Linux Advanced Routing & Traffic Control HOWTO

Bert Hubert

bert.hubert [at] netherlabs.nl

Thomas Graf

tgraf%suug.ch

Gregory Maxwell

Remco van Mook

remco [at] virtu.nl

Martijn van Oosterhout

kleptog [at] cupid.suninternet.com

Paul B Schroeder

paulsch [at] us.ibm.com

Jasper Spaans

jasper [at] spaans.ds9a.nl

Pedro Larroy

piotr%member.fsf.org

Перевод выполнили:

(C) **Андрей Киселёв** (kis_an [at] mail.ru), **Иван Песин** (ipesin [at] n-ix.com.ua)

Оригинальную версию документа вы найдете по адресу http://lartc.org/.

Практическое руководство по применению **iproute2** (и в меньшей степени **netfilter**) для управления трафиком.

Содержание

1. Посвящение.
2. Введение.
    2.1. Ограничения и лицензионное соглашение
    2.2. Предварительные сведения
    2.3. Что может предложить вам Linux
    2.4. Дополнительные замечания
    2.5. Доступ к CVS и передача обновлений
    2.6. Список рассылки
    2.7. Структура документа.
3. Введение в **iproute2**
    3.1. Почему **iproute2?**
    3.2. Краткий обзор iproute2
    3.3. Необходимые условия
    3.4. Текущая конфигурация
        3.4.1. Просмотр списка сетевых интерфейсов с помощью утилиты **ip**
        3.4.2. Просмотр списка IP-адресов с помощью утилиты **ip**
        3.4.3. Просмотр списка маршрутов с помощью утилиты **ip**
    3.5. ARP
4. Правила - база политик маршрутизации
    4.1. Простая маршрутизация по источнику.
    4.2. Маршрутизация через несколько каналов/провайдеров.
        4.2.1. Раздельный доступ
        4.2.2. Распределение нагрузки.
5. GRE и другие тоннели.
    5.1. Несколько общих замечаний о тоннелях:
    5.2. Тоннелирование IP в IP.
    5.3. GRE тоннели.
        5.3.1. Тоннелирование IPv4.
        5.3.2. Тоннелирование IPv6.
    5.4. Тоннели неядерного уровня.
6. Тоннелирование IPv6 при помощьи Cisco и/или 6bone.
    6.1. Тоннелирование IPv6.
7. IPSEC: безопасная передача данных протоколами IP через Интернет
    7.1. Пример ручного конфигурирования безопасного соединения между двумя хостами.
    7.2. Пример автоматического конфигурирования безопасного соединения между двумя хостами.
        7.2.1. Теория.
        7.2.2. Пример.
        7.2.3. Автоматизация с использованием сертификатов X.509
        7.2.4. Как сохранить настройки туннеля в безопасности.
    7.3. Туннели IPSEC
    7.4. Другое программное обеспечение для работы с IPSEC
    7.5. Взаимодействие с другими операционными системами через IPSEC.
        7.5.1. Windows.
        7.5.2. Check Point VPN-1 NG
8. Маршрутизация групповых сообщений.
9. Дисциплины обработки очередей для управления пропускной способностью
    9.1. Понятие очереди и дисциплины обработки
    9.2. Простые бесклассовые дисцплины обработки очереди.
        9.2.1. pfifo_fast
        9.2.2. Token Bucket Filter
        9.2.3. Stochastic Fairness Queueing.
    9.3. Какие типы дисциплин нужно использовать.
    9.4. Терминология
    9.5. Классовые дисциплины обработки очередей.
        9.5.1. Порядок движения пакетов внутри полноклассовых дисциплин и классов.

9.5.2. Элементы дисциплины: корень, дескриптор, родительские элементы и элементы одного уровня.
9.5.3. Дисциплина PRIO.
9.5.4. Дисциплина CBQ.
9.5.5. Hierarchical Token Bucket
9.6. Классификация пакетов с помощью фильтров.
9.6.1. Ряд простых примеров фильтрации.
9.6.2. Наиболее употребимые способы фильтрации.
9.7. Intermediate queueing device.
9.7.1. Пример конфигурирования.
10. Распределение нагрузки по нескольким интерфейсам.
10.1. Предостережение.
10.2. Другие возможности.
11. Netfilter и iproute -- маркировка пакетов.
12. Расширенная фильтрация.
12.1. Классификатор u32.
12.1.1. Селектор U32.
12.1.2. Селекторы общего назначения.
12.1.3. Селекторы специального назначения.
12.2. Классификатор route.
12.3. Фильтры-ограничители трафика.
12.3.1. Способы ограничения.
12.3.2. Действия в случае превышения ограничения.
12.3.3. Примеры.
12.4. Хеш-фильтры.
12.5. Фильтрация трафика IPv6.
12.5.1. Почему не работают tc-фильтры в IPv6?
12.5.2. Маркировка пакетов IPv6 средствами ip6tables.
12.5.3. Использование селектора u32 для пакетов IPv6.
13. Параметры настройки сети в ядре.
13.1. Reverse Path Filtering.
13.2. Малоизвестные настройки.
13.2.1. Параметры настройки IPv4.
13.2.2. Параметры настройки устройств.
13.2.3. Параметры сетевых политик.
13.2.4. Параметры маршрутизации.
14. Специализированные дисциплины управления очередями.
14.1. bfifo/pfifo
14.1.1. Параметры и порядок использования.
14.2. Алгоритм Кларка-Шенкера-Чанга.
14.3. DSMARK.
14.3.1. Введение.
14.3.2. Что такое Dsmark и с чем его "едят"?
14.3.3. Основные принципы.
14.3.4. Как работать с Dsmark.
14.3.5. Как работает SCH_DSMARK.
14.3.6. Фильтр TC_INDEX.
14.4. Ingress qdisc.
14.4.1. Параметры и порядок использования.
14.5. Random Early Detection (RED)
14.6. Generic Random Early Detection.
14.7. Эмуляция VC/ATM.
14.8. Weighted Round Robin (WRR).
15. Решебник.
15.1. Запуск нескольких сайтов с различными SLA.
15.2. Защита от SYN flood.

15.3. Ограничение пропускной способности для ICMP-пакетов, с целью предотвращения dDoS атак.
15.4. Управление приоритетами для трафика различных типов.
15.5. Прозрачное проксирование с помощью netfilter, iproute2, ipchains и squid.
15.5.1. Схема движения пакетов после настройки.
15.6. Решение проблемы с Path MTU Discovery путем настройки MTU.
15.6.1. Решение
15.7. Решение проблемы с Path MTU Discovery путем настройки MSS.
15.8. Формирователь трафика: Низкая задержка, максимальная производительность.
15.8.1. Почему все так сложно?
15.8.2. Формирователь трафика на базе CBQ.
15.8.3. Формирователь трафика на базе HTB.
15.9. Ограничение скорости для отдельного хоста или подсети.
15.10. Пример подключения локальной сети к Интернет через NAT, с организацией QoS.
15.10.1. Начнем с оптимизации пропускной способности.
15.10.2. Классификация пакетов.
15.10.3. Дополнительная оптимизация
15.10.4. Выполнение настроек во время загрузки системы.
16. Построение мостов и псевдо-мостов с Proxy ARP.
16.1. Бриджинг и iptables.
16.2. Бриджинг и шейпинг.
16.3. Псевдо-мосты с проксированием ARP.
16.3.1. ARP и проксирование ARP
16.3.2. Реализация
17. Динамическая маршрутизация -- OSPF и BGP.
17.1. Настройка OSPF в Zebra
17.1.1. Предварительные условия.
17.1.2. Конфигурирование.
17.1.3. Запуск Zebra
17.2. Настройка BGP4 с помощью Zebra.
17.2.1. Конфигурация сети (пример).
17.2.2. Конфигурирование (пример).
17.2.3. Проверка конфигурации.
18. Прочие возможности.
19. Рекомендуемая литература.
20. Благодарности.

Глава 1. Посвящение.

Посвящается большому количеству людей. Перечислим лишь некоторых из них:

- Rusty Rassel
- Алексею Н. Кузнецову
- Отличным ребятам из Google
- Сотрудникам из Casema Internet.

Глава 2. Введение.

Добро пожаловать, уважаемый читатель!

Этот документ расскажет вам -- как повысить эффективность управления трафиком в системах, построеных на базе Linux 2.2/2.4. Многие из вас даже не подозревают, что уже работают с инструментальными средствами, которые позволяют делать весьма интересные вещи. Широко известные команды, такие как **route** и **ifconfig** -- фактически являются довольно тонкой оберткой вокруг очень мощной инфраструктуры **iproute2**.

Я надеюсь, что это HOWTO станет столь же востребованным как и документация к netfilter от Rusty Rassel.

Вы всегда можете связаться с нами. Однако, если ваши вопросы непосредственно не связаны с этим документом, вам следует обращаться к соответствующему списку рассылки. Мы не настолько свободны, чтобы заниматься исключительно ответами на вопросы, но нередко оказываем помощь участникам списков рассылки.

Если вас интересуют лишь простейшие приемы управления трафиком, то прежде, чем вы начнете "блуждать" по этому HOWTO, загляните в главу "Прочие возможности" и прочитайте раздел, посвященный CBQ.init.

2.1. Ограничения и лицензионное соглашение

Этот документ распротраняется в надежде на то, что он окажется полезным для вас, но без каких-либо гарантий, явных или подразумеваемых, включая, но не ограничиваясь ими, подразумеваемые гарантии коммерческого успеха и пригодности в конкретных целях.

Короче говоря, если ваша магистраль STM-64 была взломана и по ней стала распространяться порнография вашим самым уважаемым клиентам -- это не наша вина. Уж простите.

2.2. Предварительные сведения

Как следует из заголовка - это "Advanced" HOWTO, т.е. рассчитано на подготовленного читателя. Хоть мы и не имеем дело с космической техникой, но некоторая база знаний вам все же потребуется.

Ниже приводится пара ссылок на документы, в которых содержится основная, необходимая вам информация:

Rusty Russell: networking-concepts-HOWTO

> Очень хорошее введение. Рассказывает -- что такое сети и как они взаимодействуют между собой.

Linux Networking-HOWTO (ранее называлось Net-3 HOWTO)

> Отличное руководство, очень подробное. После его прочтения вы самостоятельно сможете подключиться к Internet. Как правило, в большинстве дистрибутивов, располагается в `/usr/doc/HOWTO/NET3-4-HOWTO.txt`, но так же доступно и в Интернет, по адресу http://www.linuxports.com/howto/networking (русский перевод: http://www.linux.opennet.ru/docs/HOWTO-RU/NET-3-HOWTO.html).

2.3. Что может предложить вам Linux

Вот далеко неполный список из того, что может предложить вам операционная система Linux:

- Управлять пропускной способностью НА отдельных компьютерах.
- Управлять пропускной способностью К отдельным компьютерам.
- Поможет "раздать" пропускную способность по-справедливости.
- Защитить вашу сеть от DoS-атак
- Предотвратить нападения из вашей сети на серверы в Интернет.
- Распараллелить несколько серверов, с целью равномерного распределения нагрузки.
- Ограничить доступ к вашим компьютерам.
- Ограничить доступ ваших пользователей к другим узлам сети.
- Выполнять маршрутизацию на основе UID (да!), MAC-адресов, исходящих IP-адресов, номеров портов, типа обслуживания, времени суток и содержимого.

На сегодняшний день эти дополнительные возможности не получили широкого распространения. На то есть ряд причин: хотя имеющаяся документация достаточно подробна, она почти не содержит практических рекомендаций. А вопросы управления трафиком вообще не освещены.

2.4. Дополнительные замечания

Есть несколько замечаний, относительно этого документа, которые хотелось бы особо отметить. По большей части этот документ был написан мною, но мне не хотелось бы, чтобы это положение вещей оставалось таким и дальше. Я являюсь убежденным сторонником идей Open Source и

потому призываю вас вносить свои изменения, исправления, дополнения и пр. Не стесняйтесь сообщать мне о допущеных ошибках, опечатках или устаревшей информации. Если мой английский кажется вам несколько корявым -- не забывайте, что я не являюсь носителем этого языка. Не бойтесь присылать мне ваши замечания и предложения.

Если вы уверены в том, что уровень вашей квалификации позволяет вам оказать поддержку тому или иному разделу документа или вы в состоянии написать и поддерживать новые разделы -- добро пожаловать! Это HOWTO доступно через CVS, в формате SGML. Я буду только рад, если число соавторов увеличится.

В поиске слабо освещенных тем вам помогут примечания "FIXME". Желаете внести исправления -- всегда пожалуйста! Везде, где вы встретите слово "FIXME", знайте, что вы вступаете на неизведанную территорию. Я не берусь утверждать, что в других местах нет ошибок, но здесь вам следует быть особенно осторожными. Если вы смогли подтвердить правильность того или иного предположения, пожалуйста, сообщите нам, чтобы мы могли удалить примечание "FIXME".

2.5. Доступ к CVS и передача обновлений

Канонический адрес этого документа: http://www.ds9a.nl/lartc.

Сейчас у нас открыт анонимный доступ к CVS из любой точки мира. Сделано это с целью предоставить вам возможность всегда иметь самую последнюю версию HOWTO и присылать свои исправления без особых затруднений.

Кроме того, это дает возможность авторам работать с исходным текстом документа независимо друг от друга.

```
$ export CVSROOT=:pserver:anon@outpost.ds9a.nl:/var/cvsroot
$ cvs login
CVS password: [введите 'cvs' (без кавычек)]
$ cvs co 2.4routing
cvs server: Updating 2.4routing
U 2.4routing/lartc.db
```

Если вы внесли какие либо изменения, просто дайте команду `cvs -z3 diff -uBb`, а полученый вывод отправьте по адресу: <howto@ds9a.nl>, это облегчит нам внесение ваших исправлений. Изменения должны вноситься в файл с расширением `.db`, все остальные файлы генерируются на его основе.

Предоставляемый `Makefile`, облегчит вам преобразование документа в форматы postscript, dvi, pdf, html и простой текст. Вам может потребоваться установить у себя **docbook**, **docbook-utils**, **ghostscript** и **tetex**, чтобы иметь возможность преобразования документа во все вышеперечисленные форматы.

Не тратьте ваши силы на файл `2.4routing.sgml` -- это устаревшая версия! Свежая версия находится в файле `lartc.db`.

2.6. Список рассылки

Поток писем к авторам документа увеличивается все больше и больше. Поэтому было решено запустить список рассылки, в котором люди могли бы поговорить друг с другом о проблемах маршрутизации и управлении трафиком. Вы можете подписаться на рассылку по адресу: http://mailman.ds9a.nl/mailman/listinfo/lartc .

Там же вы найдете архив, который, в свое время, задумывался как, своего рода, база знаний. Авторы документа предпочли бы отвечать на вопросы, не задававшиеся ранее, поэтому, пожалуйста, прежде чем послать свой вопрос -- загляните сначала в архив, возможно там вы найдете нужный ответ.

2.7. Структура документа.

В самом начале находятся разделы, представляющие наибольший интерес, но, к сожалению, менее точные и менее полные.

Маршрутизация и фильтрация -- суть разные понятия. Фильтрация достаточно хорошо описана в HOWTO от Rusty Rassel, которые вы найдете по адресу http://netfilter.samba.org/unreliable-guides/ .

Наше внимание главным образом будет сконцентрировано на связке **netfilter - iproute2**.

Глава 3. Введение в iproute2

3.1. Почему iproute2?

Большинство дистрибутивов Linux, как и большинство ОС UNIX, в настоящее время используют довольно древние утилиты **arp**, **ifconfig** и **route**. Пока что эти инструменты работают достаточно адекватно, но иногда, на ядрах Linux версии 2.2 и выше, они могут вести себя довольно неожиданно.

Сетевая подсистема, в ядрах 2.2 и выше, была полностью переписана. Новый сетевой код дал увеличение производительности и более высокие эксплуатационные характеристики, что делает Linux еще более привлекательным на рынке операционных систем.

Фактически, реализация сетевой подсистемы в Linux, выполняющая классификацию, маршрутизацию и фильтрацию, оказалась даже более полной, чем в специализированных маршрутизаторах, межсетевых экранах и других устройствах управления трафиком.

По мере появления новых разработок, они "наслаивались" поверх существующих реализаций в существующих операционных системах. Это постоянное наслаивание привело к тому, что код, решающий задачи управления сетевым трафиком, временами проявлял весьма странное поведение.

Эта, заново переписанная, реализация сетевой подсистемы позволила достичь таких характеристик, которые раньше были просто недоступны.

3.2. Краткий обзор iproute2

Linux обладает довольно сложной системой управления пропускной способностью, названной Traffic Control (Управление Трафиком). Она поддерживает различные методы классификации, деления по приоритетам, совместного использования, и ограничения как входящего, так и исходящего трафика.

Мы начнем обсуждение проблемы с краткого обзора **iproute2** и ее возможностей.

3.3. Необходимые условия

Вам следует установить необходимый инструментарий -- набор утилит командной строки. Этот пакет называется **iproute**, по крайней мере в *Red Hat* и *Debian*. Если в вашем дистрибутиве его нет, то пакет можно скачать по адресу: `ftp://ftp.inr.ac.ru/ip-routing/iproute2-2.2.4-now-ss??????.tar.gz`.

Можете попробовать взять самую последнюю версию.

Отдельные утилиты пакета требуют, чтобы в ядре были разрешены некоторые опции. Следует отметить, что все дистрибутивы *Red Hat*, до версии 6.2 включительно, поставляются с ядром, в котором по-умолчанию большинство необходимых опций отключено.

В *Red Hat 7.2* такое положение вещей исправлено.

Кроме того, в ядре должна быть включена поддержка **netlink**, этого требует **iproute2**.

3.4. Текущая конфигурация

Для вас может оказаться сюрпризом, но **iproute2** уже сконфигурирована! Существующие команды **ifconfig** и **route** уже используют расширенные системные вызовы, но главным образом с настройками, заданным по-умолчанию.

Утилита **ip** является основной в пакете. Попробуем с ее помощью исследовать имеющиеся в системе сетевые интерфейсы.

3.4.1. Просмотр списка сетевых интерфейсов с помощью утилиты ip

```
[ahu@home ahu]$ ip link list
1: lo: <LOOPBACK,UP> mtu 3924 qdisc noqueue
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: dummy: <BROADCAST,NOARP> mtu 1500 qdisc noop
```

```
    link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff
3: eth0: <BROADCAST,MULTICAST,PROMISC,UP> mtu 1400 qdisc pfifo_fast qlen 100
    link/ether 48:54:e8:2a:47:16 brd ff:ff:ff:ff:ff:ff
4: eth1: <BROADCAST,MULTICAST,PROMISC,UP> mtu 1500 qdisc pfifo_fast qlen 100
    link/ether 00:e0:4c:39:24:78 brd ff:ff:ff:ff:ff:ff
3764: ppp0: <POINTOPOINT,MULTICAST,NOARP,UP> mtu 1492 qdisc pfifo_fast qlen 10
    link/ppp
```

Здесь приведен результат работы команды **ip link list** на моем домашнем маршрутизаторе (с "поднятым" NAT), у вас он может несколько отличаться. Я поясню часть того, что вы видите, но не все, а только то, что нас интересует в данный момент.

Первым в списке находится локальный (loopback) интерфейс. В принципе, при крнфигурировании ядра, можно отключить поддержку этого интерфейса, но я бы не советовал этого делать. Размер *максимального блока передачи данных* (MTU -- Maximum Transfer Unit) для этого интерфейса составляет 3924 октета, и для него отсутствует очередь, поскольку он не соответствует никакому физическому устройству и существует только в "воображении" ядра.

Я пропущу фиктивный (dummy) интерфейс, так как он может отсутствовать на вашем компьютере. Дальше у меня идут два физических сетевых интерфейса, один -- со стороны модема, другой -- обслуживает мою домашнюю локальную сеть. И наконец последним в списке стоит интерфейс ppp0.

Обратите внимание на отсутствие IP-адресов в листинге. **iproute** отделяет понятие "интерфейса" от понятия "IP-адреса". При назначении нескольких IP-адресов одному интерфейсу (IP-алиасинг), понятие IP-адреса становится достаточно расплывчатым.

Зато показываются MAC-адреса -- аппаратные идентификаторы сетевых интерфейсов.

3.4.2. Просмотр списка IP-адресов с помощью утилиты ip

```
[ahu@home ahu]$ ip address show
1: lo: <LOOPBACK,UP> mtu 3924 qdisc noqueue
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 brd 127.255.255.255 scope host lo
2: dummy: <BROADCAST,NOARP> mtu 1500 qdisc noop
    link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff
3: eth0: <BROADCAST,MULTICAST,PROMISC,UP> mtu 1400 qdisc pfifo_fast qlen 100
    link/ether 48:54:e8:2a:47:16 brd ff:ff:ff:ff:ff:ff
    inet 10.0.0.1/8 brd 10.255.255.255 scope global eth0
4: eth1: <BROADCAST,MULTICAST,PROMISC,UP> mtu 1500 qdisc pfifo_fast qlen 100
    link/ether 00:e0:4c:39:24:78 brd ff:ff:ff:ff:ff:ff
3764: ppp0: <POINTOPOINT,MULTICAST,NOARP,UP> mtu 1492 qdisc pfifo_fast qlen 10
    link/ppp
    inet 212.64.94.251 peer 212.64.94.1/32 scope global ppp0
```

Этот листинг содержит более подробную информацию. Здесь показаны все IP-адреса, и каким интерфейсам они принадлежат. Здесь "inet" соответствует термину "Internet (IPv4)". Существует целый ряд типов сетевых адресов, но нас они пока не интересуют.

Взглянем поближе на интерфейс *eth0*. Из листинга видно, что ему назначен адрес "inet" -- 10.0.0.1/8, где "/8" определяет число бит, соответствующих адресу сети. Таким образом, для адресации хостов в сети у нас остается 32 - 8 = 24 бита, что соответствует адресу сети -- 10.0.0.0 и маске сети -- 255.0.0.0.

Это говорит о том, что любой хост в этой сети, например 10.250.3.13, будет непосредственно доступен через наш интерфейс с IP-адресом 10.0.0.1.

Для *ppp0* применима та же концепция, хотя числа в IP-адресе отличаются. Ему присвоен адрес - 212.64.94.251, без маски сети. Это означает, что он обслуживает соединение типа "точка-точка" (point-to-point), и что каждый адрес, за исключением 212.64.94.251, является удаленным. Но и это еще не все. Для этого интерфейса указывается адрес другого конца соединения -- 212.64.94.1. Здесь число "/32" говорит о том, что это конкретный IP-адрес и он не содержит адреса сети.

Очень важно, чтобы вы поняли суть этой концепции. Если у вас возникают какие либо затруднения, обращайтесь к документации, упомянутой в начале этого HOWTO.

Вы наверняка обратили внимание на слово "qdisc". Оно обозначает дисциплину обработки очереди (Queueing Discipline). Позднее мы коснемся этой темы подробнее.

3.4.3. Просмотр списка маршрутов с помощью утилиты ip

Итак, мы теперь знаем, как можно отыскать адреса 10.x.y.z, и как обратиться к адресу 212.64.94.1. Однако этого недостаточно, поскольку нам необходимо иметь возможность общения с внешним миром. Интернет доступен нам через ppp-соединение, которое объявляет, что хост с адресом 212.64.94.1, готов передать наши пакеты во внешний мир и вернуть результаты обратно.

```
[ahu@home ahu]$ ip route show
212.64.94.1 dev ppp0  proto kernel  scope link  src 212.64.94.251
10.0.0.0/8 dev eth0  proto kernel  scope link  src 10.0.0.1
127.0.0.0/8 dev lo  scope link
default via 212.64.94.1 dev ppp0
```

Этот листинг достаточно "прозрачен". Первые 3 строки сообщают сведения, которые нами уже обсуждались выше. Последняя строка говорит о том, что внешний мир доступен через 212.64.94.1 -- шлюз, заданный по-умолчанию. То что это шлюз, видно благодаря наличию слова "via" (в переводе с англ. -- "через"). Этот шлюз (с адресом 212.64.94.1) готов перенаправлять наши пакеты в Интернет и возвращать обратно результаты наших запросов.

Для примера, более "старая" утилита **route**, дает такой результат на моей машине:

```
[ahu@home ahu]$ route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use
Iface
212.64.94.1     0.0.0.0         255.255.255.255 UH    0      0        0 ppp0
10.0.0.0        0.0.0.0         255.0.0.0       U     0      0        0 eth0
127.0.0.0       0.0.0.0         255.0.0.0       U     0      0        0 lo
0.0.0.0         212.64.94.1     0.0.0.0         UG    0      0        0 ppp0
```

3.5. ARP

ARP -- Address Resolution Protocol (Протокол Определения Адреса) описан в RFC 826. Он

используется для определения ethernet-адреса по IP-адресу. Машины в Интернет более известны под именами, которые преобразуются в IP-адреса, благодаря чему узел сети, скажем с именем foo.com, имеет возможность связаться с другой машиной, например с именем bar.net. Но в ethernet-сетях для адресации используется не IP-адрес, а ethernet-адрес и здесь на сцену выходит протокол ARP.

Рассмотрим простой пример. Предположим, что имеется сеть из нескольких компьютеров. В ней находятся компьютеры *foo*, с адресом 10.0.0.1 и *bar*, с адресом 10.0.0.2. Пусть *foo* хочет послать пакет *ICMP Echo Request* (**ping**) компьютеру *bar*, чтобы проверить -- работает ли он, но увы, *foo* не знает ethernet-адрес компьютера *bar*. Таким образом, прежде чем **ping**-ануть *bar*, *foo* должен отослать ARP-запрос. Этот запрос очень похож на то, что обычно кричит человек, пытаясь отыскать в толпе своего товарища: "Bar (10.0.0.2)! Ты где?". В результате все машины в сети услышат "крик" *foo*, но только *bar* (10.0.0.2) откликнется на него, послав обратно ARP-ответ, который можно трактовать как: "Foo (10.0.0.1)! Я - здесь! Мой адрес 00:60:94:E9:08:12.". После этой "переклички" *foo* будет знать ethernet-адрес компьютера *bar* и сможет связаться с ним, пока опять не "забудет" (в кэше ARP) адрес компьютера *bar* (обычно записи в ARP-кэше удаляются через 15 минут).

Содержимое ARP-кэша можно просмотреть так:

```
[root@espa041 /home/src/iputils]# ip neigh show
9.3.76.42 dev eth0 lladdr 00:60:08:3f:e9:f9 nud reachable
9.3.76.1 dev eth0 lladdr 00:06:29:21:73:c8 nud reachable
```

Как видите, мой компьютер *espa041* (9.3.76.41) "знает", как найти компьютер *espagate* (9.3.76.1). А теперь добавим еще один адрес в наш кэш:

```
[root@espa041 /home/paulsch/.gnome-desktop]# ping -c 1 espa043
PING espa043.austin.ibm.com (9.3.76.43) from 9.3.76.41 : 56(84) bytes of data.
64 bytes from 9.3.76.43: icmp_seq=0 ttl=255 time=0.9 ms

--- espa043.austin.ibm.com ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.9/0.9/0.9 ms

[root@espa041 /home/src/iputils]# ip neigh show
9.3.76.43 dev eth0 lladdr 00:06:29:21:80:20 nud reachable
9.3.76.42 dev eth0 lladdr 00:60:08:3f:e9:f9 nud reachable
9.3.76.1 dev eth0 lladdr 00:06:29:21:73:c8 nud reachable
```

В результате попытки взаимодействия компьютера *espa041* с *espa043*, ethernet-адрес последнего был добавлен в кэш. По истечении некоторого тайм аута (если между этими двумя компьютерами больше не было передано ни одного пакета), *espa041* "забудет" адрес компьютера *espa043* и для того чтобы что-то сообщить ему, опять потребуется послать ARP-запрос.

Удалим адрес компьютера *espa043* из кэша:

```
[root@espa041 /home/src/iputils]# ip neigh delete 9.3.76.43 dev eth0
[root@espa041 /home/src/iputils]# ip neigh show
9.3.76.43 dev eth0  nud failed
9.3.76.42 dev eth0 lladdr 00:60:08:3f:e9:f9 nud reachable
9.3.76.1 dev eth0 lladdr 00:06:29:21:73:c8 nud stale
```

Теперь *espa041* "забыл" адрес компьютера *espa043*. Если *espa041* опять "захочет" что-то сообщить *espa043*, он будет вынужден вновь послать ARP-запрос. В этом листинге также видно, что в записи для *espagate* (9.3.76.1), состояние `reachable` (доступно) изменилось на `stale` (устарело). Это означает, что ethernet-адрес все еще является допустимым, но он должен быть

подтвержден при первой же попытке обмена.

Глава 4. Правила - база политик маршрутизации

Если ваш маршрутизатор обслуживает сложную сеть, то нужно удовлетворять нужды разных людей, обслуживание которых, вероятно, должно отличаться. База политик маршрутизации позволяет реализовать это с помощью набора таблиц маршрутизации.

Если вы хотите использовать эту возможность, убедитесь что ядро собрано с поддержкой "IP: advanced router" и "IP: policy routing".

Когда ядру необходимо выбрать маршрут, оно определяет в соответствии с какой таблицей это нужно делать. По-умолчанию, определены три таблицы. Старая утилита **route** изменяет таблицы main и local, как и утилита **ip** (по-умолчанию).

Правила по-умолчанию:

```
[ahu@home ahu]$ ip rule list
0:      from all lookup local
32766:  from all lookup main
32767:  from all lookup default
```

В этом листинге приведены приоритеты всех правил. Мы видим, что правила применяются ко всем пакетам (`from all`). Мы уже видели таблицу 'main', она выводится командой **ip route ls** , но таблицы 'local' и 'default' для нас новые.

Если мы хотим сделать что-то интересное, то нужно задать правила, использующие разные таблицы маршрутизации. Это позволит нам переопределить общесистемную таблицу маршрутизации.

За точной семантикой происходящего в ядре, когда есть несколько подходящих правил, обратитесь к документации ip-cref Алексея Кузнецова.

4.1. Простая маршрутизация по источнику.

Давайте опять рассмотрим реальный пример. У меня есть 2 (вообще-то 3, пора бы вернуть их) кабельных модема, подключенных к маршрутизатору Linux с NAT ('masquerading'). Люди с которыми я живу в одном доме, платят мне за использование Internet. Допустим одни из моих соседей ходит только на hotmail и хочет платить меньше. Мне это подходит, но при этом будет использоваться медленный канал.

Быстрое соединение имеет с моей стороны адрес 212.64.94.251, а с другой -- 212.64.94.1. Медленное соединение получает динамический адрес, в данном примере это 212.64.78.148, адрес провайдера -- 195.96.98.253.

Таблица local:

```
[ahu@home ahu]$ ip route list table local
broadcast 127.255.255.255 dev lo  proto kernel  scope link  src 127.0.0.1
local 10.0.0.1 dev eth0  proto kernel  scope host  src 10.0.0.1
broadcast 10.0.0.0 dev eth0  proto kernel  scope link  src 10.0.0.1
local 212.64.94.251 dev ppp0  proto kernel  scope host  src 212.64.94.251
broadcast 10.255.255.255 dev eth0  proto kernel  scope link  src 10.0.0.1
broadcast 127.0.0.0 dev lo  proto kernel  scope link  src 127.0.0.1
local 212.64.78.148 dev ppp2  proto kernel  scope host  src 212.64.78.148
local 127.0.0.1 dev lo  proto kernel  scope host  src 127.0.0.1
local 127.0.0.0/8 dev lo  proto kernel  scope host  src 127.0.0.1
```

Много очевидных вещей, но они должны быть где-то указаны. Вот здесь они и заданы. Таблица default -- пустая.

Посмотрим теперь на таблицу main:

```
[ahu@home ahu]$ ip route list table main
195.96.98.253 dev ppp2  proto kernel  scope link  src 212.64.78.148
212.64.94.1 dev ppp0  proto kernel  scope link  src 212.64.94.251
10.0.0.0/8 dev eth0  proto kernel  scope link  src 10.0.0.1
127.0.0.0/8 dev lo  scope link
default via 212.64.94.1 dev ppp0
```

Создадим новое правило для нашего гипотетического соседа, которое будет называться 'John'. Хотя мы можем работать просто с числами, намного проще и понятней если мы определим названия наших таблиц в файле /etc/iproute2/rt_tables.

```
# echo 200 John >> /etc/iproute2/rt_tables
# ip rule add from 10.0.0.10 table John
# ip rule ls
0:      from all lookup local
32765:  from 10.0.0.10 lookup John
32766:  from all lookup main
32767:  from all lookup default
```

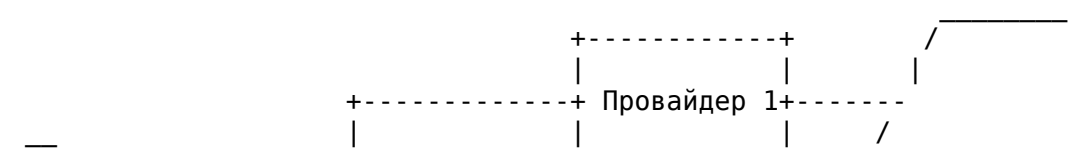
Теперь нам нужно лишь сгенерировать таблицу John и очистить кэш маршрутов:

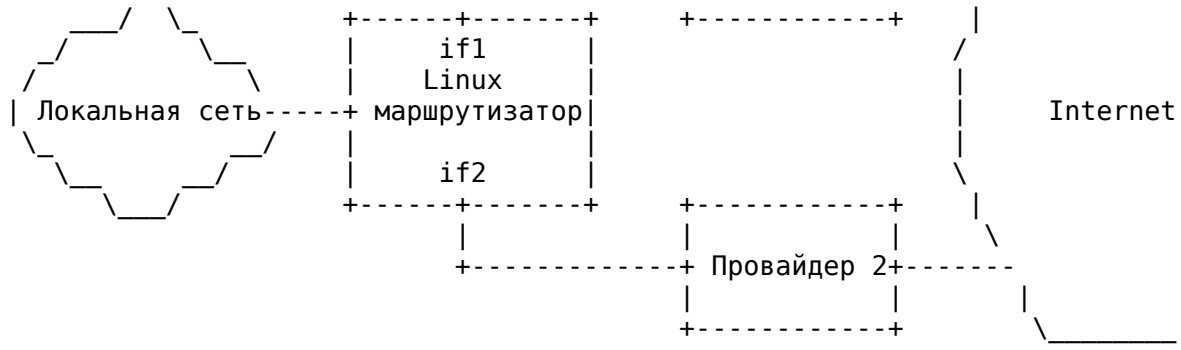
```
# ip route add default via 195.96.98.253 dev ppp2 table John
# ip route flush cache
```

Вот и все. В качестве упражнения вы можете добавить это в скрипт ip-up.

4.2. Маршрутизация через несколько каналов/провайдеров.

Ниже представлена обычная конфигурация, когда локальная сеть (или даже одна машина) подключена к Internet через двух провайдеров.





В этом случае обычно возникает два вопроса.

4.2.1. Раздельный доступ

Первый вопрос заключается в том, как организовать маршрутизацию таким образом, чтобы ответы на запросы, приходящие через определенного провайдера, скажем ровайдера 1, уходили через того же провайдера.

Давайте определим некоторые переменные. Пусть `$IF1` будет именем первого интерфейса (if1 на рисунке), а `$IF2` -- именем второго. Тогда `$IP1` будет IP адресом `$IF1` , а `$IP2` -- IP адресом `$IF2` . Далее, `$P1` это IP-адрес шлюза провайдера 1, а `$P2` -- IP адрес шлюза провайдера 2. Наконец, `$P1_NET` это IP сеть, к которой принадлежит `$P1` , а `$P2_NET` -- сеть, к которой принадлежит `$P2` .

Создадим две дополнительные таблицы маршрутизации, скажем `T1` и `T2`. Добавим их в файл `/etc/iproute2/rt_tables`. Теперь можно настроить эти таблицы следующими командами:

```
ip route add $P1_NET dev $IF1 src $IP1 table T1
ip route add default via $P1 table T1
ip route add $P2_NET dev $IF2 src $IP2 table T2
ip route add default via $P2 table T2
```

Ничего особо эффектного, маршрут к шлюзу и маршрут по-умолчанию через этот шлюз. Точно так же, как и в случае одного провайдера, но по таблице на каждого провайдера. Заметьте, что маршрута к сети, в которой находится шлюз достаточно, потому что он определяет как найти все хосты в этой сети, включая сам шлюз.

Теперь нужно настроить главную таблицу маршрутизации. Хорошо бы маршрутизировать пакеты для сетей провайдеров через соответствующие интерфейсы. Обратите внимание на аргумент `src', который обеспечивает правильный выбор исходного IP-адреса.

```
ip route add $P1_NET dev $IF1 src $IP1
ip route add $P2_NET dev $IF2 src $IP2
```

Теперь задаем маршрут по умолчанию:

```
ip route add default via $P1
```

Зададим правила маршрутизации. Они будут отвечать за то, какая таблица будет использоваться

при маршрутизации. Вы хотите, чтобы пакет с определенным адресом источника маршрутизировался через соответствующий интерфейс:

```
ip rule add from $IP1 table T1
ip rule add from $IP2 table T2
```

Этот набор команд обеспечивает маршрутизацию ответов через интерфейс, на котором был получен запрос.

Заметка читателя Рода Роака (Rod Roark): ' если `$P0_NET` это локальная сеть, а `$IF0` -- соответствующий ей интерфейс, желательно задать следующие команды:

```
ip route add $P0_NET     dev $IF0 table T1
ip route add $P2_NET     dev $IF2 table T1
ip route add 127.0.0.0/8 dev lo   table T1
ip route add $P0_NET     dev $IF0 table T2
ip route add $P1_NET     dev $IF1 table T2
ip route add 127.0.0.0/8 dev lo   table T2
```

Итак, мы рассмотрели очень простой пример. Он будет работать для всех процессов, выполняющихся на маршрутизаторе и для локальной сети, если настроено преобразование адресов (NAT/masquerading). В противном случае, вам будет необходим диапазон IP адресов обоих провайдеров, или выполнять маскирование для одного из провайдеров. В любом случае, вы можете задать правила выбора провайдера для каждого конкретного адреса вашей локальной сети.

4.2.2. Распределение нагрузки.

Второй вопрос заключается в балансировке нагрузки между двумя провайдерами. Это не сложно, если у вас уже настроен раздельный доступ, описанный в предыдущем разделе.

Вместо выбора одного из провайдеров в качестве маршрута по-умолчанию, вы настраиваете т.н. многолучевой (multipath) маршрут. В стандартном ядре это обеспечит балансировку нагрузки между двумя провайдерами. Делается это следующим образом (повторюсь, мы основываемся на примере из раздела Раздельный доступ):

```
ip route add default scope global nexthop via $P1 dev $IF1 weight 1 \
nexthop via $P2 dev $IF2 weight 1
```

Результатом команды будет попеременный выбор маршрута по-умолчанию. Вы можете изменить параметр *`weight`*, так чтобы один из провайдеров получал большую нагрузку.

Обратите внимание, что балансировка не будет идеальной, так как она основывается на маршрутах, а маршруты кэшируются. Это означает, что маршруты к часто посещаемым сайтам не будут проходить через разных провайдеров.

Если вы действительно интересуетесь этим, вам стоит посмотреть на патчи Юлиана Анастасова (Julian Anastasov), расположеные по адресу http://www.ssi.bg/~ja/#routes. Они могут вам помочь.

Глава 5. GRE и другие тоннели.

В ОС Linux поддерживаются 3 типа тоннелей. Это тоннелирование IP в IP, GRE тоннелирование и тоннели не-ядерного уровня (как, например, PPTP).

5.1. Несколько общих замечаний о тоннелях:

Тоннели могут использоваться для очень необычных и интересных вещей. Также они могут усугубить ситуацию, если они сконфигурированы неправильно. Не задавайте маршрут по умолчанию через тоннель, если только вы *ТОЧНО* не уверены в том, что делаете :-). Далее, тоннелирование увеличивает нагрузку на систему и сеть, потому что добовляются дополнительные IP-заголовки. Обычно, это 20 байт на пакет. Таким образом, если обычный размер пакета (MTU) в сети равен 1500 байтам, то при пересылке по тоннелю, пакет может содержать только 1480 байт. Это не обязательно становится проблемой, но помните о необходимости правильной настройки фрагментации пакетов, если вы соединяете большие сети. Ах да, и конечно самый быстрый способ "прорыть" тоннель -- это "рыть" с обоих сторон.

5.2. Тоннелирование IP в IP.

Этот тип тоннелирования доступен в Linux уже давно. Для его работы требуются два модуля ядра: `ipip.o` и `new_tunnel.o`.

Допустим у вас есть три сети: внутренние сети A и B, и промежуточная сеть C (например, Internet). Итак, сеть A:

```
сеть          10.0.1.0
маска         255.255.255.0
маршрутизатор 10.0.1.1
```

Адрес маршрутизатора в сети C -- 172.16.17.18.

сеть B:

```
сеть          10.0.2.0
маска         255.255.255.0
маршрутизатор 10.0.2.1
```

Адрес маршрутизатора в сети C -- 172.19.20.21.

Мы полагаем, что сеть C передает пакеты от A к B и наоборот. Такой сетью может служить даже Internet.

Теперь, что нам нужно сделать?

Убедитесь, что все необходимые модули загружены:

```
insmod ipip.o
insmod new_tunnel.o
```

Теперь на маршрутизаторе сети A выполните:

```
ifconfig tunl0 10.0.1.1 pointopoint 172.19.20.21
route add -net 10.0.2.0 netmask 255.255.255.0 dev tunl0
```

A на маршрутизаторе сети B:

```
ifconfig tunl0 10.0.2.1 pointopoint 172.16.17.18
route add -net 10.0.1.0 netmask 255.255.255.0 dev tunl0
```

Когда вам нужно будет "разрушить" тоннель, выполните:

```
ifconfig tunl0 down
```

Вот и все. Через тоннель IP в IP нельзя передавать широковещательные пакеты или пакеты IPv6. Вы можете только соединить 2 сети IPv4, которые в обычной ситуации не могли бы работать друг с другом. При нынешнем положении вещей, совместимость этого кода доходит до ядер версии 1.3. Насколько я знаю, тоннелирование Linux IP-в-IP не работает с другими операционными системами и маршрутизаторами. Очень простое решение, если оно вам подходит -- используйте его, если вам нужно больше -- используйте GRE.

5.3. GRE тоннели.

GRE это протокол тоннелирования, который был разработан фирмой Cisco. Он может немного больше чем тоннелирование IP-в-IP. Например, вы можете пересылать широковещательную передачу и IPv6 через тоннель GRE.

В ОС Linux вам будет нужен модуль `ip_gre.o`.

5.3.1. Тоннелирование IPv4.

Давайте сначала разберемся с тоннелированием IPv4:

Допустим у вас есть три сети: внутренние сети A и B, и промежуточная сеть C (например, Internet).

Сеть A:

```
сеть          10.0.1.0
маска         255.255.255.0
маршрутизатор 10.0.1.1
```

Адрес маршрутизатора в сети C -- 172.16.17.18. Назовем эту сеть neta (крайне оригинально)

сеть B:

```
сеть          10.0.2.0
маска         255.255.255.0
маршрутизатор 10.0.2.1
```

Адрес маршрутизатора в сети C -- 172.19.20.21. Назовем эту сеть netb

Мы полагаем, что сеть C передает пакеты от A к B и наоборот. Как и почему -- это нас не интересует.

На маршрутизаторе сети A, вам необходимо сделать следующее:

```
ip tunnel add netb mode gre remote 172.19.20.21 local 172.16.17.18 ttl 255
ip link set netb up
ip addr add 10.0.1.1 dev netb
ip route add 10.0.2.0/24 dev netb
```

Давайте немного обсудим эти команды. В первой строке мы добавляем тоннельное устройство и присваиваем ему имя netb (имея при этом ввиду место, куда мы хотим попасть). Потом мы сообщаем, что хотим использовать протокол GRE (mode gre), удаленный адрес 172.19.20.21 (второй маршрутизатор), и адрес с которого должны отправлятся данные, предназначенные для передачи по этому тоннелю -- 172.16.17.18 (это позволяет вашему маршрутизатору иметь несколько IP-адресов в сети C и оставлять возможность выбора конкретного адреса для тоннеля) и, наконец, TTL-поле пакета должно равняться 255 (ttl 255).

Вторая строка переводит устройство в активное состояние.

