В. А. Крищенко, А. О. Крючков

Стек сетевых протоколов TCP/IP: теория и практика компьютерных сетей

Учебное пособие

Черновик от 10 августа 2014 г.

Рассмотрена теория работы компьютерных сетей, использующих стек TCP/IP. Большое количество практических заданий охватывает ключевые протоколы сетевого и транспортного уровня, ряд протоколов прикладного уровня, некоторые моменты канального уровня, а также основы сетевого программирования. Для изучения работы компьютерных сетей используются виртуальные сети, объединяющие машины с операционной системой Debian GNU/Linux.

Для студентов, обучающихся по направлениям «Информатика и вычислительная техника» и «Программная инженерия».

Авторы: Крищенко Всеволод Александрович, к.т.н., доцент кафедры «Программное обеспечение ЭВМ и информационные технологии» (ИУ-7) МГТУ им. Н. Э. Баумана; Крючков Алексей Олегович, магистр, выпускник МГТУ им. Н. Э. Баумана. Сообщить авторам своим предложения и замечания можно эл. почтой на адрес mstu@sevik.ru.

Содержание

Обозначения и сокращения

ARP — протокол поиска сетевых адресов (англ. *Address Resolution Protocol*).

CIDR — безклассовая IP-маршрутизация на основе масок сетей (англ. *Classless Inter-Domain Routing*).

DHCP — протокол динамической конфигурации сетевых настроек компьютера (англ. *Dynamic Host Configuration Protocol*).

DNS — система серверов доменных имён, а также используемый ими протокол (англ. *Domain Name System*).

HTTP — протокол передачи гипертекста (англ. *Hypertext Transfer Protocol*).

ICMP — протокол служебных межсетевых сообщений (англ. *Internet Control Message Protocol*).

IEEE — Институт инженеров электротехники и электроники (англ. *Institute of Electrical and Electronics Engineers*).

IP — межсетевой протокол передачи данных (англ. *Internet Protocol*).

ISO — Международная организация по стандартизации (англ. *International Organization for Standardization*).

MAC — управление доступом к среде передачи данных (англ. *Media access control*).

MTU — максимальный размер полезного блока данных на канальном уровне (англ. *Maximum Transfer Unit*).

OSI — сетевой стек, разрабатываемый под эгидой ISO (англ. *Open Systems Interconnection*).

RFC — документы, описывающие стандарты, рекомендации и предложения по сетевому стеку TCP/IP (англ. *Request for Comments*).

POP — протокол получения эл. почты клиентом (англ. *Post Office Protocol*).

RIP — протокол обмена маршрутной информацией (англ. *Routing Information Protocol*).

SMTP — протокол передачи почты почтовому серверу (англ. *Simple Mail Tranfer Protocol*).)

STP — протокол построения связующего дерева портов коммутаторов (англ. *Spanning Tree Protocol*).

TCP — транспортный протокол с механизмами управлением передачей (англ. *Transfer Control Protocol*).

TTL — оставшееся время жизни IP-пакета (англ. *time to live*).

UDP — протокол негарантированной доставки коротких сообщений (англ. *User Datagram Protocol*).

VPN — виртуальная частная сеть англ. *Virtual private network*.

Введение

Понимание принципов и механизмов функционирования современных компьютерных сетей важно как для специалистов, связанных с их настройкой и эксплуатацией, так и программистов, связанных с разработкой любого сетевого программного обеспечения. Изучение сетевых протоколов стека TCP/IP должно сопровождаться практическими занятиями с компьютерной сетью, без которых весь теоретический материал будет выглядеть далёким от жизни. Практические занятия также позволяют убедится в отсутствии рассогласований между изложенной теорией и практикой современных компьютерных сетей.

В данном пособии теоретический материал тесно связан с практическими заданиями, что позволяет на практике закрепить теоретические знания и проверить их корректность. Первая глава пособия является кратким введением в сети TCP/IP, а все остальные главы включают как теоретический материал, так и практические задания. Каждая глава охватывает свой фрагмент сетевого стека TCP/IP и завершается выводами и списком контрольных вопросов. Пособие охватывает вопросы от основ канального уровня до протоколов прикладного уровня и может быть использовано и как основа для проведения лабораторных работ, так и база для создания лекционного курса.

К сожалению, в качестве платформы для практических занятий студентов зачастую трудно использовать реальные физические компьютеры и сетевое оборудование. При рассмотрении большинства протоколов их можно было бы заменить виртуальными сетями, но применение для этого средств виртуализации общего назначения (например, VirtualBox) затруднено из-за высоких требований к аппаратным средствам, необходимых для запуска десяти и более виртуальных машин на одном компьютере. К счастью, существует технология, позволяющая создавать легковесные виртуальные машины путём запуска ядра Linux как пользовательского процесса в основной операционной системе, в роли которой тоже должна выступать система на ядре Linux.

Эта технология называется User Mode Linux; для неё разработаны достаточно удобные средства управления виртуальными машинами и быстрого создания виртуальных сетей на их основе. Такие виртуальные машины при желании можно подключить и к реальной сети.

Одной из таких разработок является пакет утилит Netkit, разработанный в третьем университете Рима под руководством профессора Массимо Римондини. Пакет позволяет быстро запустить целую виртуальную сеть из машин на основе операционной системы Debian GNU/Linux на одном компьютере с достаточно скромными характеристиками.

Данное пособие основывается на версии пакета Netkit, используемой на кафедре «Программное обеспечение ЭВМ и информационные тех-

нологии» (ИУ7) в МГТУ им. Н. Э. Баумана. Для выполнения заданий пособия необходимо скачать и установить эту версию на свой компьютер. Подробнее о настройке рабочего места написано в приложении А. В приложении Б подробно описаны команды пакета Netkit, а в приложении В приведена справочная информация о программе tcpdump, используемой в пособии для перехвата сетевого трафика.

Основная часть пособия организована следующим образом. Первые две глав являются вводными: глава 1 пособия кратко и упрощённо описывает назначение основных протоколов TCP/IP и даёт картину работы компьютерной сети в целом, а глава 2 знакомит читателя с используемым симулятором компьютерных сетей и основами протокола IP. После двух вводных глав начинаются главы с подробной теорией и практическими занятиями по основным протоколам стека TCP/IP. Глава 3 подробно рассматривает сетевой протокола IPv4 и IP-маршрутизацию. Глава 4 рассматривает канальный уровень и проблему построения его логической топологии с помощью протокола STP. В главе 5 рассмотрены основы транспортного уровня и протокол UDP. В главе 6 описано использование динамической маршрутизации на основе протокола RIP. Глава 7 посвящена основам протокола и использованию сетевых сокетов, а глава 8 рассматривает подробно работу протокола TCP. В главе 10 рассмотрены вопросы работы локальных сетей, включая трансляцию сетевых адресов, фильтрацию сетевого трафика и создание виртуальных частных сетей. В главе 11 рассмотрена организация системы DNS, а в главе 12 — организация почтовой системы на основе протокола SMTP. Глава 9 рассматривает основы протокола HTTP.

Все главы заканчиваются вопросами для контроля изученного материала. В главах 3, 4, 6, 10 и 12 приведены варианты индивидуальных заданий. Преподаватель может выдать эти варианты студентам для самостоятельного выполнения после того, как они разобрались в основном материале главы.

Читатель должен иметь представление об основах программирования на языке Си, структурах данных и системах счисления, графах и конечных автоматах, азах архитектуры ЭВМ, основных понятиях теории операционных систем. Предполагается знакомство читателя с командным интерфейсом Unix-подобных систем. Общее представление об организации таких систем и их пользовательских и программных интерфейсов можно почерпнуть, например, из [1]. Для заполнения отчета желательно минимальное представление о системе подготовки текстов Latex [**?**]. Читателю необходимо также иметь минимальные навыки чтения английского текста.

Пособие может быть, при желании преподавателя, использовано и без заданий, касающихся основ сетевого программирования. Это может быть разумно для аудитории, не имеющей навыков программирования на языке Си, или если вопросы сетевого программирования рассматриваются в другом курсе. Следует отметить, что данное пособие не ставит целью

привить навыки сетевого администрирования — использование программ управления сетью и настроек сетевых служб сокращено до требуемого минимума, а в разделах контроля знаний нет посвящённым им вопросов. Тем не менее, в начале глав с практическими занятиями указываются программы и службы, встречающиеся в ней впервые.

Авторы хотели бы выразить свою благодарность Короткову Ивану Андреевичу, а также всем студентам, высказывающим свои замечания и пожелания по курсу лабораторных работ и лекций, которые авторы проводили в МГТУ им. Н. Э. Баумана.

1 Основы компьютерных сетей

Перед тем, как начать детально и подробно изучать конкретные сетевые протоколы, мы дадим некоторые основные определения, а также рассмотрим общую и достаточно упрощённую картину работы компьютерной сети на основе стека протоколов TCP/IP.

1.1 Проблема обмена информацией между процессами на различных ЭВМ

В жизни часто удобна возможность обмена информацией между процессами, запущенными на удалённых друг от друга ЭВМ. Например, одним из таких процессов может быть веб-браузер на компьютере студента, а вторым — веб-служба на сервере МГТУ им. Н. Э. Баумана. Для обмена информации между такими процессами необходима *компьютерная сеть*.

Сеть передачи данных (компьютерная сеть) — совокупность линий связи, специализированного сетевого оборудования, компьютеров и программного обеспечения, используемая для автоматической передачи цифровой информации между процессами, запущенными на удалённых друг от друга компьютерах.

Сигнал — физический процесс, несущий информацию. Физики ещё в XIX веке открыли несколько способов передачи информации, которые могут использоваться в компьютерной сети. Носителем сигнала в электрических цепях является электрический ток, в оптоволоконных кабелях — луч света, а в пространстве — радиоволны

Среда передачи данных — физическая субстанция, по которой происходит передача электрических или электромагнитных сигналов, использующихся для передачи информации. В случае проводной сети средой является кабель (электрический или оптический), в случае беспроводной — любая среда, не препятствующая распространению волн.

Сами по себе среда и сигнал передачи данных не могут решить задачу обмена информацией между двумя запущенными процессами. От передачи сигнала надо сделать много шагов до создания компьютерной сети, позволяющей читать студенту информацию на веб-сервере. Такую сложную задачу имеет смысл разделить на меньшие так, чтобы каждая полученная подзадача была относительно небольшой. Формальное описание решения каждой подзадачи передачи данных оформляется в виде описания *сетевого протокола*.

Сетевой протокол — совокупность соглашений о формате и правилах обмена *сообщениями*. Составляющие протокол соглашения описываются в документе, называемом *спецификацией протокола*.

Сообщение сетевого протокола — минимальная единица обмена информацией для данного протокола. Это определение означает, что протокол не рассматривает случай «получена половина сообщения».

Процесс передачи информации между двумя удалёнными ЭВМ традиционно разделяют на несколько уровней. Самый верхний уровень будет реализован в прикладной программе, а самый нижний будет связан с физической средой передачи данных. Тогда протокол каждого уровня, кроме самого «нижнего», будет использовать один или несколько «нижестоящих» протоколов. В качестве довольно близкой аналогии можно взять работу с файлами, когда каждый уровень (аппаратный, драйвер устройства, файловая подсистема ОС, прикладная программа) решает какую-то свою задачу и взаимодействует с другими через некоторый интерфейс.

