Маршрутизаторы NSG

Программное обеспечение NSG Linux 2.0

Руководство пользователя

Часть 3

Обработка IP-трафика

Версия программного обеспечения 2.0 build 7

Обновлено 18.04.2017

Москва 2017

АННОТАЦИЯ

Данный документ содержит руководство по настройке и применению маршрутизаторов NSG, оснащенных программным обеспечением NSG Linux 2.0. Документ имеет следующую структуру:

Руководства по применению маршрутизаторов под управлением NSG Linux 1.0 и базового программного обеспечения NSG, а также других продуктов NSG (модемов, мостов и т.п.), содержатся в отдельных документах.

ВНИМАНИЕ

Данное Руководство пользователя предназначено для лучшего понимания процедуры настройки в целом и описывает сущность выполняемых действий — а именно, *что* необходимо настраивать. Рассматриваемые вопросы относятся, как правило, к сущности используемой технологии и являются общими для любых её реализаций, независимо от конкретного производителя и устройства. (Исключением являются вопросы, специфичные для оборудования NSG — таких, как организация пользовательского интерфейса или настройка системы бесперебойных соединений ***ui*TCP**.)

Основной документацией по NSG Linux 2.0 является встроенная справка на борту устройства. Она описывает конкретные команды и параметры настройки — т.е. *как* настраивать функции и возможности, описанные в данном Руководстве. Для просмотра справки по каждому из параметров следует использовать кнопку в Web-интерфейсе или команду _manual (_m) в консольном интерфейсе. Если справка на вашем языке отсутствует, следует установить на устройстве русскую локаль.

ВНИМАНИЕ Продукция компании непрерывно совершенствуется, в связи с чем возможны изменения отдельных аппаратных и программных характеристик по сравнению с настоящим описанием. Сведения о последних изменениях приведены в файлах README.TXT, CHANGES, а также в документации на отдельные устройства.

Замечания и комментарии по документации NSG принимаются по адресу: doc@nsg.net.ru.

ООО «Эн–Эс–Джи»
Россия 105187 Москва
ул. Вольная, д.35
Тел./факс: (+7–495) 727–19–59 (многоканальный)

http://www.nsg.ru/
mailto:info@nsg.net.ru
mailto:sales@nsg.net.ru
mailto:support@nsg.net.ru

§ СОДЕРЖАНИЕ §

Часть 3. Обработка IP-трафика

§3.1. Общие сведения о процедурах обработки IP-трафика

В данной части Руководства пользователя рассмотрены настройки, относящиеся к процедурам обработки пакетов IP в NSG Linux 2.0. Все эти процедуры осуществляются по общему принципу:

— Анализируется набор *критериев*, содержащихся в заголовке 3 уровня модели TCP/IP, или 3 и 4 уровней модели OSI (т.е. заголовки собственно IP и следующего вложенного в него протокола — TCP, UDP, ICMP и т.п.). Наиболее важным из параметров заголовка является IP-адрес назначения, поскольку основная функция сети передачи данных состоит именно в том, чтобы доставить пакет на этот адрес. Наряду с ним, могут учитываться IP-адрес источника, протокол 4 уровня, номера портов TCP или UDP источника и назначения, флаги TCP-соединения и ряд других полей.

  В некоторых случаях учитываются более сложные зависимости: обмен пакетами через взаимосвязанный порт (например, для FTP — порты TCP 20 и 21), принадлежность пакета к некоторому выявленному потоку трафика и т.п.

— В зависимости от совокупности этих критериев выполняется некоторое *действие* с пакетом: передать через некоторый выходной интерфейс устройства в следующую сеть, уничтожить, изменить некоторые параметры заголовка пакета, поставить в очередь, но не передавать до выполнения каких-либо дополнительных условий, и т.п.

Особо следует подчеркнуть, что суть описываемых процедур и их параметров никак не связана с синтаксисом команд того или иного конкретного устройства или программного средства. В данном документе основное внимание уделено именно описанию процедур обработки пакетов по существу; детальное описание отдельных параметров приведено во встроенной справке непосредственно в составе NSG Linux 2.0, и нет особого смысла дублировать его на бумаге.

В список возможных процедур входят:

— Маршрутизация
— Фильтрация
— Преобразование сетевых адресов и портов (NAT)
— Программная коммутация пакетов 3 уровня
— Управление качеством услуг (QoS)

Механизмы мониторинга и отладки сети, в т.ч. в части обработки IP-пакетов, описаны в Части 4.

§3.2. IP-маршрутизация

§3.2.1. Общие сведения о сетях и межсетевом взаимодействии

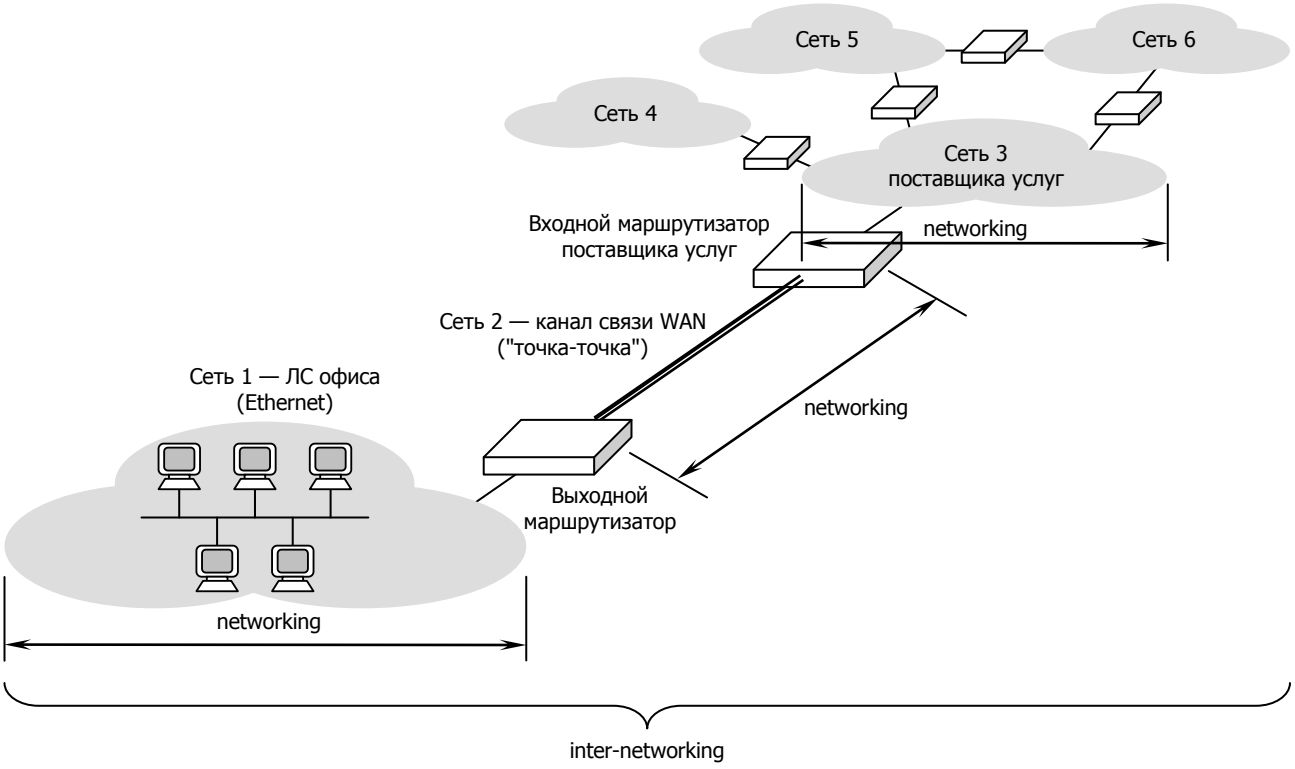
Прежде чем описывать настройку маршрутизации, уместно вкратце сформулировать сущность межсетевого взаимодействия в целом, напомнить структуру адресного пространства IP и уточнить обсуждаемые понятия.

Для начала, уточним, что подразумевается под понятием "сеть". В узком смысле этого термина, сеть — это некоторая совокупность компьютеров, подключённых к одной физической среде для передачи данных[1]. В этой среде компьютеры, точнее, их сетевые адаптеры, идентифицируют друг друга (если в этом есть необходимость) по аппаратным адресам (MAC-адресам). *В пределах сети* каждый хост непосредственно доступен каждому через общую среду передачи. Это есть обмен данными на 2 уровне, или *networking*. Заметим, что оболочка пакетов 2 уровня имеет смысл только в пределах одной сети.

Сети (а именно, сети 2 уровня) разделяются на два принципиально различных класса. Первый — это каналы "точка-точка", соединяющие всегда 2 и только 2 хоста. К ним относятся синхронные и асинхронные последовательные соединения, модемные соединения всех типов, каналы сетей PDH операторов связи (G.703.*x*, E1, E3 и др.). Такие соединения отличаются тем, что не требуют адресации ни на каком уровне: очевидно, что если пакет передать в линию, то он попадёт на соседнее устройство и никуда более.

Другой класс — это широковещательные сети, в которых к общей среде передачи подключено (или может быть подключено) более 2 хостов. На сегодняшний день это почти исключительно Ethernet или эмуляция Ethernet через различные другие среды — Wi-Fi, WiMAX, Ethernet-over-HDLC, Ethernet-over-*x*DSL и др. В такой сети отправитель и получатель пакета обязательно должны идентифицировать друг друга, т.е. адресация необходима. При этом адреса 2 и 3 уровней — различные по существу, и требуются определённые процедуры для определения одного по другому (ARP, RARP).

Из общих физических соображений очевидно, что число хостов в сети 2 уровня не может быть неограниченно большим. Другие хосты, с которыми может потребоваться обмен данными, будут находиться в других сетях, тоже ограниченного размера. Для передачи пакетов *между сетями* предназначены специально выделенные компьютеры с несколькими сетевыми интерфейсами, подключенные к нескольким сетям одновременно. Они принимают пакет из одной сети, разбирают его до 2 уровня включительно, по заголовку 3 уровня определяют его дальнейший путь, заново упаковывают его в оболочку 2 уровня, соответствующую новой сети, и отправляют дальше. Именно эта процедура называется *межсетевым взаимодействием*, или *inter-networking*. Протокол, её регламентирующий — это и есть Internetworking Protocol, IP.



[1] Внутри сети могут находиться также устройства 1 уровня (концентраторы, медиа-конвертеры, модемы) и 2 уровня (коммутаторы, мосты), но сути дела это не меняет.

§3.2.2. IP-адреса

На 3 уровне все хосты объединённых сетей рассматриваются как единая мета-сеть. Адресация 3 уровня в IP-сетях ориентирована на построение иерархической структуры, где каждая подсеть является частью другой. Предполагается, что, например, крупный поставщик услуг Интернет получает из общего адресного пространства большой блок адресов, который он делит на части и раздаёт более мелким дочерним провайдерам. Те, в свою очередь, делят свои адреса на более мелкие блоки и раздают клиентам. Каждый клиент также делит выделенный ему блок адресов, например, по своим структурным подразделениям, и внутри подразделения раздаёт адреса своим компьютерам.

Логичной и естественной является ситуация, когда каждая из сетей 2 уровня, показанных на рисунке выше, использует один непрерывный диапазон IP-адресов, и этот диапазон не пересекается с адресами, принадлежащими другим сетям. Это значительно упрощает ряд задач, в особенности:

— Автоматическое построение маршрутов (см. §3.2.6). Маршруты в смежные сети создаются в соответствии с адресными параметрами каждого интерфейса.

— Построение туннелей IPsec (см. Часть 5). Технология IPsec принимает в качестве постулата, что две приватные сети, соединяемые через безопасный туннель, имеют непересекающиеся пространства IP-адресов. Невыполнение этого требования может приводить к непредсказуемым результатам в зависимости от конкретной реализации IPsec.

Адрес IPv4 состоит из 32 бит и записывается, как правило, в т.н. *десятично-дотовой нотации* — в виде 4 чисел от 0 до 255, разделённых точками. Адрес разделяется на две части: адрес сети и адрес хоста внутри сети; граница между ними — подвижная и устанавливается *маской*. Маска также содержит 32 бита, причём сколько-то начальных бит подряд могут быть единицами, все остальные — только нулями. Часто вместо самой маски пишется, для краткости, только её длина (число единиц), поскольку по существу это единственная значимая величина в ней. Пример:

адрес	192	168	12	34
	11000000.10101000.00001100.00100010			
маска (/24)	255	255	255	0
	11111111.11111111.11111111.00000000			
адрес сети	192	168	12	0
	11000000.10101000.00001100.00000000			
локальный адрес хоста в сети	.	.	.	34
	.	.	.	.00100010
широковещательный адрес	192	168	12	255
	11000000.10101000.00001100.11111111			

Адрес сети является общим для всех хостов, входящих в данную сеть. Предположим, для определённости, что длина маски n бит. При наложении маски на адрес получается часть адреса, в которой последние 32−n бит заменены нулями. Это и есть адрес сети.

Оставшаяся часть адреса — это *локальный адрес* компьютера в сети. В чистом виде он практически никогда не используется, поскольку IP-адреса, по определению, предназначены для межсетевого взаимодействия.

Адрес, в котором все свободные биты, т.е. 32−n последних, заменены единицами, имеет особое назначение — это *широковещательный адрес*. Широковещательный адрес и адрес сети не могут назначаться никаким реальным хостам в сети.

Всего при длине маски n бит сеть может содержать $2^{(32-n)}$ адресов, из которых первый всегда является адресом сети, последний — широковещательным. Остальные адреса могут назначаться отдельным хостам в сети.

Использовать маски исключительно длиной /8, /16 или /24, т.е. по границам целых байт в адресе, в общем случае необязательно. Это было задумано когда-то на заре Интернета исключительно ради удобочитаемости. Ныне, в связи с дефицитом IPv4 адресов, широко используются дробные маски.

Особыми случаями являются самые длинные маски. Маска длиной /30 (255.255.255.252) широко используется для соединений "точка-точка", в которых задействованы ровно 2 хоста. Она содержит всего 4 адреса: сетевой, широковещательный, и два адреса для одного и для другого хоста. Создать реально значимую подсеть меньшего размера, с учётом вышеизложенных правил, уже невозможно.

Маска /32 (255.255.255.255) однозначно определяет ровно 1 IP-адрес. Такая запись обычно используется для указания, что в некотором контексте, где по смыслу должна быть указана некоторая сеть, речь в данном случае идёт строго о единственном хосте.

Маска /31 (255.255.255.254) формально не запрещена, но по существу она практически равнозначна /32. Отличие состоит в том, что интерфейс имеет свой уникальный IP-адрес, а не "одолженный" у другого. Другой адрес в этой подсети фактически не используется никак.

Как это ни тривиально выглядит, но необходимо напомнить, что каждый хост в сети должен иметь уникальный адрес, не совпадающий с адресом никакого другого хоста. Более подробно, это означает, что:

— Все IP-подсети должны иметь разные сетевые адреса.
— Все хосты в пределах одной подсети должны иметь одинаковую маску.
— Все хосты в пределах одной подсети должны иметь одинаковый адрес сети (начальную часть адреса, ограниченную маской).
— Все хосты в пределах одной подсети должны иметь различные локальные адреса (конечная часть адреса после маски).

Специальными являются следующие три группы подсетей:

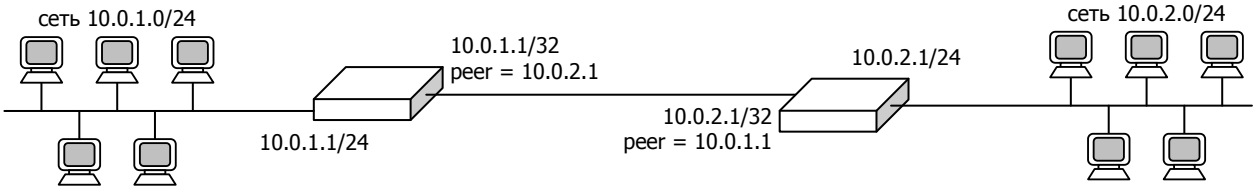
10.0.0.0/8
172.16.0.0/16 … 172.16.31.0/16
192.168.0.0/24 … 192.168.255.0/24

Эти адреса зарезервированы для применения в приватных сетях, не имеющих прямого выхода в сети общего пользования. На жаргоне администраторов они называются "серыми". Пакеты с такими адресами не должны циркулировать в Интернете вообще. На выходе из приватных сетей в Интернет производится процедура NAT (преобразование сетевых адресов), при которой эти адреса заменяются на корректные (реальные, глобальные, "белые"). Сегодня, в связи с дефицитом IPv4 адресов, а также по соображениям безопасности, это общепринятая практика построения корпоративных сетей и сетей поставщиков услуг. Зато использовать эти адреса можно многократно, в каждой корпоративной сети, не задумываясь о том, что эти же адреса используются и в других таких же сетях: напрямую эти сети никогда не взаимодействуют. В пределах одной корпоративной сети администратор может использовать их по своему усмотрению, следя только за тем, чтобы они не конфликтовали друг с другом.

§3.2.3. Соединения "точка-точка" и ненумерованные интерфейсы

В соединениях "точка-точка", состоящих всегда ровно из 2 хостов, адресация, как уже говорилось, в принципе является излишней: все, что передаётся одной стороной соединения, принимается другой, и наоборот. IP-адреса обеим сторонам соединения нужны только для того, чтобы посылать и принимать пакеты от своего собственного лица, например, *ping* с одного конца соединения на другой. Кроме того, наличие адресов может быть формальным требованием протоколов (например, PPP) или их реализации в конкретных продуктах. По сравнению с широковещательными сетями, для соединений "точка-точка" широко используются следующие особенные методы назначения IP-адресов:

1. Сеть с маской /30. Хотя этот вариант весьма неэффективен с точки зрения расходования IP-адресов, он удобен тем, что логика их назначения и использования не отличается от широковещательных сетей. Хотя, с другой стороны, в приватных сетях ничто не мешает использовать и подсети с более короткой маской, например, /24.
2. Указание адреса с маской /32 и явное указание адреса удалённой стороны (peer). При этом указанные два адреса уже никак не связаны друг с другом.
3. Использование адреса, "позаимствованного у" ("*borrowed from*") другого интерфейса этого же устройства. Как и в предыдущем случае, адрес указывается с маской /32. Интерфейс, не имеющий собственного адреса, называется *ненумерованным* (*unnumbered*). Причины, требующие наличия адреса, в данном случае также удовлетворены: требования протокола формально соблюдены, есть что подставить в качестве *source-address* в пакеты, отправляемые локально с этого интерфейса, и локально же принимаются и обрабатываются пакеты, приходящие на этот адрес.
   Адрес удалённой стороны в этом случае также указывается явно, причём на той стороне также может быть ненумерованный интерфейс. Например, два маршрутизатора, соединённые каналом "точка-точка", могут использовать для своих ненумерованных интерфейсов адреса от своих же портов Ethernet.

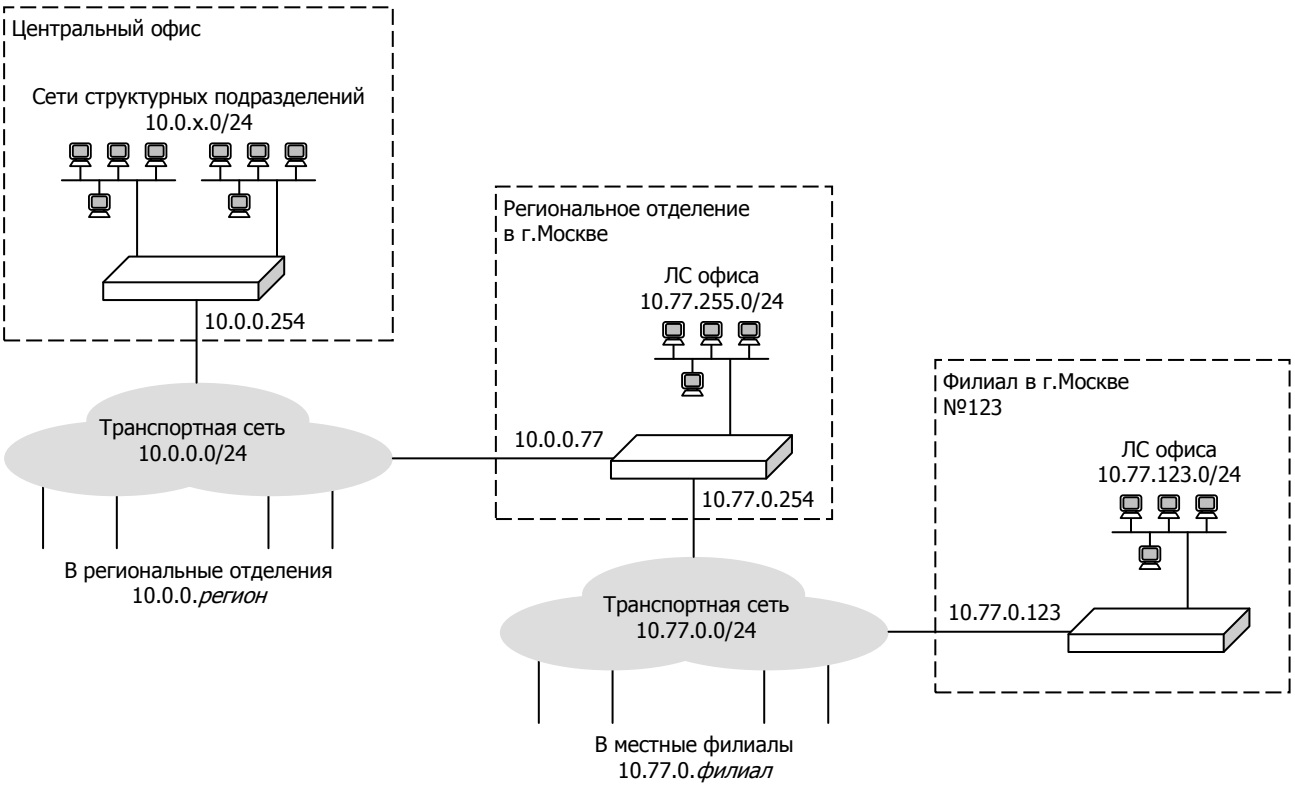


Ненумерованные интерфейсы, в частности, широко используются для серверов PPP, PPPoE, PPTP. В этом случае всем клиентским интерфейсам сервера назначается адрес, например, от его магистрального порта Ethernet, а удалённым клиентам назначаются обычные IP-адреса.

Физические соединения WAN "точка-точка" и соответствующие им протоколы 2 уровня в данной версии NSG Linux 2.0 не поддерживаются и целесообразность их поддержки в будущем не очевидна ввиду их невысокой востребованности в современных решениях. Тем не менее, всё сказанное выше относительно IP-адресов сохраняет актуальность, поскольку относится в равной степени и к асинхронным соединениям PPP (в т.ч. через сотовые сети), и к туннелям различных типов: PPPoE, PPTP, GRE, ***ui*TCP** в режиме виртуальных интерфейсов.