В третьей строке мы присваиваем созданному интерфейсу born адрес 10.0.1.1. Это нормально для небольших сетей, но когда вы становитель "шахтером" (т.е. создаете МНОГО тоннелей), возможно вам нужно будет выбрать другой диапазон адресов для тоннельных интерфейсов (в этом примере мы могли бы использовать 10.0.3.0).

В четвертой строке определяется маршрут к сети B. Обратите внимание на формат представления сетевой маски. Если вы не знакомы с такой нотацией, краткое пояснение: записываете сетевую маску в двоичной форме и считаете все "единички". Если вы не знаете как это делается, тогда просто запомните, что 255.0.0.0 это /8, 255.255.0.0 -- /16, а 255.255.255.0 -- /24. Да, а 255.255.254.0 выглядит как /23, если вам интересно.

Но хватит об этом, продолжим настройку маршрутизатора сети B.

```
ip tunnel add neta mode gre remote 172.16.17.18 local 172.19.20.21 ttl 255
ip link set neta up
ip addr add 10.0.2.1 dev neta
ip route add 10.0.1.0/24 dev neta
```

Когда захотите уничтожить тоннель -- выполните на маршрутизаторе A:

```
ip link set netb down
ip tunnel del netb
```

Конечно, вы можете изменить netb на neta и выполить это на маршрутизаторе B.

5.3.2. Тоннелирование IPv6.

За кратким описанием адресации IPv6 обратитесь к шестой главе: Тоннелирование IPv6 при помощьи Cisco и/или 6bone.

Продолжим с тоннелями.

Предположим у вас есть сеть IPv6 и вы хотите подключить ее к 6bone, или к другу.

```
Network 3ffe:406:5:1:5:a:2:1/96
```

Ваш адрес IPv4 это 172.16.17.18, а маршрутизатор 6bone имеет адрес 172.22.23.24.

```
ip tunnel add sixbone mode sit remote 172.22.23.24 local 172.16.17.18 ttl 255
ip link set sixbone up
ip addr add 3ffe:406:5:1:5:a:2:1/96 dev sixbone
ip route add 3ffe::/15 dev sixbone
```

Рассмотрим детальнее эти команды. В первой строке мы создали тоннельное устройство с именем sixbone. Тоннелю задан режим sit (что значит тоннелирование IPv6 в IPv4), целевой адрес и адрес источника. TTL установлен в максимальное значение, 255. Далее, мы активируем устройство. После этого задаем наш сетевой адрес и определяем маршрут для 3ffe::/15 (что есть вся сеть 6bone) через тоннель.

Тоннели GRE на сегодняшний день являются самыми предпочтительными. Это стандарт, который широко применяется за пределами сообщества Linux, а потому представляет собой ***Хороший выбор***.

5.4. Тоннели неядерного уровня.

Существует буквально масса реализаций тоннелей неядерного уровня. Наиболее известными являются, конечно, PPP и PPTP, но их много больше (некоторые проприетарные, некоторые высокозащищенные, некоторые даже не используют IP) и это определенно выходит за рамки этого документа HOWTO.

Глава 6. Тоннелирование IPv6 при помощьи Cisco и/или 6bone.

Марко Давидс (Marco Davids) <marco@sara.nl>

Мэйнтейнеру:

Насколько я понимаю, этот способ тоннелирования IPv6-IPv4 по определению не GRE-тоннелирование. Вы можете тоннелировать IPv6 по IPv4 с помощью тоннельных устройств GRE (GRE тоннелирует что угодно через IPv4), но в этом варианте используются интерфейс ("sit")

тоннелирующий только IPv6 через IPv4, а потому это что-то другое.

6.1. Тоннелирование IPv6.

Это еще однин вариант применения возможностей тоннелирования Linux. Он популярен среди пионеров во внедрении протокола IPv6. Практический пример, описанный ниже, конечно не единственный способ организовать тоннелирование IPv6. Однако, этот метод часто используется при создании тоннеля между Linux и маршрутизатором Cisco. Опыт говорит, что многих пользователей интересует именно этот. Десять к одному -- это относится и к вам ;-)

Небольшое отступление об адресах IPv6:

По сравнению с адресами IPv4, адреса IPv6 действительно большие: 128 бит по сравнению с 32 битами. Это дает нам именно то, что нужно -- много, очень много IP-адресов, если быть более точным: 340,282,266,920,938,463,463,374,607,431,768,211,465. Кроме этого, IPv6 (или IPng, сокращение от IP Next Generation) позволяет уменьшить таблицы маршрутизации на магистральных маршрутизаторах Internet, упростить конфигурацию оборудования, увеличить безопасность на уровне IP и улучшить качество обслуживания (QoS).

Пример: 2002:836b:9820:0000:0000:0000:836b:9886

Использование адресов IPv6 может оказазаться весьма трудным. Потому, существуют некоторые правила:

- Не используйте лидирующие нули. Аналогично с IPv4.
- Используйте двоеточия для отделения каждой 16-битной (2-байтной) группы .
- Если в адресе есть последовательность нулей, ее можно записать как ::. Это можно сделать только один раз в адресе и только для кратных 16-ти битам последовательностей.

Так, например, адрес 2002:836b:9820:0000:0000:0000:836b:9886 может быть записан как 2002:836b:9820::836b:9886, что гораздо короче и удобней.

Другой пример, адрес 3ffe:0000:0000:0000:0000:0020:34A1:F32C может быть записан как 3ffe::20:34A1:F32C, что значительно короче.

IPv6 предназначен для замены текущего IPv4. Поскольку это относительно новая технология, пока не существует всемирной сети, где в качестве основного протокола используется IPv6. Для быстрейшего продвижения этой технологии, была создана 6bone.

Сети основанные на IPv6 соединяются одна с другой инкапсуляцией IPv6 в IPv4 и пересылкой по существующей инфраструктуре IPv4.

Именно здесь применяются тоннели.

Для того, чтобы использовать IPv6, наше ядро должно иметь его поддержку. Существует много хороших описаний, как это сделать. Но все сводится к нескольким шагам:

- Возьмите достаточно новую версию дистрибутива Linux, с соответствуюшей glibc.
- Обновите исходные тексты ядра.

После этого можно приступать к сборке ядра, с включенной поддержкой IPv6:

- Перейдите в каталог `/usr/src/linux` и запустите:
- **make menuconfig**
- Перейдите в раздел "Networking Options"
- Отметьте пункты "The IPv6 protocol", "IPv6: enable EUI-64 token format", "IPv6: disable provider based addresses"

Не используйте "модули". Часто они плохо работают.

Другими словами, код поддержки IPv6 должен быть связан с ядром статически. Теперь сохраните конфигурацию и соберите ядро.

Перед сборкой измените строку в Makefile: EXTRAVERSION = -x ; --> ;
EXTRAVERSION = -x-IPv6

Есть большое количество хорошей документации по сборке и установке ядра, но это выходит за рамки данного документа. Если вы будете испытывать трудности на этом этапе, обратитесь к соответствующей документации.

Начать стоит с файла `/usr/src/linux/README`. После того, как вы разберетесь с пересборкой ядра и перезагрузитесь с новым ядром, выполните команду **/sbin/ifconfig -a**. Вы должны будете увидеть новый класс устройств `sit0-device`, где SIT означает Simple Internet Transition. Поздравляю, вы сделали большой шаг в направлении IP Next Generation ;-)

Теперь следующий шаг. Вы хотите подключить ваш хост или сеть к другой сети IPv6. Это может быть "6bone", которая и была создана для таких задач.

Предположим, что у вас имеется сеть IPv6 3ffe:604:6:8::/64 и вы хотите подключить ее к 6bone или аналогичной ей. Обратите внимание, что нотация /64 означает тоже самое, что и в обычном IP вдресе.

Ваш адрес IPv4 -- 145.100.24.181, а адрес маршрутизатора 6bone -- 145.100.1.5

```
# ip tunnel add sixbone mode sit remote 145.100.1.5 [local 145.100.24.181 ttl 255]
# ip link set sixbone up
# ip addr add 3FFE:604:6:7::2/126 dev sixbone
# ip route add 3ffe::0/16 dev sixbone
```

Обсудим приведенные команды. Первой строкой создается интерфейс тоннеля с именем sixbone и определяется его режим -- sit (т.е. тоннелирование IPv6 в IPv4), а так же указываются адреса отправителя и получателя. Напоследок устанавливается максимальный TTL -- 255.

В следующей строке интерфейс активируется (up). Далее, добавляется наш сетевой адрес и задается маршрут к сети 3ffe::/15 (что на сегодняшний день является всей 6bone) через тоннель. Если машина, на которой выполняются эти команды, является шлюзом IPv6, то нужно будет выполнить еще и такие команды:

```
# echo 1 >/proc/sys/net/ipv6/conf/all/forwarding
# /usr/local/sbin/radvd
```

radvd, как и zebra, демон маршутизации, поддерживающий автоконфигурационные возможности IPv6. Более детальную информацию ищите в Internet. Проверить настройку системы можно командой:

```
# /sbin/ip -f inet6 addr
```

Если на вашем шлюзе IPv6 запущен radvd и вы загрузите в вашей сети систему Linux с поддержкой IPv6, то сможете оценить возможности автонастройки IPv6:

```
# /sbin/ip -f inet6 addr
1: lo: <LOOPBACK,UP> mtu 3924 qdisc noqueue inet6 ::1/128 scope host

3: eth0: <BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast qlen 100
inet6 3ffe:604:6:8:5054:4cff:fe01:e3d6/64 scope global dynamic
valid_lft forever preferred_lft 604646sec inet6 fe80::5054:4cff:fe01:e3d6/10
scope link
```

Теперь можно сделать следующий шаг и настроить bind для адресов IPv6. У типа A существует эквиваоент для IPv6: AAAA. Для in-addr.arpa эквивалентом является: ip6.int. По данной теме доступно множество информации.

Существует много приложений поддерживающих IPv6 и их количество все время увеличивается. Это secure shell, telnet, inetd, браузер Mozilla, веб-сервер Apache и многие другие. Но все это выходит за рамки данного документа ;-)

На стороне Cisco конфигурация должна выглядеть примерно так:

```
!
interface Tunnel1
description IPv6 tunnel
no ip address
no ip directed-broadcast
ipv6 address 3FFE:604:6:7::1/126
tunnel source Serial0
tunnel destination 145.100.24.181
tunnel mode ipv6ip
!
ipv6 route 3FFE:604:6:8::/64 Tunnel1
```

Если в вашем распоряжении нет Cisco, можно попробовать использовать тоннельные брокеры IPv6, которых сейчас в Internet существует предостаточное количество. В большинстве случаев они не откажут вам в создании дополнительного тоннеля к вам на своих маршрутизаторах Cisco. Чаще всего это можно сделать при помощи веб-интерфейса. Поищите фразу "ipv6 tunnel broker" в вашей любимой поисковой системе.

Глава 7. IPSEC: безопасная передача данных протоколами IP через Интернет

На сегодняшний день существует две реализации IPSEC в Linux. Для ядер 2.2 и 2.4 -- это пакет *FreeS/WAN*, который является первой основной реализацией IPSEC. Этот проект имеет два сайта:

официальный -- http://www.freeswan.org/ и неофициальный -- http://www.freeswan.ca/. *FreeS/WAN* не вошел в состав ядра по ряду причин. Наиболее часто упоминается причина "политического" характера, связанная с Американскими законодательством, запрещающим экспорт технологий шифрования. Кроме того, этот пакет довольно трудно интегрируется в ядро Linux, что еще больше осложняет их слияние.

В добавок ко всему, многие высказывают обеспокоенность качеством кода. В Интернет существует достаточное количество документации, описывающей процесс установки *FreeS/WAN*.

Начиная с версии 2.5.47, в ядре Linux появилась встроенная поддержка IPSEC. Реализация ее была выполнена Алексеем Кузнецовым и Дэйвом Миллером (Dave Miller), на основе разработок USAGI IPv6 group. С этой версии, CryptoAPI Джеймса Морриса (James Morris), так же стал частью ядра -- его средствами выполняется собственно шифрование.

Этот документ будет описывать только IPSEC версии 2.5 (и выше). *FreeS/WAN* рекомендуется для пользователей Linux 2.4, но вам следует знать, что его конфигурирование сильно отличается от "родного" IPSEC. И наверное следует упомянуть о существовании "заплаты", которая делает возможной работу пользовательских утилит *FreeS/WAN* с "родным" для Linux IPSEC.

Начиная с версии ядра Linux 2.5.49, для работы IPSEC уже не требуется наложения заплат.

Пользовательские утилиты, для работы с IPSEC, вы найдете по адресу: http://sourceforge.net/projects/ipsec-tools. В этот пакет, кроме всего прочего, входит ряд программ, основанных на Racoon.

При пересборке ядра обязательно включите опции **PF_KEY**, **AH**, **ESP** и все опции в **CryptoAPI**!

Автор этой главы полный кретин (впрочем у меня есть причины сомневаться в этом -- *прим. перев.*) в области IPSEC! Если в этой главе вы столкнетесь с неизбежными ошибками, пожалуйста, сообщите о них Берту Хьюберту (Bert Hubert), по адресу: <ahu@ds9a.nl>.

Для начала мы рассмотрим -- как вручную настроить безопасное соединение между двумя хостами. Большая часть этого процесса может быть автоматизирована, но мы будем все делать вручную, чтобы показать всю его подноготную.

Если вас интересует только автоматическое формирование ключей, то можете смело пропустить следующий раздел. Однако мы находим полезным знать принципы "ручного" подхода.

7.1. Пример ручного конфигурирования безопасного соединения между двумя хостами.

IPSEC -- довольно сложная тема. В Интернет вы найдете множество разнообразной информации по ней. Этот документ концентрируется на проблемах настройки, запуска и объяснении основных принципов функционирования. Все примеры базируются на Racoon, который уже упоминался выше.

Наиболее распространенные примеры настройки **iptables** не позволяют прохождение пакетов IPSEC! Чтобы сделать это возможным, вам следует добавить следующие

правила: **iptables -A xxx -p 50 -j ACCEPT** и **iptables -A xxx -p 51 -j ACCEPT**.

IPSEC представляет собой безопасную версию протокола IP. Понятие "безопасность", в данном случае, означает возможность шифрования и аутентификации. В чистом виде, с технической точки зрения, "безопасность" означает только шифрование, однако, довольно легко показать, что этого недостаточно -- вы можете обмениваться шифрованными данными, но не иметь при этом гарантий, что удаленная сторона именно та, которую вы ожидаете.

Шифрование, в IPSEC, выполняется протоколом ESP (Encapsulating Security Payload -- Инкапсулированные Защищенные Данные), аутентификация -- протоколом AH (Authentication Header -- Заголовок Аутентификации). Вы можете сконфигурировать их оба, или один из них.

И ESP, и AH опираются на Security Association (защищенный виртуальный канал, или контекст безопасности). Security Association (SA) -- однонаправленное логическое соединение (от отправителя к получателю) между двумя системами, поддерживающими протокол IPSec, которое однозначно идентифицируется следующими тремя параметрами:

- индексом защищенного соединения (Security Parameter Index, SPI -- 32-битная константа, используемая для идентификации различных SA с одинаковыми IP-адресом получателя и протоколом безопасности);
- IP-адресом получателя IP-пакетов (IP Destination Address);
- протоколом безопасности (Security Protocol -- AH или ESP).

Пример создания защищенного канала (SA) для протокола AH может выглядеть следующим образом:

```
add 10.0.0.11 10.0.0.216 ah 15700 -A hmac-md5 "1234567890123456";
```

Эта строка говорит нам, что: "Создается защищенный канал от узла сети с адресом 10.0.0.11, к узлу сети с адресом 10.0.0.216. Что это данные аутентификации, которая выполняется протоколом AH, с использованием алгоритма шифрования hmac-md5 и ключом 1234567890123456.". Эта инструкция помечена индексом SPI -- 15700. Здесь есть один интересный момент -- каждый защищенный виртуальный канал используется для передачи данных только в одном направлении (в данном примере: от 10.0.0.11 к 10.0.0.216). В случае необходимости двустороннего обмена создаются два виртуальных канала. Более того, для каждого из протоколов создаются свои виртуальные защищенные каналы передачи данных. Таким образом, если предполагается двусторонний обмен, с использованием обеих протоколов -- и AH, и ESP, то создаются 4 защищенных канала, по одному на каждое направление для каждого из протоколов.

Пример создания защищенного канала (SA) для протокола ESP:

```
add 10.0.0.11 10.0.0.216 esp 15701 -E 3des-cbc "123456789012123456789012";
```

Этот пример говорит: "Создается защищенный канал от 10.0.0.11 к 10.0.0.216, в котором данные должны шифроваться алгоритмом 3des-cbc с ключом 123456789012123456789012". Индекс SPI -- '15701'.

Фактически, может существовать любое число практически идентичных SA, но при этом, все они отличаются различными индексами SPI. Пока что мы видели только то, как описываются виртуальные каналы безопасности (SA), но до сих пор не встретили ни одного описания политик безопасности (правил обработки получаемых пакетов). Для того, чтобы назначить эти правила, необходимо описать политику безопасности. Она может включать в себя такие действия, как: "обрабатывать пакет с помощью IPSEC, если это возможно" или "сбросить пакет".

Типичный пример назначения политики безопасности:

```
spdadd 10.0.0.216 10.0.0.11 any -P out ipsec
   esp/transport//require
   ah/transport//require;
```

Применительно к хосту 10.0.0.216, это означает, что весь трафик, отправляемый хосту 10.0.0.11 должен быть зашифрован и "обернут" протоколом AH, подтверждающим подлинность. Обратите внимание, этот пример не описывает, какой из имеющихся каналов (SA) должен использоваться, это означает, что право выбора оставляется за ядром.

Другими словами, Политика Безопасности описывает ЧТО следует предпринять в том или ином случае, а защищенный канал (SA) -- КАК получить данные.

Исходящие пакеты "подписываются" соответствующим SPI (КАК). Ядро использует его для нужд шифрования и аутентификации так, чтобы удаленный хост мог выполнить необходимые проверки и дешифрацию.

Ниже следует пример простой конфигурации, обеспечивающей передачу данных от 10.0.0.216 к 10.0.0.11 с аутентификацией, в зашифрованном виде. Обратите внимание на то, что в обратном направлении данные передаются в открытом виде. Это лишь первая версия примера и она не должна рассматриваться вами как пригодная к употреблению.

Для хоста 10.0.0.216:

```
#!/sbin/setkey -f
add 10.0.0.216 10.0.0.11 ah 24500 -A hmac-md5 "1234567890123456";
add 10.0.0.216 10.0.0.11 esp 24501 -E 3des-cbc "123456789012123456789012";

spdadd 10.0.0.216 10.0.0.11 any -P out ipsec
   esp/transport//require
   ah/transport//require;
```

Для хоста 10.0.0.11, те же самые SA, но без политики безопасности:

```
#!/sbin/setkey -f
add 10.0.0.216 10.0.0.11 ah 24500 -A hmac-md5 "1234567890123456";
add 10.0.0.216 10.0.0.11 esp 24501 -E 3des-cbc "123456789012123456789012";
```

В этой конфигурации попробуем дать команду **ping 10.0.0.11** с хоста 10.0.0.216. В результате получим такой вывод от **tcpdump**:

```
22:37:52 10.0.0.216 > 10.0.0.11: AH(spi=0x00005fb4,seq=0xa):
ESP(spi=0x00005fb5,seq=0xa) (DF)
22:37:52 10.0.0.11 > 10.0.0.216: icmp: echo reply
```

Обратите внимание на то, как возвращается ответ на запрос -- он передается в открытом виде. В то время как сам запрос не распознается утилитой **tcpdump**, но зато она показывает SPI для AH и ESP, которые инструктируют хост 10.0.0.11, КАК выполнить проверку подлинности и дешифровать данные.

Следует сделать несколько замечаний по поводу этих примеров. Такая конфигурация не обеспечивает достаточный уровень безопасности. Проблема состоит в том, что политика

безопасности описывает только -- ЧТО должен сделать хост 10.0.0.216 при передаче данных хосту 10.0.0.11 и КАК их передать, а для хоста 10.0.0.11 описывается КАК он может получить эти данные, но нет правил, описывающих ЧТО он должен предпринять при получении нешифрованных пакетов, идущих от 10.0.0.216!

Таким образом, если злоумышленник будет в состоянии выполнить "подмену" (spoofing) IP-адреса, то он сможет отправлять незашифрованные данные хосту 10.0.0.11, который примет их! Для устранения этой проблемы нужно прописать политику безопасности для хоста 10.0.0.11:

```
#!/sbin/setkey -f
spdadd 10.0.0.216 10.0.0.11 any -P IN ipsec
   esp/transport//require
   ah/transport//require;
```

Это описание сообщает хосту 10.0.0.11, что весь входящий трафик от хоста 10.0.0.216 должен иметь подтверждение подлинности (AH) и должен быть зашифрован (ESP).

Ниже приводится полная конфигурация сетевых узлов 10.0.0.11 и 10.0.0.216, для двустороннего обмена данными. Полная конфигурация для хоста 10.0.0.216:

```
#!/sbin/setkey -f
flush;
spdflush;

# AH
add 10.0.0.11 10.0.0.216 ah 15700 -A hmac-md5 "1234567890123456";
add 10.0.0.216 10.0.0.11 ah 24500 -A hmac-md5 "1234567890123456";

# ESP
add 10.0.0.11 10.0.0.216 esp 15701 -E 3des-cbc "123456789012123456789012";
add 10.0.0.216 10.0.0.11 esp 24501 -E 3des-cbc "123456789012123456789012";

spdadd 10.0.0.216 10.0.0.11 any -P out ipsec
           esp/transport//require
           ah/transport//require;

spdadd 10.0.0.11 10.0.0.216 any -P in ipsec
           esp/transport//require
           ah/transport//require;
```

И для 10.0.0.11:

```
 #!/sbin/setkey -f
flush;
spdflush;

# AH
add 10.0.0.11 10.0.0.216 ah 15700 -A hmac-md5 "1234567890123456";
add 10.0.0.216 10.0.0.11 ah 24500 -A hmac-md5 "1234567890123456";

# ESP
add 10.0.0.11 10.0.0.216 esp 15701 -E 3des-cbc "123456789012123456789012";
add 10.0.0.216 10.0.0.11 esp 24501 -E 3des-cbc "123456789012123456789012";


spdadd 10.0.0.11 10.0.0.216 any -P out ipsec
           esp/transport//require
           ah/transport//require;
```

```
spdadd 10.0.0.216 10.0.0.11 any -P in ipsec
           esp/transport//require
           ah/transport//require;
```

Обратите внимание -- в этом примере использованы идентичные ключи шифрования для обоих направлений. Однако, это не является обязательным требованием.

Чтобы проверить только что созданную конфигурацию, достаточно дать команду **setkey -D**, которая выведет перечень контекстов защиты (виртуальных защищенных каналов) или **setkey -DP** , которая отобразит сконфигурированные политики безопасности.

7.2. Пример автоматического конфигурирования безопасного соединения между двумя хостами.

В предыдущем разделе, шифрование было сконфигурировано с применением простых ключей. По идее, чтобы обеспечить необходимый уровень безопасности, мы должны бы передавать сведения о конфигурации по надежным каналам. Если бы нам пришлось настраивать удаленный хост через **telnet**, то любое третье лицо запросто могло бы получить секретные сведения, и такая конфигурация будет далеко не безопасна.

Кроме того, как только секретная информация становится известной кому-либо, она перестает быть секретной. Знание секретных сведений даст не так много удаленному пользователю, но мы должны быть абсолютно уверены в том, что каналы связи с нашими партнерами действительно надежно защищены. Эта уверенность требует большого количества ключей, если у нас есть 10 партнеров, то необходимо иметь не менее 50 различных ключей.

Помимо проблемы, связанной с необходимостью согласования ключей, существует также необходимость в периодическом их изменении. Если третья сторона сможет перехватить наш трафик, то рано или поздно она будет в состоянии "расколоть" ключ. Это может быть предотвращено за счет периодического изменения ключей, но этот процесс уже требует автоматизации.

Другая проблема состоит в том, что при работе с ключевой информацией "вручную", как это описано выше, мы заранее точно определяем алгоритмы и используемую длину ключа, что в свою очередь требует тесной координации с удаленной стороной. Желательно было бы иметь возможность определения более широкой политики назначения ключей, например так: "Мы можем использовать алгоритмы 3DES и Blowfish, с длиной ключа не менее, чем...".

Решение этих проблем берет на себя Протокол Обмена Ключами -- IKE (Internet Key Exchange), позволяющий обмениваться сгенерированными, автоматически и случайным образом, ключами. Передача ключей осуществляется с помощью асимметричной технологии кодирования, в соответствии с предопределенными алгоритмами.

В Linux IPSEC 2.5, реализация этих возможностей выполнена в виде демона KAME 'racoon' IKE. Начиная с 9 ноября 2003 года, доступны исходные тексты **racoon**, в пакете **iptools**, распространяемом Алексеем, хотя, возможно, вам придется удалить строки `#include <net/route.h>` в двух файлах. В качестве альтернативы я могу предложить откомпилированную версию.

Протокол IKE работает через UDP порт 500, предоставьте ему такую возможность, внеся соответствующие изменения в ваш набор правил для **iptables**.

7.2.1. Теория.

Как уже говорилось ранее, в случае автоматической настройки, обновление и обмен ключевой информацией выполняется без нашего участия. Очевидно, что при этом "на лету" создаются защищенные каналы (SA), которые, как это ни странно, не обеспечиваются какой-либо политикой безопасности.

Таким образом, чтобы воспользоваться преимуществами IKE, необходимо установить для него политику безопасности. Для этого создается такая политика, которая не связана с каким-либо конкретным защищенным каналом (SA). Когда ядро обнаружит такую политику, то оно передаст ее демону IKE, который в свою очередь будет использовать ее в своей работе.

Повторюсь еще раз: Политика Безопасности определяет -- ЧТО следует предпринять в том или ином случае, а Контекст Безопасности (защищенный канал) определяет -- КАК производить обмен данными. Автоматическое управление ключевой информацией позволяет нам избежать неприятностей только с определением "ЧТО предпринять".

7.2.2. Пример.

Kame racoon имеет достаточно неплохие настройки по-умолчанию, так что мы не будем их касаться. Как уже говорилось выше, мы должны определить только политику безопасности, оставив право на создание защищенных каналов за демоном IKE.

В этом примере, мы опять вернемся к нашим хостам 10.0.0.11 и 10.0.0.216, и еще раз попробуем установить защищенное соединение между ними, но на сей раз с помощью **racoon**. Для упрощения конфигурации будем использовать предопределенные ключи, сертификаты X.509 будут обсуждаться в отдельном разделе (см. раздел Автоматизация с использованием сертификатов X.509).

Мы будем придерживаться конфигурации, заданной практически по-умолчанию и идентичной для обоих хостов:

```
path pre_shared_key "/usr/local/etc/racoon/psk.txt";

remote anonymous
{
        exchange_mode aggressive,main;
        doi ipsec_doi;
        situation identity_only;

        my_identifier address;

        lifetime time 2 min;   # sec,min,hour
        initial_contact on;
        proposal_check obey;    # obey -- повиноваться, требованиям и ограничениям

        proposal {
                encryption_algorithm 3des;
```

```
                hash_algorithm sha1;
                authentication_method pre_shared_key;
                dh_group 2 ;
        }
}

sainfo anonymous
{
        pfs_group 1;
        lifetime time 2 min;
        encryption_algorithm 3des ;
        authentication_algorithm hmac_sha1;
                compression_algorithm deflate ;
}
```

Многие из параметров настройки, на мой взгляд, могут быть удалены, поскольку они достаточно близки к заданным по-умолчанию. Здесь приведены два анонимных параметра, которые применяются ко всем удаленным хостам, за счет чего достигается простота конфигурации. На данный момент пока нет особой потребности в создании настроек для каждого конкретного хоста.

Кроме того, в настройках задана самоидентификация на основе собственного IP-адреса (`my_identifier address`). Затем объявляются используемые алгоритмы шифрования -- 3des и sha1 и указывается, что будут использоваться предопределенные ключи, расположенные в файле `psk.txt`.

Содержимое файлов `psk.txt` с ключами приводится ниже. Для разных хостов они различны. Для хоста 10.0.0.11:

```
10.0.0.216      password2
```

Для хоста 10.0.0.216:

```
10.0.0.11       password2
```

Назначте владельцем файла суперпользователя (root), и установите права доступа 0600, в противном случае **racoon** откажется доверять их содержимому.

Теперь можно приступить к формированию политик безопасности, которые в данной ситуации достаточно просты и незамысловаты. На хосте 10.0.0.216:

```
#!/sbin/setkey -f
flush;
spdflush;

spdadd 10.0.0.216 10.0.0.11 any -P out ipsec
        esp/transport//require;

spdadd 10.0.0.11 10.0.0.216 any -P in ipsec
        esp/transport//require;
```

На хосте 10.0.0.11:

```
#!/sbin/setkey -f
flush;
spdflush;

spdadd 10.0.0.11 10.0.0.216 any -P out ipsec
        esp/transport//require;

spdadd 10.0.0.216 10.0.0.11 any -P in ipsec
```

```
        esp/transport//require;
```

Теперь все готово к запуску **racoon**! После того, как он будет запущен, попробуем установить сеанс связи, через **telnet**, с хоста 10.0.0.11 на хост 10.0.0.216, впрочем можно и наоборот. **racoon** тут же начнет "переговоры" с удаленным хостом:

```
12:18:44: INFO: isakmp.c:1689:isakmp_post_acquire(): IPsec-SA
  request for 10.0.0.11 queued due to no phase1 found.
12:18:44: INFO: isakmp.c:794:isakmp_ph1begin_i(): initiate new
  phase 1 negotiation: 10.0.0.216[500]<=>10.0.0.11[500]
12:18:44: INFO: isakmp.c:799:isakmp_ph1begin_i(): begin Aggressive mode.
12:18:44: INFO: vendorid.c:128:check_vendorid(): received Vendor ID:
  KAME/racoon
12:18:44: NOTIFY: oakley.c:2037:oakley_skeyid(): couldn't find
  the proper pskey, try to get one by the peer's address.
12:18:44: INFO: isakmp.c:2417:log_ph1established(): ISAKMP-SA
  established 10.0.0.216[500]-10.0.0.11[500] spi:044d25dede78a4d1:ff01e5b4804f0680
12:18:45: INFO: isakmp.c:938:isakmp_ph2begin_i(): initiate new phase 2
  negotiation: 10.0.0.216[0]<=>10.0.0.11[0]
12:18:45: INFO: pfkey.c:1106:pk_recvupdate(): IPsec-SA established:
  ESP/Transport 10.0.0.11->10.0.0.216 spi=44556347(0x2a7e03b)
12:18:45: INFO: pfkey.c:1318:pk_recvadd(): IPsec-SA established:
  ESP/Transport 10.0.0.216->10.0.0.11 spi=15863890(0xf21052)
```

Если теперь дать команду **setkey -D**, чтобы просмотреть список созданных защищенных каналов, мы получим такой вывод:

```
10.0.0.216 10.0.0.11
        esp mode=transport spi=224162611(0x0d5c7333) reqid=0(0x00000000)
        E: 3des-cbc  5d421c1b d33b2a9f 4e9055e3 857db9fc 211d9c95 ebaead04
        A: hmac-sha1  c5537d66 f3c5d869 bd736ae2 08d22133 27f7aa99
        seq=0x00000000 replay=4 flags=0x00000000 state=mature
        created: Nov 11 12:28:45 2002   current: Nov 11 12:29:16 2002
        diff: 31(s)     hard: 600(s)    soft: 480(s)
        last: Nov 11 12:29:12 2002      hard: 0(s)      soft: 0(s)
        current: 304(bytes)     hard: 0(bytes)  soft: 0(bytes)
        allocated: 3    hard: 0 soft: 0
        sadb_seq=1 pid=17112 refcnt=0
10.0.0.11 10.0.0.216
        esp mode=transport spi=165123736(0x09d79698) reqid=0(0x00000000)
        E: 3des-cbc  d7af8466 acd4f14c 872c5443 ec45a719 d4b3fde1 8d239d6a
        A: hmac-sha1  41ccc388 4568ac49 19e4e024 628e240c 141ffe2f
        seq=0x00000000 replay=4 flags=0x00000000 state=mature
        created: Nov 11 12:28:45 2002   current: Nov 11 12:29:16 2002
        diff: 31(s)     hard: 600(s)    soft: 480(s)
        last:                           hard: 0(s)      soft: 0(s)
        current: 231(bytes)     hard: 0(bytes)  soft: 0(bytes)
        allocated: 2    hard: 0 soft: 0
        sadb_seq=0 pid=17112 refcnt=0
```

А команда **setkey -DP** -- список политик безопасности, даст такой результат:

```
10.0.0.11[any] 10.0.0.216[any] tcp
        in ipsec
        esp/transport//require
        created:Nov 11 12:28:28 2002 lastused:Nov 11 12:29:12 2002
        lifetime:0(s) validtime:0(s)
        spid=3616 seq=5 pid=17134
        refcnt=3
10.0.0.216[any] 10.0.0.11[any] tcp
        out ipsec
        esp/transport//require
```

```
        created:Nov 11 12:28:28 2002 lastused:Nov 11 12:28:44 2002
        lifetime:0(s) validtime:0(s)
        spid=3609 seq=4 pid=17134
        refcnt=3
```

7.2.2.1. Известные проблемы и недостатки.

Если этот вариант у вас не работает, проверьте -- все ли файлы конфигурации принадлежат суперпользователю (root) и доступны на чтение только ему. Чтобы запустить **racoon** в приоритетном режиме, используйте ключ '-F'. Чтобы указать ему местоположение файла конфигурации -- ключ '-f'. Для того, чтобы увеличить детальность, добавьте инструкцию 'log debug;' в `racoon.conf`.

7.2.3. Автоматизация с использованием сертификатов X.509

Как уже говорилось, использование предопределенных ключей осложнено необходимостью тесной координации с удаленной стороной, что не всегда бывает удобно. Кроме того, при согласовании ключевой информации она становится известной нескольким лицам (по крайней мере двоим), а секрет, который знают несколько человек, перестает быть секретом. К счастью существуют асимметричные технологии кодирования, которые помогают снять эту проблему.

Если каждая из сторон, использующих IPSEC, создаст открытый и секретный ключи, то обе стороны, обменявшись своими открытыми ключами и настроив политику безопасности, смогут установить защищенное соединение.

Процедура создания ключей относительно проста, хотя и требует выполнения некоторых дополнительных действий. Ниже рассматривается пример использования, для этих целей, утилиты **openssl**.

7.2.3.1. Создание собственного сертификата X.509.

OpenSSL обладает разветвленной инфраструктурой поддержки ключей, используемых для подтверждения сертификатов подлинности. Прямо сейчас, мы с вами пройдем всю процедуру создания сертификатов и настройки защищенного соединения.

Для начала создадим сертификат для нашего хоста, с именем laptop. Начнем с генерации "запроса на сертификат":

```
$ openssl req -new -nodes -newkey rsa:1024 -sha1 -keyform PEM -keyout \
  laptop.private -outform PEM -out request.pem
```

Здесь будет предложено ответить на ряд вопросов:

```
Country Name (2 letter code) [AU]:NL
State or Province Name (full name) [Some-State]:.
Locality Name (eg, city) []:Delft
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Linux Advanced
Routing & Traffic Control
```

```
Organizational Unit Name (eg, section) []:laptop
Common Name (eg, YOUR name) []:bert hubert
Email Address []:ahu@ds9a.nl

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

Заполните поля запроса по своему усмотрению.

Теперь создадим собственно сертификат, подписанный самим собой:

```
$ openssl x509 -req -in request.pem -signkey laptop.private -out \
  laptop.public
Signature ok
subject=/C=NL/L=Delft/O=Linux Advanced Routing & Traffic \
  Control/OU=laptop/CN=bert hubert/Email=ahu@ds9a.nl
Getting Private key
```

После этого файл `request.pem` можно удалить.

Повторите эту процедуру для всех ваших компьютеров. Вы можете свободно передавать сгенерированные открытые ключи (файлы `.public`), но сохраняйте файлы `.private` в секрете!

7.2.3.2. Настройка и запуск.

Теперь, после того как мы создали открытый и секретный ключи, мы можем передать их **racoon**.

Вернемся к нашей предыдущей конфигурации хостов 10.0.0.11 ('upstairs') и 10.0.0.216 ('laptop').

В файл `racoon.conf`, на 10.0.0.11, добавим:

```
path certificate "/usr/local/etc/racoon/certs";

remote 10.0.0.216
{
        exchange_mode aggressive,main;
        my_identifier asn1dn;
        peers_identifier asn1dn;

        certificate_type x509 "upstairs.public" "upstairs.private";

        peers_certfile "laptop.public";
        proposal {
                encryption_algorithm 3des;
                hash_algorithm sha1;
                authentication_method rsasig;
                dh_group 2 ;
        }
}
```

Тем самым сообщив **racoon**, что сертификаты находятся в каталоге `/usr/local/etc/racoon/certs/`. Кроме того, добавим раздел, описывающий удаленный компьютер 10.0.0.216.

Строки *asn1dn* говорят о том, что локальный и удаленный идентификаторы следует извлекать из открытых ключей. Это -- 'subject=/C=NL/L=Delft/O=Linux Advanced Routing & Traffic Control/OU=laptop/CN=bert hubert/Email=ahu@ds9a.nl' из листинга, приведенного выше.

Строка **certificate_type** указывает имена файлов с локальными открытым и секретным ключами. **peers_certfile** указывает, что открытый ключ удаленного узла следует взять из файла `laptop.public`.

Блок **proposal** остается, по сравнению с предыдущей конфигурацией, практически без изменений, за исключением **authentication_method** для которого теперь указано **rsasig**, что означает -- использовать для аутентификации RSA открытый/секретный ключи.

Аналогичные изменения вносятся и в конфигурацию хоста 10.0.0.216:

```
path certificate "/usr/local/etc/racoon/certs";

remote 10.0.0.11
{
        exchange_mode aggressive,main;
        my_identifier asn1dn;
        peers_identifier asn1dn;

        certificate_type x509 "laptop.public" "laptop.private";

        peers_certfile "upstairs.public";

        proposal {
                encryption_algorithm 3des;
                hash_algorithm sha1;
                authentication_method rsasig;
                dh_group 2 ;
        }
}
```

После внесения изменений в конфигурацию, нам необходимо разместить файлы ключей. На машине 'upstairs', в каталоге `/usr/local/etc/racoon/certs`, следует разместить файлы `upstairs.private`, `upstairs.public` и `laptop.public`.

ВНИМАНИЕ! Владельцем этого каталога должен быть суперпользователь (root) и ему должны быть назначены права доступа 0700, иначе **racoon** откажется работать с ним!

А на машине 'laptop', опять же в каталоге `/usr/local/etc/racoon/certs`, нужно разместить файлы `laptop.private`, `laptop.public` и `upstairs.public`. Короче говоря, на каждом хосте должны размещаться его собственные открытый и секретный ключи, а так же открытые ключи удаленных систем.

Проверьте настройки политик безопасности (выполните строки **spdadd**, из раздела Пример). Теперь запустите **racoon** -- все должно работать.

7.2.4. Как сохранить настройки туннеля в безопасности.

Чтобы иметь возможность установить защищенное соединение с удаленной стороной, необходимо обменяться открытыми ключами. Несмотря на то, что открытый ключ можно распространять свободно, очень важно сохранить его целостность. Другими словами, всегда следует проверять --

не получили ли вы "кота в мешке".

Сделать это довольно просто, OpenSSL имеет команду **digest**, которая вычисляет контрольную сумму файла с ключом:

```
$ openssl dgst upstairs.public
MD5(upstairs.public)= 78a3bddafb4d681c1ca8ed4d23da4ff1
```

После получения вашего ключа, удаленная сторона так же вычисляет его контрольную сумму и, если они совпали (сообщить свою контрольную сумму вы может при личной встрече или по телефону), то все в порядке!

Другой вариант -- воспользоваться услугами третьей доверенной стороны -- Certificate Authority, которая может подписать созданные вами сертификаты.

7.3. Туннели IPSEC

Выше мы обсуждали работу с IPSEC в так называемом *транспортном* режиме, где обе стороны взаимодействуют друг с другом посредством протоколов IPSEC. Однако, это довольно редкий случай, чаще встречается другой вариант -- когда в транспортном режиме работают только маршрутизаторы, обеспечивая защищенными каналами связи компьютеры, находящиеся за ними. Такой режим работы называется *туннельным режимом*.

Настройка такого режима выполняется достаточно просто. Предположим, что нам необходимо "проложить" туннель от хоста, с адресом 10.0.0.216, к хосту, с адресом 10.0.0.11, через сеть 130.161.0.0/16. Для этого, на хосте 10.0.0.216 выполним следующие действия:

```
#!/sbin/setkey -f
flush;
spdflush;

add 10.0.0.216 10.0.0.11 esp 34501
        -m tunnel
        -E 3des-cbc "123456789012123456789012";

spdadd 10.0.0.0/24 130.161.0.0/16 any -P out ipsec
           esp/tunnel/10.0.0.216-10.0.0.11/require;
```

Обратите внимание на параметр *-m tunnel* -- это очень важно. Сначала конфигурируется шифрование протоколом ESP для защищенного канала (SA) между конечными точками туннела -- 10.0.0.216 и 10.0.0.11

Затем создается собственно туннель. Эта инструкция указывает ядру на необходимость шифрования всего трафика, который маршрутизируется из сети 10.0.0.0/24 в сеть 130.161.0.0/16. И наконец этот трафик должен быть отправлен хосту 10.0.0.11.

На компьютере 10.0.0.11 также необходимо выполнить некоторую настройку:

```
#!/sbin/setkey -f
flush;
spdflush;

add 10.0.0.216 10.0.0.11 esp 34501
        -m tunnel
```

```
        -E 3des-cbc "123456789012123456789012";

spdadd 10.0.0.0/24 130.161.0.0/16 any -P in ipsec
           esp/tunnel/10.0.0.216-10.0.0.11/require;
```

Если вы были внимательны, то наверняка заметили, что конфигурации обоих узлов сети практически одинаковые. Исключение составляет аргумент *-P out*, который для 10.0.0.11 изменился на *-P in*. В отличие от предыдущих примеров, на этот раз мы настроили передачу данных только в одном направлении. "Постройку" второй половины туннеля оставляем вам, в качестве самостоятельного упражнения.

У такой конфигурации есть еще одно название -- *proxy ESP*, которое более точно отражает ее назначение.

Для того, чтобы туннель заработал, в ядре должна быть разрешена возможность форвардинга (IP Forwarding)!

7.4. Другое программное обеспечение для работы с IPSEC

Томас Уолпаски (Thomas Walpuski) написал "заплату" для **isakpmd** (из OpenBSD), которая делает возможной совместную работу этого пакета и Linux 2.5 IPSEC. И даже больше! Теперь исходный код **isakpmd** в CVS уже содержит все необходимые изменения! Дополнительную информацию по этой теме вы найдете по адресу: http://bender.thinknerd.de/%7Ethomas/IPsec/isakmpd-linux.html.

isakpmd очень сильно отличается от **racoon**, но он многим нравится. Вы сможете найти его здесь: http://www.openbsd.org/cgi-bin/cvsweb/src/sbin/isakmpd/. Получить более подробную информацию об OpenBSD CVS -- здесь: http://www.openbsd.org/anoncvs.html. Томас также собрал тарболл (http://bender.thinknerd.de/%7Ethomas/IPsec/isakmpd.tgz) для тех, кто лишен возможности работы с CVS.

Кроме того, имеются "заплаты", которые обеспечивают взаимодействие **FreeS/WAN** с Linux 2.5 IPSEC. Вы найдете их здесь: http://gondor.apana.org.au/%7Eherbert/freeswan/.

7.5. Взаимодействие с другими операционными системами через IPSEC.

FIXME: Этот раздел ждет своего писателя!

7.5.1. Windows.

Андреас Йеллингаус (Andreas Jellinghaus) <aj@dungeon.inka.de> утверждает: "win2k: работает для случая с предопределенными ключами и IP-адресом в качестве идентификатора (не

думаю, что Windows поддерживает FQDN или строки USERFQDN). Сертификаты также должны работать, но я их не пробовал. ".

7.5.2. Check Point VPN-1 NG

Питер Бирингер (Peter Bieringer) сообщает:

```
Ниже приводятся некоторые результаты (тестировался только туннельный режим,
auth=SHA1):

DES:     ok
3DES:    ok
AES-128: ok
AES-192: not supported by CP VPN-1
AES-256: ok
CAST* :  not supported by used Linux kernel

Tested version: FP4 aka R54 aka w/AI
```

Дополнительная информация -- по адресу: http://www.fw-1.de/aerasec/ng/vpn-racoon/CP-VPN1-NG-Linux-racoon.html.

Глава 8. Маршрутизация групповых сообщений.

FIXME: Место редактора вакантно!

Multicast-HOWTO достаточно сильно устарел и по этой причине может быть местами неточен, а иногда и совершенно неверным.

Для того, чтобы иметь возможность маршрутизации групповых сообщений, необходимо определенным образом сконфигурировать ядро Linux. Это, в свою очередь, требует, чтобы вы определились -- какой из протоколов групповой маршрутизации будет использоваться. На сегодняшний день, существует четыре основных протокола: DVMRP (групповая версия протокола RIP unicast), MOSPF (то же самое, но для OSPF), PIM-SM (Protocol Independent Multicast-Sparse Mode) -- Протокол Групповой Маршрутизации, не зависящий от "обычного" протокола маршрутизации, который применяется для маршрутизации групповых сообщений в малочисленные группы (слово "sparse" можно перевести как "рассеянные", или "разреженные", или "малочисленные". *прим перев.*) и PIM-DM (то же самое, но для групп с компактным расположением).

Однако, в ядре Linux, эти опции отсутствуют, потому что собственно протокол обслуживается отдельными приложениями, выполняющими маршрутизацию, такими как *Zebra*, *mrouted* или *pimd*. Тем не менее включение дополнительных опций в ядре необходимо.

Независимо от протокола групповой рассылки необходимо включить опции `IP: multicasting` и `IP: multicast routing`. Для *DVMRP* и *MOSPF* этого будет вполне достаточно. Если вы предполагаете использовать протокол маршрутизации PIM, то вы дополнительно должны разрешить опции *PIM-SM version 1* или *PIM-SM version 2*, в зависимости от версии протокола *PIM*, используемого в вашей сети.

После того, как вы разберетесь с необходимыми опциями и пересоберете ядро, вы заметите, что во время загрузки, среди прочих, в списке присутствует протокол *IGMP*. Это Протокол Управления Группами (IGMP -- Internet Group Management Protocol). К моменту написания этих строк, Linux поддерживал только версии 1 и 2 протокола *IGMP*, хотя версия 3 уже вышла. Однако для нас это не имеет большого знаения, поскольку *IGMP v3* достаточно нов и его дополнительные возможности еще не скоро найдут широкое применение. В дальнейшем описании мы будем предполагать использование самых основных характеристик протокола *IGMPv2*, хотя будем встречаться и с *IGMPv1*.

Итак, мы разрешили групповую маршрутизацию в ядре. Теперь необходимо "сообщить" ему -- что собственно со всем этим делать. Это означает, что нужно добавить в таблицу маршрутизации виртуальную сеть для групповых рассылок.