Стек сетевых протоколов — совокупность связанных сетевых протоколов различных уровней, обеспечивающих решение задачи передачи информации в компьютерной сети. Вышестоящие протоколы стека используют сервисы, предоставляемые нижестоящими протоколами. Для решения проблемы передачи информации между процессами, таким образом, необходимо разработать и реализовать весь стек сетевых протоколов.

Спецификация протокола содержит полный и непротиворечивый набор указаний, достаточный для создания реализации протокола и обычно включает в себя описание следующие моментов.

1) Назначение протокола и область его применения, предоставляемый протоколом сервис.
2) Используемые нижестоящие протоколы, требования к их сервису, или используемая среда передачи данных.
3) Формат сообщений: соглашения о двоичном представлении сообщения (числе битов в байте, последовательности байтов в слове) или соглашения о используемой кодировке символов или её задании.
4) Словарь (конечное множество) возможных видов сообщений протокола.
5) Правила обмена сообщениями протокола, включая алгоритмы обработки и создания сообщений.
6) Соглашения об адресации: для идентификации сторон во многих протоколах используются адреса из некоторого множества.

Первые два пункта определяют место протокола в сетевом стеке, третий и четвертый пункты определяют возможные сообщения и их представление. Далее мы увидим как протоколы с достаточно сложными алгоритмами работы, так и весьма простыми.

Отметим, в спецификацию протокола не входит спецификация программного интерфейса, через который реализация одного протокола взаимодействует с реализацией другого в пределах одного компьютера. В спецификации лишь указывается, какую информацию должен получить протокол «сверху» для своей работы. Поскольку две вполне соответствующие одинаковому набору спецификаций реализации сетевого стека могут корректно взаимодействовать даже при разных межуровневых интерфейсах,

то их включение спецификацию протоколов, вероятно, даже было бы вредным. В главах 5 и 7 мы познакомимся с единственным подобным стандартизованным интерфейсом.

В обмене сообщениями в протоколе участвуют две или более *стороны* — далее мы увидим примеры, когда сторонами являются процессы, ядро ОС или сетевые устройства, в зависимости от уровня протокола. В спецификации протокола может быть и не указано конкретное «воплощение» стороны, чтобы не сужать возможности реализации.

Часть протоколов предполагает, что для общения по протоколу стороны должны установить *соединение*. Наличие соединения означает, что весь обмен сообщениями между двумя сторонами можно разделить на три фазы: установка соединения, основной обмен сообщениями, завершение соединения. В главе 8 мы увидим наиболее характерный пример такого протокола. Другие протоколы обходятся без понятия соединения, разрешая передачу и приём любых сообщений протокола в произвольный момент времени. Далее мы увидим многочисленные примеры таких протоколов.

Ряд протоколов используют большее, чем три, число фаз обмена, состояние стороны для них описывается довольно объёмным конечным автоматом. В таких протоколах со сложным жизненным циклом состояния стороны часто также часто можно выделить установку и разрыв соединения, хотя и на этапе основного сообщения не все сообщения являются допустимыми в некоторый момент времени. В главе 12 мы увидим характерные примеры таких протоколов.

1.2 Стек сетевых протоколов TCP/IP

В настоящее время основным используемым стеком протоколов является сетевой стек TCP/IP [2], создание которого началось ещё в 70-ые годы XX века по заказу министерства обороны США.

При разработке стека TCP/IP были выделены четыре следующих уровня передачи информации между процессами (снизу вверх, рисунок 1.1).

1) Канальный уровень: передача данных между сетевыми адаптерами в одном *сегменте сети*, например, по технологии Ethernet. Сегмент сети формально будет определён в разделе 1.4.

2) Сетевой уровень (протокол IP): передача данных между компьютерами в разных сегментах.

3) Транспортный уровень: передача данных между процессами на разных компьютерах. Существует два основных транспортных протокола — с установкой соединения (протокол TCP) и без неё (протокол UDP).

4) Прикладной уровень: «полезные» протоколы, ради которых сеть передачи данных и создавалась (например, протокол HTTP, используемый веб-браузером и веб-сервером).

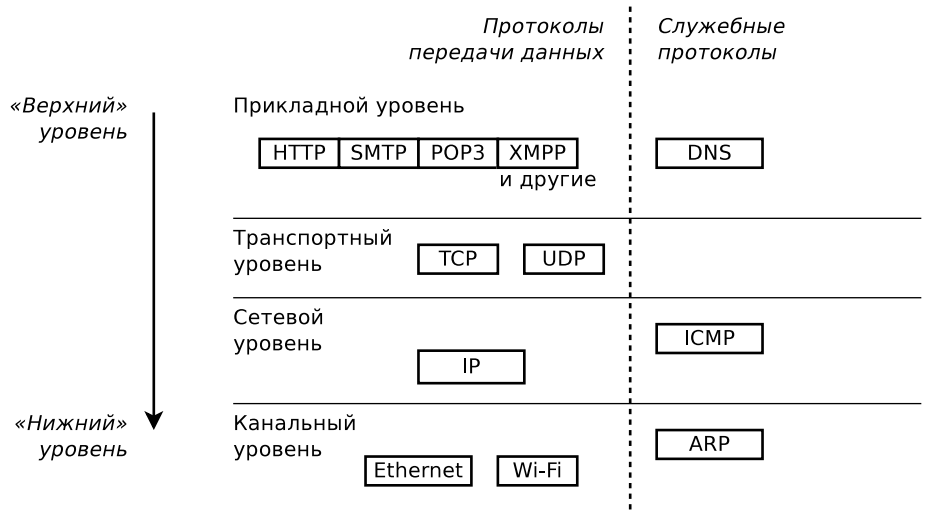


Рисунок 1.1 — Уровни и основные протоколы стека TCP/IP

Интернет — совокупность всех связанных компьютерных сетей, использующих протоколы стека TCP/IP. Отметим, что хотя название протокола IP и расшифровывается как *Internet Protocol*, на момент его создания это означало просто «межсетевой протокол».

Спецификации большинства рассматриваемых протоколов описаны в документах, известных как RFC (англ. *Request for Comments*, дословно «запрос комментариев»). Все протоколы стека TCP/IP (рисунок 1.1) можно разделить на две группы *протоколы передачи данных*, передающие полезные данные между двумя сторонами, и *служебные протоколы*, необходимые для корректной работы сети.

Как протоколы передачи данных (за исключением канального уровня), так и служебные протоколы используют некоторый протокол передачи данных для передачи своих сообщений. При этом в случае служебного протокола это протокол может относится и к тому же уровню стека, а не к нижестоящему: например, служебный протокол ICMP использует сетевой протокол передачи данных IP. Почему же протокол ICMP не отнесли к более высокому уровню? Причина проста: для корректной работы протокола IP компьютеру нужно, как мы увидим в дальнейшем, иметь возможность отравлять ICMP-сообщения, а при разработке стека было бы разумно, чтобы нижние уровни «не знали» про верхние.

Уже на примере взаимной связи протоколов IP и ICMP можно сказать, что разделение на уровни в стеке TCP/IP не носит совершенного строго характера. Насколько можно судить, разработчиков стека TCP/IP более волновал вопрос работоспособности сетевого стека, нежели вопрос его полной идеологической чистоты и формализации терминологии, поэтому в

нём легче ответить на вопросы «для чего нужен» или «как работает», чем на вопрос «что такое». В разделе 1.11 мы упомянем и попытку разработки сетевого стека, проходившую под флагом иной расстановки приоритетов.

Сетевой и транспортный уровни стека TCP/IP обычно реализованы программно на уровне операционной системы, канальный — на уровне сетевого оборудования, к которому относятся рассматриваемые далее коммутаторы, мосты, точки доступа, а также сетевые адаптеры и их драйверы. Протоколы прикладного уровня реализуются прикладными программами, в том числе системными службами. Отметим, что граница между понятиями «компьютер» и «сетевое оборудование» весьма размыта: в настоящее время почти любое подключаемое к электропитанию сетевое оборудование, за исключением сетевых адаптеров и простых коммутаторов, является по сути компактным компьютером с ЦП, ОЗУ, ПЗУ, операционной системой (обычной, например на основе ядра Linux, или специализированной) и набором прикладных программ.

В главе 10 мы увидим, что уровни стека TCP/IP могут быть связаны и более сложным образом, поскольку можно создать протокол прикладного уровня, который может быть использован как некоторый виртуальный канальный уровень.

1.3 Сообщения протоколов передачи данных

Для краткости далее мы будем использовать краткие названия сообщений протоколов стека TCP/IP. Так, сообщения канального уровня принято называть *кадрами*, а сетевого уровня — *пакетами*. Сообщения транспортного протокола TCP называют *сегментами*[1], сообщения протокола UDP — *датаграммами*.

Сообщения протоколов трёх нижних уровней стека TCP/IP состоят из заголовка, содержащего служебную информацию, и *полезной нагрузки* — некоторых передаваемых по протоколу данных. Данными обычно являются сообщения протоколов более высокого уровня (рисунок 1.2).

Несколько упрощая, можно сказать, что сообщения протокола передачи данных одного уровня ложатся в протоколы нижнего уровня в качестве полезной нагрузки. Максимальный размер кадра канального уровня ограничен сверху возможностями сетевой аппаратуры. Максимальный размер IP-пакета ограничен полезной нагрузкой кадра. Транспортный протокол TCP ликвидирует это неудобство, позволяя организовать передачу упорядоченного набора байт произвольного размера. Это позволяет прикладным протоколам передавать сообщения произвольного размера.

На рисунке 1.2 показано, что происходит с данными, передаваемыми от одного процесса другому по протоколам стека TCP/IP. Процессом-

[1]Сегменты протокола TCP не имеют, разумеется, никакого отношения к сегментам сети. Данная коллизия терминов, к сожалению, имеет место быть и в оригинальной английской терминологии.

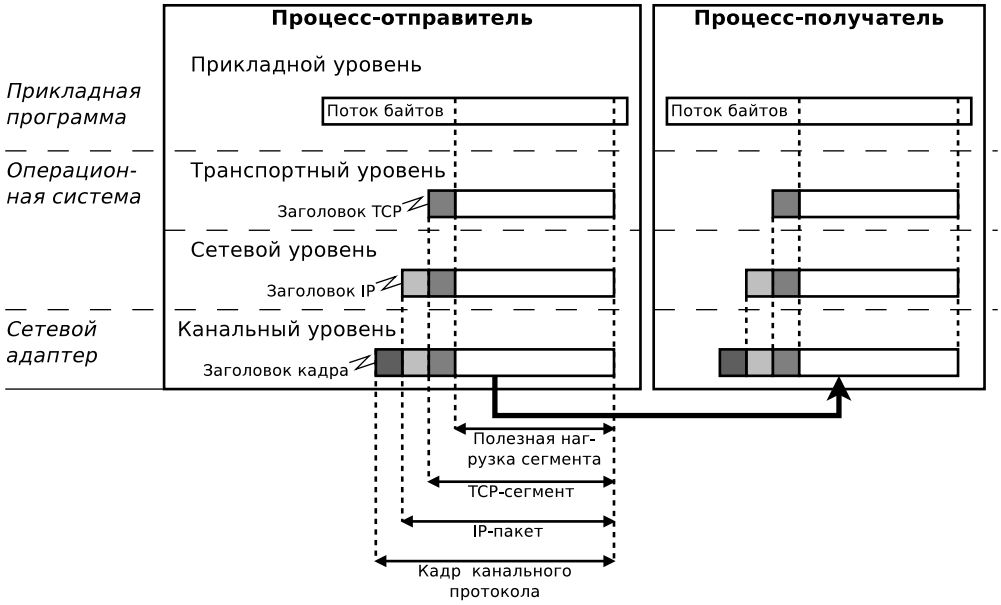


Рисунок 1.2 — Передача данных между двумя процессами по стеку TCP/IP

отправителем может быть, например, всё тот же веб-сервер, отправляющий по протоколу сообщение с веб-страницей внутри, а процессом-получателем — веб-браузер. Для взаимодействия веб-браузера и веб-сервера служит протокол HTTP.