§3.2.4. Пример адресного плана для корпоративной сети

При построении корпоративной сети крайне важно изначально заложить в неё корректный план адресации. Это поможет избежать в будущем многочисленных проблем, в первую очередь, как сказано выше, в части маршрутизации и построения туннелей IPsec. Для этой цели можно использовать любой из приватных диапазонов, перечисленных в предыдущем параграфе. Удобно использовать сеть 10.0.0.0/8 и структурировать её таким образом, чтобы отдельные байты несли понятную мнемоническую информацию о каждой подсети: номер структурного подразделения, код региона и т.п. Например, для некоторой организации, имеющей разветвлённую сеть филиалов по России, можно предложить такую схему:



Если вместо широковещательных транспортных сетей, например, в регионы, используются, везде или частично, каналы "точка-точка", то можно в каждом канале использовать адрес 10.0.0.1*xx* на стороне центрального офиса и 10.0.0.*xx* на стороне регионального, где *xx* — код региона. Оба адреса задаются с маской /32, и в конфигурации интерфейса явным образом указывается адрес партнёра (*peer address*). Такая схема несколько сложнее привычных подсетей с маской /30, но зато позволяет легко идентифицировать каждый хост и подсеть.

§3.2.5. Процедура IP-маршрутизации

Маршрутизация IP-пакетов — это выбор следующего этапа передачи пакета от данного устройства, в зависимости от его адреса назначения. Настройка маршрутизации во всех случаях заключается в составлении *таблицы маршрутизации*, определяющей, куда передавать пакет, в следующем виде:

> "Если адрес назначения пакета находится в сеть_назначения, то передать его через интерфейс device маршрутизатору gateway. Дополнительные параметры маршрута: метрика, …"

Собственно *маршрут* и представляет собой набор указанных параметров. Обязательным критерием маршрута является описание сети назначения, без него маршрут не имеет смысла. Маршрут на некоторый одиночный хост можно описать как маршрут в сеть с маской 255.255.255.255, или /32.

Один и тот же пакет может подпадать одновременно под несколько записей маршрутной таблицы, например, для нескольких сетей, из которых одна входит в другую. В этом случае безусловно выбирается наиболее специфичный маршрут, т.е. маршрут, в котором сеть назначения указана с самой длинной маской. Таким образом, маршрут на явно заданный хост, т.е. с маской /32, имеет приоритет перед всеми остальными.

Противоположный особый случай — это сеть 0.0.0.0/0. Эта сеть включает в себя все возможные адреса, поэтому такие маршруты имеют особый смысл и называются *default route* — маршрут, который следует использовать по умолчанию. Если для пакета не найдено ни одного действующего маршрута с явным описанием хоста или сети назначения, то он будет отправлен по этому маршруту.

ВНИМАНИЕ Необходимо различать *статическую* таблицу маршрутизации, заданную вручную в конфигурации устройства, и *текущую* таблицу маршрутизации, действующую в данный момент. Текущая таблица формируется и изменяется в процессе работы устройства с учётом состояния интерфейсов, данных от протоколов динамической маршрутизации, и других факторов.

Два параметра device и gateway определяют действие, которое надлежит выполнить с пакетом. Из них обязательным и, как правило, достаточным является один, в зависимости от используемой среды передачи:

— Для каналов WAN "точка-точка" и виртуальных туннелей достаточно указать выходной интерфейс. Адрес шлюза практического значения не имеет. Можно указать даже шлюз, который не соединён с данным маршрутизатором непосредственно, но маршрут к нему разрешается в некоторое соединение типа "точка-точка" — такой маршрут создастся и будет работать, хотя на практике такая замысловатая конфигурация требуется крайне редко и обычно заменяется по существу более прямолинейными механизмами.

— Для широковещательных интерфейсов, наоборот, необходимо и достаточно указать адрес следующего шлюза, причём находящийся в одной сети с данным маршрутизатором. Выходной интерфейс в направлении этого шлюза определяется автоматически по текущей таблице маршрутизации.

ПРИМЕЧАНИЕ Наряду с обычными маршрутами через заданный шлюз или интерфейс, могут создаваться специальные типы маршрутов, например, административный запрет на передачу пакетов в указанную сеть (*blackhole, phohibited* или *unreachable* — в зависимости от того, какая диагностика возвращается отправителю пакета). Действие с пакетом, попадающим под такое правило маршрутизации, обычно полностью определено его типом, поэтому дополнительные параметры (выходной интерфейс и/или шлюз) для него уже не требуются.

Метрика маршрута — дополнительный параметр, используемый на этапе выбора маршрута. Она позволяет установить приоритет каждого маршрута по сравнению с другими маршрутами при прочих равных условиях. Изначально она означала число промежуточных маршрутизаторов на пути к заданной сети, но ныне является чисто условной величиной. Важно только её относительное значение для сравниваемых маршрутов. Именно, если пакет подпадает под две или более записей маршрутной таблицы, имеющие *строго одинаковую* сеть назначения, то будет выбран маршрут с наименьшей метрикой.

Если маршрутов с идентичной сетью назначения *и* идентичной метрикой оказывается более одного, то результат может быть разным в зависимости от того, какой алгоритм заложен в конкретной реализации маршрутизатора на этот случай. В простейшем варианте пакет будет отправлен по первому найденному маршруту, но порядок следования записей в *текущей* таблице маршрутизации определяется рядом случайных факторов: последовательностью поднятия/опускания интерфейсов, загрузкой процессора в момент отработки конфигурации и др. Возможен и другой алгоритм обработки пакетов в таком случае — равномерная передача пакетов по всем этим маршрутам. В терминологии ОС Linux он называется *equal cost multipath*, в продуктах некоторых производителей — *load balancing in per-session mode*. В NSG Linux 2.0 используются оба алгоритма, в зависимости от используемого механизма управления маршрутизацией.

То и другое относится, в частности, и к маршрутам по умолчанию, которых тоже может быть более одного.

Просмотреть текущую таблицу маршрутизации можно командой .ip.route.show .

§3.2.6. Источники информации о маршрутах

Маршрутная таблица формируется из нескольких источников. Перечислим их в том порядке, в каком они примерно используются по мере старта системы.

1. Маршруты в непосредственно подключённые сети. При поднятии каждого сетевого интерфейса система формально рассчитывает возможное адресное пространство сети, в которую он включён, исходя из его адреса и маски, и создаёт соответствующие записи в таблице маршрутизации, указывающие в этот интерфейс:
   — на адрес данного интерфейса с маской /32 — со специфическим атрибутом local; благодаря ему, пакет в конечном счёте перенаправляется в локальный псевдоинтерфейс lo, для передачи прикладным процессам на данном устройстве.
   — на широковещательный адрес и на адрес сети с маской /32 — с атрибутами local и broadcast; эти пакеты направляются и на псевдоинтерфейс lo, и в сеть, причём с широковещательным MAC-адресом назначения.

   Эти три маршрута в ОС Linux хранятся в отдельной, приоритетной, таблице маршрутизации local и не выводятся командой .ip.route.show , но их следует держать в уме. Подробнее об организации таблиц маршрутизации в ОС Linux см. §3.6.
   — на адрес сети с маской, указанной для данного интерфейса — в данный интерфейс. Это маршрут на все *другие* хосты данной сети. Поскольку маска более короткая, этот маршрут имеет низший приоритет по сравнению с предыдущими. Маршрут хранится в основной таблице main, которая используется по умолчанию в случае отсутствия особых настроек. Содержимое данной таблицы выводится командой .ip.route.show .

   Для интерфейсов с длиной маски /32 или /31 вместо второй и третьей записей создаётся маршрут на адрес удалённой стороны (*peer address*, указанный в конфигурации интерфейса), с маской /32. Если интерфейс ненумерованный, то первая запись не создаётся, потому что она уже имеется для того интерфейса, от которого "одолжен" этот адрес.
2. Статическая таблица маршрутизации. Это список маршрутов, который составляется администратором вручную при конфигурировании устройства. В процессе работы он не изменяется, но из него в текущую таблицу маршрутизации переносятся только действующие маршруты. Если указанный маршрут, например, проходит через неработающий интерфейс, то он удаляется из текущей таблицы маршрутизации.
3. Механизмы динамической конфигурации интерфейсов — PPP и DHCP.

   Протокол DHCP используется на широковещательных интерфейсах. Клиент DCHP, работающий на интерфейсе, получает от централизованного сервера DHCP ряд настроек, в т.ч. IP-адрес, маску подсети (в соответствии с которыми создаются маршруты согласно п.1) и адрес шлюза по умолчанию. Указанный маршрут включается в таблицу маршрутизации (если не установлен параметр discard-default-gw).

   Протокол PPP используется для сеансовых соединений "точка-точка", клиентов PPPoE и PPTP. При установлении соединения создаются два маршрута на локальный адрес и адрес удалённой стороны (не важно, были ли они настроены статически или получены динамически). Кроме того, установленное соединение может быть объявлено маршрутом по умолчанию, это устанавливается параметром default-route = true в настройках PPP на соответствующем порту. Этому маршруту может быть принудительно назначена метрика, отличная от 0.

   ПРИМЕЧАНИЕ Интерфейсы PPP, настроенные на соединение по требованию (connection = "on-demand"), формально находятся в состоянии UP постоянно и на них создаются соответствующие маршруты — в противном случае в порт не попадали бы никакие пакеты, и он бы никогда не смог инициировать реальное соединение. Если IP-адреса должны назначаться им динамически, то в отсутствие соединения генерируются случайные адреса для формальных целей, а после установления соединения соответствующие записи в маршрутной таблице удаляются и создаются заново.
4. Протоколы динамической маршрутизации обеспечивают обмен информацией между соседними маршрутизаторами. Таким образом, сведения об имеющихся подсетях (изначально полученные каждым маршрутизатором из первых 3 источников) и маршрутах к ним автоматически распространяются по сети и изменяются по мере необходимости.

ПРИМЕЧАНИЕ Разные протоколы динамической маршрутизации используют различные определения для метрики или её эквивалентов (distance и т.п.) — хотя бы потому, что они изначально предназначались для сетей различного масштаба, с различным максимально возможным числом промежуточных шагов. Поэтому при использовании динамической маршрутизации метрика, сообщаемая каждым протоколом, имеет смысл только для сравнения с другими маршрутами, построенными с помощью этого же протокола. Для сравнения маршрутов, построенных по разным протоколам, или полученных другими способами, необходимо вводить правила преобразования метрик, или, по крайней мере, устанавливать безусловный приоритет одного протокола перед другим.

5. Некоторые специфические средства NSG Linux изменяют или позволяют изменять таблицу маршрутизации (при помощи скриптов) в процессе работы устройства. Наиболее часто эта возможность используется в службе *netping*, в технологии бесперебойных туннелей ***ui*TCP** и в клиенте PPTP.

§3.2.7. Статические маршруты

Статические маршруты в NSG Linux 2.0 создаются и удаляются в узле .ip.route . Для создания нового маршрута необходимо использовать команду _insert или нажать на кнопку ✚, для удаления использовать команду _remove или кнопку ━. После создания нового маршрута необходимо отредактировать его параметры — как минимум, сеть назначения и шлюз или выходной интерфейс. Это новая информация, которая заранее системе не известна и не может быть известна, поэтому не может быть настроена автоматически.

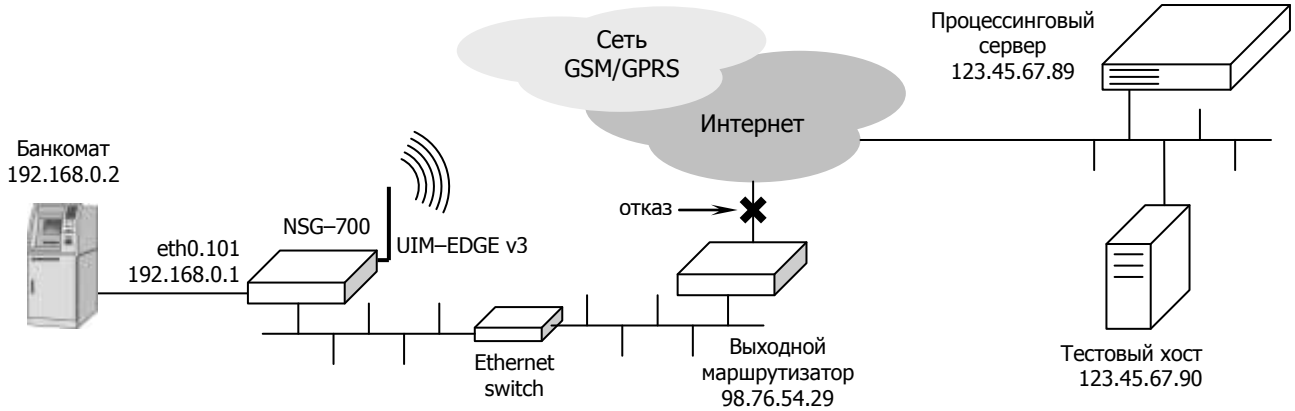
ПРИМЕЧАНИЕ Маршруты, создаваемые в Web-интерфейсе или командной оболочке nsgsh, маркируются специальным селектором proto nsg. Это позволяет отличать их от маршрутов, созданных другими способами. Демон конфигурации nsgconfd следит за этими и только за этими маршрутами, удаляет и восстанавливает их по мере необходимости. Остальные маршруты контролируются теми службами и механизмами, которые их создали.

Узел .ip.route предназначен для настройки маршрутизации непосредственно в ядре Linux. Режим *equal cost multipath* для этой простой маршрутизации не поддерживается, при наличии нескольких равноценных маршрутов пакет отправляется по первому найденному. Помимо этого, статические маршруты могут настраиваться в узле .ip.dynamic-routing посредством пакета *bird* (см. след. раздел). В этом узле возможны более сложные конфигурации и алгоритмы, в т.ч. использование *equal cost multipath*.

§3.2.8. Настройка статической маршрутизации командами ОС Linux

Пользователи, знакомые с ОС Linux, могут настраивать маршруты вручную при помощи стандартных команд route или ip route (рекомендуется использовать последнюю). Однако злоупотреблять этой возможностью не следует, поскольку в этом случае легко запутаться с маршрутами, которые созданы разными средствами. Более важная область применения этих команд — скрипты, которые могут исполняться по различным событиям, например, по срабатыванию механизма *netping* (более подробно о службе *netping* см. Часть 4).

В примере, приведённом ниже, служба создаётся и удаляется маршрут к процессинговому серверу через основное подключение Ethernet в зависимости от того, проходит ли через это соединение *ping* на контрольный хост или нет. Если маршрут существует, то он имеет по умолчанию метрику 0 и получает приоритет перед резервным маршрутом через сотовый интерфейс s1.




```
ip
: route
: 1
: : device          = "s1"
: : metric          = 3
: : network         = "123.45.67.89/32"
: 2
: : gateway         = "98.76.54.29"
: : network         = "123.45.67.90/32"
services
: netping
: : route_swap
: : : adm-state     = "up"
: : : description   = "Re-route ATM packets to GPRS back-up"
: : : destination   = "123.45.67.90"
: : : failure-script = "ip route del 123.45.67.89/32 via 98.76.54.29"
: : : restore-script = "ip route add 123.45.67.89/32 via 98.76.54.29"
```

§3.3. Динамическая и сложная статическая IP-маршрутизация

§3.3.1. Общие сведения

Протоколы динамической маршрутизации предназначены для автоматического обмена маршрутной информацией между маршрутизаторами. В NSG Linux 2.0 поддерживаются протоколы RIP v2, OSPF, BGP. Реализация данных протоколов, в отличие от NSG Linux 1.0, основана на более современном пакете *bird* (BIRD Internet Routing Daemon). Соответственно, структура настроек в целом соответствует синтаксису файла конфигурации данного проекта.

Узел меню .ip.dynamic-routing содержит наиболее употребительные параметры *bird*. Параметры и опции, не представленные явно, в любом случае можно задать в полях extra (вплоть до того, что вписать в него все опции запуска *bird* полностью и оставить остальные поля пустыми). Подробно о настройках *bird* см. оригинальную документацию на сайте проекта: http://bird.network.cz .

Помимо динамической маршрутизации, пакет обеспечивает развитые средства для сложной статической маршрутизации с использованием нескольких таблиц, встроенные опции для фильтрации пакетов по IP-адресам и др. В некотором смысле, он заменяет собой ручную настройку маршрутов в узле .ip.route . Рекомендуется использовать узел .ip.route для простой статической маршрутизации (к ней относится большинство случаев), а возможности *bird* — для настройки более сложных случаев (например, с использованием алгоритма *equal cost multipath*) в сочетании с динамической маршрутизацией (это позволяет, в частности, унифицировать метрики маршрутов, созданных различными способами). Маршруты, созданные в узле .ip.route , также интегрируются в маршрутные таблицы *bird* со статусом *kernel routes*.

§3.3.2. Принципы работы пакета BIRD

Пакет BIRD представляет собой надстройку над подсистемой маршрутизации, встроенной в ядро ОС Linux. Он выполняет обработку всей маршрутной информации, доступной из различных источников, вычисление маршрутов и, в конечном счёте, вносит изменения в маршрутные таблицы ядра Linux.

BIRD включает в себя следующие основные компоненты:

— Собственные маршрутные таблицы BIRD (в общем случае, не тождественные маршрутным таблицам в ядре Linux). Как в ядре Linux, так и в BIRD предусмотрены множественные маршрутные таблицы.

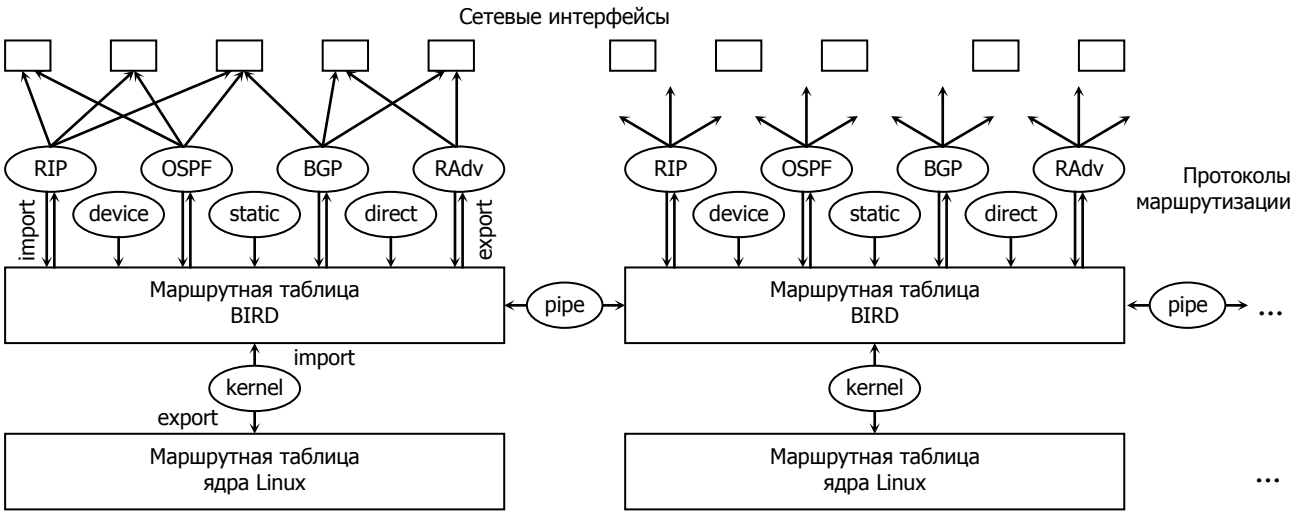
— Стандартные протоколы маршрутизации (BGP, OSPF, RIP2), а также IPv6 Router Advertisement — механизмы, предназначенные для динамического наполнения маршрутных таблиц BIRD. Относительно простые и доступные описания работы данных протоколов для начинающих сетевых администраторов можно найти, например, в Википедии (на английском и частично на русском языках).

— Псевдо-протоколы *static*, *device*, *direct*, предназначенные для внесения в маршрутные таблицы BIRD записей, генерируемых иными средствами (маршрутов в непосредственно подключённые сети, явно заданных статических маршрутов и т.п.).

— Псевдо-протоколы *pipe* для обмена между маршрутными таблицами BIRD и *kernel* для синхронизации маршрутных таблиц BIRD и ядра Linux.

— Фильтры для импорта маршрутов из отдельных протоколов в маршрутные таблицы BIRD и экспорта маршрутов из таблиц BIRD в заданные протоколы динамической маршрутизации для раздачи партнёрам.

Схема взаимодействия различных составляющих BIRD показана на рисунке.



Каждая из маршрутных таблиц BIRD может синхронизироваться или не синхронизироваться с таблицами в ядре Linux и друг с другом. Каждый маршрут в таблице BIRD содержит следующую информацию:

— Префикс IP-подсети назначения.
— Административный приоритет (*preference*) данного маршрута. *Preference* является в BIRD универсальной метрикой, применимой для сопоставления маршрутов, полученных из любых источников.
— IP-адрес маршрутизатора, от которого получена данная информация.
— IP-адрес шлюза, которому следует пересылать пакеты, адресованные в данную подсеть.
— Другие атрибуты, общие для всех маршрутов (например, название протокола, по которому данный маршрут был получен).
— Динамические атрибуты, специфические для каждого из протоколов маршрутизации (например, метрику маршрута в рамках данного протокола).

Для одной сети в маршрутной таблице BIRD может быть несколько записей, но не более одной записи с одной и той же подсетью и одним и тем же протоколом. Для отправки пакетов выбирается маршрут, имеющий наибольший приоритет (*selected route*) . Если в BIRD имеется информация о нескольких маршрутах в одну и ту же сеть, полученная по одному и тому же протоколу, то сам протокол выбирает из них оптимальный (как правило, по своей локальной метрике) до экспорта в маршрутную таблицу BIRD. Если маршруты получены по разным протоколам, то для выбора используются глобальные приоритеты (параметр preference).

Каждый протокол соединяется с маршрутной таблицей BIRD посредством двух фильтров. Каждый из этих фильтров может пропускать (*accept*), не пропускать (*deny*) или модифицировать маршруты по установленным правилам. Фильтр для экспорта передаёт маршруты из маршрутной таблицы в протокол, фильтр для импорта — из протокола в маршрутную таблицу. Когда маршрутная таблица получает от какого-либо протокола новый маршрут, она заново вычисляет действующий маршрут (*selected route*) и рассылает его во все протоколы, подключённые к ней. После этого протоколы рассылают свои обновления маршрутов другим маршрутизаторам в сети.

Список таблиц BIRD создаётся в узле .ip.dynamic-routing.table . В системе всегда неявно существует одна таблица; другие могут добавляться по мере необходимости и обязательно должны иметь имена, по которым они будут идентифицироваться. В описаниях каждой копии каждого протокола указывается, с какой из таблиц она работает (если не указано — то, соответственно, с основной, неявно существующей, таблицей).