```
ip route add 224.0.0.0/4 dev eth0
```

(Естесственно, в данном случае предполагается, что маршрутизация производится через устройство eth0! Если вы используете другое устройство -- измените команду соответствующим образом)

А теперь "включим" перенаправление (forwarding):

```
echo 1 > /proc/sys/net/ipv4/ip_forward
```

и попробуем -- что у нас получилось. Для этого **ping**-анем предопределенную группу, с адресом 224.0.0.1 (этоть адрес зарезервирован *IANA* и обозначает "все узлы в данной сети". *прим. перев.*). В результате все машины в локальной сети, на которых включена поддержка возможности обслуживания групповых рассылок, *должны* ответить. Вы наверняка заметите, что все ответившие "подписываются" своим собственным IP-адресом, а не адресом 224.0.0.1. Вот это сюрприз! :) Все члены группы устанвливают в качестве исходящего, свой адрес, а не адрес группы.

```
ping -c 2 224.0.0.1
```

С этого момента вы готовы производить групповую маршрутизацию. При этом предполагается, что у вас имеется две сети, между которыми и выполняется маршрутизация.

(Продолжение следует...)

Глава 9. Дисциплины обработки очередей для управления пропускной способностью

Когда я узнал об этом, я был в полнейшем восторге. Linux 2.2/2.4 содержит все необходимое для управления полосой пропускания на уровне специализированных систем управления пропускной способностью.

Linux предоставляет даже больше возможностей, чем FrameRelay и ATM.

Чтобы избежать путаницы, утилита **tc** использует следующие единицы измерения для задания пропускной способности:

```
mbps = 1024 kbps = 1024 * 1024 bps => byte/s
mbit = 1024 kbit => kilo bit/s.
mb = 1024 kb = 1024 * 1024 b => byte
mbit = 1024 kbit => kilo bit
```

Хранятся данные в bps и b.

Но при выводе, **tc** использует сделующее соглашение:

```
1Mbit = 1024 Kbit = 1024 * 1024 bps => byte/s
```

9.1. Понятие очереди и дисциплины обработки

Организация очереди (очередизация) определяет способ отсылки данных. Важно понимать, что мы можем управлять лишь скоростью передачи отправляемых данных.

В том виде, в каком сейчас существует Internet, мы не можем контролировать объем входящего трафика. Это что-то вроде почтового ящика (не электронного!). Нет никакого способа влиять на то, какой объем почты приходит к вам, разве что общаясь с каждым респондентом.

Однако, Internet, в большистве своем, основан на протоколе TCP/IP, а у него есть несколько свойств, которые могут нам помочь. TCP/IP не может узнать пропускной способности сети между двумя хостами, поэтому он начинает передавать данные все быстрее и быстрее (это называется "медленный старт"). Когда пакеты начинают теряться из-за перегрузки передающей среды, передача тормозится. На самом деле все немного сложнее и умнее, но об этом позже.

Продолжая нашу аналогию, это можно сравнить с выбрасыванием половины вашей почты в надежде на то, что люди перестанут вам писать. Разница лишь в том, что в случае с Internet этот прием срабатывает :-)

Если у вас есть маршрутизатор и вы хотите ограничить скорость загрузки во внутренней сети, вам нужно это делать на внутреннем интерфейсе маршрутизатора, с которого данные передаются вашим компьютерам.

Кроме того, вы должны быть уверенны, что контролируете узкое место соединения. Так, если у вас есть 100-мегабитная сетевая карта и маршрутизатор с соединением в 256 Кбит/сек, вы должны убедиться, что не посылаете данных больше, чем ваш маршрутизатор может передать. Иначе канал будет контролировать маршрутизатор и именно он будет ограничивать доступную пропускную способность. Нам нужно, так сказать, "владеть очередью" и быть самым медленным звеном. К счастью это легко реализуется.

9.2. Простые бесклассовые дисцплины обработки очереди.

Как уже говорилось, дисциплины обработки очереди определяют способ передачи данных.

Бесклассовые дисциплины, в общем, получают данные, переупорядочивают, вносят задержку или уничтожают их.

Они могут использовать для ограничения пропускной способности интерфейса целиком, без какого-либо разделения по классам. Крайне важно, чтобы вы поняли назначение этого типа очередей перед тем, как мы перейдем к классовым дисциплинам!

Наиболее распространенной дисциплиной является pfifo_fast -- она используется по-умолчанию.

Каждая из дисциплин имеет свои достоинства и недостатки. Не все из них досконально протестированы.

9.2.1. pfifo_fast

Эта дисциплина работает, как видно из названия, по принципу "первым пришел, первым ушел" (First In, First Out). Это означает, что ни один пакет не получает специальной обработки. Однако это не совсем так. Данная очередь имеет три, так называемых, "полосы". В каждой "полосе" пакеты обрабатываются по принципу FIFO. Но полоса 1 не будет обслуживаться до тех пор, пока есть пакеты в полосе 0. Аналогично, пока есть пакеты в полосе 1, не обрабатывается полоса 2.

Ядро учитывает значение поля пакета Type of Service, и направляет пакеты с установленным флагом 'минимальная задержка' в полосу 0.

Не путайте эту простую бесклассовую дисциплину с классовой дисциплиной PRIO! Хотя они ведут себя похожим образом, pfifo_fast является бесклассовой и вы не можете добавлять к ней другие дисциплины командой **tc**.

9.2.1.1. Параметры и использование

Вы не можете конфигурировать pfifo_fast, поскольку ее параметры жестко "зашиты". Вот ее конфигурация по умолчанию:

priomap

> Определяет отображение пакетных приоритетов, присвоенных ядром, в полосы.
> Отображение основывается на значении поля TOS, которое выглядит следующим образом:

```
   0     1     2     3     4     5     6     7
+-----+-----+-----+-----+-----+-----+-----+-----+
|                 |                       |     |
|   PRECEDENCE    |          TOS          | MBZ |
|                 |                       |     |
+-----+-----+-----+-----+-----+-----+-----+-----+
```

> Четыре бита TOS (поле TOS) определяются так:

```
Двоичн Десятичн  Значение
----------------------------------------------------------
1000   8         Минимизировать задержку (md)
0100   4         Максимальная пропускная способность (mt)
0010   2         Максимальная надежность (mr)
```

```
0001   1         Минимальная стоимость (mmc)
0000   0         Обычное обслуживание
```

Поскольку справа еще есть 1 бит, реальное значение поля TOS вдвое больше значения битов TOS. Команда tcpdump -v -v выводит значение всего поля TOS, а не только четырех бит. Оно приведено в первой колонке таблицы:

```
TOS     Биты  Значение                Приоритет Linux  Полоса
------------------------------------------------------------
0x0     0     Normal Service          0 Best Effort     1
0x2     1     Minimize Monetary Cost  1 Filler          2
0x4     2     Maximize Reliability    0 Best Effort     1
0x6     3     mmc+mr                  0 Best Effort     1
0x8     4     Maximize Throughput     2 Bulk            2
0xa     5     mmc+mt                  2 Bulk            2
0xc     6     mr+mt                   2 Bulk            2
0xe     7     mmc+mr+mt               2 Bulk            2
0x10    8     Minimize Delay          6 Interactive     0
0x12    9     mmc+md                  6 Interactive     0
0x14    10    mr+md                   6 Interactive     0
0x16    11    mmc+mr+md               6 Interactive     0
0x18    12    mt+md                   4 Int. Bulk       1
0x1a    13    mmc+mt+md               4 Int. Bulk       1
0x1c    14    mr+mt+md                4 Int. Bulk       1
0x1e    15    mmc+mr+mt+md            4 Int. Bulk       1
```

Куча цифр. Вторая колонка содержит значение четырех значимых битов поля TOS, а в третьей расшифровывается значение. Например, 15 означает минимальную стоимость (Minimal Monetary Cost), максимальную надежность (Maximum Reliability), максимальную полосу пропускания(Maximum Throughput) И минимальную задержку (Minimum Delay).

В четвертой колонке приведены соответствующие уровни приоритетов, выставляемые ядром Linux.

В последней колонке демонстрируется результат отображения в полосы по-умолчанию. В командной строке это выглядит так:

```
1, 2, 2, 2, 1, 2, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1
```

Это означает, что, например, приоритет 4 отображается в первую полосу. **priomap** позволяет задавать и более высокие приоритеты (> 7) , которые не соответствуют полю TOS, но они используются в других целях.

Ниже приводится таблица из документа RFC 1349 (за подробной информацией обратитесь к этому документу). Она показывает, каким образом приложения могут выставлять биты TOS:

```
TELNET                  1000           (minimize delay)
FTP
        Control         1000           (minimize delay)
        Data            0100           (maximize throughput)

TFTP                    1000           (minimize delay)

SMTP
        Command phase   1000           (minimize delay)
        DATA phase      0100           (maximize throughput)

Domain Name Service
        UDP Query       1000           (minimize delay)
```

```
        TCP Query       0000
        Zone Transfer   0100            (maximize throughput)

NNTP                    0001            (minimize monetary cost)

ICMP
        Errors          0000
        Requests        0000 (mostly)
        Responses       <same as request> (mostly)
```

txqueuelen

Длина этой очереди определяется конфигурацией интерфейса, просмотреть которую можно командами **ifconfig** и **ip**. Для задания очереди длиной 10, выполните: **ifconfig eth0 txqueuelen 10**.

Вы не можете управлять этим параметром при помощи утилиты **tc**.

9.2.2. Token Bucket Filter

Token Bucket Filter (TBF) простая дисциплина очереди, которая передает поступающие пакеты со скоростью не превышающей административно заданный порог, но с возможностью превышающих его коротких всплесков.

TBF очень точная дисциплина, при этом она не создает серьезных нагрузок на сеть и процессор. Если вам нужно просто ограничить скорость на интерфейсе, то это первый кандидат на использование.

Реализована TBF в виде буфера, постоянно заполняющегося токенами с заданой скоростью. Наиболее важным параметром буфера является его размер, определяющий количество хранимых токенов.

Каждый прибывающий токен сопоставляется с одним пакетом данных из очереди после чего удаляется. Связав этот алгоритм с двумя потоками -- токенов и данных, получим три возможных ситуации:

- Данные прибывают со скоростью равной скорости входящих токенов. В этом случае каждый пакет имеет соответствующий токен и проходит очередь без задержки.

- Данные прибывают со скоростью меньшей скорости поступления токенов. В этом случае лишь часть существующих токенов будет уничтожаться, потому они станут накапливаться до размера буфера. Далее, накопленные токены могут использоваться при всплесках, для передачи данных со скоростью превышающей скорость пребывающих токенов.

- Данные прибывают быстрее, чем токены. Это означает, что в буфере скоро не останеться токенов, что заставит дисциплину приостановить передачу данных. Эта ситуация называется "превышением". Если пакеты продолжают поступать, они начинают уничтожаться.

Последняя ситуация очень важна, поскольку позволяет административно ограничивать доступную полосу пропускания.

Накопленные токены позволяют пропускать короткие всплески, но при продолжительном

превышении пакеты будут задерживаться, а в крайнем случае -- уничтожаться.

Учтите, что в реальной реализации дисциплины, токены соответствуют байтам, а не пакетам.

9.2.2.1. Параметры и использование

Не смотря на то, что вам вероятно ничего не придется менять, дисциплина TBF имеет определенные параметры. В первую очередь это:

limit или latency

> *Limit* -- это количество байт, которые могут быть помещены в очередь ожидания токенов. Эту же величину можно задать параметром *latency*, который определяет максимальный "возраст" пакета в очереди TBF. В последнем случае, во внимание принимается размер буфера, скорость и, если задана, пиковая скорость (*peakrate*).

burst/buffer/maxburst

> Размер буфера в байтах. Максимальное количество байт, для которых токены могут быть доступны мгновенно. В целом, чем больше граничная скорость, тем больше должен быть размер буфера. Например, для ограничения на скорости 10 мбит/с на платформе Intel, вам нужен буфер размером как минимум 10 Кбайт, чтоб достичь заявленной скорости!
>
> Если буфер слишком мал, пакеты могут уничтожаться. Это связано с тем, что каждый тик таймера будет генерироваться больше токенов, чем может поместиться в вашем буфере.

mpu

> Пакет нулевого размера все равно использует полосу пропускания. В сетях ethernet, любой пакет имеет размер не менее 64 байт. *MPU* задает минимальное количество токенов для пакета.

rate

> Ограничение скорости.

Если буфер заполнен токенами, то поступающие пакеты будут проходить очередь без всяких задержек. Если вас это не устраивает, используйте следующие параметры:

peakrate

> Если на момент поступление пакета есть свободные токены, пакет пройдет очередь без каких-либо задержек. Так сказать, со скоростью света. Возможно, это не совсем то, чего вы хотите, особенно если вы используете большой буфер.
>
> Параметр *peakrate* задает скорость, с которой элемент может проходить очередь. Согласно теории, это достигается организацией достаточной задержки между проходящими пакетами.

mtu/minburst

> Очевидно, что максимальное значение *peakrate*, равное 1 Мбит/сек, накладывало бы сильное ограничение на область применения этой дисциплины. Однако, задание больших

значений *peakrate* возможно. Достигается это за счет прохождения за один интервал времени более одного пакета данных.

По умолчанию, значение *mtu* равно одному пакету, т.е. за раз проходит только один пакет.

Для расчета максимально возможного значения *peakrate*, умножьте *mtu* на 100 (или, точнее, на HZ, которое равно 100 для платформы Intel и 1024 для Alpha)

9.2.2.2. Пример конфигурации

Простая, но очень полезная конфигурация:

```
# tc qdisc add dev ppp0 root tbf rate 220kbit latency 50ms burst 1540
```

Чем же она так замечательна? Если у вас есть сетевое устройство с большой очередью, такое как DSL или кабельный модем, а вы обмениваетесь с ним данными через быстрое соединение, например ethernet, вы обнаружите, что закачки полностью уничтожают возможность интерактивной работы.

Это связано с тем, что закачки заполняют очередь модема, которая часто очень большая. Большой размер очереди позволяет достичь хороших результатов при передаче больших объемов данных, но ведь это не совсем то, что вам нужно. Вы хотите, чтобы оставалась интерактивность и вы могли бы делать еще что-то полезное во время закачки.

Приведенная команда ограничивает скорость отправки данных так, чтобы очередь в модеме не образовывалась. Таким образом, очередь будет находится в Linux, где мы можем контролировать ее размер.

Замените 220 Кбит на реальную скорость, минус несколько процентов. Если у вас действительно быстрый модем, можете немного увеличить параметр *burst*.

9.2.3. Stochastic Fairness Queueing.

Stochastic Fairness Queueing (SFQ) -- простая реализация семейства алгоритмов справедливой очередизации. Она не так точна, как другие дисциплины, но требует меньше рассчетов, и при этом поровну распределяет доступную полосу пропускания между сеансами.

Ключевым понятием в SFQ является диалог (или поток), который приблизительно соответствует сеансу TCP или потоку UDP. Трафик делится на достаточное количество очередей типа FIFO, по одной на каждый диалог. После этого, все очереди обрабатываются в циклическом порядке, тем самым обеспечивая каждому сеансу равные шансы на передачу данных.

Благодаря этому достигается очень ровное поведение, которое не позволяет какому-либо диалогу подавлять остальные. SFQ называется "стохастической", т.к. на самом деле для каждого сеанса очередь не формируется, а трафик делится на ограниченое количество очередей на основе хеш-алгоритма.

Из-за использования хеша, несколько сессий могут попасть в одну и ту же очередь, что уменьшает шансы на передачу каждого сеанса. Для того, чтобы эта проблема не ощущалась, SFQ часто меняет алгоритм хеширования, поэтому, если сессии и попадут в одну очередь, длиться это будет

лишь несколько секунд.

Стоит заметить, что SFQ эффективен только если исходящий интерфейс полностью загружен! В противном случае очередь будет отсутствовать и, следовательно, никакого положительного эффекта наблюдаться не будет. Позже мы рассмотрим варианты комбинирования SFQ с другими дисциплинами для достижения наилучшего результата.

В частности, применение SFQ на ethernet интерфейсе к которому подключен кабельный модем или DSL маршрутизатор совершенно бессмыслено без органичения полосы пропускания!

9.2.3.1. Параметры и использование

SFQ в значительной степени самоконфигурирующаяся:

perturb

> Интервал изменения алгоритма хеширования. Если не задан -- алгоритм меняться не будет, что не рекомендуется. Хорошим значением является 10 секунд.

quantum

> Количество байт выводимых из очереди за один раз. По-умолчанию равно 1 пакету максимально возможного размера (MTU). Не устанавливайте этот параметр меньшим этого значения!

limit

> Общее количество пакетов, которые могут быть помещены в очередь SFQ (последующие пакеты будут уничтожаться).

9.2.3.2. Пример конфигурации

Если у вас есть устройство, скорость соединения которого равна доступной полосе пропускания, например модем, следующий пример обеспечит разделение его возможностей между всеми пользователями:

```
# tc qdisc add dev ppp0 root sfq perturb 10
# tc -s -d qdisc ls
qdisc sfq 800c: dev ppp0 quantum 1514b limit 128p flows 128/1024 perturb 10sec
 Sent 4812 bytes 62 pkts (dropped 0, overlimits 0)
```

Число 800c: это автоматически присваеваемый дескриптор, параметр *`limit`* говорит, что в очереди может находиться до 128 пакетов. Доступно 1024 хеш-буфера, из которых 128 может быть активно (максимальное число пакетов в очереди). Каждые 10 секунд хеши будут перенастраиваться.

9.3. Какие типы дисциплин нужно использовать.

Обобщая вышесказанное: мы рассмотрели простые дисциплины очередей, которые управляют трафиком переупорядочиванием, задержкой или уничтожением пакетов.

Следующие советы могут помочь при выборе типа применяемой дисциплины. Здесь также упоминаются некоторые дисциплины, описываемые в главе 14.

- Чтобы просто ограничить скорость интерфейса, используйте Token Bucket Filter. Маштабируется до больших скоростей, при соответствующем увеличении буфера.

- Если ваш канал полностью загружен и при этом вы не желаете допустить доминирование какого-либо сеанса, используйте SFQ.

- Если вы хотите управлять скоротью магистральных каналов и хорошо понимаете как это делается, используйте Random Early Drop (описывается в главе 14).

- Для управления скоростью входящего трафика, который не пересылается, используйте ограничитель (Ingress Policer).

- Если вы пересылаете трафик, используйте дисциплину TBF на интерфейсе с которого отправляются данные. Этот подход работает, если вам не нужно пересылать трафик через разные интерфейсы. В этом случае единственным общим фактором является входящий интерфейс, потому используйте ограничитель.

- Если вам не нужно ограничивать полосу пропускания, но вы хотите видеть, справляется ли ваш интерфейс с нагрузкой, используйте очередь pfifo (не pfifo_fast). У нее нет внутренних полос, но она ведет учет использования очереди.

- Наконец, вы можете использовать "человеческий" фактор. Использование технологических решений не всегда приводит к желаемым результатам. Пользователи враждебно относятся к техническим ограничениям. Доброе слово тоже может помочь в правильном распределении пропускной способности!

9.4. Терминология

Прежде чем приступить к обсуждению более сложных тем, считаю своим долгом дать описание некоторых основных понятий, поскольку из-за сложности и относительной новизны, разные разработчики обозначают одни и те же понятия различными терминами.

Следующие описания являются вольной трактовкой документа `draft-ietf-diffserv-model-06.txt` -- *An Informal Management Model for Diffserv Routers* (Неформальная Модель Управления для Маршрутизаторов, работающих под управлением Diffserv). Этот документ находится по адресу: http://www.ietf.org/internet-drafts/draft-ietf-diffserv-model-06.txt В нем вы найдете строгие определения используемых терминов.

Queueing Discipline (qdisc)

Алгоритм управления очередью устройства, как входящей (ingress), так и исходящей (egress).

root qdisc

Корневая дисциплина организации очереди (qdisc), т.е. дисциплина применяемая к устройству.

Classless qdisc

Бесклассовая дисциплина организации очереди -- дисциплина, которая воздействует на устройство целиком, без возможности классификации трафика.

Classful qdisc

Полноклассовая дисциплина организации очереди (qdisc). Может содержать множество классов и других дисциплин, которые в свою очередь могут являться полноклассовыми. Согласно этому определению, дисциплина pfifo_fast -- полноклассовая, поскольку она содержит три полосы пропускания, которые фактически являются классами. Однако, с точки зрения пользователя (администратора сети), она является бесклассовой, так как принцип действия этих классов не может быть изменен с помощью утилиты **tc**.

Classes

Классы. Полноклассовые дисциплины организации очереди могут состоять из множества классов, каждый из которых является внутренним, по отношению к данной дисциплине. В свою очередь, классы так же могут содержать другие классы. Таким образом, класс может иметь, в качестве "родительского", другой класс или дисциплину. Краевой класс -- это класс, который не имеет вложенных классов. Такой класс имеет единственную, присоединенную к нему, дисциплину организации очереди. Эта дисциплина отвечает за передачу данных из этого класса. При создании класса, к нему по-умолчанию присоединяется дисциплина fifo. При добавлении вложенноного класса, дисциплина удаляется. Для краевых классов, созданная по-умолчанию дисциплина fifo может быть заменена на более подходящую вашим требованиям, вплоть до того, что можно назначить полноклассовую дисциплину, с набором своих внутренних классов!

Classifier

Классификатор. Каждая полноклассовая дисциплина должна "разбить" трафик по классам. Выполняется это с помощью классификаторов.

Filter

Фильтр. Классификация может выполняться и с помощью фильтров. Фильтры представляют собой ряд условий, благодаря которым выясняется -- подпадает ли тот или иной пакет под действие фильтра.

Scheduling

Планирование. Дисциплины, с помощью классификаторов, могут изменять порядок прохождения пакетов. Процесс переупорядочивания называется Планированием. К слову, дисциплина pfifo_fast, упоминавшаяся выше, производит планирование.

Shaping

Шейпинг (формирование). Под этим термином подразумевается процесс задержки пакетов, с целью ограничения пропускной способности канала. Шейпинг выполняется на исходящих очередях. Сброс отдельных пакетов, с целью снижения пропускной способности, так же иногда называют Шейпингом.

Policing

> Ограничение. Под этим термином подразумевается процесс задержки или сброса (умышленной потери) пакетов, с целью ограничения пропускной способности. В Linux под термином Policing подразумевается только сброс пакетов, если ограничение накладывается не на входящую очередь.

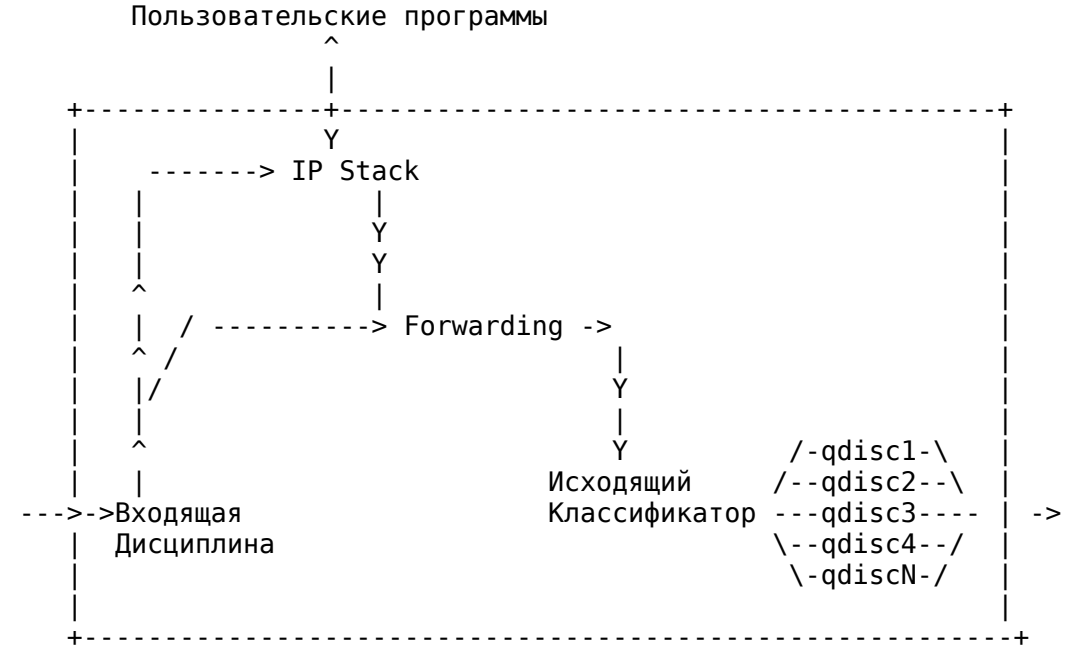
Work-Conserving

> Под этим термином подразумеваются такие дисциплины, которые всегда передают пакет без задержки дальше, если он поступил на обработку. Другими словами, эти дисциплины никогда не выполняют задержку пакетов, если устройство готово отправить его (в случае исходящей дисциплины).

non-Work-Conserving

> Некоторые виды дисциплин, например Token Bucket Filter, могут выполнять задержку пакетов на некоторое время, с целью ограничения пропускной способности. Это означает, что пакеты могут быть задержаны, даже не смотря на то, что устройство готово к передаче.

Теперь, когда мы определились с терминологией, рассмотрим все вышесказанное на следующей диаграмме:



Большой объемлющий блок, на диаграмме, представляет собой ядро. Стрелка слева показывает входящий трафик. Он подается во входящую дисциплину, которая может содержать ряд фильтров, посредством которых отдельные пакеты могут быть отброшены (потеряны, отфильтрованы). Это называется Ограничением (Policing).

Все это происходит на самой ранней стадии, прежде чем пакет будет передан для дальнейшей обработки. Таким образом достигается уменьшение нагрузки на центральный процессор.

Если пакет благополучно миновал эту стадию, то далее он может быть передан либо локальным приложениям (в этом случае он попадает в стек IP для дальнейшей обработки), либо в сеть, через исходящий классификатор, на другой узел сети. Пользовательские приложения так же могут отправлять данные в сеть, которые аналогичным образом движутся через исходящий классификатор. Исходящий классификатор "разбивает" трафик по очередям, каждая из которых имеет свою дисциплину организации. В случае конфигурации по-умолчанию имеется

единственная исходящая очередь -- pfifo_fast. Этот процесс называется "постановкой в очередь".

Попав в соответствующую очередь, пакет ожидает, пока ядро передаст его сетевому интерфейсу. Этот процесс называется "выборка из очереди".

Эта диаграмма годится и для случая с одним сетевым интерфейсом -- не следует рассматривать стрелки на ней слишком буквально.

9.5. Классовые дисциплины обработки очередей.

Классовые дисциплины широко используются в случаях, когда тот или иной вид трафика необходимо обрабатывать по разному. Примером классовой дисциплины может служить **CBQ** -- Class Based Queueing (дисциплина обработки очередей на основе классов). Она настолько широко известна, что многие идентифицируют понятие "Дисциплина Обработки Очередей" с названием **CBQ**, однако это далеко не так.

CBQ -- один из старейших алгоритмов и кроме того -- один из самых сложных. К сожалению он может далеко не все. Это может оказаться неожиданностью для тех, кто свято верит в то, что если какая-либо достаточно сложная технология распространяется без документации, то это лучшая технология из имеющихся вариантов.

Чуть ниже мы поближе рассмотрим CBQ и его альтернативы.

9.5.1. Порядок движения пакетов внутри полноклассовых дисциплин и классов.

Когда трафик передается на обработку классовой дисциплине, он должен быть отнесен к одному из классов (классифицирован). Определение принадлежности пакета к тому или иному классу выполняется фильтрами. Очень важно понимать, что именно фильтры вызываются из дисциплины, а не наоборот!

Фильтры, присоединенные к дисциплине, возвращают результат классификации (грубо говоря -- класс пакета), после чего пакет передается в очередь, соответствующую заданному классу. Каждый из классов, в свою очередь, может состоять из подклассов и иметь свой набор фильтров, для выполнения более точной классификации свой доли трафика. В противном случае пакет обслуживается дисциплиной очереди класса.

Кроме того, в большинстве случаев классовые дисциплины выполняют шейпинг (формирование) трафика, с целью переупорядочивания пакетов (например, с помощью SFQ) и управления скоростью их передачи. Это определенно необходимо в случае перенаправления трафика с высокоскоростного интерфейса (например, ethernet) на медленный (например, модем).

9.5.2. Элементы дисциплины: корень, дескриптор, родительские элементы и элементы одного уровня.

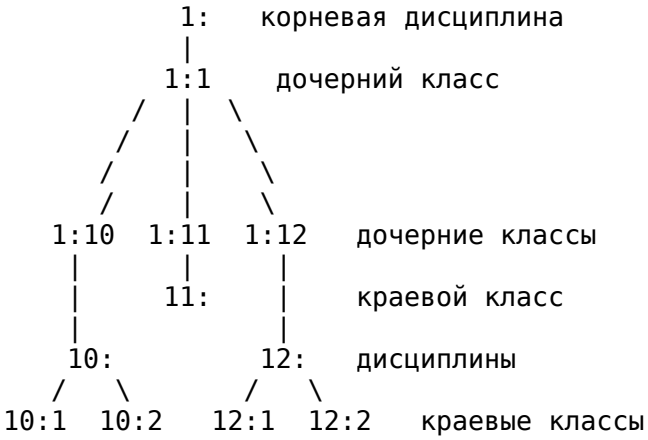
Каждый из интерфейсов имеет одну исходящую корневую дисциплину. По-умолчанию это, упоминавшаяся ранее дисциплина -- pfifo_fast. Каждой дисциплине и каждому классу назначается уникальный дескриптор, который который может использоваться последующими инструкциями для ссылки на эти дисциплины и классы. Помимо исходящей дисциплины, интерфейс так же может иметь и входящую дисциплину, которая производит управление входящим трафиком.

Дескрипторы дисциплин состоят из двух частей -- старшего и младшего номеров, в виде: <старший>:<младший>. Корневой дисциплине общепринято присваивать дескриптор '1:', что эквивалентно записи '1:0'. Младший номер в дескрипторе любой дисциплины всегда '0'.

Старшие номера дескрипторов классов всегда дублируют старший номер дескриптора своего "родителя". Старший номер должен быть уникальным в пределах блоков настроек для входящего и исходящего трафиков. Младший номер должен быть уникальным в пределах дисциплины и ее классов.

9.5.2.1. Как выполняется классификация с помощью фильтров.

Типичная иерархия может выглядеть следующим образом:



Не дайте ввести себя в заблуждение! Вы не должны полагать, что ядро находится на вершине диаграммы, а сеть под ней! Пакеты ставятся в очередь и извлекаются из очереди корневой дисциплиной, она единственная, с которой работает ядро.

Порядок классификации некоторого пакета может быть представлен в виде цепочки, например:

1: -> 1:1 -> 1:12 -> 12: -> 12:2

Таким образом, пакет помещается в очередь, которая управляется дисциплиной, присоединенной к классу 12:2. В данном примере, к каждому узлу дерева присоединен фильтр, который принимает решение -- по какой ветви продолжить классификацию пакета. Но возможен и другой вариант:

1: -> 12:2

В этом случае, фильтр, присоединенный к корню, сразу же классифицировал пакет, как принадлежащий классу 12:2.

9.5.2.2. Как выполняется извлечение пакетов из очереди.

Когда ядро решает, что пора извлечь очередной пакет из очереди и передать его сетевому интерфейсу, оно посылает запрос на извлечение корневой дисциплине. Далее этот запрос передается по цепочке классу 1:1, а от него всем элементам одного уровня -- 10:, 11:, и 12:. В данном случае запрос полностью обходит все дерево, поскольку только класс 12:2 имеет пакет.

Проще говоря, вложенные классы "общаются" ТОЛЬКО со своими "родительскими" дисциплинами и никогда не работают напрямую с интерфейсом. Только корневая дисциплина получает запрос на извлечение пакета из очереди напрямую от ядра!

В результате, классы никода не получат запрос на извлечение раньше, чем это будет сделано их "родителями". А это как раз и есть то, что нам нужно: мы можем определить внутренний класс, который выполняет только планирование и дисциплину, которая отвечает за шейпинг.

9.5.3. Дисциплина PRIO.

Дисциплина PRIO фактически никак не ограничивает трафик, она лишь выполняет его классификацию на основе присоединенных к ней фильтров. Вы можете рассматривать дисциплину PRIO как более мощную версию pfifo_fast, в которой каждая из полос является отдельным классом, а не простой очередью FIFO.

Постановка пакета в очередь выполняется дисциплиной PRIO на основе фильтров, заданных вами. По-умолчанию создаются три класса. Эти классы по-умолчанию содержат обычные дисциплины FIFO, но они могут быть заменены дисциплинами любого типа, какие вам только доступны.

Когда необходимо извлечь пакет из очереди, то первым проверяется класс :1. Каждый последующий класс проверяется только в том случае, если в предыдущем нет ни одного пакета.

Эта дисциплина может с успехом применяться в тех случаях, когда необходимо "раскидать" трафик по приоритетам, основываясь не только на флагах TOS. Вы можете так же добавить другие дисциплины к предопределенным классам, что повысит возможности управления трафиком, по сравнению с pfifo_fast.

Поскольку данная дисциплина не имеет возможности шейпинга трафика, считаю своим долгом предупредить вас: используйте эту дисциплину только в том случае, если она полностью соответствует вашим требованиям, либо присоединяйте ее к классовым дисциплинам, которые могут выполнять шейпинг. Последнее замечание относится к владельцам кабельных модемов или DSL устройств.

Формально, дисциплина PRIO относится к разряду планировщиков типа Work-Conserving.

9.5.3.1. Параметры и порядок использования дисциплины PRIO.

Применительно к данной дисциплине, утилита **tc** допускает следующие параметры:

bands

Число создаваемых полос. Каждая полоса фактически является классом. Если вы изменяете это число, то вы должны так же изменить и следующий параметр.

priomap

Если ваша конфигурация не предусматривает наличие фильтров, выполняющих классификацию трафика, то дисциплина PRIO присваивает приоритеты по-умолчанию.

Все это работает точно так же, как и в случае с pfifo_fast.

Каждая полоса является классом и имеет свой дескриптор, начиная с <старший_номер>:1 и заканчивая <старший_номер>:3, по-умолчанию. Таким образом, если дисциплине PRIO присвоен дескриптор 12: , то класс-полоса с наивысшим приоритетом получит дескриптор 12:1.

Повторюсь еще раз, полоса 0 получит младший номер дескриптора -- 1! Полоса 1 -- 2 и так далее.

9.5.3.2. Пример конфигурации.

В качестве примера создадим такое дерево:

```
             1:   корневая дисциплина
           / | \
          /  |  \
         /   |   \
       1:1  1:2  1:3    классы
        |    |    |
       10:  20:  30:    дисциплины
       sfq  tbf  sfq
полоса  0    1    2
```

Объемный трафик будет обслуживаться дисциплиной 30: , интерактивный -- 20: или 10:.

Конфигурирование:

```
# tc qdisc add dev eth0 root handle 1: prio
## Эта команда создаст классы 1:1, 1:2, 1:3

# tc qdisc add dev eth0 parent 1:1 handle 10: sfq
# tc qdisc add dev eth0 parent 1:2 handle 20: tbf rate 20kbit buffer 1600 limit 3000
# tc qdisc add dev eth0 parent 1:3 handle 30: sfq
```

Теперь посмотрим -- что у нас получилось:

```
# tc -s qdisc ls dev eth0
qdisc sfq 30: quantum 1514b
 Sent 0 bytes 0 pkts (dropped 0, overlimits 0)

 qdisc tbf 20: rate 20Kbit burst 1599b lat 667.6ms
 Sent 0 bytes 0 pkts (dropped 0, overlimits 0)

 qdisc sfq 10: quantum 1514b
 Sent 132 bytes 2 pkts (dropped 0, overlimits 0)

 qdisc prio 1: bands 3 priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
 Sent 174 bytes 3 pkts (dropped 0, overlimits 0)
```

Как видите, через полосу 0 уже "проскочил" какой-то трафик, пока отрабатывали наши команды!

Теперь выполним передачу достаточно большого объема данных неким инструментом, который корректным образом устанавливает флаги TOS и проверим еще раз:

```
# scp tc ahu@10.0.0.11:./
ahu@10.0.0.11's password:
tc                    100% |*****************************|   353 KB    00:00
# tc -s qdisc ls dev eth0
qdisc sfq 30: quantum 1514b
 Sent 384228 bytes 274 pkts (dropped 0, overlimits 0)

 qdisc tbf 20: rate 20Kbit burst 1599b lat 667.6ms
 Sent 2640 bytes 20 pkts (dropped 0, overlimits 0)

 qdisc sfq 10: quantum 1514b
 Sent 2230 bytes 31 pkts (dropped 0, overlimits 0)

 qdisc prio 1: bands 3 priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
 Sent 389140 bytes 326 pkts (dropped 0, overlimits 0)
```

На этот раз видно, что весь трафик был отправлен через дисциплину 30:, которая в нашем случае имеет наименьший приоритет. Чтобы убедиться в том, что интерактивный трафик поступает в высокоприоритетные полосы, выполним следующую команду:

```
# tc -s qdisc ls dev eth0
qdisc sfq 30: quantum 1514b
 Sent 384228 bytes 274 pkts (dropped 0, overlimits 0)

 qdisc tbf 20: rate 20Kbit burst 1599b lat 667.6ms
 Sent 2640 bytes 20 pkts (dropped 0, overlimits 0)

 qdisc sfq 10: quantum 1514b
 Sent 14926 bytes 193 pkts (dropped 0, overlimits 0)

 qdisc prio 1: bands 3 priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
 Sent 401836 bytes 488 pkts (dropped 0, overlimits 0)
```

Как видите, все работает правильно, весь трафик был отправлен в 10: -- через самую высокоприоритетную дисциплину.

9.5.4. Дисциплина CBQ.

Как уже упоминалось ранее, CBQ -- одна из самых сложных дисциплин. Пожалуй я не погрешу против истины, если заявлю, что это самая объемная, самая непонятная и самая запутанная дисциплина организации очередей. Это не потому, что авторы алгоритма некомпетентны, а потому, что идеология этого алгоритма абсолютно не совпадает с идеологией Linux.

Кроме того, что эта дисциплина является классовой, она так же может выполнять и шейпинг трафика, правда именно эта ее сторона является самым слабым местом. Если вы попробуете ограничить 10 мегабитный канал величиной в 1 мегабит, то окажется, что соединение будет просто простаивать 90% всего времени. Вместо определения объема трафика, CBQ измеряет время в микросекундах между запросами и на основе полученного времени рассчитывается средняя загруженность канала.

Такой алгоритм работы не всегда дает нужные результаты. Например, что если сетевой интерфейс

не может обеспечить полную загрузку канала на всю его возможную ширину, из-за некачественного драйвера? Как тогда правильно определить время простоя?

Проблема становится еще острее, если вам приходится иметь дело с такими вещами, как PPP через Ethernet или PPTP через TCP/IP. Эффективная пропускная способность в данном случае может быть определена как пропускная способность канала в пространство пользователя, а это величина очень не маленькая.

Те, кто близко сталкивался с CBQ отмечают, что эта дисциплина не всегда точна, и иногда допускает грубые просчеты при измерении времени.

Однако, в других случаях она показывает неплохие результаты. Прочитав этот документ, вы сможете сконфигурировать эту дисциплину и получить неплохие результаты в случае отказа от шейпинга.

9.5.4.1. Шейпинг в CBQ.

Как уже говорилось выше, ограничение пропускной способности в CBQ выполняется за счет определения промежутка времени между прохождением соседних пакетов среднего размера.

В процессе работы измеряется эффективное время простоя, как экспоненциальное взвешенное среднее по скользящему окну. Кстати, UNIX рассчитывает величину loadaverage (средняя величина нагрузки) аналогичным способом.

Расчетное время простоя вычитается из взвешенного среднего, в результате получается величина *avgidle*. Полностью загруженный канал имеет величину *avgidle* равную нулю -- промежуток времени между пакетами точно совпадает с расчетным. В случае превышения заданного ограничения, величина *avgidle* становится отрицательной. Если превышение достигает некоторого порога, CBQ приостанавливает передачу.

С другой стороны, после нескольких часов простоя, величина *avgidle* может получиться слишком большой и это приведет к тому, что канал "распахнется" на всю ширину. Чтобы этого не происходило, величина *avgidle* ограничивается числом *maxidle*.

В случае перегрузки, теоретически, алгоритм CBQ должен приостанавить передачу на вычисленный временной интервал, потом передать следующую порцию данных и снова остановиться. Но в реализации алгоритма есть свои нюансы, смотрите ниже описание параметра *minburst*.

Рассмотрим параметры дисциплины CBQ, позволяющие настроить ограничение полосы пропускания:

avpkt

> Усредненный размер одного пакета. Совместно с величиной *maxburst* (которая измеряется в пакетах) используется для вычисления значения *maxidle*.

bandwidth

> Физическая пропускная способность устройства. Необходима для вычисления времени простоя.

cell

Время, необходимое на передачу пакета через интерфейс может увеличиваться с определенным шагом. Например, передача пакетов с размерами 800 и 806 байт займет одно и тоже время, а пакетов с размерами 810 байт -- немного больше. Данный параметр задает размер шага, с которым увеличивается расчетное время передачи. Чаще всего устанавливается значение '8'. Значение должно быть степенью числа 2.

maxburst

Параметр используется для вычисления значения *maxidle*. Он задает количество усредненных пакетов, которые будут обработаны до того, как значение *avgidle* станет равным нулю. Параметр *maxidle* нельзя задать напрямую, только косвенно, с помощью этого параметра.

minburst

Как я уже говорил, в случае перегрузки, CBQ должна прекратить передачу данных. Идеальным решением является ожидание в течение расчетного промежутка времени, после этого передать один пакет и снова приостановить передачу. Однако, в Unix существует определенные сложности с измерением интервалов времен, короче 10 мс, потому лучше увеличить промежуток ожидания, после которого можно будет выполнить передачу уже не одного, а несколько пакетов, количество которых определяется параметром *minburst*. Соответственно, промежуток ожидания между передачей пакетов увеличивается пропорционально значению данного параметра.

Промежуток времени между передачей пакетов называется *offtime*. Большие значения параметра *minburst* позволяют более точно ограничить полосу пропускания на длительных временных интервалах, но на коротких интервалах трафик будет вести себя скачкообразно.

minidle

Отрицательное значение *avgidle* указывает, что канал перегружен, и нужно прекратить передачу на время, пока *avgidle* не увеличится достаточно для передачи следующего пакета. Однако, это может вызвать длительные периоды ожидания после интенсивных всплесков трафика. В таких случаях *avgidle* присваивается значение параметра *minidle*.

Величина *minidle* измеряется в отрицательных микросекундах, следовательно значение "10" говорит о том, что параметр *avgidle* не может опуститься ниже -10 микросекунд.

mpu

Минимальный размер пакета -- необходим, поскольку даже "пустой" пакет дополняется до 64 байт для передачи по сети ethernet, и передача такого пакета занимает определенное время. Алгоритм CBQ должен знать минимальный размер, чтобы правильно рассчитать время простоя.

rate

Желаемая величина пропускной способности. Для данной дисциплины -- это "педаль газа"!

Внутрення реализация CBQ имеет ряд дополнительных настроек. Например, классы, в очередях которых нет данных, не опрашиваются. Для классов, допустивших перегрузку, понижается значение эффективного приоритета. В принципе все достаточно грамотно, хотя и очень сложно.

9.5.4.2. Характеристики классов в CBQ.

Кроме ограничения полосы пропускания, CBQ имеет возможность классифицировать трафик, подобно дисциплине PRIO, и назначать классам приоритеты.

Каждый раз, когда нужно передать пакет, запускается процесс взвешенной циклической выборки (WRR) из очередей, начиная с высокоприоритетных классов. Классы группируются по приоритетам, затем, после выборки определенного количества данных из одного класса, выполняется попытка получить данные из другого класса с тем же приоритетом.

Следующие параметры позволяют управлять процессом выборки данных:

allot

> Когда производится выбор пакета для передачи, CBQ начинает опрашивать свои подклассы в соответствии с их приоритетами. Когда классу предоставляется возможность передачи, выбирается определенный объем данных. Базовая единица этого объема определяется параметром *allot*.

prio

> Приоритет. Дисциплина CBQ может присваивать классам приоритеты. Чем меньше значение -- тем выше приоритет. Пока не будет обработан трафик с высшим приоритетом, трафик с меньшим приоритетом не обрабатывается.

weight

> Вес. Каждому из имеющихся классов предоставляется возможность передать данные. Если у вас есть классы, которые значительно отличаются полосой пропускания, имеет смысл разрешить классам с большой пропускной способностью посылать за раз больше данных, чем классам с небольшой пропускной способностью
>
> Дисциплина CBQ вычисляет сумму весов всех классов и затем нормирует их полученной величиной, потэому, в качестве весов, можно использовать любые значения: важно отношение этих значений. Обычно используется простое правило: вес = полоса пропускания / 10. Для определения количества данных, посылаемых классом за раз, нормированное значение умножается на величину *allot*.

Обратите внимание: все внутренние классы дисциплины должны иметь одинаковый старший номер дескриптора!

9.5.4.3. Параметры CBQ, управляющие характером заимствования.

Кроме простого ограничения полосы пропускания для определенных типов трафика, можно давать возможность классам занимать часть полосы пропускания у других классов.

Isolated/sharing

> У класса, созданного с параметром *isolated*, нельзя занимать полосу пропускания. То есть другие классы, настроенные на заем доступной полосы пропускания никогда не смогут занять полосу этого класса, даже если она будет свободной.
>
> Наоборот, класс созданный с параметром *sharing* будет предоставлять неиспользуемую часть

своей пропускной способности другим классам.

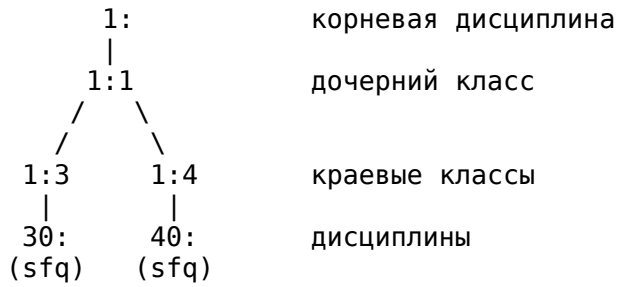
bounded/borrow

Эти два параметра определяют, может ли класс занимать пропускающую способность других классов. Параметр *bounded* запрещает, а *borrow* разрешает занимать неиспользуемую часть полосы пропускания классов, сконфигурированных с параметром *sharing*.

Примером, когда используются классы с параметрами *bounded* и *isolated*, может послужить ситуация использования одного канала двумя организациями. В этом случае они действительно будут ограничены заданной полосой пропускания.

Внутри каждого такого класса могут находиться подклассы с опциями *sharing* и *borrow*. В этой ситуации заем будет выполняться в пределах родительского класса.

9.5.4.4. Пример конфигурирования.



Рассмотрим реализацию следующего сценария. Необходимо ограничить полосу пропускания веб-трафика до 5 мегабит, а SMTP -- до 3 мегабит. Суммарная полоса пропускания не должна превышать 6 мегабит. На сервере стоит 100-мегабитная сетевая карта, классы могут занимать пропускную способность друг у друга.