Отметим, что веб-сервером называют как ПО и процесс для обработки запросов веб-браузеров, так и компьютер, где такой процесс выполняется. Здесь и далее под сервером некоего протокола будет пониматься процесс его сетевой службы, запущенный на некотором компьютере.

С точки зрения транспортного уровня передаваемое сообщение является некоторой упорядоченной последовательностью байтов, а не просто массивом байт в памяти. Дело в том, что сервер передаёт HTTP-сообщение операционной системе не обязательно целиком, а некоторыми частями: ведь внутри сообщения HTTP может находиться не только небольшой HTML-документ, но и большое изображение, и даже образ DVD-диска, поэтому HTTP-сообщение, возможно, даже не содержится целиком в памяти процесса-отправителя. Такую упорядочную последовательность байтов, которая передаётся некоторыми порциями, часто называют *потоком*.

1) Передаваемые программой-отправителем данные делятся на части на уровне протокола TCP, каждая часть заключается в один сегмент протокола TCP. Сегмент, таким образом, содержит полезную полезную нагрузку — фрагмент потока — и заголовок.

2) Сегменты TCP помещаются в IP-пакеты в качестве их полезной нагрузки, каждый сегмент — в один пакет. Пакет также имеет собственный заголовок.

3) IP-пакеты затем помещаются в кадры канального уровня в качестве их полезной нагрузки, каждый пакет — в свой кадр. Кадры также начинаются с собственного заголовка.

4) Кадры передаются сетевому адаптеру и в итоге преобразуются в физические сигналы, передаваемые по среде передачи данных.

Принимающая сторона последовательно извлекает полезную нагрузку из сообщений канального, сетевого и транспортного уровня, и в итоге данные из сегмента TCP занимают своё место в полученном наборе байт. Поскольку на стороне получателя полученные данные проходят уровни в обратном по сравнению с отправителем порядке, то стек сетевых протоколов действительно напоминает именно стек как структуру с дисциплиной LIFO.

Здесь, конечно, мы очень упрощённо описали отправку информации HTTP-сервером. В главе 7 мы познакомимся с тем, как использующие TCP прикладные программы передают сообщения прикладного протокола операционной системе, а в главе 9 рассмотрим протокол HTTP более подробно. Далее в этой главе мы рассмотрим кратко уровни стека протоколов TCP/IP от нижних к верхним.

1.4 Канальный уровень

Назначением протоколов канального уровня (англ. *link layer*) является передача небольших блоков данных между машинами, соединённых, в простейшем случае, одной средой передачи данных. Данные передаются единожды: канальный уровень не повторяет передачу испорченных или потерянных кадров, это задача других уровней.

Протоколы этого уровня обычно реализуются сетевыми устройствами. Для подсоединения компьютера к среде передачи используется устройство, называемое *сетевым адаптером* или сетевой картой.

Для идентификации сетевого адаптера при передаче ему кадра обычно нужно использовать некоторый адрес. Исключением являются протоколы канального уровня, служащие для соединения друг с другом ровно двух сетевых адаптеров: примером такого протокола может служить протокол PPP (англ. *Point-to-Point Protocol*).

Адреса устройств канального уровня называются *MAC-адресами* (англ. *Media Access Control* — контроль доступа к среде передачи). MAC-адрес состоит из шести байт, которые принято записывать в шестнадцатеричной форме через двоеточие, например: aa:bb:c0:56:78:90.

Операционная система представляет подключение к некоторому сегменту сети в виде *сетевого интерфейса*.

Сетевой интерфейс — это абстрактное сетевое устройство канального уровня, за работу которого отвечает ядро операционной системы. Сетевой интерфейс имеет единственный MAC-адрес и соответствует либо

физическому устройству (сетевому адаптеру), либо некоторому виртуальному устройству — реализация такого устройства может быть самой разнообразной. В некоторых ОС используется термин *сетевое подключение* в качестве синонима сетевого интерфейса.

Типичным примером среды передачи данных является медный кабель, содержащий четыре скрученные пары жил диаметром 0,5 мм каждая. Поскольку кабель состоит из четырёх скрученных «косичками» медных пар, он также называется «витая пара». Такой кабель используется для сетей Ethernet различных стандартов.

Мы уже определили канальный уровень как уровень передач в пределах сегмента сети, но ещё не определили последний термин. Связано это с тем, что его развитие было достаточно сложным. Изначально под сегментом сети понималось множество узлов сети, имеющих доступ к одной и той же среде передачи данных, то есть, в случае проводной сети на основе витой пары, имеющих прямое электрическое соединение. Стандарты организации IEEE используют именно это определение, но стандарты RFC его не используют. Далее мы будем называть такой сегмент *физическим сегментом сети*.

Как мы увидим ниже, такие устройства, как коммутаторы и мосты позволяют создать сегмент из машин, не имеющих прямого электрического соединения, а главе 10 мы увидим создание виртуального сегмента сети. Поэтому, с точки зрения вышестоящего сетевого уровня, определение сегмента сети не связано уже с разделяемой физической средой, нам видимо ничего не остаётся, как связать в единое целое понятия канального уровня и сегмента сети.

Сегмент сети — совокупность соединённых с помощью сетевых интерфейсов машин, между которыми может быть передано сообщение канального уровня. Все сетевые интерфейсы в пределах сегмента должны иметь уникальные MAC-адреса. Сегмент сети может совпадать с физическим сегментом, или состоять из нескольких физических сегментов, объединённых оборудованием канального уровня, или быть виртуальным, то есть не имеющим прямого отношения к среде передачи данных. К сегменту сети относится и всё сетевое оборудование, обеспечивающее соединение сетевых интерфейсов.

Отметим, что стандарты RFC определяют сегмент как то, к чему подключены сетевые интерфейсы. Это определение соответствует данному выше, но определяет сегмент по его назначению, а не по его составу.

В пределах сегмента (и только в пределах сегмента) может работать *широковещание* (англ. *broadcast*), когда один отправленный кадр приходит ко всем подключенным к сегменту машинам.

На канальном уровне можно выделить несколько решаемых задач. Во-первых, протокол канального уровня должен решить задачу преобразования цифровой информации в физический сигнал. Для этого отправ-

ляемая последовательность нулей и единиц кодируется так, чтобы свести возможные искажения при передаче к минимуму, а затем передаётся с использованием модуляции сигнала.

Во-вторых, при передаче сигнала в среде, одновременный доступ к которой имеют сразу несколько машин (например, в беспроводных сетях или в сетях старого стандарта Ethernet-10Mbit), нужно организовать контроль доступа к среде (англ. *media access control*, MAC, отсюда и термин «MAC-адрес») так, чтобы сигналы не накладывались друг на друга.

В-третьих, ряд служебных протоколов канального уровня решает задачу управления избыточными линиями связи в пределах сегмента и переключения на запасные линии при повреждении основной. В главе 4 мы увидим, как решается эта задача с помощью протокола STP.

Почти каждый протокол канального уровня решает приведённые задачи по-своему. Детальное рассмотрение протоколов канального уровня и проблем передачи информации можно найти, например, в стандартах группы IEEE 802 [3], описывающих проводные протоколы Ethernet и беспроводные протоколы Wi-Fi. К счастью, для остальных уровней стека TCP/IP не важно, как именно канальный уровень решает свои задачи, поэтому мы уделим ему сравнительно мало внимания.

Из наиболее распространенных «проводных» протоколов канального уровня можно отметить Fast Ethernet и Gigabit Ethernet со скоростью передачи данных 100 Мбит/с и 1 Гбит/с соответственно. Более поздние технологии семейства Ethernet достигают скоростей в 10, 40 и 100 Гбит/с (и называются соответствующим образом). В современных протоколах Ethernet используются как медные кабели, так и оптоволокно.

Современное беспроводное (Wi-Fi) оборудование стандарта 802.11n имеет теоретическую скорость передачи данных 600 Мбит/с, предыдущего стандарта 802.11g — 54 Мбит/с. Практическая скорость передачи ограничивается аппаратными возможностями компьютеров и эффективностью программного обеспечения.

Сетевой адаптер Gigabit Ethernet стоит на всех современных стационарных компьютерах и ноутбуках, обычно он интегрирован с материнской платой. Адаптер беспроводной сети Wi-Fi (стандарта 802.11g или более современного 802.11n) имеется во всех ноутбуках и мобильных устройствах. Сетевой адаптер подключается к шине PCI или USB, на серверах — к шине PCI-E. Кабель витой пары соединяет два сетевых адаптера. Чтобы несколько ЭВМ объединить в один сегмент сети Ethernet, необходимо иметь устройство, которое имеет несколько *портов* стандарта Ethernet. Порт канального уровня является точкой подключения сетевого кабеля к сетевому оборудованию канального уровня. Отметим, что порты коммутаторов не имеют MAC-адресов — в этом нет нужды, поскольку коммутатор не является конечным получателем пакетов, а только пересылает их дальше.

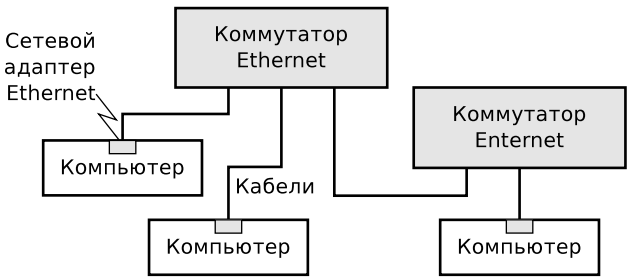


Рисунок 1.3 — Соединение коммутаторов и сетевых адаптеров в один сетевой сегмент

Концентратор (англ. *hub*) — сетевое устройство для объединения нескольких машин с проводными сетевыми адаптерами в один физический сегмент сети. Поскольку в случае концентратора сегмент сети является физическим сегментом, то только одна машина в нём может передавать данные в любой момент времени, а сигналы с двух машин будут накладываться. Сети стандартов Fast Ethernet и более поздних не используют концентраторы в принципе: здесь они упомянуты в качестве исторического примера оборудования, создающего большой проводной физический сегмент с коллизиями при передаче данных.

Коммутатор (англ. *switch*) — сетевое устройство для объединения нескольких машин с проводными сетевыми адаптерами в один сегмент сети на основе пересылки кадров. В отличие от концентратора, коммутатор не создаёт единого физического сегмента: каждая машина независимо отправляет коммутатору кадр, который он затем пересылает получателю.

Коммутаторы могут быть соединены друг с другом, как показано на рисунке 1.3, где сегмент сети состоит из трёх машин с адаптерами и двух коммутаторов. Коммутатор принимает и перенаправляет Ethernet-кадры их адресату, выбирая из своих портов тот, через который достижим адресат. В случае широковещания пакет будет отправлен на все порты, кроме того, через который он пришёл. В главе 4 мы разберёмся, как же коммутаторы связывают MAC-адреса получателей со своими портами.

