IPv6 для знатоков IPv4

Ярослав Тихий

16 августа 2013 г.

Предисловие

Умный поймет с полуслова.
(Пословица)

Хотя IPv6 уже давно у всех на слуху, многие сетевые инженеры по-прежнему не спешат с головой погрузиться в новую технологическую реальность, и у них на то есть уважительная причина: банальный недостаток времени на изучение новой архитектуры. Безусловно, не осталось никого, кто не слышал бы о пресловутом 128-битном адресе IPv6; но, увы, слишком часто познания о новом протоколе этим и ограничиваются. По злой иронии, сейчас, когда адресное пространство IPv4 фактически исчерпано и все поголовно рвутся осваивать IPv6, вынужденный и поспешный переход меньше всего способствует глубокому пониманию достоинств нового протокола. Ведь и правда, кто же станет ночи напролет корпеть над многочисленными тонкостями, когда обстоятельства требуют только удлинить адрес IP и как можно скорее? Такой поворот событий оказывает медвежью услугу всем участникам процесса. С одной стороны, авторы IPv6 проделали огромную работу по созданию новой архитектуры сетевого уровня для *Internet*, но их фундаментальные идеи так и не получают широкого признания — совершенно невозможного без *знания*. С другой стороны, сетевые инженеры погрязли в рутине и лишены того удовольствия, которое испытывает специалист, только вполне осмыслив и оценив по достоинству новую технологию — не говоря уже о том, что само эффективное ее использование невозможно без всестороннего понимания.

Как ни парадоксально это звучит, едва ли не труднее всего освоить IPv6 практикующим экспертам отрасли. Во-первых, им просто некогда учить новый предмет с азов, потому что их время расписано на месяцы, если не годы, вперед. А во-вторых, в их случае неинтересно и непродуктивно заниматься по учебнику для начинающих, потому что им давно уже не надо объяснять, что такое байт или пакет.

В полной мере ощутив эти трудности на собственном опыте, мы осмелимся предложить старой гвардии игру поинтереснее. В этой книге мы поставим себя на место создателей IPv6 и вообразим, будто это мы сами — вместе с читателем, конечно же! — по частям разрабатываем и собираем механизм IPv6, опираясь только на наш опыт с IPv4. Поспешим заверить: мы вовсе не пытаемся присвоить чужие лавры, а лишь воздаем дань гению настоящих авторов IPv6. Ведь, как мы сами скоро убедимся, практически все аспекты IPv6 можно понять и объяснить с четких технических позиций. Пока индустрия TCP/IP не торопилась принять IPv6 на вооружение, у его разработчиков было достаточно времени, чтобы старательно отшлифовать каждую деталь. Если IPv4 — это был, по сути, плод эксперимента, создание первопроходцев, у которых еще не было опыта и готовых ответов на многие вопросы, то IPv6 — тщательно продуманный механизм, в котором каждая деталь находится на своем месте.

Нам остается надеяться, что такой не совсем традиционный подход к изложению материала поможет воздать всем по заслугам. Прежде всего, эксперты по TCP/IP, стремящиеся включить IPv6 в свой арсенал, будут вознаграждены за потраченные на учебу усилия. По ходу курса у них будет возможность насладиться знакомством с новыми интересными идеями и еще раз подтвердить собственный уровень, а в результате они за короткое время досконально разберутся в основных деталях IPv6 и поймут их как по отдельности, так и во взаимосвязи. По мере того, как спрос на IPv6 день ото дня растет, высокая квалификация отдельных специалистов станет почвой для формирования

культуры, основанной на продуманном и эффективном применении возможностей IPv6. А это, в свою очередь, и будет самый лучший знак благодарности создателям IPv6, на какой только способно инженерное сообщество IT.

Содержание

1. Предпосылки

> Революция — это не торжественный обед и не литературный вечер, не рисунок и не вышивка; ее нельзя вершить изысканно и учтиво. Революция — это акт насилия. (Мао Цзе-Дун)

Чтобы разработать для *Internet* сетевой протокол нового поколения, нужен весьма серьезный толчок. Ведь вложения труда и средств в уже существующий протокол IP версии 4 весьма и весьма велики. Пусть стартовым импульсом нам послужит тот прискорбный факт, что глобальное адресное пространство IPv4 уже практически исчерпано [1, 2, 3].

> Обратите внимание: мы используем необходимость восполнения адресного пространства только как отправную точку, повод начать работу. Нашей главной целью будут новые возможности во всех аспектах работы протокола.

Прежде чем мы двинемся дальше, нам следует разобраться, почему адресное пространство IPv4 считается исчерпанным. Разумеется, это вовсе не означает, что число хостов в *Internet* уже достигло 2^{32}. Куда же тогда делись адреса? Мы должны разгадать тайну пропавших адресов, чтобы по возможности избежать той же проблемы в новом протоколе.

Часть адресов IPv4 зарезервирована для специальных целей [RFC 5735]; часть теряется на границах подсетей; и, наконец, какое-то их число просто стало жертвой неоптимального распределения в те времена, когда недостатка в адресах еще не ощущалось: организации получили большие блоки адресов, но назначили узлам[4] далеко не все из них.

Главный же источник потерь кроется в иерархической структуре индивидуальных (*unicast*) адресов IPv4 [RFC 3194]. Как мы знаем, их распределяют не по одному, а путем последовательного деления бо́льших блоков на меньшие [§4.2 RFC 4632]: IANA «раскраивает» все адресное пространство IPv4 и выдает самые большие блоки в ведение RIR; те делят полученные блоки и выдают в руки LIR блоки поменьше; а LIR уже выдают адреса или напрямую конечным пользователям, или провайдерам *Internet* без статуса LIR, чтобы те передали их пользователям. Конечный пользователь в этой схеме тоже может быть организацией, в которой есть своя сетевая под-иерархия.

На каждом уровне иерархии существует некий запас блоков, потому что ни один администратор в здравом уме не рискнет выделить все доступные ему блоки до того, как получит новый блок «сверху». И, чем выше уровень в иерархии, тем дороже обходится такой запас, потому что число адресов в среднестатистическом блоке экспоненциально растет при движении вверх по этой иерархии.

> Деление и слияние как способ эволюции объектов имеет и другие, не столь важные для нас теперь, но оттого не менее интересные последствия для численного выражения их размера [5].

Потери адресов при иерархическом распределении неизбежны; это плата за то, что адреса IP не приходится выпрашивать по одному непосредственно у IANA. К счастью, мы не первые, кто столкнулся с этой проблемой: она хорошо знакома телефонистам. Ведь

---

[1] RIPE Community Resolution on IPv4 Depletion and Deployment of IPv6. RIPE, 2007. <http://www.ripe.net/news/community-statement.html>

[2] IPv4 Exhaustion. RIPE. <http://www.ripe.net/internet-coordination/ipv4-exhaustion>

[3] G. Huston. IPv4 Address Report <http://www.potaroo.net/tools/ipv4/>

[4] То есть хостам и маршрутизаторам.

[5] В. Арнольд. Статистика первых цифр степеней двойки и передел мира. Квант, 1998, №1. http://kvant.mccme.ru/pdf/1998/01/kv0198arnold.pdf

телефонные номера тоже основаны на иерархии префиксов, только десятичных, а не двоичных, и номера эти уже не раз приходилось удлинять, чтобы поддержать рост телефонных сетей. Этот опыт для нас особо ценен, так как он говорит, что потери адресного пространства поддаются эмпирическому прогнозированию с помощью логарифмического коэффициента плотности узлов *HD* [RFC 3194]:

$$HD = \frac{\ln N_{тек}}{\ln N_{макс}},$$

где $N_{тек}$ — текущее число занятых адресов, а $N_{макс}$ — максимальное их число, то есть сколько их всего в адресном пространстве. *HD* — это отношение не самих величин, а их логарифмов как раз для того, чтобы учесть экспоненциальный характер данной системы.

> Очевидно, что основание логарифма в этом отношении может быть любым, пока оно одинаково в числителе и знаменателе.

Ссылаясь на опыт телефонных сетей и вычислительных сетей предыдущего поколения, [RFC 3194] делает вывод, что дальнейший рост сети становится практически невозможным, когда *HD* достигает порогового значения 0,87. Для 32-битного адреса это означает порядка 240 миллионов узлов, тогда как адресов IPv4 уже выделено, по разным оценкам, 2–3 миллиарда [6]. Проблема налицо!

> Между тем, заметьте: *Internet* достиг ранее неслыханного значения *HD* 0.96–0.98, и только тогда начались настоящие трудности с адресным пространством IPv4. Что может быть лучшим комплиментом механизму распределения адресов, принятому в *Internet*?

> Есть даже мнение, что управление ростом сетей IP на основе пороговых значений *HD*, взятых из истории, может оказаться чрезмерно консервативным и недооценить возможности технологии IP по поддержанию густонаселенных сетей [RFC 4692].

> Обратите внимание, что применять коэффициент *HD* имеет смысл только к иерархическим системам. Если распределение одноуровневое, то ничто не мешает вам раздать адреса, или что вы там раздаете, до конца. Например, если у вас есть пачка сигарет, то вы вполне можете раздать эти сигареты все до одной. Однако если у вас есть целый ящик сигарет, а раздача идет блоками, дальше пачками, и только затем отдельными сигаретами, то изрядная доля сигарет останется в пачках и блоках, потому что их новые хозяева решат приберечь немножко на черный день.

Нехватка адресов всерьез сказывается не только на практике, но и на философии *Internet*. Получение глобальных адресов IPv4 становится все более хлопотным делом, потому что администраторы всех уровней иерархии стараются экономить этот ценный ресурс. В ответ рождаются технические решения, которые позволяют обходиться всего несколькими глобальными адресами на сеть, независимо от числа хостов в ней. Самые популярные приемы — это NAT и proxy в сочетании с частным адресным пространством. В свою очередь, повсеместное их применение приводит к неявному отказу от главного принципа *Internet*: сквозной прозрачности на сетевом уровне [RFC 2775], когда любой хост А может послать пакет IP любому хосту Б (где А может быть равно Б). В TCP/IP этот принцип осуществим, только пока у каждого хоста есть собственный глобальный адрес.

> Можно сказать и так: исторически хосту *Internet* полагалось иметь глобальный адрес, чтобы сквозная прозрачность стала реальностью. Ведь пока хосты А и Б находятся в одном адресном пространстве IP, адрес Б — единственное, что должен знать хост А, чтобы однозначно передать пакет хосту Б.

А следующим шагом сетевые инженеры открыто хоронят сквозную прозрачность: они объявляют ее вредной для безопасности и конфиденциальности доступа к *Internet*, а

[6] G. Huston. IPv4 Address Report <http://www.potaroo.net/tools/ipv4/>

затем провозглашают NAT и proxy универсальными защитниками частной сети. Давайте посмотрим, можно ли согласиться с такой точкой зрения.

Прежде всего, надо иметь в виду, что экономия адресного пространства, защита безопасности и обеспечение конфиденциальности доступа — это совершенно разные задачи, каждую из которых можно решить только целенаправленно. Поэтому NAT и proxy экономят сетевые адреса, и не более того. Злоумышленники уже научились преодолевать границы частных сетей, например, заражая компьютер внутри такой сети с помощью электронного письма и заставляя его самостоятельно выходить на связь с «подпольным центром управления». А маскировка сетевого адреса еще не гарантирует, что сеансы пользователя останутся конфиденциальными: пользователя может выдать не только его адрес IP, но также данные прикладного уровня, например, заголовки электронных писем и поля запросов HTTP, и даже значения полей в заголовках сетевого и транспортного уровней, если провести их корреляцию во времени. (Простейший пример — когда хост последовательно увеличивает значение идентификатора в заголовке IPv4, так что его пакеты легко выделить из общего потока даже после NAT.)

> Еще популярно мнение, что NAT обеспечивает безопасность частной сети при минимальных затратах на настройку и поддержку, по принципу: установил и забыл. Сторонники этой точки зрения вообще склонны к забывчивости и потому не помнят, сколько человеко-часов уходит на то, чтобы «подружить» NAT с некоторыми протоколами, из которых вспомним хотя бы IPsec [[7], RFC 3715], FTP и SIP.
>
> Прочие сетевые приложения, для полноценной работы которых необходим известный или хотя бы фиксированный адрес IP, сами идут на различные трюки, собирательно известные как «односторонняя фиксация собственного адреса» (*Unilateral Self-Address Fixing*, **UNSAF**), чтобы обхитрить NAT [RFC 3424]. В свою очередь, это создает дополнительные бреши в и без того слабой системе безопасности, так как долговременным адресом слабо защищенного приложения может воспользоваться и злоумышленник.
>
> Чисто практические затруднения, вызванные NAT, — это только видимая верхушка айсберга. У применения NAT есть и менее очевидные последствия на уровне архитектуры *Internet*. Их развернутый анализ можно найти в [RFC 2993].

Теперь перейдем к тезису о том, что сквозная прозрачность *Internet* опасна. Может ли быть опасным принцип? Наверное, да, если он однозначно ведет к опасным результатам, но это едва ли относится к сквозной прозрачности. Речь ведь идет не о полной доступности, когда всем хостам позволен бесконтрольный обмен данными, а о *сквозной* адресации. Это значит, что непосредственное взаимодействие хостов всегда возможно *технически*, а разрешать его или нет — вопрос чисто административный. Если же мы отступимся от принципа сквозной прозрачности, то обязательно возникнут ситуации, когда прямое взаимодействие узлов необходимо, но технически невозможно. Любой инженер-практик знает, как неприятно оказаться в подобном тупике.

> Конечно, NAT и proxy — далеко не единственные враги сквозной прозрачности *Internet*. За более подробным «черным списком» мы отсылаем читателя к [RFC 2775].

Восстановить сквозную прозрачность *Internet* можно, только увеличив число глобальных сетевых адресов, — иного пути здесь нет. Число адресов IPv4 ограничено, потому что у их двоичного представления фиксированная длина. Мы не собираемся отказываться от двоичного представления информации, так что нам придется просто

---

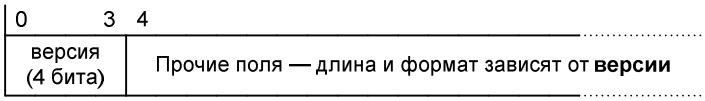
[7] L. Phifer. The Trouble with NAT. The Internet Protocol Journal - Volume 3, No. 4. <http://www.cisco.com/web/about/ac123/ac147/ac174/ac182/about_cisco_ipj_archive_article09186a00800c83ec.html>

увеличить число двоичных разрядов в адресе IP. Насколько радикальные изменения это вызовет в технологии TCP/IP?

Попробуем для начала консервативный подход. Можно ли расширить адрес IP, не отказываясь от протокола IPv4 и, в частности, от его формата заголовка? Например, мы могли бы поместить новые, расширенные адреса в специальные опции IPv4. ARP справился бы с новыми адресами, так как длина адреса в нем явно указывается. С ICMP ситуация сложнее, так как некоторые типы сообщений ICMP включают в себя адреса IPv4. Наконец, фиксированное смещение адресов от начала заголовка облегчает быструю аппаратную маршрутизацию пакетов, тогда как размещение адресов в опциях значительно усложнило бы ее. Ну, и вообще говоря, опции потому так и называются, что они должны быть факультативными, то есть необязательными для исполнения. Это окончательный аргумент против того, чтобы новые адреса находились в опциях IPv4. В то же время, основной заголовок IPv4 рассчитан только на 32-битные адреса. Вывод такой, что текущим форматом заголовка IPv4 нам все-таки придется пожертвовать, поскольку в нем нет места для новых адресов.

Теперь, когда мы готовы распрощаться с заголовком IPv4, у нас возникает противоположное искушение: не просто расширить адреса, а радикально переделать протокол IP, чтобы исправить в нем существующие недостатки и добавить новые возможности. Этим мы и займемся в рамках данного курса.

Но, несмотря на наши революционные настроения, давайте сохраним один элемент протокола, а именно поле версии в заголовке, как показано на Фиг. 1. С его помощью можно будет отличить новые пакеты IP от старых, не привлекая дополнительных сведений: сетевому стеку будет достаточно проверить значение этого поля. Если мы так поступим, то будущий протокол формально окажется новой версией IP. По историческим причинам он получил от IANA номер версии 6 [8], поэтому и назовем мы его «IP версии 6», или кратко **IPv6**.



Фиг. 1. Формат пакета IP, не зависящий от версии протокола

Мы скоро убедимся, тем не менее, что архитектурный фундамент IPv4 и IPv6 не так уж сильно разнится. Говоря об их общих чертах, мы будем использовать собирательное наименование: **IP**.

2. Адрес IPv6

2.1. Длина адреса IPv6

— Скажите, Шура, честно, сколько вам нужно денег для счастья? — спросил Остап. — Только подсчитайте все.

— Сто рублей, — ответил Балаганов, с сожалением отрываясь от хлеба с колбасой.

— Да нет, вы меня не поняли. Не на сегодняшний день, а вообще. Для счастья. Ясно? Чтобы вам было хорошо на свете.

Балаганов долго думал, несмело улыбаясь, и, наконец, объявил, что для полного счастья ему нужно шесть тысяч четыреста рублей и что с этой суммой ему будет на свете очень хорошо.

[8] http://www.iana.org/assignments/version-numbers

— Ладно, — сказал Остап, — получите пятьдесят тысяч.

(И. Ильф, Е. Петров. «Золотой теленок»)

Наша задача номер один — расширить адресное пространство. Поэтому начнем мы нашу «революцию в IP», конечно же, с сетевого адреса. А самый первый вопрос касательно нового адреса звучит предельно просто: сколько адресных битов будет достаточно, чтобы избежать в обозримом будущем новых «революций»? Вариантов ответа на этот вопрос есть немало, однако почти все их можно отнести к одной из трех больших групп:

1) «не надо гадать на кофейной гуще»: оставим попытки предсказать длину нового адреса IP и сделаем ее переменной;
2) «к природе за советом»: давайте посмотрим, сколько всего песчинок в пустыне Сахара, молекул воды в мировом океане или протонов в наблюдаемой вселенной, и решим, что большего числа адресов IP человечеству никогда не понадобится;
3) «возможности во главе угла»: длина адреса IP должна стать такой, чтобы это открыло новые возможности на уровне работы протокола.

Первый подход, безусловно, очень хорош в теории, так как он позволил бы просто увеличивать длину адреса по мере расходования адресного пространства. Более того, он не нов и применяется в протоколе ISO CLNP. С другой стороны, адреса переменной длины значительно труднее обрабатывать аппаратно. Вдобавок, длина сетевого заголовка тоже становится переменной, а это еще сильнее затрудняет быструю аппаратную обработку пакетов. Так как IP уже давно применяют в областях, где требования к скорости обработки пакетов очень высоки (телефонные магистрали, системы управления реального времени), нам придется отказаться от этого подхода, несмотря на его привлекательность.

> Кстати, и интегральные схемы, и программы для работы с адресами переменной длины стали бы не только сложнее, но и дороже. Учитывая, что скоро стек TCP/IP будет встроен буквально в каждую кофеварку, фактор стоимости тоже играет большую роль.
>
> Тем не менее, любопытно отметить, что эта простая идея: «TCP и UDP поверх ISO CLNP», — легла основу одной из альтернативных разработок «IP нового поколения», известной как **TUBA** [RFC 1347]. Это было изящно, не правда ли? Если бы проект TUBA удался, мы бы с вами сейчас «разрабатывали» IPv9 вместо IPv6 [9]. Однако даже сами авторы TUBA выражали опасение, что гибкость адресации CLNP будет в ущерб производительности сети [§6.5 RFC 1347].

Второй подход привлекателен тем, что сразу же дает нам численные оценки, но насколько безупречна логическая связь между числом каких-то частиц и размером нового адресного пространства? Конечно, интуитивно понятно: возможностям человека по назначению адресов IP есть передел, и он вряд ли превышает знакомые нам «астрономические» величины. Однако кажущаяся простота этого подхода может оказаться ширмой, которая заслонит от нас технически обоснованные критерии выбора из области TCP/IP, а не космологии.

> Кроме того, нечеткость мысли нередко превращается в привычку и влечет за собой грубые просчеты. Выбирая наугад верхнюю оценку необходимого числа адресов, «природники» порой ленятся подвергнуть критическому анализу даже собственные гипотезы. Яркий пример этого можно найти в одном из ранних проектов «нового IP», чьи авторы рассчитали необходимый размер адресного пространства, исходя из суммарного числа байтов на всех сетевых ресурсах *Internet*

[9] http://www.iana.org/assignments/version-numbers

[RFC 1475]. При этом они предполагали, что один ресурс никогда не станет содержать больше 4 гигабайт [§2.1 RFC 1475]. Очевидно, что проблема здесь вовсе не в точности оценки размера сетевого ресурса через 10 лет, а в самой постановке задачи.

И все-таки любопытно, каковы природные сверхвеличины? Оказывается, их порядок был известен еще Архимеду. Демонстрируя возможности науки перед царем Гелоном, Архимед построил весьма убедительную для своего времени модель [[10]], согласно которой в объеме вселенной умещается 10^{63} песчинок. Современные астрофизические теории говорят не о песчинках, а об атомах, которых в модельной вселенной насчитывается, по разным оценкам, порядка 10^{71}–10^{87} штук. Если принять средний диаметр песчинки равным 300 мкм, плотность — 2,4 г/см^3 (кварц), а средний вес составляющих ее атомов — ~30 атомных единиц (SiO_2), то в одной такой песчинке содержится порядка 10^{18} атомов, а в Архимедовой вселенной их всего 10^{81}, что не выходит за пределы сегодняшних представлений. Таким образом, оказывается, что сила научного воображения — величина практически постоянная еще с античных времен.

А что же заслоняет от нас хромая логика второго подхода? Она неявно примиряет нас с той мыслью, что все адреса IPv6 тоже когда-то будут розданы и все, на что мы способны, это отодвинуть роковой день как можно дальше в будущее. Иначе говоря, этот подход рассматривает адресное пространство как аморфную кучу песчинок, из которой их черпают, неважно, по одной, ведрами или ковшом экскаватора. В действительности же у адресного пространства всегда возникает определенная структура, и нам пора ее сознательно формировать, потому что именно от нее будет напрямую зависеть срок службы адресного пространства IPv6. Взамен мы будем вознаграждены вполне обоснованной оценкой его требуемого размера.

Недостаток «природнического» подхода — архитектурный: он не привел бы к преждевременному исчерпанию нового адресного пространства, но вполне мог бы направить нашу мысль по ложному пути.

Прежде всего, обратим внимание, что интересующий нас адрес в любом случае будет двоичный (цепочка битов), а распределять двоичные адреса удобнее всего не по одному или произвольными диапазонами, а в виде префиксов. Мы уже знакомы с этой процедурой в контексте IPv4. Субъект распределительной иерархии получает сверху определенный двоичный префикс, дополняет его несколькими битами и спускает удлиненный префикс уровнем ниже. Для простоты допустим, что он дополняет префикс постоянным числом битов n. Тогда данный субъект сможет превратить полученный сверху префикс в не более чем 2^n префиксов для субъектов нижележащего уровня. Скажем, получив «сверху» (например, от LIR) префикс IPv4 *198.51.100.128/25* и полагая $n = 2$, можно произвести четыре префикса */27* для раздачи нижестоящим субъектам (например, рядовым пользователям): *198.51.100.128/27*, *198.51.100.160/27*, *198.51.100.192/27*, и *198.51.100.224/27*.

Что делать нашему субъекту, когда он исчерпает выданный ему префикс? Видимо, запросить сверху еще один. Затем он тоже исчерпывается, и опять требуется новый префикс, чтобы поддержать рост сети. Через некоторое время каждый субъект обладает набором разрозненных префиксов, которые не агрегируются, потому что получены из разных префиксов вышестоящего уровня. К примеру, *198.51.100.128/25* не агрегируется с *203.0.113.0/25*, хотя он мог бы агрегироваться с *198.51.100.0/25*, дав один префикс *198.51.100.0/24*, — но, увы, к этому моменту *198.51.100.0/25* уже выдан в совсем другие руки. Именно так произошло дробление адресного пространства в IPv4.

Теоретически, альтернативой мог бы стать возврат текущего префикса вышестоящему субъекту и запрос нового префикса меньшей длины, чтобы

[10] Архимед. Исчисление песчинок (псаммит). М.-Л.: Гос. Технико-теоретическое изд-во, 1932. 104 с.

увеличить число доступных битов *n*; но практически это осуществимо только на самом нижнем уровне иерархии, потому что на других уровнях субъекту пришлось бы *отобрать* у подчиненных выданные префиксы. И даже на нижнем уровне это затруднительно, так как означает перенумерацию сети — смену адресов на всех узлах.

Но что плохого в дроблении адресного пространства? Во-первых, это увеличивает административные расходы, так как каждому субъекту приходится учитывать множество полученных и выданных префиксов, хотя в идеале он мог бы получить сверху ровно один префикс и выдать по одному префиксу каждому подчиненному субъекту. Во-вторых, числу неагрегируемых префиксов пропорционально число маршрутов, которые возникнут в сети. К примеру, в «зоне без умолчания» (*default free zone*), составляющей ядро *Internet*, число субъектов (автономных систем) на январь 2010 года составило примерно 30 000, а число неагрегируемых маршрутов IPv4 — 150 000 [[11]], и это различие в пять раз вызвано, в основном, дроблением адресного пространства IPv4.

Чтобы эта проблема не возникла вновь, адресное пространство IPv6 должно быть практически неисчерпаемым на всех уровнях распределительной иерархии: каждому субъекту иерархии должно хватить одного префикса. Обратите внимание: из сказанного не следует, что мы должны запретить назначение субъекту нескольких префиксов. В некоторых случаях это полезный и технически обоснованный прием. Однако плохо, когда хроническая нехватка адресов вынуждает к постоянному и повсеместному его использованию: пользователи выпрашивают дополнительные префиксы у провайдера, тот — у региональной регистратуры и т.д.

Тем не менее, мы не поощряем разбазаривание адресов IPv6 даже при неисчерпаемом адресном пространстве. Неисчерпаемым оно останется только при обдуманном, рациональном использовании [RFC 5375, [12]].

Оценить необходимое число битов в адресе IPv6 нам позволит модель распределительной иерархии. Начнем мы с самого нижнего уровня в ней, который делить дальше нет смысла. Как мы знаем по опыту IPv4, субъект этого уровня — канал, а назначаемый ему префикс — подсеть. Иными словами, давайте допустим, что мы сохраним в IPv6 деление битов адреса на префикс подсети и номер узла в подсети.

Хотя каналу вполне можно назначить несколько подсетей, подсеть IP не может охватывать больше одного канала [§2.1 RFC 4291, RFC 4903]; это один из фундаментальных постулатов TCP/IP. Какое максимальное число узлов способен соединить один канал? Разумная оценка здесь — это число адресов MAC. Сегодня традиционные 48-битные адреса IEEE 802 уступают место 64-битным, т.н. EUI-64 [[13]]. Теоретически канал с адресацией EUI-64 может соединить порядка 2^{63} узлов, так что пусть номер узла занимает для ровного счета 64 бита, то есть 8 байт.

Как это уже было в адресе IEEE 802, один бит EUI-64 занят, чтобы отличать групповые адреса от индивидуальных. Отсюда 63-я, а не 64-я степень двойки в числе узлов EUI-64.

Отлично, с номером узла мы разобрались. А какая длина потребуется префиксу подсети? Чтобы ответить и на этот вопрос, мы двинемся вверх по распределительной иерархии. Оставшаяся часть пирамиды носит скорее административный, нежели технический характер, и поэтому нам придется принять ее как данность. В ее современной структуре можно выделить четыре основных уровня [[14]]:

---

[11] P. Smith. BGP Routing Table Analysis. APNIC. http://thyme.apnic.net/

[12] Conservation. IPv6 Address Allocation and Assignment Policy. RIPE. http://www.ripe.net/ripe/docs/ipv6policy.html#conservation

[13] Guidelines for 64-bit Global Identifier (EUI-64™) Registration Authority. IEEE. http://standards.ieee.org/regauth/oui/tutorials/EUI64.html

[14] IPv6 Address Allocation and Assignment Policy. RIPE. http://www.ripe.net/docs/ipv6policy.html

1) конечный пользователь, например, организация или частный клиент;
2) местная регистратура *Internet* — LIR (провайдер);
3) региональная регистратура *Internet* — RIR;
4) IANA.

Обратите внимание, что конечный пользователь здесь — это не человек за персональным компьютером, которому достаточно одного адреса IP, а сеть. Во-первых, провайдеры обслуживают не только частных лиц; а кроме того, и это даже важнее, все большее распространение получают домашние и персональные сети, когда один или несколько человек опутаны многоуровневой сетью, состоящей не только из традиционных ЭВМ, но также из телефонов, фото- и видеокамер, стиральных машин, кофеварок и тостеров. Сложная структура персональной сети нужна, как минимум, для безопасности. Вот представьте себе такую ситуацию. Снимая занятную сцену на камеры мобильных телефонов, горожане не замечают, что в ее фоне террорист расположил вирусный QR-код. Вечером по персональным сетям вирус перебирается из телефонов в кофеварки, и в понедельник утром жертвы получают вместо живительного напитка полную чашку водянистой бурды, отдающей горелым. Вечерние газеты пестрят заголовками о таинственной волне самоубийств… А ведь трагедию можно было предотвратить, разграничив сеть межсетевым экраном и поместив телефон и кофеварку в разные зоны безопасности.

Мы не поскупились, выдав 64 бита номеру узла, и теперь у нас возникает искушение продолжить оперировать порциями этой длины. Если мы так поступим, адрес получится длиною 320 бит, поскольку в нашей модели всего пять уровней иерархии: канал, пользователь, LIR, RIR и IANA. Безусловно, этого хватит на века, но по зубам ли будет столь длинный адрес современным вычислительным системам? Ведь пока длина адреса не превосходит разрядности вычислительной системы, элементарные операции над адресом можно выполнить, условно говоря, за единичное время. Иначе время операции начинает расти вместе с длиной адреса. Это справедливо как для традиционных ЭВМ с центральным процессором, так и для специализированных интегральных схем. Сегодня новые системы только-только подбираются к планке 256 бит [15]; так что давайте умерим нашу щедрость и посмотрим, нельзя ли будет обойтись более реальной величиной 128 бит.

Не должны мы забывать и о встроенных системах, которые до сих пор оперируют словами в 32 бита, а то и в 16 бит, потому что это уменьшает физические размеры системы и ее энергопотребление. Несмотря на свою маломощность, такие системы не отстают от своих «старших братьев» в том, что касается TCP/IP! Однако работа со слишком длинными адресами может оказаться им не по плечу ввиду ограниченного объема ОЗУ, так как вырастут в размере и сетевые структуры данных, и работающий с ними код.

Если весь адрес IPv6 окажется 128-битным, то на префикс подсети останется 64 бита. Допустим для первоначальной оценки, что это число битов поровну делят между четырьмя уровнями пирамиды. Тогда каждый уровень сможет получить 16 бит: в сети конечного пользователя будет до 2^{16} подсетей (по 2^{64} узлов каждая); у провайдера — до 2^{16} пользователей; каждая RIR сможет поддержать до 2^{16} LIR; наконец, IANA достанется 2^{16} префиксов как для выделения их RIR, так и для служебных целей.

Чтобы почувствовать, насколько это много или мало — 16 бит на каждом уровне иерархии, — давайте переведем несколько примеров на доступный нам язык IPv4. Так, если бы конечный пользователь назначал своим подсетям исключительно префиксы */24*, то для выделения 2^{16} префиксов ему понадобился бы блок */8* (или, по старинке, сеть класса A) — предел мечтаний любой корпорации. При этом одна подсеть была бы ограничена всего лишь 254 узлами. В IPv6 же 16 бит на подсети — это минимум, что может получить пользователь, а в каждой его подсети будет до 2^{64} узлов! Что касается

[15] Например, см. расширения SSE5 от *AMD* и AVX от *Intel*.

LIR, то до начала жесткой экономии адресов IPv4 новая LIR сразу получала блок */19*, а он содержал всего лишь 2^{13} *отдельных адресов*, или же 32 подсети */24*. Так что даже при делении префикса подсети IPv6 поровну LIR достанется втрое больше доступных битов, чем было во времена IPv4. Выходит, даже 16 бит на уровень могло бы хватить.

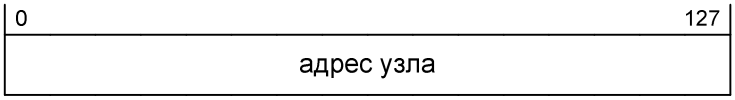
Конечно же, на практике число подчиненных субъектов будет расти при движении вниз. Так, IANA обслуживает всего несколько RIR, в ведении каждой RIR находится порядка сотен или тысяч LIR, а крупная LIR может поставлять услуги *Internet* миллионам конечных пользователей. Эту неравномерность шкалы вполне можно отразить в числе битов, которыми станет распоряжаться субъект каждого уровня, например, так:

- IANA — 8 бит (256 RIR и служебных префиксов);
- RIR — 16 бит (65 тысяч LIR);
- LIR — 24 бит (16 миллионов сетей-клиентов);
- конечный пользователь — 16 бит (65 тысяч подсетей).

Кроме того, основанное на префиксах распределение адресов позволяет варьировать размер блока в зависимости от запросов получателя, так что приведенные нами границы — лишь иллюстрация, а вовсе не строгое правило [§5 RFC 6177]. Небольшой LIR вполне хватит 16 бит, тогда как крупный корпоративный пользователь сможет получить у LIR в свое распоряжение, скажем, 24 бита и даже более того за счет слияния смежных 16-битных блоков. Тем не менее, *ориентировочно* в нашей модели конечный пользователь IPv6 получит от LIR префикс */48* и еще 16 бит останутся на внутреннюю структуру его сети.

> В практике IPv6 уже есть прецеденты, когда особо крупная LIR получила у RIR */19* [§2.4.1 RFC 5375], что означает 29 доступных битов — конечно же, не считая тех битов, которые будут отданы в распоряжение конечных пользователей этой LIR. В терминах нашей оценочной модели, эта LIR сможет подключить до 2^{29} пользователей, причем каждый получит довольно щедрую долю в адресном пространстве IPv6.
>
> Чтобы выданный блок можно было укрупнить по требованию за счет слияния с соседним(и), достаточно следовать известной практике, когда выдающая сторона вводит единичные биты не в младшие, а в старшие разряды префикса [RFC 1219, RFC 3531]. Например, распоряжаясь 4 битами, их значения следует выдавать не как 0, 1, 2, 3…, а как 0, 8, 4, 12, 2, 10, 6, 14… Тогда при доле заполнения до $1/2^N$ у каждого выданного префикса заведомо будут свободны 2^N–1 префиксов, следующих за ним, и у всей группы N младших битов будут нулевые, что обеспечит агрегирование. Например, пока выдано не более четверти доступных префиксов, каждый выданный блок можно укрупнить вдвое или вчетверо. Мы предлагаем читателю самостоятельно разобраться с двоичной арифметикой этого полезного приема. (Подсказка: рассмотрите «зеркальное отражение» *n*-разрядной двоичной записи 0, 1, 2, 3…)

Таким образом, мы видим, что префикс подсети длиной 64 бита оставляет достаточно пространства для гибкого распределения битов согласно принципу: каждому по потребностям. Еще 64 бита мы заняли тем, что мы условно назвали номером узла в подсети. О структуре индивидуального адреса IPv6 мы еще поговорим в §2.6, а пока что сделаем наш первый технический вывод: всякий адрес IPv6 будет цепочкой из 128 битов [RFC 4291], как это показано на Фиг. 2.



Фиг. 2. Адрес IPv6 в самом общем виде

> Мы отложим численную интерпретацию этой цепочки до §2.2. Тем не менее, такая цепочка — несомненно, упорядочена: ее биты не «перетасованы», а следуют в строго определенном порядке.

Конечно, если бы IPv6 впервые возник в 2010 году, его адрес вполне мог бы удлиниться до 256 бит. Во-первых, это дало бы больше свободы в числе уровней распределения и маршрутизации. Во-вторых, разговоры об экономии префиксов [16] можно было бы отложить на неопределенный срок. С другой стороны, достаточное, но все же ограниченное число префиксов заставит инженера лишний раз подумать, а думать ему полезно. Избыток вычислительных ресурсов — это не оправдание бездумному их расходованию.

Любопытно заметить, что «ничто не ново под луной»: завершенный 1 января 1983 года переход *Internet* от NCP к TCP/IP решал ту же самую задачу по расширению адресного пространства, а длина адреса тогда тоже увеличилась вчетверо, с одного байта до четырех.

Давайте оценим, сколько адресов IPv6 надо выдать, чтобы коэффициент HD (см. §1) достиг величины 0,8 (пороговое значение «пора задуматься» [§4 RFC 3194]). Нам будет удобно работать с двоичными логарифмами. Логарифм искомого числа равен $0{,}8 \times 128 \approx 102$. То есть выходит, что «первый звонок» прозвенит за *26 двоичных порядков* до абсолютного предела. В этом парадокс иерархического распределения: адреса фактически исчерпываются заметно раньше, чем можно было бы предвидеть на основании их общего числа.

2.2. Текстовое представление адреса IPv6

Итак, адрес IPv6 получил свое первичное, то есть двоичное воплощение — простите за невольный каламбур. Теперь подходящее время привести на печатной странице пример адреса IPv6, но как нам его записать? В виде нулей и единиц? Да, это возможно, но довольно неудобно. Почему? Во-первых, избыточность такой записи очень велика, ведь эта запись основана на двухбуквенном алфавите, тогда как в распоряжении систем человеческой письменности куда больше символов. Во-вторых, направление письменности не одинаково в разных традициях, и упорядоченную цепочку символов разные народы будут вправе записать по-разному, а в управлении вычислительными системами такой разнобой совершенно неприемлем.

Поэтому давайте следующим шагом предложим лаконичное и однозначное текстовое представление адреса IPv6. В первую очередь, оно поможет людям записывать, читать и, по мере возможности, запоминать такие адреса. Однако записывают адреса не только на бумаге — их также вводят в память ЭВМ. В частности, именно к этому сводится ручная настройка адресов. Кроме того, адреса считывают с экрана и выводят на печать. Следовательно, текстовое представление адреса должно использовать только печатаемые символы, но в какой кодировке? Ведь их набор в разных кодировках отличается. К счастью, на практике можно выбрать подмножество символов, которое будет «по зубам» любой существующей ЭВМ, например: буквы латинского алфавита (без определенного регистра), десятичные цифры и основные знаки препинания.

Какие знаки препинания считать основными? Сегодня можно сказать, что это знаки препинания из кодировки ASCII — базовой кодировки, принятой в *Internet* [RFC 20] и до сих пор не сдавшей все свои позиции универсальной кодировке UTF-8 [RFC 5198]. А еще более мудрый выбор — это символы пунктуации, присутствующие одновременно и в ASCII, и в EBCDIC.

В копилке нашего опыта уже есть подобное решение для адреса IPv4, а именно формат «десятичный с точками» (*dotted decimal*):

- На входе дан адрес IPv4 как *упорядоченная* цепочка из 32 бит.
- Сначала биты адреса делят на четыре группы по восемь бит в каждой, то есть байты. При этом сохраняют относительный порядок битов.

[16] Conservation. IPv6 Address Allocation and Assignment Policy. RIPE. http://www.ripe.net/docs/ipv6policy.html#conservation

- Затем находят численное значение каждого байта и записывают его как десятичное число, руководствуясь при этом целым беззнаковым кодом и сетевым порядком битов: первый бит по порядку — самый старший, то есть наиболее значимый (MSB).
- Наконец, перечисляют эти значения байтов так, чтобы слева стоял самый первый, он же наиболее значимый, байт. Между значениями ставят точки.

> Любопытно отметить, что традиция позиционной записи чисел, когда слева оказывается самый старший разряд, а справа самый младший, не связана напрямую с направлением письменности. Например, арабская письменность тоже следует этому правилу. Видимо, эта традиция — более древняя, чем современные письменности, и восходит к тому единому первоисточнику, откуда разные культуры позаимствовали десятичную систему счисления.
>
> Обсудите применение такой нотации в системах вертикальной письменности. ☺
>
> В связи с десятичной записью байтов адреса IPv4 возникает вот какой вопрос: может ли она содержать ведущие нули? С одной стороны, арифметика позиционной записи гарантирует, что такие нули не меняют численного значения. Но, с другой стороны, на практике ведущие нули могут вызывать сложности. Скажем, в языке Си ведущий ноль означает, что число записано по основанию 8, а из языка Си через его стандартную библиотеку (см. функцию *strtol*) эта интерпретация вполне может проникнуть и в приложения, если программист не зафиксирует основание счисления явным образом, а позволит библиотеке угадывать его. Например, функции *inet(3)* из библиотеки *BSD libc* позволяют вводить октеты адреса IPv4 по любому основанию, принятому в языке Си: 8, 10, 16 [17], — используя префиксы `0` и `0x` для выбора оснований 8 и 16, соответственно. В результате существует риск, что разные приложения могут интерпретировать ведущий ноль по-разному. Чтобы избежать неоднозначности, современная трактовка, основанная на формате URI, такова: в общепринятой записи адреса IPv4 ведущие нули недопустимы [§3.2.2 RFC 3986]. Конечно, эти соображения обусловлены историческими причинами, а не желанием оправдать небрежность разработчиков. Поэтому в IPv6 мы оставим корректную обработку ведущих нулей в текстовой записи адреса на совести программиста, а нам останется четко определить, что эти нули означают.

Что в этом рецепте можно улучшить? Во-первых, у десятичной записи довольно высокая избыточность; шестнадцатеричная запись может быть короче в $\lg 16 \approx 1{,}2$ раза. Для длинного адреса это плюс.

Во-вторых, текстовую запись адреса IPv4 не очень удобно сопоставлять с двоичными масками подсетей и шестнадцатеричными дампами пакетов при отладке сети. Хотя регулярное преобразование чисел из десятичного представления в двоичное и обратно — неплохая гимнастика для ума, с длинными адресами IPv6 упражняться придется слишком часто и помногу, если мы не откажемся от десятичной записи в пользу шестнадцатеричной.

Наконец, шестнадцатеричное представление удобно делить на отдельные байты, так как каждая цифра представляет полубайт, а их пара — целый байт. Напротив, опустив в записи адреса IPv4 точки, мы в большинстве случаев получим другое десятичное число, нежели численное значение исходного адреса; и наоборот, общепринятую запись адреса IPv4 в общем случае нельзя получить, просто вставив точки в десятичную запись его численного значения.

К примеру, адрес *203.0.113.42* как 32-битное целое беззнаковое число равен 3405803818 десятичному или 0xCB00712A шестнадцатеричному, и только в

[17] http://www.freebsd.org/cgi/man.cgi?query=inet&apropos=0&sektion=3&manpath=4.4BSD+Lite2&arch=default&format=html

шестнадцатеричной записи значения отдельных байтов видны невооруженным глазом: 0xCB — 203, 0x00 — 0, 0x71 — 113 и 0x2A — 42.

Вместе этих аргументов более чем достаточно, чтобы выбрать основанием текстовой нотации IPv6 число 16.

> Тем временем в Австралии ведутся эксперименты с записью адресов IPv6 по основанию 85 [RFC 1924]. ☺

Пообещав наглядную связь между численным значением адреса IPv6 и его записью, перейдем к осуществлению нашего плана. Что мы понимаем под численным значением адреса? Как мы уже говорили, цепочку битов можно рассматривать как число в целом беззнаковом коде, если определить старшинство битов в ней. В TCP/IP принят сетевой порядок старшинства: первый бит — наиболее значимый. Это позволяет нам найти численное значение адреса IPv6, равно как и адреса IPv4, поскольку оба они — упорядоченные цепочки битов.

Следующий шаг — от абстрактного численного значения к его шестнадцатеричной записи. Чтобы записать цепочку из 128 битов как беззнаковое число в шестнадцатеричном представлении, потребуется 32 знака, включая возможные ведущие нули. Давайте для удобства добавим нейтральные символы-разделители через каждые несколько знаков, как это делают с телефонными номерами. Пусть в группе будет четыре знака, а разделяет группы знак двоеточия, а не точка — это позволит не спутать адрес IPv6 с адресом IPv4. Тогда, например, адрес с таким численным значением по основанию 16:

```
20010DB80000000000000000100006789
```

можно записать так:

```
2001:0DB8:0000:0000:0000:0001:0000:6789
```

> Выбор двоеточия на роль разделителя групп может показаться неудачным, потому что двоеточие уже нагружено несколькими смежными ролями и возможна путаница. Во-первых, двоеточие разделяет адрес хоста и номер порта в формате URL. По этой причине адрес IPv6 в составе URL приходится заключать в квадратные скобки [§3.2.2 RFC 3986].[18] Во-вторых, двоеточие — это альтернативный разделитель в текстовой записи адресов MAC.[19] Но, если перебрать все символы пунктуации ASCII, то среди них окажется не так уж много удобных вариантов. Одно из требований к хорошему разделителю — это чтобы он был нейтральным в командной строке *Unix*. Ведь адреса часто бывают аргументами сетевых утилит, например, *ping*, *telnet* и *tcpdump*. Если теперь адреса IPv6 придется постоянно закавычивать или экранировать, то это нарушит уже сложившуюся практику, что было бы крайне нежелательно (т.н. правило наименьшего удивления).

Это был наш первый пример *настоящей* текстовой записи адреса IPv6. Он уже вполне отвечает формату и готов к использованию. Но мы видим, что его избыточность все еще велика: в нем много раз повторяются нули. А ведь по нашему плану адресное пространство IPv6 будет неисчерпаемым, а значит, разреженным, так что и на практике в адресах будут встречаться длинные цепочки нулей. Как бы нам сократить их запись? Для этого нам понадобится пара дополнительных правил.

Во-первых, давайте рассматривать каждую группу из четырех знаков как отдельное 16-битное беззнаковое число. Цепочка битов, представленная таким числом, не

---

[18] Обратите внимание, как создатели электронной почты *Internet* за много лет до IPv6 предусмотрели и обошли эту трудность: формат доменного литерала с самого начала требовал заключать адрес узла в квадратные скобки, независимо от сетевого протокола [§3.3 и §6.2.3 RFC 822].

[19] Согласно букве IEEE 802, стандартный разделитель — это дефис, тогда как двоеточие указывает на обратный порядок битов [§3.1.2 и §3.1.8 IEEE 802-2001]. Тем не менее, на практике двоеточие часто применяется вместо тире при прямом порядке битов. (В IEEE 802 принят порядок, обратный сетевому, то есть LSB.) Например, этой традиции следуют операционные системы семейства *Unix*.

изменится, если мы отбросим ведущие нули, поскольку мы уже зафиксировали ее длину. Так что мы вправе объявить ведущие нули группы необязательными.

> Таким образом, в записи группы ведущие нули вполне допустимы и не меняют интерпретации группы. Но, строго говоря, в текстовом формате адреса IPv6 их число не может быть произвольным, так как группа не должна содержать более четырех шестнадцатеричных цифр. Это ограничение подчеркивает тот факт, что одна группа — это всегда 16 бит адреса.

После этой простой поправки тот же самый адрес можно записать заметно короче:

```
2001:DB8:0:0:0:1:0:6789
```

Теперь мы замечаем, что в этом адресе подряд идут три нулевых группы. А что будет, если мы их вообще опустим? Мы точно знаем, что в полной записи адреса IPv6 восемь групп; благодаря этому мы можем элементарно вычислить длину пропущенной цепочки нулей. Например, давайте опустим нулевые группы, идущие подряд в нашем образцовом адресе:

```
2001:DB8::1:0:6789
```

Так как всего групп должно быть восемь, а в записи осталось только пять, то на месте пропуска должны быть недостающие три нулевых группы. Позицию же пропуска однозначно показывает пара знаков двоеточия.

При необходимости пара двоеточий может находиться не только в середине, но также и в начале или в конце адреса. Например, следующие две записи эквивалентны между собой — но отличны от *2001:DB8::1:0:6789*:

```
2001:DB8:1:0:6789::
2001:DB8:1:0:6789:0:0:0
```

А что будет, если мы сделаем больше одного пропуска? Например, так:

```
~~2001:DB8::1::6789~~
```

Попробуем восстановить полный адрес. Мы легко видим, что пропущено четыре группы, но мы не знаем, как они распределены между пропусками: одна и три, две и две, три и одна. Эта неоднозначность мешает нам восстановить полный адрес. Так мы убеждаемся, что в сокращенной записи допустим только один пропуск, и никак не более.

Между тем, опустить можно любую цепочку нулей, а не только самую длинную. К примеру, наш образцовый адрес *2001:DB8::1:0:6789* можно записать и так:

```
2001:DB8:0:0:0:1::6789
```

Как нетрудно видеть, теперь опущена всего одна нулевая группа, потому что остальные семь записаны явно.

> Только что составленное нами правило сокращенной записи позволит легче запоминать адреса IPv6. Если вам пожалуются на то, что адрес IPv6 невозможно запомнить, расскажите собеседнику об этом правиле.

> Некоторые парсеры формата «десятичный с точками» IPv4 допускали сокращенную запись адресов на входе. Наиболее известный, а возможно, и единственный случай такого поведения — это функции *inet(3)* из библиотеки *BSD libc*, которые позволяли, к примеру, сократить запись адреса *10.0.0.1* до `10.1`, а *192.168.0.1* до `192.168.1`. Правила такого сокращения были сложнее, чем простой пропуск нулевых октетов — см. соответствующую страницу руководства *man* [20]. Однако все подобные расширения нотации IPv4 были нестандартными и зависели от реализации, в отличие от *общепринятых* правил записи адресов IPv6.

> Проверьте себя, ответив на такой вопрос: какое максимальное число двоеточий может содержать запись адреса IPv6, отвечающая стандарту? (Ответ: 8. Пример можно найти в [§2.7.1 RFC 4291].)

[20] http://www.freebsd.org/cgi/man.cgi?query=inet&apropos=0&sektion=3&manpath=4.4BSD+Lite2&arch=default&format=html

Помимо полных, 128-битных адресов, в ряде случаев нам понадобится запись префиксов IPv6. Так, к примеру, префиксами будут представлены записи в таблицах маршрутов или просто блоки адресов, выделенные для каких-то целей. По своей сути, префикс — это цепочка битов, длина которой не больше, чем длина адреса. Как мы знаем, одна такая цепочка обозначает все адреса, старшие разряды которых совпадают с этой цепочкой. Из-за их тесной связи с адресами, префиксы удобно записывать в том же формате, только с явным указанием длины. Пусть десятичная длина префикса пишется после адреса через косую черту, как и в нотации IPv4 CIDR [§3.1 RFC 4632].

Допустим, нам дан такой двоичный префикс длиной 58 бит:

```
00100000000000010000110110111000000000000000000001100110110
```

Чтобы превратить его в полный, 128-битный адрес IPv6, который мы должны записать перед косой чертой, этот префикс надо дополнить 70 нулевыми битами *справа* — ведь префикс означает старшие биты адреса. Шестнадцатеричное значение искомого адреса таково:

```
20010DB80000CD8000000000000000000
```

Записав этот адрес по недавно сформулированным правилам, мы ставим косую черту и пишем длину, 58. Вот что у нас получается:

```
2001:0DB8:0000:CD80:0000:0000:0000:0000/58
```

Это и есть текстовое представление данного префикса. А теперь мы можем переписать адрес IPv6 лаконичнее и получить краткий, но полностью эквивалентный вариант:

```
2001:DB8:0:CD80::/58
```

Наконец, иногда удобно одновременно записать адрес IP и его префикс. (Главное применение этой записи мы встретим при назначении адреса сетевому интерфейсу.) По определению, префикс уже содержится в старших разрядах адреса, так что достаточно дополнить адрес длиной префикса, применив ту же самую нотацию. Например, запись:

```
2001:DB8:0:CD9F::123/58
```

означает адрес *2001:DB8:0:CD9F::123* в контексте префикса *2001:DB8:0:CD80::/58*.

> Убедитесь, что адрес *2001:DB8:0:CD9F::123* действительно содержит префикс *2001:DB8:0:CD80::/58*.

Отличить запись адреса с префиксом от записи чистого префикса в общем случае можно только по контексту. Конечно, в записях адресов довольно часто встречается ненулевой остаток (разряды справа от префикса), а в правильных записях префиксов его нет никогда; однако адрес, в котором остаток нулевой, тоже имеет право на существование.

Типичная ошибка, которую можно допустить, записывая префикс, — это отбросить в последней значащей группе концевые нули [§2.3 RFC 4291]. Например, следующие две записи *не эквивалентны*:

```
2001:DB8:0:CD80::/60
2001:DB8:0:CD8::/60
```

Чтобы убедиться в этом, достаточно преобразовать каждую из текстовых записей обратно в двоичное или шестнадцатеричное представление префикса и остатка, как показано в Табл. 1, и сравнить непосредственно их.

Табл. 1. Развернутое представление префиксов IPv6

Текстовая нотация	Префикс (по основанию 16)	Остаток (по основанию 16)
2001:DB8:0:CD80::/60	20010DB80000CD8	0000000000000000
2001:DB8:0:CD8::/60	20010DB800000CD	8000000000000000

Более краткое шестнадцатеричное представление применимо, когда длина префикса кратна четырем. В этом случае вместо каждых четырех двоичных разрядов мы можем записать один шестнадцатеричный знак. В противном случае мы рискуем потерять информацию о длине префикса.

«Составленные» нами правила записи адресов и префиксов IPv6 зафиксированы в §2.2 и §2.3 RFC 4291. А вот что RFC 4291 забывает сказать, так это что текстовая нотация IPv6 нечувствительна к регистру: знаки-буквы могут быть как прописными (`A`–`F`), так и строчными (`a`–`f`) [§2.3 RFC 5952]. Это обычная практика шестнадцатеричной нотации.

Читая RFC 5952, имейте в виду, что речь идет только о формате адреса IPv6 *на выходе*, то есть в выводе программ. Авторы стандарта делают на этом слабое ударение, из-за чего §4.3 можно трактовать так, что заглавные буквы A–F в текстовой нотации IPv6 вообще запрещены. Не стоит удивляться, если завтра появится приложение или библиотека IPv6, которые откажутся принимать заглавные буквы *на входе*.

Существует еще одна нотация, в которой младшие 4 байта адреса IPv6 записаны как адрес IPv4 [п. 3 §2.2 RFC 4291]. Например, наш первый адрес IPv6 будет записан как *2001:DB8::1:0.0.103.137*. Она может быть полезна во время перехода, когда в сети сосуществуют IPv4 и IPv6, а некоторые адреса IPv6 содержат в себе адреса IPv4.

2.3. Типы адресов IPv6

Теперь мы можем заняться первичной «раскройкой» адресного пространства IPv6. Какие типы адресов нам понадобятся? Основываясь на нашем опыте с IPv4, мы составим такой пробный список:

- глобальные индивидуальные адреса (*global unicast*);
- частные индивидуальные адреса (*private-use unicast*);
- внутриканальные индивидуальные адреса (*link-local unicast*);
- групповые адреса (*multicast*);
- адрес обратной связи (*loopback*);
- неопределенный адрес (*unspecified*).

Начнем движение с конца этого списка, поскольку там находятся более простые типы, отвечающие меньшим множествам адресов.

Неопределенный адрес (это был *0.0.0.0* в IPv4) означает, что адрес не установлен, не настроен, неизвестен. Например, в API сокетов Беркли приложение использует неопределенный локальный адрес сокета, чтобы сказать сетевому стеку: выбери подходящий адрес самостоятельно. Очевидно, что неопределенный адрес нельзя назначать интерфейсам или помещать в заголовок IP, кроме нескольких особо оговоренных случаев. Например, в самом начале автоматической настройки адресов узла адрес источника пакета может быть неопределенным, чтобы обойти «проблему курицы и яйца», когда узлу надо послать пакет, хотя у него еще нет собственных адресов — мы об этом еще поговорим. Напротив, адрес назначения пакета неопределенным быть не должен никогда, потому что иначе в нем не будет никакого смысла: это все равно, что пустое поле «куда/кому» на почтовом конверте. Поскольку неопределенный адрес играет довольно важную роль, его значение должно быть общепринятым и уникальным; нам надо утвердить его с самого начала. Пусть в IPv6 неопределенным тоже будет адрес с нулевым численным значением:

```
0:0:0:0:0:0:0:0
```

Благодаря правилам сокращения из §2.2, его можно записать и проще, всего парой двоеточий:

```
::
```

Адрес обратной связи необходим, чтобы сетевые приложения TCP/IP могли работать и на отдельной машине, у которой нет настоящих сетевых интерфейсов. Кроме того, удобно, когда один и тот же общепринятый адрес всегда указывает на локальный узел: меньше придется менять настройки сетевых приложений при их переносе с одного узла на другой, если части одного приложения тоже общаются по TCP/IP. Очевидно, что адрес обратной связи никогда не должен появляться в заголовках пакетов, путешествующих между узлами, потому что он имеет смысл только в пределах одного узла. В IPv4 для этой цели зарезервировали большой блок, *127.0.0.0/8*; но, как показал опыт, достаточно было бы одного адреса *127.0.0.1*. Поэтому в IPv6 мы ограничимся ровно одним адресом обратной связи. Ради простоты и краткости пусть это будет адрес с численным значением 1:

```
::1
```

Групповым адресам стоит назначить уникальный префикс, чтобы по одному виду адреса можно было определить, групповой ли он. Ведь правила обработки индивидуальных и групповых пакетов существенно различаются. Пусть у всех групповых адресов IPv6, и только у них, будет двоичный префикс `1111 1111`. В текстовой нотации мы запишем его так:

```
FF00::/8
```

> Обратите внимание, что эту запись нельзя сократить до `FF::/8`. Если вам это не очевидно, повторите правила записи адресов IPv6 из §2.2.

Теперь мы можем перейти к индивидуальным адресам (*unicast*). Наша главная цель — обеспечить *Internet* максимальным числом глобальных индивидуальных адресов; поэтому мы выделим в доступном адресном пространстве относительно небольшие множества адресов, не являющихся глобальными, а все остальные адреса пусть по умолчанию будут глобальными. Так мы максимизируем число последних.

Внутриканальные адреса нам знакомы еще по IPv4. Тогда это был блок *169.254.0.0/16*, который полагалось использовать при автоматической настройке узлов в простой одноранговой сети [RFC 3927]. Пакеты с адресами из этого блока не должны покидать пределов одного канала и подключенных к нему узлов. Задача автоматической настройки наверняка возникнет и после перехода на IPv6, так что давайте зарезервируем для внутриканальных адресов такой префикс:

```
FE80::/10
```

> Хотя по нашему генеральному плану длина префикса подсети составит 64 бита, это будет решение на уровне политики, а не протокола. Поэтому сейчас мы резервируем для внутриканальных адресов не /64, а блок существенно большего размера, и это не противоречит сказанному ранее.

Частные адреса IPv4 были нужны для развертывания сетей типа интранет, когда по своей технологии и архитектуре сеть устроена точно так же, как *Internet*, но физически отделена от него. Если бы было точно известно, что такой интранет никогда не станет частью *Internet*, блок адресов для него можно было бы выбрать совершенно произвольный, вплоть до *0.0.0.0/0*. Но в действительности интранет рано или поздно соединяют с *Internet* через шлюз, и тогда возникает неоднозначность адресов: один адрес А оказывается назначен как внутри, в пределах интранета, так и снаружи, в *Internet*. После этого другие узлы интранета не могут быть уверены, куда они попадут, обращаясь по адресу А. Избежать этого сценария позволили четко оговоренные частные адреса [RFC 1918], которые никогда не применяются в роли глобальных и потому не встречаются в *Internet*. Этот подход не лишен недостатков, которые мы обсудим в §2.10. Тем не менее, он решает задачу уникальности адресов с точки зрения интранета. Сам термин «интранет» в большой мере устарел, так как сетей, полностью отделенных от *Internet*, осталось мало, а подавляющее их число просто подключено через шлюз, скрывающий частные адреса при помощи NAT или proxy. Поэтому примем на вооружение вместо него термин «**сайт**».

Типичный пример сайта — это организация или ее часть, всей сетью которой управляет один отдел.

> Когда в организации несколько сайтов, по-русски мы бы называли каждый из них просто площадкой, но термин «сайт» уже вошел в обиход.

Соответственно, в IPv6 вместо частных адресов возникают **внутрисайтовые адреса** (*site-local address*), которые значимы только в пределах данного сайта и не могут встречаться в глобальном пространстве *Internet*. Для них мы зарезервируем префикс, смежный предыдущему:

```
FEC0::/10
```

> В конечном итоге недостатки внутрисайтовых адресов перевесили их преимущества, так что на сегодняшний день рекомендовано отказаться от их применения [RFC 3879]. Мы обсудим в §2.10, что именно привело к этому решению и чем мы располагаем взамен. Тем временем, как это ни странно, понятие внутрисайтовых адресов поможет нам разобраться с темой §2.4.

Наконец, все остальные адреса IPv6 относятся к типу глобальных индивидуальных. Ну, или практически все из них.

> На самом деле, есть еще пара исключений, которые мы подробно рассматривать не будем. Адреса IPv6 с префиксом *::/96* содержат в себе адреса IPv4, но только глобальные индивидуальные [§2.5.5.1 RFC 4291], поэтому и внешний адрес IPv6 в известном смысле остается глобальным индивидуальным. Адреса IPv6 с префиксом *::FFFF:0:0/96* могут содержать в себе любой адрес IPv4 [§2.5.5.2 RFC 4291]. На практике префикс *::/96* больше не применяется, а префикс *::FFFF:0:0/96* используется только в API, чтобы «подружить» приложения IPv6 с сетевой средой IPv4 [RFC 4038]. Например, приложение без поддержки IPv4 все-таки можно заставить соединиться с хостом *192.0.2.1*, подсунув ему такой адрес IPv6: *::FFFF:192.0.2.1*. (О нотации см. примечание в конце §2.2.) Конечно, сам сетевой стек должен при этом поддерживать оба протокола.
>
> Пока что для применения в *Internet* раздают блок *2000::/3* [[21], [22]]. Остальные глобальные адреса IPv6 зарезервированы на случай, если этот блок исчерпается слишком скоро и надо будет пересмотреть политику выдачи адресов IPv6.

Среди глобальных индивидуальных адресов IPv6 нам следует сразу же зарезервировать диапазон для примеров. Его назначение понятно: даже если читатель бездумно введет адреса из примеров в конфигурацию живой сети, это не приведет к глобальному конфликту. Как мы помним, в IPv4 эту роль играли блоки *192.0.2.0/24*, *198.51.100.0/24* и *203.0.113.0/24* [RFC 5735, RFC 5737]. Несколько штук их понадобилось, чтобы было удобнее составлять наглядные примеры взаимодействия разных сетей. Теперь же достаточно выделить один блок побольше, и авторы сами раскроят его по своему усмотрению. Пускай это будет *2001:DB8::/32* [RFC 3849]. Мы уже начали использовать его, говоря о текстовой нотации адреса IPv6.

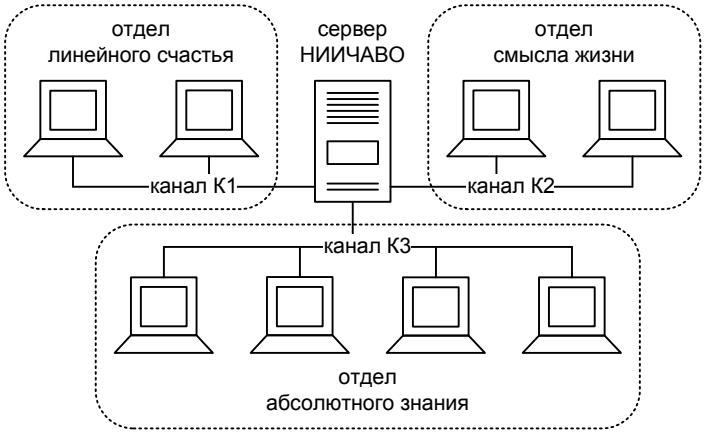
> Блок *192.0.2.0/24* был назначен для этой цели на высочайшем уровне, то есть IANA. Блоки *198.51.100.0/24*, *203.0.113.0/24* и *2001:DB8::/32* выделены региональной регистратурой Азии и Океании APNIC.
>
> Как мы помним, у IPv4 был сводный документ, посвященный адресам особого назначения [RFC 5735]. Аналогичный документ есть и у IPv6 [RFC 5156].
>
> В *Internet* уже опубликовано немало технической документации, где для иллюстраций выбран префикс IPv6 *2001:DB80::/32*. Обсудите правомерность и удачность такого выбора.

[21] http://www.iana.org/assignments/ipv6-address-space
[22] http://www.iana.org/assignments/ipv6-unicast-address-assignments

2.4. Область и зона действия адреса IPv6

Затронутая нами в §2.3 концепция внутриканальных адресов подводит нас к одной любопытной и довольно общей проблеме. Пока что мы знакомы только с тем, как эта концепция реализована в IPv4, и реализация эта предполагает, что у каждого узла ровно один активный сетевой интерфейс [§3 RFC 3927]. Это позволяет объединить все узлы многоадресным каналом и обойтись одним префиксом, *169.254.0.0/16*, без дальнейшего его дробления. Есть ли у нас основания для того, чтобы расширить эту схему на случай N каналов, где $N > 1$?

Для затравки мы можем представить себе простейший сценарий: в центре сети находиться сервер, подключенный к нескольким каналам, а рабочие станции обращаются к нему по этим каналам. Примером послужит гипотетическая сеть НИИЧАВО [23], показанная на Фиг. 3. Если любая рабочая станция в такой сети обращается только к своим соседям (*neighbor*) по каналу, включая сервер, то для работы сети достаточно внутриканальных адресов. Каждый канал может отвечать, скажем, независимому отделу организации; тогда подсети отделов будут изолированы друг от друга, и маршрутизация пакетов IP между ними не потребуется.

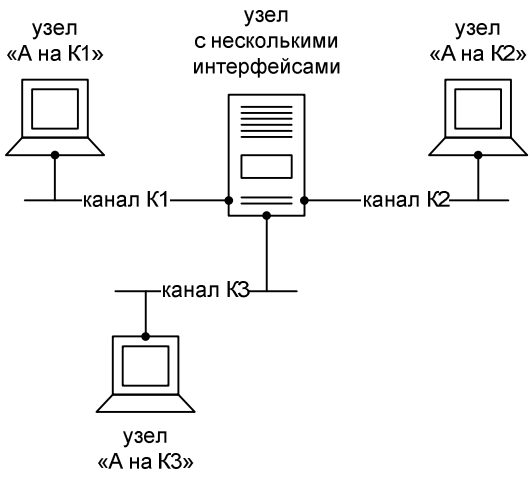


Фиг. 3. Гипотетическая сеть НИИЧАВО

Пока мы работаем в рамках IPv4, осуществить эту схему на практике мы не сможем, потому что нам пришлось бы назначить каждому сетевому интерфейсу сервера адрес из одной и той же подсети, *169.254.0.0/16*, а сетевой уровень IPv4 такой конфигурации не поддерживает по причине ее неоднозначности. А именно сетевой стек IPv4 не смог бы определить, через какой интерфейс надо передавать исходящие пакеты, адресованные *169.254.x.y*.

Если же мы выйдем за пределы IPv4 и рассмотрим задачу на чисто логическом уровне, то внутриканальные адреса разных каналов вполне могут быть независимы, поскольку область действия каждого из них ограничена его каналом. Техническая сложность здесь заключается в том, что узлу, подключенному к нескольким каналам, надо научиться отличать адрес A на канале К1 от адреса A на канале К2, как показано на Фиг. 4. То есть указатель на канал должен стать частью внутриканального адреса. Если это произойдет, то один и тот же префикс подсети можно будет назначить разным интерфейсам узла, и это не приведет к неоднозначности.

[23] А. и Б. Стругацкие. «Понедельник начинается в субботу»: Сказка для научных работников младшего возраста — М.: Детская литература, 1965. — 224 с, ил.



Фиг. 4. Разрешение неоднозначности адреса А указанием канала

Подобное решение давно применяют географы: городам можно давать одинаковые имена, если они стоят на разных реках. Чтобы не возникло путаницы, к собственному имени города добавляют название реки. Например, так появились Франкфурт-на-Майне и Франкфурт-на-Одере, Комсомольск-на-Амуре и Комсомольск-на-Днепре.

К сожалению, подобного механизма вообще не было в IPv4 — об этом подробно говорится в [§3 RFC 3927], — так что стек IPv4 не различал внутриканальные адреса или префиксы подсетей на разных каналах, если их двоичное значение совпадало. Хотя менять что-либо в поведении IPv4 уже поздно, нам ничто не мешает освободить IPv6 от этого ограничения.

Однако мы пойдем дальше по пути обобщения и не станем ограничивать эту интересную идею одними только каналами. Скажем так: пускай у *каждого* адреса IPv6 будет определенная **область действия**, или просто **область** (*scope*) [§4 RFC 4007]. Например, у внутриканального адреса это канал, у внутрисайтового — сайт, а у глобального — целая планета или даже вселенная (в зависимости от высоты наших звездных амбиций). Единственное исключение — это неопределенный адрес *::*, ввиду его особого статуса: область действия адреса *::* в общем случае не определена.

Если вам нужно освежить память насчет типов адресов IPv6, повторите §2.3.

Вернемся пока что к внутриканальным адресам и снова рассмотрим наш пример — сервер в центре организации (Фиг. 3). Допустим, всем его интерфейсам назначен один и тот же численный адрес *FE80::1/64*, область действия которого — канал.[24] Но, по нашему замыслу, адрес *FE80::1* в отделе линейного счастья (канал К1) и адрес *FE80::1* в отделе смысла жизни (канал К2) — суть *разные* адреса, несмотря на одинаковое численное значение и равновеликую область действия. Чтобы зафиксировать это различие, мы скажем, что канал К1 — это **зона действия** (*scope zone*), или просто **зона**, адреса «*FE80::1* на К1». То же самое можно сказать о любом адресе из подсети «*FE80::/64* на К1» — у всех у них зона совпадает. В то же время, у адресов «*FE80::1* на К2» и, например, «*FE80::C0DE* на К2» будет другая зона, а именно канал К2.

Чем зона отличается от области? Область — это характеристика величины, размера: один канал, один сайт, одна вселенная. А зона — это конкретная территория, где актуален данный адрес IPv6: канал К1 в Отделе линейного счастья НИИЧАВО или облако *Wi-Fi* у меня дома; академическая сеть НИИЧАВО или корпоративная сеть компании *Yoyodyne*; наконец, наша Вселенная.

[24] В §2.3 мы зарезервировали *FE80::/10* для внутриканальных адресов, помните?

Адепты объектно-ориентированного программирования без труда заметят такую параллель: область — это класс, а зона — это объект.

Особенность такого подхода в том, что область адреса IPv6 можно определить по его двоичному префиксу (*FE80::/10*,— канал, *FEC0::/10* — сайт, и т.д.), тогда как его зона зависит от конфигурации конкретной сети. Информация о зоне не содержится в численном значении адреса, а значит, ее надо хранить и сообщать дополнительно, например, как мы делали это выше, снабжая внутриканальный адрес уточнением: «на таком-то канале».

Означает ли это, что имя зоны становится неотъемлемой частью адреса IPv6? Придется ли поместить такие имена в заголовок IPv6 наряду с численными адресами источника и назначения? Чтобы ответить на этот вопрос, нам надо принять и понять простой факт: адресация между разными зонами одной области[25] невозможна, потому что адреса в их объединении не уникальны. Иными словами, если в заголовке пакета указан адрес источника или назначения, принадлежащий данной зоне, то пакет не должен покидать пределов этой зоны, потому что иначе адрес утратит свой смысл. То, что мы сейчас обнаружили — это фундаментальное свойство зонной архитектуры IPv6. Мы станем называть его «принцип изоляции зон».

Вообще-то, нам с самого начала было очевидно, что в нашем примере сети НИИЧАВО обмен трафиком между разными ЛВС невозможен, поскольку в них применяются только внутриканальные адреса. Сейчас же мы обобщили этот вывод на любую область.

Благодаря этому свойству узел-получатель всегда знает, из какой зоны пришел пакет. Например, если сервер НИИЧАВО получит из канала К2 пакет, созданный узлом *FE80::DADA* и адресованный *FE80::F00D*, то сервер немедленно отнесет адреса источника и назначения к зоне К2: они станут не просто *FE80::DADA* и *FE80::F00D*, а «*FE80::DADA* на канале К2» и «*FE80::F00D* на канале К2». Никакой дополнительной информации в заголовке самого пакета при этом не потребуется.

Неоднозначность может возникнуть при отправке пакета и состоит она в выборе выходного интерфейса, потому что посредством интерфейсов узел подключен к каналам, а через них и к зонам большей величины. Здесь возможны два случая:

1) данный узел — транзитный на пути данного пакета;
2) данный узел — создатель пакета.

В первом из них пакет был принят из какого-то интерфейса, а значит, узел уже отнес адреса источника и назначения в нем к определенным зонам, как мы только что говорили. Следовательно, на момент передачи пакета узел точно знает зону его назначения и может правильно выбрать выходной интерфейс. Скажем, если адрес назначения внутриканальный, то пакет может только вернуться в тот же канал, откуда он пришел — через тот же самый интерфейс или другой подключенный к тому же каналу.

И только во втором случае сведения о зоне назначения пакета заранее недоступны. Здесь нет другого входа, как явным образом указать зону назначения. Например, если администратор сети НИИЧАВО захочет проверить с помощью *ping* доступность хоста *FE80::Baa1* в недавно подключенном отделе научного атеизма (новый канал К4 — на схеме сети отсутствует), то ему придется указать в командной строке не только численный адрес, но и зону. Нам предстоит сконструировать стандартный механизм, который позволит это сделать.

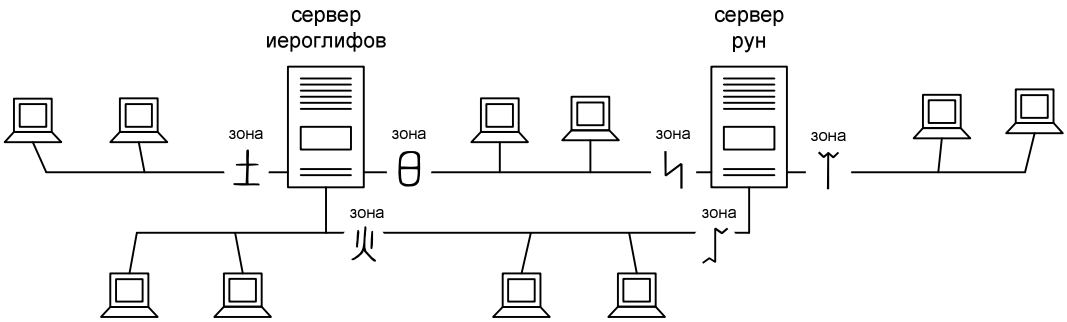
Выше мы рассмотрели все основные случаи, когда узел должен определить зону адреса: прием, транзит и создание пакета. В каждом из них узел располагал необходимыми сведениями *локально* и мог обойтись без внешней информации, явно сообщаемой другими узлами. На этом основании мы можем, наконец, сделать

[25]Или, говоря проще, одинаковой величины.

долгожданный вывод: указывать идентификаторы зон в заголовке IPv6 не нужно — достаточно численных адресов источника и назначения.

> Грубо говоря, включение имен зон в состав заголовка IPv6 означало бы просто отказ от зонной архитектуры и расширение адреса IPv6 еще несколькими битами.

Выходит, что идентификация зон — это частное дело отдельно взятого узла, и нет нужды заботиться о ее согласовании между узлами. Например, если бы в сети НИИЧАВО было два сервера, то один из них мог бы обозначать подключенные к нему каналы древнекитайскими иероглифами, а другой — древнегерманскими рунами, несмотря на то что у них есть общие каналы, как изображено на Фиг. 5. Главное, чтобы каждый узел сам не запутался в собственных обозначениях или, выражаясь строже, чтобы идентификатор зоны оставался уникальным в пределах узла.



Фиг. 5. Идентификаторы зон могут быть обозначены даже рунами или иероглифами

> Пусть читатель найдет эти иероглифы и руны в Unicode.

> На первый взгляд, мы сами себе противоречим. Ведь мы сказали в §2.2, что в текстовой записи адреса IPv6 допустимы только самые основные символы, а теперь говорим о каких-то рунах и иероглифах. Дело здесь в том, что идентификатор зоны, взятый на локальном узле, никогда не придется вводить в настройки другого узла, потому что там он не будет иметь никакого смысла. Именно поэтому локальный узел волен выбрать для идентификации зон любую знаковую систему, с которой он сам способен работать.

Как именно узел будет вести учет подключенных зон, мы оставим на усмотрение реализации. Тем не менее, можно предложить типовую модель этого механизма, чтобы не обсуждать воздушные замки. Итак, для того чтобы достоверно отличить разные зоны друг от друга, узлу достаточно:

а) знать область (величину) каждой из них: канал, сайт, …, вселенная;
б) пронумеровать зоны одной области (одинаковой величины).

Сочетание области и номера зоны мы обозначим как **индекс зоны** (*zone index*). Наши обозначения каналов в примере: «канал К1», «канал К2», «канал К3», — были ни чем иным как разновидностью таких индексов.

Теперь мы достаточно оснащены понятиями и терминами, чтобы двинуться дальше. Нашим первым важным решением было не ограничивать области IPv6 одним только каналом. Однако пока что наше слово расходится с делом: все наши примеры были основаны исключительно на каналах, а об областях большей величины мы практически не сказали ни слова. На самом деле, нам повезло, потому что, заранее не подумав, мы могли наговорить чепухи. Дело в том, что при переходе к областям большей величины возникают нетривиальные вопросы, на которые мы должны сперва ответить. Посыл здесь, на первый взгляд, прост: область большей величины содержит в себе одну или несколько областей меньшей величины. Например, сайт состоит из одного или больше каналов. Однако более строгий анализ вызывает, по меньшей мере, вот какие два вопроса:

- может ли область меньшей величины быть разделена между областями большей величины;[26]
- могут ли адрес источника и адрес назначения пакета IPv6 принадлежать разным зонам?

Хотя мы еще не работали над заголовком IPv6, уже сейчас достаточно ясно, что в нем будут адреса источника и назначения пакета.

Для ответа на первый вопрос рассмотрим гипотетический канал К, который принадлежит одновременно двум сайтам, С1 и С2. Это значит, что к каналу К могут быть одновременно подключены два узла У1 и У2 с внутрисайтовым адресом, скажем, *FEC0::C001*. Отличаться эти адреса будут, конечно же, зоной: у одного это «сайт С1», а у другого — «сайт С2». Теперь представим себе, что третий узел, принадлежащий сайту С1, передает в канал К пакет, адресованный *FEC0::C001*. Согласно нашей рабочей модели, идентификатор зоны назначения в этом пакете не упоминается. В результате, хотя этот пакет предназначался узлу У1 по принципу изоляции зон, его примут оба узла, У1 и У2, потому что каждый из них решит так: «Этот пакет пришел из канала К, и значит, он относится к моему сайту». В такой схеме адрес назначения теряет однозначность. Как восстановить ее?

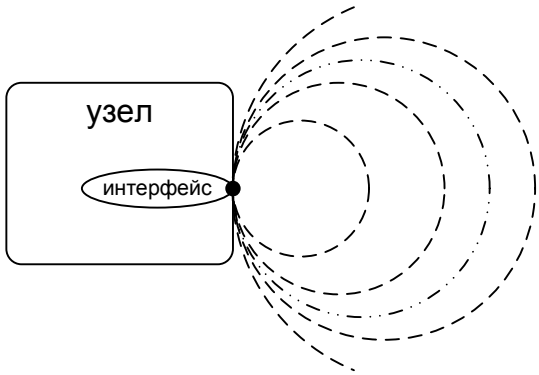
Для этого необходимо и достаточно, чтобы заголовок и входной интерфейс принятого пакета вместе однозначно определяли зоны адреса источника и адреса назначения [§7 и 8 RFC 4007]. Сам по себе численный адрес IPv6 из заголовка пакета указывает только на свою область, но не на зону. Значит, отображение области в зону надо провести на основе информации о входном интерфейсе. Чтобы это отображение оставалось однозначным и определенным для любой области, понадобится вот какое дополнительное условие: каждый интерфейс узла IPv6 находится ровно в одной зоне каждой области, независимо от назначенных ему адресов [§5 RFC 4007]. Предыдущий пример нарушал это требование, так как подключенные к каналу К интерфейсы узлов У1 и У2 находились одновременно в двух зонах области «сайт».

Да, сетевой интерфейс может находиться в зоне, даже не обладая ни одним адресом из этой зоны. Так, интерфейс, которому назначен только внутриканальный адрес, все равно находится в глобальной зоне и способен принимать и передавать глобальный трафик, например, если узел маршрутизирует транзитные пакеты.

Не ошиблись ли мы, сказав «ровно в одной зоне» вместо «не больше чем в одной зоне»? На самом деле, нет. Ведь иначе пострадала бы определенность соответствия: <численный адрес и входной интерфейс> → <зона адреса>, — так как некоторым адресам во входящих пакетах не отвечала бы никакая зона.

Обратите внимание, что мы заранее не знаем всех возможных областей. Можно полагать, что их множество не только бесконечно, но даже обладает свойствами континуума. Почувствовать это поможет иллюстрация Фиг. 6, где зоны разных областей обозначены касающимися пунктирными окружностями: между любыми двумя из них мы можем провести еще одну. Выражаясь философски, зона существует на интерфейсе независимо от того, помыслил ли кто-нибудь о ней.

[26] Тот же вопрос можно поставить так: могут ли зоны перекрываться?



Фиг. 6. Интерфейс принадлежит зонам всех областей

Последнее требование выполнимо, только если сетевая топология зон удовлетворяет ряду условий:

- границы зон проходят сквозь узлы, а не сквозь каналы;
- зоны никогда не перекрываются частично;
- зона большей величины может *полностью* заключать в себя зоны меньшей величины.

Зона не может включить в себя другую зону той же области, однако зоны разных областей вполне могут иметь одинаковую границу. Скажем, зона «корпоративная сеть компании *Yoyodyne*» может совпадать по границам с зоной «канал *Ethernet* компании *Yoyodyne*», если сеть компании *Yoyodyne* состоит из одной ЛВС.

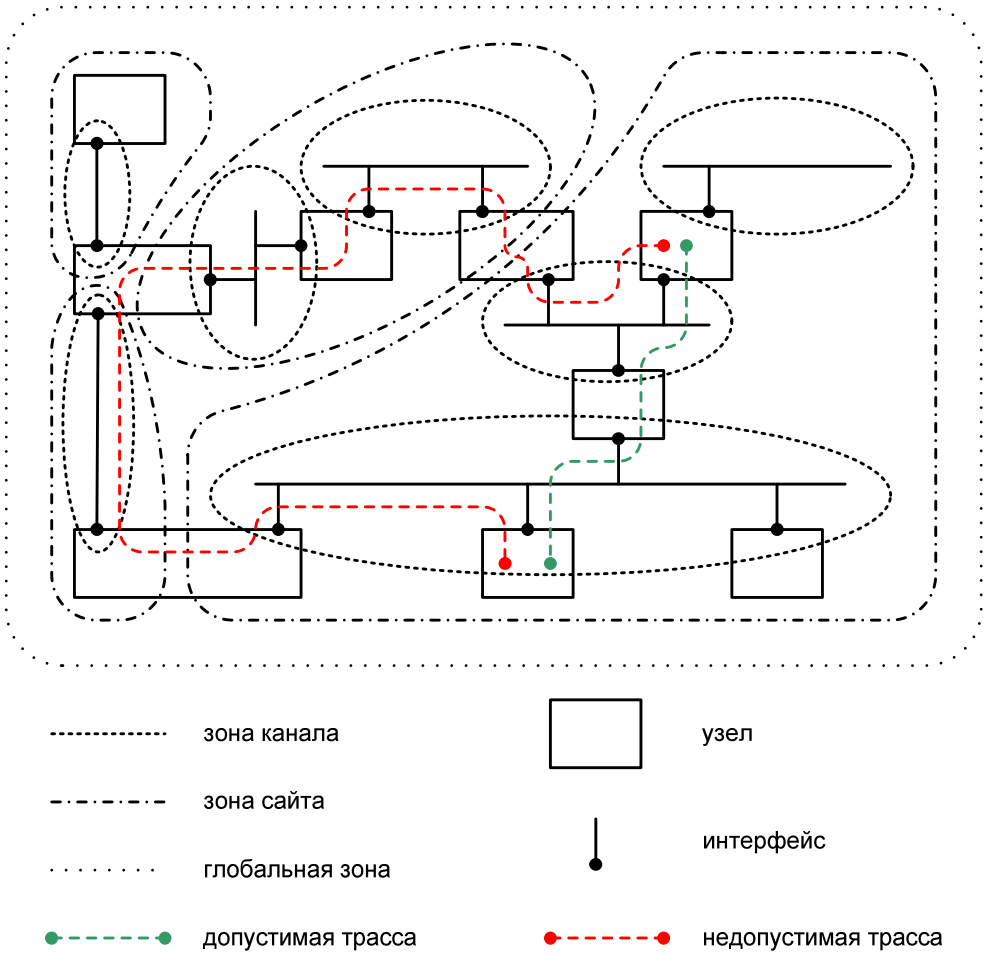
Любопытно отметить, что при такой топологии зона меньшей величины может одновременно принадлежать нескольким зонам большей величины, но это будут зоны разных областей. Двум зонам одной области она принадлежать не может. Так, например, канал может одновременно принадлежать сайту, государству и вселенной, но двум государствам он принадлежать не может. Чтобы соединить два государства, понадобится пограничный узел между ними, или же буферный канал, не принадлежащий ни одному из них.

Вдобавок, еще одно топологическое условие нам дает принцип изоляции зон:

- зона должна быть связной с точки зрения маршрутизации пакетов, то есть трасса пакета, адрес источника или назначения которого принадлежит данной зоне, должна полностью лежать внутри этой зоны.

Физическая схема сети вполне может допускать разные трассы между парой интерфейсов. Не исключено, что некоторые из них нарушат это правило, если пакеты будут путешествовать по ним. Задача администратора сети — исключить подобные трассы-нарушители при настройке маршрутизации. Например, не следует маршрутизировать свой внутрисайтовый трафик через соседнюю организацию, как показано на Фиг. 7. В свою очередь, сетевой стек IPv6 должен блокировать такие нарушения и не пропускать пакет-нарушитель из одной зоны в другую.

Чтобы лучше понять все эти условия, рассмотрите пример сложной карты зон на Фиг. 7 и установите, как эта карта выполняет каждое из них.



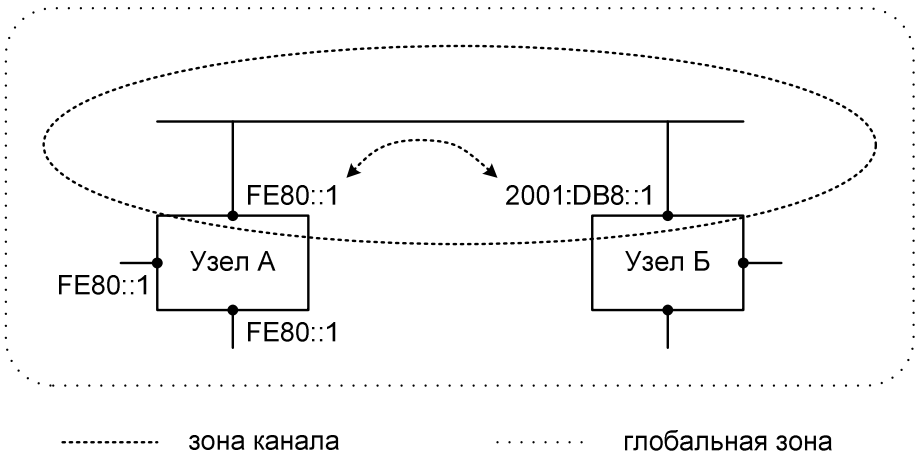
Фиг. 7. Пример топологии зон. Допустимая и недопустимая внутрисайтовые трассы в нем

Разобравшись с топологией зон, мы можем ответить на наш второй вопрос, о взаимоотношении зон адреса источника и адреса назначения пакета IPv6. Хотя, на первый взгляд, они обязаны совпасть, на самом деле принцип изоляции зон этого не требует. Необходимым и достаточным условием будет, чтобы *интерфейс* источника был в зоне адреса назначения, а *интерфейс* назначения — в зоне адреса источника.

> В то же время благодаря самому устройству зонной архитектуры интерфейс источника автоматически находится в зоне адреса источника, а интерфейс назначения — в зоне адреса назначения.

Это можно сформулировать и по-другому. «Дальность полета» пакета IPv6 ограничена двумя зонами, адреса источника и адреса назначения. Топология этих зон такова, что либо их границы совпадают, либо одна зона заключена внутри другой. Очевидно, что в последнем случае более сильное ограничение накладывает зона меньшей величины. Именно она и будет лимитирующей. Следовательно, ей должны принадлежать интерфейсы источника и назначения, чтобы пакет благополучно «долетел».

Например, на Фиг. 8 узлы А и Б соединены каналом. Поэтому узел А может направить узлу Б пакет, выбрав внутриканальный адрес источника и глобальный адрес назначения. Благодаря топологии зон IPv6 это не вызовет неоднозначности, и узел Б, зная входной интерфейс пакета, сможет ответить на него, используя глобальный адрес источника и внутриканальный адрес назначения. В свою очередь, и узел А отнесет адреса в пакете-ответе к их зонам на основании входного интерфейса.



Фиг. 8. Обмен данными между адресами разных областей

На пути от интерфейса источника к интерфейсу назначения пакету могут встретиться маршрутизаторы. Каковы будут правила работы с зонами для них? Определить зону адреса источника и адреса назначения маршрутизатор может по тому же принципу: численный адрес указывает на область, а входной интерфейс уточняет ее до конкретной зоны. Или наоборот, входной интерфейс указывает на множество зон, все они разных областей, а численный адрес выбирает из этого множества одну зону по ее области.

> Обратите внимание, что именно *входной* интерфейс пакета уточняет как адрес источника, так и адрес назначения до определенной зоны.

Далее маршрутизатор находит пакету адрес следующего шага и выходной интерфейс, используя таблицу маршрутов и, возможно, какие-то дополнительные правила политики узла. В условиях зонирования выходной интерфейс больше не может быть произвольным — он должен удовлетворять принципу изоляции зон. То есть маршрутизатор не имеет права продвинуть пакет так, чтобы он покинул зону адреса источника или адреса назначения. И снова благодаря топологии зон выполнить это условие просто: для сохранения зон адреса источника и адреса назначения достаточно, чтобы найденный выходной интерфейс находился в них обеих одновременно.

Если маршрутизатор нашел несколько альтернативных маршрутов, но не все из них сохраняют зоны данного пакета, то разумно будет оставить в рабочем множестве только походящие маршруты, а остальные исключить из рассмотрения. Если же ни один из найденных маршрутов не сохраняет зоны пакета, то придется отбросить сам пакет.

> [§9 RFC 4007] предлагает, чтобы маршрутизатор использовал отдельную таблицу маршрутов для каждой подключенной к нему зоны. Это позволит с самого начала исключить из рассмотрения маршруты, нарушающие зону назначения.
>
> [§9 RFC 4007] предписывает действовать по шагам: сначала по адресу назначения пакета маршрутизатор находит подходящий маршрут, который не нарушит зону назначения, а затем он проверяет, сохранит ли этот маршрут зону источника. То есть зона источника не влияет на окончательный выбор маршрута из множества альтернатив. На практике можно построить сеть, в которой такой маршрутизатор станет отбрасывать пакеты, хотя у него будет возможность продвинуть их с сохранением обеих зон. Пусть читатель сделает это на бумаге в качестве упражнения, используя области «канал», «сайт» и «вселенная».

Наконец мы можем сказать, что поняли структуру областей и зон IPv6. Теперь пора выполнить наше обещание и дать администратору сети НИИЧАВО необходимые инструменты, чтобы провести *ping* узла (а точнее, его интерфейса) *FE80::Baa1* в отделе атеизма (канал К4). Какой информации не хватает в командной строке: «`ping6 fe80::baa1`»? Индекса зоны?

Чуть позже мы узнаем, что в IPv6 есть свой протокол управления, отличный от ICMP. Поэтому *ping* теперь называется `ping6`. Впрочем, без новой утилиты можно было и обойтись: выбор протокола могла бы выполнить одна и та же утилита `ping` по версии адреса назначения. К примеру, именно так поступает командный интерфейс *JunOS*, хотя его команда `ping` все равно вызывает за кулисами */sbin/ping* или */sbin/ping6* в зависимости от версии данного адреса.

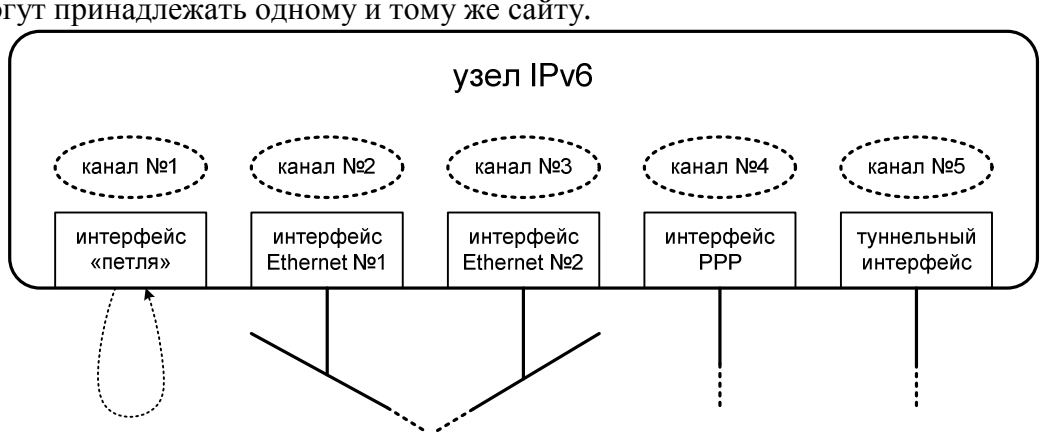
Как мы уже не раз говорили, численный адрес IPv6 сам указывает на свою область, в данном случае канал; индекс зоны тоже содержит в себе эти сведения. Выходит, что дополнить численный адрес индексом зоны было бы избыточно. Чтобы устранить эту избыточность, давайте еще раз просмотрим наши выкладки и решим такой ребус: чему равен полный зонный адрес IPv6 за вычетом адреса численного? Напрашивается такой ответ: он равен сетевому интерфейсу в данной зоне. Ведь именно знание входного интерфейса позволяет узлу дополнить численный адрес его зоной в принятом пакете. Поэтому кандидат №1 на роль удобного и понятного идентификатора, который расширит численный адрес IPv6 до зонного, — это системное имя сетевого интерфейса, подключенного к требуемой зоне. Например, если канал К4 подключен к серверу НИИЧАВО через интерфейс `en3`, то зонный адрес «*FE80::Baa1* на канале К4» достаточно записать в командной строке, например, так:

```
ping6 FE80::BAA1%en3
```

В этой нотации между численным адресом и именем интерфейса стоит символ процента, который реже других встречается в сетевом букваре.

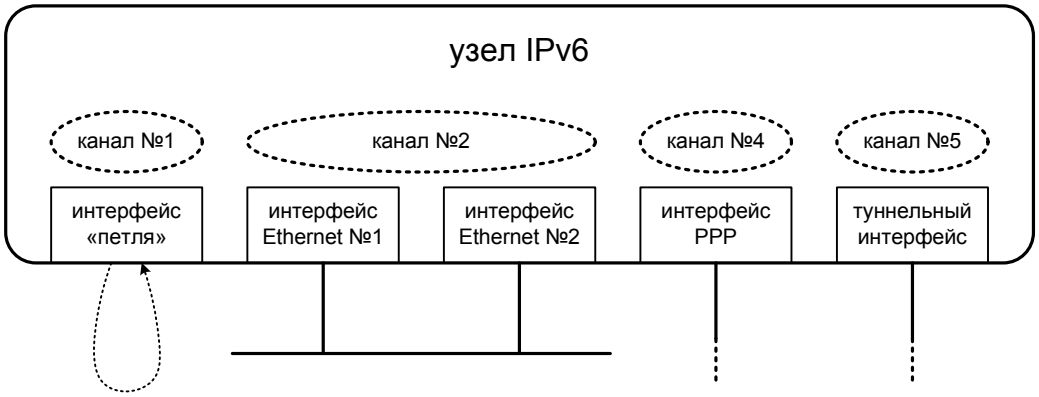
Следует признать, что этот выбор символа-разделителя не идеален. Так, процент — это метасимвол в формате URI, так что зонный адрес IPv6, строго говоря, нельзя указать в URI буквально. Работа над оптимальным решением этого противоречия еще не завершена [draft-fenner-literal-zone, draft-ietf-6man-uri-zoneid].

Но насколько безупречна эта конструкция? Ведь все наши предыдущие выкладки насчет топологии зон относились к входящим пакетам, а теперь мы вдруг перескочили к пакету исходящему! Когда пакет — входящий, то все соответствия однозначны: пакет входит через ровно один сетевой интерфейс,[27] который находится в ровно одной зоне каждой области. Но однозначность эта не взаимная, и поэтому ее нельзя просто так обратить для передачи: в одной зоне может находиться несколько интерфейсов данного узла. Например, он может быть подключен двумя интерфейсами к одному и тому же каналу в целях отказоустойчивости и распределения нагрузки. Наконец, все интерфейсы узла могут принадлежать одному и тому же сайту.



Фиг. 9. Начальная настройка узла: гипотеза взаимно однозначного соответствия между интерфейсами и внутриканальными зонами

[27] Возможно, в квантовых сетях будущего это будет не так.



Фиг. 10. Разрешение неоднозначности: два интерфейса подключены к одному каналу

Поэтому окончательную картину соответствия между интерфейсами и зонами способен построить только инженер-проектировщик данной сети. Имена же интерфейсов в системе дают лишь хорошее начальное приближение, которое система способна построить автоматически (см. Фиг. 9), так что администратору останется только скорректировать его, пользуясь картой сети [§6 RFC 4007], — например, как это изображено на Фиг. 10.

> Как следствие, сетевой стек IPv6 должен предоставить инструменты для подстройки соответствий между интерфейсами и зонами.

Чтобы отразить эту особенность в терминологии, мы дадим текстовому идентификатору зоны отдельное название: **zone_id** [§11.2 RFC 4007] Такой идентификатор чаще всего совпадает с системным именем одного из интерфейсов, потому что это удобно и понятно, но жесткой связи между ними нет. В результате мы получаем такую текстовую нотацию зонного адреса IPv6:

```
<численный_адрес>%<zone_id>
```

Если мы хотим записать зонный префикс, то его длину следует поместить, как и прежде, после адреса [§11.7 RFC 4007]:

```
<численный_адрес>%<zone_id>/<длина_префикса>
```

Вот пример сочетания внутриканального адреса и длины префикса в одной строке:

```
FE80::DADA:C001%en5/64
```

Поскольку системные имена интерфейсов в разных системах выглядят по-разному, то и формат *zone_id* у них разный. В Табл. 2 приведено несколько примеров, как была бы введена команда *ping6* по внутриканальному адресу хоста НИИЧАВО в разных ОС.

Табл. 2. *Ping* по внутриканальному адресу в разных ОС

ОС	Команда
FreeBSD	`ping6 FE80::Baa1%fxp3`
Linux	`ping6 FE80::Baa1%eth3`
Mac OS X	`ping6 FE80::Baa1%en3`
MS Windows	`ping6 FE80::Baa1%5`

Кстати, это еще одна иллюстрация того, как идентификация зон локальна для каждого узла.

Чтобы завершить нашу работу над зонной архитектурой IPv6, нам надо рассмотреть последний вопрос: всегда ли в настройках и командной строке адрес IPv6 придется дополнять *zone_id*? На самом деле, нет. Во-первых, среди всевозможных областей IPv6 есть две области с особыми свойствами:

- У адреса обратной связи *::1* область действия ограничена виртуальным интерфейсом типа «петля». Узлу нужен ровно один такой интерфейс, так что адрес обратной связи дополнять *zone_id* не надо. Можно сказать, что адрес *::1* — это особый случай внутриканального адреса, который

встречается только на виртуальном канале интерфейса «петля» [§6 RFC 4007].

- Глобальный адрес, такой как *2001:DB8::1*, наделен в космическую эпоху даже не глобальной, а вселенской областью действия, так что никаких дополнительных указателей он тоже не требует — по крайней мере, пока к *Internet* не подключатся параллельные вселенные, жители которых тоже изобрели IPv6.

> Здесь будет уместно замечание по поводу термина **зонный адрес** (*scoped address*), который встречается не только в нашем изложении, но и в документах RFC. Не следует понимать этот термин как «неглобальный адрес». Суть зонной адресной архитектуры IPv6 как раз в том, что у *всех* адресов кроме неопределенного есть область действия, просто у некоторых она глобальная. Поэтому «зонный адрес» — это адрес, который может и не быть глобальным, адрес произвольной области. Этот термин — памятка разработчику протокола или реализации, выросшему в среде IPv4, чтобы тот не забывал о существовании адресов IPv6, область действия которых меньше глобальной, и подвергал тщательному анализу детали своей разработки, связанные с этим необычным свойством адресации IPv6.

Кроме того, на практике удобно опустить *zone_id*, если эта информация избыточна. Например, какой смысл указывать его для внутрисайтового адреса, если все интерфейсы нашего узла принадлежат одному сайту, или для внутриканального адреса, если узел снабжен ровно одним интерфейсом помимо «петли»? Обобщить эту мысль можно, рекомендовав реализациям IPv6 поддержку зон по умолчанию, чтобы у каждой области была одна такая зона [§6 RFC 4007]. Тогда сетевой стек по численному значению адреса IPv6 без *zone_id* сможет «угадать» его зону.

> Обратное тоже верно: на практике может быть удобно снабдить определенным *zone_id* адрес, у которого вообще нет области, — неопределенный адрес. Как мы уже сказали, в API сокетов Беркли неопределенным адресом приложение говорит сетевому стеку: выбери адрес сам. Если же приложение снабдит неопределенный адрес определенным значением *zone_id*, это может означать выбор подходящего адреса в пределах данной зоны [§4 и §11.1 RFC 4007].

Хотя до сих пор мы говорили про области и зоны индивидуальных адресов IPv6, ограничить область действия мы можем и у группового адреса. Тогда, к примеру, мы сможем назначить общепринятый групповой адрес «все маршрутизаторы данного канала». Подробнее об областях групповых адресов мы поговорим ниже, в §2.8.

2.5. Общепринятые функции внутриканальных адресов

Мы с вами «построили» мощный механизм зонирования адресов IPv6. Найдутся ли у нас для него более важные и распространенные применения, нежели создание вложенных частных сетей? Зоны величиной больше канала, но меньше вселенной находятся в ведении отдельных администраторов, которые сами решают, для чего именно применять зонный механизм. Поэтому для стандартных, общепринятых функций остается только внутриканальная область; но, как мы сами сейчас убедимся, этого вполне достаточно.

> Напомним себе, что значит «общепринятый» в контексте TCP/IP. В *Internet* никогда не было стандартов в полном смысле этого слова, потому что никто никому их не навязывал и никто не следил за их соблюдением. Каждый сам решал, придерживаться ли RFC или нет, но тем, кто выдумывал свои собственные технические условия, не стоило удивляться, что их система ни с чем больше не совместима. Отсюда и появились *общепринятые* форматы данных, значения параметров и правила поведения сторон: они всего лишь известны и добровольно приняты теми, кто желает, чтобы их система могла взаимодействовать с остальным

миром. Хотя «стандартный» и «общепринятый» для нас синонимы, последний гораздо лучше передает техническую философию Сети.

Внутриканальная область выгодно отличается тем, что ее свойства заранее четко определены: это всегда ровно один канал. Каждый реально существующий канал представляет собой зону этой области, потому что зоны IPv6 существуют независимо от назначенных адресов. Какие фундаментальные аспекты работы сети IP ограничены одним каналом?

В первую очередь, это продвижение пакетов (*forwarding*). Согласно фундаментальной модели IP — а IPv6 наследует ее без изменений, — пакеты перемещаются между узлами по каналам. Поэтому элементарным актом перемещения пакета от источника к адресату выступает шаг (*hop*) — передача пакета другому узлу IP в пределах одного канала, — а адрес сетевого интерфейса-получателя называют адрес следующего шага (*next hop address*), таким образом подчеркивая, что этот шаг может быть не последним на пути пакета.

Эти азбучные истины TCP/IP мы повторили только ради «синхронизации» с читателем.

Очевидно, что адресу следующего шага *достаточно* быть внутриканальным, потому что его функция — указать на сетевой интерфейс в пределах одного канала. Адреса большей области тоже можно применять в этой роли, однако содержащаяся в них информация явно избыточна.

Если мы рассмотрим маршрутизатор как роль узла, то она сводится только к продвижению транзитных пакетов, тогда как созданием и поглощением пакетов ведает роль, известная нам как хост. Следовательно, для выполнения функции маршрутизатора узлу вполне достаточно внутриканальных адресов. Преимущество подобного подхода в том, что транзитная инфраструктура сети[28] вообще не зависит от адресов, которыми хосты пользуются для сквозной передачи данных между собой. К примеру, если организация переходит к другому провайдеру услуг *Internet* и получает новый диапазон глобальных адресов взамен старого, ей не придется менять адреса на интерфейсах своих маршрутизаторов. Чтобы поощрить эту практику, мы выдвинем такое требование: каждый интерфейс маршрутизатора IPv6 обязан обладать как минимум одним индивидуальным внутриканальным адресом [§2.3 RFC 4861].

Второе возможное применение внутриканальных адресов — это распространение сведений, необходимых для автоматической настройки хостов. Здесь внутриканальные адреса способны разрешить «проблему курицы и яйца», когда хост не может получить из сети информацию для автоматической настройки своего сетевого интерфейса до тех пор, пока он не назначил интерфейсу адрес. Эта тема заслуживает развернутого обсуждения, и мы к ней еще вернемся ниже, в §5.4; а пока что, немного забегая вперед, скажем: хост IPv6 будет обязан поддерживать механизм автоматической настройки, а для этого его сетевому интерфейсу понадобится внутриканальный адрес [RFC 4862].

Подведем итог: каждый сетевой интерфейс IPv6 обязан иметь по меньшей мере один индивидуальный внутриканальный адрес [§2.1 RFC 4291]. Помимо него, интерфейсу понадобятся адреса в зонах большей величины. Ведь это лишь частный случай назначения одному интерфейсу нескольких адресов — полезной практики, принятой еще во времена IPv4 [§3.3.4.1 RFC 1122], — и мы пока что не обнаружили никаких причин отказаться от нее. Поэтому пусть у сетевого интерфейса IPv6 может быть несколько индивидуальных адресов, области[29] и подсети которых могут как совпадать, так и быть разными [§2.1 RFC 4291].

[28] То, что раньше называли «подсеть», то есть сеть за вычетом хостов.

[29] Мы говорим «области», а не «зоны», потому что два адреса одного интерфейса никак не могут находиться в разных зонах одной области. И наоборот, если их области одинаковы, то и зона совпадает. Это следует из правил зонной топологии IPv6.

Говоря проще, у сетевого интерфейса IPv6 может быть много индивидуальных адресов, а у некоторых или всех из них вполне может совпадать область, или даже префикс подсети. Подобный вывод о групповых адресах последует из этого, когда в §5.1 мы доберемся до розыска соседей, так как тогда интерфейсу IPv6 потребуется вплоть до одного группового адреса на каждый индивидуальный.

У этого простого, на первый взгляд, решения о поддержке множества адресов на интерфейсе IPv6 есть далеко идущие последствия для других протоколов. Вот только один пример. OSPF для IPv6 (OSPFv3) оперирует целыми каналами [§2.1 RFC 5340], а не отдельными подсетями, как делал его предшественник OSPFv2. Когда на каждом канале целое множество подсетей, прежний подход был бы неэффективен.