§3.3.3. Общие настройки BIRD и протоколов

К глобальным настройкам BIRD относится, в первую очередь, идентификатор данного маршрутизатора, необходимый для работы ряда протоколов. Идентификатор — 32-битное число, которое устанавливается параметром router-id . Для удобочитаемости принято записывать его в десятично-дотовом формате, подобно IP-адресу. В общем случае, это может быть произвольное число, но по здравому смыслу логично использовать в качестве идентификатора один из IP-адресов данного устройства (по крайней мере, в этом случае можно быть уверенным в его уникальности). Если идентификатор не назначен явно, то в этом качестве используется наименьший из всех адресов IPv4, присвоенных интерфейсам данного маршрутизатора, за исключением интерфейса lo . Для IPv6 данный параметр является обязательным.

Помимо глобального идентификатора, для любой копии любого протокола может быть установлен отдельно свой собственный идентификатор устройства. В этом случае он имеет приоритет над глобальным.

Чтобы использовать подсистему динамической маршрутизации, необходимо настроить хотя бы один из входящих в неё протоколов или псевдо-протоколов. Для каждого протокола в системе может быть создано несколько независимых копий (как минимум, одна, в противном случае протокол не будет работать) для работы с разными интерфейсами, областями маршрутизации и/или маршрутными таблицами. Каждая копия автоматически получает в системе имя в формате протоколНомер (однако номера упорядочиваются автоматически, поэтому имена копий могут меняться при переконфигурировании). Для удобства администрирования в каждой копии предусмотрено поле description — её текстовое описание.

Включить BIRD, как целое, можно параметром enable. По умолчанию подсистема выключена и не содержит никаких протоколов. При создании любой копии любого протокола она автоматически включается с самыми общими настройками, принятыми по умолчанию; отключить конкретную копию можно параметром disable внутри его узла конфигурационного дерева.

Каждая копия каждого протокола должна быть связана с некоторой таблицей маршрутизации BIRD. Для этого используется параметр table внутри данной копии. Он должен указывать на таблицу из числа созданных в узле .ip.dynamic-routing.table . Если параметр не указан или таблица с таким именем не существует, то данная копия связывается с основной таблицей BIRD, неявно существующей всегда.

Приоритет маршрутов, полученных с помощью данной копии, по сравнению с маршрутами, полученными из других протоколов, устанавливается параметром preference (бóльшие значения соответствуют более предпочтительным маршрутам). Если приоритеты не указаны явно, то, по умолчанию, используется следующий список приоритетов между протоколами, заложенный в BIRD:

Directly connected	240
Static	200
OSPF intra-area, inter-area и внешние с метрикой тип 1	150
RIP	120
BGP	100
Полученные из других таблиц	70
Полученные из других источников	10

Для передачи маршрутов из протокола в таблицу BIRD служит поле import, в обратном направлении — export . Поле может содержать ключевое слово all или none, имя фильтра, описание фильтра или выражение фильтра (*filter expression*). По умолчанию, для фильтров импорта установлено значение all, для фильтров экспорта — none .

Любые дополнительные параметры BIRD, не предусмотренные явно в конфигурационном дереве NSG, могут быть заданы с помощью полей extra в узле dynamic-routing или в описаниях конкретных копий протоколов, соответственно.

Список интерфейсов, на которых работает данная копия протокола, со специфическими опциями для этого протокола на этих интерфейсах, задаётся в узле interface . Именем каждого элемента списка является один или несколько *шаблонов* интерфейсов (в терминологии BIRD — *clauses*):

[-] ["маска"] [префикс] [, ...]

Каждая запись в данном списке предназначена для описания интерфейсов, имеющих одинаковые параметры, и может содержать один или несколько *шаблонов*. Шаблон может содержать маску имени интерфейса (допускаются подстановочные символы *, ? и \), префикс, или то и другое. Интерфейс соответствует шаблону, если он совпадает по маске (если она задана) и по префиксу (если задан) одновременно. Для IPv6 используется только маска. Маска и префикс разделяются пробелом, шаблоны отделяются друг от друга запятыми.

ВНИМАНИЕ В синтаксисе BIRD, если указывается имя или маска имени интерфейса, то оно должно быть заключено в двойные кавычки. В связи с особенностями синтаксиса командного языка NSG Linux 2.0, вводить их необходимо с использованием *escape*-символа \, а затем брать всю запись в двойные кавычки уже без \, например:

"\"eth*\" 192.168.1.0/24, \"br*\""

Если все шаблоны состоят только из префиксов, то такую запись можно вводить без кавычек.

Знак "-" исключает из списка интерфейсы, соответствующие данному шаблону. Список шаблонов интерпретируется слева направо до первого совпадения интерфейса с одним из шаблонов. Например, запись

"eth0", -"eth*", "*"

содержит 3 шаблона (первый и третий включительно, второй исключительно) и означает eth0 и все интерфейсы, имя которых не начинается с символов eth.

Записи интерпретируются для каждого конкретного интерфейса в порядке нумерации. Если в списке содержится несколько записей с разными параметрами, то к интерфейсу применяется первая запись, которой он будет соответствовать.

Список интерфейсов содержится в описаниях каждой копии протоколов Direct, RIP и RAdv, а также в описаниях каждой области (*area*) протокола OSPF, и является обязательным параметром. По умолчанию, список пустой, т.е. без явно заданного списка эти протоколы функционировать не могут.

Примеры описания интерфейсов (из документации BIRD):

```
interface
: "*"                          (вводится как "\"*\"")
: : type = "broadcast"
```

Протокол (OSPF) работает на всех интерфейсах данного устройства с опцией type broadcast .

```
interface
: "eth1", "eth4", "eth5"       (вводится как "\"eth0\", \"eth4\", \"eth5\"")
: : type = "ptp"
```

Протокол (OSPF) работает на трёх перечисленных интерфейсах с опцией type ptp .

```
interface
: -192.168.1.0/24, 192.168.0.0/16
```

Протокол работает на всех интерфейсах с адресами из диапазона 192.168.0.0/16, за исключением более узкого диапазона 192.168.1.0/24.

```
interface
: "eth*" 192.168.1.0/24        (вводится как "\"eth*\" 192.168.1.0/24")
```

Протокол работает на всех интерфейсах Ethernet с адресами из диапазона 192.168.1.0/24.

ВНИМАНИЕ В заключение необходимо во всех случаях настроить протокол Kernel и в его рамках — экспорт из таблиц BIRD в маршрутные таблицы ядра Linux. Без этого никакие маршруты, известные BIRD, не будут использованы в реальности.

§3.3.4. Протокол Static

Static в терминах BIRD — это псевдопротокол, предназначенный для ручного создания статических маршрутов в таблицах BIRD. Несмотря на внешнее сходство, он имеет более широкие возможности по сравнению с прямым указанием маршрутов в таблицах ядра (узел .ip.route). С его помощью могут быть описаны следующие специфические маршруты (тип маршрута определяется параметром action):

— Маршруты через заданный шлюз (via-gateway).
— Маршруты через заданный интерфейс устройства NSG (via-interface).
— Многовариантные маршруты (*equal cost multipath* в терминологии Linux, или *load balancing in per-session mode* в продуктах некоторых производителей). Для маршрута такого типа создаётся список из произвольного числа шлюзов; дополнительно, каждому шлюзу можно назначить "вес", пропорционально которому будет распределяться нагрузка между шлюзами.
— Тупиковые маршруты ("посылка в *null*"). Такие маршруты требуются на тот случай, если адрес назначения пакета находится в вашем адресном блоке, но маршрут к нему неизвестен (например, такой хост вообще не существует), и в то же время нежелательно возвращать пакет по маршруту по умолчанию (иначе он так и будет бессмысленно циркулировать до истечения TTL). Тупиковые маршруты могут быть 3 типов:

drop	Молча уничтожить пакет.
reject	Уничтожить пакет и отправить в ответ сообщение ICMP Destination Unreachable.
prohibit	Уничтожить пакет с сообщением Administratively Prohibited.

Если указанная в маршруте сеть недоступна в данный момент (интерфейс находится в состоянии DOWN, или указанный шлюз не найден в списке смежных узлов), то протокол Static просто удаляет маршрут из таблицы BIRD и добавляет его снова, как только это будет возможно.

Маршруты, созданные протоколом Static, не имеют никаких специфических атрибутов.

Пример конфигурации (из документации BIRD):

```
static
: 1
: : table              = "testable"              # Connect to a non-default routing table
: : route
: : : 0.0.0.0/0                                  # Default route
: : : : action         = "via-gateway"
: : : : : gateway      = "62.168.0.13"
: : : 10.0.0.0/8
: : : : action         = "via-multipath"         # Multipath route
: : : : : multipath
: : : : : : 1
: : : : : : : gateway  = "62.168.0.14"
: : : : : : : weight   = 2
: : : : : : 2
: : : : : : : gateway  = "62.168.1.10"
: : : : : : 3
: : : : : : : gateway  = "62.168.1.11
: : : 62.168.0.0/25
: : : : action         = "reject"                # Sink route
: : : 10.2.0.0/24
: : : : action         = "via-interface"
: : : : : interface    = "ppp1"                  # Secondary network
```

§3.3.5. Протокол Device

Псевдо-протокол Device запускается в единственной копии и не создаёт никаких маршрутов. Это модуль, передающий из ядра Linux в BIRD информацию о существующих сетевых интерфейсах и их состояниях. Как правило, он должен быть включён, поскольку работа всех остальных протоколов опирается именно на эту информацию.

Протокол имеет два вспомогательных параметра: интервал опроса ядра (scan-time) и список сетей (primary), адреса из которых BIRD следует считать первичными, если интерфейс имеет более одного адреса; список может относиться к определённой маске имени интерфейса. Настройка их требуется только в специфических случаях.

Пример конфигурации. В отличие от BIRD, при явном указании одиночного IP-адреса его необходимо вводить всё равно как префикс с маской /32:

```
device
: scan-time            = 10
: primary
: : 1                  = "\"eth0\" 192.168.1.1/32"
: : 2                  = "192.168.0.0/16"
```

ВНИМАНИЕ Принудительный выбор адреса primary необходим, в частности, в том случае, если какое-либо другое приложение (например, *ui*TCP) манипулирует IP-адресами интерфейсов. Если при этом удаляется адрес, который был выбран BIRD в качестве первичного, то BIRD рестартует интерфейс и выбирает из оставшихся адресов новый первичный. То же самое происходит, если добавляется адрес, который должен был бы быть выбран BIRD в качестве первичного по умолчанию. Рестарты интерфейса, вероятнее всего, нежелательны для работы этого приложения и системы в целом. Чтобы избежать их, следует назначить интерфейсу формальный адрес, который не будет динамически назначаться и удаляться, и указать именно его в качестве первичного. (Или выбрать первичный адрес из числа реальных адресов интерфейса, сконфигурированных статически.)

§3.3.6. Протокол Direct

Протокол Direct генерирует в таблицах BIRD маршруты в непосредственно подключённые сети.

В большинстве случаев протокол Direct является излишним и использовать его не следует, потому что нет необходимости держать эти маршруты в таблицах BIRD. Механизм маршрутизации в ядре операционной системы самостоятельно генерирует эти маршруты и вносит их непосредственно в свои маршрутные таблицы, и экспортировать их из таблиц BIRD в таблицы ядра уже не требуется. Протокол OSPF также самостоятельно генерирует эти маршруты, а BGP экспортирует, как правило, агрегированные маршруты. Включать протокол Direct следует только в некоторых специфических конфигурациях (например, когда eBGP используется, вопреки своему предназначению, в качестве IGP для маршрутизации внутри автономной системы).

Список интерфейсов, за которыми должен следить данный протокол, содержится в узле interfaces. По умолчанию, Direct работает со всеми интерфейсами устройства; явно создавать этот список нужно только в том случае, если нужно выбрать только часть интерфейсов (например, определённые интерфейсы для определённой таблицы BIRD).

Маршруты, созданные протоколом Direct, не имеют никаких специфических атрибутов.

Пример конфигурации:

```
direct
: 1
: : interface
: : : "\"gre*\" = false                    # исключить туннели GRE
: : : "\"*\""    = true
```

§3.3.7. Протокол RIP2

При использовании протокола RIP каждый маршрутизатор регулярно (а также по особому запросу вне расписания) рассылает своим соседям оповещения, содержащие расстояние (измеренное в числе шагов маршрутизации) до всех известных ему сетей. Сосед увеличивает расстояние на единицу (если говорить точнее, то на метрику интерфейса), вносит эти маршруты в свою таблицу и рассылает их всем своим соседям (в т.ч. и предыдущему, но тот игнорирует их, потому что уже знает маршруты от себя напрямую). Максимальная длина маршрута в RIP составляет 15 шагов маршрутизации (*hops*).

Если какая-то сеть становится недоступной, то маршрутизаторы продолжают пересылать друг другу маршруты, каждый раз увеличивая метрику. В конце концов метрика достигает предельной величины 16, означающей в терминах RIP бесконечность, или недоступную сеть. Процедура сходится довольно медленно, и именно поэтому максимальная длина маршрутов ограничена таким небольшим числом.

RIP — самый простой из протоколов динамической маршрутизации, но он имеет существенные недостатки: медленную сходимость, ограниченный размер сети, большой объём служебного трафика (каждый раз рассылается маршрутная таблица полностью). Последнее делает использование RIP весьма дорогостоящим, например, в сотовых сетях). Тем не менее, RIP до сих пор может быть актуален в относительно простых и широкополосных инсталляциях.

Реализация RIP в BIRD и NSG Linux 2.0 поддерживает RIPv2 в применении к IPv4 и IPv6, а также аутентификацию сообщений с помощью MD5. Устаревшая версия RIPv1 не поддерживается.

Для минимальной настройки RIP необходимо создать одну копию протокола и определить интерфейсы, на которых она работает.

Маршруты, полученные по RIP, имеют два специфических атрибута, которые могут быть использованы в фильтрах экспорта-импорта:

rip_metric Локальная метрика маршрута в рамках RIP — целое число от 1 до 16 (условной бесконечности в RIP). Если в таблице BIRD имеются маршруты в одну и ту же сеть назначения от различных копий RIP с равными приоритетами, то будет выбран маршрут с наименьшей RIP-метрикой. При импорте в RIP маршрутов, полученных от других протоколов, им по умолчанию назначается RIP-метрика 5.

rip_tag Тег RIP — 16-битное число. Может быть использован для передачи дополнительной информации о маршруте (например, для внешних маршрутов — номера автономной системы-источника). Для маршрутов, импортированных в RIP из других протоколов, тег равен 0.

Пример конфигурации (из документации BIRD):

```
rip
: 1
: : debug              = "all"
: : interface
: : : "eth0"
: : : : metric         = 3
: : : : mode           = "multicast"
: : : "eth*"
: : : : metric         = 2
: : : : mode           = "broadcast "
: : honor              = "neighbor"
: : authentication     = "none"
: : import             = "filter { print "importing"; accept; }"      # Выводится в stdout, т.е. в консольный порт
: : export             = "filter { print "exporting"; accept; }"
```

§3.3.8. Протокол OSPF

Open Shortest Path First (OSPF) относится к протоколам внутренней маршрутизации (*interior gateway protocols*, IGP), т.е. предназначен для использования в пределах одной автономной системы. С помощью этого протокола каждый маршрутизатор поддерживает свою копию базы данных, описывающей топологию автономной системы. При штатной работе протокола все маршрутизаторы в системе имеют одинаковые копии этой базы данных и исполняют одинаковый алгоритм вычисления маршрутов от себя до других узлов. В качестве действующего выбирается маршрут с наименьшей длиной (или стоимостью, в терминах некоторой метрики). BIRD и NSG Linux 2.0 поддерживают версии OSPFv2 для сетей IPv4 и OSPFv3 для сетей IPv6, а также аутентификацию сообщений с помощью MD5.

Чтобы уменьшить потребление вычислительных ресурсов, OSPF предусматривает возможность разбиения автономной системы на зоны (*area*), каждая из которых обрабатывается независимо от других. Топология одной зоны остаётся скрытой от остальной части AS. Это также предотвращает распространение некорректной маршрутной информации из одной зоны в другие.

Важная особенность OSPF — его способность учитывать маршрутную информацию, полученную из других протоколов (например, Static или BGP), в качестве внешних маршрутов. Узел, рассылающий внешний маршрут, может пометить его специальным тегом, что позволяет организовать обмен дополнительной информацией между маршрутизаторами на границе AS.

OSPF отличается быстрым обнаружением изменений в топологии AS (например, отказов интерфейсов маршрутизатора) и хорошей сходимостью, т.е. новые маршруты пересчитываются за относительно короткое время. При этом генерируется минимальный объём служебного трафика, поскольку рассылаются не маршруты целиком, а только изменения в базе данных OSPF. Это делает OSPF предпочтительным (по сравнению, в частности, с RIP) для дорогостоящих и/или низкоскоростных каналов связи, таких как сотовые соединения.

Каждый маршрутизатор, на котором исполняется OSPF, периодически рассылает со всех своих интерфейсов сообщения Hello. С их помощью смежные маршрутизаторы могут динамически обнаруживать друг друга. После этого соседи обмениваются своими частями топологической базы данных и далее продолжают синхронизировать свои копии, рассылая соседям все изменения в ней. Принцип несанкционированной рассылки (*flooding*) с каждого узла прост и надёжен и гарантирует, что каждый из маршрутизаторов рано или поздно получит информацию обо всех изменениях.

Настройка OSPF отличается от других протоколов, в первую очередь, тем, что в каждой копии протокола необходимо создать зоны (area). Идентификатором зоны является 32-битное число, которое в NSG Linux 2.0 следует записывать в десятично-дотовом формате, подобно IP-адресу. (В частности, это может быть адрес одного из интерфейсов маршрутизатора, подключённого к данной зоне, что автоматически гарантирует уникальность идентификатора.) Большинство настроек выполняется отдельно внутри каждой зоны; вне зон настраиваются только некоторые общие параметры для данной копии протокола. Внутри зоны содержится описание сетей и/или интерфейсов, входящих в неё, а также ряд других параметров.

OSPF определяет 4 атрибута для маршрутов:

ospf_metric1
ospf_metric2 Каждый внутренний маршрут имеет метрику в диапазоне от 1 до 65535 (бесконечность). Для внешних маршрутов может использоваться метрика типа 1 или типа 2. Метрика типа 1 и метрика внутренних маршрутов хранятся в одном атрибуте и могут сравниваться друг с другом. Метрика типа 2 всегда больше, чем любая метрика типа 1 или внутренняя. Если указаны обе метрики, то используется только метрика типа 1. Значение по умолчанию ospf_metric2 = 10000.

ospf_tag Тег OSPF — 32-битное число. Используется только при экспорте маршрутов в другие протоколы; непосредственно на маршрутизацию в OSPF-домене не влияет. Значение по умолчанию 0.

ospf_router_id
Идентификатор маршрутизатора, объявляющего о данном маршруте или сети. Используется только для чтения.

Пример конфигурации (из документации BIRD):

```
ospf
: 1
: : rfc1583compat          = true
: : tick                   = 2
: : export                 = "filter { if source = RTS_BGP then { ospf_metric1 = 100; accept; } reject; }"
: : area
: : : 0.0.0.0
: : : : interface
: : : : : "eth*"
: : : : : : cost           = 11
: : : : : : hello          = 15
: : : : : : priority       = 100
: : : : : : retransmit     = 7
: : : : : : authentication = "simple"
: : : : : : password       = "aaa"
: : : : : "ppp*"
: : : : : : cost           = 100
: : : : : : authentication = "cryptographic"
: : : : : : password       = "def"
: : : : : "gre1"
: : : : : : cost           = 10
: : : : : : stub           = true
: : : : : "gre2"
: : : 0.0.0.120
: : : : stub-cost          = 50
: : : : networks
: : : : : 172.16.1.0/24
: : : : : 172.16.2.0/24
: : : : : : hidden         = true
```

§3.3.9. Протокол BGP

Border Gateway Protocol предназначен для управления маршрутизацией на уровне сетевых магистралей. В отличие от других протоколов, для сходимости этого протокола не требуется, чтобы все маршрутизаторы использовали для выбора маршрутов одни и те же правила. Таким образом, на любом маршрутизаторе в сети может использоваться любая политика маршрутизации. Единственное обязательное требование состоит в том, что если маршрутизатор объявляет некоторый маршрут, то он должен принимать и обрабатывать пакеты в соответствии с этим маршрутом.

Работа BGP опирается на понятие *автономной системы* (AS). Каждая AS представляет собой часть сети с общим управлением и общей политикой маршрутизации и идентифицируется уникальным номером (ASN, изначально 16-битным, но в современных системах — 32-битным). Как правило, маршрутизаторы внутри одной AS обмениваются друг с другом маршрутной информацией при помощи какого-либо из протоколов внутренней маршрутизации (*interior gateway protocol*, IGP), например, OSPF или RIP. Граничные маршрутизаторы, соединяющие данную AS со смежными, обмениваются глобальной маршрутной информацией (между AS) друг с другом при помощи внешнего BGP (exterior BGP, eBGP) и распространяют эту информацию внутри своей AS при помощи внутреннего BGP (interior BGP, iBGP).

Каждый BGP-маршрутизатор рассылает своим соседям обновления своей маршрутной таблицы (в той части, в какой он должен её экспортировать) вместе с полной информацией о маршруте, т.е. списком AS, через которые пройдёт пакет, если он будет послан по данному маршруту. Такой алгоритм позволяет избежать зацикливания маршрутов.

Пакет BIRD и NSG Linux 2.0 поддерживают все требования стандарта BGP4, а также атрибуты сообществ (*community attributes*), согласование возможностей (*capability negotiation*), аутентификацию MD5, репликаторы маршрутов (*route reflectors*), мультипротокольные расширения для IPv4 и IPv6, и 32-битные номера AS.

Для выбора оптимального маршрута в BIRD используется более сложный алгоритм, нежели одно только сравнение метрик. Если в протоколе имеется несколько маршрутов в одну и ту же сеть с одинаковым приоритетом (*preference*), то они последовательно пропускаются через цепочку правил до тех пор, пока из них не останется один:

— Выбирается маршрут с наибольшим локальным приоритетом (Local Preference).
— Выбирается маршрут с кратчайшим путём в терминах AS (shortest AS path).
— Маршрут, полученный от IGP, имеет приоритет над полученным от EGP, а полученный от EGP — над маршрутом из неизвестного источника (*incomplete*).
— Выбирается маршрут с минимальным значением атрибута Multiple Exit Discriminator.
— Маршрут, полученный от eBGP, имеет приоритет над полученным от iBGP.
— Выбирается маршрут с минимальным внутренним расстоянием до граничного маршрутизатора.
— Если все предшествующие правила не дали однозначного ответа, то выбирается маршрут с наименьшим значением идентификатора (Router ID) того маршрутизатора, которым он создан.