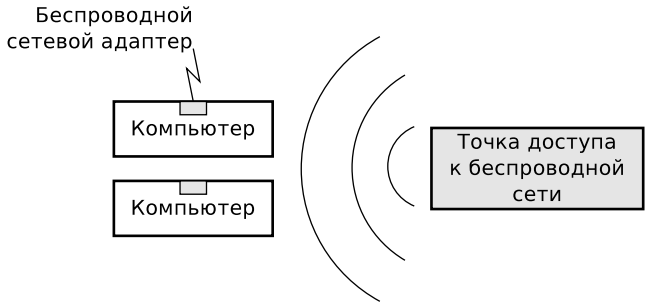


Рисунок 1.4 — Сегмент беспроводной сети

В случае беспроводных сетей базовым оборудованием является *точка доступа* к беспроводной сети (рисунок 1.4). Отметим, что в случае

беспроводной точки доступа сегмент сети вполне совпадает с физическим сегментом.

В один сегмент сети могут быть объединены устройства и адаптеры различных протоколов канального уровня. Для этого необходимо иметь устройство, называющееся *мостом*.

Мост — bridge.устройство, объединяющее несколько сетевых сегментов в один сегмент Для такого объединённого сегмента, конечно же, остаётся в силе требование уникальности MAC-адресов.

Типичная точка доступа беспроводных устройств является на самом деле мостом: она имеет как адаптер Ethernet, так и беспроводной адаптер Wi-Fi, и объединяет два сегмента двух различных протоколов в один. В итоге объединённый сегмент сети на рисунке 1.5 состоит из кабельного фрагмента, беспроводного фрагмента и соединяющего их моста.

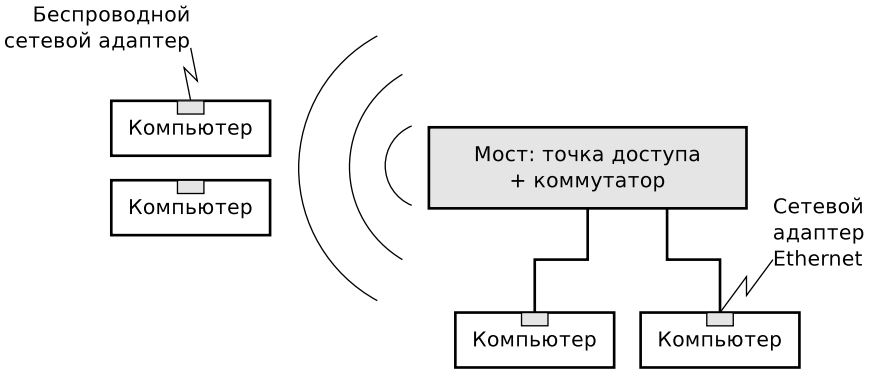


Рисунок 1.5 — Объединение в один сегмент проводной и беспроводной сети

Отметим, что в определении моста не требуется, чтобы объединяемые сегменты непременно относились к различным протоколам канального уровня, и с точки зрения данного определения коммутатор можно рассматривать и как мост, что пригодится в главе 4.

Кадр типичного канального уровня делится на две части — область данных и заголовок, в котором присутствуют, в частности, поля для адреса отправителя и адреса получателя кадра, а также длина передаваемой полезной нагрузки. Величина максимального объёма полезных данных в одном кадре называется MTU (Maximum Transmission Unit). Она зависит от конкретного канального уровня, достаточно типичным её значением будет 1500 байт. Минимальное стандартное значение MTU для сетей TCP/IP — 576 байт [4]. Для работы моста, соединяющих два сегмента в один, необходимо, чтобы величины MTU у подключённых к нему сегментов совпадали.

1.5 Сетевой уровень: протокол IPv4

Канальный уровень решает задачу передачи в пределах одного сегмента сегмента. Задачей протоколов сетевого уровня (англ. *internet layer*)

является передача данных между различными сегментами сети. В стеке TCP/IP таких протоколов два: протокол IP версии 4 (кратко обозначается как IPv4) и протокол IP версии 6 (IPv6). Протокол IPv4 морально устарел, но наиболее активно эксплуатируется, поэтому мы рассмотрим его и далее будем называть его просто «протокол IP».

Используемый протоколом IPv4 адрес состоит из четырёх байтов, которые принято записывать в виде четырёх десятичных чисел через точку, например: 127.0.0.1 (старшие байты — слева).

Разумно как-то объединить все IP-адреса компьютеров, находящихся в одном сегменте сети: это позволит понять, относятся ли два адреса к одному и тому же сегменту или нет, ведь в первом случае задача передачи данных уже решена канальным уровнем. Для этого IP-адрес делится на две части (рисунок 1.6):

– адрес сети: старшая (левая) часть адреса;
– адрес машины[1] в сети: младшая (правая) часть адреса.

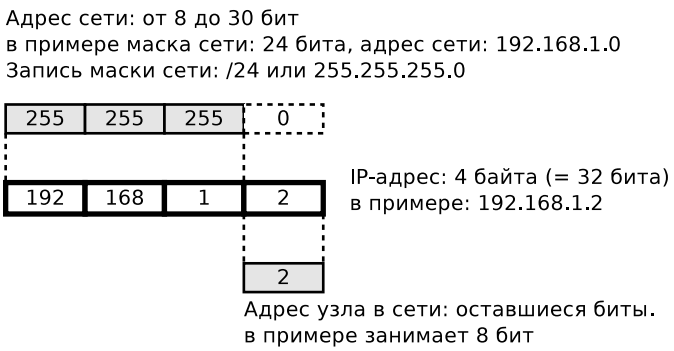


Рисунок 1.6 — Структура IP-адреса

Для указания границы между этими частям нужно указать количество битов, ушедших на адрес сети. Это количество называется *маской* IP-сети. Записать маску можно двумя способами:

– первый способ записи IP-адреса с маской: 192.168.1.2/24 (24 старших бита отводится для сети);
– второй способ записи: IP-адрес 192.168.1.2, маска сети 255.255.255.0; в этой маске старшие 24 разрядов — единицы, младшие 8 разрядов — нули, единичные биты в маске показывают часть адреса, относящуюся к сети.

Обе эти записи означают одно и тоже: в IP-адресе 192.168.1.2 старшие 24 бит (здесь это «192.168.1») уходят на адрес сети, а оставшиеся 8 бит (для данного адреса это «2») — на адрес узла. IP-сеть в данном понимании обычно (в том числе в данном пособии) совпадает с сегментом сети

[1] Используют также термин *адрес хоста*. Кроме того, *хостом* часто называют любой компьютер с IP-адресом, как правило — не являющийся маршрутизатором.

канального уровня, хотя, вообще говоря, связь между ними может быть и более сложной.

Когда нужно указать адрес некоторой сети, то он просто дополняется нулями до 32-х бит. Корректный адрес машины (правая часть IP-адреса) не может быть равен нулю, поэтому такие адреса, как 192.168.1.0/24 или 10.0.0.192/28 могут обозначать только сеть в целом, а не конкретный адрес машины.

В отличие от MAC-адреса, адрес сетевого уровня в идеале должен идентифицировать компьютер во всём мире. С другой стороны, если мы хотим организовать сеть из нескольких машин дома, то хотелось бы назначать в ней IP-адреса как можно проще, не ставя в известность международные организации, контролирующие их распределение.

Для этой цели было выделено несколько групп так называемых *частных IP-адресов*[1]: это сети 192.168.0.0/16, 10.0.0.0/8 и 172.16.0.0/12. Кроме того, любой адрес из сети 127.0.0.0/8 считается адресом той же самой машины, с которой обращаются по данному адресу.

У этого решения есть явный недостаток: раз в мире может быть сколько угодно машин с адресом, например, 192.168.1.2, то как же передать информацию для процесса на такой машине через интернет? Ответ на этот вопрос мы узнаем в разделе 1.9.

Если IP-адреса получателя и отправителя принадлежат одной сети, то отправитель может просто послать получателю кадр с нужным IP-пакетом внутри. Для того, чтобы передавать данные между разными IP-сетями, нам необходим *маршрутизатор*.

Маршрутизатор (англ. *router*) — компьютер или устройство с двумя или более сетевыми адаптерами с IP-адресами в разных IP-сетях, передающее данные между ними. Как мы видим, IP-маршрутизатор будет иметь несколько адресов в разных IP-сетях. Например с одним адаптером может быть связан адрес и маска 10.0.10.1/24, а с другим — 10.0.20.1/24.

Учитывая, что IP-сеть определена в пределах сегмента сети, можно отметить, что маршрутизатор связывает и два сегмента при помощи протокола сетевого уровня. На пути от отправителя до получателя IP-пакета может находится множество маршрутизаторов и сетевых сегментов. Один шаг передачи IP-пакета по этому пути, охватывающий передачу в пределах одного сегмента, мы назовём *маршрутизацией*.

Маршрутизация IP-пакета (англ. *IP routing*) — определение адреса очередного получателя IP-пакета и последующая передача пакета очередному получателю в кадре канального уровня.

Маршрутизацией, по данному определению, занимаются все машины, поддерживающие протокол IP. Далее мы будем для краткости называть маршрутизацию IP-пакетов просто «маршрутизацией». На пути от отправи-

[1] Частные адреса также иногда называют *серыми IP-адресами*.

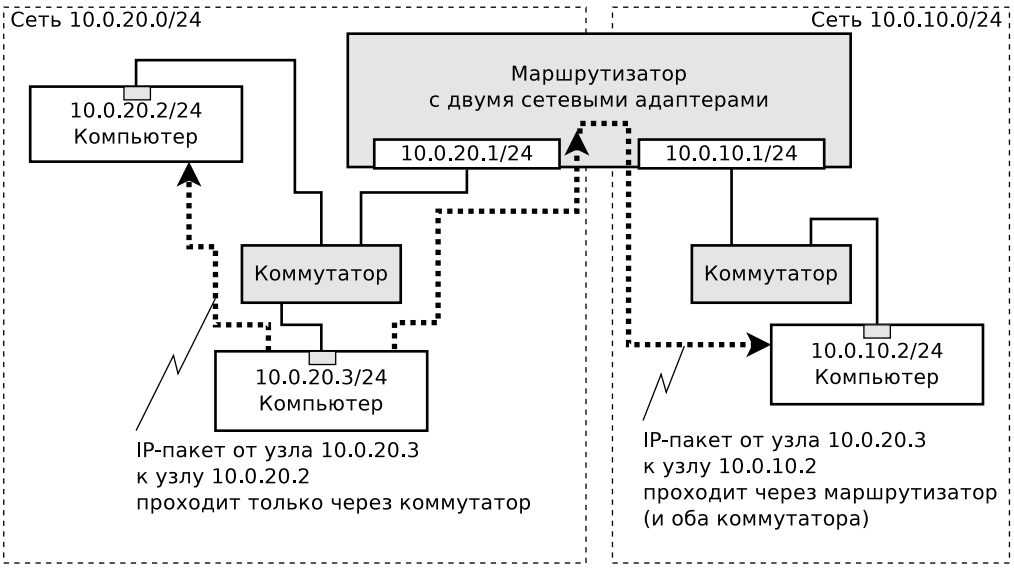


Рисунок 1.7 — Передача IP-пакета с использованием и без использования маршрутизатора

теля к получателю маршрутизацией занимается отправитель и все маршрутизаторы, через которые проходит пакет.