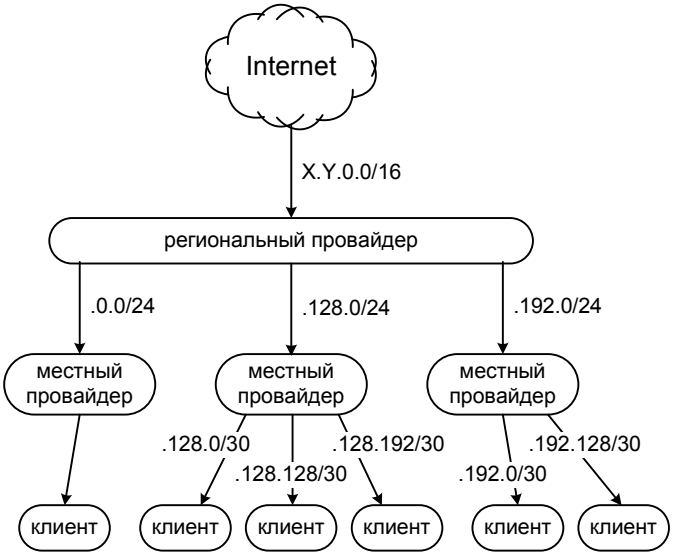
2.6. Структура индивидуального адреса IPv6

В IPv4 структуру индивидуального адреса определил процесс иерархического распределения префиксов: каждый уровень этой иерархии добавлял к префиксу несколько битов справа, а затем отдавал удлиненный префикс на более низкий уровень. В самом конце этого пути префикс назначали каналу, и он становился префиксом подсети. Поэтому, с точки зрения конечного участника, индивидуальный адрес IPv4 состоял из трех полей: назначенного «сверху» префикса, номера подсети и номера узла в пределах подсети; первые два поля вместе давали префикс подсети. Это касалось не только глобальных адресов: рекомендованные префиксы частных адресов IPv4 тоже были «спущены сверху», только не при посредстве провайдера или LIR, а непосредственно из рук IANA. Благодаря CIDR [RFC 4632], длина этих полей в адресе IPv4 не была фиксированной с момента отказа от маршрутизации по классам. Так, длина назначенного префикса зависела от «пути» по иерархии, приведшей к данному адресу; затем конечный владелец префикса делил остаток битов адреса по своему усмотрению, балансируя между тонким дроблением сети и экономией адресов, так как каждая подсеть «съедала» два ценных адреса IPv4.

У этой схемы есть, как минимум, два важных преимущества. Во-первых, такое распределение адресного пространства основано всего на одной элементарной операции, а именно на делении блока адресов пополам. Как нетрудно убедиться, именно это происходит, когда к данному префиксу добавляют справа один бит, а затем перебирают его возможные значения, то есть 0 и 1. Конечно, у этой процедуры есть известное ограничение: она работает только с блоками по 2^n адресов IPv4, имеющих общий префикс длины $32-n$; исходным блоком служит все адресное пространство IPv4, которому отвечает пустой префикс нулевой длины. Если бы блок был произвольным диапазоном адресов, то мы не смогли бы привести его к единому префиксу; поэтому он должен удовлетворять вышеуказанному условию. Именно это позволяет нам поставить знак эквивалентности между префиксами и блоками адресов.

Во-вторых, когда иерархия распределения адресов совпадает с иерархией маршрутизации, появляется возможность для агрегирования маршрутов. Допустим, к примеру, что региональный провайдер получил блок *X.Y.0.0/16*, а теперь делит его на блоки *X.Y.Z.0/24* и выдает их местным провайдерам, подключенным к нему. Те, в свою очередь, выдают каждому конечному клиенту блок /30. В результате только местный провайдер должен работать с мелкими маршрутами /30, указывающими на его собственных клиентов. Вышестоящий провайдер имеет дело с более крупными маршрутами *X.Y.Z.0/24*, каждый из которых указывает на одного местного провайдера, независимо от числа его клиентов. Наконец, в ядре *Internet* (зоне без умолчания) региональный провайдер представлен одним маршрутом *X.Y.0.0/16*, независимо от числа местных провайдеров и их клиентов. В идеале, это позволяет избежать экспоненциального

роста числа маршрутов при движении вверх по иерархии маршрутизации, как показано на Фиг. 11.



Фиг. 11. Агрегирование маршрутов IPv4 в *Internet*

Недостатки же этой схемы вызваны, по большей части, дефицитом адресов IPv4. С одной стороны, возможность выделить только 2^n адресов превращается в проблему, потому что точность порционирования кажется слишком низкой. С другой стороны, никакое деление доступных битов адреса IPv4 на номер подсети и номер узла не оптимально в растущей сети, потому что подсети все равно оказываются переполнены вопреки тщательному планированию, а расширить их обратным слиянием блоков нельзя, потому что смежный блок к тому времени уже занят.

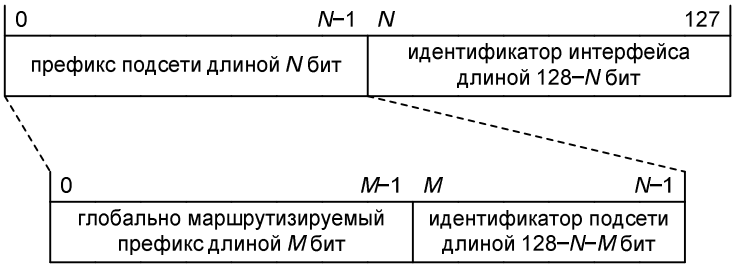
> У каждого блока адресов есть ровно один смежный блок. Это прямое следствие из распределения адресов делением блоков пополам. Чтобы найти префикс смежного блока, достаточно инвертировать младший бит текущего префикса. Например, двоичный префикс 110 смежен 111, а 1001 смежен 1000. Очевидно, что такое отношение смежности взаимно: если блок А смежен блоку Б, то блок Б смежен блоку А. Смежные блоки можно объединить и получить вдвое больший блок, просто отбросив младший бит префикса. Поскольку смежные префиксы отличаются только в младшем бите, то неважно, какой из них вы выберете для этой операции. А вот несмежные блоки вообще не объединяются.

Переходя к IPv6, мы получаем в распоряжение намного большее адресное пространство. Поэтому мы вправе допустить, что недостатки существующей структуры адреса больше не проявят себя. В то же время, положительные стороны этой структуры остаются весьма привлекательными. Поэтому разумно будет сохранить эту структуру адреса хотя бы в общем виде: назначенный префикс, номер подсети и номер узла. А о деталях мы сейчас поговорим.

Начнем мы с младших битов и посмотрим на ту часть, что в IPv4 была номером узла в подсети. В современной практике IP не только у маршрутизаторов, но и у хостов может быть несколько сетевых интерфейсов, в том числе и в одной подсети [§3.3.4 RFC 1122]. Например, это важно для подключения с резервированием. При этом у каждого из интерфейсов должен быть адрес, уникальный в пределах зоны. Если интерфейсы узла окажутся в одной подсети, то их адреса смогут отличаться только в младших разрядах, потому что префикс подсети у них будет одинаковый. То есть интерфейсам понадобятся разные номера узла, хотя физически узел один и тот же. Чтобы избавиться от этой путаницы, скажем так: номер в подсети принадлежит не всему узлу, а только одному из его интерфейсов. Поэтому самая младшая часть адреса IPv6 называется **идентификатор интерфейса** (*interface ID*) [§2.5 RFC 4291], а не номер узла. Соответственно, и вместо номера подсети в IPv6 говорят: **идентификатор подсети** (*subnet*

ID). Ну, а старшие биты адреса IPv6 занимает назначенный тем или иным образом префикс, например, глобально маршрутизируемый. Этот префикс и идентификатор подсети вместе образуют **префикс подсети** (*subnet prefix*). Такую структуру адреса иллюстрирует Фиг. 12.

А вот про маску сети (*netmask*) в IPv6 можно благополучно забыть. Архитектура IPv6 даже не пытается поддерживать «рваные» сетевые маски, в которых группы нулей и единиц перемежаются, так что достаточно указать число битов префикса вместо длинной побитовой маски.



Фиг. 12. Общая структура глобального индивидуального адреса IPv6

Теперь пришло время вспомнить, что в длину адреса IPv6 мы заложили длину идентификатора интерфейса, равную 64 битам. Соответственно, на префикс подсети тоже оставалось 64 бита. Тем не менее, это были просто размышления, не имеющие силу закона. Поэтому бывалый сетевой администратор, съевший собаку на IPv4, все еще может выбирать длину префикса подсети по старинке, исходя из личных прогнозов роста подсети. Но чем это плохо? Дело в том, что устаревший подход перечеркнет наши благие намерения, потому что подсети будут по-прежнему страдать от перенаселения узлами. Чтобы полностью исключить человеческий фактор и навсегда избавить IPv6 от перенаселения подсетей, мы в приказном порядке зафиксируем длины идентификатора интерфейса и префикса подсети на значении 64 бита [§2.5.1 RFC 4291], как показано на Фиг. 13.

Размер адресного пространства IPv6 настолько огромен, что мы можем позволить себе это расточительство. Взамен сетевые администраторы будут навсегда избавлены от сомнительных прогнозов роста подсетей. Да-да, и на канал «точка-точка» (*point-to-point*), в котором по определению всего два интерфейса, тоже надо назначить префикс /64. Если кто-то скажет, что это уже чересчур, у нас готов встречный вопрос: «А что, если завтра на месте канала «точка-точка» возникнет широковещательный канал со многими интерфейсами? Опять менять адреса?»

Типичный пример такого расширения канала — это внедрение в него межсетевых экранов и прочих систем сетевой безопасности.

Так мы отвязываем длину идентификатора интерфейса не только от планируемого размера подсети, но и от технологии канала. Позже мы убедимся, что фиксированная длина идентификатора интерфейса открывает нам и другие интересные возможности.

Фиг. 13. Фактический формат индивидуального адреса IPv6

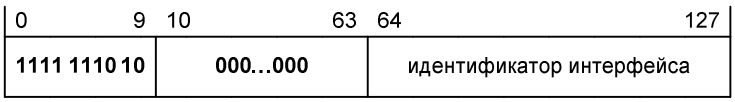
На сегодняшний день у правила «длина идентификатора интерфейса IPv6 равна 64 битам» сложился неоднозначный статус. Изначально это было правило политики в чистом виде. То есть технические элементы ядра IPv6 были разработаны так, чтобы не зависеть от этой длины. Она оставалась параметром *N*, который можно было установить индивидуально для каждой подсети, и только затем говорилось: а давайте положим *N* = 64 *для всех подсетей*. (Представьте себе механизм с колесом управления, которое опломбировано в определенном положении, хотя механизм остается в штатном режиме при любом положении колеса.) Тем не менее, позже возникли детали, которые работают только при

$N = 64$. (Как если бы в тот механизм добавили новый блок, которые немедленно сломается, если сорвать пломбу и повернуть колесо управления.) В частности, это групповые адреса UBM (§2.9), а также крипто-адреса CGA (§6.3) и HBA (§6.8), с которыми мы в свое время познакомимся.

С другой стороны, есть сторонники применения более длинных префиксов на каналах «точка-точка», вплоть до */127*, — чему отвечает $N = 1$, — и их аргументы тоже весьма убедительны [RFC 6164]. Поэтому вопрос, в каких случаях и насколько жестко надо ограничивать длину префикса подсети IPv6, остается предметом исследования. Это вполне естественно для развивающегося протокола. А мы тем временем не погрешим против реалий IPv6, приняв $N = 64$.

Небольшая оговорка, которую мы должны сделать, — это что среди всех индивидуальных адресов IPv6 есть один префикс, зарезервированный для особых целей. Адреса в нем не обязаны содержать 64-битный идентификатор интерфейса. Этот префикс — `000`, или же *::/3* в текстовой нотации IPv6 [§2.5.4 RFC 4291].

Что же касается внутриканальных адресов, то для них мы исключения делать не будем: каждый из них содержит 64-битный идентификатор интерфейса. А каково значение оставшихся 54 битов, идентификатора подсети? Пусть, как показано на Фиг. 14, это будет ноль. Остальные значения этого поля во внутриканальных адресах зарезервированы [§2.5.6 RFC 4291]. Таким образом, на практике внутриканальные адреса ограничены подсетью *FE80::/64*.



Фиг. 14. Формат внутриканального адреса IPv6

2.7. Идентификатор интерфейса и EUI-64

Фиксируя длину идентификатора интерфейса, мы подразумевали, что теперь это поле сможет вместить в себя канальный адрес в формате EUI-64. Это было бы в высшей степени удобно, ведь адреса EUI-64 призваны быть глобально уникальными — по крайней мере, пока они получены у *IEEE*. Например, чтобы автоматически назначить интерфейсу уникальный внутриканальный адрес,[30] в первом приближении будет достаточно соединить префикс *FE80::/64*. и адрес EUI-64 этого интерфейса. Например, интерфейс с адресом EUI-64 *00-11-22-33-44-55-66-77* получил бы, по такой предварительной схеме, внутриканальный адрес *FE80::11:2233:4455:6677*.

Однако не все канальные адреса — EUI-64. Как быть, если канальный адрес интерфейса отвечает другому стандарту? Это тоже не беда, так как вполне можно сформулировать правило отображения таких адресов в EUI-64. Пока в исходном адресе меньше 64 битов, это отображение может быть инъективным; то есть разные исходные адреса могут быть отображены в разные адреса EUI-64. Простейший и наиболее удобный для нас вид такого отображения — это инкапсуляция исходного адреса в EUI-64. К примеру, ее правила уже заданы для адресов EUI-48 и MAC-48 [[31]]. Они очень просты — см. Табл. 3.

Табл. 3. Отображение EUI-48 и MAC-48 в EUI-64

Тип адреса	Исходный вид	Отображение в EUI-64
EUI-48	*xx-yy-zz-ww-vv-tt*	*xx-yy-zz*-FF-FE-*ww-vv-tt*
MAC-48	*xx-yy-zz-ww-vv-tt*	*xx-yy-zz*-FF-FF-*ww-vv-tt*

[30] Разумеется, ему достаточно уникальности в пределах канала.

[31] Guidelines for 64-bit Global Identifier (EUI-64™) Registration Authority. IEEE. http://standards.ieee.org/regauth/oui/tutorials/EUI64.html

То есть между OUI и добавочным идентификатором возникают еще два байта с определенными значениями: `FF-FE` в случае EUI-48 и `FF-FF` в случае MAC-48. Например, из EUI-48 *02-35-AB-00-64-C1* получится такой EUI-64: *02-35-AB-FF-FE-00-64-C1* (добавочные байты подчеркнуты).

> По этой причине *IEEE* не разрешает производителям оборудования расходовать добавочные идентификаторы EUI-64, которые начинаются с `FF-FE` и `FF-FF`.

Постойте, а в чем состоит разница между EUI-48 и MAC-48? Пока мы не запутались в этом вопросе, давайте изучим его современную интерпретацию, потому что она нам сейчас же понадобится. Оба типа адресов находятся в ведении IEEE. По первоначальной задумке, MAC-48 служили канальными адресами IEEE 802, а EUI-48 — идентификаторами в прочих технологиях. Вначале *IEEE* даже допускал, что пространства этих адресов могут быть разными, несмотря на одинаковый формат.

> Циник заметил бы, что *IEEE* хотел продать одни и те же байты дважды.

Однако сегодня из документов *IEEE* однозначно следует, что MAC-48 — это устаревший термин для подмножества EUI-48 [[32]]. Так что мы вправе говорить «EUI-48», подразумевая «MAC-48 и Все-все-все». Кроме того, нам можно забыть о правиле преобразования MAC-48 в EUI-64 и пользоваться только правилом для EUI-48.

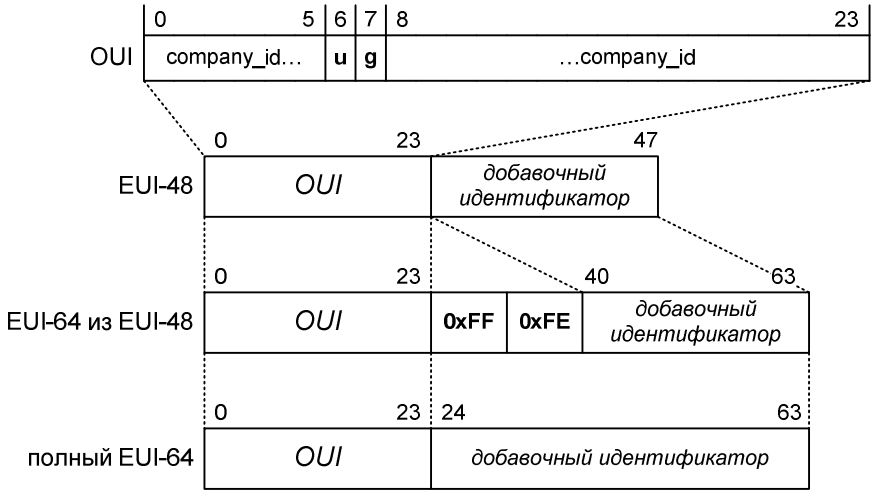
> Придирчивый читатель может заметить, что адрес и идентификатор — не одно и то же. Тем не менее, идентификатор может быть адресом при соблюдении дополнительных условий. Например, в классической технологии Ethernet идентификатор может быть адресом благодаря тому, что изначально *Ethernet* был широковещательной средой. В этих условиях адрес мог не обладать свойствами локатора; ему не нужно было отвечать на вопрос: «Где находится данный узел?» Эта особенность не могла не наложить отпечаток на последующее развитие коммутируемого *Ethernet*: именно из-за нее коммутаторам *Ethernet* приходится динамически изучать расположение станций на своих портах.

Отлично, теперь вернемся к нашим текущим делам и внимательнее посмотрим на свойства гипотетического адреса IPv6, автоматически полученного соединением префикса подсети и идентификатора EUI-64. Благодаря уникальности EUI-64, мы надеемся, что полученный адрес IPv6 окажется уникальным; но как мы сможем гарантировать, что на другом узле тот же самый адрес не будет назначен вручную?

Поясним наше опасение примером. Допустим, что к одному каналу подключены два узла IPv6, А и Б. Их интерфейсам требуются внутриканальные адреса. Узел А получает внутриканальный адрес от нас, а узел Б пытается выбрать его автоматически, располагая адресом EUI-64 на своем интерфейсе. Мы решили назначить узлу А адрес *FE80::11:2233:4455:6677*. Как нам удостовериться, что узел Б не выберет тот же самый адрес? Заранее убедиться, что его EUI-64 не равен *00-11-22-33-44-55-66-77*? Очевидно, что такая «автоматика» недорого стоит, потому что мы обеспечиваем ее работу вручную. Представьте, то таких узлов не два, а тысяча, и 999 из них выбирают адрес автоматически, на основе EUI-64. Неужели нам придется обойти все 999 узлов, чтобы правильно выбрать тысячный адрес?

Чтобы разрешить это затруднение, нам надо вспомнить, что мы знаем об адресе MAC-48, он же EUI-48. Такой адрес обладает определенной структурой, и, к нашему счастью, она сохранилась в EUI-64 без изменений, с точностью до длины поля «добавочный идентификатор», как это показано на Фиг. 15.

[32] Guidelines for use of the 24-bit Organizationally Unique Identifiers (OUI). IEEE. http://standards.ieee.org/regauth/oui/tutorials/UseOfEUI.html



Фиг. 15. Формат OUI и соответствие между разными типами EUI

Сейчас основной интерес для нас представляет бит **U/L**, он же **u**, который отличает адреса, выданные глобально (полученные из рук *IEEE*), от адресов, назначенных локально (по усмотрению администратора сети). Второй бит, который нам придется учитывать в дальнейшем, — бит **I/G**, он же **g**, — отличает групповые адреса от индивидуальных. Преобразование из EUI-48 в EUI-64 сохраняет эти биты.

Благодаря биту **U/L** мы можем избежать конфликтов хотя бы с EUI-64, назначенными производителем аппаратуры. Например, в EUI-64 *00-11-22-33-44-55-66-77* этот бит сброшен, а значит, это глобальный идентификатор, и нам следует избегать его. И наоборот, EUI-64 *02-11-22-33-44-55-66-77* — локальный, так что мы можем выбрать его в качестве идентификатора интерфейса, не опасаясь конфликта с адресом какой-нибудь сетевой карты.

Обобщим эту мысль: идентификатор интерфейса в адресе IPv6 должен основываться на идентификаторе EUI-64 и наследовать его формат. Однако если мы перенесем этот формат без изменений, то окажется, что мы больше не вправе назначать интерфейсам простые, короткие номера: *2001:DB8::1, 2001:DB8::2* и т.д. Ведь если мы рассмотрим, к примеру, число 1 с позиций EUI-64, то это окажется *00-00-00-00-00-00-00-01* — глобально выданный идентификатор, и мы нарушим наше собственное правило. Иначе говоря, нам придется всегда устанавливать бит **U/L** в идентификаторах интерфейсов, которые мы выдумываем сами, например, *2001:DB8::1* нам придется заменить на *2001:DB8:0:0:200:0:0:1*.

Как нам избежать этого неудобства? Возможный выход здесь — это задать взаимно однозначное отображение между настоящим EUI-64 и идентификатором интерфейса IPv6. Ведь нас никто не заставляет точно копировать формат EUI-64 — нам достаточно сохранить содержащуюся в нем информацию. Правило этого отображения очевидно: инвертировать бит **U/L**, с тем чтобы удобные короткие идентификаторы 1, 2, 3… стали доступны для локального назначения. В результате мы получаем так называемый **модифицированный формат EUI-64** (*modified EUI-64 format*), который отличается от EUI-64 только интерпретацией значений бита **U/L**: теперь 0 означает «локальный», а 1 — «глобальный» [§2.5.1 и Приложение A RFC 4291].

Окончательное требование таково: идентификатор интерфейса в адресе IPv6 обязан отвечать модифицированному формату EUI-64. Это правило распространяется на все индивидуальные адреса IPv6 кроме блока *::/3*, зарезервированного для особых целей [§2.5.1 RFC 4291].

Чтобы по этому правилу «изготовить» из EUI-64 адрес IPv6, надо сначала инвертировать бит **U/L**, а затем добавить перед полученной цепочкой битов (то есть в старших разрядах) данный префикс подсети.

Тогда аналогичный рецепт для EUI-48 (MAC-48) может быть комбинацией двух уже известных нам шагов: сначала от EUI-48 к EUI-64, а затем от EUI-64 к IPv6:

1) EUI-48→EUI-64:
    a) вставить между третьим и четвертым байтами EUI-48 (то есть между OUI и добавочным идентификатором) последовательность двух байтов FF-FE;
2) EUI-64→IPv6:
    a) инвертировать бит **U/L**;
    б) присоединить спереди префикс подсети.

Любителям истории будет небезынтересно узнать, что в те времена, когда разработчики IPv6 впервые сформулировали правила для работы с модифицированным форматом EUI-64, MAC-48 и EUI-48 считались разными сущностями. Тогда разработчики IPv6 допустили ошибку в прочтении документов *IEEE* и перепутали алгоритмы преобразования MAC-48 и EUI-48 в EUI-64: говоря о MAC-48, они привели алгоритм для EUI-48, который и вошел в реализации IPv6. (См. замечание в конце Приложения A RFC 4291.) Впоследствии были разговоры о том, не исправить ли эту ошибку [33, 34], но оказалось проще оставить все как есть. И, наконец, сегодня EUI-48 и MAC-48 слились воедино, и все неожиданно встало на свои места, а старая ошибка превратилась в мудрый выбор.

Вот несколько примеров прямого преобразования:

- Из EUI-64 *00-11-22-33-44-55-66-77* и префикса *2001:DB8::/64* получается адрес IPv6 *2001:DB8::211:2233:4455:6677*.
- Из EUI-48 (или MAC-48) *10-20-30-40-50-60* и внутриканального префикса (*FE80::/64*) получается внутриканальный адрес IPv6 *FE80::1220:30FF:FE40:5060*.

Возможен и обратный анализ:

- Адрес IPv6 *2001:DB8::D00F* — назначен локально, так как бит **U/L** в идентификаторе интерфейса сброшен.
- Адрес IPv6 *2001:DB8::200:FF:FE00:D00F* — получен из глобального EUI-48 *00-00-00-00-D0-0F*.
- Адрес IPv6 *2001:DB8::200:0:0:D00F* — получен из глобального EUI-64 *00-00-00-00-00-00-D0-0F*.

Хотя бо́льшая часть локальных идентификаторов интерфейса (**U/L** = 0) доступна администраторам и пользователям для их собственных нужд, среди них есть несколько зарезервированных [RFC 5453]. Объяснить, откуда они возникают, мы сможем позже. В частности, зарезервировано нулевое значение, *0000:0000:0000:0000*.

В завершение раздела зададим себе такой вопрос: *достаточно* ли сформулированных нами правил, чтобы автоматически гарантировать уникальность адреса IPv6? Для ответа на него рассмотрим вот какой сценарий. Представим себе, что к одному каналу подключены два интерфейса, А и Б, причем интерфейсу А заранее назначен *локальный* EUI-64 *02-00-00-00-00-00-00-01* — это вполне законно. Далее администратор сети назначает интерфейсу Б внутриканальный адрес IPv6 *FE80::1*. Он тоже имеет на это полное право, потому что такой адрес отвечает нашим требованиям. И наконец узел, управляющий интерфейсом А, автоматически назначает своему интерфейсу внутриканальный адрес IPv6 на основе уже доступного EUI-64. Как нетрудно видеть, это окажется то же самый адрес *FE80::1*, и в зоне данного канала возникнет конфликт адресов.

---

[33] См. обсуждение «*RFC 3513 EUI-48/MAC-48 confusion*» в списке рассылки IETF IPng: http://www.atm.tut.fi/list-archive/ipng/msg10039.html

[34] См. обсуждение «*why 0xFFFE is used in the modified EUI-64 format*» в списке рассылки IETF IPv6: http://www.ietf.org/mail-archive/web/ipv6/current/msg06035.html

Корень проблемы здесь состоит в том, что локально назначенные идентификаторы интерфейсов IPv6 неявно занимают ячейки в подмножестве локальных идентификаторов EUI-64.

Вот мы и обнаружили как минимум один проблемный случай — применение локальных EUI-64. Это значит, что автоматическое назначение адресов IPv6 останется ненадежным, пока мы целенаправленно не поработаем над механизмом предотвращения конфликтов. Тем не менее, структура идентификатора интерфейса на основе EUI-64 сыграет роль хорошего фундамента для наших последующих разработок — в этом мы убедимся в §5.4.

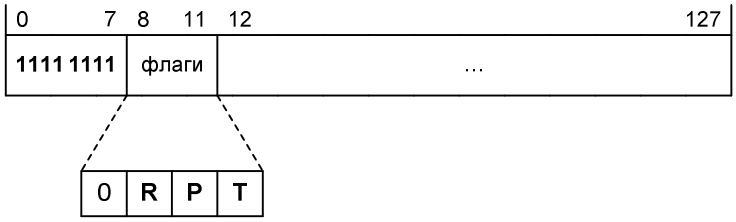
2.8. Структура группового адреса IPv6

До сих пор мы сказали о групповых адресах IPv6 только то, что они полностью займут префикс *FF00::/8* и что у них тоже будут области действия. Теперь нам пора уточнить детали.

Прежде всего, заметим, что групповые адреса бывают двух видов: общепринятые и выбранные для каких-то частных целей. Первые из них назначает IANA, внося в соответствующий реестр [35]. Эти назначения носят долговременный характер, так что общепринятые групповые адреса можно охарактеризовать как **постоянно назначенные** (*permanently-assigned*). Например, в IPv4 к этой категории относились адреса *224.0.0.1* (все узлы подсети) и *224.0.0.2* (все маршрутизаторы подсети) [36]. Адреса второго вида каждый выбирает для своих собственных целей или нужд своей организации, не претендуя на монопольное владение ими, и освобождает, когда они больше не нужны. Такие адреса, в противоположность первым, мы назовем **временно занятыми** (*transient*). Четко разграничить эти два вида адресов необходимо, поскольку общепринятые адреса важны для работы стандартных протоколов, а значит, конфликты между адресами двух видов способны причинить заметный вред сети.

Групповых адресов у нас довольно много, так что мы не будем ломать голову над оптимальной пропорцией между общепринятыми и временно занятыми адресами: пусть их будет поровну. Тогда надежно отличить временно занятый адрес от общепринятого нам позволит один бит-флаг, который мы обозначим **T** (*transient*). Его смысл прост: **T** = 0 означает общепринятый адрес, а **T** = 1 — временно занятый [§2.7 RFC 4291].

Раз уж дело дошло до флагов, то давайте выделим для них целый полубайт, как показано на Фиг. 16. На сегодняшний день в нем занято еще два флага, **P** и **R**. О назначении флага **P** мы поговорим в §2.9, а в случае **R** ограничимся его нулевым значением.



Фиг. 16. Флаги в групповом адресе IPv6

О назначении флага **R** можно узнать из RFC 3956. Знакомство с этим документом пройдет легче и интереснее, если читатель сперва завершит вместе с нами «работу» над основными элементами IPv6 в рамках данной книги.

Распределение временно занятых адресов никто не координирует, и это палка о двух концах. С одной стороны, это удобно и облегчает использование группового вещания для самых разных целей. Действительно, глупо было бы требовать запроса в IANA для того, чтобы провести небольшую видеоконференцию или испытать новый

[35] http://www.iana.org/assignments/ipv6-multicast-addresses
[36] http://www.iana.org/assignments/multicast-addresses

протокол узкого применения. С другой стороны, разрешение конфликтов между сторонами, использующими временно занятые адреса, ложится на плечи самих сторон. В отсутствие публичного реестра этих адресов, даже установить сам факт конфликта непросто, поскольку это может потребовать низкоуровневой отладки сети. К счастью, в нашем арсенале уже есть инструмент, который позволит избежать конфликтов даже при одинаковом численном значении адреса. Это, конечно же, будет ограничение области действия временно занятых адресов.

Но это вовсе не значит, что общепринятым групповым адресам области не требуются. Напротив, ограничение области позволяет более тонко управлять множеством узлов — а точнее, их интерфейсов, — на которые указывает тот или иной общепринятый адрес. В групповом вещании IPv4 эта концепция имела рудиментарную форму: общепринятые адреса с префиксом *224.0.0.0/24* относились только к текущей подсети, то есть были внутриканальными. Теперь же, варьируя область, мы сможем избирательно указывать на группу нужного нам «диаметра». Например, если нам дан прикладной протокол FOO [RFC 3092], то у нас должна быть возможность составить как минимум такие адреса:

- все серверы FOO на данном канале;
- все серверы FOO в данном сайте;
- все серверы FOO в *Internet*.

Чтобы решение этой задачи подошло и общепринятым, и временно занятым адресам, область группового адреса надо закодировать в нем самом как можно проще и независимо от прочих полей. Поступим так: вслед за полубайтом флагов отведем полубайт области. Стандартные значения этого полубайта [§2.7 RFC 4291] приведены в Табл. 4.

Табл. 4. Общепринятые коды области группового адреса IPv6 [§2.7 RFC 4291]

Код	Значение
0	(резерв)
1	область интерфейса
2	внутриканальная область
3	(резерв)
4	область администратора
5	область сайта
6–7	(доступно)
8	область организации
9–13	(доступно)
14	глобальная область (вселенная)
15	(резерв)

Прокомментируем эти области.

Область интерфейса (*interface-local scope*) позволяет ограничить групповую рассылку единственным интерфейсом узла источника. Такой групповой трафик не покинет пределов узла источника; в этом он аналогичен индивидуальному трафику в адрес *::1*.

Мы, конечно же, помним, что узел вступает в группу IP не целиком и полностью, а на отдельных сетевых интерфейсах. В терминах общепринятого группового API, ссылка на интерфейс, такая как его номер или имя, — это обязательный аргумент вызовов «вступить в группу *X*» и «выйти из группы *X*» наряду с адресом группы [§7.1 RFC 1112, §5.2 RFC 3493]. Именно поэтому область с номером 1 проводит границу вокруг одного интерфейса, а не вокруг всего узла.

Внутриканальная область (*link-local scope*) нам уже знакома: она охватывает один канал и все интерфейсы, подключенные к нему.

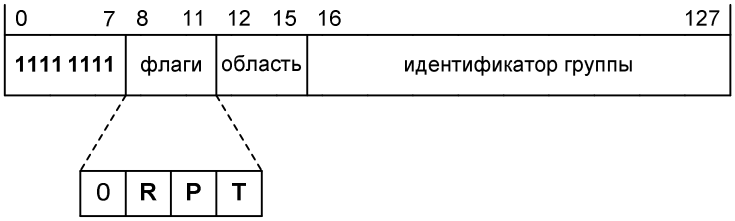
Области с кодами 4–13 призваны соответствовать организационным структурам разной величины. Границы их зон уже не следуют из конфигурации каналов или других косвенных источников, а зависят от политики управления сетью, так что настроить эти границы можно только явным образом. Как именно проводится подобная настройка, зависит от программного обеспечения, но в принципе ее можно свести к перечислению всех сетевых интерфейсов, входящих в данную зону.

В частности, как следует из их названий, область администратора (*admin-local scope*) отвечает наименьшему участку административной ответственности, область сайта (*site-local scope*) охватывает один **сайт** (площадку), а область организации (*organization-local scope*) — целую организацию, у которой, вероятно, есть несколько сайтов (площадок). Коды, помеченные как «(доступно)», можно назначать областям нестандартной величины, если это необходимо для каких-то частных задач. Разумеется, при этом больший код должен отвечать области большей величины.

> Справедливости ради заметим, что для групповых адресов IPv4 были также предложены административно заданные области действия [RFC 2365].

Наконец, глобальная область (*global scope*) не накладывает никаких ограничений на распространение группового трафика.

Мы уже израсходовали в групповом адресе 16 старших разрядов: восемь ушло на префикс *FF00::/8*; следующие четыре заняты флагами; и еще четыре занимает код области. Значит, у нас осталось 112 младших разрядов. Они-то и составят **идентификатор группы** (*group ID*), который позволит различать группы, сосуществующие в пределах зоны. В результате мы получим структуру группового адреса IPv6, изображенную на Фиг. 17.



Фиг. 17. Формат группового адреса IPv6

Для простоты и единообразия положим, что если адрес общепринятый (**T** = 0), то группа получает идентификатор во всех областях сразу. К примеру, если группа «все серверы mDNSv6» получит идентификатор `0xFB` [37], то мы без труда сможем составить ее адреса разных областей (см. Табл. 5), даже не вполне представляя себе, что такое mDNSv6.

Табл. 5. Групповые адреса серверов mDNSv6 разных областей действия

Адрес	Точная группа
FF01::FB	Все серверы mDNSv6 на том же интерфейсе, что и источник пакета
FF02::FB	Все серверы mDNSv6 на том же канале, что и источник пакета
FF05::FB	Все серверы mDNSv6 в том же сайте, что и источник пакета
FF0E::FB	Все серверы mDNSv6 в *Internet*

> На практике может оказаться, что общепринятая группа имеет смысл не во всех областях. Например, группа *FF02::5*, «все маршрутизаторы OSPF», определена только в области канала. В этом случае идентификатор группы зарезервирован в других областях. Так, группы *FF01::5* и *FF05::5* никогда назначены не будут, и это позволит избежать путаницы.
>
> Составьте групповые адреса разных областей для протокола ASAP, получившего *десятичный* идентификатор группы 307.

[37] http://www.iana.org/assignments/ipv6-multicast-addresses/

С временно занятыми адресами (**T** = 1) так поступить нельзя, потому что сетевые администраторы разных уровней вполне могут применить один и тот же номер группы для различных целей, если у них нет общей политики в этом аспекте. Здесь области нужны именно для того, чтобы заранее предотвратить возможный конфликт. Благодаря областям, любой сетевой администратор может занять идентификатор группы в своей зоне без оглядки «вниз», «вверх», или «на соседа». Поэтому неудивительно, что временно занятый адрес имеет смысл только в данной зоне и его трактовка никоим образом не распространяется на другие зоны той же или иной величины.

При всех преимуществах этой схемы, ее возможности по разрешению конфликтов не безграничны. Так, в каждой зоне только кто-то один может выбирать временно занятые адреса, а в противном случае снова возникает опасность конфликта. В частности, неясно, кто распоряжается временно занятыми адресами в глобальной зоне. Мы посвятим решению этой проблемы §2.9.

А в завершение текущего раздела давайте «назначим» важнейшие общепринятые группы [§2.7.1 RFC 4291], как показано в Табл. 6.

Табл. 6. Адреса важнейших общепринятых групп IPv6 [§2.7.1 RFC 4291]

ID	Адрес	Группа
0	*FF0x::*	(резерв)
1	*FF0x::1*	«все узлы»; определена для таких значений кода области *x*: 1 (интерфейс) и 2 (канал)
2	*FF0x::2*	«все маршрутизаторы»; определена для таких значений кода области *x*: 1 (интерфейс), 2 (канал) и 5 (сайт)

> А почему нет стандартных групп *FF04::2* и *FF08::2*? Дело в том, что группа «все маршрутизаторы» охватывает только маршрутизаторы индивидуального трафика (*unicast*), а потому она имеет смысл только в областях, определенных для индивидуальных адресов, тогда как области администратора (*x* = 4) и организации (*x* = 8) для индивидуальных адресов не определены. Область интерфейса (*x* = 1) для них тоже не определена, но, как мы уже сказали, эта область играет роль интерфейса «петля» для группового трафика.
>
> Еще одно общепринятое назначение — особенно важное для нас — это групповые адреса для примеров в учебниках и документации. Для них выделено целое семейство префиксов *FF0x::DB8:0:0/96* [§3 RFC 6676].

В отличие от IPv4, где поддержка группового вещания рекомендована, но все же носит факультативный статус [§3 RFC 1112, §3.3.7 RFC 1122], в IPv6 групповое вещание — это неотъемлемая часть протокола. Поэтому каждый узел IPv6 обязан состоять в группах «все узлы» на всех активных интерфейсах (*FF01::1*) и подключенных к ним каналах (*FF02::1*). В дополнение к этому, маршрутизаторы IPv6 должны вступать в относящиеся к ним группы «все маршрутизаторы» (*FF01::2*, *FF02::2*, *FF05::2*).

> Одним из плодов этого решения будет то, что IPv6 сможет полностью избавиться от широковещания. Широковещательный адрес IPv4 превращался в такой же канальный адрес, например, MAC *FF-FF-FF-FF-FF-FF*, создавая паразитную нагрузку на станции канала, которые не имели дела с IPv4. Так что теперь, чтобы направить пакет всем узлам IPv6 определенной области, следует использовать подходящий групповой адрес «все узлы». Однако окончательный отказ от идеи широковещания в среде IPv6 произойдет только в §5.3, когда мы рассмотрим все основные ситуации, в которых наш опыт IPv4 подталкивает нас к применению этого режима адресации.

2.9. Расширение возможностей групповой адресации с помощью индивидуальных адресов

В §2.8 мы обнаружили, что даже области действия не защитят от конфликтов, когда независимые пользователи группового вещания IPv6 выбирают себе временно занятые адреса из одной и той же зоны. Так, например, все мы находимся в одной глобальной зоне, и произвольный выбор глобального временно занятого адреса заведомо чреват конфликтом, даже если вероятность его низка. Чтобы пользователи группового вещания IPv6 не работали по принципу «авось пронесет», нужен глобальный арбитр, который бы управлял распределением и оборотом глобальных временно занятых адресов. Но кто возьмет на себя этот труд и связанные с ним расходы? Добровольцев пока нет, и даже не предвидится. В конечном итоге, именно об это препятствие разбились все попытки создать выделенную систему управления временно занятыми групповыми адресами в *Internet*.

> Из таких попыток стоит упомянуть протокол **MADCAP** (*multicast address dynamic client allocation protocol*) [RFC 2730], который имеет статус предлагаемого стандарта. Тем не менее, насколько нам известно, распространения он так и не получил [§3.5 RFC 6308].

Но даже если препятствие нельзя преодолеть в лоб, его частенько можно обойти сбоку. Как бы нам сделать это? Например, можно воспользоваться уже существующей службой распределения чего-либо. Как сейчас окажется, у нас уже есть отменный кандидат на эту роль.

Для работы в *Internet* нового поколения конечному пользователю нужен, по меньшей мере, глобальный индивидуальный адрес IPv6, но гораздо лучше будет выделить ему целый префикс. Ведь, как мы сказали еще в §2.1, адресов IPv6 достаточно много, чтобы поощрять пользователей к созданию целых сетей вокруг себя. Для распределения индивидуальных префиксов уже создана международная система, включающая в себя IANA, несколько RIR и множество LIR. Одна из ее задач состоит как раз в том, чтобы обеспечить уникальность выдаваемых префиксов и избежать каких-либо конфликтов в подпространстве индивидуальных адресов.

Теперь обратим внимание, что выданный пользователю индивидуальный префикс должен быть не длиннее стандартного префикса подсети IPv6 (64 бита — см. §2.7), так как подсеть дальше не делится. В противном случае пользователь вряд ли сможет найти применение выданному префиксу.

Поэтому каждый пользователь, законным путем получивший глобальный индивидуальный префикс IPv6, может составить заведомо уникальный групповой адрес, взяв за основу идентификатора группы свой префикс, например, поместив его в старшие биты такого идентификатора. Длина идентификатора группы составляет 112 бит, а этого с запасом достаточно, чтобы вместить любой индивидуальный префикс IPv6. Максимальная длина такого префикса — */64*, так что пользователю даже остается 48 бит идентификатора группы, которыми он может распоряжаться по своему усмотрению, а это означает 2^{48} глобально уникальных групп!

Более внимательный анализ находит в таком изящном трюке две проблемы:

- во-первых, возможен конфликт между двумя сообществами пользователей, когда одни станут производить временно занятые групповые адреса от своих префиксов, а другие продолжат выбирать их наугад;
- во-вторых, префиксы могут быть разной длины, и, скажем, *2001:DB8:a1ef::/48* — не одно и то же, что *2001:DB8:a1ef::/64*, поскольку длина префикса — его неотъемлемый атрибут. Вдобавок, нам не следует забывать, что система распределения префиксов — многоуровневая, и владелец префикса *2001:DB8:a1ef::/48* вполне может отличаться от владельца *2001:DB8:a1ef::/64*, хотя они и находятся на одной вертикали

иерархии: последний получил префикс от первого, возможно, через посредников.

К счастью, это проблемы практического характера, и нам вполне по силам их решить. Так, для решения первой проблемы достаточно, чтобы эти два сообщества пользователей оперировали непересекающимися подмножествами адресов. Обеспечить это способен еще один бит в поле флагов группового адреса, обозначаемый **P**. Пусть значение **P** по умолчанию, то есть нулевое, означает прежний, анархический способ выделения — или, скорее уж, захвата — временно занятых адресов. Напротив, если в групповом адресе IPv6 **P** = 1, то перед нами адрес, основанный на уникальном индивидуальном префиксе. Очевидно, что установленный флаг **P** имеет смысл только при **T** = 1, так как речь идет только о временно занятых адресах. В противном случае значение флага **P** должно быть нулевым. Вот мы и справились с первой проблемой.

Вторую проблему тоже решить несложно. В идентификаторе группы остались незанятыми 48 бит, и мы можем «отрезать» от них поле длины префикса, а уж остальные биты достанутся пользователю. Так как префикс может быть длиной от 0 до 64 бит включительно, нам понадобится минимум 7 бит, чтобы представить все возможные значения целым беззнаковым полем. Но ради удобства пользователей и программистов мы расширим это поле до целого байта, то есть 8 бит.

> Не востребованные немедленно значения диапазона пригодятся для расширений. Например, особое значение длины префикса 255 означает внутриканальный групповой адрес, основанный на идентификаторе интерфейса [RFC 4489]. Хост может сам создать такой адрес, например, для нужд Zeroconf.

В результате мы получаем изображенный на Фиг. 18 формат **группового адреса, основанного на индивидуальном префиксе** (*unicast-prefix-based multicast address*, **UBM**) [RFC 3306]. Несмотря на сокращение поля доступных бит, пользователь получает в свое распоряжение 32 бита, что тоже весьма неплохо!

Фиг. 18. Формат адреса UBM

> Откуда взялось поле «резерв» между кодом области и длиной префикса, мы, по правде говоря, не знаем. Возможно, оно выравнивает поле «префикс сети» на границу 32 бит, чтобы такие адреса было удобнее составлять. Как и у всех полей «резерв», его биты обязаны быть нулевыми.

> Адреса UBM определены не только в IPv6, но и в IPv4 [RFC 6034].

Например, обладатель глобального префикса *2001:DB8:D0D0::/47* получит в свое нераздельное пользование все групповые адреса общего вида `FF3`*x*`:002F:2001:DB8:D0D0:0:`*zzzz*`:`*zzzz*, где *x* — любой допустимый код области (см. Табл. 4 на стр. 42), а *zzzz:zzzz* — произвольный идентификатор группы.

> Можно сказать, что вместе с индивидуальным префиксом *2001:DB8:D0D0::/47* его обладатель получил все групповые префиксы *FF3x:002F:2001:DB8:D0D0::/96*, где *x* — допустимый код области.

> Любой технический автор может свободно распоряжаться всеми адресами UBM, основанными на префиксе для документации *2001:DB8::/32* и его дочерних префиксах большей длины [§3.1 RFC 6676].

Благодаря удачному выравниванию все элементы адреса видны невооруженным глазом. В частности, `002F` — это длина префикса (47), записанная по основанию 16, а сам

префикс находится сразу за полем длины. Интересным свойством адресов UBM будет то, что по значению адреса легко определить, из какой сети он произошел, хотя сама группа может включать в себя интерфейсы из разных сетей по всему миру.

Нам осталось поставить последний штрих в этой схеме, ответив на такой вопрос: как быть, если индивидуальный префикс — не глобальный, а меньшей области? Допустим, к примеру, что он внутрисайтовый. Откуда пользователь взял его? Скорее всего, пользователь принадлежит какой-то организации, или части организации, сеть которой основана на внутрисайтовых адресах и представляет собой зону величины «сайт». В этой организации тоже есть ответственный за распределение адресного пространства, например, администратор сети или отдел технического обеспечения. Именно он и выдал пользователю префикс, уникальный в пределах данного сайта. Поэтому произведенный от такого префикса групповой адрес тоже будет уникальным в пределах сайта. Выходит, что групповые адреса, основанные на неглобальных префиксах, получат право на существование, если мы ограничим их хождение зоной префикса. Для этого нам достаточно ввести такое правило: область группового адреса, основанного на индивидуальном префиксе, должна быть не больше, чем область самого префикса [§4 RFC 3306]. Например, пользователь может создать на основе внутрисайтового индивидуального префикса групповые адреса с областями «интерфейс», «канал», «администратор» и «сайт», а вот «организация» и «вселенная» — не имеет права.

> В свою очередь, групповые маршрутизаторы на границах сайта должны отфильтровывать пакеты с адресом назначения, нарушающим это правило.

Закончив с этим решением, посмотрим на возможные альтернативы ему. Так, в некоторых прикладных задачах достаточно, чтобы групповой трафик исходил из единственного источника. Например, к этому типу относятся задачи вещания, построенного по той же схеме, что радио и телевидение: у так называемого **канала вещания** (*channel*) есть ровно один источник и от нуля до бесконечности слушателей. Этот режим заметно отличается от традиционного группового вещания IP, когда любой источник может передавать пакеты данной группе.

При вещании из одного источника идентификатором последнего выступает адрес IP. Если той же группе станет вещать источник с другим адресом, то это будет другой канал вещания. Поэтому идентификатором канала вещания выступает упорядоченная пара (S,G), где S — адрес источника, а G — адрес группы назначения. Например, в этом режиме узел-слушатель проверяет, адресован ли ему данный групповой пакет, не по списку групп G, в которых он состоит на входном интерфейсе, а по списку каналов вещания (S,G) на этом интерфейсе. Поэтому данный режим называют **групповое вещание с настройкой источника** (*source-specific multicast*, **SSM**) [RFC 4607].

По аналогии, традиционный режим, когда вещать группе может любой источник — это **групповое вещание из произвольного источника** (*any-source multicast*, **ASM**) [§2 RFC 3569].

Есть ли у адресации SSM какие-нибудь интересные особенности? В режиме SSM уникальным[38] должен быть канал (S,G). Иными словами, каналы (S_1,G_1) и (S_2,G_2) совпадают, только если $S_1 = S_2$ и $G_1 = G_2$. Поэтому, если вещают источники S_1 и S_2, где $S_1 \neq S_2$, то все их каналы автоматически будут разными, независимо от выбранного адреса назначения G. А значит, в режиме SSM источнику не надо заботиться об уникальности адреса G! Точнее, разные адреса назначения G_1, G_2 и т.д. понадобятся только разным каналам одного и того же источника S.

> Возможен и промежуточный режим, когда слушатели принимают групповые пакеты только от заданного множества источников. Его называют **групповое вещание с фильтрацией источника** (*source-filtered multicast*, **SFM**) [§2

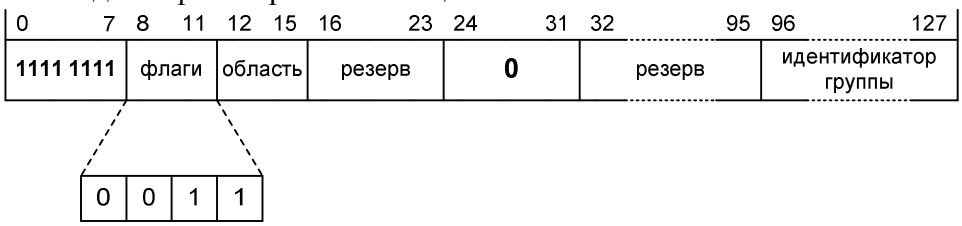
[38] Конечно же, в пределах зоны.

RFC 3569]. В простейшем случае этот режим можно реализовать локально, в самих слушателях, как программируемый фильтр пакетов. Однако из соображений эффективности сеть группового вещания тоже участвует в этой фильтрации, чтобы не передавать групповой трафик туда, где его все равно потом отбросят. В частности, IGMPv3 и MLDv2 (§6.4) поддерживают SFM. По своей адресации SFM гораздо ближе к ASM, чем к SSM, так как по-прежнему оперирует группами с уникальным адресом *G*, тогда как источник *S* интересен только фильтру.

Единственная наша проблема состоит в том, что до сих пор мы говорили о типах группового вещания как о *режимах*. Означает ли это, что каждой реализации сетевого стека IPv6 понадобится тумблер ручного управления с положениями «ASM» и «SSM»? Конечно, это было бы очень неудобно и даже глупо. Сетевой стек должен сам отличать на входе групповые пакеты, переданные в режиме ASM, от пакетов, переданных в режиме SSM.[39] Тогда и источник сможет выбирать режим для каждой передачи отдельно, по запросу приложения. Пока мы работаем над адресами IPv6, давайте закодируем различие между ASM и SSM в адресе назначения *G*. Иными словами, адресу *G* в составе канала SSM нужна отличительная черта, которая выделит его среди обычных групповых адресов ASM.

Можно сказать, что каналы (*S*,*G*) в чем-то родственны групповым адресам UBM: и здесь, и там индивидуальный адрес или префикс составляет часть группового адреса. Только у вторых индивидуальный префикс «зашит» непосредственно в групповой адрес назначения, тогда как у первых индивидуальный адрес передается отдельно — по видимому, в поле «источник» заголовка IPv6, над которым нам еще предстоит работать. Поэтому вряд ли кто-то возразит, если мы сделаем адрес назначения SSM частным случаем группового адреса UBM.

Поставив себя на место IANA, мы пожертвуем пустым префиксом нулевой длины *::/0* ради SSM.[40] То есть, когда в групповом адресе IPv6 **T** = 1 и **P** = 1, а поле «длина префикса» равно нулю, как на Фиг. 19, то перед нами адрес SSM. Нетрудно подсчитать, что все адреса SSM охвачены префиксами *FF3x::/96*, где *x* — допустимый код области из Табл. 4 на стр. 42. В каждой области доступно 2^{32} адресов, что не так уж и много по современным меркам. Однако особенность адресации SSM такова, что эти адреса полностью доступны каждому источнику SSM, поскольку адрес источника тоже входит в окончательный идентификатор канала вещания.



Фиг. 19. Формат группового адреса SSM

Как можно заметить, сама идея SSM никак не зависит от особенностей IPv6. Поэтому неудивительно, что механизм SSM определен и для IPv4 [RFC 4607]. Адресам SSM выделен блок *232.0.0.0/8* [[41]].

Последний наш вопрос по SSM традиционно будет насчет областей. В канале (*S*,*G*) элементы *S* и *G* — это не что иное, как адреса IPv6, и у каждого из них есть определенные область и зона действия. Нужны ли особые правила, связывающие эти области и зоны? На самом деле, нет — достаточно универсального правила, основанного на принципе изоляции зон из §2.4: интерфейс источника *S* должен находиться в зоне адреса *G*, а все интерфейсы *G* — в зоне адреса *S*.

[39] Такое решение будет в духе архитектурных принципов *Internet* [§3.8 RFC 1958].
[40] Ведь пустой префикс *::/0* находится в безраздельном ведении IANA.
[41] http://www.iana.org/assignments/multicast-addresses/

В завершение раздела отметим, что значение поля «идентификатор группы» у всех адресов с **T** = 1, включая UBM и SSM, ограничено диапазоном 0x8000 0000–0xFFFF FFFF на уровне политики распределения групповых адресов [§4.3 RFC 3307]. Зачем это было сделано, мы увидим в §4.1.1, когда рассмотрим разрешение групповых адресов IPv6 в адреса MAC.

2.10. Проблема внутрисайтовых адресов и ее решение

Последний вопрос, связанный с адресацией IPv6, может показаться неожиданным. Он таков: как нам быть с внутрисайтовыми индивидуальными адресами? Постойте, ведь совсем недавно мы сами же ввели их в обсуждение, а затем активно использовали в примерах, когда нам требовались области разной величины…

Проблема здесь исключительно в том, что мы опирались на идею сайта, которая имеет не так уж много общего с практикой эксплуатации сетей. Наш идеальный сайт представлял собой четко обозначенную территорию с незыблемыми границами, тогда как на практике оба качества, определенность и неизменность, подвержены постоянным испытаниям со стороны объективной реальности. Самое опасное из них — это объединение сайтов. Представьте себе, что две организации, сети которых построены на внутрисайтовых адресах, решили объединиться. Естественно, им захочется провести слияние сетей. Если организациям повезет, то их множества подсетей окажутся непересекающимися. Тогда уже назначенные адреса сохранят уникальность, и достаточно будет пересмотреть индексы зон на бывших пограничных узлах, чтобы вместо двух сайтов образовался один. Но, несмотря на внушительную величину блока внутрисайтовых адресов, */10*, вероятность конфликта велика благодаря человеческому фактору. Как вы думаете, какие префиксы предпочитают администраторы сетей? Правильно, те из них, что можно покороче записать и полегче запомнить: *FEC0:0:1::/48*, *FEC0:0:0:1::/64* и т.п. Поэтому при объединении двух больших сайтов наверняка возникнет хотя бы несколько конфликтов в адресном пространстве.

> Те, кто хотя бы раз пробовал объединить две частных сети *10.0.0.0/8*, не дадут нам соврать.
>
> Пусть читатель самостоятельно найдет еще несколько примеров, когда внутрисайтовые адреса создают больше проблем, чем решают.

Так мы приходим к довольно неожиданному выводу: частные адреса IPv6 должны обладать глобальной уникальностью. Конечно, они по-прежнему непригодны для использования в *Internet*, но их уникальность значительно облегчит сосуществование и взаимодействие разных сайтов. Например, несколько сайтов по взаимной договоренности смогут открыть доступ друг другу к своим ресурсам и создать сеть сетей, этакий «частный *Internet*». Безусловно, это потребует работы на разных уровнях, но, по крайней мере, у них не будет проблем с адресацией.

Решить поставленную задачу нам поможет относительно большая длина адреса IPv6. Согласно нашим рассуждениям о ней, каждому сайту полагается префикс длиной 48 бит. Если 8 старших бит занять общепринятым префиксом, единым для всех частных адресов, то под уникальный идентификатор сайта остается 40 бит. Какова будет вероятность конфликта, если N сайтов выберут эти 40 бит совершенно случайным образом? Эта задача известна в теории вероятности как парадокс именин или задача о днях рождения. Парадокс здесь заключается в том, что искомая вероятность намного выше, нежели вероятность угадать заранее выбранную величину из того же диапазона, которая в данном случае составляет 2^{-40}. Проведем вычисление по приближенной формуле [42]:

[42] M. Sayrafiezadeh. The Birthday Problem Revisited. Math. Mag. 67, 220-223, 1994.

$$p \approx 1 - \left(1 - \frac{N}{2M}\right)^{N-1},$$

где N — число участников (сайтов), а M — число вариантов (в нашем случае 2^{40}). Эта формула дает вероятность парного конфликта, когда один и тот же случайный выбор делают два сайта. В Табл. 7 приведены значения такой вероятности для разных порядков величины N (числа участников). Вероятность конфликта более высокого порядка (тройного, четверного и т.д.) будет существенно ниже, так что ей можно пренебречь.

Табл. 7. Вероятность парного конфликта при случайном выборе 40 бит

N	p
2	$9{,}09 \cdot 10^{-13}$
10	$4{,}09 \cdot 10^{-11}$
10^2	$4{,}50 \cdot 10^{-9}$
10^3	$4{,}54 \cdot 10^{-7}$
10^4	$4{,}55 \cdot 10^{-5}$
10^5	$4{,}54 \cdot 10^{-3}$
10^6	0,365
10^7	>0,999

Достаточно ли низки найденные нами вероятности? Для настоящей глобальной уникальности, видимо, нет. Так, миллион сайтов создаст конфликт с вероятностью треть. Однако этот конфликт вызовет практические трудности, только если весь миллион сайтов решит объединиться, что крайне маловероятно.[43] Это наблюдение помогает нам лучше понять наши собственные намерения. Нет, мы не готовим техническую базу для создания альтернативной глобальной сети на частных адресах. Мы всего лишь хотим, чтобы взаимодействие разумного числа сайтов происходило, с хорошей вероятностью, без проблем с адресацией. Но какое число сайтов считать разумным, а какую вероятность хорошей? Достаточно очевидно, что взаимодействие сайтов потребует не только технических, но и административных механизмов. Если число сайтов действительно велико (100, 1000…), то им окажется проще действовать посредством *Internet*, где все необходимые механизмы уже созданы и отлажены. Так что интуитивно понятно, что 1000 сайтов — это уже практически недостижимая величина. И, тем не менее, при объединении 1000 сайтов вероятность конфликта — всего лишь один случай из более чем двух миллионов.

> Много это или мало, пусть читатель решит сам. Для сравнения, тот же порядок у вероятности погибнуть в авиакатастрофе, сев на данный рейс [44].

Мы признаем, что проверка дала удовлетворительные результаты, а адреса на основе случайных префиксов при длине случайного поля 40 бит обладают требуемыми свойствами. Нам осталось зафиксировать их название и формат.

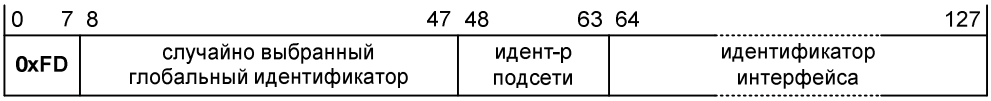
Адрес на основе случайного префикса получил название **уникальный локальный адрес** (*unique local address*, **ULA**) [RFC 4193]. Хотя вероятность конфликта между случайными префиксами пренебрежимо мала, она все же не равна нулю, и поэтому ULA можно считать глобально уникальным лишь условно.

Теперь завершим работу над техническими деталями ULA [§3 RFC 4193]. Восьмибитный общепринятый префикс, отличающий ULA от других адресов — это *FD00::/8*. За ним следуют 40 случайных бит — глобальный идентификатор. Вместе префикс *FD00::/8* и глобальный идентификатор образуют префикс */48*. Следующие 16 бит, как обычно, доступны сайту для деления на подсети, */64* каждая. Тогда

[43] Численную оценку этой маловероятности мы оставим читателю.

[44] Accident statistics. Plane Crash Info.com. http://planecrashinfo.com/cause.htm

окончательный формат адреса ULA будет таким, как показано на Фиг. 20. Например, если случайный идентификатор сайта оказался равен 0x123456789A, то префикс сайта будет таким: *FD12:3456:789A::/48*. Разумеется, на практике следует использовать настоящий, качественный генератор случайных значений, чтобы вероятность конфликта оставалась минимальной.



Фиг. 20. Формат адреса ULA

Какова область действия ULA? Мы хотели отказаться от понятия сайта как точного технического и географического определения и дать частным сетям возможность объединяться и взаимодействовать без границ, так что область действия ULA глобальная, а зона та же самая, что и у публичных адресов IPv6. Тем не менее, это вовсе не означает, что ULA имеют право появляться в *Internet*, ибо они остаются уникальными, только пока встречается ограниченное число сайтов.

> RFC 4193 также зарезервировал смежный блок, *FC00::/8*. Адреса в нем будут иметь тот же формат ULA: общепринятый префикс 8 бит, глобальный идентификатор 40 бит, идентификатор подсети 16 бит, идентификатор интерфейса 64 бита. Разница в том, что глобальный идентификатор будет выбираться каким-то другим способом, например, с помощью центрального арбитра [draft-hain-ipv6-ulac]. Вместе эти блоки ULA полностью занимают префикс *FC00::/7*.

> Самостоятельно обсудите границы применимости групповых адресов UBM (§2.9), основанных на индивидуальных адресах ULA.

Теперь, когда мы располагаем ULA, можно позволить себе отказаться от применения внутрисайтовых адресов на практике. Хотя IETF официально упразднил внутрисайтовые адреса еще раньше [§4 RFC 3879], только разумная альтернатива способна побудить пользователей IPv6 наконец-то последовать этому указанию.

2.11. Представление адресов IPv6 в DNS

Несмотря на трюки сокращенной записи, запоминать адреса IPv6 значительно труднее, чем имена узлов. Кроме того, работать с именами узлов удобнее, потому что они практически никогда не меняются.

> Администраторы флотских сетей даже убеждены, что переименовать узел — крайне дурная примета.

Хранить соответствия между именами и адресами узлов надо, конечно же, в DNS. Какие изменения потребуются, чтобы инфраструктура DNS приобрела поддержку IPv6?

Первый кандидат на пересмотр — это прямая зона. В ней соответствия «имя → адрес IPv4» хранились в виде записей RR A. У такой записи формат данных RDATA совершенно негибкий, так что правая сторона записи **A** может содержать только адрес IPv4. Следовательно, нам нужен новый тип записи RR, способный хранить адрес IPv6.

Если мы примем как неоспоримую истину, что IPv6 — это последнее и окончательное слово в протоколах сетевого уровня, то новый тип RR тоже не потребует особой гибкости: его поле данных **RDATA** будет просто адресом IPv6 в двоичном представлении с сетевым порядком байтов [§2.2 RFC 3596]. Адрес IPv6 вчетверо длиннее адреса IPv4, так что дадим новому типу RR остроумное имя **AAAA** [§2.1 RFC 3596]. Его общепринятый численный код — 28.

> Конечно, мы съязвили, оправдывая негибкость записи **AAAA**. Форматам сетевых данных гибкость нужна практически всегда, а в данном случае ее принесли в жертву простоте. Идеальным для нас решением была бы обобщенная запись RR, в которой тип адреса передавался бы как отдельное поле внутри **RDATA**.

Текстовый формат записи **AAAA** очевиден: слева доменное имя, справа адрес IPv6 в стандартной нотации из §2.2. Например:

```
www.example.org.      AAAA      2001:db8:c001::f00d
```

Все, прямая зона к IPv6 готова. Перейдем к обратной зоне, хранящей соответствия «адрес → имя». Эта зона привлекает записи **PTR**, а они не привязаны к определенному семейству адресов. На самом деле, эти записи вообще не имеют прямого отношения к адресам, так как их данные **RDATA** — доменные имена [§3.3.12 RFC 1035]. Однако сейчас нам нужно поместить адрес в *левую* часть такой записи, а там доменное имя содержится вообще у всех записей DNS.

Чтобы закодировать адрес IPv4 как доменное имя, пригодное для этой роли, потребовались следующие инструменты: обратная запись по байтам и служебный домен *in-addr.arpa*. Обратная запись позволяет делегировать префиксы IP в виде доменов. Это полезная особенность, и мы ее сохраним в IPv6. Между тем, мы понизим гранулярность делегирования до полубайта. Вдобавок, чтобы не путать IPv4 и IPv6, мы выделим новый служебный домен: *ip6.arpa* [§2.5 RFC 3596].

Каждому полубайту отвечает цифра в полной записи адреса IPv6 (без сокращений). Поэтому закодировать адрес IPv6 несложно: надо развернуть все сокращения, затем выписать цифры в обратном порядке через точку и, наконец, добавить справа домен *ip6.arpa*. Только и всего!

Например, приведенный выше адрес *2001:db8:c001::f00d* в полной записи выглядит так:

```
2001:0db8:c001:0000:0000:0000:0000:f00d
```

Далее он превращается в такое доменное имя:

```
d.0.0.f.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1.0.0.c.8.b.d.0.1.0.0
.2.ip6.arpa.
```

Всего получится 32 уровня за вычетом *ip6.arpa* — по числу полубайтов в адресе IPv6. Так как это доменное имя, сокращенной записи для него не предусмотрено: все символы придется выписать полностью.

> К счастью, при составлении зонного файла общий доменный суффикс можно не повторять в каждой записи, а вынести его в директиву `$ORIGIN` [§5.1 RFC 1035].

Естественным для нас вопросом будет, как соотносятся публикация адресов IPv6 в DNS и их зонная архитектура. В частности, будет ли иметь смысл внесение в DNS адресов, область действия которых меньше глобальной? Развернутый ответ на эти вопросы мы дадим в §6.7, когда у нас будут все необходимые инструменты. Однако уже сейчас нам интуитивно понятно, что в глобальных, публично доступных зонах DNS не следует размещать сведения о неглобальных адресах IPv6.

> Это правило справедливо и для общепринятых адресов, таких как адрес обратной связи *::1*. Даже если возникнет необходимость назначить им глобально доступные имена, это должны сделать центральные органы *Internet*, то есть IANA и ICANN, а не каждый оператор DNS.

Помимо этого, даже в частном DNS адреса ограниченной области действия не может сопровождать *zone_id*, потому что он имеет смысл только в пределах одного узла, тогда как DNS поставляет свои сведения многим узлам сразу. Публиковать в DNS адреса с идентификаторами зон совершенно бессмысленно, и мы не ошиблись, отведя полю **RDATA** записи **AAAA** только 128 бит и полностью проигнорировав возможный *zone_id*. Зона действия полученного из DNS адреса IPv6 должна быть ясна из контекста, например, внутрисайтовые адреса принадлежат тому сайту, которых их размещает на своем частном сервере DNS; очевидно, они не должны быть видны другим сайтам во избежание двусмысленности.

Обратите внимание на конфликт терминов: зона (действия) адреса IPv6 и зона DNS — это разные и независимые понятия.

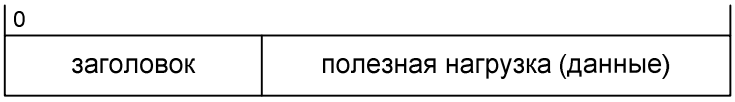
3. Пакет IPv6

3.1. Структура пакета

Разобравшись в первом приближении с адресом, мы можем перейти к следующему ключевому элементу IPv6 — пакету.

Несмотря на наш революционный настрой, саму концепцию сетевого пакета мы пересматривать не станем. IP был и останется протоколом коммутации пакетов.

Общепринятое сегодня деление пакета на заголовок и полезную нагрузку если и не является оптимальным в каждом возможном случае, то, по крайней мере, всех устраивает как самый простой компромисс. Поэтому мы можем не изобретать велосипед, а сохранить структуру пакета в самом общем ее виде, показанном на Фиг. 21.



Фиг. 21. Структура пакета IPv6 в первом приближении вполне традиционна

Следующим встает такой вопрос: как устроить заголовок, чтобы он был достаточно гибким и расширяемым? Ведь, как хорошо известно, без этих качеств наш протокол долго не протянет. В IPv4 для этой цели существовал механизм опций, который вполне справлялся со своей задачей. Чего в нем не хватало?

В первую очередь, более четкой структуры опций. Вообще говоря, опции IP могут быть двух видов. Опции первого вида представляют интерес для каждого узла по трассе пакета, а опции второго вида важны только для адресата. Первые мы назовем **пошаговыми опциями** (*hop-by-hop options*), а вторые — **опциями адресата** (*destination options*). Если опции этих видов перемежаются, то маршрутизаторам приходится просматривать их список полностью, чтобы найти в нем все пошаговые опции. Эта лишняя работа замедляет продвижение пакетов. Данную проблему еще можно было бы обойти, запретив источнику размещать пошаговую опцию после опции адресата.

Вообще-то говоря, у пакета IPv4 могло быть больше одного адресата. Так, групповой пакет был адресован целой группе узлов. Помимо этого, каждый адрес в опции семейства «маршрут от источника» (*source route*) был адресом назначения, хотя только один из них был адресом *конечного* назначения. Как оказывается, эти соображения вполне применимы и к IPv6: групповые адреса IPv6 у нас уже появились в §2.3 и §2.8, а маршрутизация от источника возникнет в §3.3.3, где мы и обсудим более подробно разные типы адресатов.

Далее, сегодня есть спрос на сквозную защиту IP. То есть информацию пакета надо защищать непосредственно на уровне IP, а не выше или ниже по стеку; а значит, защитные данные, такие как цифровая подпись, должны находиться в заголовке IP или его расширении.

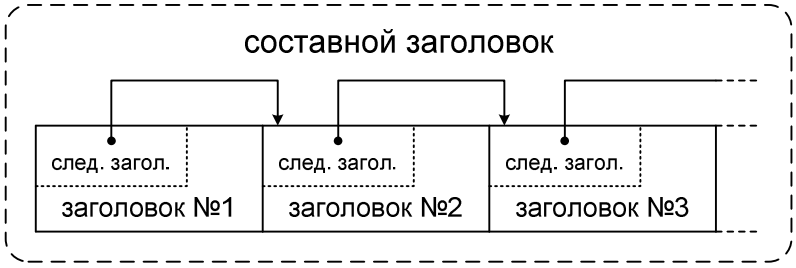
Здесь под защитой IP мы понимаем аутентификацию, шифрование и гарантию целостности пакетов. В TCP/IP эту задачу решает семейство протоколов IPsec, которого мы кратко коснемся в §3.3.5.

Определение «**сквозная**» означает, что в такой защите IP участвуют только источник и адресат, а транзитные узлы даже не отличают защищенные пакеты от обычных и обрабатывают их по одним и тем же правилам.

Допустим на минутку, что в нашем распоряжении — только опции IP. Тогда элементы защиты нам придется тоже выполнить в виде опций. Между тем, если в пакете уже есть опции адресата, то их можно было бы зашифровать, чтобы скрыть от посторонних глаз; ведь промежуточным узлам не нужно анализировать их. В этом случае

часть заголовка IP окажется открытой, а часть зашифрованной, что неизбежно собьет с толку маршрутизаторы. Чтобы такой подход мог хоть как-то работать, опции адресата должны были бы находиться строго после опций защиты, а все маршрутизаторы обязаны были бы знать о существовании опций защиты: тогда сканирование списка опций маршрутизатор прерывал бы на первой же опции защиты, потому что дальше может идти шифрованная «абракадабра». Однако данный подход в открытую нарушает сквозной характер защиты: она теряет свою прозрачность для транзитных узлов. Кроме того, подобное решение «в лоб» начисто лишено технического изящества.

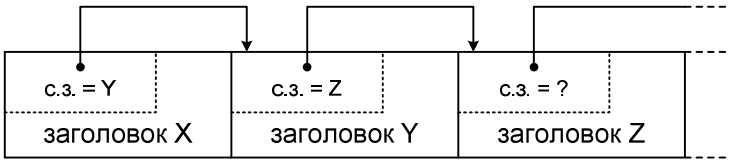
Поэтому мы не станем ограничивать себя опциями и пересмотрим саму структуру заголовка в пакете IPv6. Пусть этот заголовок будет модульным и состоит из более мелких, простых заголовков, каждый из которых решает ровно одну задачу. Такой составной заголовок схематически показан на Фиг. 22.



Фиг. 22. Составной заголовок пакета IPv6

Чтобы число простых заголовков было практически неограниченным, их можно «сцепить» вместе способом, в котором различимы мотивы связанного списка и кодирования TLV. А именно каждый простой заголовок содержит поле «следующий заголовок» (*Next Header*), в котором закодирован тип заголовка, идущего непосредственно за этим [§4 RFC 2460], как показано на Фиг. 23. Кроме того, заголовки переменной длины содержат поле длины данного заголовка, тогда как длина фиксированных заголовков заранее задана протоколом. Сведения о длине понадобятся, чтобы однозначно определить, где заканчивается текущий заголовок и начинается следующий.

Можно сказать, что IPv6 достигает модульности заголовка посредством рекурсивной инкапсуляции.



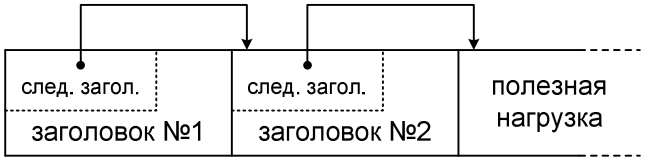
Фиг. 23. Тип простого заголовка IPv6 надо искать в предыдущем заголовке (с.з. = следующий заголовок)

А теперь обратите внимание: в отличие от кодирования TLV, где тип объекта содержится в самом объекте, тип заголовка IPv6 находится в *предшествующем* заголовке. Как же тогда получатель пакета IPv6 узнает, каков тип самого первого простого заголовка? Ему не придется гадать, если протокол заранее зафиксирует этот тип. Действительно, нам в любом случае потребуется обязательный заголовок, куда мы поместим адреса источника и назначения. Без этого заголовка не обойдется ни один пакет IPv6, так что пускай он и будет самым первым. Именно этот простой заголовок и называют **заголовком IPv6** (*the IPv6 header*), тогда как прочие простые заголовки называют **заголовками расширения** IPv6 (*IPv6 extension headers*). Чтобы избежать двусмысленности, мы довольно часто будем уточнять: *основной* заголовок IPv6, — в противовес заголовкам расширения.

Весь составной заголовок в пакете IPv6 собственного имени не удостоился, потому что его рассматривают как последовательность простых заголовков.

А что содержит поле «следующий заголовок» в самом последнем заголовке? Это поле отлично подходит для типа полезной нагрузки, которая расположена за последним

заголовком. Для этих типов уже существует реестр протоколов Internet [45]. Чтобы не нарушать стройной картины, пусть типы простых заголовков IPv6 вносятся в тот же самый реестр. Тогда любой простой заголовок сможет быть последним, как показано на Фиг. 24. Таким образом, поле «следующий заголовок» становится неотличимо от поля «тип полезной нагрузки», оно же «протокол». Иначе нам пришлось бы завести специальный заголовок расширения с полем «тип полезной нагрузки» вместо поля «следующий заголовок».

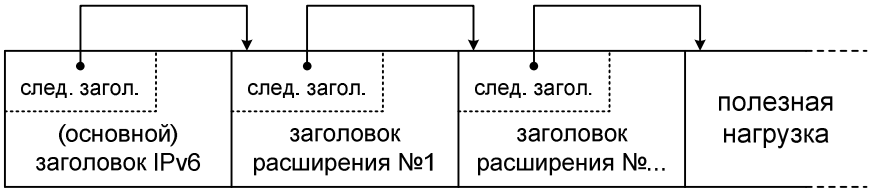


Фиг. 24. Соединение составного заголовка IPv6 с полезной нагрузкой

Отсюда немедленно следует длина поля «следующий заголовок» — 8 битов, — потому что в реестре протоколов IP всего 256 значений. Это совсем немного, так что мы должны ограничить свои аппетиты небольшим числом разных типов простых заголовков.

> Мы сознательно сказали «типы простых заголовков», а не «типы заголовков расширения». Основному заголовку IPv6 тоже назначен отдельный код, 41. Это пригодится нам позже, в §4.2, для туннельной инкапсуляции.
>
> К счастью, на сегодняшний день в реестре протоколов *Internet* еще достаточно свободных значений: из них занято чуть больше половины.
>
> Как мы увидим в §3.3.5, заголовки защиты IPsec — это всего лишь частный случай заголовков расширения. Но если это так, каким образом IPsec применяют к IPv4? Нетрудно увидеть, что цепочку заголовков расширения вполне можно прицепить и к заголовку IPv4. Ведь реестр протоколов один независимо от версии IP. Другое дело, что большинство заголовков расширения имеет отношение только к IPv6 и лишено смысла в среде IPv4. Заголовки IPsec — важное исключение из этого.

Собрав воедино сказанное в этом разделе, мы получаем структуру пакета IPv6, изображенную на Фиг. 25.



Фиг. 25. Окончательная структура пакета IPv6 (число заголовков расширения переменно)

3.2. Заголовок IPv6

Итак, каждый пакет IPv6 начинается с заголовка IPv6 (неожиданно, правда?) Но теперь, в отличие от IPv4, все необязательные элементы мы можем перенести из основного заголовка в заголовки расширения. Без каких полей мы абсолютно не сможем обойтись в основном заголовке? Используя наш опыт IPv4, а также не забывая сказанное в §1 и §3.1, составим предварительный неупорядоченный список:

- версия IP;
- адрес источника;
- адрес назначения;
- TTL;
- длина пакета;
- следующий заголовок.

[45] http://www.iana.org/assignments/protocol-numbers/

Теперь давайте рассмотрим подробнее эти поля и, если понадобится, скорректируем их свойства.

> Целочисленные поля, «созданные» нами по ходу нашей работы, будут по умолчанию в беззнаковом коде. Как вы думаете, почему отрицательные и дробные числа редко встречаются в сетевых протоколах?

Как мы уже обсудили в §1, поле версии IP поможет надежно отличить пакет IPv6 от пакета IPv4 в переходных условиях, пока новая версия будет работать наряду со старой. Учитывая масштабы применения IPv4 и финансовые вложения в технологии на его основе, этот переходный период может растянуться на многие годы. Очевидно, что длина и позиция поля версии обязаны быть такими же, как в заголовке IPv4. То есть длина этого поля — 4 бита, а находится оно в самом начале заголовка (учитывая сетевой порядок битов и байтов). Теперь его новое значение — 6.

Также нам никуда не деться без двух адресов IPv6, источника и назначения. Вместе они занимают 256 бит, или 32 байта, что весьма немало. Не надо ли нам попытаться сжать их двоичное представление путем какого-то кодирования, подобно тому как мы поступили с текстовым представлением? Нет, делать этого ни в коем случае не стоит; иначе ради выигрыша в несколько байтов мы пожертвуем простотой заголовка и затрудним его обработку, особенно на аппаратном уровне. Что мы еще должны сказать об этих адресах? Прежде всего, у пакета ровно один интерфейс источника, и поэтому адрес источника не может быть групповым. Если узел принял пакет с групповым адресом источника, его необходимо отбросить, никак не реагируя на него, потому что любая реакция по групповому адресу может открыть лазейку для атаки типа «отказ в обслуживании» против других узлов. Далее, адрес обратной связи *::1* можно использовать только в трафике через интерфейс типа «петля», будь то адрес источника или назначения, — прийти в пакете извне узла он не имеет права. Прочие требования к адресам в заголовке IPv6 мы сформулируем по ходу работы.

> Кстати, именно благодаря тому, что адрес источника в пакете IP не бывает групповым, мы можем говорить о типе всего пакета в зависимости от типа его адреса назначения, как то «индивидуальный пакет» и «групповой пакет».

Без поля TTL или аналогичного ему возникает риск бесконечного зацикливания пакетов в сети. При неудачном стечении обстоятельств это может также сопровождаться дублированием пакетов, что означает экспоненциальный рост паразитного трафика и полный крах сети. Так что без эквивалента TTL нам никак не обойтись. Однако, перенося поле TTL в заголовок IPv6, мы внесем в его свойства долгожданную поправку. Как показал опыт работы с IPv4 [§5.3.1 RFC 1812], измерить время удержания транзитного пакета на узле затруднительно и, к тому же, оно редко превышает одну секунду. Де-факто поле TTL ведет обратный отсчет не секунд жизни пакета, а его шагов по транзитным узлам. Мы не можем не учесть этот опыт в IPv6, и нам остается только переименовать поле сообразно его функции: «предельное число шагов» (*Hop Limit*). Длина его по-прежнему будет 8 бит. По смыслу это поле не может быть дробным или отрицательным, поэтому оно целое беззнаковое.

> По первоначальному замыслу, поле TTL в IPv4 отсчитывало именно время, а не шаги, чтобы заодно управлять интервалами таймеров на узле назначения. Так, в первом эталонном алгоритме сборки пакета IPv4 из фрагментов интервал таймера сборки был функцией от остаточного значения TTL [§3.2 RFC 791]. Однако впоследствии от этого подхода отказались в пользу других методов. В частности, значение тайм-аута сборки стало постоянным [§3.3.2 RFC 1122].

Правила работы с полем «предельное число шагов» [§3 RFC 2460] не отличаются от общепринятой практики IPv4:

- источник пакета помещает в это поле начальное значение, выбранное по своему усмотрению;

- каждый транзитный узел уменьшает значение этого поля на единицу и, если оно упало до нуля, уничтожает пакет и, возможно, шлет источнику аналог извещения ICMP;
- конечный адресат пакета игнорирует это поле.

В каких комментариях нуждаются эти простые требования?

- Во-первых, если узел-источник намерен передать пакет в сеть, то пакету предстоит хотя бы один шаг. Поэтому начальное значение поля «предельное число шагов» обязано быть больше нуля.
- Во-вторых, узел проверяет данное поле только после того, как он убедится, что пакет транзитный и подлежит дальнейшему продвижению. Напротив, если пакет только что создан текущим узлом или адресован ему, то узел это поле вообще не проверяет.
- В-третьих, у транзитного пакета это поле может оказаться нулевым, например, из-за сбоя на предыдущем узле или по злому умыслу отправителя. Тогда в беззнаковой 8-битной арифметике декремент нулевого значения даст положительное значение 255, хотя пакет явно подлежит уничтожению. Следовательно, на практике значение поля «предельное число шагов» надо проверить перед декрементом, а не после него. Если его значение до декремента один или ноль, то пакет надо уничтожить.

> Эталонная реализация IPv6 *KAME* [46] хитрит: она сначала сравнивает значение поля с единицей и только затем уменьшает его (или уничтожает пакет, если значение уже единица или меньше, то есть ноль). Алгоритм, приведенный в стандарте [§4.4 RFC 2460], тоже применяет этот трюк. Тем не менее, основной текст стандарта [§3 RFC 2460] обходит молчанием случай, когда поле «предельное число шагов» заранее содержит ноль. Это, конечно же, недоработка.

Поле длины пакета необходимо на тот случай, если канальный уровень не сохраняет точную длину полезной нагрузки кадра. Хрестоматийный пример — это Ethernet, где длина полезной нагрузки не может быть ниже 46 байт из-за наследия CSMA/CD. Другой возможный пример — это канальный протокол, длина полезной нагрузки в котором должна быть кратна, скажем, восьми байтам.

> Подобное ограничение можно встретить в защищенных протоколах с блочным шифрованием, но оно не всегда заметно их потребителям. Например, подсистема шифрования PPP маскирует ее, сохраняя и восстанавливая точную длину полезной нагрузки с помощью особой набивки, в которой закодирована информация о числе добавленных в конец полезной нагрузки байтов [§6.1 RFC 2419]. Располагая такой информацией, приемник PPP может точно «срезать» набивку без вреда для полезной нагрузки.

Трасса пакета заранее неизвестна, так что поле длины потребуется в каждом пакете — ведь никто знает, сквозь какие каналы ему придется пройти. Пусть единицей измерения длины пакета остается байт, а пакет состоит из целого числа байтов; ведь это очень удобно, учитывая ориентацию всей технологии обработки данных на байты.

> Хотя исторически байт не всегда был одной и той же длины, и потому возник альтернативный термин «октет», обозначавший именно цепочку из 8 бит, по состоянию на сегодняшний день это полные синонимы. Мы предпочитаем термин «байт», так как он понятен большей аудитории, а «октет» уже звучит старомодно, почти как французские реплики героев Льва Толстого.

При целом беззнаковом кодировании полю длины вполне хватит 16 битов, так как, с одной стороны, существующие канальные протоколы не поддерживают MTU больше $\sim 2^{16}$ байт, а с другой стороны, транспортные протоколы TCP и UDP рассчитаны на

[46] http://www.kame.net/

максимальную длину пакета $\sim 2^{16}$ байт. Кроме того, чем длиннее пакет, тем выше вероятность его повреждения при передаче. В этом-то и состоит работа пакетных сетей, чтобы даже огромные порции данных передавать в виде пакетов разумного размера.

> А в чем же заключается зависимость транспортных протоколов от максимальной длины пакета? Так, в заголовке UDP есть поле длины, и оно длиной 16 бит. Зависимость TCP не столь очевидна, поскольку TCP не хранит длину сегмента, полагаясь на IP в этом вопросе. Единственное ее проявление в заголовке TCP — это неотложный указатель (*urgent pointer*) длиной 16 бит, так что неотложные данные не смогут простираться далее $\sim 2^{16}$ байт от начала сегмента. Однако сама длина сегмента не может превышать 65535 байт ввиду 16-битного поля MSS в одноименной опции.
>
> Тем не менее, RFC 2675 предлагает опцию IPv6 «пакет-слон» (*jumbogram*), расширяющую поле длины до 32 бит. Тот же RFC обсуждает препятствия на пути к применению этой опции совместно с TCP и UDP. Эталонная реализация IPv6 *KAME* на всякий случай поддерживает опцию «пакет-слон».

Опций в пакете вполне может не быть, так что мы их отправим в необязательные заголовки расширения. В результате число и длина полей в заголовке IPv6 оказываются строго постоянными.

Мы уже заняли совершенно необходимыми полями 36,5 байта. Чтобы помочь современным вычислительным системам, давайте выровняем конец заголовка IPv6 на границу 32-битного слова. Мы не можем выбросить из заголовка даже полбайта, так что придется нарастить заголовок до 40 байт. Тогда конец заголовка IPv6 окажется выровнен на границу 64-битного слова, от чего особенно выиграют 64-разрядные платформы. После выравнивания у нас образуются незанятые три с половиной байта.

Давайте уступим моде на управление качеством обслуживания (QoS) и отдадим свободные разряды полям «класс трафика» (8 бит) и «метка потока» (20 бит). Первое из них аналогично полю TOS в IPv4, а второе позволяет, пометив последовательность пакетов одним и тем же значением, превратить их в единый объект политики обслуживания (поток). Уточнять подробности применения этих полей мы сейчас не будем.

> Как мы помним, сегодня общепринятая интерпретация поля TOS основана на DiffServ [RFC 2474] и ECN [RFC 3168]. То же самое касается и поля «класс трафика».
>
> Общие правила работы с полем «метка потока» изложены в [RFC 6437]. Одно из перспективных направлений его возможного применения — это оптимизация распределения нагрузки между каналами [RFC 6438] и серверами [draft-carpenter-v6ops-label-balance].
>
> Мы вовсе не хотели сказать, что управление качеством обслуживания в сети — это фикция или шарлатанство. Однако нельзя не признать, что слишком часто за него берутся именно из-за моды, без должного понимания его причин, законов и механизмов. По иронии, противники QoS зачастую столь же невинны, как и их оппоненты.

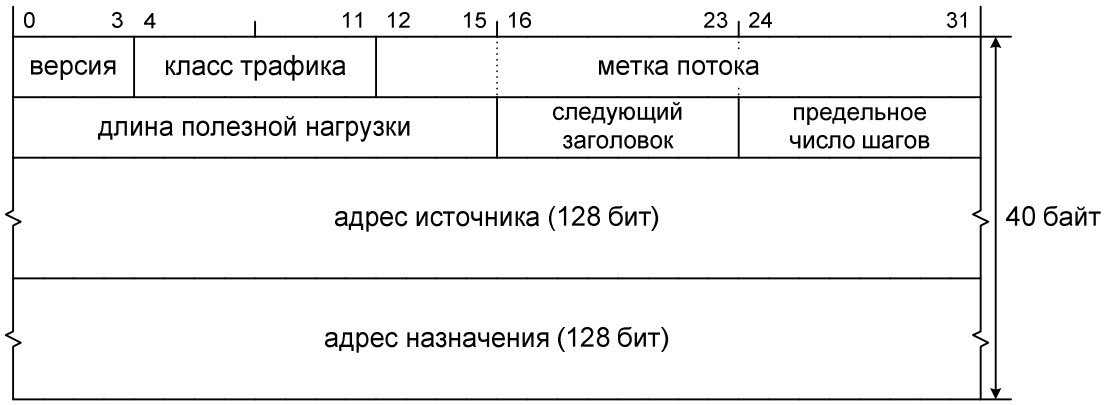
И, наконец, последний штрих: раз длина заголовка IPv6 всегда составляет 40 байт, то стоит ли включать эту величину в значение поля длины пакета? Пусть она входит туда неявно, и тогда максимальный пакет сможет стать длиннее на 40 байт. Так что давайте переименуем это поле в «длину полезной нагрузки» (*Payload Length*). В данном контексте «полезная нагрузка» — это пакет за вычетом основного заголовка IPv6; заголовки расширения входят в «полезную нагрузку». Максимальная длина такой «полезной нагрузки» — 65535 байт, а всего пакета — на 40 байт больше, то есть 65575 байт. Это ограничение чисто теоретическое, оно следует из того, как мы кодируем поле длины; об ограничениях практического характера мы поговорим в §6.5.

Сканируя заголовки, реализация IPv6 обязана следить, чтобы их цепочка не вышла за границы полезной нагрузки. Если байты полезной нагрузки уже исчерпаны, а заголовки еще не закончились, то пакет, очевидно, поврежден.

Теперь давайте составим полный, но еще неупорядоченный список полей в заголовке IPv6:

- версия IP (4 бита);
- адрес источника (128 бит);
- адрес назначения (128 бит);
- предельное число шагов (8 бит);
- длина полезной нагрузки (16 бит);
- следующий заголовок (8 бит);
- класс трафика (8 бит);
- метка потока (20 бит).

В каком порядке их лучше всего разместить? В современных вычислительных системах надлежащее выравнивание данных значительно повышает скорость доступа к ним, так что начать анализ нам надо с самых востребованных полей, а именно с адресов. В идеале, их надо было бы выровнять на границу 128-битного слова; однако тогда нам пришлось бы поместить их в самое начало заголовка, а эта позиция уже прочно занята коротеньким полем «версия IP». Значит, нам придется довольствоваться границей 64-битного слова, проходящей в заголовке по смещению 8 байт; то есть адреса отправляются в конец заголовка. Далее, поле «длина полезной нагрузки» следует выровнять на границу 16-битного слова. Для этого достаточно поместить между ним и полем «версия IP» поля качества обслуживания. Наконец, в зазор между полем «длина полезной нагрузки» и адресами умещаются поля «следующий заголовок» и «предельное число шагов». Вот мы и получили полный формат заголовка IPv6 [§3 RFC 2460], показанный на Фиг. 26.



Фиг. 26. Основной заголовок IPv6

Какие поля, знакомые нам по IPv4, не вошли в заголовок IPv6? Во-первых, больше не нужна длина заголовка: она стала постоянной величиной 40 байт. Заголовок IPv4 был переменной длины, так как включал в себя опции; теперь же опции IPv6 займут отдельный заголовок расширения (или даже несколько таких заголовков — см. §3.3.2 и §3.3.6).

Во-вторых, исчезли все поля, относящиеся к фрагментации: идентификатор, смещение, флаги. Так как большинство пакетов при оптимальной работе сети не фрагментировано, эти поля стоит вынести в собственный заголовок расширения. О факте фрагментации будет говорить само наличие такого заголовка. А как же флаг **DF**, запрещающий фрагментацию? Вопрос резонный; о судьбе флага **DF** мы скажем в §3.3.4 и §6.5.

В-третьих, совсем пропала контрольная сумма заголовка. Это решение вызвано несколькими причинам. Прежде всего, модульная структура заголовков в пакете IPv6 размывает границу между их уровнями: если конечный адресат пакета еще может

разобрать все заголовки сетевого уровня и определить, где начинаются данные следующего уровня, например, TCP, то транзитный узел не может и не должен делать этого. Тогда контрольная сумма уже не может защищать все заголовки сетевого уровня, и область ее покрытия придется ограничить только основным заголовком IPv6. Так само ее существование в значительной мере теряет смысл. Затем, современные канальные технологии сами обеспечивают пошаговую защиту пакетов от повреждения, а протоколы транспортного уровня защищают важнейшие поля заголовка IP сквозным образом. Благодаря этому, собственная роль контрольной суммы IP не так уж велика. И, наконец, отказ от контрольной суммы IP ускоряет аппаратное продвижение пакетов и делает проще соответствующие интегральные схемы.

> Пусть читатель изобразит форматы заголовков IPv4 и IPv6, а затем раскрасит их поля в зависимости от того, сохранилось ли поле при смене версии IP, исчезло или возникло.
>
> Еще одним упражнением будет выяснить, какие поля заголовка окажутся повреждены, если модуль маршрутизатора «забудет» проверить поле «версия» и обработает пакет IPv6, как если бы это был пакет IPv4. Рассмотрите два конкретных сценария: а) декремент поля TTL перед продвижением пакета; б) перезапись значения DSCP правилом QoS. (Не забудьте, что в обоих случаях также произойдет пересчет контрольной суммы заголовка.) Затем определите, какие поля заголовка будут повреждены, если пакет IPv4 будет по ошибке обработан согласно формату заголовка IPv6. При этом ограничьтесь одним сценарием: декремент поля «предельное число шагов».

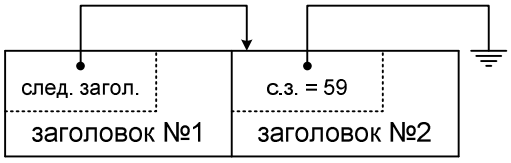
3.3. Заголовки расширения

3.3.1. Заголовок, которого нет

Представим себе пакет IPv6, который состоит из одного основного заголовка. Хотя нам не совсем ясно, зачем такой пакет мог бы понадобиться, формального повода запретить его у нас тоже нет.

> Обсуждая заголовки IPsec в §3.3.5, мы найдем применение даже такому вот «пустому» пакету: обеспечение конфиденциальности на уровне потока пакетов.

Однако само существование пакета без полезной нагрузки поднимает любопытный вопрос: чему в нем должно быть равно значение поля «следующий заголовок»? Мы могли бы сказать, что оно неважно, потому что длина полезной нагрузки равна нулю; но зачем нам лишняя неоднозначность? Чтобы не провоцировать разночтения стандарта и несовместимость реализаций, давайте назначим этому случаю вполне определенный код из реестра протоколов IP. Это код 59, «следующего заголовка нет» (*No Next Header*) [§4.7 RFC 2460]. Таким кодом можно завершить и цепочку заголовков расширения, если за ней в пакете ничего нет, как это показано на Фиг. 27.



Фиг. 27. Пакет без полезной нагрузки

Теперь представим себе, что получатель сканирует пакет и обнаруживает заголовок со значением 59 в поле «следующий заголовок», а затем еще какие-то байты (об этом скажет длина полезной нагрузки IPv6). Хотя такая ситуация неоднозначна, в ней нет явного криминала: она не ведет к осложнениям или опасным последствиям, если обработать ее правильно. А принцип Постела подсказывает, как именно следует поступить: пусть транзитные узлы сохраняют эти концевые байты, а конечный адресат игнорирует их [§4.7 RFC 2460].

Напомним себе современную трактовку **принципа Постела** (*Postel's principle*), также известного как **принцип устойчивости** (*robustness principle*). Основоположник *Internet* Джонатан Постел (*Jonathan Postel*) сформулировал этот принцип одной лаконичной фразой: «*Будь консервативен в собственных действиях и либерален к тому, что принимаешь от других*» [§2.10 RFC 793]. Сегодня мы добавим к этому: «*…но никогда не забывай о безопасности*». Этот принцип требовательности к себе и терпимости к другим — один из главных в TCP/IP. Если сторона протокола следует ему, она не только соблюдает протокол, но также приобретает гибкость и устойчивость к неполадкам, а эти качества особенно важны, когда разработчики протокола не могут заранее предусмотреть все варианты развития событий.

3.3.2. Опции

Мы предварительно отметили в §3.1, что опции можно поделить на пошаговые и актуальные только для адресата. Требования к обработке этих видов опций существенно разнятся. Во-первых, маршрутизаторы в целях производительности должны сканировать только пошаговые опции. Во-вторых, когда у IP появятся собственные механизмы сквозной защиты, опции адресата можно будет шифровать. В-третьих, при фрагментации пакета пошаговые опции надо целиком копировать в каждый фрагмент, тогда как опции адресата достаточно привести один раз и даже можно разбить на части, если они окажутся слишком длинными и пересекут границы одного фрагмента.

Пока что мы лишь излагаем общие соображения, а точные правила фрагментации заголовков мы сформулируем, когда будем готовы к этому, в §3.3.4.

По всей видимости, каждый вид опций заслуживает собственного заголовка расширения. Форматы этих заголовков могут полностью совпадать, но им будут отвечать разные коды «следующий заголовок». Так, заголовок опций адресата получает код 60 [§4.6 RFC 2460].

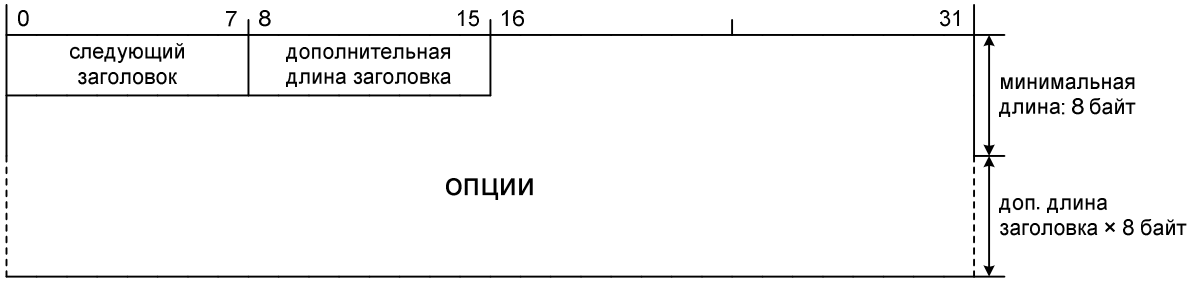
Заголовок пошаговых опций, ввиду своей особой роли, получает отличительный код 0 [§4.3 RFC 2460] и позицию непосредственно после заголовка IPv6. Иными словами, код 0 может встретиться *только* в поле «следующий заголовок» основного заголовка IPv6. Если же пакет составлен правильно, но первое поле «следующий заголовок» ненулевое, то в пакете пошаговых опций нет. С помощью этой простой логики маршрутизаторы могут быстро обнаруживать пошаговые опции в транзитных пакетах.

Пусть по своему формату заголовки опций будут контейнерами, в которые можно вложить больше одной опции. Если для отдельных опций мы применим кодирование TLV, то в фиксированной части заголовка нам потребуются только поля «следующий заголовок» и «длина» — ведь число опций может быть переменным.

Будь у нас такое желание, мы могли бы обойтись без поля длины за счет подходящего кодирования опций. Но благодаря полю длины мы сможем просто и однозначно указать конец списка опций, так что давайте не будем создавать себе трудности. Кроме того, поле длины позволяет «перескочить» через заголовок опций, не сканируя его.

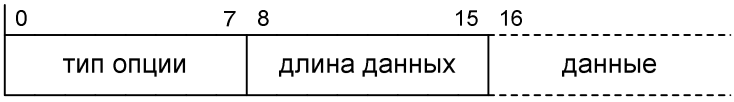
Поле «следующий заголовок» занимает один байт. Пусть поле «длина» тоже займет один байт, и тогда действительное начало опций будет выровнено на границу 16-битного слова. Код обоих полей будет целый беззнаковый. Но не слишком ли мало 255 байт для всего списка опций? Удлинить его можно, если измерять длину не в байтах, а в бо́льших единицах. Заодно так мы выровняем конец заголовка опций. Конец заголовка IPv6 выровнен на границу 64-битного слова, и большего выравнивания мы уже не достигнем. Поэтому пусть длина заголовка опций измеряется в 8-байтных единицах. Еще заметим, что длина такого заголовка не меньше единицы из-за обязательных полей. Эту единицу можно учесть неявно и выиграть еще 8 байт. То есть поле «длина» заголовка опций ведет учет 8-байтных отрезков, начиная со второго. Тогда длина всего заголовка в байтах может

меняться от 8 до 2048 с шагом 8 байт. Окончательный формат заголовка опций показан на Фиг. 28.



Фиг. 28. Заголовок опций IPv6

Сами опции мы станем кодировать широко известным способом TLV: поле типа, поле длины, данные, — как показано на Фиг. 29. Поля типа и длины данных занимают по одному байту. Длина данных измеряется в байтах и, по традиции IPv6, не включает фиксированные поля типа и длины.



Фиг. 29. Общий вид опции IPv6

Когда получатель пакета сканирует список опций, то он вполне может встретить незнакомую опцию. Благодаря одинаковому кодированию TLV, через опцию-незнакомку легко «перескочить»; но где гарантия, что ее можно безопасно игнорировать? Чтобы избежать этой неоднозначности, давайте сразу же закодируем требуемое поведение в старших двух битах типа опции, как показано в Табл. 8.

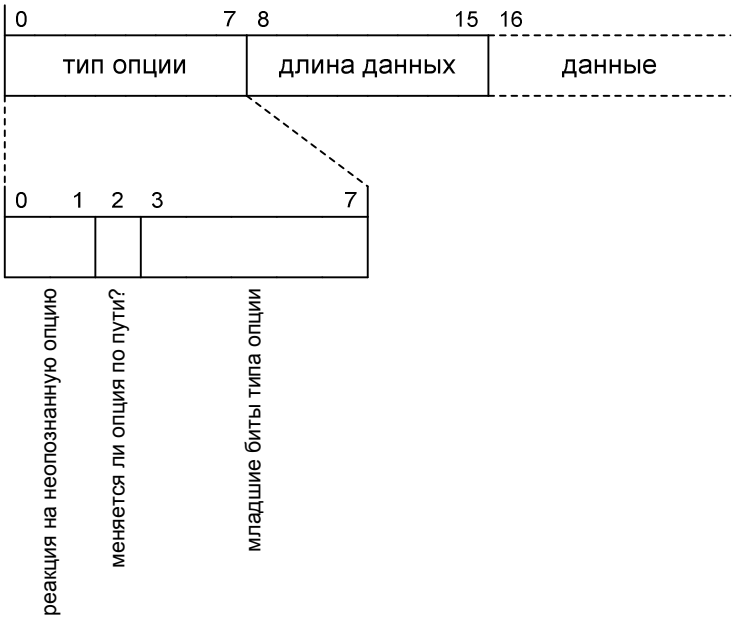
Табл. 8. Смысл старших битов в численном коде опции IPv6

Биты	Предписание
00	«Перескочить» через незнакомую опцию и продолжить обработку заголовка
01	Уничтожить пакет; не высылать никаких извещений
10	Уничтожить пакет; выслать извещение «ICMP версии 6» (независимо от адреса назначения исходного пакета)
11	Уничтожить пакет; выслать извещение «ICMP версии 6», только если адрес назначения исходного пакета не групповой

Здесь мы вполне обоснованно предположили, что у IPv6 будет управляющий протокол, аналогичный ICMP. Над ним мы поработаем в §4.3. Он и правда будет называться ICMPv6.

Еще одна трудность возникнет, когда на уровне IP появится сквозной механизм защиты пакетов от умышленной модификации. Сможет ли он защищать опции? Если транзитные узлы модифицируют защищенную опцию, то конечному адресату пакет покажется подложным. Напротив, если опция по своему определению не изменяется в пути, то она вполне подлежит защите. Отражает это свойство опции третий по старшинству бит ее типа: если он сброшен, то опция заведомо не меняется по пути, а если установлен, то может меняться. Теперь механизму защиты не нужно знать свойства отдельных опций — ему достаточно проверить этот бит в поле типа, чтобы узнать, можно ли защищать данную опцию.

Например, реакция механизма защиты IPsec AH [RFC 4302] выглядит так: если данный бит установлен, то механизм подставляет вместо данных опции нулевые байты, когда вычисляет для пакета его защитную цепочку. Таким образом, под защитой AH оказываются хотя бы код и значение длины, а также фактическая длина опции. Предполагается, что длина опции не меняется, а алгоритм защиты чувствителен к числу нулевых байтов, идущих подряд.



Фиг. 30. Кодирование свойств опции в ее типе

В трех старших битах типа (см. Фиг. 30) закодированы неотъемлемые свойства каждой опции, и посему эти биты нельзя рассматривать как отдельное поле. Когда новой опции назначают численное значение типа, то в нем сразу же учитывают ее свойства. Например, если мы увидим опцию с десятичным кодом 95, то немедленно[47] скажем о ней следующее:

- транзитные узлы могут модифицировать эту опцию;
- узел, который не опознал эту опцию, должен молча уничтожить пакет.

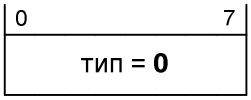
Если же мы, к примеру, инвертируем в этом коде бит «меняется по пути», то получим совершенно другой тип опции, никак не связанный с первоначальным.

Вычислите десятичное значение кода новой опции. (127)

О поле типа опции осталось сказать, что пошаговые опции и опции адресата хранятся в пакете раздельно, но получают численные коды из одного и того же реестра [[48]], так что один код не может означать две разных опции.

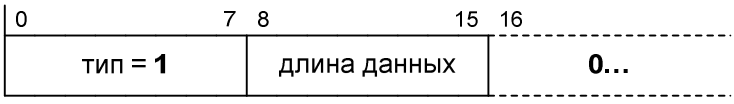
Суммарная длина списка опций может быть любой с точностью до байта, тогда как заголовок опций должен занимать число байт, кратное восьми, и оставляет $8N - 2$ байт на, собственно, опции. Кроме того, для некоторых опций может быть желательно то или иное выравнивание относительно начала заголовка. Чтобы удовлетворить эти требования, давайте «создадим» две опции-пустышки, обязательных к реализации и опознанию.

Первая из них, **Pad1**, в виде исключения займет ровно один байт, к чему и сводится вся ее функция (см. Фиг. 31). Формально тип опции **Pad1** — 0, но она нарушает формат TLV: в ней нет полей, кроме поля типа.



Фиг. 31. Опция Pad1

Вторая же опция, **PadN**, вполне отвечает формату TLV (см. Фиг. 32). Ее тип — 1, и она может занять от 2 до 257 байт. Если ее поле «длина» больше нуля, то пусть поле данных ради единообразия будет заполнено нулевыми байтами.



Фиг. 32. Опция PadN

[47] Точнее, как только справимся с переводом числа 95 в двоичное представление.

[48] http://www.iana.org/assignments/ipv6-parameters

Согласно своей роли, опции **Pad1** и **PadN** могут встречаться в заголовке опций несколько раз и располагаться в начале заголовка, в его конце, или же перемежать другие опции. Они могут возникать и среди пошаговых опций, и среди опций адресата.

> Проанализируйте свойства опций **Pad1** и **PadN**, закодированные в старших битах их типов.

Все остальные опции IPv6 — суть расширения базового протокола; их технические условия приведены в отдельных документах RFC. Поэтому сейчас мы их рассматривать не будем, но надеемся, что теперь читатель сможет разобраться в них самостоятельно, когда ему это понадобится.