Основная задача BGP — контролировать глобальную связность сетей и маршруты к другим автономным системам. Когда эти маршруты распространяются с помощью BGP между маршрутизаторами внутри AS, то в них указывается дополнительный атрибут NEXT_HOP — адрес соответствующего граничного маршрутизатора. Для выбора маршрутов BGP протокол опирается на имеющуюся информацию о внутренних маршрутах AS, которая содержится в таблице IGP. По ней определяются непосредственные следующие шлюзы для каждого маршрута и внутренние расстояния до граничных маршрутизаторов. В BIRD и NSG Linux 2.0 обычно используется одна и та же таблица маршрутизации для IGP- и BGP-маршрутов.

Каждая копия BIRD соответствует одному смежному маршрутизатору. Это позволяет установить политики маршрутизации и другие параметры индивидуально для каждого соседа. Обязательными параметрами конфигурации являются номер своей автономной системы, номер смежной автономной системы и IP-адрес соседа.

BGP использует для каждого маршрута большое число атрибутов, причём часть из них относится только к внутренним соединениям BGP. Некоторые атрибуты являются необязательными.

bgp_path — Последовательность номеров AS, в том порядке, в каком их будет проходить пакет, посланный по данному маршруту. Для внутренних BGP-маршрутов путь не содержит номера локальной AS.

bgp_local_pref
: Только для внутренних маршрутов: локальный приоритет, используемый в пределах BGP для выбора оптимального маршрута (из нескольких равноценных по более существенным критериям, см. выше). По существу, служит внутренней метрикой, распространяемой в пределах всей локальной AS.

bgp_med — (Опциональный) Multiple Exit Discriminator — дополнительный атрибут, используемый на внешних соединениях (т.е. соединениях между автономными системами) для того, чтобы указать соседу оптимальную точку входа в локальную AS. Значение этого атрибута, полученное от соседа, также распространяется по внутренним соединениям AS. При экспорте маршрутов из таблицы BIRD во внешнюю копию BGP данный атрибут устанавливается в 0, чтобы значения, полученные от одной смежной AS, не распространялись к другим соседям. При этом в фильтре экспорта может быть явно указано новое значение.

bgp_origin — Источник информации о маршруте. Может принимать одно из следующих значений:

ORIGIN_IGP	Маршрут создан одним из IGP-протоколов.
ORIGIN_EGP	Маршрут импортирован от одного из EGP-протоколов (де-факто это значение не имеет особого смысла).
ORIGIN_INCOMPLETE	Источник происхождения маршрута неизвестен.

bgp_next_hop
: IP-адрес следующего шлюза, которому следует передавать пакеты по данному маршруту. Для внутренних соединений BGP это будет либо адрес маршрутизатора, объявляющего данный маршрут — если этот маршрутизатор находится внутри AS, либо адрес граничного маршрутизатора, через который этот пакет должен покинуть данную AS — если это внешний маршрут. Таким образом, каждый источник (*speaker*) BGP-маршрутов может определить кратчайший путь внутри локальной AS к данной точке.

bgp_atomic_aggr
: (Опциональный). Пустой атрибут, не имеющий никакого значения. Сам факт присутствия этого атрибута сигнализирует о том, что данный маршрут был агрегирован из нескольких каким-то промежуточным узлом по пути от источника маршрута к данному устройству.

bgp_community
: (Опциональный). Список значений сообществ (BGP *comminuties*), ассоциированных с данным маршрутом. Каждое значение представляет собой пару 16-битных чисел и используется в фильтрах как тип данных pair. Первое число содержит номер AS, определяющей данное сообщество, а второе — идентификатор, имеющий некоторый смысл в пределах этой AS. Механизм сообществ может использоваться различными способами, но чаще всего он применяется для передачи дополнительной информации, например, о политике экспорта маршрутов для того или иного соседа. Поскольку каждая AS вправе самостоятельно определять свою политику, она также вправе самостоятельно определять атрибуты сообщества и их семантику. (За исключением нескольких стандартных сообществ, определённых в RFC–1997.)

bgp_originator_id
: (Только для внутренних маршрутов, опциональный). 32-битное число в десятично-дотовой нотации. Данный атрибут создаётся репликатором маршрутов и содержит Router ID маршрутизатора, являющегося оригинальным создателем данного маршрута, в локальной AS.

bgp_cluster_list
: (Только для внутренних маршрутов, опциональный). Список идентификаторов кластеров (Cluster ID) для репликаторов маршрутов. Каждый репликатор маршрутов указывает в реплицируемом маршруте свой идентификатор кластера.

Пример конфигурации (из документации BIRD, с нетривиальным фильтром экспорта):

```
bgp
: 1
: : local-as          = 65000                 # Use a private AS number
: : neighbor-ip       = "62.168.0.130"        # Our neighbor ...
: : neighbor-as       = 5588
: : multihop
: : : enable          = "on"                  # ... which is connected indirectly
: : export            = "filter { if source = RTS_STATIC then { bgp_community.add((65000,5678));
bgp_path ~ [= 65000 =] then bgp_path.prepend(65000); accept; } reject; }
: : import            = "all"
: : source-address    = "62.168.0.1"          # Use a non-standard source address
```

§3.3.10. Протокол Router Advertisement

Поддержка IPv6 и связанных с ним функций в NSG Linux 2.0 находится в разработке.

§3.3.11. Протокол Pipe

Псевдо-протокол Pipe осуществляет обмен маршрутами между двумя таблицами BIRD. Настройки протокола выполняются применительно к одной из таблиц, которая является для него первичной (table); помимо неё, обязательным параметром является вторичная таблица (peer-table). Направление операций экспорта-импорта подразумевается по отношению к первичной таблице, т.е. перенос маршрутов из первичной таблицы во вторичную называется экспортом, из вторичной в первичную — импортом.

Протокол имеет два режима работы: прозрачный и непрозрачный. Различие состоит в том, как передаются маршруты в одни и те же сети, если их несколько. В непрозрачном режиме из одной таблицы в другую передаётся только один оптимальный маршрут, так же, как и при работе других протоколов. В качестве создателя этого маршрута указывается протокол Pipe (что может ограничить доступ к атрибутам маршрута, которые являются специфическими для других протоколов). Непрозрачный режим используется по умолчанию.

В прозрачном режиме протокол Pipe ретранслирует из одной таблицы в другую все имеющиеся маршруты, с сохранением их оригинального создателя и других атрибутов. Если при этом оба фильтра (экспорта и импорта) имеют простейшую настройку "пропускать все маршруты без модификации", то содержимое обеих таблиц будет идентичным.

Основное назначение множественных таблиц и протокола Pipe состоит в обеспечении маршрутизации на основе совокупности критериев (*policy routing*), при которой решение относительно дальнейшей передачи пакета принимается на основе не только его адреса назначения, но также адреса источника, входного интерфейса, протокола 4 уровня и других параметров. Ядро Linux позволяет реализовать такие политики с помощью множественных таблиц маршрутизации, выбор между которыми производится именно на основе этих расширенных критериев. Правила для такого выбора (*routing rules*) устанавливаются вне рамок BIRD; для того, чтобы отразить эту многозначность в BIRD, следует создать несколько таблиц BIRD, связать их с соответствующими таблицами ядра и управлять передачей маршрутов из одной таблицы в другую при помощи фильтров. При этом Pipe используется для передачи выбранного подмножества маршрутов из одной таблицы BIRD в другую.

Маршруты, созданные протоколом Pipe, не имеют никаких специфических атрибутов.

Пример (из документации BIRD). Маршрутизатор является граничным между двумя автономными системами. Каждая AS подключена к своей группе интерфейсов и имеет свой собственный выход во внешний мир. Требуется использовать каждую из систем в качестве резервного выхода для другой на случай, если у той откажет штатный канал связи с внешним миром.

Простейшее решение состоит в том, чтобы создать две таблицы маршрутизации (as1 и as2) и настроить правила маршрутизации в ядре Linux (*в данной версии — только средствами ОС Linux*) таким образом, чтобы пакеты, входящие с интерфейсов первой AS, маршрутизировались по таблице ядра 1, а с интерфейсов второй AS — по 2. Таким образом, маршрутизатор, по существу, разделяется на два логических устройства, каждое из которых имеет свою таблицу маршрутизации и использует свой собственный набор протоколов маршрутизации на своём собственном наборе интерфейсов. Для того, чтобы добавить в эту таблицу маршруты через другую автономную систему как резервные, они передаются между таблицами BIRD по протоколу Pipe, при этом понижается их приоритет и корректируются пути BGP в соответствии с переходом через границу AS.

```
dynamic-routing
: table                                  # Define the tables
: : 1                 = "as1"
: : 2                 = "as2"
: kernel
: : 1                                    # Synchronize them with the kernel
: : : table           = "as1"
: : : kernel-table    = 1
: : 2
: : table             = "as2"
: : kernel-table      = 2
: bgp                                    # The outside connections
: : 1
: : : table           = "as1"
: : : local-as        = "0.0.0.1"
: : : neighbor-ip     = "192.168.0.1"
: : : neighbor-as     = "0.0.3.233"
: : : export          = "all"
: : : import          = "all"
: : 2
: : : table           = "as2"
: : : local-as        = "0.0.0.2"
: : : neighbor-ip     = "10.0.0.1"
: : : neighbor-as     = "0.0.3.234"
: : : export          = "all"
: : : import          = "all"
: pipe                                   # The Pipe
: : 1
: : : table           = "as1"
: : : peer-table      = "as2"
: : : export          = "filter { if net ~ [ 1.0.0.0/8+] then { if preference>10 then preference = preference-10; ↵
if source=RTS_BGP then bgp_path.prepend(1); accept; } reject; }"
: : : import          = "filter { if net ~ [ 2.0.0.0/8+] then { if preference>10 then preference = preference-10; ↵
if source=RTS_BGP then bgp_path.prepend(2); accept; } reject; }"
```

§3.3.12. Протокол Kernel

Псевдо-протокол Kernel осуществляет синхронизацию маршрутных таблиц BIRD и ядра операционной системы. В конфигурации протокола, помимо таблицы BIRD (table), обязательным параметром является соответствующая ей таблица ядра (kernel-table). По существу, он выполняет две задачи:

— Передаёт все изменения, произведённые в таблицах BIRD, в таблицы ядра, по которым непосредственно производится маршрутизация пакетов.

— Через определённые интервалы времени (scan-time) считывает маршрутные таблицы ядра и сравнивает их с таблицами BIRD на предмет возможных расхождений. Например, в ядре могут исчезать и появляться какие-то маршруты из-за перехода интерфейсов UP/DOWN, и эти переходы могли остаться незамеченными для BIRD. Или же маршрутная таблица ядра могла быть модифицирована какими-либо другими механизмами (в частности, в устройствах NSG такими механизмами являются *netping* и *ui*TCP); такие изменения учитываются или не учитываются в таблицах BIRD в зависимости от параметра learn .

При синхронизации маршрутных таблиц сеть одна сложность. В таблицах ядра всегда присутствуют маршруты в непосредственно подключённые сети. Этими маршрутами операционная система управляет самостоятельно (как составная часть управления IP-адресами интерфейсов) и от BIRD не требуется вмешиваться в этот процесс. Поэтому при сканировании таблиц ядра (импорте в BIRD) такие маршруты игнорируются безусловно (если это всё-таки требуется, то для этой цели существует отдельный псевдо-протокол Direct). При экспорте из BIRD они, по умолчанию, игнорируются тоже, во избежание конфликтов, но их экспорт можно явным образом разрешить с помощью параметра device-routes .

Поскольку ОС Linux поддерживает множественные таблицы маршрутизации (*в данной версии NSG Linux 2.0 управление ими не поддерживается в командных оболочках NSG*), то в системе могут работать одновременно несколько копий протокола Kernel, но при соблюдении обязательного условия: каждая копия должна быть связана со своей собственной таблицей BIRD и со своей собственной маршрутной таблицей в ядре.

Поскольку протокол Kernel тесно интегрирован с конкретной маршрутной таблицей ядра, он имеет два принципиальных ограничения. Во-первых, с одной и той же таблицей не может быть связана другая копия Kernel. Во-вторых, при экспорте маршрутов невозможно изменение их атрибутов (которое может быть предусмотрено фильтрами в общем случае). То и другое ограничения можно обойти с помощью дополнительной таблицы и протокола Pipe.

Протокол Kernel предусматривает для маршрутов два специфических атрибута, которые взаимно преобразуются в соответствующие атрибуты в ядре Linux:

krt_prefsrc Предпочтительный IP-адрес источника. Используется при генерации исходящих пакетов и должен принадлежать одному из интерфейсов данного устройства.

krt_realm Целое число. Обозначает область применения (*realm*) данного маршрута. Может быть использовано для классификации трафика.

Пример конфигурации (из документации BIRD, для системы с 2 маршрутными таблицами):

```
kernel
: 1                                     # Primary routing table
: : learn           = true              # Learn alien routes from the kernel
: : scan-time       = 10                # Scan kernel routing table every 10 seconds
: : import          = "all"
: : export          = "all"
: 2                                     # Secondary routing table
: : table           = "auxtable"
: : kernel-table    = 100
: : export          = "all"
```

§3.3.13. Протокол BFD

Bidirectional Forwarding Detection (BFD) — это не протокол маршрутизации по существу, а независимый от протоколов инструмент, обеспечивающий контроль за тем, жив ли сосед, и оперативное обнаружение отказов. Протоколы маршрутизации, такие как BGP или OSPF, как правило, имеют для этой цели встроенные механизмы типа Hello, но время срабатывания таких механизмов довольно велико. Например, для OSPF оно составляет, по умолчанию, 40 сек. и не может быть уменьшено далее нескольких секунд. BFD представляет универсальный и быстрый механизм с малыми накладными расходами, который может быть подключён к любому протоколу маршрутизации в качестве вспомогательного инструмента.

Собственно BFD представляет собой совокупность отдельных сессий, мало связанных друг с другом. Каждая BFD-сессия контролирует одноадресный маршрут между двумя маршрутизаторами. Для этого партнёры периодически посылают друг другу управляющие пакеты в обоих направлениях. При этом BFD не занимается обнаружением соседей, но и не требует для них статической конфигурации. BFD-сессии устанавливаются по требованию, по инициативе других протоколов(BGP, OSPF), и эти протоколы предоставляют всю необходимую информацию — IP-адреса, интерфейсы и т.п. Если состояние BFD-сессии меняется, то эти протоколы получают уведомление и соответствующим образом реагируют на него (например, аннулируют факт соседства в OSPF, если BFD-сессия перешла в состояние DOWN).

ПРИМЕЧАНИЕ Для того, чтобы протоколы маршрутизации (OSPF, BGP) использовали механизм BFD, они также должны быть настроены на создание BFD-сессий. Как правило, это делается посредством параметра bfd в конфигурации протокола.

Пакеты BFD посылаются с динамическими адресами портов источника. На Linux-системах, по умолчанию, для этого используется несколько иной диапазон портов, нежели рекомендовано IANA (49152–65535). Настройка этого диапазона возможна средствами ОС Linux, но в данном дереве конфигурации не предусмотрена.

Настройка BFD состоит, в основном, из множественных определений интерфейсов. Большинство настроек BFD относятся к конкретным сессиям. Когда запрашивается новая сессия BFD, она динамически создаётся на основе этих настроек. При этом для сессий с непосредственно подключёнными соседями используются определения интерфейсов, ассоциированных с требуемой сессией. Для сессий, в которых явно указан факт доступности соседа через несколько промежуточных узлов, используются определения из узла multihop. Если для сессии не указано явно ни то, ни другое, то используется конфигурация по умолчанию. По этой причине во многих случаях конфигурация BFD по существу не требуется.

Для отдельных соседей сессия может быть установлена принудительно в узле neighbor.

ПРИМЕЧАНИЕ В отличие от списка интерфейсов в других протоколах (собственно маршрутизации), BFD будет работать и на интерфейсах, не входящих явным образом в данный список, используя настройки по умолчанию.

В отличие от оригинального BIRD, в котором параметры времени BFD могут указываться в секундах, милли- или микросекундах, в пользовательском интерфейсе NSG Linux все они указываются в миллисекундах. Для большинства параметров и практических ситуаций это наиболее адекватный масштаб времени.

ПРИМЕЧАНИЕ Данная реализация BFD в BIRD и, соответственно, в NSG Linux является экспериментальной и находится в состоянии активного развития. В ней реализованы основные функциональные возможности BFD согласно RFC 5880, IP-транспорт для BFD согласно RFC 5881, 5883, и интеграция с клиентскими протоколами согласно RFC 5882. Отдельные расширенные возможности (например, режим эхо или аутентификация) не реализованы и планируются в последующих версиях.
В данной версии BIRD поддерживается только одна копия BFD.

Ввиду экспериментального статуса, возможны некоторые проблемы, а также изменения структуры данной ветви конфигурации в будущем.

§3.3.14. Отладка динамической маршрутизации

BIRD и NSG Linux 2.0 предлагают развитые средства отладки динамической маршрутизации. Как глобально, так и на уровне отдельных копий каждого протокола, можно включить вывод в журнал всей отладочной информации полностью, либо по отдельности:

— изменений состояния протокола
— обмена маршрута между протоколом и таблицей BIRD
— результатов работы фильтров
— изменений состояния интерфейсов, зарегистрированных протоколом
— внутренних событий протокола
— пакетов, принятых и переданных протоколом

Кроме того, глобально для всей системы динамической маршрутизации можно регистрировать в журнале действия пользователя.

Просмотр журнала BIRD по отдельным категориям производится командами в узле .ip.dymanic-routing.show .

§3.4. Фильтрация пакетов и NAT

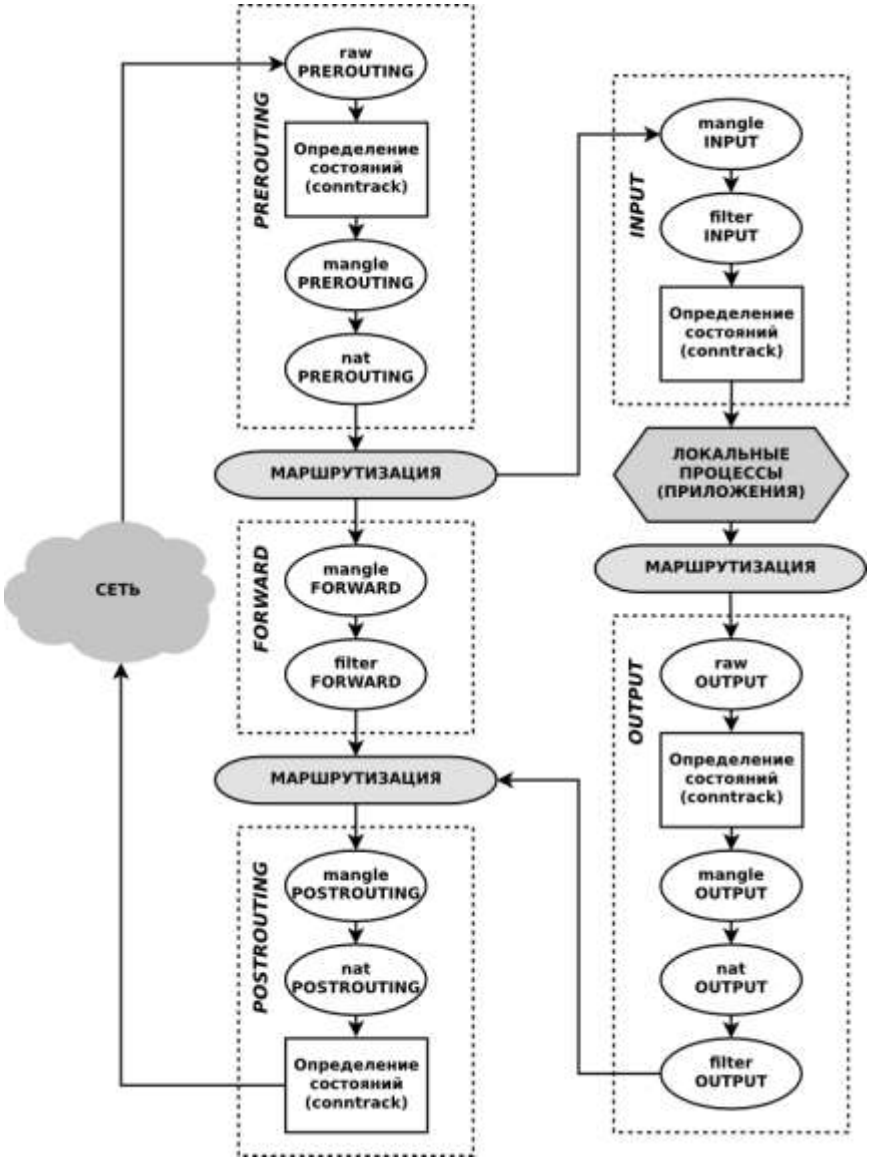
§3.4.1. Общие сведения

Фильтрация пакетов и преобразование сетевых адресов (NAT) в Linux — две тесно связанные процедуры. По сути своей, обе они используют один и тот же набор критериев для анализа пакетов; различие состоит только в действиях, выполняемым над пакетами. По форме, обе эти процедуры реализуются одним и тем же встроенным модулем ядра *netfilter* и настраиваются с помощью утилиты *iptables*.

Средства управления NSG Linux 2.0 — консольная оболочка nsgsh и Web-интерфейс — предлагают упрощённый пользовательский интерфейс для настройки *iptables* в большинстве практических задач. Это освобождает пользователя от необходимости освоения *iptables* и запоминания его опций в полном объёме, но не избавляет от необходимости понимания выполняемых настроек по существу.

Структура узлов .ip.filter и ip.nat в целом соответствует синтаксису командной строки *iptables*. Оба узла содержат наиболее употребительные параметры. Из них автоматически составляется командная строка. Параметры и опции, не представленные явно, в любом случае можно задать в полях extra-options и extra-target (вплоть до того, что вписать в них все опции *iptables* полностью и оставить остальные поля пустыми).

С точки зрения *iptables*, все пакеты разделяются на 3 типа: входящие (адресованные одному из локальных интерфейсов устройства), исходящие (генерируемые одним из интерфейсов устройства) и транзитные (передаваемые с одного стороннего хоста на другой). Правила *iptables* оформляются в виде т.н. *таблиц*, каждая из которых применяется на определённом этапе обработки пакетов того или иного типа. Каждая таблица содержит *цепочки* правил, исполняемых последовательно. Некоторые цепочки предопределены и существуют всегда, другие могут создаваться пользователем. Порядок применения различных таблиц показан на рисунке.



Путь проверки пакета в системе netfilter
Источник: ru.wikipedia.org
Автор Tetromino

Подробно о возможностях *iptables* и её настройках см. соответствующие *man pages*. Настоятельно рекомендуется ознакомиться с этим документом хотя бы наскоро, чтобы получить представление об общей структуре настроек. Весьма содержательное и доходчивое описание *iptables* приведено также в русскоязычном разделе Википедии:

http://ru.wikipedia.org/wiki/Iptables

Если этот документ и не достаточен для исчерпывающего овладения *iptables*, то, по крайней мере, он позволяет увидеть возможности этого механизма и понять, зачем всё так сложно и почему такие вещи, как NAT и фильтры, не могут быть настроены одной кнопочкой [ХОЧУ!] .