Можно выделить два разных случая маршрутизации, иногда называемых прямой и косвенной маршрутизацией. В первом случае, при *прямой маршрутизации*, конечный получатель IP-пакета находится в том же сегменте сети, что и отправитель: на рисунке 1.7 это случай передачи пакета от узла 10.0.20.3 к узлу 10.0.20.2, а также передача маршрутизатором кадра с пакетом от 10.0.20.3 к узлу 10.0.10.2. Во втором случае, при *косвенной маршрутизации*, получатель не находится в непосредственно подключённом сегменте, и пакет необходимо передать посреднику — маршрутизатору. На рисунке 1.7 это случай соответствует передаче маршрутизатору кадра с пакетом для узла 10.0.10.2 от узла 10.0.20.3. Отметим, что на пути IP-пакета косвенная маршрутизация может происходить несколько раз, а прямая — только на последнем этапе, при передаче получателю.

Если адрес сети узла назначения пакета совпадает с сетью, к которой подключён отправитель, то это значит, что мы можем отправить пакет непосредственно получателю, если узнаем его MAC-адрес. В противном случае нам нужно получить MAC-адрес маршрутизатора и послать пакет ему. Таким образом, и при прямой, и при косвенной маршрутизации нам нужно знать MAC-адрес следующего получателя кадра[1].

Для получения MAC-адреса по известному IP-адресу в пределах сегмента сети используется протокол разрешения адресов ARP [5] (англ. *Address Resolution Protocol*, (глава 2), который посылает широковещательный запрос всему сегменту сети и затем ждёт ответ от обладателя искомого IP-адреса (рисунок 1.8). Отметим, что в ходе маршрутизации IP-адрес

[1]За исключением случая канального уровня, не использующего MAC-адреса в принципе.

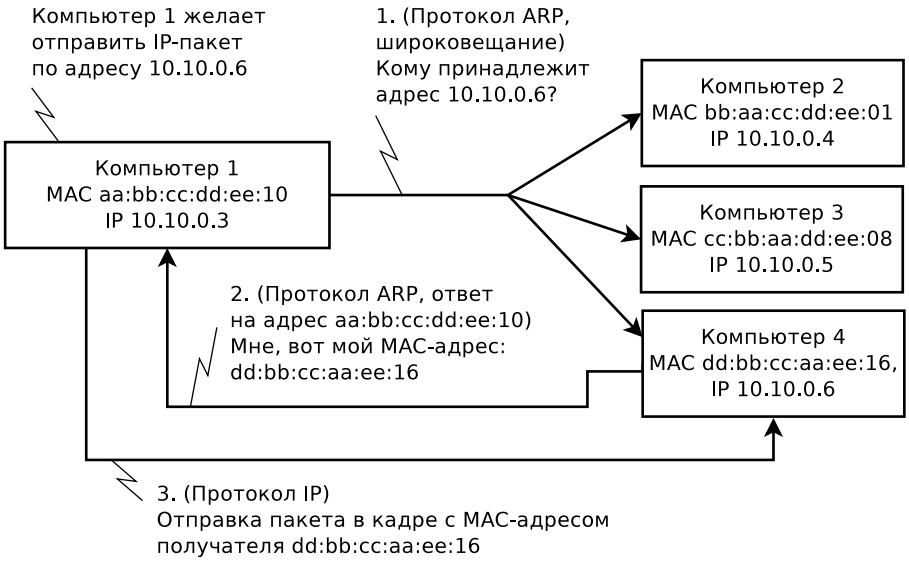


Рисунок 1.8 — Использование протокола ARP для поиска MAC-адреса

получателя в пакете не меняется: для указания получателя кадра на очередном шаге маршрутизации используется MAC-адрес получателя в кадре канального уровня.

Обычным компьютерам для маршрутизации обычно достаточно знать IP-адрес маршрутизатора в своём сегменте, которому они и передают все пакеты для машин вне этого сегмента. Маршрутизаторы же должны иметь таблицу соответствия адресов сетей и адресов маршрутизаторов, которым они и будут пересылать пакеты для получателей из этой сети. Эти таблицы и проблемы их составления рассмотрены в главах 2, 3 и 6.

Как было отмечено ранее, величина MTU сегмента сети определяет максимальный размер IP-пакета, который может быть передан поверх канального уровня. Что же делать маршрутизатору, если он подключен к сегментам с разной величиной MTU, и пришедший через один сетевой интерфейс пакет не помещается в кадр другого интерфейса? К счастью, протокол IP допускает фрагментацию пакетов, когда один пакет разбивается на два или более мелких пакета-фрагмента. При обсуждении сетевого (глава 3) и транспортного (глава 8) уровней мы увидим и другие решения этой же проблемы, поскольку фрагментация имеет ряд крупных недостатков.

Отметим, что некоторые источники дают иные определения маршрутизации: например, как только косвенную маршрутизацию или как определение всех маршрутизаторов на пути к получателю, или только определение очередного маршрутизатора, но не пересылку пакета и т.д. К счастью, из контекста обычно всегда ясно, какое определение маршрутизации подразумевается в том или ином случае. Однако, если вы услышите на собеседовании вопрос «что такое IP-маршрутизация?», то сложно сказать, какое из многочисленных существующих определений ожидается от вас,

поскольку в стандартах RFC это термин используется, но не определяется формально.

1.6 Транспортный уровень

Задачей протоколов транспортного уровня (англ. *transport layer*) является организация обмена данными между произвольными процессами, выполняющимися на разных машинах. У каждого настроенного сетевого адаптера компьютера есть IP-адрес, но нужно ещё как-то указать, что данные передаются для некоторого конкретного процесса на этом компьютере. Для этого на транспортном уровне была введена дополнительная сущность — *сетевой порт*[1].

Сетевой порт (или порт транспортного уровня, для определённости) является двухбайтовым беззнаковым числом. Заголовок сообщения транспортного уровня содержит порт отправителя и порт получателя. Таким образом, соединение между двумя процессами определяется IP-адресами получателя и отправителя и номерами их портов. Пару из IP-адреса и номера порта можно считать адресом транспортного уровня, часто адрес и порт записывают через двоеточие (например, 195.19.50.247:80).

Чтобы одной программе передать данные другой, ей нужно знать IP-адрес машины, где запущена программа-получатель, и порт, с которым она связана. Чтобы вторая задача решалась просто, с большинством протоколов прикладного уровня связаны стандартные номера портов. Например, с прикладным протоколом HTTP, по которому работают веб-браузеры и веб-серверы, связан порт 80 протокола TCP. Конечно, при использовании протокола HTTP может использоваться и иной порт сервера, число 80 лишь является значением по умолчанию. Полный список типовых значений портов можно посмотреть в файле **services**, который есть на всех типовых настольных ОС.

На рисунке 1.9 условно показан процесс обмена данными по транспортному протоколу между двумя процессами. Один из них ожидает приход данных на некотором сетевом порту. Этот процесс (и соответствующую ему программу) называют сервером[2]или *сетевой службой*.

Процесс на другой машине — клиент — выступает инициатором обмена. Он отправляет данные на IP-адрес и порт сервера, используя транспортный протокол. Для получения данных от сервера с процессом-клиентом так же должен быть связан порт, который выделяется из числа свободных на данной ЭВМ. В отличие от портов сервера, которые обычно фиксированы, порт клиента просто выбирается из свободных.

---

[1] Порт протокола транспортного уровня не следует путать, например, с портами коммутатора или портами ввод-вывода.

[2] Обычно из контекста ясно, идёт ли речь идёт о программе или о компьютере.

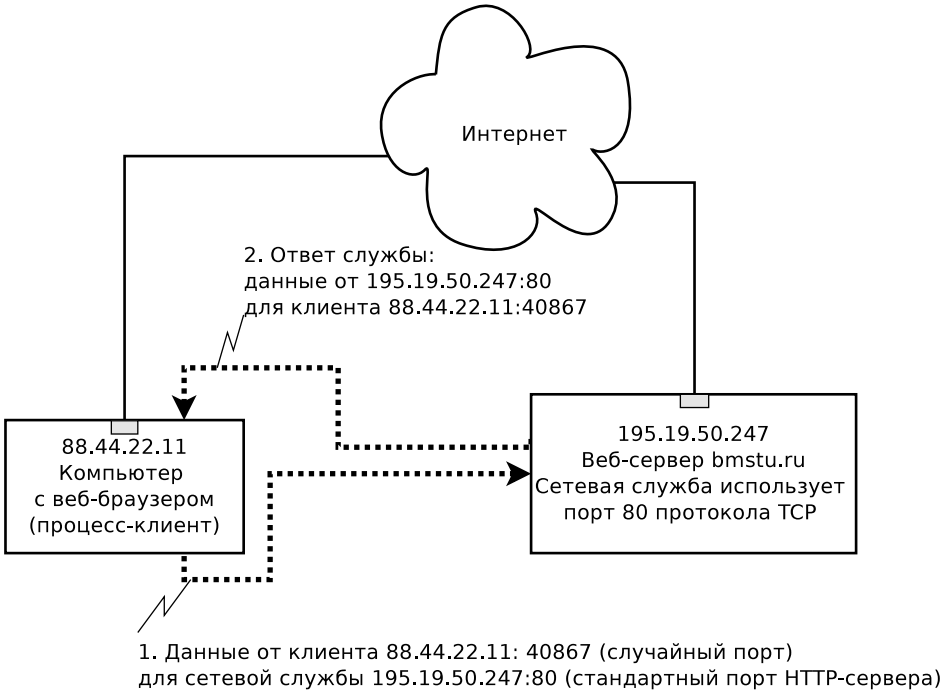


Рисунок 1.9 — Сетевая служба и её клиент

В стеке TCP/IP есть два основных транспортных протокола: TCP и UDP. Протокол UDP (англ. *User Datagram Protocol*) очень прост: он служит для передачи небольших фрагментов данных, размер которых на практике не должен превышать примерно пол-килобайта, чтобы они могли поместиться в сеть с наихудшим значением MTU. Каждая датаграмма помещается ровно в один IP-пакет, который либо доходит до получателя, либо теряется, причём протоколу UDP его судьба безразлична. Подробнее протокол UDP рассмотрен в главе 5.

Протокол TCP (англ. *Transmission Control Protocol*) весьма сложен, поскольку позволяет надежно передать поток данных, при этом пытаясь не допустить перегрузки сети. Свою работу протокол TCP начинает с установки соединения между клиентом и сервером. После установки соединения начинается передача данных. Передаваемые данные делятся на сегменты так, чтобы каждый сегмент умещался в один IP-пакет. С целью обеспечения надежной передачи протокол TCP ожидает подтверждения переданных сегментов. Если оно не приходит за некоторое время или по иным причинам становится ясно, что сегмент был потерян, то он передается заново. По сигналу от прикладной программы TCP-соединение разрывается (в нормальном случае — по инициативе клиента).

В протокол TCP входят алгоритмы регулирования темпа передачи данных. Поскольку пропускная способность сетей и маршрутизаторов на пути до получателя неизвестна, не стоит сразу передавать все данные, а лучше начать с малого и при получении подтверждения увеличивать число сегментов, которые можно послать до ожидания подтверждения. Каждая

из сторон обмена так же сообщает, какой объём данных она готова принять без исчерпания памяти, выделенной для TCP-соединения в ядре ОС: процесс-получатель может перестать считывать данные из ядра, в результате чего буфер в ядре будет исчерпан. В главе 8 мы подробно рассмотрим эти механизмы протокола TCP.