> В §6.4 нам встретится опция «сигнал маршрутизатору».

3.3.3. Маршрутный заголовок

Важное следствие сквозной модели, принятой в *Internet*, состоит в том, что конечные узлы явным образом не участвуют в маршрутизации пакетов. Источник создает пакет IP и отправляет его в сеть, а та уже сама выбирает подходящую трассу, чтобы пакет дошел до адресата. Конечно, инженер понимает, что простирающаяся между конечными узлами сеть состоит из каналов и маршрутизаторов, работающих по вполне определенным правилам, однако узлу-источнику это неинтересно; все, что от него требуется, — это создать пакет и выбрать для него первый шаг. В этой модели источник управляет дальнейшей маршрутизацией пакета только неявно, устанавливая в нем адрес назначения.

> В общем случае маршрутизаторы IP могут рассматривать и другие поля заголовка и параметры пакета, например, класс обслуживания или полную длину. Это естественное обобщение простого алгоритма маршрутизации, ответ которого (следующий шаг пакета) зависит только от адреса назначения. Такую обобщенную маршрутизацию часто называют **маршрутизация согласно политике** (*policy routing*). Тем не менее, и в этом случае источник пакета влияет на его трассу только неявно.

Тем не менее, еще давным-давно разработчики IPv4 подумали: а вдруг источник иногда бывает осведомлен лучше сети? Например, из-за сбоя или атаки протокол маршрутизации распространил ложные сведения, и маршрутизаторы направляют пакеты не туда, куда надо. Тогда источник мог бы подсказать им верную трассу для своих пакетов, включив эту информацию в заголовок IP. Так появились опции IPv4 LSRR (нестрогая маршрутизация от источника) и SSRR (строгая маршрутизация от источника) [§3.1 RFC 791].

> Если обратиться к первоисточнику [49], то можно выделить две задачи маршрутизации от источника в IP. Первая из них состоит в том, чтобы направить пакет по определенной трассе независимо от состояния транзитных узлов. Иными словами, в трассу пакета надо включить желательные транзитные узлы в определенном порядке. Вторая же задача сводится к тому, чтобы исключить из трассы нежелательные маршрутизаторы или сети, направив пакет в обход них. Техническое решение этих задач одинаково и сводится к перечислению транзитных узлов, через которые должен проследовать пакет. Дело в том, что указание транзитному узлу вида: «Ни в коем случае не продвигай пакет через следующий шаг X», — слабо выполнимо в рамках маршрутизации IP, поскольку у транзитного узла может не быть других вариантов. По этой причине исключение узлов из трассы пакета достигается путем включения в нее других, альтернативных узлов.

[49] J. Postel, J. Reynolds. Comments on the IP Source Route Option. ISI, 1987. http://www.rfc-editor.org/in-notes/museum/ip-source-route-comments.txt

Как мы помним, суть работы SRR была довольно простой. Допустим, источник И хотел направить пакет конечному адресату К через транзитные узлы T_1, T_2, T_3,…T_N. Тогда он составлял заголовок пакета так:

- помещал адрес T_1 в поле «адрес назначения»;
- помещал список адресов {T_2, T_3,…T_N, К} в опцию SRR;
- устанавливал указатель опции SRR так, чтобы тот указывал на первый адрес в списке, то есть T_2.

В результате узел T_1 значился адресатом пакета и поэтому получал его первым. Он поступал с заголовком так:

- определял, на какой адрес в списке указывает указатель опции SRR — T_2;
- копировал адрес T_2 в поле «адрес назначения»;
- определял выходной интерфейс пакета по его новому адресу назначения;
- записывал локальный адрес выходного интерфейса на место T_2 в опции SRR;
- увеличивал указатель на один адрес.

Далее пакет приходил узлу T_2, и тот действовал аналогичным образом… И так далее, пока пакет не добирался наконец до узла К. Разница между строгим и нестрогим режимами состояла в том, мог ли пакет посещать транзитные узлы вне заданного источником списка (в нестрогом мог, а в строгом не мог).

Сегодня мы спросили бы разработчиков этого механизма: а почему сеть должна *настолько* доверять источнику? Действительно, злонамеренный источник вполне может использовать опции SRR, по меньшей мере, для двух типов атак:

- Обход системы безопасности. Допустим, сеть спроектирована так, чтобы весь трафик проходил через один или несколько узлов-контролеров, например, межсетевых экранов, которые анализируют трафик и применяют к нему ту или иную политику. Но с помощью маршрутизации от источника злоумышленник сможет направить трафик в обход межсетевых экранов, если в физической топологии сети есть такая лазейка.
- Усиление атак типа «отказ в обслуживании» [[50]]. Положим, злоумышленник хочет подавить маломощный узел А потоком пакетов. Если он включит в каждый пакет опцию LSRR со списком вида «Б, А, Б, А, Б, А…», где Б — сторонний узел, то каждый пакет посетит А не один раз, а несколько! В результате злоумышленнику достаточно создать поток пакетов в N раз меньше, а значит, этот поток может пройти незамеченным через межсетевые экраны, реагирующие на слишком большой трафик между парой узлов.

Пусть читатель сам вычислит максимально достижимый коэффициент усиления атаки N_{max} в IPv4.

Решение: Максимальная длина заголовка IPv4 — 60 байт (4-битное поле, 4-байтная единица длины). Из них 20 байт занимает обязательная часть, так что на опции остается самое большее 40 байт. Заголовок LSRR занимает 3 байта, но наименьшее допустимое значение смещения — 4 байта [§3.1 RFC 791], поэтому список адресов занимает до 36 байт, а это означает 9 адресов IPv4. Еще один адрес — это адрес назначения в основном заголовке. Так что каждый из адресов А и Б может встретиться самое большее 5 раз. Это и будет искомое значение N_{max}.

Поэтому маршрутизация от источника в глобальной сети приносит больше вреда, чем пользы, и было бы неразумно повторить в IPv6 ошибки IPv4. Из этого затруднения мы выкрутимся вот как: давайте в самом общем виде зададим гибкий формат заголовка,

[50] P. Biondi, A. Ebalard, "IPv6 Routing Header Security", CanSecWest Security Conference 2007, April 2007. http://www.secdev.org/conf/IPv6_RH_security-csw07.pdf

управляющего маршрутизацией пакета IPv6, а всю конкретику оставим на усмотрение дополнительных протоколов. Тогда, в случае чего, нас винить будет не в чем.

Искомый заголовок расширения IPv6 мы назовем **маршрутный заголовок** (*routing header*) [§4.4 RFC 2460]. Ему отвечает значение «следующий заголовок» 43 — разумеется, в предшествующем ему заголовке, а не в нем самом.

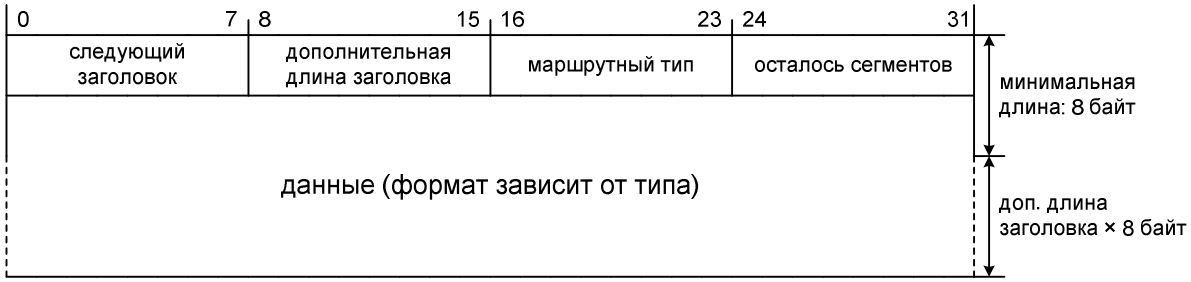
Так как длина маршрутной информации заранее неизвестна, маршрутный заголовок будет переменной длины. Значит, помимо обязательного поля «следующий заголовок», маршрутному заголовку потребуется поле длины. По аналогии с заголовками опций из §3.3.2, пусть это поле длиной один байт хранит длину в 8-байтных словах, причем первое слово не учитывается, так как оно всегда присутствует и частично занято фиксированной частью заголовка.

Далее мы поместим поле маршрутного типа, которое и определит, как надо трактовать дальнейшие данные в этом заголовке. Ему будет вполне достаточно одного байта. Конечно же, для общепринятых значений этого поля понадобится реестр [51].

Как быть получателю пакета, если он встретил неизвестный маршрутный тип? Придется ли ему безусловно отбросить этот пакет? Ответить на этот вопрос нам поможет опыт маршрутизации от источника IPv4. (Да, учиться можно и на ошибках!) Когда список транзитных узлов в опции SRR исчерпан и пакет направляется к конечному адресату К, опция SRR отработала и уже не имеет значения для маршрутизации пакета. В этом случае опцию SRR можно игнорировать.

Чтобы перенести эту схему в маршрутный заголовок IPv6, нам придется принять определенную модель его данных. Пусть они состоят из последовательности сегментов, где каждый сегмент может быть, например, адресом транзитного узла. Как обозначить в этой модели, что все сегменты уже обработаны и дальнейшие получатели пакета могут игнорировать маршрутный заголовок, независимо от его типа? Вот одно из возможных решений: хранить в фиксированной части маршрутного заголовка число еще необработанных сегментов. Когда значение этого поля упадет до нуля, получатель пакета надежно определит, что маршрутный заголовок уже отработал свое, даже если получатель не знаком с этим маршрутным типом и не знает, как устроен его сегмент. Если же это поле ненулевое, то пакет с незнакомым типом маршрутного заголовка придется отбросить.

В результате мы приходим к формату маршрутного заголовка, изображенному на Фиг. 33.



Фиг. 33. Маршрутный заголовок

Из уже существующих маршрутных типов мы упомянем типы 0 (т.н. **RH0** — *Routing Header 0*) и 2 (**RH2**). RH0 — это прямой аналог LSRR. Неудивительно, что в конце концов его упразднили ради безопасности [RFC 5095]. Об RH2 см. примечание в конце раздела.

> Одно отличие RH0 от LSRR состояло в том, что в текущий сегмент RH0 переносился предыдущий адрес назначения из основного заголовка IPv6, а не адрес выходного интерфейса. Таким образом, в RH0 достаточно было поменять местами адрес назначения и адрес из текущего сегмента, а затем провести декремент счетчика «осталось сегментов». Как следствие, в RH0 происходит простой

[51] http://www.iana.org/assignments/ipv6-parameters

циклический сдвиг списка адресов, тогда как опция LSRR записывала фактические интерфейсы, в которые продвигался пакет. Точно так же поступала опция SSRR, а нужно это было именно для нее, чтобы потом можно было послать ответный пакет с записанным строгим маршрутом от источника.

При должной политике безопасности на границах опорной сети, запрещающей проникновение сомнительных RH0 извне, внутри такой сети RH0 мог бы стать ценным инструментом для обнаружения скрытых дефектов. Ведь с его помощью можно было бы из одной точки проверять разные пути сквозь сеть, включая неактивные в данный момент, запасные. Корреляция результатов подобных проверок вдоль множества путей позволяет довольно точно определить, где локализован дефект [52]. Так что приходится признать, что RH0 был упразднен не по причине его бесполезности или неизлечимых проблем, а из-за низкой культуры сетевой безопасности, по-прежнему довлеющей над *Internet* и делающей данный инструмент опасным для его же владельца.

Существование маршрутного заголовка IPv6 заставляет нас пересмотреть такое понятие, как адресат пакета. До сих пор мы полагали, что у пакета ровно один адресат, хотя он может быть индивидуальным узлом или группой. Теперь же нам придется различать два типа адресатов. Если в пакете вообще нет маршрутного заголовка или же он есть, но его поле «осталось сегментов» равно нулю, то в основном заголовке IPv6 записан **конечный адресат** пакета. Это узел, который поглотит пакет и обработает его полезную нагрузку. Если же в маршрутном заголовке все еще есть активные сегменты, то в основном заголовке IPv6 значится **текущий адресат** пакета. Этот узел примет пакет, только чтобы сменить в нем адрес назначения и передать дальше; он будет транзитным на пути пакета. При этом конечный адрес назначения пакета скрыт в сегменте маршрутного заголовка.

Вообще говоря, ни один из адресов назначения в пакете с маршрутным заголовком не должен быть групповым. Это следует из соображений безопасности.

Нужны ли дополнительные правила, если у адреса назначения в маршрутном заголовке область действия не глобальная? На самом деле, нет, достаточно уже знакомого нам принципа изоляции зон из §2.4. Однако, строго говоря, этот принцип должен выполняться — а значит, и проверяться — для всех адресов назначения одновременно. В противном случае возможна атака засылкой пакета в удаленную зону. Например, если текущий адресат пакета глобальный, а конечный адресат внутриканальный, то произойдет засылка внутриканального пакета в канал, не подключенный непосредственно к источнику пакета.

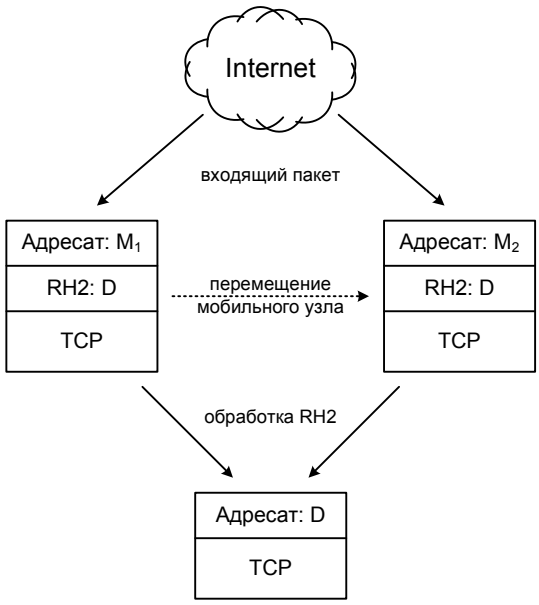
Нормативный документ [§9 RFC 4007] требует, чтобы следующий адрес назначения из маршрутного заголовка имел область не меньшей величины, чем у текущего адреса назначения в основном заголовке IPv6. Пока принцип изоляции зон выполняется и все интерфейсы назначения пакета находятся в зоне его адреса источника, это требование избыточно. К примеру, вполне возможна «эстафета» по адресам, область которых сужается, если все они назначены интерфейсам в тот же канал, где находится интерфейс-источник.

Маршрутизация от источника обрела новую жизнь в MPLS-TE, где вполне можно задать трассу LSP по узлам, перечислив их адреса в ERO. Конечно, этот механизм выгодно отличается от RH0 в IPv6 в плане безопасности, так как пограничный узел (LER), устанавливающий LSP, — заведомо доверенный, в отличие от абстрактного источника IPv6. Однако на поверку MPLS не столь уж независим от IP: не имея собственной адресации, он опирается в этом на традиционный протокол сетевого уровня, и обычно это IP. Поэтому, если узлам MPLS назначены зонные адреса IPv6, то ERO подчиняется все тому же принципу

[52] N. Guilbaud. Google Backbone monitoring. Localizing packet loss in a large complex network. AusNOG 2012.

изоляции зон! А именно входной LER должен иметь возможность адресовать каждый узел по трассе LSP, иначе ERO будет лишен смысла. В частности, именно поэтому нельзя проложить трассу LSP через несколько каналов, используя только внутриканальные адреса: для составления ERO потребуются адреса большей области, по крайней мере, начиная со второго узла вдоль трассы. Это в высшей степени любопытно, что наши разработки для маршрутизации от источника IPv6 также применимы к MPLS!

Как мы знаем, узел может адресовать пакеты IP самому себе. Точно так же, текущий и конечный адреса назначения могут принадлежать одному и тому же узлу. Но зачем это нужно? Например, чтобы текущий адрес назначения обладал свойствами локатора, а конечный — идентификатора. Это полезно, если узел мобильный и перемещается по *Internet*. Текущий адрес мобильного узла *M* в каждый момент времени доступен обычным порядком, с помощью маршрутизации, и может меняться, когда узел меняет свое местоположение. В то же время, его постоянный адрес *D*, с которым работают вышестоящие протоколы, доступен только посредством маршрутного заголовка. Тогда перемещения узла будут прозрачны для протоколов уровня выше IP, как показано на Фиг. 34. Очевидно, что такой маршрутный заголовок может содержать ровно один сегмент, причем адрес в нем обязан совпасть с постоянным адресом текущего адресата *D*. В противном случае налицо сбой или нарушение безопасности. Даная разновидность маршрутного заголовка получила маршрутный код 2 и потому известна как RH2. Подробнее с ней можно познакомиться в RFC 3775, посвященном мобильности узлов IPv6.



RH2 = маршрутный заголовок, тип 2
M_1, M_2 = текущие адреса мобильного узла
D = постоянный адрес мобильного узла

Фиг. 34. Принцип работы RH2

3.3.4. Заголовок фрагмента. Фрагментация и сборка в IPv6

Как мы помним, в IPv4 модуль фрагментации был обязательным компонентом стека и в хостах, и в маршрутизаторах, потому что фрагментировать длинный пакет должен был узел, стоящий непосредственно на входе в канал с низким MTU. Это усложняло логику маршрутизаторов IPv4: они не могли рассматривать транзитный пакет как элементарную единицу обмена данными, которую надо или целиком продвинуть дальше, или отбросить. Кроме того, нагрузить маршрутизаторы фрагментацией было не самой лучшей мыслью, потому что они обрабатывают трафик многих хостов, а сами

хосты обслуживают только собственные нужды (по крайней мере, на сетевом уровне стека).

Вспоминается пословица: «На одного с сошкой семеро с ложкой».

Теперь, когда у нас есть приятная возможность перевернуть все вверх дном, мы можем «примерить» на каркас IPv6 альтернативные правила фрагментации:

1) провести фрагментацию пакета может только его источник;
2) транзитные узлы продвигают фрагменты, не отличая их от цельных пакетов;
3) собирает пакет из фрагментов по-прежнему конечный адресат.

Так вся работа по фрагментации ложится на источники пакетов, что не только справедливо, но и выгодно с точки зрения производительности сети.

К сожалению, RFC 2460 нигде явно не говорит, что фрагменты собирает именно конечный адресат, хотя это косвенно следует из рекомендованного порядка заголовков расширения [§4.1 RFC 2460], который мы обсудим в §3.3.6.

Но не стоит сомневаться, что новый подход к фрагментации создаст дополнительные проблемы и поставит неожиданные вопросы на уровне работы протокола. Что ж, от этого никуда не уйти, такова цена прогресса. Нам остается только проанализировать возникшие затруднения и найти решения для них.

Если длина пакета L (цельного или фрагмента) превращается в сквозную характеристику,[53] то пакет может достичь конечного адресата только при очевидном условии:

$$L \le PMTU\ . \qquad (1)$$

Здесь $PMTU$ — MTU трассы от данного источника до конечного адресата пакета:

$$PMTU = \min_i MTU_i\ , \qquad (2)$$

где индекс i пробегает все выходные интерфейсы этой трассы.

В новых условиях сквозную величину L задает источник пакета, так что выполнение условия (1) становится его личной заботой, а это связано с поиском значения $PMTU$, или хотя бы его оценки снизу $\underline{PMTU}$:

$$L \le \underline{PMTU} \le PMTU \Rightarrow L \le PMTU\ . \qquad (3)$$

Простые арифметические соображения подсказывают нам, что есть две стратегии выполнения условия (3). Первая из них состоит в том, чтобы источник приложил усилия и нашел хорошее приближение PMTU, вплоть до точного значения. Для этого существует процедура PMTUD. Чтобы она работала, источник должен получать извещения, когда его пакеты уничтожаются из-за чрезмерной длины. В IPv4 эту роль выполняли сообщения ICMP «пакет слишком велик»; подобный механизм нам понадобится и в IPv6. Сразу же заметим, что PMTUD сходится быстрее, если каждое извещение подсказывает очередную величину MTU_i: узел, высылающий это извещение, включает в него значение MTU того интерфейса, куда вышел бы пакет, если бы он не оказался слишком длинным. В IPv4 это было необязательным расширением [§4 RFC 1191], но теперь нам стоит ввести это в стандарт — отметим себе на будущее (§4.3). Далее, PMTUD предполагает, что протокол более высокого уровня устойчив или безразличен к потере пакетов. Это условие не ново, потому что пакеты в сетях TCP/IP теряются по самым разным причинам. Наконец, PMTUD работает *надежно* только при том условии, что хотя бы часть извещений доходит до источника, или же что вышестоящий протокол предполагает отклик удаленной стороны. В противном случае возникает «черная дыра PMTUD», куда бесследно, с точки зрения источника, пропадают длинные пакеты.

Для образования классической «черной дыры PMTUD» достаточно блокировать извещения ICMP, однако для двунаправленного протокола не все еще

[53] Характеристика, которая не меняется на всем пути от источника до конечного адресата.

потеряно. Отклик удаленной стороны (точнее, его неожиданное отсутствие) позволяет обнаружить и обойти «черную дыру», уменьшив длину пакета. Если же вышестоящий протокол однонаправленный, такой как Syslog или Netflow, то «черная дыра» станет поглощать пакеты до тех пор, пока извещения ICMP не будут разблокированы.

Вторая стратегия значительно проще и потому подходит для источников, которые не хотят или не могут обременять себя процедурой PMTUD. Она заключается в том, чтобы использовать минимальное значение MTU, указанное протоколом IP, как «габаритные ворота» для исходящих пакетов и их фрагментов:

$$L \le MTU_{\min} . \quad (4)$$

Этот трюк работает, потому что минимальное значение MTU служит оценкой снизу для MTU любой трассы, пока не нарушен протокол:

$$\forall i, MTU_{\min} \le MTU_i \Rightarrow MTU_{\min} \le PMTU . \quad (5)$$

Чтобы при этом не страдала эффективность передачи данных, минимальное значение MTU должно быть разумно большим. Как мы помним, в IPv4 это была микроскопическая величина 68 байт. Теперь же, в IPv6, MTU канала не может опуститься ниже 1280 байт [§5 RFC 2460].

Мы точно не знаем, почему была выбрана именно величина 1280 байт. Из ее очевидных численных свойств можно заметить разве что такое: $1280 = 2^{10} + 2^8$. Но практически важное свойство данного выбора состоит в том, что он оставляет весьма приличный допуск на многоуровневую инкапсуляцию, например, туннели, при распространенном значении MTU физического канала 1500 байт.

А как нам быть с каналом, который по своему устройству неспособен переместить пакет длиной 1280 байт? (Например, см. [RFC 4919].) Пусть расширение эффективного MTU такого канала до требуемой величины, например, за счет фрагментации кадров, будет задачей нижележащего уровня. Обычно это часть протокола канальной инкапсуляции (пример см. в [RFC 4944]).

Выше, в §3.2, мы спросили себя, куда из заголовка IPv6 потерялся флаг **DF**. Теперь мы видим, что необходимость в нем отпала. Можно сказать, что флаг **DF** неявно установлен в каждом пакете IPv6, потому что транзитные узлы фрагментировать его не могут.

Остальные поля, управляющие сборкой фрагментов, мы решили переместить в отдельный заголовок расширения. Он называется **заголовок фрагмента** (*fragment header*). Его код в реестре протоколов IP — 44. Если в пакете IPv6 этот заголовок есть, то перед нами фрагмент, а если нет, то цельный пакет.

Так как собирает пакет только его конечный адресат, каждый фрагмент должны сопровождать служебные сведения, представляющие интерес для транзитных узлов. Это необходимо, потому что каждый фрагмент — это тоже пакет IPv6, путешествующий сквозь сеть независимо от других фрагментов.

Какие это будут сведения? В первую очередь, это основной заголовок IPv6, так как он содержит важнейшие сведения о пакете. Далее, пошаговые опции по самому их смыслу также интересны всем узлам на пути фрагмента. Маршрутный заголовок, если он есть, заведомо нужен промежуточным адресатам, хотя маршрутизаторы его не рассматривают. Наконец, об опциях адресата мы заранее не можем сказать, будут ли они нужны промежуточным адресатам. Возможно, появятся опции двух видов: первые будут интересны всем адресатам подряд, а вторые — только последнему из них, то есть конечному. Тогда будет достаточно снабдить каждый фрагмент опциями первого типа, заключив их в отдельный заголовок расширения «опции адресата». Остальные заголовки и полезная нагрузка нужны только конечному адресату, так что их вполне можно разбить между фрагментами.

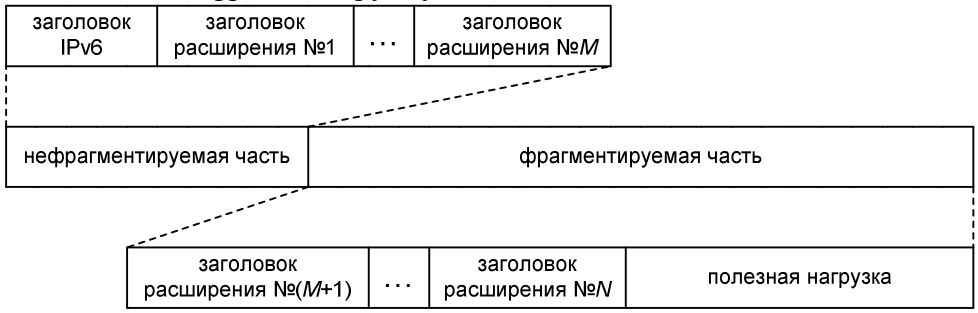
Обратите внимание, что конечному адресату интересны все заголовки расширения.

Опции для всех адресатов и опции для конечного адресата будут заключены в заголовки расширения одного и того же типа, «опции адресата» (код 60). Отличие будет только в том, как они размещены относительно других заголовков. А именно первые должны предшествовать заголовку фрагмента, а последние — идти за ним.

Мы питаем слабость к обобщениям, так что давайте сначала опишем эту картину в целом, а затем уже перейдем к ее деталям. Итак, исходный пакет IPv6 состоит из двух частей, **нефрагментируемой** (*unfragmentable*) и **фрагментируемой** (*fragmentable*), как показано на Фиг. 35. Мы уже выяснили, что нефрагментируемая часть состоит из таких заголовков:

- основной заголовок IPv6;
- пошаговые опции (если есть);
- опции адресата, важные для промежуточных адресатов (если есть);
- маршрутный заголовок (если есть).

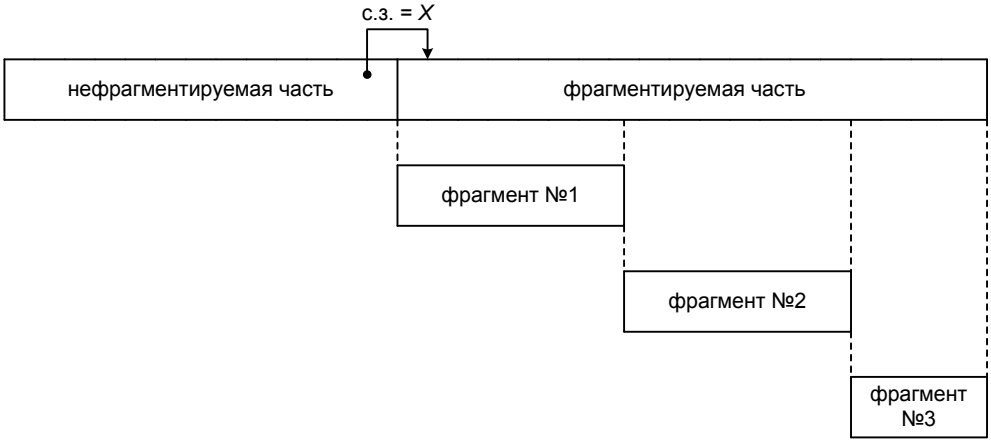
Все прочие заголовки расширения, если они есть в данном пакете, а также полезная нагрузка составляют его фрагментируемую часть.



Фиг. 35. Структура пакета IPv6 с точки зрения его фрагментации

Пусть читатель подумает, где и в каком виде существует исходный пакет, если по сети передаются только его фрагменты. (Как образ в памяти источника и конечного адресата.)

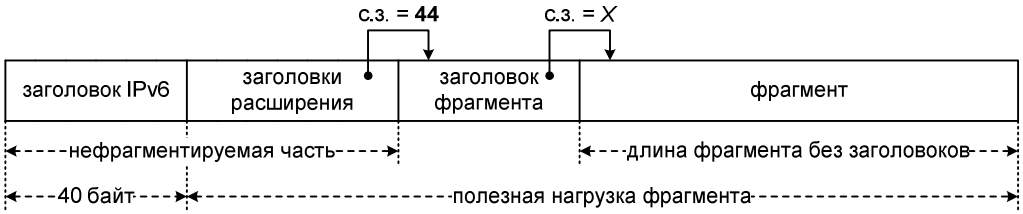
Первым делом в процессе фрагментации вторую часть делят на фрагменты ненулевой длины, так чтобы окончательная длина каждого фрагмента после инкапсуляции не превысила текущей оценки PMTU, — эта операция схематически представлена на Фиг. 36. Но как связать длины фрагмента до и после инкапсуляции? Ответ на этот вопрос мы получим, рассмотрев следующий шаг процедуры фрагментации.



Фиг. 36. Деление пакета IPv6 на фрагменты (с.з. = следующий заголовок)

Следующим шагом каждый фрагмент превращают в полноценный пакет IPv6. Для этого к нему спереди добавляют заголовок фрагмента, а перед ним — полную копию нефрагментируемой части, как изображено на Фиг. 37. Чтобы такой пакет стал годным, остается скорректировать пару полей в нем, а именно поле «длина полезной нагрузки» в

заголовке IPv6 и поле «следующий заголовок» в заголовке перед заголовком фрагмента. Первая корректировка нужна, так как поле «длина полезной нагрузки» всегда относится к текущему пакету, даже если это фрагмент. Второй корректировкой мы учитываем тот факт, что цепочка заголовков изменилась и теперь после нефрагментируемой части идет заголовок фрагмента.



Фиг. 37. Фрагмент после инкапсуляции (с.з. = следующий заголовок)

В других стеках встречаются протоколы сетевого уровня, заголовки которых отдельно хранят длины данного фрагмента и исходного пакета. Пример такого протокола — ISO CLNP.

Теперь мы видим, что первоначальная длина фрагмента до инкапсуляции должна быть не больше, чем текущая оценка PMTU за вычетом длин нефрагментируемой части и заголовка фрагмента. Насколько сложно будет провести это вычисление заранее, до начала фрагментации? Длина нефрагментируемой части зависит только от структуры исходного пакета, она не изменяется от фрагмента к фрагменту. Что касается заголовка фрагмента, то в наших силах сделать его длину фиксированной и тем упростить алгоритм фрагментации.

Законопослушный узел *на выходе* не должен менять состав и содержание нефрагментируемой части от фрагмента к фрагменту. Ведь зачем это ему? Тем не менее, *на входе* может оказаться, что у разных фрагментов одного исходного пакета разные нефрагментируемые части. Как мы скажем, когда дойдем до процедуры сборки, в этом случае безусловное преимущество у фрагмента со смещением ноль.

Пришло время подумать над форматом заголовка фрагмента. Он зависит от той системы параметров, которой мы станем описывать каждый фрагмент. Впрочем, готовую систему мы вполне можем взять из IPv4, потому что она удовлетворяет ключевому требованию: устойчивости к нарушению порядка фрагментов. Итак, вот параметры фрагмента IPv6:

- идентификатор исходного пакета;
- смещение;
- флаг «еще фрагменты».

К этим полям мы также должны добавить поле «следующий заголовок», чтобы обеспечить сцепление заголовков и полезной нагрузки в пакете IPv6. Таким образом, общее число полей постоянно, а значит, и заголовок фрагмента сможет обладать постоянной длиной. Теперь давайте рассмотрим каждый параметр по очереди и уточним его свойства.

Идентификатор пакета занимал в IPv4 16 бит, и этого оказалось мало: при большой скорости трафика вероятны конфликты, которые приводят к перекрестной сборке пакетов [RFC 4963]. Хуже того, при дальнейшем росте скорости трафика неверно собранные пакеты возникают достаточно часто, чтобы начали сбоить контрольные суммы TCP и UDP: происходят коллизии, когда сбойный пакет обладает верной контрольной суммой. Поэтому давайте расширим это поле до 32 бит. Но будет ли их достаточно?

Чтобы дать на это ответ, мы должны сначала определить, какова область уникальности идентификатора. В IPv4 идентификатор ведь не был самостоятельным — он объединялся с адресами источника и назначения, а также номером протокола. Только весь этот кортеж должен был оставаться уникальным. Перенести это правило в IPv6 без изменений нам мешает то, что номер протокола больше не характеризует пакет как целое:

теперь пакет состоит из модулей, каждому из которых отвечает свой номер протокола. Поэтому пусть в IPv6 идентификатор будет уникальным только для заданных адресов источника и назначения.

> Если пакет содержит маршрутный заголовок, то это будут адрес источника и адрес конечного назначения [§4.5 RFC 2460], поскольку собирать пакет IPv6 будет его конечный адресат.

> Если у фрагмента адрес источника или адрес назначения имеет ограниченную область действия, то индекс его зоны тоже придется учесть при идентификации такого фрагмента [54]. Если этого не сделать, то возможна перекрестная сборка пакетов из разных зон одной области. Скажем, фрагменты с адресом источника *FE80::1*, адресом назначения *FE80::2* и идентификатором *42* — суть части разных исходных пакетов, если они получены из разных каналов. Это очевидное следствие из зонной адресной архитектуры IPv6 (§2.4).

Во времени же идентификатор должен оставаться уникальным, пока «живы» фрагменты данного пакета. Оценить время жизни фрагментов сверху можно суммой времени доставки фрагмента по сети и величиной тайм-аута сборки. Если вторая величина — настраиваемый параметр сетевого стека, то первая может принимать сколь угодно большие значения, по крайней мере, в теории.

> Здесь бы нам пригодилась интерпретация поля TTL как времени жизни пакета, но мы от нее отказались по практическим соображениям [§8.2 RFC 2460].

Поэтому нам остается сделать эмпирическое предположение: чем выше скорость трафика, тем меньше вероятная задержка пакета в сети. Здесь мы имеем в виду не время распространения сигнала, а непроизводительные расходы времени, такие как ожидание в буферах транзитных узлов. При большой скорости трафика пакет скорее будет потерян, нежели задержится где-то на многие секунды.

Значит, в случае большой скорости трафика — а именно он нам интересен — главным фактором в управлении надежной сборкой будет ее тайм-аут. Он должен быть существенно меньше, чем период, за который идентификатор заведомо повторится. Этот период сокращается при росте скорости трафика, так что его можно оценить снизу, зная верхнюю оценку скорости, с которой один источник готов передавать пакеты в адрес данного узла.

Попробуем оценить худший случай, когда скорость расходования значений идентификатора максимальна при постоянной скорости трафика, выраженной, например, в бит/с. Одно значение идентификатора расходуется на один исходный пакет, так что мы ищем условие максимальной частоты исходных пакетов при постоянной скорости трафика. Для этого необходимо, чтобы сумма длин фрагментов была минимальной.

Пакет IPv6, который фрагментирован согласно протоколу, состоит не менее чем из двух фрагментов, а длина первого фрагмента не менее 1280 байт. Длина же второго фрагмента не может опуститься ниже суммы длин заголовка IPv6 и заголовка фрагмента плюс один байт.[55] Если мы чуть забежим вперед и скажем, что заголовок фрагмента занимает 8 байт, то наименьшая сумма длин фрагментов составит $1280 + 40 + 8 + 1 = 1329$ байт.

> Сейчас мы не рассматриваем «патологическое» поведение источника, когда он создает пакеты из одного фрагмента или чрезмерно короткие фрагменты. Тем не менее, алгоритм сборки фрагментов должен быть устойчив к этим аномалиям, так как они явным образом ничему не противоречат. Так завещает нам принцип Постела.

---

[54] См. сообщение *JINMEI Tatuya* «*Re: IPv6 Scoped Address Architecture (RFC 2710)*» в списке рассылки IETF IPng: http://www.mail-archive.com/ipng@sunroof.eng.sun.com/msg00465.html

[55] Годный фрагмент не может быть пустым.

Теперь мы без труда вычислим, что, например, при передаче со скоростью 1 Тбит/с значение 32-битного идентификатора неизбежно повторится только через такой интервал времени:

$$\frac{8\,{}^{бит}\!/_{байт}\times 1329\,{}^{байт}\!/_{пакет}\times 2^{32}\ пакет}{10^{12}\,{}^{бит}\!/_{с}}\approx 46\,с\,.$$

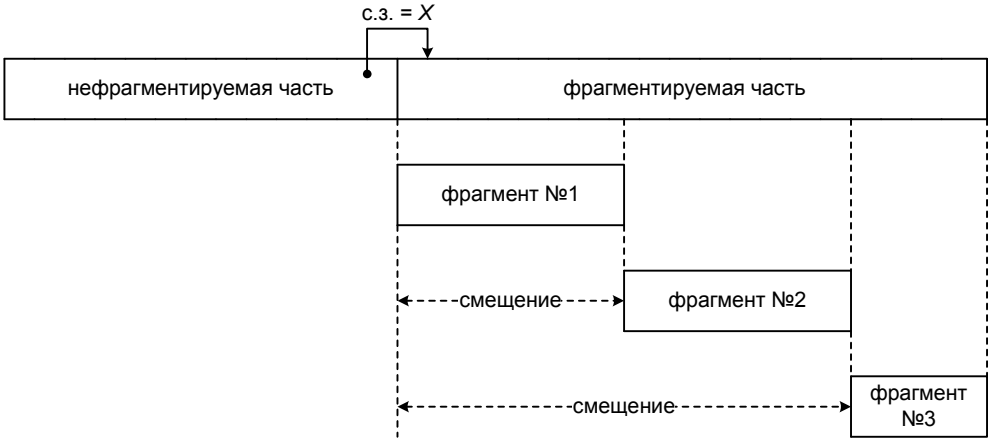
Также можно вычислить, какая скорость фрагментированного трафика станет опасной, если тайм-аут сборки составляет 60 секунд, как предписано в §4.5 RFC 2460:

$$\frac{8\,{}^{бит}\!/_{байт}\times 1329\,{}^{байт}\!/_{пакет}\times 2^{32}\ пакет}{60\,с}\approx\frac{46}{60}\times 10^{12}\,{}^{бит}\!/_{с}\approx 7{,}6\times 10^{11}\,{}^{бит}\!/_{с}\,.$$

Запас еще в несколько порядков, конечно, не помешал бы; но и такое соотношение скорости трафика и периода повтора идентификатора[56] можно назвать удовлетворительным, потому что этот трафик — не магистральный, создаваемый многими узлами, а только от одного узла к другому.

> Реализация IPv6 *KAME* пошла на сознательное упрощение. Вместо того чтобы генерировать независимые цепочки идентификаторов для каждой пары адресов источника и назначения, она использует одну цепочку на весь стек. Раньше это был простой счетчик, тогда как современное развитие *KAME* в *FreeBSD* применяет генератор псевдослучайных чисел с заведомо большим периодом. Конечно, такой трюк не может не привести к сокращению периода повтора идентификатора.

Смещение фрагмента, с нашей тягой к экономии единиц, пусть отсчитывается не от начала пакета, а от начала фрагментируемой части, как показано на Фиг. 38.



Фиг. 38. Как вычисляется смещение фрагментов (с.з. = следующий заголовок)

> Побочным результатом этого выбора будет то, что теперь смещение фрагмента не может указать внутрь основного заголовка IPv6. Как мы помним, в IPv4 смещение фрагмента отсчитывалось от самого начала пакета, и это позволяло злоумышленнику составить такую последовательность фрагментов, в которой последующие фрагменты накладывались на важнейшие поля заголовка, в частности, протокол и адреса источника и назначения. Чуть позже мы сделаем фрагментацию IPv6 еще безопаснее, наложив полный запрет на перекрывающиеся фрагменты.

Единицу смещения и длину поля «смещение» мы оставим такими же, как в IPv4: 8 байт и 13 бит, соответственно. Это позволит смещению принимать значения вплоть до $\left(2^{13}-1\right)\times 8=65528$ байт включительно. Такого диапазона достаточно, потому что фрагментируемая часть исходного пакета не длиннее максимальной «полезной нагрузки» IPv6, составляющей 65535 байт; а 7 байт разницы заведомо поместятся в один фрагмент,

[56] Если быть точным, постоянно *произведение* скорости трафика на период повтора идентификатора.

если нефрагментируемая часть состоит только из основного заголовка IPv6 — условие максимально длинной фрагментируемой части.

> Как мы помним, значение «длина полезной нагрузки» в основном заголовке IPv6 включает заголовки расширения.

Полученный диапазон смещений даже чуть шире, чем нужно: он позволяет создать последовательность фрагментов, которая соберется в пакет-великан длиной более 65575 байт. Для этого достаточно, чтобы у любого фрагмента сумма его смещения в байтах и длины без заголовков превысила 65575 байт за вычетом длины нефрагментируемой части (см. Фиг. 36 и Фиг. 37). Если в ходе сборки получится пакет-великан, то следует уничтожить его и, возможно, выслать источнику извещение об ошибке.[57]

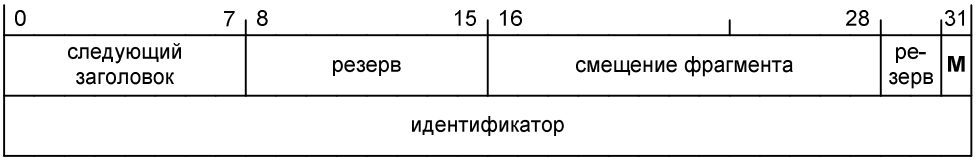
> Пусть читатель сам установит, какова максимальная длина такого пакета-великана в предположении, что PMTU не меньше 65575 байт. Указания к решению: Прочтите до конца раздел. Затем докажите формулу:
>
> $$L = L_1 - F_1 - 8 + 8O_N + F_N \ ,$$
>
> где L — длина полезной нагрузки исходного пакета, L_1 — длина полезной нагрузки первого фрагмента, F_1 — длина первого фрагмента без заголовков, O_N — смещение последнего фрагмента в 8-байтных словах, F_N — длина последнего фрагмента без заголовков. Максимизируем все параметры, кроме F_1, который минимизируем, поскольку он со знаком «минус»: max L_1 = 65535, min F_1 = 8 (из-за выбора единицы смещения), max O_N = 8191 (13 бит), max F_N = 65535 − 8 = 65527 (учитываем длину заголовка фрагмента 8 байт). В результате получаем, что максимальная полезная нагрузка пакета-великана составляет 196574 байт, а весь пакет еще на 40 байт длиннее! Такая последовательность фрагментов основана на трюке с разным составом нефрагментируемой части: в первом фрагменте она будет максимально длинной, например, за счет опций PadN (§3.3.2), а в последнем там будет только заголовок IPv6. Очевидно, что послать такую серию фрагментов может только экспериментатор или злоумышленник, потому что она — испытание узла-адресата на устойчивость.

Флаг «еще фрагменты» (**M**) обладает той же семантикой, что и флаг **MF** в IPv4: если он установлен, то это не последний фрагмент, а если сброшен, то последний.

Длина фрагмента до инкапсуляции (см. Фиг. 36) должна быть кратна 8 байтам из-за выбора единицы смещения, если только это не самый последний фрагмент. Ведь иначе смещение следующего фрагмента не будет делиться на 8. Поэтому, если вдруг адресат получит фрагмент с установленным флагом «еще фрагменты», и его длина без заголовков (см. Фиг. 37) окажется не кратной 8 байтам, то значит, где-то произошел сбой. Такой фрагмент надо отбросить, а в адрес источника выслать извещение об ошибке.

> Конечно же, характеристики «последний» и «не последний» относятся здесь к позиции фрагмента в исходном пакете, а не к порядку поступления фрагментов из сети. Работа механизма фрагментации и сборки IP не зависит от порядка фрагментов, потому что сеть вправе непредсказуемо нарушать его.

Все эти поля мы уместим в заголовок длиной 8 байт, ради выравнивания, как показано на Фиг. 39. Это и будет заголовок фрагмента [§4.5 RFC 2460].



Фиг. 39. Заголовок фрагмента

[57] Используя аналог ICMP, над которым мы еще поработаем.

Поле «смещение» расположено так, чтобы удобно было переводить его значение в байты: достаточно прочесть все выровненное 16-битное слово, а затем обнулить три младших бита операцией «логическое И» с маской `0xFFF8`. Это эквивалентно сдвигу значения «смещение» влево на 3 двоичных разряда, то есть умножению его на 8.

Сборка выполняется в обратном порядке:

1) получив на входе фрагмент, сборщик составляет кортеж: <адрес источника, адрес назначения, идентификатор>, — и использует его как ключ при поиске буфера сборки;
    а) если буфер сборки по этому ключу не найден, надо создать новый буфер, так как перед нами новый пакет;
2) если смещение фрагмента равно нулю:
    а) его начало, вплоть до заголовка фрагментации, но не включая его, становится нефрагментируемой частью вновь собираемого пакета и хранится вне буфера сборки;

> В принципе, реализация может хранить нефрагментируемую часть и в буфере сборки, с тем чтобы в итоге сборки получить полный пакет. Однако тогда на шаге 3 ей придется внести поправку в позицию фрагмента внутри буфера. Ведь в заголовке фрагмента смещение указано относительно начала фрагментируемой части, а не всего пакета. Кроме того, понадобится буфер сборки с подвижным началом, так как длина нефрагментируемой части неизвестна до приема фрагмента со смещением ноль, а он может прийти не первым, если сеть переупорядочит фрагменты.

    б) сборщик запоминает значение «следующий заголовок» из заголовка фрагмента;

> §4.5 RFC 2460 допускает случай, когда фрагменты одного пакета содержат разные заголовки расширения и значения «следующий заголовок», хотя на практике такие вариации весьма и весьма подозрительны. Но какие заголовки и значения полей попадут в собранный пакет? Здесь преимущество будет у самого первого фрагмента — то есть с нулевым смещением, а не пришедшего раньше других.

3) данные после заголовка фрагментации помещаются в буфер сборки по смещению, указанному в заголовке фрагментации;
4) если фрагмент — последний (флаг **M** сброшен): длина буфера сборки «замораживается», чтобы прочие фрагменты не могли оказаться за последним;
5) если все «пробелы» в буфере сборки заполнены: фрагментируемая часть пакета собрана; осталось поместить перед ней нефрагментируемую часть (см. шаг 2(а)) и провести обратную корректировку полей:
    а) «длина полезной нагрузки» — теперь относится к вновь собранному пакету;
    б) «следующий заголовок» в последнем заголовке нефрагментируемой части — используя значение, которое запомнили на шаге 2(б).

По ходу сборщик должен обнаруживать возможные ошибки, которые мы уже обсудили выше:

- тайм-аут сборки, чтобы недособранные пакеты не занимали память навечно;
- фрагмент с установленным флагом **M**, длина которого не кратна 8 байтам;
- вновь собранный пакет-великан длиной более 65575 байт.

Естественной реакцией на эти ошибки будет отбросить пакет и, возможно, выслать источнику извещение об ошибке.

> Мы говорим: «Возможно», — потому что извещения не стоит слать в ответ на групповые пакеты и другие извещения. Об этом мы еще поговорим в §4.3.

Кроме того, возможны отклонения, связанные с позицией и ролью фрагмента, как то:

- больше одного фрагмента со сброшенным флагом **M**;
- перекрывающиеся фрагменты.

В обоих случаях, чтобы вызвать подозрение, это должны быть разные фрагменты, а не дубликаты одного и того же фрагмента, возникшие в сети.

В лучшем случае их может вызвать совпадение кортежа <адрес источника, адрес назначения, идентификатор> у разных фрагментированных пакетов, а это чревато перекрестной сборкой и повреждением данных.

В худшем же случае это может быть злонамеренная попытка обойти системы сетевой безопасности. Вот пример такой атаки. Допустим, веб-сервер защищен простым межсетевым экраном, так что входящие соединения TCP разрешены только с портом 80. Злоумышленник посылает первый фрагмент с заголовком TCP, где порт назначения равен 80, так что межсетевой экран считает этот пакет безопасным и пропускает его дальнейшие фрагменты. Тогда следующим шагом злоумышленник шлет перекрывающийся фрагмент, который заменяет порт назначения 80 на 22 (или любой другой) в буфере сборки веб-сервера.

Поэтому самым безопасным решением будет полностью запретить перекрытие фрагментов IPv6 [RFC 5722]:

- законопослушный источник не имеет права создавать перекрывающиеся фрагменты;
- конечный адресат обязан «молча»[58] отбросить весь собираемый пакет, если в нем обнаружено перекрытие фрагментов; для полноты защиты следует отбросить и последующие сегменты этого пакета, пришедшие в пределах тайм-аута сборки.

Рассмотрите и другие возможные атаки на механизм сборки IPv6. Например: Злодей шлет фрагменты в обратном порядке: №2, а затем №1. У фрагмента №2 сумма смещения и длины не превышает дозволенного, и адресат создает буфер сборки. Затем приходит фрагмент №1, у которого нефрагментируемая часть максимально возможной длины или около того. Вместе эти фрагменты образуют пакет-великан, причем он «растет» не с конца, а с головы буфера сборки, за счет нефрагментируемой части. Это вполне может вызвать крах неаккуратной реализации. Конечно, для такой атаки нужен достаточно большой «потолок» PMTU.

3.3.5. Заголовки безопасности IPsec

В контексте заголовков расширения IPv6 нам будем уместно упомянуть **IPsec** (*IP security*, безопасность IP). Ведь IPsec — это рекомендованный компонент всякого узла IPv6 [§11.1 RFC 6434].

В предыдущей редакции технических условий узла IPv6 поддержка IPsec была даже обязательной [§8.1 RFC 4294]. Хотя в новой редакции [§11.1 RFC 6434] она стала рекомендованной, а не обязательной, это вовсе не перечеркнуло давней связи между IPv6 и IPsec. IPsec был «разжалован» ради двух вполне оправданных целей [§11 RFC 6434]. Во-первых, IPsec — это не панацея и не монопольный поставщик безопасности в среде TCP/IP. В некоторых случаях стоит предпочесть безопасность на другом уровне, скажем, транспортном (TLS) или прикладном (SSH). А во-вторых, обязательность IPsec — это серьезное препятствие на пути встроенных реализаций IPv6 для платформ с ограниченными ресурсами, например, кофеварок и сенсоров. Уместить IPsec в такую минимальную реализацию практически невозможно, а без него реализация IPv6 не соответствует

[58] То есть без каких-либо извещений в адрес источника.

действующему стандарту. Новая редакция технических условий узла IPv6 разрешила это противоречие.

В целом IPsec [RFC 4301] — это архитектура, то есть целая система моделей, механизмов, алгоритмов и протоколов, направленная на сквозную защиту пакетов IP. Слово «сквозная» здесь значит, что защиту пакета выполняют только его источник и конечный адресат, как то: источник подписывает пакет, а конечный адресат проверяет подпись; источник шифрует пакет, а конечный адресат расшифровывает его. Между тем, транзитные узлы, будь то маршрутизаторы или промежуточные адресаты, обрабатывают защищенные пакеты точно так же, как обычные.

Хотя исторически IPsec возник именно для защиты IPv6 и только затем был адаптирован к среде IPv4, подробно рассматривать его мы не будем хотя бы потому, что он заслуживает отдельного курса лекций. Сейчас мы ограничимся только тем, что посмотрим, какие черты объединяют заголовки безопасности IPsec с другими заголовками IPv6.

Среди протоколов IPsec с IPv6 непосредственно соприкасаются протоколы **AH** (*authenticated header*, аутентифицированный заголовок) [RFC 4302] и **ESP** (*encapsulating security payload*, инкапсулирующая защитная полезная нагрузка) [RFC 4303]. Первый из них, AH, обеспечивает только аутентификацию и защиту целостности пакетов, тогда как ESP позволяет, кроме того, зашифровать большую часть пакета. Так как ESP умеет все, на что способен AH, и даже сверх того, применять их вместе смысла нет. Давайте посмотрим, сможем ли мы составить заголовки расширения для этих протоколов, исходя из общих соображений.

По одной из версий, AH возник отдельно от ESP, только чтобы обойти ограничения на экспорт средств шифрования из США [59]. Лазейка в законе состояла в том, что для средств аутентификации делалось исключение, хотя они зачастую опирались на те же самые математические конструкции, что и средства шифрования.

Какие поля потребуются заголовку AH? В первую очередь, по смыслу AH, это цифровая подпись пакета (или же, в терминологии IPsec, значение защитного кода — *integrity check value*, **ICV**). Алгоритмы безопасности и способы атаки на них находятся в постоянном развитии, подобно броне и снарядам; поэтому нам не следует заранее фиксировать длину цифровой подписи и алгоритм ее вычисления. Как следствие, заголовок AH будет переменной длины, и ему понадобится поле «длина». В отличие от других заголовков расширения IPv6, единицей длины AH служит 32-битное слово.

В подражание другим заголовкам, значение поля длины AH — это полная длина заголовка AH минус два слова [§2.2 RFC 4302]. Конечно, логичнее было бы вычесть длину постоянной части AH, то есть три слова. С другой стороны, вычитая только два слова, мы гарантируем, что значение поля длины AH никогда не будет нулевым. Предубеждение против нулевых значений основано на том, что во многих протоколах нулевое значение означает «нет сведений». См., например, поле контрольной суммы UDP. Кроме того, в IPv6 длина заголовка AH должна быть кратна 64 битам, чтобы сохранить выравнивание последующих данных [§2.2 RFC 4302].

Называется поле длины AH «длина полезной нагрузки» [§2.2 RFC 4302] — наверное, чтобы сбить врагов с толку.

Как нам быть с алгоритмом подписи и его возможными параметрами, такими как секретный ключ? Модель IPsec полагает, что эти сведения заранее находятся в памяти источника и адресата как специальная запись, **связь безопасности** (*security association*,

[59] M. Kaeo. IPsec Analysis in an IPv6 Context–v01. NAv6TF, 2007. http://www.nav6tf.org/documents/nav6tf.draft_ipsec_analysis_v1.0.pdf (зеркало: https://docs.google.com/file/d/0B85W8noVmG5RMzN3VUEzZlBwNkk/edit?usp=sharing)

SA) [§4 RFC 4301]. Если провести параллель с кинофильмом «про шпионов», обмен шифроблокнотами происходит за кадром, потому что так безопаснее. В простейшем случае, связь безопасности вводят вручную на каждом из узлов.

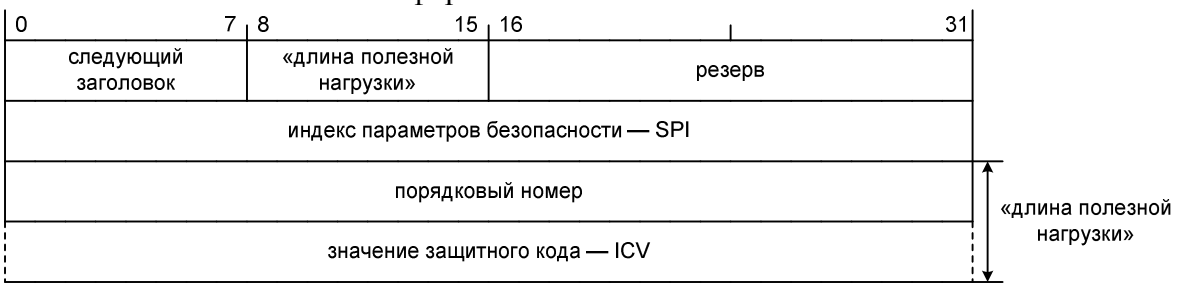
> Конечно, на практике связями безопасности удобнее управлять автоматически, с помощью дополнительного протокола, такого как IKE.

Поэтому, когда приходит время передавать данные, источнику достаточно сослаться на нужную связь (у шпионов это была бы строка в шифроблокноте), чтобы адресат смог правильно трактовать заголовок безопасности. Такой ссылкой служит 32-битный **индекс параметров безопасности** (*security parameter index*, **SPI**). Он явно заслуживает отдельного поля в заголовке AH [§2.4 RFC 4302].

Далее, чтобы злоумышленник не смог провести атаку воспроизведением, послав копию аутентичного пакета, полезно будет включить в заголовок AH порядковый номер, монотонно растущий на протяжении жизни данной связи безопасности [§2.5 RFC 4302].

> На первый взгляд, атака воспроизведением основана на свойстве неидемпотентности некоторых сообщений. Например, дважды предъявив копии чека на 1 рубль, мы получим 2 рубля. Поэтому чеки обычно номерные. С другой стороны, повторный сигнал «поднять железнодорожный семафор» ничего не изменит, потому что семафор уже поднят. Тем не менее, злоумышленник может сохранить копию этого сигнала и послать ее позже, когда семафор закроется, вызвав тем самым крушение поезда. Поэтому неудивительно, что система безопасного обмена данными должна противостоять таким атакам.
>
> Так как сеть способна изменять порядок доставки пакетов, нельзя надеяться, что порядковый номер будет монотонно расти в *принимаемых* пакетах. Поэтому проверка порядкового номера AH адресатом происходит при помощи скользящего окна [§3.4.3 RFC 4302]: пакеты с номером меньше нижней границы окна игнорируются, в пределах окна запоминается каждый принятый номер, а получение номера больше верхней границы окна сдвигает окно вперед.

Наконец, AH — это заголовок расширения, за которым могут находиться другие заголовки и полезная нагрузка. Поэтому в заголовке AH необходимо поле «следующий заголовок», чтобы адресат смог правильно интерпретировать последующие данные. Как можно убедиться в [§2 RFC 4302], наши общие соображения позволили обосновать все поля заголовка AH. Его точный формат показан на Фиг. 40.



Фиг. 40. Заголовок AH

Для полноты картины осталось заметить, что цифровая подпись ICV защищает не только «хвост» пакета, начиная с заголовка AH, но и предшествующие заголовки — по мере возможности, конечно. Трудность здесь в том, что некоторые поля в этих заголовках могут меняться по пути следования пакета. Если бы все их удалось идентифицировать, то при вычислении ICV через них можно было бы «перескочить» или подставить вместо данных то же число нулевых битов. Мы выберем замещение нулями, потому что оно защищает хотя бы длину переменного поля, при условии что значение ICV зависит от *числа* замещающих нулевых битов. Вернемся к идентификации переменных полей. Мы уже решили эту задачу для опций IPv6, закодировав нужную информацию в одном бите кода опции. Заголовки IPv6, в отличие от опций, — стандартные, и их свойства можно

просто перечислись в соответствующем документе [Приложение А2 RFC 4302]. Здесь у нас возникает искушение провести троичную, а не двоичную классификацию полей:

1) неизменные;
2) предсказуемые;
3) непредсказуемые.

Новый класс, предсказуемые поля и заголовки, — это те, значение которых меняется предсказуемым образом, так что источник, вычисляя ICV, может использовать их финальное значение, которое увидит конечный адресат. На практике, единственным стандартным представителем этого класса оказывается маршрутный заголовок RH0 (и связанный с ним адрес назначения).

> Любитель биологии заметил бы, что даже этот представитель — ныне ископаемый.

Окончательно классифицируем поля основного заголовка IPv6 в Табл. 9.

Табл. 9. Предсказуемость полей основного заголовка IPv6

Неизменные	Предсказуемые	Непредсказуемые
• «версия» • «длина полезной нагрузки» • «следующий заголовок» • «адрес источника» • «адрес назначения» — в отсутствие маршрутного заголовка	• «адрес назначения» — при наличии маршрутного заголовка	• «класс трафика» • «метка потока» • «предельное число шагов»

Что касается заголовков расширения, то здесь работы нам практически не осталось. В заголовках опций эта задача решена на уровне отельных опций; предсказуемый заголовок RH0 упразднен; а заголовок фрагмента вообще не подлежит защите, потому что фрагментация происходит после защиты, а сборка, соответственно, перед проверкой AH [Приложение А2 RFC 4302], так что модулю IPsec никогда не встретится заголовок фрагмента, если только не произошел сбой в стеке.

> Пакет-фрагмент может попасть на вход защищенного туннеля, но тогда он подвергнется туннельной инкапсуляции, и полученный туннельный пакет уже не будет фрагментом. Это применимо и к туннельному режиму IPsec, который лишь объединяет логические операции туннелирования и защиты в одну удобную услугу.

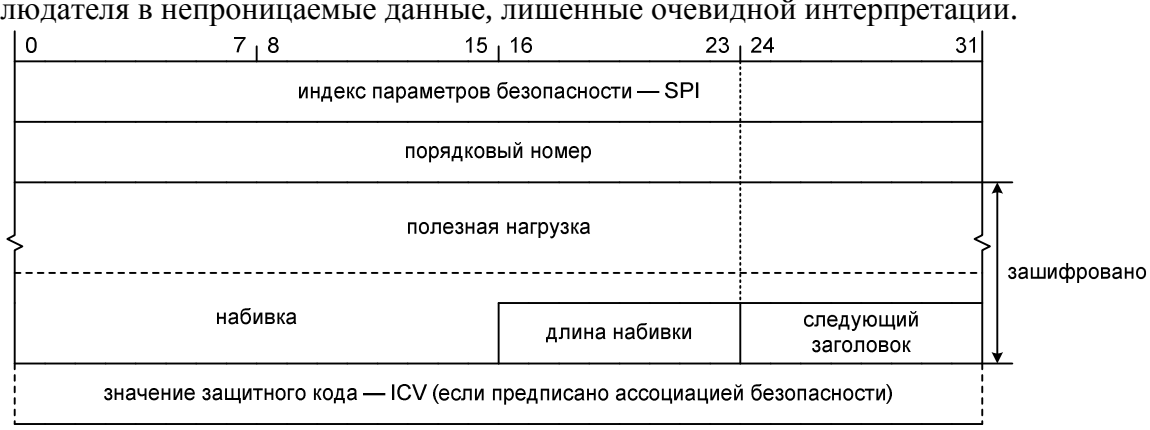
Перейдем теперь к ESP. Новой задачей, по сравнению с AH, будет скрыть содержимое пакета от посторонних глаз, применив шифрование. Чтобы сторонний наблюдатель мог извлечь как можно меньше сведений о содержимом пакета, следует зашифровать не только байты, следующие за заголовком ESP, но и сведения об их смысле, то есть поле «следующий заголовок». Тогда, в теории, наблюдателю придется только гадать, что же содержится в данном пакете. Однако на практике выбор значений поля «следующий заголовок» не так уж велик. Более того, сегодня в прикладных пакетах чаще всего это будет 6 (TCP) или, заметно реже, 17 (UDP), а в туннельных 4 (IPv4) или 41 (IPv6). Если начать шифрование с этого поля, то ограниченный набор его вероятных значений откроет лазейку для атак на шифротекст, потому что многие алгоритмы шифрования работают над входными данными последовательно, байт за байтом или блок за блоком.

Это касается даже шифрования в режиме сцепления блоков (CBC), потому что первый блок ни с чем не сцеплен, и предсказуемое начало входного текста все так же понижает надежность защиты.

Чтобы этого не произошло, применим простой трюк: пусть поле «следующий заголовок» находится в ESP *после* защищаемых данных и потому шифруется последним [§2 RFC 4303]. Туда же, в конец шифротекста, мы поместим поля «набивка» и «длина набивки», необходимые для блочных шифров. И тогда перед шифротекстом достаточно будет поместить только SPI и порядковый номер — конечно, открытым текстом.

Если данная связь безопасности предписывает не только шифровать пакеты, но и проверять их подлинность, то цифровую подпись ICV можно добавить после шифротекста. Тогда сторонний наблюдатель, не имея доступа к параметрам безопасности, даже не сможет определить, подписан ли данный пакет.

Именно этой структуре пакета, представленной на Фиг. 41, ESP обязан своим названием. В отличие от AH, он не встраивается в цепочку заголовков расширения, а полностью инкапсулирует защищаемую часть пакета, превращая ее для стороннего наблюдателя в непроницаемые данные, лишенные очевидной интерпретации.



Фиг. 41. Инкапсуляция ESP

Опытный взломщик, не имея доступа к самим данным, может делать выводы из размера пакетов и их распределения по оси времени. Полное сокрытие полезной нагрузки в ESP позволяет перекрыть и этот канал утечки информации. Вариации размера пакетов можно маскировать достаточно объемистой набивкой [§2.7 RFC 4303]. А чтобы не было очевидных закономерностей в ритме обмена пакетами, стороны могут вставлять в поток пакеты-пустышки, когда нет настоящих данных для передачи. На эту роль прекрасно подойдут пакеты, в которых цепочка заголовков завершается кодом 59 — «следующего заголовка нет» (§3.3.1). Вместе эти приемы обеспечивают конфиденциальность на уровне потока пакетов (*traffic flow confidentiality*, **TFC**).

У такой изоляции ESP есть и обратная сторона. В отличие AH, в ESP цифровая подпись ICV не защищает внешний заголовок IP, предшествующий данным ESP [§2 и §3.3.2 RFC 4303]. Поэтому информации из внешнего заголовка в пакете ESP слепо доверять нельзя. Тем не менее, возможности злоумышленника по его подделке не так уж велики. Если поле SPI указало на годную связь безопасности SA в памяти адресата, а проверка целостности и расшифровка прошли успешно, то адресат пакета может сделать следующие выводы:

- полезная нагрузка пакета подлинная;
- пакет пришел из доверенного источника;
- возможно, связь SA указывает на источник пакета.

Архитектура IPsec допускает, но не требует, чтобы связь SA была привязана к конкретным адресам источника и назначения [§4.4.1 RFC 4301].

Тем не менее, злоумышленник может проводить ограниченные атаки на протоколы, чувствительные к точным значениям адресов источника и назначения в пакете. К примеру, модифицируя эти адреса, можно перемещать сегменты TCP из одного соединения в другое, при условии, что эти соединения различаются только адресами, но не портами.

> Как мы помним, идентификатором сеанса TCP выступает кортеж <адрес №1, порт №1, адрес №2, порт №2>. Поэтому вполне возможна одновременная работа двух сеансов TCP, которые отличаются, скажем, только адресом №1.

Чтобы блокировать и этот вектор атаки, нам достаточно превратить заголовок IP в полезную нагрузку перед тем, как над ним поработает ESP. Мы уже умеем это делать с помощью туннельной инкапсуляции, так что сейчас нам достаточно соединить вместе два решения: туннель и IPsec. Если мы ограничимся простейшим туннелем «IP-в-IP», то получим конструкцию, известную как **туннельный режим** (*tunnel mode*). Очевидно, что он применим не только к ESP, но и к AH, хотя для ESP он важнее. В противоположность ему, непосредственная защита прикладных пакетов называется **транспортный режим** (*transport mode*), возможно, потому что в этом режиме под защитой чаще всего оказываются дейтаграммы транспортных протоколов.

> На практике под защитой IPsec вполне могут быть и другие туннельные протоколы, например, GRE или L2TP. Ведь для IPsec безразлично, что защищать, пока это IP.

> Пусть читатель самостоятельно изобразит структуру пакетов AH и ESP в туннельном режиме. Какие значения полей «следующий заголовок» в них возникнут? Рассмотрите IPv4 и IPv6.

Теперь, обезопасив себя от подделки адресов, спокойно посмотрим на ту же проблему с другой стороны. Кто еще, кроме злодеев, может вносить изменения во внешний заголовок IP? Конечно же, трансляторы NAT. В некоторых случаях NAT — это полезный инструмент, однако его работа основана на подмене одних адресов IP другими. И если ESP спокойно отнесется к трансляции адресов во внешнем заголовке, то AH неизбежно забракует пакет, прошедший NAT.

> Искушенный читатель может вспомнить еще одно средство подружить IPsec и NAT: NAT-T. Однако в действительности протокол NAT-T решает довольно узкую задачу: он позволяет IPsec проходить сквозь трансляторы NAT в режиме маскарада, когда трансляция происходит по принципу «многие в один» (сюръекция). Очевидно, что при этом происходит потеря адресной информации на уровне IP, и недостающие сведения транслятор скрывает в номерах портов TCP или UDP (или других аналогичных полях, таких как идентификатор ICMP *ping*). Поэтому пакеты IPsec приходится дополнительно инкапсулировать в UDP. Принципиальной несовместимости между AH и NAT это не решает, так что инкапсуляция NAT-T определена только для ESP [RFC 3948].

3.3.6. Порядок заголовков

Теперь мы составили полную картину заголовков расширения IPv6 и назначили каждому из них определенную роль. Весьма вероятно, что эта роль определяет не только трактовку данного заголовка, но и его предпочтительную позицию в пакете. В то же время, единообразный формат заголовков расширения IPv6 никак не ограничивает их число и порядок. Действительно, в каждом таком заголовке есть поле «следующий заголовок», и это позволяет источнику пакета сцепить заголовки в произвольном порядке и даже повторить заголовок данного типа несколько раз. Вопрос в том, всегда ли будет иметь смысл такая перестановка. Вероятно, нет. Поэтому сейчас нам надо найти и сформулировать дополнительные правила, которые продиктуют порядок заголовков

расширения в пакете IPv6. При этом мы, конечно же, воспользуемся нашими предыдущими наработками.

Для начала заметим, что существует ровно один заголовок расширения, который представляет интерес для всех узлов на пути пакета; это заголовок пошаговых опций. Один такой заголовок может включать в себя переменное число опций, так что ему совершенно незачем встречаться в пакете больше одного раза. Значит, мы вправе повторить сказанное выше: заголовок пошаговых опций, если он вообще есть в пакете, обязан быть вторым по порядку, следуя непосредственно за основным заголовком IPv6. Это правило упростит логику работы маршрутизаторов IPv6, так как им незачем заглядывать дальше первого заголовка расширения.

Еще два заголовка расширения нужны промежуточным адресатам: заголовок опций адресата и маршрутный заголовок. Маршрутный заголовок с ненулевым числом активных сегментов — это сигнал текущему адресату, что пакет надо продвинуть дальше. Поэтому будет вполне естественно, если промежуточный адресат остановится на маршрутном заголовке и не станет заглядывать дальше. Следовательно, опции, предназначенные всем адресатам, надо расположить перед маршрутным заголовком.

Все остальные заголовки расширения и полезная нагрузка несут информацию только конечному адресату. Если исходный пакет подлежит фрагментации, то они составляют его фрагментируемую часть и становятся, вообще говоря, недоступны транзитным узлам. Среди них особо выделяются заголовки безопасности, с помощью которых источник может защитить пакет. В частности, шифрование способно скрыть остаток пакета от посторонних глаз без какого-либо вреда для работы протокола. Поэтому естественное место для заголовков безопасности — в самом начале фрагментируемой части.

> Это же положение заголовков безопасности гарантирует нам, что последующие данные заведомо не изменяются по пути следования пакета, — одно из требований IPsec.

Наконец, в §3.1 мы допустили, что возникнут опции, имеющие смысл только для конечного адресата. Их будет разумно поместить в отельный заголовок опций адресата, расположенный после заголовка безопасности и защищенный им.

Нам осталось указать место заголовку фрагмента, но мы фактически уже сделали это выше: он возникает непосредственно за нефрагментируемой частью исходного пакета, то есть после маршрутного заголовка.

Конечно же, ни один из заголовков расширения не обязан встречаться в пакете, поэтому установленный нами порядок заголовков IPv6 — относительный, а не абсолютный. Вот его окончательный вид [§4.1 RFC 2460]:

- основной заголовок IPv6 (всегда);
- заголовок пошаговых опций (если есть);
- заголовок опций каждого адресата (если есть);
- маршрутный заголовок (если есть);
- заголовок фрагмента (если есть);
- заголовки безопасности (если есть);
- заголовок опций конечного адресата (если есть);
- полезная нагрузка (если есть).

Таким образом выходит, что ни один заголовок расширения не должен встретиться в пакете IPv6 больше одного раза, кроме заголовка опций адресата, который может встретиться самое большее дважды [§4.1 RFC 2460].

В завершение раздела отметим, что при этом порядке заголовков расширения в пакете IPv6 их можно обрабатывать последовательно, без необходимости забегать вперед или возвращаться назад за информацией.

Отклонение от указанного выше порядка заголовков может вызывать нежелательные последствия. Например, если маршрутный заголовок окажется скрыт за заголовком фрагмента, то текущий адресат сначала соберет пакет, а затем среагирует на маршрутный заголовок. Такое поведение не только грубо противоречит правилам фрагментации и сборки IPv6, но и нарушает работу PMTUD: источник фрагментирует пакет, чтобы он прошел трассу, а текущий адресат сводит его работу на «нет». Конечно, в этом примере скорее виноват сам источник, который разместил заголовки расширения в неверном порядке. Тем не менее, «игры» с порядком заголовков могут быть использованы в злонамеренных целях, и поэтому адресат обязан оставаться начеку. К сожалению, эталонная реализация *KAME* не проверяет порядка заголовков в принятых пакетах, а слепо обрабатывает их подряд. Стандарт, в свою очередь, ограничивается рекомендациями и не выдвигает четких требований к порядку заголовков в пакете IPv6. [§4.1 RFC 2460] даже требует принимать и обрабатывать заголовки в любом порядке. Эта деталь работы IPv6 явно требует уточнения.

3.3.7. Реакция на неизвестный заголовок

Модульная структура пакета IPv6 предполагает, что появятся и новые типы заголовков расширения. Следовательно, устаревшие узлы будут сталкиваться с незнакомыми заголовками. Какова будет безопасная реакция узла на это событие? Сможет ли он сигнализировать источнику о проблеме? Удастся ли ему перескочить через неизвестный заголовок? Давайте разберем эти вопросы, чтобы поставить точку в нашей работе над заголовками IPv6.

Прежде всего, давайте определимся, какие именно типы узлов на пути пакета могут столкнуться с подобной проблемой. Вряд ли это будет источник, ведь он сам составил этот пакет и знает смысл каждого заголовка в нем. Обычные маршрутизаторы не анализируют пакет дальше заголовка пошаговых опций, который они обязаны поддерживать, так что они просто не доберутся до незнакомого заголовка. А вот текущий адресат пакета обязан пройти по цепочке заголовков IPv6 до тех пор, пока он не встретит маршрутный заголовок или полезную нагрузку; вот тут-то его и может поджидать неизвестный заголовок расширения IPv6. Кроме того, в сети могут быть различные «теневые» устройства, например, межсетевые экраны или системы обнаружения и предотвращения атак. Такое устройство обычно маскируется под маршрутизатор или даже коммутатор (мост), однако ведет подробный анализ и учет протоколов на нескольких уровнях стека, отслеживая не только отдельные пакеты, но также транспортные соединения и прикладные сеансы. С появлением IPv6 ему придется разбирать всю цепочку заголовков расширения в каждом пакете, чтобы добраться до полезной нагрузки, а заодно проверить весь пакет на соответствие многоуровневой политике безопасности. Без сомнения, заголовок нового типа может озадачить программу такого устройства, если она была выпущена раньше.

Следующим шагом мы рассмотрим возможность перескочить через неизвестный заголовок расширения. Для этого представим себе процесс разбора пакета IPv6. Его алгоритм, если опустить детали, довольно прост:

1) начать с основного заголовка IPv6, который обязан быть по смещению ноль;
2) для текущего заголовка:
    а) найти поле «следующий заголовок» в текущем заголовке и запомнить его значение;
    б) найти длину текущего заголовка, чтобы определить, где его конец;
    в) перейти к следующему заголовку, который начинается сразу за текущим;
3) повторять пункт 2, пока не перейдем к протоколу следующего уровня.

Чтобы выполнить пункты 2а и 2б, надо знать формат текущего заголовка. Действительно, разные заголовки расширения по-разному кодируют свою длину и

размещают поле «следующий заголовок» по разным смещениям. Это очевидное, но не главное препятствие к перескоку через неизвестный заголовок.

Главное же препятствие скрыто в пункте 3. Допустим, мы знаем формат текущего заголовка и успешно выполнили пункты 2а–2в, но тип следующего заголовка (п. 2а) нам ничего не говорит. Поскольку заголовки IPv6 и протоколы следующего уровня заносятся в один и тот же реестр (см. §3.1), мы даже не сможем определить, что представляет собой следующий заголовок, расширение IPv6 или его полезную нагрузку. Ведь перед нами только значение байта «следующий заголовок», например, 254. Это с равным успехом может быть как новый заголовок расширения IPv6, так и экспериментальный транспортный протокол.

Невозможность перескочить через неопознанный заголовок вполне согласуется с тем, что заголовки расширения IPv6 несут в себе инструкции, обязательные для выполнения. Это отличает их от опций.

Поэтому, когда адресат пакета встречает незнакомое значение «следующий заголовок», у него нет другого выбора, как отбросить этот пакет и, возможно, выслать источнику служебное извещение «неизвестный следующий заголовок».

> Как обычно, мы говорим: «Возможно», — потому что в ряде случаев извещение высылать не следует из соображений устойчивости и безопасности. Например, плохой идеей будет извещение в ответ на групповой пакет или другое извещение. Подробности этого мы еще обсудим во время работы над протоколом служебных извещений IPv6.

А как быть в том же случае межсетевому экрану? Он не сможет провести глубокий анализ пакета, так что ответную реакцию ему должна подсказать текущая политика безопасности. Если эта политика либеральна, то пакет можно пропустить, надеясь на лучшее. Если же политика безопасности строгая, то пакет будет уничтожен.

Обратите внимание, что реакцию адресата здесь диктует протокол, а реакцию межсетевого экрана — политика.

> Напомним себе разницу между протоколом и политикой. Протокол позволяет разным сторонам вести разговор и однозначно понимать друг друга, а политика накладывает на ход их разговора дополнительные ограничения, из протокола не следующие. К примеру, русский язык — это протокол, тогда как дипломатический протокол, с нашей колокольни, никакой не протокол, а всего лишь политика. Помимо этого, политика способна указать, каким образом надо установить параметры протокола, допускающие настройку. Так, политика может потребовать определенного значения TCP MSS — или запретить громкие разговоры после отбоя.
>
> Чтобы реакция межсетевых экранов и подобных им «теневых» устройств была более гибкой, можно инкапсулировать все новые заголовки расширения в ровно один новый мета-заголовок с элементами TLV [draft-krishnan-ipv6-exthdr]. Впрочем, нам не совсем ясно, чем это лучше механизма опций, который работает по тому же принципу и уже готов к использованию.

4. IPv6 в стеке протоколов

4.1. Инкапсуляция на канальном уровне

Формально IPv6 — это всего лишь новая версия IP, которая может свободно сосуществовать с IPv4 благодаря полю «**версия**». Поэтому IPv6 не требует обязательных изменений в уже существующих протоколах канальной инкапсуляции IP, если только эти протоколы не содержат явных привязок к тем или иным аспектам IPv4.

Возьмем, к примеру, протокол SLIP [RFC 1055]: он уже был готов передавать пакеты IPv6, когда IPv6 еще не было и в помине. Более того, SLIP способен передавать

вперемежку пакеты любых версий IP, так как приемник всегда может узнать версию IP данного пакета из первого полубайта в его заголовке.

> Как мы помним, в SLIP тип инкапсулируемого протокола указывается неявно, по предварительному соглашению сторон. Поэтому канал SLIP без перенастройки сторон может передавать только один протокол сетевого уровня. Однако это не значит, что в кадр SLIP можно вложить только пакет IP; например, без малого 20 лет назад будущий автор этого курса на спор написал драйвер *MS DOS*, инкапсулирующий IPX в SLIP, и потом сыграл с друзьями-спорщиками немало партий в *Doom* по этому каналу, вполне отвечавшему модели OSI.
>
> Между прочим, это неплохой аргумент в споре о том, являются ли IPv4 и IPv6 разными протоколами или же версиями одного протокола: если бы они были разными протоколами, то не смогли бы работать вместе по одному каналу SLIP, а значит, это версии одного протокола.

4.1.1. Ethernet

Из тех же соображений мы могли бы сказать, что инкапсуляция IP в кадры Ethernet [RFC 894] применима к любой версии IP. Однако в этом случае практика вносит свои коррективы. Оказывается, существуют сетевые устройства, так называемые «коммутаторы четвертого уровня», которые анализируют разные поля в заголовках *Ethernet*, IPv4, TCP, UDP, но забывают проверить поле версии IP [[60], [61]]! То есть такое устройство, видя значение 0x800 в поле **Ether Type**, немедленно решает, что внутри кадра содержится пакет IPv4. Масштаб этой проблемы оказывается достаточно большим, так что придется оставить **Ether Type** 0x800 для IPv4 и назначить IPv6 отдельное значение **Ether Type**: 0x86DD [§3 RFC 2464].

> Радя Перлман в своей традиционной лекции [[62]] сетует, что текст стандартов IPv4 [RFC 791] и IPv6 [RFC 2460] нигде явным образом не предписывает *проверять* поле версии IP на входе. Мы готовы согласиться с уважаемой Радей, что это явная недоработка. Видимо, виновата здесь опять история. Ведь изначально стандарты *Internet* были ориентированы на высококвалифицированных и здравомыслящих специалистов, для которых строгий и безопасный алгоритм приема однозначно следовал из формата передаваемых данных.[63] Увы, сегодня уровень среднего читателя RFC заметно понизился по сравнению с 1970-м годом, и это можно компенсировать только четкостью и строгостью самих стандартов, не оставляющих места разночтениям.[64]
>
> Самостоятельно обсудите, как следует *на входе* обрабатывать пакет IPv6, принятый с Ether Type 0x800, и пакет IPv4, пришедший с Ether Type 0x86DD. Оптимальное решение вам подскажет принцип Постела.

Еще один аспект инкапсуляции IP в *Ethernet* — это разрешение групповых адресов. Наивным решением было бы привлечь для этой цели ARP или аналогичный механизм, когда источник группового пакета сначала спрашивал бы, кто состоит в данной группе, и узнавал бы таким способом адреса MAC-48 ее членов. Однако на поверку это решение никуда не годится, потому что групповое вещание — основа для автоматического обнаружения сетевых ресурсов. Необходимость ARP в данном случае будет означать

---

[60] См. сообщение *Steve Deering* «*Why a new etype of IPv6*» в списке рассылки *internet-history* <http://www.postel.org/pipermail/internet-history/2010-September/001437.html>.

[61] Radia Perlman. Interconnections: Bridges, Routers, Switches, and Internetworking Protocols (2nd Edition). §9.8.1.

[62] Radia Perlman. Mythology and Folklore of Network Protocol Design. http://www.isoc-au.org.au/Events/PerlmanNov05.pdf

[63] Именно поэтому «протокол» и «формат сообщения» в TCP/IP долгое время были синонимами.

[64] Справедливости ради заметим, что достаточно сложный протокол уже нельзя свести к формату его сообщений — необходимы также явные правила поведения сторон.

обнаружение с целью обнаружения — верный путь к «проблеме курицы и яйца», когда узлу надо послать пакет, чтобы запросить недостающие сведения, а эти сведения нужны, чтобы послать пакет. Поэтому давайте выдвинем такое требование: пусть групповое вещание IP привлекает только локальные механизмы разрешения адреса. Источник группового пакета должен быть способен преобразовать групповой адрес назначения IP в отвечающий ему канальный адрес MAC-48 своими силами, не опрашивая другие узлы. Тогда исходящий групповой пакет можно будет передать на канальный уровень сразу же, опираясь только на информацию, которая уже содержится в нем.

Как мы помним, групповое вещание IPv4 вполне отвечало этому требованию, достигая разрешения групповых адресов простой подстановкой битов. А именно 23 младших бита группового адреса IPv4 помещались в соответствующие младшие биты группового адреса MAC *01-00-5E-00-00-00* [§6.4 RFC 1112]. Такое преобразование не было взаимно однозначным, поскольку в групповом адресе IPv4 переменные 28 бит.[65] В результате узел мог получить чужой групповой пакет, и поэтому он обязательно проверял адрес назначения: действительно ли он состоит в данной группе IP на данном интерфейсе?

Давайте оставим подобный механизм разрешения групповых адресов и в IPv6. Но, чтобы уменьшить вероятность конфликтов, мы расширим поле соответствия. Пусть из группового адреса IPv6 в групповой адрес MAC попадают 4 младших байта, то есть 32 бита. Тогда нам потребуется другой префикс MAC, длиной 16 бит. На эту роль назначен префикс 0x3333, отвечающий адресам от *33-33-00-00-00-00* до *33-33-FF-FF-FF-FF*.

> Убедитесь, что перед вами групповые адреса MAC.
>
> Еще можно заметить, что это локальные, а не глобальные адреса MAC. В отличие от *01-00-5E*, новый диапазон OUI *33-33-XX* не пришлось покупать у *IEEE*.
>
> «3333 Coyote Hill Road» — это был адрес Исследовательского центра *Xerox* в Пало-Альто, откуда появился *Ethernet* [§2.3.1 RFC 5342].
>
> Обсудите недостатки использования в данном случае локальных адресов MAC. (Возможен конфликт с протоколами местного значения; однако фильтрация по адресам сетевого уровня предотвратит нежелательные последствия.)

Вот пример преобразования: групповым адресам IPv6 *FF02::C001:DEAD:BEEF* и *FF0E::BA1D:DEAD:BEEF* отвечает один и тот же групповой адрес MAC-48 *33-33-DE-AD-BE-EF*.

Хотя конфликты при разрешении групповых адресов почти безвредны и лишь слегка понижают эффективность работы сети, их вероятность можно понизить, ограничив диапазон значений идентификатора группы IPv6 на уровне политики, а не протокола. Так, [RFC 3307] требует придерживаться таких диапазонов:

- общепринятые групповые адреса: `0x0000 0000`–`0x3FFF FFFF`;
- постоянно назначенные группы: `0x4000 0000`–`0x7FFF FFFF`;
- временно занятые адреса: `0x8000 0000`–`0xFFFF FFFF`.

Разница между общепринятыми групповыми адресами и постоянно назначенными группами такова: общепринятые групповые адреса назначены в системных целях и нужны для работы протоколов стека IPv6, тогда как постоянно назначенные группы просто созданы для поддержки глобальных проектов, важных для всего *Internet*. Например, `0x0000 0001` («все узлы») — это общепринятый групповой адрес, тогда как `0x4040 4040` (NTP) — всего лишь постоянно назначенная группа.

Те же самые требования закреплены в реестре IANA для групп IPv6 [[66]].

Как нетрудно видеть, ни одно предписанное значение не выходит за пределы 32 бит, так что конфликты при разрешении адресов оказываются исключены. Тем не менее,

[65] 4 старших бита фиксированы и составляют префикс `1110` класса D.

[66] http://www.iana.org/assignments/ipv6-multicast-addresses/

групповой слушатель IPv6 по-прежнему обязан фильтровать входящие пакеты, принимая только действительно адресованные его группам на данном интерфейсе.

4.1.2. PPP

Пример канального протокола, поддержка IPv6 в котором неизбежно требует дополнительной работы, — это PPP. Чтобы обеспечить высокий уровень услуг, PPP готовит канал к работе каждого сетевого протокола, например, согласуя адреса на его концах. Это задание PPP делегирует дочернему протоколу семейства NCP. Уже существующий NCP для IP, IPCP [RFC 1332], может работать только с адресами длиной 32 бита, так что он не годится для IPv6. Посему для IPv6 необходим свой собственный NCP — **IPV6CP** [RFC 5072].

Прежде всего, назначим IPv6 и его NCP уникальные номера из реестра параметров PPP [67], чтобы кадры PPP с ними можно было однозначно демультиплексировать. У IPv6 это 0x0057, а номер его NCP по традиции на 0x8000 больше, то есть 0x8057. Теперь мы можем инкапсулировать оба протокола в PPP, и пора подумать над функциями IPv6CP.

Какое задание, специфичное именно для IPv6, мы можем поставить перед IPV6CP? По нашей задумке (см. §2.7), сетевому интерфейсу IPv6 в высшей степени полезно иметь какой-то эквивалент адреса EUI-64, чтобы на его основе можно было создать условно уникальный идентификатор интерфейса. Хорошо, когда у интерфейса есть настоящий, глобально уникальный EUI-64. Канальный адрес MAC-48 — тоже неплохой вариант, потому что его можно превратить в EUI-64 по общепринятому рецепту, который мы обсудили в §2.7. Но как быть с PPP, где явных канальных адресов вообще нет? Ведь каналу PPP вполне достаточно неявной адресации: всякий кадр PPP, переданный в канал, будет принят узлом на другом конце канала, и только им. Вот пусть это и будет работой IPV6CP: согласовать два уникальных идентификатора на концах канала PPP. Так как в конечном итоге нам нужны идентификаторы интерфейса в формате «модифицированный EUI-64», то пусть IPV6CP согласует непосредственно их, а не промежуточные EUI-64 или MAC-48.

Каким требованиям должны удовлетворять эти идентификаторы [§4.1 RFC 5072]?

- Идентификаторы интерфейсов на концах канала обязаны быть разными. В то же время, глобальная уникальность им совершенно не требуется. Ведь уникальность составленных из них адресов IPv6 обеспечит префикс подсети, назначенный каналу.

  Зона уникальности полученных адресов IPv6 будет зависеть от области действия префикса подсети: из глобального префикса мы получим глобально уникальные адреса, из внутрисайтового префикса — адреса, уникальные в пределах данного сайта, а из внутриканального префикса — внутриканальные адреса для данного канала PPP.

- Желательно, чтобы идентификаторы были постоянными и не менялись, к примеру, при перезагрузке узлов. Каждый узел вправе использовать все доступные ему источники уникальности, чтобы построить такой идентификатор. В первую очередь, на эту роль претендуют аппаратные адреса EUI-64 и MAC-48 других сетевых интерфейсов узла. Если таковых нет, то следующий кандидат — это серийные номера основных компонентов узла. Наконец, как крайнее средство, идентификатор все же может быть случайным.
- Составляя идентификатор, узел обязан соблюдать формат «модифицированный EUI-64» (см. §2.7). В частности, бит **U/L** нужно установить в единицу тогда, и только тогда, когда идентификатор создан на основе настоящего глобального адреса EUI-64 или MAC-48, назначенного

[67] http://www.iana.org/assignments/ppp-numbers

именно этому узлу. Само преобразование EUI-64 и MAC-48 в «модифицированный EUI-64» мы уже подробно обсудили там же, в §2.7.

Само согласование идентификаторов представляет собой простой и доступный пример работы механизма опций PPP. В данном случае достаточно сообщений **Conf-Req**, **Conf-Rej**, **Conf-Nak** и **Conf-Ack** (см. Табл. 10), а код соответствующей опции IPV6CP — 1 [§4 RFC 5072].

Табл. 10. Сообщения IPV6CP при согласовании идентификатора интерфейса

Сообщение	**Смысл**	**Допустимые варианты ответа**
Запрос		
Conf-Req	Источник хотел бы присвоить *себе* (не удаленной стороне!) такой-то идентификатор интерфейса (прилагается к сообщению). Пока что это **пробный идентификатор** (*tentative identifier*), потому что источник только испытывает его. Если идентификатор в сообщении нулевой, то источник не может сам справиться с выбором и просит удаленную сторону о помощи.	**Conf-Rej**, **Conf-Nak**, **Conf-Ack**
Варианты ответа		
Conf-Rej	Источник вообще не поддерживает данную опцию. Удаленная сторона должна продолжить согласование сообщением **Conf-Req** без этой опции или прервать сеанс.	**Conf-Req**
Conf-Nak	Источник поддерживает опцию, но не удовлетворен ее значением. Видимо, значение было нулевым, или источник сам хотел назначить себе такой же идентификатор. Поэтому источник предлагает *удаленной стороне* свой вариант *ее,* удаленной стороны, идентификатора (прилагается к сообщению). Удаленная сторона должна рассмотреть этот вариант и послать новый **Conf-Req**.	**Conf-Req**
Conf-Ack	Источник отвечает, что согласен с выбором удаленной стороны, поскольку он не нулевой и отличается от пробного идентификатора самого источника. На этом согласование завершено.	Нет

Как мы помним, согласование опции PPP проходит независимо в каждом из двух возможных направлений. Поэтому стороны А и Б ведут два, на первый взгляд, независимых диалога IPV6CP. Итогом одного из них будет идентификатор стороны А, а другого — идентификатор стороны Б.

Механизм согласования IPV6CP допускает следующий сценарий: узел Б объявляет нулевой идентификатор, и тогда узел А подбирает идентификатор не только для себя, но и для удаленного узла Б. В этом случае бит **U/L** в идентификаторе для Б надо сбросить, даже если он получен из глобального EUI-64. Ведь этот EUI-64 принадлежит узлу А, а не узлу Б. (Как исключение, этот бит можно установить в единицу, если узел А имел в запасе еще один глобальный EUI-64, а теперь передает его узлу Б в безраздельное пользование на время данного сеанса PPP [§4.1 RFC 5072].)

Чтобы процесс согласования идентификаторов не зациклился, сформулируем несколько дополнительных правил поведения стороны IPV6CP:

- не повторяться, то есть не предлагать один и тот же идентификатор дважды в **Conf-Req** или **Conf-Nak**;

Тем не менее, идентификатор, который уже фигурировал в **Conf-Nak**, вполне может появиться в ответном **Conf-Req**. Однако в этом случае идентификатор придет от удаленной стороны, а мы говорим о поведении одной стороны, которую мы по традиции считаем нашей локальной.

- не капризничать, то есть соглашаться с первым же подходящим идентификатором:
  - если идентификатор в принятом нами **Conf-Req** не совпадает с нашим собственным пробным значением, которое мы объявили в исходящем **Conf-Req**, то мы немедленно соглашаемся с ним, отвечая **Conf-Ack**. Так мы соглашаемся с выбором удаленной стороны для нее самой;
  - если идентификатор во входящем **Conf-Nak** не совпадает с тем, что мы сами предложили удаленной стороне в последнем переданном нами **Conf-Nak**, то мы обязаны принять его в качестве нашего пробного значения и выслать об этом **Conf-Req**. Здесь мы соглашаемся с предложением удаленной стороны для нас.

Обратите внимание, как между встречными диалогами IPV6CP возникает скрытая зависимость. Чтобы правильно ответить на входящее сообщение **Conf-Req**, нам надо помнить наш собственный пробный идентификатор, который мы наверняка сами объявили в исходящем **Conf-Req**. А правильная реакция на входящий **Conf-Nak** возможна только при условии, что мы запомнили значение опции из посланного нами самими **Conf-Nak**.

Эти правила склоняют согласование идентификаторов к одному из двух вероятных путей:

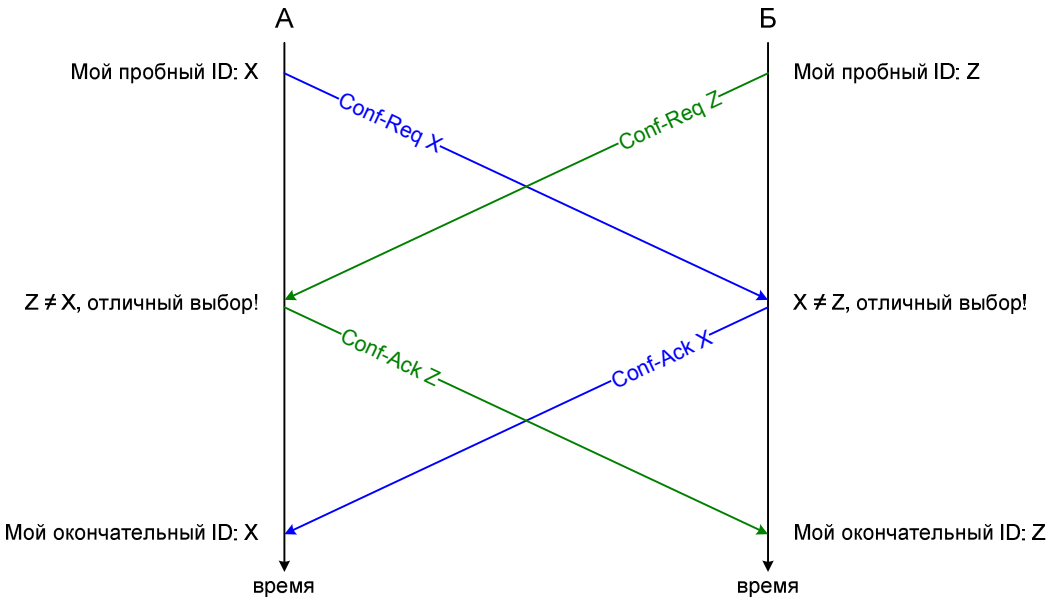
- или обе стороны с самого начала выбирают себе разные пробные идентификаторы и просто соглашаются друг с другом;
- или же сначала возникает конфликт пробных идентификаторов, вслед за которым стороны предлагают разные идентификаторы друг другу.

Конечно, конфликт может возникнуть больше одного раза подряд, но вероятность этого события низка. Точнее, она может быть высокой, но только если мы имеем дело с неудачной реализацией протокола. В частности, возможен перманентный конфликт, если стороны выбирают одно и то же пробное значение, а затем используют один и тот же детерминистический алгоритм, чтобы генерировать новые пробные значения для себя и партнера. Но на этом «патологическом» случае мы останавливаться не будем. У любой современной системы есть достаточно источников уникальности или случайности, а в крайнем случае она может запросить идентификатор у удаленной стороны, передав нулевое значение в своем первом сообщении **Conf-Req**.

Поэтому, если обе стороны передают нулевые значения в своих сообщениях **Conf-Req**, согласование надо немедленно прекратить ради устойчивости протокола [§4.1 RFC 5072].

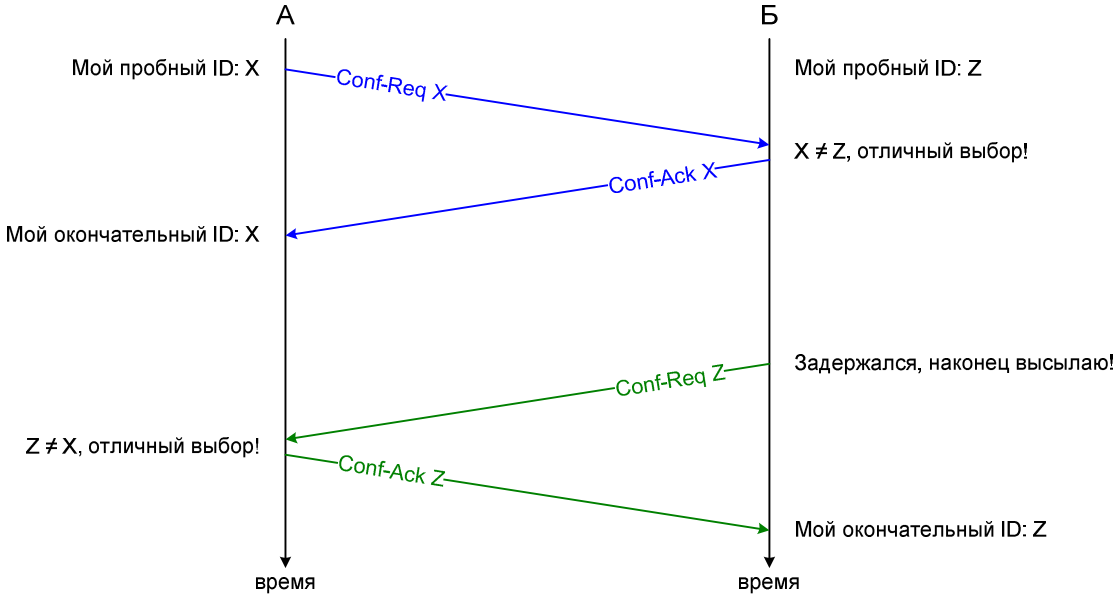
Гораздо большее влияние на ход процедуры IPV6CP окажет то, насколько одновременны встреченные диалоги. Поэтому, чтобы лучше почувствовать работу механизма IPV6CP, давайте рассмотрим процессы нулевого и первого порядков (немедленное согласие; конфликт, а затем согласие с предложением удаленной стороны) в двух крайних случаях: когда встречные диалоги совершенно синхронны или строго последовательны.

Если стороны с самого начала выбрали себе разные пробные идентификаторы, то встречные диалоги независимы, а поэтому не имеет значения, одновременны ли они, или же один следует за другим. Итог будет один: стороны утвердят самостоятельно выбранные идентификаторы.



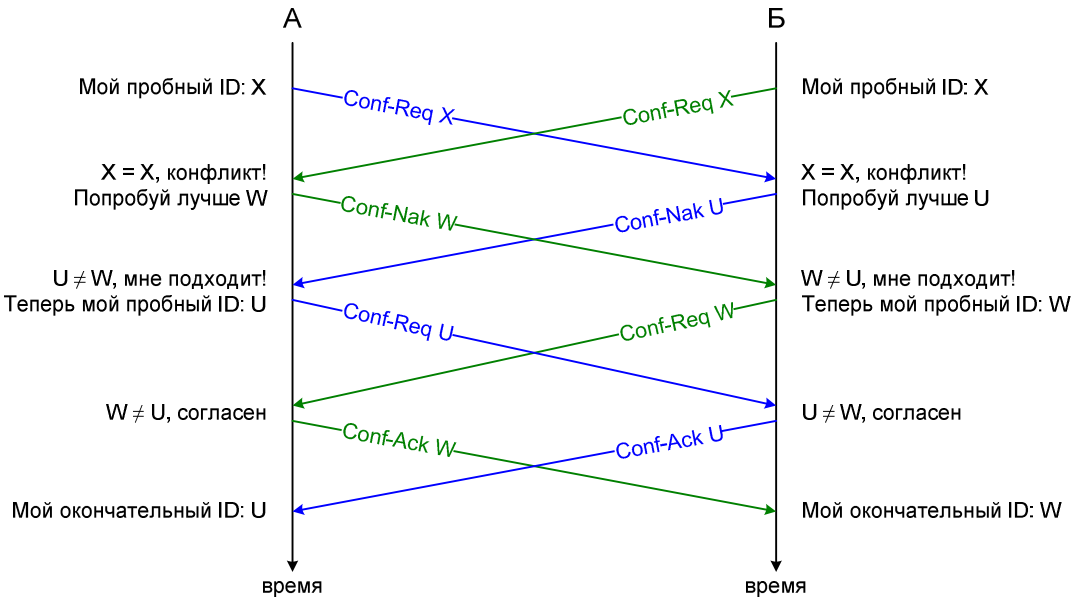
Фиг. 42. IPV6CP: Параллельное согласование без конфликта

Параллельный сценарий (Фиг. 42) в этом случае преобразуется в свой последовательный эквивалент (Фиг. 43) простым сдвигом одного из диалогов вдоль оси времени.



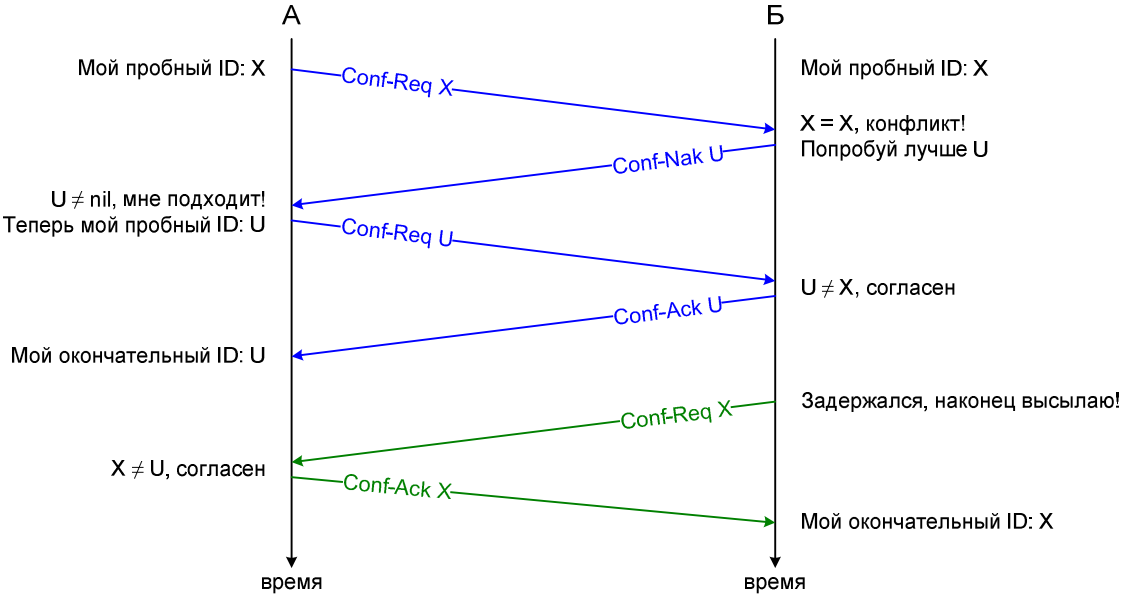
Фиг. 43. IPV6CP: Последовательное согласование без конфликта

Если же стороны случайно выбрали один и тот же идентификатор, исход согласования зависит от того, как встречные диалоги расположены друг относительно друга на оси времени. Так, если они одновременны (Фиг. 44), то каждая из сторон согласится с подсказкой удаленной стороны и выберет именно ее собственным идентификатором.



Фиг. 44. IPV6CP: Параллельное согласование и преодоление конфликта

Ну, а если диалоги будут вестись последовательно (Фиг. 45), то сторона А, открывшая согласование первой, будет вынуждена принять подсказку удаленной стороны Б, тогда как сторона Б сможет настоять на своем первоначальном выборе идентификатора.



Фиг. 45. IPV6CP: Последовательное согласование и преодоление конфликта

Пусть читатель самостоятельно составит такие же две схемы для случая, когда одна из сторон просит своего визави о помощи в выборе идентификатора, объявляя в **Conf-Req** нулевое значение.

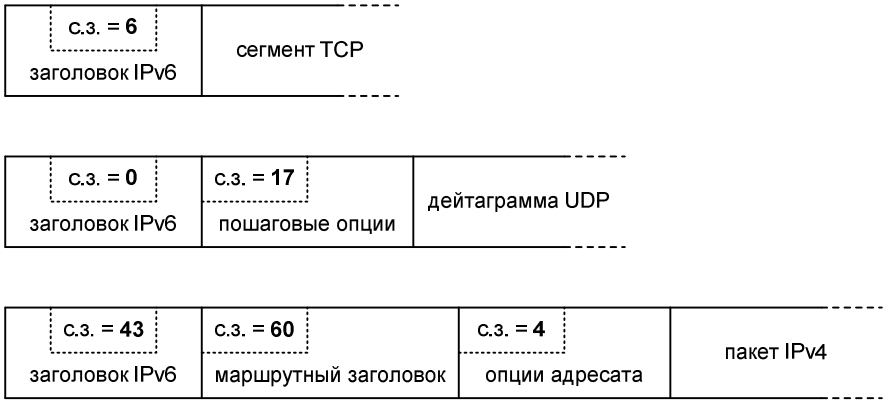
На практике сторона IPV6CP может прибегать к безобидным трюкам, чтобы обойтись без конфликта уже в нулевом порядке согласования. Так, она может отложить выбор своего пробного идентификатора до тех пор, пока не придет запрос **Conf-Req** удаленной стороны. После этого ей достаточно инвертировать хотя бы один бит в чужом идентификаторе, чтобы получить заведомо уникальный идентификатор для себя самой. Читатель может составить схему такого диалога самостоятельно. Конечно, в этом случае встает вопрос, кто пошлет **Conf-Req** первым и как избежать возможного тупика в развитии сеанса, когда ни одна из сторон не проявляет инициативу. Хотя, судя по выдачам *tcpdump*, этот трюк использует *Cisco IOS* версии 12.4, независимый обмен запросами **Conf-Req** нам кажется более надежным.

4.2. Взаимодействие с вышестоящими протоколами

В стеке TCP/IP на уровень IP непосредственно опираются протоколы нескольких видов: управляющие, то есть ICMP; транспортные, такие как TCP, UDP и SCTP; туннельные, например, IP-IP, GRE, EtherIP. О том, как нам быть с ICMP, мы поговорим в §4.3, а сейчас давайте рассмотрим вопрос инкапсуляции IPv6 в целом, сделав ударение на протоколах транспортного уровня.