```
# tc qdisc add dev eth0 root handle 1:0 cbq bandwidth 100Mbit         \
  avpkt 1000 cell 8
# tc class add dev eth0 parent 1:0 classid 1:1 cbq bandwidth 100Mbit  \
  rate 6Mbit weight 0.6Mbit prio 8 allot 1514 cell 8 maxburst 20      \
  avpkt 1000 bounded
```

В этой части устанавливается корневая дисциплина и класс 1:1, пропускная способность которого ограничена величиной в 6 мегабит.

Как видите, CBQ требует много больше настроек по сравнению с HTB.

```
# tc class add dev eth0 parent 1:1 classid 1:3 cbq bandwidth 100Mbit  \
  rate 5Mbit weight 0.5Mbit prio 5 allot 1514 cell 8 maxburst 20      \
  avpkt 1000
# tc class add dev eth0 parent 1:1 classid 1:4 cbq bandwidth 100Mbit  \
  rate 3Mbit weight 0.3Mbit prio 5 allot 1514 cell 8 maxburst 20      \
  avpkt 1000
```

Здесь создаются два класса, управляющие веб и почтовым трафиками. Обратите внимание на то, как указаны веса классов. Пропускная способность классов не ограничивается, но они подчинены классу 1:1, который имеет ограничение по полосе пропускания. Таким образом, сумма пропускных способностей этих классов не сможет превысить ограничение родительского класса.

Старшие номера дескрипторов дочерних классов (classid) наследуют старший номер родительского класса.

```
# tc qdisc add dev eth0 parent 1:3 handle 30: sfq
# tc qdisc add dev eth0 parent 1:4 handle 40: sfq
```

При создании, к каждому из классов, по-умолчанию присоединяется дисциплина FIFO, однако, для более равномерного распределения пропускной способности между соединениями, присоединим к каждому из классов дисциплину обработки очереди SFQ.

```
# tc filter add dev eth0 parent 1:0 protocol ip prio 1 u32 match ip \
  sport 80 0xffff flowid 1:3
# tc filter add dev eth0 parent 1:0 protocol ip prio 1 u32 match ip \
  sport 25 0xffff flowid 1:4
```

В заключение, трафик классифицируется с помощью фильтров и направляется в нужные классы.

Обратите внимание: команда **tc class add** СОЗДАЕТ класс в пределах дисциплины, а **tc qdisc add** -- добавляет дисциплину к классу.

У вас может возникнуть резонный вопрос: "Что будет с трафиком, который не подпадает под условия установленных фильтров?". В этом случае трафик останется неклассифицированым и будет обработан корневой дисциплиной 1:0, т.е. пройдет без ограничений.

Если сумма SMTP+web трафиков превысят сконфигурированные 6 мегабит, то вся полоса пропускания будет разделена между классами, в соответствии с их весами. Таким образом WEB-сервер получит 5/8 ширины канала, а SMTP-сервер -- 3/8.

В соответствии с данной конфигурацией можно утверждать, что WEB-сервер всегда будет иметь полосу, как минимум 5/8*6=3.75 мегабита.

9.5.4.5. Прочие параметры настройки CBQ: split и defmap.

Как уже говорилось выше, для классификации трафика, полноклассовые дисциплины используют фильтры.

Но кроме фильтров, CBQ может предложить вам параметры split и defmap. Хотя назначение этих параметров достаточно сложно понять, и к тому же они не являются жизненно необходимыми, тем не менее я постараюсь описать их.

Так как наиболее часто классификация трафика производится только на основе поля TOS, в заголовке пакета, то предусматривается специальный синтаксис команд. Каждый раз, когда CBQ сталкивается с необходимостью определения -- в какую из очередей поставить пакет, она проверяет -- является ли этот узел "узлом разбиения" и если это так, то выбирается подочередь с заданным приоритетом, который может быть рассчитан исходя из значения поля TOS.

Значение приоритета пакета складывается по "И" с параметром *defmap* и проверяется -- есть ли совпадение. Проще говоря -- это самый простой способ создания высокоскоростных фильтров, которые работают с незначительным числом приоритетов. С параметром *defmap*, равным 0xFF будет совпадать любой пакет, 0x00 -- ни один. Возможно пример настройки поможет вам полнее понять вышесказанное:

```
# tc qdisc add dev eth1 root handle 1: cbq bandwidth 10Mbit allot 1514 \
```

```
  cell 8 avpkt 1000 mpu 64

# tc class add dev eth1 parent 1:0 classid 1:1 cbq bandwidth 10Mbit    \
  rate 10Mbit allot 1514 cell 8 weight 1Mbit prio 8 maxburst 20        \
  avpkt 1000
```

Самое обычное начало для CBQ. Значения для параметра *defmap* можно определить из следующей таблицы:

```
TC_PRIO..        Число  Значение поля TOS
------------------------------------------------
BESTEFFORT         0    Maximize Reliablity (0x04) (Максисальная надежность)
FILLER             1    Minimize Cost (0x02)       (Минимальная стоимость)
BULK               2    Maximize Throughput (0x08) (Максимальная пропускная
способность)
INTERACTIVE_BULK   4
INTERACTIVE        6    Minimize Delay (0x10)      (Минимальная задержка)
CONTROL            7
```

Уровень приоритета TC_PRIO.. рассчитывается исходя из значения поля TOS (за дополнительной информацией о значениях приоритета пакета, обращайтесь к разделу pfifo_fast).

Теперь создадим классы, через которые пойдет интерактивный и объемный трафик:

```
# tc class add dev eth1 parent 1:1 classid 1:2 cbq bandwidth 10Mbit     \
  rate 1Mbit allot 1514 cell 8 weight 100Kbit prio 3 maxburst 20        \
  avpkt 1000 split 1:0 defmap c0

# tc class add dev eth1 parent 1:1 classid 1:3 cbq bandwidth 10Mbit     \
  rate 8Mbit allot 1514 cell 8 weight 800Kbit prio 7 maxburst 20        \
  avpkt 1000 split 1:0 defmap 3f
```

В данном случае "узлом разбиения" назначается дисциплина 1:0, это та точка, где будет делаться выбор. Число 0xC0 в двоичном представлении имеет вид 11000000, а 0x3F -- 00111111, таким образом оба класса перекрывают весь диапазон возможных приоритетов. Первому классу будут соответствовать пакеты, приоритеты которых имеют 6 и/или 7 биты в установленном состоянии, что соответствует интерактивному и управляющему трафику. Ко второму классу будут отнесены все остальные пакеты.

Таблица выбора для узла 1:0 теперь будет иметь следующий вид:

```
приоритет       класс
0                     1:3
1                     1:3
2                     1:3
3                     1:3
4                     1:3
5                     1:3
6                     1:2
7                     1:2
```

Кроме того, можно изменять приоритеты отдельных видов трафика. Для этого используется команда вида: **tc class change**, например, чтобы повысить приоритет трафика best effort и классифицировать его, как принадлежащий классу 1:2, нужно дать следующую команду:

```
# tc class change dev eth1 classid 1:2 cbq defmap 01/01
```

В этом случае, таблица выбора будет иметь следующий вид:

```
приоритет       класс
0                       1:2
1                       1:3
2                       1:3
3                       1:3
4                       1:3
5                       1:3
6                       1:2
7                       1:2
```

FIXME: Корректность работы команды **tc class change** не проверена. Выводы были сделаны исключительно на основе изучения исходных текстов.

9.5.5. Hierarchical Token Bucket

Мартин Девера (Martin Devera) aka <devik> справедливо отмечает, что CBQ слишком сложна и слабо оптимизирована для большинства типичных ситуаций. Его подход более точно соответствует конфигурациям, когда необходимо распределить заданную полосу пропускания между различными видами трафика на полосы гарантированной ширины, с возможностью заимствования.

HTB работает точно так же, как и CBQ, но, в отличие от последней, принцип работы основан не на вычислении времени простоя, а на определении объема трафика, что полностью соответствует названию Token Bucket Filter. Эта дисциплина имеет незначительное число параметров настройки, которые достаточно хорошо описаны на сайте http://luxik.cdi.cz/~devik/qos/htb/.

Хотя конфигурирование HTB -- задача достаточно сложная, тем не менее конфигурации хорошо масштабируются. В случае же с CBQ процесс конфигурирования становится слишком сложным даже в самых простых случаях! HTB3 теперь стала частью ядра (начиная с версий 2.4.20-pre1 и 2.5.31). Однако, вам может потребоваться пропатченная версия утилиты **tc**: старший номер версии HTB в ядре и пользовательских утилит должны совпадать, в противном случае **tc** откажется работать с HTB.

Если у вас установлено достаточно свежее или пропатченное ядро, вам определенно стоит посмотреть в сторону HTB!

9.5.5.1. Пример конфигурации.

Конфигурация практически идентична вышеприведенному примеру:

```
# tc qdisc add dev eth0 root handle 1: htb default 30

# tc class add dev eth0 parent 1: classid 1:1 htb rate 6mbit burst 15k

# tc class add dev eth0 parent 1:1 classid 1:10 htb rate 5mbit burst 15k
# tc class add dev eth0 parent 1:1 classid 1:20 htb rate 3mbit ceil 6mbit burst 15k
# tc class add dev eth0 parent 1:1 classid 1:30 htb rate 1kbit ceil 6mbit burst 15k
```

Автор рекомендует устанавливать дисциплину SFQ для этих классов:

```
# tc qdisc add dev eth0 parent 1:10 handle 10: sfq perturb 10
# tc qdisc add dev eth0 parent 1:20 handle 20: sfq perturb 10
# tc qdisc add dev eth0 parent 1:30 handle 30: sfq perturb 10
```

Добавим фильтры, которые будут выполнять классификацию трафика:

```
# U32="tc filter add dev eth0 protocol ip parent 1:0 prio 1 u32"
# $U32 match ip dport 80 0xffff flowid 1:10
# $U32 match ip sport 25 0xffff flowid 1:20
```

В результате получаем ясную и понятную конфигурацию -- никаких малопонятных чисел, никаких недокументированных параметров.

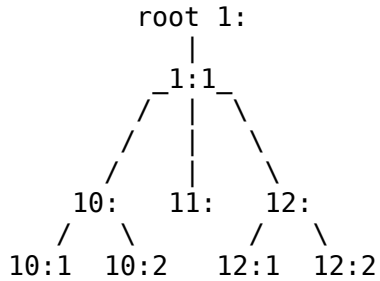
В HTB все выглядит достаточно прозрачно -- классы 10: и 20: имеют гарантированную пропускную способность, при наличии свободной части пропускной способности они заимствуют ее в отношении 5:3.

Неклассифицированый трафик будет отнесен к классу 30:, который имет достаточно небольшую ширину, но может заимствовать незанятую часть канала.

9.6. Классификация пакетов с помощью фильтров.

Для классификации того или иного пакета, всякий раз вызывается так называемая "цепочка классификации". Эта цепочка состоит из всех фильтров, присоединенных к полноклассовой дисциплине.

Вернемся к дереву:



Когда пакет необходимо поставить в очередь, проверяется каждая ветвь в цепочке фильтров. Например, некоторый пакет мог бы быть направлен фильтром 1:1 в класс 12: и далее в 12:2.

Фильтр, который сразу отправляет пакет в 12:2, мог бы быть сразу присоединен к 1:1, но в этом случае пакет минует дополнительные проверки, которые могли бы быть сделаны в 12:.

Вы не можете выполнять фильтрацию в обратном порядке, только вниз! Кроме того, все фильтры в HTB должны присоединяться к корню!

Повторюсь еще раз. Пакеты могут фильтроваться (классифицироваться) только в направлении сверху-вниз.

9.6.1. Ряд простых примеров фильтрации.

Для начала продемонстрируем самый обычный случай, который, к счастью, достаточно прост. Допустим, что у нас имеется дисциплина PRIO, с дескриптором 10:, которая содержит три класса и нам нужно весь трафик, отправляемый на 22 порт и с 80 порта, отдать в самую высокоприоритетную полосу, тогда набор фильтров мог бы выглядеть так:

```
# tc filter add dev eth0 protocol ip parent 10: prio 1 u32 match \
  ip dport 22 0xffff flowid 10:1
# tc filter add dev eth0 protocol ip parent 10: prio 1 u32 match \
  ip sport 80 0xffff flowid 10:1
# tc filter add dev eth0 protocol ip parent 10: prio 2 flowid 10:2
```

О чем говорят эти строки? Они говорят: "Присоединить к eth0, к узлу 10:, фильтр u32, с приоритетом 1, который отбирает пакеты, направляемые на порт 22, и передает их в полосу 10:1". Аналогичное высказывание делается относительно пакетов, отправленных с порта 80. И последняя строка говорит о том, что весь неклассифицированный трафик должен отправляться в полосу 10:2.

Вы обязательно должны указывать имя сетевого интерфейса, поскольку каждый из них имеет свое уникальное пространство имен дескрипторов.

Отбор пакетов по IP-адресам выполняется аналогичным образом:

```
# tc filter add dev eth0 parent 10:0 protocol ip prio 1 u32 \
  match ip dst 4.3.2.1/32 flowid 10:1
# tc filter add dev eth0 parent 10:0 protocol ip prio 1 u32 \
  match ip src 1.2.3.4/32 flowid 10:1
# tc filter add dev eth0 protocol ip parent 10: prio 2      \
  flowid 10:2
```

В этой конфигурации весь трафик, идущий на хост 4.3.2.1 и с хоста 1.2.3.4, ставится в очередь с наивысшим приоритетом.

Допускается смешивать проверку IP-адреса и номера порта:

```
# tc filter add dev eth0 parent 10:0 protocol ip prio 1 u32 match ip src
4.3.2.1/32 \
  match ip sport 80 0xffff flowid 10:1
```

9.6.2. Наиболее употребимые способы фильтрации.

В большинстве случаев, команды фильтрации начинаются так:

```
# tc filter add dev eth0 parent 1:0 protocol ip prio 1 u32 ..
```

Это так называемый селектор u32, который может выполнять отбор пакетов по любой его части.

По IP-адресу

> По исходящему адресу: **match ip src 1.2.3.0/24**, по адресу назначения: **match ip dst 4.3.2.0/24**. Чтобы отобрать пакеты, отправляемые с/на единственный узел сети нужно указать сетевую маску /32 или вообще опустить ее.

По номеру порта

Исходящий порт: **match ip sport 80 0xffff**, порт назначения: **match ip dport 80 0xffff**.

По типу протокола, из семейства IP (tcp, udp, icmp, gre, ipsec)

Номера протоколов вы найдете у себя, в файле `/etc/protocols`. Например, отбор ICMP-пакетов (номер протокола icmp -- 1) можно выполнить командой **match ip protocol 1 0xff**.

По метке пакета

Пакеты могут маркироваться либо средствами **ipchains**, либо средствами **iptables**. Это бывает полезно, например, для наложения ограничений на трафик, который следует с интерфейса eth1 на интерфейс eth0:

```
# tc filter add dev eth1 protocol ip parent 1:0 prio 1 handle 6 fw flowid 1:1
```

Обратите внимание -- этот фильтр не является селектором u32!

Установить метку можно следующим образом:

```
# iptables -A PREROUTING -t mangle -i eth0 -j MARK --set-mark 6
```

В данном случае число 6 выбрано произвольным образом.

Если вы не хотите с головой погружаться в синтаксис **tc**, тогда просто пользуйтесь возможностями **iptables** и запомните принцип отбора по метке.

По полю TOS

Отбор трафика с минимальной задержкой:

```
# tc filter add dev ppp0 parent 1:0 protocol ip prio 10 u32 \
      match ip tos 0x10 0xff \
     flowid 1:4
```

Дополнительные сведения о фильтрации трафика вы найдете в главе Расширенная фильтрация

9.7. Intermediate queueing device.

Устройство IMQ не является дисциплиной обработки очереди, но тесно с ними связано. В Linux, дисциплины обработки очередей присоединяются к сетевым устройствам и все, что помещается в очередь устройства, попадает сначала в очередь дисциплины обработки очереди. Из-за этого подхода существуют два ограничения:

1. Ограничение пропускной способности полноценно работает только для исходящего трафика (дисциплина обработки входящего трафика существует, но ее возможности мизерны по сравнению с полноклассовыми дисциплинами).

2. Дисциплина обработки очереди обслуживает трафик только для одного интерфейса, нет никакой возможности задать глобальные ограничения.

Устройство IMQ пытается решить эти проблемы. С помощью подсистемы фильтрации ОС Linux можно определенные пакеты направлять через этот псевдо-интерфейс, к которому подключаются различные дисциплины обработки очередей. Таким образом, можно управлять полосой пропускания, как входящего, так и общего трафика.

9.7.1. Пример конфигурирования.

Первое, что может прийти на ум -- попытаться сформировать входящий трафик так, чтобы дать себе наивысший приоритет ;) Конфигурация IMQ очень похожа на конфигурацию любого другого интерфейса:

```
tc qdisc add dev imq0 root handle 1: htb default 20

tc class add dev imq0 parent 1: classid 1:1 htb rate 2mbit burst 15k

tc class add dev imq0 parent 1:1 classid 1:10 htb rate 1mbit
tc class add dev imq0 parent 1:1 classid 1:20 htb rate 1mbit

tc qdisc add dev imq0 parent 1:10 handle 10: pfifo
tc qdisc add dev imq0 parent 1:20 handle 20: sfq

tc filter add dev imq0 parent 10:0 protocol ip prio 1 u32 match \
                ip dst 10.0.0.230/32 flowid 1:10
```

В этом примере для классификации пакетов используется селектор u32, однако, при работе с imq, могут использоваться и другие селекторы.

Отбираемый трафик маркируется средствами **iptables**:

```
iptables -t mangle -A PREROUTING -i eth0 -j IMQ --todev 0

ip link set imq0 up
```

Действие IMQ в **iptables** допускается использовать только в цепочках PREROUTING и POSTROUTING, таблицы mangle. Его синтаксис:

```
IMQ [ --todev n ]
```

где n -- это номер устройства imq.

Это действие существует и в **ip6tables**.

Обратите внимание: трафик попадает на интерфейс не в тот момент, когда переходит к цели IMQ, а после обработки соответствующей цепочки. Точное место входа на устройство imq зависит от типа трафика (входящий/исходящий).

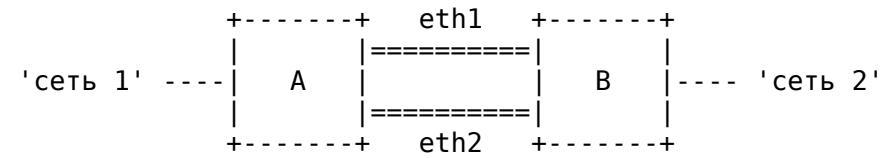
Для входящего трафика imq регистрируется с приоритетом NF_IP_PRI_MANGLE + 1. Это означает, что пакеты попадают на устройство сразу после прохождения цепочки PREROUTING. Для исходящего трафика, imq использует приоритет NF_IP_PRI_LAST, который гарантирует, что пакеты, уничтоженные пакетным фильтром не будут занимать полосу пропускания устройства.

Дополнительные сведения и "заплаты" вы найдете на сайте http://luxik.cdi.cz/~patrick/imq/

Глава 10. Распределение нагрузки по нескольким интерфейсам.

Существует несколько способов реализации такой схемы. Простейший и непосредственный способ представляет 'TEQL'. Это простейший компенсатор (True/Trivial link Equalizer). Как и большая часть того, что связано с очередями, балансировка нагрузки работает в обе стороны. Для полного эффекта обе стороны соединения должны работать совместно.

Представим ситуацию:



A и *B* это маршрутизаторы, и пока мы будем предполагать, что оба работают под управлением Linux. Для передачи данных из сети 1 в сеть 2, маршрутизатору *A* необходимо распределить пакеты по обоим каналам. Маршрутизатор *B* должен быть настроен соответствующим образом, чтобы поддерживать это. Тоже самое относится и к передаче данных из сети 2 в сеть 1. Маршрутизатор *B* должен передавать данные по двум интерфейсам -- eth1 и eth2.

Распределение выполняется устройством 'TEQL'. Вот необходимые команды (куда уж проще):

```
# tc qdisc add dev eth1 root teql0
# tc qdisc add dev eth2 root teql0
# ip link set dev teql0 up
```

Не забудьте команду **ip link set up**!

Приведенные команды должны быть выполнены на каждом из маршрутизаторов. Устройство *teql0* представляет собой циклический распределитель пакетов по интерфейсам eth1 и eth2. На устройство teql никогда не приходят данные, их получают интерфейсы eth1 и eth2.

Но пока что у нас есть только устройства и нам нужно настроить маршрутизацию. Один из способов сделать это -- назначить сеть /31 каждому соединению, в том числе и устройствам *teql0*:

На маршрутизаторе *A*:

```
# ip addr add dev eth1 10.0.0.0/31
# ip addr add dev eth2 10.0.0.2/31
# ip addr add dev teql0 10.0.0.4/31
```

На маршрутизаторе *B*:

```
# ip addr add dev eth1 10.0.0.1/31
# ip addr add dev eth2 10.0.0.3/31
# ip addr add dev teql0 10.0.0.5/31
```

Теперь маршрутизатор *A* должен "пинговать" адреса 10.0.0.1, 10.0.0.3 и 10.0.0.5 по двум реальным

каналом и одному компенсаторному устройству. Аналогично, маршрутизатор *B* должен пинговать 10.0.0.0, 10.0.0.2 и 10.0.0.4.

Если все работает, на маршрутизаторе *A* маршрут к сети 2 должен проходить через 10.0.0.5, а маршрутизатор *B* должен иметь адрес 10.0.0.4 в качестве маршрута к сети 1. Для отдельного случая, когда сеть 1 это ваша домашняя сеть, а сеть 2 -- Internet, маршрутизатор *A* должен иметь шлюз по умолчанию 10.0.0.5.

10.1. Предостережение.

Не все так просто как кажеться. На интерфейсах eth1 и eth2, маршрутизаторов *A* и *B*, необходимо отключить фильтрацию по адресу возврата (return path filtering), иначе будут фильтроваться пакеты, предназначеные для другого интерфейса:

```
# echo 0 > /proc/sys/net/ipv4/conf/eth1/rp_filter
# echo 0 > /proc/sys/net/ipv4/conf/eth2/rp_filter
```

Следующая проблема -- это порядок пакетов. Допустим нужно передать 6 пакетов от *A* к B. По интерфейсу eth1 могут пройти пакеты 1, 3 и 5. Соответственно по eth2 пройдут 2, 4 и 6. В идеальном мире, маршрутизатор B получил бы эти пакеты в порядке 1, 2, 3, 4, 5, 6. Но в нашем мире вероятность того, что порядок изменится, очень велика. Например, порядок может быть таким: 2, 1, 4, 3, 6, 5. И это может запутать стек TCP/IP. Хотя эта проблема не так важна, для каналов с большим количеством разных сеансов TCP/IP, тем не менее вы не сможете ощутить значительный прирост скорости при передаче одного файла по ftp, разве только отпровляющая или получающая ОС -- не Linux, которую весьма не просто шокировать простым изменением порядка пакетов.

Однако, для многих приложений балансировка нагрузки является хорошим решением.

10.2. Другие возможности.

Уильям Стирнс (William Stearns) использовал сложную конфигурацию тоннелей для достижения эффективного использования нескольких несвязанных подключений к сети Internet. Информацию об этом можно найти здесь.

В дальнейшем данный документ, вероятно, будет содержать больше информации по этому вопросу.

Глава 11. Netfilter и iproute -- маркировка пакетов.

До сих пор мы детально разбирались с работой **iproute** и лишь вскользь упомянули **netfilter**. Теперь настало самое время поговорить о нем. Для начала рекомендую вам прочитать Remarkably Unreliable Guides.

Netfilter позволяет выполнять фильтрацию трафика и вносить изменения в заголовки пакетов. Одна из замечательных особенностей netfilter -- это возможность устанавливать числовые метки на пакеты.

Например, следующее правило пометит все пакеты, отправляемые на порт 25:

```
# iptables -A PREROUTING -i eth0 -t mangle -p tcp --dport 25 \
 -j MARK --set-mark 1
```

Допустим, что у нас имеется два подключения к Интернет -- одно быстрое, но дорогое, другое медленное, зато дешевое. Естественно, что мы предпочтем отправлять почту по дешевому маршруту. Командой выше, мы уже пометили все пакеты исходящей почты числовой меткой -- 1, теперь попробуем направить эти пакеты по дешевому маршруту:

```
# echo 201 mail.out >> /etc/iproute2/rt_tables
# ip rule add fwmark 1 table mail.out
# ip rule ls
0:      from all lookup local
32764:  from all fwmark        1 lookup mail.out
32766:  from all lookup main
32767:  from all lookup default
```

и зададим пусть и медленный, зато недорогой маршрут в таблице mail.out:

```
# /sbin/ip route add default via 195.96.98.253 dev ppp0 table mail.out
```

Это собственно все, что нам пришлось сделать! Если нужно сделать исключения для отдельных хостов, то можно несколько модифицировать набор правил для netfilter, пропуская пакеты с отдельных хостов без метки, или можно добавить правило с более низким приоритетом, которое отправляет отдельные хосты через таблицу main.

Можно так же использовать поле TOS, в заголовке пакета, задавая различный тип обслуживания для трафика различного рода, и создавая правила, которые реагируют на это поле.

Само собой разумеется, все это прекрасно работает и на хостах, под NAT ('masquerading').

Некоторые наши читатели отмечают, что как минимум MASQ и SNAT конфликтуют с механизмом маркировки пакетов. Расти Рассел (Rusty Russell) описал эту проблему (http://lists.samba.org/pipermail/netfilter/2000-November/006089.html). Просто, отключите фильтр проверки обратного адреса (см. главу Параметры настройки сети в ядре) и все должно заработать.

Чтобы иметь возможность маркировать пакеты, вы должны собрать ядро с рядом включенных опций:

```
IP: advanced router (CONFIG_IP_ADVANCED_ROUTER) [Y/n/?]
IP: policy routing (CONFIG_IP_MULTIPLE_TABLES) [Y/n/?]
IP: use netfilter MARK value as routing key (CONFIG_IP_ROUTE_FWMARK) [Y/n/?]
```

См. так же раздел Прозрачное проксирование с помощью netfilter, iproute2, ipchains и squid

Глава 12. Расширенная фильтрация.

Как уже говорилось в разделе Классовые дисциплины обработки очередей, для того, чтобы определить в какую из подочередей направить пакет, используются фильтры-классификаторы.

Ниже приводится неполный список доступных классификаторов:

fw

Решение принимается на основе маркера пакета, установленного брандмауэром (например -- iptables). Наиболее простой классификатор, который можно рекомендовать в том случае, если вам не хочется изучать синтаксис **tc**. За дополнительной информацией обращайтесь к главе 9.

u32

Решение принимается на основе значений полей в заголовке пакета (например, исходящий IP-адрес и т.п.).

route

Решение принимается на основе маршрута, по которому движется пакет.

rsvp, rsvp6

Маршрутизация пакетов производится на базе RSVP. Применимо только в том случае, если управление сетью полностью находится в ваших руках. В Интернет RSVP не поддерживается.

tcindex

Используется в **DSMARK** qdisc, см. соответствующий раздел.

Вообще есть множество способов классификации пакетов, но практически все они находятся в прямой зависимости от предпочитаемой вами системы.

Классификаторы, как правило, принимают некоторое количество аргументов. Перечислим их здесь, для удобства.

protocol

Протокол, принимаемый классификатором. Как правило вы будете принимать только IP-трафик.

parent

Существующий класс, к которому должен быть присоединен данный классификатор.

prio

Приоритет классификатора. Чем меньше число -- тем выше приоритет.

handle

Назначение и смысл аргумента зависит от контекста использования.

Во всех следующих разделах мы будем исходить из условия, что формируется трафик, идущий к хосту *HostA*, что корневой класс сконфигурирован как 1:, а класс, которому посылается выбранный трафик -- как 1:1.

12.1. Классификатор u32.

Фильтр **U32** наиболее гибкий из доступных в текущей конфигурации. Он целиком основан на хеш-таблицах, которые повышают устойчивость фильтра при значительном количестве правил фильтрации.

В простейшем виде, фильтр **U32** -- это набор записей, каждая из которых состоит из двух полей: селектора и действия. Селекторы, описанные ниже, проверяют обрабатываемый IP-пакет до тех пор, пока не будет встречено первое совпадение, после чего выполняется соответствующее селектору действие. Самый простой тип действия -- перенаправление пакета в определенный класс.

Для конфигурирования фильтра используется команда **tc filter**, состоящая из трех частей: определение фильтра, селектор и действие. Определение фильтра может быть записано как:

```
tc filter add dev IF [ protocol PROTO ]
                     [ (preference|priority) PRIO ]
                     [ parent CBQ ]
```

Поле *protocol* описывает обслуживаемый протокол. Здесь мы будем обсуждать исключительно протокол IP. Поле *preference* (в качестве синонима можно использовать *priority*) описывает приоритет определяемого фильтра, что позволяет задавать несколько фильтров (списков правил) с различными приоритетами. Вообще, правила обслуживаются в порядке добавления в список, в случае с приоритетами -- первыми обслуживаются правила, имеющие наивысший приоритет (чем меньше число, тем выше приоритет). Поле *parent* определяет вершину дерева CBQ (например 10:1), к которой должен быть присоединен данный фильтр.

Описаные выше опции применимы ко всем фильтрам, а не только к **U32**.

12.1.1. Селектор U32.

Селектор **U32** содержит определение шаблона, который будет сопоставляться с обрабатываемым пакетом. Если быть более точным, он определяет -- какие биты в заголовке пакета будут проверяться и не более того, но, не смотря на свою простоту, это очень мощный и гибкий метод. Рассмотрим примеры, взятые из реально работающего и достаточно сложного фильтра:

```
# tc filter add dev eth0 protocol ip parent 1:0 pref 10 u32 \
  match u32 00100000 00ff0000 at 0 flowid 1:10
```

Оставим пока первую строку в покое, эти параметры описывают хеш-таблицы фильтра, и сконцентрируем свое внимание на строке селектора, которая содержит ключевое слово *match*. Этот селектор будет отбирать пакеты, в IP-заголовках которых второй байт будет содержать число 0x10 (0010). Как вы уже наверняка догадались, 00ff -- это маска, которая точно определяет проверяемые биты. Ключевое слово *at* означает, что поиск совпадения должен начинаться с указанного смещения (в байтах), в данном случае -- с начала пакета. Переведя все это, на

человеческий язык, можно сказать, что пакет будет соответствовать селектору, если в его поле TOS (Type of Service) будет установлен бит *Minimize-Delay* (минимальная задержка). Проанализируем еще одно правило:

```
# tc filter add dev eth0 protocol ip parent 1:0 pref 10 u32 \
  match u32 00000016 0000ffff at nexthdr+0 flowid 1:10
```

Параметр *nexthdr* означает переход к следующему заголовку в IP-пакете, т.е. к заголовку протокола более высокого уровня. Опять же, в данной ситуации поиск будет вестись с начала заголовка. Анализу будет подвергнуто второе 32-х битное слово в заголовке. В протоколах TCP и UDP это поле содержит порт назначения. Число записывается в формате *big-endian*, т.е. первым указывается старший байт. Таким образом мы получаем номер порта назначения -- 0x0016, или 22 (в десятичной форме). В случае протокола TCP, этот порт соответствует службе SSH. Надеюсь вы понимаете, что данное соответствие бессмысленно обсуждать вне контекста применения, поэтому отложим эту дискуссию на более позднее время.

Уловив все, что говорилось выше, вы без труда поймете смысл следующего селектора: `match c0a80100 ffffff00 at 16`. Данный селектор будет пытаться найти 3-х байтовую последовательность в IP-заголовке, начиная с 17-го байта, отсчитываемого от начала заголовка, что соответствует любому адресу назначения в сети 192.168.1.0/24.

12.1.2. Селекторы общего назначения.

Селекторы общего назначения задают шаблон, маску и смещение. Используя эти селекторы вы сможете выполнять проверку практически любого, отдельно взятого бита в заголовке IP (или протокола более высокого уровня). При написании и чтении они более сложны, чем селекторы специального назначения, которые будут обсуждаться в следующем разделе. Синтаксис селекторов общего назначения:

```
match [ u32 | u16 | u8 ] PATTERN MASK [ at OFFSET | nexthdr+OFFSET]
```

Ключевое слово **u32**, или **u16**, или **u8** указывает длину шаблона в битах. PATTERN и MASK в обязательном порядке должны иметь длину, указанную в предыдущем ключевом слове. Параметр OFFSET задает смещение от начала заголовка в байтах. Если присутствует ключевое слово *nexthdr+*, то смещение начинает отсчитываться от начала заголовка протокола более высокого уровня.

Приведем несколько примеров:

Этим селектором будут отобраны пакеты, у которых "время жизни" (поле TTL) равно 64. Поле TTL находится в 9-м (в 8-м, если считать с нуля) байте IP-заголовка.

```
# tc filter add dev ppp14 parent 1:0 prio 10 u32 \
     match u8 64 0xff at 8 \
     flowid 1:4
```

Следующие селекторы отберут TCP-пакеты, в которых установлен бит ACK:

```
# tc filter add dev ppp14 parent 1:0 prio 10 u32 \
     match ip protocol 6 0xff \
     match u8 0x10 0xff at nexthdr+13 \
     flowid 1:3
```

Отбор ACK-пакетов, длина которых меньше 64 байт:

```
## отбор ack-пакетов более сложным способом,
## IP protocol 6,
## Длина IP-заголовка 0x5 (32-х битных слов),
## Общая длина 0x34 (ACK + 12 байт опций TCP)
## TCP ACK (бит 5, смещение 33)
# tc filter add dev ppp14 parent 1:0 protocol ip prio 10 u32 \
            match ip protocol 6 0xff \
            match u8 0x05 0x0f at 0 \
            match u16 0x0000 0xffc0 at 2 \
            match u8 0x10 0xff at 33 \
            flowid 1:3
```

Это правило отберет пакеты TCP, с установленным битом ACK, и не несущие в себе данных. Это яркий пример использования двух селекторов. Конечный результат будет получен логическим умножением (операция "И") результатов каждого из селекторов. Если вы посмотрите на диаграмму TCP-заголовка, то увидите, что флаг ACK -- это младший бит старшей тетрады (0x10) 14-го байта в TCP-заголовке (*at nexthdr+13*). Что касается второго селектора, то если бы мы хотели усложнить себе жизнь, можно было бы записать `match u8 0x06 0xff at 9`, вместо специального селектора *protocol tcp* (здесь число 6 -- это номер протокола TCP), в 10-м байте IP-заголовка. С другой стороны, в этом примере мы не использовали специальный селектор, для отбора по биту ACK, просто потому, что такого селектора не существует.

Фильтр, приведенный ниже, является модификацией предыдущего примера. На этот раз не производится проверка длины IP-заголовка. Вы спросите -- почему? Потому, что фильтр, приведенный выше, работает только на 32-х битных системах.

```
tc filter add dev ppp14 parent 1:0 protocol ip prio 10 u32 \
     match ip protocol 6 0xff \
     match u8 0x10 0xff at nexthdr+13 \
     match u16 0x0000 0xffc0 at 2 \
     flowid 1:3
```

12.1.3. Селекторы специального назначения.

Следующая таблица содержит перечень селекторов специального назначения, которые автор данного раздела нашел в исходном коде утилиты **tc**. Они облегчат вам жизнь и повысят удобочитаемость ваших фильтров.

FIXME: таблица находится в отдельном файле `selector.html`.

FIXME: Она по-прежнему остается на польском языке :-(

FIXME: надо перевести в формат sgml.

Несколько примеров:

```
# tc filter add dev ppp0 parent 1:0 prio 10 u32 \
     match ip tos 0x10 0xff \
     flowid 1:4
```

FIXME: шаблон *tcp dport*, в примере ниже, не работает.

Правило выше отберет пакеты, в которых поле TOS имеет значение 0x10. Поле TOS -- это второй байт в IP-заголовке, причем это однобайтовое поле (8 бит). Таким образом, эквивалентный селектор можно было бы записать как: *match u8 0x10 0xff at 1*. Это наталкивает на мысль, что селекторы специального назначения, внутри фильтров **U32**, преобразуются в селекторы общего назначения и в таком виде сохраняются в памяти ядра. Отсюда следует другое заключение -- селекторы *tcp* и *udp* суть есть одно и то же. По этой причине вы не сможете использовать единичный селектор *match tcp dport 53 0xffff*, для отбора TCP-пакетов, направляющихся на 53-й порт. Этим селектором будут отобраны как TCP-пакеты, так и UDP-пакеты. Вы обязательно должны добавлять селектор типа протокола:

```
# tc filter add dev ppp0 parent 1:0 prio 10 u32 \
        match tcp dport 53 0xffff \
        match ip protocol 0x6 0xff \
        flowid 1:2
```

12.2. Классификатор route.

Этот классификатор тесно связан с таблицами маршрутизации. Когда пакет достигает такого фильтра, то дальнейшая его судьба зависит от информации, находящейся в таблицах маршрутизации.

```
# tc filter add dev eth1 parent 1:0 protocol ip prio 100 route
```

В данном случае, к узлу 1:0 добавлен классификатор с приоритетом 100. Когда пакет достигнет этого узла (поскольку он является корневым, то это произойдет немедленно), то классификатор проанализирует таблицу маршрутизации. Если совпадение будет подтверждено, то пакет будет передан данному классу с приоритетом, равным 100. Однако, для того, чтобы все это заработало, необходимо добавить соответствующую запись в таблицу маршрутизации:

Тут вся хитрость состоит в том, чтобы определить некую 'область' (realm), основываясь на адресе отправителя или получателя. Примерно таким образом:

```
# ip route add Host/Network via Gateway dev Device realm RealmNumber
```

Для примера зададим номер 'области', равным 10, для нашей сети 192.168.10.0.

```
# ip route add 192.168.10.0/24 via 192.168.10.1 dev eth1 realm 10
```

В фильтрах, основанных на классификаторе *route*, мы можем использовать номер 'области' (realm) для представления сетей или отдельных узлов сети.

```
# tc filter add dev eth1 parent 1:0 protocol ip prio 100 \
  route to 10 classid 1:10
```

Под действие данного правила попадут пакеты, направляющиеся в сеть 192.168.10.0

Фильтры данного типа можно строить и на основе адреса отправителя. Рассмотрим пример, когда к маршрутизатору, под управлением Linux, через интерфейс eth2 подключена сеть.