1.7 Служба доменных имён

Служебный протокол прикладного уровня DNS (англ. *Domain Name Service*) и использующая его служба доменных имён имеет очень важное значение для функционирования интернета в целом. Здесь мы кратко рассмотрим его применение, а в главе 11 мы подробно расскажем об организации системы DNS.

Как мы уже знаем, клиентскому процессу для соединения с процессом-сервером по протоколу TCP или UDP нужно знать его порт и IP-адрес его машины. К сожалению, люди плохо запоминают четырехбайтные числа, и для решения проблемы запоминания адресов машин еще в 70-ые годы были введены символьные имена машин. Первоначально они просто записывались в файле **hosts** — вы и сейчас можете найти его во всех ОС для настольных компьютеров. Найдя и просмотрев этот файл на своём компьютере, вы можете узнать, что имя **localhost** связано с адресом 127.0.0.1, и в настоящее время содержимое этого файла часто этим и ограниченно. Стоит отметить, что компьютерные вирусы иногда связывают в файле **hosts** адреса веб-сайтов антивирусов с адресом 127.0.0.1.

По мере роста числа машин интернета использование файла **hosts** для хранения имен всех машин стало невозможным, и был создан служебный протокол прикладного уровня DNS. В настоящий момент в интернете для решения проблемы преобразования мнемонических имён в IP-адреса развернута иерархическая система серверов доменных имен, использующих этот протокол. Каждый из таких серверов имеет зону ответственности: например, сервер имён с IP-адресом 195.19.32.2 отвечает за домен **bmstu.ru**, сервер с адресом 193.232.128.6 — один из многих серверов, отвечающих за зону **ru**, а сервер с адресом 198.41.0.4 является одним из корневых, т.е. главных серверов в иерархии (рисунок 1.10). Такие серверы называют *авторитетными серверами имён*.

Компьютеру пользователя для работы достаточно знать адрес *кеширующего сервера имён*: такой DNS-сервер по запросу DNS-клиента будет опрашивать авторитетные серверы имён, начиная, если понадобится, с корневых. При этом он будет кешировать полученную от авторитетных серверов информацию с целью уменьшения числа запросов к ним в будущем. На рисунке 1.10 показан наихудший случай поиска адреса домена **bmstu.ru**, когда опрос начинается с корневого сервера имён. Служба DNS использует протокол UDP, порт 53.

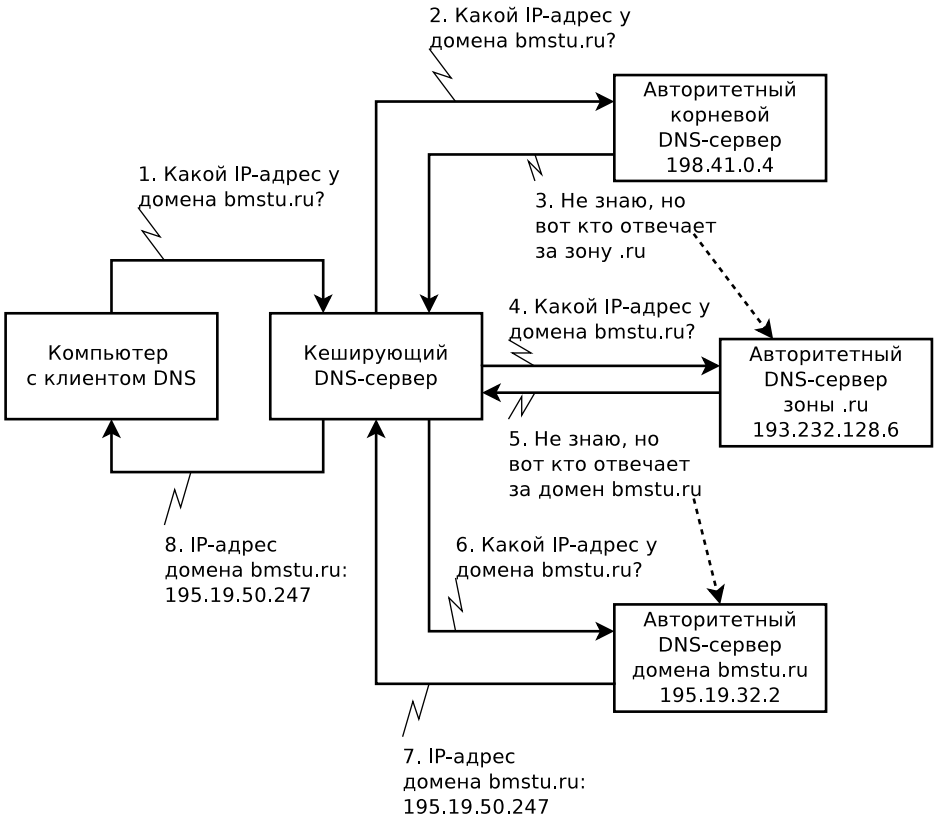


Рисунок 1.10 — Работа служба доменных имён

Отметим, что DNS-клиент, кеширующий сервер и авторитетные серверы используют один и тот же протокол для общения друг с другом, причём кеширующий сервер сам выступает в роли клиента при общении с авторитетными серверами. Система DNS позволяет не только преобразовывать имя в IP-адрес, но и выполнять обратное преобразование, а также хранить иную информацию, как мы увидим в главе 11.

1.8 Протоколы прикладного уровня

Протоколы прикладного уровня (англ. *application layer*) чрезвычайно многочисленны и разнообразны. Кроме того, спецификации ряда протоколов (например, протокола программы Skype) не описаны в открытых источниках.

Протокол HTTP используется для передачи информации между веб-серверами и веб-браузерами: именно этот протокол и его клиенты для просто пользователя стали синонимом слова «интернет». Протокол HTTPS является безопасным вариантом протокола HTTP и позволяет защитить передаваемые данные от перехвата злоумышленниками, поэтому рекомендуется использовать веб-службы, в которых передача паролей и иных личных данных происходит по протоколу HTTPS. В главе 9 мы рассмотрим эти протоколы более подробно.

Протокол SMTP используется для передачи электронной почты от почтового клиента к почтовому серверу и от одного почтового сервера к другому. В главе 12 работа протокола SMTP будет обсуждена подробнее. Для получения почты клиентом существуют протоколы POP3 и IMAP (отметим, что если и получатель, и отправитель почты используют почтовую службу с веб-интерфейсом, то для передачи почты между ними, возможно, используется только протокол SMTP).

Для обмена «мгновенными» сообщениями в настоящий момент существует несколько протоколов: XMPP (клиенты Jabber и ряд других систем), Oscar (протокол системы ICQ), неопубликованный протокол Skype и многие другие.

Надо отметить, что протокол прикладного уровня не обязательно является самым «верхним» протоколом, используемым прикладной программой. Например, поверх протокола HTTP могут работать протоколы SOAP и XML-RPC, созданные для облегчения обмена структурированными данными между программами, а поверх них будет реализован некоторый протокол конкретной информационной системы.

1.9 Локальные сети и преобразование сетевых адресов

Локальная компьютерная сеть (англ. *Local Area Network, LAN*) неформально определяется как компьютерная сеть небольшой организации, офиса, кафедры университета или даже квартиры. Большинство локальных сетей объединяет от пары-тройки до нескольких десятков компьютеров конечных пользователей. Локальная сеть обычно подключёна к интернету через поставщика услуги доступа в интернет (интернет-провайдера) посредством некоторого маршрутизатора. Его сетевой интерфейс, подключённый к локальной сети, и соответствующий IP-адрес принято называть *внутренними*, а интерфейс и адрес, относящиеся к сети провайдера — *внешними*.

Далее мы рассмотрим типичную небольшую локальную сеть, состоящую всего из одного сегмента сети, использующую частные IP-адреса и соединённую с интернетом через единственный маршрутизатор. Отметим, что в общем случае локальная сеть содержит несколько сегментов сети и не обязательно использует только частные адреса.

Как мы знаем, для нормальной работы каждой такой машине нужно присвоить IP-адрес и маску сети, а также указать адрес маршрутизатора (для передачи данных за пределы локальной сети) и кеширующего сервера DNS (для преобразования мнемонических имён в IP-адреса). Даже в небольшой локальной сети число компьютеров может измеряться десятками и меняться во времени (например, если в дом пришли гости со своими телефонами и ноутбуками), поэтому для упрощения задачи сетевой настройки был создан протокол DHCP (англ. *Dynamyc Host Configuration*

Protocol). Благодаря ему DHCP-клиент может получить все необходимые настройки от DHCP-сервера, если таковой существует в локальной сети. Типичный домашний маршрутизатор поэтому выполняет и функцию DHCP-сервера, а также функцию кеширующего сервера доменных имён — чтобы просто сообщить клиентам свой внутренний IP-адрес в качестве используемого DNS-сервера.

В силу многочисленности пользователей локальной сети им в большинстве случаев назначаются частные IP-адреса. Например типичный внутренний адрес маршрутизатора локальной сети — 192.168.1.1/24, а его DHCP-сервер будет выдавать клиентам адреса в диапазоне 192.168.1.2 — 192.168.1.254. Поскольку в мире есть множество сетей с такими адресами, то IP-пакеты с ними в принципе не могут передаваться в интернете, ведь частные адреса решают задачу идентификации только в пределах локальной сети. Несмотря на это, пользователям в локальной сети наверняка хочется обмениваться данными с серверами в интернете и даже запускать сетевые службы на компьютере с частным адресом — скажем, раздавая торренты.

Рисунок 1.11 — Трансляция сетевых адресов

Для разрешения этого противоречия был придуман механизм преобразования сетевых адресов (англ. *Network Address Translation*, NAT), который заменяет в IP-пакете частный адрес клиента на внешний адрес маршрутизатора (рисунок 1.11). Поскольку внешний адрес у маршрутизатора как правило только один, то при такой трансляции меняется и транспортный порт клиента на некоторый, связанный с данным соединением транспортного уровня. Маршрутизатор, выполняющий преобразование сетевых адресов, называют *NAT-маршрутизатором*. В случае протокола UDP, не устанавливающего явного соединения, NAT-маршрутизатор считает соединением активный обмен между парой адресов. При получении пакета-ответа

от сервера NAT-маршрутизатор выполнит обратное преобразование адреса и порта получателя, что позволит доставить пакет клиенту (рисунок 1.11). В результате преобразования адресов клиенты с частными IP-адресами могут обмениваться данными с внешними серверами по протоколам TCP и UDP.

Как видно из описании механизма трансляции, находящиеся за NAT-маршрутизатором машины недоступны как серверы для внешних клиентов, пока доступ к ним открыт явно. Для открытия доступа NAT-маршрутизатор должен пересылать пакеты, пришедшие на некоторый его порт, на некоторый адрес внутри локальной сети. В случае домашней сети для автоматической настройки таких пересылок используется протокол UPnP, с помощью которого сервер может «попросить» маршрутизатор открыть доступ к его TCP-портам.

В типичном случае внешний адрес маршрутизатора не является частным[1]. Однако, в некоторых локальные сетях и внешний адрес маршрутизатора может быть частным, что означает, что и сам интернет-провайдер использует преобразование адресов для своих клиентов: механизм NAT может использоваться многократно на пути IP-пакета.