> Вопреки убеждению отдельных неофитов, у IPv6 нет собственных транспортных протоколов, отличных от тех, к которым мы привыкли во времена IPv4. Задача IPv6 состоит лишь в том, чтобы модернизировать уровень IP, сохранив при этом, по возможности, его интерфейс для вышестоящих протоколов.

Прежде всего, мы должны сказать, как блок данных протокола (БДП), стоящего над IP, следует инкапсулировать в пакет IPv6. На самом деле, правила этой инкапсуляции мы уже задали, когда говорили в §3.1 о структуре заголовков пакета IPv6. Тогда мы сказали, что пакет IPv6 содержит в своем начале цепочку из одного или более простых заголовков. Каждый простой заголовок содержит поле «следующий заголовок», возможные значения которого хранятся в реестре протоколов IP [68]. Значит, последний заголовок в цепочке может ссылаться на тип полезной нагрузки пакета точно так же, как это делал заголовок IPv4. Например, если пакет IPv6 содержит в себе сегмент TCP, то последнее поле «следующий заголовок» содержит значение 6; если дейтаграмму UDP, то 17; а если пакет IPv4, то 4. Эти примеры иллюстрирует Фиг. 46.



Фиг. 46. Примеры инкапсуляции IPv6 (с.з. = следующий заголовок)

Следующим шагом будет найти и исправить все явные и неявные зависимости вышестоящих протоколов от IPv4. К счастью, у стандартных и активно применяемых протоколов их оказывается немного.

> Кстати, была предпринята попытка составить полный список зависимостей от IPv4 у протоколов, описанных в RFC. Отчет занял восемь (!) документов RFC, 3789–3796.

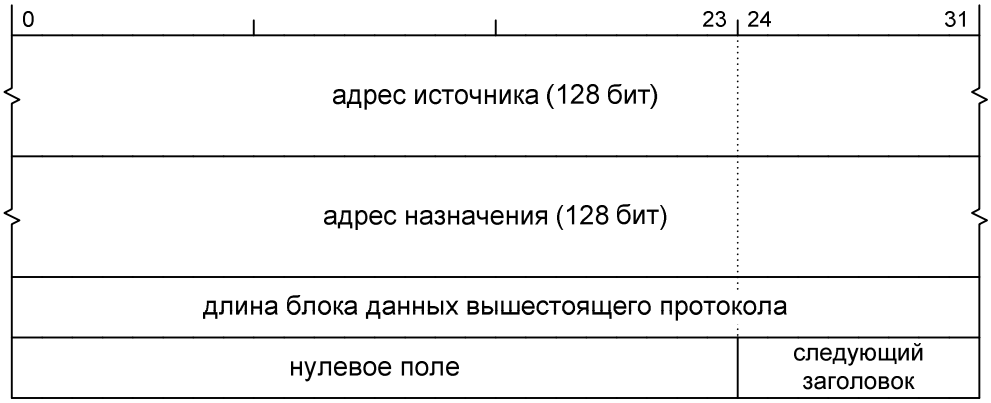
Главная зависимость TCP и UDP от IPv4 заключается в формате псевдозаголовка, который служит для вычисления контрольной суммы. Этот псевдозаголовок включает в себя адреса IPv4 источника и назначения. Чтобы TCP и UDP смогли работать поверх IPv6, потребуется новый формат псевдозаголовка, который учтет специфику нового протокола [§8.1 RFC 2460], как это показано на Фиг. 47.

> Самостоятельно изучите вопрос, нужно ли задавать новый псевдозаголовок для SCTP. (Подсказка: начните с вводной статьи [69].)

---

[68] http://www.iana.org/assignments/protocol-numbers/

[69] R. Stewart, M. Tüxen, P. Lei. SCTP: What is it, and how to use it? BSDCan 2008. http://www.bsdcan.org/2008/schedule/attachments/44_bsdcan_sctp.pdf



Фиг. 47. Псевдозаголовок IPv6 для вычисления контрольных сумм

Как и раньше, поле длины в этом псевдозаголовке содержит длину дейтаграммы вышестоящего протокола, выраженную в байтах, и не учитывает заголовки IPv6. Например, UDP может взять это значение непосредственно из заголовка UDP. В то же время, TCP явным образом не хранит длину сегмента, так что ему придется вычислить искомую длину.

> «В лоб» это вычисление можно провести, вычтя длины всех заголовков *расширения* IPv6 в данном пакете — не считая основного заголовка! — из значения поля «длина полезной нагрузки» в его основном заголовке IPv6. Альтернативный способ, близкий знатокам языка Си, — это найти разность указателей на конец пакета и заголовок TCP, приведенных к типу *char* *. Главное — не перепутать при этом конец пакета с последним байтом пакета, потому что их смещения отличаются на один байт.

Что касается поля «следующий заголовок», то оно теперь хранит тип вышестоящего протокола, а не просто копирует такое же поле основного заголовка IP; ведь в пакете могут быть заголовки расширения, а тогда тип вышестоящего протокола будет скрыт в самом последнем из них.

Кроме того, протокол UDP неявно полагается на контрольную сумму IPv4, когда объявляет вычисление контрольной суммы UDP необязательным: он предполагает, что заголовок IP защищает себя от повреждения. Но заголовок IPv6 сам по себе не защищен, и поэтому вычисление контрольной суммы UDP при работе поверх IPv6 становится обязательным [§8.1 RFC 2460]. Если дейтаграмма UDP с нулевым значением в поле «контрольная сумма» получена в пакете IPv6, то ее следует уничтожить.

> Освежим нашу память и вспомним несколько фактов. Контрольная сумма TCP/IP вычисляется в 16-битном обратном коде, где есть два арифметически эквивалентных нулевых значения, 0x0000 и 0xFFFF. Значение поля «контрольная сумма» в TCP и UDP — это результат побитовой инверсии вычисленной суммы. (В обратном коде инверсия битов приводит к смене знака числа, так что неудивительно, что сумма всех слов сообщения, включая его контрольную сумму, должна быть нулевой.) Поэтому мы можем отличить нулевую контрольную сумму от неизвестной, задав дополнительное правило. Для этого мы предписываем хранить истинно нулевую сумму только как 0x0000, что после инверсии даст нам 0xFFFF. А значение суммы 0xFFFF, что после инверсии дает 0x0000, мы назначаем особому случаю, когда источник дейтаграммы сумму вообще не посчитал. Просто и изящно, не правда ли? ☺
>
> Трансляцию NAT вполне можно обобщить на обе версии IP и применить, чтобы обеспечить взаимодействие хостов IPv4 с хостами IPv6. Но как быть транслятору NAT46 или NAT64, если со стороны IPv4 он получает дейтаграмму UDP с неопределенной контрольной суммой? В этом случае при ее трансляции в IPv6 придется вычислить значение контрольной суммы и поместить в заголовок UDP. Тем самым транслятор заверит целостность дейтаграммы без достаточных на то оснований, ведь дейтаграмма могла поступить к транслятору на вход уже

поврежденной! Вот нам еще один пример, как NAT нарушает сквозной принцип, лежащий в основе архитектуры *Internet*.

А как появление IPv6 скажется на работе туннельных протоколов? Если такой протокол сконструирован надлежащим образом, то изменений в нем не потребуется. Так, например, протоколу GRE достаточно учесть, что IPv6 получил свой собственный **Ether Type** 0x86DD. Еще при туннелировании IPv6 будет полезно включить вычисление контрольной суммы GRE, чтобы обеспечить дополнительную защиту пакетов от повреждения.

Туннельный протокол IP-IP тоже готов к работе с IPv6 как снаружи [RFC 2473], так и внутри [§3 RFC 3056, §3.1 RFC 4213] туннеля. Достаточно учесть при инкапсуляции, что исходному пакету IPv4 отвечает номер протокола IP 4, а исходному пакету IPv6 — 41 [70]. Вполне можно строить и смешанные туннели «IPv4 внутри IPv6» и «IPv6 внутри IPv4». Сводная таблица таких способов инкапсуляции — Табл. 11.

Табл. 11. Туннельная инкапсуляция IP-IP после появления IPv6

Заголовки	IPv6-в-IPv6	IPv6-в-IPv4	IPv4-в-IPv6
Внешний	IPv6: «след. заголовок» = 41	IPv4: «протокол» = 41	IPv6: «след. заголовок» = 4
Внутренний	IPv6	IPv6	IPv4

Наконец, у всех типов туннелей есть общая важная деталь — управление фрагментацией. Когда внешний протокол туннеля — IPv4, то узел на входе в туннель (инкапсулятор) может разрешить или запретить транзитную фрагментацию туннельных пакетов с помощью флага **DF**. Напротив, в среде IPv6 транзитная фрагментация всегда запрещена; это касается и туннельных пакетов, внешний протокол которых — IPv6, а поэтому их длиной обязан управлять сам инкапсулятор. Здесь его поджидает, как минимум, одна сложность: внутренний MTU туннеля обязан быть не меньше 1280 байт, тогда как и внешний MTU трассы (PMTU) туннеля может оказаться равным 1280 байт. Инкапсуляция исходного пакета длиной 1280 байт заведомо даст туннельный пакет длиной более 1280 байт, который не пройдет данную трассу без фрагментации. Это означает, что в среде IPv6 инкапсулятор обязан быть готов к фрагментации туннельных пакетов, потому что в общем случае без нее не обойтись. Самый простой, хотя и не слишком эффективный подход к выбору длины туннельного пакета — это принять нижнюю оценку PMTU туннеля равной 1280 байт и фрагментировать туннельные пакеты исходя из этого. Тогда внутренний MTU туннеля будет ограничен только диапазоном длины пакета IPv6 до фрагментации за вычетом длины туннельных заголовков: $65575 - X$. Подход посложнее — это выполнение процедуры PMTUD инкапсулятором.

В RFC 4459 можно найти более обширный анализ теории и практики управления фрагментацией в туннеле.

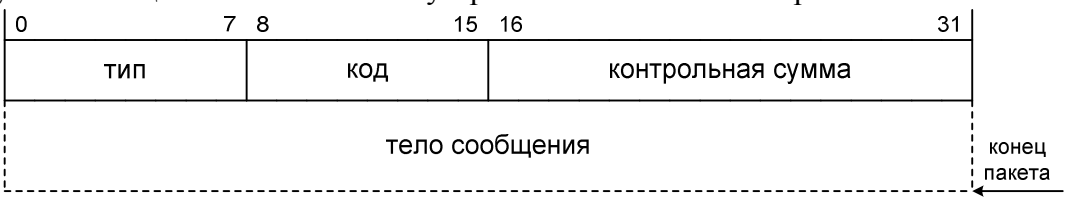
4.3. Управление

Хотя ICMP [RFC 792] вполне можно было бы «притянуть за уши» к управлению IPv6, новая версия IP — это хороший повод пересмотреть и сопряженный с ним протокол управления. Чтобы не заботиться о совместимости с ICMP, давайте назначим новому протоколу другой номер, 58, из реестра протоколов IP. Новорожденный протокол мы назовем «ICMP версии 6», а кратко — **ICMPv6** [RFC 4443], хотя заголовок ICMP и не содержит поля версии. Чтобы избежать разночтений, «старый» ICMP для IPv4 мы теперь обозначим как ICMPv4, по крайней мере, при его сравнении с ICMPv6.

[70] http://www.iana.org/assignments/protocol-numbers/

Мы предполагаем, что сообщение ICMPv6 никогда не окажется внутри пакета IPv4 и наоборот, потому что такие сочетания лишены смысла. По-хорошему, реализации сетевого стека должны явно проверять это условие на входе. В особых случаях, например, при трансляции NAT из IPv6 в IPv4 и обратно, придется также транслировать и сообщения управляющих протоколов. Процедуру и саму возможность такой трансляции мы обсуждать не будем.

Какие требования мы предъявим к протоколу управления IPv6? В первую очередь, он не должен отставать от IPv6 в плане гибкости и расширяемости. Это требование мы выполним, «собрав» сообщение ICMPv6 из двух частей. У первой части формат будет постоянным, а у второй — переменным, как изображено на Фиг. 48. Это не новость, потому как сообщения ICMPv4 были устроены тем же самым образом.



Фиг. 48. Общая структура сообщения ICMPv6

Значение в поле «тип» относит сообщение ICMPv6 к одной из функциональных групп. Например, сообщения «адресат недоступен» и «запрос эха» несут совершенно разную смысловую нагрузку и явно претендуют на разные типы. Кроме того, именно поле «тип» определяет формат второй части сообщения.

Поскольку ICMPv6 — отдельный протокол, номера его типов никак не связаны с номерами типов ICMPv4. Номера типов ICMPv6 мы будем распределять с нуля, так что у нас есть возможность придать им определенную структуру. В ICMPv4 типы извещений об ошибках шли вперемежку с чисто справочными типами. Теперь ради удобства давайте проведем между ними границу. Пусть типы извещений об ошибках получают номера из диапазона 0–127, а справочные типы — из диапазона 128–255. Тогда, чтобы узнать характер сообщения ICMPv6, достаточно взглянуть на старший бит в его поле «тип»: 0 означает произошедшую ошибку, а 1 — справочные сведения [§2.1 RFC 4443].

Толкование поля «код» зависит от типа сообщения. Например, с его помощью можно поделить сообщения одного типа на более узкие группы, чем мы воспользуемся в текущем разделе.

Формат переменной части также целиком зависит от типа сообщения. Эту часть мы назовем **телом сообщения** (*message body*). Она простирается до конца пакета, а длину ее можно вычислить, исходя из длины полезной нагрузки в заголовке IPv6.

Согласно принятой в IPv6 модели защиты данных от повреждения, контрольная сумма ICMPv6 должна защищать не только само сообщение, но и заголовок IPv6. Достичь этого можно, используя при вычислении контрольной суммы псевдозаголовок стандартного формата (см. §4.2). В остальном правила обычные: на время расчета контрольной суммы псевдозаголовок условно помещается перед заголовком ICMPv6, а само поле «контрольная сумма» явно или «мысленно» обнуляется [§2.3 RFC 4443].

Какое значение надо поместить в поле «следующий заголовок» такого псевдозаголовка? (Ответ: 58 — ICMPv6.)

Заметим, что ICMPv4 не использовал псевдозаголовка и вычислял только контрольную сумму собственного БДП, вложенного в пакет IPv4. Так что применение псевдозаголовка в ICMP — это новшество IPv6, вызванное отсутствием контрольной суммы в основном заголовке IPv6.

Сообщение ICMPv6 составляет всю полезную нагрузку пакета IPv6, так что поле длины заголовку ICMPv6 не требуется: длину сообщения ICMPv6 можно вычислить, зная длину всего пакета и структуру его заголовков. Это вычисление понадобится, чтобы

заполнить псевдозаголовок, так как он содержит длину блока данных вышестоящего протокола (см. Фиг. 47).

> Напомним о двусмысленности термина «полезная нагрузка» в IPv6. В общеупотребительном смысле, это последняя секция пакета, после основного заголовка и заголовков расширения IPv6, занятая блоком данных вышестоящего протокола: дейтаграммой UDP, сегментом TCP, сообщением ICMPv6. В узком смысле поля «длина полезной нагрузки» из основного заголовка IPv6 это весь пакет за вычетом основного заголовка. При заполнении псевдозаголовка эти два толкования встречаются, так как в заголовке IPv6 указана «длина полезной нагрузки» (смысл №2), а надо найти длину блока данных вышестоящего протокола (смысл №1). Это касается, в первую очередь, узла назначения, так как он должен разобрать и проверить входящий пакет.

Перейдем теперь к **типам** сообщений ICMPv6. Для начала, в Табл. 12, мы определим только самые необходимые из них [§2.1 RFC 4443], а для будущих назначений создадим реестр [71].

Табл. 12. Основные типы сообщений ICMPv6

Тип	Название (рус.)	Название (англ.)	Ссылка
Извещения об ошибках			
1	Адресат недоступен	*Destination Unreachable*	[§3.1 RFC 4443]
2	Пакет слишком велик	*Packet Too Big*	[§3.2 RFC 4443]
3	Время истекло	*Time Exceeded*	[§3.3 RFC 4443]
4	Проблема в параметре	*Parameter Problem*	[§3.4 RFC 4443]
100	Для частных экспериментов		[§2.1 RFC 4443]
101	Для частных экспериментов		[§2.1 RFC 4443]
127	Резерв для расширения		[§2.1 RFC 4443]
Справочные сообщения			
128	Запрос эха	*Echo Request*	[§4.1 RFC 4443]
129	Ответ эхом	*Echo Reply*	[§4.2 RFC 4443]
200	Для частных экспериментов		[§2.1 RFC 4443]
201	Для частных экспериментов		[§2.1 RFC 4443]
255	Резерв для расширения		[§2.1 RFC 4443]

> Тип 0 не назначен, скорее всего, из-за уже знакомого нам предубеждения перед нулевыми значениями, и это вполне оправдано: ноль часто означает «нет данных».

Что еще изменилось по сравнению с основными типами ICMPv4? Прежде всего, извещения «пакет слишком велик» и «время истекло» получили отдельные типы. Теперь «адресат недоступен» говорит только о невозможности доставить пакет из-за проблем на уровне адресации и маршрутизации, или же ввиду ограничений политики [§3.1 RFC 4443], например:

- код 0: нет маршрута к адресу назначения, то есть неизвестно, куда продвигать пакет;
- код 1: обмен пакетами с адресатом запрещен в административном порядке, например, политикой сетевой безопасности;
- код 2: адресат находится за пределами зоны источника, то есть предотвращено нарушение границ зоны, такое как попытка направить пакет узлу совершенно другой сети, используя внутриканальный адрес источника;

[71] http://www.iana.org/assignments/icmpv6-parameters

- прочие коды «адресат недоступен» приведены в [§3.1 RFC 4443] и реестре [72].

Какую практическую пользу принесет то, что у сообщения «пакет слишком велик» теперь есть свой тип, а не только код в рамках типа «адресат недоступен», как это было в ICMPv4? Прежде всего, инженерам по сетевой безопасности будет легче составлять политики, которые запрещают произвольные извещения «адресат недоступен», но при этом не блокируют работу PMTUD. В IPv4 это было серьезной проблемой. Во-первых, сами инженеры часто забывали разрешить именно это сочетание типа и кода ICMPv4. А во-вторых, было много примеров сетевого программного обеспечения, которое позволяло фильтровать ICMPv4 только по значению поля «тип», но не «код», так что невозможно было избирательно разрешить PMTUD. В ICMPv6 этот опыт нельзя было не учесть, так как важность правильной работы PMTUD возросла многократно из-за новых правил фрагментации, когда решение о размере пакета принимает только его источник (см. §3.3.4).

В идеале, фильтрация ICMP не должна представлять сложности для сетевого экрана, полноценно отслеживающего прикладные сеансы. Ему достаточно пропускать сообщения ICMP, относящиеся к существующим сеансам, и блокировать «залетные» сообщения. К сожалению, не все межсетевые экраны рассматривают сигнализацию ICMP как неотъемлемую часть сеанса, но это всего лишь практическое ограничение, которое можно было бы устранить доработкой программы. Более фундаментальная проблема состоит в том, что сообщение ICMPv4 может содержать недостаточно сведений, чтобы точно отнести его к определенному прикладному сеансу. Мы устраним эту проблему в ICMPv6 еще до конца данного раздела.

Далее, как мы уже отметили, говоря о фрагментации и сборке в §3.3.4, извещение «пакет слишком велик» обязано содержать в себе значение MTU выходного канала, куда «не уместился» пакет. Благодаря этой информации, процедура PMTUD сойдется гораздо быстрее.

Извещение «проблема в параметре» подходит для того, чтобы сообщать источнику, помимо прочих проблем в составленном им пакете, о неопознанных опциях и неизвестных значениях «следующий заголовок». Чтобы различать эти проблемы, можно назначить им разные коды [§3.4 RFC 4443]:

- код 0: недопустимое значение какого-либо поля в заголовке либо неверно сформированный пакет, например, длина фрагмента с **M** = 1 не кратна восьми;
- код 1: неопознанное значение «следующий заголовок» — возможно, адресат не поддерживает это расширение или протокол;
- код 2: неопознанная опция IPv6 — высылается, только если старшие биты кода опции требуют такого извещения (см. §3.3.2).

Но почему тип 3 называется «время истекло»? Ведь в §3.2 мы приняли решение, что теперь вместо поля «время жизни пакета» будет поле «предельное число шагов». Конечно, это так, но тип 3 охватывает два разных события, и второе из них по-прежнему связано со временем [§3.3 RFC 4443]:

- код 0: превышено предельное число шагов, то есть пакет предположительно зациклился в сети, отчего значение его поля «предельное число шагов» достигло нуля;
- код 1: истек тайм-аут сборки, то есть адресат не дождался прихода всех фрагментов пакета.

[72] http://www.iana.org/assignments/icmpv6-parameters

К сожалению, текст [§3.3 RFC 4443] можно трактовать так, будто маршрутизатор должен проверить поле «предельное число шагов» во входящем пакете, как только он принят из сети. Как мы знаем из §3.2 настоящего курса, на самом деле этому обязательно предшествует вывод, что пакет транзитный и подлежит продвижению, так как иначе поле «предельное число шагов» вообще не проверяется. Видимо, авторы [RFC 4443] руководствовались моделью идеального маршрутизатора, у которого все пакеты — транзитные, но забыли сделать на этом акцент. Увы, в наши дни такие недомолвки чреваты неправильными реализациями.

Дополнительные типы сообщений ICMPv6 возникнут по мере необходимости. Хотя у каждого из них будет свой формат, возможны и общие детали. Одну важную деталь мы сейчас обнаружим в извещениях об ошибках.

Модель управления ошибками IPv6 остается такая же, как и в IPv4: извещение об ошибке высылается только при невозможности обработать принятый пакет IP. То есть за каждым сообщением об ошибке стоит определенный пакет-виновник. А значит, к извещению об ошибке обязательно надо приложить хотя бы начало пакета-виновника, чтобы его источник смог опознать пакет и принять необходимые меры. Это правило касается всех извещений об ошибках ICMPv6. Оно не ново — подобное правило было и в ICMPv4.

По одной из точек зрения, даже минимальный MTU IPv4 был выбран в связи с сигнализацией ошибок таким образом. Ведь в 68 байт как раз умещается внешний заголовок IPv4 (20 байт), заголовок ICMPv4 (8 байт), заголовок IPv4 виновника (20 байт), и остается еще 20 байт для заголовка TCP.

Однако теперь у нас есть шанс устранить одну застарелую проблему. Она была в том, что сообщение ICMPv4 содержало слишком мало байтов из пакета-виновника: стандартом требовались лишь его заголовок IPv4 и 8 байт полезной нагрузки [§3.2.2 RFC 1122]. Когда в сети возникали сложные схемы многоуровневой инкапсуляции, например, с привлечением туннелей, информативные части пакета-виновника просто не помещались в сообщение.

Теперь же мы вправе — и обязаны — потребовать, чтобы всякое сообщение об ошибке ICMPv6 содержало как можно б*о*льшую часть пакета-виновника [§2.4(c) RFC 4443]. Очевидно, что это должно быть начало пакета [§2.1 RFC 4443], а не середина или конец, потому что именно начало пакета дает ключ к его содержимому.

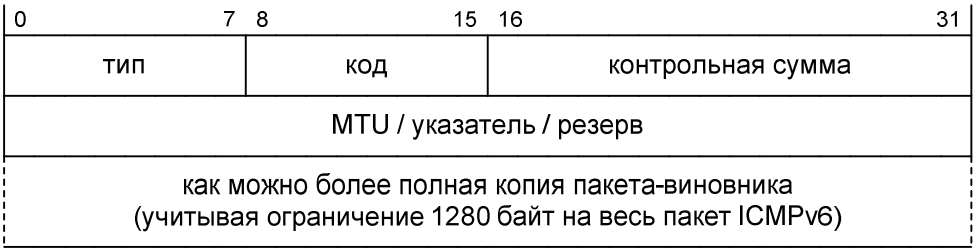
Подумайте, что можно включить в копию пакета-виновника, когда происходит тайм-аут сборки, а первый фрагмент так и не получен. (Вариант: образец нефрагментируемой части и заголовок фрагмента.)

Насколько велика может быть эта часть? Мы заметили в §3.3.4, что ICMPv6 поможет нам управлять фрагментацией. Но если извещения ICMPv6 сами будут подвержены фрагментации, то наша схема зациклится. Поэтому фрагментировать извещения ICMPv6, вообще говоря, нельзя. Следовательно, ограничением сверху здесь будет минимальный MTU IPv6, 1280 байт. Конечно, в этот MTU должен умещаться весь пакет ICMPv6, включая заголовок IPv6 и возможные заголовки расширения.

Нетрудно убедиться, что вместе эти два условия (как можно больше, но не больше чем) допускают только два случая:

- извещение содержит пакет-виновник целиком (в том виде как он был доступен извещающему узлу);
- извещение содержит начало пакета-виновника, а весь пакет ICMPv6 занимает 1280 байт.

В результате извещения ICMPv6 будут иметь единый формат, представленный на Фиг. 49.



Фиг. 49. Структура извещения об ошибке ICMPv6

Однако полнота сообщений об ошибках оборачивается другой проблемой. Теперь сообщения об ошибках имеют относительно большой размер, а значит, они смогут расходовать заметную часть ресурсов сети, если частота ошибок будет высокой, например, вследствие атаки. Хуже того, возможны косвенные атаки, когда злоумышленник «бомбардирует» узел А сбойными пакетами от имени источника Б, вызывая шквал ICMPv6 от А к Б. Предотвратить эту проблему можно, если все узлы IPv6 ограничат частоту, с которой они высылают сообщения об ошибках. Пусть это ограничение будет обязательным [§2.4(f) RFC 4443], хотя конкретный алгоритм мы оставим на усмотрение реализаций.

Пример такого алгоритма в общих чертах описан в [§2.4(f) RFC 4443].

Продолжая наш ход мысли, мы приходим к общему вопросу [§2.4(e) RFC 4443]: в каких случаях узел IPv6 обязан воздержаться от уведомления об ошибке? Прежде всего, чтобы избежать бесконечной «игры в пинг-понг», извещение об ошибке ни в коем случае нельзя высылать в ответ на другое извещение об ошибке. Возможно, в будущем у нас возникнут и справочные типы сообщений ICMPv6, которые могут спровоцировать «пинг-понг», так что за этим моментом надо внимательно следить.

Например, в §5.2 нам встретится сообщение типа «переадресовка», которое высылается в ответ на другой пакет, хотя и не является извещением об ошибке.

Также не следует отвечать ошибкой на пакет, адрес источника в котором не индивидуальный. Ведь извещение об ошибке уходит по адресу, который был источником пакета-виновника. Например, если пакет отправлен с группового адреса, то налицо явное нарушение протокола (см. §3.2); если это не результат сбоя, то попытка атаки «отказ в обслуживании». Помимо того, в некоторых вполне законных случаях — с ними мы встретимся ниже, это будут DAD в §5.4.1 и MLD в §6.4 — адрес источника в пакете может быть неопределенным, *::*. На такой пакет отвечать ошибкой тоже нельзя, потому что это означало бы указать неопределенный адрес *назначения* в пакете ICMPv6.

Также не следует в общем случае отвечать на групповые и широковещательные пакеты, поскольку иначе один ошибочный пакет вызовет целую бурю ответов ICMPv6. Чтобы определить, относится ли пакет к групповым или широковещательным, недостаточно проверить адрес источника в заголовке IPv6. Необходимо также убедиться, что пакет не был разослан группе или всем узлам на канальном уровне. К примеру, если пакет получен из канала *Ethernet*, то надо проверить адрес MAC назначения в заголовке кадра. Таким образом, для данной проверки нужна помощь со стороны канального уровня сетевого стека.

Например, в сетевом стеке *BSD Unix* канальный уровень помечает принятый кадр как групповой или широковещательный с помощью битовых флагов **M_MCAST** и **M_BCAST** [[73]], так что сетевому уровню достаточно проверить эти флаги — ему не нужно изучать заголовок кадра самостоятельно.

Из последнего правила есть всего пара исключений [§2.4(e) RFC 4443]. Во-первых, если код нераспознанной опции IPv6 содержит двоичный префикс `10`, то

[73] Gary R. Wright, W. Richard Stevens. TCP/IP Illustrated, Volume 2: The Implementation. Addison Wesley, 1995. p. 39.

сообщение об ошибке требуется независимо от сетевого и канального адресов назначения пакета — об этом случае мы уже сказали в §3.3.2 настоящего курса. Во-вторых, ошибка «пакет слишком велик» высылается независимо от адресов назначения пакета [§3.2 RFC 4443], чтобы процедуру PMTUD можно было применять и к групповому трафику.

Когда маршрутизатор отвечает извещением ICMPv6, его адрес назначения очевиден: это адрес источника из пакета-виновника. А какой надо выбрать адрес источника для самого извещения? В общем случае это нетривиальный вопрос [§2.2(b) RFC 4443]; ответ на него дает процедура DAS, с которой мы познакомимся в §6.6. Главная сложность здесь в том, чтобы зона адреса источника позволяла коммуникацию с адресом назначения. Этой особенности не было в IPv4, где маршрутизатор просто использовал адрес выходного интерфейса извещения [§4.3.2.4 RFC 1812]. Также сравните это со случаем высылающего ICMPv6 хоста: ему достаточно поменять местами адреса источника и назначения, при условии что пакет-виновник был адресован индивидуально хосту, а не группе [§2.2(a) RFC 4443].

5. Протокол розыска соседей

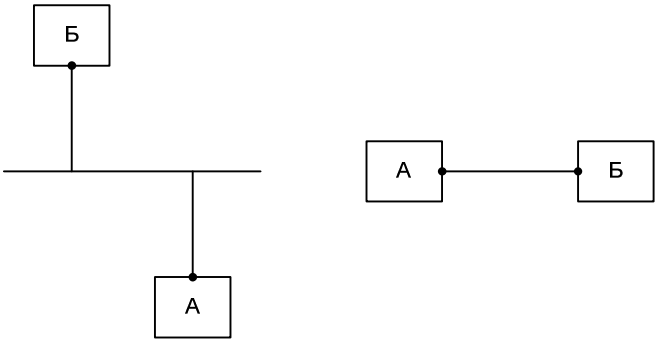
5.1. Розыск соседей и разрешение адресов

В нашей модели компьютерных сетей пакеты не перелетают с узла на узел волшебным образом, а путешествуют по каналам. Пока мы работаем на уровне IP, непосредственная передача пакета напрямую, «из рук в руки», возможна только узлу, который подключен к тому же каналу, что и наш локальный узел.

Нам удобнее думать о сетевом взаимодействии узлов, поставив себя на место одного из них и назвав его локальным узлом. Тогда все прочие узлы — удаленные.

Сам процесс передачи пакета по каналу может быть сколь угодно сложным и даже рекурсивно вызывать уровень IP, как это происходит в туннелях. Однако благодаря сознательному и целенаправленному разделению уровней в сетевой архитектуре это происходит прозрачно и выглядит для уровня-потребителя как один элементарный акт.

Чтобы нам вскоре не пришлось опровергать наши собственные утверждения, мы сразу же должны оговорить, что подключение узлов к общему каналу — это необходимое, но еще недостаточное условие для прямой передачи пакета между ними. Дело в том, что существуют «неклассические» каналы, в которых не все пары узлов могут обмениваться пакетами напрямую. Подробному обсуждению таких каналов в контексте IPv6 мы посвятим §5.2, а пока что давайте предположим, что наш локальный узел А априори знает, что он может передать пакет узлу Б напрямую, используя определенный канал К. Если это так, то узлы А и Б — **соседи** (*neighbors*) по каналу К. Пока наше воображение еще не смущено «неклассическими» каналами, мы можем позволить себе упрощенную картину, где соседи А и Б соединены каналом «точка-точка» — например, старым добрым PPP — или же широковещательным каналом — скажем, привычной локальной сетью *Ethernet*. Эти хрестоматийные случаи иллюстрирует Фиг. 50.



Фиг. 50. Узлы-соседи по широковещательному каналу (слева) и по каналу «точка-точка» (справа)

Тем не менее, давайте с самого начала иметь в виду, что, с математической точки зрения, отношение соседства коммутативно, но не транзитивно: если узел Б — сосед узла А, то узел А — сосед узла Б; но если узлы А и Б соседи и узлы Б и В соседи, то это еще не значит, что узлы А и В тоже соседи.

> Коммутативность соседства предполагает, что канал двунаправленный. Само понятие соседства нам нужно для разработки механизмов, ориентированных на двунаправленные каналы, так что однонаправленные каналы мы рассматривать не станем.

> Еще одно свойство соседства, кажущееся самоочевидным, — это его рефлексивность: всякий узел является собственным соседом и не нуждается в маршрутизаторе, чтобы разговаривать сам с собой. Но в действительности оно не возникает из воздуха, а реализовано с помощью сетевого интерфейса типа «петля» и связанного с ним виртуального канала из одного узла, на котором и возникает этот вырожденный случай соседства. Когда эта схема дает сбой, возникают артефакты, такие как разговор узла PPP с самим собой через удаленного соседа.

Теперь подумаем, как локальный узел А обращается к соседу Б. Использует ли он для этого титул, кличку или радиопозывные? Конечно же, нет. Вместо этого узел А применяет адрес IP, присвоенный интерфейсу узла Б. Если у узла Б несколько интерфейсов, то мы имеем в виду, конечно же, один из подключенных к общему каналу К.

Таким образом, узел IP, намереваясь передать индивидуальный пакет, вначале знает только адрес IP удаленного соседа. Так как сосед может быть не только адресатом пакета, но и маршрутизатором, его называют «следующий шаг» (*next hop*), тем самым подчеркивая, что он может быть не последним на пути пакета.

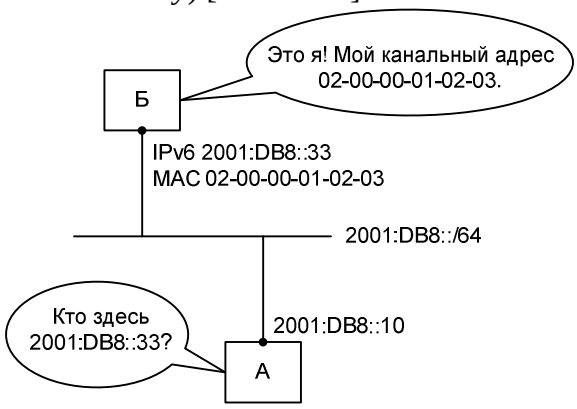
> Следующий шаг пакета от создавшего его хоста иногда называют «первый шаг» (*first hop*). На самом деле, эта деталь редко имеет значение, так что первый шаг пакета — всего лишь частный случай следующего шага.

Однако, чтобы на самом деле передать пакет дальше, уровень IP должен обратиться к канальному уровню, а тот использует только канальную адресацию и ожидает канального адреса соседа, будь то в явном или неявном виде. Поэтому локальный узел должен найти среди всех соседей по данному каналу именно того, кто обладает адресом следующего шага, и узнать его канальный адрес. Эта процедура известна нам как разрешение адресов (*address resolution*).

Если канал — типа «точка-точка», то это тривиальная задача, поскольку явный адрес IP и неявный канальный адрес единственного соседа заранее известны.

> Напомним себе и читателям, что в определенных условиях бывает достаточно неявной адресации. Так, в канале «точка-точка» явная канальная адресация излишня, потому что любая передача в этот канал будет принята ровно одним узлом, которому она и предназначалась. Можно сказать, что адрес соседа по каналу «точка-точка» звучит как «вон тот узел на другом конце провода». Поэтому, например, присваивать адреса MAC интерфейсам на концах канала «точка-точка» было бы избыточным, хотя на это нет принципиального запрета.

Если же канал многоадресный, то задача существенно затрудняется; ее решение зависит от свойств этого канала. В худшем случае, когда нет никаких вспомогательных механизмов, поможет только заданная вручную таблица соответствия между адресами сетевого уровня и канальными адресами. Но, к счастью, такой случай практически не встречается, а зато мы довольно часто сталкиваемся с каналами, где есть возможность обратиться сразу ко всем соседям и спросить, кому принадлежит искомый сетевой адрес. Эта идея, представленная на Фиг. 51, открывает нам путь к автоматизированному **розыску соседей**[74] (*neighbor discovery*) [RFC 4861].



Фиг. 51. Идея автоматического розыска соседей

> Этим свойством, когда узел может одновременно обратиться ко всем соседям, обладают не только широковещательные каналы, такие как *Ethernet*. Скажем, канал «точка-точка» тоже поддерживает широковещательную и групповую адресацию, поскольку сосед заведомо один. Что же касается широковещательного канала, то он дает более сильные гарантии: все подключенные к нему узлы, независимо от их числа, — попарно соседи, и вдобавок между ними возможно широковещание.

Хотя на первый взгляд может показаться, что разрешение адресов и розыск соседей — одна и та же задача, на самом деле они перекрываются, но не совпадают. Розыск соседей, в общем случае, можно вести по любым критериям, а не только по сетевому адресу. Например, узел может спросить у своих соседей: «Эй, кто из вас — маршрутизатор по умолчанию?» — или: «У кого это из вас жужжит вентилятор?» С другой стороны, разрешение адресов не всегда требует розыска соседей. Так, мы обсудили в §4.1.1, что разрешение групповых адресов IPv4 и IPv6 в групповые же адреса MAC происходит локально, простой подстановкой битов. Розыск не потребуется и тогда, когда требуемые сведения заранее внесены в локальную конфигурацию — то, что мы знали в IPv4 как статический ARP (*static ARP*).

> История вычислительных сетей знает и другие примеры локального разрешения адресов. Так, в адресе IPX идентификатор интерфейса всегда совпадал с его канальным адресом MAC, так что найти канальный адрес соседа можно было, опустив те старшие биты адреса IPX, где находился префикс подсети.

Это логическое разделение полезно иметь в виду, чтобы наше решение не вышло узким и ограниченным. Тем не менее, первой прикладной задачей у нас стоит именно разрешение индивидуальных адресов IPv6 в канальные адреса, когда узел А спрашивает: «Дорогие соседи, кому из вас принадлежит сетевой адрес Б?» — и получает в ответ канальный адрес искомого соседа, если тот на месте.

> Нетерпеливые читатели могут найти полный список функций, которые выполняет розыск соседей IPv6, в [§3 RFC 4861]. Но не стоит торопиться, ибо мы придем к этим функциям естественным путем.

[74] Можно встретить термин-синоним «обнаружение соседей».

Для начала мы рассмотрим уже готовое решение из нашего арсенала: ARP. Этот протокол достаточно гибок, чтобы работать с разными типами канальных и сетевых адресов, так что мы могли бы легко приспособить его и для нужд IPv6. Тем не менее, у нас есть повод, чтобы отказаться от ARP и создать новый механизм розыска соседей IPv6. В вину ARP мы поставим его… широковещание. Как так? Ведь мы сами только что сказали, что широковещание открывает нам светлый путь к розыску соседей! Не подлежит сомнению, что это была плодотворная идея. Но, чтобы перейти «от идеи к модели», нам надо учесть современные возможности каналов. В самом начале они поддерживали только два режима передачи, широковещательный и индивидуальный; однако с тех пор появился и прочно утвердился еще один режим, групповой.

В чем широковещательный режим проигрывает групповому? Прежде всего, у широковещания нулевая избирательность. Представим себе канал в смешанной сети, где одни узлы поддерживают только IPv4, другие — только IPv6, третьи — только ISO и т.д. Всякий раз, когда узел IPv4 посылает в такой канал запрос ARP, его принимают, анализируют и только затем отбрасывают все те узлы, которым вообще нет дела до IPv4; их вычислительные ресурсы расходуются при этом совершенно впустую, а в большой сети непроизводительный расход «виртуальной энергии» достигает заметной величины.

> Ходит легенда о больших плоских ЛВС, настолько перегруженных широковещательным трафиком, что только самый современный и мощный компьютер может справиться с его приемом. Если же компьютер слабоват, то он практически повиснет, пытаясь принять, чтобы тут же отбросить, бесконечный поток не интересных ему широковещательных кадров.
>
> Кстати, потребление электроэнергии компонентами и узлами сети тоже зависит от того, насколько оптимизировано в ней распространение трафика.

Это положение можно было бы значительно улучшить, если бы розыск соседей перешел на групповое вещание. В простом канале типа «общая шина» нежелательный групповой трафик отфильтруют сетевые адаптеры, а в современной коммутируемой ЛВС он даже не попадет в те сегменты, где его никто не ожидает. Вот нам задача номер один.

> Например, коммутируемая ЛВС Ethernet сможет оптимизировать групповой трафик, если в ней применяется «подслушивание IGMP/MLD» (*IGMP/MLD snooping*) [RFC 4541] (см. §6.4) или специализированный протокол GMRP [IEEE 802.1ak-2007].

В IPv4 существовала группа «все узлы данной подсети», *224.0.0.1*, а ей отвечал определенный групповой канальный адрес. Например, в *Ethernet* это был адрес MAC *01-00-5E-00-00-01*. Но, увы, построить на этом *стандартный* механизм розыска соседей было нельзя, потому что поддержка группового вещания в IPv4 так и осталась необязательной [§3.3.7 RFC 1122].

Теперь же, в IPv6, групповое вещание — обязательная возможность (см. §2.8), и мы могли бы основать розыск соседей на групповом адресе «все узлы канала», *FF02::1*. Это сразу ограничило бы круг вовлеченных в него узлов только теми, что поддерживают IPv6. Но не можем ли мы придумать решение еще лучше?

Подсказкой нам послужит внушительная длина адреса IPv6. А что если мы закодируем в групповом адресе самую характерную часть искомого индивидуального адреса? Положим, канал объединяет какое-то число узлов IPv6. Их интерфейсам, смотрящим в этот канал, назначена одна и та же подсеть, но идентификатор интерфейса у каждого свой. Кроме того, мы заранее знаем, что длина этого идентификатора 64 бита. Значит, мы могли бы перенести их во внутриканальный групповой адрес с определенным префиксом, назначенным для этой цели. Допустим, если бы это был префикс *FF02::/64*, то из индивидуального адреса *2001:DB8:1:2:3:4:5:6* (*И*) мы получили бы групповой адрес *FF02::3:4:5:6* (*Г*). Теперь узел Б, владелец адреса *И*, должен вступить в группу *Г* и принимать запросы о розыске соседей. Его сосед А, разыскивающий адрес *И*, пошлет

запрос по адресу *Г* и прицельно попадет запросом в узел Б, потому что в данной подсети ни у кого больше нет такого идентификатора интерфейса.

На словах звучит эффектно; а что произойдет в действительности? Прежде всего, узел А должен преобразовать адрес *Г* в групповой канальный адрес. Эта процедура не представляет сложности, потому что она выполняется локально (см. §4.1.1), однако часть битов идентификатора группы при этом, как правило, теряется. Так, например, при инкапсуляции *Ethernet* из группового адреса IPv6 в групповой адрес MAC переходят только младшие 32 бита. Кроме того, каждая группа занимает какое-то количество ресурсов в интеллектуальной ЛВС, оптимизирующей групповой трафик. Говоря проще, коммутаторам такой ЛВС приходится следить за «географией» каждой группы. Поэтому давайте ограничим общее число групп, которые могут возникнуть на одном канале при розыске соседей. Для этого пусть в групповой адрес *Г* попадают только избранные биты адреса *И*.

Но сколько и каких битов искомого адреса будет лучше всего взять? Мы уже выяснили, что они должны принадлежать идентификатору интерфейса. Когда администратор сети назначает эти идентификаторы вручную, то, скорее всего, он будет выбирать небольшие числа: 1, 2, 3… Если же идентификатор создается из адреса MAC-48 по рецепту §2.7, то мы должны учесть структуру этого адреса (см. Фиг. 15). Старшие 24 бита в нем — это OUI. Если, например, организация закупает для своей сети оборудование избранного производителя, то поле OUI в его адресах MAC окажется одинаковым (или будет принимать ограниченный набор значений). В то же время, младшие 24 бита в адресе MAC (добавочный идентификатор) носят куда более случайный характер, потому что в пределах одного OUI они уникальны, а каждый производитель назначает их независимо от своих конкурентов. Наконец, если у интерфейса есть адрес EUI-64, то добавочный идентификатор занимает в нем 40 бит. Их распределение зависит от политики производителя; в частности, при последовательном назначении старшие биты могут совпадать в пределах одной партии оборудования, а младшие будут разными. Таким образом, младшие 24 бита искомого адреса — это хороший выбор при разных политиках назначения идентификаторов интерфейсам. Тогда общее число групп, которые могут возникнуть в одном канале при розыске соседей, не превысит 2^{24}. Да, это тоже немало, но, тем не менее, существенно меньше, чем первоначальные 2^{63}.

> Попытайтесь вспомнить или выяснить, почему 2^{63}, а не 2^{64}. (Подсказка: в формате «модифицированный EUI-64» тоже есть бит **I/G**, он же **g**, а розыск ведется только по индивидуальному адресу.)

Итак, мы вполне обоснованно решили, что в особый групповой адрес *Г* попадают только 24 младших бита искомого адреса *И*. Соответственно, заранее известный групповой префикс должен быть длиной 104 бита. На эту роль назначен общепринятый префикс *FF02::1:FF00:0/104* [§2.7.1 RFC 4291]. Например, из нашего адреса *И*, то есть *2001:DB8:1:2:3:4:5:6*, получится такой адрес *Г*: *FF02::1:FF05:6*. Общепринятый термин для внутриканального группового адреса, составленного по этому рецепту, — **групповой адрес искомого узла** (*solicited-node multicast address*).

> Таково общепринятое английское название, тогда как русскоязычная терминология IPv6 еще только формируется. Обнаруженный нами у других авторов [75] вариант «адрес активного узла» кажется нам не совсем удачным, так как он плохо передает смысл данного понятия.

Теперь мы знаем, по какому адресу надо слать запрос о розыске соседа. Это адрес группы искомого узла, объединяющей все интерфейсы канала с индивидуальными адресами, у которых младшие 24 бита такие же, как в искомом адресе. Хотя в такую

[75] Ю.А. Семенов. Протоколы и алгоритмы маршрутизации в Интернет. Лекция 1. http://www.intuit.ru/department/network/pami/1/6.html

группу могут попасть несколько интерфейсов с разными адресами, розыск соседей становится значительно прицельнее, нежели при широковещании или рассылке по групповому адресу «все узлы». Конечно, теперь всякий узел IPv6, назначая своему интерфейсу индивидуальный адрес, обязан вступить на этом интерфейсе в соответствующую группу искомого узла, чтобы принимать запросы от соседей. Когда же адрес удаляется, узел должен выйти из этой группы, если только на интерфейсе не остается других адресов с теми же 24-мя младшими битами. На этом наша первая задача решена.

> На первый взгляд, число групп искомого узла, в которые узлу надо вступить на данном интерфейсе, равно числу индивидуальных адресов на этом интерфейсе. Но чуть более внимательный анализ обнаруживает известное нам свойство: разные индивидуальные адреса отображаются в один и тот же групповой адрес искомого узла, если в них совпадают 24 младших бита. Таким образом, аккуратный выбор идентификаторов интерфейса может сократить число групп искомого узла, которые придется слушать. Простейший подход сводится к тому, чтобы все идентификаторы интерфейса сделать одинаковыми, например, основанными на модифицированном EUI-64 этого интерфейса. Тогда адреса IPv6 этого интерфейса будут отличаться только префиксами подсети, а их группы искомого узла заведомо совпадут. Почему число таких групп стоит ограничивать? Во-первых, как мы сказали на с. 105, группы расходуют ресурсы сети. А во-вторых, современные сетевые адаптеры могут сами фильтровать групповой трафик, чтобы освободить шину и центральный процессор, но длина аппаратного фильтра в них ограничена. Когда узел слушает много разных групп и длина такого фильтра оказывается превышена, адаптер вынужден принимать весь групповой трафик в надежде, что центральный процессор разберется.
>
> Группа искомого узла — это лишь частный случай применения группового вещания IPv6 для направленного розыска сетевых ресурсов. В общем случае, узел, обладающий ресурсом, вычисляет групповой адрес как функцию от имени ресурса и вступает в эту группу, а узел-искатель, зная имя ресурса, проводит то же самое вычисление и направляет по полученному групповому адресу запрос. К примеру, так можно разыскивать хост по его символьному имени: от имени хоста в заданной кодировке вычисляют хэш-функцию подходящей длины и добавляют к ее значению общепринятый групповой префикс. На этой идее уже основан один экспериментальный протокол для запроса информации об узлах IPv6 [RFC 4620].

Следующий вопрос заключается в том, как будет устроен протокол розыска соседей. Пока что мы не определились даже с его уровнем в стеке. Привлечение групповых адресов сетевого уровня означает, что новый протокол мы должны разместить над IPv6, а не рядом с ним, в отличие от IPv4 и ARP. По сути, розыск соседей — это один из аспектов управления IPv6, так что давайте поступим просто: пусть розыск соседей станет подмножеством ICMPv6. Это подмножество кратко обозначают **ND** (от *neighbor discovery*) [RFC 4861].

> Как следствие, сообщения ND будут инкапсулированы в пакеты IPv6, а тогда на них смогут распространяться пакетные фильтры и прочие правила сетевой политики. Поэтому придется быть вдвойне осторожным, чтобы по ошибке не блокировать легитимный трафик ND и тем самым не «сломать» механизм разрешения адресов — ведь от его сбоя пострадает передача всего индивидуального трафика!

Такая позиция ND в стеке протоколов позволит применять его на любых каналах, способных инкапсулировать IPv6. Хотя сам по себе розыск соседей необходим далеко не всем типам каналов, например, он явно избыточен на каналах «точка-точка», мы создадим механизмы на основе протокола ND, которые будут уместны независимо от типа канала. Поэтому сразу выдвинем вот какое положение: ND будет работоспособен на всех каналах IPv6, которые поддерживают групповое вещание [§3.2 RFC 4861].

Означает ли наш выбор, что сообщения ND смогут пересекать границы каналов? Ведь в общем случае пакеты ICMPv6 подлежат маршрутизации наравне с другими пакетами IPv6. Между тем, из самого определения соседства следует, что пакеты ND должны распространяться только в пределах одного канала. Опасность здесь не только в утечке ND за границы канала, но и в проникновении фальшивых сообщений ND из-за этих границ.

Сможет ли адресат пакета ND узнать, что источник пакета — не его сосед? Если такой адресат не доверяет никому, то задача без дополнительных механизмов неразрешима. Но если адресат доверяет маршрутизаторам данного канала, то он может учесть, что всякий законопослушный маршрутизатор уменьшает значение «предельное число шагов» в заголовке IPv6 транзитного пакета. Сами по себе эти сведения еще ничем не помогают, так как адресат не знает, каким было начальное значение этого поля. Можем ли мы просто зафиксировать это значение? Допустим, мы решили, что начальное значение «предельного числа шагов» в пакетах ND всегда будет X. Теперь адресат пакета сравнивает видимое им значение с X и, если они совпадают, заключает, что пакет пришел от соседа. Но злоумышленник может как-нибудь узнать число шагов H до атакуемого узла и установить поле «предельное число шагов» равным $Y = X + H - 1$, если $H = 1$ отвечает соседству. Как помешать злоумышленнику? Если школьная арифметика помогла ему, то она поможет и нам! По условию задачи, $H > 1$, так как злоумышленник — не сосед атакуемого узла. (Если бы он оказался соседом, узлу пришлось бы совсем несладко!) Значит, злоумышленнику придется увеличить значение «предельного числа шагов» относительно X: $Y > X$. Единственный случай, когда он не сможет этого сделать, — это если значение X уже будет максимально возможным, а именно 255.

> Этот трюк работает, только если маршрутизаторы правильно обрабатывают пакеты с нулевым значением «предельного числа шагов», как мы обсудили в §3.2. В противном случае декремент этого значения даст 255 согласно беззнаковой арифметике по модулю 256.
>
> Атаки на ND со стороны соседей нам еще предстоит рассмотреть.

Обнаруженный нами трюк способен защитить от дистанционных атак не только ND и не только в среде IPv6. Ведь все, что ему нужно, — это поле «предельное число шагов» или «TTL» в сетевом заголовке. За свою универсальность этот трюк получил пышноватое название «обобщенный механизм безопасности на основе TTL» (*Generalized TTL Security Mechanism*), сокращенно **GTSM** [RFC 5082].

> Другой пример протокола, который может выиграть от применения GTSM, — это BGP. Исторически идея GTSM появилась именно в применении к BGP [§4 RFC 5082], а затем ее обобщили и на другие случаи.
>
> Конечно, GTSM не заменяет собой прочие механизмы сетевой безопасности, такие как фильтрация пакетов на границах канала. Просто любая хорошо построенная оборона обязана быть глубоко эшелонированной.

Итак, участники ND обязаны применять GTSM: источник пакета ND должен установить поле «предельное число шагов» равным 255, а его адресат обязан убедиться, что это значение сохранилось [§6.1, §7.1, §8.1 RFC 4861].

Теперь можно перейти к отдельным сообщениям ND. Чтобы провести разрешение сетевого адреса, достаточно двух видов сообщений, которые в первом приближении выглядят как запрос и ответ. Но, как мы помним, даже в ARP строгую последовательность «запрос-ответ» приходилось нарушать, когда узлу надо было послать «добровольный ARP» (*gratuitous ARP*), чтобы самому объявить соседям свой канальный адрес.

> В ARP не имело значения, представлял ли собой добровольный ARP запрос или ответ, потому что алгоритм получателя ARP [RFC 826] проверял код операции только *после* того, как обновлял локальный кэш.

Не Конфуций ли сказал: если название неправильное, то и слова не повинуются? Поэтому давайте для начала исправим терминологию. Сообщение, которое несет сведения о сетевом и канальном адресах узла, мы назовем **объявлением соседа** (*neighbor advertisement*), сокращенно **NA**. Узел вправе выслать это объявление добровольно или по запросу. Другие же узлы могут запросить такое объявление, когда им это нужно, послав **вызов соседа** (*neighbor solicitation*), сокращенно **NS**.

Снова представим себе, что узел А хочет передать пакет своему соседу Б, но не знает его канального адреса. Теперь у узла А есть простая методика:

1) составить групповой адрес искомого узла Г, взяв младшие 24 бита сетевого адреса Б и добавив их к общепринятому префиксу *FF02::1:FF00:0/104*;
2) составить вызов соседа NS (о его формате мы еще не говорили, так как не знаем всех требований к нему);
3) послать этот вызов по адресу Г;
4) ожидать объявления соседа NA от узла Б;
5) извлечь из принятого объявления NA канальный адрес Б;
6) направить по канальному адресу Б кадр с пакетом.

А что узел А должен делать после? Если он немедленно забудет канальный адрес Б, то указанную процедуру ему придется повторять на каждый пакет для соседа Б, а это будет совершенно неэффективно. С другой стороны, узел А не должен хранить канальный адрес Б в своей памяти вечно, потому что тот может впоследствии измениться, например, если в узле Б «на лету» сменят сетевой адаптер.

> Подумайте, какая из возникших проблем была бы самой острой, если бы процедуру розыска соседа пришлось бы выполнять для каждого исходящего пакета? (Авторский вариант: возникновение задержки, равной ПКО (*RTT*) данного канала.)

Нашего опыта достаточно, чтобы сказать: здесь нужен кэш. Действительно, кэш соответствия между сетевыми и канальными адресами уже был в ARP. Политика управления им была весьма простой [RFC 826]:

- новая запись создавалась по любому сообщению ARP от любого соседа, если сообщение было в наш адрес (поле ARP «сетевой адрес цели»);
- существующая запись обновлялась по любому сообщению ARP от этого соседа, даже если оно было адресовано кому-то еще;
- устаревшая запись удалялась по тайм-ауту.

В отсутствие дополнительных сообщений ARP, запись неизбежно удалялась по тайм-ауту, даже если тем временем протокол более высокого уровня, например, TCP, успешно вел обмен данными с узлом Б, чем подтверждал актуальность данной записи. Хуже того, при активном обмене данными с узлом Б широковещательная процедура ARP немедленно повторялась после удаления его записи, создавая паразитную нагрузку на других соседей. Так было потому, что между вышестоящими уровнями стека и ARP отсутствовала обратная связь.

> Этот неоптимальный подход появился с одобрения [§2.3.2.1 RFC 1122] — см. п. 1. Хотя в том же разделе RFC были предложены и более эффективные приемы работы с кэшем ARP, п. 1 распространился благодаря сетевому стеку *BSD Unix* [76].

> Конечно же, уровни сетевого стека выше IP, например, транспортный уровень SCTP, TCP и UDP, обязаны быть только у хоста, тогда как маршрутизатору в теории достаточно продвигать транзитные пакеты IP.

[76] Gary R. Wright, W. Richard Stevens. TCP/IP Illustrated, Volume 2: The Implementation. Addison Wesley, 1995.

Мы обязательно примем меры, чтобы ND не унаследовал этот недостаток. Сам же кэш канальных адресов был важной и полезной деталью; он нам необходим и сейчас. Назовем мы его просто: **кэш соседей** (*neighbor cache*), сокращенно **NC**.

Работу над NC мы начнем именно с того, что постулируем связь между ND и вышележащими уровнями в сетевом стеке хоста IPv6. Эта связь носит локальный характер и не выходит за пределы одного узла, поэтому ее можно реализовать по-разному. В простейшем случае, это будет просто вызов функции ядра. Главное, что теперь более высокий уровень в стеке способен сообщить модулю ND: да, я уверен, что данный адресат доступен. Например, если этим уровнем будет TCP, то он сможет утверждать доступность адресата, если от того поступят новые данные или новые квитанции (не дубликаты).

> На первый взгляд, TCP делает необоснованный вывод; ведь прием пакетов вовсе не означает, что мы знаем верный канальный адрес их передатчика. Однако TCP — это протокол с обоюдным квитированием, и приход новых данных или квитанций возможен, только если удаленная сторона успешно получает наши квитанции или данные, соответственно.
>
> Подобную обратную связь можно было реализовать и в ARP. П. 4 §2.3.2.1 RFC 1122 вполне допускает связь между вышележащими уровнями и модулем ARP, хотя и говорит о передаче отрицательных, а не положительных сигналов: протокол более высокого уровня сообщает модулю ARP, что данный узел недоступен, а модуль ARP удаляет запись об этом узле.
>
> Подробному обсуждению тонкостей обратной связи между вышестоящими протоколами и ND посвящено [Приложение E.1 RFC 4861].

Однако у нас есть одно затруднение: вышестоящий уровень подтверждает доступность адресата, а не соседа. Это вполне закономерно: ведь выбор соседа происходит на уровне IP, в процессе маршрутизации исходящего пакета. Поэтому мы должны обеспечить логический переход от адресата к соседу.

> Здесь под маршрутизацией мы понимаем выбор следующего шага пакета, а не продвижение транзитных пакетов (*forwarding*). В этом смысле маршрутизировать надо все исходящие пакеты, включая созданные на данном узле.

Если адресат окажется соседом локального узла, то модуль ND может сразу же обновить его запись NC. Но если адресат доступен только косвенно, через маршрутизатор, то перед модулем ND встанет такой вопрос: через какого соседа мы последний раз слали пакет данному адресату? Ответить на него поможет еще один кэш, который станет хранить самые свежие пары «адресат-сосед». Он так и называется: **кэш адресатов** (*destination cache*), кратко **DC**.

> Также запись в кэше адресатов может хранить дополнительные сведения о данном адресате, например, текущую оценку величины PMTU. К этой идее мы придем естественным образом в §6.5.
>
> Хотя модуль ND маршрутизатора не получает сигналов от вышележащих уровней стека по причине отсутствия оных, кэш адресатов может пригодиться и маршрутизатору. Например, при распределении трафика по равноценным путям (*equal-cost multipath*, **ECMP**) важно, чтобы все пакеты одного прикладного потока, такого как сеанс TCP, направлялись по одному пути. В противном случае в сети могут возникнуть трудно диагностируемые проблемы [RFC 2991]. Самая распространенная из них — это переупорядочение пакетов из-за того, что задержки разных путей всегда немного отличаются. Конечно, переупорядочение не фатально для TCP/IP, но, происходя постоянно, оно значительно понижает эффективность транспортных протоколов. Например, для механизмов управления перегрузкой TCP приход пакета вне очереди — это признак потери предыдущего пакета, а значит, перегрузки в сети [§3.2 RFC 5681]. Еще одна серьезная проблема — это видимые флуктуации PMTU, если PMTU вдоль разных путей отличается. Кэш

адресатов поможет направить все потоки одного адресата по одному и тому же пути.

Чтобы нам стало понятнее, как возникшие структуры данных связаны между собой, приведем иллюстрацию — Фиг. 52. В качестве отправной точки мы включили в эту схему хорошо знакомую нам таблицу маршрутов. Однако давайте иметь в виду, что в §5.2 мы пересмотрим эталонную модель хоста IPv6 и придем к неожиданному выводу, что таблица маршрутов для него необязательна. С другой стороны, программная реализация может совместить все три логических структуры в одной [§5.1 RFC 4861], например, в базисном дереве (*radix tree*) по образу и подобию таблицы маршрутов *BSD*.

Фиг. 52. Связь между структурами данных в памяти узла: таблица маршрутов, DC и NC

Мы еще обсудим кэш адресатов в §5.2, а сейчас перейдем к структуре и жизненному циклу записи NC. Сама роль такой записи требует, чтобы ключом в ней выступил адрес IPv6 соседа; то есть в NC не может быть двух записей об одном и том же соседском адресе. В то же время, механизм ND даже не пытается выяснить, принадлежат ли разные адреса IPv6 одному соседскому интерфейсу или узлу, и поэтому, говоря «запись NC о соседе Б», мы подразумеваем запись об индивидуальном адресе Б, ставя условный знак равенства между узлом-соседом и его индивидуальным адресом. Это одно из допущений модели ND, которое, не будучи универсальной истиной, вполне работает в частном случае разрешения индивидуальных адресов IP в канальные адреса.

Говоря об использовании множественных подключений в §6.8, мы встретим случай, в котором принципиально важно учитывать, какие адреса IPv6 принадлежат одному и тому же удаленному узлу. Однако тот случай не будет иметь прямого отношения к розыску соседей.

В своем исходном состоянии запись NC о соседе Б просто не существует. Ее небытие продолжается до тех пор, пока у локального узла А впервые не возникнет необходимость передать пакет соседу Б.

Обратите внимание, как это отличается от поведения ARP. В кэше ARP запись создавалась по приходу любого сообщения ARP, чей сетевой адрес цели (*target protocol address*) принадлежал локальному узлу. Поэтому в кэше ARP могли возникать ненужные записи. Напротив, входящее объявление NA *никогда* не служит поводом создать новую запись NC.

Тогда узел А создает запись, но канальный адрес в ней еще неизвестен. Это состояние **НЕПОЛНАЯ** (*INCOMPLETE*). Чтобы заполнить запись, узел А должен получить от Б объявление соседа. О своем желании он сообщает, направляя вызов соседа на групповой адрес искомого узла, полученный из индивидуального адреса «Б». Дальше узлу А остается только ожидать. Если ожидание затянется,[77] то надо повторить групповой вызов еще несколько раз[78] — ведь пакеты и кадры иногда все-таки теряются. Если уж и это не поможет, то запись NC останется только удалить, исходный пакет — отбросить, а его источник — известить по ICMPv6 (в отсутствие противопоказаний — см. §4.3).

[77] Параметр **RetransTimer**, по умолчанию **RETRANS_TIMER**, 1 с [§6.3.2 RFC 4861].
[78] Дважды, то есть всего до 3 вызовов — параметр **MAX_MULTICAST_SOLICIT** [§10 RFC 4861].

Для этого служит сообщение ICMPv6 тип 1, код 3 [§3.1 RFC 4443].

У записей в кэше ARP тоже было неполное состояние — по крайней мере, начиная с сетевого стека BSD [[79]]. В ARP это состояние не было совершенно необходимым, так как запись все равно возникла бы снова, даже если бы ответ пришел после исчезновения неполной записи по тайм-ауту. Стек BSD использовал неполные записи в собственных целях, например, чтобы хранить ссылки на исходные пакеты, ожидающие отправки.

Но вот, допустим, от узла Б своевременно поступило объявление соседа. Значит ли это, что сосед Б доступен и его канальный адрес можно с уверенностью использовать? Строго говоря, нет. Ведь у нас нет гарантии, что это объявление выслано в ответ на вызов узла А. Может быть, вызовы не доходят до Б из-за одностороннего сбоя в канале, но Б сам почему-то решил выслать объявление NA (ему это не возбраняется). Надежно различить эти два случая можно, если формат NA предусмотрит специальный флаг. Скажем, если он сброшен, то объявление добровольное, а если установлен — то по вызову. Обозначим этот флаг как **S** (*solicited*, по вызову).

По протоколу, узел Б устанавливает флаг **S**, *только* если сообщение NA адресовано персонально узлу А в ответ на запрос NS узла А. Именно эту информацию флаг **S** и доносит до узла А. Поэтому дополнительные проверки со стороны узла А, как то сверка адреса назначения NA, в этом случае избыточны.

Еще раз обратим внимание: хотя приход «непрошенного» объявления NA — это вполне допустимое событие, оно не приводит к созданию новой записи NC, а только влияет на уже существующую запись.

Теперь, в зависимости от состояния флага **S** в полученном NA, перед узлом А открываются два пути. Если флаг установлен, то канал между А и Б явно работает в двух направлениях, и записью NC можно смело пользоваться хотя бы некоторое время. В нее надо поместить канальный адрес Б, взятый из NA, и перевести запись в состояние **ДОСТУПНАЯ** (*REACHABLE*).

Какое время запись NC будет оставаться **ДОСТУПНОЙ** в отсутствие дополнительных сигналов? Чтобы избежать синхронизации соседей между собой, это время надо сделать разным на разных узлах, например, случайным.[80] Такая синхронизация нежелательна, потому что способна охватить многие узлы и привести к резким всплескам трафика [[81]]. Достаточно будет, если каждый узел выберет случайное значение из определенного диапазона[82] и будет использовать его в своих записях NC некоторое время, порядка нескольких часов, после чего выберет новое случайное значение и т.д. [§6.3.2 RFC 4861].

Если же флаг **S** сброшен, то о доступности узла Б по-прежнему ничего не известно, но узел А все-таки узнал канальный адрес Б и может рискнуть передать по нему ожидающий пакет в канальном обрамлении. В этом случае узел А тоже сохраняет канальный адрес Б в записи NC, чтобы информация не пропала зря, но переводит запись в состояние **ПРОСРОЧЕННАЯ** (*STALE*), на котором мы остановимся подробнее буквально через несколько абзацев. В двух словах, сведения из **ПРОСРОЧЕННОЙ** записи имеют хороший шанс оказаться актуальными, но, тем не менее, нуждаются в подтверждении, чтобы запись стала **ДОСТУПНОЙ**.

---

[79] G. R. Wright, W. R. Stevens. TCP/IP Illustrated, Volume 2: The Implementation. Addison Wesley, 1995. §21, “ARP: Address Resolution Protocol”.

[80] Параметр **ReachableTime**, случайный [§6.3.2 RFC 4861].

[81] S. Floyd, V. Jacobson, "The Synchronization of Periodic Routing Messages", IEEE/ACM Transactions on Networking, April 1994. <http://ee.lbl.gov/papers/sync_94.pdf>

[82] От **MIN_RANDOM_FACTOR** × **BaseReachableTime** до **MAX_RANDOM_FACTOR** × **BaseReachableTime**, по умолчанию от 0.5 × 30 до 1.5 × 30 с [§6.3.2 RFC 4861].

Все состояния, в которых запись заполнена (в ней указан не только сетевой, но и канальный адрес), мы будем собирательно называть **полными**.

ДОСТУПНАЯ запись освежается по сигналу от вышестоящих протоколов, или же дополнительными объявлениями NA с **S** = 1 — то есть сбрасывается тайм-аут записи. Но допустим, что такую запись давно[80] не освежали. Означает ли это, что сосед Б куда-то пропал? В аналогичных условиях ARP полагал именно так и удалял запись. Но на самом деле это может означать всего лишь, что текущий диалог завершился и соседи решили помолчать, занявшись другими делами. Поэтому узел А не удаляет запись NC о соседе Б, а просто переводит ее в состояние **ПРОСРОЧЕННАЯ** (*STALE*).

> Так как в ND активная **ДОСТУПНАЯ** запись постоянно освежается, ее тайм-аут может быть существенно меньше, чем знакомое нам время жизни записи в кэше ARP. Так что не стоит удивляться, что простаивающие **ДОСТУПНЫЕ** записи NC становятся **ПРОСРОЧЕННЫМИ** гораздо скорее, нежели предсказывает наш опыт IPv4.

ПРОСРОЧЕННАЯ запись хранит сведения, в которых узел А не уверен. Без проверки пользоваться ими нежелательно, и поэтому запись остается **ПРОСРОЧЕННОЙ** ровно до тех пор, пока к ней не будет обращений. Но, с другой стороны, и обязательного тайм-аута у **ПРОСРОЧЕННОЙ** записи нет, потому что она содержит полезные сведения.

Проходит время, и вот узел А опять готов передать новый пакет соседу Б, запись о котором **ПРОСРОЧЕННАЯ**. Весьма вероятно, что узлы А и Б находятся там же, где они были и раньше. Поэтому узел А шлет пакет по канальному адресу Б, взятому из записи. Он надеется, что вскоре вышестоящий протокол подаст сигнал о доступности соседа Б. Такое событие узел А отмечает, переводя запись в состояние **ЗАДЕРЖКА** (*DELAY*) и устанавливая ее таймер.[83]

А что если интервал этого таймера истек, а сигнала сверху по-прежнему нет? Даже тогда для записи в состоянии **ЗАДЕРЖКА** не все потеряно: ведь, возможно, переданный пакет не предполагал никакого отклика, или он потерялся где-то дальше в сети. Чтобы разрешить возникшую неоднозначность, узел А целенаправленно проверяет годность записи NC: он направляет вызов NS по *индивидуальному* адресу узла Б, используя канальный адрес из записи, и переводит запись в состояние **ИСПЫТАНИЕ** (*PROBE*). И, наконец, если даже после нескольких попыток[84] вызова и ожидания[77] сосед Б так и не ответил объявлением NA, то остается только удалить его запись из NC и тем самым начать розыск данного соседа с нуля в надежде обнаружить его новый канальный адрес.

> Современные реализации IPv4 тоже освежают записи в кэше ARP по истечении тайм-аута, индивидуально обращаясь к соседям. Например, так поступает ядро *Linux* [[85]]. Этот прием был описан в п. 2 [§2.3.2.1 RFC 1122] под названием **индивидуальный опрос** (*unicast poll*).

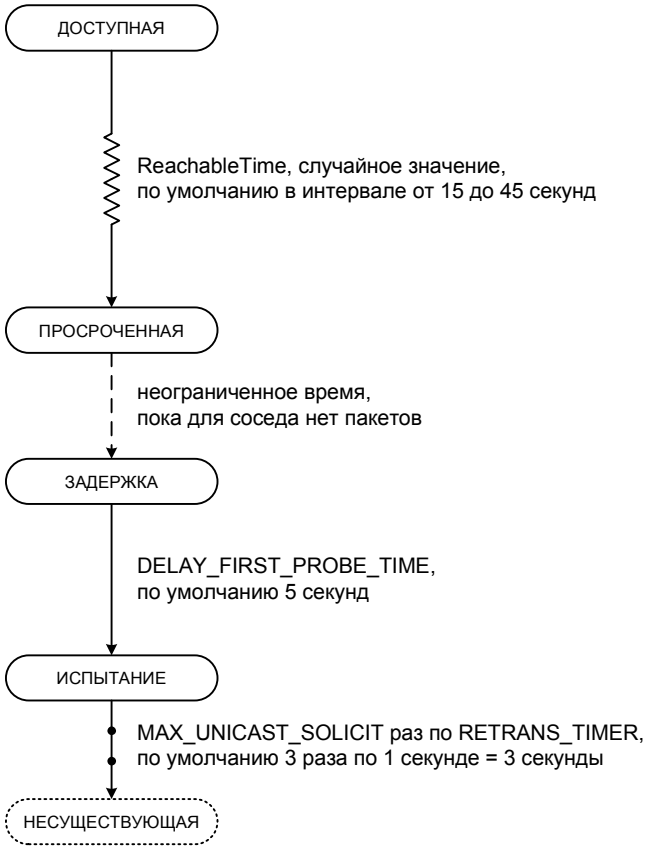
Окончательная хронология не обновляемой и потому обреченной исчезнуть записи NC показана на Фиг. 53.

---

[83] Параметр **DELAY_FIRST_PROBE_TIME**, 5 с [§10 RFC 4861].

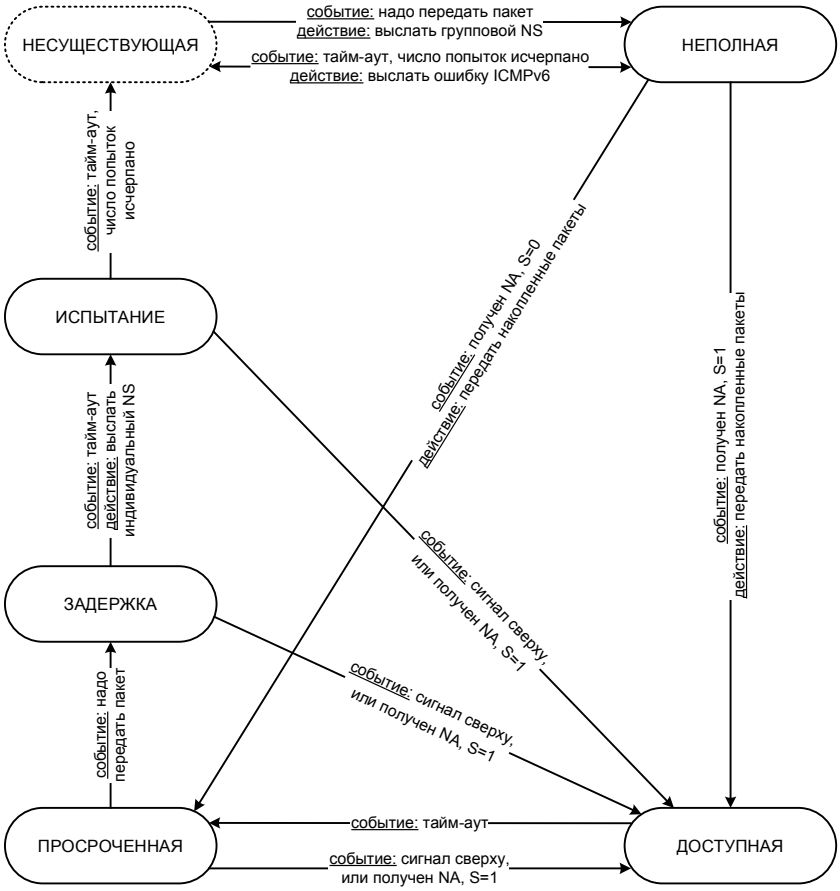
[84] Параметр **MAX_UNICAST_SOLICIT**, 3 раза [§10 RFC 4861].

[85] Klaus Wehrle, Frank Pählke, Hartmut Ritter, Daniel Müller, Marc Bechler. The Linux Networking Architecture: Design and Implementation of Network Protocols in the Linux Kernel. Prentice Hall, 2004. §15.3.



Фиг. 53. Наглядная хронология устаревающей записи NC

Пока запись NC полная, но проходит проверку, в очереди узла А могут возникнуть прикладные пакеты, требующие передачи соседу Б. Должен ли узел А приостановить их отправку до тех пор, пока он не убедится в достоверности записи? На самом деле, пауза здесь не требуется и может быть даже вредна, так что узел А продолжает передавать пакеты соседу Б, используя канальный адрес из записи. Благодаря этому, у вышестоящих протоколов тоже появляется возможность освежить запись. Сигнал вышестоящего протокола переводит полную запись из любого состояния в состояние **ДОСТУПНАЯ**. Очевидно, что на несуществующие и **НЕПОЛНЫЕ** записи сигнал не действует, даже если он вдруг поступит, поскольку в таких записях нет информации о канальном адресе соседа.



Фиг. 54. Упрощенная диаграмма состояний ND

Хотя начинали мы с простой экономии ресурсов канала и узлов-соседей, в результате у нас получился полнофункциональный механизм наблюдения за состоянием соседей, которым локальный узел передает пакеты. Этот механизм называют **выявление недоступности соседей** (*neighbor unreachability detection*), сокращенно **NUD** [§7.3 RFC 4861]. По сути, механизм NUD — это и есть конечный автомат ND, а информация о доступности соседей — один из основных продуктов его работы. На текущий момент у нас готов упрощенный вариант этого механизма, схематически показанный на Фиг. 54.

> Более полную диаграмму состояний ND можно найти, например, в [86], однако эта диаграмма покажется вам заметно доступнее и осмысленнее, если вы сначала завершите вместе с нами работу над текущим разделом.

Ценой определенных усилий мы победили в ND недостатки ARP и теперь можем спокойно взглянуть на плодотворные идеи старого протокола, чтобы не потерять их в ND.

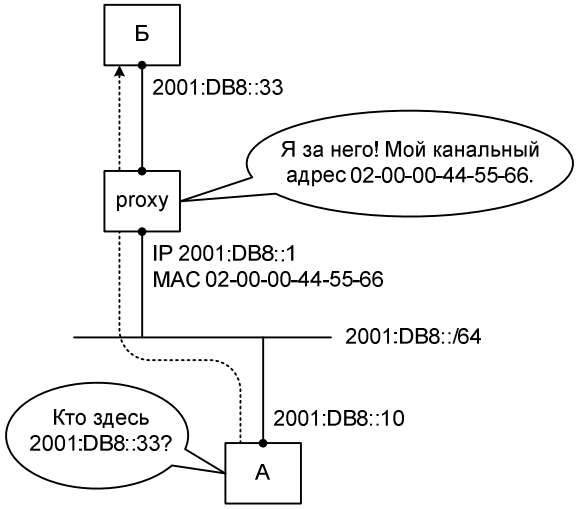
Первая идея состоит в том, что большинство запросов ARP, а теперь и вызовов NS открывает некий сетевой диалог. Иными словами, очень часто узел А проводит розыск соседа Б и передает ему пакет, а вскоре уже узел Б имеет что передать своему соседу А. Если А и Б — хосты, то это естественный стиль работы большинства прикладных протоколов. Если же один из них или оба — маршрутизаторы, то здесь проявляется не только двунаправленная работа прикладных протоколов, но и практическая симметрия маршрутов. Поэтому модуль ARP, получив запрос, предусмотрительно создавал запись о его источнике, если запрос касался локального узла: он предвидел, что скоро роли поменяются и источник запроса станет искомым соседом, получателем пакета IP.

В среде IPv6 намек на предстоящий диалог — это вызов NS. Если формат сообщения NS предусмотрит место для канального адреса источника (А), то получатель NS (Б) сможет сразу же создать о нем полную запись NC. По правилам NUD, эта запись —

[86] Qing Li, Tatuya Jinmei, Keiichi Shima. IPv6 Core Protocols Implementation. Morgan Kaufmann, 2007. P. 502.

ненадежная, потому что проверена только односторонняя проходимость канала: А→Б. Значит, такая запись заслуживает состояния **ПРОСРОЧЕННАЯ**. Это состояние записи вполне позволит сетевому диалогу начаться без задержки на еще один цикл ND между А и Б, когда узел Б станет разыскивать соседа А. Далее, если все будет в порядке, запись о соседе А перейдет в состояние **ДОСТУПНАЯ** по стандартной процедуре, не задерживая диалог.

Вторая идея заключается в том, что на розыск соседа не всегда откликается сам владелец искомого сетевого адреса. Искомый узел может находиться вне канала, а ответит за него маршрутизатор, у которого есть сведения о доступности искомого узла, как то маршрут к нему, более точный (*specific*), чем префикс подсети канала. Поскольку такой маршрутизатор отвечает от имени искомого узла и выступает посредником между искателем и искомым, то его обозначают популярным термином **proxy**. В частности, в ARP такой маршрутизатор назывался ARP proxy. Задача подобного proxy — привлечь к себе трафик для искомого узла, чтобы затем продвинуть его согласно таблице маршрутов, как показано на Фиг. 55. Поэтому proxy должен ответить, что искомому сетевому адресу отвечает его собственный канальный адрес. Особенность IPv6 здесь только в том, что ND proxy должен вступить в группу искомого узла, отвечающую адресу «Б», чтобы принимать вызовы NS к Б и отвечать на них объявлениями NA от имени Б.



Фиг. 55. Идея: ND proxy отвечает за другой узел

Если ND proxy отвечает от имени многих узлов, например, целой подсети, ему может пригодиться режим работы сетевого интерфейса, когда тот принимает все групповые кадры подряд (*allmulti*). Однако это может вызвать осложнения в коммутируемой ЛВС, использующей сообщения IGMP или MLD (§6.4) для изучения географии групп IP. В такой ЛВС трафик группы просто не дойдет до портов, из которых не поступало уведомления о вступлении в данную группу.

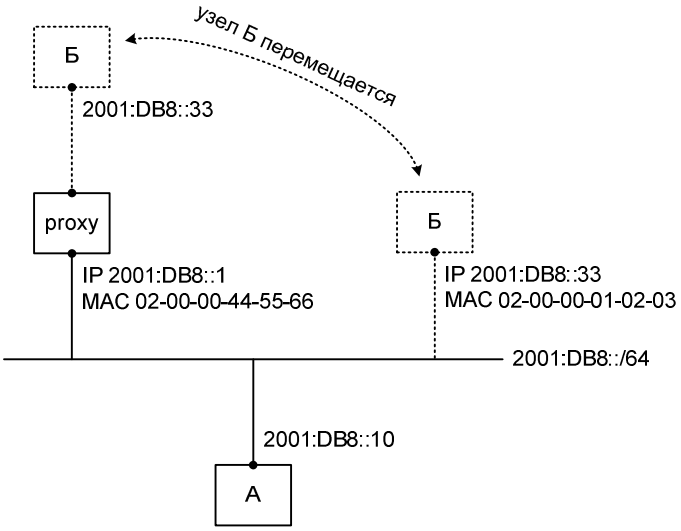
Функция ND proxy должна быть настраиваемой. Как минимум, ей требуется «выключатель», потому что по умолчанию она должна быть *выключена*.

А как быть, если искомый узел — мобильный и может находиться то по соседству, то вне данного канала, не меняя при этом своего адреса? Сохраним наши обозначения: пусть Б — это искомый узел, тогда как А — наш локальный узел, который хочет передать пакет непосредственно узлу Б, полагая при этом, что Б — его сосед. Этот сценарий представлен на Фиг. 56.

Хотя такая мобильность еще не позволяет узлу Б свободно перемещаться в пределах *Internet*, она отвечает основному требованию мобильного IP (*mobile IP*): при перемещении узел Б не меняет своего адреса [87]. Альтернативный подход —

[87] Introduction to Mobility in IP. Configuring Proxy Mobile IP. Cisco Aironet 1200 Series Access Point Software Configuration Guide for VxWorks.

это т.н. кочевой IP (*nomadic IP*), когда узел меняет свой адрес в зависимости от точки подключения к сети [ibid]. Очевидно, что «кочевой IP» гораздо проще реализовать, так как он опирается только на обычную маршрутизацию, но он не обеспечит долгоживущих прикладных сеансов, способных пережить переезды узла, если только прикладной протокол сам не поддерживает смены адреса на лету.



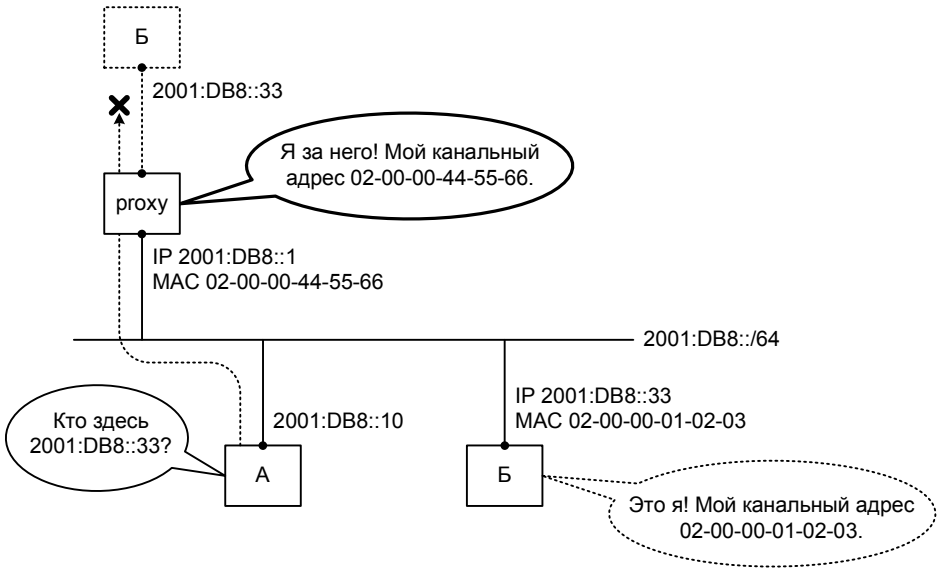
Фиг. 56. Мобильность узла, за которого отвечает ND proxy

Пусть сначала узел Б находится в другой части сети. Узел А начинает розыск узла Б запросом ARP в IPv4 или вызовом NS в IPv6. От имени узла Б отвечает proxy и сообщает свой канальный адрес. Узел А думает, что это ответил его сосед Б, и передает пакет по канальному адресу proxy. Так proxy перехватывает пакеты, предназначенные узлу Б, и поступает с ними согласно своим текущим настройкам. В это время узел Б вообще не видит запросов ARP или вызовов NS, которые посылает узел А, потому что узлы А и Б на самом деле не являются соседями.

Теперь допустим, что узел Б незаметно перемещается на данный канал и действительно становится соседом узла А, как показано на Фиг. 57. В этом случае он уже самостоятельно отвечает на запросы (вызовы) узла А. Но proxy тем временем продолжает отвечать от его имени, потому что ничего не знает о перемещении узла Б. Это значит, что на каждый запрос узла А придет два ответа (объявления), сообщающих разные канальные адреса искомого узла Б! Какой окажется запись об узле Б в кэше узла А, если он успешно получит оба ответа? В ARP все решает ее величество фортуна: окончательный вид записи зависит от порядка, в котором поступят ответы, а он, можно сказать, случайный. «Победит» ответ, пришедший последним: именно его канальный адрес останется в записи. Таким образом, примерно в половине случаев в запись об узле Б попадет канальный адрес proxy, и пакеты пойдут туда, где узла Б в данный момент нет. Проблема налицо, и нам обязательно надо исправить ее в ND, потому что сегодня мобильные узлы получают все большее распространение.

Точнее, в IPv4 один или несколько первых пакетов пойдут по канальному адресу из ответа ARP, пришедшего первым, а затем придет другой ответ, и последующие пакеты пойдут по канальному адресу из него.

http://www.cisco.com/en/US/docs/wireless/access_point/1200/vxworks/configuration/guide/bkscgch6.html#wp1032681



Фиг. 57. Мобильность приводит к сбою: трафик уходит не туда

К счастью, перед нами всего два уровня в ранжире узлов, «борющихся» за искомый адрес. Первый, с наибольшим приоритетом — это действительный владелец адреса Б. По определению индивидуального адреса, владелец у него ровно один. Второй уровень — это proxy, которых может быть и несколько. Относительный приоритет одних proxy перед другими мы определять не будем, так как в данном случае proxy — это вспомогательный механизм, и его не стоит усложнять сверх меры. Поэтому нам достаточно одного бита в формате NA, который и укажет, пришло ли объявление NA от владельца адреса или от proxy.

Теперь давайте подумаем, как получатель обязан реагировать на то или иное значение этого бита, чтобы получился желаемый результат. Нам необходимо, чтобы в итоге «выигрывал» владелец адреса, независимо от последовательности объявлений:

- если первым приходит объявление NA от владельца:
  1) по приходу объявления владельца узел А заполняет запись о соседе Б, внося в нее канальный адрес из NA;
  2) последующее объявление proxy не меняет уже полной записи о Б;
- когда первым приходит объявление NA от proxy:
  1) по приходу объявления proxy узел А заполняет запись о соседе Б, внося в нее канальный адрес из NA;
  2) объявление владельца изменяет канальный адрес в записи о Б, несмотря на то что она уже полная.

То есть объявление владельца вытесняет данные, полученные от proxy, но не наоборот. Обеспечить это можно было бы, снабдив запись в NC атрибутом: заполнена ли она по объявлению владельца или proxy. Но нельзя ли обойтись без дополнительного атрибута? Если объявление послал владелец, то оно безусловно обновляет запись в кэше, и происхождение этой записи неважно. Если же объявление исходит от proxy, а полная запись с другим канальным адресом уже существует в NC, то достаточно воспользоваться конечным автоматом NUD и просто перевести запись в состояние **ПРОСРОЧЕННАЯ**, не изменяя других ее полей.

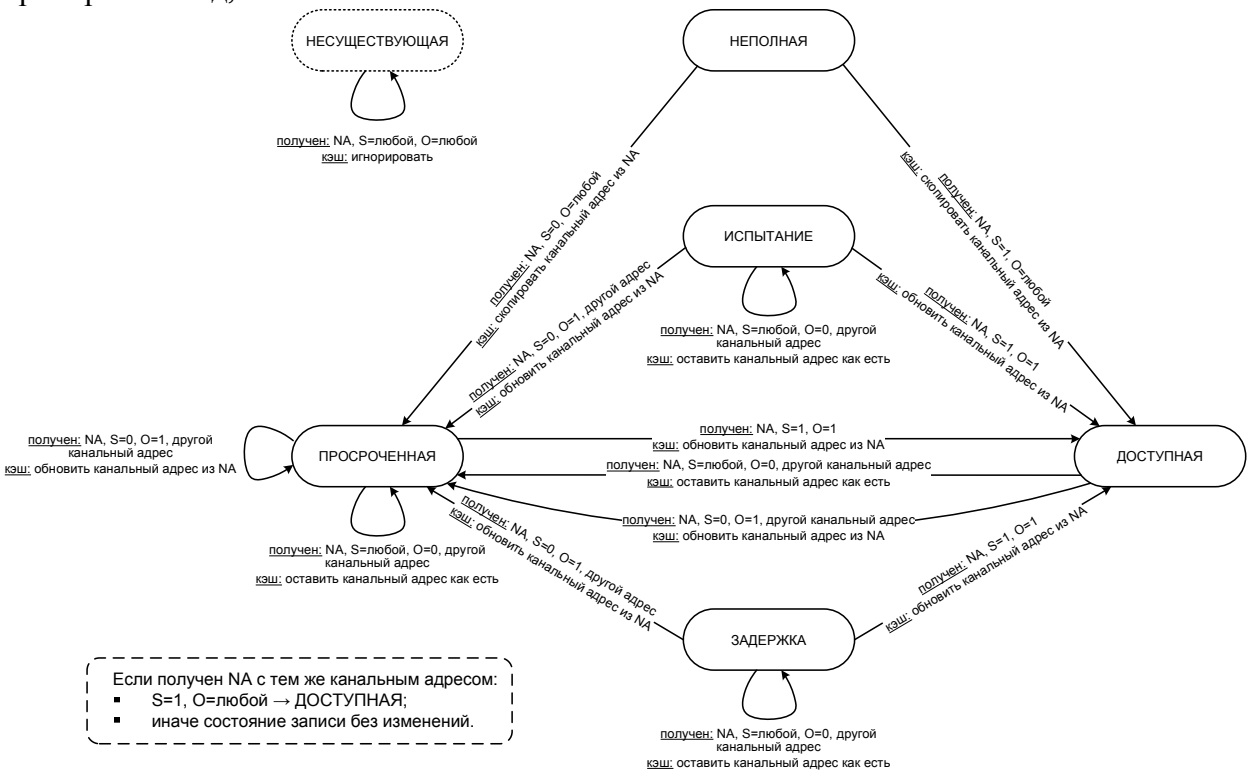
> Пожалуйста, не забывайте, что, в отличие от ARP, запись в кэше соседей ND никогда не создают по приходу объявления NA. Объявление NA может повлиять только на уже существующую запись, будь то полную или НЕПОЛНУЮ.

Став **ПРОСРОЧЕННОЙ**, запись проходит процедуру проверки. Если указанный в ней канальный адрес по-прежнему доступен, то запись просто возвращается в состояние **ДОСТУПНАЯ**. Так произойдет, если запись указывает на «живого» владельца искомого адреса, или же на «живой» proxy, который предлагает доступ к владельцу искомого

адреса. В противном случае запись исчезнет, а узел А повторит процедуру ND с самого начала. Это случится, в частности, когда узел Б снова уйдет с канала и станет доступен только через proxy.

> Поскольку все proxy равноправны, в рамках ND нет способа определить, какой именно proxy дает доступ к узлу Б, когда тот вне канала и за него отвечают несколько proxy. Эту проблему придется решать надлежащей настройкой самих proxy.
>
> Обратите внимание: механизм NUD следит только за состоянием подлинных соседей. Так, если запись NC об узле Б указывает на proxy, то ее состояние — это состояние proxy, а не самого узла Б. О доступности узла Б в этом случае NUD информации не предоставляет, потому что он ограничен рамками канала и не рассчитан на слежение за *сквозной* доступностью. Эту *архитектурную* особенность стоит иметь в виду, даже если вы не собираетесь применять ND proxy на практике.

Окончательный смысл нового флага в сообщении NA сводится к тому, замещать ли уже существующую полную запись. Поэтому данный флаг мы обозначим как «замещение» (*override*), сокращенно **O**. Если в принятом сообщении NA флаг **O** установлен (**O** = 1), то сведения из этого сообщения имеют приоритет перед записью в NC. Канальный адрес из сообщения немедленно перемещается в запись. Сама запись становится **ДОСТУПНОЙ**, если сообщение по вызову (**S** = 1), или **ПРОСРОЧЕННОЙ**, если оно добровольное (**S** = 0). Если же флаг **O** сброшен (**O** = 0), то новый канальный адрес в сообщении NA не приводит к немедленному обновлению записи NC. Вместо этого **ДОСТУПНАЯ** запись просто становится **ПРОСРОЧЕННОЙ**; в других полных состояниях запись вообще остается без изменений, потому что она уже проходит проверку [§7.2.5 RFC 4861]. После дополнения этими правилами конечный автомат NUD приобретает вид, показанный на Фиг. 58.



Фиг. 58. Диаграмма состояний ND с учетом флага O **и смены канального адреса**

Теперь, благодаря флагу **O** и связанным с ним правилам обновления канального адреса, мы можем отразить в ND еще одну полезную идею ARP. Если у интерфейса данного узла менялся канальный адрес, то узел высылал широковещательный «добровольный ARP», в котором разыскивал собственный сетевой адрес или отвечал о

нем сам себе. Это вызывало немедленное обновление существующих записей у его соседей — точнее, у тех из них, до кого добровольный ARP успешно дошел. Почему происходило обновление, мы уже обсудили: по правилам ARP, получатель сообщения первым делом искал сетевой адрес источника в своем кэше и обновлял его канальный адрес [RFC 826]. Хотя этот механизм вполне работал, сейчас давайте позаботимся, чтобы каждый тип сообщения ND делал, по возможности, свое дело. К счастью, узел IPv6 уже вправе выслать объявление NA, когда ему этого захочется. Единственная обязанность узла, когда объявление добровольное, — это сбросить в нем флаг **S**.

> Добровольный запрос ARP служил также и для того, чтобы обнаруживать конфликты адресов IP, но об этой функции мы поговорим в §5.4.1.

Поэтому, когда происходит смена канального адреса, узел IPv6 может выслать добровольное объявление NA о своем новом канальном адресе. Подходящими флагами в этом объявлении будут сброшенный флаг **S** (**S** = 0) и установленный флаг **O** (**O** = 1), поскольку объявление не по запросу, однако его источник самый авторитетный. А по какому адресу следует выслать такое объявление? Так как данный узел не знает, кто из его соседей хранит в NC запись о нем, объявление придется выслать по групповому адресу «все узлы канала», *FF02::1*. Чтобы оно наверняка не потерялось, его можно повторить несколько раз,[88] разделив передачи паузами.[77]

> Некоторые операционные системы на всякий случай шлют такое объявление NA и сразу после назначения сетевому интерфейсу адреса IPv6. В частности, за этим замечена *Cisco IOS* версии 12.4.

На этом мы исчерпали первую порцию задач, связанных с розыском соседей в IPv6, и можем наконец-то собрать вместе требования к формату сообщений NS и NA.

По нашему плану, центральный элемент сообщения NS — это адрес IPv6 искомого соседа, о котором происходит запрос. Нам уже сейчас нетрудно предвидеть, что сетевой или канальный адрес соседа, о котором идет речь в данном сообщении, будет элементом в формате многих сообщений ND. Поэтому давайте называть его одним термином: **адрес цели** (*target address*). В сообщении NS (Фиг. 59) центральное место займет сетевой адрес цели, то есть ее адрес IPv6 [§4.3 RFC 4861].

Флаги в сообщении NS нам не понадобились. Зато выше, на с. 114, мы запланировали, что в целях оптимизации сообщение NS может включать в себя канальный адрес источника (при этом сетевой адрес источника находится в заголовке IPv6). Насколько эта информация необходима в теле NS и есть ли у нее другие применения?

Мы настолько сконцентрировались на поведении узла-искателя, что совсем забыли об искомом узле и упустили из виду вот какую важную деталь: чтобы ответить индивидуальным объявлением NA на вызов NS, необходимо сперва узнать канальный адрес узла-искателя. Очевидно, что применить ND для этой цели нельзя, так как протокол зациклится и возникнет тупиковая ситуация. Поэтому верным решением будет указать канальный адрес узла-искателя, источника NS, непосредственно в самом NS. Конечно, канальный кадр, доставивший вызов NS, должен содержать в себе адрес источника, но с уровня IP доступ к канальному заголовку может быть затруднен ввиду программных ограничений стека. Поэтому вполне оправдано повторить канальный адрес источника в теле вызова NS.

Таким образом, получается, что типичное сообщение NS не только может, но и должно содержать в себе канальный адрес источника. Из этого правила будет всего одно серьезное исключение. Когда мы будем говорить в §5.4 об автоматической настройке адресов, у нас возникнет вызов NS с неопределенного адреса и без канального адреса источника. Уже и сейчас понятно, что такой особенный вызов NS понадобится, чтобы обойти очередную «проблему курицы и яйца» на нашем пути.

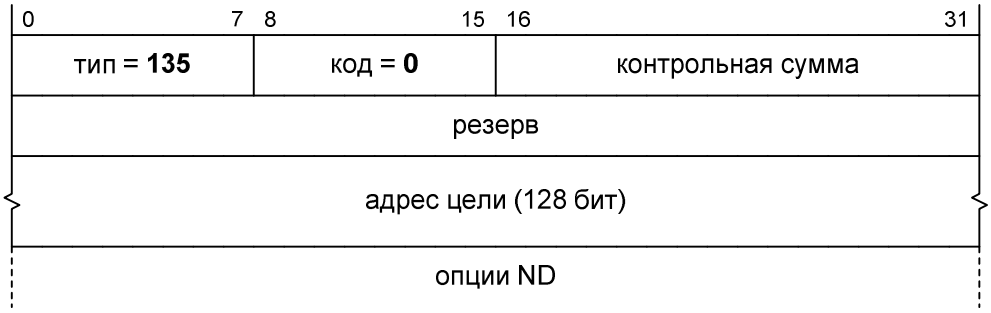
---

[88] Параметр **MAX_NEIGHBOR_ADVERTISEMENT**, 3 раза [§10 RFC 4861].

Еще есть один случай, когда канальный адрес источника в вызове NS рекомендован, но не обязателен: когда вызов NS индивидуальный [§4.3 RFC 4861]. Ведь такой вызов шлют соседу, когда его запись NC уже существует в заполненном виде, а значит, велик шанс, что у соседа тоже есть запись NC о нашем локальном узле [§7.2.2 RFC 4861]. Сосед должен был создать такую запись по приходу от нас самого первого вызова NS, который был групповым и содержал канальный адрес источника.

> Самостоятельно убедитесь, что эта оптимизация сработает и в случае однонаправленного прикладного протокола, например, Syslog или Netflow.
>
> Если вдруг окажется, что у соседа все-таки нет записи NC о нашем узле, например, потому что он перезагрузился, то ему придется провести встречный розыск соседа, послав групповой вызов NS [§7.2.4 RFC 4861]. От зацикливания протокола здесь спасет то, что групповой NS обязан содержать в себе канальный адрес источника.

В итоге, канальный адрес источника — это не вполне обязательный элемент NS, а значит, нам понадобятся опции в формате ND.



Фиг. 59. Вызов соседа — NS

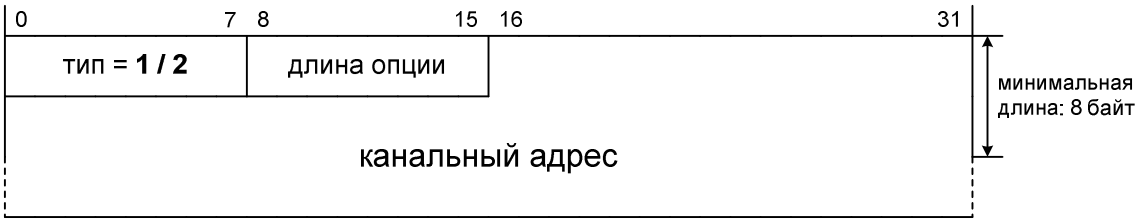
Пусть общий вид опций ND (Фиг. 60) будет основан на уже знакомом нам формате TLV: тип (1 байт), длина (1 байт), а затем переменное количество данных, формат которых зависит от типа [§4.6 RFC 4861]. Длина опции ND измеряется в 8-байтных словах и, разнообразия ради, включает в себя байты типа и длины, так что нулевое значение длины свидетельствует о сбое. Узнать о наличии опций в сообщении ND можно, определив, где находится его конец: если он совпадает с концом фиксированной части формата, то опций нет, а иначе они присутствуют. Как определить границы сообщения ND, мы уже сказали в §4.3; ведь ND — это подмножество ICMPv6.

Фиг. 60. Общий вид опции ND

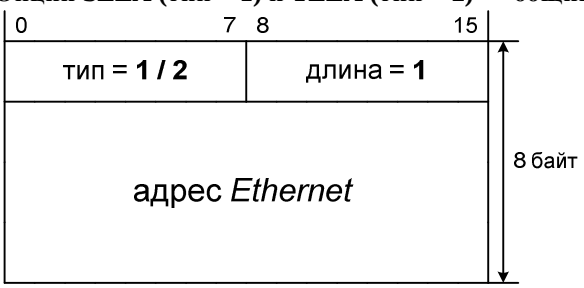
В уже потребовавшейся нам опции «канальный адрес источника» (*Source Link-Layer Address Option*, **SLLA** или **SLLAO**) роль данных играет канальный адрес, а ее тип равен 1 [§4.6.1 RFC 4861]. Такой формат опции SLLA представлен на Фиг. 61. В частности, на канале *Ethernet* длина данных будет равна шести байтам, а всей опции — единице (8 байт), как показано на Фиг. 62. Поэтому, к примеру, адрес MAC *02-03-04-05-06-07* будет инкапсулирован в эту опцию так: <0x01, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07>.

> Предполагается, что у каждой канальной технологии есть стандарт RFC, где зафиксированы детали работы IPv6 поверх каналов этого типа. Точный формат опций ND «канальный адрес» — одна из таких деталей. Например, см. [§6 RFC 2464] для *Ethernet*.

В отличие от ARP, ND не указывает явным образом тип канального адреса. Ведь он однозначно следует из типа канала, откуда пришло сообщение ND.



Фиг. 61. Опции SLLA (тип = 1) и TLLA (тип = 2) — общий формат



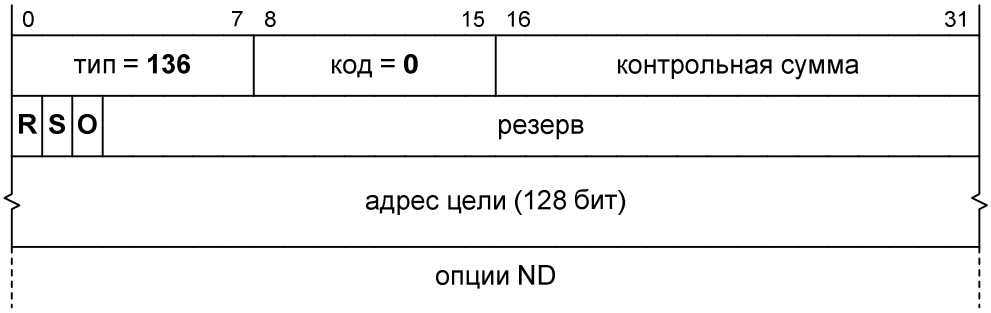
Фиг. 62. Опции SLLA (тип = 1) и TLLA (тип = 2) для *Ethernet*

Сообщение NA содержит сведения о соседе, а именно его сетевой и канальный адреса. Согласно нашей терминологии, это адреса цели. Хотя сетевой адрес источника сообщения приведен в заголовке IPv6, он может и не совпадать с сетевым адресом цели, так как сообщение мог послать proxy, а не сам искомый узел. Чтобы proxy не пришлось «подделывать» адрес в заголовке IPv6, мы поместим сетевой адрес цели отдельно, в тело сообщения NA [§4.4 RFC 4861].

Форматы NS и NA могут стать довольно похожими, если мы приложим к этому немного усилий ради упрощения будущих реализаций. Для этого пусть канальный адрес цели разместится в опции, а не в фиксированной части сообщения NA. Когда искомый узел или proxy отвечает на групповой вызов NS, он обязан включить эту опцию в объявление NA, чтобы сообщить канальный адрес цели. В то же время, когда узел отвечает на индивидуальный запрос NS, он вправе предположить, что источник запроса уже располагает правильным канальным адресом цели; в этом случае канальный адрес цели в объявлении NA можно опустить, хотя особой выгоды это не принесет. Соответствующая опция — «канальный адрес цели» (*Target Link-Layer Address Option*, **TLLA** или **TLLAO**) — отличается от опции «канальный адрес источника» только типом, который равен 2. Благодаря этому, формат обеих опций можно представить на одной иллюстрации; их общий формат показан на Фиг. 61, а формат для *Ethernet* — на Фиг. 62.

> Еще объявление NA без опции TLLA можно встретить в нестандартных применениях ND. К примеру, *Cisco IOS* версии 12.4 после настройки сетевого интерфейса PPP шлет NA с **S** = 0, **O** = 0 и без канального адреса цели — видимо, потому что у интерфейса PPP его просто нет. Адрес назначения при этом — «все узлы канала», *FF02::1*. Этим объявлением *IOS*, по-видимому, хочет сообщить узлу на другом конце канала свой локальный адрес.

Кроме того, в сообщении NA мы запланировали несколько флагов. Помимо уже знакомых нам флагов **S** и **O**, мы отведем бит для флага **R**, необходимость в котором возникнет ниже, в §5.4.2. Окончательный формат NA будет таким, как показано на Фиг. 63.



Фиг. 63. Объявление соседа — NA

Прекрасно, теперь наш узел IPv6 может составить сообщение NS или NA, когда потребуется. А в какой сетевой интерфейс он отправит это сообщение, если у него их несколько?

Что касается NS, то это сообщение уходит в интерфейс, подключенный к тому каналу, где проводится розыск соседа. Ведь мы с самого начала отметили, что отношение соседства между узлами определено только на заранее заданном канале. Поэтому розыск соседа проводится не глобально, а только после того, как узел-искатель определился с каналом. В свою очередь, соседа разыскивают, когда для него есть исходящий пакет и уже известен выходной интерфейс. Тогда будет вполне резонно передать и сообщение NS в тот интерфейс, на который указала маршрутизация исходящего пакета. Это обеспечит однозначный выбор канала для проведения розыска соседа, поскольку в архитектуре IP к интерфейсу нельзя одновременно подключить больше одного канала.