Правила фильтрации и NAT, как и правила маршрутизации, по сути своей являются глобальными, т.е. относятся к системе в целом, а не к какому-либо одному её интерфейсу. По этой причине они вынесены в отдельные узлы конфигурации в ветке .ip, а не внутри меню портов. Чтобы ограничить действие правила одним интерфейсом, необходимо и достаточно указать этот интерфейс в критериях фильтрации.

ПРИМЕЧАНИЕ Настройку фильтров следует производить в последнюю очередь, предварительно убедившись в нормальной работе остальных компонент системы.

§3.4.2. Критерии отбора пакетов в *iptables*

Для отбора пакетов, к которым будут применяться те или иные правила обработки, может использоваться широкий набор критериев, включающий практически любую информацию о пакете. В первую очередь, это поля заголовка сетевого уровня:

— IP-адреса источника и назначения пакета. Адреса указываются в полях source и destination в формате одиночного IP-адреса (x.x.x.x) либо префикса сети (x.x.x.x/m).

— Протокол транспортного подуровня, т.е. вложенный непосредственно в IP. Протокол может указываться по имени (поле protocol) или по номеру (protocol-num). Некоторые наиболее важные номера протоколов:

1	ICMP
6	TCP
17	UDP
47	GRE (туннели GRE и PPTP)
50	ESP (IPsec)
51	AH (IPsec)
115	L2TP

Приоритетным из этих двух является поле protocol.

— Протокольно-зависимые параметры:

  — Для протоколов TCP и UDP — номера портов источника и назначения. Указываются в полях source-port и destination port. Можно указывать одиночный порт или диапазон портов в формате nnnn:mmmm.

  — Дополнительно для протокола TCP — флаги TCP-соединения (в данной версии NSG Linux 2.0 указываются в поле extra-options). Учёт этих флагов позволяет организовать фильтрацию в зависимости от наличия и состояния TCP-соединения (т.н. *stateful packet inspection*).

  — Для протокола ICMP — тип пакета (icmp-type).

— Любые другие параметры заголовка (например, значения поля ToS/DiffServ) могут быть указаны в поле extra-options.

Помимо информации, содержащейся в самом пакете, может учитываться другая информация, имеющаяся о нём в системе:

— Имя интерфейса, через который данный пакет попал в систему (поле in-interface).

— Имя интерфейса, через который данный пакет должен быть отправлен в соответствии с таблицей маршрутизации (поле out-interface).

— Маркеры, выставленные ранее для данного пакета предыдущими цепочками (только вручную или через extra-options).

Интерфейсы можно указывать в формате имя+, в этом случае фильтр будет действовать на все интерфейсы, у которых начальная часть имени совпадает с заданной. Например, запись in-interface = "eth+" означает, что фильтр действует на пакеты, входящие в устройство через любой из интерфейсов eth0, eth1 и т.д.

Все критерии можно инвертировать, т.е. отбирать только те, у которых критерий *не* соответствует выбранному значению. Например, запись source-port = "!22" означает, что под действие данного правила будут подпадать все пакеты, *кроме* пакетов SSH.

Помимо основных критериев, перечисленных выше, пакет может проверяться на соответствие большому числу дополнительных, которые можно определить в узле explicit-matches .

Если правило содержит несколько критериев, то они объединяются оператором И, т.е. пакет подпадает под действие правила, если он удовлетворяет всем этим критериям одновременно.

Любой из вышеперечисленных частных критериев может быть опущен, в этом случае данный параметр пакета не рассматривается и считается, что пакет удовлетворяет совокупному критерию при любом значении этого параметра.

§3.4.3. Простые фильтры

Узел меню .ip.filter соответствует таблице filter в *iptables* и содержит три предопределённые цепочки:

INPUT	Для пакетов, адресованных локальным службам данного устройства.
OUTPUT	Для пакетов, генерируемых локальными службами и приложениями на данном устройстве.
FORWARD	Для пакетов, передаваемых с одного стороннего хоста на другой через данное устройство.

Каждая цепочка может содержать неограниченное число правил, создаваемых пользователем. Правила применяются в порядке их нумерации. Каждое правило содержит набор критериев, по которым выбираются пакеты, и действие (поле target), которое должно быть выполнено с этими пакетами. Основные действия, составляющие собственно процесс фильтрации:

ACCEPT	Пропустить пакет для дальнейшей обработки.
DROP	Уничтожить пакет безо всяких уведомлений.
REJECT	Уничтожить пакет, но при этом послать отправителю уведомление о том, что он не может быть доставлен. Тип уведомления (icmp-port-unreachable и др.) можно указать в поле extra-target в формате: --reject-with тип_пакета_ICMP.
RETURN	Остановить применение правил этой цепочки и вернуть пакет в предыдущую (вызывающую) цепочку. Как правило, в простых конфигурациях используется только одна цепочка, и пакет в этом случае передаётся из фильтра в следующую процедуру.

Другие возможные действия *iptables* могут быть определены в поле extra-target вместо target.

Каждая цепочка содержит также действие по умолчанию (default-target), которое определяет, что делать с пакетами, не попавшими под действие ни одного из созданных правил.

ПРИМЕЧАНИЕ Не следует злоупотреблять глобальным значением default-target = "drop", поскольку оно запрещает сразу слишком много пакетов и работа устройства становится невозможной без большого числа явно разрешающих фильтров. Для "параноидального" подхода к фильтрации пакетов обычно бывает более целесообразно установить в конце цепочки запрещающие фильтры на какие-либо определённые интерфейсы (например, на интерфейс, подключённый к сети общего пользования).

§3.4.4. Stateful Packet Inspection и другие сложные фильтры

Правила *iptables* позволяют учитывать флаги TCP, значения поля ToS/DiffServ и другие поля заголовка. В NSG Linux 2.0 эти критерии всегда можно указать в поле extra-options в таком же формате, в каком они записываются в командной строке *iptables*, например, extra-options = "--tos Minimize-Delay" или extra-options = "--tos 16".

Важный частный случай — это фильтрация пакетов TCP с учётом состояния TCP-соединения (*stateful packet inspection*, SPI). Состояние соединения, к которому принадлежит пакет, указывается набором флагов в заголовке TCP. Наиболее частая практическая задача — это запретить установление входящих TCP-соединений из внешнего мира в защищаемую сеть. Для установления новых TCP-соединений хост-инициатор посылает пакет с единственным выставленным флагом SYN. Именно такие пакеты следует запретить, не мешая прохождению всех остальных.

Пример. Пусть, для определённости, порт eth0 подключён к внутренней сети офиса, имеющей реальные IP-адреса (ситуация крайне редкая на сегодняшний день, но теоретически корректная), порт eth4 — к поставщику услуг Интернет. Следующий фильтр:

```
ip
: filter
: : FORWARD
: : : 1
: : : : protocol        = "tcp"
: : : : in-interface    = "eth4"
: : : : extra-options   = "--syn"
: : : : target          = "DROP"
```

запрещает прохождение пакетов TCP с установленным флагом SYN и не установленными другими флагами, полученных через интерфейс eth4. Таким образом, получается эффект "полупрозрачного зеркала": хосты из внешнего мира не могут устанавливать соединения с хостами во внутренней сети офиса, в то время как для соединений в обратном направлении никаких ограничений не устанавливается. (Аналогичная ситуация имеет место, как побочный эффект, при использовании *source NAT* или *masquerading*, что является намного более частой задачей.)

ПРИМЕЧАНИЕ Данный фильтр не запрещает установление TCP-соединений из внешнего мира к самому устройству NSG (в т.ч. Telnet, SSH, HTTP и HTTPS), поскольку такие пакеты маршрутизируются на локальный интерфейс устройства и обрабатываются в цепочке INPUT, а не FORWARD.

§3.4.5. Служебные фильтры

Цепочки ip.filter.INPUT и ip.filter.OUTPUT содержат два служебных фильтра с номером 0, разрешающих приём пакетов с локального адреса 127.0.0.1 и передачу данных на этот адрес. Данные фильтры необходимы для того, чтобы гарантировать обмен пакетами между компонентами NSG Linux внутри устройства. В противном случае ошибки пользователя легко могут привести к тому, что взаимодействие между nsgconfd, nsgsh и Web-интерфейсом, а также между некоторыми другими компонентами, будет нарушено и доступ к устройству будет утрачен безвозвратно. По этой причине данные фильтры являются обязательными и не могут быть удалены пользователем. Они выводятся в общем списке исключительно для сведения пользователя.

§3.4.6. NAT и коммутация IP — общие сведения

Механизм трансляции сетевых адресов и портов (Network Address Translation, NAT) предназначен для преобразования IP-адресов и номеров портов TCP/UDP при прохождении пакетов через маршрутизатор. Как правило, он используется для подключения некоторой ограниченной подсети, адресное пространство которой не соответствует глобальному распределению IP-адресов, ко всему внешнему миру, использующему единое иерархическое адресное пространство. Соответственно, две сети, между которыми производится трансляция адресов, называются обычно *внутренней* и *внешней* (хотя в общем случае это могут быть и две сети с глобальными IP-адресами, и две сети с приватными IP-адресами).

Понятие NAT является собирательным и включает в себя целый ряд различных процедур, по которым производится преобразование IP-адресов и портов при обращении из внутренней сети во внешнюю, из внешней во внутреннюю, или в обоих направлениях. Использование трансляции сетевых адресов решает три важные задачи:

— Позволяет использовать одни и те же IP-адреса во многих подсетях, поскольку адресное пространство каждой внутренней подсети является локальным, а не частью глобального адресного пространства всего Интернет. Уникальными должны быть только внешние IP-адреса, назначенные интерфейсу с поддержкой NAT; эти адреса распределяются централизованно поставщиками услуг Интернет. Во внутренней сети могут использоваться любые адреса, назначенные администратором или, например, унаследованные в ходе какой-либо реструктуризации сети.
— Устраняет нехватку глобальных IP-адресов, присущую протоколу IPv4. За одним или несколькими глобальными адресами может стоять намного большее число хостов внутренней сети.
— Скрывает истинную структуру внутренней сети от внешнего мира и тем самым повышает ее безопасность.

Преобразование адресов, а также коммутация IP-пакетов, могут применяться как ко всему потоку трафика, проходящему через некоторый интерфейс, так и избирательно к пакетам, которые удовлетворяют определенным критериям. Для практических применений наиболее важны следующие механизмы NAT:

— Преобразование адреса источника в статически заданный IP-адрес или диапазон адресов (*Source NAT*, *SNAT*).
— Преобразование адреса источника в текущий IP-адрес того интерфейса, через который отсылается данный пакет (*IP Masquerading*). В отличие от SNAT, сам адрес выходного интерфейса при этом может назначаться динамически.
— Преобразование адреса назначения в статически заданный IP-адрес или диапазон адресов (*Destination NAT*, *DNAT*, или *виртуальные сервера*). Для протоколов TCP или UDP может быть преобразован также номер порта назначения.
— Принудительная коммутация (в обход нормальной процедуры маршрутизации) или копирование пакета на заданный выходной интерфейс и/или шлюз (*Switching*).

Мнемоническое правило при настройке SNAT, DNAT и IP-маскарадинга состоит в том, что все они включаются на интерфейсе, подключённом к *внешней* сети.

§3.4.7. Потоки трафика

Ключевым понятием для работы механизма NAT является *поток*. В общем случае этим термином обозначается некоторая совокупность взаимосвязанных пакетов. Для каждого потока создается соответствующая запись в таблице NAT, непосредственно определяющая уникальное сочетание адресов и портов источника и назначения. Строгая интерпретация этого понятия зависит от типа трафика. Поток имеет ограниченное время жизни, по истечении которого запись из таблицы NAT удаляется. Подробное описание механизма идентификации потоков в Linux содержится в документе:

Oskar Andreasson. Iptables Tutorial 1.2.0. Chapter 7. The state machine.
https://www.frozentux.net/iptables-tutorial/iptables-tutorial.html#STATEMACHINE

Для пакетов TCP поток определяется наиболее просто, поскольку данный протокол по природе своей ориентирован на соединения. Потоком считается вся последовательность отправленных и полученных пакетов, относящихся к одному TCP-соединению, начиная от трехшаговой процедуры установления соединения (SYN — SYN/ACK — ACK) и заканчивая разрывом соединения. Между этими двумя процедурами соединение находится в состоянии ESTABLISHED и по нему передаются данные. Время жизни соединения зависит от текущей фазы соединения, в частности:

— для установленного соединения — 5 суток
— для соединения после нормального завершения (процедура FIN — ACK) — 2 минуты (чтобы пропустить все остаточные пакеты, которые могли задержаться на промежуточных узлах)
— для соединения после аварийного завершения (RST) — 10 секунд

После прохождения очередного пакета отсчет времени жизни соединения начинается заново. Полный список таймаутов для различных фаз TCP-соединения, установленных по умолчанию, приведен в вышеупомянутом документе. При необходимости эти значения могут быть изменены средствами ОС Linux.

Для пакетов UDP принадлежность пакетов к тому или иному потоку определяется на основе сочетания адресов и номеров портов. Время жизни потока после посылки первого пакета (т.е. пакета, который не ассоциируется ни с одним существующим потоком) — 30 секунд. Если за это время получен один ответный пакет, то поток считается установленным и время жизни увеличивается до 180 секунд. Если в течение 180 секунд не принято и не получено ни одного пакета, относящегося к данному потоку, запись о потоке уничтожается.

Для пакетов ICMP понятие потока применяется только к типам пакетов, подразумевающим парное взаимодействие Request/Reply (например, Echo Request / Echo Reply). После посылки пакета Request время ожидания ответа — 30 секунд. После получения соответствующего пакета Reply запись о потоке уничтожается, поскольку никаких последующих пакетов не предполагается.

Смешанные потоки TCP/ICMP или UDP/ICMP могут возникать, например, когда в ответ на попытку установления TCP-соединения или попытку отправки UDP-датаграммы приходит пакет ICMP Host Unreachable или Network Unreachable. Такие пакеты учитываются в рамках ранее открытого потока TCP или UDP, соответственно, и перенаправляются соответствующему хосту — инициатору потока. После этого запись о потоке уничтожается.

Для пакетов остальных протоколов транспортного подуровня, работающих поверх IP (например, для NETBLT, MUX или EGP) используется механизм, аналогичный UDP. Время жизни потока по умолчанию — 600 секунд.

Сложные протоколы прикладного уровня подразумевают или допускают использование одновременно нескольких портов TCP или UDP. К таким протоколам относятся FTP, TFTP, IRC и Amanda. Например, FTP использует порты TCP 21 (ftp) и 20 (ftp-data).

§3.4.8. Source NAT и IP-маскарадинг

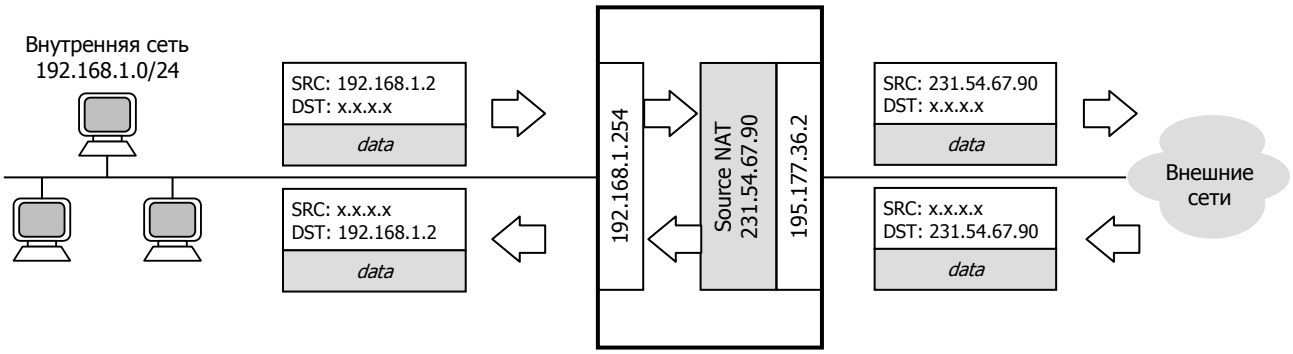
Трансляция адреса источника позволяет нескольким хостам внутренней сети выходить во внешнюю сеть, используя один или несколько внешних адресов. Обмен IP-пакетами происходит следующим образом:

— Внутренний интерфейс маршрутизатора получает пакет от хоста во внутренней сети, адресованный во внешнюю сеть. Этот пакет маршрутизируется на некоторый внешний интерфейс.

— Внутренний IP-адрес источника (как правило, приватный) заменяется указанным внешним адресом. В общем случае, для пакетов TCP и UDP номер порта источника может заменяться некоторым новым номером порта, относящимся к данному интерфейсу (см. ниже).

— В списке потоков NAT создается запись о вновь созданном потоке, после чего пакет отправляется во внешнюю сеть.

— Когда из внешней сети поступает пакет, в котором адрес назначения равен IP-адресу Source NAT на данном интерфейсе, номер порта назначения или идентификатор пакета, указанные в заголовке, проверяются по списку потоков NAT. Если запись о потоке с таким номером порта или идентификатором найдена, то выполняется обратное преобразование, и пакет маршрутизируется во внутреннюю сеть. Если такая запись не найдена, т.е. ни один из внутренних хостов ранее не обращался к данному внешнему хосту (в пределах контролируемых таймаутов), пакет уничтожается.



Если от внешнего хоста будет получен пакет с адресом назначения, принадлежащим внутренней сети, то такой пакет считается некорректным и будет безусловно уничтожен. Если извне получен пакет с адресом назначения, не принадлежащим внутренней сети, то в NSG Linux он будет пропущен через интерфейс без изменений и отправлен далее в соответствии с таблицей маршрутизации.

Внешний IP-адрес или диапазон адресов для Source NAT задается статически и, в общем случае, никак не связан с адресами, назначенными интерфейсам устройства NSG. При этом, однако, необходимо следить за тем, чтобы на вышестоящих маршрутизаторах был известен обратный маршрут к этим адресам.

Для пакетов TCP и UDP номер порта источника по возможности сохраняется. Однако если два хоста из внутренней сети пытаются одновременно посылать пакеты с одним и тем же номером порта, то для одного из них номер порта будет изменен. При этом все возможные номера портов разбиваются на три диапазона: менее 512, от 512 до 1023 включительно, и старше 1023. Эти диапазоны инвариантны, т.е. порты из каждого диапазона отображаются только на порты из того же диапазона (если только диапазон не задан принудительным образом).

IP-адрес и номер порта назначения в любом случае остаются неизменными.

NAT в форме *IP-маскарадинга* работает аналогично Source NAT и используется для той же самой цели. Различие состоит в том, что вместо фиксированного внешнего адреса (или диапазона адресов) используется IP-адрес выходного интерфейса. Если интерфейс имеет несколько IP-адресов, то для маскарадинга используется первый по списку адрес из сети, подходящей для отправки данного пакета.

Кроме того, если используемый IP-интерфейс переходит в состояние DOWN, то все записи IP-маскарадинга, относящиеся к нему, уничтожаются. После включения интерфейса все потоки, проходящие через него, идентифицируются заново.

IP-маскарадинг предназначен, в основном, для ситуации, когда IP-адрес выходного интерфейса назначается динамически и априори неизвестен, а маршрутизация в вышестоящей сети производится только на этот адрес. Типичный пример — PPP-подключение к входному пулу поставщика услуг Интернет. Для статических задач рекомендуется использовать Source NAT.

Допускается также использовать IP-маскарадинг в том случае, когда адреса выходных интерфейсов являются статическими. С качественной точки зрения, это решает поставленную задачу. Однако маскарадинг требует больше вычислительных ресурсов и сильнее сказывается на производительности устройства: вместо заранее известного значения, внешний адрес интерфейса приходится искать заново для каждого пакета.

В Linux правила Source NAT и маскарадинга содержатся в цепочке *iptables* POSTROUTING; соответственно, в NSG Linux 2.0 настройка производится в узле .ip.nat.POSTROUTING . В большинстве практических задач NAT-ируется весь трафик, проходящий через интерфейс устройства, подключённый к внешней сети. Поэтому для настройки NAT достаточно создать новое правило и указать в нём имя интерфейса (критерий) и процедуру NAT (действие) в поле target: SNAT либо MASQUERADE.

Для SNAT необходимо указать также используемый внешний адрес или диапазон адресов (через дефис) в поле to-source. Если в критериях отбора пакетов явно указан протокол TCP либо UDP, то после адреса можно указать, через двоеточие, диапазон номеров портов, которые будут принудительно подставляться в качестве портов источника. (Если диапазон не указан, то номера портов по возможности сохраняются, либо изменяются, как описано выше.) Дополнительно можно установить правило выбора портов в этом диапазоне: последовательно (по кругу, т.е., после достижения конца диапазона перебор портов начинается с начала) или случайным образом.

§3.4.9. Упрощённая настройка маскарадинга для интерфейсов PPP и портов LTE

Для интерфейсов, работающих по протоколу PPP и его производным, в NSG Linux 2.0 предусмотрена упрощённая настройка NAT Masquerading. В первую очередь, это сотовые интерфейсы GPRS/EDGE, 3G и CDMA/EV-DO. В большинстве задач требуется выполнять маскарадинг для всех пакетов, отправляемых через данный интерфейс, поэтому настройки NAT в этом случае можно считать априори известными, например:

```
ip
: nat
: : POSTROUTING
: : : 1
: : : : out-interface       = "cdma"
: : : : target              = "MASQUERADE"
```

Включить и выключить такую настройку можно разовыми командами add-nat/del-nat в меню любого порта (сотового, асинхронного) с инкапсуляцией PPP, туннеля PPTP или PPPoE.

Аналогичным образом ставится задача для портов LTE, несмотря на то, что они имеют Ethernet-подобный протокол. Для них команда add-nat/del-nat находится в узле service.