1.10 Пример работы компьютерной сети

Мы рассмотрели достаточно, чтобы в общих чертах представлять, что происходит в домашней сети от момента подключения компьютера к сетевому кабелю и до посещения пользователем веб-сайта **bmstu.ru**.

Типичная домашняя «коробочка» обычно объединяет в себе NAT-маршрутизатор, DNS-сервер и DHCP-сервер, а также коммутатор Ethernet. Внутренний сетевой адаптер домашнего маршрутизатора обычно имеет IP-адрес типа 192.168.1.1/24 (как в примере далее) или 192.168.0.1/24. После своего включения маршрутизатор получает у интернет-провайдера свой внешний IP-адрес как DHCP-клиент или с помощью специальных сетевых протоколов, таких как L2TP или PPPoE, использующие выданные клиенту имя пользователя и пароль.

Изначально у домашнего компьютера есть только MAC-адрес его сетевого адаптера, в примере на рисунке 1.12 это 00:aa:bb:cc:dd:ee:ff. Затем в компьютер вставляется кабель, соединяющий его с домашним маршрутизатором, точнее, со встроенным в последний сетевым коммутатором. Компьютер получает от локального DHCP-сервера следующую информацию:

– частный IP-адрес и маску сети, в примере это 192.168.1.2/24;
– IP-адрес используемого маршрутизатора (в примере — 192.168.1.1);

[1] Адреса, пригодные для передачи информации в интернете без трансляции адресов, иногда называют для краткости *белыми*.

– IP-адрес кеширующего DNS-сервера, в нашем случае он совпадает с адресом маршрутизатора.

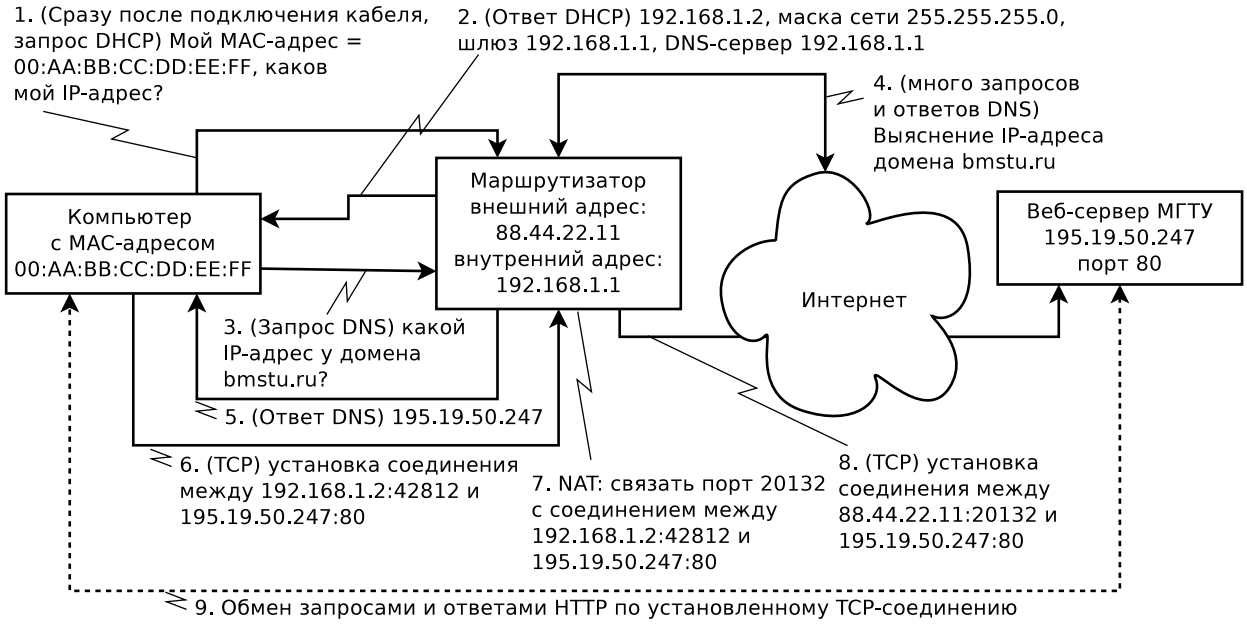


Рисунок 1.12 — Домашняя локальная сеть

Теперь на подключённом к локальной сети компьютере можно вбить в строке веб-браузера адрес **http://bmstu.ru**. Браузер поймет, что префкис **http://** — это указание на соединение по протоколу HTTP, а далее указано доменное имя. Затем браузер отправит кеширующему DNS-серверу запрос на получение IP-адреса доменного имени **bmstu.ru**. Кеширующий DNS-сервер имеет IP-адрес 192.168.1.1. Как мы помним, для передачи данных в сети Ethernet нужно указать MAC-адрес получателя, поэтому до физической передачи DNS-запроса нужно узнать MAC-адрес DNS-сервера по протоколу ARP. Затем компьютер сможет передать пакет с DNS-запросом адреса. Заметьте, что данный DNS-сервер в нашем примере находится в нашей же сети (192.168.1.0/24), и нам достаточно прямой IP-маршрутизации.

Кеширующий сервер DNS, если не найдет этот адрес в своём кеше, будет опрашивать авторитетные серверы имен (как в нашем примере) или кеширующий DNS-сервер провайдера, который, в свою очередь, будет опрашивать авторитетные серверы. В итоге кеширующий сервер опросит авторитетный сервер зоны **bmstu.ru** и получит ответ с адресом имени **bmstu.ru**. При этом будет использоваться IP-маршрутизация и протокол ARP для получения MAC-адреса следующего маршрутизатора на пути к авторитетным серверам.

Получив от кеширующего DNS-сервера ответ с IP-адресом (для имени **bmstu.ru** это адрес 195.19.50.247), веб-браузер соединяется с ним по протоколу HTTP, который работает поверх протокола TCP и использу-

ет порт сервера 80. На этот раз IP-пакеты пойдут от нашей машины с частным IP-адресом на машину с «нормальным» IP-адресом, поэтому наш NAT-маршрутизатор будет осуществлять трансляцию частных адресов в свой внешний адрес.

1.11 Альтернативный сетевой стек

Ключевые протоколы стека TCP/IP, включая протоколы TCP, IPv4, UDP и DNS, были разработаны по заказу Пентагона во второй половине 70-ых — начале 80-ых годов. Начальной целью данной программы, известной как ARPANET, было создание децентрализованной глобальной компьютерной сети, которая уже к концу 70-ых годов охватывала десятки ЭВМ в ведущих университетах США и Великобритании.

К сожалению, мировая организация по стандартизации (англ. *International Organization for Standardization*, ISO) довольно своеобразно отреагировала на успех проекта ARPANET. Вместо того, чтобы рассмотреть возможность стандартизации уже апробированного на практике стека TCP/IP, ответственный комитет ISO приступил к разработке **??** собственного сетевого стека, названного OSI (англ. *Open System Interconection*). Общее описание этого семиуровневого стека известно как «модель OSI/ISO» [6].

Попытка создания сетевого стека OSI была неудачной ([7], стр. 126), от нее остался лишь протокол сетевого уровня CLNP, все еще поддерживаемый рядом производителей. Также получили некоторое распространение термины типа «L2 switch» («L» — от «Layer»), которым иногда называют сетевые коммутаторы, и «L3 switch», которым иногда называют специализированные IP-маршрутизаторы: такая нумерация связана с тем, что семиуровневый стек ISO начинался с физического уровня, отделенного от канального. Эти термины, что интересно, используются именно для оборудования стека TCP/IP некоторыми производителями.

Довольно распространено заблуждение, что семь уровней модели ISO можно как-то связать со стеком TCP/IP. Для понимания его ошибочности (отмеченной, например, в [7], стр. 130) достаточно изучить модель OSI/ISO подробнее и убедиться, что в ней нет места, например, простому транспортному протоколу типа UDP, поскольку в итоге стеке OSI для передачи всегда устанавливается логическое соединение. С другой стороны, в стеке TCP/IP нет прямых аналогов многих базовых понятий стека OSI/ISO, которые описаны в статье [8]. Таким образом, модель OSI/ISO является именно (и только) моделью нереализованного стека OSI/ISO.

Следует отметить, что модель OSI была создана буквально вопреки стеку TCP/IP, как это видно по [8], где даже нет прямого упоминания проекта ARPANET и протоколов стека TCP/IP. В итоге разработка сетевого стека ISO все ещё была далека от завершения к моменту, когда число

машин в интернете начало измеряться сотнями тысяч, и была тихо прекращена в середине 90-ых за явным преимуществом протоколов стека TCP/IP. Несмотря на это, организация по стандартизации ISO так и не признала стек TCP/IP стандартом «де-юре». Отсутствие такого статуса, однако, не мешает стеку TCP/IP быть общепринятым открытым стандартом «де-факто» и поддерживаться многочисленным оборудованием и программным обеспечением различных производителей. Отметим, что, в отличие от общедоступных документов RFC, содержащих спецификации протоколов стека TCP/IP, стандарты протоколов OSI/ISO даже для ознакомления с ними необходимо приобретать за довольно значительную сумму.

1.12 Контрольные вопросы

Для самоконтроля полученных знаний рекомендуется ответить на следующие вопросы.

1) Что такое протокол, сообщение протокола, стек протоколов?
2) Из чего состоит спецификация сетевого протокола?
3) Что такое заголовок и полезная нагрузка?
4) Что такое сегмент сети? Для чего служат протоколы канального уровня?
5) В чем различия между MAC-адресом и IP-адресом?
6) Какую функцию выполняют протоколы канального уровня? Что означает величина MTU сегмента сети?
7) Какую функцию выполняет протокол ARP?
8) Что такое сетевой интерфейс?
9) Какую функцию выполняет протокол IP? Что такое IP-маршрутизация и маршрутизатор?
10) Что такое маска сети и для чего она служит?
11) Какие сообщения называют кадром, пакетом, сегментом?
12) Перечислите все упомянутые в этой главе протоколы передачи данных и все служебные протоколы.
13) Для чего на транспортном уровне стека TCP/IP введено понятие порта?
14) Почему в протоколе TCP используется установка соединения, а протокол UDP — нет?
15) Для чего были выделены диапазоны частных IP-адресов?
16) Какую проблему решает трансляция сетевых адресов и на чём она основана?
17) Для чего служит протокол DNS и сервер DNS?
18) Почему компьютеры пользователей обращаются к кеширующему DNS-серверу, а не обращаются напрямую к авторитетным серверам имён?

2 Простейшая сеть передачи данных

В этой главе мы создадим простейшую сеть из двух виртуальных машин, соединённых виртуальным сегментом протокола Ethernet и присвоим их сетевым интерфейсам адреса. В результате работы должна быть получена сеть передачи данных, показанная на рисунке 2.1.

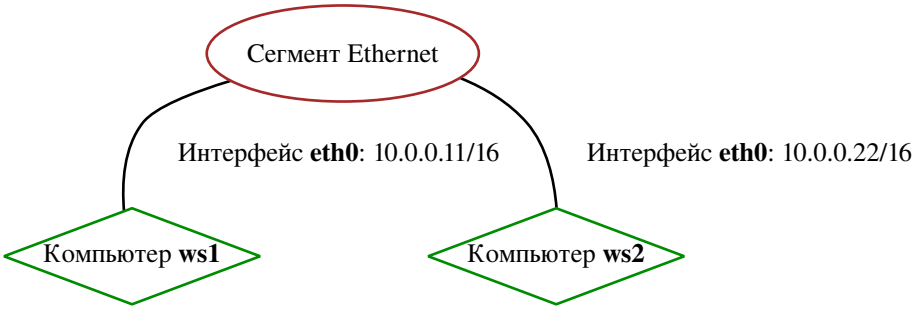


Рисунок 2.1 — Компьютерная сеть из двух машин, подключенных к одному сегменту

Затем мы увидим работу протокола ARP, который позволяет найти MAC-адрес по IP-адресу. Также мы рассмотрим функционирование протокола IP в простейшем случае, когда данные передаются в пределах одного сегмента, и познакомимся с некоторыми сообщениями служебного протокола ICMP.