Что же касается сообщения NA, то в этом случае выбор интерфейса зависит от того, по запросу ли высылается данное сообщение. Если NA — это ответ на запрос NS (**S** = 1), то его достаточно передать в тот же сетевой интерфейс, откуда пришел запрос, и тогда оно заведомо попадет в нужный контекст канала. Ну, а если NA не по запросу (**S** = 0), то контекст канала известен из других соображений. Например, если это объявление о смене канального адреса, то его надо передать именно в тот интерфейс, на котором эта смена произошла.

Вот теперь розыск соседей на самом деле возможен: у нас есть форматы сообщений ND и правила работы с ними. А какую еще информацию извлекает из успешного обмена сообщениями NS и NA локальный узел А, кроме канального адреса искомого соседа Б? Очевидно, он узнаёт, что передача данных по каналу от А к Б и обратно, от Б к А, была возможна хотя бы на момент розыска соседа Б. Таким образом, ND позволяет узлу А убедиться в том, что доступность соседа Б симметричная, а NUD поддерживает актуальность этих сведений. В случае же асимметричной доступности, когда канал однонаправленный, ND просигнализирует полную недоступность узла Б. Иными словами, ND поддерживает только двунаправленные каналы [§3.2 RFC 4861].

> Возможно, когда-нибудь в будущем состоится попытка распространить ND и на асимметричные каналы [§3.2 RFC 4861]. Произойдет это, конечно же, когда возникнет спрос на такие каналы в среде IPv6.

На этом мы завершили работу над первой частью протокола розыска соседей, посвященной разрешению индивидуальных адресов IPv6 в канальные адреса. Ниже мы встретим еще несколько задач, решение которых будет удобным и логически обоснованным, если его оформить в рамках того же самого протокола. Поэтому протокол розыска соседей IPv6 (*Neighbor Discovery Protocol*, **NDP**) не ограничен одним только разрешением адресов [§3 RFC 4861]. Но, как говорится, всему свое время.

> Завершим раздел мы тем, что обратим внимание, как полезные разработки IPv6 постепенно находят себе дорогу обратно в IPv4. Так, в современных версиях *Linux* [89] и *MS Windows* [90] к модулю ARP фактически приделали конечный

[89] Klaus Wehrle, Frank Pählke, Hartmut Ritter, Daniel Müller, Marc Bechler. The Linux Networking Architecture: Design and Implementation of Network Protocols in the Linux Kernel. Prentice Hall, 2004. §15.3.

автомат ND, который включает в себя обратную связь с вышестоящими протоколами, тонкое управление кэшем соседей с помощью NUD и прочие новшества, которых не хватало в традиционных реализациях ARP.

5.2. Модели хоста, канала и подсети IPv6

Если птица ходит уткой, плавает уткой и крякает уткой, то ее называют уткой.
(Дж. У. Райли)

CROWD: A witch! A witch! A witch! We've got a witch! A witch!
VILLAGER #1: We have found a witch, might we burn her?
CROWD: Burn her! Burn!
BEDEMIR: How do you know she is a witch?
VILLAGER #2: She looks like one.
BEDEMIR: Bring her forward.
WITCH: I'm not a witch. I'm not a witch.
BEDEMIR: But you are dressed as one.
WITCH: They dressed me up like this.
(Monty Python and the Holy Grail)

Вертит, как цыган солнцем.
(Украинская поговорка)

До сих пор мы говорили об узлах IPv6 в целом, практически не проводя границ между хостом и маршрутизатором, и это было вполне оправданно, пока мы обсуждали базовые механизмы IPv6, не зависящие от типа узла.

Нам не повредит напомнить себе лишний раз определения хоста и маршрутизатора IP. В идеале, хост только создает и потребляет пакеты, тогда как маршрутизатор занят исключительно тем, что продвигает пакеты хостов. Конечно, на практике маршрутизатор обычно содержит в себе элементы хоста, необходимые для обработки служебных сообщений, удаленного управления и участия в протоколах маршрутизации.

Однако рано или поздно перед нами должен встать вопрос, можем ли мы положиться на наш опыт IPv4 и по-прежнему считать различия в устройстве и поведении хоста и маршрутизатора незначительными во всем, что не касается функции продвижения транзитного трафика.

Ведь, как мы знаем, в IPv4 произошла конвергенция этих двух типов узла, в результате чего между ними не осталось принципиальных отличий, а выбор роли узла свелся к настройке флажка, пропускать ли транзитный трафик. Современные операционные системы готовы обеспечить работу и хоста, и маршрутизатора IPv4, используя один и тот же программный код, а за этим стоит единство алгоритмов и структур сетевого стека. В частности, оба типа узлов IPv4:

а) пользуются таблицей маршрутов одного и того же вида: «префикс → следующий шаг» [ср. §3.3.1.2 RFC 1122 и §5.2.4.3 RFC 1812];

Хотя [§3.3.1.2 RFC 1122] говорил, что полноценная таблица маршрутов — необязательный элемент в сетевом стеке хоста IPv4, на практике она встречается повсеместно.

[90]Description of Address Resolution Protocol (ARP) caching behavior in Windows Vista TCP/IP implementations. Microsoft KB article 949589. http://support.microsoft.com/kb/949589

б) одинаковым способом определяют, является ли удаленный узел соседом, сопоставляя префиксы на своих сетевых интерфейсах с адресом удаленного узла [ср. §3.3.1.1 RFC 1122 и §5.2.4.2 RFC 1812].

Мы сознательно не упоминаем ненумерованные интерфейсы «точка-точка», так как они не требуют розыска соседей.

А на самом глубоком уровне такую конвергенцию обеспечивает определенная модель взаимодействия хостов и маршрутизаторов одной подсети. Согласно этой модели, хост сам принимает решение, кому из соседей передать исходящий пакет, тогда как маршрутизатор только продвигает транзитный трафик, который был передан ему хостом. Чтобы эта модель работала, хост должен обладать полной информацией о логической топологии канала, которому назначена подсеть. Только тогда хост сможет правильно принять решение о том, является ли удаленный узел соседом. Это очень важное решение; ведь если хост ошибется и посчитает соседом узел, который на самом деле доступен только через маршрутизатор, то попытка передать ему пакет будет обречена на неудачу.

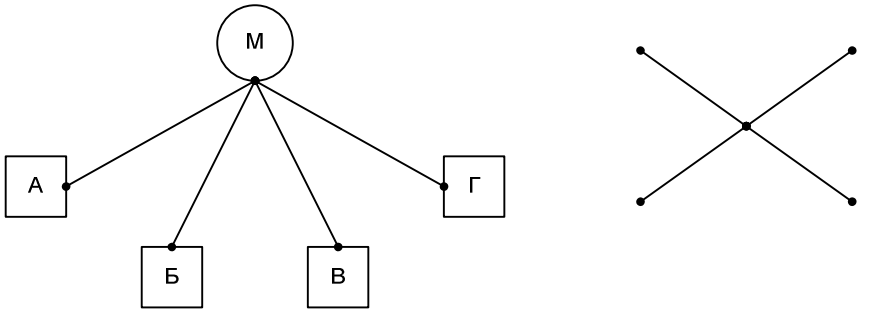
Сейчас мы снова опираемся на постулат, который утверждает, что одна подсеть отвечает не более чем одному каналу [§2.1 RFC 4291].

Проанализируем с этой точки зрения тест на соседство IPv4. Допустим, что узел А собирается передать пакет, адресованный узлу Б, посредством одного из своих интерфейсов И, которому назначены адреса *192.0.2.3/28* и *203.0.113.131/28*. Следовательно, с точки зрения узла А, каналу, к которому подключен интерфейс И, назначены подсети *192.0.2.0/28* и *203.0.113.128/28*. Если адрес назначения пакета Б равен *192.0.2.8* или, скажем, *203.0.113.136*, то узел А немедленно определит, что этот адрес принадлежит соседу, потому что он содержит префикс непосредственно подключенной подсети. С точки зрения IP, такой пакет можно передать напрямую его адресату, полагаясь на канальный уровень стека. В то же время, ни один из адресов *192.0.2.150*, *198.51.100.1* или *203.0.113.2* не принадлежит соседу, и, следовательно, такие адресаты доступны только косвенно, через маршрутизатор.

На самом деле, тому же каналу могут быть назначены и другие подсети IPv4, о которых узел А не знает, потому что из них ему адрес не назначили.

Эта логика IPv4 основана на гипотезе, что все узлы одной подсети — попарно соседи и могут передавать друг другу пакеты напрямую, не задействуя маршрутизатор. Такое допущение, безусловно, справедливо для широковещательных каналов, например, *Ethernet*, равно как и для соединений «точка-точка». Но насколько оно универсально? Существуют ли каналы, в которых оно не выполняется?

Оказывается, что в действительности спрос на такие каналы есть. Их принято обозначать термином «**каналы множественного доступа без широковещания**» или аббревиатурой **NBMA** (*Non-Broadcast Multiple Access*). Это не совсем точный термин, потому что на практике в каналах NBMA обычно отсутствует не только широковещание, но и свободная адресация одних узлов другими. Простейший пример такого канала — это топология «звезда» (Фиг. 64), где концевые узлы могут передавать пакеты центральному узлу (концентратору), тогда как напрямую обмениваться данными друг с другом концевые узлы не могут. В топологии «звезда» концентратор — сосед любому концевому узлу, и наоборот, но никакие концевые узлы — не соседи друг другу.



Фиг. 64. Канал NBMA «звезда» и его граф

Классические примеры каналов NBMA — это «звезды» на основе Frame Relay и ATM, где каждый «луч» был виртуальным соединением (*virtual circuit*).

Современными примерами каналов NBMA типа «звезда» служат кабельные, сотовые и xDSL-сети. В отличие от *Ethernet* и *WiFi*, в них концентратор не играет роли коммутатора. Поэтому обмен пакетами между конечными узлами в них возможен только на сетевом уровне, при поддержке маршрутизатора, логика которого часто встроена в сам концентратор.

Но даже локальную сеть *Ethernet* можно превратить в «звезду» NBMA с помощью частных VLAN. Делают это из соображений безопасности, чтобы централизованно управлять доступом между конечными узлами, не отказываясь при этом от простой схемы «одна ЛВС и одна подсеть».

Если бы мы проводили теоретический анализ топологии канала, то нам, возможно, следовало бы различать схемы «общая среда без широковещания» (когда возможна индивидуальная передача любому узлу, если его канальный адрес заранее известен) и «точка-многоточка» (звезда) в чистом виде [§4.1 RFC 4903]. Однако на практике чаще всего встречается та или иная их комбинация, когда только некоторые пары узлов — соседи между собой. Поэтому для нас сейчас эти две схемы — только частные, крайние случаи более общей модели, в которой возможность узлов канала адресовать друг друга индивидуально и группами ограничена заданным образом. Такую модель вполне допустимо причислить к NBMA.

Конечно, внимательный читатель обоснованно возразит, что такая «звезда» — это вовсе не один канал, а комбинация нескольких каналов «точка-точка», и он будет абсолютно прав, по крайней мере, в теории. Однако, с практической точки зрения, превратить каждый «луч» в отдельный канал — это неудачный выбор. Представьте себе концентратор, к которому уже подключены тысячи абонентов и ежедневно подключаются сотни новых. Если действовать прямолинейно и работать с каждым подключением как с отдельным каналом «точка-точка», то сегодня концентратору потребуется несколько тысяч сетевых интерфейсов, а завтра это число может достичь и миллиона. Хотя теоретик не увидит в этом ничего страшного, на практике число сетевых интерфейсов узла не может расти неограниченно.

Число сетевых интерфейсов узла ограничено не только потому, что программисты-разработчики сетевого стека поленились и использовали неоптимальный, немасштабируемый алгоритм работы с ними. Более фундаментальная проблема в том, что сетевой интерфейс — это центральный объект управления, с которым связана определенная конфигурация практически в каждом компоненте сетевого стека. Впрочем, этот аргумент можно отклонить на том основании, что объем ОЗУ в современных вычислительных системах практически неограничен, а любое число однотипных интерфейсов можно настроить с помощью одного шаблона. Еще один веский аргумент за ограничение числа сетевых интерфейсов заключается в том, что политику сетевой безопасности удобно формулировать в терминах входного и выходного интерфейсов пакета, но тогда число правил такой политики может расти как квадрат числа интерфейсов.

> Но даже эту трудность можно обойти, объединив интерфейсы в зоны безопасности или перейдя к иной модели управления доступом. Однако в конечном итоге оказывается, что проще ограничить число интерфейсов, нежели постоянно преодолевать препятствия.

Именно поэтому возникает спрос на искусственное решение, когда одна подсеть назначена множеству подключений, а это автоматически превращает их в один канал согласно фундаментальному правилу IP: «Одна подсеть — это один канал» [§2.1 RFC 4291]. Конечно, такая конструкция напоминает цирковой фокус с легким привкусом обмана. Тем не менее, давайте посмотрим, чего будет стоить ее реализация в IPv6. Ведь мы находимся в поле эксперимента, а искомая возможность вполне востребована. Как обычно, в процессе мы встретим и решим более глубокие и общие задачи.

Новые условия таковы, что канал может обладать нетривиальной логической топологией, когда не все его узлы — соседи. Ранее, когда модель канала была простой и требовала соседства между всеми узлами, эта топология канала была скрыта в самом тесте на соседство IPv4. Теперь же информацию о топологии канала придется задавать и хранить явным образом, например, в виде матрицы соседства, где 1 означает пару соседей, а 0 — разъединенную пару узлов. Такие матрицы для канала «звезда» (Фиг. 64) и широковещательного канала с тем же набором узлов приведены в Табл. 13 и Табл. 14, соответственно. Придется ли вводить подобную матрицу в настройки каждого из узлов канала?

> Интересующие нас матрицы соседства — симметрические, потому что, согласно нашей модели из §5.1, отношение соседства коммутативно.

Табл. 13. Матрица соседства для канала «звезда»

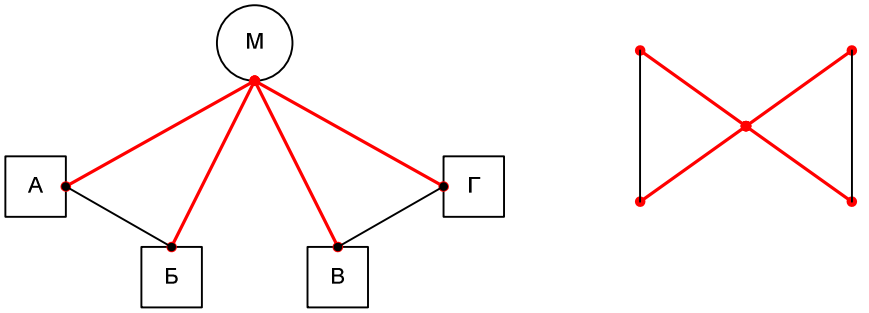
	А	Б	В	Г	М
А	1	0	0	0	1
Б	0	1	0	0	1
В	0	0	1	0	1
Г	0	0	0	1	1
М	1	1	1	1	1

Табл. 14. Матрица соседства для широковещательного канала

	А	Б	В	Г	М
А	1	1	1	1	1
Б	1	1	1	1	1
В	1	1	1	1	1
Г	1	1	1	1	1
М	1	1	1	1	1

Для начала мы ограничимся каналом, который обслуживает один маршрутизатор. Остальные узлы такого канала — хосты. Если бы топология этого канала была совершенно произвольной, то, наверное, обойтись без дублирования топологической информации было бы нельзя, и понадобился бы какой-то протокол ее распространения.

Однако для IP практическую ценность представляют только топологии, в которых маршрутизатор — сосед каждого хоста. Действительно, в противном случае для беспрепятственной работы канала потребовался бы транзитный трафик через хосты, а это по определению невозможно. Говоря языком элементарной теории графов, граф «звезда» с маршрутизатором в центре будет подграфом любой топологии канала, представляющей интерес в контексте TCP/IP. Это правило иллюстрирует Фиг. 65.



Фиг. 65. Канал NBMA «бабочка» содержит подграф «звезда»

Тогда маршрутизатор IPv6 будет естественным хранителем информации о топологии канала, потому что он может обслуживать практически неограниченное число хостов (до ~2^{63}) и при этом он заведомо способен передать сведения о топологии канала каждому подключенному хосту. Механизм распространения этих сведений нам сейчас предстоит разработать.

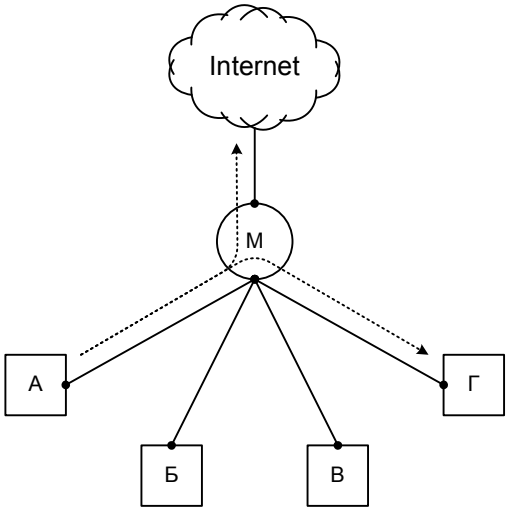
Каких начальных настроек будет достаточно хосту IPv6 с одним сетевым интерфейсом, чтобы начать работу с сетью в таких условиях? Нетрудно убедится, что абсолютный минимум — это его собственный адрес и адрес маршрутизатора. В то же время, длина префикса подсети здесь излишня, так как она больше не содержит информации о топологии канала.

> Здесь мы имеем в виду адрес хоста, область которого больше, чем внутриканальная. Помимо него, у хоста будет как минимум один внутриканальный адрес — к этому требованию мы уже пришли в §2.5.
>
> В то же время, адрес маршрутизатора вполне может быть внутриканальным. Мы обсудили это там же, в §2.5.
>
> Конечно же, как это было сказано в §2.6, адрес IPv6 назначают не всему узлу в целом, а определенному его интерфейсу — в данном случае, интерфейсу в интересующий нас канал. По текущим условиям, такой интерфейс у каждого узла один, и это позволяет нам для простоты говорит «адрес узла».

В отсутствие других сведений хосту IPv6 не остается ничего, кроме как передавать *весь* исходящий трафик через этот маршрутизатор, так что для хоста это будет **маршрутизатор по умолчанию** (*default router*).

> Очевидное исключение — это пакеты, в которых хост адресует сам себя. Их не следует отправлять на маршрутизатор по умолчанию, потому что такие пакеты вообще не покидают границ хоста.
>
> Любой имеющий опыт работы с IPv4 сказал бы, что у нашего модельного хоста есть только маршрут по умолчанию, *::/0*, и нет маршрута на непосредственно подключенную подсеть. Однако суть как раз в том, что модель хоста IPv6 целенаправленно избегает появления таблицы маршрутов среди структур данных хоста. Сделано это, не в последнюю очередь, для упрощения минимальной реализации, которая найдет себе место во встроенных системах. Фактически, происходит попытка снять с хоста все функции, которые будут заведомо доступны в маршрутизаторе. Реализация хоста IPv6 может включать в себя эти функции факультативно, но они не обязательны, и их отсутствие не повредит работоспособности хоста.

Чтобы начать передачу данных через маршрутизатор, хосту надо выяснить его канальный адрес, а для этого, в зависимости от типа канала, может потребоваться розыск соседа. Однако успех этой операции обеспечен, несмотря на сложную топологию канала, тем, что маршрутизатор — заведомо сосед хоста, если канал отвечает базовым требованиям TCP/IP. Так оказывается, что для начала работы хосту IPv6 вообще не нужны дополнительные сведения о топологии канала: ему достаточно слать весь исходящий трафик через маршрутизатор по умолчанию, и он уже способен это сделать.



Фиг. 66. Маршрутизация обратно в канал NBMA и за его пределы

Для топологии «звезда» этого решения будет вполне достаточно, так как у хостов все равно нет иного способа обмениваться данными, кроме как через маршрутизатор, хотя формально они подключены к одному каналу. Такая конфигурация представлена на Фиг. 66.

Теперь усложним топологию канала и допустим, что она включает в себя прямые связи между некоторыми хостами. Для примера рассмотрим топологию «бабочка» (Фиг. 65), где две пары хостов снабжены такими связями. Ее матрица соседства приведена в Табл. 15. Как мы уже отметили, «звезда» обязательно будет подграфом такой топологии (см. Фиг. 65), и наша простейшая схема работы хоста, с одной стороны, останется возможной. С другой стороны, она утратит свою оптимальность, так как хосты А и Б или В и Г смогли бы обмениваться пакетами напрямую, не создавая нагрузки на маршрутизатор М. Сведения о соседстве хостов А и Б, а также В и Г, хранятся маршрутизатором М. Каким образом М мог бы сообщить, скажем, хосту А, что тот вполне способен передавать пакеты хосту Б напрямую? И когда М следует это сделать? Потребуется ли особый протокол, по которому А запросит недостающую информацию у М?

Табл. 15. Матрица соседства для канала «бабочка»

	А	Б	В	Г	М
А	–	1	0	0	1
Б	1	–	0	0	1
В	0	0	–	1	1
Г	0	0	1	–	1
М	1	1	1	1	–

На самом деле, здесь достаточно простой **переадресовки** (*redirect*), которую будет естественно оформить в виде сообщения ICMPv6, принадлежащего к семейству ND [§4.5 RFC 4861]. Последовательность событий может быть такой:

1) хост А передает пакет, адресованный Б, через маршрутизатор М, и вносит запись вида «Б→М» в свой кэш адресатов DC;
2) маршрутизатор М замечает, что в матрице соседства есть прямая связь А–Б;
3) тем не менее, он не отбрасывает пакет, а исправно передает его Б;
4) затем, чтобы оптимизировать будущий трафик, М направляет хосту А сообщение ND типа «переадресовка», которое уведомляет, что адресат Б доступен через следующий шаг Б, то есть напрямую;
5) хост А принимает переадресовку к сведению, исправляя запись Б в кэше адресатов на «Б→Б»;

6) пока кэш адресатов хоста А содержит запись «Б→Б», хост А передает пакеты хосту Б напрямую;
7) когда запись устаревает и удаляется из кэша, вся процедура повторяется с самого начала;
8) если же переадресовка потерялась и не дошла до хоста А, процедура повторится, как только А передаст следующий пакет в адрес Б;
9) наконец, если сеть вдруг продублирует пакет с переадресовкой, то запись DC «Б→Б» не изменится, потому что переадресовка обладает свойством идемпотентности.

Конечно, маршрутизатор М не должен слать переадресовку на *каждый* пакет от А к Б, если хост А игнорирует переадресовки и продолжает слать пакеты через М. Из соображений устойчивости и безопасности частоту переадресовок следует ограничивать [§8.2 RFC 4861]. Впрочем, это справедливо практически для всех сообщений ICMPv6.

У этого решения есть несколько очевидных преимуществ. Во-первых, оно использует уже готовые структуры данных, в частности, кэш адресатов хоста. Во-вторых, оно требует всего лишь одного нового сообщения ND с простыми свойствами. В-третьих, оно устойчиво к потере и дублированию переадресовок. Наконец, в-четвертых, матрица соседства, хранимая маршрутизатором М, может изменяться по ходу работы канала, и хост А оперативно обновит свой взгляд на топологию канала без внешнего вмешательства благодаря работе кэша адресатов. Например, если связь А–Б исчезнет, то хост А станет слать трафик в адрес Б через маршрутизатор М, как только устареет прежняя запись в кэше адресатов DC.

Своевременно обнаружить исчезновение связи А–Б и переключиться обратно на маршрутизатор М хосту А помогут механизм NUD (§5.1) и обратная связь между записями NC и DC. Так, когда NUD сигнализирует недоступность соседа Б, можно удалить заодно и все записи DC, чей следующий шаг равен Б. Тогда время жизни самих записей DC можно и не ограничивать. Мы обобщим эту идею ближе к концу раздела.

Данный механизм настолько прост и универсален, что он заслуживает роли стандартного для всех хостов IPv6. Кроме того, он предъявляет весьма низкие требования к вычислительным возможностям хоста, а это как нельзя лучше соответствует тенденции применять IP в промышленных сетях управления, где хосты — устройства маломощные и с низким энергопотреблением.

Таким образом, хост IPv6 при передаче исходящих пакетов руководствуется существенно иными правилами, чем хост IPv4. Это различие нам следует четко отразить в терминологии и процедурах, чтобы избежать путаницы.

Независимо от версии IP, перед хостом стоит одна и та же основная задача:

Дано: индивидуальный адрес назначения исходящего пакета IP.

Вопрос: доступен ли сетевой интерфейс с этим адресом непосредственно, на канальном уровне?

Что означает для хоста *H* доступность удаленного сетевого интерфейса *I* на канальном уровне? Очевидно, это значит, что хост *H* посредством одного из его собственных, локальных интерфейсов *J* может обмениваться данными канального уровня с интерфейсом *I*, а значит, и его узлом-владельцем. Ведь всякий сетевой интерфейс а) подключен ровно к одному каналу и б) принадлежит ровно одному узлу. (Когда интерфейс не подключен, он не готов к работе и, с нашей точки зрения, не существует.) Необходимое, но недостаточное условие доступности на канальном уровне — это чтобы у хоста *H* нашелся интерфейс *J*, подключенный к тому же каналу *L*, что и интерфейс *I*. Дополнительное условие — это фактическая возможность обмена данными на канальном уровне между интерфейсами *I* и *J* по

каналу *L*. Если, например, в локальных сетях второе следует из первого, то в каналах NBMA все зависит от логической топологии данного канала.

Будет любопытно отметить, что понятие «петли» (*loopback*) позволяет обобщить эту конструкцию и на случай, когда интерфейс назначения *I* принадлежит локальному хосту *H*. В этом случае тривиальное решение выглядит как $J = I$. Впрочем, если у хоста несколько интерфейсов, подключенных к каналу *L*, то возможны и нетривиальные решения, где $J \neq I$.

Если ответ на этот ключевой вопрос утвердительный, то хост проведет разрешение сетевого адреса назначения в отвечающий ему канальный адрес с помощью ARP или ND, в зависимости от версии IP, и сможет передать пакет его адресату напрямую, из интерфейса в интерфейс. В противном случае хост должен воспользоваться услугами маршрутизатора.

А вот общепринятые способы найти ответ на данный вопрос в двух версиях IP существенно разнятся.

Как мы уже сказали, хост IPv4 исходил из предположения, что все адреса в одной с ним подсети доступны напрямую. О таких адресах мы говорили, что они непосредственно подключенные (*directly connected*). У непосредственно подключенного адреса может быть всего два состояния: или он вообще не назначен, или он принадлежит интерфейсу, который доступен локальному хосту на канальном уровне. Все прочие адреса не являются непосредственно подключенными и доступны хосту только через маршрутизатор. Классификацию адресов хост IPv4 проводил с помощью несложной побитовой арифметики, используя список префиксов подсетей, назначенных его собственным интерфейсам. Вдобавок, если результат теста был положительный, то наиболее точный префикс однозначно указывал на интерфейс, сквозь который доступен непосредственно подключенный адрес.

Напротив, хост IPv6 больше не доверяет информации о подсетях, потому что она не всегда отражает действительную топологию каналов. Хост IPv6 не может априори определить, принадлежит ли произвольный адрес соседу, и потому рассчитывает на дополнительные источники информации, чтобы классифицировать каждый адрес назначения индивидуально. Если об адресе нет иной информации, то хост IPv6 по умолчанию полагает, что соответствующий сетевой интерфейс недоступен на канальном уровне, а адрес находится «**вне канала**» (*off-link*) [§3 RFC 5942].

Обратите внимание на это правило. Оно существенно отличается от общепринятой практики IPv4.

Из каких источников хост может почерпнуть иные сведения? Во-первых, адреса его собственных интерфейсов доступны ему напрямую, потому что хост всегда может послать пакет сам себе, минуя маршрутизатор. Во-вторых, если в настройках хоста указан адрес маршрутизатора по умолчанию, то он также заведомо доступен напрямую. В свою очередь, эта настройка открывает хосту возможность обмена данными с другими хостами, даже если их адреса «вне канала». В процессе этого обмена хост может получить от маршрутизатора переадресовку вида «Б→Б» и таким образом узнать, что интерфейс с адресом Б доступен ему на канальном уровне. Все эти источники информации сообщают хосту IPv6, что адрес находится «**на канале**» (*on-link*).

Модель, в которой все адреса по умолчанию находятся «вне канала», довольно нова и опыт работы с ней относительно невелик, так что даже некоторые стандарты потребовали корректив. Скажем, все еще действующий стандарт ND предписывал, что адрес — «на канале», если о нем пришло объявление NA, или же если он выступает источником любого сообщения ND [§2.1 RFC 4861]. Однако это поведение противоречит модели, в которой единственный достоверный источник такой информации — маршрутизатор. Кроме того, оно открывает возможность для атаки, когда злоумышленник на канале перехватывает трафик, предназначенный узлу вне канала. Поэтому [п.3 §3 RFC 5942] запрещает хосту подобный трюк.

В то же время, хосту не возбраняется «разогревать» свой кэш соседей NC, заранее создавая в нем записи, но не делая при этом выводов о непосредственной доступности узлов. К примеру, получив запрос NS с опцией SLLA, хост создает о его источнике запись NC в состоянии **ПРОСРОЧЕННАЯ** [§3 RFC 5942 и §7.2.3 RFC 4861], потому что в будущем это позволит сэкономить время и обойтись без явного розыска соседа, как мы обсудили в §5.1. Но воспользоваться этой записью NC для прямой передачи хост будет вправе, только когда маршрутизатор подтвердит, что адресат «на канале», и будет создана соответствующая запись в кэше адресатов DC.

Только теперь, поняв и сформулировав главное различие между алгоритмами теста на соседство IPv4 и IPv6, мы можем безопасно оптимизировать алгоритм IPv6, не рискуя навредить нашему пониманию основ.

На практике о *некоторых* диапазонах адресов может быть известно, что они «на канале», с точки зрения данного хоста. К их числу относятся:

1) все внутриканальные адреса, *FE80::/10*;
2) подсети, назначенные подлинно широковещательным каналам;
3) диапазоны адресов меньше подсети, внутри которых топология канала гарантирует прямую доступность.

Кроме того, условно считается, что групповые адреса всегда «на канале», поскольку групповой трафик IP распространяется в сети не так, как индивидуальный. Групповые маршрутизаторы слушают все обслуживаемые ими группы и потому, с точки зрения источника трафика, они ничем не отличаются от рядовых членов группы на том же канале. Подробнее мы поговорим об этом в §6.4. Кроме того, к групповым адресам никогда не применяют ND. Поэтому групповые адреса стоят особняком, и нет смысла пытаться включить их в наш текущий дискурс о критериях «на канале/вне канала».

В первых двух случаях диапазоны адресов уже представлены префиксами. Третий случай не столь важен, но в нем диапазон тоже можно представить набором префиксов. Поэтому вместо произвольных диапазонов мы можем говорить о префиксах «на канале». Если список этих префиксов сообщить хосту, то он сможет эффективнее использовать свое соседство с другими узлами и реже обращаться к услугам маршрутизатора по умолчанию. Соответствующая структура в памяти хоста IPv6 так и называется: **список префиксов** (*Prefix List*). Если адрес назначения содержит в себе один из этих префиксов, то он для данного хоста — «на канале».

Сделаем замечание по терминологии. Строго говоря, префикс может *содержаться в адресе*, тогда как адрес может *принадлежать блоку*, представленному этим префиксом. Поэтому говорить, что «адрес принадлежит префиксу», как советовал нам редактор, было бы искажением технической сути этих отношений.

Свежеполученный опыт подсказывает нам, что список префиксов надо заполнять с осторожностью, чтобы не навредить работе каналов нетривиальной топологии. По умолчанию в этом списке ровно один элемент — внутриканальный префикс *FE80::/10* [§5.1 RFC 4861, §3 RFC 5942].

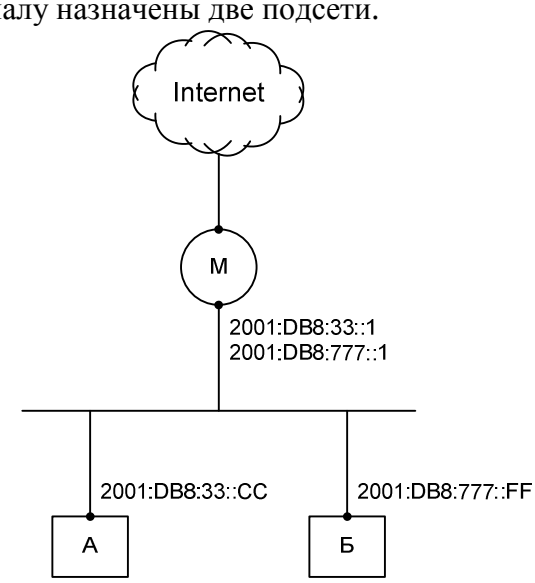
По этой причине хосты на канале NBMA не могут свободно взаимодействовать друг с другом, используя внутриканальные адреса. На это ограничение приходится идти ввиду особой роли внутриканальных адресов. Хотя, зонная архитектура IPv6 вполне допускает маршрутизацию внутриканальных адресов маршрутизатором обратно в тот же канал, использовать эту возможность по умолчанию не следует.

В то же время, скажем, интерфейс, подключенный к каналу *Ethernet*, еще ничего не говорит о фактической топологии этого канала. Например, с помощью частных VLAN топологию такого канала можно ограничить и превратить его из широковещательного в

NBMA. По этой причине хосту не следует автоматически вносить в список префиксов подсети только потому, что они назначены интерфейсам определенного типа.

Согласно нашему генеральному плану, точными сведениями о топологии канала обладает маршрутизатор. Поэтому самая достоверная информация о списке префиксов могла бы исходить именно от маршрутизатора. Пока что мы не располагаем подходящим механизмом для этого, но скоро он у нас тоже появится (в §5.4.2).

С другой стороны, теперь подсеть, в которой находится интерфейс хоста IPv6, больше не влияет на классификацию «на канале» или «вне канала», как это было в IPv4. Поэтому мы вполне можем допустить случай, когда удаленный адрес оказывается «на канале», хотя не принадлежит подсети хоста. Для примера рассмотрим сценарий на Фиг. 67, в котором одному каналу назначены две подсети.

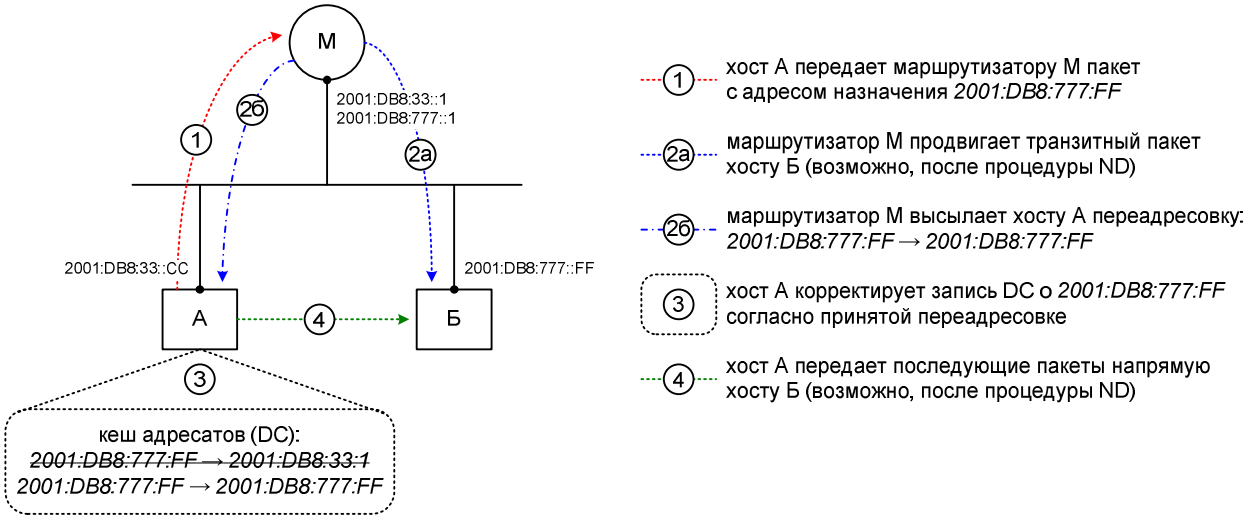


Фиг. 67. Один канал и один маршрутизатор, но две подсети

Как работала бы подобная схема в IPv4? Допустим, у хоста А был бы адрес *192.0.2.33/24*, у хоста Б — *203.0.113.44/24*, а у маршрутизатора М — *192.0.2.1/24* и *203.0.113.1/24*. В этой схеме весь трафик между хостами А и Б неизбежно проходил бы через маршрутизатор М. Почему? Во-первых, если бы хосту А пришла переадресовка ICMP вида «*203.0.113.44→203.0.113.44*», то хост А проигнорировал бы ее, потому что следующий шаг в ней не являлся непосредственно подключенным [§3.2.2.2 RFC 1122]: *203.0.113.44* не принадлежал подсети *192.0.2.0/24*. А во-вторых, маршрутизатор М не стал бы слать хосту А такую переадресовку [§5.2.7.2 RFC 1812], потому что он заранее знал: она будет проигнорирована. Те же соображения, с точностью до перестановки адресов, определили бы и обратный ход трафика от Б к А.

Теперь же, в IPv6, у хоста А не будет причин не принять переадресовку «Б→Б» просто потому, что именно маршрутизатор — достоверный источник сведений о топологии канала [§8.1 RFC 4861]. В свою очередь, маршрутизатор М не должен ограничивать переадресовки одной подсетью, если он располагает информацией о том, что хосты А и Б — соседи. И тогда хост А сможет узнать и принять к сведению, что хост Б «на канале» и доступен напрямую, как это показано на Фиг. 68.

> Обратите особое внимание на это важное различие между переадресовками IPv4 и IPv6 [§3.1 RFC 4861].



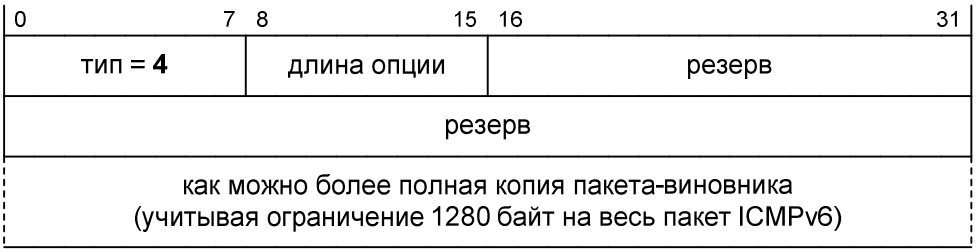
Фиг. 68. Механизм переадресовки

Это новое свойство нам следует перенести и на список префиксов. Для этого скажем: элементы списка префиксов не обязаны принадлежать подсетям, в которых находится сетевой интерфейс хоста. К примеру, список префиксов хоста X с адресом *2001:DB8:33::33/64* вполне может содержать префиксы *2001:DB8:33::/48* (префикс короче подсети) и *2001:DB8:777::/64* (другая подсеть). Все адреса с такими префиксами будут для хоста X «на канале», и тот сможет слать пакеты их владельцам напрямую.

Теперь нам пора поработать над форматом переадресовки. Как мы уже сказали, она будет разновидностью сообщения ICMPv6 из семейства ND [§4.5 RFC 4861]. Переадресовка — это сообщение с любопытными свойствами. С одной стороны, переадресовка — это полезная подсказка хосту, а вовсе не извещение об ошибке. Поэтому ее численный тип находится в диапазоне справочных сообщений 128–255 и равен 137. С другой стороны, переадресовка высылается в ответ на произвольный пакет IPv6, что отличает ее от других справочных сообщений и роднит с извещениями об ошибках. В этом случае, как мы знаем, служебное сообщение должно содержать в себе начало пакета, который его вызвал, чтобы получатель смог более точно отреагировать на него, например, определив, к какому сеансу TCP относится это сообщение.

Разрешить это противоречие в свойствах переадресовки будет довольно просто, если мы воспользуемся механизмом опций ND и поместим пакет-виновник (или его начало) в специально отведенную для этой роли опцию «переадресованный заголовок» (*Redirected Header*), показанную на Фиг. 69. Правила работы с ней очевидны:

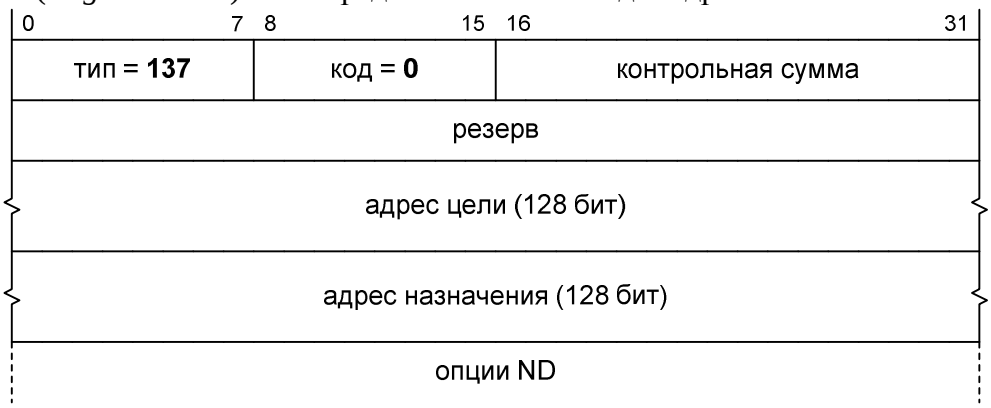
- опция «переадресованный заголовок» имеет смысл только в сообщении «переадресовка», а в прочих сообщениях ND ее следует игнорировать;
- полезная нагрузка этой опции содержит весь пакет-виновник или его начало, так чтобы суммарная длина пакета ICMPv6 вместе с заголовками IPv6 не превысила максимально допустимое ограничение в 1280 байт, заданное в §3.3.4.



Фиг. 69. Опция ND «переадресованный заголовок»

А какие обязательные поля мы поместим в тело переадресовки? Назначение переадресовки заключается в том, чтобы сообщить получателю, что оптимальный путь к адресату Б пролегает через следующий шаг Ш, где Б и Ш — адреса IPv6. Поэтому фиксированная часть тела переадресовки состоит, помимо выравнивания, всего из двух

адресов IPv6. Как показано на Фиг. 70, первым из них расположен адрес следующего шага Ш. Так как его владелец — заведомо сосед получателя, то, по терминологии ND, это **адрес цели** (*target address*). Непосредственно за ним идет адрес назначения Б.



Фиг. 70. Переадресовка IPv6

Адрес цели идет первым для единообразия с другими сообщениями ND.

Хотя адрес назначения также содержится в копии пакета-виновника (опция «переадресованный заголовок»), его дублирование в фиксированной части переадресовки облегчает работу ее получателя.

Здесь мы видим еще одно важное отличие между переадресовками IPv6 и IPv4. Как мы помним, переадресовка IPv4 могла быть для одного адреса или целой сети, хотя формат переадресовки для сети был классовым и не имел смысла в CIDR. Переадресовка же IPv6 — исключительно для одного адреса, и это позволяет хранить ее сведения в кэше DC.

Адреса Б и Ш вполне могут совпадать. Именно этот случай нам сейчас особенно интересен. Ведь такой переадресовкой маршрутизатор сообщает, что узел Б, он же Ш, находится «на канале».

Случай, когда адреса Б и Ш не совпадают, тоже, безусловно, важен на практике. Такая переадресовка говорит хосту, что оптимальный путь к адресату Б пролегает через другой маршрутизатор на том же канале, а именно Ш.

Прочие требования к годной переадресовке вполне предсказуемы [§8.1 и §8.2 RFC 4861]:

- она защищена механизмом GTSM от засылки извне канала, так что значение «предельного числа шагов» в заголовке IPv6 должно быть равно 255 — см. §5.1;
- в заголовке IPv6 адрес назначения не имеет права быть групповым, потому что иначе это явная атака на безопасность;

Это требование в RFC явно не присутствует, но доказать его нетрудно. Адрес назначения IPv6 переадресовки всегда равен адресу источника пакета-виновника [§8.2 RFC 4861], а тот ни при каких условиях не может быть групповым — мы говорили об этом в §3.2.

- в теле переадресовки адрес назначения Б не может быть групповым, так как переадресовку никогда не высылают в ответ на групповой пакет;
- хост-получатель проверяет с помощью своего кэша DC, что адрес источника IPv6 переадресовки совпадает с текущим адресом следующего шага для адреса назначения Б из тела переадресовки. Смысл этой проверки в том, что достоверная переадресовка могла прийти только от маршрутизатора, которым хост действительно пользовался;

Эта проверка также защищает от переадресовок, которые задержались в сети. Например, если хост А был переадресован маршрутизатором М1 к

маршрутизатору М2, а тем, с свою очередь, к маршрутизатору М3, то запоздавшая копия переадресовки М1 будет хостом А проигнорирована.

А как быть хосту, если от его текущего маршрутизатора по умолчанию пришла переадресовка вида «Б→Ш», но в кэше DC еще нет записи об адресате Б? Хотя стандарт признает такую переадресовку законной [§8.1 RFC 4861] и рекомендует создать новую запись «Б→Ш» [§8.3 RFC 4861], подобная переадресовка подозрительна, потому что она содержит сведения об адресате, которому хост в обозримом прошлом никаких пакетов не слал. Разобраться в этой ситуации хосту поможет копия пакета-виновника в теле переадресовки.

- пошаговая адресация маршрутизаторов происходит по их внутриканальным адресам [§8 RFC 4861], и поэтому:
  - в заголовке IPv6 адрес источника должен быть внутриканальным, так как переадресовка исходит от маршрутизатора данного канала;
  - в теле переадресовки адрес цели Ш должен или совпадать с адресом назначения Б (переадресовка на хост), или быть внутриканальным (переадресовка на другой маршрутизатор).

Под пошаговой адресацией мы имели в виду обращение к узлу, которое происходит, когда другой сосед использует его в качестве следующего шага пакета. Архитектура IPv6 самым настоятельным образом содействует тому, чтобы хост в этом случае обращался к маршрутизатору по его внутриканальному адресу.

Та же тенденция прослеживается и во взаимодействии между маршрутизаторами IPv6. Так, отношения смежности между маршрутизаторами IS-IS [§3 RFC 5308] и OSPF [§2.5 RFC 5340] устанавливаются именно по их внутриканальным адресам.

Пока хосты IPv6 обращаются к маршрутизаторам по внутриканальным адресам, хостам не придется делать исключения из правил «на канале» — «вне канала». Ведь внутриканальный префикс *FE80::/10* автоматически попадает в список префиксов «на канале». В противном же случае хост должен был бы постоянно «помнить», что адрес маршрутизатора по умолчанию находится «на канале». Это несложно, но немного нарушает стройность алгоритма передачи пакетов хостом.

Полученный формат переадресовки уже позволяет сообщить сам факт, что адрес назначения — «на канале». Но всегда ли этого достаточно? Допустим, хост А послал пакет адресату Б через маршрутизатор М, получил от маршрутизатора переадресовку вида «Б→Б» и сохранил это соответствие в своем кэше адресатов DC. Каковы его дальнейшие действия? Когда придет время послать следующий пакет в адрес Б, кэш адресатов DC укажет хосту А, что адрес Б — «на канале» и принадлежит соседу. То есть придет время для розыска соседа Б с помощью протокола ND. В свою очередь, протокол ND подразумевает, что сосед Б может получать групповые пакеты, рассылаемые хостом А.

Но что будет, если канал между А и Б — NBMA без группового вещания между хостами? Тогда на канальном уровне хост А мог бы передавать соседу Б индивидуальные кадры, если бы знал его канальный адрес. В то же время, групповой запрос NS не дойдет до Б. В этом случае розыск соседа закончится неудачей и обмен данными между А и Б окажется невозможным.

Чтобы обойти и эту особенность каналов NBMA, нам достаточно довести до логического завершения новую функцию маршрутизатора IPv6. Как мы уже отмечали, маршрутизатор IPv6 — это координатор передачи данных по каналу между хостами. Тогда именно он и должен сообщить хосту А недостающие сведения. Для этого достаточно дополнить переадресовку уже доступной нам опцией ND «канальный адрес цели» (§5.1) и поместить в нее канальный адрес узла Б.

Эта опция в переадресовке не помешает и при передаче по широковещательному каналу, так как хосту А не придется самостоятельно разыскивать соседа Б. Тем не менее,

для широковещательных каналов это просто оптимизация, тогда как для каналов NBMA это важный механизм, который позволяет IPv6 работать с такими каналами без их искусственной адаптации к широковещательной семантике.

Что хост А сделает с информацией из этой опции? Очевидно, он поместит ее в свой кэш соседей NC. Так как информация исходит от маршрутизатора и еще не подтверждает обоюдной доступности А и Б, подходящим состоянием для записи NC будет **ПРОСРОЧЕННАЯ**. Этой записью уже можно воспользоваться для непосредственной передачи пакета. Когда хост Б откликнется на этот пакет, то по сигналу вышестоящего протокола запись NC станет **ДОСТУПНОЙ**. Если же вышестоящий протокол — строго однонаправленный, то хосту Б все равно придется ответить на *индивидуальный* запрос NS с опцией «канальный адрес источника» (§5.1), который пошлет хост А, когда запись NC перейдет по тайм-ауту в состояние **ИСПЫТАНИЕ**.

> Обратите внимание: хост Б будет обязан пройти тут же самую процедуру переадресовки, чтобы получить право слать прикладные пакета напрямую хосту А, полагая того соседом «на канале».

Последний вопрос о работе этого механизма состоит в том, как маршрутизатор М узнает канальный адрес узла Б в отсутствие полноценного группового вещания. Конечно, последнее средство, пригодное для всех разновидностей NBMA — это ручная настройка кэша соседей М. Однако М сможет найти канальный адрес Б автоматически, если канал NBMA эмулирует групповое вещание по принципу веера, известное как «псевдошироковещание» (*pseudobroadcast*) [91]: групповые пакеты, рассылаемые М, доходят до всех узлов канала, а групповые пакеты, рассылаемые другими узлами канала, доходят как минимум до М. В этом случае М узнает канальный адрес Б, когда попытается передать узлу Б первый пакет от А и проведет розыск соседа. Кроме того, не исключено, что М уже знает канальный адрес Б, потому что хост Б сам проводил розыск соседа М. Ведь пришедший от хоста Б групповой запрос NS позволит М не только ответить информацией о себе, но и создать запись NC о соседе Б в состоянии **ПРОСРОЧЕННАЯ**, поскольку такой запрос NS обязательно содержит опцию «канальный адрес источника».

До сих пор мы сознательно ограничивались сценарием, когда канал обслуживает один маршрутизатор, а у каждого хоста один сетевой интерфейс, помимо «петли». Насколько трудно будет снять эти ограничения? Начнем с первого из них и рассмотрим канал с несколькими маршрутизаторами. Чтобы подключенный к тому же каналу хост смог пользоваться их услугами, ему, прежде всего, понадобится список их адресов вместо одного адреса маршрутизатора по умолчанию. Так возникает новая структура данных в памяти хоста: **список маршрутизаторов по умолчанию** (*Default Router List*). Сейчас мы допустим, что этот список указан вручную в начальных настройках хоста, а над его автоматическим заполнением мы поработаем чуть позже, в §5.4.2. Мы не станем усложнять нашу конструкцию системой приоритетов и положим, что все маршрутизаторы в списке эквивалентны.

Не имея оснований предпочесть какой-то конкретный маршрутизатор из списка, хост может начать работу, просто выбрав запись случайным образом. Этого уже будет достаточно, скажем, чтобы распределить нагрузку густозаселенного канала на несколько маршрутизаторов. Кроме того, дублирование маршрутизаторов повысит отказоустойчивость сети, если каждый хост сможет самостоятельно обнаружить, что его текущий маршрутизатор «пропал без вести», и выбрать из списка новый, «живой» маршрутизатор. Придется ли нам изобрести для этого новый протокол? К счастью, нет. Ведь в нашем распоряжении уже есть замечательный механизм NUD, для работы которого достаточно протокола ND — см. §5.1. Когда NUD сигнализирует о недоступности текущего маршрутизатора, хост может проверить работоспособность

[91] Using IP Multicast Over Frame Relay Networks. Cisco technology white paper. http://www.cisco.com/en/US/docs/ios/solutions_docs/ip_multicast/White_papers/frm_rlay.html

следующего по списку, и так далее. Совсем удалять недоступный маршрутизатор из списка, конечно же, не следует. Ведь вполне может оказаться, что к тому времени, когда последний маршрутизатор в списке засбоит, первый уже вернется в строй. На этом задача о нескольких маршрутизаторах решена.

> Конечно, поиск «живого» маршрутизатора открыт для оптимизации. К примеру, хост может активно следить за всеми маршрутизаторами в списке с помощью все того же механизма NUD и ранжировать их по состоянию записи в кэше соседей NC: **ДОСТУПНАЯ** запись лучше **ПРОСРОЧЕННОЙ**, а та предпочтительнее **НЕПОЛНОЙ**.
>
> Справедливости ради заметим, что §3.3.1.2 RFC 1122 требовал от хоста IPv4 обязательной поддержки нескольких маршрутизаторов по умолчанию. В то же время, §3.3.1.4 того же RFC 1122 признавал, что в IPv4 нет надежного и общепринятого способа узнать о недоступности текущего маршрутизатора. Этот конфликт между желаемым и действительным неизбежно сказался на практике IPv4, где список маршрутизаторов по умолчанию можно встретить редко.

Сможем ли мы так же легко и изящно снять второе ограничение и обеспечить эффективную работу хоста IPv6, подключенного к нескольким каналам? Для начала мы вновь обратим внимание на уже известный нам факт: отношение соседства определено только в пределах заданного канала. Иными словами, узел Б не может быть соседом узла А «вообще» — они будут соседями только на вполне определенном канале К. В свою очередь, окном в канал для хоста выступает сетевой интерфейс, а структуры данных хоста, которые управляют передачей пакетов в нашей модели, самым непосредственным образом работают с отношением соседства:

- запись NC содержит сведения о соседе;
- запись DC указывает на соседа — следующий шаг;
- маршрутизатор обязан быть соседом;
- префикс «на канале» содержится только в адресах соседей.

Поэтому все эти структуры привязаны к определенному сетевому интерфейсу и не имеют смысла для других интерфейсов того же хоста.[92] Пока хост обладает помимо «петли» ровно одним сетевым интерфейсом, эта связь может оставаться неявной; но, обзаведясь несколькими подключениями, хост должен вести отдельный экземпляр каждой структуры для каждого из своих интерфейсов, не имея при этом центральной структуры данных над ними. В результате у эталонного хоста IPv6 нет универсального критерия, по которому он смог бы предпочесть для передачи пакета один из своих интерфейсов. Этот выбор остается задачей других механизмов и протоколов [Приложение А RFC 4861].

> К примеру, «продвинутый» хост с несколькими подключениями может участвовать в протоколе маршрутизации и самостоятельно вычислять оптимальные направления передачи пакетов вместо того чтобы надеяться на маршрутизаторы по умолчанию.
>
> На первый взгляд, есть случаи, когда и наш эталонный хост может сам выбрать оптимальный выходной интерфейс. Например, вот типичное заблуждение: «Если адресат исходящего пакета — «на канале» на одном из интерфейсов, то надо выбрать именно этот интерфейс». Но как быть, если этот канал засбоил и в данный момент адресат доступен только через маршрутизатор другого канала? Дело в том, что множественное подключение применяют для решения довольно сложных задач, таких как отказоустойчивость высокой категории, и простые решения здесь не срабатывают. В то же время базовые механизмы хоста IPv6 должны оставаться простыми, чтобы они были по силам не только серверу, но и кофеварке.

Мы же предположим, что у эталонного хоста IPv6 в каждый момент времени есть один выходной интерфейс по умолчанию, который и используется для передачи пакетов.

[92] Если только эти интерфейсы не подключены к тому же самому каналу.

Теперь нам пора свести воедино наши соображения о том, каков принципиальный алгоритм передачи пакета эталонным хостом IPv6 [§5.2 RFC 4861]. Вызов этого алгоритма будет происходить на каждый исходящий пакет. Входными данными алгоритма послужат сетевой адрес назначения пакета и требуемый выходной интерфейс, а вернуть он должен канальный адрес следующего шага, или же ошибку, если следующий шаг найти не удалось.

Первое решение хоста состоит в том, является ли адрес назначения индивидуальным или групповым. Определить это просто благодаря тому, что все групповые адреса имеют префикс *FF00::/8*. Дальнейшая работа над групповым адресом для нас сейчас интереса не представляет, так как, согласно нашему плану, групповые адреса IPv6 всегда «на канале», а канальный групповой адрес просто *вычисляется* из сетевого группового адреса по правилам инкапсуляции IPv6 для каналов данного типа. Например, в *Ethernet* это уже знакомая нам элементарная операция: взять 32 младших бита группового адреса IPv6 и присоединить к ним префикс `33-33`. Поэтому сосредоточим наше внимание на индивидуальных адресах назначения.

Далее хосту надо найти сетевой адрес следующего шага. Здесь хост пытается сэкономить время, применяя кэш адресатов DC, привязанный к выходному интерфейсу. Если адрес назначения уже содержится в левой части записи DC, то правая часть этой записи сразу же отвечает на два вопроса: явным образом она сообщает сетевой адрес следующего шага, а неявно говорит, «на канале» ли адресат. Впрочем, хосту уже неважен ответ на второй вопрос, потому что он знает ответ на первый.

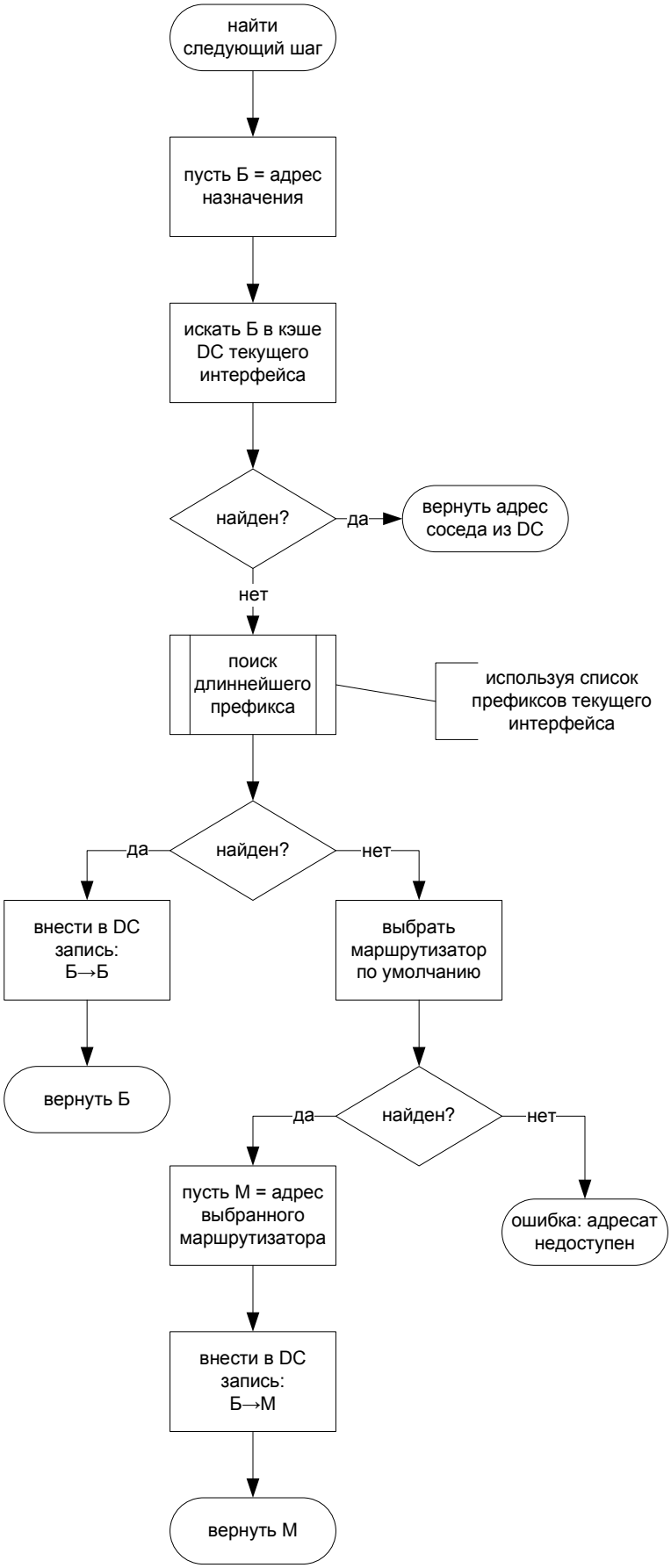
Если же записи об адресе назначения в кэше DC нет, хост должен определить, «на канале» ли адресат. Здесь безопаснее ошибочный диагноз: «Вне канала», — потому что в этом случае пакет уйдет через маршрутизатор. Напротив, ошибочный диагноз: «На канале», — приведет к невозможности передать пакет. Поэтому хост сопоставляет адрес назначения со списком префиксов выходного интерфейса и делает вывод: «На канале», — только если адрес назначения содержит хотя бы один из этих префиксов. В противном же случае хост обязан действовать так, как если бы адресат был «вне канала».

> Действующий стандарт ND предписывает поиск длиннейшего префикса [§5.2 RFC 4861], хотя эта информация здесь явно избыточна. Ведь даже если список префиксов содержит более короткий префикс наряду с содержащим его префиксом большей длины, результат проверки не зависит от того, какой именно префикс был обнаружен в адресе назначения. Допустим, например, что список префиксов состоит всего из двух пунктов: *2001:DB8::/48* и *2001:DB8::/64*. Тогда адрес *2001:DB8::1234* будет «на канале» вне зависимости от того, какой из префиксов «сработал» при проверке. Информация о длиннейшем префиксе могла бы понадобиться, если бы надо было распространить поиск на несколько сетевых интерфейсов и выбрать интерфейс с самым точным префиксом. Однако выбор выходного сетевого интерфейса сознательно не включен в современную версию ND. По-видимому, поиск длиннейшего префикса в стандарте ND остался с тех пор, когда поддержка нескольких интерфейсов на уровне ND была предметом исследования. Кроме того, поиск самого точного префикса может быть нужен, если хост ведет статистику для каждого префикса на канале: счетчик числа обращений и т.п.

Если по итогам этой проверки адрес назначения находится «на канале», то сетевой адрес следующего шага совпадает с адресом назначения. В противном случае хост выбирает кандидата из списка маршрутизаторов по умолчанию, например, руководствуясь сведениями NUD. Окончательный результат помещается в кэш DC.

> Если сведений NUD ни об одном маршрутизаторе нет, то достаточно перебирать список маршрутизаторов по кругу [§6.3.6 RFC 4861]. Тогда довольно скоро в кэше NC появится актуальная информация о доступности хотя бы некоторых маршрутизаторов.

В этом и состоит первая часть эталонного алгоритма передачи пакета хостом IPv6: выбор следующего шага (маршрутизация). Давайте представим его в виде блок-схемы Фиг. 71.



Фиг. 71. Алгоритм маршрутизации выходного пакета эталонным хостом IPv6

Теперь сетевой адрес следующего шага выбран, и слово за модулем ND, чьей задачей будет выяснить отвечающий ему канальный адрес. Процесс ND тоже старается сэкономить время и силы, на этот раз с помощью кэша соседей NC при выходном интерфейсе. Если данный адрес уже закэширован и запись о нем полная, то есть содержит канальный адрес, то достаточно вернуть последний. Конечно, сам факт обращения к записи NC может включить фоновый механизм ее освежения согласно NUD, например, если запись **ПРОСРОЧЕННАЯ**. Тем временем алгоритм передачи IPv6 успешно завершает работу, так как любой полной записью NC можно немедленно пользоваться для передачи пакета.

И, наконец, если полной записи о следующем шаге в кэше NC еще нет, то хосту остается только поместить пакет в буфер и начать процедуру розыска соседа. В результате ее работы возникнет полная запись NC, если искомый узел доступен, конечно. Так задача сведется к предыдущей, когда запись NC уже есть.

Какими любопытными свойствами обладает этот алгоритм? С одной стороны, он вполне способен распределить нагрузку на несколько маршрутизаторов [RFC 4311], но при этом он не страдает от недостатков случайного распределения, потому что кэширует следующий шаг для каждого адресата.

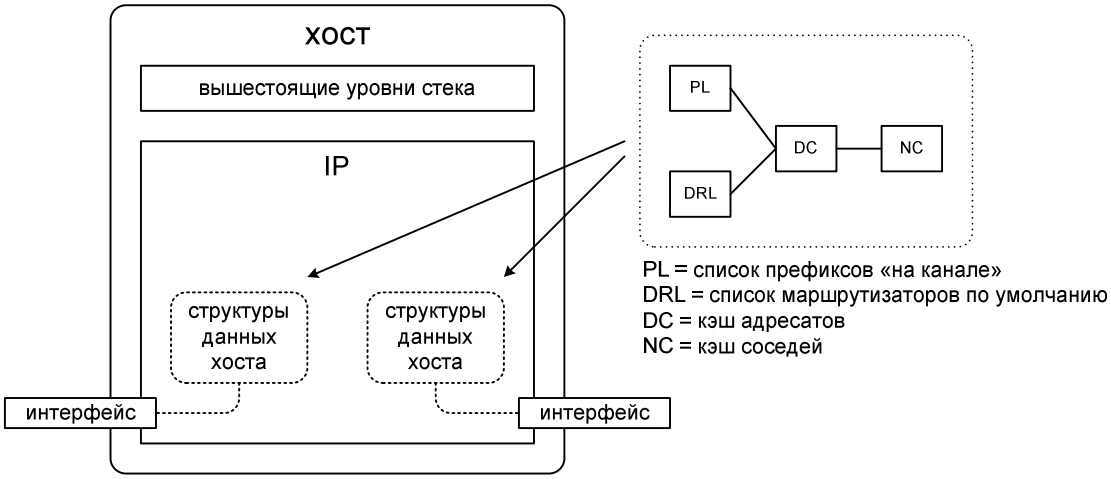
> Проблемы, которые влечет за собой бессистемное распределение пакетов по разным путям, мы уже упомянули в §5.1.

С другой стороны, этот алгоритм не обременен мгновенной проверкой доступности следующего шага. Допустим, хост до некоторых пор пользовался одним маршрутизатором, а затем его маршрутизатор-фаворит неожиданно взяли и выключили. Тогда алгоритм передачи хоста продолжит возвращать канальный адрес выключенного маршрутизатора до тех пор, пока фоновый механизм NUD не сигнализирует о его недоступности и пока не исчезнут указывающие на него записи кэша DC. И только после этого произойдет полное переключение на новый маршрутизатор.

Конечно, это компромисс между надежностью и производительностью: маршрутизаторы не исчезают ежесекундно, тогда как проверять доступность маршрутизатора перед каждым пакетом было бы накладно. К тому же, благодаря непрерывной обратной связи с вышестоящими протоколами, механизм NUD может установить свои тайм-ауты весьма короткими и обнаружить пропажу соседа в считанные секунды после того, как прекратятся сигналы вышестоящих протоколов о его доступности.

Дальнейшая оптимизация состоит в том, чтобы удалить все записи кэша DC, которые ссылаются на пропавшего соседа, как только механизм NUD обнаружил его недоступность. Это вызовет пересмотр следующего шага для всех активных адресатов, ранее доступных через пропавшего соседа.

Кроме того, если пропавший сосед — маршрутизатор по умолчанию, то надо на какое-то время исключить его из кандидатов для алгоритма передачи. Например, при циклическом обходе списка маршрутизаторов достаточно перейти к следующему элементу в нем. Тогда пропавший маршрутизатор окажется в конце списка и хост выберет его в последнюю очередь; у бедняги будет время на то, чтобы прийти в себя.



Фиг. 72. Структуры данных эталонного хоста IPv6 и связь между ними

Как мы видим, структуры данных эталонного хоста IPv6 не только привязаны к определенному сетевому интерфейсу, но и логически связаны между собой, как это схематически показано на Фиг. 72.

Прямая связь заключается в следующем:

- хост использует список маршрутизаторов по умолчанию и список префиксов «на канале», чтобы выбрать следующий шаг пакета, и сохраняет свой выбор как запись в кэше DC;
- запись в кэше DC указывает на сетевой адрес следующего шага для данного адреса назначения;
- затем хост проводит розыск соседа для адреса следующего шага и сохраняет результат в кэше NC;
- в результате хост получает возможность передать пакет по определенному канальному адресу.

Обратная связь не менее важна и сводится вот к чему:

- механизм NUD в фоне следит за состоянием соседей, записи о которых есть в кэше NC хоста;
- если NUD обнаруживает, что сосед стал недоступен, то это повод удалить из кэша DC все записи, ссылающиеся на него как на следующий шаг;

Это позволит хосту адаптироваться как к сбою маршрутизатора, так и к изменению топологии канала, когда бывший сосед становится доступен только через маршрутизатор.

- кроме того, если сосед — маршрутизатор по умолчанию, хосту следует принять его сбой во внимание при выборе следующего шага для адресатов «вне канала».

Как именно хост примет во внимание недоступность маршрутизатора, зависит от его алгоритма управления списком маршрутизаторов по умолчанию. Принципиальный алгоритм [§6.3.6 RFC 4861] рекомендует только, чтобы, во-первых, заведомо или вероятно доступные маршрутизаторы получали приоритет и, во-вторых, в отсутствие заведомо или вероятно доступных маршрутизаторов список перебирался по кругу. Смысл этого алгоритма в том, чтобы не выбрать снова сбойный маршрутизатор до тех пор, пока хост не попытает счастья со всеми остальными пунктами списка.

Сведения о доступности маршрутизаторов поставляет механизм NUD, и может так оказаться, что в данный момент полной записи NC ни об одном из них нет. Это вовсе не значит, что все маршрутизаторы недоступны, — просто стали недоступны те из них, которыми хост до сих пор пользовался. Это повод проверить на работоспособность остальные маршрутизаторы в списке. Отсюда и возникает перебор списка по кругу.

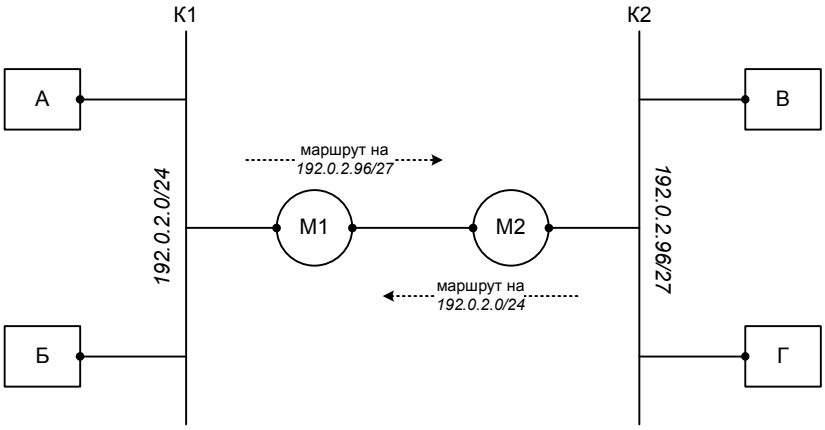
Подведем итоги. В §5.1 мы «сконструировали» механизм розыска соседей, однако еще не сформулировали правил, когда именно узел IPv6 вправе считать удаленный адрес соседским и поводить его розыск. А розыск соседа, который соседом не является — это все равно, что охота на несуществующую черную кошку в темной комнате. Поэтому сейчас нам пришлось целенаправленно поработать над этим вопросом. В процессе мы пересмотрели простую модель канала и подсети, принятую в IPv4, и признали ее слишком ограниченной и не отвечающей требованиям современных канальных технологий. Взамен мы сформулировали для IPv6 принципиально иную модель, в которой маршрутизаторы координируют передачу данных хостами по каналу и за его пределы, а хосты, в свою очередь, обращаются к маршрутизаторам за сведениями о топологии канала, а не пытаются самостоятельно вычислить ее. Затем мы «создали» довольно простой протокол в рамках ND, который обеспечивает работу канала IPv6 согласно этой модели.

> Исторической справедливости ради заметим, что хост IPv4 тоже мог следовать подобной линии поведения, когда у него не было полноценной таблицы маршрутов, а вместо нее были список маршрутизаторов по умолчанию и кэш маршрутов, уточняемый переадресовками [§3.3.1.2 RFC 1122]. Тем не менее, критерий соседства IPv4 был задан вполне однозначно и отвечал модели, когда все адреса в подсети, и только они, принадлежат соседями [§3.3.1.1 RFC 1122].

Как вы думаете, насколько оригинальна наша модель? Конечно же, ничто не ново под луной, и на ту же самую модель давно опирается стек протоколов ISO. В частности, взаимодействие хостов (конечных систем, ES) и маршрутизаторов (транзитных систем, IS) в нем обеспечивает протокол ES-IS [ISO 9542; предыдущая версия доступна как RFC 995].

> Подход ES-IS к решению данной задачи можно даже признать более изящным и логически завершенным. Он не сводится к одним только переадресовкам, а включает в себя периодические объявления, которыми ES и IS извещают друг друга о своем существовании. Поэтому некое подобие розыска соседей в ES-IS необходимо, только когда на канале нет ни одной IS. Даже в этом случае ES-источник не тратит время на предварительный розыск соседа, а просто шлет пакет с данными по групповому канальному адресу «все ES». ES-адресат, если он действительно подключен к каналу, принимает пакет в обработку и заодно высылает ES-источнику объявление, содержащее его индивидуальный канальный адрес. Тогда ES-источник кэширует канальный адрес соседа, и последующие пакеты уже идут по нему, а не группе «все ES».
>
> Таким образом, ES по умолчанию полагает, что все адреса «на канале». В противоположность этому, хост IPv6 обязан по умолчанию считать, что «вне канала» находятся все адреса, кроме внутриканальных. Когда нет ни маршрутизатора, ни ручной настройки списка префиксов, хосты IPv6 могут обмениваться индивидуальными пакетами, только используя внутриканальные адреса, так как в этих условиях все остальные индивидуальные адреса находятся «вне канала» и потому недоступны.
>
> Любопытно, что ранние стандарты IPv6 следовали модели ES-IS, в которой все адреса по умолчанию были «на канале», однако впоследствии от нее полностью отказались [RFC 5942], потому что за ее внешней простотой скрываются различные осложнения [RFC 4943].
>
> Для общего знакомства со стеком протоколов ISO и, в частности, с протоколом ES-IS мы настоятельно рекомендуем замечательную книгу Ради Перлман [60].

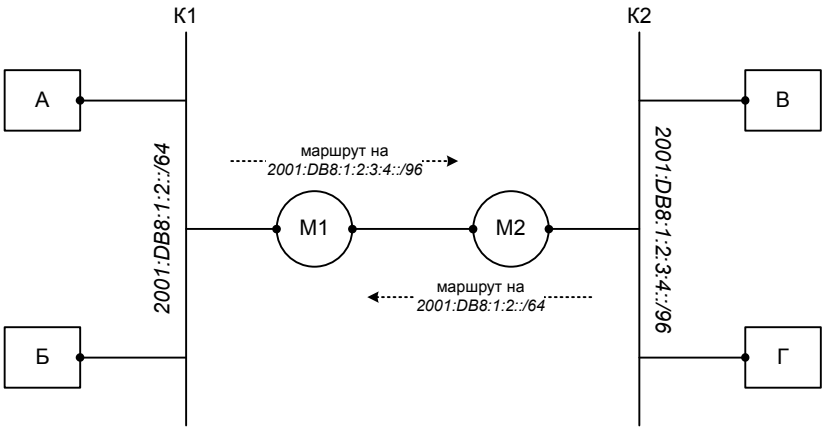
В завершение рассмотрим любопытный пример того, как отличается практика управления подсетями в IPv4 и IPv6. Как мы помним, в IPv4 была вполне допустима конфигурация сети, когда из подсети был «вырезан» префикс большей длины и назначен другому каналу. Скажем, префикс *192.0.2.0/24* назначен каналу К1, а *192.0.2.96/27* — каналу К2, как это показано на Фиг. 73.



Фиг. 73. Удаленный под-префикс в IPv4

Маршрутизаторы М1 и М2 естественно работали с такой конфигурацией, потому что префикс канала К2 был точнее, чем префикс канала К1, и диапазон адресов *192.0.2.96–192.0.2.127* однозначно маршрутизировался в направлении канала К2, тогда как остальные адреса вида *192.0.2.X* оставались на канале К1. Тем не менее, хосты канала К1 не могли передавать пакеты хостам канала К2, потому что они разыскивали их на канале К1, вместо того чтобы обратиться к маршрутизатору М1. Обычным решением этой проблемы было или включить ARP proxy на маршрутизаторе М1, или распространить точный маршрут на *192.0.2.96/27* через М1 по всем хостам канала К1, вручную или протоколом маршрутизации.

Как бы мы стали решать ту же проблему в IPv6? Допустим, каналу К1 назначена подсеть *2001:DB8:1:2::/64*, а каналу К2 — *2001:DB8:1:2:3:4::/96*, как показано на Фиг. 74. Прежде всего, мы не можем не заметить, что длина префикса подсети, отличная от 64, нарушает адресную политику IPv6 из §2.6, хотя и не вступает в противоречие с базовым протоколом, так что назначение такого префикса остается на совести администратора сети.



Фиг. 74. Удаленный под-префикс в IPv6

Что же касается технического решения, то здесь мы видим, как минимум, три альтернативных подхода на основе стандартных механизмов IPv6:

- Во-первых, можно оставить списки префиксов хостов канала К1 пустыми, не считая внутриканального префикса, и тогда хосты будут сначала обращаться к маршрутизатору М1, а тот уже при необходимости переадресует их к соседу.
- Во-вторых, вполне допустимо считать весь префикс *2001:DB8:1:2::/64* «на канале» и включить на маршрутизаторе М1 ND proxy для префикса *2001:DB8:1:2:3:4::/96*.
- В-третьих, можно составить и распространить по хостам канала К1 сложный список префиксов, который будет эквивалентен *2001:DB8:1:2::/64* за вычетом *2001:DB8:1:2:3:4::/96*.

Пусть читатель найдет способ составить такой список префиксов, используя минимально возможное число элементов. (Начните с доказательства того, что минимально необходимое число префиксов в таком списке равно 32. Затем попробуйте доказать, что кратчайшее решение — единственное. По ходу вы поймете, как устроен этот список.)

5.3. Пересмотр типов вещания IPv6. Вещание наугад — anycast

Поработав над розыском соседей, мы можем пересмотреть типы вещания в IPv6. Прежде всего, что произошло с широковещанием? До сих пор мы старательно избегали этой темы. Однако теперь мы убедились, что необходимость в широковещании полностью отпала благодаря обязательной поддержке группового вещания. Широковещание IPv4 опиралось на широковещание канального уровня, а с ним мы активно боремся, чтобы не расходовать впустую ресурсы сети. Обратиться же ко всем узлам IPv6 в определенной зоне всегда можно с помощью общепринятой группы «все узлы» (*FF0x::1*). Итак, в IPv6 нет широковещания; его функции выполняет групповое вещание.

А когда нет широковещания, то не нужны и особые широковещательные адреса. В частности, идентификатор интерфейса *FFFF:FFFF:FFFF:FFFF* (все биты установлены) никоим образом не зарезервирован.

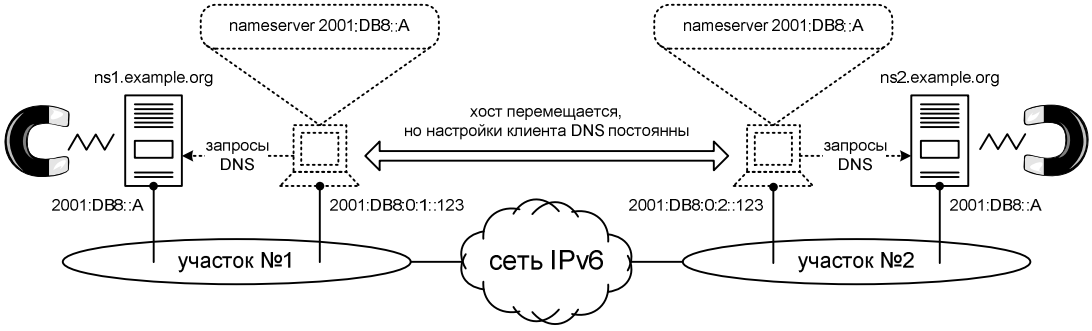
Это не значит, что такой идентификатор можно свободно использовать. Чтобы понять, какие с ним связаны сложности, проанализируйте его с позиций модифицированного формата EUI-64 (§2.7). (Подсказка: обратите внимание на значения битов **U/L** и **I/G** в нем.)

Групповое вещание полезно для протоколов не только служебных, но и прикладных. Оно выручает в тех случаях, когда один пакет надо с минимальными затратами разослать нескольким узлам. Например, сервер NTP может передавать «сигналы точного времени» сразу группе клиентов; клиент DHCP может обратиться одним запросом к группе серверов и получить несколько предложений, а затем выбрать среди них наиболее подходящее.

Однако возникают похожие задачи, которые групповое вещание решить не может. К примеру, нет смысла создавать в организации группу серверов DNS хотя бы потому, что протокол DNS рассчитан на индивидуальное вещание и клиенту не будет никакой выгоды от нескольких ответов.

Искушенный читатель возразит, что DNS плюс групповое вещание — это протокол mDNS. Однако последний решает немного иную задачу, а именно он позволяет находить имена в локальной сети, где вообще нет сервера DNS.

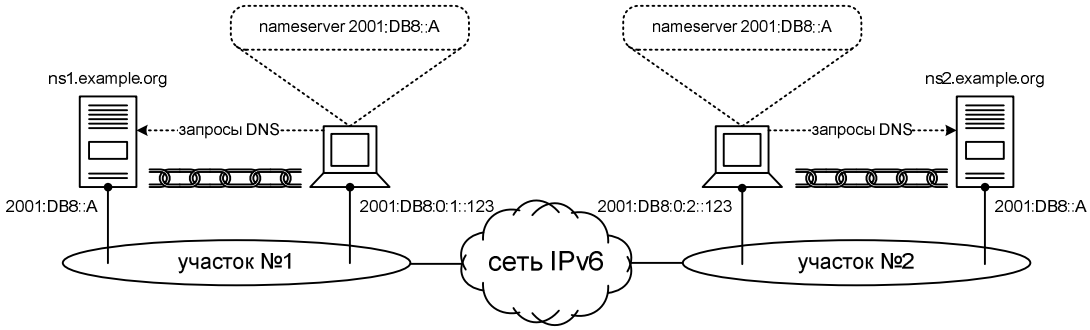
Также групповое вещание совершенно бессильно в протоколах, работающих поверх TCP. Тем не менее, сама концепция сетевой группы, когда за одним адресом стоят несколько узлов (точнее, интерфейсов), сохраняет свое значение и в этом случае. Скажем, в крупной организации можно было бы назначить всем серверам DNS или SMTP один «волшебный» адрес *X*, так чтобы клиент, независимо от местоположения в сети и не меняя настроек, всегда обращался бы к ближайшему серверу (см. Фиг. 75). С точки зрения источника пакета, это будет **вещание наугад** (*anycast*), по принципу «кому попало».



Фиг. 75. Идея anycast для DNS

Тем не менее, роль случайности в этом типе вещания не так велика, как кажется источнику. Для простых протоколов, где весь сеанс состоит из обмена парой пакетов UDP по схеме «запрос-ответ», действительно неважно, в какой именно узел anycast попадет запрос. Между тем, TCP не играет в кости: для его работы надо, чтобы пакеты данного соединения попадали в один и тот же узел anycast хотя бы при условии постоянной конфигурации сети. Получается оксюморон: вещание наугад должно быть воспроизводимым (см. Фиг. 76).

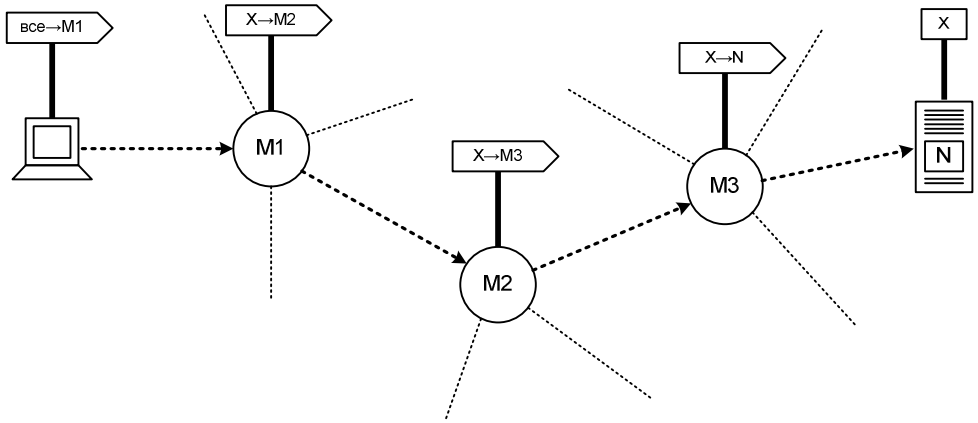
> Примером простого протокола «запрос-ответ» мог бы служить DNS поверх UDP, если бы его стороны вообще не поддерживали транспорт «виртуальная цепь» при поддержке TCP [RFC 5966]. Но, как мы помним, поддержка этого транспорта важна для передачи длинных сообщений DNS. Так что даже DNS требует от вещания наугад воспроизводимости.
>
> И даже простейший протокол вида «запрос-ответ» потребует воспроизводимого вещания наугад, если длина его полностью сформированного пакета может превышать минимальное значение MTU. Дело здесь, конечно же, во фрагментации и сборке. Ведь после фрагментации каждый фрагмент представляет собой отдельный пакет и проходит путь к адресату независимо от других фрагментов, а встретиться они должны на одном и том же узле назначения, чтобы сборка прошла успешно.



Фиг. 76. Anycast должен быть воспроизводимым

Как нам выполнить это противоречивое условие? Давайте попробуем свести задачу к двум крайним случаям. В первом из них источник и узлы anycast будут разбросаны по сети, так что они не окажутся соседями друг друга. Во втором же случае источник и все узлы anycast будут соседями по каналу.

В первом случае задачу можно решить обычной настройкой маршрутов в сети. Для простоты положим, что источник — это хост, у которого есть только маршрутизатор по умолчанию. По условию задачи, источник — не сосед ни одному из узлов anycast, а значит, первым шагом пакет попадет на маршрутизатор по умолчанию. Если у того будет маршрут для адреса anycast *X* в сторону ближайшего узла anycast *N*, то следующим шагом пакет достигнет его или хотя бы приблизится к нему и т.д. То есть достаточно выстроить на маршрутизаторах цепочку маршрутов, которая приведет пакет к ближайшему узлу anycast, как показано на Фиг. 77. Принципиально это ничем не отличается от процесса маршрутизации в сложной сети, где к данному адресату можно проложить более одного пути и потому пакеты от разных источников ходят к нему через совершенно разные участки сети. Следовательно, и осуществить такую схему можно теми же инструментами: статическими маршрутами или протоколом динамической маршрутизации.



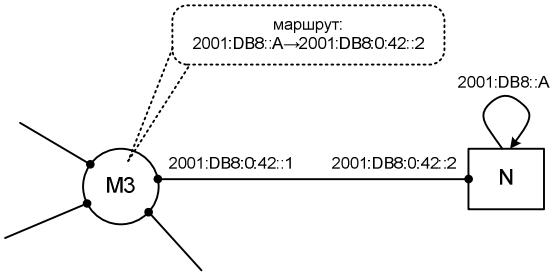
Фиг. 77. Трасса к адресу anycast

Все так просто, пока у каждого маршрутизатора один маршрут на данный адрес anycast. Если же маршрутизатор поддерживает распределение нагрузки по нескольким путям, то придется каким-то образом позаботиться, чтобы все пакеты одного сеанса приходили на один и тот же узел anycast. Однако мы знаем, к тому же, что пакеты одного сеанса должны идти по одному пути во избежание нежелательных побочных эффектов, таких как переупорядочение. В точности та же задача возникает и при обработке индивидуального трафика, так что маршрутизатору не придется делать исключений для пакетов anycast. Например, будет достаточно выбрать алгоритм балансировки, в котором выбор маршрута есть однозначная функция от адреса источника пакета. А это автоматически обеспечит попадание пакетов одного сеанса в один и тот же узел anycast.

Тогда вопрос для нас представляет только финал, когда последний маршрутизатор в цепочке передает пакет непосредственно узлу anycast. Здесь возможны два очевидных подхода.

Согласно первому из них, маршрутизатор должен полагать, что адрес *X* — «на канале». Как сообщить эту информацию маршрутизатору, зависит от реализации. К примеру, можно назначить каналу между маршрутизатором и узлом anycast префикс *X*/64. Так или иначе, маршрутизатор проведет розыск адреса *X*, а узел anycast откликнется и получит пакет.

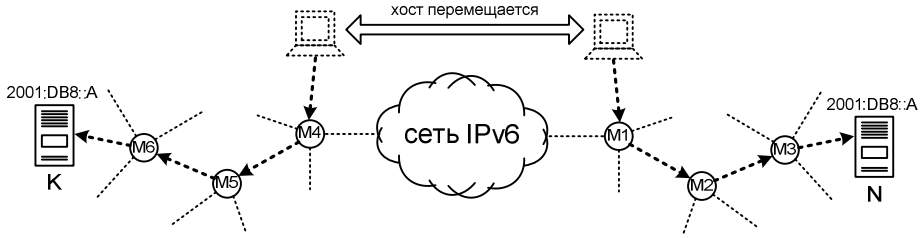
Второй подход состоит в том, чтобы каждому узлу anycast *J* назначить, помимо адреса *X*, обычный индивидуальный адрес A_J. Тогда в таблице маршрутов последнего маршрутизатора достаточно будет создать маршрут $X \to A_N$, и по нему пакет достигнет узла anycast *N*. В этой схеме адрес A_J вполне может быть внутриканальным, а хотя бы один такой адрес — это обязательный атрибут всякого интерфейса IPv6 (см. §2.5); так что назначать дополнительные адреса только ради работы anycast не придется. Сам же адрес *X* можно назначить любому сетевому интерфейсу узла, включая «петлю», как это показано на Фиг. 78.



Фиг. 78. Последний шаг к адресу anycast

Когда же источник переместится в другую часть сети, как показано на Фиг. 79, там у него будет новый маршрутизатор по умолчанию, с которого начнется иная цепочка маршрутов к ближайшему узлу anycast *K*. Так как цепочка маршрутов зависит только от

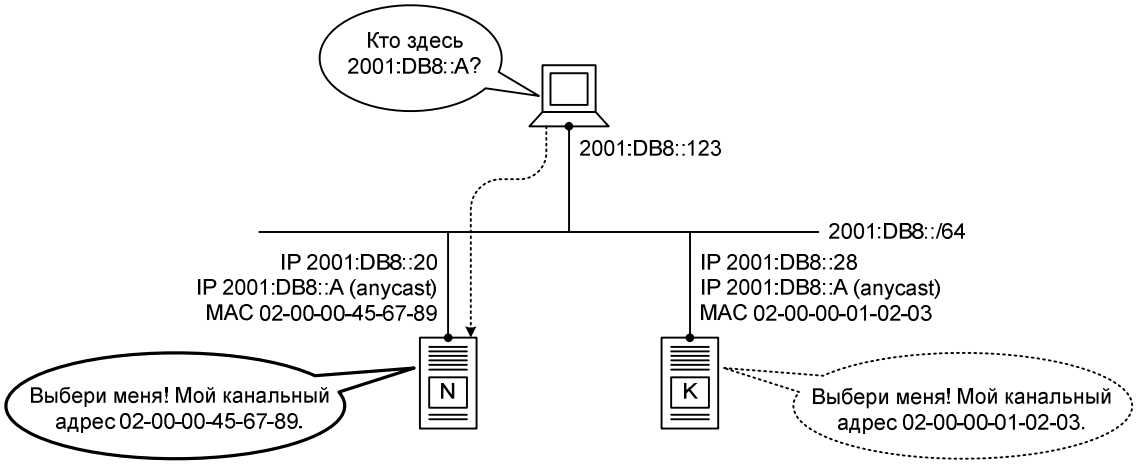
точки подключения источника, вещание наугад будет воспроизводимым, пока источник находится на одном месте, а это нас вполне устраивает.



Фиг. 79. Сеть выбирает узел anycast в зависимости от точки подключения хоста

Перейдем теперь ко второму случаю, когда источник и все узлы anycast — соседи по какому-то каналу. В этом случае трудно сказать, какой из узлов anycast ближе к источнику, так что пусть это будет настоящее вещание наугад, когда первичный выбор узла anycast происходит случайно. Тем не менее, после случайного выбора должен идти воспроизводимый этап передачи пакетов. Решим ли мы и эту задачу уже доступными нам средствами?

Допустим, что источник по-прежнему не подозревает об особенностях адреса *X* и работает с ним, как с обычным индивидуальным адресом. По условию задачи, адрес *X* — «на канале». Поэтому источник первым делом проведет розыск соседа *X*, направив вызов NS группе искомого узла Г(*X*), как мы обсудили в §5.1. Если наши узлы anycast заранее вступят в эту группу, то все они, в идеале, получат NS и ответят на него объявлениями NA. В результате источник получит несколько объявлений NA об одном и том же сетевом адресе цели *X*, но все они будут с разными канальными адресами, как показано на Фиг. 80.



Фиг. 80. Anycast в пределах канала

Потенциально эти объявления могут состязаться за запись в кэше NC источника, меняя ее непредсказуемым образом. Поэтому нам надо позаботиться, чтобы узлы anycast не воевали между собой. Гарантией мира будет сброшенный флаг **O** (**O** = 0) в их объявлениях NA насчет адреса цели *X*. Тогда в кэше NC закрепится канальный адрес из объявления, пришедшего самым первым. Последующие объявления станут переводить запись в состояние **ПРОСРОЧЕННАЯ**, но она будет возвращаться в состояние **ДОСТУПНАЯ**, не меняя канального адреса, благодаря процедуре NUD. Таким образом, вещание наугад будет воспроизводимым и в этом случае.

> Обратите внимание на прямое сходство между механизмами ND proxy и anycast [§7.2.8 RFC 4861].

Если на канале будет много источников вещания наугад и все они обратятся по адресу *X*, то их обращения распределятся между узлами anycast случайным образом. Так вещание наугад можно использовать для распределения нагрузки на узлы.

Пока что в нашей схеме узлы anycast отвечают на каждый вызов NS «хором», а это приведет к пикам нагрузки на канал. Чтобы сгладить эти пики, пусть каждый узел anycast

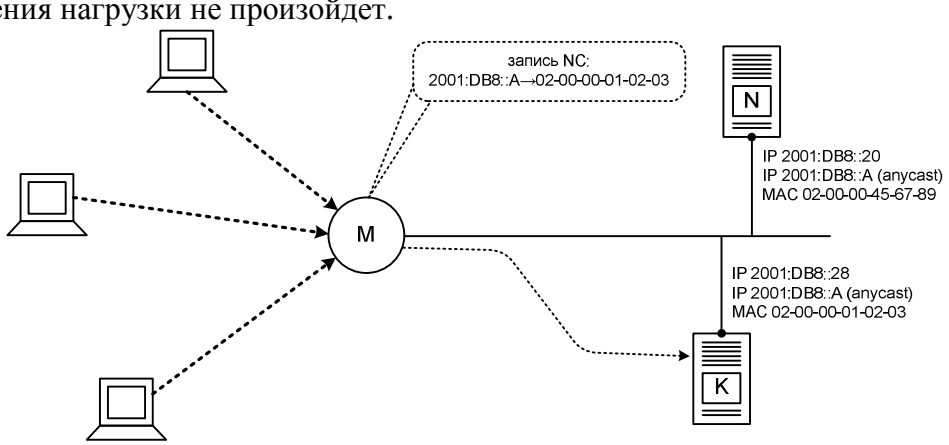
задерживает объявление NA на случайное время, от 0 до заданного «потолка»[93] [§7.2.4 RFC 4861].

По-хорошему, параметр **MAX_ANYCAST_DELAY_TIME** должен быть меньше параметра **RETRANS_TIMER**, чтобы оставить немного времени на передачу по каналу, но в §10 RFC 4861 их значения совпадают.

С другой стороны, если бы узлы anycast отвечали на вызов NS без искусственной задержки, это позволило бы источнику выбрать ближайший или наименее загруженный узел, так как выигрывает ответ NA, пришедший первым.

А что произойдет, если выбранный узел anycast через какое-то время станет недоступен, скажем, его просто выключат? Тогда источник вычистит запись об адресе *X* из NC благодаря NUD, и цикл розыска повторится с начала. В результате источник переключится на другой, доступный узел anycast. Так оказывается, что вещание наугад в рамках одного канала открывает путь не только к распределению нагрузки, но и к резервированию.

Итак, мы рассмотрели два независимых сценария с вещанием наугад. Можно ли их объединить, чтобы охватить все возможные случаи? Конечно! Если мы поместим между источником и каналом с узлами anycast маршрутизатор, как показано на Фиг. 81, то выбирать узел anycast придется маршрутизатору, но он с этой задачей вполне справится. Ведь выбор узла anycast по адресу ничем не отличается от обычного розыска соседа. Однако у гибридного сценария есть важное отличие от сценария «все на одном канале»: если несколько источников станут обращаться по адресу anycast через один и тот же маршрутизатор, то все обращения пойдут на один и тот же узел anycast. В этом случае распределения нагрузки не произойдет.



Фиг. 81. Anycast на канале через маршрутизатор

Остался последний вопрос касательно вещания наугад: какова структура «волшебного» адреса anycast? Возникнет ли у нас новый тип адреса IPv6? Нет, не возникнет. «Построенная» нами схема вещания наугад в IPv6 явно требует, чтобы адрес anycast ничем не отличался от индивидуального адреса. Только при этом условии вещание наугад можно вести на основе уже доступных механизмов, маршрутизации и ND. Узнать, что данный адрес — anycast, можно только из настроек узла, которому он назначен, и только такой узел работает с адресом anycast чуть иначе.

«Разрабатывая» вещание наугад, мы почти не ограничивали себя деталями IPv6. Поэтому вещать наугад по той же схеме можно и в среде IPv4, с точностью до замены ND на ARP [RFC 1546]. Вещание наугад уже пробовали использовать в DNS [RFC 3258].

О назначении общепринятых адресов anycast в IPv6 см. §6.1 и [RFC 2526].

Некоторые реализации не поддерживают передачу пакетов с адресом источника anycast [[94], [95]], хотя такие пакеты вполне имеют право на существование,

[93] Параметр **MAX_ANYCAST_DELAY_TIME**, 1 с [§10 RFC 4861].

если они идут в ответ, а не открывают сетевой диалог. Единственная их проблема в том, что адрес источника потенциально указывает на более чем один сетевой интерфейс. Тем не менее, в каждом конкретном случае эта неоднозначность устраняется теми же механизмами, что обеспечивают воспроизводимости anycast.

Обсудите самостоятельно, можно ли считать применение адреса обратной связи *::1* особым случаем anycast.

Можно подвести итоги раздела. В IPv6 исчезло широковещание, но зато появилось вещание наугад. Индивидуальный и групповой режимы вещания продолжают играть свои роли, причем групповое вещание IPv6 также выполняет те важные функции, которые традиционно были свойственны широковещанию.

Как показывает практика, реализации группового вещания на канальном уровне зачастую менее отлажены, чем остальной код, и потому они далеко не всегда надежны. Не стоит сомневаться, что «глюки» канального группового вещания неизбежно выливаются в труднодиагностируемые проблемы на уровне IPv6. Поэтому производителям канального оборудования придется хорошенько потрудиться над тем, чтобы их продукция стала действительно пригодна к работе в эпоху IPv6.

5.4. Автоматическая настройка IPv6

5.4.1. Выбор идентификатора интерфейса и обнаружение адресных конфликтов

Давным-давно существовал протокол IPX, в чем-то похожий на IPv6.[96] Адрес IPX тоже состоял из двух частей фиксированной длины, отвечающих каналу и интерфейсу. Интересной особенностью адреса IPX было то, что идентификатор интерфейса *всегда* совпадал с адресом MAC-48 того же интерфейса. Благодаря этому, упрощались два аспекта работы сети. Во-первых, розыск соседей становился тривиальной операцией: достаточно было выделить в адресе IPX идентификатор интерфейса, чтобы получить канальный адрес искомого соседа. Во-вторых, при настройке сети IPX не приходилось назначать каждому интерфейсу уникальный адрес, а достаточно было назначить идентификатор (префикс) всему каналу, потому что об уникальности адресов MAC — глобальной! — уже позаботился IEEE. Платой за эту простоту была жесткая связь между канальными и сетевыми адресами. Так, например, при смене сетевого адаптера у узла менялся адрес IPX.

Если адаптер позволяет программировать свой адрес MAC, то можно было бы сохранить старый адрес, однако это означало бы переход к самостоятельному управлению адресами MAC и — прощай, простота IPX! В действительности, реализации даже позволяли выбирать адрес IPX на свой вкус, но это вызывало программную смену связанного с ним адреса MAC.

Конечно, еще одно очевидное ограничение IPX состояло в том, что он мог работать только поверх каналов с адресацией MAC-48 или совместимой.

К счастью, ограничения на идентификатор интерфейса IPv6 гораздо мягче, чем это было в IPX. Единственное условие, на котором мы настаиваем, — это чтобы идентификатор интерфейса в индивидуальном адресе IPv6[97] был длиной 64 бита и отвечал формату модифицированного EUI-64. В §2.7 мы даже пошли на отклонение от *IEEE* EUI-64, чтобы с адресами, назначенными вручную, было удобнее работать. Благодаря

[94] http://www.freebsd.org/cgi/man.cgi?query=ifconfig

[95] Anycast IPv6 addresses. Microsoft TechNet. http://technet.microsoft.com/en-us/library/cc759104%28WS.10%29.aspx

[96] Точнее, это IPv6 немного похож на IPX, потому что позаимствовал его хорошие стороны.

[97] Кроме особых адресов с двоичным префиксом `000`.

этой модификации, идентификатор интерфейса вполне может быть небольшим числом: 1, 2, 3 и т.д. Такая политика назначения адресов необходима, когда речь идет, например, о серверах, которым требуется постоянный и мало-мальски удобный сетевой адрес.

> Чаще всего серверу достаточно постоянного имени, которое зарегистрировано в DNS. Тем не менее, в некоторых случаях нужен именно постоянный адрес IP, который будет зафиксирован в настройках клиентов. Например, это касается серверов DNS. Ведь узнать из DNS адрес сервера DNS — это все равно, что вытащить себя из болота за собственные волосы.

Между тем, технология TCP/IP находит применение не только в традиционных вычислительных сетях. Не исключено, что скоро все интеллектуальные бытовые приборы и личные электронные устройства будут общаться друг с другом по IPv6, и движение в эту сторону заметно уже сегодня. Управление такими сетями требует нетрадиционного подхода, потому что у рядового пользователя не будет ни желания, ни навыков, чтобы вручную назначить адрес IPv6 каждому из окружающих его «маленьких цифровых друзей». Но даже в традиционных сетях далеко не все узлы требуют адресов, назначенных вручную. Например, такие адреса не нужны большинству рабочих станций и терминалов. Для них, равно как и для персональных электронных устройств, будет достаточно *предсказуемой* автоматической настройки адресов.

> Предсказуемость здесь нужна, чтобы облегчить отладку сети, если уж до этого дойдет дело.

Итак, мы видим, что нам пора заняться процедурой автоматической настройки индивидуальных адресов в IPv6. На какие части можно разбить ее? Прежде всего, узел должен найти для каждого из активных сетевых интерфейсов один или несколько адресов IPv6. Чтобы составить хотя бы один адрес IPv6, узел должен откуда-то узнать значения его фундаментальных частей — префикса подсети и идентификатора интерфейса. Посмотрим на эти части по отдельности, начав с последней.

Если у интерфейса, подлежащего настройке, есть уникальный канальный адрес, такой как MAC-48 или EUI-64, то это отличный кандидат на роль идентификатора интерфейса. Все, что требуется в этом случае — преобразовать канальный адрес к формату «модифицированный EUI-64» по общепринятой методике, которую мы «разработали» в §2.7. Эта часть задачи не представляет трудности.

> Строго говоря, методику преобразования канального адреса в идентификатор интерфейса должен приводить документ, посвященный передаче IPv6 в данной канальной среде. Например, для *Ethernet* это [RFC 2464]. Тем не менее, случаи EUI-64 и MAC-48 также рассмотрены в центральном документе «Адресная архитектура IPv6» потому что эти виды идентификаторов популярны и применяются в самых разных технологиях канального уровня [Приложение A, RFC 4291]. Там же приводятся рекомендации, что делать, если у интерфейса нет уникального канального адреса.
>
> Мы уже встретили пример, когда, не имея готовых адресов, канальный протокол согласует уникальные идентификаторы интерфейса для IPv6. Это, конечно же, IPV6CP в рамках PPP — см. §4.1.2.

Как выбрать префикс подсети, зависит от области адреса. Если надо составить внутриканальный адрес, то искомый префикс заранее известен: *FE80::/64*. Благодаря этому, узел может самостоятельно выбрать внутриканальные адреса для своих активных интерфейсов, не пользуясь никакими дополнительными сведениями.

> Хотя для внутриканальных адресов был зарезервирован префикс *FE80::/10*, на интерфейсах мы встречаем только *FE80::/64* в согласии с тем правилом, что длина префикса подсети IPv6 составляет 64 бита. Но, строго говоря, внутриканальный префикс подсети может зависеть от типа канала и уточняется соответствующим документом о канальной инкапсуляции IPv6.

Если же область адреса больше, чем канал, то префикс подсети выбирает и сообщает узлу администратор сети. Здесь наш путь снова раздваивается: или администратор вручную заносит префикс подсети в настройки узла, или же префикс настраивается централизованно, а узел узнает его автоматически, следуя некому протоколу. Такой протокол вполне сможет работать поверх IPv6, так как узел уже выбрал себе внутриканальные адреса.

> Искушенный читатель спросит, почему бы не выбрать этим протоколом DHCP (точнее, его вариант для IPv6 — DHCPv6). Сейчас мы убедимся, что в NDP уже заложено достаточно возможностей, чтобы обойтись без еще одного протокола, по крайней мере, для распространения самых необходимых настроек.

Мы посвятим §5.4.2 вопросу о точном механизме автоматической настройки префикса подсети, а сейчас давайте переключимся на следующую проблему. Предположим, что узел уже автоматически составил адрес IPv6. Например, это может быть внутриканальный адрес, префикс которого известен заранее: *FE80::/64*. В отличие от IPX, где идентификатор интерфейса обязан совпадать с его адресом MAC, в IPv6 такого требования нет. Поэтому нельзя исключить ситуацию, когда автоматически выбранный идентификатор интерфейса уже занят другим узлом-соседом.

Скажем, узел собирается назначить внутриканальный адрес интерфейсу *Ethernet*, адрес MAC которого *00-01-02-03-04-05*. Согласно правилам «модифицированного EUI-64», ему отвечает идентификатор интерфейса *02-01-02-FF-FE-03-04-05*. Сцепляя его с внутриканальным префиксом, узел получает адрес *FE80::0201:02FF:FE03:0405*. Несмотря на уникальность исходного адреса MAC, тот же самый адрес IPv6 уже может быть назначен другому соседу, например, вручную. Таким образом, было бы опрометчиво назначить автоматически составленный адрес IPv6, не проверив его на уникальность в пределах канала.

> Строго говоря, рассмотренный в примере конфликт адресов возникает из-за нарушения формата «модифицированный EUI-64»: администратор не должен назначать «с потолка» идентификаторы интерфейсов, у которых установлен бит **U/L**. Тем не менее, на практике случается всякое. Даже производители сетевых адаптеров не застрахованы от ошибок, и у разных адаптеров могут совпасть адреса MAC. Подобные проблемы желательно выявлять до того, как они навредят работе сети.