§3.4.10. Процедура Destination NAT

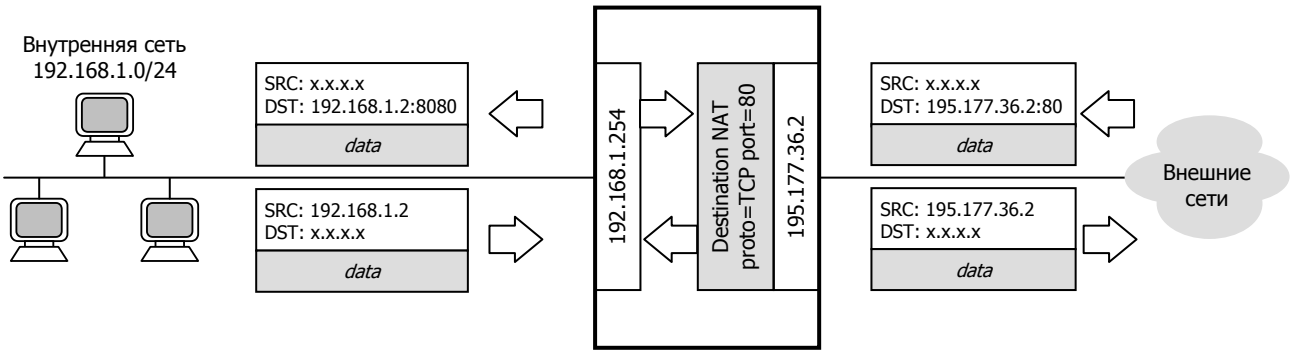
Процедура Destination NAT (DNAT), иначе называемая "виртуальными серверами", предназначена для доступа из внешней сети к определенным хостам во внутренней сети по явно указанным IP-адресам, протоколам и портам. При этом для внешнего мира все эти серверы будут расположены по внешнему IP-адресу интерфейса NSG. Обработка пакетов производится следующим образом:

— Внешний интерфейс маршрутизатора получает пакет, не относящийся ни к одному из потоков, инициированных со стороны внутренней сети.

— Если для данного протокола, работающего поверх IP (TCP, UDP, ICMP, GRE, PPTP, VPN и др.) и для данного номера порта назначения (TCP, UDP) или типа пакетов (ICMP) имеется запись в таблице NAT, то IP-адрес назначения заменяется на указанный адрес во внутренней сети; если указано, заменяется также порт назначения.

— Пакет передается на указанный адрес и порт во внутренней сети.

— Когда от внутреннего хоста приходит ответный пакет, IP-адрес источника заменяется внешним IP-адресом интерфейса, через который пакет отправляется во внешний мир, и пакет отправляется по назначению.



В Linux правила Destination NAT содержатся в цепочке *iptables* PREROUTING; соответственно, в NSG Linux 2.0 настройка производится в узле .ip.nat.PREROUTING . Как правило, Destination NAT используется в качестве виртуальных серверов, ожидающих подключения по определённому протоколу и, для TCP или UDP, на порту с определённым номером. Таким образом, правило DNAT обычно содержит критерии:

— Имя интерфейса, подключённого к внешней сети.

— IP-адрес, под которым виртуальный сервер известен внешнему миру (адрес должен принадлежать данному интерфейсу).

— Протокол и, для TCP/UDP, номер порта назначения, известный внешнему миру.

В качестве действия необходимо указать:

— В поле target — процедура DNAT

— В поле to-destination — новый IP-адрес сервера во внутренней сети. Можно указать диапазон адресов (через дефис), в этом случае создаваемые потоки будут равномерно распределяться между всеми адресами и

получится, что единственный виртуальный сервер на самом деле обслуживается целой фермой, или кластером, физических серверов.

Если в критериях отбора пакетов явно указан протокол TCP либо UDP, то после адреса можно указать, через двоеточие, диапазон номеров портов, которые будут принудительно подставляться в качестве портов назначения. Дополнительно можно установить правило выбора портов в этом диапазоне: последовательно (по кругу, т.е., после достижения конца диапазона перебор портов начинается с начала) или случайным образом. Если диапазон не указан явным образом, номера портов передаются без изменений.

Пример. На вышеприведённом рисунке клиент из внешнего мира обращается к виртуальному серверу HTTP (порт TCP 80) по публичному адресу 195.177.36.2. Реально он обслуживается физическим сервером с адресом 192.168.1.2 во внутренней сети, работающим на порту 8080.

```
: ip
: : nat
: : : PREROUTING
: : 1
: : : protocol          = "tcp"
: : : in-interface      = "eth1"
: : : destination       = "195.177.36.2"
: : : destination-port  = "80"
: : : target            = "DNAT"
: : : to-destination    = "192.168.1.2:8080"
```

ПРИМЕЧАНИЕ Процедура DNAT принципиально может применяться только к пакетам, направляемым на другие хосты. Если требуется перенаправить пакет, адресованный самому устройству NSG, на другой TCP или UDP порт этого же устройства, то следует использовать процедуру REDIRECT, например:

```
protocol          = "tcp"
destination-port  = "10023"
target            = "REDIRECT"
extra-target      = "--to-ports 10024"
```

Если подходящей записи в таблице Destination NAT не найдено, пакет передаётся дальше в процедуру маршрутизации без изменений. В частности:

— Если адрес назначения находится во внутренней сети, пакет будет передан в эту сеть без изменений.

— Если маршрут на адрес назначения не указан явно, то пакет будет возвращён, в большинстве практических задач, по умолчательному маршруту на вышестоящий узел сети (к поставщику услуг и т.п.). В результате он будет циркулировать между двумя узлами до тех пор, пока у него не истечёт TTL.

Отрегулировать прохождение таких пакетов можно с помощью фильтров. Как правило, NAT используется между внутренней сетью с приватными адресами и внешней сетью общего пользования. Пакеты с приватными адресами во внешнем мире циркулировать не должны — если они поступают, значит, это результат либо неправильной конфигурации какого-то узла, либо чьих-то злонамеренных действий. Поэтому целесообразно дополнить DNAT фильтром, запрещающим прохождение пакетов с адресом назначения в приватной сети (или с любым адресом назначения, кроме публичного адреса интерфейса), на этом же интерфейсе.

§3.4.11. Автоматическая настройка Destination NAT с помощью службы UPnP

Для автоматической настройки правил Destination NAT (виртуальных серверов) на устройствах NSG может использоваться служба Universal Plug and Play. В этом случае хосты-клиенты UPnP, находящиеся во внутренней сети, посылают устройству NSG запросы на добавление и удаление соответствующих правил NAT для протоколов TCP и UDP в зависимости от приложений, работающих на них и подразумевающих доступ к ним из внешнего мира. Демон UPnP, работающий на устройстве NSG, обрабатывает эти запросы и генерирует/удаляет соответствующие правила в цепочках ip.nat.UPnP и ip.filter.UPnP . Обращения к этим цепочкам создаются, соответственно, в цепочках ip.nat.PREROUTING и ip.filter.FORWARD и имеют наивысший приоритет; таким образом, входящие транзитные пакеты сначала передаются на обработку в цепочки UPnP, а затем возвращаются в исходную цепочку.

Служба UPnP в NSG Linux 2.0 поддерживает как спецификацию Internet Gateway Device (IGD), являющуюся составной частью общей спецификации UPnP, так и альтернативный протокол NAT Port Mapping Protocol (NAT–PMP). Подробное описание службы UPnP приведено в Части 4.

§3.4.12. Другие действия в цепочках *iptables*

Помимо нескольких основных действий, перечисленных выше, в цепочках *iptables* могут использоваться также и другие операции с пакетами. Полный перечень действий, допустимых в том или ином узле, зависит от конкретной цепочки и составляется динамически. Подробно обо всех возможных действиях см. первоисточник — *man pages* по *iptables*. Краткий перечень возможных действий приведён ниже.

CLASSIFY — Назначить пакету определённую выходную очередь и приоритет в этой очереди. Только для цепочек FORWARD, OUTPUT, POSTROUTING таблицы mangle.

CLUSTERIP — Организовать простейший кластер из нескольких узлов, работающих на одном IP и MAC-адресе, со статическим распределением соединений между ними.

CONNMARK — Установить метку соединения для данного пакета.

DSCP — Изменить значение поля DSCP (6 бит) в заголовке пакета IPv4. Только для таблицы mangle.

ECN — Принудительно очистить поле ECN (Explicit Congestion Notification) в заголовке TCP. Только для таблицы mangle.

LOG — Внести данный пакет в лог на уровне ядра (получаемый с помощью dmesg или syslogd). Данное действие не является терминирующим, т.е. после срабатывания данного правила обработка пакета продолжается в данной цепочке. Если после этого требуется выполнить какое-либо другое действие, например, уничтожить нежелательный пакет, то его следует задать в виде отдельного правила с тем же набором критериев и меньшим приоритетом (т.е. бóльшим номером в списке).

MARK — Установить метку netfilter для данного пакета. Следует использовать только в таблице mangle.

NETMAP — Статическое отображение адресов из одной сети на адреса в другой сети. Только для таблицы nat.

NFLOG — Более универсальный вариант ULOG, обеспечивающий передачу информации о пакете не напрямую в сокет *netlink*, а через специальную подсистему — *logging backend*.

RATEEST — Сбор статистики, оценка скорости передачи и сохранение результатов для последующих вычислений с помощью критерия rateest .

REDIRECT — Перенаправить пакет на локальную машину. Только для цепочек PREROUTING и OUTPUT таблицы nat.

RETURN — Вернуться в предыдущую цепочку.

TOS — Изменить значение поля TOS (8 бит) в заголовке пакета IP. Только для таблицы mangle.

TCPMSS — Изменить значение MSS (Maximum Segment Size) в пакетах TCP SYN, чтобы вручную ограничить максимальный размер для данного соединения (обычно — MTU выходного интерфейса минус 40). Только для правила, содержащего протокол TCP в качестве одного из критериев.

TCPOPTSTRIP — Удалить заданные опции TCP из заголовка пакета. Только для правила, содержащего протокол TCP в качестве одного из критериев, и только в таблице mangle.

TEE — Отправить копию пакета через заданный шлюз.

TOS — Изменить значение поля Type of Service заголовка IPv4 или поля Priority IPv6. Только в таблице mangle.

TPROXY — Перенаправить пакет на локальный сокет, не изменяя его заголовок никоим образом. Допускается изменять только метку netfilter, которая может быть учтена средствами расширенной маршрутизации. Только для цепочки PREROUTING и вызываемых из неё пользовательских цепочек таблицы mangle.

TTL — Изменить значение поля TTL в заголовке пакета IP.

ВНИМАНИЕ Данная операция может привести к непредсказуемым последствиям. Запрещается явно устанавливать или увеличивать поле TTL в пакетах, отправляемых за пределы вашей локальной сети.

ULOG — Ретранслировать данный пакет из ядра Linux через сокет *netlink* с тем, чтобы сделать его доступным для пользовательских процессов. Как и LOG, данное действие не является терминирующим.

§3.5. Резервирование сетевых соединений на абонентских устройствах

§3.5.1. Особенности задачи, специфические проблемы

Задача использования нескольких альтернативных каналов связи (основного и резервного/-ых) с сетями общего пользования для устройств, предполагающих использование конечным абонентом, существенно отличается от классической постановки для операторских устройств. Использование динамической маршрутизации (RIP, OSPF, BGP и др.) в данном случае не представляется возможным, поскольку требует, для начала, наличия у клиента собственного IP-адреса или пространства адресов, независимого от того, через какого оператора он в данный момент подключён. (В терминах IP-сетей, сеть абонента должна являться *автономной системой*.) Это достаточно сложная и дорогая услуга, доступная в реальности только операторам связи и крупным корпоративным пользователям. (По крайней мере, до широкого распространения протокола IPv6.) Обыкновенные конечные абоненты получают IP-адреса из пула адресов того или иного оператора — т.е. адреса будут свои для каждого соединения и принципиально не могут быть одними и теми же.

Как следствие вышесказанного, услуги протоколов динамической маршрутизации конечным абонентам не предоставляются ввиду их бессмысленности в такой ситуации.

Задача усложняется ещё более, если конечным пользователям выделяются не реальные статические, а динамические и/или приватные IP-адреса в сетях операторов, что является ныне общепринятой практикой. В этом случае абонент виден в Интернет под, вообще говоря, непредсказуемым и непостоянным адресом из пула оператора.

По этим причинам конечному абоненту приходится идти существенно другим путём: контролировать состояние доступных каналов связи и из фактически работающих выбирать один по некоторым приоритетам. Задача, таким образом, распадается на две:

— Регулярный мониторинг состояния каналов связи
— Переключение трафика с основного канала на резервный и обратно

В относительно простых ситуациях эти задачи решаются штатными средствами стека IP. В более сложных (более двух каналов связи с разными приоритетами, и/или отсутствие средств контроля для всех или некоторых каналов) приходится прибегать к помощи дополнительных скриптов. Но даже в этом случае возникает ещё и третья составляющая проблемы: при переключении на другой канал (или даже при переподключении по прежнему) клиент получает новый IP-адрес. Для серверов, с которыми он работает, он становится, как правило, уже новым клиентом, старая пользовательская сессия должна быть разорвана, и установлена новая. Штатные средства IP данную задачу не решают и даже не ставят — предполагается, что приложения должны самостоятельно обнаруживать потерю связи и реинициализировать сессию. Но в ряде корпоративных приложений, в частности, банковских, это нежелательно или недопустимо. Эта часть задачи, а также две предыдущие в сколь угодно сложных ситуациях, решаются фирменной разработкой NSG — *ui*TCP (Un-Interruptible TCP).

§3.5.2. Средства мониторинга работы соединений

Мониторинг физического уровня контролирует связность устройства с непосредственно сопряжённым физическим устройством и производится по сигналу Carrier Detect физического порта. Однако для разных типов портов он работает по-разному:

— Для портов PPP сигнал Carrier Detect модема учитывается всегда (если только он не отключён сознательно при помощи опции local). Однако для наиболее актуального случая — сотовых соединений — он не полностью информативен и не достаточен. Падение сигнала DCD безусловно означает разрыв связи, но вместе с тем в сотовых сетях возможны ситуации, когда соединение формально не разорвано, DCD поднят, но фактическая скорость соединения равна нулю. Таким образом, он, безусловно, полезен (чтобы обеспечить реакцию в кратчайшее время при корректном разрыве соединения), но его необходимо дублировать средствами более высоких уровней на случай некорректной работы сети или "зависания" внутреннего ПО сотового модема.

— Для портов Ethernet сигнал DCD контролирует, по существу, связь устройства с ближайшим коммутатором. Если выдернуть кабель между ними, то сигнал, естественно, упадёт. Но в реальности проблемы возникают, как правило, не в непосредственно подключённом кабеле, а где-то в вышестоящих промежуточных соединениях, отделённых от порта устройства NSG одним или несколькими коммутаторами и маршрутизаторами. Отследить их средствами физического уровня Ethernet принципиально невозможно, поэтому вместо них необходимо использовать средства мониторинга вышестоящих уровней. Например, это может быть LCP Echo в соединении PPPoE, или *netping* в общем случае.

  Помимо этого, отдельные порты отдельных моделей устройств (NSG–700, NSG–1800) оснащены встроенным коммутатором, и отключить их конструктивно невозможно. Выдергивание кабелей из внешних портов коммутатора никак не сказывается при этом на состоянии внутреннего порта процессора.

— Для туннелей аналогом сигнала CD является готовность нижележащего сетевого интерфейса, через который работает туннель. Однако средства для такого мониторинга имеются не во всех типах туннелей.

Мониторинг канального уровня контролирует связность устройства со смежным устройством этого уровня — например, с сервером PPP, минуя все промежуточные устройства нижележащих уровней (сотовые и телефонные сети с их собственными инкапсуляциями и протоколами, и т.п.) Из числа технологий, актуальных на сегодняшний день, он предусмотрен только в PPP в виде обмена пакетами LCP Echo Request / Echo Reply. Если не получено несколько пакетов подряд, соединение считается разорванным.

Мониторинг на канальном и на любых вышестоящих уровнях принципиально не может быть мгновенным, а может основываться, так или иначе, исключительно на посылке контрольных пакетов (в отсутствие полезных данных). Либо каждая сторона, независимо от другой, посылает через установленные интервалы времени пакеты *Request* некоторого протокола и ждёт ответных *Reply* (так сделано, например, в PPP), либо обе стороны посылают пакеты *keepalive* и следят за тем, чтобы они регулярно приходили от другой стороны (например, в Cisco-HDLC). В обоих случаях время срабатывания контрольных механизмов всегда конечное и равно, в наихудшем случае, произведению интервала посылки на допустимое число потерянных пакетов.

Мониторинг сетевого уровня контролирует связность устройства с конечным IP-хостом, минуя все промежуточные узлы IP-сети и все промежуточные устройства нижележащих уровней (коммутаторы Ethernet, системы доступа Ethernet-over-*x*DSL, сотовые сети и т.п.) в каналах связи между этими узлами. Он также возможен только по принципу *keepalive* либо *echo* и имеет конечное время срабатывания.

Наиболее употребительным вариантом является *netping* — скрипт на основе утилиты *ping*, использующий пакеты ICMP Echo Request / Echo Reply. Именно его удобно использовать для контроля каналов доступа Ethernet. Возможны также механизмы и на основе других типов пакетов.

Мониторинг прикладного уровня реализуется непосредственно в приложениях, работающих поверх сети, и контролирует связность конечных прикладных хостов — клиента и сервера (например, банкомата и процессингового центра), или узлов одноранговой сети. Поскольку особенности работы конкретных приложений находятся, по существу, вне компетенции администратора транспортной сети, то единственное, что возможно — принять его к сведению. В частности, именно этот механизм должен управлять переустановлением TCP-соединений и датаграммных потоков после перехода на другой канал связи при наличии NAT (см. §3.5.4).

Встроенные средства мониторинга в туннелях (разнообразные *keepalive*, *echo*, *dead peer detection* — в зависимости от технологии) работают аналогично механизмам прикладного уровня. Однако существенная разница состоит в том, что концы туннеля находятся непосредственно на сетевых устройствах и могут настраиваться сетевым администратором. С другой стороны, следует помнить, что поверх них могут работать ещё механизмы мониторинга прикладного уровня.

При одновременной работе нескольких механизмов мониторинга возникает дополнительная проблема — выбор корректных таймаутов на разных уровнях. Практическое инженерное правило состоит в том, что время срабатывания контрольного механизма на каждом вышестоящем уровне должно быть примерно в 3 раза больше, чем на нижележащем, чтобы они не дёргали систему вразнобой. Предположим, например, что банкомат работает по туннелю, устанавливаемому через сотовую сеть 2G. Для устойчивого, без ложных срабатываний, обнаружения отказов сети рекомендуется, по практическому опыту, таймаут LCP Echo 3×15 сек = 45 сек. Тогда тайм-аут разрыва туннеля должен составлять порядка 2 мин, а таймаут прикладного ПО банкомата — 5–6 мин, что находится уже на пределе разумного. Именно по этой причине, в частности, актуален переход на сети 3G (UMTS, CDMA EV-DO), которые обеспечивают существенно меньшие задержки пакетов и, как следствие, позволяют сократить таймаут LCP Echo до 10 сек; пропорционально ему сокращаются и все остальные таймауты, что позволяет, в конечном счёте, вписаться в таймауты прикладного ПО банкомата.

Во всех случаях целесообразно использовать механизмы мониторинга соединений самого нижнего уровня из доступных. Для соединений PPP это комбинация DCD и LCP Echo — чтобы обеспечить гарантированное срабатывание плюс, по возможности, минимальное время срабатывания. Для соединений Ethernet приходится использовать, как правило, *netping*. Для соединений псевдо-Ethernet "точка-точка" и "точка-многоточка" (Ethernet-over-G.703.6, туннели GRE с инкапсуляцией Ethernet, LTE, клиенты Wi-Fi) возможно использовать контроль DCD или его эквивалента.

ПРИМЕЧАНИЕ В сетях отдельных операторов наблюдалась ситуация, когда при исчерпании средств на счету PPP-соединение устанавливается успешно, но при попытке выхода в Интернет клиент безусловно перенаправляется на платёжный портал оператора. Такие случай не могут быть обнаружены встроенными средствами мониторинга PPP 1–2 уровней и требуют применения механизмов более высоких уровней (*netping* и др.).

В технологии NSG *ui*TCP для мониторинга используется односторонний механизм *keepalive* на уровне пакетов TCP: сервер судит о состоянии соединения по получению запросов от клиента, клиент — по получению ответов от сервера. Таким образом, контролируется туннель на всём протяжении от клиента до сервера, поверх всех транспортных сетей.

§3.5.3. Средства управления трафиком

Метрика. Штатный и простейший инструмент управления выбором маршрутов — это метрики. В таблице маршрутизации одновременно могут находиться несколько маршруты с разными метриками, приоритетен тот, у которого метрика меньше. Если приоритетный интерфейс переходит в состояние *down* или удаляется, этот маршрут удаляется из таблицы и из-под него всплывает маршрут с бóльшей метрикой. Он будет действующим до тех пор, пока приоритетный маршрут снова не перейдёт в состояние *up*. Очевидно, что резервных маршрутов в такой схеме может быть несколько. Очевидно также, что она работоспособна только в том случае, если приоритетный интерфейс имеет встроенные средства для мониторинга соединения на физическом и/или канальном уровне (либо их эквивалентах для туннелей).

Для статических маршрутов метрика назначается вручную. Для маршрутов, получаемых динамически (по протоколам PPP, DHCP и т.п.), также предусмотрен соответствующий параметр. Пример:

```
port
: m1
: : type                    = "cdma"
: : encapsulation           = "ppp"
: : ppp
: : : main
: : : : default-route       = true
: : : : default-route-metric = 1
: m2
: : type                    = "3g"
: : encapsulation           = "ppp"
: : ppp
: : : main
: : : : default-route       = true
: : : : default-route-metric = 3
```

В данном случае порт CDMA является основным. Порт 3G живёт своей собственной жизнью, падает, восстанавливается, и никак не используется до тех пор, пока работает основной порт.

Скрипты *netping*. Механизм *netping* в NSG Linux 2.0 предусматривает два произвольных скрипта — на случай пропадания связности и её восстановления. В скриптах можно использовать любые команды оболочки ash ОС Linux, а также вызов nsgsh в пакетном режиме (см. Часть 1), с оглядкой на права доступа для исполнения требуемых команд. Для управления маршрутами рекомендуется использовать непосредственно утилиты route или (рекомендуется) ip route, не затрагивая nsgsh и прав доступа в ней. Однако для этого необходимо знать, хотя бы с минимальном объёме, синтаксис их команд:

```
ip route add X.X.X.X/M dev интерфейс
ip route add X.X.X.X/M via G.G.G.G
ip route del X.X.X.X/M dev интерфейс
ip route del X.X.X.X/M via G.G.G.G

ip route add default dev интерфейс
ip route add default via G.G.G.G
ip route del default dev интерфейс
ip route del default via G.G.G.G
```

Если удаляемый маршрут — единственный в данную сеть, то выходной интерфейс и адрес шлюза указывать необязательно, например:

```
ip route del default
```

Пример использования ip route в скриптах *netping* см. в Части 4.

Подробное описание синтаксиса ip route см. в *man pages*.

События интерфейса. Любой из IP-интерфейсов в NSG Linux 2.0 можно рассматривать как генератор событий, которые, в свою очередь, могут анализироваться обработчиком событий и приводить к исполнению заданных скриптов (см. Часть 4). Явной необходимости в манипулировании таблицей маршрутизации при помощи данного механизма нет, поскольку маршруты добавляются и удалятся автоматически. Однако его можно использовать, например, для управления сложной маршрутизацией на основе правил, или для каких-либо других действий.