```
# ip route add 192.168.2.0/24 dev eth2 realm 2
# tc filter add dev eth1 parent 1:0 protocol ip prio 100 \
  route from 2 classid 1:2
```

Этот фильтр говорит о том, что пакеты из подсети 192.168.2.0 (область 2) будут отнесены к идентификатору класса 1:2.

12.3. Фильтры-ограничители трафика.

Более сложные системы управления трафиком можно строить на основе фильтров, которые зависят от пропускной способности и загруженности канала. Вы можете объявить фильтр, который не будет срабатывать, если загрузка канала превысила некоторую величину или наоборот, срабатывать только в том случае, если пропускная способность превысила заданное значение.

Так, если вы решили ограничить 5 Мбитный канал величиной в 4 Мбит/сек, вы можете вообще прекратить прием пакетов, если загруженность канала превысит 4 Мбит/сек, либо отбросить 1 Мбит/сек, а 4 Мбит/сек передать заданному классу.

При превышении заданного порога вы можете отбрасывать "лишние" пакеты, переклассифицировать их или передать другим фильтрам.

12.3.1. Способы ограничения.

Существует две основных возможности ограничения. Если вы собрали ядро с "функциями оценки" (*estimators*), то оно сможет более или менее точно измерять объем трафика, переданного тому или иному фильтру. Устройство функции оценки достаточно несложное -- 25 раз в секунду подсчитывается объем переданных данных, на основе которого вычисляется загруженность канала.

Другой способ -- Token Bucket Filter, который реализуется в пределах вашего фильтра. Это типичный шейпер, который может использоваться, если вам нужно просто ограничить полосу пропускания по какому-нибудь критерию.

12.3.1.1. Оценочная функция в ядре (estimator).

Тут все очень просто и имеется всего один параметр: *avrate*. Либо объем трафика остается ниже *avrate* и фильтр классифицирует его как сконфигурированный classid, либо, в случае превышения заданной границы, выполняется указанное действие, которое по-умолчанию выполняет переклассификацию.

Для оценки загруженности канала, ядро использует экспоненциальное взвешенное смещенное среднее значение, которое менее чувствительно к коротким пикам.

12.3.1.2. Token Bucket Filter.

Используются следующие параметры:

- burst/buffer/maxburst
- mtu/minburst
- mpu
- rate

Которые ведут себя идентично описанным в разделе Token Bucket Filter. Обратите внимание, если вы установите *mtu* слишком низким, то входящий трафик будет подавлен, в то время как исходящий TBF qdisc передаст с более низкой скоростью.

Еще одно важное отличие такого фильтра -- он может только либо пропустить пакет, либо отбросить. Он не может задержать пакет на какое-то время.

12.3.2. Действия в случае превышения ограничения.

Если правило "решит", что произошло превышение заданного предела, то оно выполнит соответствующее действие. Имеются четыре вида действий:

continue

Выполняется в случае несовпадения с условной частью правила, чтобы передать пакет следующему фильтру.

drop

Очень жестокое действие, которое просто отказывает "в праве на жизнь" трафику, объем которого превысил заданную величину. Часто используется во входных фильтрах и имеет ограниченное применение. Например, представим, что имеется сервер имен, который не в состоянии работать при нагрузке выше чем 5Мбит/сек, в этом случае можно построить входной фильтр, который ограничит входящий трафик для нашего сервера.

Pass/OK

Пропустить трафик. Может использоваться для того, чтобы отключить сложный фильтр, оставив его на месте.

reclassify

Действие, заданное по-умолчанию. Наиболее часто сводится к переклассификации в *Best Effort* (в данном контексте фразу "Best Effort" следует понимать как -- "лучшее из оставшегося". *прим. перев.*)

12.3.3. Примеры.

Единственный, известный нам, реальный пример приведен в разделе Защита от SYN flood

Ограничение входящего icmp-трафика до 2 Кбит. При превышении предела -- пакеты отбросить.

```
tc filter add dev $DEV parent ffff: \
    protocol ip prio 20 \
    u32 match ip protocol 1 0xff \
    police rate 2kbit buffer 10k drop \
    flowid :1
```

Ограничить размер пакетов (т.е. все пакеты, имеющие размер более чем 84 байта, будут сброшены)

```
tc filter add dev $DEV parent ffff: \
   protocol ip prio 20 \
   u32 match tos 0 0 \
   police mtu 84 drop \
   flowid :1
```

Этот метод может использоваться для полного подавления icmp-трафика:

```
tc filter add dev $DEV parent ffff: \
   protocol ip prio 20 \
   u32 match ip protocol 1 0xff \
   police mtu 1 drop \
   flowid :1
```

Фактически означает: "отбросить все icmp-пакеты, размер которых превышает 1 байт". Чисто теоретически, пакеты могут иметь размер в 1 байт, но на практике вы с ними никогда не встретитесь.

12.4. Хеш-фильтры.

Если у вас возникла потребность в большом количестве правил, например, когда у вас много клиентов, причем все имеют различные спецификации QoS (от англ. Quality of Service -- Качество Обслуживания), то может сложиться ситуация, когда ядро будет тратить недопустимо большое количество времени на поиск подходящего правила в наборе.

По-умолчанию, все фильтры находятся в одной большой цепочке, и располагаются в порядке убывания приоритетов. Если набор содержит 1000 правил, то для некоторых пакетов может потребоваться выполнить 1000 проверок, чтобы решить, что с ними делать дальше.

Поиск шел бы гораздо быстрее, если бы было 256 цепочек, по четыре правила в каждой, при условии, что вы сможете направить пакет в нужную цепочку.

В этом случае вам поможет хеширование. Представим, что у нас имеется сеть из 1024 компьютеров, с IP-адресами от 1.2.0.0 до 1.2.3.255, причем каждый компьютер может быть отнесен к одному из 3-х предопределенных классов качества обслуживания, например 'lite', 'regular' и 'premium'. Решение "в лоб" дает 1024 правила:

```
# tc filter add dev eth1 parent 1:0 protocol ip prio 100 match ip src \
  1.2.0.0 classid 1:1
# tc filter add dev eth1 parent 1:0 protocol ip prio 100 match ip src \
  1.2.0.1 classid 1:1
...
# tc filter add dev eth1 parent 1:0 protocol ip prio 100 match ip src \
  1.2.3.254 classid 1:3
# tc filter add dev eth1 parent 1:0 protocol ip prio 100 match ip src \
  1.2.3.255 classid 1:2
```

Чтобы уменьшить число проверок, можно использовать последний байт IP-адреса в качестве хеш-ключа. В результате получается 256 таблиц, первая из которых:

```
# tc filter add dev eth1 parent 1:0 protocol ip prio 100 match ip src \
  1.2.0.0 classid 1:1
# tc filter add dev eth1 parent 1:0 protocol ip prio 100 match ip src \
  1.2.1.0 classid 1:1
# tc filter add dev eth1 parent 1:0 protocol ip prio 100 match ip src \
  1.2.2.0 classid 1:3
# tc filter add dev eth1 parent 1:0 protocol ip prio 100 match ip src \
  1.2.3.0 classid 1:2
```

Следующая:

```
# tc filter add dev eth1 parent 1:0 protocol ip prio 100 match ip src \
  1.2.0.1 classid 1:1
...
```

Таким образом каждый пакет должен пройти не более 4-х проверок.

Реальная конфигурация намного сложнее, но она стоит того. Первым создается корневой фильтр, а затем -- таблица на 256 записей:

```
# tc filter add dev eth1 parent 1:0 prio 5 protocol ip u32
# tc filter add dev eth1 parent 1:0 prio 5 handle 2: protocol ip u32 divisor 256
```

После этого добавляются правила в созданные таблицы:

```
# tc filter add dev eth1 protocol ip parent 1:0 prio 5 u32 ht 2:7b: \
        match ip src 1.2.0.123 flowid 1:1
# tc filter add dev eth1 protocol ip parent 1:0 prio 5 u32 ht 2:7b: \
        match ip src 1.2.1.123 flowid 1:2
# tc filter add dev eth1 protocol ip parent 1:0 prio 5 u32 ht 2:7b: \
        match ip src 1.2.3.123 flowid 1:3
# tc filter add dev eth1 protocol ip parent 1:0 prio 5 u32 ht 2:7b: \
        match ip src 1.2.4.123 flowid 1:2
```

Это записи в таблице с номером 123, которые выполняют проверку на принадлежность адресам 1.2.0.123, 1.2.1.123, 1.2.2.123, 1.2.3.123, и в случае совпадения передают пакеты в классы 1:1, 1:2, 1:3 и 1:2 соответственно. Обратите внимание на то, как задается номер таблицы, шестнадцатеричное число 0x7b соответствует числу 123, в десятичном представлении.

И наконец создается хеш-фильтр, который перенаправит трафик в нужную таблицу:

```
# tc filter add dev eth1 protocol ip parent 1:0 prio 5 u32 ht 800:: \
        match ip src 1.2.0.0/16 \
        hashkey mask 0x000000ff at 12 \
```

```
        link 2:
```

А теперь поясним некоторые моменты. Заданной по-умолчанию хеш-таблице присвоен идентификатор 800:: и вся фильтрация начинается отсюда. Затем выбирается IP-адрес отправителя, который находится в 12, 13, 14 и 15 байтах в IP-заголовке и указывается, что нас интересует только последний байт. После чего трафик передается в хеш-таблицу 2:, которая была создана ранее.

Все это выглядит довольно сложным, но действительно работает и дает ошеломляющую производительность. Обратите внимание, этот пример может быть оптимизирован еще больше и сведен к идеальному случаю, когда каждая цепочка содержит 1 фильтр!

12.5. Фильтрация трафика IPv6.

12.5.1. Почему не работают tc-фильтры в IPv6?

Дело в том, что в ядре Linux, модель маршрутизации и адресации IPv4 (замечательные особенности которой описывает этот HOWTO) строилась на основе Базы Политик Маршрутизации (RPDB -- Routing Policy Database). К сожалению, модель IPv6 в Linux была реализована совершенно иным образом. Хотя они используют совместно некоторые средства, но основа основ -- RPDB, не принимает участия в адресации и маршрутизации IPv6.

Надеюсь, что такое положение дел наверняка изменится, надо только подождать.

FIXME: : Ждем ваших замечаний и предложений по этому поводу, может кто-то работает над этим?

12.5.2. Маркировка пакетов IPv6 средствами ip6tables.

ip6tables имеет возможность пометить пакеты:

```
# ip6tables -A PREROUTING -i eth0 -t mangle -p tcp -j MARK --mark 1
```

Но тем не менее, это крайне слабое утешение, поскольку пакет пройдет мимо RPDB.

12.5.3. Использование селектора u32 для пакетов IPv6.

Для передачи по сетям IPv4, трафик IPv6 обычно инкапсулируется в *SIT*-туннель. За дополнительной информацией по созданию такого рода туннелей, обращайтесь к разделу Тоннелирование IPv6. Это позволяет выполнять фильтрацию IPv4-пакетов, которые, в качестве полезной нагрузки, несут пакеты IPv6.

Следующий фильтр отберет все пакеты IPv6, инкапсулированные в IPv4:

```
# tc filter add dev $DEV parent 10:0 protocol ip prio 10 u32 \
            match ip protocol 41 0xff flowid 42:42
```

Продолжим в том же духе. Предположим, что пакеты IPv6 передаются по сети IPv4, и не имеют набора опций. Тогда, для обнаружения пакетов ICMPv6 можно использовать следующий фильтр. 0x3a (58) -- Next-Header для ICMPv6.

```
# tc filter add dev $DEV parent 10:0 protocol ip prio 10 u32 \
           match ip protocol 41 0xff \
           match u8 0x05 0x0f at 0 \
           match u8 0x3a 0xff at 26 \
           flowid 42:42
```

Фильтрация по адресу получателя потребует от нас дополнительных усилий. Например, следующий фильтр отбирает пакеты с адресом получателя 3ffe:202c:ffff:32:230:4fff:fe08:358d:

```
# tc filter add dev $DEV parent 10:0 protocol ip prio 10 u32 \
            match ip protocol 41 0xff \
            match u8 0x05 0x0f at 0 \
            match u8 0x3f 0xff at 44 \
            match u8 0xfe 0xff at 45 \
            match u8 0x20 0xff at 46 \
            match u8 0x2c 0xff at 47 \
            match u8 0xff 0xff at 48 \
            match u8 0xff 0xff at 49 \
            match u8 0x00 0xff at 50 \
            match u8 0x32 0xff at 51 \
            match u8 0x02 0xff at 52 \
            match u8 0x30 0xff at 53 \
            match u8 0x4f 0xff at 54 \
            match u8 0xff 0xff at 55 \
            match u8 0xfe 0xff at 56 \
            match u8 0x08 0xff at 57 \
            match u8 0x35 0xff at 58 \
            match u8 0x8d 0xff at 59 \
            flowid 10:13
```

Эту же методику можно использовать для отбора по адресу подсети, например 2001::

```
# tc filter add dev $DEV parent 10:0 protocol ip prio 10 u32 \
            match ip protocol 41 0xff \
            match u8 0x05 0x0f at 0 \
            match u8 0x20 0xff at 28 \
            match u8 0x01 0xff at 29 \
            flowid 10:13
```

Глава 13. Параметры настройки сети в ядре.

В ядре имеется масса параметров, которые могут быть изменены под различные нужды. Хотя, заданные по-умолчанию параметры удовлетворяют потребности в 99% случаев, но не зря же мы назвали это руководство Advanced HOWTO!

Очень интересные настройки вы найдете в `/proc/sys/net`, загляните туда. Конечно же, изначально не все тонкости будут описаны здесь, но мы работаем над этим.

Между прочим, значительная часть настроек описана в файле `Documentation/filesystems/proc.txt`, в дереве каталогов с исходными текстами ядра.

13.1. Reverse Path Filtering.

Reverse Path Filtering -- Проверка Обратного Адреса, хотя это слишком вольный перевод термина, но мне он кажется наиболее близким по смыслу. *прим. перев.*.

По-умолчанию, маршрутизаторы перенаправляют все подряд, даже пакеты, которые не принадлежат вашей сети. В качестве примера можно привести утечку локального трафика в Интернет. Если у вас имеется интерфейс с маршрутом к нему 195.96.96.0/24, то вы наверняка не ожидаете получить на него пакеты от 212.64.94.1.

В файловой системе `/proc` лежит файл, изменив который вы сможете отключить или включить эту проверку. Смысл этого параметра достаточно прост -- все, что поступает к нам, проходит проверку на соответствие исходящего адреса с нашей таблицей маршрутизации и такая проверка считается успешной, если принятый пакет предполагает передачу ответа через тот же самый интерфейс.

Следующий фрагмент включит проверку исходящего адреса для всех существующих интерфейсов:

```
# for i in /proc/sys/net/ipv4/conf/*/rp_filter ; do
>  echo 2 > $i
> done
```

Возвращаясь к примеру выше: пусть имеется маршрутизатор, построенный на базе Linux, который обслуживает две подсети -- *net1* и *net2*, причем *net1* подключена к интерфейсу eth0, а *net2* -- к интерфейсу eth1. Теперь, если на eth0, придет пакет с обратным адресом *net2*, то он будет сброшен. Аналогичным образом будет отвергнут пакет, с обратным адресом *net1*, пришедший на интерфейс eth1.

Это есть полная проверка обратного адреса. Но такая фильтрация возможна только на основе анализа IP-адресов сетей связанных напрямую маршрутизатором. Это потому, что полная фильтрация становится невозможной в случае асимметричной маршрутизации (когда пакеты приходят через один интерфейс, а ответный трафик через другой), например через спутник (или в случае динамической маршрутизации (bgp, ospf, rip) в вашей сети), когда данные поступают со спутниковой антенны, а ответы отправляются по обычной наземной линии связи.

Если у вас как раз такой случай (вы наверняка точно знаете об этом), то можете просто выключить `rp_filter` на интерфейсе, куда приходят данные со спутника. Если вам интересно узнать -- сбрасываются ли какие-нибудь пакеты, файл `log_martians`, в том же самом каталоге, сообщит ядру о необходимости журналирования таких пакетов.

```
# echo 1 >/proc/sys/net/ipv4/conf/<interfacename>/log_martians
```

FIXME: достаточно ли настроек в файлах `conf/[default,all]/*` ? - martijn

13.2. Малоизвестные настройки.

Итак, имеется целый ряд дополнительных настроек , которые вы можете изменить. Попробуем их перечислить. Кроме того, они частично описаны в файле `Documentation/ip-sysctl.txt`.

Некоторым из этих параметров присвоены такие значения по-умолчанию, которые напрямую зависят от того как вы сконфигурировали свое ядро.

Оскар Андриссон (Oskar Andreasson) написал руководство, в котором значительно лучше нас описал все эти настройки, так что не поленитесь -- загляните на http://ipsysctl-tutorial.frozentux.net/. (Русский перевод руководства "ipsysctl tutorial", о котором идет речь, вы найдете по адресу: http://gazette.linux.ru.net/rus/article/index-ipsysctl-tutorial.html, тарболл с исходными текстами документа (на русском языке) -- http://gazette.linux.ru.net/archive/ipsysctl-tutorial.1.0.4.tar.gz.

13.2.1. Параметры настройки IPv4.

Небольшое примечание: в большинстве своем, приводимые здесь временные параметры не воздействуют на локальный (loopback) интерфейс, так что вы не сможете проверить их, не имея реального сетевого интерфейса. Кроме того, указываемые пределы измеряются в 'тиках', и предполагают использование ранее упомянутого *Token Bucket Filter*.

Ядро имеет встроенные 'часы', которые 'тикают' определенное число раз в секунду. Для платформы Intel -- обычно 100 раз в секунду (в ядрах серии 2.6.x этот параметр можно изменить с помощью утилит конфигурирования. **прим. перев.**). Таким образом, число 50, в некоем параметре `*_rate`, будет означать '2 пакета в секунду'. *Token Bucket Filter* сконфигурирован так, чтобы иметь возможность превышения лимита не более чем на 6 пакетов.

Некоторые пункты, из приводимого ниже списка, мы просто скопировали из `/usr/src/linux/Documentation/networking/ip-sysctl.txt`, написанного Алексеем Кузнецовым <kuznet@ms2.inr.ac.ru> и Энди Клином (Andi Kleen) <ak@muc.de>.

От переводчика (А.К.): Я взял на себя смелость привести описание части параметров из *"ipsysctl tutorial"*, поскольку они более точные и полные.

/proc/sys/net/ipv4/icmp_destunreach_rate

> Если ядро решит, что оно не в состоянии доставить пакет, то пакет будет сброшен, а отправителю будет послано соответствующее ICMP-сообщение.

/proc/sys/net/ipv4/icmp_echo_ignore_all

> Параметр может принимать два значения -- 0 (выключено) и 1 (включено). Значение по-умолчанию -- 0 (выключено). Если записана 1, то ядро просто игнорирует все *ICMP Echo Request* запросы и тогда никто не сможет **ping**-ануть вашу машину, чтобы проверить ее наличие в сети, что само по себе не есть хорошо. Однако, у каждого из нас может быть свое собственное мнение по поводу этой опции. С одной стороны -- окружающие лишены возможности проверить ваше присутствие в сети, с другой -- существует масса приложений, которые используют *ICMP Echo Request* запросы далеко не в благовидных целях. Вобщем все как всегда -- что-то плохо, а что-то хорошо.

/proc/sys/net/ipv4/icmp_echo_ignore_broadcasts

Эта переменная очень близка по смыслу к `icmp_echo_ignore_all`, только в данном случае будут игнорироваться ICMP-сообщения, отправленные на широковещательный или групповой адрес. Вполне очевидно, почему полезно включить этот параметр -- защита от smurf атак.

/proc/sys/net/ipv4/icmp_echoreply_rate

Частота генерации эхо-ответов, посылаемых по любому адресу.

/proc/sys/net/ipv4/icmp_ignore_bogus_error_responses

Отдельные маршрутизаторы, вопреки стандартам, описанным в RFC 1122, отправляют фиктивные ответы в широковещательном диапазоне. Обычно эти ошибки заносятся в системный журнал. Если вы не желаете регистрировать их, то включите этот параметр и тем самым сбережете некоторый объем дискового пространства в своей системе. Параметр может принимать два значения -- 0 (выключено) и 1 (включено). Значение по-умолчанию -- 0 (выключено)

/proc/sys/net/ipv4/icmp_paramprob_rate

Относительно малоизвестное ICMP-сообщение, которое посылается в ответ на неправильные пакеты с искаженными IP или TCP заголовками. Этот параметр регулирует частоту генерации подобных сообщений.

/proc/sys/net/ipv4/icmp_timeexceed_rate

Ограничивает частоту генерации сообщений *ICMP Time Exceeded*.

От переводчика (А.К.): По поводу двух предыдущих параметров (`icmp_paramprob_rate` и `icmp_timeexceed_rate`) у меня есть большие сомнения в том, что они присутствуют в файловой системе `/proc`. Вероятно в более ранних версиях ядра (до 2.4) эти параметры присутствовали, но у себя я их не нашел, зато имеются два других параметра, которые обеспечивают данную функциональность.

icmp_ratelimit

Максимальная частота генерации ICMP-пакетов с типом, указанным в `icmp_ratemask` (см. ниже). Это значение задается в "тиках" и устанавливает временной интервал между ICMP-посылками. Таким образом, значение 0 означает отсутствие ограничений. Обычно 1 "тик" равен 0.01 секунды, так значение 1 в этой переменной ограничивает скорость передачи не более 100 посылок в секунду, а значение 100 -- не более 1 посылки в секунду. Значение по-умолчанию -- 100 (зависит от конфигурации ядра), что означает не более 1 ICMP посылки за интервал в 100 "тиков".

icmp_ratemask

Маска ICMP типов, на которые накладывается ограничение по частоте генерации переменной icmp_ratelimit. Каждый из ICMP типов маскируется своим битом.

`icmp_ratemask` -- это битовая маска, где каждый ICMP тип представлен своим битом. Соответствие между символическим названием ICMP типа и его порядковым номером вы найдете в заголовочном файле `netinet/ip_icmp.h` (обычно это `/usr/include/netinet/ip_icmp.h`). За дополнительной информацией обращайтесь к RFC 792 - Internet Control Message Protocol. Математически маска определяется так:

```
ratemask = SUM 2^n
```

где n принимает значения всех типов ICMP, которые должны быть ограничены.

Например: Ограничим передачу сообщений *ICMP Destination Unreachable*. В `/usr/include/netinet/ip_icmp.h` этому типу соответствует число 3. Подсчитаем значение 2**3, это будет число 8. Теперь нужно прибавить это число к имеющейся битовой маске. Допустим, что маска уже равна числу 6160 (в двоичном виде 0001100000010000), тогда результирующая маска получится: 6160 + 8 = 6168 (в двоичном виде 0001100000011000).

Перед добавлением маски подобным образом убедитесь сначала, что нужный вам бит сброшен, иначе вы получите неверную маску! К примеру, если у вас маска является числом 256 и вы еще добавите число 256, то это приведет к размаскированию *ICMP Echo Request* и маскировке 9-го, отсутствующего типа ICMP.

Значение по-умолчанию -- 6168 (в двоичном виде 0001100000011000), что подразумевает наложение ограничений на *ICMP Destination Unreachable*, *ICMP Source Quench*, *ICMP Time Exceeded* и *ICMP Parameter Problem*, где *ICMP Destination Unreachable* = 3, *ICMP Source Quench* = 4, *ICMP Time Exceeded* = 11 и *ICMP Parameter Problem* = 12. Таким образом, значение по-умолчанию соответствует выражению:

```
ratemask = 2^3 + 2^4 + 2^11 + 2^12
```

Злоумышленник может "заставить" некоторый маршрутизатор или хост "затопить" жертву ICMP-посылками, передаваемыми в ответ на поддельный ICMP-пакет с обратным адресом жертвы. Поэтому очень важно ограничить частоту генерации отдельных видов ICMP-сообщений.

На сайте http://www.frozentux.net вы найдете небольшую программу -- **ratemask**, которая может оказаться полезной при создании маски для переменной `icmp_ratemasks` или для дешифрации существующей маски.

/proc/sys/net/ipv4/igmp_max_memberships

Максимальное число групп на каждый сокет. Значение по-умолчанию -- 20 и может быть изменено по мере необходимости. FIXME: Это действительно так?

/proc/sys/net/ipv4/inet_peer_gc_maxtime

Параметр `inet_peer_gc_maxtime` определяет частоту "сборки мусора" при незначительном объеме данных. Эта переменная имеет то же предназначение, что и `inet_peer_gc_mintime` (см. ниже), только выбор, каким параметром пользоваться, производится в зависимости от величины нагрузки на систему. Значение параметра измеряется в так называемых "тиках" (jiffies), понятие "тика" описывалось выше.

Параметр принимает целое число, измеряемое в "тиках". Значение по-умолчанию -- 120 "тиков", чего вполне достаточно для большинства серверов и рабочих станций.

/proc/sys/net/ipv4/inet_peer_gc_mintime

Параметр `inet_peer_gc_mintime` устанавливает минимальное время между проходами "сборщика мусора" при значительном объеме данных, хранящихся в "inet peer storage". Когда система находится под тяжелыми нагрузками и появляется множество факторов,

ограничивающих размер пула памяти, то частота "сборки мусора" определяется этой переменной. Значение переменной измеряется в "тиках".

Параметр принимает целое число, измеряемое в "тиках". Значение по-умолчанию -- 10 "тиков", чего вполне достаточно для большинства серверов и рабочих станций.

/proc/sys/net/ipv4/inet_peer_maxttl

Это максимальное время хранения записей. При незначительных нагрузках на систему неиспользуемые записи будут удаляться через данный промежуток времени.

/proc/sys/net/ipv4/inet_peer_minttl

Параметр определяет минимальное время хранения данных в "inet peer storage". Это время должно быть достаточно большим, чтобы перекрыть время сборки фрагментов на противоположной стороне. Минимальное время хранения является гарантией того, что размер пула будет меньше чем величина `inet_peer_threshold`.

/proc/sys/net/ipv4/inet_peer_threshold

Здесь хранится приблизительный размер памяти для "inet peer storage". Когда размер пула достигает этой величины, то "сборщик мусора" переводится в более агрессивный режим работы -- с периодом прохода `inet_peer_gc_mintime`. Кроме того, этот порог влияет на срок хранения записей. Чем выше этот порог, тем больше срок хранения.

/proc/sys/net/ipv4/ip_autoconfig

Этот параметр содержит номер протокола, по которому получил сведения о конфигурации (RARP, BOOTP, DHCP или нечто подобное). Иначе -- ноль.

/proc/sys/net/ipv4/ip_default_ttl

Устанавливает значение по-умолчанию для величины *Time To Live* исходящих пакетов. Это число определяет продолжительность "жизни" пакета в Internet. Каждый раз, когда пакет попадает на очередной роутер (брандмауэр и т.п.), величина TTL пакета уменьшается на 1.

Значение по-умолчанию -- 64. Это достаточно приемлемое значение. Очень трудно представить себе ситуацию, когда это число оказалось бы недостаточным. Переменная принимает целое число без знака в диапазоне от 0 до 255 включительно, однако, значение 255 слишком велико, а если поставить 0 -- то вы вообще лишитесь выхода в сеть. Число 64 достаточно хорошее, если вы не пытаетесь установить соединение с компьютером, отстоящим от вас на значительном растоянии, измеряемом в "переходах" (hops). Тогда этой вличины TTL может и не хватить. Однако, на практике едва ли вам встретятся компьютеры, отстоящие от вас более чем на 30 "переходов" (hops).

Увеличение TTL пакета до 255 может иметь свои негативные последствия. Если представить себе, что где-то произошел сбой в 2-х маршрутизаторах и пакет "зациклился" между ними, тогда пакет начнет "бегать" туда-сюда между маршрутизаторами, поглощая пропускную способность канала, до тех пор, пока TTL пакета не истечет. Как правило, величину TTL не устанавливают больше 100.

/proc/sys/net/ipv4/ip_dynaddr

Параметр используется для разрешения некоторых проблем, связанных с динамической адресацией. Позволяет демону **diald** одновременно устанавливать соединение и изменять исходящий адрес в пакетах (и сокетах для локальных процессов). Эта возможность была

реализованв для поддержки TCP по коммутируемым соединениям и соединениям с маскарадингом (masqueradig). Эта опция позволяет "маскарадить" исходящий адрес пакета при изменении динамического IP-адреса.

В переменную можно записать одно из 3-х значений: 0, 1 или 2.

- 0 -- опция выключена, это значение по-умолчанию.
- 1 -- опция включена.
- Любое другое значение, отличающееся от 0 и 1 подразумевает включение этой опции в "многословном" (verbose) режиме, что приводит к записи в системный журнал отладочных сообщений.

Что происходит, если изменяется адрес исходящего интерфейса при включенной опции ip_dynaddr:

- Исходящий адрес пакета и сокета изменяется ПРИ ПОВТОРНОЙ ПЕРЕДАЧЕ, когда он находится в состоянии SYN_SENT. Это касается локальных процессов.
- Для транзитных пакетов -- исходящий адрес пакета изменяется на выходе, когда внутренний хост выполняет ПОВТОРНУЮ ПЕРЕДАЧУ, пока не будет принят внешний пакет.

Эта опция особенно полезна для случаев автодозвона, когда в момент установления связи исходящий адрес еще не известен. Любой запрос на соединение заставляет diald "поднять" исходящий интерфейс, после чего пакеты отправляются через него. Это означает, что нет необходимости сначала выдавать запрос который "поднимет" исходящий интерфейс, а затем выдавать "реальный" запрос на соединение, вместо этого мы просто устанавливаем соединение.

/proc/sys/net/ipv4/ip_forward

Включает/отключает функцию форвардинга (передачу транзитных пакетов между сетевыми интерфейсами), которая позволяет компьютеру выступать в роли брандмауэра или маршрутизатора. Эта переменная очень важна для Network Address Translation (Трансляция Сетевых Адресов), брандмауэров, маршрутизаторов и всего того, что должно передавать пакеты между сетями.

Это булева переменная. Или, другими словами, она может принимать два значения -- 0 или 1. Значение по-умолчанию -- 0, "запрещено". То есть 0 означает запрет форвардинга, а 1 -- разрешает его.

/proc/sys/net/ipv4/ip_local_port_range

Содержит два целых числа, которые определяют диапазон локальных портов, которые используются в клиентских соединениях, т.е. для исходящих соединений, которые связывают нашу систему с некоторым узлом сети, где мы выступаем в качестве клиента. Первое число задает нижнюю границу диапазона, второе -- верхнюю.

Значения по-умолчанию зависят от имеющегося объема ОЗУ. Если установлено более чем 128 Мб, то нижняя граница будет 32768, а верхняя -- 61000. При меньшем объеме ОЗУ нижняя граница будет 1024 а верхняя -- 4999 или даже меньше.

Этот диапазон определяет количество активных соединений, которые могут быть запущены одновременно, с другой системой, которая не поддерживает TCP-расширение timestamp.

Диапазона 1024-4999 вполне достаточно для установки до 2000 соединений в секунду с системами, не поддерживающими *timestamp*. Проще говоря, этого вполне достаточно для большинства применений.

/proc/sys/net/ipv4/ip_no_pmtu_disc

Параметр `ip_no_pmtu_disc` запрещает поиск PMTU (от англ. Path Maximum Transfer Unit -- Максимальный Размер Пакета для выбранного Пути). Для большинства случаев лучше установить в эту переменную значение FALSE, или 0 (т.е. система будет пытаться определить максимальный размер пакета, при котором не потребуется выполнять их фрагментацию, для передачи на заданный хост). Но иногда, в отдельных случаях, такое поведение системы может приводить к "срывам" соединений. Если у вас возникают такие проблемы, то вам следует попробовать отключить эту опцию (т.е. записать в переменную число 1) и установить нужное значение MTU.

Обратите внимание на то, что MTU и PMTU -- это не одно и то же! MTU -- (от англ. Maximum Transfer Unit -- максимальный размер пакета) определяет максимальный размер пакета для наших сетевых интерфейсов, но не для сетевых интерфейсов на другом конце. PMTU -- опция, которая заставляет систему вычислять максимальный размер пакета, при котором не потребуется фрагментация пакетов, для маршрута к заданному хосту, включая все промежуточные переходы.

Значение по-умолчанию -- FALSE (0), т.е. функция определения разрешена. Если записать число 1 в этот файл, то функция определения PMTU будет запрещена. Параметр `ip_no_pmtu_disc` может принимать значение 0 или 1.

/proc/sys/net/ipv4/ipfrag_high_thresh

Параметр задает максимальный объем памяти, выделяемый под очередь фрагментированных пакетов. Когда длина очереди достигает этого порога, то обработчик фрагментов будет отвергать все фрагментированные пакеты до тех пор, пока длина очереди не уменьшится до значения переменной `ipfrag_low_thresh`. Это означает, что все отвергнутые фрагментированные пакеты должны быть повторно переданы узлом-отправителем.

Пакеты фрагментируются в том случае, если их размер слишком велик, чтобы быть переданными по данному каналу. Узел-отправитель, в этом случае, "режет" пакеты на более мелкие части и затем передает их одну за другой. На узле-получателе эти фрагменты опять собираются в полноценный пакет. Примечательно, что идея фрагментации пакетов, сама по себе, вещь замечательная, но, к сожалению, может быть использована в весьма неблаговидных целях.

Параметр содержит целое число в диапазоне 0 .. 2147483647 и означает верхний порог длины очереди фрагментов в байтах. Значение по-умолчанию -- 262144 байт, или 256 Кб. Этого количества, как правило, вполне достаточно даже для самых крайних случаев.

/proc/sys/net/ipv4/ip_nonlocal_bind

Установка этого параметра позволяет отдельным локальным процессам выступать от имени внешнего (чужого) IP-адреса. Это может оказаться полезным в некоторых случаях, когда необходимо "прослушивать" внешние (чужие) IP-адреса, например -- сниффинг чужого траффика. Однако, эта опция может оказывать отрицательное влияние на работоспособность отдельных приложений.

Может иметь два значения -- 0 или 1. Если установлено значение 0, то опция отключена, 1 -- включена. Значение по-умолчанию -- 0.

/proc/sys/net/ipv4/ipfrag_low_thresh

Этот параметр очень тесно связан с переменной `ipfrag_high_thresh`. Он устанавливает нижний порог, при достижении которого опять разрешается прием фрагментов в очередь. Обработчик фрагментов имеет свою очередь, в которой находятся фрагментированные пакеты, ожидающие сборки. Когда длина очереди достигает верхнего порога (`ipfrag_high_thresh`), то прием фрагментов в очередь приостанавливается до тех пор, пока длина очереди не уменьшится до величины `ipfrag_low_thresh`. Это предохраняет систему от "затопления" фрагментированными пакетами и, тем самым, является, в своем роде, защитой от некоторых видов DoS-атак.

Целое число в диапазоне 0 .. 2147483647 и означает нижний порог длины очереди фрагментов в байтах, по достижении которого, фрагментированные пакеты снова будут приниматься в очередь. Значение по-умолчанию -- 196608 байт, или 192 Кб. Это число обязательно должно быть меньше, чем `ipfrag_high_thresh`.

/proc/sys/net/ipv4/ipfrag_time

Определяет максимальное время "хранения" фрагментов в секундах. Это относится только к тем фрагментам, которые пока невозможно собрать, поскольку собранные пакеты к этому сроку скорее всего уже будут переданы дальше (на следующий уровень или в сеть).

Принимает целое значение и определяет предельное время хранения фрагментов в секундах. Если записать туда число 5, то это будет означать 5 секунд.

/proc/sys/net/ipv4/tcp_abort_on_overflow

Заставляет ядро отвергать новые соединения, если их поступает такое количество, что система не в состоянии справиться с таким потоком. Что это означает? Допустим, что на систему обрушивается шквал запросов на соединение, тогда они могут быть просто отвергнуты, если эта опция будет включена, поскольку система не в состоянии обработать их все. Если не установлена, то система будет пытаться обслужить все запросы.

Может иметь два значение -- 0(выключено) или 1(включено). Значение по-умолчанию -- 0. Включение этой опции следует расценивать как крайнюю меру. Перед этим необходимо попытаться поднять производительность сервисов.

/proc/sys/net/ipv4/tcp_fin_timeout

Задает максимальное время пребывания сокета в состоянии FIN-WAIT-2. Используется в тех случаях, когда другая сторона по тем или иным причинам не закрыла соединение со своей стороны. Каждый сокет занимает в памяти порядка 1.5 Кб, что может привести к значительным утечкам памяти в некоторых случаях.

Принимает целое число. Значение по-умолчанию -- 60 секунд. В ядрах серии 2.2 это значение было равно 180 секундам, но было уменьшено, поскольку иногда возникали проблемы, связанные с нехваткой памяти, на web-серверах, которые, как правило, обслуживают огромное количество подключений.

За дополнительной информацией -- обращайтесь к описанию параметров `tcp_max_orphans` и `tcp_orphan_retries`.

/proc/sys/net/ipv4/tcp_keepalive_time

Определяет -- как часто следует проверять соединение, если оно давно не используется. Значение параметра имеет смысл только для тех сокетов, которые были созданы с флагом

SO_KEEPALIVE.

Принимает целое число секунд. Значение по-умолчанию -- 7200, т.е. 2 часа. Не уменьшайте это число без необходимости, поскольку это может привести к увеличению бесполезного трафика.

/proc/sys/net/ipv4/tcp_keepalive_intvl

Определяет интервал проверки "жизнеспособности" сокета. Это значение учитывается при подсчете времени, которое должно пройти перед тем как соединение будет разорвано.

Принимает целое число. Значение по-умолчанию -- 75 секунд. Это достаточно высокое значение, чтобы рассматривать его как нормальное. Значения параметров `tcp_keepalive_probes` и `tcp_keepalive_intvl` могут использоваться для определения времени, через которое соединение будет разорвано.

Со значениями по-умолчанию (9 попыток с интервалом 75 секунд) это займет примерно 11 минут. Попытки определения "жизнеспособности", в свою очередь, начнутся через 2 часа после того, как через данное соединение проследовал последний пакет.

/proc/sys/net/ipv4/tcp_keepalive_probes

Определяет количество попыток проверки "жизнеспособности" прежде, чем будет принято решении о разрыве соединения.

Принимает целое число, которое не следует устанавливать больше 50-ти. Значение по-умолчанию -- 9. Это означает, что будет выполнено 9 попыток проверки соединения, чтобы убедиться в том, что соединение разорвано.

/proc/sys/net/ipv4/tcp_max_orphans

Задает максимальное число "осиротевших" (не связанных ни с одним процессом) сокетов. Если это число будет превышено, то такие соединения разрываются, а в системный журнал пишется предупреждение.

Это ограничение существует исключительно ради предотвращения простейших разновидностей DoS-атак. Вообще вы не должны полагаться на эту переменную! Не рекомендуется уменьшать это число. Сетевая среда может потребовать увеличение этого порога, однако, такое увеличение может привести к необходимости увеличения объема ОЗУ в системе. Прежде чем поднимать этот предел -- попробуйте перенастроить сетевые сервисы на более агрессивное поведение по отношению к "осиротевшим" сокетам.

Принимает целое число. Значение по-умолчанию -- 8192, однако оно очень сильно зависит от объема памяти в системе. Каждый "осиротевший" сокет "съедает" примерно 64 Кб памяти, которая не может быть сброшена в своп (swap).

При возникновении проблем, связанных с этим ограничением -- в системный журнал будет записано сообщение, подобное этому:

```
TCP: too many of orphaned sockets
```

Это может служить поводом к тому, чтобы пересмотреть значения переменных `tcp_fin_timeout` или `tcp_orphans_retries`.

/proc/sys/net/ipv4/tcp_orphan_retries

Количество попыток закрыть соединение перед тем как оно будет разорвано принудительно. Если вы администрируете http-сервер, который испытывает большие нагрузки, то стоит подумать об уменьшении этого значения.

Принимает целое число. Значение по-умолчанию -- 7, что соответствует, примерно, от 50 секунд до 16 минут, в зависимости от внличины Retransmission Timeout (RTO -- Таймаут для Повторной Передачи. прим. перев.). Детальное описание RTO вы найдете в разделе "3.7. Data Communication" RFC 793 - Transmission Control Protocol.

Кроме того, посмотрите описание переменной `tcp_max_orphans`.

/proc/sys/net/ipv4/tcp_max_syn_backlog

Определяет максимальное время хранения SYN-запросов в памяти до момента получения третьего, завершающего установление соединения, пакета. Эта опция работает только тогда, когда включен параметр `tcp_syncookies`. Если сервер испытывает серьезные нагрузки, то можно попробовать немного увеличить этот параметр.

Значение по-умолчанию зависит от количества памяти, имеющейся в системе. Если объем памяти менее 128 Мб, то значение по-умолчанию равно 128, если больше, то значение по-умолчанию равно 1024.

Если вы увеличиваете эту переменную до величины более чем 1024, то было бы неплохо изменить величину TCP_SYNQ_HSIZE и пересобрать ядро. TCP_SYNQ_HSIZE находится в файле linux/include/tcp.h. Эта величина рассчитывается по формуле:

```
TCP_SYNQ_HSIZE*16 <= tcp_max_syn_backlog
```

/proc/sys/net/ipv4/tcp_max_tw_buckets

Максимальное число сокетов, находящихся в состоянии TIME-WAIT одновременно. При превышении этого порога -- "лишний" сокет разрушается и пишется сообщение в системный журнал. Цель этой переменной -- предотвращение простейших разновидностей DoS-атак.

Целое число. Значение по-умолчанию -- 180000. На первый взгляд может показаться, что это очень много, но на самом деле это не так. Если у вас начинают возникать ошибки, связанные с этим параметром, то попробуйте увеличить его.

Вам не следует уменьшать значение этой переменной. Вместо этого, если начали поступать сообщения в системный журнал, ее следует увеличить, однако, это может потребовать наращивания памяти в системе.

/proc/sys/net/ipv4/tcp_retrans_collapse

Включает/выключает эмуляцию ошибки протокола TCP, делая возможным сетевое взаимодействие с некоторыми устройствами, в которых реализация стека TCP имеет эту ошибку. Без ее эмуляции было бы невозможным работать с отдельными моделями принтеров. Ошибка заключается в отправке полноразмерных пакетов при повторной передаче.

Может принимать два значения -- 0 (выключено) и 1 (включено). Значение по-умолчанию -- 1 (включено). Эмуляция ошибки никак не повредит взаимодействию с другими узлами сети, но при этом позволит общаться с устройствами, имеющими ошибку в реализации стека TCP. Вообще эта опция совершенно безопасна, однако, если в журнал пишутся непонятные сообщения -- можете попробовать отключить ее.

/proc/sys/net/ipv4/tcp_retries1

Максимальное количество попыток повторной передачи пакетов по установленному соединению прежде, чем сообщение об ошибке будет передано сетевому уровню, в результате чего может быть выбран другой маршрут для отправки последующих пакетов. Минимальное значение этого параметра равно 3. Это число является значением по-умолчанию, что соответствует интервалу времени от 3 секунд до 8 минут, в зависимости от величины Retransmission timeout (RTO). Детальное описание RTO вы найдете в разделе "3.7. Data Communication" RFC 793 - Transmission Control Protocol.

Целое число. Значение по-умолчанию -- 3. Стандарты определяют диапазон изменения этого параметра от 3 до 100.

/proc/sys/net/ipv4/tcp_retries2

Максимальное количество попыток повторной передачи пакетов, до того как соединение будет считаться разорванным. Это ограничение определено в RFC 1122 и равно 100, но обычно его уменьшают.

Значение по-умолчанию -- 15, что соответствует примерно 13-30 минутам в зависимости от величины Retransmission timeout (RTO).

/proc/sys/net/ipv4/tcp_rfc1337

Является реализацией решения проблемы, описываемой в "RFC 1337 - TIME-WAIT Assassination Hazards in TCP". Проблема связана с "устаревшими" дубликатами пакетов, которые могут вносить помехи во вновь устанавливаемые соединения и порождать три различные проблемы. Первая -- "устаревший" дубликат пакета с данными может быть ошибочно воспринят в новом соединении, что приведет к передаче неверных данных. Вторая -- соединение может быть десинхронизировано и "уйти" в ACK-цикл из-за "устаревших" дубликатов, которые порождают новые соединения (здесь автор имеет ввиду "устаревшие" дубликаты SYN-пакетов, прим. перев.). И третья проблема -- "устаревшие" дубликаты могут "проникнуть" в недавно созданное соединение и ошибочно уничтожить его.

Согласно упомянутому RFC существуют три возможных решения, однако, одно из них решает эту проблему лишь частично, другие требуют внесения значительных изменений в протокол TCP.

Окончательное решение состоит в том, что RST-пакеты должны просто игнорироваться, пока сокет находится в состоянии TIME_WAIT. Вместе с установкой параметра Maximum Segment Life (MSL -- максимальное время жизни сегмента) равным 2 мин. такой подход решает все три проблемы, описанные в RFC 1337.

/proc/sys/net/ipv4/tcp_sack

Разрешает Selective Acknowledgements (SACK -- Выборочное Подтверждение), детальное описание вы найдете в RFC 2883 - An Extension to Selective Acknowledgement (SACK) Option for TCP и RFC 2883 - An Extension to Selective Acknowledgement (SACK) Option for TCP.

Если этот параметр включен (1), то в TCP-заголовке будет устанавливаться SACK-флаг при передаче SYN-пакета, сообщая тем самым удаленному хосту, что наша система в состоянии обрабатывать SACK, на что удаленный хост может ответить ACK-пакетом с установленным флагом SACK. Этот режим выборочно подтверждает каждый сегмент в TCP-окне. Это особенно полезно на неустойчивых соединениях, поскольку позволяет производить повторную передачу лишь отдельных, не подтвержденных фрагментов, а не всего TCP-окна, как это диктуется более старыми стандартами. Если какой либо сегмент TCP-окна был

"утерян", то приемная сторона не пришлет на него SACK-подтверждение о приеме. Отправитель, поняв это, повторит передачу "потерявшихся" сегментов. Избыточные данные сохраняются в TCP-заголовке, 40 байт на сегмент. Подтверждение каждого сегмента -- это два 32-битных беззнаковых целых числа, таким образом в заголовке может разместиться подтверждение 4-х сегментов. Однако, как правило, совместно с опцией SACK используется опция timestamp, которая занимает 10 байт и поэтому в одном пакете может быть подтверждено не более 3 сегментов.

Рекомендуется включать эту опцию, если вы имеете неустойчивые соединения. Однако, если вы соединены 1.5-метровым кабелем с другой машиной, то в таком случае, для достижения наивысшей скорости обмена, следует эту опцию отключить. Обычно эта опция не нужна, но лучше ее включить. Она обеспечивает 100% обратную совместимость, т.е. вы не должны испытывать никаких проблем при соединении с хостами, которые эту опцию не поддерживают.

Значение по-умолчанию --1 (включено).

/proc/sys/net/ipv4/tcp_stdurg

Разрешает/запрещает соответствие стандарту RFC 1122. Поведение по-умолчанию соответствует стандарту использования флага URG -- BSD 4.2, описанному в RFC 793. Если этот параметр включен, то возможны сбои при работе с отдельными узлами Интернета, точнее -- с узлами, которые придерживаются стандарта BSD 4.2. Значение по-умолчанию -- 0 (выключено).

/proc/sys/net/ipv4/tcp_syn_retries

Количество попыток передачи SYN-пакета при установлении нового соединения. Это число не должно устанавливаться больше чем 255, поскольку каждая повторная попытка отнимает значительное время. На каждую попытку отводится примерно 30-40 секунд. Значение по-умолчанию -- 5, что соответствует, примерно, 180 секундам.

/proc/sys/net/ipv4/tcp_synack_retries

Количество попыток передачи SYN,ACK-пакета в ответ на SYN-запрос. Другими словами -- максимальное число попыток установить пассивное TCP-соединение, инициированное другим хостом. Это число не должно устанавливаться больше чем 255. Значение по-умолчанию -- 5.

/proc/sys/net/ipv4/tcp_timestamps

Разрешает/запрещает использование временных меток (timestamps), в соответствии с RFC 1323. Если коротко, то это расширение TCP используется для расчета Round Trip Measurement (определение времени возврата) лучшим способом, нежели метод Retransmission timeout (RTO). Эта опция должна сохранять обратную совместимость в большинстве случаев, так что лучше оставить ее включенной, особенно если вы работаете в высокоскоростной сети (например LAN или 10mb Интернет). В случае низкоскоростного оединения (скажем модемное) -- вы прекрасно обойдетесь и без этой опции, и будет даже лучше, если вы ее отключите. Значение по-умолчанию -- 1 (включено).

/proc/sys/net/ipv4/tcp_tw_recycle

Разрешает/запрещает быструю утилизацию сокетов, находящихся в состоянии TIME-WAIT. Если вы не уверены в своих действиях, то вам лучше не трогать эту переменную. Значение по-умолчанию -- 0.

/proc/sys/net/ipv4/tcp_window_scaling

Разрешает/запрещает масштабирование TCP-окна, как определено в RFC 1323. В этом документе описано как производится масштабирование TCP-окна при передаче по Large Fat Pipes (LFP -- "толстый" канал). При передаче TCP-пакетов по "толстым" каналам возникают существенные потери пропускной способности из-за того, что они не загружены полностью во время ожидания подтверждения о приеме предыдущего TCP-окна. Основная проблема состоит в том, что окно не может иметь размер больше, чем 2**16 байт (65 Кб). Разрешая масштабирование TCP-окна мы, тем самым, можем увеличить его размер и таким образом уменьшить потери пропускной способности. Значение по-умолчанию -- 1 (включено).

13.2.2. Параметры настройки устройств.

Каталог `conf/DEV/`, где под DEV следует понимать название того или иного устройства, содержит настройки для конкретного сетевого интерфейса. Настройки в каталоге `conf/all/` применяются ко ВСЕМ сетевым интерфейсам. Настройки в каталоге `conf/default/` -- настройки по-умолчанию, они не влияют на настройки существующих интерфейсов, но используются для первоначальной настройки вновь устанавливаемых устройств.

/proc/sys/net/ipv4/conf/DEV/accept_redirects

Управляет приемом ICMP-сообщений о переадресации. Сообщения *ICMP Redirect ...* используются для уведомления маршрутизаторов или хостов о существовании лучшего маршрута движения пакетов к заданному хосту, который (маршрут) может быть быстрее или менее загружен.

Может иметь два значения -- 0 (выключено -- сообщения о переадресации игнорируются) и 1 (включено -- сообщения о переадресации принимаются). Значение по-умолчанию -- 1 (включено), однако я посоветовал бы отключать эту опцию, поскольку она далеко небезопасна. В подавляющем большинстве случаев необходимость в переадресации отсутствует, поэтому лучше держать эту переменную выключенной, если конечно вы на 100% не уверены в обратном.

/proc/sys/net/ipv4/conf/DEV/accept_source_route

Разрешает/запрещает "маршрутизацию от источника". Маршрутизация от источника весьма небезопасна. Значение по-умолчанию -- 1 (включено).

/proc/sys/net/ipv4/conf/DEV/bootp_relay

Разрешает/запрещает форвардинг пакетов с исходящими адресами 0.b.c.d. Демон BOOTP relay должен перенаправлять эти пакеты на корректный адрес. Значение по-умолчанию -- 0 (выключено), поскольку реализация обработки этого параметра еще отсутствует (kernel v2.2.12).

/proc/sys/net/ipv4/conf/DEV/forwarding

Включает/отключает функцию форвардинга (передачу транзитных пакетов) между сетевыми интерфейсами. Могут использоваться для включения/выключения функции форвардинга для отдельных интерфейсов. По-умолчанию все параметры conf/DEV/forwarding принимают значение, установленное в `ipv4/ip_forward` так, если этот параметр включить, то и все параметры conf/DEV/forwarding будут включены, если выключить, то и conf/DEV/forwarding окажутся выключены.

/proc/sys/net/ipv4/conf/DEV/log_martians

Включает/выключает функцию журналирования всех пакетов, которые содержат неправильные (невозможные) адреса (так называемые martians -- "марсианские" пакеты). Под невозможными адресами, в данном случае, следует понимать такие адреса, которые отсутствуют в таблице маршрутизации. (см. раздел Reverse Path Filtering).

/proc/sys/net/ipv4/conf/DEV/mc_forwarding

Включает/выключает поддержку маршрутизации групповых рассылок для заданного интерфейса. Кроме того, чтобы иметь поддержку маршрутизации групповых рассылок, необходимо собрать ядро с включенной опцией CONFIG_MROUTE. Дополнительно в системе должен иметься демон, осуществляющий групповую маршрутизацию. Значение по-умолчанию -- 0 (выключено). Обратите внимание -- нет никакой необходимости включать эту опцию, если вы желаете лишь получать групповые пакеты. Она необходима только если вы собираетесь перенаправлять групповой трафик через вашу систему.

/proc/sys/net/ipv4/conf/DEV/proxy_arp

Включает/выключает проксирование arp-запросов для заданного интерфейса. ARP-прокси позволяет маршрутизатору отвечать на ARP запросы в одну сеть, в то время как запрашиваемый хост находится в другой сети. С помощью этого средства происходит "обман" отправителя, который отправил ARP запрос, после чего он "думает", что маршрутизатор является хостом назначения, тогда как в действительности хост назначения находится по другую сторону маршрутизатора. Маршрутизатор выступает в роли уполномоченного агента хоста назначения, перекладывая пакеты от другого хоста. Значение по-умолчанию -- 0 (выключено). Дополнительную информацию вы найдете в Proxy-ARP mini HOWTO.

/proc/sys/net/ipv4/conf/DEV/rp_filter

См. раздел Reverse Path Filtering

/proc/sys/net/ipv4/conf/DEV/secure_redirects

Включает/выключает режим безопасной переадресации. Если параметр выключен, то будут приниматься любые сообщения ICMP Redirect ... от любого хоста из любого места. Если включен, то сообщения о переадресации будут восприниматься только от тех шлюзов (gateways), которые имеются в списке шлюзов по-умолчанию. С помощью этой опции можно избежать большинства ложных переадресаций, которые могут быть использованы для перехвата трафика. Значение по-умолчанию -- 1 (включено). Обратите внимание -- действие этой параметра отменяется параметром `shared_media`, так что, если вы включаете `secure_redirects`, то необходимо включить и `shared_media`.

/proc/sys/net/ipv4/conf/DEV/send_redirects

Включает/выключает выдачу ICMP Redirect ... другим хостам. Эта опция обязательно должна быть включена, если хост выступает в роли маршрутизатора любого рода. Как правило ICMP-сообщения о переадресации отправляются в том случае, когда необходимо сообщить хосту о том, что он должен вступить в контакт с другим сервером. Значение по-умолчанию -- 1 (включено). Если компьютер не выступает в роли маршрутизатора, то этот параметр можно отключить.

/proc/sys/net/ipv4/conf/DEV/shared_media

Включает/выключает признак того, что физическая сеть является носителем нескольких

логических подсетей, например, когда на одном физическом кабеле организовано несколько подсетей с различными сетевыми масками. Этот признак используется ядром при принятии решения о необходимости выдачи ICMP-сообщений о переадресации. Значение по-умолчанию -- 0 (выключено). Этот параметр влияет на установку параметра `secure_redirects`.

/proc/sys/net/ipv4/conf/DEV/tag

FIXME: Заполните этот пробел.

13.2.3. Параметры сетевых политик.

Каталог `neigh/DEV/`, где под DEV следует понимать название того или иного устройства, содержит настройки для конкретного сетевого интерфейса. Настройки в каталоге `neigh/all/` применяются ко ВСЕМ сетевым интерфейсам. Настройки в каталоге `neigh/default/` -- настройки по-умолчанию, они не влияют на настройки существующих интерфейсов, но используются для первоначальной настройки вновь устанавливаемых устройств.

/proc/sys/net/ipv4/neigh/DEV/anycast_delay

Максимальное значение случайной задержки ответов в "тиках" (1/100 секунды). Пока не реализовано (Linux пока еще не имеет поддержки anycast).

/proc/sys/net/ipv4/neigh/DEV/app_solicit

Определяет количество запросов, посылаемых демону ARP пользовательского уровня. 0 -- отключено.

/proc/sys/net/ipv4/neigh/DEV/base_reachable_time

Базовое значение, используемое для расчета случайного времени доступа, в соответствии с RFC2461.

/proc/sys/net/ipv4/neigh/DEV/delay_first_probe_time

Задержка перед первой попыткой проверки доступности "соседей". (см. `gc_stale_time`)

/proc/sys/net/ipv4/neigh/DEV/gc_stale_time

Определяет частоту проверки записей в таблице ARP на предмет "устаревания". Если запись "устарела", то она должна быть подтверждена при первой же возможности (бывает полезным в случае динамического распределения IP-адресов). В случае, если параметр `ucast_solicit` имеет значение больше 0, то сначала выполняется попытка послать ARP-пакет напрямую известному хосту, а в случае неудачи, когда `mcast_solicit` больше 0, выполняется широковещательный ARP-запрос.

/proc/sys/net/ipv4/neigh/DEV/locktime

Записи в таблице ARP обновляются только в том случае, если "возраст" записи превышает этот параметр. Предотвращает слишком частую "продувку" кэша ARP.

/proc/sys/net/ipv4/neigh/DEV/mcast_solicit

Максимальное количество повторов передачи широковещательного запроса к маршрутизаторам.

/proc/sys/net/ipv4/neigh/DEV/proxy_delay

Максимальное время (выбирается случайно из диапазона [0 .. proxytime]) задержки перед отправкой ответа на ARP-запрос, для которого имеется запись в proxy ARP. В некоторых случаях может использоваться для предотвращения сетевых атак.

/proc/sys/net/ipv4/neigh/DEV/proxy_qlen

Максимальная длина очереди для задержанных ARP-откликов (см. `proxy_delay`).

/proc/sys/net/ipv4/neigh/DEV/retrans_time

Время, измеряемое в "тиках", между повторными передачами запросов к "соседям". Используемых для определения адресов.

/proc/sys/net/ipv4/neigh/DEV/ucast_solicit

Максимальное количество повторов передачи уникастного запроса.

/proc/sys/net/ipv4/neigh/DEV/unres_qlen

Максимальная длина очереди для ожидающих arp-запросов - количество пакетов, которые приняты от протоколов других уровней, ожидающие разрешения адреса.

13.2.4. Параметры маршрутизации.

/proc/sys/net/ipv4/route/error_burst и /proc/sys/net/ipv4/route/error_cost

`error_burst` используется в паре с `error_cost` для ограничения количества генерируемых сообщений *ICMP Destination Unreachable*. `error_burst` несет в себе смысл верхнего предела "стоимости" передачи сообщений, в то время как `error_cost` обозначает "цену" одного сообщения. Когда `error_burst` "опустошается", то передача сообщений *ICMP Destination Unreachable* прекращается.

Сообщения *ICMP Destination Unreachable* обычно отсылаются тогда, когда невозможно определить дальнейший маршрут движения пакета. Этому могут быть три причины:

1. Невозможно выполнить передачу хосту.
2. Не известен маршрут к заданному сегменту сети или хосту.
3. Если данный маршрут запрещен набором правил маршрутизации.

В этих трех случаях сетевая подсистема генерирует сообщение ICMP Destination Unreachable, свое для каждого случая:

1. *ICMP Host Unreachabl*e -- когда хост, находящийся в той же сети, что и наш роутер -- недоступен.

2. *ICMP Network Unreachable* -- когда в таблице маршрутизации роутера нет ни одного маршрута, по которому пакет мог бы быть отправлен дальше.

3. *ICMP Communication Administratively Prohibited By Filtering* -- когда пакет не может быть переправлен из-за наличия правил маршрутизации явно запрещающих передачу.

Значение по-умолчанию -- 500. Учитывая значение по-умолчанию переменной `error_cost` (100) это соответствует 5-ти сообщениям *ICMP Destination Unreachable* в секунду.

/proc/sys/net/ipv4/route/flush

Запись любой информации в эту переменную (само собой разумеется, запись может быть произведена только, если вы обладаете правами root) приведет к очистке кэша маршрутов.

/proc/sys/net/ipv4/route/gc_elasticity

Значения, управляющие частотой и поведением алгоритма сборки "мусора" в кэше маршрутизации. Может оказаться полезным при восстановлении после сбоев. Прежде чем Linux сможет перейти к другому маршруту, пройдет не менее `gc_timeout` секунд (по-умолчанию -- 300), если использовавшийся ранее оказался недоступным. Это число можно немного уменьшить, чтобы ускорить восстановление после сбоев.

См. также это сообщение, которое отправил Ard van Breemen.

/proc/sys/net/ipv4/route/gc_interval

См. описание `/proc/sys/net/ipv4/route/gc_elasticity`.

/proc/sys/net/ipv4/route/gc_min_interval

См. описание `/proc/sys/net/ipv4/route/gc_elasticity`.

/proc/sys/net/ipv4/route/gc_thresh

См. описание `/proc/sys/net/ipv4/route/gc_elasticity`.

/proc/sys/net/ipv4/route/gc_timeout

См. описание `/proc/sys/net/ipv4/route/gc_elasticity`.

/proc/sys/net/ipv4/route/max_delay

Максимальный размер задержки перед сбросом кэша маршрутов.

/proc/sys/net/ipv4/route/max_size

Максимальный размер кэша маршрутов. Устаревшие записи будут удаляться из кэша только после достижения размера кэша этой величины

/proc/sys/net/ipv4/route/min_adv_mss

FIXME: Восполните этот пробел.

/proc/sys/net/ipv4/route/min_delay

Минимальный размер задержки перед сбросом кэша маршрутов.

/proc/sys/net/ipv4/route/min_pmtu

FIXME: Восполните этот пробел.

/proc/sys/net/ipv4/route/mtu_expires

FIXME: Восполните этот пробел.

/proc/sys/net/ipv4/route/redirect_load

Факторы, влияющие на принятие решения о пересылке сообщений перенаправления на заданный хост. Перенаправление не производится в случае, если величина нагрузки или количество отправленных сообщений достигли своего предела.

/proc/sys/net/ipv4/route/redirect_number

См. описание `/proc/sys/net/ipv4/route/redirect_load`

/proc/sys/net/ipv4/route/redirect_silence

Таймаут перенаправления. По истечении этого периода времени, сообщение о переадресации посылается снова, независимо от того были ли достигнуты пределы `redirect_load` или `redirect_number`.

Глава 14. Специализированные дисциплины управления очередями.

Если вы придете к выводу, что ранее упомянутые дисциплины организации очередей вас не устраивают, то ядро может предложить вам ряд более специализированных дисциплин.

14.1. bfifo/pfifo

Эти бесклассовые дисциплины еще более просты, чем `pfifo_fast` -- они не имеют внутренних полос, все виды трафика в них равноправны. Единственное их преимущество -- возможность получения некоторых статистик. Таким образом, если вы не собираетесь ограничивать трафик или раскидывать его по приоритетам, то использование этой дисциплины позволит вам получить дополнительную информацию, которая поможет выявить узкие места на интерфейсе.

Длина `pfifo` измеряется в пакетах, `bfifo` -- в байтах.

14.1.1. Параметры и порядок использования.

limit

> Задает длину очереди. Для `pfifo` измеряется в пакетах, для `bfifo` -- в байтах. По-умолчанию имеет значение: для `pfifo` -- *`txqueuelen`* интерфейса (см. раздел pfifo_fast), в пакетах, и *`txqueuelen * mtu`* байт -- для `bfifo`.

14.2. Алгоритм Кларка-Шенкера-Чанга.

Этот алгоритм настолько "заумный", что даже Алексей (главный автор CBQ) утверждает -- будто бы не до конца понял его суть. С его слов:

> David D. Clark, Scott Shenker и Lixia Zhang *Поддержка Приложений Реального Времени в Пакетных Сетях с Интеграцией Сервисов: Архитектура и Механизм*.
>
> Насколько я понял, основная идея заключается в создании WFQ-потоков для каждого из сервисов с гарантированным качеством обслуживания и привязка оставшейся пропускной способности к фиктивному потоку *flow-0*. В поток *flow-0* отправляется весь предиктивный трафик и трафик, который обслуживается по принципу "лучшее из оставшегося" (best effort). Планировщик потока в первую очередь пропускает предиктивный трафик, а оставшаяся пропускная способность отдается трафику "best effort".
>
> Примечательно, что в CSZ-потоках (Clark-Shenker-Zhang) НЕ производится наложения ограничений пропускной способности. Предполагается, что поток уже прошел входной контроль QoS сети и не нуждается в дополнительном формировании. Любая попытка что-то улучшить или ограничить, на промежуточных переходах, может привести к нежелательным задержкам и увеличению нестабильности.
>
> На сегодняшний день CSZ - единственный планировщик, который может дать настоящую гарантию качества обслуживания. Другие схемы (включая CBQ) не гарантируют величину задержки.
>
> В настоящее время не может быть рекомендован к использованию, если вы не читали или не достаточно четко понимаете содержимое упомянутого документа.

14.3. DSMARK.

Esteve Camps

<marvin@grn.es>

Этот текст -- отрывки из моих тезисов *Поддержка QoS в Linux*, сентябрь 2000 года.

Исходные документы:

- Draft-almesberger-wajhak-diffserv-linux-01.txt

- Примеры, прилагаемые к дистрибутиву **iproute2**
- White Paper-QoS protocols and architectures и IP QoS Frequently Asked Questions.

Автор главы: Esteve Camps <esteve@hades.udg.es>.

14.3.1. Введение.

Прежде всего, было бы неплохо, если бы вы предварительно ознакомились с RFC, посвященными данной теме (RFC2474, RFC2475, RFC2597 и RFC2598) по адресам: IETF DiffServ working Group и домашняя страничка проекта diffserv.

14.3.2. Что такое Dsmark и с чем его "едят"?

Dsmark -- это дисциплина организации очереди, которая предлагает возможности, необходимые в "Differentiated Services" (известной также, как DiffServ, или просто -- DS). DiffServ -- фактически одна из двух архитектур QoS (вторая называется "Integrated Services"), которая базируется на значении поля DS в заголовке IP-пакетов.

Одним из первых решений в IP, которое предлагало некоторый уровень QoS, был "Type of Service" (Тип Обслуживания) -- поле TOS в IP-заголовке. Изменяя это поле, можно было выбрать высокую/низкую пропускную способность, минимальную задержку или высокую надежность. Но это решение не обеспечивало достаточной гибкости, которую требовали вновь появляющиеся услуги (например, приложения реального времени, интерактивные приложения и т.п.). С появлением новых требований, появились и новые архитектуры. Одна из них -- DiffServ, которая подменяет первые шесть битов ToS в пакете IPv4 или октет "класс трафика" в пакете IPv6, полем, с названием DS, в котором можно указать до 64 классов трафика.

14.3.3. Основные принципы.

Differentiated Services (Дифференцированное Обслуживание) ориентирован на группы. Имеется ввиду, что эта технология ничего не знает о потоках, она ориентирована на группы, а применяемые правила зависят от того, к какой группе направляется пакет.

Сеть маршрутизаторов с поддержкой механизмов DiffServ называют "облаком DiffServ" (или "доменом DiffServ"). Классификация, формирование и установка меток (под установкой меток понимается установка значений в поле DS) происходит на входе в "облако". Внутри домена метка определяет -- какой уровень QoS должен применяться к трафику внутренними маршрутизаторами сети.

Самым большим преимуществом модели DiffServ является то, что она действует на границе "облака". После того как данные пересекли границу, внутренним маршрутизаторам можно не заниматься поддержанием информации о статусе QoS и полностью сосредоточиться на своей основной функции -- маршрутизации.

Фактически, внутри своих локальных доменов, вы можете диктовать любую политику обслуживания, но при соединении с другими DS-доменами вы должны следовать *Соглашению об*

Уровне Обслуживания (SLA).

К этому моменту у вас наверняка возникла масса вопросов. Diffsrv -- это много больше, чем я смог сказать. Вы должны понять, что я не в состоянии в 50 строках изложить содержимое трех RFC. :-)

14.3.4. Как работать с Dsmark.

Как уже было определено выше, в случае с DiffServ, пограничные и внутренние узлы различаются между собой. Это два важных пункта на пути трафика. Оба типа узлов выполняют классификацию трафика. Результат классификации может использоваться для различной DS-обработки, прежде чем пакет уйдет в сеть. Код diffserv представляет пакет в виде структуры `sk_buff`, в которой имеется поле `skb->tc_index`. В данном поле сохраняется результат начальной классификации, который может использоваться для различной интерпретации DS на пограничных и внутренних маршрутизаторах.

Значение `skb->tc_index` изначально устанавливается дисциплиной DSMARK qdisc для каждого входящего пакета, в соответствии с полем DS в IP-заголовке. Кроме того, классификатор cls_tcindex считывает, целиком или частично, значение skb->tcindex и использует его для выбора нужного класса.

Для начала рассмотрим команду DSMARK qdisc и ее параметры:

```
... dsmark indices INDICES [ default_index DEFAULT_INDEX ] [ set_tc_index ]
```

Каково назначение этих параметров?

- ***indices:*** размер таблицы пар маска-значение. Максимальное значение 2^n, где n >= 0.
- ***Default_index:*** индекс в таблице, принимаемый по-умолчанию, если классификатор не находит ни одного совпадения.
- ***Set_tc_index:*** инструкция, которая считывает значение поля DS и записывает его в skb->tc_index.

14.3.5. Как работает SCH_DSMARK.

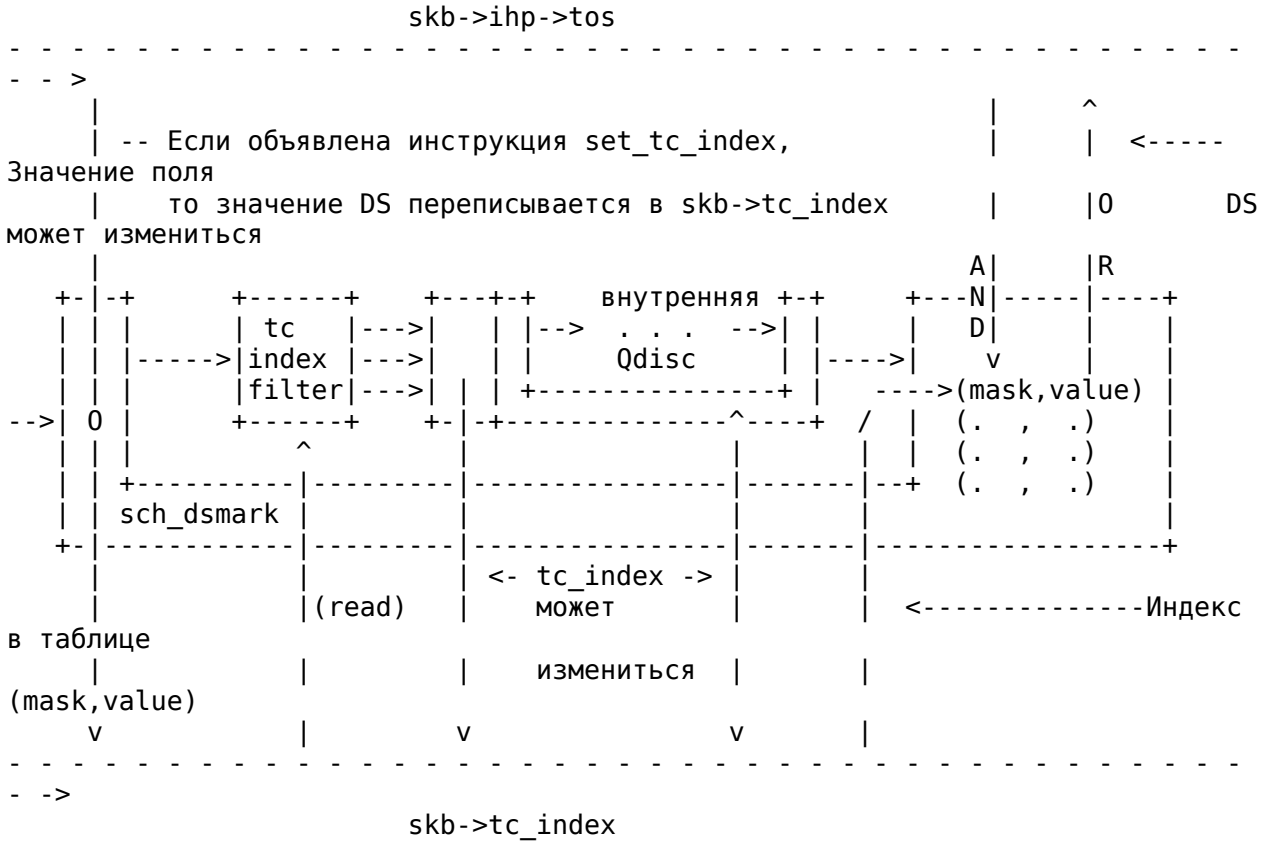
Эта дисциплина выполняет следующие шаги:

- Если вставлена инструкция set_tc_index, то считывается поле DS и сохраняется в skb->tc_index.
- Вызывается классификатор. Он возвращает идентификатор класса, который будет сохранен в skb->tc_index. Если такой класс не найден, то используется класс по-умолчанию из параметра default_index. Если ни set_tc_index, ни default_index не объявлены, то результат может оказаться непредсказуемым.
- После этого управление передается внутренней qdisc, где вы можете повторно использовать результаты фильтрации. Идентификатор класса, возвращаемый внутренней qdisc, запоминается в skb->tc_index. Это значение будет использоваться в качестве индекса

таблицы маска-значение. Конечный результат, который будет связан с пакетом, получается из выражения:

```
New_Ds_field = ( Old_DS_field & mask ) | value
```

- Таким образом, конечный результат получается в результате объединения по "И" ds_field и маски, и затем объединения по "ИЛИ" с параметром value. Следующая диаграмма иллюстрирует этот процесс:



Как установить метку? Просто измените mask и value класса. См. следующий код:

```
tc class change dev eth0 classid 1:1 dsmark mask 0x3 value 0xb8
```

Это изменение пары (mask,value) в хеш-таблице, пометит пакеты, принадлежащие классу 1:1.

Теперь перейдем к описанию фильтра TC_INDEX. Кроме всего прочего, фильтр TC_INDEX может использоваться и в других конфигурациях, а не только в тех, которые включают DS услуги.

14.3.6. Фильтр TC_INDEX.

Базовый синтаксис команды, объявляющей фильтр TC_INDEX:

```
... tcindex [ hash SIZE ] [ mask MASK ] [ shift SHIFT ]
            [ pass_on | fall_through ]
            [ classid CLASSID ] [ police POLICE_SPEC ]
```

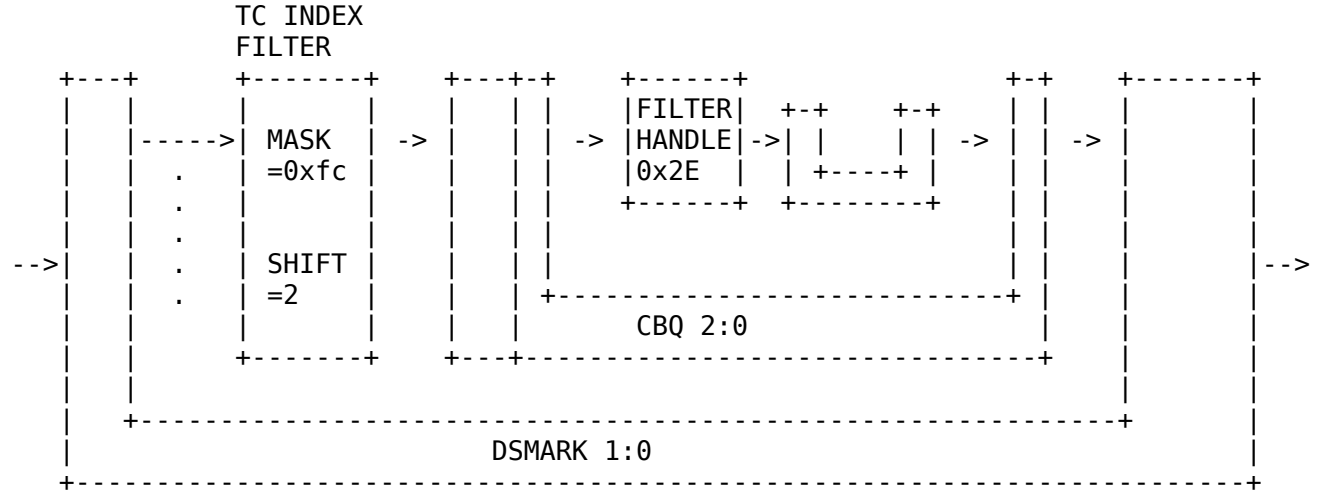
Ниже приводится пример, который описывает работу TC_INDEX (обратите внимание на места,

выделенные жирным шрифтом:

```
tc qdisc add dev eth0 handle 1:0 root dsmark indices 64 set_tc_index

tc filter add dev eth0 parent 1:0 protocol ip prio 1 tcindex mask 0xfc  shift 2

tc qdisc add dev eth0 parent 1:0 handle 2:0 cbq bandwidth 10Mbit cell 8 avpkt 1000
mpu 64

# EF traffic class

tc class add dev eth0 parent 2:0 classid 2:1 cbq bandwidth 10Mbit rate 1500Kbit
avpkt 1000 prio 1 bounded isolated allot 1514 weight 1 maxburst 10

# Packet fifo qdisc for EF traffic

tc qdisc add dev eth0 parent 2:1 pfifo limit 5

tc filter add dev eth0 parent 2:0 protocol ip prio 1 handle 0x2e tcindex classid 2:1
pass_on
```

(Это неполный код, я просто привел часть примера EFCBQ, включенного в состав дистрибутива **iproute2**).

Будем исходить из предположения, что мы получаем пакет, помеченный как EF. Если вы прочитаете RFC2598, то увидите, что рекомендуемое значение DSCP для EF трафика -- 101110. Это означает, что в поле DS будет записано 10111000 (не забывайте, что младшие биты в поле TOS не используются в DS), или 0xb8, в шестнадцатиричном представлении.



Полученный пакет имеет значение 0xb8 в поле DS. Дисциплина с идентификатором 1:0 считывает это значение и помещает его в skb->tc_index. На следующем шаге (вторая строка в примере), описанный фильтр выполняет следующие действия:

```
Value1 = skb->tc_index & MASK
Key = Value1 >> SHIFT
```

В нашем примере MASK=0xFC и SHIFT=2.

```
Value1 = 10111000 & 11111100 = 10111000
Key = 10111000 >> 2 = 00101110 -> 0x2E (в шестнадцатиричном виде)
```

Возвращаемое значение будет соответствовать фильтру внутренней qdisc (в примере,

идентификатор 2:0). Если фильтр с заданным идентификатором найден, то условия фильтра будут проверены (в случае, если фильтр включает в себя эти условия) и будет возвращен classid (в нашем примере classid 2:1), который далее будет записан в skb->tc_index. Если фильтр не будет найден, то результат будет зависеть от объявления флага fall_through. Если объявление fall_through присутствует, то его значение будет воспринято как classid. В противном случае продолжится просмотр остальных фильтров. Будьте предельно внимательны, при использовании флага fall_through -- его использование рекомендуется только в том случае, когда значение skb->tc_index и идентификаторы классов ссвязаны простыми (в смысле несложными) отношениями.

И последние два параметра, которые мы опишем, это *hash* и *pass_on*. Первый из них определяет размер хеш-таблицы. *Pass_on* -- означает, что если не будет найден classid, равный результату этого фильтра, то необходимо попробовать применить следующий фильтр. Действие по-умолчанию -- fall_through (см. следующую таблицу).

В заключение посмотрим -- какие значения параметров TCINDEX допустимы:

TC Name	Value	Default
Hash	1...0x10000	Зависит от реализации
Mask	0...0xffff	0xffff
Shift	0...15	0
Fall through / Pass_on	Flag	Fall_through
Classid	Major:minor	None
Police	.....	None

Это очень мощный тип фильтров. Кроме того, он может использоваться не только в конфигурации DiffServ, но и как любой другой тип фильтров.

Я настоятельно рекомендую вам внимательно просмотреть все примеры DiffServ, включаемые в дистрибутив **iproute2**. Со своей стороны я обещаю, что дополню этот текст, как только найду время. Все, что я здесь описал -- есть результат длительных экспериментов. Я буду весьма признателен, если вы сообщите мне об обнаруженных ошибках.

14.4. Ingress qdisc.

До сих пор, все дисциплины, которые обсуждались, были исходящими (egress) дисциплинами. Однако, каждый из интерфейсов может иметь и входящие (ingress) дисциплины. Они не применяются к исходящему трафику, но они позволяют устанавливать tc-фильтры для входящих пакетов, вне зависимости от того, предназначены ли они для данного хоста или должны быть перенаправлены дальше.

Поскольку tc-фильтры имеют полную реализацию Token Bucket Filter, то они так же могут ограничивать входящий трафик на основе "функций оценки" (estimators). Это позволяет выстраивать эффективную политику обслуживания входящего трафика, до того как он попадет "в руки" стека протоколов IP.

14.4.1. Параметры и порядок использования.

Входные дисциплины, сами по себе не требуют каких-либо параметров. Пример создания входной дисциплины:

```
# tc qdisc add dev eth0 ingress
```

Кроме входящей дисциплины вы можете добавлять к устройству и исходящие дисциплины.

Примеры использования входящей дисциплины вы найдете в главе Решебник.

14.5. Random Early Detection (RED)

Этот раздел служит в качестве введения в организацию очередей в магистральных сетях, которые зачастую имеют полосу пропускания более 100 мегабит, что требует иного подхода чем в случае с домашним ADSL-модемом.

Нередко на маршрутизаторах в Интернет возникает так называемая проблема *tail drops* -- отсечения конца очереди. Когда очередь полна, ни один вновь поступивший пакет туда уже не помещается, а потому отбрасывается. Такое управление очередью приводит к повторной синхронизации параметров соединения. После синхронизации TCP сразу посылает столько пакетов, сколько допускает размер окна подтверждения. Подобный всплеск нагрузки опять приводит к отсечению конца очереди, что опять порождает необходимость повторной синхронизации... Такое хождение по кругу может продолжаться довольно долго.

Чтобы избежать возникновения заторов, на маршрутизаторах зачастую организуются очереди большого размера. К сожалению, не смотря на то, что увеличение размеров очереди благоприятно сказывается на пропускной способности, большие очереди могут приводить к увеличению времени задержки, что становится причиной взрывоподобного поведения TCP-соединений.

Проблема отсечения хвоста очереди с каждым днем становится все более неприятной. Для предотвращения перегрузок, ядро Linux предоставляет в наше распоряжение механизм RED, сокращенно от Random Early Detect (Случайное Раннее Обнаружение), которое иногда называется как Random Early Drop (Случайное Раннее Отсечение), последнее определение более точно описывает принцип работы.

RED -- это не панацея от всех бед, но позволяет более "справедливо" разделить канал между TCP-соединениями.

Он позволяет контролировать нагрузку с помощью выборочного случайного уничтожения некоторых пакетов до того, как очередь будет заполнена полностью, что заставляет протоколы, подобные TCP, снижать скорость передачи и предотвращает повторную синхронизацию. Кроме того, выборочная "потеря" пакетов помогает TCP быстрее найти подходящую скорость передачи данных, а так же удерживать размер очереди и время задержки на разумном уровне. Вероятность "потери" пакета конкретного соединения прямо пропорциональна пропускной способности, используемой этим соединением, а не числу пакетов, т.е. большие пакеты уничтожаются чаще маленьких, что дает достаточно справедливое распределение полосы пропускания.

RED хорошо подходит для обслуживания очередей на магистральных линиях, где отслеживание сессий (с целью справедливого распределения канала) является непозволительной роскошью.

При работе с RED вы должны будете определиться со значениями трех параметров: *Минимум (min), Максимум (max)* и *Превышение (burst)*. *Минимум* -- это минимальный размер очереди в байтах, выше которого начнется выборочная потеря пакетов. *Максимум* -- это "мягкий" максимум, алгоритм будет пытаться удержать размер очереди ниже этого предела. *Превышение* -- максимальное число пакетов, которые могут быть приняты в очередь сверх установленного максимального предела.

Минимальный размер очереди рассчитывается, исходя из максимально допустимого времени задержки в очереди и пропускной способности канала. Например, на моем 64Кбит/сек (8 Кбайт/сек) соединении я хочу получить максимальное время задержки 200 мсек, тогда 8 * 0.2 = 1.6 Кбайт (т.е. примерно 1600 байт). Если установить минимальный предел слишком маленьким, это приведет к снижению пропускной способности, слишком большим -- к увеличению времени задержки. Уменьшение размера очереди не может служить заменой уменьшению размера MTU, которое используется для уменьшения времени отклика на медленных линиях связи.

Максимальный размер очереди нужно задавать по меньшей мере в два раза большим минимального, чтобы снизить вероятность повторной синхронизации. На медленных линиях, с небольшим минимальным пределом размера очереди, максимальный предел следует задавать в четыре, а иногда и более раз больше минимального.

Предел превышения отвечает за поведение RED на пиковых нагрузках. Размер превышения должен устанавливаться больше, чем min/avpkt. Экспериментальным путем я пришел к выражению, устанавливающему размер превышения, (min+min+max)/(3*avpkt), которое дает неплохие результаты.

Кроме того, вам необходимо будет определиться с предельным размером очереди (limit) и средним размером пакета (avpkt). По достижении очередью предельного размера, RED переходит к алгоритму "отсечения конца". Обычно я устанавливаю предельный размер очереди в 8 раз больше максимального. Значение 1000, для avpkt (средний размер пакета), дает неплохие результаты на высокоскоростных линиях, при размере MTU = 1500.

Техническое описание RED (авторы: Sally Floyd и Van Jacobson) вы найдете в документе the paper on RED queueing.

14.6. Generic Random Early Detection.

Этот алгоритм известен не так широко. Он напоминает RED с несколькими внутренними очередями, распределение пакетов по очередям производится на базе поля tcindex Diffserv.

Каждая из виртуальных очередей может иметь собственные параметры конфигурации.

FIXME: убедительная просьба к Джамалу (Jamal) и Вернеру (Werner) дополнить этот раздел

14.7. Эмуляция VC/ATM.

Возможность строить Виртуальные Соединения через сокеты TCP/IP, появилась целиком и полностью благодаря усилиям Вернера Альмсбергера (Werner Almesberger). Виртуальные соединения -- это концепция, пришедшая из теории построения сетей ATM.

Дополнительную информацию по этой теме вы найдете по адресу: http://linux-atm.sourceforge.net/.

14.8. Weighted Round Robin (WRR).

Эта дисциплина организации очередей не включена в состав ядра Linux, однако вы можете скачать все необходимое по адресу: http://wipl-wrr.dkik.dk/wrr/. Она была протестирована только на ядрах серии 2.2, но возможно будет работать и на ядрах серий 2.4/2.5

WRR qdisc распределяет пропускную способность по классам, используя схему взвешенного циклического обхода. Т.е., подобно CBQ qdisc, она содержит классы, которые могут включать в себя любые другие дисциплины. Все классы получают ширину канала, пропорциональную присвоенным им весам. Весовые коэффициенты могут быть установлены как вручную, с помощью утилиты **tc**, так и автоматически, в этом случае величина весового коэффициента устанавливается обратно пропорциональной объему передаваемых через класс данных.

Дисциплина имеет внутренний классификатор, который распределяет пакеты, идущие на/из разные узлы сети, по разным классам. Определение отправителя/получателя может производиться на основе MAC или IP адреса. Однако, MAC адреса могут использоваться только в ethernet сетях. Привязка хостов к классам происходит автоматически, при появлении первых пакетов, прошедших по соединению.

Она неплохо зарекомендовала себя при обслуживании небольших локальных сетей, например в студенческих общежитиях, когда между несколькими индивидами разделяется единственное соединение с Интернет. Центральной частью дистрибутива WRR является набор сценариев, выполняющих настройку алгоритма для таких сетей.

Глава 15. Решебник.

В этой главе приводятся решения типовых задач, которые смогут помочь вам в преодолении некоторых проблем. Она не дает универсальных рецептов на все случаи жизни, но тем не менее, изучение чужого опыта никогда не бывает лишним.

15.1. Запуск нескольких сайтов с различными SLA.

***От переводчика (А.К.)**: SLA (от англ. Service Level Agreement) означает "Соглашение об Уровне Обслуживания" -- основной документ, регламентирующий взаимоотношения между ИТ-компанией и заказчиком.*

Сделать это можно несколькими способами. Прежде всего следует упомянуть, что **Apache** поддерживает подобную функциональность в виде модулей, но мы продемонстрируем как добиться этого средствами операционной системы. Эти строки взяты из примера, представленного Джамалом Хади (Jamal Hadi).

Допустим, что у нас есть два клиента, которые арендуют некоторую долю нашего канала под http, ftp и потоковое audio. Первый клиент арендует полосу в 2 Мбита, второй -- 5 Мбит. Для начала назначим нашим клиентам виртуальные IP-адреса на нашем сервере:

```
# ip address add 188.177.166.1 dev eth0
```

```
# ip address add 188.177.166.2 dev eth0
```

Решение проблемы о том, как назначить правильные IP-адреса различным службам, оставляем за вами. Практически все популярные демоны поддерживают такую возможность.

Присоединяем CBQ qdisc к eth0:

```
# tc qdisc add dev eth0 root handle 1: cbq bandwidth 10Mbit cell 8 avpkt 1000 \
  mpu 64
```

Создаем классы клиентов:

```
# tc class add dev eth0 parent 1:0 classid 1:1 cbq bandwidth 10Mbit rate \
  2MBit avpkt 1000 prio 5 bounded isolated allot 1514 weight 1 maxburst 21
# tc class add dev eth0 parent 1:0 classid 1:2 cbq bandwidth 10Mbit rate \
  5Mbit avpkt 1000 prio 5 bounded isolated allot 1514 weight 1 maxburst 21
```

И добавляем фильтры к классам:

```
##FIXME: Для чего нужна первая строка, что она делает? Каково назначение "делителя"
(divisor)?:
##FIXME: Делитель имеет отношение к хеш-таблице и номеру пула
##       (bucket) - ahu
# tc filter add dev eth0 parent 1:0 protocol ip prio 5 handle 1: u32 divisor 1
# tc filter add dev eth0 parent 1:0 prio 5 u32 match ip src 188.177.166.1
  flowid 1:1
# tc filter add dev eth0 parent 1:0 prio 5 u32 match ip src 188.177.166.2
  flowid 1:2
```

FIXME: Почему нет token bucket filter?

15.2. Защита от SYN flood.

Пример взят из документации к **iproute**, написанной Алексеем и адаптирован для совместной работы с **netfilter**. Если этот пример заинтересует вас, измените числовые значения на наиболее подходящие для вашей системы.

Этот сценарий был написан для защиты отдельного хоста, а не сети. Учитывайте это обстоятельство.

Для его работы желательно иметь самую последнюю версию **iproute2**.