Непроверенный адрес мы обозначим как **пробный** (*tentative*). Требуется ли новый протокол, чтобы проверить такой адрес на уникальность? На самом деле, нет — мы можем обойтись протоколом ND из §5.1, слегка доработав его. В общих чертах схема ясна: узел посылает запрос NS касательно пробного адреса и ожидает, не ответит ли кто-то из соседей объявлением NA; если такой ответ поступит, то пробный адрес, очевидно, уже занят. Однако этот случай применения ND обладает рядом отличительных особенностей, которые выделяют его в под-протокол с названием **DAD** (*duplicate address detection*, выявление дубликатных адресов) [§5.4 RFC 4862].

> Как мы знаем, в IPv4 для той же цели служил добровольный запрос ARP. Хотя практика IPv4 уже давно держит этот механизм в своем арсенале, формальная процедура была опубликована относительно недавно под названием «**обнаружение конфликтов адресов IPv4**» (*IPv4 Address Conflict Detection*, **ACD**) [RFC 5227].

Первое затруднение состоит в том, что участие в ND, каким мы его знаем, требует уже назначенного индивидуального адреса. Ведь у пакета ND должен быть определенный адрес источника, не так ли? В то же время, узел не может назначить себе адрес, не проверив его по DAD. Это рождает «проблему курицы и яйца», для решения которой DAD должен работать, даже когда узел-инициатор вообще не обладает индивидуальными адресами. Пусть узел, выполняющий DAD, *всегда* посылает запрос NS с неопределенным адресом источника IPv6 *::* [§5.4.2 RFC 4862] и без опции «канальный адрес источника»

[§4.3 RFC 4861]. Тогда неопределенный адрес источника выделит NS для DAD среди обычных вызовов NS, передаваемых с целью разрешения адреса или NUD.

Но как узел-инициатор получит возможный ответ NA? Ведь пакет IPv6 нельзя направить по неопределенному адресу! Чтобы обойти и эту преграду, пусть отвечающий узел направляет ответ NA по групповому адресу «все узлы канала», *FF02::1*, если вызов NS пришел с неопределенного адреса. Следовательно, узел-инициатор должен вступить в эту группу перед началом DAD [§5.4.2 RFC 4862].

> Как следствие, протокол управления группами IPv6 MLD должен работать, когда на интерфейсе еще нет индивидуальных адресов. Возьмем себе это на заметку, чтобы вспомнить позже, когда дело дойдет до MLD (§6.4).

> Благодаря тому, что запрос NS в рамках DAD не использует опцию «канальный адрес источника» и ответ на него подлинно групповой, механизм DAD устойчив к конфликтам канальных адресов. Если бы мы «оптимизировали» DAD так, чтобы ответ NA шел по индивидуальному адресу MAC, коммутируемая локальная сеть направила бы его одному случайному владельцу этого адреса MAC, и DAD с большой вероятностью дал бы сбой. А ведь конфликт канальных адресов приведет и к конфликту адресов IPv6, составленных на их основе посредством «модифицированного EUI-64».

Отвечающий узел шлет объявление NA в ответ на запрос NS, однако в случае DAD это объявление направляется всем узлам канала, а не только узлу-инициатору DAD. Такое объявление NA получат и те узлы, которые о нем не просили. Поэтому, отвечая на вызов DAD, то есть вызов NS с неопределенного адреса, узел должен сбросить флаг **S** (**S** = 0). Вот еще одна особенность применения ND для DAD.

Дополнительное затруднение возникает, когда два узла одновременно пытаются проверить один и тот же незанятый адрес. Если следовать простейшей схеме, то ни один из узлов не получит объявления NA, и оба узла решат, что пробным адресом можно пользоваться. В результате возникнет конфликт адресов. Чтобы избежать такого исхода, узел-инициатор должен обращать внимание не только на объявления NA, но и на вызовы NS. Если узел получит чужой вызов NS касательно пробного адреса, исходящий с неопределенного адреса, то значит, еще один узел пытается проверить тот же адрес с помощью DAD, и назревает конфликт адресов [§5.4.3 RFC 4862]. Если же чужой вызов NS касательно пробного адреса поступит с определенного адреса, то это обычный розыск соседа, и его следует игнорировать, потому что право нашего узла на пробный адрес еще не доказано. В свою очередь, чтобы принимать интересующие его вызовы NS, узел-инициатор DAD должен заранее вступить в группу искомого узла, отвечающую пробному адресу [§5.4.2 RFC 4862].

> Вообще говоря, узел может получать и свои собственные групповые сообщения. Так как в данном случае адрес источника IPv6 неопределенный, источник вызова DAD надо идентифицировать, если это возможно, по его канальному адресу [Приложение A RFC 4862]. Альтернативный подход, если локальный уровень IPv6 уверен, что получает собственные вызовы DAD, — это подсчитать разность между числом посланных и принятых сообщений [там же]; очевидно, в отсутствие конфликта она должна оставаться нулевой. В любом случае, верная обработка этой ситуации зависит от точных свойств канала и интерфейса в него.

> Интересная идея, подсказанная нам редактором, состоит вот в чем: узел мог бы отличать собственные групповые вызовы NS от чужих, если бы он включал в свои вызовы особую опцию ND, содержащую достоверный «отпечаток пальцев» узла, например, подпись HMAC.

> Узел-владелец пробного адреса, отвечая на вызов DAD, мог бы послать объявление NA более узкой группе искомого узла, а не группе «все узлы канала». Ведь узел-инициатор DAD все равно обязан состоять в обеих группах. Мы не

видим препятствий для такого подхода, однако стандарты о нем не говорят. По-видимому, мысль разработчиков IPv6 шла по тому же пути, что и наша: сначала они решали проблему доставки объявления NA инициатору DAD, а только потом озаботились проблемой одновременных сеансов DAD. Затем надо было соотнести полученные решения между собой и оптимизировать первое из них в свете второго, но до этого, видимо, просто не дошли руки.[98]

Соберем воедино все наши замечания и оформим их как процедуру DAD, в ходе которой узел А проверяет пробный адрес П на интерфейсе И:

1) узел А вступает на интерфейсе И в следующие группы:
   а) «все узлы канала», *FF02::1*;
   б) группа искомого узла, отвечающая адресу П (группа Г);
2) узел А в фоне ожидает сообщений ND;
3) узел А направляет группе Г вызов DAD, то есть NS касательно адреса П с неопределенным адресом источника;
4) узлы, уже занявшие адрес П,[99] если таковые есть, отвечают на вызов DAD объявлениями NA (**S** = 0), направляя их группе «все узлы канала»;
5) узел А отказывается назначить себе адрес П и ожидает ручного вмешательства оператора, если до истечения тайм-аута приходит хотя бы одно из таких сообщений ND:
   а) чужой вызов NS с неопределенного адреса касательно адреса П;
   б) объявление NA об адресе П;
6) DAD дает отрицательный результат, если истекает тайм-аут.[100]

Поскольку сообщения ND могут теряться, а именно отсутствие сообщений означает отрицательный результат, процедуру DAD следует повторить несколько раз.[101] Так узел достовернее убедится, что опасности конфликта нет.

По умолчанию параметр **DupAddrDetectTransmits** равен единице [§5.1 RFC 4862], и повтора нет.

Вооружившись такой процедурой DAD, мы можем проверять любые индивидуальные адреса IPv6, а не только составленные автоматически. Поэтому пусть узел IPv6 выполняет DAD перед назначением каждого индивидуального адреса независимо от того, настроен ли он вручную или получен автоматически [§5.4 RFC 4862]. Так DAD поможет выявлять ошибки в настройке сети и минимизировать ущерб от них.

Тем не менее, администратор узла может избирательно заблокировать DAD на интерфейсе, установив параметр **DupAddrDetectTransmits** в ноль [§5.4 RFC 4862].

Обратите внимание: проверка DAD — только для индивидуальных адресов. В частности, DAD никогда не применяют к адресам anycast, потому что те по определению не уникальны [§5.4 RFC 4862]. Между тем, DAD вполне способен обнаружить и предотвратить случай, когда узел назначает своему интерфейсу некий адрес А как индивидуальный, хотя тот уже назначен другим интерфейсам канала как anycast. Ведь, как мы обсудили в §5.3, принадлежность внешне индивидуального адреса IPv6 к anycast — это локальное знание обладающего им узла. В случае назревающего конфликта узел, который назначает себе индивидуальный адрес, просто получит объявления NA с **O** = 0, а не 1. Условие надежной работы этого механизма таково: **MAX_ANYCAST_DELAY_TIME** < **RetransTimer**.

---

[98] См. обсуждение «*why is NA to DAD NS sent to all-nodes address?*» в списке рассылки IETF IPV6: http://www.ietf.org/mail-archive/web/ipv6/current/msg06287.html (зеркало: http://www.mail-archive.com/ipv6@ietf.org/msg06101.html).

[99] Их может быть несколько, если адрес П назначен им как anycast.

[100] Параметр **RetransTimer**, по умолчанию **RETRANS_TIMER** (1 с) [§6.3.2 RFC 4861].

[101] Параметр **DupAddrDetectTransmits**, всего 1 раз [§5.1 RFC 4862].

Обратная ситуация, когда адрес A уже назначен узлу Y как индивидуальный, а сосед X пытается назначить его себе как anycast, приведет к тому, что назначение anycast завершится успешно, но все пакеты от соседей в адрес A пойдут исключительно узлу Y. Самостоятельно обсудите, можно ли обнаружить эту ситуацию автоматически. (Указание: не забудьте о возможном присутствии на канале ND proxy.)

Процедура DAD универсальна по отношению не только к адресам, но и к каналам. Ведь она основана на протоколе ND, а тот поддерживает все каналы с групповым вещанием, а не только широковещательные каналы, подобные *Ethernet*. Как следствие, и процедура DAD применима ко всем каналам этого типа без исключения. Это как раз тот случай, когда ND полезен даже на каналах «точка-точка».

С другой стороны, на канале NBMA процедура DAD полноценно работать не сможет, потому что нет возможности обратиться ко всем узлам такого канала сразу. Это ограничивает применимость к NBMA протоколов, которые зависят от работоспособности DAD. Такие протоколы автоматически выбирают идентификатор интерфейса и потому вдвойне обязаны проверить его на возможный конфликт. К этой категории относятся SLAAC (§5.4.2), расширения для конфиденциальности (§6.2), CGA (§6.3) и HBA (§6.8). Мы познакомимся с ними ниже, а сейчас просто запомним эту особенность: если адрес составляет сам узел, то ему необходим работоспособный механизм DAD.

Пожалуй, единственный случай, в котором требование DAD можно ослабить, — это когда уникальность адресов обеспечивает другой протокол. Уже известный нам пример — это внутриканальные адреса на канале PPP. Они содержат заведомо уникальные идентификаторы интерфейсов, согласованные с помощью IPV6CP (§4.1.2), и поэтому применение к ним DAD избыточно [§5 RFC 5072]. Тем не менее, прочим индивидуальным адресам даже на интерфейсах PPP может потребоваться проверка DAD [там же].

Располагая DAD, узел IPv6 уже способен сам назначить внутриканальные адреса своим сетевым интерфейсам. Так узел может самостоятельно, без вмешательства оператора, выполнить то требование §2.5, что у каждого сетевого интерфейса IPv6 должен быть внутриканальный адрес [§2.1 RFC 4291]. Это важный шаг на пути к автоматической настройке адресов большей области, так как после него узел может свободно общаться со своими соседями.

Кроме того, внутриканальных адресов уже достаточно, чтобы смогла работать простейшая сеть без маршрутизаторов, такая как была показана на Фиг. 3 (стр. 22). То есть мы походя «создали» для IPv6 механизм, аналогичный динамической настройке адресов *169.254.0.0/16* в IPv4 [RFC 3927].

5.4.2. Розыск маршрутизаторов и префиксов

Теперь пришло время сделать то, что мы уже давно внесли в наш план, а именно обеспечить централизованное управление конфигурацией хостов IPv6 в пределах канала.

Первым шагом в этом направлении давайте поручим какому-то из узлов-соседей распространение информации о префиксах подсети, назначенных данному каналу. Этой информацией в любом случае должны располагать маршрутизаторы канала, поэтому они и будут идеальными кандидатами на роль ее распространителей, а дополнительные источники этой информации, такие как серверы DHCP, нам сейчас не понадобятся.

Как мы уже упоминали в §5.4.1, вариант DHCP для IPv6 известен под именем DHCPv6 [RFC 3315].

Рассмотрим простейший случай, когда на канале есть всего один выделенный маршрутизатор, а остальные узлы (хосты) получают от него сведения о префиксах и выбирают его маршрутизатором по умолчанию. Благодаря ND мы уже решили несколько

важных задач, так что пусть распространение информации о префиксах будет очередным подмножеством ND [§6 RFC 4861]. Оформить его можно следующим образом:

- маршрутизатор время от времени рассылает группе «все узлы канала», *FF02::1*, особое сообщение ND, которое мы назовем **объявление маршрутизатора**, сокращенно **RA** (*router advertisement*);
- узел может запросить объявление RA, не дожидаясь периодической рассылки, с помощью другого сообщения ND, которое мы обозначим как **вызов маршрутизатора**, сокращенно **RS** (*router solicitation*); вызов RS логично направить группе «все маршрутизаторы канала», *FF02::2*.

Если принять во внимание нашу текущую задачу, то главная информация, которую мы хотим видеть в объявлении RA, — это список префиксов, доступных узлам канала для автоматической настройки адресов. Подобный список вполне может быть частью объявления RA. А какие еще сведения мы хотели бы видеть в объявлении RA?

В §5.2 мы запланировали, что список маршрутизаторов по умолчанию и список префиксов в хостах IPv6 тоже можно будет настраивать автоматически. Однако, пока еще не принято окончательных решений касательно протокола, нам следует осознать, что «самореклама» маршрутизаторов, объявление префиксов и автоматическая настройка адресов — это взаимосвязанные, но все же разные механизмы, основанные на протоколе ND. Поэтому нам не следует объединять их в один монолитный блок. Рассмотрим сейчас каждый механизм по отдельности.

Розыск маршрутизаторов (*router discovery*) — это интересная возможность, которая существовала, но так и не получила распространения в IPv4 [RFC 1256]. На практике есть два способа сообщить хосту IPv4 адрес маршрутизатора по умолчанию: или он указывается в административном порядке,[102] или хост выбирает его, участвуя в протоколе маршрутизации, таком как RIP или OSPF. Если же мы желаем построить сеть с резервированием, когда маршрутизаторов по умолчанию несколько, то у нас остается только второй путь, а это означает лишние хлопоты по настройке и поддержке службы RIP или OSPF на каждом хосте.

Для IPv6 мы уже «создали» механизм на основе NUD, с помощью которого хост IPv6 может выбрать «живой» маршрутизатор из списка, не прибегая к другим протоколам кроме ND; это произошло в §5.2. Поэтому хост IPv6 вполне готов работать со списком маршрутизаторов по умолчанию вместо единственного адреса, как это было в IPv4.

> Мы имеем в виду именно список разных индивидуальных адресов, а не один адрес anycast, за которым стоят несколько маршрутизаторов.

Оборотная сторона такого списка — это расход усилий на его настройку и поддержку. Почему бы хосту IPv6 не вести этот список самостоятельно, используя сведения из объявлений RA? В принципе, уже сам факт, что узел рассылает соседям объявления RA, может говорить о его готовности работать маршрутизатором по умолчанию. Но, чтобы жестко не связывать между собой сообщения RA и функцию маршрутизатора по умолчанию, мы чуть позже выделим в формате RA поле со смыслом «я — маршрутизатор по умолчанию». Пока что просто отметим его необходимость. Благодаря такой дифференциации, узел сможет распространять среди соседей полезные сведения, не становясь автоматически их маршрутизатором по умолчанию.

Перейдем к следующему из намеченных нами механизмов, а именно к **розыску префиксов** (*prefix discovery*). Есть ли у него другие применения, кроме автоматической настройки адресов? Иными словами, какие еще префиксы могут быть интересны хосту? Конечно же, это префиксы «на канале». Ведь, как мы обсудили в §5.2, префикс подсети IPv6 не всегда автоматически находится «на канале», и наоборот, вполне возможны префиксы «на канале», не совпадающие с префиксом подсети. Мы даже отметили себе на

[102] Возможно, централизованно на сервере DHCP или BOOTP.

будущее, что эти сведения могли бы распространять маршрутизаторы, если бы у нас был походящий механизм.

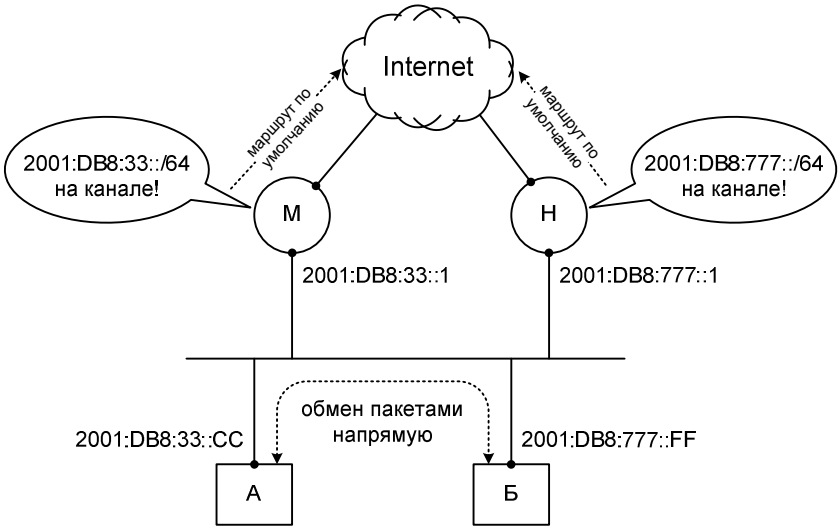
Сейчас мы как раз над ним работаем, и возможность сообщать узлам префиксы «на канале» возникает у нас естественным образом. Для этого будет достаточно, чтобы в объявлении RA каждый префикс сопровождали его атрибуты. В частности, префикс может обладать такими свойствами:

а) быть пригодным для автоматической настройки адресов;
б) находится «на канале»;
в) и то, и другое.

То есть (а) и (б) — это независимые свойства префикса, и потому каждое из них заслуживает отдельного флага длиной один бит. Обозначим флаг автоматической настройки как **A**, а флаг «на канале» как **L**.

> Случай, когда оба флага сброшены (**A** = 0 и **L** = 0), сейчас не имеет смысла, но он может пригодиться для расширения протокола, когда у префикса появятся новые свойства.

Как и прочая информация в сообщениях RA, сведения о префиксах «на канале» вполне могут исходить от нескольких маршрутизаторов. Рассмотрим это на примере сценария, где канал обслуживают два маршрутизатора, и каждый из них отвечает за свой префикс «на канале», как показано на Фиг. 82. Теперь сведения о префиксах «на канале» полностью рассредоточены, так что маршрутизатор М больше не играет роль единственного координатора и не может выслать переадресовку, когда хост А вручает ему очередной пакет для хоста Б. Ведь информация, что *2001:DB8:777::/64* на канале, находится у маршрутизатора Н. В отсутствие объявлений RA, пакеты от А к Б пойдут окольным путем, посетят маршрутизаторы М и Н и даже проделают часть пути по *Internet*, хотя А и Б — соседи по каналу!



Фиг. 82. Два маршрутизатора на канале, и у каждого своя подсеть

Допустим теперь, что маршрутизаторы М и Н рассылают объявления RA. Они адресованы группе «все узлы канала», так что хост А будет получать объявления не только от М, но и от Н. Поэтому маршрутизатор Н сможет оптимизировать направление трафика, объявив, что префикс *2001:DB8:777::/64* — «на канале» (**L** = 1). Тогда хост А зафиксирует *2001:DB8:777::/64* в своем списке префиксов, и трафик от А к Б пойдет по кратчайшему пути. Точно так же хост Б получит объявление RA от маршрутизатора М, поместит *2001:DB8:33::/64* в свой список префиксов, и обратный трафик от Б к А тоже пойдет напрямую.

> Обратите внимание: в отличие от хостов, маршрутизаторы никогда не используют чужие сообщения RA как источник сведений о канале. Тем не менее,

как мы увидим ниже, маршрутизатору следует пассивно наблюдать за чужими сообщениями RA и извещать администратора сети о замеченных разногласиях.

Прекрасно, теперь хост IPv6 может в автоматическом режиме получать собственные адреса, адреса маршрутизаторов по умолчанию и префиксы «на канале». А как быть, если какие-то из этих параметров изменят свои значения в ходе работы сети? Ведь маршрутизаторы и префиксы вполне могут возникать и исчезать по желанию сетевого администратора. Очевидно, что в этом случае вся сеть должна подстраиваться к новым условиям, чтобы администратору не пришлось вручную вычищать старые параметры из конфигурации хостов. В противном случае ценность механизмов автоматической настройки IPv6 будет равна одному ломаному грошу, да и то в базарный день.

Решить эту задачу можно, если параметры хоста не будут вечными. Вот возможная схема [§6.3.5 RFC 4861, §5.5.3 и §5.5.4 RFC 4862]:

- в конфигурации хоста с каждым параметром связан таймер;
- в объявлении RA каждый параметр снабжен временем жизни;
- хост перезапускает таймер для тех параметров, которые приведены в RA;
- хост удаляет параметр из своих настроек, если:
  - его время жизни истекает по таймеру, или
  - объявление RA сообщает, что новое время жизни параметра нулевое.

Здесь под параметром мы понимаем автоматически настроенный адрес, или элемент в списке маршрутизаторов по умолчанию, или префикс «на канале». Конечно, автоматически настроенный адрес наследует свое время жизни у префикса с установленным флагом **A**, из которого он образован.

Добавление нового параметра проблем не вызывает. Автоматически составив пробный адрес из нового префикса с **A** = 1, хост должен всего лишь убедиться в его уникальности с помощью DAD, прежде чем назначит его интерфейсу (см. §5.4.1). Адрес нового маршрутизатора достаточно внести в список маршрутизаторов по умолчанию, чтобы можно было им воспользоваться, когда NUD обнаружит сбой текущего маршрутизатора (см. §5.2). Новым префиксом «на канале» (**L** = 1) можно пользоваться немедленно, как только он будет внесен в список префиксов (см. §5.2).

Напомним себе, что все структуры данных эталонного хоста IPv6 привязаны к определенному интерфейсу, как было показано на Фиг. 72 (стр. 141). Поэтому объявление RA влияет только на структуры того интерфейса, через который оно было получено.

Будет ли так же проста процедура удаления этих параметров? Два последних вида параметров, маршрутизаторы по умолчанию и префиксы «на канале», определяют только мгновенное решение хоста, как маршрутизировать исходящий пакет, — они явным образом не влияют на долговременное состояние хоста. Более того, исчезновение маршрутизатора или префикса «на канале» — это внешнее событие, на которое хост повлиять неспособен. Поэтому параметры этих двух видов можно и нужно удалить, как только истечет время их жизни по таймеру или объявлению RA.

В то же время, у локального адреса IP свойства особенные. Во-первых, к нему привязаны протяженные во времени сеансы вышестоящих протоколов, а безусловное удаление адреса вызовет их обрыв. Во-вторых, хост все-таки сам управляет своими адресами, пусть даже на основании внешних сведений, таких как административная конфигурация или объявления RA. Благодаря этому перед хостом открывается возможность провести удаление просроченного адреса по более сложной, но менее болезненной процедуре.

Чтобы «прощание» прошло гладко, пусть хост расстается с просроченным адресом в два этапа [§1 и §5.5.4 RFC 4862].

Здесь не работает пословица: долгие проводы — лишние слезы. ☺

Пока время жизни адреса еще не истекло, он доступен как для создания новых сеансов,[103] так и для продолжения уже начатых. Такой адрес называют **предпочтительным** (*preferred*). Когда его время жизни истекает, адрес становится **устаревшим** (*deprecated*): с него больше нельзя устанавливать новые сеансы, но старые, тем не менее, продолжаются и завершаются — по крайней мере, мы на это надеемся.

> В качестве исключения, с устаревшего адреса можно устанавливать новые сеансы, если у хоста нет подходящего предпочтительного адреса, например, когда все его адреса в требуемой зоне устарели [§5.5.4 RFC 4862]. Также приложение может явно запросить определенный локальный адрес посредством API, даже если он устаревший.
>
> Все это касается только исходящих сеансов; входящие сеансы на устаревшие адреса по умолчанию разрешены, так как удаленный хост ничего не знает о состоянии адреса, по которому он обращается.

Какое-то время спустя устаревший адрес превращается в **недействительный** (*invalid*), полностью исчезая с интерфейса; теперь незавершенные сеансы разрываются в аварийном порядке. В противоположность недействительному адресу, мы назовем адрес **действительным** (*valid*), пока он назначен интерфейсу. Предпочтительные и устаревшие адреса — действительные.

Осуществить подобную схему можно, связав с каждым адресом хоста два таймера, первый из которых отсчитывает **предпочтительное время жизни** адреса (*preferred lifetime*), а второй — **действительное** (*valid lifetime*). Предпочтительное время жизни не может превышать действительное, потому что недействительный адрес не может оставаться предпочтительным.

> Состояние адреса «пробный» из §5.4.1 можно рассматривать наряду с состояниями «предпочтительный» и «устаревший». Пробный адрес назначается интерфейсу «не до конца»: в этом состоянии его нельзя использовать для обмена данными, и хост не отвечает на обычные вызовы NS касательно пробного адреса. Затем проходит DAD, и по его результатам адрес превращается в предпочтительный или помечается как дубликат. В этом случае времена жизни можно назначить пробному адресу еще до проведения DAD. Небольшая сложность заключается в том, что один или оба таймера могут сработать до завершения DAD.
>
> Самостоятельно разберите оптимизацию механизма DAD, известную как «оптимистический DAD» [RFC 4429], и дайте ей оценку.

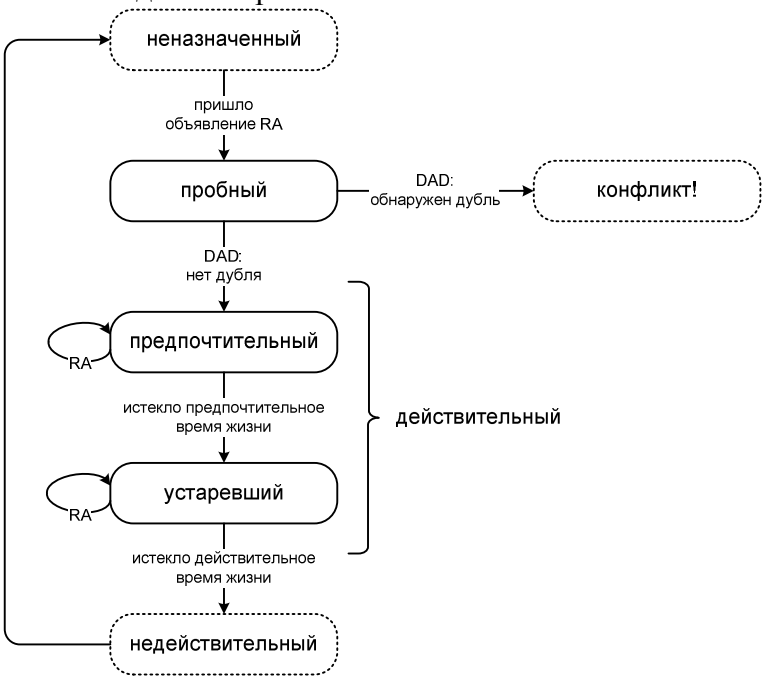
Чтобы реализациям IPv6 не пришлось делать большого различия между адресами, назначенными вручную и настроенными автоматически, пусть особое значение времени жизни означает бесконечность. Так как в объявлении RA время жизни будет представлено в виде двоичного беззнакового поля, пусть бесконечность обозначает максимальное значение этого поля, «все единицы». Тогда действительное и предпочтительное времена жизни смогут быть атрибутами всякого локального адреса, но некоторые адреса устаревать или исчезать никогда не будут.

Но даже адресам с конечным временем жизни не обязательно исчезать, пока их префикс анонсируется маршрутизатором. Как мы только что сказали, объявление RA вызывает перезапуск таймеров на соответствующих параметрах. Поэтому, например, если адресу оставалось жить 5 секунд, но пришло объявление RA, в котором время жизни соответствующего префикса 1 час, то таймер будет перезапущен с интервалом 1 час, и адрес в ближайшее время не исчезнет с интерфейса. Благодаря этому, объявления RA могут сколь угодно долго регенерировать автоматически назначенные адреса.

В результате жизненный цикл индивидуального адреса IPv6 выглядит, как показано на Фиг. 83. Состояния «недействительный» и «неназначенный» в нем — это, по

[103] Определение сеанса остается на усмотрение вышестоящих протоколов.

сути, одно и то же. Можно для удобства полагать, что адрес переходит из первого во второе, когда его удаление с интерфейса полностью завершено: сеансы закрыты, структуры данных освобождены и проч.



Фиг. 83. Жизненный цикл индивидуального адреса IPv6

Наконец-то мы можем составить форматы сообщений RS и RA. Сообщение RS [§4.1 RFC 4861] — это просто вызов, так что главную информацию несет сам факт отправки и приема этого сообщения. Тем не менее, мы учтем полученный в §5.1 опыт с форматом NS и позволим сообщению RS содержать опцию ND «канальный адрес источника» (SLLA), как показано на Фиг. 84. Действительно, раз хост занялся розыском маршрутизаторов, то он наверняка собирается начать сетевой диалог. Поэтому маршрутизаторы смогут поступить предупредительно и заранее создать для этого хоста запись NC в состоянии ПРОСРОЧЕННАЯ, чтобы позднее не тратить времени на его розыск. А вдобавок, на такой вызов RS можно будет ответить индивидуальным, а не групповым объявлением RA, чтобы не тревожить понапрасну другие узлы канала.

> Конечно, у опции «канальный адрес источника» есть смысл, только если адрес источника IPv6 определенный, потому как вносить в NC неопределенный адрес было бы эксцентричным поступком. По стандарту, RS допустимо слать с неопределенного адреса источника, ::, чтобы избежать «проблемы курицы и яйца» [§4.1 RFC 4861], хотя, как нам кажется, хосту ничто не мешает назначить себе внутриканальный адрес и слать RS с него. Точнее, если хост не способен по какой-то причине назначить себе уникальный внутриканальный адрес, то у него будут те же самые сложности и с назначением себе других адресов.

Фиг. 84. Вызов маршрутизатора — RS

Формат RA более богат [§4.2 RFC 4861]. Прежде всего, мы решили, что в нем должно быть поле «время жизни маршрутизатора». Пусть нулевое значение этого поля означает, что данный маршрутизатор не хочет, чтобы хосты вносили его в свои списки маршрутизаторов по умолчанию; а если данный маршрутизатор уже в списке, то его следует оттуда удалить.

В данном протоколе все маршрутизаторы эквивалентны. Однако существует расширение ND [RFC 4191], которое позволяет назначить маршрутизаторам разные приоритеты, а также распространить более точные маршруты на некоторые префиксы. У хостов, поддерживающих это расширение, может появиться еще одна структура данных, а именно явная таблица маршрутов.

Еще механизм розыска маршрутизаторов поможет централизованно настраивать параметры ND и даже всего IPv6, если в сообщении RA будут соответствующие поля. Выбор этих полей мы подробно обсуждать не будем, а приведем только их список:

- предельное число шагов IPv6 (§3.2) — значение по умолчанию в создаваемых хостом пакетах, если иное не указано приложением посредством сетевого API;
- время доступности соседа (в миллисекундах) — параметр **BaseReachableTime** (§5.1): основа для вычисления случайного параметра **ReachableTime** (тайм-аут записи NC в состоянии **ДОСТУПНАЯ**);
- период повтора — параметр **RetransTimer** (§5.1): пауза между повторными вызовами NS, пока на них нет ответа NA.

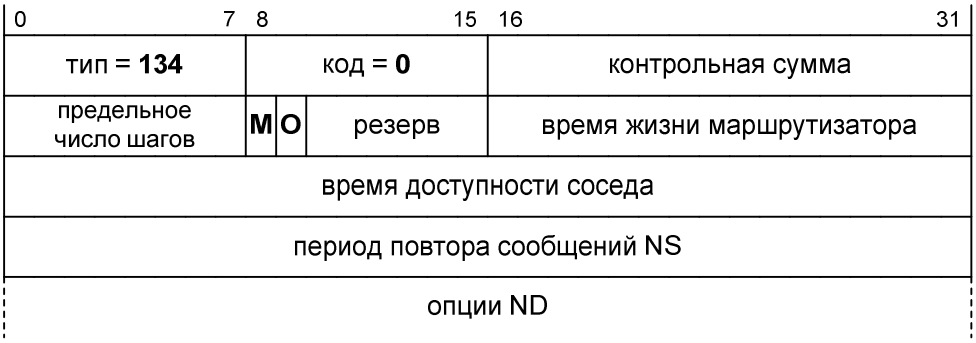
Если маршрутизатор не настаивает на определенном значении какого-то из параметров, то пусть он помещает в это поле ноль.

По-видимому, выбор пал на **BaseReachableTime** и **RetransTimer**, потому что именно эти параметры мы бы стали подстраивать, если бы, к примеру, нам пришлось оптимизировать работу канала на несколько тысяч хостов. В таком канале накладные расходы на NUD можно понизить, подняв **BaseReachableTime**, а задержку отклика на сообщения NS, вызванную их высокой частотой, можно компенсировать, увеличив **RetransTimer**.

В принципе, эти поля можно было вынести в опции ND, как мы поступим с полем «MTU канала».

Если у хоста несколько интерфейсов, то данные параметры относятся к тому интерфейсу, через который было получено объявление RA [§6.3.2 RFC 4861].

Еще в объявлении RA есть два флага, **M** и **O**, которые служат для того, чтобы отсылать хосты к протоколу DHCPv6 за настройками. Установленный флаг **M** говорит, что адрес IPv6 хост может получить по DHCPv6, а не настраивать автоматически, тогда как установленный флаг **O** означает, что из DHCPv6 доступны дополнительные настройки, такие как адреса серверов DNS, SIP и проч. Поскольку DHCPv6 возвращает хосту все настройки сразу, флаг **O** при установленном флаге **M** избыточен.

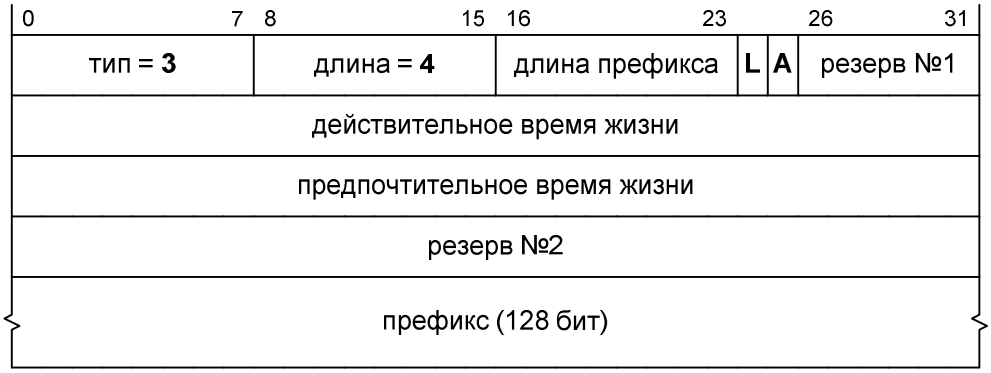


Фиг. 85. Объявление маршрутизатора — RA

Это была фиксированная часть RA, постоянной длины, — показана на Фиг. 85. Но мы, вообще-то, начинали с того, что хотели распространять список префиксов. Длина этого списка, очевидно, переменная, так что давайте поместим этот список среди опций ND.

Можно поступить еще проще: пускай одна опция содержит один префикс, как показано на Фиг. 86. Она так и называется: **опция сведений о префиксе** (*Prefix*

Information Option, **PIO**) [§4.6.2 RFC 4861]. Если префиксов несколько, то в хвосте сообщения RA появится несколько опций PIO.

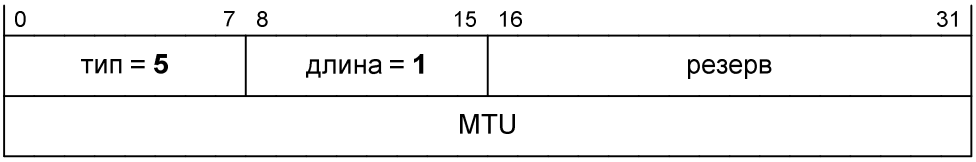


Фиг. 86. Опция сведений о префиксе — PIO

По нашему плану, атрибутами каждого префикса будут его длина, действительное и предпочтительное времена жизни, а также флаги **A** и **L**, говорящие, в каких целях хостам можно использовать данный префикс: для автоматической настройки адресов, для ведения списка префиксов «на канале».

А если маршрутизатор заодно сообщит свой канальный адрес в опции «канальный адрес источника» (SLLA), то хостам не придется дополнительно разрешать его сетевой адрес — они сразу же смогут создать запись NC, хотя бы в состоянии ПРОСРОЧЕННАЯ. Так маршрутизатор «убьет нескольких зайцев» одним объявлением RA.

Еще одна опция ND, специфичная для сообщения RA — это «MTU канала» [§4.6.4 RFC 4861]. Она призвана помочь работе IPv6 поверх каналов с переменным значением MTU. Формат этой опции показан на Фиг. 87.



Фиг. 87. Опция «MTU канала»

> Пример такого канала можно найти даже в современной технологии *Ethernet*. Представьте себе два коммутатора, А и Б, первый с поддержкой сверхбольших кадров («кадр-слон», *jumbo frame*), а второй без нее. Если их соединить между собой, то получится ЛВС, в которой PMTU пары портов зависит от того, на каких коммутаторах выбраны эти порты. Между всеми портами А значение PMTU будет 9000 байт, между портами А и Б — 1500 байт, между портами Б тоже 1500 байт. Допустим, что все интерфейсы, подключенные к А, получили значение MTU 9000 байт, а все интерфейсы, подключенные к Б, — 1500 байт. Механизмы TCP MSS и PMTUD не смогут преодолеть такую асимметрию, потому что извещения ICMP «пакет слишком большой» слать будет некому и большие кадры от А к Б будут просто теряться. (Последняя надежда на детектор «черной дыры PMTUD».) Зато если все хосты, подключенные к А и Б, получат одно и то же значение MTU 1500 байт, то канал сможет беспрепятственно работать, несмотря на особенности его реализации.

С какого адреса источника маршрутизатор должен вести рассылку RA? Ведь именно этот адрес хосты сохранят в своих списках маршрутизаторов по умолчанию, если «время жизни маршрутизатора» в объявлении RA больше нуля. Логично было бы использовать тот адрес, вероятность смены которого минимальна. Этим свойством обладает внутриканальный адрес, так как он не зависит от адресного плана сети, текущего провайдера *Internet* и проч. Поэтому маршрутизатор обязан рассылать RA с внутриканального адреса [§4.2 RFC 4861].

> Конечно, это не произвольный внутриканальный адрес, а назначенный тому интерфейсу, через который ведется рассылка.

Такое же правило в §5.2 мы сформулировали для переадресовок [§4.5 RFC 4861].

Обратите внимание, как работает зонная архитектура адресов IPv6 в случае обмена RS и RA. Хост вправе отправить вызов RS с любого из своих адресов на данном интерфейсе [§4.1 RFC 4861], а маршрутизатор может ответить индивидуальным объявлением RA по этому адресу [§6.2.6 RFC 4861]. Таким образом, вполне возможно, что адреса источника и назначения RA будут из зон разной величины. То есть адрес источник RA всегда будет внутриканальным, а вот адрес назначение может быть, например, глобальным. Тем не менее, такой пакет не покинет пределов данного канала и потому не нарушит принципа изоляции зон (§2.4).

Те же соображения применимы и к переадресовке (§5.2). Ее адрес назначения совпадает с адресом источника пакета-виновника, а тот с большой вероятностью принадлежит области больше внутриканальной, так как пакет был направлен через маршрутизатор. В то же время, адрес источника переадресовки — заведомо внутриканальный.

Распространение параметров IPv6 и ND в сообщениях RA влечет за собой еще одну проблему. Если канал обслуживают несколько маршрутизаторов, то вполне может оказаться, что по недосмотру они объявляют разные значения параметров. В этом случае объявления RA от разных маршрутизаторов будут непрерывно менять соответствующие настройки хостов, а флуктуации настроек чреваты нестабильностью в работе протоколов. По этой причине необходимо, чтобы все маршрутизаторы канала объявляли согласованные наборы параметров. В первую очередь, это касается численных параметров в фиксированной части RA, а также опции «MTU канала»: их значения обязаны совпадать.

Разные маршрутизаторы вполне могут объявлять разные списки префиксов, но и здесь необходимо определенное единообразие. А именно времена жизни одного и того же префикса, объявляемого разными маршрутизаторами, должны совпадать, чтобы не было нежелательных флуктуаций в настройках хостов. В то же время, флаги префиксов вполне могут отличаться, потому что их можно объединить операцией «побитовое ИЛИ». К примеру, если префикс *2001:DB8:0:123::/64* объявлен одним маршрутизатором как доступный для автоматической настройки адресов (**A** = 1), а другим — как расположенный «на канале» (**L** = 1), то хосты вправе как внести его в свои списки префиксов, так и назначить себе адреса на его основе.

Интересный вопрос состоит в том, как именно удалить префикс «на канале» с помощью RA [§6.3.4 RFC 4861]. Сброс флага **L** в объявлении не даст желаемого результата, потому что **L** = 0 значит «нет информации», а не «вне канала». Поэтому верным решением будет объявить префикс с **L** = 1 и нулевым временем жизни. Из двух времен жизни префикса, атрибут «на канале» использует действительное время жизни [§4.6.2 RFC 4861]. Если тот же префикс доступен для автоматической настройки адресов (**A** = 1), мгновенная установка его действительного времени жизни в ноль не приведет к потере уже настроенных адресов благодаря защитному правилу «2 часа», которое мы обсудим в конце раздела. Но еще надежнее будет объявить один и тот же префикс в двух опциях PIO, из которых одна имеет **L** = 1 и нулевое время жизни, а другая — **A** = 1 и ненулевое время жизни.

Чтобы помочь администратору в согласовании параметров, каждому маршрутизатору рекомендовано следить за объявлениями соседей, сравнивать их со своими и сообщать в системном журнале о найденных противоречиях [§6.2.7 RFC 4861].

А как должен поступить хост, если разные маршрутизаторы объявляют разную длину одного и того же префикса? Здесь мы сами себе поставили небольшую ловушку, как хитрый экзаменатор студенту. На самом деле, длина префикса — это его неотъемлемая часть, так что один и тот же префикс не может иметь разные длины: это будут разные цепочки битов, разные префиксы.

Между тем, один префикс может включать в себя другой. Например, двоичный префикс `01` содержится в префиксе `0100`, а префикс IPv6 *2001:DB8::/32* — в префиксе *2001:DB8::/48*. При этом префикс большей длины точнее, так как ему отвечает меньшее множество адресов. Когда такие префиксы находятся на одном канале, это тривиальный случай: множество адресов короткого префикса полностью содержит в себе множество адресов длинного префикса. Тем не менее, это разные префиксы, у каждого из которых свой набор флагов и свое время жизни. Например, префикс «на канале» *2001:DB8::/32* может устареть раньше, чем *2001:DB8::/48*, и тогда адреса, составляющие разность этих множеств, окажутся «вне канала».

> Затруднение могло бы возникнуть, если бы такие префиксы разной длины были доступны для авто-настройки адресов IPv6. Однако здесь нас спасает правило, что длина префикса подсети IPv6 всегда равна 64.

Когда канал обслуживают несколько маршрутизаторов, то их немедленный ответ на вызов RS приведет к пику нагрузки на канал. Это может даже открыть дверь атаке типа «отказ в обслуживании», когда злоумышленник подключился к каналу или захватил уже подключенный хост и теперь бомбардирует его маршрутизаторы вызовами RS, чтобы спровоцировать усиленный в несколько раз поток ответов RA.

> От засылки извне канала пакеты RS и RA защищены при помощи GTSM: поле «предельное число шагов» в них обязано равняться 255 — см. §5.1.

Маршрутизатору не возбраняется всегда направлять объявления RA по адресу «все узлы канала» независимо от того, шлет ли он это объявление по таймеру или по вызову [§6.2.6 RFC 4861, §5.5.1 RFC 4862]. Так злоумышленник или даже просто сбойный хост сможет вызвать целую бурю группового трафика. Чтобы этого не произошло, каждый маршрутизатор обязан ограничить частоту своих объявлений RA [104].

Тем не менее, проблема нежелательной синхронизации RA остается. С ней надо бороться, сделав время отправки RA случайным. Если RA по вызову, то надо внести случайную задержку, а если по таймеру, то достаточно сделать его интервал случайным [§6.2.4 и §6.2.6 RFC 4861]. Последний штрих состоит в том, чтобы вообще не слать RA по вызову, если задержка помещает его на оси времени после ближайшего RA по таймеру. Благодаря этим предосторожностям, групповая рассылка объявлений RA не повредит каналу даже при возникновении сбойных или злонамеренных хостов.

> Интересно, что разные периодические события в сети неожиданно проявляют склонность к самопроизвольной синхронизации во времени [105], и чаще всего это нежелательное явление. Поэтому приходится явным образом вносить случайность в длины задержек и прочие подобные параметры протоколов, как мы это сделали для **ReachableTime** в §5.1. В свою очередь, хорошее качество случайных величин можно обеспечить, только опираясь на теоретическую базу; действовать наобум здесь нельзя [RFC 4086].

Последний вопрос, связанный с розыском маршрутизаторов, заключается в том, может ли узел IPv6 впоследствии отказаться от роли маршрутизатора. Согласно общепринятой модели [§6.2.1 RFC 4861], маршрутизатор IPv6 — это функция узла, которую можно избирательно включить или выключить на каждом из его сетевых интерфейсов, при условии что узел вообще способен быть маршрутизатором. Из соображений устойчивости и безопасности, по умолчанию эта функция выключена.

Но допустим теперь, что на данном интерфейсе некого узла M эта функция была когда-то явным образом включена и узел какое-то время исполнял все обязанности маршрутизатора: продвигал транзитный трафик, высылал переадресовки, рассылал

---

[104] Параметр **MIN_DELAY_BETWEEN_RAS**, 3 с [§6.2.6 RFC 4861].

[105] S. Floyd, V. Jacobson, "The Synchronization of Periodic Routing Messages", IEEE/ACM Transactions on Networking, April 1994. <http://ee.lbl.gov/papers/sync_94.pdf>

объявления RA с информацией о настройках сети. Чтобы теперь корректно уйти со сцены и превратиться в скромный хост, узел M должен аннулировать всю информацию, которую он явно или неявно сообщил хостам.

С явной информацией все довольно просто: достаточно разослать несколько раз объявление RA, в котором список параметров все тот же, но время их жизни нулевое. Так узел M заставит хосты удалить информацию о префиксах и исключить себя из списка маршрутизаторов по умолчанию.

Однако в памяти хостов все еще остаются записи DC, которые указывают на узел M. Подобные записи могли возникнуть, даже если M не был маршрутизатором по умолчанию, а значит, их нельзя убрать объявлением RA. Например, эти записи могли быть созданы по переадресовке другого маршрутизатора H. В результате окажется, что хосты продолжают слать транзитный трафик через узел M, хотя тот его просто игнорирует. Если бы узел M вообще исчез, то сработал бы механизм NUD, и хосты бы вычистили устаревшие записи DC, связанные с недоступным соседом. Но в нашем случае узел M будет по-прежнему доступен, и образуется «черная дыра».

Чтобы разрешить это затруднение, нам пригодится флаг **R** в объявлении NA, который мы предварительно зарезервировали. Пусть узел устанавливает этот флаг в своих сообщениях NA, когда он маршрутизатор на данном интерфейсе, и сбрасывает, когда он просто хост. Тогда хосты-соседи смогут более надежно узнать об отказе узла M продвигать транзитный трафик. Как только флаг **R** в его сообщениях NA обнулится, полагается вычистить все записи DC, указывающие на узел M, потому что он добровольно сложил с себя полномочия маршрутизатора [§7.2.5 RFC 4861].

> Таким образом, оказывается, что хост IPv6 должен следить не только за доступностью соседей, но и за их состоянием «хост/маршрутизатор». Этой цели служит булево поле **IsRouter** в эталонной записи NC [§5.1 RFC 4861].
>
> Определить, что сосед — маршрутизатор, хост может по прямым или косвенным признакам [Приложение D RFC 4861]. Прямой признак один, это флаг **R** в объявлении NA, тогда как косвенных признаков несколько. Скажем, получение объявления RA от соседа говорит о том, что он — маршрутизатор. На то же самое указывает переадресовка *на* соседа, если это переадресовка на маршрутизатор (адрес цели не совпадает с адресом назначения). (Переадресовку *от* соседа хост может получить, только если он уже использует его как маршрутизатор.) Косвенные признаки, в отличие от прямого, не могут подтвердить обратное, а именно что сосед — это хост, а не маршрутизатор.

На этом мы закончили с розыском маршрутизаторов, и пришло время ответить на вопрос о более подробном алгоритме автоматической настройки адресов. Чтобы освежить нашу память, вернемся к моменту, когда хост провел первичную инициализацию стека IPv6 и активировал сетевой интерфейс. Вот что хост делает дальше:

1) преобразует канальный адрес интерфейса в идентификатор интерфейса IPv6 длины *N*;

> Еще раз напомним себе, что *N = 64* — это чисто административное решение, которое не следует «зашивать» в протоколы стека IPv6.
>
> Строго говоря, хост не обязан использовать канальный адрес интерфейса при выборе его идентификатора IPv6 [Приложение A RFC 4291], однако это повсеместная практика, если канальный адрес у интерфейса имеется.

2) создает пробный внутриканальный адрес IPv6, сцепляя префикс *FE80::/(128−N)* и идентификатор интерфейса, полученный на шаге №1;
3) проверяет пробный внутриканальный адрес на уникальность с помощью DAD;
4) если адрес уникальный, назначает его интерфейсу и устанавливает для него бесконечные времена жизни. В противном случае работа IPv6 через

данный интерфейс невозможна и происходит останов процедуры [§5.4.5 RFC 4862];

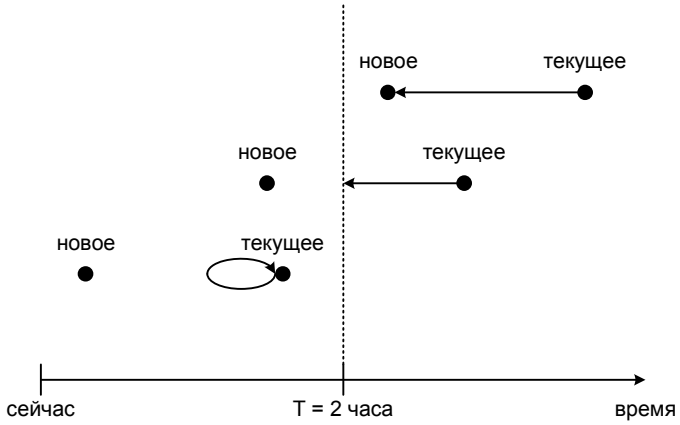
5) возможно, посылает вызов RS группе «все маршрутизаторы канала», *FF02::2*;
6) по мере получения объявлений RA сканирует опции PIO [§5.5.3 RFC 4862]:
    а) игнорирует опцию, если в ней сброшен флаг **A**;

Опция PIO с **A** = 0 может пригодиться для заполнения списка префиксов «на канале», если у нее **L** = 1; но мы сейчас говорим только об автоматическом назначении адресов сетевому интерфейсу.

    б) игнорирует опцию, если префикс внутриканальный;
    в) игнорирует опцию, если действительное время жизни префикса меньше предпочтительного;
    г) если префикс еще не назначен:
        i) игнорирует префикс, если сумма длин префикса и идентификатора интерфейса не равна длине адреса IPv6, 128 бит;
        ii) игнорирует префикс, если его действительное время жизни равно нулю;
        iii) создает пробный адрес, сцепляя префикс из RA и идентификатор интерфейса из шага №1;
        iv) проверяет пробный адрес на уникальность с помощью DAD;
        v) если адрес уникальный, то назначает его интерфейсу и устанавливает для него времена жизни согласно принятой опции RA;
    д) иначе, если префикс уже назначен в составе ранее созданного адреса:
        i) устанавливает предпочтительное время жизни адреса согласно опции;
        ii) устанавливает действительное время жизни на основании опции (см. примечание).

Хотя вопрос о безопасности ND мы еще не обсуждали, уже сейчас понятно, что без дополнительных механизмов защиты сообщения ND уязвимы к обычной подделке. Подделка RA, помимо прочего, позволяет провести очень простую атаку типа «отказ в обслуживании», когда хост получает RA, в котором действительное время жизни его текущих префиксов очень короткое. В результате хост сбрасывает свои адреса и некоторое время остается полностью недоступен. Обратите внимание, что хост не нужно «бомбардировать» — достаточно единственного фальшивого RA. Чтобы блокировать этот вектор атаки, стоит ограничить возможность RA сокращать действительное время жизни адресов [§5.5.3 RFC 4862]. Суть этого ограничения довольно проста, ее иллюстрирует Фиг. 88. Прежде всего, RA может беспрепятственно удлинять время жизни адреса. Например, если к приходу RA адресу оставалось жить 5 минут, а новое время жизни 10 минут, то таймер адреса будет установлен на 10 минут. Если же RA на самом деле пытается сократить время жизни адреса, то оно может укоротить его только до 2 часов. К примеру, если адресу оставалось жить 1 час, а новое время жизни 10 минут, то адрес проживет целый час. Если адресу оставалось жить 3 часа, а новое время жизни 45 минут, то адрес проживет 2 часа. И только если новое время жизни больше 2 часов, оно вступит в силу как есть. Когда у нас появится надежная защита сообщений RA от подделки, это ограничение можно будет снять. Иными словами, ограничение действует только на те сообщения RA, подлинность которых никак не подтверждена. Кроме того, оно действует только на

действительное время жизни как наиболее уязвимый параметр, тогда как предпочтительным временем жизни адресов можно управлять без ограничений.



Фиг. 88. Защита от атаки на время жизни адреса

Хост выполняет шаг 6 в фоне, пока интерфейс активен, так как периодические объявления RA сообщают текущую конфигурацию данного канала, а она может изменяться по желанию администратора сети. Маршрутизатор может сам назначить своим интерфейсам внутриканальные адреса, выполнив шаги 1–4; шаги 5 и далее — только для хостов [§4 RFC 4862].

У такого механизма автоматической настройки адресов есть одна любопытная особенность: ни маршрутизатор, ни хост не фиксируют назначение адреса в долговременной памяти. Когда хост включают, он проводит процедуру автоматической настройки с самого начала и, как нетрудно убедиться, получает тот же самый набор адресов, пока постоянны его канальный адрес и конфигурация маршрутизаторов. Можно сказать, что у такого механизма нет внутреннего состояния. Поэтому полностью его называют так: **автоматическая настройка адресов без внутреннего состояния**, сокращенно **SLAAC** (*stateless address autoconfiguration*) [RFC 4862].

> В качестве безвредной оптимизации, [§5.7 RFC 4862] разрешает хосту хранить текущие настройки SLAAC в долговременной памяти (ППЗУ, диск) в пределах времени их жизни.

Важное условие работы SLAAC — чтобы канал поддерживал групповое вещание на сетевом уровне. Если это не так, то и SLAAC на таком канале работать не сможет.

> Это не значит, что для SLAAC нужно настоящее групповое вещание на канальном уровне, например, каким мы его знаем в *Ethernet*. Требуется групповое вещание именно на сетевом уровне, а канальный уровень может его эмулировать. Так, например, в канале «точка-точка» групповое вещание сводится к передаче пакетов удаленному узлу. А если канал многоадресный и широковещательный, но без группового вещания, то групповые пакеты можно передавать всем узлам в широковещательных кадрах, полагаясь на отбор нужных пакетов сетевым уровнем. Пример этого подхода мы находим в ARCNET [§4.3 RFC 1201, §7 RFC 2497].

Как и всякий сложный автоматический механизм, SLAAC должен быть, в идеале, полностью управляемым, а по меньшей мере — отключаемым. У оператора узла должна быть возможность блокировать работу SLAAC на сетевом интерфейсе, если она нежелательна по тем или иным соображениям. Это касается как хостов, так и маршрутизаторов. Хост должен быть готов работать с ручными настройками, игнорируя объявления RA. Что же касается маршрутизатора, то рассылка RA может повлиять на конфигурацию многих хостов, и поэтому ее надо вообще отключить по умолчанию [[106]] и включать явным образом только при необходимости и только после надлежащей настройки всех параметров объявления RA.

[106] Параметр **AdvSendAdvertisements**, по умолчанию FALSE [§6.2.1 RFC 4861].

Это вовсе не противоречит обязательному статусу поддержки элементов SLAAC узлами IPv6. Одно дело, когда функцию можно избирательно отключить, а совсем другое, когда она вообще недоступна. То же касается DAD [§5.4 RFC 4862] и ND в целом.

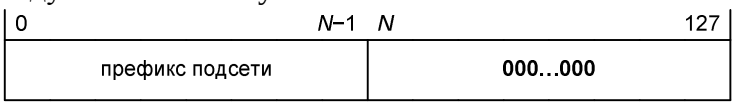
Интересное применение возможностей SLAAC — это перенумерация сети IPv6, например, при переходе от одного провайдера к другому [§4.1 RFC 4862]. Ввиду постоянной длины префикса подсети, все идентификаторы интерфейсов могут остаться теми же, и тогда задача сводится к перенумерации префиксов подсетей. Гладко провести эту операцию можно, применяя в переходный период два префикса с установленным флагом **A**. Тогда каждый хост назначит своему интерфейсу два адреса, старый и новый. Далее, управляя предпочтительным временем жизни старого адреса, можно добиться, чтобы он больше не выступал источником новых сеансов. Завершить операцию можно, полностью упразднив старый адрес с помощью его действительного времени жизни. При этом достаточно менять настройки маршрутизаторов, а хосты сами их подхватят из объявлений RA. Подробно безболезненная процедура перенумерации разобрана в [RFC 4192].

6. Вспомогательные механизмы

6.1. Выбор маршрутизатора наугад

В §5.4.2 мы «создали» механизм, с помощью которого хост может составить список маршрутизаторов по умолчанию, обслуживающих данный канал. В ходе работы этого механизма хост явным образом узнает индивидуальные адреса таких маршрутизаторов. Далее он может применить к ним определенную политику выбора, предпочитая одни маршрутизаторы другим.

Но если определенной политики нет и хосту безразлично, какой именно маршрутизатор он выберет, то возникает хороший повод применить вещание наугад (§5.3). Для этого всем маршрутизаторам канала надо назначить общепринятый адрес anycast, так чтобы он был заранее известен хостам. Адрес anycast по внешним свойствам не отличается от индивидуального адреса, поэтому он вполне может принадлежать подсети, назначенной данному каналу. Следовательно, общепринятое значение мы можем выбрать только для идентификатора интерфейса, так как префикс подсети следует из адресного плана данной сети, его выбирает администратор. Пусть во всякой подсети на маршрутизаторы указывает адрес anycast с нулевым идентификатором интерфейса [§2.6.1 RFC 4291] — см. Фиг. 89. Это адрес «маршрутизатор подсети». Как следствие, нулевой идентификатор интерфейса зарезервирован, и его нельзя использовать в обычных адресах IPv6, будь то индивидуальных или anycast.



Фиг. 89. Адрес anycast «маршрутизатор подсети»

По изначальной задумке [§2.6 RFC 4291], адрес anycast «маршрутизатор подсети» был назначен, не только чтобы хост внутри подсети мог обратиться к произвольному маршрутизатору-соседу, но и чтобы хост извне подсети мог направить пакет на ее границу, скажем, при помощи маршрутного заголовка (§3.3.3). Этот план распространялся и на административные единицы больше подсети. Практического применения эта идея не получила.

Довольно узкоспециализированное применение адреса anycast «маршрутизатор подсети» — это отладка туннелей, в частности, 6rd [§5 RFC 5969]. Суть его сводится к тому, что стороны туннеля могут обращаться друг к другу, например, проводить *ping* удаленного конца, используя один заранее известный адрес, близкий по свойствам к индивидуальному, — тот на случай, если инструменты отладки не поддерживают групповое вещание.

Позже появились и другие общепринятые адреса anycast. В итоге сошлись на том, что 128 самых старших значений идентификатора интерфейса[107] в любой подсети зарезервированы для этой роли [RFC 2526]. Адрес «маршрутизатор подсети» был назначен самым первым и потому он — исключение из этого правила. Интересно отметить, что в зарезервированных таким образом идентификаторах (кроме нулевого) бит **I/G** установлен. Этим как бы подчеркивается, что основанные на них адреса — не индивидуальные.

То, что общепринятый адрес anycast «маршрутизатор подсети» уже существует и принят стандартами, — один из аргументов против назначения каналам «точка-точка» префиксов */127* [§3 RFC 3627]. При таком назначении один интерфейс неизбежно получает нулевой идентификатор. Хуже того, второй интерфейс может получить при этом настоящий адрес anycast «маршрутизатор подсети», с тем же численным значением, и тогда возникнет конфликт, потому что адреса anycast не подлежат проверке DAD (см. §5.4.1). Ответный аргумент сторонников */127* — что на практике адрес anycast «маршрутизатор подсети» используют исчезающее редко [§4 RFC 6164]. (Мы не выступаем ни за одну из сторон в этом споре, а предлагаем читателю решать данный вопрос в каждом конкретном случае сообразно техническим условиям и обстоятельствам — надеемся, что к концу курса он будет на это способен.)

6.2. Конфиденциальность на уровне адресов

СТАРШИЙ БРАТ СМОТРИТ НА ТЕБЯ.
(Дж. Оруэлл, «1984»)

Если у вас паранойя, то это еще не значит, что за вами не следят.
(Дж. Хеллер, «Уловка-22»)

Возникший в §5.4 механизм SLAAC не только облегчает настройку хостов IPv6, но и подстегивает манию преследования у пользователей *Internet*. Вот представим себе: пользователь купил портативный компьютер, сетевой интерфейс которого обладает постоянным адресом MAC-48. Это может быть адаптер *Ethernet* или *802.11 WiFi*. Путешествуя по миру, пользователь подключается к местным сетям IPv6, и всякий раз его компьютер настраивает адрес с помощью SLAAC. Нетрудно видеть, что идентификатор интерфейса в таком адресе всегда будет один и тот же. Скажем, если адрес MAC — *00-01-02-03-04-05*, то идентификатор будет такой: *02-01-02-FF-FE-03-04-05*.

Если вам это не очевидно, повторите §2.7.

Благодаря совместным усилиям IEEE и IETF, этот идентификатор глобально уникален и однозначно указывает на данный компьютер. Поэтому, пока пользователь возит с собой кремниевого любимца, у него тоже есть личный номер. Далее, префикс подсети в адресе IPv6 весьма часто указывает на географическое местоположение узла. Таким образом, по текущему адресу можно сказать, кто этот пользователь и где он находится! А текущий адрес не хранится в секрете, он известен владельцу любого сетевого ресурса, к которому пользователь обращается со своего компьютера. Теперь достаточно взять журнал ресурса, который пользователь регулярно посещает, чтобы узнать маршрут его путешествия. Связать же идентификатор интерфейса с личностью пользователя часто помогает сам пользователь, регистрируясь в различных интерактивных системах, таких как форумы WWW. Выходит, что сегодня за нами могут следить не только спецслужбы, но и пользователи *Internet*, у которых есть немного административных привилегий!

[107] С учетом того, что бит **U/L** остается сброшен.

Насколько серьезна подобная угроза, пусть каждый решает сам, а наше дело — удовлетворить пожелания пользователей, благо у нас уже есть почти все необходимые инструменты.

Прежде всего, еще раз назовем корень зла: постоянный и глобально уникальный идентификатор интерфейса. На самом деле, ни постоянство, ни *глобальная* уникальность нам от него не требуются — мы их получили в нагрузку от *IEEE*, когда скромно просили в §5.4 уникальность в пределах данного канала. Тем не менее, в §5.4.1 нам пришлось «разработать» механизм DAD, чтобы дополнительно проверять уникальность адресов на канале. Не применить ли его сейчас?

Допустим, хост генерирует совершенно случайный, непредсказуемый идентификатор интерфейса и использует его в SLAAC. Не исключено, что пробный идентификатор уже занят соседом, но это можно проверить с помощью DAD. Как быть, если DAD дал положительный результат, то есть адрес дубликатный? В обычном SLAAC (§5.4), где идентификатор интерфейса однозначно зависит от его аппаратного адреса, это была бы тупиковая ситуация, и хост ожидал бы административного вмешательства. Теперь же хост может сгенерировать другой случайный идентификатор, снова создать адрес, проверить его по DAD и так далее до тех пор, пока DAD не даст отрицательный результат. Эта простая идея и лежит в основе механизма IPv6, известного как **расширения для конфиденциальности** (*Privacy Extensions*) [RFC 4941].

> Конечно, даже случайный идентификатор интерфейса должен удовлетворять требованиям формата «модифицированный EUI-64» (§2.7). В частности, бит **U/L** должен быть сброшен, так как этот идентификатор не происходит от официально полученного глобального EUI-64 [§3.2.1 RFC 4941]. В то же время, стандарт не предписывает сбрасывать в ноль бит **I/G**; почему — нам не вполне ясно. По крайней мере, родственный формат CGA из §6.3 не забывает потребовать этого.
>
> Кроме того, надо проверить, что случайный идентификатор не совпадает ни с одним из зарезервированных значений и еще не присутствует ни на одном из интерфейсов хоста. Ведь колесо фортуны может повернуться неожиданной стороной.
>
> Хотя текущий протокол расширений для конфиденциальности основан на 64-битных идентификаторах интерфейса [§1 RFC 4941], в принципе длина идентификатора интерфейса могла бы быть произвольной, так как не представляет труда сгенерировать случайное значение любой заданной длины.

Конечно, хосту потребуется качественный генератор (псевдо)случайных чисел. Также надо исключить сценарий, когда два хоста одновременно генерируют одну и ту же псевдослучайную последовательность. Чтобы этого не произошло, генератор псевдослучайных чисел надо инициализировать каким-то заведомо уникальным и трудно предсказуемым значением. Эти проблемы давно известны и встречаются в самых разных областях технологии *Internet*, так что методы их решения тоже хорошо разработаны [RFC 4086].

> Как раз здесь может пригодиться глобальный адрес MAC или EUI. Он поможет инициализировать генератор (псевдо)случайных чисел так, чтобы тот выдал достаточно уникальную последовательность. Конечно, адрес надо смешать хэш-функцией с другими, менее предсказуемыми источниками случайности, но он послужит неплохой основой ввиду своей высокой уникальности.
>
> Нормативный документ расширений для конфиденциальности предписывает использовать одну вполне определенную хэш-функцию [§3.2 RFC 4941], но это сделано, по-видимому, только чтобы избежать самодельных, ненадежных алгоритмов в реализациях. В теории же выбор хэш-функции может быть совершенно произвольным, пока результат обладает достаточной степенью случайности и непредсказуемости.

Подобный механизм защищает конфиденциальность подвижных, странствующих пользователей, которые возят с собой свой компьютер. А можем ли мы подобным образом помочь маскировке оседлых пользователей? Да, если позволим хосту время от времени менять адрес, даже пока он связан с одной и той же точкой подключения. В общем случае хост не может свободно сменить префикс подсети, так как он задан точкой подключения. Нам придется ограничиться все тем же идентификатором интерфейса. Весь адрес, составленный из префикса подсети и случайного идентификатора интерфейса, мы обозначим как **временный** (*temporary*), потому что время его жизни заведомо ограничено. Этим он отличается от обычного автоматически назначенного адреса, который может существовать на интерфейсе неограниченно долго благодаря постоянной регенерации объявлениями RA.

Давайте интегрируем эту схему в уже существующий цикл жизни адреса: пробный, предпочтительный, устаревший, недействительный, — как показано на Фиг. 90. Временный адрес начинает свою жизнь как пробный, потому что его обязательно надо проверить на уникальность по DAD. Если адрес действительно уникальный, то он становится предпочтительным, и с него можно устанавливать исходящие сеансы. В противном случае надо просто сгенерировать новый случайный идентификатор интерфейса и начать все сначала.

> На всякий случай число таких попыток создать уникальный идентификатор ограничено величиной **TEMP_IDGEN_RETRIES** — по умолчанию 3 раза [§3.3 RFC 4941]. Действительно, если несколько попыток подряд не удались, то налицо какой-то непредусмотренный нами сбой.

Временный адрес обречен на то, чтобы рано или поздно стать устаревшим. Он не может оставаться предпочтительным дольше, чем указано в настройках хоста.[108] После этого адрес уже не годится для открытия новых прикладных сеансов. Поэтому было бы неплохо создать и назначить новый адрес чуть заранее,[109] чтобы у сетевого интерфейса всегда был хотя бы один предпочтительный временный адрес. У сетевого интерфейса IPv6 может быть много адресов, и не будет никакой беды в том, что времена жизни старого и нового адресов перекроются.

По нашей задумке новый адрес должен быть другим, так что ему потребуется новый случайный идентификатор интерфейса. Конечно, его можно генерировать каждый раз, когда в нем возникнет необходимость. Но если вычислительные ресурсы хоста ограничены, а требования к степени конфиденциальности не настолько высоки, чтобы каждый адрес хоста был случайным, достаточно время от времени генерировать один случайный идентификатор для каждого из сетевых интерфейсов [§3.5 RFC 4941]. Период регенерации должен быть таким, чтобы преемник устаревшего адреса отличался от него.[110]

> Еще новый идентификатор следует сгенерировать, когда хост переключается с одного канала на другой [§3.5 RFC 4941]. Впрочем, достоверно определить, что канал другой, — нетривиальная задача [RFC 4135].

Тем временем устаревший адрес продолжит свое существование, чтобы активные прикладные сеансы спокойно завершились штатным порядком. В них этот адрес уже «засвечен», и нет никакой необходимости торопить его полное исчезновение. Тем не менее, рано или поздно его стоит объявить недействительным и удалить, чтобы избежать слишком большого числа адресов на интерфейсе. Ведь с каждым адресом связаны

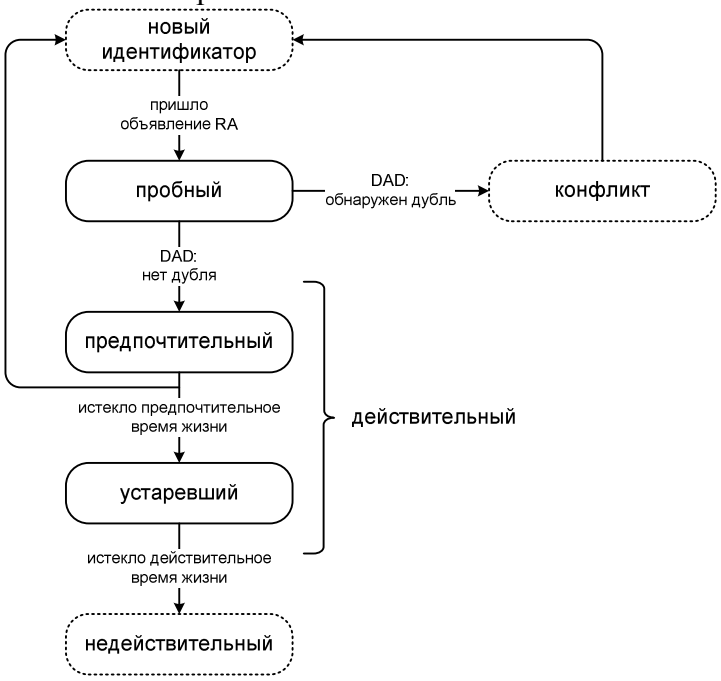
---

[108] Время жизни равно **TEMP_PREFERRED_LIFETIME** − **DESYNC_FACTOR**, где **TEMP_PREFERRED_LIFETIME** по умолчанию 1 день, а **DESYNC_FACTOR** — элемент случайности для борьбы с нежелательной синхронизацией [§3.3, §5 RFC 4941].

[109] Параметр **REGEN_ADVANCE**, по умолчанию 5 с [§3.4, §5 RFC 4941].

[110] То есть он должен быть короче, чем **TEMP_PREFERRED_LIFETIME** − **REGEN_ADVANCE** − **DESYNC_FACTOR**.

ресурсы хоста и сети: структуры в оперативной памяти, группа искомого узла и т.п. Поэтому действительное время жизни временного адреса тоже ограничено, хотя и довольно продолжительным интервалом.[111]



Фиг. 90. Жизненный цикл временного адреса IPv6

В остальном временный адрес следует тем же правилам, что и другие автоматически назначенные адреса. В частности, сообщения RA тоже управляют его временем жизни. Это можно представить себе вот каким образом. У простого адреса всего два таймера, один предпочтительного времени жизни, а другой действительного, и оба они перезапускаются по сообщению RA, если в нем упомянут соответствующий префикс. А у временного адреса вместо этого две пары таймеров. В каждой паре один таймер по-прежнему перезапускается сообщениями RA, зато второй взведен единожды, в момент создания адреса, и ведет монотонный отсчет времени. Тем не менее, сработать первым может любой из спаренных таймеров, и тогда адрес перейдет в следующее состояние.

Вот так незначительной доработкой уже готовых механизмов нам удалось решить задачу конфиденциальности адресов IPv6. Конечно, выдать пользователя может не только его сетевой адрес, но в многоуровневой системе защиты все элементы важны, так как злоумышленник может атаковать на любом уровне. Поэтому расширения для конфиденциальности — если и не достаточное, то уж точно необходимое средство, чтобы противостоять охоте за персональной информацией. Но пока за вами не следят, или если вас это не очень беспокоит, расширения для конфиденциальности удобнее держать выключенными, потому что они затрудняют анализ и отладку работы сети. Поэтому у этих расширений должен быть выключатель, и его положение по умолчанию должно быть «выкл» [§3.6 RFC 4941].

> Еще одна особенность применения временных адресов вот в чем. Если сетевой стек хоста принимает входящие соединения и сеансы на этих адресах, то политику безопасности хоста надо формулировать с большей осторожностью. В частности, надо иметь в виду, что не все локальные адреса хоста заранее известны.
>
> Также стоит учесть слабо обоснованную, но популярную практику ограничивать доступ к информационным ресурсам с адресов, у которых нет записи PTR в DNS. Это затруднение можно обойти с помощью DDNS, автоматически регистрируя для временных адресов случайные имена. (Очевидно, что регистрировать постоянное имя — значит, нарушить конфиденциальность.)

[111] Параметр **TEMP_VALID_LIFETIME**, по умолчанию неделя [§3.3, §5 RFC 4941].

6.3. Безопасность ND

Протокол ND получился более гибким и универсальным, чем его предшественник ARP, но устойчивость ND к умышленным атакам осталась низкой. Мы не станем разбирать все возможные векторы атаки, а ограничимся самым очевидным из них: узел злоумышленника подключается к каналу и отвечает на вызовы NS касательно чужих адресов. В своих ответах NA узел злоумышленника сообщает, что искомому адресу IPv6 отвечает его собственный канальный адрес. В результате прочие узлы канала сами направляют трафик узлу злоумышленника. К примеру, так узел злоумышленника может прикинуться маршрутизатором по умолчанию и перехватить весь трафик от узлов канала в *Internet*. Чтобы настоящий владелец адреса не мешал атаке своими ответами NA, можно подвергнуть его параллельной атаке типа «отказ в обслуживании» или выбрать момент, когда он отключен для профилактики.

> Этот и прочие векторы атаки на ND разобраны в RFC 3756.
>
> Убедитесь с помощью конечного автомата NUD (Фиг. 58 на стр. 118), что злоумышленнику достаточно выслать сообщение NA с установленными флагами **S** и **O**, чтобы «исправить» в свою пользу уже существующую запись NC, хотя на пустом месте создать запись NC он таким образом не может.
>
> Предложите способ принудительно создать запись NC в памяти атакуемого хоста. (Подойдут такие сообщения: NS, RS или RA с опцией SLLA; переадресовка с опцией TLLA.)

В отличие от ARP, который стоял в иерархии стека рядом с IP, ND — это подмножество ICMPv6, а значит, в теории его можно защитить на уровне IPv6. Такую защиту обеспечивает протокол IPsec, рассмотренный нами в §3.3.5. Но, увы, на практике это слишком тяжеловесное, хотя и надежное решение. На нижнем уровне работой IPsec управляют связи безопасности — грубо говоря, элементарные правила о том, как именно защищать пакеты из данного источника к данному адресату. Если к каналу подключены N узлов, а у каждого из них M адресов, то для полноценной работы ND надо будет настроить до $2 \cdot (N \cdot M + 1)$ связей *на каждом узле* [§6 draft-arkko-manual-icmpv6-sas]. Это трудоемкая и неблагодарная работа сложности $O(N^2)$. На более высоком уровне связями безопасности управляет удобный протокол IKE, но он требует полноценно работающего сетевого уровня, что неминуемо приводит нас к «проблеме курицы и яйца». В частности, IKE не позволит защитить самый уязвимый элемент ND — разрешение адресов [draft-arkko-icmpv6-ike-effects].

Можно ли защитить ND, обойдясь при этом ручными настройками трудоемкости $O(N)$? Давайте рассуждать. Для начала проверим, что нам даст симметричная криптография. Допустим, каждый законный узел хранит общий ключ и пользуется им для подписи сообщений ND (на выходе) и для проверки их подлинности (на входе). Эта схема будет работать до тех пор, пока злоумышленник не похитит ключ. После этого безопасность канала рассыплется, словно карточный домик: злоумышленник сможет прикидываться любым узлом. Восстановление безопасности потребует смены ключа на каждом узле канала. Плохая схема, двигаемся дальше.

Следующий кандидат — асимметричная криптография: каждый законный узел получает свою личную пару ключей, закрытый и открытый. На выходе узел подписывает сообщения ND своим закрытым ключом, на входе он проверяет подпись открытым ключом соседа. Кража закрытого ключа ставит под удар только его владельца, но не весь канал, и это хорошо. Но как узел узнает открытые ключи своих соседей? Если их распространять предварительно, сложность настройки всего канала подскочит до $O(N^2)$. А что будет, если источник сообщения ND сам приложит свой открытый ключ к сообщению? Положим, адресат сообщения ND не располагает никакой априорной информацией о законном соседе. Тогда злоумышленник сгенерирует собственную пару

ключей и сможет подписывать сообщения ND как ни в чем не бывало. Такой вариант не годится.

> Мы даже не пытаемся применить на данном этапе инфраструктуру открытых ключей (PKI), так как она потребовала бы работающей сети. Почему это так, мы увидим чуть позже в данном разделе, при обсуждении авторизации маршрутизаторов.

Однако мы ошибаемся, говоря, что априорной информации нет. Ведь при розыске соседа узел заранее знает соседский адрес IPv6! На самом деле, это допущение, и допущение очень важное: адрес IPv6 соседа — доверенный. Ведь сейчас мы работаем над защитой протокола ND, и нам нужны определенные «очаги доверия», гарантированные нам другими протоколами и уровнями стека, — иначе нам не на что будет опереться.

Что означает доверенность адреса IPv6? Это значит, что мы узнали его из доверенного источника. Скажем, мы хотим провести *ping* одного из интерфейсов узла-соседа Б. Как нам достоверно узнать его адрес IPv6? Во-первых, мы можем сами пойти к консоли этого узла, локально войти в систему и прочесть адрес в настройках.

> Конечно, возможны атаки и на локальный вход в систему, но сейчас мы их не рассматриваем: это один из наших «очагов доверия».

Во-вторых, если у нас есть основания доверять DNS[112] и мы знаем доменное имя узла Б, то программа *ping6* сама узнает адрес по имени:

```
$ ping6 b.example.org
PING6(56=40+8+8 bytes) 2001:db8::2 --> 2001:db8::1
...
```

В-третьих, адрес Б мы можем узнать от доверенного лица, из рук в руки или в подписанном электронном письме… Читатель может самостоятельно продолжить этот список.

Так или иначе, мы полагаем априори, что к началу розыска соседа разыскиваемый адрес IPv6 известен достоверно. Почему бы нам не превратить адрес IPv6 в контейнер для нужной информации? Если префикс подсети задан точкой подключения узла, то идентификатор интерфейса по-прежнему в нашем распоряжении. Мы уже помещали в него канальные адреса (§5.4.1) и даже случайные числа (§6.3), а теперь сохраним в нем отпечаток (*fingerprint*) открытого ключа. Тогда другие соседи смогут проверить, своими ли ключами пользуется данный узел.

Если следовать этой схеме, то идентификатор интерфейса уже нельзя назначить «с потолка», так как он зависит от открытого ключа узла. Процедура назначения адреса узлу и его интеграции в сеть будет примерно такой:

1) сгенерировать пару ключей для узла;
2) вычислить идентификатор интерфейса как отпечаток открытого ключа, используя заранее утвержденный алгоритм;
3) составить адрес из префикса подсети (согласно точке подключения) и идентификатора интерфейса (получен на шаге 2);
4) внести ключи в настройки узла;
5) назначит адрес сетевому интерфейсу узла;
6) распространить сведения о новом адресе в сети:
    а) зарегистрировать в DNS (см. §2.11);
    б) внести в настройки удаленных приложений;
    в) сообщить заинтересованным лицам.

Так мы получаем **криптографически произведенный адрес**, сокращенно **CGA** (*cryptographically generated address*). Полные правила работы с CGA существенно

[112] Например, благодаря применению DNSSEC.

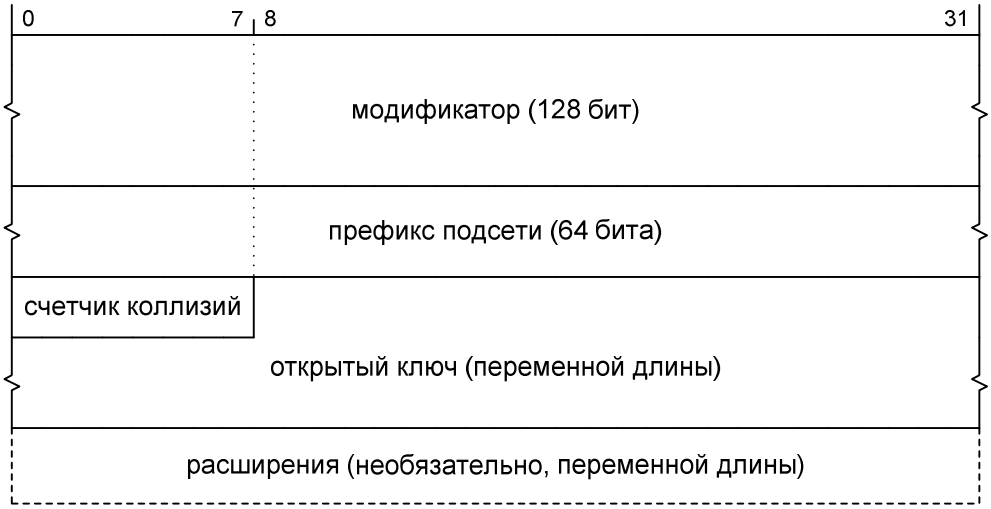
сложнее, они приведены в RFC 3972. Стоящая за ними идея, тем не менее, довольно прозрачна. А среди интересных подробностей RFC 3972 мы отметим две.

Во-первых, если узел — это рабочая станция пользователя, которая устанавливает только исходящие сеансы, то ее текущий адрес вполне может время от времени меняться. Именно этим мы уже пользовались в §6.2, когда работали над расширениями для конфиденциальности. Для такого узла шаг 6 вышеуказанной процедуры не имеет особого смысла.

> Тем не менее, даже в этом случае адрес полезно зарегистрировать в DDNS. Причину мы обсудили в §6.2, говоря о расширениях для конфиденциальности.

Шаги 2, 3 и 5 такой узел может выполнить самостоятельно, располагая стандартным алгоритмом. При постоянных ключах это позволяет узлу автоматически менять свой адрес CGA. Но как отпечаток ключа может стать переменным? Для этого он должен зависеть от дополнительного аргумента, известного всем сторонам протокола. Этот аргумент, **модификатор** CGA, не представляет тайны, так что передавать его можно вместе с открытым ключом. Вместе с еще несколькими величинами он составляет **параметры** CGA — см. Фиг. 91. Отпечаток зависит от всех этих параметров, поэтому они обязательно сопровождают открытый ключ.

> Так, префикс подсети входит в параметры CGA, чтобы отпечаток открытого ключа зависел от него. Благодаря этой зависимости, злоумышленник не сможет подменить в CGA префикс подсети.
>
> Еще один параметр CGA — это счетчик коллизий. Адрес CGA зависит от случайного модификатора и потому сам он тоже случайный, а значит, есть вероятность коллизии с уже существующим адресом. Эта коллизия не вызовет проблем, так как ее вовремя обнаружит DAD (§5.4.1), но что делать дальше? В этом случае достаточно увеличить на единицу счетчик коллизий и повторить вычисление хэш-функции, дающей отпечаток ключа. Если хэш-функция сильная, то новый идентификатор интерфейса окажется совершенно другим. Почему нельзя просто сгенерировать новый случайный модификатор, мы увидим в следующем пункте.



Фиг. 91. Блок параметров CGA

Во-вторых, в идентификаторе интерфейса доступны только 62 бита, так как значения битов **U/L** и **I/G** заданы форматом «модифицированный EUI-64» (§2.7). То есть без дополнительных ухищрений атака на CGA методом грубой силы имела бы сложность порядка 2^{62}. Учитывая рост вычислительных мощностей по закону Мура, это не так уж много. Поэтому три старших бита в идентификаторе интерфейса CGA выделены под специальное поле **Sec**. Протокол CGA остроумно использует эти три бита, чтобы плавно управлять сложностью грубой атаки на адрес за счет более трудоемкой генерации такого адреса [§7.2 RFC 3972].

Техника этого трюка напоминает поиск прообраза хэш-функции перебором. А именно ставится задача найти такой модификатор CGA, для которого хэш H(модификатор | открытый ключ | прочее) содержит ноль в $16 \times Sec$ старших разрядах. Все прочие модификаторы считаются недействительными. Это повышает сложность грубой атаки на отпечаток ключа до $O(2^{59+16\times Sec})$ [§7.2 RFC 3972, [113]]. Создание CGA усложняется до $O(2^{16\times Sec})$, но проверить уже подобранный модификатор можно за время O(1): достаточно вычислить указанный хэш и проверить, что $16 \times Sec$ старших разрядов в нем нулевые.

Очевидный недостаток исходного протокола CGA — это фиксированная хэш-функция, SHA-1. Исследователи активно ищут слабые стороны распространенных хэш-функций, так что нельзя исключать, что бастион SHA-1 когда-нибудь падет. Если это произойдет, то перейти на новую хэш-функцию, не расширяя протокола и не ломая совместимости, будет нельзя. Чтобы убедиться в этом, рассмотрим наивный подход: новые реализации на выходе используют более устойчивую хэш-функцию, скажем, SHA-3, а на входе пробуют и старую, и новую функции, так как заранее не знают, какой функцией создан данный CGA. Тогда злоумышленник сможет провести разновидность атаки понижением уровня безопасности (*downgrade attack*): взять доверенный адрес CGA, созданный при помощи SHA-3, и найти для него пару ключей, дающую такой же отпечаток при использовании уже взломанной функции SHA-1. Обратно совместимая реализация примет подпись злоумышленника, потому что попробует SHA-1 наряду с SHA-3.

Отметим, что для взлома CGA необходимо найти второй прообраз SHA-1, но исследователи достаточно близко подобрались только к поиску коллизий этой хэш-функции. О типах атак на хэш-функции см. [RFC 4270] и [[114]].

Чтобы не дожидаться взлома SHA-1, технические условия протокола CGA уже пришлось модифицировать [RFC 4982]. Увы, расширение CGA для поддержки разных хэш-функций — типичный пример того, как потом приходится расплачиваться за негибкость первой версии протокола. Предложенное решение лишено технического изящества и изменяет уже опубликованный протокол. А именно у поля **Sec** теперь новая, довольно запутанная интерпретация. К счастью, CGA все еще носит де-факто экспериментальный статус, и цена таких изменений в протоколе относительно невелика.

Механизм CGA явным образом рассчитан на то, что длина идентификатора интерфейса N равна 64 битам. Это упрощает устройство CGA, хотя и нарушает исходное положение [§2.5.4 RFC 4291], что на архитектурном уровне N — по-прежнему величина переменная. Впрочем, с самого начала было понятно, что это положение рано или поздно принесут в жертву хотя бы в частном случае. Так, принятие CGA в качестве стандарта еще не ставит крест на $N \neq 64$, потому что IPv6 вполне может работать и без CGA. С другой стороны, на тех каналах, где CGA действительно нужен, $N = 64$.

Отлично, мы наметили решение поставленной задачи: настройка CGA требует усилий порядка $O(N)$, где N — число узлов на канале. Теперь давайте в общих чертах скажем, какие изменения потребуются в протоколе ND.

Чтобы проверить подлинность подписанного сообщения ND, получатель рассматривает само сообщение, его цифровую подпись, открытый ключ источника, параметры CGA, а также адрес IPv6, на который претендует источник этого сообщения. Этот заявленный адрес зависит от типа сообщения [§5.1.1 RFC 3971], как показано в Табл. 16.

---

[113] T. Aura, M. Roe. Strengthening Short Hash Values <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.145.7681>

[114] A.J. Menezes, P.C. van Oorschot, S.A. Vanstone. Handbook of Applied Cryptography <http://www.cacr.math.uwaterloo.ca/hac/>

Табл. 16. Заявленные адреса разных типов сообщений ND

Тип сообщения	Заявленный адрес
NS с неопределенного адреса (DAD)	Адрес цели
NS с определенного адреса	Адрес источника
NA	Адрес источника = адрес цели (см. примечание)
RS	Адрес источника (если он определенный)
RA	Адрес источника
Переадресовка	Адрес источника

Данный механизм не поддерживает защиту ответов proxy, потому что proxy не знает закрытого ключа узла, от имени которого отвечает. Поэтому защита NA возможна только при условии, что адрес источника равен адресу цели [§7.4 RFC 3971]. Метод защиты ответов proxy находится в разработке [draft-krishnan-cgaext-proxy-send].

Проработав текущий раздел до конца, подумайте, на каком основании хост может доверять адресу маршрутизатора, приславшего подписанную переадресовку. (Ответ: Если адрес маршрутизатора известен заранее из ручной настройки или сертифицированного объявления RA.)

Первым шагом получатель проверяет соответствие между заявленным адресом (в предположении, что он — CGA) и открытым ключом. Для этого надо по стандартному алгоритму вычислить отпечаток ключа как функцию ключа и параметров CGA. Если идентификатор интерфейса в заявленном адресе совпадает с отпечатком, то сообщение успешно прошло первый этап проверки. Вторым шагом получатель проверяет цифровую подпись сообщения с помощью открытого ключа.

Таким образом, к дополнительным сведениям, которые еще нет в сообщении ND, относятся: параметры CGA, открытый ключ и цифровая подпись. Эти сведения источник сообщения помещает в специально отведенные для них опции ND. Правила работы с такими опциями составляют часть протокола **SEND** (*Secure ND*, безопасный ND) [RFC 3971].

Точнее, параметры CGA и открытый ключ попадают в одну опцию «CGA». Защищенное сообщение ND может и не содержать опции «CGA», если открытый ключ соседа известен из других источников. В таком случае достаточно одной цифровой подписи.

Механизм CGA в составе SEND работает при условии, что доверенный адрес предлагает узел, проводящий аутентификацию. В ответ он должен получить сообщение ND, касающееся именно этого адреса. Только в этом случае весь механизм надежно опирается на доверенность исходного адреса IPv6. Например, мы начинаем *ping* вполне определенного доверенного адреса, и наш локальный узел, рассылая вызовы NS, добивается объявления NA именно об этом адресе, а не каком-то другом. Таким образом, при разрешении адресов это условие выполняется.

Между тем, злоумышленник вполне может сгенерировать новый адрес CGA и вызвать создание записи NC о нем, например, послав NS с опцией SLLA. Однако такая запись не принесет заметного вреда, если не будет обращений по этому адресу на прикладном уровне или попыток использовать его как маршрутизатор. В этом и состоит модель защиты при помощи CGA: обращение идет только по доверенным адресам IPv6.

Хотя CGA можно применять не только в рамках SEND, делать это надо с большой осторожностью, так как разные применения одного и того же механизма защиты могут открыть лазейку для перекрестных атак, например, атак воспроизведением, когда злоумышленник предъявляет в одном протоколе защитные данные, перехваченные из другого протокола [§7.4 RFC 3972].

В то же время, при автоматическом розыске маршрутизаторов хост заранее не знает, какие адреса ему предъявят в объявлениях RA. Не исключено, что часть объявлений принадлежит незаконным маршрутизаторам [RFC 6104], но механизм CGA бессилен их разоблачить. Все, на что способен механизм CGA, — это помешать краже *заранее известного* адреса.

Чтобы защитить розыск маршрутизаторов, можно сертифицировать их в этой роли. То есть законному маршрутизатору выдают сертификат, подписанный удостоверяющим центром (возможно, по цепочке доверия). В процессе предварительной ручной настройки каждый хост получает копию корневого сертификата. После этого хост может проверить сертификат маршрутизатора, от которого пришло подписанное объявление RA.

Закрытый ключ удостоверяющего центра заперт в надежном сейфе, так что злоумышленник не может создать новый сертификат — он способен только похитить сертификат и закрытый ключ законного маршрутизатора, например, взломав его. Если злоумышленник сделает это, то он сможет присвоить себе полномочия данного маршрутизатора, но не более того. Когда взлом будет раскрыт, старый сертификат придется отозвать с помощью списка отозванных сертификатов (*certificate revocation list*, **CRL**) — это обычная практика работы с цифровыми сертификатами. Поэтому хосты должны проверять, не отозван ли сертификат маршрутизатора. Нет ли здесь «проблемы курицы и яйца»? Список CRL доступен для загрузки по сети,[115] и загрузить его в общем случае можно, только выбрав маршрутизатор. Если же маршрутизатор незаконный или взломан, то он может манипулировать загрузкой CRL. К счастью, список CRL подписан удостоверяющим центром, сертификат которого есть у хоста, так что фальсифицировать CRL нельзя. Все, что остается незаконному маршрутизатору — это помешать загрузке CRL. Следовательно, хост должен прекратить работу через выбранный маршрутизатор, если загрузка CRL не удалась. То есть хост вначале выбирает маршрутизатор условно, чтобы загрузить CRL, и переходит к обычной работе в сети, только если выполнены все перечисленные условия:

- список CRL успешно загружен;
- список CRL заверен действительной подписью удостоверяющего центра;
- сертификат выбранного маршрутизатора не значится в CRL.

В противном случае хосту следует сообщить оператору о проблеме и попытать счастья с другим маршрутизатором, рассылающим объявления RA [§6.3.1 RFC 3971].

На этих довольно простых идеях и основана авторизация маршрутизаторов в рамках SEND [§6 RFC 3971]. Дальше углубляться в нее мы не будем, а только отметим, что львиную долю в ней занимает механизм розыска и передачи цепочек доверия **ADD** (*Authorization Delegation Discovery*).

> К сожалению, ND подвержен не только прямым атакам путем подделки сообщений протокола. Конечный автомат ND реагирует и на другие события. Например, приход транзитного пакета IPv6 может вызвать создание записи NC в состоянии **НЕПОЛНАЯ**. Теперь представьте себе, что злодей бомбардирует маршрутизатор транзитными пакетами, адресованными несуществующим узлам подсети [§4.3.2 RFC 3756]. Очевидно, что в этом случае неосторожная реализация маршрутизатора может исчерпать записями **НЕПОЛНАЯ** всю доступную память и потерпеть крах. Помочь здесь может только предусмотрительное управление кэшем NC, например, адаптивное удаление записей **НЕПОЛНАЯ** по мере роста их числа. Те же соображения актуальны и для хоста, если злоумышленник может запускать на нем приложения, передающие пакеты произвольным адресатам.

[115] Например, по HTTP или LDAP.

6.4. Управление групповым трафиком

«Ешь, кума, девятую шанежку — я ведь не считаю». (Пословица)

Между групповым вещанием в IPv4 и IPv6 нет принципиальных отличий благодаря целенаправленной конвергенции их линий развития. Поэтому, даже если читатель не до конца знаком со всеми механизмами группового вещания IPv4, данный раздел не вызовет у него затруднений, потому что он не опирается на знания из IPv4. Как раз наоборот, у такого читателя будет возможность наверстать отставание в данной теме сразу для обеих версий IP.

С приходом IPv6 групповое вещание больше нельзя считать второстепенным механизмом и обходить его вниманием даже при начальном изучении темы — ведь без него IPv6 практически неработоспособен.

«Конструируя» такой важный механизм IPv6, как проколы ND, мы активно применяли групповое вещание, практически не задумываясь над тем, готово ли оно служить нам без каких-либо усилий с нашей стороны. Дабы потом не обнаружилось, что мы построили замок на песке, нам надо уделить немного внимания нашему верному помощнику и убедиться, что у него есть все необходимое для полноценной работы.

Находясь на сетевом уровне стека, мы склонны допустить, что на канальном уровне групповое вещание уже полностью работоспособно. Это естественное предположение, которое опирается на принятую нами модель сетевого стека, и отклонения от него могут быть вызваны только чисто практическими деталями. Но, к сожалению, эти детали слишком серьезны, чтобы обойти их вниманием.

Поэтому давайте для начала вспомним, каким образом устроена поддержка группового вещания на канальном уровне сетевого стека. По этому критерию канальные технологии можно разбить на четыре больших группы.

- В первой из них канальный уровень поддерживает групповое вещание, по крайней мере, в теории. К примеру, идеальная ЛВС *Ethernet* готова разослать групповой кадр, как только мы укажем верный адрес назначения MAC, в котором установлен бит **I/G**. На практике же необходимы различные оговорки. Пока сеть *Ethernet* реализована как общая шина, групповой кадр попадает на вход ко всем станциям, и задача фильтрации группового трафика ложится на сетевые адаптеры. Это означает, что аппаратные фильтры надо явным образом программировать, а когда число групп превышает возможности фильтра, то остается только отключить фильтр. В коммутируемой сети *Ethernet* проблема неизбирательной доставки групповых кадров по-прежнему остается, потому что коммутаторы заранее не знают, к каким портам подключены члены данной группы. Чуть позже мы посмотрим, как ее можно решить и чего это будет нам стоить.
- Во второй группе находятся широковещательные канальные технологии без групповой адресации. Типичный представитель этой группы — ныне вышедшая из употребления ЛВС ARCNET. У каждой станции *ARCNET* был свой канальный адрес длиной 8 бит. Кроме того, нулевой адрес был широковещательным. Однако групповых адресов канального уровня у *ARCNET* вообще не было. Поэтому групповые адреса сетевого уровня оставалось транслировать в широковещательный адрес *ARCNET*. Вполне естественно, что эта группа канальных технологий тоже страдает от неизбирательной доставки группового трафика.
- Третью группу составляют каналы «точка-точка», в которых вся адресация — неявная, например, PPP. Когда локальный узел передает пакет в такой канал, он тем самым как бы заявляет: «Пакет адресован не мне». Канальный уровень, за неимением других вариантов, делает такой вывод: «Раз не мне,

то удаленному узлу». И снова групповой пакет попадает на вход узлу, членство которого в группе возможно, но далеко не гарантировано.

- Наконец, четвертая группа — это каналы NBMA, в которых полноценное групповое вещание невозможно (см. §5.2). Точнее, в них возможно только ограниченное групповое вещание, когда узел рассматривает канал NBMA как множество каналов «точка-точка» и передает групповой пакет всем своим непосредственным соседям. То, что некоторые узлы канала вообще не получат пакет, хотя они, может быть, и состоят в данной группе, — проблема данного типа каналов, и мы с ней вряд ли что-то можем поделать, не меняя тип канала с NBMA на широковещательный при помощи дополнительного протокола (например, LANE в ATM или VPLS в MPLS). В то же время, и проблема неизбирательной доставки тоже остается, потому что узел-источник не обладает сведениями о членстве соседей в данной группе. Хотя такой режим вещания можно назвать групповым только условно, он может пригодиться, например, хостам для розыска маршрутизатора по умолчанию, так как этот маршрутизатор — всегда сосед хоста (см. §5.2).

Сухой остаток из этого сравнения таков: если канал вообще поддерживает групповое вещание, то он сделает все возможное, чтобы доставить групповой пакет всем членам группы, но он вправе пожертвовать избирательностью, доставив пакет и узлам, в данной группе не состоящим. Поэтому узел обязан фильтровать входящий групповой трафик по адресу назначения на всех доступных ему уровнях стека. Так как возможность фильтровать его на канальном уровне ограничена, в основном из-за рудиментарной групповой адресации, главная доля ответственности ложится здесь на сетевой уровень, то есть IP.

> Мы сформулировали это требование фильтрации пакетов на входе для группового трафика, потому что именно о нем сейчас идет речь. Однако на практике то же самое требование справедливо и для индивидуального трафика. Так, хост не должен полагать, что любой входящий пакет адресован именно ему. Хост может получать чужие пакеты по ошибке, например, из-за неправильных или устаревших маршрутов в сети. Требования устойчивости и безопасности в этом случае очевидны: прежде чем принять пакет в обработку, хост должен убедиться, что адрес назначения пакета — действительно его. Приведем аналогию из обычной жизни: перед тем как вскрыть найденный в почтовом ящике конверт, хорошим тоном будет проверить адрес получателя.

Теперь мы можем двинуться вверх по стеку, и наше внимание немедленно привлекает интерфейс между канальным и сетевым уровнями. Общепринятая модель инкапсуляции такова, что превратить групповой кадр в групповой пакет очень просто: достаточно удалить обрамление канального уровня.

> Строго говоря, нет абсолютной гарантии, что внутри принятого группового кадра окажется именно групповой, а не индивидуальный пакет. Как мы упоминали в примечании к §5.2, некоторые семейства протоколов за пределами TCP/IP даже пользуются этим трюком, чтобы сэкономить на явном разрешении индивидуальных адресов. К их числу относится, например, ISO.

В то же время, обратная операция может потребовать дополнительных усилий, если канальный уровень поддерживает явную адресацию. Какой канальный адрес назначения мы укажем в групповом кадре? Как раз здесь нам пригодятся правила разрешения групповых адресов IPv6 для данного типа канальной инкапсуляции, а они заведомо локальны и сводятся к какому-то вычислению (см. §4.1.1). Скажем, в *ARCNET* любой групповой адрес IPv6 превратится в адрес *0* [§7 RFC 2497], а в *Ethernet* это будет полученный подстановкой битов адрес *33-33-XX-XX-XX-XX* [§7 RFC 2464]. Хотя даже в

последнем случае отображение адресов далеко от взаимной однозначности, это не вызовет проблем благодаря обязательной фильтрации на входе.

> На этот аспект можно посмотреть и с другой стороны: потеря информации при разрешении групповых адресов IP в канальные адреса — еще одна весомая причина фильтровать входящие групповые пакеты по их адресу назначения. Это касается не только IPv6, но и IPv4. Так, групповой адрес IPv4 тоже отображается с потерей битов, например, в тот же нулевой адрес *ARCNET* [§4.3 RFC 1201] или в адрес MAC-48 *01-00-5E-XX-XX-XX* [§6.4 RFC 1112].

Наконец мы снова вынырнули на сетевой уровень. Здесь задача группового вещания IPv6 сводится к трем очевидным подзадачам, которые уже знакомы нам по индивидуальной передаче:

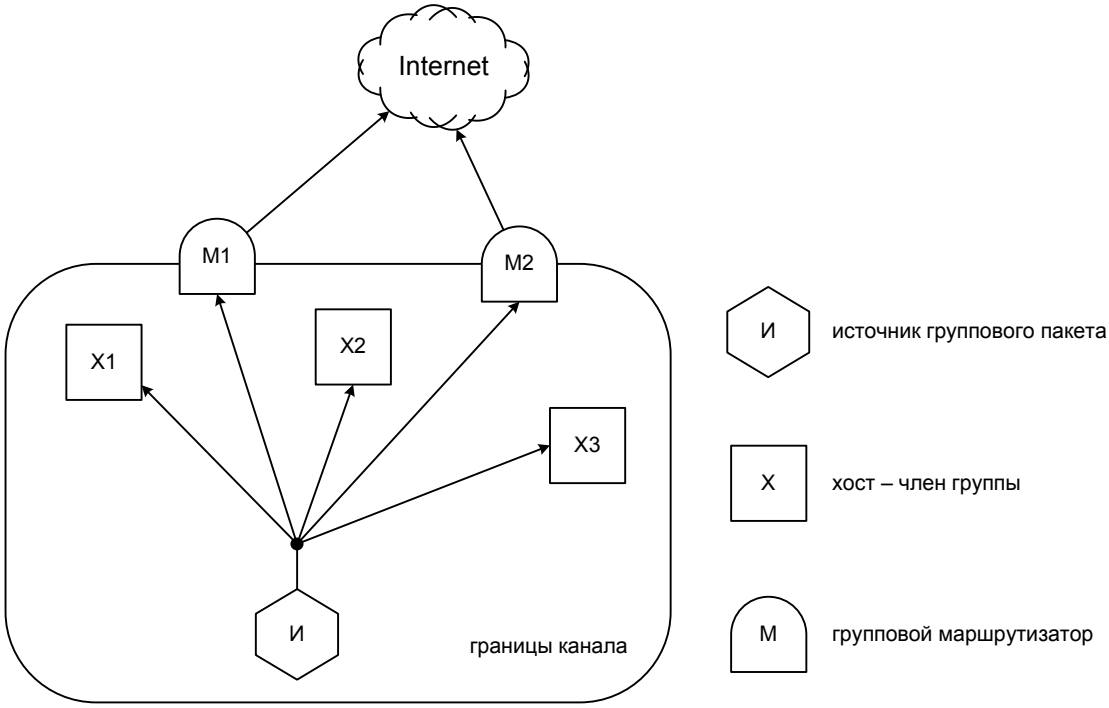
1) передача в пределах канала;
2) прием;
3) маршрутизация.

С передачей по каналу мы уже практически разобрались. Передающий узел (источник или маршрутизатор) выбирает по каким-то правилам выходной сетевой интерфейс, а затем просто отдает групповой пакет в ведение соответствующего канального модуля. Тот, если нужно, вычисляет канальный адрес назначения и — в добрый путь!

Процедура группового приема сложнее, потому что требует предварительной подготовки. Мы не раз повторяли фразу о том, что узел должен вступить в некую группу Г на сетевом интерфейсе И. Пришло время посмотреть, что за этим стоит на самом деле. В первую очередь, это означает: настроить входные фильтры канального и сетевого уровней так, чтобы адресованные группе Г пакеты, войдя через интерфейс И, достигали локального уровня IP и принимались им в обработку. На поверхности это выглядит так: узел вносит адрес Г в список групп данного сетевого интерфейса И. Поскольку членство в группах динамическое, то необходима и обратная операция, то есть покинуть группу, убрав адрес Г из списка групп на интерфейсе И.

> В групповом вещании IP передача и прием — операции независимые, так что вещать группе вполне может узел, в ней не состоящий. Членство в группе влияет только на прием.