Рестарт портов и туннелей. Одно из возможных действий при переходе с одного канала связи на другой — переустановление туннелей по новому маршруту. Это может потребоваться по двум причинам:

— Для возобновления работы с новым IP-адресом клиента, под которым он виден серверу. При этом на стороне сервера следует использовать механизмы DPD, *keepalive* и т.п., чтобы своевременно удалять "мёртвые" сессии.

— Для создания необходимых записей NAT на промежуточных узлах, например, на выходе из приватной сети поставщика услуг в публичный Интернет.

Также может потребоваться рестарт проблемного порта, как один из способов восстановить его работу.

Для рестарта портов и туннелей в командных оболочках NSG Linux 2.0 имеется команда restart в этих узлах. Команда выполняется с правами пользователя и, следовательно, может применяться в скриптах без ограничений.

Маршрутизация туннелей *ui*TCP. В технологии NSG *ui*TCP при выборе некоторого канала связи принудительно создаётся маршрут до сервера, с маской /32, через этот канал. Соответственно, при переходе на другой канал этот маршрут удаляется и заменяется новым. Данный маршрут применяется только для отправки публичных пакетов туннеля через сети общего пользования и не имеет отношения к полезному трафику в туннеле.

После этого выполняется переустановление только транспортного TCP-соединения с сервером. Собственно туннель не инициализируется полностью, а переходит в "подвешенное" состояние и пытается возобновить работу без потери полезных данных.

Помимо этого, при уходе с некоторого канала связи может выполняться некоторый скрипт, предназначенный для рестарта этого порта. Это актуально, в первую очередь, для сотовых интерфейсов и для выбора SIM-карты для очередного подключения. Вместе с тем в данном скрипте могут выполняться и другие действия, например, запись в лог, включение индикации и т.п.

§3.5.4. Поведение NAT при изменении маршрутизации

При использовании NAT на устройствах, имеющих к одному назначению несколько маршрутов (основной и резервные) через разные интерфейсы, необходимо иметь следующую особенность работы NAT в ОС Linux. После того, как поток идентифицирован и для него создана запись NAT, эта запись функционирует независимо от маршрутизации и, соответственно, от текущего выходного интерфейса и его IP-адреса. Если устройство переходит, например, с основного канала Ethernet на резервное сотовое соединение, или наоборот, то процедура NAT не меняется и пакеты начинают уходить в новый интерфейс с адресами, NAT-ированными по-старому, уже не адекватно реальности. В результате пакеты не доходят до назначения, или ответные пакеты не могут вернуться на устройство, и соединение теряется. Например, единожды запущенный *ping* перестаёт проходить.

Предполагается, что эта потеря связи будет обнаружена средствами прикладного уровня (или туннеля — *keepalive, echo request/reply, dead peer detection* и т.п.) или вручную, после чего прикладное соединение будет переустановлено, идентифицировано механизмом NAT как новый поток, и для него будет создана новая запись NAT. Таким образом, поведение сети с точки зрения её конечной функции — передачи данных — определяется в данном случае особенностями и алгоритмами работы каждого конкретного приложения в отдельности.

Поскольку в первую очередь для проверки доступности узлов сети используется *ping*, то необходимо особо обратить внимание на его работу в вышеописанной ситуации. NAT идентифицирует пакеты ICMP Echo Request/Echo Reply как поток, основываясь на идентификаторе *ping*-а (PingID), которое утилита ping вставляет в пакет. В одном запуске утилиты PingID во всех пакетах одинаковый. Однако алгоритм его назначения может быть разным в разных реализациях *ping*, что приводит к существенно различному поведению.

Реализации *ping* для Linux при каждом новом запуске утилиты используют новый PingID. Таким образом, если через устройство NSG проходит *ping* с ПК или иного хоста под управлением Linux на третье устройство, то после смены канала связи он перестаёт проходить. Необходимо остановить его и запустить заново. Новые пакеты пойдут с новым PingID, механизм NAT идентифицирует их как новый поток, и *ping* будет успешно проходить.

Реализация *ping* в Windows, наоборот, использует один и тот же PingID и старается сохранить его до последней возможности. Перезапуск утилиты *ping* и даже переоткрытие консольного окна не гарантируют изменения PingID (хотя и приводят к этому в отдельных случаях, вероятно, по совокупности с какими-то неучтёнными факторами); единственным способом изменить PingID в Windows, известным автору, является перезагрузка Windows.

Как можно видеть, в данном случае *ping* не является достоверным показателем работоспособности сети и судить об успешности переходов на резервный канал и обратно следует исключительно по работе конкретных приложений.

Технология NSG *ui*TCP основана на использовании TCP-соединений и регулярном контроле за прохождением пакетов. При смене канала связи происходит переустановление TCP-соединения; как следствие — всегда создаётся корректная новая запись в таблице NAT.

Начиная с версии 2.0 *build 7*, в состав NSG Linux включена утилита conntrack, которая позволяет контролировать, в частности, состояние текущей таблицы NAT. Рекомендуется использовать её в скриптах netping или обработчика событий для полной очистки таблицы NAT. В этом случае для следующего пакета будет создана новая запись и либо работа продолжится (если это допускается приложением — например, *ping*), либо приложение получит ответ, что пакет принадлежит к несуществующей сессии (с нового IP-адреса и т.п.) и сможет инициализировать новую сессию немедленно, а не по тайм-ауту.

Подробно о параметрах данной утилиты см. conntrack --help или соответствующую *man page*.

§3.5.5. Примеры дополнительных действий при изменении канала связи

Одиночный сотовый модуль или соединение PPPoE, PPTP; рестарт IPsec:

```
services
: event-handler
: : 1
: : : virt-sensor       = ifstate.m1
: : : state             = up
: : : prev-state        = other
: : : script            = "ipsec setup restart"
```

Два сотовых модуля с разной метрикой маршрутов (см. пример §3.5.3), светодиодная индикация:

```
services
: event-handler
: : 1
: : : virt-sensor       = ifstate.m1
: : : state             = up
: : : prev-state        = other
: : : script            = "nsgsh -q .tools.led.green=on .tools.led.red=off"
: : 2
: : : virt-sensor       = ifstate.m1
: : : state             = other
: : : prev-state        = up
: : : script            = "nsgsh -q .tools.led.green=off .tools.led.red=on"
```

Мониторинг основного соединения Ethernet с помощью netping (см. Часть 4) и очистка таблицы NAT:

```
services
: netping
: : route_swap
.....................
: : : failure-script    = "ip route del default via 98.76.54.29; conntrack -F;"
: : : restore-script    = "ip route add default via 98.76.54.29; conntrack -F;"
```

§3.6. Множественные таблицы и маршрутизация на основе правил

§3.6.1. Множественные маршрутные таблицы

Дальнейшим развитием принципа маршрутизации пакетов является маршрутизация на основе установленных правил (набор правил иногда называется *политикой*). В этом случае при анализе пакета учитывается не только адрес назначения, указанный в заголовке IP-пакета, но и, в общем случае, любые другие поля, и маршрут выбирается в зависимости от совокупности этих параметров.

Расширенная маршрутизация в Linux реализуется путём создания нескольких таблиц маршрутизации; каждая таблица сама по себе работает традиционным образом — по адресу назначения пакета. Всего в системе существует 256 таблиц, идентифицируемых номерами от 0 до 255. По умолчанию, в системе автоматически формируются две основные таблицы local и main (см. ниже), которые и используются, если никакие политики и правила не настроены специально. Для них зарезервированы номера 255 и 254, соответственно; также зарезервированы номера 253 (специальная таблица default) и 0. Дополнительные таблицы, формируемые пользователем, могут иметь номера от 1 до 252.

Маршруты, создаваемые пользователем в узле .ip.route , попадают в таблицу main. Туда же попадают маршруты, создаваемые утилитами Linux (*route*, *ip route*) без явного указания таблицы, или формируемые автоматически при построении туннелей (IPsec, *ui*TCP и др.).

Таблица local создаётся автоматически и содержит обязательные маршруты на широковещательные и локальные интерфейсы; её модификация пользователем, хотя и не запрещена полностью, но настоятельно не рекомендуется.

Все остальные маршрутные таблицы, включая default, по умолчанию пустые. Наполнение этих таблиц может производиться, в рамках CLI и Web-интерфейсов NSG, в узле .ip.dynamic-routing : записи сначала создаются, средствами протоколов маршрутизации (RIP, OSPF, BGP) или псевдо-протоколов (*static* и др.), в маршрутных таблицах BIRD (не путать с таблицами в ядре Linux, по которым фактически производится маршрутизация!), а затем копируются из заданной таблицы BIRD в заданную таблицу ядра. Для манипуляций с маршрутами из-под командной оболочки Linux, в т.ч. в скриптах, можно использовать утилиту ip route .

Другой способ появления маршрутов в этих таблицах — клиент DHCP, в котором надо явно указать параметр dhcp-options.default-gw-table .

Помимо номеров, маршрутные таблицы в ядре Linux могут иметь имена, по которым (вместо номеров или наряду с ними) к ним могут обращаться некоторые утилиты. Имена назначаются в узле .ip.route-tables . В этом же узле можно просмотреть заданную таблицу маршрутизации или полный список маршрутов по всем таблицам. Непосредственно на работу маршрутизации данный узел не влияет и служит лишь для информационных и вспомогательных целей.

§3.6.2. Правила маршрутизации

В дополнение к таблицам, создаются *правила*, определяющие выбор той или иной таблицы в зависимости от всех интересующих пользователя параметров пакета. Настройка правил производится в узлах .ip.rules и .ipv6.rules, соответственно.

В качестве *критериев* для отбора пакетов непосредственно в правилах могут использоваться IP-адреса источника и назначения, имя интерфейса, через который поступил пакет, и значение поля ToS пакета. Если требуется учитывать какие-либо другие поля заголовка (например, протокол 4 уровня и номер порта), то это можно сделать с помощью цепочек IPtables (рекомендуется использовать для этой цели таблицу mangle), в которых для заданного набора критериев следует указать действие

target = "MARK"

и желаемую метку. Эта же метка указывается в параметре fwmark правила. Таким образом, пакет сначала анализируется IPtables и получает соответствующую метку, а затем по этой метке подбирается нужная таблица маршрутизации.

Если некоторое из полей не указано, то данный критерий не учитывается при анализе пакетов, т.е. по нему проходят все пакеты без исключения.

Действием для каждого правила является либо поиск маршрута по некоторой таблице маршрутизации, либо уничтожение пакета (молча или с отправкой тех или иных сообщений по ICMP отправителю пакета). Правила проверяются в порядке их нумерации. Если для пакета найдено подходящее правило и поиск в таблице, указанной этим правилом, дал положительный результат, или если пакет уничтожен, то дальнейшая проверка не производится. Если маршрут не найден, продолжается проверка по следующим правилам, вплоть до конца списка.

По умолчанию, в системе существуют всего три таблицы маршрутизации и пять правил (три для IPv4 и два для IPv6), соответствующих им:

0 Правило с наивысшим приоритетом, указывающее на специальную таблицу маршрутизации local (номер 255). Данная таблица содержит маршруты на локальные и широковещательные адреса.

32766 Правило для маршрутизации по основой таблице main (номер 254). Это "обычная" таблица маршрутизации, которая подразумевается при отсутствии каких-либо явно указанных правил.

32767 Только для IPv4: Правило для маршрутизации по маршрутной таблице default (номер 253). Данная таблица зарезервирована для использования в тех случаях, когда ни одно из предыдущих правил не позволило выбрать маршрут. По умолчанию, эта таблица пустая.

Все три правила не имеют критериев, т.е. действуют на все пакеты. Таким образом, если множественные таблицы и правила маршрутизации не настроены пользователем сознательно некоторым специальным образом, то сначала срабатывает первое правило и все пакеты проверяются по таблице local; если данный пакет не является широковещательным и не адресован ни одному из локальных интерфейсов устройства, то проверка продолжается и по второму правилу пакет передаётся в таблицу main, которая, как правило, должна содержать маршрут по умолчанию (с сетью назначения 0.0.0.0/0). Если ни в одной из таблиц, указанных правилами, не нашлось подходящего маршрута, пакет уничтожается.

§3.7. Управление качеством услуг (QoS)

§3.7.1. Общие сведения о процедурах QoS

Управление качеством услуг состоит в применении специфических алгоритмов, определяющих порядок обработки пакетов в выходных и входных очередях. Эти алгоритмы называются политиками (*policies*), или дисциплин управления очередями (*queueing disciplines*) и настраиваются на портах устройства NSG в ветви link.qdisc .

Критерии для обработки пакета тем или иным образом могут настраиваться:

— Явным образом с помощью *iptables* (узлы .ip.filter, .ip.mangle) на основе практически любых критериев, по которым возможно различать и классифицировать пакеты.

— Явным образом с помощью фильтров утилиты tc (*в данной версии NSG Linux 2.0 доступны только средствами ОС Linux*).

— Для пакетов Ethernet VLAN — явным образом с помощью правил in-CoS2Class-map .

— Неявным образом с помощью встроенных фильтров той или иной дисциплины (например, по значению поля ToS).

В результате пакеты распределяются в выходной очереди по различным классам (см. след. параграф), каждый из которых может обслуживаться по индивидуальным правилам. Алгоритм обработки пакетов может предусматривать следующие действия:

— Формирование (*shaping*) исходящего трафика: ограничение скорости и сглаживание пиковых всплесков.

— Планирование (*scheduling*) исходящего трафика, называемое также приоритизацией или переупорядочением. Пакеты, требующие каких-либо особых условий передачи (минимального времени задержки и т.п.), отбираются по определённым критериям (например, по полю ToS) и обслуживаются в первую очередь, в то время как пакеты, посланные на условиях *best effort* ("уж как получится…"), задерживаются в очереди или сбрасываются вообще. С другой стороны, дисциплина приоритизации должна всё-таки предусматривать какие-то гарантии для низкоприоритетного трафика, чтобы ему доставалась хоть какая-то часть полосы пропускания.

— Сброс — уничтожение пакетов, не подпадающих под установленные ограничения. В этом случае, в зависимости от характера трафика, производится либо повторная передача (например, средствами TCP), либо некоторая экстраполяция потерянных данных (например, в приложениях VoIP и IPTV).

— Ограничение (*policing*) входящего трафика. По существу, данное понятие является избыточным, поскольку управление потоком входящих пакетов уже не подконтрольно данному устройству. Задерживать на устройстве уже принятый пакет — занятие вполне бессмысленное; пакет надлежит либо передать по назначению (и, если это требуется, пропустить через формирователь и планировщик трафика на *выходном* интерфейсе), либо сбросить. Поскольку сброс пакета — тривиальная операция, доступная непосредственно в настройках фильтров, то отдельной дисциплины для управления входящим трафиком в данной версии NSG Linux не предусмотрено; в случае необходимости, ограничение входящего трафика может быть настроено средствами *iptables*.

В подавляющем большинстве задач, связанных с управлением QoS, требуется организовать обработку трафика по тому или иному алгоритму на *выходном* интерфейсе устройства. Например, чтобы ограничить скорость загрузки данных из Интернета некоторыми клиентами, находящимися во внутренней сети за устройством, следует ограничить скорость передачи на интерфейсе, обращённом к клиенту.

Управление трафиком эффективно только в том случае, если оно производится на устройстве, подключённом к самому медленному отрезку на пути дальнейшего следования пакетов. Например, не имеет смысла настраивать его на интерфейсе Fast Ethernet, если где-то далее используется канал WAN 2 Мбит/с; следует управлять трафиком на маршрутизаторе, который непосредственно передаёт данные в этот канал. В большинстве случаев таким "узким местом" является канал от площадки пользователя до поставщика сетевых услуг, т.е. именно от отрезок, в который передаются данные устройством NSG. Скорость передачи, формируемую устройством, рекомендуется устанавливать немного ниже физического быстродействия канала.

ПРИМЕЧАНИЕ Управление трафиком относится к процедурам 2–3 уровней и имеет своим результатом то, что пакет помещается в выходную очередь драйвера порта. После того, как пакет передан в драйвер физического уровня, дальнейший контроль над ним невозможен, например, если поступит высокоприоритетный пакет VoIP, а в драйвер к этом моменту уже передано несколько больших пакетов FTP, то этот пакет может быть передан только после них. Чтобы обеспечить адекватную и предсказуемую работу алгоритмов QoS, обычно следует уменьшить размер очереди на физическом уровне — параметр qlen .

§3.7.2. Очереди и классы

Дисциплины управления трафиком могут быть классовыми и безклассовыми.

Безклассовые дисциплины управляют потоком трафика на интерфейсе в целом. Большинство из них не предусматривает сортировки пакетов на какие-либо категории; отдельные дисциплины (*pfifo_fast*) имеют внутренние категории (полосы, *bands*), но они никак не настраиваются пользователем.

Классовые дисциплины предусматривают разделение трафика на несколько категорий (классов), каждая из которых обслуживается по определённым правилам. Каждый класс может содержать либо дисциплину для обслуживания пакетов, попадающих в этот класс, либо дочерние классы. Таким образом, система классов имеет древовидную структуру, имеющую одну корневую очередь (*root*) и неограниченное число ветвей и уровней. Число классов, их подчинённость, правила для отбора пакетов в каждый класс и дисциплина обслуживания каждого класса полностью настраиваются пользователем.

Для настройки механизмов QoS каждая дисциплина, или очередь, имеет идентификатор (*handle*), уникальный в пределах данного интерфейса. Если *handle* не назначен пользователем явно, система генерирует некоторое значение самостоятельно. Идентификатором класса является его номер, который должен быть уникальным в пределах данной очереди. Для того, чтобы поместить пакет в определённый класс (например, в результате действия CLASSIFY в IP-фильтрах), необходимо указать этот класс в формате очередь:класс (иногда называемых также major:minor). В этом случае пакет будет обработан согласно дисциплине, назначенной данному классу.

Вполне очевидно, что в классовой дисциплине следует определять не менее двух классов. Дисциплина, содержащая только один класс, не имеет смысла (по сравнению с аналогичной безклассовой), а не содержащая классов вообще — только создаёт напрасную нагрузку на процессор.

Номер очереди и номер класса в этой очереди задаются 16-битными числами (т.е. могут иметь значения от 1 до 65535). Теоретически они могут указываться совместно в виде одного 32-битного числа, но такой формат не рекомендуется и допустим не во всех полях интерфейса. Для безклассовых дисциплин идентификатор класса всегда указывается равным нулю, например, 12:0.

§3.7.3. Краткий обзор дисциплин управления очередями

Для управления очередями в NSG Linux 2.0 предусмотрены следующие дисциплины:

pfifo и bfifo — Простые дисциплины FIFO (First In — First Out), без ограничения скорости или приоритизации трафика. Размер буфера указывается в пакетах или в байтах, соответственно. Пакеты, не помещающиеся в данный буфер, сбрасываются; таким образом, ограничивается размер всплесков трафика. Эти политики могут также использоваться для сбора статистики или для одного только контроля за нагрузкой на интерфейсе (а именно, загружен ли он настолько, что пакеты начинают накапливаться в буфере данной дисциплины).

pfifo_fast — Ненастраиваемая безклассовая дисциплина для приоритизации трафика. Все пакеты разделяются, в зависимости от значения поля TOS, на 3 полосы (*bands*) с безусловным приоритетом: трафик из полосы 0 отправляется в первую очередь, из полосы 1 — только если нет пакетов в полосе 0, из полосы 2 — только если нет пакетов в обеих приоритетных. Внутри каждой полосы действует принцип FIFO. Данная политика устанавливается по умолчанию для всех интерфейсов, если пользователем явно не установлено иное.

SFQ — Stochastic Fairness Queueing — безклассовая дисциплина приоритизации, направленная на предоставление всем приложениям более или менее "справедливого" доступа к полосе пропускания на полностью загруженных каналах связи. Препятствует узурпации всей полосы пропускания одним или несколькими приложениями, генерирующими большой трафик, но только при условии, что приложение не создаёт много потоков пакетов одновременно.

RED — Random Early Drop (или Random Early Detection) — безклассовая дисциплина формирования трафика, направленная на то, чтобы сдерживать размер очереди. По мере роста трафика начинают сбрасываться отдельные пакеты случайным образом, что приводит к более плавной регулировке средствами вышестоящих уровней (TCP и др.) и менее болезненной деградации сети. Эффективна для широкополосных интерфейсов (~100 Мбит/с и более).

TBF — Token Bucket Filter — простая безклассовая дисциплина формирования трафика, основанная на алгоритме "дырявого ведра". Позволяет ограничить информационную скорость и размер всплесков трафика. По существу своему, разработана в эпоху узкополосных каналов WAN n×64 Кбит/с, поэтому не всегда эффективна в современных широкополосных сетях MAN и LAN в силу своих внутренних ограничений.

HTB Hierarchical Token Bucket — относительно простая (по сравнению с CBQ и др.) классовая дисциплина, являющаяся расширением TBF. Трафик разделяется на классы, для каждого из которых устанавливается своя информационная скорость и размер всплеска. При нехватке полосы пропускания для дочернего класса она может быть "одолжена" у родительского, если она у него имеется. Эффективное решение для распределения фиксированной и заранее известной полосы пропускания между приложениями и/или пользователями.

PRIO Классовая дисциплина — аналог pfifo_fast, полностью настраиваемая пользователем: число и структура классов, правила распределения пакетов по классам, дисциплина обслуживания каждого дочернего класса. Неизменным является только принцип безусловного приоритета каждого класса перед всеми последующими. Рекомендуется для каналов с заранее неизвестной или переменной полосой пропускания.

Другие дисциплины, реализованные в ядре ОС Linux (CBQ, WRR, DSMARK и др.), на практике используются крайне редко, ввиду их сложности или по другим причинам. Настройка этих дисциплин в пользовательском интерфейсе NSG Linux 2.0 не предусмотрена, однако при необходимости они могут быть задействованы средствами командной строки Linux (с правами пользователя root).

§3.7.4. Дисциплины pfifo и bfifo

Обе дисциплины представляют собой простейший буфер FIFO, который начинает заполняться, если пакеты поступают на передачу со скоростью, бóльшей физического быстродействия линии (и в количестве более qlen пакетов подряд). Пакеты, не помещающиеся в буфер, сбрасываются. Единственное различие между дисциплинами состоит в том, что для pfifo размер буфера указывается в пакетах, а для bfifo — в байтах.

Использование данных дисциплин позволяет установить факт перегрузок линии (размер данных в буфере отображается в логах), наблюдать статистику интерфейса, а также ограничить всплески трафика. Уменьшение размера буфера физического уровня (qlen) для них не требуется.

§3.7.5. Дисциплина pfifo_fast

Простая безклассовая дисциплина, используемая по умолчанию на всех интерфейсах, если для них не установлена иная дисциплина явным образом. Создаёт 3 выходных полосы (*bands*) с идентификаторами 0, 1 и 2. Пакеты из полосы 0 отправляются в первую очередь, из полосы 1 — только если нет пакетов в полосе 0, из полосы 2 — только если нет пакетов в обеих приоритетных. Пакеты распределяются между полосами в зависимости от внутреннего приоритета пакета в Linux, который, в свою очередь, может быть установлен в зависимости от поля ToS (по умолчанию) или иными способами. Максимальная длина очереди в каждой полосе равна значению qlen интерфейса.