```
#! /bin/sh -x
#
# демонстрация возможностей по управлению входящим (ingress) трафиком
# здесь приводится пример ограничения пропускной способности для входящих SYN-
пакетов
# Может оказаться полезным для защиты от TCP-SYN атак.
#
#пути к различным утилитам;
#укажите правильные значения.
#
TC=/sbin/tc
IP=/sbin/ip
```

```
IPTABLES=/sbin/iptables
INDEV=eth2
#
# пометить все SYN-пакеты, пришедшие через $INDEV, числом 1
############################################################
$iptables -A PREROUTING -i $INDEV -t mangle -p tcp --syn \
  -j MARK --set-mark 1
############################################################
#
# установить ingress qdisc на входящий интерфейс
############################################################
$TC qdisc add dev $INDEV handle ffff: ingress
############################################################

#
#
# SYN-пакет имеет размер 40 байт (320 бит), отсюда -- три пакета
# имеют размер 960 бит (примерно 1 Кбит); ограничим скорость поступления
# 3-мя пакетами в секунду ( точнее -- 1 Кбит/сек )
############################################################
$TC filter add dev $INDEV parent ffff: protocol ip prio 50 handle 1 fw \
police rate 1kbit burst 40 mtu 9k drop flowid :1
############################################################


#
echo "---- qdisc parameters Ingress  ----------"
$TC qdisc ls dev $INDEV
echo "---- Class parameters Ingress  ----------"
$TC class ls dev $INDEV
echo "---- filter parameters Ingress ----------"
$TC filter ls dev $INDEV parent ffff:

#Удаление ingress qdisc
#$TC qdisc del $INDEV ingress
```

15.3. Ограничение пропускной способности для ICMP-пакетов, с целью предотвращения dDoS атак.

Недавние распределенные атаки, типа "Отказ в обслуживании", стали основной "головной болью" для Интернет. Настроив соответствующую фильтрацию вы сможете предотвратить наступление катастрофических последствий, вызванных такого рода атаками.

Основная задача -- настроить фильтры таким образом, чтобы пакеты, с исходящими адресами, не принадлежащими вашей сети, не смогли бы покинуть ее. Это предотвратит возможность отправки всякой "гадости" в Интернет.

Прежде, чем приступить к делу, нарисуем схему подключения локальной сети к Интернет:

```
[Интернет] ---<E3, T3...>--- [Linux router] --- [Офис]
                           eth1          eth0
```

Зададим начальные условия:

```
# tc qdisc add dev eth0 root handle 10: cbq bandwidth 10Mbit avpkt 1000
# tc class add dev eth0 parent 10:0 classid 10:1 cbq bandwidth 10Mbit rate \
  10Mbit allot 1514 prio 5 maxburst 20 avpkt 1000
```

Если у вас более высокоскоростное подключение -- измените эти цифры соответствующим образом. Теперь необходимо определиться с "шириной" канала для ICMP-трафика. Чтобы найти типовое значение для вашей сети, можно воспользоваться утилитой **tcpdump**, запустив ее с перенаправлением вывода в файл. Затем, с помощью этого файла, вы сможете подсчитать количество ICMP-пакетов, отправляемых вашей сетью в единицу времени.

Если вариант подсчета экспериментальным путем вам не подходит, попробуйте ограничиться 10% общей пропускной способности. Построим наш класс:

```
# tc class add dev eth0 parent 10:1 classid 10:100 cbq bandwidth 10Mbit rate \
  100Kbit allot 1514 weight 800Kbit prio 5 maxburst 20 avpkt 250 \
  bounded
```

Он ограничивает пропускную способность канала величиной 100 Кбит/сек. А теперь подключим к нему фильтр для ICMP-пакетов:

```
# tc filter add dev eth0 parent 10:0 protocol ip prio 100 u32 match ip
  protocol 1 0xFF flowid 10:100
```

15.4. Управление приоритетами для трафика различных типов.

Если канал практически полностью забит отправляемыми/получаемыми данными, то работа через **telnet** или **ssh** становится практически невозможной. Как было бы здорово, если бы интерактивный трафик не блокировался другими пакетами. Linux поможет вам в этом!

Как и прежде, необходимо настроить обслуживание трафика на обоих концах канала. Наилучший вариант -- когда с обоих концов установлена операционная система Linux, однако UNIX тоже может выполнять управление приоритетами трафика.

Стандартный планировщик *pfifo_fast* имеет три различных "полосы". В первую очередь обслуживается полоса 0, а затем полосы 1 и 2. Поэтому, необходимо весь интерактивный трафик отправить в полосу 0!

Отталкиваясь от "Ipchais HOWTO" (уже довольно устаревшего):

В IP-заголовке имеется 4 редко используемых бита -- TOS (Type of Service -- Тип Обслуживания). Они задают способ обслуживания пакета: "Minimum Delay" (минимальная задержка), "Maximum Throughput" (максимальная пропускная способность), "Maximum Reliability" (максимальная надежность) и "Minimum Cost" (минимальная стоимость канала). Причем одновременно может быть установлен только один из этих битов. Роб ван Ньюкерк (Rob van Nieuwkerk), автор кода ipchains TOS-mangling, дает следующее пояснение:

> Наиболее важным для меня, является флаг "Minimum Delay" (минимальная задержка). Я включаю его в пакетах интерактивного трафика на моем роутере, работающем под управлением Linux. Я подключен к сети через модем 33.6. Linux "раскидывает" пакеты по 3-

м очередям. Таким образом я получаю вполне приемлемую скорость обслуживания интерактивного трафика при большой загрузке канала.

Как правило, флаг "Minimum Delay" устанавливается в пакетах для **telnet** и **ftp-control**, а в пакетах **ftp-data** -- "Maximum Throughput". Делается это следующим образом:

```
# iptables -A PREROUTING -t mangle -p tcp --sport telnet \
  -j TOS --set-tos Minimize-Delay
# iptables -A PREROUTING -t mangle -p tcp --sport ftp \
  -j TOS --set-tos Minimize-Delay
# iptables -A PREROUTING -t mangle -p tcp --sport ftp-data \
  -j TOS --set-tos Maximize-Throughput
```

Эти правила прописываются на удаленном хосте и воздействуют на входящие, по отношению к вашему компьютеру, пакеты. Для пакетов, отправляемых в обратном направлении, эти флаги (вроде бы) устанавливаются автоматически. Если это не так, можете на своей системе прописать следующие правила:

```
# iptables -A OUTPUT -t mangle -p tcp --dport telnet \
  -j TOS --set-tos Minimize-Delay
# iptables -A OUTPUT -t mangle -p tcp --dport ftp \
  -j TOS --set-tos Minimize-Delay
# iptables -A OUTPUT -t mangle -p tcp --dport ftp-data \
  -j TOS --set-tos Maximize-Throughput
```

15.5. Прозрачное проксирование с помощью netfilter, iproute2, ipchains и squid.

Этот раздел написал Рэм Нарул (Ram Narula), из "Internet for Education" (Таиланд).

Прозрачное проксирование -- это обычное перенаправление серверу **squid** трафика, отправляемого на порт 80 (web).

Существует 3 общеизвестных способа такого перенаправления:

Шлюз.

Вы можете настроить шлюз таким образом, что он будет перенапрвлять все пакеты, адресованные на порт 80, сереверу **squid**

НО

Это увеличит нагрузку на шлюз. Кроме того, некоторые коммерческие роутеры не поддерживают такую возможность.

Layer 4 switch

Свичи выполняют такое перенаправление без особых проблем.

НО

Стоимость этого оборудования очень высока и может быть сопоставима с ценой связки

"обычный роутер" + "Linux-сервер"

Использовать кэш-сервер в качестве шлюза.

Вы можете отправить ВЕСЬ трафик через кэш-сервер.

НО

Это довольно рисковано, поскольку **squid** довольно значительно нагружает CPU, что может привести к снижению производительности сети. Кроме того, **squid** может "рухнуть" и тогда никто из сети не сможет выйти в Интернет.

Мы предлагаем 4-й вариант:

Linux+NetFilter.

Эта методика предполагает установку меток на пакеты, с портом назначения равным числу 80, и дальнейшая маршрутизация помеченных пакетов, с помощью **iproute2**, на **squid**.

```
|----------------|
|   Реализация   |
|----------------|

 Используемые адреса
 10.0.0.1 naret (NetFilter)
 10.0.0.2 silom (Squid)
 10.0.0.3 donmuang (Router, соединенный с Интернет)
 10.0.0.4 kaosarn (некий сервер в сети)
 10.0.0.5 RAS
 10.0.0.0/24 main network
 10.0.0.0/19 total network

|---------------|
|Структура сети |
|---------------|

Internet
|
donmuang
|
------------hub/switch----------
|        |             |       |
naret   silom        kaosarn  RAS etc.
```

Прежде всего -- весь трафик должен идти через *naret*, для этого на всех компьютерах пропишем его в качестве шлюза по-умолчанию. Исключение составляет *silom*, для которого шлюз по-умолчанию -- *donmuang* (в противном случае web-трафик зациклится).

(на всех компьютерах в моей сети был прописан шлюз по-умолчанию -- 10.0.0.1, который ранее принадлежал *donmuang*, поэтому я изменил IP-адрес у *donmuang* на 10.0.0.3, а серверу *naret* присвоил адрес 10.0.0.1)

```
Silom
-----
-настройка squid и ipchains
```

Настройте прокси-сервер **squid** на *silom*. Убедитесь, что он поддерживает прозрачное проксирование. Обычно прокси-серверу, для приема запросов от пользователей, назначают порт 3128, поэтому весь трафик, отправляемый на порт 80 будет перенаправлен на порт 3128. С

помощью **ipchains** это делают следующие правила:

```
silom# ipchains -N allow1
silom# ipchains -A allow1 -p TCP -s 10.0.0.0/19 -d 0/0 80 -j REDIRECT 3128
silom# ipchains -I input -j allow1
```

iptables:

```
silom# iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 80 -j REDIRECT --to-port
3128
```

За информацией по настройке сервера **squid**, обращайтесь на http://squid.nlanr.net/.

Убедитесь, что на этом сервере разрешен форвардинг, и заданный по умолчанию шлюз для него -- *donmuang* (НЕ *naret*!).

```
Naret
-----
-настройка iptables и iproute2
-запрет ICMP-сообщений о перенаправлении (если необходимо)
```

1. Пометить пакеты, с портом назначения 80, числовой меткой 2.

   ```
   naret# iptables -A PREROUTING -i eth0 -t mangle -p tcp --dport 80 \
    -j MARK --set-mark 2
   ```

2. Настроить маршрутизацию так, чтобы помеченные пакеты отправлялись на *silom*

   ```
   naret# echo 202 www.out >> /etc/iproute2/rt_tables
   naret# ip rule add fwmark 2 table www.out
   naret# ip route add default via 10.0.0.2 dev eth0 table www.out
   naret# ip route flush cache
   ```

   Если *donmuang* и *naret* находятся в одной подсети, то *naret* не должен выдавать ICMP-сообщения о перенаправлении. Запрет можно наложить так:

   ```
   naret# echo 0 > /proc/sys/net/ipv4/conf/all/send_redirects
   naret# echo 0 > /proc/sys/net/ipv4/conf/default/send_redirects
   naret# echo 0 > /proc/sys/net/ipv4/conf/eth0/send_redirects
   ```

На этом настройку можно считать завершенной. Проверим конфигурацию

```
Для naret:

naret# iptables -t mangle -L
Chain PREROUTING (policy ACCEPT)
target     prot opt source               destination
MARK       tcp  --  anywhere             anywhere           tcp dpt:www MARK set 0x2

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination

naret# ip rule ls
0:      from all lookup local
32765:  from all fwmark        2 lookup www.out
32766:  from all lookup main
```

```
32767:  from all lookup default

naret# ip route list table www.out
default via 203.114.224.8 dev eth0

naret# ip route
10.0.0.1 dev eth0  scope link
10.0.0.0/24 dev eth0  proto kernel  scope link  src 10.0.0.1
127.0.0.0/8 dev lo  scope link
default via 10.0.0.3 dev eth0

|-------|
|-КОНЕЦ-|
|-------|
```

15.5.1. Схема движения пакетов после настройки.

```
|----------------------------------------|
|         Схема движения пакетов         |
|----------------------------------------|

ИНТЕРНЕТ
/\
||
\/
---------------------donmuang-------------------------
/\                                    /\         ||
||                                    ||         ||
||                                    \/         ||
naret                                silom      ||
*destination port 80 ================>(cache)    ||
/\                                    ||         ||
||                                    \/         \/
\\==================================kaosarn, RAS, и пр.
```

Обратите внимание, в данном случае сеть получилась асимметричной, т.е. добавился один лишний переход для исходящих пакетов.

Ниже приводится путь, проделываемый пакетами в/из Интернет для компьютера *kaosarn*.

Для web/http трафика:

```
kaosarn http request->naret->silom->donmuang->internet
http replies from Internet->donmuang->silom->kaosarn
```

Для прочего трафика:

```
kaosarn outgoing data->naret->donmuang->internet
incoming data from Internet->donmuang->kaosarn
```

15.6. Решение проблемы с Path MTU Discovery путем настройки MTU.

Общеизвестно, что скорость передачи данных возрастает с увеличением размера пакета. Это вполне естественно, ведь для каждого пакета в потоке принимается решение о выборе маршрута. К примеру, файл, размером в 1 мегабайт, будет передан быстрее, если он будет разбит на 700 пакетов (при максимальном размере пакета), а не на 4000.

Однако, не все сегменты Интернет могут передавать пакеты, с максимальной полезной нагрузкой в 1460 байт. Поэтому необходимо найти такой размер, который будет максимально возможным для заданного маршрута.

Процесс поиска такого размера называется 'Path MTU Discovery' (Поиск Максимального Размера Пакета для выбранного Пути), где MTU означает 'Maximum Transfer Unit' (Максимальный Размер Блока передачи данных).

Когда на маршрутизатор поступает пакет, который не может быть передан по выбранному маршруту целиком (без фрагментации) И у него установлен флаг "Don't Fragment" (Не фрагментировать), то в ответ отправляется ICMP-сообщение о том, что пакет был сброшен из-за невозможности "протолкнуть" его по выбранному маршруту. Компьютер, выполнивший посылку, начинает последовательно уменьшать размер пакетов до тех пор, пока они не смогут быть переданы по выбранному маршруту.

Все бы ничего, если бы не появились злоумышленники, которые задались целью нарушить нормальную работу Сети. Это, в свою очередь, вынуждает администраторов ограничивать или вообще блокировать ICMP-трафик с целью повысить отказоустойчивость вверенного им фрагмента сети.

В результате таких действий процесс поиска оптимального размера пакета работает все хуже и хуже, а на некоторых маршрутизаторах вообще не работает, что в свою очередь порождает сеансы TCP/IP с весьма странным поведением, которые "умирают" спустя некоторое время.

Хотя у меня нет никаких доказательств, но я знаю по крайней мере два сайта, с которыми наблюдается подобная проблема и перед обоими стоит Alteon Acedirectors -- возможно кто-то имеет более богатый опыт и сможет подсказать как и почему это происходит.

15.6.1. Решение

Если вы столкнетесь с подобной проблемой, то можно посоветовать отключить 'Path MTU Discovery' и установить MTU вручную. Koos van den Hout пишет:

> У меня возникла следующая проблема: для арендованного мною канала, работающего через ppp, на скорости 33.6К, я установил величину MTU/MRU, равную 296. Это дает мне достаточно приемлемое время отклика.
>
> Со моей стороны установлен роутер (с маскарадингом), работающий под управлением Linux.
>
> Недавно я вынес маршрутизатор на отдельный компьютер, так что теперь большая часть приложений выполняется на отдельной машине.
>
> После этого возникла масса проблем с авторизацией в irc. В процессе длительных поисков я

установил, что само соединение проходит и даже показывается системой как 'connected', но я не получал motd от irc (motd -- от англ. Message of The Day, которое демонстрируется после успешной авторизации, *прим. перев.*). Кроме того, помятуя о проблеме, связанной с MTU, я определил, что проблема исчезала только в том случае, когда MTU устанавливался равным 296. А так как серверы irc блокируют весь трафик, который напрямую не связан с их работой, то в числе блокируемых оказался и ICMP.

Мне удалось убедить администратора WEB-сервера в истинных причинах возникновения проблем, но администратор IRC-сервера отказался устранять их.

Таким образом, передо мной встала необходимость уменьшить MTU для внешнего трафика и оставить нормальное значение для локального.

Решение:

```
ip route add default via 10.0.0.1 mtu 296
```

(10.0.0.1 -- шлюз по-умолчанию, внутренний адрес маршрутизатора с маскарадингом)

Вообще, можно запретить 'PMTU Discovery' путем настройки специфических маршрутов. Например, если проблемы наблюдаются только с определенной подсетью, то это должно помочь:

```
ip route add 195.96.96.0/24 via 10.0.0.1 mtu 1000
```

15.7. Решение проблемы с Path MTU Discovery путем настройки MSS.

Как уже говорилось выше, Path MTU Discovery не работает в Интернет должным образом. Если вам известны факты существования сегментов в вашей сети, где размер MTU ограничен, то вы уже не можете полагаться на безотказную работу Path MTU Discovery.

Однако, помимо MTU, есть еще один способ ограничения размера пакета -- это, так называемый MSS (Maximum Segment Size -- Максимальный Размер Сегмента). MSS -- это поле в заголовке TCP-пакета SYN.

С недавних пор, ядра Linux и некоторые драйверы PPPoE, стали поддерживать такую особенность, как 'clamp the MSS' (ограничение размера MSS).

В этом есть свои плюсы и минусы. С одной стороны, устанавливая MSS, вы недвусмысленно извещаете удаленную сторону о том, что размер пакета не должен превышать заданную величину. Причем для этого не требуется передачи ICMP-сообщений.

С другой стороны -- за атакущим сохраняется возможность нарушить связь путем модификации пакетов. Однако, следует заметить, что мы довольно часто используем эту возможность и это приносит свои положительные плоды.

Чтобы иметь возможность манипулировать размером сегмента, у вас должны быть установлены **iptables**, не ниже 1.2.1a и ядро Linux, не ниже 2.4.3. Основная команда **iptables**:

```
# iptables -A FORWARD -p tcp --tcp-flags SYN,RST SYN -j TCPMSS  --clamp-mss-to-pmtu
```

Она рассчитает правильный MSS для вашего соединения. Если вы достаточно уверены в себе и в своих знаниях, можете попробовать нечто подобное:

```
# iptables -A FORWARD -p tcp --tcp-flags SYN,RST SYN -j TCPMSS --set-mss 128
```

Это правило устанавливает MSS равным 128. Очень полезно, если вы наблюдаете разрывы при передаче голосовых данных, когда поток небольших пакетов VoIP прерывается "огромными" http-пакетами.

15.8. Формирователь трафика: Низкая задержка, максимальная производительность.

Этот сценарий претерпел существенные изменения. Ранее он предназначался только для Linux-клиентов в вашей сети! Теперь же он может влиять на Windows и Mac машины

При разработке сценария преследовались следующие цели:

Обеспечить низкую задержку для интерактивного трафика.

> Это означает, что перекачка больших объемов данных не должна отрицательно сказываться на работе через **ssh** или **telnet**. Это очень важно, поскольку в период интерактивного взаимодействия с удаленной системой даже незначительные задержки, скажем в 200 мсек, вызывают у пользователя чувство раздражения.

Обеспечить приемлемую скорость web-серфинга

> Даже учитывая тот факт, что http-трафик сам по себе является довольно объемным, он не должен подвергаться значительным задержкам.

Обеспечить достаточно высокую скорость передачи больших объемов данных.

> Довольно часто возникает ситуация, когда исходящий трафик практически полностью блокирует входящий.

Оказывается, что все поставленные цели вполне достижимы. Основная причина, по которой перекачка значительных объемов вызывает задержки интерактивного трафика, заключается в наличии больших очередей во многих устройствах доступа, таких как модемы DSL.

В следующем разделе дается детальное описание причин, которые вызывают задержки, и даются практические рекомендации по их устранению. Однако, если вас не интересуют пространные размышлизмы, то можете просто пропустить его и сразу перейти к разделам со сценариями.

15.8.1. Почему все так сложно?

Поставщикам услуг Интернет хорошо известно, что пользователей в основном интересует

скорость, с которой они могут получать данные. Но помимо пропускной способности канала, на скорость скачивания очень сильно влияют факты потери пакетов. Увеличение очередей способствует уменьшению потерь, что в свою очередь приводит к увеличению скорости скачивания. Поэтому поставщики услуг, как правило, создают очереди очень большого объема.

Однако, увеличение очередей приводит к появлению задержек в интерактивном трафике. Интерактивные пакеты сначала попадают в исходящую очередь, где они ожидают отправки на удаленную систему, в результате может пройти до нескольких секунд (!), пока они достигнут места назначения. То же самое повторяется на обратном пути -- сначала пакеты попадают в огромную очередь для входящего трафика, у поставщика услуг, долго ожидают отправки и только потом попадают к вам.

В этом руководстве мы расскажем вам, о способах обслуживания очередей, но, к большому сожалению, не все очереди нам доступны. Так, очереди поставщика услуг нам не подвластны совершенно, в то время, как очередь исходящего трафика, скорее всего находится внутри вашего кабельного или DSL модема. Некоторые модели допускают возможность конфигурирования очередей, но чаще всего такая возможность отсутствует.

Итак, что же дальше? Поскольку мы лишены возможности управлять этими очередями, то напрашивается очевидное решение -- они должны быть перемещены на наш Linux-маршрутизатор. К счастью это возможно. Для этого необходимо:

Ограничить скорость исходящего трафика.

> Ограничивая скорость исходящего трафика, величиной несколько меньшей, чем пропускная способность канала, мы тем самым ликвидируем исходящую очередь в модеме. Таким образом, исходящая очередь как бы перемещается в маршрутизатор.

Ограничить скорость входящего трафика.

> Это немного сложнее, поскольку действительно отсутствует возможность влияния на скорость поступления данных. Тем не менее, можно попробовать сбрасывать пакеты, если скорость поступления слишком высока, это заставит TCP/IP снизить скорость передачи до желаемого уровня. Поскольку в данном вопросе чрезмерное усердие может только навредить, то необходимо предусмотреть величину возможного превышения на коротких отрезках времени.

При выполнении этих условий, входящая очередь будет ликвидирована полностью (за исключением коротких пиков), и появляется возможность управления исходящей очередью, используя всю мощь, которую предоставляет операционная система Linux.

После этого остается обеспечить первоочередную передачу интерактивного трафика. А для того, чтобы входящий трафик не блокировался исходящим, необходимо так же обеспечить первоочередную передачу ACK-пакетов. При наличии объемного трафика в обоих направлениях, возникают значительные задержки, поэтому входящие ACK-пакеты не должны проигрывать в конкурентной борьбе с исходящим трафиком.

После выполнения необходимых настроек, мы получили следующие результаты на ADSL соединении в Нидерландах:

```
Базовое время ожидания отклика:
туда-обратно мин/ср/макс = 14.4/17.1/21.7 мсек

Во время скачивания, без формирователя трафика:
туда-обратно мин/ср/макс = 560.9/573.6/586.4 мсек

Во время отправки большого объема, без формирователя трафика:
```

```
туда-обратно мин/ср/макс = 2041.4/2332.1/2427.6 мсек

С формирователем трафика, при отправке большого файла на скорости 220 Кбит/сек:
round-trip min/avg/max = 15.7/51.8/79.9 мсек

С формирователем трафика, при скачивании на скорости 850 Кбит/сек:
туда-обратно мин/ср/макс = 20.4/46.9/74.0 мсек

При наличии исходящего трафика, скорость входящего достигает ~80%
от максимально возможного значения. Скорость исходящего трафика
колеблется около 90%. При этом время ожидания подскакивает до 850 мсек,
причина пока не выяснена.
```

Чего можно ожидать от этого сценария, во многом зависит от фактической пропускной способности канала для исходящего потока. При наличии объемного исходящего трафика, перед исходящим интерактивным пакетом практически всегда будет стоять какой либо другой пакет, что и обусловливает нижний предел времени ожидания. Вы можете рассчитать этот предел, разделив MTU на максимальную скорость для исходящего потока. Типичные значения будут несколько выше. Чтобы достичь лучшего эффекта, можно попробовать несколько уменьшить MTU!

Ниже приводятся две версии сценария формирователя трафика. Одна версия построена на базе HTB, разработанной Девиком (Devik), другая -- на базе CBQ, которая, в отличие от HTB, включена в состав ядра Linux. Оба сценария проверены и дают прекрасные результаты.

15.8.2. Формирователь трафика на базе CBQ.

Может работать практически с любой версией ядра. В данной реализации, внутри CBQ qdisc размещаются две SFQ (Stochastic Fairness Queues), что даст возможность равноправного сосуществования нескольких потоков данных.

Входящий трафик формируется с помощью **tc**-фильтров, содержащих Token Bucket Filter.

Вы можете улучшить сценарий за счет добавления ключевых слов *bounded* в строках, начинающихся со слов **tc class add .. classid 1:20**. Если вы предполагаете уменьшать MTU, не забудьте уменьшить и значения **allot** и **avpkt**!

```
#!/bin/bash

# Формирователь трафика для домашнего соединения с Интернет
#
#
# Установите следующие параметры так, чтобы они были немного меньше фактических
# Единицы измерения -- килобиты
DOWNLINK=800
UPLINK=220
DEV=ppp0

# очистка входящей и исходящей qdisc
tc qdisc del dev $DEV root    2> /dev/null > /dev/null
tc qdisc del dev $DEV ingress 2> /dev/null > /dev/null

###### исходящий трафик

# установка корневой CBQ

tc qdisc add dev $DEV root handle 1: cbq avpkt 1000 bandwidth 10mbit

# ограничить общую исходящую скорость величиной $UPLINK -- это предотвратит
```

```
# появление огромных очередей в DSL модеме,
# которые отрицательно сказываются на величине задержки:
# базовый класс

tc class add dev $DEV parent 1: classid 1:1 cbq rate ${UPLINK}kbit \
allot 1500 prio 5 bounded isolated

# высокоприоритетный (интерактивный) класс 1:10:

tc class add dev $DEV parent 1:1 classid 1:10 cbq rate ${UPLINK}kbit \
   allot 1600 prio 1 avpkt 1000

# класс по-умолчанию 1:20 -- получает немного меньший объем трафика
# и имеет более низкий приоритет:

tc class add dev $DEV parent 1:1 classid 1:20 cbq rate $[9*$UPLINK/10]kbit \
   allot 1600 prio 2 avpkt 1000

# оба получают дисциплину Stochastic Fairness:
tc qdisc add dev $DEV parent 1:10 handle 10: sfq perturb 10
tc qdisc add dev $DEV parent 1:20 handle 20: sfq perturb 10

# определения фильтров
# TOS = Minimum-Delay (ssh, НО НЕ scp) -- в 1:10:
tc filter add dev $DEV parent 1:0 protocol ip prio 10 u32 \
      match ip tos 0x10 0xff  flowid 1:10

# ICMP (ip protocol 1) -- в интерактивный класс 1:10
# так мы сможем удивить своих друзей:
tc filter add dev $DEV parent 1:0 protocol ip prio 11 u32 \
        match ip protocol 1 0xff flowid 1:10

# Поднять скорость входящего трафика, при наличии исходящего -- передать ACK-пакеты
# в интерактивный класс:

tc filter add dev $DEV parent 1: protocol ip prio 12 u32 \
   match ip protocol 6 0xff \
   match u8 0x05 0x0f at 0 \
   match u16 0x0000 0xffc0 at 2 \
   match u8 0x10 0xff at 33 \
   flowid 1:10

# остальной трафик не является интерактивным поэтому передаем его в 1:20

tc filter add dev $DEV parent 1: protocol ip prio 13 u32 \
   match ip dst 0.0.0.0/0 flowid 1:20

########## входящий трафик #############
# необходимо несколько уменьшить скорость поступления входящего трафика,
# это предотвратит задержку пакетов в очередях у поставщика услуг.
# Поставщики имеют обыкновение увеличивать размеры очередей,
# поэтому, экспериментальным путем подберите требуемые значения,
# при которых скачивание будет происходить с максимальной скоростью.
#
# присоединить входной ограничитель:

tc qdisc add dev $DEV handle ffff: ingress

# сбрасывать все подряд (0.0.0.0/0), что приходит со слишком большой скоростью.

tc filter add dev $DEV parent ffff: protocol ip prio 50 u32 match ip src \
   0.0.0.0/0 police rate ${DOWNLINK}kbit burst 10k drop flowid :1
```

Если вы собираетесь использовать этот сценарий совместно с *ppp* -- скопируйте его в `/etc/ppp/ip-up.d`.

Если последние две строки в сценарии порождают сообщения об ошибке -- обновите версию **tc**!

15.8.3. Формирователь трафика на базе HTB.

Следующий вариант сценария достигает поставленных целей за счет использования замечательных особенностей HTB (см. раздел Hierarchical Token Bucket). Требует наложения "заплаты" на ядро!

```
#!/bin/bash

# Формирователь трафика для домашнего соединения с Интернет
#
#
# Установите следующие параметры так, чтобы они были немного меньше фактических
# Единицы измерения -- килобиты
DOWNLINK=800
UPLINK=220
DEV=ppp0

# очистка входящей и исходящей qdisc
tc qdisc del dev $DEV root    2> /dev/null > /dev/null
tc qdisc del dev $DEV ingress 2> /dev/null > /dev/null

###### исходящий трафик

# установка корневой HTB, отправить трафик по-умолчанию в 1:20:

tc qdisc add dev $DEV root handle 1: htb default 20

# ограничить общую исходящую скорость величиной $UPLINK -- это предотвратит
# появление огромных очередей в DSL модеме,
# которые отрицательно сказываются на величине задержки:

tc class add dev $DEV parent 1: classid 1:1 htb rate ${UPLINK}kbit burst 6k

# высокоприоритетный (интерактивный) класс 1:10:

tc class add dev $DEV parent 1:1 classid 1:10 htb rate ${UPLINK}kbit \
   burst 6k prio 1

# класс по-умолчанию 1:20 -- получает немного меньший объем трафика
# и имеет более низкий приоритет:

tc class add dev $DEV parent 1:1 classid 1:20 htb rate $[9*$UPLINK/10]kbit \
   burst 6k prio 2

# оба получают дисциплину Stochastic Fairness:
tc qdisc add dev $DEV parent 1:10 handle 10: sfq perturb 10
tc qdisc add dev $DEV parent 1:20 handle 20: sfq perturb 10

# TOS = Minimum-Delay (ssh, НО НЕ scp) -- в 1:10:
tc filter add dev $DEV parent 1:0 protocol ip prio 10 u32 \
      match ip tos 0x10 0xff  flowid 1:10

# ICMP (ip protocol 1) -- в интерактивный класс 1:10
# так мы сможем удивить своих друзей:
tc filter add dev $DEV parent 1:0 protocol ip prio 10 u32 \
        match ip protocol 1 0xff flowid 1:10

# Поднять скорость входящего трафика, при наличии исходящего -- передать ACK-пакеты
# в интерактивный класс:
```

```
tc filter add dev $DEV parent 1: protocol ip prio 10 u32 \
   match ip protocol 6 0xff \
   match u8 0x05 0x0f at 0 \
   match u16 0x0000 0xffc0 at 2 \
   match u8 0x10 0xff at 33 \
   flowid 1:10

# остальной трафик не является интерактивным поэтому он попадает в 1:20


########## входящий трафик #############
# необходимо несколько уменьшить скорость поступления входящего трафика,
# это предотвратит задержку пакетов в очередях у поставщика услуг.
# Поставщики имеют обыкновение увеличивать размеры очередей,
# поэтому, экспериментальным путем подберите требуемые значения,
# при которых скачивание будет происходить с максимальной скоростью.
#
# присоединить входной ограничитель:

tc qdisc add dev $DEV handle ffff: ingress

# сбрасывать все подряд (0.0.0.0/0), что приходит со слишком большой скоростью.

tc filter add dev $DEV parent ffff: protocol ip prio 50 u32 match ip src \
   0.0.0.0/0 police rate ${DOWNLINK}kbit burst 10k drop flowid :1
```

Если вы собираетесь использовать этот сценарий совместно с *ppp* -- скопируйте его в `/etc/ppp/ip-up.d`.

Если последние две строки в сценарии порождают сообщения об ошибке -- обновите версию **tc**!

15.9. Ограничение скорости для отдельного хоста или подсети.

Хотя очень подробные описания темы раздела можно найти в разных местах, в том числе и в справочном руководстве **man**, тем не менее этот вопрос задается очень часто. К счастью, ответ на него настолько прост, что не нуждается в полном понимании принципов управления трафиком.

Эти три строки сделают все что вам нужно:

```
tc qdisc add dev $DEV root handle 1: cbq avpkt 1000 bandwidth 10mbit

tc class add dev $DEV parent 1: classid 1:1 cbq rate 512kbit \
allot 1500 prio 5 bounded isolated

tc filter add dev $DEV parent 1: protocol ip prio 16 u32 \
match ip dst 195.96.96.97 flowid 1:1
```

Первая строка -- назначает базовую дисциплину на заданный интерфейс и сообщает ядру, что пропускная способность интерфейса составляет 10 Мбит/сек. Если вы установите неверное значение, то особого вреда не будет, однако, всегда стремитесь устанавливать верные значения, это повысит точность вычислений.

Вторая строка создает класс с полосой пропускания 512 Кбит/сек. Подробное описание CBQ содержит раздел Дисциплины обработки очередей для управления пропускной способностью.

Последняя строка говорит о том, какой трафик должен формироваться классом, определенным выше. Трафик, не подпадающий под заданное в фильтре условие, НЕ ОГРАНИЧИВАЕТСЯ! Более сложные примеры назначения условий (подсети, порт отправителя, порт адресата), вы найдете в разделе Наиболее употребимые способы фильтрации.

Если вы внесли какие-либо изменения в сценарий и желаете перезапустить его -- предварительно запустите команду **tc qdisc del dev $DEV root**, чтобы очистить существующую конфигурацию.

Сценарий может быть немного улучшен, за счет добавления в конец дополнительной строки **tc qdisc add dev $DEV parent 1:1 sfq perturb 10**. За подробным описанием обращайтесь к разделу Stochastic Fairness Queueing.

15.10. Пример подключения локальной сети к Интернет через NAT, с организацией QoS.

Меня зовут Педро Ларрой (Pedro Larroy) <piotr%member.fsf.org>. Здесь я расскажу об общих принципах настройки соединения локальной сети, в которой имеется большое число пользователей, к Интернет через маршрутизатор, работающий под управлением Linux. Маршрутизатор имеет реальный IP-адрес и производит Трансляцию Сетевых Адресов (NAT). Я живу в университетском общежитии, где проложена локальная сеть на 198 пользователей. Эта сеть соединена с Интернет через маршрутизатор, который я администрирую. Пользователи очень интенсивно работают в пиринговых сетях, что требует соответствующего управления трафиком. Надеюсь, что этот пример будет интересен читателям lartc.

Прежде всего я опишу процесс настройки своего маршрутизатора шаг за шагом, и в заключение расскажу, как сделать этот процесс автоматическим, выполняющимся в процессе загрузки системы. Сеть, к которой относится этот пример, является локальной (LAN). Она подключена к Интернет через маршрутизатор, который имеет единственный реальный IP-адрес. Разделение единственного реального IP-адреса между всеми пользователями в локальной сети осуществляется с помощью нескольких правил **iptables**. Для этого необходимо:

Ядро Linux 2.4.18 или выше

> На ядро нужно наложить заплату, для поддержки HTB.

iproute

> Убедитесь, что **tc** поддерживает HTB. Скомпилированная версия распространяется вместе с HTB.

iptables

15.10.1. Начнем с оптимизации пропускной способности.

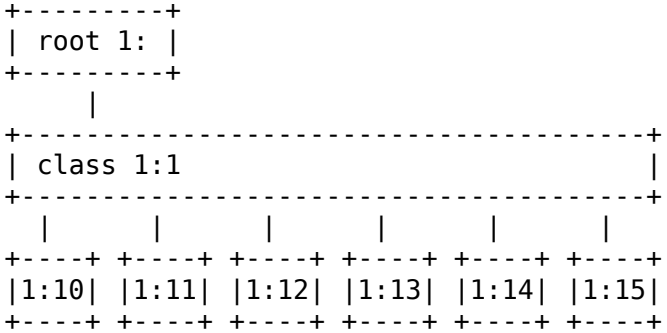
Для начала создадим несколько дисциплин (qdiscs), которые будут обслуживать трафик. Первой создается htb qdisc с 6-ю классами и различными приоритетами. Каждому классу назначена определенная пропускная способность, но при этом они могут задействовать неиспользуемую пропускную способность, если она не занята другими классами. Напомню, что классы с более высоким приоритетом (т.е. с более низким числом prio) будут получать "излишек" канала

первыми. Подключение к Интернет осуществляется через модем ADSL, с пропускной способностью для входящего трафика 2 Мбит/сек, исходящего -- 300 Кбит/сек. Я ограничиваю исходящую пропускную способность величиной в 240 Кбит/сек по той простой причине, что это максимальное значение, при котором время ожидания отклика остается минимальным. Величина этот параметра может быть определена экспериментально, путем наблюдения за изменением времени отклика при изменении величины пропускной способности.

Для начала, присвойте переменной CEIL величину, составляющую 75% от общей пропускной способности для исходящего трафика. Там, где я использую eth0 -- назначьте свой интерфейс, который "смотрит" в Интернет. Сценарий (на языке командной оболочки), выполняющий настройку, начинается со следующих строк:

```
CEIL=240
tc qdisc add dev eth0 root handle 1: htb default 15
tc class add dev eth0 parent 1: classid 1:1 htb rate ${CEIL}kbit ceil ${CEIL}kbit
tc class add dev eth0 parent 1:1 classid 1:10 htb rate 80kbit ceil 80kbit prio 0
tc class add dev eth0 parent 1:1 classid 1:11 htb rate 80kbit ceil ${CEIL}kbit prio
1
tc class add dev eth0 parent 1:1 classid 1:12 htb rate 20kbit ceil ${CEIL}kbit prio
2
tc class add dev eth0 parent 1:1 classid 1:13 htb rate 20kbit ceil ${CEIL}kbit prio
2
tc class add dev eth0 parent 1:1 classid 1:14 htb rate 10kbit ceil ${CEIL}kbit prio
3
tc class add dev eth0 parent 1:1 classid 1:15 htb rate 30kbit ceil ${CEIL}kbit prio
3
tc qdisc add dev eth0 parent 1:12 handle 120: sfq perturb 10
tc qdisc add dev eth0 parent 1:13 handle 130: sfq perturb 10
tc qdisc add dev eth0 parent 1:14 handle 140: sfq perturb 10
tc qdisc add dev eth0 parent 1:15 handle 150: sfq perturb 10
```

Эти строки создают одноярусное дерево HTB:



classid 1:10 htb rate 80kbit ceil 80kbit prio 0

> Это класс с наивысшим приоритетом. Пакеты, попадающие в этот класс, будут иметь самую низкую задержку и получат избыток канала в первую очередь. Сюда будет направляться интерактивный трафик: **ssh**, **telnet**, **dns**, **quake3**, **irc**, а так же пакеты с установленным флагом SYN.

classid 1:11 htb rate 80kbit ceil ${CEIL}kbit prio 1

> Это первый класс, через который будет проходить довольно объемный трафик. В моем случае -- это трафик от локального WEB-сервера и запросы к внешним WEB-серверам, исходящий порт 80 и порт назначения 80, соответственно.

classid 1:12 htb rate 20kbit ceil ${CEIL}kbit prio 2

В этот класс помещаются пакеты, с установленным битом Maximize-Throughput в поле TOS, а так же иной трафик, который генерируется *локальными процессами* на маршрутизаторе, отправляемый в Интернет. Таким образом, все последующие классы будут иметь дело только с перенаправляемым трафиком.

classid 1:13 htb rate 20kbit ceil ${CEIL}kbit prio 2

Высокоприоритетный класс, обслуживающий объемный трафик, поступающий от компьютеров из локальной сети.

classid 1:14 htb rate 10kbit ceil ${CEIL}kbit prio 3

Этот класс обслуживает почтовый трафик (SMTP,pop3...) и пакеты, с установленным битом Minimize-Cost в поле TOS.

classid 1:15 htb rate 30kbit ceil ${CEIL}kbit prio 3

Последний класс. Он обслуживает прочий трафик, поступающий от компьютеров из локальной сети. Сюда попадает все, что относится к работе в пиринговых сетях, т.е. kazaa, edonkey и пр.

15.10.2. Классификация пакетов.

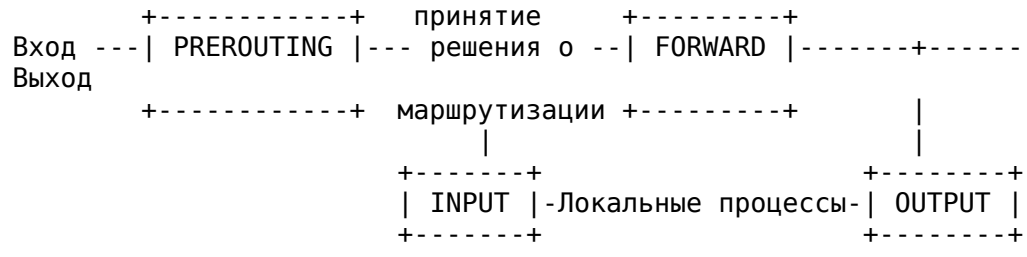
Мы создали различные классы обработки трафика, но классификация пока отсутствует, поэтому, к настоящему моменту, весь трафик пойдет через класс 1:15 (который назначен классом по-умолчанию: **tc qdisc add dev eth0 root handle 1: htb default 15**). Теперь самое главное -- нужно распределить трафик по имеющимся классам.

Устанавим фильтры, которые будут выполнять классификацию пакетов, основываясь на метках **iptables**. Мне нравятся **iptables** за их чрезвычайную гибкость и за возможность подсчитывать количество пакетов, пропущенных тем или иным правилом. Добавим в сценарий следующие строки:

```
tc filter add dev eth0 parent 1:0 protocol ip prio 1 handle 1 fw classid 1:10
tc filter add dev eth0 parent 1:0 protocol ip prio 2 handle 2 fw classid 1:11
tc filter add dev eth0 parent 1:0 protocol ip prio 3 handle 3 fw classid 1:12
tc filter add dev eth0 parent 1:0 protocol ip prio 4 handle 4 fw classid 1:13
tc filter add dev eth0 parent 1:0 protocol ip prio 5 handle 5 fw classid 1:14
tc filter add dev eth0 parent 1:0 protocol ip prio 6 handle 6 fw classid 1:15
```

Здесь задаются соответствия между специфическими значениями FWMARK (handle x fw) и классами (classid x:x). Теперь рассмотрим процесс установки меток на пакеты.

Для начала необходимо разобраться с тем, как движутся пакеты через **iptables**:



Далее я буду исходить из предположения, что всем таблицам назначена политика по-умолчанию **-P ACCEPT**. Наша локальная сеть относится к классу B, с адресами 172.17.0.0/16. Реальный IP-адрес -- 212.170.21.172

Добавим правило **iptables**, которое будет выполнять SNAT, что позволит пользователям локальной сети общаться с внешним миром, и разрешим форвардинг пакетов:

```
echo 1 > /proc/sys/net/ipv4/ip_forward
iptables -t nat -A POSTROUTING -s 172.17.0.0/255.255.0.0 -o eth0 -j SNAT --to-source
212.170.21.172
```

Проверим, что пакеты уходят через класс 1:15:

```
tc -s class show dev eth0
```

Добавим в цепочку PREROUTING, таблицы mangle, правила для установки меток на пакеты:

```
iptables -t mangle -A PREROUTING -p icmp -j MARK --set-mark 0x1
iptables -t mangle -A PREROUTING -p icmp -j RETURN
```

Теперь вы должны наблюдать увеличение значения счетчика пакетов в классе 1:10, при попытке **ping**-ануть из локальной сети какой-нибудь сайт в Интернете.