Прекрасно, групповое вещание IPv6 по каналу уже работает. Теперь попробуем выйти за пределы одного канала и подумаем над маршрутизацией групповых пакетов. Допустим, у группы есть члены за пределами канала, к которому подключен источник. Понадобится ли источнику таблица групповых маршрутов, или хотя бы групповой маршрутизатор по умолчанию? Придется ли ему отдельно работать с маршрутизатором и членами группы на канале? На самом деле, всех этих сложностей можно избежать, если маршрутизатор сам притворится одним из членов группы и станет принимать групповой трафик наравне с ними (см. Фиг. 92), вместо того чтобы служить явным следующим шагом (*next hop*) групповых пакетов. Поскольку маршрутизатор на самом деле не потребляет принятые групповые пакеты сам, а продвигает их дальше, признать его полноценным членом группы мы не можем. Он — только **слушатель** (*listener*) данной группы. Действительные члены группы — тоже ее слушатели, однако не все слушатели группы — ее члены.



Фиг. 92. С точки зрения источника, групповые маршрутизаторы неотличимы от хостов

Чтобы такая схема работала, групповой маршрутизатор должен каким-то образом узнать, какие группы ему слушать и куда их продвигать. Конечно, эти сведения можно сообщить ему путем ручной настройки, но это плохо вяжется с динамическим характером группового вещания. Допустим, маршрутизатор соединяет несколько каналов: К1, К2, К3,… К*N*. Если текущий источник группы Г находится на канале К1, а слушатели есть только на К2, то маршрутизатор должен слушать группу Г, как минимум, на К1 и продвигать ее трафик в К2. Самое примитивное решение могло бы состоять в том, чтобы слушать группу Г на всех N каналах, а продвигать групповой пакет в $N-1$ каналов, за исключением того, откуда пришел данный пакет. Это напоминает нам процесс лавинной рассылки (*flooding*) в работе коммутатора ЛВС.

Возможно ли оптимизировать этот процесс? Как мы помним, обучаемый коммутатор ЛВС извлекает информацию о местонахождении узла из того, через какой интерфейс (порт) приходят от него кадры, в предположении, что путь к данному узлу совпадает с путем от него. Поэтому коммутатор делает, к примеру, такой вывод: «Если кадр от источника У2 пришел через интерфейс И5, то впредь я стану продвигать кадры, адресованные У2, только в интерфейс И5». Но, увы, сейчас нам от этого трюка нет никакой пользы. Ведь входной интерфейс группового пакета не дает маршрутизатору никаких сведений о том, где находятся *слушатели* группы. Корень проблемы здесь кроется в том, что слушатели никак себя не проявляют, а косвенных признаков явно недостаточно. Во-первых, источник группового пакета — далеко не всегда слушатель этой группы. Во-вторых, групповой адрес никогда не возникнет в поле «источник», потому что архитектура IP не допускает коллективного авторства пакетов. В-третьих, не все слушатели группы — ее члены и могут говорить от имени группы.

> Кадр, на котором обучается коммутатор ЛВС, может быть индивидуальным или групповым, но адрес источника в нем однозначно индивидуальный, и потому процесс обучения тоже ограничен только индивидуальными адресами. Следовательно, та же самая проблема управления групповым трафиком стоит и перед коммутаторами. Мы к этому вопросу еще вернемся.

Чтобы восполнить этот недостаток информации, слушателям группы придется явным образом заявить о себе, а для этого им понадобится соответствующий протокол. Так как это будет очередной аспект управления IPv6, мы поместим его в удобные рамки ICMPv6 и назовем **розыск групповых слушателей** (*Multicast Listener Discovery*, **MLD**).

В то же время, источники группового трафика обнаружат себя, передавая пакеты, и дополнительный протокол их розыска не требуется. Групповому маршрутизатору достаточно настроить свои активные сетевые интерфейсы так, чтобы они принимали все групповые пакеты. Например, пока маршрутизатор работает только с IPv6 и *Ethernet*, ему достаточно принимать все кадры MAC, у которых адрес назначения начинается на *33-33*. Такой теневой прием всего группового трафика еще не делает маршрутизатор слушателем всех возможных групп. Групповой маршрутизатор слушает группу только тогда, когда он готов продвигать для нее трафик. Это различие важно, так как настоящий слушатель должен заявить о себе с помощью MLD, а не только втихомолку принимать пакеты.

> Аналог MLD в IPv4 — это протокол IGMP. Его современные версии основаны на точно тех же идеях, а сообщения IGMP играют те же роли, что сообщения MLD. Более того, версии IGMP и MLD фактически синхронизированы, поскольку групповое вещание IPv4 и IPv6 развивается параллельно. Так, IGMPv2 отвечает MLDv1, а IGMPv3 — MLDv2. Что скрывается за этими номерами версий, мы сейчас увидим.

Работу над MLD мы начнем с того, что уточним, какой объем сведений о слушателях группы Г на канале К действительно необходим маршрутизатору. Будет ли это поименный список? Когда маршрутизатор продвигает групповой пакет в канал К, он передает в этот канал ровно одну копию пакета, а всю остальную работу делают канальные механизмы. Поэтому на самом деле групповому маршрутизатору достаточно знать, есть ли вообще слушатели группы Г на данном канале К, а их число и состав совершенно неважны. Это простое соображение и послужит нашей отправной точкой.

> В идеале групповой маршрутизатор оперирует именно каналами, а не интерфейсами, чтобы избежать ненужного дублирования пакетов, а в худшем случае и лавины группового трафика. Такую лавину может вызвать конфигурация с более чем двумя подключениями к каналу, в которой маршрутизатор получает обратно несколько копий группового пакета, который он только что продвинул в канал, и поступает с ними так же. Если несколько интерфейсов маршрутизатора подключены к одному и тому же каналу, то для оптимальной работы эту особенность надо как-то отразить в настройках маршрутизатора, но как именно это сделать, остается на усмотрение реализации.

Следующее очевидное соображение состоит в том, что областью распространения MLD будет один канал. Поэтому адресам источника и назначения в сообщениях MLD прекрасно подойдет внутриканальная область. Кроме того, надежность передачи пакетов в пределах канала обычно удовлетворительная, и нам можно не обременять себя и протокол средствами гарантированной доставки данных, а будет достаточно повторять сообщения время от времени.

> Любителям истории TCP/IP будет небезынтересно отметить, что самая ранняя версия IGMP [Приложение I RFC 988], условно известная как IGMPv0, как раз обеспечивала гарантированную доставку сообщений от слушателя к маршрутизатору. На практике оказалось, что без этого можно обойтись и тем упростить реализацию сторон протокола. Ведь гарантированная доставка неизбежно требует хранить историю передачи, тогда как периодический повтор вполне обходится без сохраненного состояния.

В первом приближении, наша пробная версия MLD могла бы работать следующим образом. Слушатели шлют **отчеты** (*Report*) о принимаемых группах, просто перечисляя адреса этих групп. Маршрутизатор вызывает отчеты слушателей, направив в канал явный **запрос** (*Query*), хотя слушатель может выслать отчет и по собственной инициативе. Чтобы на первых порах не заботиться о размере отчета в контексте MTU, пусть каждый отчет содержит ровно один групповой адрес. Если слушатель принимает несколько групп, то он вышлет по отдельному отчету о каждой из них. Кто в этой схеме будет задавать ритм? Пока на канале нет группового маршрутизатора, в периодических отчетах смысла тоже

нет. Поэтому пусть именно маршрутизатор периодически высылает запрос, а слушатели отвечают на него.

> Конечно, ради выравнивания нагрузки на канал слушателю следует слегка задерживать объявление по запросу на случайное время, а повторные сообщения надо рассылать через случайные интервалы.

Тем не менее, в одном случае оправдан немедленный и добровольный отчет о группе, а именно когда новый слушатель начинает принимать данную группу. Вот нам еще одно важное действие хоста при вступлении в группу Г на интерфейсе И: направить отчет MLD о группе Г в интерфейс И. Чтобы отчет не потерялся, его стоит повторить несколько раз.

Какой адрес источника IPv6 будет указан в пакете с отчетом? Если у интерфейса И уже есть внутриканальный адрес, то надо использовать его. Однако на момент вступления в группу Г интерфейс И может быть еще без адресов, например, если группа Г — это группа искомого узла, которая отвечает пробному внутриканальному адресу интерфейса И (см. процедуру DAD в §5.4.1). Выходит, чтобы вступить в группу, нужен адрес, а чтобы назначить адрес, нужна группа? Это типичная «проблема курицы и яйца», для которой у нас уже есть типовое решение: в пакете с отчетом MLD допустим неопределенный адрес источника *::* [RFC 3590, §5.2.13 RFC 3810]. Это вполне приемлемо для MLD, поскольку маршрутизатор все равно не ведет поименного списка слушателей группы.

> Все группы искомого узла — внутриканальные. Тогда зачем их вообще объявлять по MLD? Как станет видно через пару абзацев, это весьма резонный вопрос, располагающий к небольшой беседе.

В такой схеме работы MLD запросу не нужны никакие дополнительные параметры, а отчет содержит всего одно информативное поле, адрес группы. Но как маршрутизатор определит, что у группы Г не осталось слушателей, когда все они прекратят ее прием? В этом случае запрос маршрутизатора не вызовет ни одного отчета об этой группе. Так как запросы и отчеты все же могут теряться, маршрутизатору не следует спешить и надо подождать еще несколько запросов подряд. А уж если и в ответ на них не пришло ни одного отчета, в котором адрес группы равен Г, то значит, ее и правда больше никто не слушает.

Наконец, нам следует определить, по какому адресу надо слать запрос и отчет. Запрос должен достичь всех узлов канала, так как заранее неизвестно, кто из них групповой слушатель. Поэтому запрос направляется группе «все узлы канала», *FF02::1*. Чтобы при этом не возникло очередной «проблемы курицы и яйца», пусть эта группа будет особенной: она никогда не объявляется по MLD. Так как она внутриканальная, то и маршрутизации она не подлежит.

> На первый взгляд, можно было бы назначить особый групповой адрес «все слушатели канала». Однако групповое вещание в IPv6 — обязательная функция, которую в любом случае применяет протокол ND (§5), так что выигрыша от такого назначения не было бы вовсе, потому что любой интерфейс IPv6 — член нескольких групп. (Перечислите, каких.)
>
> Маршрутизатор вправе продвинуть индивидуальный пакет в тот же канал, откуда он пришел, даже если адрес назначения в нем внутриканальный; ведь зонная архитектура IPv6 вполне допускает такое поведение (см. §2.4). В то же время, продвигать обратно в канал групповой пакет точно не следует, потому что это прямая дорога к зацикливанию трафика. А если групповых маршрутизаторов на канале будет несколько, то может даже возникнуть лавинный эффект, когда число копий пакета растет с каждым циклом экспоненциально. Чтобы понизить вероятность такого сбоя, «предельное число шагов» в пакетах MLD устанавливают равным 1, а значит, не применяют к ним GTSM (§5.1).

Выходит, если трафик внутриканальных групп все равно не подлежит маршрутизации, то и сведения о них маршрутизатору не нужны. Может, их надо вообще исключить из объявлений MLD? На самом деле, нет. Дело здесь в том, что отчеты о внутриканальных группах интересны интеллектуальным коммутаторам ЛВС, которые заняты «подслушиванием» MLD. До этой темы мы доберемся буквально через несколько абзацев.

Что касается групп из области интерфейса (*FFx1*:…), то они имеют значение только в пределах данного интерфейса, а адресованные им пакеты никогда не покидают пределов узла. Поэтому естественно, что группы из этой области никогда не упоминаются в сообщениях MLD. То же справедливо для групп зарезервированной области `0`.

Что касается отчета, то у нас целых три кандидата на адрес его назначения: индивидуальный адрес маршрутизатора, который был источником запроса, группа «все маршрутизаторы канала» и… группа Г, о которой дается отчет. Первый кандидат, то есть адрес маршрутизатора, плох тем, что отчет не получат другие маршрутизаторы того же канала, когда они есть, а слушателям придется отвечать на запрос каждого маршрутизатора отдельно. Маршрутизаторы у нас уже получают запросы друг друга, и они могли бы выбрать, кто из них служит «метрономом» данного канала (**генератор запросов**, *querier*), чтобы понизить суммарную нагрузку. Вопрос о точном механизме этих выборов мы пока отложим в наш «внутренний стек», но сама идея оптимизации довольно прозрачна: только один маршрутизатор запрашивает, но отчеты получают все. Второй кандидат, «все маршрутизаторы канала», допускает такую оптимизацию, но вовлекает в прием отчетов маршрутизаторы индивидуального трафика, которым эти отчеты неинтересны. Наконец, группа Г, о которой дается отчет, позволит сообщению достичь всех групповых маршрутизаторов, потому что они уже ведут теневой прием всех возможных групп на всех своих активных интерфейсах; ведь именно так маршрутизаторы следят за источниками группового трафика. Кроме того, отчет получат другие слушатели группы Г, а это может пригодиться для дополнительной оптимизации трафика MLD. Поэтому пусть отчет о группе Г адресуется этой же самой группе — по крайней мере, в нашей самой первой версии MLD.

Суть дополнительной оптимизации вот в чем. Раз маршрутизатору неинтересен поименный список слушателей, то о каждой группе достаточно одного отчета на весь канал. Пока слушатель задерживает свой отчет о группе Г на случайное время, он заодно ожидает, не придут ли отчеты о Г от других ее слушателей, у которых задержка оказалась короче. Если такой отчет получен, значит, кто-то другой успел отрапортовать о группе Г первым, и повторный отчет избыточен. Как мы скоро увидим, у этого трюка есть и обратная, отрицательная сторона: коммутаторам сложнее следить за топологией группы в пределах ЛВС.

На схеме, которую мы наметили под видом «пробной версии MLD», основана работа протокола IGMPv1 в IPv4 [Приложение I RFC 1112]. В нем выборы генератора запросов оставлялись вышестоящему протоколу групповой маршрутизации.

В результате у нас получился протокол, в котором групповой слушатель просто заявляет о своем текущем состоянии, сообщая адреса всех интересных ему групп. Маршрутизатор своим запросом способен только вызвать отчеты слушателя, но никак не влияет на их содержание. Фактически, у нашей текущей версии MLD нет полноценного состояния, в том смысле, что очередное сообщение стороны протокола не влияет на долговременное состояние других сторон и не изменяет текста будущих сообщений. Условно говоря, стороны не ведут диалог, а бросают отдельные независимые реплики. Следовательно, принимать и обрабатывать отчеты слушателей легко может сторонний узел, которому даже не нужно следить за встречным потоком запросов, потому что тот не содержит никакой полезной информации.

Позднее мы увидим, как прием запросов наряду с отчетами позволяет точнее управлять таймерами протокола. Тем не менее, для правильной интерпретации отчета MLD знать текст запроса совершенно не нужно. Более того, отчет MLD может быть и не по запросу, например, когда слушатель начинает прием новой группы.

Это свойство MLD открывает нам путь к оптимизации группового трафика не только на сетевом, но и на канальном уровне. Скажем, интеллектуальный коммутатор *Ethernet* вполне может следить за отчетами MLD, которые входят через его порты, и таким образом вести учет слушателей групп на этих портах. Конечно, тогда коммутатору придется нарушить границу между уровнями в стеке протоколов, потому что он будет сначала выделять пакеты IPv6 с отчетами MLD среди всех возможных кадров *Ethernet*, а затем анализировать их. Последним шагом будет преобразование групповых адресов IPv6 в групповые адреса MAC, так как коммутатор все же управляет трафиком на основании канальных, а не сетевых адресов. Тем не менее, этот трюк вполне по силам современным коммутаторам, вычислительные возможности которых давно превосходят самые смелые мечты пионеров ЭВМ. Поскольку коммутатор в такой схеме подслушивает чужие сообщения, хотя ему это и не положено по его сетевой роли, данный прием известен как **подслушивание MLD** (*MLD snooping*).

Конечно, тот же самый прием коммутаторы используют и по отношению к IPv4, подслушивая IGMP.

Обратите внимание, какую роль в подслушивании IGMP и MLD играют правила преобразования групповых адресов IP в групповые адреса MAC. Они сводятся к простой подстановке битов (§4.1.1) и потому не представляют трудности для коммутатора. Иначе коммутатор не смог бы воспользоваться информацией из сообщений IGMP или MLD, потому что не знал бы, какой канальный адрес отвечает данной группе IP.

Хотя с архитектурной точки зрения подслушивание MLD и IGMP — всего лишь сомнительный трюк, на практике оно стало самым популярным подходом к управлению групповым трафиком IP на канальном уровне. Даже общепринятые протоколы вынуждены считаться с ним. В частности, слушатели обязаны объявлять свои внутриканальные группы, кроме *FF02::1* и *224.0.0.1*, и это явная дань подслушиванию MLD и IGMP.

Предложите способ управления на канальном уровне группами «все узлы IP», *FF02::1* и *224.0.0.1*, при том что они не объявляются слушателями. Иными словами, по каким признакам коммутатор может своевременно установить, что на данном порту есть хотя бы один узел IPv4 или IPv6? (Подсказка: ARP с протоколом 0x800 или BOOTP/DHCPv4 — это IPv4; ND или DHCPv6 — это IPv6.)

Получив работоспособный прототип MLD, мы можем заняться его оптимизацией. Прежде всего, обратим внимание на одну серьезную проблему, которую мы сами же и создали. Когда слушатель группы Г направляет отчет MLD по адресу Г, он рассчитывает, что этот отчет примут и обработают все групповые маршрутизаторы канала, а не только слушатели группы Г. Это различие важно, например, когда группа Г впервые появляется на данном канале и маршрутизаторы о ней еще ничего не знают. Чтобы эта схема сработала, маршрутизаторам будет недостаточно настроить свои канальные фильтры на прием всех групповых пакетов, — им также придется вести анализ всех входящих групповых пакетов, и анализ этот будет более глубоким и трудоемким, чем того требует обычная адресация IPv6. Ведь до сих пор узел, будь то хост или маршрутизатор, заглядывал дальше основного заголовка IPv6 только в том случае, если адрес назначения IPv6 в основном заголовке пакета принадлежал узлу.

В IPv4 это было, строго говоря, не так, поскольку маршрутизатор просматривал все опции IP.

В IPv6 из этого правила есть ровно одно исключение — пошаговые опции. Мы вспомним о них и тут же воспользуемся ими через пару абзацев.

Обычно, если пакет был индивидуальный, его адрес назначения должен был быть назначен одному из интерфейсов узла; если же пакет был групповой, то узел должен был быть членом данной группы на входном интерфейсе пакета. В противном случае пакет был чужим; хост отбраковывал его, а маршрутизатор продвигал согласно текущей политике. Этот простой критерий «свой/чужой» позволял эффективно выделять пакеты, требующие анализа, из всего трафика, что особенно важно для маршрутизаторов. Теперь же мы фактически обязали групповые маршрутизаторы полностью разбирать всю цепочку заголовков IPv6 в каждом входящем групповом пакете, чтобы выделить из общего потока сообщения ICMPv6, а затем анализировать заголовок ICMPv6, чтобы отделить и обработать интересные маршрутизатору сообщения MLD.

Вспомним важный факт, который мы узнали из §3.1: в отличие от IPv4, тип полезной нагрузки пакета IPv6 не всегда можно определить прямо из его основного заголовка.

Простой, но действенный прием против этой проблемы сводится к тому, чтобы заранее помечать пакеты MLD. Делать это должен, конечно же, их источник, но как? Среди всех значений «следующий заголовок» в §3.3.2 мы особо выделили одно, а именно нулевое. Это значение отвечает заголовку пошаговых опций, представляющих интерес для всех узлов по пути пакета. Его особенность в том, что встретиться оно может только в основном заголовке IPv6 ради легкого доступа к пошаговым опциям.

Почему бы нам не воспользоваться этим механизмом для маркировки служебных пакетов, нарушающих правила адресации? Ведь на самом деле маршрутизаторы IPv6 уже проверяют, нет ли в транзитном пакете пошаговых опций, а стоимость этой проверки низка, потому что ограничена основным заголовком пакета. Просто до сих пор у нас не возникало задач, где бы это пригодилось. Итак, пришло время «сконструировать» пошаговую опцию, которая скажет маршрутизатору: обрати внимание на этот пакет, он может содержать интересные тебе сведения, хотя он и не адресован тебе явным образом. За свою роль эта опция называется «сигнал маршрутизатору» (*Router Alert*) [RFC 2711].

Сначала нам надо выбрать численный тип этой опции. Как мы помним из §3.3.2, три старших бита в нем отражают свойства опции. Первое свойство — это можно ли игнорировать опцию, если узел не поддерживает ее. В данном случае это так. Второе свойство — неизменность. У транзитных узлов нет никакой причины изменять значение данной опции, поэтому она будет неизменной. Такой комбинации свойств отвечают три нулевых бита, `000`, в старших разрядах типа опции, а значит, значение надо выбрать из диапазона от 0 до 31. Общепринятый тип этой опции — 5.

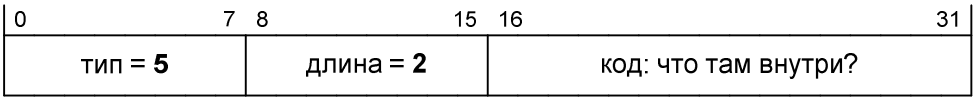
Какие данные будет содержать эта опция, и нужны ли они ей вообще? Хотя само ее присутствие — это уже сигнал для маршрутизатора, давайте оптимизируем наш подход и точнее укажем тип сведений, которые маршрутизатор сможет извлечь из данного пакета (см. Фиг. 93). По сути, это классификация протоколов под несколько другим углом зрения. Если протоколы IP (значения для поля «следующий заголовок») делают ударение на инкапсуляции, то сейчас мы перенесем его на способы управления трафиком. Каждый протокол управления получит свой код из соответствующего реестра [116], и тогда маршрутизатор сможет быстро узнать, интересен ли ему этот пакет, без того чтобы разбирать всю цепочку заголовков. Протокол MLD пришел за своим кодом самым первым и потому получил почетное нулевое значение. Для простоты и определенности пусть все сообщения MLD несут пошаговую опцию «сигнал маршрутизатору» с кодом 0 внутри.

Чтобы проще объяснить, зачем понадобился еще один реестр протоколов, применим кулинарную параллель: опция «сигнал маршрутизатору» не только

[116] http://www.iana.org/assignments/ipv6-routeralert-values

выделяет пирожки с начинкой среди обычных булочек, но и говорит, какая там начинка внутри. Ведь гурману важен не способ начинки, а ее вкус.

Обсудите, как применение опции «сигнал маршрутизатору» влияет на устойчивость и безопасность сети, а также что этому можно противопоставить [§4 RFC 2711].



Фиг. 93. Пошаговая опция IPv6 «сигнал маршрутизатору»

Поле кода внутри опции «сигнал маршрутизатору» — длиной 2 байта. Как читатель уточнит его спецификацию, чтобы она стала однозначной? (Ответ: способ кодирования целый беззнаковый, порядок байтов сетевой.)

Упражнение для читателя: составить заголовок пошаговых опций, содержащий *только* опцию «сигнал маршрутизатору» с кодом MLD, и заполнить все поля. Сможете ли вы это сделать? Если нет, то как обойти возникшее затруднение? (Ответ: придется добавить два **Pad1** или один пустой **PadN**.)

Перейдем к следующему усовершенствованию. В предварительной версии MLD мы отложили вопрос о том, как групповые маршрутизаторы канала выберут ровно одного из своего числа на роль **генератора запросов** (*querier*). Вот простой и остроумный подход к этой, казалось бы, сложной задаче: пускай роль генератора играет маршрутизатор с наименьшим адресом IP.

Мы, конечно же, помним из §2.2, что адрес IP (v4 или v6) — это цепочка битов с определенным порядком старшинства, а значит, ее можно рассматривать как двоичную запись целого неотрицательного числа. Однозначность позиционной записи чисел гарантирует нам, что соответствие между побитовым представлением адреса и его численным значением взаимно однозначно, а значит, у разных адресов заведомо разное численное значение. Точнее говоря, позиционная запись однозначна с точностью до незначащих нулей, но адресов IP эта оговорка не касается ввиду их фиксированной разрядности. Если бы адреса IP были переменной длины, нам пришлось бы явным образом уточнить, имеют ли значение нули в старших разрядах адреса, например, являются ли 1 и 01 суть разными адресами или же эквивалентными представлениями одного и того же адреса. Разумеется, все эти соображения справедливы только в пределах определенной версии IP.

Чуть выше мы позаботились, чтобы маршрутизаторы получали запросы друг друга. Каждый из них шлет свои запросы с определенного внутриканального адреса, который, несомненно, уникален в пределах канала — об этом позаботился механизм DAD из §5.4.1. Теперь представим себе, что маршрутизатор получил запрос, адрес IP источника в котором численно меньше, чем его собственный. Это значит, что есть более вероятный претендент на роль генератора, и собственные запросы надо приостановить на какое-то время, большее, чем стандартный период повтора запросов. В результате самый подходящий кандидат станет непрерывно подавлять своими запросами другие маршрутизаторы, и те будут оставаться пассивными **наблюдателями** (*non-querier*). Математическая основа этого трюка — транзитивность отношений «больше» и «меньше» на множестве целых чисел: $A > B, B > C \Rightarrow A > C$. Благодаря этому свойству процесс выборов гарантированно сойдется, как только все маршрутизаторы получат все неподавленные запросы своих соперников. Когда же текущий генератор выйдет из игры, сработают таймеры, процедура выборов повторится, и место генератора займет новый победитель. В результате мы получаем простой и устойчивый механизм.

Очевидно, что в общем случае подобные выборы можно провести, используя любой уникальный идентификатор и любое транзитивное отношение над ним.

Так как у наблюдателя может быть несколько внутриканальных адресов на данном интерфейсе, для сравнения он выбирает и использует именно тот из них, с которого он слал бы — и шлет во время выборов — свои собственные запросы.

Еще один заслуживающий оптимизации аспект — это окончание приема группы. Информация о начале приема группы не задерживается, потому что новый слушатель первым делом высылает добровольный отчет об этой группе. Напротив, окончание приема проходит втихомолку. Маршрутизатор делает вывод о том, что все слушатели прекратили прием группы, когда об этой группе давно не было отчетов. Сколько времени ему на это понадобится? Запрос к слушателям — довольно дорогая операция, так как сначала пакет с запросом направляется всем узлам канала, а затем слушатели разных групп отвечают на него одним пакетом на каждую группу. Поэтому период повтора запросов должен быть достаточно большим, порядка минут. Следовательно, и уход всех слушателей будет обнаружен не раньше, чем через несколько минут.

Чтобы ускорить этот процесс, пусть слушатель явным образом извещает маршрутизаторы о прекращении приема данной группы Г. Для этого он высылает сообщение нового типа, **итог** (*Done*). Как и отчет, это сообщение содержит в себе адрес группы Г, о которой идет речь. По какому адресу лучше всего направить итог? Здесь у нас пока всего два варианта, «все маршрутизаторы канала», *FF02::2*, и адрес данной группы Г. Обычно маршрутизаторов на канале меньше, чем потенциальных слушателей группы, а сообщения типа «итог» будут относительно редки, так что мы остановим свой выбор на первом варианте, *FF02::2*.

Теперь представим себе, что групповые маршрутизаторы канала получили итоговое сообщение о группе Г от некоего слушателя. Поскольку маршрутизаторы не ведут поименный список слушателей и даже не считают их «по головам», то само по себе это сообщение еще не дает им всей необходимой информации — оно только извещает, что произошли изменения в числе слушателей группы Г. Естественной реакцией маршрутизаторов будет проверить, остались ли в группе Г еще слушатели. Для этого генератору запросов надо послать в канал несколько запросов с короткими паузами между ними.

Если это будут обычные запросы, они вызовут целый поток информации, не относящейся к делу. Поэтому нам следует ограничить эти запросы группой Г, о которой пришло итоговое сообщение. Во-первых, такие запросы надо направлять по адресу группы Г, а не «все узлы канала». Во-вторых, в самих запросах надо указать, что они касаются только группы Г. То есть тело запроса тоже должно включать в себя поле «адрес группы». В обычном, **общем запросе** (*General Query*) значение этого поля будет нулевым, тогда как в **запросе, ограниченном группой** (*Multicast-Address-Specific Query*), это поле содержит действительный адрес группы.

Мелкая, но важная деталь — это взаимодействие между процедурой быстрой проверки группы и выборами генератора запросов. Как быть текущему генератору, если он еще не послал положенное число запросов группе Г, а тем временем пришел чужой запрос с еще меньшим адресом источника? Пусть ради простоты и устойчивости протокола текущий генератор всегда доводит процедуру быстрой проверки до конца. Ведь выборы генератора — это лишь оптимизация, и не будет никакой беды в том, что на протяжении нескольких секунд запросы будут слать два маршрутизатора.

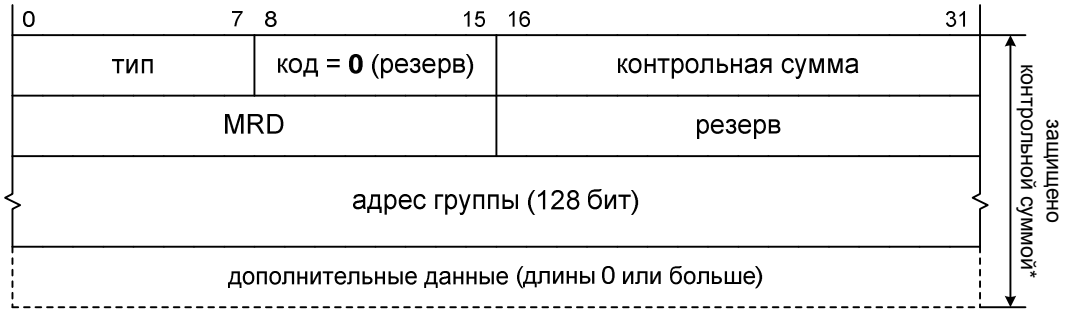
Последний штрих в нашей схеме быстрой проверки группы коснется координации генератора запросов с наблюдателями. Как последние узнают, что идет оперативная проверка группы и связанные с ней таймеры надо установить на более короткий интервал? По нашему плану, наблюдатели продолжают получать запросы, а ритм задает генератор запросов, так что пускай он заодно сообщает наблюдателям, какой тайм-аут надо связать с данным запросом. Для этого достаточно одного целочисленного поля в формате запроса. В свою очередь, слушателям оно пригодится, чтобы равномернее

распределить отчеты в доступном времени, выбирая случайную задержку от нуля до значения этого поля. За эту роль данное поле известно как **MRD** (*Maximum Response Delay*, максимальная задержка отклика). Размерность значения **MRD** — миллисекунды.

Маршрутизаторы обращают внимание на поле **MRD**, только если принят запрос, ограниченный группой. Общие запросы на интервалы таймеров не влияют.

Строго говоря, слушателям следует вносить поправку на задержку передачи по каналу, выбирая задержку отчета на основании **MRD**. Ведь иначе отчет может запоздать.

Перейдем к формату сообщений. Формат запроса и ответа будет одним и тем же, чтобы упростить реализацию (см. Фиг. 94). Конечно, поле **MRD** в ответе или итоге не несет никакой смысловой нагрузки. Длина такого унифицированного сообщения фиксирована и составляет 24 байта. Как быть, если пакет содержит еще какие-то данные вслед за фиксированным сообщением? Эту возможность мы узаконим, но оставим для будущих расширений протокола. На входе дополнительные данные надо просто игнорировать, а на выходе пока что никаких дополнительных данных быть не должно.



*) Не забудьте: контрольная сумма также защищает псевдозаголовок.

Фиг. 94. Сообщение MLDv1

«Сконструированные» нами механизмы составляют костяк первой версии протокола розыска слушателей IPv6, **MLDv1** [RFC 2710].

Тем же самым образом работает IGMPv2 в IPv4 [RFC 2236].

Любопытно отметить, что у сообщений MLDv1 типы численно меньше, чем у сообщений ND. Это можно объяснить тем, что MLD логически предшествует ND. К примеру, группу искомого узла надо объявить по MLD до того, как ею можно пользоваться в целях ND.

Мы провели довольно много времени, пересматривая и оттачивая детали протокола MLDv1, так что давайте соберем самые главные его черты в одной таблице, Табл. 17, а заодно сравним его с нашей пробной «нулевой версией», аналогичной IGMPv1 в IPv4, чтобы лучше увидеть проделанный путь.

Табл. 17. Сравнение возможностей «MLDv0» и MLDv1

	“MLDv0”	**MLDv1**
Типы сообщений ICMPv6	«запрос MLD» (130), «отчет MLD» (131)	«запрос MLD» (130), «отчет MLD» (131), «итог MLD» (132)
Виды запроса	Только общий	Общий и ограниченный группой
Форма отчета	Одна группа на сообщение	Одна группа на сообщение
Адрес назначения в отчете	Объявляемая группа	Объявляемая группа
Выборы генератора запросов	Механизм не определен	По адресу IPv6
Подавление избыточных отчетов	Да	Да

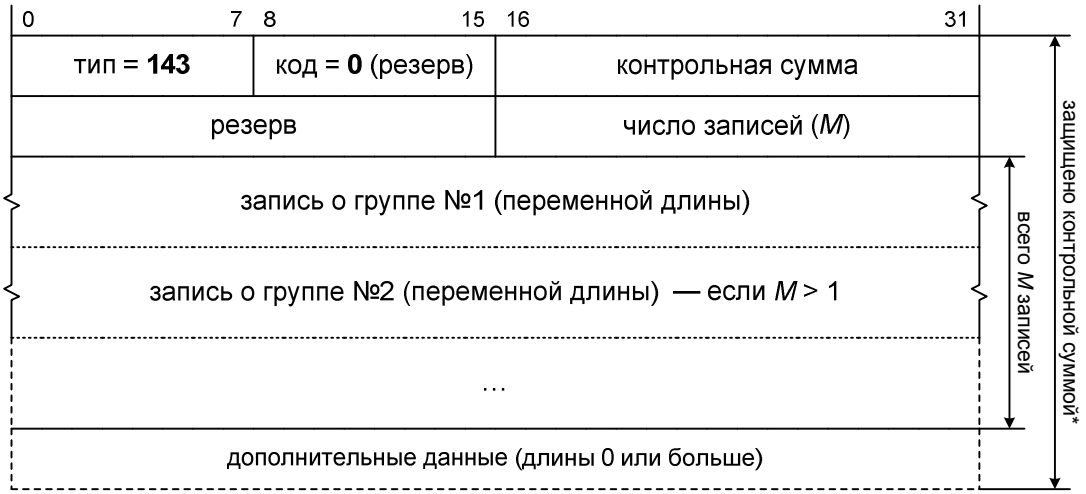
Управление интервалом таймера	Нет	Да

Способны ли мы еще улучшить наш результат во второй версии протокола, **MLDv2** [RFC 3810]? Конечно! Мы даже готовы немедленно выступить с черновым списком усовершенствований:

1. Центральным объектом MLDv1 выступает группа. В частности, сообщения MLDv1 относятся к одной группе (или ко всем группам сразу), а механизм подавления избыточных отчетов достигает того, что о каждой группе дается один отчет. Следовательно, накладные расходы на работу такого протокола растут пропорционально числу активных групп, а оно может быть велико, особенно если на канале в основном расположены маршрутизаторы, каждый из которых продвигает трафик многих разных групп. Напротив, число слушателей на канале вряд ли будет очень большим из чисто практических соображений. Поэтому имеет смысл перенести фокус протокола с группы на слушателя. Для начала надо переработать формат отчета так, чтобы он смог нести информацию о многих группах сразу.
2. MLDv1 поддерживает только один режим группового вещания, а именно ASM. В новой версии MLD совершенно необходима поддержка SSM и SFM. О различиях между этими режимами см. §2.9.
3. В MLDv1 механизм координации генератора запросов с наблюдателями находится в зачаточном состоянии. Его надо распространить на прочие важные параметры протокола: интервал между запросами, число повторов.

   Аналогичные усовершенствования вошли в IGMPv3 для IPv4 [RFC 3376].

Первый пункт мы осуществим, усложнив формат отчета (см. Фиг. 95). Теперь он будет содержать переменное число записей о группах. Однако тогда нам придется пересмотреть адрес назначения отчета MLDv2. В MLDv1 это был адрес группы, о которой дается отчет, поскольку он был ровно один. Когда отчет дается о переменном числе групп, нам ничего не остается, как адресовать его фиксированной общепринятой группе «все маршрутизаторы с поддержкой MLDv2», *FF02::16*. Другие слушатели больше не получат этот отчет, а значит, механизм подавления избыточных отчетов работать не сможет. Его ценность уже упала благодаря понижению числа отчетов, так что в MLDv2 мы от него полностью откажемся. Пусть каждый слушатель MLDv2 говорит сам за себя, без оглядки на других. Это заодно упростит реализацию слушателя.



*) Не забудьте: контрольная сумма также защищает псевдозаголовок.

Фиг. 95. Отчет MLDv2

Еще один аргумент против подавления избыточных отчетов в MLDv2 связан с подслушиванием MLD, которое ведут интеллектуальные коммутаторы ЛВС [§A.2 RFC 3810]. В простейшем случае такой коммутатор продвигает групповые кадры только в те порты, откуда были замечены отчеты MLD о данной группе. Если же отчет подавлен, то коммутатор решит, что на этом порту слушателей нет, и продвигать групповые кадры туда не станет. Чтобы обойти эту проблему, коммутатор, в свою очередь, подавляет продвижение отчетов MLD в те порты, где, как ему кажется, нет групповых маршрутизаторов [§2.1.1 RFC 4541]. То же справедливо для IGMP. Не стоит сомневаться, что такой трюк только усложняет настройку сети и дестабилизирует ее работу.

Родственная проблема состоит в том, что коммутатор должен копировать весь групповой трафик в порты, где есть групповые маршрутизаторы, поскольку те ведут теневой прием всех групп, не вступая в них явным образом. В противном случае, например, маршрутизаторы не смогут получать отчеты MLDv1. Увы, коммутатор не может надежно обнаружить групповые маршрутизаторы по их запросам MLD, так как только генератор запросов ведет их постоянную передачу, а наблюдатели молча следят за чужими сообщениями MLD. Поэтому, чтобы коммутатор мог самостоятельно определить, на каких портах есть групповые маршрутизаторы, предложен отдельный простой протокол **MRD** (*Multicast Router Discovery*) [RFC 4286].

Объединение информации о многих группах в один отчет может привести к тому, что полная длина пакета с отчетом превысит MTU канала, а фрагментировать отчет было бы нежелательно. Чтобы обойти и эту трудность, один длинный отчет о множестве групп будет эквивалентен нескольким отчетам поменьше, каждый о подмножестве этого множества, вплоть до одного отчета на группу. Это позволит слушателю разбить длинный список групп на порции подходящего размера, не применяя никаких дополнительных механизмов, а просто управляя числом групп на отчет [§5.2.15 RFC 3810].

Перейдем ко второму пункту. Главная трудность здесь в том, что теперь каждый слушатель потенциально способен сделать сложный заказ: эти источники я принимаю, а те блокирую… Как говорится: «Здесь играть, здесь не играть, а здесь рыбу заворачивали». Задача маршрутизатора в том, чтобы по возможности удовлетворить все эти запросы сразу, расходуя минимум своих ресурсов.

Так как слушатель все равно фильтрует входящий групповой трафик, проникновение в канал пакетов от нежелательного источника не вызовет никаких проблем, кроме расхода ресурсов сети. Напротив, заблокировать желательный источник означало бы сделать невозможным какое-то нужное и запланированное сетевое взаимодействие. Поэтому маршрутизатор имеет право на погрешность, но только в сторону разрешения трафика. Чтобы гарантировать это, мы построим определенную модель фильтрации группового трафика по адресу источника и заставим обе стороны MLD следовать ей.

Для начала заметим, что для маршрутизатора нет принципиальной разницы между SSM и SFM. Это хост знает, что в режиме SSM разные источники S в комбинированном адресе (S,G) означают разные каналы вещания, которые могут принимать совершенно разные приложения, как мы обсудили в §2.9. Маршрутизатор же просто оптимизирует распространение группового трафика. Поэтому, когда хост принимает два канала, (S_1,G) и (S_2,G), которые отличаются только адресом источника, но не адресом назначения, маршрутизатору достаточно допустить источники S_1 и S_2 к группе G, а остальные блокировать, чтобы не расходовать впустую ресурсы сети. То же самый подход применим и к SFM.

Остаточные особенности фильтрации SSM обсуждаются в [RFC 4604].

В общем, картина такая: маршрутизатор допускает некое *множество* адресов источника **S** к данной группе G, а остальные адреса блокирует. Поэтому нашу модель

фильтрации мы выразим на языке элементарной теории множеств, а оперировать она будет только адресами источника, так как у каждого группового адреса назначения будет свой независимый фильтр. Множество разрешенных адресов **S** мы станем называть первичным, потому что в нашей модели оно составляет истинный смысл фильтра, хотя способы фиксации (кодирования) этой информации могут быть разными.

> Так, буквально через абзац мы обнаружим альтернативный способ кодирования фильтра при помощи множества *блокируемых* адресов.

Множество всех адресов IPv6 — это универсум данной задачи, то есть множество всех возможных значений. Оно конечно, и поэтому теоретически любой фильтр можно записать, просто перечислив адреса источника. Так, режим вещания ASM обеспечивает фильтр, содержащий *все* адреса IPv6, а полное прекращение приема группы равноценно фильтру, в котором нет ни одного адреса (пустое множество). А когда несколько слушателей выражают свои пожелания касательно фильтрации трафика в виде множеств, маршрутизатору достаточно найти их объединение, чтобы удовлетворить все запросы и не заблокировать ни один желательный источник: $S = \bigcup_i S_i$. На бумаге выглядит неплохо!

> Как следствие, в MLDv2 можно вообще отказаться от сообщений типа «итог» и сигнализировать о выходе из группы на языке множеств.

> Мы сознательно не пытаемся исключить недопустимые адреса источника, такие как групповые адреса, из первичного фильтра, поскольку наша работа ведется в предположении, что входящий пакет с недопустимым адресом источника будет отбракован на самой ранней стадии его обработки. Это позволяет упростить наши теоретико-множественные выкладки и сделать их более прозрачными.

Но как нам быть со случаем, когда слушатель SFM на самом деле хочет заблокировать несколько источников, не препятствуя остальным? Этому отвечает первичное множество, равное дополнению множества блокируемых адресов.[117] Нетрудно подсчитать, что при блокировании N адресов в первичном фильтре остается $2^{128} - N$ элементов, и при небольшом N это будет астрономическое число! Если множество всех адресов для режима ASM ($N = 0$) еще можно было бы кодировать кратко, как особый случай, режим SFM с блокированием потребует явного перечисления этого фантастического количества адресов, потому что мы заранее не знаем, какие источники предпочтет исключить слушатель. Очевидно, что ни передавать, ни хранить такое множество невозможно. Поэтому настало время нам перейти от красивой теории к практическим хитростям и трюкам.

Главный наш трюк мы позаимствуем из сценария с блокированием источников. Пока степень наполнения фильтра адресами близка к одной из двух крайностей, все или ни одного, состояние фильтра можно записать довольно кратко. Когда наш первичный фильтр содержит всего несколько адресов, мы перечисляем именно их. Когда же первичный фильтр включает в себя почти все возможные адреса, то достаточно перечислить множество недостающих адресов $\overline{S}$, потому что оно связано с первичным множеством фильтра однозначным образом: $\overline{S} = \backslash S$, где $\backslash S$ означает дополнение множества S до универсума (множества всех адресов IPv6).

Переводя это в практическую плоскость, мы скажем, что фильтр группового маршрутизатора может работать в одном из двух режимов, **включающем** (*INCLUDE*) либо **исключающем** (*EXCLUDE*). Во включающем режиме фильтр допускает перечисленные источники, а в исключающем — блокирует их. Эти режимы взаимоисключающие, потому что каждый из них однозначно преобразует множество перечисленных адресов во множество допущенных адресов.

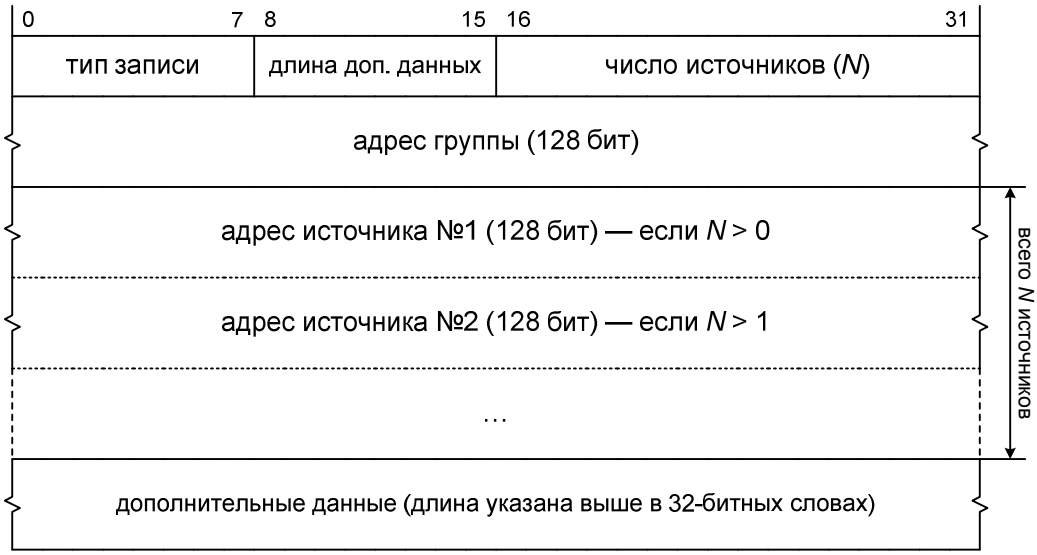
[117] То есть разности множества всех адресов и множества блокируемых адресов.

Сочетание этих режимов фильтрации имеет смысл, только когда элементами фильтра выступают диапазоны адресов разной величины, например, префиксы. Этот случай мы встречаем в настройках систем сетевой безопасности, таких как межсетевые экраны. Однако SFM оперирует только отдельными адресами.

Со своей стороны, слушатель тоже фильтрует трафик группы в одном из этих двух режимов, потому что его касаются те же самые практические соображения насчет размера фильтра. И тогда поддержка SFM в MLDv2 сводится к следующему: во-первых, передать информацию о фильтрах от слушателей к маршрутизатору, а во-вторых, свести вместе информацию от разных слушателей, чтобы объединенный фильтр маршрутизатора удовлетворил их всех.

В такой модели группового фильтра режим приема ASM (прием от любого источника) — это подмножество SFM и выражается фильтром EXCLUDE($\varnothing$), где $\varnothing$ — пустое множество. Говоря по-русски, ASM — это прием от всех источников без исключения (подразумевая, что технически такое исключение возможно).

Чтобы передавать такого рода информацию, каждая запись в отчете MLDv2 (см. Фиг. 96) будет содержать, помимо адреса группы, список источников. В первом приближении, этот список и есть фильтр данной группы у слушателя, составившего отчет. Как мы только что установили, этот фильтр надо снабдить указанием, в каком режиме он должен работать, включающем или исключающем. Сделаем мы это про помощи поля «тип записи».



Фиг. 96. Запись о группе в отчете MLDv2

Чтобы провести начальную загрузку своего фильтра на маршрутизаторы, слушатель должен дословно перечислить адреса источника в фильтре и указать его режим работы. Для этой цели достаточно двух типов записи, **CHANGE_TO_INCLUDE_MODE** (тип 3) и **CHANGE_TO_EXCLUDE_MODE** (тип 4). Первый из них говорит, что слушатель перевел свой фильтр во включающий режим и загрузил в него приведенный список источников. Второй тип сообщает то же самое для фильтра в исключающем режиме. Чтобы нам было удобнее обсуждать теоретико-множественные свойства типов записей, мы станем их кратко записывать как *TO_IN*(*S*) и *TO_EX*(*S*), соответственно, где *S* — множество источников, перечисленное в данной записи.

Хотя этих двух типов уже вполне хватило бы, перезагружать с нуля весь список каждый раз, когда надо послать отчет, было бы излишне. Поэтому мы предусмотрим еще два случая.

В первом из них слушатель хочет разрешить или блокировать несколько дополнительных источников, не меняя остальной их список. Так возникают типы записи

ALLOW_NEW_SOURCES (тип 5) и **BLOCK_OLD_SOURCES** (тип 6), сокращенно *ALLOW*(*S*) и *BLOCK*(*S*).

Второй заслуживающий оптимизации случай — это периодический ответ на запрос маршрутизатора, которым слушатель сообщает свой текущий фильтр, хотя тот остается без изменений. Для этого он использует отдельные типы записи, **MODE_IS_INCLUDE** (тип 1) и **MODE_IS_EXCLUDE** (тип 2), сокращенно *IS_IN*(*S*) и *IS_EX*(*S*). Благодаря этому маршрутизаторы могут оптимизировать обработку фильтров, пока те остаются постоянными. Кроме того, маршрутизатор не должен реагировать на эти записи дополнительными запросами, чтобы не вызвать зацикливание протокола.

Чтобы не повторять все время «запись отчета» и не путать ее с записью о группе в оперативной памяти узла, впредь мы будем говорить просто «отчет», например, «отчет IS_EX», памятуя при этом, что соответствующее сообщение MLDv2 составное и может содержать в себе более одного отчета, каждый из них об определенной группе.

Слушатель вполне может комбинировать разные отчеты об одной и той же группе, чтобы гибко уведомлять маршрутизаторы об изменениях в состоянии своего фильтра данной группы, используя при этом минимум сообщений MLDv2. К примеру, когда фильтр группы Г переходит из состояния INCLUDE(A) в состояние INCLUDE(B), где A и B — вообще говоря, разные множества источников, допущенных к данной группе Г, можно выделить два подмножества источников, затронутых этой операцией. Множество A за вычетом множества B, в математической нотации $A \setminus B$, больше не интересно слушателю, и его можно заблокировать. Напротив, множество B за вычетом множества A, то есть $B \setminus A$, раньше не интересовало слушателя, так что его надо разблокировать и допустить к группе. В то же время, пересечение множеств A и B, то есть $A \cap B$, остается интересным слушателю, и его статус не изменяется. Поэтому, чтобы объявить о данном переходе, достаточно два отчета в любом порядке: BLOCK($A \setminus B$) и ALLOW($B \setminus A$). Эти отчеты можно объединить в одно сообщение MLDv2, если позволяет их длина.

> Пусть читатель наглядно изобразит этот пример с помощью кругов Эйлера.

> В принципе, то же самое изменение можно было бы выразить всего одним отчетом TO_IN(*B*), однако этот отчет подразумевает, что изменился режим работы фильтра. В ответ маршрутизаторы начнут дополнительные проверки. Кроме того, если фильтр сам по себе длинный, а изменился он всего на пару источников, то суммарная длина двух отчетов ALLOW и BLOCK будет меньше, чем длина одного TO_IN, так как ALLOW и BLOCK передают только изменения, а TO_IN и TO_EX — фильтр целиком. Поэтому слушателю будет лучше воздержаться от применения TO_IN или TO_EX, когда подстраивается уже существующий фильтр.

Используя наши соображения, составим сводку основных правил для слушателя группы в Табл. 18. Здесь $A \setminus B$ обозначает разность множеств A и B (множество всех элементов A, которые не принадлежат также B), а $\varnothing$ — пустое множество.

Табл. 18. Правила отчетности для слушателя MLDv2

Текущее состояние	Новое состояние	Отчет
начало приема или EXCLUDE(A)	INCLUDE(B)	TO_IN(B)
начало приема или INCLUDE(A)	EXCLUDE(B)	TO_EX(B)
INCLUDE(A)	INCLUDE(B)	BLOCK($A \setminus B$), ALLOW($B \setminus A$)
EXCLUDE(A)	EXCLUDE(B)	ALLOW($A \setminus B$), BLOCK($B \setminus A$)
INCLUDE(A) или EXCLUDE(A)	конец приема	TO_IN($\varnothing$)

На практике дополнительная сложность возникает из-за того, что источник должен повторить свой отчет несколько раз. Если группа изменит свое состояние во время повтора отчета о ней, надо объединить старый и новый отчет, используя все те же соображения теории множеств [§6.1 RFC 3810].

Очевидная особенность SSM состоит в том, что о групповых адресах *FF3x::/96* не должно быть отчетов IS_EX или TO_EX. Дело в том, что слушатель SSM заведомо заинтересован в ограниченном числе источников, и поэтому он может вести прием группы только в режиме INCLUDE [§2.2.2 RFC 4604].

Теперь нам осталось понять, как маршрутизатор сведет вместе отчеты разных слушателей об одной и той же группе. Поскольку это довольно непростая задача, давайте начнем ее исследование с частных случаев, легче поддающихся анализу.

Пока все слушатели работают только во включающем или только в исключающем режиме, решение нам дает элементарная теория множеств. Как мы уже говорили, маршрутизатор обязан допустить к группе объединение первичных множеств источников, чтобы случайно не заблокировать желательный источник, а окончательную фильтрацию проведут сами слушатели. При включающем режиме искомая операция и есть объединение включающих фильтров I_i:

$$R = \bigcup_i I_i \ .$$

Если же все источники работают в исключающем режиме, то лучшее, что может сделать маршрутизатор, это блокировать пересечение всех исключающих фильтров E_i:

$$\overline{R} = \bigcap_i E_i \ .$$

Ведь для множеств справедливо такое следствие законов де Моргана:

$(A \setminus B) \cup (A \setminus C) = A \setminus (B \cap C)$.

Если A — это множество всех адресов IPv6, а B и C — множества двух исключающих фильтров, то смысл этого равенства именно в том, что пересечение исключающих фильтров эквивалентно объединению включающих фильтров.

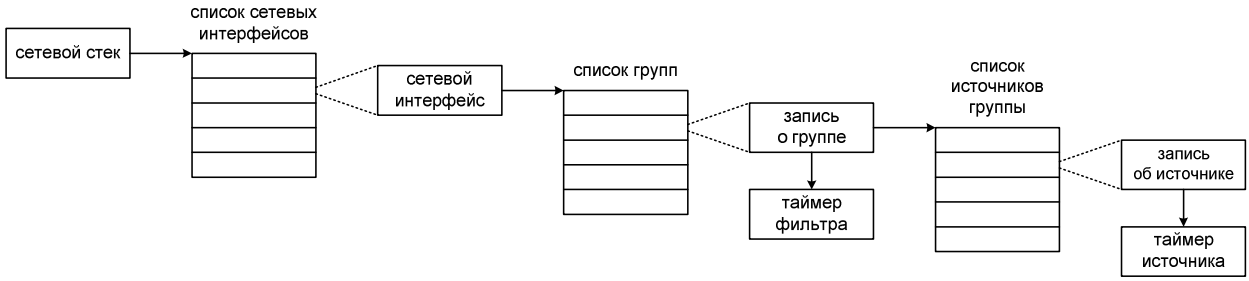
А как быть, когда на канале есть слушатели группы в разных режимах? Снова применим правило объединения первичных множеств для создания удовлетворительного фильтра. При объединении множеств число элементов не уменьшается, а первичное множество любого исключающего фильтра, пока он разумной длины, уже содержит почти все адреса IPv6. Значит, и суммарный фильтр в своем первичном виде будет содержать почти 2^{128} элементов. Как мы знаем, в этом случае надо просто блокировать остальные адреса, чтобы не иметь дела с фильтром чудовищной длины. Поэтому подходящим режимом суммарного фильтра будет именно исключающий. Иными словами, как только хотя бы один из слушателей переходит в исключающий режим, то же самое вынуждены сделать и маршрутизаторы. При этом блокировать надо пересечение всех исключающих фильтров за вычетом объединения всех включающих фильтров:

$$\overline{R} = \bigcap_i E_i \setminus \bigcup_i I_i \ .$$

В частности, пока есть слушатели в режиме ASM ($\exists n : E_n = \varnothing$), этот фильтр будет пустой и маршрутизатор не сможет блокировать ни один источник. Когда же последний слушатель в исключающем режиме прекратит прием группы, маршрутизатору следует вернуться во включающий режим, чтобы эффективнее фильтровать трафик группы. Если затем и слушатели во включающем режиме завершат прием, включающий фильтр опустеет: $R = \varnothing$, — что отвечает концу продвижения группы в данный канал.

Это и есть теоретико-множественная основа поддержки SFM и SSM в MLDv2. Все остальное в ней — суть важные, но не несущие фундаментального значения детали. Среди них выделяется вопрос о том, как маршрутизатор станет управлять текущей информацией о группе, чтобы она оставалась актуальной. В MLDv1 для этого было достаточно связать с

группой один таймер, а когда кто-то из слушателей прекращал прием группы, запросом о данной группе проверить, остались ли в ней другие слушатели. То есть элементарным объектом управления выступала группа. Теперь же разные слушатели изъявляют желание, или нежелание, принимать трафик группы из разных источников, но маршрутизатор по-прежнему не ведет поименного списка слушателей и не запоминает, кто именно из них заинтересован в данном источнике. Поэтому отдельный таймер потребуется каждой записи об источнике группы в памяти маршрутизатора [§7.2 RFC 3810], а элементарным объектом управления в MLDv2 станет источник данной группы, как это показано на Фиг. 97.



Фиг. 97. Структуры данных группового маршрутизатора — модель MLDv2

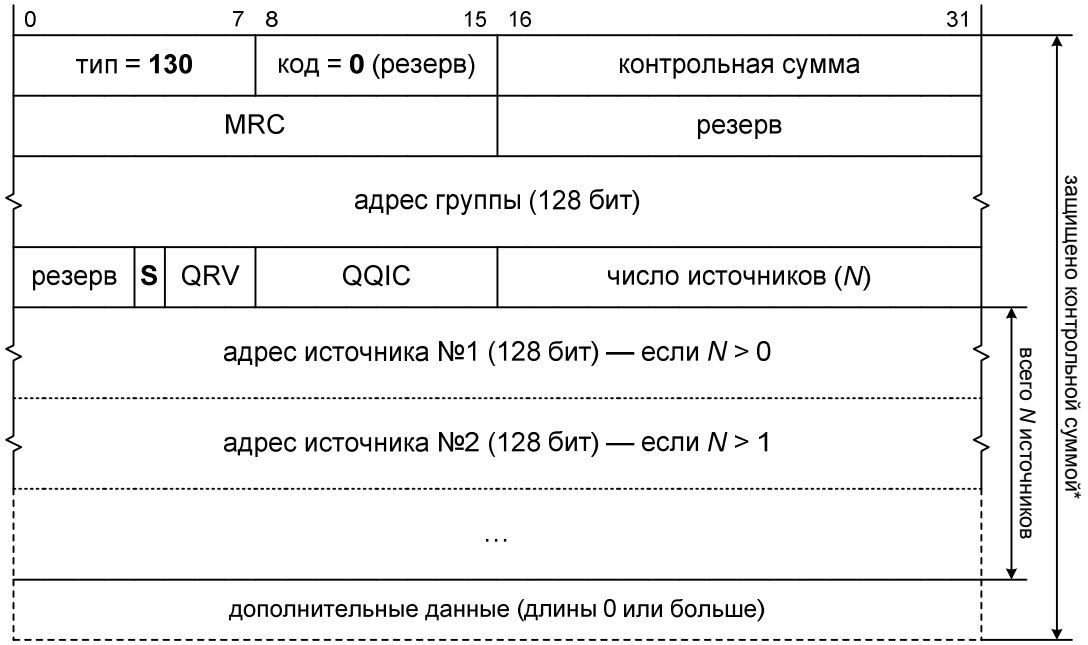
Обратите внимание: у разных групп могут быть разные предпочтения насчет одного и того же источника, а их фильтры независимы друг от друга. Поэтому один и тот же адрес IP источника вполне может фигурировать в разных записях, если они связаны с разными группами.

Когда слушатель группы заявляет отчетом *IS_IN(A)*, *TO_IN(A)* или *ALLOW(A)* о своем желании принимать пакеты из некоторого множества источников *A*, маршрутизатор обязан немедленно допустить трафик из источников *A* в канал. Напротив, когда слушатель группы не желает вести прием из данного множества источников *B* и включает в свой отчет элемент *IS_EX(B)*, *TO_EX(B)* или *BLOCK(B)*, маршрутизатор не вправе тут же заблокировать все множество источников *B*, потому что, возможно, у некоторых источников в нем есть и другие слушатели. Точно так же в MLDv1 маршрутизатор не мог прекратить продвижение трафика группы, едва получив итоговое сообщение от одного слушателя, — он был обязан проверить, остались ли у группы другие слушатели. Это вызвано тем, что маршрутизатор не ведет поименного учета слушателей.

Тем не менее, по приходу отчета *IS_EX*(*B*), *TO_EX*(*B*) или *BLOCK*(*B*) маршрутизатору не обязательно проверять все множество блокируемых источников *B*. Ведь в этот момент у маршрутизатора уже есть определенное первичное множество допущенных источников *S*, закодированное в терминах *INCLUDE* или *EXCLUDE*. (Как мы говорили, во включающем режиме (*INCLUDE*) первичное множество просто равно фильтру, который маршрутизатор хранит в своей памяти: $S = R$, — а в исключающем режиме (*EXCLUDE*) оно равно дополнению этого фильтра до множества всех адресов IPv6, нашего универсума: $S = \overline{R}$.) Поэтому проверить запросом необходимо только пересечение множеств B и S: $Q = B \cap S$, — так как адреса вне этого подмножества в любом случае сохраняют свое состояние: подмножество $S \setminus B$ по-прежнему остается допущенным к группе, а $B \setminus S$ уже заблокировано.

Чтобы провести проверку источников направленно, маршрутизатору понадобится уточнить запрос MLD не только группой, о которой идет речь, но и множеством источников *Q*, состояние которых его интересует. Поэтому в MLDv2 у запроса появляется новая разновидность: **запрос, ограниченный группой и источниками** (*Multicast Address and Source Specific Query*). Нужно ли уточнять в таком запросе режим фильтрации, *INCLUDE* или *EXCLUDE*? На самом деле, групповой маршрутизатор всегда спрашивает только о желательных источниках — он никогда не ставит вопрос: «Кто блокирует данный источник?» — потому что блокировка происходит по умолчанию. Это следует из самого предназначения MLD: избавиться от как можно большей доли ненужного

группового трафика. Множество же $Q = B \cap S$ содержит не больше элементов, чем множество *B*, приведенное в отчете. Поэтому множество *Q* тоже можно указать простым перечислением его элементов. Следовательно, запрос MLDv2 ставится только в режиме *INCLUDE*, и явное указание режима в нем излишне. Такой запрос о группе Г, ограниченный списком источников *Q*, мы будем обозначать Q(Г,*Q*), а запрос, ограниченный только группой Г — просто Q(Г). Запрос Q(Г) отличается от Q(Г,*Q*) тем, что содержит ноль источников, то есть список *Q* в нем пуст. Их общий формат показан на Фиг. 98 и содержит несколько полей, которые понадобятся нам чуть позднее.



*) Не забудьте: контрольная сумма также защищает псевдозаголовок.

Фиг. 98. Запрос MLDv2

Хотя ограниченность запроса Q(Г,*Q*) режимом *INCLUDE* вполне обоснована, она приходит в противоречие с исключающим режимом работы фильтра и вызывает дополнительную сложность в управлении проверкой источников. Пока фильтр самого маршрутизатора работает во включающем режиме, отслеживать состояние проверки множества источников *Q* просто: *Q* — это заведомо подмножество фильтра ($B \cap S \subseteq S$), о каждом элементе *Q* уже есть запись в памяти маршрутизатора, и потому при проверке достаточно понизить тайм-ауты на всех записях *Q*. Напротив, в исключающем режиме элементы *Q* заведомо отсутствуют в фильтре ($B \cap S \not\subset \setminus S$), и маршрутизатору придется хранить записи об элементах *Q* отдельно от рабочего фильтра.

Поэтому во включающем режиме состояние группы, с точки зрения маршрутизатора, можно задать всего одним списком источников, тогда как в исключающем режиме понадобятся целых два списка. Общепринятые обозначения этих состояний [§2.3 RFC 3810] приведены и прокомментированы в Табл. 19.

Табл. 19. Режимы фильтра группы в маршрутизаторе MLDv2

Обозначение	Режим	Смысл параметров
INCLUDE(*A*)	Включающий	• *A* — «список включений» (*Include List*). Это список источников, допущенных к группе. Все прочие источники заблокированы.
EXCLUDE(*X*,*Y*)	Исключающий	• *X* — «список требований» (*Requested List*). Это список источников, которые все еще могут быть интересны некоторым слушателям. Поэтому их нельзя немедленно блокировать, а необходимо

		проверить с помощью запроса MLD. Ниже мы встретим еще одну роль списка требований. • *Y* — «список исключений» (*Exclude List*). Это список заблокированных источников. Все прочие источники, включая элементы *X*, допущены к группе.

В состоянии *EXCLUDE*(*X*,*Y*) пересечение множеств *X* и *Y*, очевидно, пусто: $X \cap Y = \varnothing$, — поскольку нельзя одновременно допускать и блокировать один и тот же источник. Следовательно, *X* вообще не влияет на работу фильтра в текущий момент. Но если на записи об элементе $x \in X$ истечет тайм-аут, в модели MLD это будет означать, что данный источник *x* больше не интересен никому. Тогда маршрутизатор сможет перенести элемент *x* из списка *X* в список *Y*, тем самым заблокировав источник *x* и избавив канал от ненужного группового трафика. В этом и состоит роль списка *X*.

Правила поведения маршрутизатора в ответ на отчеты слушателей группы оказываются довольно сложными, потому что они должны учесть все наши предыдущие соображения. Волноваться из-за этого не стоит: мы разберем и прокомментируем свод этих правил. Однако прежде нам надо уточнить еще несколько деталей.

Как мы уже сказали, в MLDv2 элементарным объектом управления групповым трафиком служит источник группы. При этом групповой маршрутизатор занимается тем, что минимизирует избыточный трафик в канал, блокируя никому не интересные источники. Условно говоря, маршрутизатор стремится заблокировать весь трафик группы, но ему мешают отчеты слушателей, требующие допустить тот или иной источник.

Пока запись об источнике группы находится в списке включений или требований, она явно разрешает трафик от этого источника, и маршрутизатор заинтересован в том, чтобы заблокировать этот источник, как только его перестанут слушать. Для этого маршрутизатор запускает на записи об источнике таймер, который срабатывает, если о данном источнике группы давно не было отчетов. Дальнейшая судьба записи зависит от режима фильтра, но в любом случае она приводит к блокировке источника. Так, во включающем режиме запись удаляется из списка включений, а в исключающем режиме она перемещается в список исключений.

> Конечно же, мы помним, что запись об источнике привязана к определенной группе. Например, один и тот же источник И может быть допущен маршрутизатором к группе Г1, но заблокирован для группы Г2. В терминах заголовка IPv6 это означает, что пакеты от И к Г1 будут продвигаться в данный канал, а пакеты от И к Г2 — нет. Это находит прямое отраженное в иерархии структур, которые маршрутизатор хранит в своей памяти — см. Фиг. 97.

Оказавшись в списке исключений, запись об источнике вызывает его блокировку, а значит, потенциальное уменьшение трафика в канал. Это желанное для маршрутизатора состояние, и нет причин прерывать его по тайм-ауту. Если вдруг у источника снова возникнет слушатель, он заявит о себе отчетом. Такова модель MLD: слушатели группы обязаны напоминать о себе, а в MLDv2 и о своем выборе источников, так как по умолчанию маршрутизатор не продвигает групповой трафик в канал. Следовательно, на записях в списке исключений таймер остановлен.

С другой стороны, список исключений — атрибут исключающего режима фильтра, а в этом режиме маршрутизатор и близко не может подойти к своей заветной мечте, а именно заблокировать все источники, потому что ему дозволено блокировать только перечисленные *N* из них, а остальные $2^{128} - N$ беспрепятственно шлют пакеты группе. Это настоящий кошмар для маршрутизатора! Поэтому в его интересах перевести фильтр группы во включающий режим, как только исчезнут слушатели в исключающем режиме.

Маршрутизатор не ведет их поименного списка, и ему остается снова положиться на таймер. Это так называемый «таймер фильтра» (*Filter Timer*) на записи о группе. Он активен только в исключающем режиме и перезапускается по приходу отчета IS_EX или TO_EX, поскольку такой отчет говорит, что на канале все еще есть слушатели группы в исключающем режиме. Но если интервал этого таймера истечет, у маршрутизатора будет полное право переключить фильтр во включающий режим за отсутствием слушателей группы в исключающем режиме.

Однако, как маршрутизатор ни стремится свести трафик группы на нет, он обязан допускать источники к группе по требованию ее слушателей. В частности, ему не пристало самовольно блокировать те источники, о которых недавно были отчеты IS_IN, TO_IN или ALLOW. Поэтому, переводя фильтр группы во включающий режим, маршрутизатор обязан позаботиться, чтобы новый список включений заранее содержал в себе, по меньшей мере, все записи о желательных источниках; ведь иначе они окажутся заблокированы по умолчанию. Выходит, что накапливать эту информацию фильтр должен заранее, еще в исключающем режиме. Понадобится ли для этого новый список источников в памяти маршрутизатора? Попробуем обойтись уже доступным списком требований, если он сможет играть обе роли. В принципе, маршрутизатору ничто не мешает сохранять записи о желательных источниках в этом же списке. Вопрос только в том, насколько оправдано хранить оба вида информации в одном списке: записи о желательных источниках и записи о тех источниках, которые маршрутизатор пытается заблокировать. Чтобы предотвратить путаницу, заметим такое простое свойство «собираемого» нами механизма: если пришел отчет IS_EX или TO_EX, то фильтр останется в исключающем режиме как минимум на протяжении интервала своего таймера. Если этот интервал будет, скажем, равен тайм-ауту записи об источнике, то у слушателей во включающем режиме будет еще достаточно времени, чтобы снова заявить о своем выборе источников в ответ на периодические запросы маршрутизатора. Поэтому маршрутизатор, получив отчет IS_EX или TO_EX, вправе вычистить из списка требований все записи, кроме текущих кандидатов на блокировку [§7.2.3 RFC 3810].

Выше мы позволили себе вольность говорить о работе таймеров маршрутизатора в обыденных терминах «давно» и «недавно». В общем, понятно, что «недавно» — это пока интервал таймера не истек, а какой таймер, следует из контекста. Тем не менее, нам явно следует добавить определенности в эту картину. Мы сошлись на том, что у каждой группы есть два вида таймеров.

Первый из них — это таймер одного источника данной группы. Во включающем режиме MLDv2 он играет ту же роль, что и таймер всей группы в MLDv1, а именно позволяет маршрутизатору определить, что пора блокировать трафик, так как его больше никто не принимает. Разница только в том, что маршрутизатор MLDv1 оперировал целыми группами, тогда как его «потомок» в MLDv2 работает тоньше и управляет отдельными источниками, вещающими группе. Поэтому, когда срабатывает такой таймер, в MLDv1 маршрутизатор удалил бы запись о группе, а в MLDv2 он удаляет вместо этого запись об одном источнике. Когда фильтр INCLUDE(*A*) опустеет, будет пора удалить и всю запись о группе, так что отельный таймер группы здесь не требуется.

Насколько большой интервал понадобится таймеру источника? Этот таймер перезапускается по приходу отчета, что данный источник интересен какому-то слушателю данной группы, и поэтому интервал должен быть таким, чтобы заведомо перекрыть период между отчетами; ведь иначе состояние фильтра группы будет неустойчивым, фильтр станет «хлопать». Пока слушатели не меняют своего состояния, их отчеты вызывает маршрутизатор, высылая периодические запросы. Это довольно дорогая операция, так как, в теории, все слушатели канала должны в ответ сообщить свои фильтры в отчетах IS_IN или IS_EX, поэтому период между запросами, *QI* (*Query Interval*), должен быть довольно большим, по умолчанию 125 секунд [§9.2 RFC 3810].

Вдобавок одиночный запрос мог не дойти до некоторых слушателей, скажем, из-за сбоя или перегрузки канала. Тем не менее, есть надежда, что какой-то из последующих запросов все-таки дойдет, и поэтому маршрутизатор должен подождать, как минимум, еще несколько интервалов *QI*, прежде чем объявить источник невостребованным и удалить его запись. Сколько всего раз надо передать сообщение, чтобы оно наверняка дошло, — это параметр *RV* (*Robustness Variable*), зависящий от ненадежности канала, по умолчанию 2 [§9.2 RFC 3810]. Такой повтор защитит от $RV-1$ потерь. Наконец, слушатели не ответят на запрос все хором, а сделают случайные паузы, чтобы распределить нагрузку на канал. Верхняя граница такой паузы — параметр *QRI* (*Query Response Interval*), по умолчанию 10 секунд [§9.3 RFC 3810].

В результате, пока сеть стабильна, интервал таймера источника должен быть не менее чем $QI \times RV + QRI$; по умолчанию выходит 260 секунд. Этот интервал обозначают *MALI* (*Multicast Address Listening Interval*) [§9.4 RFC 3810].

> Интервал *QI* входит в *MALI* ровно *RV* раз, а не $RV-1$, и вот почему. Нам никто не гарантирует, что возникновение записи об источнике совпадает с отправкой периодического запроса. Поэтому до ближайшего запроса придется подождать, самое большее, время *QI*. Затем следует $RV-1$ интервалов длиной *QI*, когда запрос повторяется. Поэтому всего выходит *RV* интервалов длины *QI*.
>
> Чем пренебрегает формула для *MALI*? (Задержкой канала.)

Но вот, представим себе, маршрутизатор получает отчет, который дает ему повод усомниться в некоторых записях, к примеру, отчет BLOCK. В ответ маршрутизатор начинает активную проверку этих записей запросами, ограниченными группой и списком источников. Эквивалентом этого сценария в MLDv1 был приход сообщения «итог». Чтобы справиться с проверкой поскорее, не жертвуя при этом достоверностью результатов, маршрутизатор передает запрос *LLQC* раз (*Last Listener Query Count*, по умолчанию равен *RV* [§9.9 RFC 3810]), но делает между передачами лишь короткие паузы длиной *LLQI* (*Last Listener Query Interval*), по умолчанию 1 секунда [§9.8 RFC 3810]. При этом слушатели задерживают свои отчеты на короткое случайное время от 0 до *LLQI*. В такой схеме самое позднее, когда может прийти отчет, это спустя время $LLQI \times LLQC$ от начала проверки, что по умолчанию составляет 3 секунды. Для этого интервала принято обозначение *LLQT* (*Last Listener Query Time*) [§9.10 RFC 3810]. И нет смысла устанавливать таймер источника во время проверки на больший интервал.

> Нам трудно избавиться от навязчивой мысли, что аббревиатуру *LLQT* следует произносить так: «*Lil' cutie*».
>
> Интервал *LLQI* входит в *LLQT* ровно *LLQC* раз, а не $LLQC+1$, потому что первый запрос в серии уходит немедленно. Затем следует $LLQC-1$ интервалов повтора и, наконец, еще один интервал отведен на случайную задержку отчетов слушателями.

Выходит, что в зависимости от ситуации оптимальный интервал таймера может быть длинным (*MALI*) или коротким (*LLQT*). Выбрать подходящий интервал довольно просто, если действовать по вот какой процедуре:

- По умолчанию маршрутизатор устанавливает таймер источника на длинный интервал (*MALI*).
- Когда маршрутизатор передает первое сообщение в серии запросов Q(Г,*S*), ограниченных данной группой Г и списком источников *S*, он понижает значение таймера на каждом источнике группы Г из списка *S* до *LLQT*, если оно выше *LLQT*.

> Чуть позднее нам придется видоизменить эту процедуру, чтобы обеспечить более надежную синхронизацию между генератором запросов и пассивными наблюдателями. В некоторых случаях маршрутизатору придется устанавливать

значение таймера на интервал *LLQT* при передаче не только первого, но и нескольких последующих запросов в серии.

То есть передача запросов, ограниченных группой и списком источников, служит прямым признаком того, что маршрутизатор усомнился в востребованности источников из этого списка.

Прекрасно, теперь маршрутизатор может гибко управлять таймерами источников, пока фильтр группы работает во включающем режиме. А как быть с исключающим режимом? Будут ли его правила игры совершенно иными? Для начала напомним себе, что в этом режиме на месте единственного списка включений, содержавшего в себе все записи об источниках, возникает целых два списка, требований и исключений, а запись об источнике находится в одном из них. Функция списка требований такова, что и управлять им надо практически так же, как списком включений. Единственное существенное отличие заключается в том, что по тайм-ауту его записи не удаляются, а перемещаются в список исключений, потому что это кандидаты на блокировку. Оказавшись в списке исключений, запись об источнике вообще становится «бессмертной», а ее таймер не работает.

Тем не менее, на сцене наконец-то появляется второй вид таймера, с которым приходится иметь дело маршрутизатору MLDv2. Это, конечно же, таймер фильтра, следящий за тем, чтобы фильтр группы не задержался в исключающем режиме после того, как с канала исчезнет последний слушатель в этом режиме. Мы уже обнаружили, что ради устойчивости протокола интервал у этого таймера должен быть таким же, как у таймера источника. Вопрос только, каким из них, коротким или длинным?

Пока конфигурация слушателей постоянна, это должен быть длинный интервал (*MALI*), так как подготовка к переходу EXCLUDE→INCLUDE включает в себя накопление записей в списке требований, а для этого каждый слушатель во включающем режиме должен получить возможность высказаться отчетом IS_IN в ответ на последний из периодических запросов маршрутизатора.

Но возможна и ситуация, когда маршрутизатор получает повод немедленно усомниться в целесообразности исключающего режима. Таким поводом служит отчет TO_IN, косвенно говорящий об уменьшении числа слушателей группы в исключающем режиме. Получив его, маршрутизатор должен каким-то образом проверить, не упало ли оно до нуля. С этой целью маршрутизатор шлет серию из запросов Q(Г), ограниченных только группой, но не списком источников, пытаясь вызвать отклик слушателей группы в исключающем режиме за время *LLQT*. Наряду с ними ответят и все остальные слушатели группы, так что время подготовки к вероятному переходу EXCLUDE→INCLUDE, если он произойдет ввиду отсутствия ожидаемого отклика, можно смело сократить до *LLQT*.

По итогам наших рассуждений составим процедуру управления интервалом таймера фильтра — она оказывается практически такой же, как у таймера источника:

- По умолчанию маршрутизатор устанавливает таймер фильтра на длинный интервал (*MALI*).
- Когда маршрутизатор передает первое сообщение в серии запросов Q(Г), ограниченных только данной группой Г, он понижает значение таймера фильтра группы Г до *LLQT*, если оно выше *LLQT*.

Позднее нам придется видоизменить и эту процедуру, чтобы обеспечить надежную синхронизацию между генератором запросов и пассивными наблюдателями. В некоторых случаях маршрутизатору придется устанавливать значение таймера фильтра на интервал *LLQT* при передаче не только первого, но и нескольких последующих запросов в серии.

Любопытно отметить, что в приведенных нами процедурах можно даже не упоминать режим работы фильтра, потому как в исключающем режиме маршрутизатор не

упоминает источники из списка исключений в запросах Q(Г,*S*), а во включающем режиме ему никогда не потребуется запрос Q(Г).

Тем не менее, на практике будут нелишними дополнительные проверки, что поведение разных компонентов одной реализации отвечает ожиданиям разработчиков.

Таким образом, у каждой разновидности запроса MLDv2 своя отдельная роль. Все они собраны в Табл. 20.

Табл. 20. Разновидности запросов MLDv2

Разновидность	Обозначение	Когда шлется	Роль
Общий запрос	Q(::)	При старте маршрутизатора, затем периодически.	Выяснить текущее состояние дел на канале.
Запрос, ограниченный группой Г	Q(Г)	Когда уменьшается число слушателей группы в исключающем режиме.	Выяснить, остались ли у данной группы слушатели в исключающем режиме.
Запрос, ограниченный группой и списком источников	Q(Г, S)	Когда уменьшается число слушателей группы, заинтересованных в данных источниках S.	Выяснить, востребованы ли все еще в данной группе данные источники.

И вот, наконец, мы готовы составить свод правил для маршрутизатора MLDv2 в Табл. 21.

Табл. 21. Правила обработки отчетов маршрутизатором MLDv2

Текущее состояние	Отчет	Новое состояние	Запросы	Удалить	Таймеры
INCLUDE(A)	ALLOW(B)	INCLUDE($A \cup B$)	—	—	(B)=MALI
INCLUDE(A)	IS_IN(B)	INCLUDE($A \cup B$)	—	—	(B)=MALI
INCLUDE(A)	TO_IN(B)	INCLUDE($A \cup B$)	Q(Г, $A \setminus B$)	—	(B)=MALI
Комментарий: Текущий фильтр пропускает только источники из множества A. Поэтому все три отчета, по сути, требуют расширения этого множества до $A \cup B$. На записях, о которых получена свежая информация, перезапускается таймер. Отчет TO_IN(B) также вызывает запрос насчет прочих допущенных источников, Q(Г, $A \setminus B$), потому что это может быть сигнал о сокращении или изменении, а не расширении множества принимаемых источников. Отчет IS_IN(B) такого запроса не вызывает, потому что он сам пришел в ответ на какой-то запрос, во избежание зацикливания протокола.					
INCLUDE(A)	BLOCK(B)	INCLUDE(A)	Q(Г, $A \cap B$)	—	—
Комментарий: Поскольку текущий фильтр пропускает только источники из множества A, подмножество $B \setminus A$ уже заблокировано, и остается рассмотреть только $A \cap B$. Маршрутизатор не может немедленно заблокировать его, так как его потенциально принимают другие слушатели, поэтому сейчас он ограничивается запросом Q(Г, $A \cap B$).					
INCLUDE(A)	IS_EX(B)	EXCLUDE($A \cap B, B \setminus A$)	—	$A \setminus B$	($B \setminus A$)=0, (фильтр)=MALI
INCLUDE(A)	TO_EX(B)	EXCLUDE($A \cap B, B \setminus A$)	Q(Г, $A \cap B$)	$A \setminus B$	($B \setminus A$)=0, (фильтр)=MALI
Комментарий: Прежде всего, маршрутизатор должен перевести фильтр в исключающий режим, так как обнаружился слушатель в этом режиме. Все источники, кроме множества A, уже были заблокированы, поэтому сейчас можно смело блокировать такую часть B:					

$B \cap (\setminus A) = B \setminus A$. Таймеры этих записей останавливаются, так как у списка исключений один таймер на весь фильтр; он как раз запускается. Остаток B, то есть $A \cap B$, немедленно блокировать нельзя, но его можно подвергнуть проверке, сохранив в списке требований. Тем не менее, запрос об источниках $A \cap B$ можно послать, только если отчет — об изменении фильтра (TO_EX), а не о его текущем состоянии (IS_EX), потому что последний сам пришел в ответ на какой-то запрос. Источники $A \setminus B$ в новом состоянии допущены по умолчанию, и записи о них можно вообще удалить.					
EXCLUDE(X, Y)	ALLOW(A)	EXCLUDE($X \cup A$, $Y \setminus A$)	—	—	(A)=MALI
EXCLUDE(X, Y)	IS_IN(A)	EXCLUDE($X \cup A$, $Y \setminus A$)	—	—	(A)=MALI
EXCLUDE(X, Y)	TO_IN(A)	EXCLUDE($X \cup A$, $Y \setminus A$)	Q(Г, $X \setminus A$), Q(Г)	—	(A)=MALI
Комментарий: Здесь мы знакомимся со второй ролью списка требований X, когда он накапливает включающие записи на случай перехода фильтра во включающий режим. Поэтому недостающие записи добавляются в список требований путем объединения множеств X и A. На всех записях, о которых получена свежая информация, перезапускается таймер. В то же время элементы A удаляются из списка исключений, который сокращается до $Y \setminus A$. Ведь отчет требует прекратить блокировку A, и это должно произойти немедленно, так как погрешность фильтра допустима только в сторону разрешения источников. Отчет IS_IN запросов не вызывает, так как сам вызван запросом. Напротив, отчет TO_IN вызывает запрос насчет тех элементов X, о которых нет свежей информации: X за исключением A. По большому счету, он нужен, чтобы список X не рос за счет устаревших записей. Кроме того, отчет TO_IN потенциально означает, что число слушателей в исключающем режиме уменьшилось, потому что один из них перешел во включающий режим. Чтобы проверить, остались ли они вообще, маршрутизатор шлет еще один запрос без уточнения источников, то есть ограниченный только группой: Q(Г). Хотя в этом случае запрос Q(Г, $X \setminus A$) может показаться избыточным, он нужен для координации таймеров между маршрутизаторами канала — о ней мы скоро поговорим.					
EXCLUDE(X, Y)	IS_EX(A)	EXCLUDE($A \setminus Y$, $Y \cap A$)	—	$X \setminus A$, $Y \setminus A$	($A \setminus X \setminus Y$)=MALI, (фильтр)=MALI
EXCLUDE(X, Y)	TO_EX(A)	EXCLUDE($A \setminus Y$, $Y \cap A$)	Q(Г, $A \setminus Y$)	$X \setminus A$, $Y \setminus A$	($A \setminus X \setminus Y$)=(фильтр), (фильтр)=MALI
Комментарий: Безопасно блокировать можно только пересечение $Y \cap A$, так как только насчет этого подмножества есть консенсус слушателей. Остаток Y, то есть $Y \setminus A$, нужно немедленно разблокировать, потому что он интересен данному слушателю. Остальную часть A, то есть $A \setminus Y$, надо сначала проверить запросом, если мы имеем дело с отчетом об изменении состояния (TO_EX). При этом роль списка требований меняется: он снова хранит записи о проверяемых источниках, кандидатах на блокировку, а все прочие записи из него удаляются. Подмножество $X \setminus A$ уходит из списка требований, потому что в новых условиях у него больше нет шансов на блокировку: слушатель хочет его принимать. В то же время, подмножество $X \cap A$ остается в списке требований, потому что это по-прежнему кандидат на блокировку. На вновь добавленных записях, которых раньше не было в списке требований (это $A \setminus X \setminus Y$), запускаются таймеры. В случае TO_EX					

они устанавливаются на интервал, равный *текущему* значению таймера фильтра. Почему так? Пока фильтр работает в исключающем режиме, его таймер отсчитывает время до желанного момента, когда маршрутизатор сможет заблокировать почти все источники. Поэтому таймер фильтра неявно «тикает» на всех источниках, которые сейчас не занесены в X или Y и потому разрешены по умолчанию. Ради сохранения информации об актуальности таких записей, мы должны скопировать значение таймера фильтра в таймер такого источника, когда мы создаем о нем явную запись в списке требований X. Именно это здесь и происходит: сохранение накопленной информации.

Случай IS_EX особенный. Если маршрутизатор получил сообщение IS_EX, то есть заявление об уже существующем состоянии слушателя, и подмножество $A \setminus X \setminus Y$ не пусто, то это значит, что маршрутизатор потерял синхронизацию со слушателем. Хотя такое может произойти, например, когда маршрутизатор едва только включился в работу, или же после сбоя канала, наиболее вероятная причина — что маршрутизатор сам удалил эти записи из списка требований, чтобы проверить другие источники. Скажем, именно это произойдет, если на канале несколько слушателей в исключающем режиме, фильтры которых не совпадают: список требований станет «осциллировать». (Убедитесь в этом, изучив сценарий, когда один слушатель блокирует множество $\{1,2\}$, а другой — $\{2,3\}$.) Так или иначе, маршрутизатор не может доверять своим текущим сведениям и устанавливает таймеры новых записей на длинный интервал MALI. Кроме того, маршрутизатор не шлет запрос, ограниченный группой и списком источников, так как отчет IS_EX уже вызван каким-то запросом.

> У нас есть гипотеза, что установка ($A \setminus X \setminus Y$)=(фильтр) и в этом случае не вызвала бы никаких проблем. Исследуйте ее, если будет время и желание.

Наконец, оба отчета свидетельствуют, что у группы есть как минимум один слушатель в исключающем режиме, а значит, надо перезапустить таймер фильтра на длинный интервал MALI.

EXCLUDE(X, Y)	BLOCK(A)	EXCLUDE($X \cup (A \setminus Y), Y$)	Q(Г, $A \setminus Y$)	—	($A \setminus X \setminus Y$)=(фильтр)

Комментарий: Это правило похоже на предыдущее, TO_EX. Часть множества A, возможно, уже заблокирована фильтром Y, поэтому изменения касаются только подмножества $A \setminus Y$. Как обычно, вместо его непосредственной блокировки следует запрос о нем. Тем не менее, отчет BLOCK может исходить от слушателя в любом режиме, и поэтому маршрутизатор только добавляет недостающие элементы в список требований X путем объединения множеств, а не сбрасывает его до $A \setminus Y$. По той же самой причине таймер фильтра продолжает отсчет времени, а не перезапускается; ведь отчет BLOCK не доказывает, что есть слушатели в исключающем режиме. Если его интервал вскоре истечет, фильтр перейдет во включающий режим, а в список включений попадут все желательные источники, потому что список требований не сбрасывался до проверяемого подмножества, а объединялся с ним. Так маршрутизатор избежит нежелательной блокировки источников при переключении фильтра. Об установке таймеров на источниках см. предыдущее обсуждение TO_EX.

В отличие от правила TO_EX, здесь список исключений Y не требует немедленных изменений. Дело в том, что объявление вида TO_EX(A) говорит, что данному слушателю интересны все источники кроме перечисленных во множестве A, а значит, их надо немедленно разблокировать. В то же время, объявление BLOCK(A) не несет в себе такого строгого требования.

Заданием для читателя будет изобразить и проанализировать диаграммы Эйлера, отвечающие этим правилам.

Также убедитесь, что отчеты идемпотентны, то есть несколько копий одного отчета подряд, приведут маршрутизатор в то же самое состояние, что и ровно один экземпляр отчета. Это свойство позволяет слушателям повторять свои отчеты не по запросу, чтобы они наверняка дошли.

А обладают ли отчеты свойствами коммутативности и транзитивности? (Повод подумать.)

Все отчеты кроме IS_EX и TO_EX обладают свойством аддитивности: если данный список источников разбить, скажем, на два подсписка и послать их в разных отчетах, то окончательное состояние маршрутизатора окажется тем же самым. Зачем это может пригодиться, догадаться несложно: если отчет о данной группе содержит так много источников, что сообщение превышает MTU канала, его можно разбить на несколько сообщений поменьше. Однако для IS_EX и TO_EX этот трюк не работает. Как тогда быть? Приемлемым решением будет передать только часть списка одним сообщением [§5.2.15 RFC 3810]. Ведь маршрутизатор MLD вправе ошибаться в разрешительную сторону, если это необходимо, а опустить часть источников в отчете IS_EX или TO_EX, значит, допустить их к группе.

Подсчитайте, сколько источников может уместиться в одно сообщение MLDv2, когда MTU канала равно минимально возможному значению 1280 байт. (Подсказка: учесть опцию «сигнал маршрутизатору. Ответ: {1280[MTU]−40[IP]−8[HopByHop-RtrAlert]−28[MLDv2]} div 16 = 75.)

Оказывается, что сложность правил MLDv2 (а также IGMPv3) обусловлена, главным образом, поддержкой слушателей в режиме EXCLUDE(A), где множество A не пусто. Рассмотрите упрощенный протокол **LW-MLDv2** (облегченный MLDv2), где слушатель может принимать или избранные источники (INCLUDE(A)), или все источники подряд (EXCLUDE(∅)), но не может блокировать избранные источники. В этом случае правила маршрутизатора становятся намного проще, потому что ему больше не нужно вести список исключений. Самое удивительное, что маршрутизатор LW-MLDv2 может быть совместим со слушателями MLDv2, потому что в MLD допустима погрешность в сторону разрешения источников. Маршрутизатору LW-MLDv2 достаточно рассматривать любые отчеты IS_EX(A) или TO_EX(A) как просьбу разрешить все источники, то есть IS_EX(∅) или TO_EX(∅), соответственно. Если вы не справитесь с этим заданием самостоятельно, обратитесь к [RFC 5790].

Такое выражение протокола MLDv2 на языке теории множеств имеет одну любопытную особенность. Когда составленная нами таблица предписывает послать запрос, ограниченный группой и списком источников, может так оказаться, что список источников пуст. Например, в последнем правиле таблицы, EXCLUDE/BLOCK, такая ситуация возникнет, если A — это подмножество Y: $A \subseteq Y \Rightarrow A \setminus Y = \varnothing$. На первый взгляд, выходит, что надо послать запрос $Q(Г, \varnothing)$, где $\varnothing$ — пустое множество. Но, если мы составим такой запрос, то он окажется неотличим от запроса, ограниченного только группой, Q(Г) — см. формат запроса MLDv2 (Фиг. 98). Как разрешить эту двусмысленность? На самом деле, очень просто: запрос $Q(Г, \varnothing)$ риторический и никакого ответа не предполагает, а значит, и слать его совершенно ненужно. Ведь запрос всегда ставится в режиме INCLUDE, и на пустой список источников ответа никогда не было бы в принципе. Иными словами, единственный режим слушателя, который мог бы вызвать ответ на $Q(Г, \varnothing)$, — это $INCLUDE(\varnothing)$, но этот режим эквивалентен прекращению приема группы.

> Выражаясь на модном сегодня новоязе а-ля «1984», это не-режим не-слушателя.

Теперь нам осталось исправить всего несколько мелких деталей, чтобы считать нашу работу над MLDv2 завершенной. Для этого вернемся к нашему списку усовершенствований на стр. 190 и поглядим, что в нем осталось без внимания. Это оказывается взаимодействие между маршрутизаторами одного канала. Чтобы начать совместную работу, они должны, прежде всего, выбрать, кто из них будет генератором запросов. У нас уже есть готовая процедура выборов (стр. 187), которая оперирует только адресами маршрутизаторов и потому нас вполне устраивает.

> В связи с этой процедурой возникает таймер, с помощью которого наблюдатель может обнаружить, что текущий генератор запросов ушел со сцены. Этот таймер перезапускается на интервал **OQPT** (Other Querier Present Timeout, тайм-аут присутствия другого генератора запросов) всякий раз, когда приходит запрос MLD. Если же происходит тайм-аут, значит, запросов давно никто не слал, и пора начать новые выборы. Стандарт предлагает величину *OQPT*, равную $QI \times RV + ½QRI$ [§9.5 RFC 3810]. Однако на наш взгляд, добавка $½QRI$ способна привести к тому, что новый генератор запросов возникнет слишком поздно и непрерывность маршрутизации группового трафика будет нарушена. Убедиться в этом просто. Рассмотрите канал с одним слушателем и двумя маршрутизаторами. Один из маршрутизаторов, очевидно, генератор, а второй наблюдатель. Когда генератор исчезает, наблюдатель ждет время *OQPT*, прежде чем послать серию запросов. Затем слушатель вправе задержать свой отчет на время вплоть до *QRI*. В результате отчет может запоздать, придя через время $OQPT + QRI - MALI = ½QRI$ после того, как оставшийся маршрутизатор прекратил продвижение группы в канал по тайм-ауту *MALI*. Более подходящим значением для *OQPT* будет просто $QI \times RV$.
>
> Когда маршрутизатор только включается в работу или участвует в выборах, ему надо позаботиться, чтобы его запрос наверняка услышали другие маршрутизаторы, а также слушатели. Ради устойчивости к потере пакетов он передает такой запрос несколько раз (по умолчанию *RV* [§9.7 RFC 3810]), разделяя повторы паузами (по умолчанию $¼QI$ [§9.6 RFC 3810]).

После выборов все маршрутизаторы кроме генератора запросов замолкают и ведут только пассивное наблюдение за чужими сообщениями MLD. Чтобы такая система работала устойчиво, состояние наблюдателей должно быть согласовано с генератором в том, что касается управляющих записей и таймеров. Для этого, в первую очередь, наблюдатели должны получить от генератора его текущие значения параметров *QI* и *RV*, так как от них зависит длина интервала *MALI*. Эти значения обозначают, соответственно, *QQI* (*Querier's QI*) и *QRV* (*Querier's RV*), чтобы подчеркнуть, что они исходят от генератора запросов.

Как следствие, формат запроса MLDv2 должен предусмотреть поля для этой информации. Значение *QRV* можно поместить в пакет как есть, в беззнаковом коде. Соответствующее поле тоже обозначают **QRV** (*Querier's Robustness Variable*, *RV* генератора запросов) [§5.1.8 RFC 3810]. В то же время, значение *QQI*, если представить его как число миллисекунд в 16-битном беззнаковом коде, может расти только до 65,535 секунд, а кроме того, миллисекундная точность на всем диапазоне значений здесь излишня. Поэтому для передачи *QQI* мы чуть позже разработаем новый код. Он же пригодится и для передачи значений *MRD* (максимальной задержки отчета). Поле для хранения *QQI* в этом коде обозначают **QQIC** (*Querier's Query Interval Code*, код интервала запросов генератора) [§5.1.9 RFC 3810].

Далее, когда дело доходит до быстрой проверки группы, в игру вступают интервал *LLQI*, счетчик повторов *LLQC* и тайм-аут *LLQT*. Значения этих параметры тоже должны быть согласованными, чтобы наблюдатель не удалил запись об источнике или не переключил режим фильтра группы раньше времени. Впрочем, эти параметры явным образом передавать не надо. Вот наше тому обоснование:

- По умолчанию значение *LLQC* совпадает с *RV*, а рекомендуемое значение *RV* уже объявляется генератором запросов в поле запроса **QRV**.
- При быстрой проверке максимальная задержка отчета, *MRD*, равна *LLQI*, а значение параметра *MRD* уже содержится в запросе. Поэтому наблюдателям достаточно выполнить обратную операцию и принять рабочее значение *LLQI* равным *MRD* из полученного запроса.
- Значение *LLQT* однозначным образом зависит от *LLQI* и *LLQC*: $LLQT = LLQI \times LLQC$.

Чтобы не быть голословными, сошлемся на существующие реализации. Например, именно так поступает модуль MLDv2 в XORP [118]: он принимает *LLQC* равным *QRV*, а *LLQI* равным *MRD*, если принят запрос с ненулевым адресом группы.

Но каким образом наблюдатели узнают, что началась быстрая проверка? Эту информацию можно извлечь из того, какого вида запрос:

- Если запрос общий, с нулевым адресом группы и пустым списком источников, то это явно не быстрая проверка, а начальный или периодический запрос. Из такого запроса надо извлечь только значения *RV* и *QI*.
- Если запрос ограничен только ненулевой группой, а список источников в нем пустой, то это быстрая проверка, остались ли на канале слушатели группы в исключающем режиме. В этом случае надо понизить интервал таймера на фильтре указанной группы до *LLQT*.

Согласно принципу Постела, получатель запроса должен сначала убедиться, что фильтр указанной группы работает в исключающем режиме.

- Если же запрос ограничен ненулевой группой и непустым списком источников, то он служит для быстрой проверки указанных источников данной группы. Для наблюдателя это повод понизить интервалы таймеров на всех перечисленных источниках данной группы до *LLQT*.

Конечно, наблюдателю не помешало бы убедиться, что запрос исходит именно от генератора запросов. Пунктуальная реализация может помнить адрес текущего генератора и обновлять его, если пришел запрос с еще меньшим адресом источника IPv6. Однако в этом случае придется также учесть сценарий с исчезновением генератора. На практике достаточно сравнить адрес источника запроса с собственным адресом наблюдателя и проигнорировать запрос, если наш

[118] http://www.xorp.org/

локальный адрес меньше. Остальную работу по стабилизации системы здесь выполнит механизм выборов генератора. Ведь он гарантирует, что в конце концов останется ровно один активный генератор, который и станет диктовать остальным значения параметров.

Эта черновая схема взаимодействия между генератором запросов и пассивными наблюдателями неплохо выглядит на бумаге, однако она не учитывает, что пакеты иногда теряются. Увы, сейчас злая фея Реальность немного спутает нам карты и усложнит полученное решение.

В действительности генератор запросов никогда не знает, что запрос дошел до всех наблюдателей. Обратной связи от наблюдателей к генератору нет — и не будет, чтобы не усложнять протокол сверх меры. Поэтому единственное средство, доступное генератору, — это повторять запрос несколько раз, повышая тем самым вероятность успешной передачи информации. И если периодические запросы не надо дублировать, поскольку они и так повторяются с интервалом *QI*, то запросы для быстрой проверки приходится повторять еще по $LLQC - 1$ раз. Это поведение для нас не ново, ведь и до слушателей запрос может дойти не с первого раза. Тем не менее, появление на сцене наблюдателей вносит в нашу схему дополнительный элемент неопределенности.

Представим себе, что на канале заведомо один и только один групповой маршрутизатор — генератор запросов. В таком умозрительном случае маршрутизатор вправе прекратить повтор запроса, как только придет первый отчет о данной группе и, возможно, источниках. Действительно, зачем повторять запрос, если искомая информация уже получена? Вернемся теперь в объективную реальность, где маршрутизаторов вполне может быть несколько. В этом случае приход отчета не дает генератору права прекратить повтор, потому что кто-то из наблюдателей мог и не получить отчет или запрос. Действуя согласно принятой модели, что только повтор обеспечивает требуемую вероятность успеха, генератор обязан передать запрос *LLQC* раз, и ни разом меньше.

Какие сложности это вызывает? В нашей рабочей схеме каждый экземпляр запроса управляет таймерами наблюдателей, понижая их интервал до небольшой величины *LLQT*. Поэтому возможен, к примеру, такой сценарий с участием генератора, наблюдателя и слушателя, составленный для $LLQC = 2$:

1. Генератор шлет первый экземпляр запроса Q(Г,*S*).
2. Наблюдатель понижает таймеры на источниках *S* до *LLQT*.
3. Слушатель после короткой паузы отвечает отчетом, подтверждая свой интерес к источникам *S*.
4. Генератор и наблюдатель получают отчет и перезапускают таймеры *S* с длинным интервалом *MALI*.
5. Генератор повторяет запрос Q(Г,*S*).
6. Наблюдатель понижает таймеры на источниках *S* до *LLQT*.
7. Слушатель не отвечает или отчет не доходит до наблюдателя.
8. Наблюдатель получает тайм-аут на источниках *S* и блокирует их.
9. Синхронизация генератор-наблюдатель нарушена!

Аналогичный сценарий можно составить и для запроса Q(Г), ограниченного только группой. В нем у наблюдателя сработает таймер фильтра.

К сожалению, генератор запросов действует в условиях недостатка информации и поэтому не может дать стопроцентную гарантию того, что наблюдатель получит все необходимые сведения даже после сбоя сети. Видимо, лучшее, что мог сделать генератор на шаге 5 нашего сценария, — это сообщить наблюдателю: больше не понижай таймеры, так как источники *S* все еще востребованы в данной группе. Эту информацию можно закодировать одним битом в формате запроса. По смыслу этот бит — просто флаг, надо ли наблюдателям понижать соответствующие таймеры. Он возникнет на месте поля «резерв», поэтому его значение 1 отвечает новому поведению: не понижать таймеры.

Отсюда его название: «подавить обработку маршрутизаторами» (*Suppress Router-Side Processing*), сокращенно «флаг **S**» [§5.1.7 RFC 3810].

Как генератор запросов определит, что пора установить этот флаг? Если мы предположим, что слушатели подтвердили свой интерес ко всем источникам множества *S* в запросе, то будет достаточно отметить этот факт установкой флага **S** в том образе запроса, который генератор хранит в своей оперативной памяти. Тогда последующие копии запроса уйдут с **S** = 1. Однако в действительности может оказаться, что подтвержден интерес только к подмножеству источников $S_1 \subset S$. Как генератору запросов разделить подмножества S_1 и $S_2 = S \setminus S_1$, когда придет время повторить запрос? Здесь ему помогут значения таймеров. Ведь у подтвержденного подмножества S_1 таймеры перезапущены и снова отсчитывают длинный интервал *MALI*, тогда как у неподтвержденного подмножества S_2 значения таймеров по-прежнему не превышают *LLQT*. Так как флаг **S** один на все сообщение, генератору придется разбить запрос на два. В первом из них будет перечислено подмножество источников S_1 и установлен флаг **S** (**S** = 1); во втором же окажется подмножество источников S_2, а флаг **S** будет сброшен (**S** = 0).

Флаг **S** имеет смысл и для запросов, ограниченных только группой. В этом случае речь идет о подтверждении исключающего режима. Так как маршрутизатор не может находиться в нем наполовину, делить будущие копии запроса ему точно не придется. Тем не менее, критерий, установить ли флаг **S**, может быть тем же, с точностью до замены таймера источника на таймер фильтра: если таймер фильтра по-прежнему не превышает *LLQT*, то **S** = 0, а если он вдруг стал больше *LLQT*, то значит, пора установить **S** = 1.

Теперь наблюдатели понижают таймеры не по любому запросу Q(Г) или Q(Г,*S*), а только если в нем **S** = 0. Для нас это повод уточнить условия, когда то же самое делает генератор запросов. Если бы наблюдателей не было вообще, генератор мог бы сделать это единожды, в самом начале серии запросов. Ведь вся наша арифметика тайм-аутов *MALI* и *LLQT* была основана именно на этой модели. С другой стороны, если бы наблюдатели существовали, но пакеты всегда достигали бы цели, генератор тоже мог бы понизить таймеры в начале серии, а флаг **S** установить сразу после первого запроса, чтобы наблюдатели поступили с таймерами так же. То есть первый запрос серии вызвал бы установку таймеров на всех маршрутизаторах канала, а последующие дубликаты запроса на них бы не влияли. Но в действительности первый запрос серии может потеряться, как любой другой. Именно поэтому генератор запросов устанавливает флаг **S** не раньше, чем он получит подтверждение, что энный запрос хоть до кого-то дошел (в данном случае до слушателя). Поэтому пусть ради синхронизации с наблюдателями генератор тоже продолжает понижать соответствующие таймеры до *LLQT* всякий раз, пока он передает запрос с **S** = 0.

> Конечно, этот трюк нарушает четкость нашего плана, как согласовать тайм-аут записи с ритмом повторяемых запросов, но он помогает синхронизации маршрутизаторов канала. С одной стороны, теперь время жизни записи может продлиться дольше, чем это необходимо. Так, если заинтересованных слушателей не осталось и на запрос ответить некому, запись проживет интервал *LLQT* после последнего запроса, хотя по нашему первоначальному плану она прекратила бы свое существование через минимально необходимое время *LLQI*. С другой стороны, теперь выше вероятность того, что тайм-аут записи произойдет практически одновременно на всех маршрутизаторах.

Окончательные правила синхронизации между генератором запросов и пассивными слушателями оказываются такими:

1. Подготовка к передаче запроса — только для генератора [§7.6.3 RFC 3810]:

a. Запрос, ограниченный только группой Г: Если таймер фильтра Г меньше или равен *LLQT*, то флаг **S** надо сбросить в 0, а иначе установить в 1.
b. Запрос, ограниченный группой Г и списком источников *S*: Если все таймеры *S* меньше или равны *LLQT*, то флаг **S** надо сбросить в 0, а если больше — установить в 1. В смешанном случае запрос надо разбить на два, один с **S** = 0, а другой с **S** = 1.

2. Действия по приему или передаче запроса — как для генератора, так и наблюдателей [§7.6.1 RFC 3810]:
   a. Если в запросе флаг **S** сброшен в 0:
      i. Запрос, ограниченный только группой Г: Понизить таймер фильтра Г до *LLQT*.
      ii. Запрос, ограниченный группой Г и списком источников *S*: Понизить таймеры источников из списка *S* до *LLQT*.
   b. Если же в запросе флаг **S** установлен в 1, то текущие таймеры не трогать.

> Эта процедура действительно способна помочь синхронизации между генератором запросов и наблюдателями после кратковременного сбоя в сети. Например, если в нашем вышеизложенном сценарии самый первый отчет слушателя дойдет до наблюдателя, но не дойдет до генератора, наблюдатель повысит таймеры до *MALI*, тогда как у генератора они продолжат отсчитывать короткий интервал *LLQT*. Однако затем генератор повторит свой запрос с **S** = 0, и наблюдатель снова понизит таймеры, так что синхронизация будет восстановлена.

На этом мы почти завершили усовершенствование MLD и практически готовы объявить вторую версию протокола готовой к применению. Последнее, о чем нам следует подумать, — это совместимость с первой версией.