Поле ToS может иметь 16 возможных значений. Они преобразуются в определённые значения внутреннего приоритета в Linux (который также может принимать значения от 0 до 15). Далее эти приоритеты отображаются на полосы обслуживания pfifo_fast.

Значение поля ToS		
bin	**dec**	**словесное**
0000	0	Normal Service
0001	1	Minimize Monetary Cost (mmc)
0010	2	Maximize Reliability (mr)
0011	3	mmc + mr
0100	4	Maximize Throughput (mt)
0101	5	mmc + mt
0110	6	mr + mt
0111	7	mmc + mr + mt
1000	8	Minimize Delay (md)
1001	9	mmc + md
1010	10	mr + md
1011	11	mmc + mr + md
1100	12	mt + md
1101	13	mmc + mt + md
1110	14	mr + mt + md
1111	15	mmc + mr + mt + md

Приоритет в Linux	
номер	**описание**
0	Best Effort
1	Filler
0	Best Effort
0	Best Effort
2	Bulk
2	Bulk
2	Bulk
2	Bulk
6	Interactive
6	Interactive
6	Interactive
6	Interactive
4	Interactive bulk
4	Interactive bulk
4	Interactive bulk
4	Interactive bulk

Выходная очередь
1
2
1
1
2
2
2
2
0
0
0
0
1
1
1
1

§3.7.6. Дисциплина PRIO

Классовая дисциплина PRIO является настраиваемым аналогом pfifo_fast и хорошо подходит для планирования трафика на соединениях с заранее неизвестной или переменной полосой пропускания. Число полос (параметр bands) определяется пользователем, при этом каждой полосе соответствует свой класс; следует обратить внимание, что полосы нумеруются от 0 до bands−1, а классы — от 1 до bands, соответственно. Каждый класс может содержать дочерние классы или политику обслуживания, создаваемые/выбираемые пользователем. Пакет может быть помещён в желаемый класс напрямую средствами IPtables, например

```
ip
: mangle
: : POSTROUTING
: : : 1
: : : : критерии фильтрации
: : : : target          = "CLASSIFY"
: : : : set-class       = "10:1"
```

или средствами управления приоритетом VLAN (IEEE 802.1p), например

```
port
: eth0
: : vlan
: : : eth0.101
: : : : in-CoS2Class-map
: : : : : 1   = "10:1"
: : : : : 2   = "20:1"
: : : : : 4   = "10:3"
```

Пакеты, не отправленные явным образом ни в один из классов, сортируются в соответствии с заданной картой приоритетов. Карта задаётся списка priormap, где элементом списка является внутренний приоритет в Linux (от 0 до 15), а значением элемента — номер полосы, в которую следует поместить данный пакет (от 0 до bands−1).

По умолчанию (при создании) очередь типа prio настроена в точности так же, как и стандартная очередь pfifo_fast интерфейса, т.е. содержит 3 класса и делит весь трафик между ними в соответствии со стандартной картой приоритетов (1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1).

Недостатки данной дисциплины следуют из её достоинств: если в приоритетных классах имеется большой объём трафика, то низкоприоритетный трафик не будет передаваться вообще. Чтобы избавиться от этой проблемы, следует назначить приоритетным классам, вместо дисциплины по умолчанию, некоторую дисциплину формирования трафика, не позволяющую им захватывать всю полосу пропускания — например, TBF.

§3.7.7. Дисциплина SFQ

Дисциплина приоритизации SFQ (Stochastic Fairness Queueing) заключается в том, что трафик разделяется на большое число потоков (по сеансам TCP, потокам UDP и т.п. — аналогично тому, как это делается в механизме NAT) и каждому потоку предоставляется полоса пропускания псевдослучайным образом. Использование такого алгоритма имеет смысл только на полностью загруженных соединениях. Он обеспечивает относительно равномерное распределение полосы пропускания между потоками и не позволяет какому-либо одному потоку полностью занять всю имеющуюся полосу. Алгоритм SFQ менее точен, чем остальные, зато требует значительно меньшего объема вычислений.

Данная дисциплина в большой степени самонастраивающаяся, поэтому для нее имеются только два необязательных специфических параметра. Количество байт, отправляемое из одной очереди за один шаг работы алгоритма, определяется параметром quantum. Не следует устанавливать данный параметр меньше максимального MTU пакетов, проходящих через данный интерфейс.

Второй параметр perturb определяет время между перестройкой алгоритма распределения полосы пропускания. Псевдослучайный алгоритм использует хэширование, поэтому является только односторонне однозначным: два различных потока могут случайно оказаться в одной "лунке рулетки" и в результате получить один и тот же квант полосы пропускания на двоих. Периодическая перестройка хэширования позволяет гарантировать, что если такая ситуация и возникает, она продлится не дольше ограниченного времени.

Ограничение данной дисциплины состоит в том, что она оперирует потоками, а не трафиком того или иного приложения в целом, поэтому она не может идентифицировать "агрессивные" приложения, создающие сразу большое число потоков (например, клиенты файлообменных сетей или менеджеры закачек). Такие приложения могут захватывать сразу много квантов полосы пропускания и затруднять работу других. Для ограничения работы таких пользователей следует комбинировать SFQ с классовыми политиками и фильтрами, а также административными мерами.

§3.7.8. Дисциплина RED

Безклассовая дисциплина Random Early Detection (или Random Early Drop) предназначена для формирования трафика на широкополосных (~100 Мбит/с и более) сетевых магистралях. Она призвана решить следующую задачу.

Без формирования трафика выходная очередь растёт, пока не будет достигнут предельный размер буфера, а затем сбрасываются все пакеты из хвоста очереди подряд (*tail-drop*). Это не всегда справедливо по отношению ко всем пользователям и приложениям, и грозит привести к резонансному эффекту: после того, как маршрутизатор сбросил большое число избыточных пакетов, примерно через одинаковый таймаут во всех соединениях начинается повторная передача этих пакетов, и магистраль снова оказывается перегруженной. Чтобы избежать переполнения очередей, приходится увеличивать размеры буферов — но это тоже оказывается палкой о двух концах: растёт время задержки пакетов в очереди, что неприемлемо для VoIP, IPTV и т.п. приложений, а трафик TCP-соединений становится очень неравномерным безо всякой разумной причины на то.

Алгоритм RED пытается удержать средний размер очереди вблизи заданного предела — существенно меньшего, чем максимально возможный размер очереди. Для этого он начинает, при приближении к этому пределу, случайным образом сбрасывать некоторую часть пакетов. Таким образом, деградация сети происходит более плавно и сказывается не на всех приложениях одновременно. Применительно к TCP-соединениям, это позволяет избежать резонансной повторной передачи, а также быстрее сойтись к "равновесной" эффективной скорости обмена пакетами. Для приложений, чувствительных к задержкам, уменьшение средней длины очереди означает уменьшение как собственно задержки пакетов, так и её вариации (джиттера). При этом вероятность сброса пакета из конкретного соединения пропорциональна не числу пакетов в нём, а занимаемой им полосе пропускания.

RED — относительно грубый метод, и не является панацеей от перегрузок. Но он вполне приемлем для магистральных соединений, где нет возможности контролировать трафик каждого из многочисленных пользователей и/или приложений в отдельности. Кроме того, он не создаёт большой дополнительной нагрузки на процессор.

RED имеет следующие основные параметры. Нижний порог (min) — это средняя длина очереди, при которой начинается случайный сброс пакетов; до этого предела пакеты не сбрасываются. Чтобы выбрать оптимальное значение, следует умножить максимальную величину задержки, допустимую при нормальных условиях работы сети (т.е. без перегрузок), на скорость линии (и поделить на 8, чтобы получить значение в байтах). Например, при допустимой задержке 200 мс и скорости 64 Кбит/с получается min = 0,200 × 64000 × 1/8 = 1600 байт.

От этого порога до некоторого желаемого размера очереди (max) вероятность сброса пакета увеличивается линейно от 0 до некоторого максимального значения probability (рекомендуется значение в диапазоне 0.01–0.02, т.е. 1–2%); таким образом, алгоритм стремится удержать среднюю длину очереди в пределах желаемой. Рекомендуется устанавливать max, как минимум, в 2 раза больше min (при малых значениях min — ещё больше, например, в 4 раза).

Третий параметр — величина всплеска (burst) — определяет влияние кратковременных всплесков трафика на среднюю скорость; чем он больше, тем медленнее реакция алгоритма, т.е. тем больше байт успеет уйти в линию прежде, чем начнётся случайный сброс. Значение параметра должно быть не менее, чем min/avpkt (см. ниже). Экспериментально рекомендуемое значение (2*min+max)/(3*avpkt) .

Далее, поскольку вероятность сброса пакетов в "мягком" режиме не доходит до 100%, то фактический размер очереди может сильно выходить за пределы max . Чтобы очередь не разрасталась до бесконечности, необходим жёсткий предел её длины, задаваемый параметром limit . После достижения этого предела сбрасываются все новые избыточные пакеты подряд, как это было бы без управления трафиком. Рекомендуется устанавливать предел в 8 раз больше max.

Два дополнительных параметра avpkt (средний размер пакета) и bandwidth (расчётная полоса пропускания) используются для вычисления средней длины очереди. Они влияют на эффективность и точность работы алгоритма, хотя и значения по умолчанию более или менее подходят для большинства практических ситуаций.

Наконец, параметр ecn позволяет сделать работу алгоритма более деликатной при условии, что оба взаимодействующих хоста поддерживают механизм явного уведомления о перегрузках (Explicit Congestion Notification). Если в поле DiffServ указано, что он поддерживается, то в пределах средней длины очереди от min до limit пакеты не уничтожаются, а отправляются адресату, но с установленным состоянием Congestion Detected. Адресат, получив такой пакет, устанавливает это же состояние в ответном пакете; источник, получив ответный пакет, должен снизить скорость отправки данных.

§3.7.9. Дисциплина TBF

Token Bucket Filter — простая безклассовая дисциплина формирования трафика, основанная на алгоритме "дырявого ведра" (token bucket). Она позволяет ограничить информационную скорость на интерфейсе некоторой заданной величиной, но при этом допускает кратковременные превышения этой величины.

Алгоритм подразумевает, что имеется некоторое виртуальное "ведро" (bucket), в которое с заданной скоростью подливаются разрешения на передачу очередной дозы информации, так называемые "токены" (token). Эта скорость и будет информационной скоростью интерфейса. Каждый токен позволяет отправить определённое число байт из очереди; после этого он считается использованным и удаляется из "ведра". Для двух потоков — токенов и данных — возможны три ситуации:

— Если скорость поступления пакетов в очередь *равна* скорости поступления токенов, то все пакеты отправляются без задержки и каждый пакет уносит с собой соответствующий токен.

— Если скорость поступления пакетов *меньше*, чем информационная скорость интерфейса (или вообще равна нулю), то токены не расходуются полностью и накапливаются в "ведре" — до тех пор, пока оно не будет заполнено до конца. После этого неизрасходованные токены просто теряются безвозвратно. Запас токенов в "ведре" может быть использован позже для отправки всплеска трафика.

— Если скорость поступления пакетов *больше*, чем информационная скорость интерфейса, то для их передачи сверх этой скорости используются токены, накопившиеся в "ведре". После того, как "ведро" опустошится, скорость передачи падает до установленной информационной скорости, а избыточные пакеты будут задерживаться в очереди и сбрасываться. Таким образом, алгоритм допускает кратковременные всплески трафика в объёме, не превышающем размер "ведра".

Для настройки алгоритма используются три основных параметра. Информационная скорость интерфейса (бит/с) устанавливается параметром rate; значение этого параметра следует выбирать несколько меньше физической скорости линии. По смыслу задачи, величина rate не может быть больше скорости линии.

Размер "ведра" (в байтах) определяется параметром burst; это максимальный размер разового всплеска, который может быть отправлен в линию немедленно (с максимальной физической скоростью интерфейса), если после долгой паузы поступит большая порция данных.

Максимальная длина очереди в байтах (limit) либо максимальное время ожидания в очереди (latency) определяют объём данных, которые могут накапливаться в очереди при нехватке токенов для их передачи. После достижения этого предела пакеты уже не задерживаются до поступления необходимого количества токенов, а сбрасываются. Два параметра являются взаимоисключающими и пересчитываются друг в друга с учётом размера "ведра", информационной скорости и, если задана, пиковой скорости (см. ниже).

Для нормальной работы алгоритма значения burst и limit должны быть не меньше максимального размера пакета (величина MTU + длина заголовков), который может быть отправлен интерфейсом. Далее, для эффективного использования памяти и достижения заданной информационной скорости рекомендуется соблюдать условия:

burst ≥ 2*packet, где packet — максимальный размер пакета
burst ≥ 2*rate/800 (это ограничение связано с дискретностью таймера Linux — 1/100 сек)
limit ≥ 2*burst

Вспомогательный параметр mpu — минимальный размер пакета — используется для корректного расчёта скорости. Дело в том, что даже пакет нулевой длины расходует, тем не менее, не нулевую полосу пропускания. Например, в сети Ethernet длина пакета не может быть меньше 64 байт.

Если в "ведре" образовался запас токенов, то вновь поступившие данные отправляются в линию с максимально возможной скоростью. Такое поведение может быть нежелательно по тем или иным причинам; в этом случае пиковую скорость для передачи разрешённого всплеска можно ограничить величиной peakrate . Однако здесь есть другой подводный камень: по существу, алгоритм передаёт пакет, и затем ждёт некоторое время, прежде чем передать следующий, чтобы средняя скорость соответствовала заданной. Но, поскольку системный таймер UNIX-систем работает с дискретностью 10 мс, то при наличии такого ограничения можно отправить не более 100 пакетов в секунду. Это означает, что, например, даже при среднем размере пакета 10000 бит (а это 1250 байт — нереально большая величина) интерфейс сможет развить фактическую скорость, (невзирая на значение peakrate, если только оно установлено) всего 1 Мбит/с, что смехотворно мало по нынешним меркам. Поэтому в паре с пиковой скоростью используется ещё один параметр — mtu. Он определяет максимальное количество данных, передаваемых в этом режиме за один такт. Для аптекарски точного расчёта полосы пропускания следует устанавливать его равным MTU интерфейса; если же увеличить его в несколько раз, это приведёт к снижению точности, но позволит отправлять каждые 10 мс не один, а несколько пакетов, и соответственно увеличить фактическую скорость передачи всплеска до величин, более или менее осмысленных в современных сетях Fast Ethernet.

§3.7.10. Дисциплина HTB

Hierarchical Token Bucket — классовая дисциплина формирования трафика, представляющая собой расширение TBF. На интерфейсе формируется дерево классов, в которые помещается трафик, например, с помощью фильтров или приоритетов IEEE 802.1p (см. §3.7.6). Для каждого класса создаются два "ведра" — токены и с-токены — и устанавливается промежуточный "потолок" (*ceiling*) скорости передачи. Класс имеет гарантированную информационную скорость rate; сверх этой скорости он может отправить не более чем burst байт на скорости ceil и cburst байт на максимальной физической скорости линии.

Непосредственно формирование трафика происходит только в конечных классах ("листьях" дерева). Если классу требуется передать данные на скорости выше rate, он "одалживает" (*borrow*) токены и/или с-токены у родительских классов. Строго говоря, вместо "одалживает" более уместным было бы слово "использует", поскольку использованные дочерним классом токены, естественно, не могут вернуться в "ведро" родительского класса. Родительский класс, если не имеет в этот момент своих собственных токенов, пытается одолжить их у своего родителя, и так далее до Адама, т.е. до корневого класса.

Таким образом, свободные токены перераспределяются от корня к "листьям" дерева. Промежуточные и корневой классы никакого формирования трафика не производят, они нужны только для управления раздачей токенов. Все использованные токены и с-токены относятся на счёт как конечного класса, так и его родителя и прародителей вплоть до того класса, у которого они были "одолжены".

По умолчанию, ceil = rate, т.е. перераспределение полосы пропускания от родительского класса к данному дочернему не производится. Для корректной и эффективной работы алгоритма следует соблюдать следующие правила:

— Сумма значений rate конечных классов не должна превышать значения rate для их общего родительского класса. В идеале, родительский класс должен иметь значение rate равным сумме rate всех своих дочерних классов, а разницу (ceil – rate) распределять между ними по требованию.

— Значения burst и cburst для родительского класса должны быть не меньше, чем максимальные значения burst и cburst, соответственно, среди всех его дочерних классов.

— Кроме того, остаётся в силе взаимосвязь между burst и rate (cburst и ceil, соответственно), приведённая в предыдущем параграфе.

Внутри каждого класса может быть определена своя дисциплина, но это уже не может быть HTB. Целесообразно использовать здесь дисциплины приоритизации трафика, поскольку HTB выполняет только формирование.

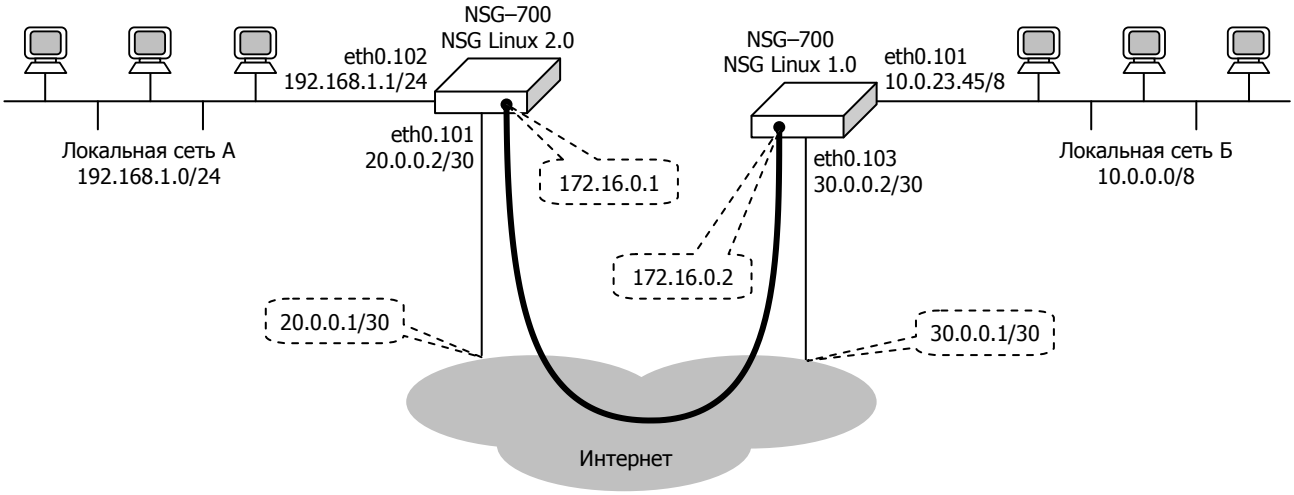
Помимо указанных, алгоритм предусматривает ещё некоторые параметры для тонкой настройки, но на практике эти параметры настраиваются автоматически и не требуют вмешательства пользователя.

Особая ситуация возникает в случае, если поступивший пакет не помещён явным образом ни в один из классов данной дисциплины. На этот случай в дисциплине HTB имеется глобальный параметр default — идентификатор класса, в который следует помещать такие пакеты. По умолчанию default = 0, т.е. указывает на саму корневую политику. В этом случае пакет передаётся с максимальной физической скоростью линии, минуя какие бы то ни было ограничения для дочерних классов.

Приложение 3–А. Примеры конфигурации

§3–А.1. OSPF over GRE

Две корпоративные площадки необходимо объединить туннелем через сеть общего пользования, с динамической маршрутизацией между ними. Обе площадки имеют статические публичные IP-адреса в сетях поставщиков услуг. Для наглядности с одной стороны используется устройство NSG–700 с программным обеспечением NSG Linux 2.0, на другой — такое же, но с NSG Linux 1.0.



Конфигурация устройства с Linux 2.0:

```
ethernet
: switch
: : phy0
: : : vlan-groups
: : : : 1               = 101
: : : : 2               = 102
: : : vlan-tagged       = true
: : phy1
: : : vlan-group        = 101
: : phy2
: : : vlan-group        = 102
: : vlan-mode           = true
ip
: route
: : 1
: : : gateway           = "20.0.0.1"
: dynamic-routing
: : enable              = true
: : kernel
: : : 1
: : : : export          = "all"
: : ospf
: : : 1
: : : : area
: : : : : 0.0.0.1
: : : : : : interface
: : : : : : : "\"gre1\""
: : : : : : networks
: : : : : : : 172.16.0.0/30
: : : : : : stubnet
: : : : : : : 192.168.1.0/24
: : : : export          = "all"
: : : : router-id       = "20.0.0.2"
```

Конфигурация устройства с NSG Linux 1.0:

```
hostname nsg
!
nsg
    tunnel  ip 1
        destination-ip 20.0.0.2
        source-ip 30.0.0.2
        keepalive 5 retry 3
        ttl 64
        ip address 172.16.0.2/32 peer 172.16.0.1
        exit
    ethernet-switch
        mode vlan
        exit
    port eth0
        vlan 101
            ip address 10.0.23.45/8
            exit
        vlan 103
            ip address 30.0.0.2/30
            exit
        exit
!
interface tuni1
    ip ospf network point-to-point
!
router ospf
    ospf router-id 30.0.0.2
    network 10.0.0.0/8 area 0.0.0.1
    network 172.16.0.1/32 area 0.0.0.1
!
ip route 0.0.0.0/0 30.0.0.1
!
```

```
port
: eth0
: : vlan
: : : eth0.101
: : : : ifAddress
: : : : : prefix          = "20.0.0.2/30"
: : : eth0.102
: : : : ifAddress
: : : : : prefix          = "192.168.1.1/24"
tunnel
: gre
: : gre1
: : : ifAddress
: : : : prefix            = "172.16.0.1/30"
: : : source              = "20.0.0.2"
: : : destination         = "30.0.0.2"
: : : keepalive           = "yes"
```

ВНИМАНИЕ Для работы механизма GRE *keepalive* создаются служебные правила в цепочках .ip.mangle.GRE_KA и ip.mangle.input . После включения/выключения *keepalive* необходимо отдельно применить изменения в этом узле или выше. Настройка данных правил пользователем не предусмотрена, поэтому в общей конфигурации они не выводятся.

ПРИМЕЧАНИЕ Начиная с версии NSG Linux 2.0 *build 4*, для OSPF допустимо также указание адреса удалённой стороны соединения "точка-точка" в качестве сети с маской /32:

```
ip
: : ospf
: : : 1
: : : : area
: : : : : 0.0.0.1
: : : : : : networks
: : : : : : : 172.16.0.2/32
tunnel
: gre
: : gre1
: : : ifAddress
: : : : prefix        = "172.16.0.1/32"
: : : : peer          = "172.16.0.2"
```