```
tc-s класс показывают dev eth0
```

Действие **-j RETURN** предотвращает движение пакетов по всем правилам. Поэтому все ICMP-пакеты будут проходить только это правило. Добавим еще ряд правил, которые будут изменять биты в поле TOS:

```
iptables -t mangle -A PREROUTING -m tos --tos Minimize-Delay -j MARK --set-mark 0x1
iptables -t mangle -A PREROUTING -m tos --tos Minimize-Delay -j RETURN
iptables -t mangle -A PREROUTING -m tos --tos Minimize-Cost -j MARK --set-mark 0x5
iptables -t mangle -A PREROUTING -m tos --tos Minimize-Cost -j RETURN
iptables -t mangle -A PREROUTING -m tos --tos Maximize-Throughput -j MARK --set-mark
0x6
iptables -t mangle -A PREROUTING -m tos --tos Maximize-Throughput -j RETURN
```

Поднимем приоритет для **ssh**-пакетов:

```
iptables -t mangle -A PREROUTING -p tcp -m tcp --sport 22 -j MARK --set-mark 0x1
iptables -t mangle -A PREROUTING -p tcp -m tcp --sport 22 -j RETURN
```

а так же для пакетов, с которых начинается TCP-соединение, т.е. SYN-пакетов:

```
iptables -t mangle -I PREROUTING -p tcp -m tcp --tcp-flags SYN,RST,ACK SYN -j MARK
--set-mark 0x1
iptables -t mangle -I PREROUTING -p tcp -m tcp --tcp-flags SYN,RST,ACK SYN -j RETURN
```

И так далее. После того, как в цепочку PREROUTING, таблицы mangle, будут внесены все необходимые правила, закончим ее правилом:

```
iptables -t mangle -A PREROUTING -j MARK --set-mark 0x6
```

Это заключительное правило отправит оставшиеся немаркированные пакеты в класс 1:15. Фактически, это правило можно опустить, так как класс 1:15 был задан по-умолчанию, но тем не менее, я оставляю его, чтобы сохранить единство настроек и кроме того, иногда бывает полезно увидеть счетчик пакетов для этого правила.

Нелишним будет добавить те же правила в цепочку OUTPUT, заменив имя цепочки PREROUTING на OUTPUT (s/PREROUTING/OUTPUT/). Тогда трафик, сгенерированный локальными процессами на маршрутизаторе, также будет классифицирован по категориям. Но, в отличие от вышеприведенных правил, в цепочке OUTPUT, я устанавливаю метку **-j MARK --set-mark 0x3**, таким образом трафик от маршрутизатора получает более высокий приоритет.

15.10.3. Дополнительная оптимизация

В результате приведенных настроек, мы получили вполне работоспособную конфигурацию. Однако, в каждом конкретном случае, эти настройки всегда можно немного улучшить. Найдите время и проследите -- куда идет основной трафик и как лучше им распорядиться. Я потратил огромное количество времени и наконец довел свою конфигурацию до оптимального уровня, практически сведя на нет бесчисленные таймауты.

Если вдруг обнаружится, что через некоторые классы проходит подавляющее большинство трафика, то к ним можно прикрепить другую дисциплину организации очереди, чтобы распределить канал более равномерно:

```
tc qdisc add dev eth0 parent 1:13 handle 130: sfq perturb 10
tc qdisc add dev eth0 parent 1:14 handle 140: sfq perturb 10
tc qdisc add dev eth0 parent 1:15 handle 150: sfq perturb 10
```

15.10.4. Выполнение настроек во время загрузки системы.

Уверен, что можно найти множество способов, чтобы произвести настройку маршрутизатора во время загрузки. Для себя я создал скрипт `/etc/init.d/packetfilter`, который принимает команды [start | stop | stop-tables | start-tables | reload-tables]. Он конфигурирует дисциплины (qdiscs) и загружает необходимые модули ядра. Этот же сценарий загружает правила **iptables** из файла `/etc/network/iptables-rules`, которые предварительно могут быть сохранены утилитой **iptables-save** и восстановлены -- **iptables-restore**.

Глава 16. Построение мостов и псевдо-мостов с Proxy ARP.

Мосты (bridges) -- это специальные устройства, которые могут быть установлены в сети и не требуют предварительной настройки. Сетевой коммутатор (swtch) -- это особый вид многопортового моста. Мост -- это чаще всего двухпортовый коммутатор (switch). На базе Linux может быть построен многопортовый (несколько интерфейсов) мост, по сути -- настоящий коммутатор (switch).

Мосты часто применяются для объединения фрагментированных стационарных сетей. Поскольку мост -- это устройство 2-го уровня (Канальный уровень по классификации OCI), который лежит ниже сетевого уровня, где "заправляют" протоколы IP, то ни серверы, ни маршрутизаторы даже не подозревают о его существовании. Это означает, что вы можете блокировать или изменять некоторые пакеты, а так же формировать трафик по своему усмотрению.

Еще одно замечательное свойство моста -- в случае выхода из строя, мост может быть заменен отрезком кабеля или сетевым концентратором (hub -- хабом).

Одна из отрицательных сторон -- мост может стать причиной большой неразберихи. **traceroute** его не "видит" и не сможет указать в каком месте теряются пакеты. Так что ничего удивительного, если какая-нибудь организация считает правильным "ничего не менять".

Мосты на базе Linux 2.4/2.5 подробно описаны на сайте http://bridge.sourceforge.net/.

16.1. Бриджинг и iptables.

Что касается Linux 2.4.20, то бриджинг и **iptables** не "видят" друг друга без установки вспомогательных модулей. Если построен мост между eth0 и eth1, то пакеты, передаваемые по мосту, проходят мимо **iptables**. Это означает, что ни фильтрация, ни NAT, ни возможность внесения изменений в заголовки пакетов (mangling) вам недоступны. Начиная с Linux 2.5.45 это было исправлено.

Хочу упомянуть еще об одном проекте -- **etables**. Он позволяет вытворять такие штуки, как MACNAT и *brouting*. Это действительно круто!

16.2. Бриджинг и шейпинг.

Принцип работы соответствует заголовку. Вы должны убедиться, что четко представляете -- с какой стороны подключен каждый из интерфейсов, иначе может получиться так, что будет производиться попытка формирования экспортируемого трафика на внутреннем интерфейсе. Если необходимо -- используйте утилиту **tcpdump**.

16.3. Псевдо-мосты с проксированием ARP.

Если вы просто хотите построить псевдо-мост, то можете сразу перейти к разделу Реализация, однако мы рекомендуем все-таки прочитать о том как все это работает на практике.

По-умолчанию, обычный мост просто передает пакеты с одного интерфейса на другой в неизменном виде. Он рассматривает только аппаратный адрес пакета, чтобы определить -- в каком направлении нужно передать пакет. Это означает, что Linux может переправлять любой вид трафика, даже тот, который ему не известен, если пакеты имеют аппаратный адрес.

Псевдо-мост работает несколько иначе и скорее больше походит на скрытый маршрутизатор, чем на мост, но подобно мосту имеет некоторое влияние на архитектуру сети.

Правда это не совсем мост, поскольку пакеты в действительности проходят через ядро и могут быть отфильтрованы, изменены, перенаправлены или направлены по другому маршруту.

Настоящий мост в принципе тоже может делать это, но для этого требуется специальное программное обеспечение, например: Ethernet Frame Diverter.

Еще одно преимущество псевдо-моста состоит в том, что он не может передавать пакеты протоколов, которые "не понимает" -- что предохраняет сеть от заполнения всяким "мусором". В случае, если вам необходимо переправлять такие пакеты (например, пакеты SAP или Netbeui), то устанавливайте настоящий мост.

16.3.1. ARP и проксирование ARP

Когда некий узел сети желает установить связь с другим узлом, находящимся в том же физическом сегменте сети, то он отсылает пакет ARP-запроса, который, в переводе на человеческий язык, может звучать примерно так: "У кого установлен адрес 10.0.0.1? Сообщите по адресу 10.0.0.7". В ответ на запрос, 10.0.0.1 ответит коротким пакетом "Я здесь! Мой аппаратный адрес xx:xx:xx:xx:xx:xx".

Когда 10.0.0.7 получит ответ, он запомнит аппаратный адрес хоста 10.0.0.1 и будет хранить его в кэше некоторое время.

При настройке моста мы можем указать ему на необходимость отвечать на ARP-запросы. Это вынудит узлы сети передавать ARP-запросы мосту, который затем обработает их и передаст на соответствующий интерфейс.

Таким образом, всякий раз, когда компьютер по одну сторону моста запрашивает аппаратный адрес компьютера, находящегося по другую сторону, то на запрос отвечает мост, как бы говоря "передавай все через меня".

Вследствие чего весь трафик попадет на мост и будет передан в нужном направлении.

16.3.2. Реализация

В ранних версиях Linux, для реализации проксирования протокола ARP, вам пришлось бы настраивать ядро. Для того, чтобы сконфигурировать псевдо-мост, вам пришлось бы вручную описать все маршруты по обе стороны моста И создать соответствующие правила. Это требовало достаточно большого объема ручного ввода, что само по себе могло повлечь за собой большое число ошибок.

В Linux 2.4/2.5 (а возможно и в 2.2) все это было заменено установкой флага `proxy_arp` в файловой системе `proc`. Теперь процедура настройки псевдо-моста выглядит так:

1. Назначить IP-адреса интерфейсам с обоих сторон.

2. Создать маршруты, так чтобы ваша машина знала, какие адреса, по какую сторону находятся.

3. Включить проксирование ARP для обоих интерфейсов, например командами:

```
echo 1 > /proc/sys/net/ipv4/conf/ethL/proxy_arp
echo 1 > /proc/sys/net/ipv4/conf/ethR/proxy_arp
```

где символами L и R обозначены интерфейсы по одну и по другую сторону моста (Left и Right -- "слева" и "справа").

Кроме того, не забывайте включить флаг ip_forwarding! Для истинного моста установка этого флага не требуется.

И еще одно обстоятельство, на которое необходимо обратить внимание -- не забывайте очистить кэши arp на компьютерах в сети, которые могут содержать устаревшие сведения и которые больше не являются правильными.

На Cisco для этого существует команда **clear arp-cache**, в Linux -- команда **arp -d ip.address**. В принципе, вы можете подождать, пока кэши очистятся самостоятельно, за счет истечения срока хранения записей, но это может занять довольно продолжительное время.

Вы можете ускорить этот процесс, используя замечательный инструмент **arping**, который во многих дистрибутивах является частью пакета **iputils**. Утилитой **arping** вы может отослать незапрошенные arp-сообщения, чтобы модифицировать удаленные кэши arp.

Это очень мощный метод, который, к сожалению, может использоваться злоумышленниками для нарушения правильной маршрутизации!

В Linux 2.4, возможно потребуется выполнить команду **echo 1>/proc/sys/net/ipv4/ip_nonlocal_bind** прежде, чем появится возможность отправлять незапрошенные arp-сообщения!

Вы можете также обнаружить некоторые проблемы с сетью, если имеете привычку к определению маршрутов без сетевых масок. При выполнении выборочной маршрутизации совершенно необходимо, чтобы сетевые маски были установлены должным образом!

Глава 17. Динамическая маршрутизация -- OSPF и BGP.

Как только ваша локальная сеть становится достаточно большой, или вы приступаете к обслуживанию некоего сегмента Интернет, у вас сразу же возникает необходимость в динамической маршрутизации ваших данных.

Интернет стандартизирован главным образом на OSPF (от англ. Open Shortest Pass First -- Открытый протокол поиска Кратчайшего Маршрута. RFC 2328) и BGP4 (Border Gateway Protocol -- Протокол Пограничных Маршрутизаторов, RFC 1771). Linux поддерживает оба, посредством **gated** и **zebra**.

Поскольку описание этих протоколов выходит за рамки данного документа, мы дадим лишь некоторые ссылки на документы, содержащие подробное описание:

Краткий обзор:

Cisco Systems Designing large-scale IP Internetworks

Протокол OSPF:

Moy, John T. "OSPF. The anatomy of an Internet routing protocol" Addison Wesley. Reading, MA. 1998.

Протокол BGP:

Halabi, Bassam "Internet routing architectures" Cisco Press (New Riders Publishing). Indianapolis, IN. 1997.

А так же:

Cisco Systems Using the Border Gateway Protocol for interdomain routing

Хотя примеры приводятся исключительно для маршрутизаторов Cisco, они практически полностью совпадают с языком конфигурирования в Zebra :-)

17.1. Настройка OSPF в Zebra

Пожалуйста, дайте мне знать -- насколько верна следующая информация, а так же присылайте ваши предложения, комментарии. Zebra -- большой программный пакет динамической маршрутизации, который разработали Кунихиро Ишигуро (Kunihiro Ishiguro), Тошиаки Такеда (Toshiaki Takada) и Ясахиро Охара (Yasuhiro Ohara). С помощью Zebra вы легко и быстро сможете настроить OSPF, но на практике, при настройке протокола под весьма специфические потребности, вы столкнетесь со значительным числом параметров. Ниже приводятся некоторые из характеристик протокола OSPF:

Иерархичность

> Сети объединяются в иерархические *области* (area) -- группу смежных сетей, которые находятся под единым управлением и совместно используют общую стратегию маршрутизации. Взаимодействие между областями осуществляется посредством стержневой части (backbone), которая обозначается как *область 0* (area 0). Все стержневые маршрутизаторы обладают информацией о маршрутах ко всем другим областям.

Быстрая сходимость

> Алгоритм поиска кратчайшего пути (SPF) обеспечивает быструю сходимость, а отсюда и более быстрый, в сравнении с протоколом RIP, выбор маршрута.

Эффективное использование пропускной способности

> Использование групповых сообщений, вместо широковещательных, предотвращает "затопление" информацией о маршрутах посторонних узлов сети, которые могут быть не заинтересованы в получении этих сведений, что значительно снижает нагрузку на каналы связи. Кроме того, *Внутренние Маршрутизаторы* (т.е. те, которые не имеют интерфейсов за пределами своей области) не обладают информацией о маршрутах в других областях, за счет чего так же достигается уменьшение трафика маршрутизации. Роутеры, имеющие несколько интерфейсов в более чем одной области, называются *Пограничными Маршрутизаторами* (Area Border Routers), они поддерживают отдельные топологические базы данных для каждой из областей, с которыми соединены.

Ресурсоемкость

Протокол OSPF основан на алгоритме Shortest Path First, предложенном Е.В.Дейкстрой (E.W.Dijkstra), который требует больших вычислительных затрат, нежели иные алгоритмы маршрутизации. Но в действительности он не так уж и плох, поскольку кратчайший маршрут рассчитывается только в пределах одной области, причем для сетей малого и среднего размеров -- это вообще не проблема, так что вы не будете даже обращать внимания на это обстоятельство.

Состояние маршрута

OSPF представляет собой протокол состояния маршрута. В качестве метрик используются -- пропускная способность, надежность и стоимость.

Открытость и наличие программного обеспечения под GPL.

OSPF -- это *открытый* протокол, а Zebra выпускается под GPL, что дает дополнительные преимущества перед проприетарными протоколами и программными продуктами.

17.1.1. Предварительные условия.

Ядро Linux:

Собранное с CONFIG_NETLINK_DEV и CONFIG_IP_MULTICAST (я не вполне уверен, возможно требуется еще что-то)

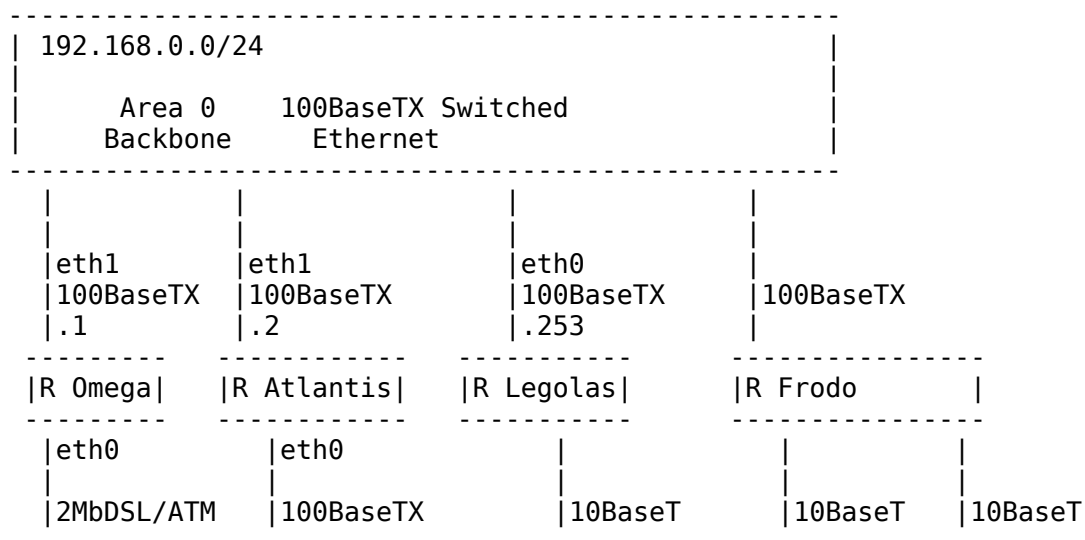
Iproute

Zebra

Пакет может входить в состав вашего дистрибутива. Если нет, обращайтесь на http://www.zebra.org/.

17.1.2. Конфигурирование.

Рассмотрим конфигурирование Zebra на примере сети:



```
------------------------------
| Internet |   | 172.17.0.0/16        Area 1      |       |  192.168.1.0/24 wlan
Area 2|
------------   |         Student network (dorm)   |       |       barcelonawireless
|
               ------------------------------------
------------------------------
```

Пусть вас не пугает эта схема -- дело в том, что большую часть работы Zebra выполнит самостоятельно и вам не потребуется вручную "поднимать" все маршруты. Самое главное, что вы должны уяснить из этой схемы -- это топология сети. И особое внимание обратите на *область 0* (area 0), как самую важную часть. Для начала сконфигурируем zebra под свои потребности (поправим файл `zebra.conf`):

```
hostname omega
password xxx
enable password xxx
!
! Описание интерфейсов.
!
!interface lo
! пример описания интерфейса.
!
interface eth1
multicast
!
! Статический маршрут по-умолчанию
!
ip route 0.0.0.0/0 212.170.21.129
!
log file /var/log/zebra/zebra.log
```

В дистрибутиве Debian, кроме того необходимо подредактировать файл `/etc/zebra/daemons`, чтобы обеспечить запуск демонов во время загрузки системы.

```
zebra=yes
ospfd=yes
```

Затем нужно внести соответствующие изменения в `ospfd.conf` (для случая IPv4) или в `ospf6d.conf` (для случая IPv6). Мой `ospfd.conf` выглядит так:

```
hostname omega
password xxx
enable password xxx
!
router ospf
  network 192.168.0.0/24 area 0
  network 172.17.0.0/16 area 1
!
! направить вывод на stdout в журнал
log file /var/log/zebra/ospfd.log
```

Здесь размещены инструкции, описывающие топологию сети.

17.1.3. Запуск Zebra

Теперь запустим Zebra. Сделать это можно вручную -- дав прямую команду **zebra -d**, либо с помощью сценария начальной загрузки -- **/etc/init.d/zebra start**. После запуска, в журнале `ospfd.log`, появятся строки, примерно с таким содержанием:

```
2002/12/13 22:46:24 OSPF: interface 192.168.0.1 join AllSPFRouters Multicast group.
2002/12/13 22:46:34 OSPF: SMUX_CLOSE with reason: 5
2002/12/13 22:46:44 OSPF: SMUX_CLOSE with reason: 5
2002/12/13 22:46:54 OSPF: SMUX_CLOSE with reason: 5
2002/12/13 22:47:04 OSPF: SMUX_CLOSE with reason: 5
2002/12/13 22:47:04 OSPF: DR-Election[1st]: Backup 192.168.0.1
2002/12/13 22:47:04 OSPF: DR-Election[1st]: DR     192.168.0.1
2002/12/13 22:47:04 OSPF: DR-Election[2nd]: Backup 0.0.0.0
2002/12/13 22:47:04 OSPF: DR-Election[2nd]: DR     192.168.0.1
2002/12/13 22:47:04 OSPF: interface 192.168.0.1 join AllDRouters Multicast group.
2002/12/13 22:47:06 OSPF: DR-Election[1st]: Backup 192.168.0.2
2002/12/13 22:47:06 OSPF: DR-Election[1st]: DR     192.168.0.1
2002/12/13 22:47:06 OSPF: Packet[DD]: Negotiation done (Slave).
2002/12/13 22:47:06 OSPF: nsm_change_status(): scheduling new router-LSA origination
2002/12/13 22:47:11 OSPF: ospf_intra_add_router: Start
```

Не обращайте внимания на строки "...SMUX_CLOSE...", поскольку они относятся к SNMP и не представляют интереса для нас. Из приведенного листинга видно, что 192.168.0.1 -- это *Выделенный Маршрутизатор* (Designated Router), а 192.168.0.2 -- *Резервный Выделенный Маршрутизатор* (Backup Designated Router).

И **zebra**, и **ospfd** допускают возможность интерактивного взаимодействия с ними через **telnet**:

```
$ telnet localhost zebra
$ telnet localhost ospfd
```

Попробуем посмотреть список установленных маршрутов, залогировавшись в **zebra**:

```
root@atlantis:~# telnet localhost zebra
Trying 127.0.0.1...
Connected to atlantis.
Escape character is '^]'.

Hello, this is zebra (version 0.92a).
Copyright 1996-2001 Kunihiro Ishiguro.

User Access Verification

Password:
atlantis> show ip route
Codes: K - kernel route, C - connected, S - static, R - RIP, O - OSPF,
       B - BGP, > - selected route, * - FIB route

K>* 0.0.0.0/0 via 192.168.0.1, eth1
C>* 127.0.0.0/8 is directly connected, lo
O   172.17.0.0/16 [110/10] is directly connected, eth0, 06:21:53
C>* 172.17.0.0/16 is directly connected, eth0
O   192.168.0.0/24 [110/10] is directly connected, eth1, 06:21:53
C>* 192.168.0.0/24 is directly connected, eth1
atlantis> show ip ospf border-routers
============ OSPF router routing table =============
R    192.168.0.253         [10] area: (0.0.0.0), ABR
                           via 192.168.0.253, eth1
                                 [10] area: (0.0.0.1), ABR
                           via 172.17.0.2, eth0
```

или напрямую, с помощью **iproute**:

```
root@omega:~# ip route
212.170.21.128/26 dev eth0  proto kernel  scope link  src 212.170.21.172
192.168.0.0/24 dev eth1  proto kernel  scope link  src 192.168.0.1
172.17.0.0/16 via 192.168.0.2 dev eth1  proto zebra  metric 20
default via 212.170.21.129 dev eth0  proto zebra
root@omega:~#
```

Отсюда видно, что **zebra** добавила ряд маршрутов, которых в таблице раньше не было. Новые маршруты появляются спустя несколько секунд после того, как были запущены **zebra** и **ospfd**. Теперь вы можете попробовать **ping**-ануть некоторые из узлов сети. **Zebra** выставляет маршруты автоматически, все что от вас требуется -- прописать маршрутизаторы в конфигурационный файл и этого будет достаточно!

Для захвата и анализа OSPF-пакетов можно воспользоваться командой:

```
tcpdump -i eth1 ip[9] == 89
```

где число 89 -- это номер протокола OSPF, а 9 -- это номер октета в ip-заголовке, где хранится номер протокола.

OSPF имеет ряд дополнительных настраиваемых параметров, имеющих особое значение при работе в больших сетях. В одном из следующих выпусков этого документа мы покажем некоторые методологии тонкой подстройки протокола OSPF.

17.2. Настройка BGP4 с помощью Zebra.

Версия 4 *Протокола Пограничных Маршрутизаторов* (BGP4) -- это протокол динамический маршрутизации, описанный в RFC 1771. Он предназначен для обмена информацией (т.е. таблицами маршрутизации) между маршрутизаторами, с целью обеспечить согласующееся представление о данной *Автономной Системе* (Autonomous System -- AS). Может использоваться в режиме EGP (от англ. Exterior Gateway Protocol -- Протокол Внешних Маршрутизаторов) или IGP (от англ. Interior Gateway Protocol -- Протокол Внутренних Маршрутизаторов). В случае EGP -- требуется, чтобы каждый из узлов сети имел свой собственный номер *Автономной Системы* (AS). BGP4 поддерживает *Безклассовую Внутридоменную Маршрутизацию* (Classless Inter Domain Routing -- CIDR) и объединение маршрутов (объединение нескольких маршрутов в один).

17.2.1. Конфигурация сети (пример).

Последующие примеры будем рассматривать на основе конфигурации сети, которая приведена ниже. AS 1 и 50 имеют еще ряд "соседей", но мы будем рассматривать только эти две Автономные Системы, в качестве своих "соседей". Взаимодействие между сетями, в данном примере, производится через туннели, но в общем случае это не обязательно.

В данном примере используются зарезервированные номера AS, поэтому вы должны будете подставить свои собственные числа.

```
          --------------------
          | 192.168.23.12/24 |
          |    AS: 23        |
          --------------------
            /              \
           /                \
          /                  \
------------------       ------------------
| 192.168.1.1/24 |-------| 10.10.1.1/16   |
|    AS: 1       |       |    AS: 50      |
------------------       ------------------
```

17.2.2. Конфигурирование (пример).

Следующая конфигурация написана для узла 192.168.23.12/24, но может быть с легкостью адаптирована и для других узлов.

Начинается с установки некоторых общих параметров, таких как: имя хоста, пароль и ключи отладки:

```
! имя хоста
hostname anakin

! пароль
password xxx

! разрешить пароль (режим суперпользователя)
enable password xxx

! путь к файлу журнала
log file /var/log/zebra/bgpd.log

! отладка: выводить отладочную информацию (этот раздел позднее может быть удален)
debug bgp events
debug bgp filters
debug bgp fsm
debug bgp keepalives
debug bgp updates
```

Список доступа, используемый для ограничения перераспределения в локальных сетях (RFC 1918).

```
! RFC 1918 networks
access-list local_nets permit 192.168.0.0/16
access-list local_nets permit 172.16.0.0/12
access-list local_nets permit 10.0.0.0/8
access-list local_nets deny any
```

Следующий шаг -- настройка каждой из AS:

```
! Собственный номер AS
router bgp 23
```

```
    ! IP-адрес маршрутизатора
    bgp router-id 192.168.23.12

    ! наша сеть, для оповещения "соседей"
    network 192.168.23.0/24

    ! объявлять о всех подключенных маршрутах (= непосредственно подключенных
интерфейсах)
    redistribute connected

    ! объявлять о корневых маршрутах (= подставленных вручную)
    redistribute kernel
```

Блок "router bgp" должен содержать сведения о своих "соседях":

```
neighbor 192.168.1.1 remote-as 1
neighbor 192.168.1.1 distribute-list local_nets in
neighbor 10.10.1.1   remote-as 50
neighbor 10.10.1.1   distribute-list local_nets in
```

17.2.3. Проверка конфигурации.

vtysh -- это мультиплексор, соединяющий все интерфейсы Zebra.

```
anakin# sh ip bgp summary
BGP router identifier 192.168.23.12, local AS number 23
2 BGP AS-PATH entries
0 BGP community entries

Neighbor        V    AS MsgRcvd MsgSent   TblVer  InQ OutQ Up/Down  State/PfxRcd
10.10.0.1       4    50      35      40        0    0    0 00:28:40        1
192.168.1.1     4     1   27574   27644        0    0    0 03:26:04       14

Total number of neighbors 2
anakin#
anakin# sh ip bgp neighbors 10.10.0.1
BGP neighbor is 10.10.0.1, remote AS 50, local AS 23, external link
  BGP version 4, remote router ID 10.10.0.1
  BGP state = Established, up for 00:29:01
  ....
anakin#
```

А теперь посмотрим -- какие маршруты были получены от "соседей":

```
anakin# sh ip ro bgp
Codes: K - kernel route, C - connected, S - static, R - RIP, O - OSPF,
       B - BGP, > - selected route, * - FIB route

B>* 172.16.0.0/14 [20/0] via 192.168.1.1, tun0, 2d10h19m
B>* 172.30.0.0/16 [20/0] via 192.168.1.1, tun0, 10:09:24
B>* 192.168.5.10/32 [20/0] via 192.168.1.1, tun0, 2d10h27m
B>* 192.168.5.26/32 [20/0] via 192.168.1.1, tun0, 10:09:24
B>* 192.168.5.36/32 [20/0] via 192.168.1.1, tun0, 2d10h19m
B>* 192.168.17.0/24 [20/0] via 192.168.1.1, tun0, 3d05h07m
B>* 192.168.17.1/32 [20/0] via 192.168.1.1, tun0, 3d05h07m
B>* 192.168.32.0/24 [20/0] via 192.168.1.1, tun0, 2d10h27m
```

Глава 18. Прочие возможности.

В этой главе перечисляются известные нам проекты, которые так или иначе связаны с темой маршрутизации и управления трафиком. Описание некоторых из них заслуживает отдельной главы, другие проекты сами предоставляют настолько качественную документацию, что не нуждаются в дополнительных HOWTO.

Реализация 802.1Q VLAN для Linux

VLAN -- очень интересный способ организации нескольких виртуальных локальных сетей в единой физической среде. Много полезной информации о виртуальных сетях вы найдете по адресу: ftp://ftp.netlab.ohio-state.edu/pub/jain/courses/cis788-97/virtual_lans/index.htm. С помощью этой реализации вы сможете использовать Linux в качестве маршрутизатора, на манер Cisco Catalyst, 3Com: [Corebuilder, Netbuilder II, SuperStack II switch 630], Extreme Ntwks Summit 48, Foundry: [ServerIronXL, FastIron].

Отличный HOWTO по организации VLAN:
http://scry.wanfear.com/~greear/vlan/cisco_howto.html.

Включено в состав ядра, начиная с версии 2.4.14 (или может быть 13).

Альтернативная реализация 802.1Q VLAN для Linux

Этот проект был начат из-за разногласий, возникших в 'официальном' проекте VLAN, связанных с архитектурными решениями и стилем кодирования.

Linux Virtual Server

Команда блестящих разработчиков. Linux Virtual Server -- высоко масштабируемый и очень перспективный сервер который основан на кластере из реальных серверов, с равномерным распределением нагрузки, под управлением операционной системы Linux. Архитектура кластера прозрачна для конечных пользователей, которые 'видят' только один-единственный виртуальный сервер.

Короче говоря, LVS предоставляет возможность равномерного распределения нагрузки при любом объеме трафика. Некоторые из решений, реализованных этой командой, очень необычны! Например, они позволяют нескольким машинам иметь один и тот же IP-адрес, при этом протокол ARP на них отключается. ARP работает только на головной LVS-машине, которая решает какому из внутренних хостов должен быть передан тот или иной входящий пакет, и передает его по MAC-адресу внутреннего сервера. Исходящий трафик передается маршрутизатору напрямую, а не через головную LVS-машину, благодаря чему для нее нет нужды отслеживать весь исходящий трафик, который по своему объему может быть просто ужасающим!

LVS реализован в виде "заплаты" на ядра версий 2.0 и 2.2. Для серий 2.4/2.5 -- в виде модуля к Netfilter, т.е накладывать "заплату" на ядра этих версий уже не требуется!

CBQ.init

Конфигурирование CBQ можно немного упростить, особенно в том случае, если вам

необходимо лишь сформировать трафик для нескольких компьютеров, стоящих позади маршрутизатора. CBQ.init поможет вам в этом, благодаря упрощенному синтаксису.

Например, если необходимо ограничить скорость загрузки, величиной 28 Кбит/сек, для всех компьютеров в локальной сети 192.168.1.0/24 (на 10 Мбитном eth1), поместите эти строки в файл конфигурации CBQ.init:

```
DEVICE=eth1,10Mbit,1Mbit
RATE=28Kbit
WEIGHT=2Kbit
PRIO=5
RULE=192.168.1.0/24
```

Мы рекомендуем эту программу тем, кого не интересуют вопросы "как" и "почему". CBQ.init давно используется нами и зарекомендовал себя с самой лучшей стороны. Здесь приведен очень простой пример конфигурирования CBQ.init, на самом деле он может много больше, например выполнять формирование трафика в зависимости от времени. Описание находится внутри сценария, по этой причине вы не найдете файл README.

Набор скриптов управления трафиком Chronox

Стефан Мюллер (Stephan Mueller smueller@chronox.de) написал два сценария `limit.conn` и `shaper`, первый из которых позволяет легко ограничить ширину канала для одиночной сессии, например так:

```
# limit.conn -s SERVERIP -p SERVERPORT -l LIMIT
```

Работают с ядрами 2.4/2.5.

Второй сценарий более сложен, и может использоваться для создания нескольких очередей, основываясь на метках пакетов, устанавливаемых **iptables**.

Реализация протокола Virtual Router Redundancy Protocol (ссылка 1, ссылка 2)

FIXME: Эта ссылка "битая". Может кто подскажет -- куда переехал проект?

Применяется исключительно для "горячего" резервирования. Две машины, имеющие свои собственные IP и MAC адреса, образуют третий, виртуальный IP и MAC адрес. Изначально этот протокол предназначался для "горячего" резервирования маршрутизаторов, которые требуют наличия постоянного MAC-адреса, но вполне подойдет и для серверов другого типа.

Вся прелесть этой реализации заключается в простоте настройки и отсутствии необходимости пересборки ядра.

Просто, на каждой из машин запускается команда, например такая:

```
# vrrpd -i eth0 -v 50 10.0.0.22
```

и все! Теперь адрес 10.0.0.22 обслуживается одним из серверов, вероятнее всего тем, на котором команда **vrrpd** была запущена первой. Теперь, если этот сервер отключить от сети, то довольно быстро обслуживание виртуальных IP и MAC адресов возьмет на себя один из оставшихся компьютеров.

Я попытался смоделировать эту ситуацию. Правда по необъяснимой причине у меня сбрасывался шлюз по-умолчанию, но флаг -n решил мои проблемы.

Ниже приводится временная диаграмма "сбойной" ситуации:

```
64 bytes from 10.0.0.22: icmp_seq=3 ttl=255 time=0.2 ms
64 bytes from 10.0.0.22: icmp_seq=4 ttl=255 time=0.2 ms
64 bytes from 10.0.0.22: icmp_seq=5 ttl=255 time=16.8 ms
64 bytes from 10.0.0.22: icmp_seq=6 ttl=255 time=1.8 ms
64 bytes from 10.0.0.22: icmp_seq=7 ttl=255 time=1.7 ms
```

НИ ОДИН ICMP-пакет не был потерян! После передачи 4-го пакета мой P200 был отключен от сети и обслуживание виртуальных адресов тут же перехватил 486-й, это можно наблюдать по увеличившемуся времени отклика.

tc-config

tc-config -- это набор сценариев для конфигурирования подсистемы управления трафиком на Linux 2.4+ для Red Hat. В качестве корневой дисциплины используется CBQ qdisc, в "листьях" -- SFQ qdisc.

Включает утилиту **snmp_pass**, которая получает статистику управления трафиком через snmp. FIXME: Необходимо дополнить.

Глава 19. Рекомендуемая литература.

http://snafu.freedom.org/linux2.2/iproute-notes.html

Содержит большое количество технических сведений и комментариев к исходным текстам ядра.

http://www.davin.ottawa.on.ca/ols/

Джамал Хади Салим (Jamal Hadi Salim), один из авторов Linux traffic control.

http://defiant.coinet.com/iproute2/ip-cref/

HTML-версия документа, написанного Алексеем -- очень подробное описание части iproute2.

http://www.aciri.org/floyd/cbq.html

Неплохая страничка Салли Флойд (Sally Floyd), посвященная CBQ, содержит оригинальные документы, написанные ее рукой. Не относится к специфике Linux, но основным предметом дискуссии является теория применения SBQ. Можно рекомендовать тем, кого интересуют сугубо технические сведения.

Differentiated Services on Linux

Этот документ (созданный Вернером Альмсбергером (Werner Almesberger), Джамалом Хади Салимом (Jamal Hadi Salim) и Алексеем Кузнецовым) описывает возможности DiffServ в ядре Linux, в том числе **TBF**, **GRED**, **DSMARK qdisc** и классификатор **tcindex**.

http://ceti.pl/~kravietz/cbq/NET4_tc.html

Еще один HOWTO, но только на польском языке! Здесь вы найдете неплохие примеры,

поскольку командная строка выглядит абсолютно одинаково на любом языке человеческого общения! Автор этого документа уже вступил с нами в контакт и возможно вскоре станет нашим соавтором!

IOS Committed Access Rate

От замечательных ребят из Cisco, которые имеют привычку, достойную похвалы, выкладывать документацию в он-лайн. Синтаксис Cisco совершенно иной, но концепция остается неизменной. Кроме того, мы можем делать и делаем больше и без роутеров, цена которых сопоставима с ценой автомобиля. :-)

Экспериментальный сайт с документацией

Стеф Коэн (Stef Coene) убедил своего босса в необходимости поддержки Linux, поэтому он очень много экспериментирует, особенно с настройками пропускной способности. Его сайт содержит большое количество практической информации, примеров, тестов, а так же описывает некоторые ошибки, встречающиеся в CBQ/tc.

TCP/IP Illustrated, volume 1, W. Richard Stevens, ISBN 0-201-63346-9

Рекомендуется к прочтению всем, кому необходимо глубокое понимание TCP/IP. Кроме того -- очень занимательная книга.

От переводчика: *(А.К.) Отличный документ той же направленности, но на русском языке: TCP/IP крупным планом.*

Policy Routing Using Linux, Matthew G. Marsh, ISBN 0-672-32052-5

Введение в политики маршрутизации с огромным количеством примеров.

Internet QoS: Architectures and Mechanisms for Quality of Service, Zheng Wang, ISBN 1-55860-608-4

Книга в твердом переплете, охватывает темы, связанные с Quality of Service (Качеством Обслуживания). Доступно описывает базовые концепции.

Глава 20. Благодарности.

Цель этой заключительной главы -- перечислить всех, кто внес свою лепту в этот HOWTO, или помог нам прояснить те или иные моменты. Здесь мы хотим выразить свою признательность всем, кто так или иначе оказал нам свою помощь.

- Junk Alins

  <juanjo@mat.upc.es>

- Joe Van Andel

- Michael T. Babcock

  <mbabcock@fibrespeed.net>

- Christopher Barton

<cpbarton%uiuc.edu>

- Peter Bieringer

  <pb:bieringer.de>

- Adam Burke

  <aburke%crg.ee.uct.ac.za>

- Ard van Breemen

  <ard%kwaak.net>

- Ron Brinker

  <service%emcis.com>

- Lukasz Bromirski

  <l.bromirski@mr0vka.eu.org>

- Lennert Buytenhek

  <buytenh@gnu.org>

- Esteve Camps

  <esteve@hades.udg.es>

- Ricardo Javier Cardenes

  <ricardo%conysis.com>

- Stef Coene

  <stef.coene@docum.org>

- Don Cohen

  <don-lartc%isis.cs3-inc.com>

- Jonathan Corbet

  <lwn%lwn.net>

- Gerry N5JXS Creager

  <gerry%cs.tamu.edu>

- Marco Davids

  <marco@sara.nl>

- Jonathan Day

  <jd9812@my-deja.com>
- Martin aka devik Devera

  <devik@cdi.cz>
- Hannes Ebner

  <he%fli4l.de>
- Derek Fawcus

  <dfawcus%cisco.com>
- David Fries

  <dfries%mail.win.org>
- Stephan "Kobold" Gehring

  <Stephan.Gehring@bechtle.de>
- Jacek Glinkowski

  <jglinkow%hns.com>
- Andrea Glorioso

  <sama%perchetopi.org>
- Thomas Graf

  <tgraf%suug.ch>
- Sandy Harris

  <sandy%storm.ca>
- Nadeem Hasan

  <nhasan@usa.net>
- Erik Hensema

  <erik%hensema.xs4all.nl>
- Vik Heyndrickx

  <vik.heyndrickx@edchq.com>
- Spauldo Da Hippie

<spauldo%usa.net>

- Koos van den Hout

  <koos@kzdoos.xs4all.nl>

- Stefan Huelbrock <shuelbrock%datasystems.de>

- Ayotunde Itayemi

  <aitayemi:metrong.com>

- Alexander W. Janssen <yalla%ynfonatic.de>

- Andreas Jellinghaus <aj%dungeon.inka.de>

- Gareth John <gdjohn%zepler.org>

- Dave Johnson

  <dj@www.uk.linux.org>

- Martin Josefsson <gandalf%wlug.westbo.se>

- Andi Kleen <ak%suse.de>

- Andreas J. Koenig <andreas.koenig%anima.de>

- Pawel Krawczyk <kravietz%alfa.ceti.pl>

- Amit Kucheria <amitk@ittc.ku.edu>

- Pedro Larroy

  <piotr%member.fsf.org>

    - Глава 15, раздел 10: Пример подключения локальной сети к Интернет через NAT, с организацией QoS

    - Глава 17, раздел 1: Настройка OSPF в Zebra

- Edmund Lau <edlau%ucf.ics.uci.edu>

- Philippe Latu <philippe.latu%linux-france.org>

- Arthur van Leeuwen <arthurvl%sci.kun.nl>

- Jose Luis Domingo Lopez

  <jdomingo@24x7linux.com>

- Robert Lowe

  <robert.h.lowe@lawrence.edu>

- Jason Lunz <j@cc.gatech.edu>
- Stuart Lynne <sl@fireplug.net>
- Alexey Mahotkin <alexm@formulabez.ru>
- Predrag Malicevic <pmalic@ieee.org>
- Patrick McHardy <kaber@trash.net>
- Andreas Mohr <andi%lisas.de>
- James Morris <jmorris@intercode.com.au>
- Andrew Morton <akpm%zip.com.au>
- Wim van der Most
- Stephan Mueller <smueller@chronox.de>
- Togan Muftuoglu <toganm%yahoo.com>
- Chris Murray <cmurray@stargate.ca>
- Takeo NAKANO <nakano@apm.seikei.ac.jp>
- Patrick Nagelschmidt <dto%gmx.net>
- Ram Narula <ram@princess1.net>
- Jorge Novo <jnovo@educanet.net>
- Patrik <ph@kurd.nu>
- P?l Osgy?ny <oplab%westel900.net>
- Lutz Preъler <Lutz.Pressler%SerNet.DE>
- Jason Pyeron <jason%pyeron.com>
- Rod Roark <rod%sunsetsystems.com>
- Pavel Roskin <proski@gnu.org>
- Rusty Russell <rusty%rustcorp.com.au>
- Mihai RUSU <dizzy%roedu.net>
- Rob Pitman <rob%pitman.co.za>
- Jamal Hadi Salim <hadi%cyberus.ca>
- Ren? Serral <rserral%ac.upc.es>

- David Sauer <davids%penguin.cz>
- Sheharyar Suleman Shaikh <sss23@drexel.edu>
- Stewart Shields <MourningBlade%bigfoot.com>
- Nick Silberstein <nhsilber%yahoo.com>
- Konrads Smelkov <konrads@interbaltika.com>
- William Stearns <wstearns@pobox.com>
- Andreas Steinmetz <ast%domdv.de>
- Matthew Strait <straitm%mathcs.carleton.edu>
- Jason Tackaberry <tack@linux.com>
- Charles Tassell <ctassell%isn.net>
- Jason Thomas <jason5intology.com.au>
- Glen Turner <glen.turner%aarnet.edu.au>
- Tea Sponsor: Eric Veldhuyzen <eric%terra.nu>
- Thomas Walpuski <thomas%bender.thinknerd.de>
- Song Wang <wsong@ece.uci.edu>
- Frank v Waveren <fvw@var.cx>
- Chris Wilson <chris@netservers.co.uk>
- Lazar Yanackiev <Lyanackiev%gmx.net>