У совместимости между протоколами, по большому счету, два аспекта. Первый из них — это совместимость сообщений, которыми обмениваются стороны протокола, а второй — совместимое поведение сторон в ответ на принятые сообщения. Конечно же, эти аспекты связаны между собой.

Совместимость сообщений — это, по сути, однозначность их интерпретации. Скажем, если в запросе MLDv2 мы изменим код поля «максимальная задержка отклика»,[119] то нам надо позаботиться, чтобы слушатель MLDv1 не применил к этому полю ошибочную интерпретацию. Продемонстрируем это на практике. Мы как раз собирались разработать новый код для значений *QI* и *MRD* в запросе, так что давайте сделаем его обратно совместимым с целым беззнаковым кодом.

Вот один из способов обеспечить такую совместимость: пусть старший бит поля указывает на способ его кодирования. Когда этот бит сброшен, остаток поля составлен в целом беззнаковом коде. Очевидно, что старая реализация не знает о такой роли старшего бита и примет его за нулевой старший разряд. В результате она получит то же самое численное значения поля, и совместимость будет достигнута. Пока маршрутизатор MLDv2 работает в режиме совместимости, он вправе пользоваться только этим кодом. Когда же совместимость не требуется, можно применить альтернативный код, на который укажет установленный старший бит поля.

Наша цель — представить широкий диапазон значений с разумной точностью, а этому условию отвечает код с плавающей запятой, похожий на экспоненциальную (научную) нотацию чисел. В этом коде число приближенно записывают как мантиссу в оптимальном диапазоне, помноженную на степень основания системы счисления, в данном случае 2: $x = M \cdot 2^E$. Здесь M — мантисса фиксированной точности, а E — целочисленный порядок числа.

[119] В MLDv1 это поле обозначалось как **MRD**.

Двоичная мантисса фиксированной разрядности будет оптимальной, если все ее биты будут значащими. Это также называют нормализованной записью. Проще говоря, в мантиссе не должно быть ведущих нулей, потому что они не содержат информации. Следовательно, старший бит в *M* обязан быть равен единице, ведь иначе он будет незначащим. Поэтому установленный старший бит можно подразумевать, а не хранить явно, и этим выиграть еще один младший бит. Так мы компенсируем расход одного бита на указание способа кодирования поля.

> Хотя обычно принимают, что в нормализованной записи запятая находится после самого старшего разряда мантиссы, нам сейчас удобнее поместить ее после самого младшего разряда, чтобы мантисса оставалась целой. Очевидно, что эти два представления однозначно преобразуются друг в друга простым сдвигом, то есть изменением порядка *E* на число явно хранимых разрядов мантиссы, если старший разряд подразумевается. Например, двоичная мантисса 1,0001 преобразуется в 10001 вычитанием 4 из порядка, чтобы значение закодированного числа оставалось неизменным.

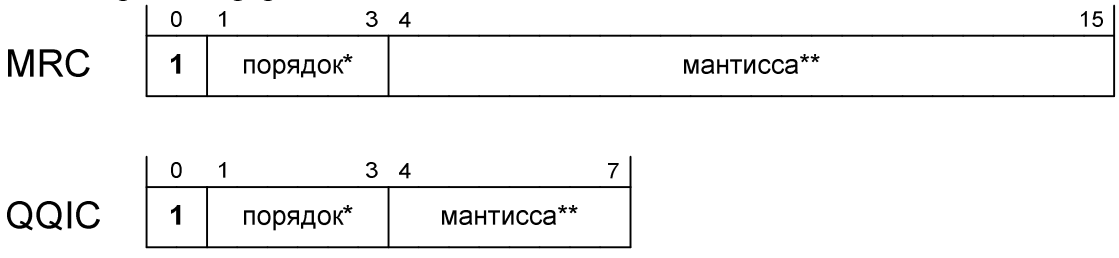
Применим такой код к 16-битному полю, где будет храниться значение *MRD*. Ввиду нового кода это поле и обозначают по-другому: **MRC** (*Maximum Response Code*, код максимальной задержки отклика) [§5.1.3 RFC 3810]. Пусть порядок занимает 3 бита. Еще один бит «съел» флаг кода. Тогда на мантиссу остается 12 бит. Значения до 32767 мы уже можем представить в старом коде, поэтому нам интересен диапазон от $32768 = 2^{15}$ и выше. Этому отвечают значения порядка от 3: $\log_2 \frac{2^{15}}{2^{12}} = 3$. Поэтому нам следует сместить порядок на 3 для расширения диапазона. Двоичные значения поля мантиссы M' и поля порядка E' будут представлены в обычном целом беззнаковом коде. Тогда численное значение *MRD* в новом коде будет равно $(0\text{x}1000 + M') \cdot 2^{E'+3}$.

> Рассчитайте, каковы минимальное и максимальное значения поля **MRC** в новом коде. (32 768 и 8 387 584.)

Тем же самым способом мы решим задачу о кодировании поля **QQIC** для хранения значений *QQI* [§5.1.9 RFC 3810]. Это поле длиной 8 бит, причем 3 из них отведены под порядок и 4 под мантиссу. Следовательно, численное значение поля, когда его старший бит установлен, следует вычислять так: $QQI = (0\text{x}10 + M') \cdot 2^{E'+3}$.

> Рассчитайте, каковы минимальное и максимальное значения поля **QQIC** в новом коде. (128 и 31 744.)

Расширенные форматы полей **MRC** и **QQIC** в отчете MLDv2 показаны на Фиг. 99.



*) Истинный порядок числа больше этого значения на 3.
**) Мантисса хранится без старшего бита, который всегда равен 1.

Фиг. 99. Код с плавающей запятой для полей MRC и QQIC

Решив вопрос о совместимости на уровне отдельных полей, перейдем к совместимости формата сообщений в целом. По своему формату отчет MLDv2 в корне отличается от отчета MLDv1, так как он содержит переменное число записей о разных группах, а каждая запись имеет определенный тип и может содержать список источников. Лучшее, что мы можем сделать ради совместимости как однозначности интерпретации, —

это полностью разделить отчеты MLDv1 и MLDv2. Для этого достаточно назначить отчету MLDv2 новый тип ICMPv6 (143).

Что же касается запросов, то их формат изменился не так радикально. Более того, он вполне отвечает модели расширения протокола MLD, когда новые данные помещаются в конец сообщения, после его совместимой части. Благодаря этому запрос MLDv2 может сохранить за собой тот же самый тип ICMPv6 (130). Если реализация поддерживает обе версии MLD, то ей просто определить версию запроса на входе: запрос MLDv1 всегда длиной 24 байта, тогда как длина запроса MLDv2 — от 28 байт и выше. Диапазон длин от 25 до 27 байт запретный и не отвечает никакой версии MLD.

> Обратите внимание, что длина сообщения MLDv2 — как запроса, так и отчета — закодирована в самом сообщении. Поэтому отличить будущие версии от MLDv2 можно будет, только декодировав совместимую с MLDv2 часть.

Наконец, мы добрались до совместимого поведения сторон протокола. Здесь нас, в первую очередь, интересуют два случая: поведение слушателей MLDv1 в ответ на запросы MLDv2 и поведение маршрутизаторов MLDv2 в ответ на отчеты MLDv1. Это видно из таблиц совместимости Табл. 22 и Табл. 23.

Табл. 22. Совместимость слушателей MLD

Версия слушателя	Запрос MLDv1	Запрос MLDv2
MLDv1	нет проблем	**как быть?**
MLDv2	отчет MLDv1	нет проблем

Табл. 23. Совместимость маршрутизаторов MLD

Версия маршрутизатора	Отчет или итог MLDv1	Отчет MLDv2
MLDv1	нет проблем	проигнорирует
MLDv2	**как быть?**	нет проблем

> Слушатели MLDv2 не должны отвечать отчетами MLDv2 на запросы MLDv1 хотя бы потому, что маршрутизатор MLDv1 проигнорирует их ввиду неизвестного типа ICMPv6.

В первом случае слушатель ответит отчетом MLDv1 о данной группе, или же целой серией отчетов обо всех своих группах, если запрос был общий. Тогда маршрутизатор MLDv2 сможет взять на заметку, что на канале есть как минимум один слушатель MLDv1, и включить режим совместимости.

Это естественным порядком приводит нас ко второму случаю. Маршрутизатор MLDv2 как более современный легко декодирует сообщение MLDv1, поэтому вопрос состоит в том, насколько усложнит реализацию режим совместимости. В частности, понадобится ли маршрутизатору MLDv2 отдельный алгоритм реакции на сообщения MLDv1? К счастью, нет, потому что по содержащейся в них информации отчет и итог MLDv1 — это лишь подмножество многофункциональных отчетов MLDv2. На самом деле, мы об этом уже говорили, так что сейчас нам осталось только составить таблицу соответствия — Табл. 24.

Табл. 24. «Апгрейд» сообщений MLDv1 до MLDv2

Сообщение MLDv1	Эквивалент MLDv2
Отчет	IS_EX(∅)
Итог	TO_IN(∅)

Благодаря такому соответствию, маршрутизатор MLDv2 может обрабатывать отчеты и итоги MLDv1 по уже готовым правилам протокола MLDv2.

Конечно, маршрутизатору MLDv2 придется учесть, что адрес назначения отчета или итога MLDv1 равен группе, о которой идет речь, а не *FF02::16*. Это не вызовет проблем, так как маршрутизатор ведет теневой прием всех групп и руководствуется, в первую очередь, опцией «сигнал маршрутизатору» в сообщениях MLD, а не их адресами назначения.

Помимо этих двух очевидных случаев взаимодействия сторон MLD, нам надо обратить внимание на «перекрестное опыление» слушателей и маршрутизаторов (Табл. 25). Ведь в принципе слушатели могут получать отчеты, а маршрутизаторы — запросы. К счастью, с первой частью проблем нет, так как отчеты MLDv2 направляются по специальному групповому адресу, и слушатели их не получат, независимо от их версии. Что касается реакции слушателя MLDv2 на отчет MLDv1, связанная с этим оптимизация (подавление отчетов) упразднена, и отчет можно безопасно игнорировать.

Тем не менее, слушателю MLDv2 не запрещено подавлять свои отчеты при получении чужих отчетов MLDv1 [§8.2.2 RFC 3810].

Табл. 25. «Перекрестное опыление» слушателей MLD

Версия слушателя	Отчет или итог MLDv1	Отчет MLDv2
MLDv1	нет проблем	не получит
MLDv2	может игнорировать	не получит

В то же время, маршрутизатор обязан реагировать на чужие запросы, так как на этом основаны процедуры выбора генератора запросов и последующей синхронизации с ним. Этот случай самый сложный, так как правила реакции маршрутизаторов на чужие запросы в MLDv1 и MLDv2 существенным образом отличаются. В частности, маршрутизаторы MLDv1 не смогут использовать дополнительную информацию из запроса MLDv2 для точной синхронизации с генератором.

Поэтому, ради устойчивой работы сети, мы объявим нежелательной смешанную конфигурацию с использованием маршрутизаторов MLDv1 и MLDv2 на одном канале. Точнее, пусть маршрутизатор MLDv2 использует только запросы MLDv1, пока на том же канале есть маршрутизаторы MLDv1 [§8.3.1 RFC 3810]. Это приемлемый компромисс, так как маршрутизаторов меньше, чем хостов, а их взаимозаменяемость выше. Если обновление хостов — процесс долгий, трудоемкий и требующий поэтапного подхода, переключение подсети[120] на MLDv2 можно провести за один шаг, например, путем установки на канал одного или нескольких новых групповых маршрутизаторов и временного отключения старых. Затем будет достаточно постепенно обновлять старые маршрутизаторы и возвращать их обратно в эксплуатацию.

Предложите решение для вот какой проблемы. Допустим, последний маршрутизатор MLDv1 только что убрали с канала. Но маршрутизаторы MLDv2, находясь в режиме совместимости, получают запросы MLDv1 друг от друга и потому пребывают в уверенности, что режим совместимости по-прежнему необходим. Могут ли маршрутизаторы MLDv2 самостоятельно выйти из режима совместимости, а если да, то как? (RFC предлагает ручную настройку.)

Сводная таблица совместимости между сторонами MLDv1 и MLDv2 будет такой, как показано в Табл. 26.

Табл. 26. Совместимость между сторонами MLD: сообщения какой версии использовать?

Хосты ↓ \ Маршрутизаторы →	Только v1	Только v2	v1 и v2
Только v1	v1	Запросы v2, отчеты v1	v1
Только v2	v1	v2	v1

[120] В старом смысле этого термина: транзитная инфраструктура сети.

v1 и v2	v1	Запросы v2, отчеты v1 и v2	v1

А вот сравнительная таблица главных характеристик этих протоколов — Табл. 27.

Табл. 27. Сравнение возможностей MLDv1 и MLDv2

	MLDv1	**MLDv2**
Типы сообщений ICMPv6	«запрос MLD» (130), «отчет MLD» (131), «итог MLD» (132)	«запрос MLD» (130), «отчет MLDv2» (143)
Виды запроса	Общий и ограниченный группой	Общий; ограниченный группой; ограниченный группой и источником
Форма отчета	Одна группа на сообщение	Несколько групп на сообщение; каждую группу сопровождает режим фильтрации и список источников
Адрес назначения в отчете и итоге	Объявляемая группа	«Все маршрутизаторы MLDv2»
Выборы генератора запросов	По адресу IP	По адресу IP
Подавление избыточных отчетов	Да	Упразднено
Поддержка SSM и SFM	Нет	Да
Кодирование интервала таймера	Целочисленное	С плавающей точкой, обратно совместимое

Последним аспектом MLD, который мы не можем обойти вниманием, будет безопасность [§10 RFC 3810]. В какой степени этот протокол защищен от атак, и, с другой стороны, насколько опасны эти атаки? В отличие от ND, состояние стороны MLD зависит только от принятых ей сообщений MLD и не зависит от других внешних событий.

> Напротив, состояние ND может зависеть от любых пакетов IPv6. Так, маршрутизатор обращается к модулю ND по приходу каждого транзитного пакета.

Поэтому единственный способ атаковать MLD — это подделка сообщений MLD. Главной мерой защиты от такой подделки будет изоляция канала от любых сообщений MLD извне. Действительно, по своему устройству MLD — стопроцентно внутриканальный протокол. Здесь можно пустить в ход весь арсенал доступных средств. Прежде всего, на выходе адрес источника сообщения MLD обязан быть внутриканальным за тем единственным исключением, что в отчете он может быть неопределенным. Тем не менее, отчеты с неопределенным адресом источника необходимы только для подготовки коммутаторов ЛВС на стадии SLAAC (§5.4.2) или DAD (§5.4.1), тогда как узлы IPv6 вполне могут игнорировать их на входе [§5.2.13 RFC 3810]. Действительно, сообщения ND, из которых состоят SLAAC и DAD, все равно не подлежат маршрутизации, и маршрутизаторам нет дела до членства слушателей в группах искомого узла и группе «все узлы канала». А позднее, когда узел назначит себе внутриканальный адрес, он повторит свои отчеты уже с определенным адресом источника. Поэтому на входе адрес источника в сообщении MLD должен быть внутриканальным, а иначе такое сообщение надо просто игнорировать.

Далее, все сообщения MLD несут опцию «сигнал маршрутизатору», и это существенно облегчает выделение этих пакетов из общего потока. Следовательно, на входе все узлы обязаны убедиться, что пакет MLD снабжен этой опцией.

Маршрутизаторам к тому же запрещено продвигать пакеты MLD куда бы то ни было. Продвижение MLD в другие каналы запрещено из соображений безопасности, а в тот же самый канал — ради устойчивости сети. Как мы уже отмечали, зацикливание группового трафика может иметь лавинный характер, когда каждая копия пакета рождает несколько копий следующего поколения и т.д. По этой причине MLD не защищают при помощи GTSM, а наоборот, устанавливают в пакетах MLD «предельное число шагов» равным 1. Ведь продвижение пакетов MLD в другие каналы уже заблокировано зонной архитектурой, тогда как продвижение в тот же самый канал она вполне допускает, как было сказано в §2.4.

Справившись с внешними врагами, мы можем перейти к врагам внутриканальным, а они, конечно, более опасны. Главная угроза исходит здесь от подделки или просто несанкционированной отправки запросов MLD. Для начала злоумышленник способен провести атаку на выборы генератора. Для этого ему достаточно вести передачу запросов с как можно меньшего внутриканального адреса, например, *FE80::* или *FE80::1*. Заполучив роль генератора запросов, или параллельно с этой атакой, злоумышленник может подавить всякую маршрутизацию группового трафика в данный канал. Для этого ему достаточно периодически высылать запрос, ограниченный никому не интересной группой. Как нетрудно убедиться, в этом случае законные маршрутизаторы будут вечно пребывать в состоянии пассивных наблюдателей, а слушатели никогда не получат повод послать свои отчеты. В результате законные маршрутизаторы будут обманным способом убеждены в том, что никаких слушателей на канале просто нет.

> На первый взгляд, такой атаке можно противопоставить модифицированную процедуру выборов, когда таймер *OQPT* перезапускается только по общему запросу. Однако у злодея в запасе есть ответный ход: слать общий запрос по адресу «все маршрутизаторы MLDv2», *FF02::16*. Маршрутизаторы обязаны принять и обработать такой запрос, как обычно [§5.1.15 RFC 3810]. В ответ злодею представим себе, что мы внесли поправку в протокол, и теперь общий запрос *обязан* быть адресован группе «все узлы», *FF02::1*. Тогда злодей разыграет козырь и пошлет такой запрос инкапсулированным в кадр *Ethernet*, где адрес назначения MAC равен *33-33-00-00-00-16*. Благодаря фильтрации группового трафика на канальном уровне, в коммутируемой ЛВС такой кадр с большой вероятностью попадет только к маршрутизаторам MLDv2 [121].
>
> Практический рецепт против такой атаки — это не принимать пакеты с адреса *FE80::*, потому что это зарезервированный в §6.1 адрес anycast, и вручную назначить законным маршрутизаторам наименьшие внутриканальные адреса: *FE80::1*, *FE80::2* и т.д. Однако у злодея снова появится шанс, как только маршрутизатор *FE80::1* выйдет из строя, например, из-за атаки типа «отказ в обслуживании».
>
> В стратегической перспективе, достойный ответ атакам на MLD — это сертификация групповых маршрутизаторов аналогично SEND из §6.3.

В то же время, подделка отчетов MLD — это оксюморон, так как вступление в группу или просто начало ее приема — добровольная операция, доступная любому узлу канала. Конечно, злоумышленник может провести атаку типа «отказ в обслуживании», послав отчет о приеме большого числа групп, а вдобавок замести следы, подделав адрес источника. С первой частью этой атаки можно бороться, ограничив на маршрутизаторе число продвигаемых групп, — это все равно придется сделать из соображений устойчивости. Что касается подделки адреса источника, то здесь она не несет критической опасности, так как в MLD «личность» слушателя практически неважна, хотя ловить злоумышленника будет чуть сложнее.

---

[121] Marc “vanHauser” Heuse. Recent advances in IPv6 insecurities. 27th Chaos Communication Congress. Berlin, 2010. <http://events.ccc.de/congress/2010/Fahrplan/events/3957.en.html>

Другое дело, что бесконтрольное вступление хостов в группы небезопасно и может привести, например, к утечке конфиденциального трафика. Чтобы управлять членством в группах, понадобится инфраструктура аутентификации и авторизации слушателей MLD. Альтернативное решение проблемы состоит в сквозной защите группового трафика [RFC 3740], а это лучше отвечает фундаментальному принципу TCP/IP: не рассчитывай на помощь транзитных узлов, если без нее можно обойтись.

Небезопасна и передача трафика группе, которую может вести любой источник. Режимы SFM и SSM сами по себе не делают ее безопасной, так как адрес источника можно подделать. Здесь тоже поможет система сквозной защиты, этакий «групповой IPsec».

6.5. Фрагментировать или нет?

Механизм управления фрагментацией IPv6 требует дополнительных соображений и комментариев, потому что он существенным образом отличается от его аналога в IPv4. Главное отличие, конечно же, в том, что фрагментацию IPv6 может вести только источник, хотя информация об оптимальной длине пакета скрыта дальше в сети.

В первую очередь, такая особенность механизма фрагментации IP влияет на работу транспортных протоколов. Начнем с хрестоматийного примера TCP. Раньше, в среде IPv4, перед TCP было два основных пути:

- сбрасывать флаг **DF** и делить поток прикладных данных на сегменты произвольного размера, поручив их дальнейшую фрагментацию сети;
- устанавливать флаг **DF** и выполнять PMTUD, избегая фрагментации IP и пытаясь оптимизировать размер сегмента.

Более хитроумная реализация TCP могла пробовать второй путь, но переключаться на первый, если PMTUD не работает, например, из-за слишком строгой фильтрации пакетов ICMP в сети.

В среде IPv6 первый путь полностью закрыт, потому что флаг **DF** неявно установлен всегда, а пакеты, длина которых больше PMTU, не достигнут адресата ни при каких условиях. Теперь сам источник пакета должен выбрать его подходящий размер. Как следствие, при переходе к IPv6 растет важность правильной и надежной работы PMTUD.

В первую очередь, этот факт касается сетевых администраторов, увлеченных борьбой с «нежелательным» трафиком ICMP.

Тем не менее, даже ненадежная сигнализация ICMP не мешает проводить PMTUD, используя сквозной принцип[122] и некоторую эвристику. Например, если «тройное рукопожатие» TCP проходит успешно, но затем оказывается, что полноразмерные сегменты куда-то пропадают — удаленная сторона не квитирует новые данные, — то здесь явно замешан PMTU, и источнику TCP следует понизить размер сегмента. Хотя подсказок в виде очередного значения MTU он не получит, его выручит двоичный поиск, ограниченный снизу заведомо «габаритным» значением (1280 байт в IPv6), а сверху — MTU локального интерфейса. Пробное значение можно повышать, пока сегменты достигают удаленной стороны. Как только оценка окажется завышенной, сегменты перестанут проходить, и надо будет перейти к понижению пробной величины. На этой идее основана так называемая процедура PMTUD уровня пакетизации [RFC 4821].

Тем не менее, PMTUD — не абсолютная необходимость даже в среде IPv6. Теперь минимальный размер MTU равен относительно большой величине, 1280 байт, так что примитивная реализация TCP может себе позволить ориентироваться на эту постоянную.

[122] «Надейся в первую очередь на себя, затем на удаленную сторону, и только в последнюю на транзитные узлы».

Так же может поступить и более сложный вариант TCP, если он обнаружит, что PMTUD все-таки не работает.

Итак, в среде IPv6 перед TCP открываются следующие пути:

- сегментировать поток, исходя из величины PMTU 1280 байт;
- оптимизировать размер сегмента с помощью PMTUD.

TCP — это пример транспортного протокола, который может плавно варьировать размер сегмента и поэтому не нуждается во фрагментации на уровне IPv6. Совсем другую картину мы увидим, если перенесем наш взгляд на блочные транспортные протоколы и их потребителей.

Представим себе, что некий прикладной протокол работает поверх UDP. В этом случае размер исходящей дейтаграммы задает приложение, и размер этот может достигать внушительной величины. Это не наша фантазия: в реально существующих протоколах, например, в NFS, возникают сообщения длиной порядка нескольких тысяч байт. Как подобная дейтаграмма достигнет адресата?

Первый подход мы уже нащупали: приложение может посредством сетевого API попросить локальный модуль IPv6, чтобы тот фрагментировал пакеты исходя из MTU 1280 байт. Это не самый эффективный, зато простой и надежный путь: из-за своего чрезмерного размера пакеты теряться заведомо не будут.

Тем не менее, пакеты теряются еще по ряду причин, и прикладной протокол должен быть устойчив (или безразличен) к таким потерям. Данное свойство прикладных протоколов, работающих поверх UDP, позволяет оптимизировать размер фрагмента IPv6. Для этого надо, чтобы модуль IPv6 сам выполнял PMTUD. Как это будет работать? Например, так:

1) приложение шлет данному адресату свое первое большое сообщение;
2) модуль IPv6 фрагментирует составленный пакет, взяв первым приближением PMTU величину MTU выходного интерфейса;
3) возможно, такие фрагменты не пройдут всю трассу; но модуль IPv6 примет извещения ICMPv6 «пакет слишком велик» и получит второе приближение PMTU;

> Это приближение будет довольно точным благодаря обязательному значению MTU в сообщении «пакет слишком велик». В классическом IPv4 источнику приходилось вести вместо этого поиск вслепую, так как маршрутизатор не подсказывал значения MTU. Впрочем, двоичный поиск сходится довольно быстро. Подумайте, в каком случае двоичный поиск вслепую выиграет у поиска с подсказкой маршрутизатора. (Например, когда число шагов достаточно велико, а MTU каждого нового шага меньше предыдущего — «телескопическая трасса».)

4) приложение обнаружит потерю сообщения, например, не получив отклика удаленной стороны в течение какого-то времени, и повторит сообщение (или пошлет следующее, если в данном протоколе потери можно игнорировать);
5) модуль IPv6 фрагментирует пакет, используя новое приближение PMTU;
6) возможно, фрагменты опять не пройдут; но в результате модуль IPv6 снова уточнит величину PMTU;
7) приложение еще раз повторит сообщение (или пошлет следующее)…

И так до тех пор, пока приближение PMTU не станет меньше или равно его истинному значению для данной трассы. После этого фрагментированные пакеты начнут доходить до адресата, и прикладной сеанс продолжится. Чтобы подобная схема работала, модуль IPv6 должен кэшировать значения PMTU для своих недавних адресатов. Подходящее место для этих сведений — кэш адресатов, DC [§5.2 RFC 1981].

> Как мы помним, трасса — это путь по сети, от источника к адресату через транзитные узлы, который проходит (или прошел бы) пакет в данный момент времени. Для простоты мы сейчас не будем говорить о маршрутизации от

источника, когда трассу выбирает источник пакета, и маршрутизации согласно политике, когда выбор следующего шага основан на произвольных характеристиках пакета и внешних параметрах. Тогда трасса IP зависит от узла-источника и от адреса назначения пакета, а также от настроек источника и каждого транзитного узла в момент обработки пакета (грубо говоря, от их таблиц маршрутов). Конечно, в «живой» сети трасса может со временем меняться вместе с настройками источника и транзитных узлов. Однако, пока сеть стабильна, трасса остается постоянной хотя бы какое-то время. Именно поэтому источник имеет право кэшировать значения PMTU, используя в роли ключа адрес назначения.

Однозначность соответствия между адресатом и наблюдаемой величиной PMTU может оказаться нарушена, если сеть использует ECMP и пакеты одному адресату могут приходить по разным путям с неодинаковым значением PMTU. Между тем, существующие реализации хоста IP кэшируют одно значение PMTU на адресата еще со времен IPv4, а IPv6 лишь стандартизует эту практику. Поэтому надо с осторожностью применять распределяющие хэш-функции ECMP, зависящие от чего-либо сверх адреса назначения пакета, например, номера вышестоящего протокола или портов TCP/UDP. (Обсудите самостоятельно, насколько допустимо, чтобы такая хэш-функция зависела от адреса источника пакета.)

Еще один путь открывается, когда прикладной протокол позволяет варьировать размер сообщений, например, если приложение сегментирует некий поток байтов или более крупных единиц данных. В этом случае приложение могло бы выбрать такой размер своих сообщений, чтобы их фрагментация на данной сетевой трассе не требовалась.[123] Но, чтобы приложение узнало подходящий размер, требуется обратная связь от локального модуля IPv6 к приложению. Ее тоже можно обеспечить в рамках сетевого API. Скажем, приложение может запрашивать у модуля IPv6 текущее приближение PMTU для данного адресата. Или же модуль IPv6 может подсказывать приложению PMTU по своей инициативе, в виде служебной структуры,[124] когда приложение ожидает ответа удаленной стороны, а вместо него приходит извещение ICMPv6 «пакет слишком велик».

Наконец, оптимистичное приложение, которое верит в возможности PMTUD, захочет запретить модулю IPv6 фрагментацию своих исходящих пакетов. Такое приложение должно активно следить за текущим приближением PMTU и корректировать размер своих сообщений. Модуль IPv6 и приложение проводят PMTUD в паре: модуль IPv6 обрабатывает извещения ICMPv6 и оценивает PMTU, а приложение генерирует новые пробные пакеты согласно последней оценке.

Именно эти идеи нашли отражение в расширенном API для IPv6, основанном на сокетах Беркли [RFC 3542]. Приложение управляет фрагментацией и PMTUD с помощью опций сокета из Табл. 28 на уровне **IPPROTO_IPV6** [§11 RFC 3542]. Некоторые опции можно избирательно устанавливать для отдельных сообщений с помощью служебной структуры *cmsghdr*.

Табл. 28. Опции сокета, управляющие фрагментацией IPv6

Опция	Смысл
IPV6_USE_MIN_MTU	Использовать минимальный MTU (1280 байт)
IPV6_PATHMTU	Запросить приближение PMTU для подключенного сокета
IPV6_RECVPATHMTU	Разрешить возврат приближения PMTU из вызова *recvfrom()*
IPV6_DONTFRAG	Запретить фрагментацию исходящих пакетов сокета

[123] Этот прием иногда называют «семантическая фрагментация», так как он требует хотя бы минимального понимания, что значат передаваемые данные. Например, мы его уже встретили в MLDv2 (§6.4).

[124] Это *cmsghdr* в терминах API сокетов Беркли.

> Обсудите преимущества и недостатки альтернативного подхода к определению PMTU, когда каждый транзитный узел по трассе пакета записывает фактическое значение MTU в специальную пошаговую опцию, если ее текущее значение больше MTU [draft-park-pmtu-ipv6-option-header], так что прошедший всю трассу короткий пакет содержит в себе точное значение PMTU.

Еще один практический способ понизить долю фрагментированных пакетов — это не занижать MTU интерфейсов без необходимости [§5 RFC 2460]. Сегодня стандарт де-факто для подключения конечных пользователей на канальном уровне — это *Ethernet*, а он диктует значение MTU 1500 байт. Это же значение мы встречаем и в других канальных протоколах, например, в PPP [§2 RFC 1661]. В центре сети может понадобиться запас MTU сверх 1500 байт, чтобы пакеты от конечных пользователей можно было подвергать дополнительной инкапсуляции, например, туннелировать, не вызывая фрагментации. Конечно, в сложной сети с несколькими административными зонами выбрать для каждого канала оптимальное значение MTU может быть непросто. Тем не менее, опускать его ниже 1500 байт точно не стоит.

Еще мы до сих пор не задали минимальный размер буфера сборки IPv6. Как мы помним, в IPv4 он составлял 576 байт. Теперь мы видим значение, которое лучше подходит современным условиям: 1500 байт. Иными словами, всякий узел IPv6 обязан собрать адресованный ему пакет, если длина пакета после сборки не превышает 1500 байт [§5 RFC 2460]. Какой вывод должны сделать вышестоящие протоколы? Не следует создавать пакеты длиннее 1500 байт, если нет уверенности, что адресат готов принимать их. Например, в TCP подобные сведения можно извлечь из опции MSS.

> В качестве упражнения вычислите наиболее вероятные длины полноразмерных сегментов TCP, которые можно будет увидеть в сети после перехода на IPv6. Не забудьте, что фактическая длина полноразмерного сегмента TCP может оказаться меньше значения опции MSS [RFC 6691]. (Подсказка: учтите возможность расширений из [RFC 1323].)

6.6. Работа с множеством локальных адресов. Выбор адресов по умолчанию

Как мы сказали в §2.5, у хоста IPv6 может быть несколько сетевых адресов. На самом деле, один адрес бывает только у хоста IPv6, который не подключен к сети, — это адрес обратной связи *::1*. Если у хоста есть хотя бы один активный сетевой интерфейс кроме «петли», то число его адресов возрастает до двух, так как этому интерфейсу обязательно назначен внутриканальный адрес IPv6. А если интерфейс хоста получает еще и глобальный адрес IPv6, то адресов у хоста становится три. Как мы видим, три индивидуальных адреса — это минимум для хоста IPv6, подключенного к *Internet*.

Однако три адреса — это еще не предел. Положим, у хоста всего один сетевой интерфейс. Он находится в разных зонах, и каждой зоне может отвечать, по меньшей мере, один локальный адрес. Хотя на практике мы встречаем только внутриканальную и глобальную области, адресация IPv6 вполне допускает области промежуточной величины и отвечающие им зоны.

> Мы помним из §2.4, что у всякого численного адреса IPv6 ровно одна область действия, а назначенный интерфейсу адрес принадлежит ровно одной зоне. В то же время, сетевой интерфейс находится в зонах всех возможных областей, независимо от назначенных ему адресов.

Далее, сетевой интерфейс хоста может получить несколько адресов из одной зоны. Для чего это может быть полезно? Адреса из одной подсети назначают, когда, например, функции хоста А переходят хосту Б, а хост А прекращает свое существование. В этом случае можно назначить хосту Б, кроме его собственного адреса, бывший адрес хоста А, и тогда не придется менять настройки других элементов сети: хост Б станет отвечать от

имени хоста А, и никто не заметит изменений. Адреса из разных подсетей открывают путь к тому, чтобы использовать одновременные подключения к нескольким провайдерам *Internet*.

> Конечно, полноценная работа с несколькими подключениями — это существенно более сложная задача. Ее анализу посвящен [RFC 4177], а мы вкратце рассмотрим ее в §6.8 настоящей книги.

Нетрудно предвидеть, что ситуация только усложнится, когда у хоста будет несколько активных сетевых интерфейсов. Каждый из них получит, помимо обязательного внутриканального адреса, ноль или больше адресов согласно адресному плану сети.

Рассмотрим с этих позиций типовой сценарий: локальный хост Л *по своей инициативе* собирается послать пакет удаленному хосту У. Допустим, что приложение, запущенное на хосте Л, хочет направить в адрес У дейтаграмму UDP или установить с У соединение TCP. Чтобы составить такой пакет, сетевой стек хоста Л — не без помощи приложения и даже самого пользователя — должен перейти от абстрактных названий «Л» и «У» к паре адресов, источника и назначения пакета.

> Нам сейчас интересен именно случай, когда у хоста Л есть выбор, какие адреса указать в пакете. Если же хост Л отвечает на пакет хоста У, то свободы у него существенно меньше. В классических протоколах единственно верным поведением хоста Л будет использовать уже приведенные в пакете адреса, просто переставив их местами. В противном случае хост У, скорее всего, не примет такой ответ. Современные протоколы, например, Shim6 (§6.8) и SCTP, поддерживают больше одного адреса на каждом конце сеанса, но их выбором руководят соображения доступности адреса в данный момент времени. Нас же сейчас интересует первоначальный выбор на основании априорных данных, таких как тип и область действия адреса.

Первым делом пользователь сообщает приложению какой-то идентификатор хоста У, пригодный к машинной обработке. Сегодня в этой роли мы используем имена хостов, так что приложение на входе получит имя У, такое как *y.example.org*. Обратившись к DNS, приложение получит список адресов, отвечающих данному имени. Как мы уже говорили в §2.11, в общедоступных зонах DNS мы надеемся увидеть только глобальные адреса; между тем, DNS — не единственная база имен, наряду с ней применяют NIS, WINS и даже файл *HOSTS.TXT*… К чему мы ведем нашу мысль? В общем случае приложение получит *множество адресов* хоста У, а область действия некоторых из них будет меньше глобальной.

Со своей стороны, хосту Л тоже назначено некое множество адресов. Если во множестве «Л» всего N адресов, во множестве «У» всего M адресов, то можно составить $N \times M$ упорядоченных пар адресов, потенциальных кандидатов.

В некоторых случаях конкретную пару выбирает пользователь или какой-то внешний механизм, и тогда приложение посредством сетевого API просит стек хоста Л использовать именно эту пару адресов для связи с хостом У. Например, команда *ping6* позволяет указать оба адреса явным образом в командной строке:

```
ping6 -S FE80::1 2001:DB8::1
```

Но обычно полагают, что приложение и сетевой стек сами сделают оптимальный **выбор адресов по умолчанию**, сокращенно **DAS** (*default address selection*), для передачи данных между локальным и удаленным хостами [RFC 6724].

> На самом деле, эта аббревиатура «DAS» встречается в литературе редко, но мы используем ее, чтобы не говорить «ВАПУ». ☺

В IPv4 процедура выбора сводилась к тому, что приложение перебирало удаленные адреса по одному, пытаясь установить сеанс. Например, так себя ведет клиент *telnet*:

```
$ telnet host.example.org
```

```
Trying 192.0.2.23...
telnet: connect to address 192.0.2.23: Operation timed out
Trying 198.51.100.7...
telnet: connect to address 198.51.100.7: No route to host
Trying 203.0.113.5...
Connected to host.example.org.
Escape character is '^]'.

NetBSD/vax (host.example.org) (ttyp0)

login:
```

Сетевой стек, в свою очередь, подбирал локальный адрес для указанного приложением удаленного адреса. К примеру, стек *BSD Unix* находил по таблице маршрутов выходной интерфейс, а затем брал из списка адресов интерфейса самый первый [[125]]. Эта процедура продолжалась до тех пор, пока приложению не удавалось установить сеанс или пока оно не исчерпывало список удаленных адресов.

> У процедуры DAS в среде IPv4 есть и нормативные документы [§3.3.4.3 RFC 1122 и §2.3 RFC 1123]. Поведение стека BSD вполне отвечает им: приложение перебирает адреса назначения, а стек подбирает адрес источника, руководствуясь таблицей маршрутов и найденным с ее помощью выходным интерфейсом для данного пакета или соединения. Для соединений TCP это выполнялось единожды, перед открытием соединения, чтобы потом изменения маршрутов не вызвали смену локального адреса и, как следствие, сбой протокола.

Когда у каждого адреса появляется область действия, такая простая схема DAS может завести в тупик. Достаточно представить себе, что список адресов выходного интерфейса возглавляет внутриканальный адрес, назначенный раньше всех остальных. В этом случае хост Л сможет общаться только со своими соседями по каналу, даже если у него будет не только внутриканальный, но и глобальный адрес. Ведь в §2.4 мы сформулировали такое ключевое правило: чтобы пакет IPv6 достиг адресата, интерфейс назначения должен находиться в зоне адреса источника. Чтобы выполнить это правило, алгоритм DAS должен рассмотреть все локальные адреса и по очереди сопоставить каждый из них с удаленным адресом.

> Если никакой выбор адресов не способен удовлетворить это требование, то обмен данными между хостами Л и У попросту невозможен. Например, это произойдет, если у хоста Л есть только внутриканальные адреса, а все адреса хоста У находятся «вне канала» (см. §5.2). Однако в общем случае на стадии DAS нет сведений о том, где находится удаленный адрес, так как привести в действие механизм ND может только передача пакета, а она еще даже и не начиналась — источник только заполняет заголовок. Поэтому источнику доступны лишь априорные критерии выбора оптимальной пары адресов.
>
> Вторая половина того же правила из §2.4 гласит, что выходной интерфейс источника должен находится в зоне адреса назначения пакета. Так как из опубликованного в DNS адреса следует только его область, но не зона, источник вправе полагать, что этот адрес автоматически попадает в одну зону с выходным интерфейсом. Ведь мы постановили в §2.4, что сетевой интерфейс находится в одной зоне каждой возможной области! Например, если запрос DNS вернул внутрисайтовый адрес *FEC0::1*, то хост сделает вывод, что это его локальный сайт, а не сайт соседней организации. Поэтому надо быть вдвойне осторожным, публикуя адреса ограниченной области в DNS.

[125] G. R. Wright, W. R. Stevens. TCP/IP Illustrated, Volume 2: The Implementation. Addison Wesley, 1995. §22.8, “in_pcbconnect Function”.

Но какой именно критерий надо использовать при таком сопоставлении? Иными словами, какой из локальных адресов лучше всего подходит данному удаленному адресу? Чтобы критерий, который мы сейчас предложим, было проще алгоритмизировать, давайте сформулируем его в виде транзитивного отношения «лучше/хуже» между двумя потенциальными адресами источника, *SA* и *SB*, при данном адресе назначения *D*. Тогда список локальных адресов можно будет отсортировать по этому критерию, и наилучший адрес сам «всплывет» в начало списка.

> Конечно, в данном случае сортировку можно заменить обычным линейным поиском. Двоичный поиск этой задаче не подходит, так как порядок элементов зависит от переменного параметра *D*, а значит, список нельзя отсортировать раз и навсегда.

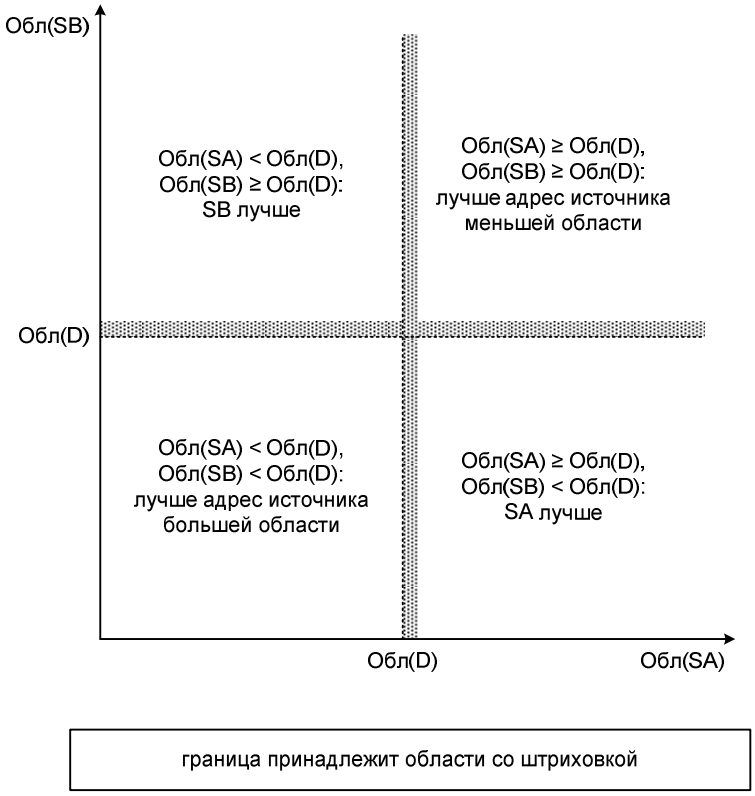
Первым мы рассмотрим случай, когда область адреса назначения Обл(*D*) — не больше,[126] чем области обоих адресов источника, Обл(*SA*) и Обл(*SB*): Обл(*D*) ≤ Обл(*SA*) ≤ Обл(*SB*) или Обл(*D*) ≤ Обл(*SB*) ≤ Обл(*SA*). Тогда шансы на успех передачи при любом выборе довольно велики. В этом случае имеет смысл выбрать адрес источника, область которого поменьше, а значит поближе к области адреса назначения. Благодаря этому, скажем, диалог с удаленным внутриканальным адресом будет вестись с локального внутриканального адреса. Ведь политика безопасности удаленного хоста может быть более либеральной к внутриканальным адресам, чем к глобальным адресам.

Когда область адреса назначения Обл(*D*) — промежуточной величины: Обл(*SA*) < Обл(*D*) ≤ Обл(*SB*) или Обл(*SB*) < Обл(*D*) ≤ Обл(*SA*), — то шансы на успех больше у адреса источника большей величины. И наоборот, у адреса источника меньшей величины они явно недостаточны, потому что его область строго меньше, чем область адреса назначения. (Случай, когда они равновелики, мы уже включили в предыдущий вариант.)

Наконец, возможна ситуация, когда у адреса назначения область строго больше, чем у потенциальных адресов источника: Обл(*SA*) ≤ Обл(*SB*) < Обл(*D*) или Обл(*SB*) ≤ Обл(*SA*) < Обл(*D*). Но даже в этом случае еще не все потеряно. Ведь может оказаться, что интерфейс назначения находится в зоне адреса источника, например, глобальный адрес назначения находится «на канале» (§5.2). Чтобы повысить вероятность благоприятного исхода, надо выбрать адрес источника, область которого больше, а значит, потенциально охватывает больше интерфейсов. Это лучшее, что может сделать источник пакета в таких условиях.

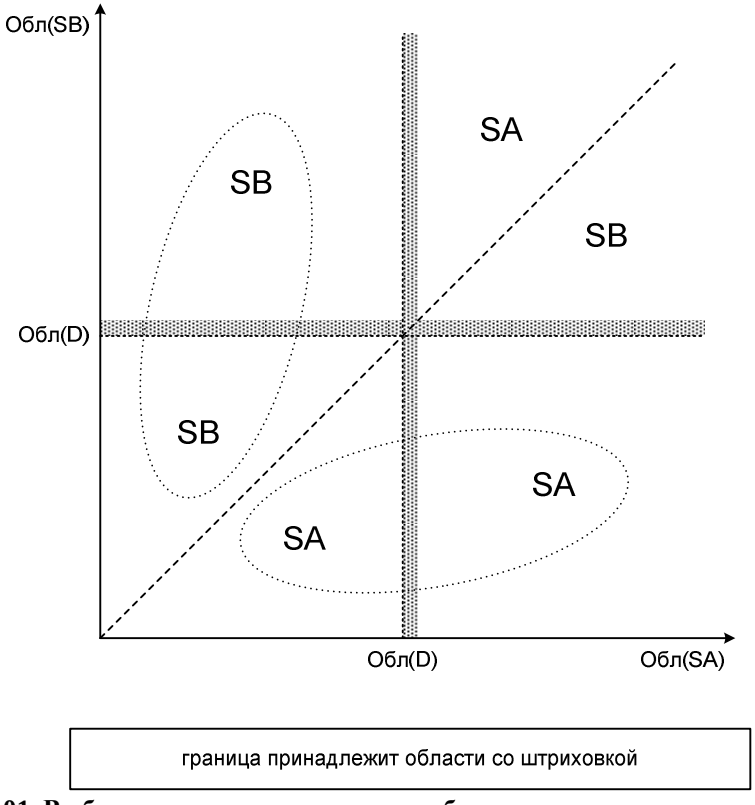
Чтобы лучше почувствовать эти правила, давайте изобразим их графически, отложив по одной оси область *SA*, а по другой — область *SB*, как показано на Фиг. 100. Тогда все пространство задачи можно разбить на четыре участка согласно нашим правилам выбора лучшего адреса источника. В двух из них выбор уже предопределен, а еще в двух зависит от отношения величин областей SA и SB между собой.

[126] То есть меньшей или равной величины.



Фиг. 100. Выбор адреса источника по его области — начальная схема

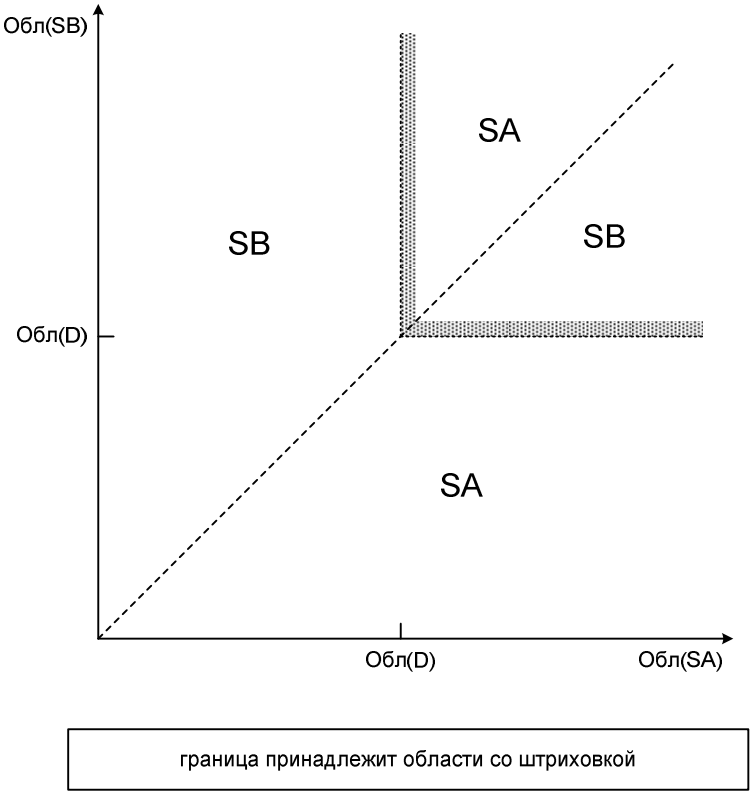
Если через начало координат этого графика провести прямую с угловым коэффициентом 1 (см. Фиг. 101), то она разобьет все пространство задачи на две половины так, что в верхней область *SA* будет меньше, чем область *SB*, а в нижней — наоборот. Фактически, эта прямая может провести для нас выбор лучшего адреса там, где он все еще условный. Поэтому мы можем пересмотреть нашу графическую схему так, что на каждом участке сразу указан окончательный выбор адреса источника.



Фиг. 101. Выбор адреса источника по его области — промежуточная схема

На первый взгляд у нас получилось шесть разных участков. Однако, присмотревшись внимательнее, мы заметим, что их можно свести к четырем (см. Фиг.

102), поскольку у двух участков *SA* есть общая граница, и то же самое справедливо для двух участков *SB*.



Фиг. 102. Выбор адреса источника по его области — окончательная схема

А уже руководствуясь последней схемой, нетрудно записать оптимальный алгоритм выбора [§5 RFC 6724]:

```
Если Обл(SA) < Обл(SB) то:
    Если Обл(SA) < Обл(D) то:
        Предпочесть SB.
    Иначе:
        Предпочесть SA.
Иначе Если Обл(SA) > Обл(SB) то:
    Если Обл(SB) < Обл(D) то:
        Предпочесть SA.
    Иначе:
        Предпочесть SB.
Иначе:
    Области SA и SB равны - ответ не определен.
```

Пусть читатель докажет, что такой критерий сравнения транзитивен: если *SA* лучше *SB*, а *SB* лучше *SC*, то *SA* лучше *SC*.

Как быть, если области *SA* и *SB* окажутся равны? В этом случае придется применить дополнительные критерии выбора. Так, один из них можно связать с жизненным циклом локального адреса IPv6 (Фиг. 83 на стр. 159). Такой адрес может устареть, а выбрать устаревший локальный адрес можно, только если нет равноценного ему предпочтительного адреса.

Еще один критерий мы найдем, если попытаемся ответить на вот какой вопрос. Мы обнаружили в §2.5, что для выполнения своей основной функции, то есть продвижения транзитных пакетов, маршрутизатору достаточно внутриканальных адресов. Действительно, если каждый активный сетевой интерфейс маршрутизатора получит внутриканальный адрес, то его соседи по каналу смогут ссылаться на него как следующий шаг. Кроме того, маршрутизатор сможет участвовать на этом канале в розыске соседей и

маршрутизаторов (§5), а также играть роль генератора запросов MLD (§6.4), потому что при участии в работе этих протоколов маршрутизатор не просто может, а даже обязан слать свои сообщения с внутриканального адреса.

Однако есть как минимум один случай, когда маршрутизатору может понадобиться адрес большей области действия. Представьте себе, что маршрутизатор попытался продвинуть транзитный пакет в канал, MTU которого меньше длины пакета. В этом случае пакет пропадает, а маршрутизатору надо выслать извещение «пакет слишком велик» в адрес источника пакета. Вообще говоря, область этого адреса может быть любой, в том числе глобальной.

Нетрудно видеть, к чему мы клоним. Даже маршрутизатору иногда приходится выполнять функции хоста, когда он создает пакет с извещением ICMPv6. Высылая такое извещение, он направляет его по адресу источника пакета-виновника. Чтобы такое извещение дошло, его собственный адрес источника должен быть подходящей области действия. Например, если пакет-виновник был послан с глобального адреса, то извещению в общем случае тоже необходим глобальный адрес источника. Ведь в противном случае нет никаких гарантий, что извещение достигнет адресата.

Откуда маршрутизатор возьмет глобальный адрес? Придется ли назначить один глобальный адрес на каждый интерфейс маршрутизатора? На самом деле, будет достаточно ровно одного адреса на весь маршрутизатор, например, на интерфейсе «петля», если маршрутизатор сможет выбрать адрес источника из всех своих адресов, а не только из адресов выходного интерфейса. Ведь тогда главный критерий выбора, основанный на сравнении областей, автоматически предпочтет глобальный адрес источника.

Обычному хосту, у которого есть несколько сетевых интерфейсов, тоже может пригодиться эта возможность выбирать из всех своих адресов, хотя ему рекомендовано ограничиться адресами выходного интерфейса [§4 RFC 6724].

> В этом разделе мы говорим «хост», а не «узел», чтобы подчеркнуть, что речь идет об одной из основных функций хоста IP — создавать новые пакеты на выходе. Вторая его функция — это потреблять адресованные ему пакеты на входе. В то же время, для маршрутизатора это нетипичные, дополнительные функции. Главная функция маршрутизатора — продвигать транзитные пакеты. Только особенности управления IP заставляют маршрутизатор заниматься иногда делом, свойственным хосту.

С другой стороны, пока сам выходной интерфейс обладает подходящим адресом, следует предпочесть его. Это необходимо, в первую очередь, для правильной работы механизма переадресовок из §5.2. Ведь маршрутизатор вернет переадресовку, только если адрес источника в заголовке пакета указывает на соседа по каналу. Поэтому пусть это будет дополнительным критерием: если один адрес-кандидат принадлежит выходному интерфейсу, а другой нет, то побеждает первый из них.

Также нам следует учесть, что расширения для конфиденциальности из §6.2 делят локальные адреса хоста на два класса: публичные и временные. Эти расширения по умолчанию должны быть выключены, так что их включение на хосте однозначно говорит о желании их активно использовать. Поэтому *по умолчанию* временный адрес выигрывает у публичного. Тем не менее, у приложения должна быть возможность запросить выбор публичного адреса посредством сетевого API.

Наконец, если иерархии распределения адресов и маршрутизации IPv6 совпадают, то можно приближенно судить об удаленности двух интерфейсов друг от друга по длине общего префикса их адресов в одной зоне: чем длиннее общий префикс, тем, вероятно, короче трасса между интерфейсами. Поэтому есть определенный резон в том, чтобы предпочесть из двух кандидатов *SA* и *SB* тот, у которого длиннее общий префикс с адресом назначения *D*, по крайней мере, если более сильные критерии выбора дают

«ничью». Например, если адрес *SB* в одной подсети с *D*, а адрес *SA* нет, то предпочесть стоит *SB*.

> Такое сравнение следует ограничить префиксом подсети локального адреса, потому как сравнение идентификаторов интерфейсов не даст требуемой информации. Грубо говоря, если первые 64 бита у сравниваемых адресов оказались равны, то сравнивать последующие биты не имеет смысла.

Именно эти критерии и именно в таком порядке составляют костяк процедуры выбора адреса источника [§5 RFC 6724]. Кратко перечислим их еще раз:

1) сопоставление областей;
2) предпочтительный адрес против устаревшего;
3) адрес выходного интерфейса пакета против прочих адресов;
4) временный адрес против публичного;
5) сравнение длины префиксов, общих с адресом назначения.

Данная процедура возвращает самый подходящий из локальных адресов хоста, но только для заданного адреса назначения *D*. Обозначим ее как Ист(*D*) — источник для *D*.

> Полный список критериев [§5 RFC 6724] длиннее. Прежде всего, он предпочитает адрес источника, равный адресу назначения, в угоду локальной передаче пакетов в пределах хоста. Ведь равенство адресов источника и назначения — самый простой и надежный признак того, что пакет локальный. Прочие критерии помогают работе механизма мобильности узлов, а также позволяют вручную подстраивать DAS с помощью меток и численных весов. Все они, безусловно, важны на практике, но не несут фундаментального значения, и поэтому мы их подробно не рассматриваем. Читатель без труда разберется в них самостоятельно.

Теперь нам пора вспомнить, что хост также располагает множеством потенциальных адресов назначения. Среди них нельзя выбрать ровно один, самый лучший, потому заранее неизвестно, по какому адресу в итоге ответит удаленная сторона. Хост обязан перебирать их до тех пор, пока попытка связи не приведет к успеху. Тем не менее, адреса назначения тоже не помешает упорядочить так, чтобы наиболее вероятные претенденты на успех оказались в начале списка.

Значит, нам надо также сформулировать транзитивные критерии сравнения двух адресов назначения, *DA* и *DB*. Сделать это можно, опираясь на уже проверенные нами приемы, потому что удаленные адреса все равно видны хосту через призму его собственных сетевых интерфейсов и локально назначенных адресов. Неудивительно, что процедура выбора адреса назначения активно пользуется процедурой Ист(*D*) и сравнивает характеристики адресов источника, чтобы принять решение насчет потенциальных адресов назначения [§6 RFC 6724].

Прежде всего, поддержим зонную архитектуру IPv6 таким критерием: выигрывает адрес назначения, адрес источника для которого имеет ту же область. Например, если Обл(*DB*) = Обл(Ист(*DB*)), а Обл(*DA*) ≠ Обл(Ист(*DA*)), то выигрывает *DB*. Безусловно, это не окончательный критерий, так как области могут совпадать, или не совпадать, у обоих кандидатов.

Если предыдущий критерий дал ничью, то проигрывает адрес назначения, адрес источника для которого устарел. Например, если Ист(*DB*) устарел, а Ист(*DA*) нет, то выигрывает *DA*. Конечно, и здесь может быть ничья, если оба адреса источника находятся на одном этапе жизненного цикла.

Следующим снова вступает в игру критерий, связанный с областями адресов. Область адреса назначения косвенно говорит о том, насколько длинен и тернист путь к сетевому интерфейсу назначения. Например, если адрес назначения — внутриканальный, то до соответствующего интерфейса всего один шаг, а если он — внутрисайтовый, то трасса до него, по крайней мере, заключена в пределах одной административной зоны, а значит, меньше вероятность сбоя из-за несогласованной политики на границах сетей.

Поэтому, при прочих равных, есть смысл предпочесть адрес назначения, область которого меньшей величины.

Если остальные критерии вернули ничью, то остается только предпочесть адрес назначения, у которого длиннее общий префикс с отвечающим ему адресом источника.

> И снова мы перечислили не все критерии выбора [§6 RFC 6724], обратив внимание только на самые основные из них.

Нет ли в «созданной» нами процедуре DAS скрытых противоречий? Одна ее деталь, которая вызовет подозрение у любого инженера, знакомого с IPv4, — это использование информации о выходном интерфейсе пакета при выборе адреса источника, то есть до того, как окончательно установлен адрес назначения. Ведь реалии IPv4 таковы, что выбор выходного интерфейса (маршрутизация) проходит на основании адреса назначения пакета.

Конечно, эту неувязку можно обойти, рассматривая выходной интерфейс как функцию от адреса назначения *D*, который процедура Ист(*D*) получает на вход в виде аргумента. В реалиях IPv4 это просто означало бы, что хост использует таблицу маршрутов, чтобы найти выходной интерфейс по адресу назначения.

Тем не менее, в архитектуру IPv6 заложен иной подход, который у нас постепенно выкристаллизовывается. Путь к нему мы начали, работая над ND и SLAAC, когда установили, что все структуры хоста, включая список маршрутизаторов по умолчанию, привязаны к определенному сетевому интерфейсу, а выбор интерфейса из нескольких доступных вариантов находится за пределами задач ND (см. §5.2). Окончательного решения мы достигнем в §6.8, а сейчас только сделаем очередной необходимый шаг по направлению к нему.

В §5.2 мы подробно обсудили, что модель хоста IPv6 существенно отличается от того, как устроены реализации хостов и маршрутизаторов IPv4. В то же время, у нас до сих пор нет модели маршрутизатора IPv6. Давайте допустим, что маршрутизатор IPv6 не столь существенно отличается от своего аналога IPv4 и его работа основана на такой же таблице маршрутов, ставящей в соответствие удаленным префиксам адреса следующего шага.

Пока что единственный случай, когда процедура DAS нужна маршрутизатору IPv6, — это отправка извещения ICMPv6. В этом случае маршрутизатор IPv6 не встретит препятствий, потому что адрес назначения извещения однозначно известен: это адрес источника в пакете-виновнике, вызвавшем извещение. Поэтому маршрутизатор вполне способен определить выходной интерфейс извещения, руководствуясь своей таблицей маршрутов, и только затем провести выбор адреса источника.

> На зону, в которой находится адресат извещения, указывает входной интерфейс пакета-виновника. Если физический маршрутизатор состоит из нескольких виртуальных маршрутизаторов, то этот интерфейс также указывает на тот виртуальный маршрутизатор, чьей таблицей маршрутов надо воспользоваться при отправке извещения ICMPv6.

Таким образом, случай многих адресов назначения на стадии DAS возникает только в работе хостов IPv6, и нам достаточно сосредоточить внимание на них. Как это ни странно звучит для эксперта по IPv4, но та архитектура хоста IPv6, к которой мы постепенно движемся, вполне допускает, что выбор выходного интерфейса происходит до маршрутизации пакета [§7 RFC 6724]. Например, приложение может само выбрать один из сетевых интерфейсов хоста и вести все свои сетевые диалоги строго через него.

> Обратите внимание, что процедура DAS сработает и в том случае, когда удаленные адреса-кандидаты — групповые, а не индивидуальные. Конечно, в этом случае пробное множество локальных адресов включает в себя только адреса, назначенные выходному интерфейсу, или другому интерфейсу в тот же самый канал, чтобы удовлетворить требованиям групповой маршрутизации [§4 RFC 6724].

Безусловно, у приложения должна быть возможность управлять аспектами DAS, скажем, предпочесть публичные адреса временным. Для этого даже существует эталонный API [RFC 5014].

Мы сознательно обошли вниманием тот случай, список удаленных адресов смешанный и содержит как адреса IPv6, так и адреса IPv4. Обсуждение механизмов перехода между IPv4 и IPv6 — очень важная тема, но она выходит за рамки нашего курса. Кроме того, для полноценного знакомства с ней не помешает сперва разобраться, как работает IPv6 в чистом виде. Тем не менее, полезно иметь в виду, что механизм DAS готов справиться и со смешанным случаем.

Механизм DAS — это по-прежнему объект исследования, поскольку есть мнение, что в сложных сценариях он не всегда дает удовлетворительный результат [RFC 5220, RFC 5221].

В завершение раздела соберем вместе и приведем все адреса, которые хост IPv6 обязан полагать своими собственными [§2.8 RFC 4291]:

- адрес обратной связи *::1* (§2.3);
- по одному обязательному внутриканальному адресу на каждый интерфейс (§2.5);
- дополнительные индивидуальные адреса (§2.6) и адреса anycast (§5.3), настроенные на интерфейсах хоста вручную (§5.4.1) или автоматически (§5.4.2);
- общепринятые групповые адреса «все узлы» (§2.8);
- групповой адрес искомого узла для каждого назначенного хосту индивидуального адреса или адреса anycast (§5.1);
- адреса всех групп, в которых хост на данный момент состоит (§2.8).

А маршрутизатор IPv6, помимо всех предписанных хосту адресов, обязан считать своими собственными еще и такие:

- адреса anycast «маршрутизатор подсети» на всех интерфейсах, где включена функция маршрутизатора (§6.1);
- общепринятые групповые адреса «все маршрутизаторы» (§2.8).

6.7. Разрешение доменных имен в среде IPv6

Предположим, серверы и клиенты DNS учли изменения, предложенные в §2.11. После этого приложение, обращаясь к DNS посредством сетевой библиотеки, может встречаться не только с адресами IPv4, но также с адресами IPv6. Более того, в переходный период вероятен «винегрет» из адресов, когда одному имени сопоставлены адреса обоих семейств. Например:

```
www.example.org.     IN   A        192.0.2.77
                          AAAA     2001:db8:c001::beef
```

Правильная работа с адресами из разных семейств пусть будет на совести программистов. Во-первых, они должны избавиться от ложного убеждения, будто все сетевые адреса — IPv4. Во-вторых, им следует принять во внимание правила выбора адресов, к которым мы пришли §6.6. Мы же сейчас позаботимся, чтобы приложение получило по своему запросу полный список адресов, отвечающих данному доменному имени.

Между клиентом DNS в составе сетевой библиотеки и приложением обычно находится дополнительный уровень абстракции, который позволяет обращаться посредством одного простого API к разным системам разрешения имен: *HOSTS.TXT*, DNS, NIS и т.п. Скажем, в классическом API *BSD Unix* приложения использовали функцию *gethostbyname()*. Оттуда эта функция проникла в стандарт *POSIX* и в API *Windows Sockets*. К сожалению, функция *gethostbyname()* не готова к работе со смешанными семействами сетевых адресов: она может вернуть список адресов только из одного семейства. По

умолчанию она возвращает адреса IPv4, а способ переключить ее на IPv6 системно-зависим.

> Например, реализация *gethostbyname()*, популяризованная пакетом *BIND*, возвращает адреса IPv6, если была выбрана опция **RES_USE_INET6** библиотеки *resolver*. Эту опцию можно указать под ее текстовым именем `inet6` в файле *resolv.conf* или переменной окружения **RES_OPTIONS**.

По всей видимости, нужны новые функции для доступа к системам разрешения имен. Как показывает опыт, пересматривать общепринятый API еще труднее и опаснее, чем модифицировать сетевой протокол. Но раз уж нам придется это сделать, новые функции должны быть максимально гибкими и универсальными. По этой причине мы не можем просто добавить к уже существующей поддержке IPv4 поддержку IPv6.

Новый API обязан одновременно работать с любыми семействами адресов. С точки зрения программиста, добиться этого несложно: достаточно снабдить каждый адрес обязательным атрибутом, его семейством. Для этого можно, к примеру, хранить двоичное представление адреса не само по себе, а в составе структуры, где еще одно поле указывает семейство данного адреса.

> Этот подход хорошо знаком тем, кто использовал API сокетов Беркли.

Именно так поступает новая функция *getaddrinfo()* [§6.1 RFC 3493]. Значение поля «семейство адреса», возвращаемое этой функцией, можно прозрачно передавать функциям из API сокетов Беркли [RFC 3493], так что приложениям не нужно обрабатывать каждое известное им семейство отдельно. Теперь правильно написанное приложение не потребует никаких изменений после появления следующей версии IP или даже перехода на новый стек протоколов с похожими свойствами.

Каковы будут отношения нового API и зонной адресной архитектуры IPv6? Если узел использует не DNS, а другую систему разрешения имен, которая каким-то образом позволяет ему получать правильные идентификаторы зон *zone_id* в адресах, то функция *getaddrinfo()* готова дать ему доступ к этой информации. Она возвращает адреса IPv6 в виде структур *sockaddr_in6*, где есть поле идентификатора зоны.

Если же адрес IPv6 получен из записи **AAAA**, опубликованной в DNS, то доступно только его численное значение, не уточненное идентификатором зоны *zone_id*. Причины этого были изложены в §2.11. Тем не менее, отсутствие *zone_id* в записи **AAAA** еще не значит, что такая запись не может содержать в правой части адрес IPv6 с ограниченной областью действия. К примеру, если организация использует внутрисайтовые адреса, то все они заведомо принадлежат одной зоне — внутренней сети организации. С одной стороны, такие адреса вполне можно поместить в частную зону DNS этой организации. С другой стороны, они не имеют смысла вне данной организации.

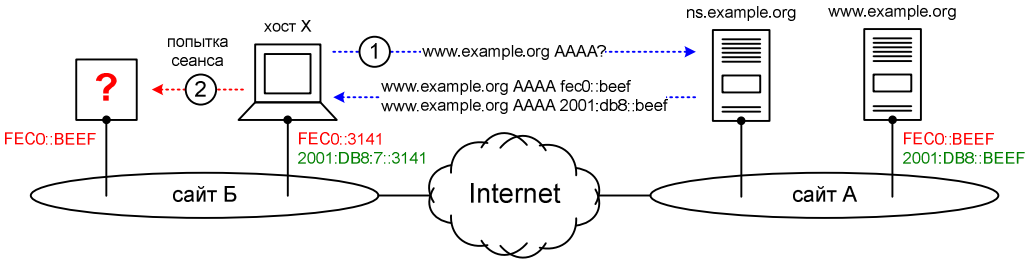
Утечка записей с внутрисайтовыми адресами в *Internet* может иметь неприятные последствия. Рассмотрим следующий пример. Организация А поместила в свою публичную зону DNS такие записи:

```
www.example.org.     IN   AAAA      2001:db8::beef
                          AAAA      fec0::beef
```

Организация Б тоже использует внутрисайтовые адреса. Ее хост X собирается обратиться к *www.example.org*. У хоста X есть внутрисайтовый адрес, и по правилам выбора (§6.6) хост X предпочтет удаленный адрес *FEC0::BEEF*, потому что у него наименьшая область. Но, с точки зрения хоста X, зона адреса *FEC0::BEEF* — сайт Б, а не сайт А! Когда хост X попытается установить сеанс по этому адресу (см. Фиг. 103), возможны три основных исхода, один другого хуже:

- хост X не сможет установить сеанс, потому что в сети Б нет узла с адресом *FEC0::BEEF*; этот исход не фатальный, так как хост X перейдет к следующему адресу *www.example.org*, *2001:DB8::BEEF*, по тем же правилам выбора;

- хост X установит сеанс с легитимным узлом сети Б, чей адрес — *FEC0::BEEF*; результат будет непредсказуемым;
- злоумышленник, забравшийся в сеть Б, присвоит себе свободный адрес *FEC0::BEEF* и без труда перехватит сеанс; вдобавок, хосты могут применять разные политики безопасности к сеансам внутри и вне сайта; например, хост X может отказаться от аутентификации и шифрования сеанса, полагая, что ведет диалог с доверенным узлом; в результате безопасность пострадает еще сильнее.



Фиг. 103. Последствия необдуманной публикации внутрисайтового адреса в DNS

Выводы напрашиваются сами собой:

- в общедоступной зоне DNS следует регистрировать только глобальные адреса IPv6;
- клиент DNS не должен возвращать приложению запись из общедоступной зоны DNS, если она содержит адрес IPv6 с ограниченной областью.

Выполнить второе правило в общем случае непросто, так как клиент DNS должен правильно классифицировать зоны на доверенные и общедоступные. Однако в типовой частной сети эту задачу можно переложить на рекурсивный сервер DNS. Такой сервер, обрабатывая запросы конечных клиентов, мог бы фильтровать записи в ответах других серверов; в то же время, собственные авторитетные ответы он фильтрации подвергать не будет. Конечно, в этом случае надо запретить прямое обращение клиентов к внешним серверам DNS, например, на уровне политики безопасности сети.

6.8. Работа хоста с несколькими подключениями

Осторожно: вы вступаете в область исследования и экспериментов. Данный аспект работы IPv6 еще находится на стадии развития.

Мы завершим нашу умозрительную работу над основами IPv6, рассмотрев такой простой вопрос: каким образом хост IPv6 с несколькими сетевыми подключениями сможет эффективно использовать их? Ведь до сих пор мы старательно закладывали фундамент для ответа на этот вопрос, но от самого ответа столь же старательно уходили.

Чтобы в полной мере оценить значение этого вопроса, давайте мысленно вернемся во времена господства IPv4 и посмотрим, как тогда обеспечивали отказоустойчивое подключение отдельной сети к *Internet*.

Мы, конечно же, помним, что *Internet* — это «сеть сетей».

Главным рецептом тогда, как и теперь, было дублирование и резервирование элементов, в данном случае, подключений к другим сетям. Проще говоря, сети было необходимо несколько подключений, желательно, к разным партнерам. Однако, чтобы эти подключения на самом деле работали, надо было каким-то образом донести информацию об адресном пространстве сети в глобальное облако маршрутизации *Internet* через каждое из подключений.

Как нетрудно убедиться, в модели маршрутизации IP прикладной трафик распространяется навстречу маршрутной информации.

Если сеть принадлежала достаточно крупному провайдеру, у которого были статус LIR, автономная система и прочие атрибуты серьезного игрока в этом бизнесе, то сеть

объявляла о себе с помощью BGP, и все образовывалось как бы само собой. Это вполне устоявшаяся схема, и LIR IPv6 сообщают о себе точно так же при помощи MBGP.

Но что если сеть принадлежала абстрактной организации, применявшей свои подключения к *Internet* для сугубо внутренних целей? Пока адреса IPv4 еще были относительно доступны, такая организация могла получить прямиком из рук RIR автономную систему и отдельный блок адресов, не привязанный к определенному провайдеру,[127] после чего подключиться к нескольким провайдерам и, по взаимному с ними соглашению, объявить о себе по BGP. Альтернативой этому было получение блока адресов из пространства провайдера №1 и анонсирование этого блока, по соглашению с провайдером №1, прочими провайдерами той же организации. Независимо от выбранного пути, глобальным следствием этой практики было дробление адресного пространства и рост числа маршрутов в глобальном облаке *Internet*. Ведь каждая такая сеть была представлена, как минимум, одним маршрутом, а число независимых сетей может на порядки превышать число провайдеров. Неудивительно, что независимые сети попали в немилость современных политик распределения адресов IPv6, принятых RIR.

Самая «интересная» участь ожидала те сети, которые не были достаточно велики или весомы, чтобы оказаться представленными в глобальном облаке маршрутизации *Internet*. Им не оставалось ничего иного, кроме как подключиться к нескольким провайдерам, получить от каждого из них по блоку адресов и после этого заняться сооружением пирамиды из различных ухищрений, чтобы обеспечить работу сети по нескольким подключениям. Вот лишь краткий и неполный список препятствий, которые им приходилось преодолевать:

- Один провайдер вряд ли пропустит от клиента транзитный трафик, адрес источника в котором принадлежит другому провайдеру. Это вполне обоснованное ограничение в рамках борьбы с подделкой адресной информации.
- Отказоустойчивость на входе, например, для публично доступного сервера внутри сети, можно обеспечить, назначив ему и опубликовав в DNS адреса от разных провайдеров. Однако тогда надо следить, чтобы ответный трафик возвращался в соответствующее подключение — иначе см. первый пункт. Так как обычная маршрутизация не учитывает адрес источника пакета, приходится использовать нестандартные трюки, например, маршрутизацию согласно политике.
- Отказоустойчивость на выходе, чтобы пользователи сети сохранили доступ к *Internet* при отказе одного из подключений, возможна только путем сочетания NAT и каких-то доморощенных механизмов наблюдения за состоянием подключений в стиле: «*Ping* на *Google* не прошел, значит, провайдер «упал». Переключаем *default route*…» Ведь у каждого пользовательского хоста IPv4 всего один адрес.

Но даже эти ухищрения обладали весьма ограниченными возможностями. В частности, никакими трюками нельзя было сохранить обычное соединение TCP при переключении между провайдерами, потому что менялся локальный адрес сокета. В результате, чем меньше была сеть, тем трудней ей давалось надежное подключение к *Internet*.

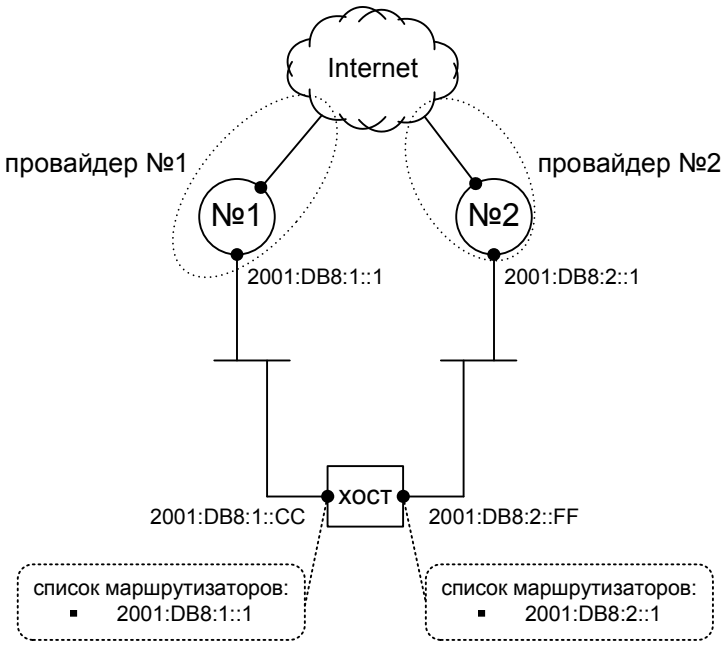
Поэтому сейчас, когда пришло время перемен, стоит перенести фокус с масштабных механизмов отказоустойчивости, таких как BGP, на средства, доступные индивидуальному пользователю. Иными словами, мы хотим осчастливить все сети независимо от их размера, а самая маленькая сеть — это один хост. Именно на хосте IPv6 мы и заострим свое внимание.

[127] Известный как **PI** (*Provider Independent*).

Поставим себя на место владельца хоста IPv6, которому необходимо надежное подключение к *Internet*, работоспособность которого не зависит от одного провайдера. Как бы мы поступили в этой ситуации?

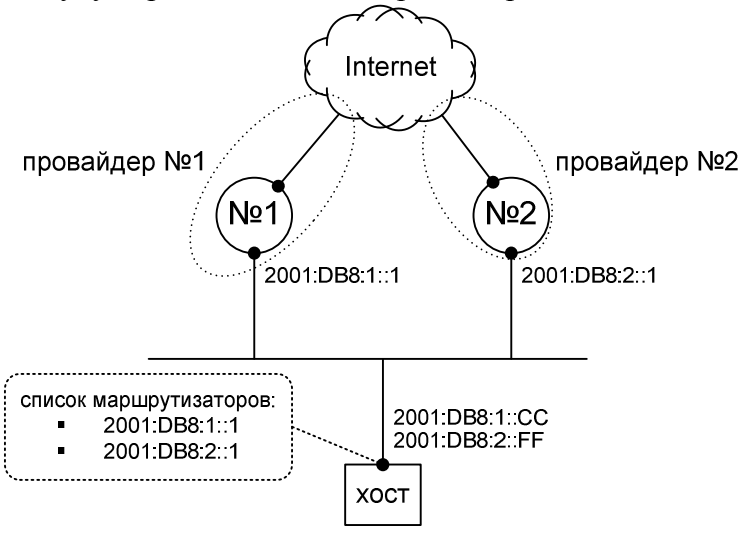
Первый вариант (Фиг. 104) состоит в том, чтобы установить два (или больше) сетевых интерфейса и одновременно подключиться к двум (или больше) разным провайдерам (в надежде, что они независимы). При этом каждый сетевой интерфейс получит адрес из блока соответствующего провайдера. Эталонный хост IPv6 на базовом уровне уже готов к работе в такой конфигурации, потому что с каждым его интерфейсом связан отдельный список маршрутизаторов по умолчанию.

> Можно сказать, что эталонный хост IPv6 с самого начала поддерживает виртуализацию сетевого стека.



Фиг. 104. Множественное подключение по разным каналам

Второй вариант (Фиг. 105) возможен, если есть канал, на котором уже присутствуют маршрутизаторы нескольких провайдеров (или маршрутизаторы, подключенные к разным провайдерам). Тогда достаточно подключить хост к одному каналу, но назначить его сетевому интерфейсу несколько адресов, по одному от каждого провайдера. В этом случае у хоста будет всего один список маршрутизаторов по умолчанию, но в нем будут представлены все провайдеры.



Фиг. 105. Множественное подключение по одному каналу

На самом деле, это только первый шаг к отказоустойчивому подключению. В обоих вариантах он известен как **множественное подключение** (*multihoming*). Но что он дает хосту? Непосредственно хост может обнаружить только сбой канала или маршрутизатора, используя механизм NUD (§5.1). Что ж, и это неплохо, так как хост может переключиться на другого провайдера, используя стандартные механизмы.

Если же сбой будет где-то дальше в сети провайдера, то явного указания на это хост не получит. Что же произойдет в этом случае? Рассмотрим для примера два сценария, в которых хост принимает соединения TCP извне или же устанавливает их с внешними хостами сам.

Допустим, что в первом сценарии все адреса хоста опубликованы в DNS под одним именем. Тогда удаленное приложение, получив на входе имя, станет пробовать эти адреса по очереди, в порядке, предписанном DAS (§6.6). Так как трасса к разным адресам назначения будет проходить через разных провайдеров ввиду самого устройства маршрутизации IP, рано или поздно удаленная сторона выберет доступный адрес назначения, и первый пакет <SYN> дойдет до хоста, скажем, через провайдера №2. Теперь хосту надо позаботиться, чтобы ответный пакет <SYN,ACK> ушел через того же провайдера №2, потому что именно он работоспособен и, вероятно, готов пропустить такой пакет. Эта задача сводится к выбору выходного интерфейса (вариант Фиг. 104) или маршрутизатора (вариант Фиг. 105). В первом случае решение таково: сокет IPv6 должен быть привязан к определенному сетевому интерфейсу хоста. Например, в данном случае пакет <SYN> придет по адресу, который назначен интерфейсу №2, а значит, и ответный пакет <SYN,ACK> надо направить в тот же интерфейс. Во втором случае хосту понадобится дополнительное правило при выборе маршрутизатора, чтобы пакет с адресом источника от провайдера №2 ушел именно через маршрутизатор №2. Благодаря записи в кэше адресатов DC, которая возникнет в результате этого выбора, последующие пакеты данного соединения пойдут через маршрутизатор №2 естественным порядком.

> В нормативных документах IPv6 такого правила пока что нет, хотя оно обсуждалось [draft-huitema-multi6-hosts, [128]].

> Если один и тот же удаленный адрес одновременно установит соединения с разными адресами локального хоста через разных провайдеров, эта схема даст сбой. Чтобы восстановить ее работоспособность, запись DC должна включать в себя адрес источника, чтобы разным локальным адресам отвечали разные записи DC даже при одном и том же удаленном адресе назначения.

Перейдем теперь ко второму сценарию, когда хост сам хочет установить соединение TCP с удаленной стороной, послав для начала пакет <SYN>. Конечно, может оказаться, что у удаленного хоста тоже есть несколько адресов и только некоторые из них недоступны через текущего провайдера. Тогда поможет обычный перебор удаленных адресов. Но как быть, если произошел полный сбой текущего провайдера? Как мы уже сказали, хост может обнаружить его только по косвенным признакам. Здесь одна из возможных линий поведения хоста — попробовать все свои выходные интерфейсы (вариант Фиг. 104) или все маршрутизаторы (вариант Фиг. 105) по очереди. В первом случае процедура DAS выберет правильный адрес источника для пакета <SYN>. Во втором случае надо, чтобы выбор маршрутизатора влиял на выбор адреса источника, а это вполне допустимое расширение процедуры DAS [§5 и §7 RFC 6724].

Таким образом, множественное подключение хоста IPv6 можно использовать, практически не выходя за рамки стандартных механизмов. Тем не менее, мы так и не обеспечили одной простой возможности: сохранить уже установленное соединение TCP при переключении хоста между провайдерами. Как нетрудно убедиться, такое

[128] Kenji Ohira. IPv6 Address Assignment and Route Selection for End-to-End Multihoming. IETF Meeting №57. <http://ops.ietf.org/multi6/ietf57/day1/IETF-57-ohira-multi6.ppt>.

переключение по-прежнему вызывает смену локального адреса, и соединение TCP рвется. С этим нельзя ничего поделать, по крайней мере, на уровне IP.

От этой проблемы проще всего отмахнуться, сказав, что прикладные протоколы должны быть устойчивы к сбоям на транспортном уровне. Несомненно, это так. Скажем, протоколы FTP и HTTP поддерживают «докачку», и эта возможность не чувствительна к адресам IP: клиент просто откроет новый сеанс и снова запросит тот же файл или объект, начиная с энного байта.

Однако это скучный подход, который не делает чести нашему воображению. Давайте лучше применим фантазию и в общих чертах набросаем решение поинтереснее. Прийти к нему несложно. Для этого представим себе, что приложение открывает одно соединение TCP сразу с множеством адресов удаленной стороны. Действительно, зачем перебирать их по одному, если можно было бы послать по пакету <SYN> сразу им всем. Тогда и локальный конец соединения мог бы иметь не один адрес, а целое их множество.

«В лоб» такое решение можно сымитировать, открыв $N \times M$ соединений TCP, где N — число локальных адресов, а M — число удаленных адресов. Однако это непрактичный подход, так как, во-первых, число таких соединений может быть велико, а во-вторых, ими надо дополнительно управлять, чтобы сохранить порядок байтов в потоке. Ведь получателю надо как-то узнать, в каком порядке были посланы байты, пришедшие по разным соединениям.

Поэтому нам понадобится более целостное решение. Так, TCP можно заменить новым транспортным протоколом, который с самого начала станет поддерживать больше одного адреса на каждом из концов соединения. Конечно, пакет IPv6 по-прежнему содержит только один адрес источника и один адрес назначения, так что вся работа по слежению за доступностью адресов и переключению между ними ляжет на этот новый транспортный протокол. На самом деле, такой протокол уже создан: SCTP. Прелесть этого решения в том, что оно сквозное: хостам безразлично, где именно в сети произошел сбой, и пока существует хотя бы одна пара адресов, межу которой пакеты все еще ходят, транспортное соединение будет жить.

Однако новый транспортный протокол — это довольное радикальное новшество, которое подходит только для новых приложений. Кроме того, с новым транспортным протоколом связана та трудность, что всевозможные межсетевые экраны, системы предотвращения вторжений и прочие «ящики» (*middleboxes*), ведущие глубокий анализ транзитного трафика, уже хорошо знакомы с TCP, но новый протокол наверняка вызовет проблемы. Поэтому нельзя ли перенести все те же идеи обратно в старый добрый TCP, чтобы и существующие приложения выиграли от новой архитектуры множественных подключений?

В основу такого «многопутевого TCP» (*multipath TCP*, **MPTCP**) следует заложить принципы совместимости с приложениями и сетью [RFC 6182]. Что это значит? С одной стороны, сетевой API, например, сокеты Беркли, должен остаться, по возможности, без изменений, чтобы уже существующие приложения не пришлось даже перекомпилировать, не говоря уже о том, чтобы их переписывать. А с другой стороны, для стороннего наблюдателя в сети каждый путь MPTCP (известный как под-поток, *sub-flow*) должен выглядеть как обычное соединение TCP. Это нужно, чтобы MPTCP беспрепятственно проходил сквозь различные «ящики».

> В частности, у каждого под-потока своя, независимая от других под-потоков и непрерывная, как и положено в TCP, нумерация байтов, то есть порядковые номера и номера квитанций. Если бы нумерация байтов была одна на все под-потоки, протокол стал бы проще, но достаточно строгий «ящик» вполне мог бы блокировать под-поток с «прорехами» в нумерации байтов, потому что тот нарушает протокол TCP. Соответствие между номерами байтов в под-потоке и всем соединении MPTCP задает та дополнительная информация, которую несут прозрачные для «ящиков» опции MPTCP.

Фактически мы вернулись к идее множественных соединений TCP между разными парами локальных и удаленных адресов, но под управлением четко заданного механизма. Чтобы работать по-настоящему эффективно, MPTCP должен управлять не только взаимной доступностью разных пар адресов, но и перегрузкой вдоль разных путей, перенося трафик на те пути, где перегрузка меньше [RFC 6356]. Для переноса служебной информации MPTCP вполне подходят опции TCP, которые активно используются и для других целей, а потому без труда преодолевают «ящики» [draft-ietf-mptcp-multiaddressed]. Благодаря дополнительной информации, MPTCP вновь собирает исходный поток байтов из нескольких под-потоков, а потому он способен вести одновременную передачу данных вдоль нескольких путей.

> Конечно, MPTCP не зависит от особенностей IPv6. Более того, он позволяет соединению TCP одновременно использовать пути IPv4 и IPv6.

А как насчет других протоколов, уже работающих непосредственно поверх IPv6? В их случае можно испытать иной подход. Представим себе, что между всеми этими протоколами и уровнем IPv6 появляется «прокладка», которая манипулирует адресами в пакетах, так что вышестоящий протокол всегда имеет дело с одной парой адресов (локальный и удаленный), как и раньше. Этой «прокладке», известной как **Shim6** (Прокладка №6) [RFC 5533, [129]], тогда придется динамически транслировать неизменные адреса «сверху» в зависящие от текущей обстановки в сети адреса «снизу» и наоборот. Эта трансляция превращает адреса, видные вышестоящим протоколам, в идентификаторы, не зависящие от сетевой топологии, тогда как адреса, действительно попадающие в сеть, становятся локаторами, чувствительными к сетевой обстановке. Поэтому первые из них получили название «идентификаторы вышележащего уровня» (*Upper-Layer Identifier*, **ULID**). Конечно, адрес ULID остается обычным адресом IPv6 и может выполнять заодно роль локатора, пока соответствующий сетевой интерфейс доступен. Так как Shim6 тесно соприкасается с IPv6, свою служебную информацию он передает в виде заголовков расширения. Информацию о доступности адресов-локаторов для нужд Shim6 поставляет вспомогательный протокол **REAP** (*Reachability Protocol*) [RFC 5534].

> Самостоятельно обсудите, вызовет ли Shim6 те же трудности, что и NAT, у прикладных протоколов, передающих адреса IP внутри своих сообщений. К таким протоколам относятся, например, FTP и SIP.

Мы не станем сейчас разбирать технические детали MPTCP и Shim6, потому что они довольно сложны и выходят за рамки нашего курса. Тем не менее, уже сказанного достаточно, чтобы увидеть, как снова победила сквозная модель: эффективное использование независимых подключений оказалось по силам самим хостам, а IPv6 предлагает почти все необходимые для этого механизмы.

Конечно, передача данных по независимым подключениям создает и новые проблемы. Одна из них состоит в том, как локальный хост убедится, что все завяленные адреса принадлежат одному удаленному хосту. Ведь иначе злоумышленник может перехватить поток данных, выдав себя за один из адресов удаленного хоста. Это в особенности касается протоколов вроде Shim6, в которых стороны не обмениваются информацией обо всех своих адресах заранее.

Связать воедино несколько адресов можно, снова воспользовавшись идентификатором интерфейса как носителем защитной информации. Допустим, хосту необходимо назначить N адресов, все в разных подсетях. То есть на входе мы получаем список префиксов подсетей $\{P_i\}$, где $1 \le i \le N$. Тогда идентификатор интерфейса L_j для префикса P_j можно вычислить как хэш-функцию *всех* префиксов, например, так:

[129] A. García-Martínez, M. Bagnulo, I. van Beijnum. The Shim6 architecture for IPv6 multihoming. IEEE Communications Magazine, 2010, v. 48, issue 9, pp. 152–157 <http://www.networks.imdea.org/Portals/8/Downloads/Publications/Shim6-Arquitecture-2010-EN.pdf>.

$$L_j = H\left(M \mid P_j \mid \{P_i\}\right).$$

То есть хэш-функция *H* вычисляется от цепочки битов, полученной сцеплением какого-то дополнительного параметра *M*, известного обеим сторонам протокола, данного префикса P_j и всего списка префиксов $\{P_i\}$. В результате мы получаем набор из *N* адресов вида:

$$A_j = P_j \mid L_j .$$

То есть каждый адрес получен сцеплением префикса подсети и отвечающего ему идентификатора интерфейса, вычисленного по вышеприведенному рецепту. Составленный таким образом адрес известен как **адрес, основанный на хэше** (*Hash-Based Address*, **HBA**) [RFC 5535]. Он имеет смысл только в составе всего набора $\{A_i\}$, известного как набор HBA (*HBA-Set*). Точнее, вне набора это обычный адрес IPv6 без дополнительных защитных свойств.

> Параметр *M* нужен, как минимум, для того, чтобы разные хосты, подключенные к одним и тем же подсетям, могли получить разные наборы HBA. Кроме того, адрес HBA случайный, а значит, есть вероятность конфликта с уже назначенным адресом. В этом случае конфликт обнаружит процедура DAD (§5.4.1). Но, чтобы разрешить конфликт, придется варьировать параметр *M* и повторить вычисление набора HBA. На практике параметр *M* составной и содержит в себе несколько полей, отвечающих за разные свойства набора HBA: уникальность, устойчивость к конфликтам, совместимость с CGA (§6.3).

> Если хэш-функция надежная, то порядок сцепления отдельных элементов в цепочку-аргумент неважен, пока один и тот же порядок принят всеми сторонами протокола. На практике одно поле *M*, а именно счетчик коллизий, возникает между префиксом P_j и списком префиксов ради совместимости с CGA. Это поле увеличивается на единицу, если процедура DAD обнаружила коллизию. Благодаря этому повторное вычисление дает другие случайные адреса.

Теперь мысленно перенесемся на другую сторону протокола и подумаем, как с помощью механизма HBA проверить адрес *A*, на который претендует удаленная сторона. Пусть этот адрес состоит из префикса подсети *P* и идентификатора интерфейса *L*. Суть проверки сводится вот к чему: принадлежит ли данный адрес *A* данному набору HBA? Чтобы провести такую проверку, проверяющей стороне потребуется параметр *M* и список префиксов $\{P_i\}$. Именно эти два элемента и определяют набор HBA, потому что все остальное можно вычислить на месте. Так, проверяющая сторона первым делом проверяет, содержится ли префикс *P* из адреса *A* в списке $\{P_i\}$. Если это не так, то налицо подлог или обычный сбой. Если же префикс — в списке, то осталось вычислить его ожидаемый идентификатор интерфейса HBA:

$$L_{HBA} = H\left(M \mid P \mid \{P_i\}\right).$$

Если адрес *A* действительно из данного набора HBA, то его идентификатор интерфейса *L* обязан совпасть с вычисленным значением L_{HBA}, с точностью до битов **U/L** и **I/G**.

Дело осталось за малым: передать параметр *M* и список префиксов $\{P_i\}$ проверяющей стороне. Оказывается, что для этого не нужен новый протокол, так как всю необходимую информацию можно поместить в блок параметров CGA (Фиг. 91 на стр. 174). Формат этого блока достаточно гибок и расширяем, чтобы в нем нашлось место для списка префиксов, а модификатор уже в нем содержится. В результате можно составлять адреса, которые одновременно удовлетворяют нормативам CGA и HBA, потому что их идентификаторы интерфейса стандартным образом зависят и от открытого ключа хоста, и от его списка префиксов. Это чудесным образом избавляет нас от конфликта между механизмами CGA и HBA.

В блоке параметров CGA уже есть случайный модификатор, префикс подсети *P*, и счетчик коллизий — они отвечают подстроке *M* | *P* в аргументе хэш-функции. Список префиксов отправится в конец блока как расширение CGA. В результате весь аргумент хэш-функции будет отформатирован как блок параметров CGA, а HBA превратится в совместимое расширение CGA.

Конечно, простой пакет HBA не пройдет криптографическую проверку CGA, поскольку он не подписан закрытым ключом. Вместо настоящего ключа HBA-без-CGA использует случайную цепочку битов в формате ключа RSA. Тем не менее, адрес HBA содержит в себе годный отпечаток этого случайного ключа.

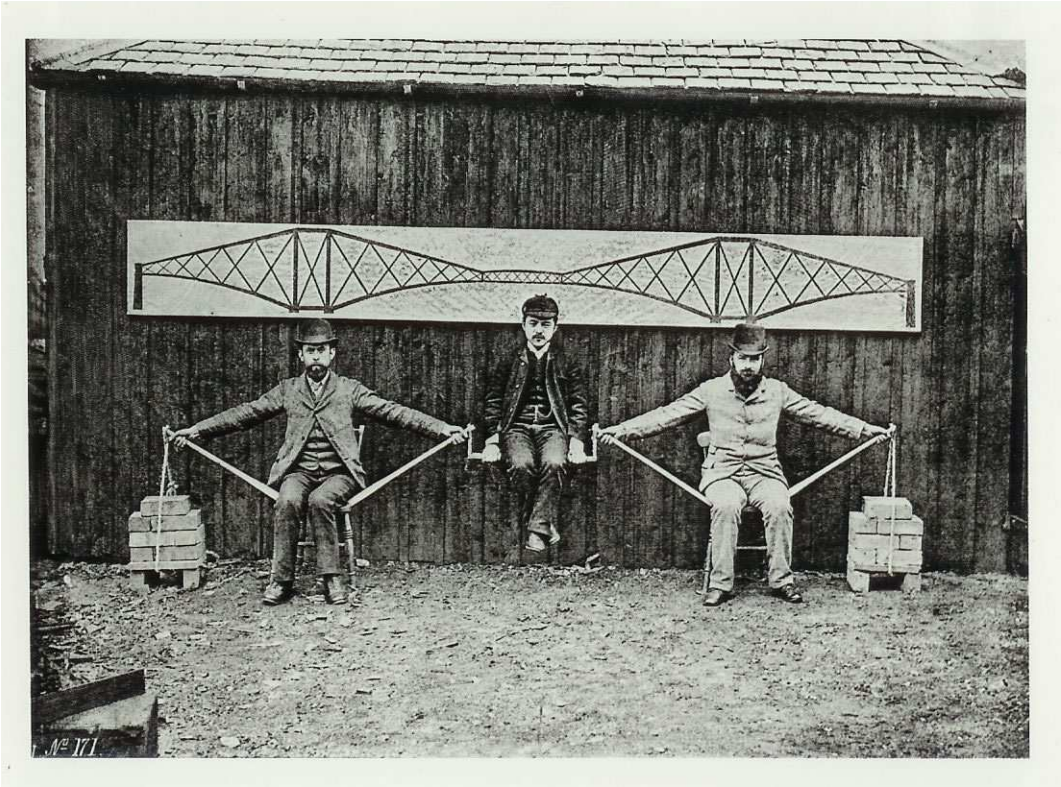
Как и можно было ожидать, HBA наследует у CGA ограничение на длину префикса подсети *P*: она обязана равняться 64 битам.

7. Заключение

На этом мы завершим наше первое «погружение» в среду IPv6 и подведем итоги. Прежде всего, отметим, что некоторые важные аспекты остались вне нашего поля зрения. Так, мы сознательно не рассматривали практические стороны эксплуатации IPv6, такие как планирование, настройка и отладка. По этим темам уже существует достаточно литературы, ориентированной на конкретные сетевые операционные системы; а нашей задачей было заложить прочный фундамент знаний, чтобы читатель смог сам решать новые задачи, которые обязательно возникнут при работе с IPv6, — и далеко не для всех из них найдутся готовые подсказки.

Кроме того, мы обошли вниманием механизмы плавного и, по возможности, безболезненного перехода между IPv4 и IPv6. Несомненно, Переход — это большая и увлекательная тема, но для знакомства с ней сначала надо разобраться в самом IPv6, чего мы вместе с читателем, как мы смеем надеяться, вполне достигли, и дальнейший путь читатель сможет проделать самостоятельно.

Ну, а чтобы поставить точку, давайте подчеркнем ту сторону IPv6, которая нам кажется наиболее значимой. Главная заслуга архитектуры IPv6, на наш взгляд, состоит в том, что она восстанавливает сквозной принцип *Internet* и расставляет по своим местам такие логически разные задачи, как адресация, безопасность и конфиденциальность [RFC 4864]. Это важно не только потому, что попытка смешать их при помощи NAT приводит к неэффективным и ненадежным решениям. Увы, современная практика IPv4 поощряет нечеткость технической мысли, когда вместо того, чтобы досконально разобраться в задании и найти структурный подход к нему (см. Фиг. 106), инженер просто составляет комбинацию малопонятных ему компонентов, которые «обычно работают»; или, хуже того, предлагает готовое «решение на все случаи жизни». В свою очередь, управленческое звено охотно делает этот ширпотреб главным продуктом, поскольку он быстр, дешев и не требует квалифицированной рабсилы. Неудивительно, что такая обстановка совершенно отучает инженеров думать и никак не способствует развитию из них хороших специалистов. Поэтому приход IPv6 — это уникальный шанс для отрасли снова вернуться к той постоянной и целенаправленной мыслительной практике, которая и создала технологию *Internet*.



Фиг. 106. Инженеры Б. Бейкер, К. Уатанабе и Дж. Фаулер демонстрируют принцип работы консольного моста [130]. Конструкциям таких инженеров можно доверять!

8. Словарик

Так как формирование русскоязычной терминологии не только IPv6, но и TCP/IP в целом по-прежнему стоит на повестке дня, ниже приведены, переведены и кратко пояснены основные термины, использованные в этой книге:

- Адрес источника (source address)
  - Адрес, указывающий на источник данного БДП.
- Адрес назначения (destination address)
  - Адрес, указывающий на адресата (или адресатов) данного БДП.
- Адрес, основанный на хэше (hash-based address, HBA)
  - Адрес IPv6, криптографически привязанный к заранее заданному списку префиксов подсети.
- Адрес цели (target address)
  - В протоколе розыска соседей IPv6 — адрес, о котором идет речь в данном сообщении.
- Адресат (destination)
  - Узел, которому адресована полезная нагрузка данного БДП.
- Блок данных протокола, БДП (Protocol Data Unit, PDU)
  - Абстракция, объединяющая сообщение, дейтаграмму, пакет, кадр и т.п.
- Вещание наугад (anycast)
  - Режим адресации, в котором БДП будет доставлен какому-то одному из интерфейсов, имеющих данный адрес. Семантически неотличим от индивидуального вещания.
- «Вне канала» (off-link)
  - Удаленный адрес IPv6 находится «вне канала» и доступен только через маршрутизатор, если не установлена возможность передать по

[130] <http://en.wikipedia.org/wiki/File:Cantilever_bridge_human_model.jpg>

нему пакет, задействуя только канальный уровень. Таким образом, все удаленные адреса IPv6 по умолчанию «вне канала».

- Внутриканальный (link-local)
  - Относящийся к области действия «один канал».
- Внутрисайтовый (site-local)
  - Относящийся к области действия «один сайт».
- Генератор запросов (querier)
  - Участник MLD, обычно групповой маршрутизатор, который в данный момент избран слать запросы MLD от имени всех групповых маршрутизаторов канала.
- Группа (multicast group)
  - Множество интерфейсов, имеющих данный групповой адрес.
- Группа искомого узла (solicited-node multicast group)
  - В розыске соседей IPv6 — внутриканальная группа, объединяющая интерфейсы с индивидуальными адресами, младшие биты которых совпадают и имеют значение, «зашитое» в адресе группы искомого узла.
- Групповой (multicast, group)
  - Режим адресации, в котором копия БДП будет доставлена всем интерфейсам, имеющим данный адрес.
- Зона действия (scope zone)
  - Участок конкретной сетевой топологии, где определен данный адрес IPv6.
- Индивидуальный (unicast, individual)
  - Режим адресации, в котором адреса интерфейсов уникальны в пределах соответствующей топологической единицы, так что БДП будет доставлен не более чем одному интерфейсу.
- Интерфейс сетевой (network interface)
  - Составная программно-аппаратная деталь сетевого стека, посредством которой узел подключен к каналу. Согласно современной концепции, адрес IP или MAC назначен определенному интерфейсу, а не всему узлу.
- Источник (source)
  - Узел, создавший данный БДП.
- Канал (link)
  - «Пассивный» элемент сетевой топологии IP, соединяющий узлы между собой.
- Канал вещания (channel)
  - Составной адрес назначения в режиме группового вещания SSM.
- Канальный (link-layer)
  - Имеющий отношение к канальному уровню сетевого стека.
- Криптографически произведенный адрес (CGA)
  - Адрес IPv6, содержащий в себе отпечаток открытого ключа.
- Локальная вычислительная сеть, ЛВС (local area network, LAN)
  - Географически ограниченный канал, поддерживающий многоадресность и широковещание.
- Локальный (local)
  - Относящийся к узлу, на котором мы сосредоточили свое внимание в данный момент.
- Маршрут (route)

    - Правило, сопоставляющее префиксу IP адрес следующего шага. Благодаря устройству адресации IP, это правило заодно выбирает выходной интерфейс в определенный канал.
- Маршрутизатор (router)
    - Узел, задача которого состоит в продвижении чужих, транзитных пакетов.
- Маршрутизация (routing)
    - Процесс выбора следующего шага для пакета.
- Многоадресный (multiaccess)
    - О канале — способный соединить более двух узлов благодаря явной адресации канального уровня.
- «На канале» (on-link)
    - Удаленный адрес IPv6 находится «на канале», если достоверно установлена возможность передать по нему пакет, привлекая только канальный уровень. В частности, такая информация может исходить от маршрутизатора в виде переадресовки.
- Наблюдатель (non-querier)
    - Участник MLD, обычно групповой маршрутизатор, который в данный момент пассивно принимает и обрабатывает сообщения MLD, потому что генератором запросов избран другой узел.
- Область действия (scope)
    - Участок абстрактной сетевой топологии, где может быть определен данный адрес IPv6.
- Обратная связь (loopback)
    - Передача пакетов, при которой источником и адресатом выступает один и тот же хост. Происходит в пределах хоста и позволяет разным приложениям на одном хосте взаимодействовать по TCP/IP.
- Общепринятый (well-known)
    - Стандартный в пределах *Internet*. Так как стандарты *Internet* не имеют силы закона, а лишь являются рекомендациями, их технические условия добровольно приняты всеми желающими полноценно участвовать в работе Сети.
- Переадресовка (redirect)
    - Сообщение ICMP, которое подсказывает более предпочтительный адрес следующего шага.
- «Петля» (loopback interface)
    - Интерфейс для обратной связи.
- Подсеть (subnet, subnetwork)
    - Часть адресного пространства IP, отвечающая одному каналу. В нотации CIDR задается префиксом подсети.
    - Подграф сетевой топологии, состоящий только из маршрутизаторов и каналов между ними; транзитная инфраструктура сети.
- Подслушивание IGMP и MLD (IGMP/MLD snooping)
    - Трюк, когда коммутатор ЛВС анализирует проходящие через него сообщения IGMP и MLD, чтобы узнать топологии существующих групп IPv4/IPv6 и оптимизировать продвижение группового трафика.
- Предпочтительный (preferred)
    - Состояние назначенного интерфейсу адреса IPv6, когда его уникальность уже подтверждена, а время жизни еще не истекло.
- Пробный (tentative)

    - Состояние назначенного интерфейсу адреса IPv6, когда его уникальность еще не подтверждена, так что пользоваться им пока нельзя.
- Продвижение (forwarding)
    - Передача полученного БДП дальше в практически неизменном виде, без работы с его полезной нагрузкой. Цель продвижения БДП — приблизить его к адресату. В контексте индивидуального вещания IP требует маршрутизации.
- Реестр (registry)
    - Список общепринятых значений какого-то параметра протокола.
- Розыск соседей (neighbor discovery, ND)
    - Процедура, в ходе которой узел IPv6 узнает адреса своих соседей по каналу, обладающих искомыми признаками. В роли такого признака может выступить обладание определенным индивидуальным адресом, готовность быть маршрутизатором по умолчанию и т.п.
- Сайт (site)
    - Условная единица сетевой топологии, охватывающая часть сети под единым административным началом.
- Связь безопасности (security association)
    - В протоколе IPsec — центральная структура данных, определяющая, какие именно правила защиты надо применять к пакету в зависимости от его адреса источника, адреса назначения и типа полезной нагрузки.
- Сквозной (end-to-end)
    - Привлекающий только источник и адресата пакета; не зависящий от трассы пакета.
- Следующий шаг (next hop)
    - Узел, принимающий пакет при совершении им очередного шага.
- Точный более или менее (more/less specific)
    - О префиксе IP — содержащий больше или меньше значащих битов и потому отвечающий меньшему или большему множеству адресов, соответственно.
- Трасса (path)
    - Последовательность шагов пакета IP сквозь сеть.
- Удаленный (remote)
    - Относящийся к другим узлам, кроме локального. В контексте парного сетевого взаимодействия это узел на дальнем, относительно локального узла, конце соединения.
- Узел (node)
    - «Активный» элемент сетевой топологии IP, принимающий и передающий пакеты по каналам.
- Устаревший (deprecated)
    - Состояние назначенного интерфейсу адреса IPv6, когда его предпочтительное время жизни уже истекло, а действительное — еще нет.
- Хост (host)
    - Узел, роль которого состоит в создании новых пакетов на выходе и в потреблении адресованных ему пакетов на входе.
- Шаг (hop)
    - Элементарный акт перемещения пакета IP от источника по направлению к адресату сквозь сеть. Сводится к передаче пакета в

пределах одного канала. В ходе этой передачи удаленный узел, который принимает пакет, известен как «следующий шаг».

- Широковещательный (broadcast)
  - Режим адресации, в котором копия БДП будет доставлена всем интерфейсам данной топологической единицы.