Для настройки сетевых параметров нам понадобится программа **ip**. Для наблюдения за работой сети мы используем программу перехвата сетевого трафика **tcpdump**. Для создания сообщений протокола ICMP мы воспользуемся программой **ping**.

2.1 Создание и запуск виртуальной машины

Для создания виртуальных сетей мы будем использовать пакет Netkit. Установите его на свой компьютер, как описано в приложении А, если вы этого ещё не сделали. Подробное описание пакета Netkit приведено в приложении Б. Изучать его сейчас нет необходимости: необходимые инструкции будут даваться по ходу изложения.

Приступим к развёртыванию виртуальной сети, для чего откроем эмулятор терминала (Gnome Terminal, Konsole или иной). Существенная часть практических заданий будет связана с работой с командой оболочкой, поэтому при наборе команд всегда используйте автодополнение, где это возможно: при нажатии клавиши «Tab» интерпретатор команд попытается дополнить неполную команду.

Для запуска виртуальной машины в данной главе используется команда **vstart**, основной параметр которой — идентификатор машины. При отсутствии виртуальной машины с указанным идентификатором она будет

создана. Кроме того, при запуске машины следует указать её сетевые интерфейсы и идентификаторы виртуальных сетей, подключённых к ним.

Для создания сети передачи данных, показанной на рисунке 2.1, нужно создать две виртуальные машины, каждую из которых следует запустить в разных терминалах или вкладках. Мы будем использовать префикс «ws» (от англ. *workstation*, «рабочая станция») для имён машин с единственным сетевым интерфейсом, который не играет роль сервера (веб, почтового, доменных имён или иного).

Задание № 1: Запуск виртуальных машин командой vstart

Сначала создадим для работы отдельный каталог (**~/tcp-ip/lab-intro**) и перейдём в него следующими двумя командами.

```
mkdir -p ~/tcp-ip/lab-intro
cd ~/tcp-ip/lab-intro
```

Выполните нижеследующую команду и вы увидите, как виртуальная машина с именем **ws1** начнёт загружаться.

```
vstart ws1 --eth0=networkA
```

В другом сеансе командной оболочке[1] запустите виртуальную машину **ws2**.

```
vstart ws2 --eth0=networkA
```

Имена сетевым интерфейсам даются в разных ОС по разному. В наших виртуальных машинах интерфейсы сети стандарта Ethernet именуются **eth0**, **eth1** и т.д. Интерфейсы нумеруются подряд в пределах одного типа сети на каждой машине.

Как видно по приведенным выше командам, каждая машина будет иметь единственный сетевой интерфейс с именем **eth0**. Сетевые интерфейсы обеих машин будут подключёны к одному и тому же виртуальному сегменту с условным именем **networkA** (это имя имеет смысл только для используемого симулятора сети и никак не связано с реальными протоколами). Обратите внимание: имена подключённых виртуальных сетей могут быть почти произвольными, но должны быть строго совпадающими при запуске разных машин, если их интерфейсы должны быть подключены к одному сегменту сети (и, разумеется, различными, в противном случаю).

Через некоторое время загрузка обеих машин завершится выводом приглашения, для первыой машины оно будет выглядеть как **ws1:~#**, для второй — как **ws2:~#**. Теперь мы сможем работать с интерпретатором команд внутри запущенных виртуальных машин. В дальнейшем все приведенные команды будут отдаваться именно в запущенных виртуальных машинах, если только явно не указано, что они запускаются на основной машине.

2.2 Утилиты для настройки и диагностики сети

В ходе всех работ нам понадобятся различные утилиты для настройки и диагностики сети. К сожалению, набор этих утилит не стандартизован даже в Unix-подобных системах. В «классический» набор сетевых утилит в POSIX-системах, часто называемый net-tools, входят программы **ifconfig**, **route**, **netstat** и другие. Аналогичные команды есть и в ОС Windows, где первая из них называется **ipconfig**.

Современные системы на основе ядра Linux включают пакет утилит iproute2, отличающийся унифицированным синтаксисом команд и призванный заменить классические утилиты. В операционных системах с ядрами, отличными от Linux, этот пакет недоступен, что является главным недостатком пакета iproute2. Таблица 2.1 описывает назначение и соответствие утилит данных пакетов.

Таблица 2.1 — Соответствие утилит из пакетов net-tools и iproute2

Назначение	**net-tools**	**iproute2**
Настройка сетевых интерфейсов	ifconfig	ip addr, ip link
Управление маршрутными таблицами	route	ip route
Управление кешем протокола ARP	arp	ip neigh
Информация об использования сети процессами ОС	netstat	ss

Далее мы будем постоянно использовать утилиту **ip** из пакета iproute2. Кроме того, мы будем использовать утилиту **ping** для посылки пакетов протокола ICMP. Отметим, что ключ **-n** заставляет команды **arp**, **ping** и некоторые другие показывать IP-адреса и MAC-адреса «как есть». В частности, он запрещает искать DNS-имена выводимых на экран IP-адресов и искать производителя сетевой карты по её MAC-адресу.

2.3 Адреса сетевых интерфейсов

Для идентификации компьютера в пределах сегмента сети на канальном уровне используется шестибайтовый MAC-адрес, традиционно записываемый как шесть октетов в шестнадцатеричном виде, разделенные двоеточиями, например: 00:1c:bf:5f:c0:50. Старшие три байта корректного MAC-адреса идентифицируют производителя оборудования. MAC-адрес идентифицирует конкретный сетевой адаптер и выдан его производителем, но может быть легко изменён при необходимости в настройках операционной системы.

Нумерация интерфейсов своя в пределах каждого компьютера, а имя зависит от ядра конкретной операционной системы: имена интерфейсов, к сожалению, не стандартизованы даже в POSIX-системах. На всякий случай подчеркнём, что одинаковое название интерфейсов на разных ма-

шинах ни означает ничего, и никак не связано с тем тем, подключены ли они к одному или к разным сетевым сегментам.

> **Задание № 2: Получение сведений о сетевых интерфейсах**
>
> Выполните в обеих виртуальных машинах команду **ip link** или её сокращённый вариант **ip l**. Сколько сетевых интерфейсов показано, какие им присвоены MAC-адреса и какое значение MTU они имеют?

В результате выполнения задания мы увидим, что у машины есть три следующих сетевых интерфейса.

1) Локальный виртуальный интерфейс с именем **lo** предназначен для связи машины с нею же самой. Его MAC-адрес состоит из одних нулей: этот интерфейс является виртуальным даже на реальных машинах, поскольку присутствует всегда и не связан ни с каким сетевым устройством, а вместо этого «замыкается» сам на себя (англ. *loopback*). Это единственный функционирующий сейчас интерфейс: в его статусе присутствует слово **UP**.

2) Виртуальный интерфейс **teqlo0** служит для объединения нескольких интерфейсов в один и использования его как единого интерфейса. В данном пособии он не используется.

3) Наконец, сетевой интерфейс **eth0** канального протокола Ethernet имеет некоторый MAC-адрес — поскольку это виртуальная машина, то этот адрес выбирается симулятором произвольным образом, но гарантируется его уникальность в пределах одного реального компьютера. В описании интерфейса также указан широковещательный MAC-адрес, в котором все двоичные разряды равны единице: посланный на такой адрес Ethernet-кадр придёт на все сетевые интерфейсы, связанные с подключёнными к этому сегменту сетевыми адаптерами.

Интерфейс **eth0** связан с сетевым адаптером в реальных машинах. В случае нашей виртуальной машины он связан с виртуальным адаптером, подключенным к виртуальному сегменту, который соединяет две запущенные виртуальные машины друг с другом.

2.4 Адреса протокола IPv4

Протокол сетевого уровня IPv4 (англ. *Internet Protocol, версия 4* [9]) позволяет передавать данные между компьютерами, находящимися в разных, с точки зрения нижестоящего канального уровня, сегментах сети. Для работы с протоколом IP компьютер должен иметь как минимум один IP-адрес для каждого используемого сетевого интерфейса. Адрес обычно записывается в виде четырёх октетов в десятичном виде, разделённых точками, например: 127.0.0.1.

Маска сети используется для определения, относятся ли два IP-адреса к одной и той же сети или же к разным сетям. Она обозначает коли-

чество бит с начала IP-адреса (то, есть, со старшего байта левого октета), который отводятся под адрес сети. Маска может записываться либо кратко, как число этих битов (например, /24), либо длинно — как четыре октета, где соответствующие маске биты равны единицы, а остальные равны нулю (например, 255.255.255.0).

Допустим, что у нас есть некоторый IP-адрес (I) и его маска сети (M), и мы хотим проверить, принадлежит ли некоторый другой IP-адрес (O) этой сети. Если результат побитовой операции «И» длинного представления маски адреса и самого адреса равен результату побитового «И» маски и другого адреса ($I * M = O * M$, где $*$ — побитовое «И»), то адреса считаются принадлежащими к одной IP-сети.

В рамках исторического экскурса следует сообщить, что изначально IP-адреса использовались без отдельных масок. При этом маска сети определялась самим IP-адресом, точнее, его старшими битами: эти биты задавали класс сети, определяющий маску. Если старший бит адреса был нулевым, то адрес относился к классу «A», в котором первый байт определял сеть (что аналогично маске 255.0.0.0). Если старший бит был единицей, а следующий — нулём, то адрес относился к сети класса «B», в котором сеть задавалась старшими двумя байтами (аналог маски 255.255.0.0). Если старшие два бита были единичными, а следующий — нулевым, то это означало сеть класса «C», в которой первые три байта адреса задавали сеть (маска 255.255.255.0). Класс «D» также был введен (и по-прежнему существует) и имеет специальное назначение, которое мы увидим в дальнейшем.

Авторы, к сожалению, не знают точного ответа на вопрос, что же подвигло создать сети, число машин в которых (читай — в одном сегменте сети) могло исчисляться десятками тысяч и даже миллионами. По всей видимости, классы сетей и вводились как некоторое временное решение до разработки и стандартизации современной маршрутизации на основе масок сетей.

Довольно быстро классы сетей были заменены современной системой с использованием масок сетей, называемой CIDR (англ. *Classless Inter-Domain Routing*). Отметим, что использование масок позволяет при необходимости разбить любую сеть на 2^p сетей, использующих на p бит более длинную маску, но до введения таких масок сетей выделить подсети в сетях, например, класса «A», было невозможно. Мы можем увидеть рудименты классовой организации сетей в диапазонах частных и локальных адресов.

С каждым сетевым интерфейсом может быть связано, вообще говоря, несколько пар из IP-адреса и маски сети, но в данном пособии всегда будет существовать только один IP-адрес для каждого настроенного сетевого интерфейса. Судя по тому, что в текущий момент сетевой интерфейс **lo** настроен, система уже присвоила ему некоторый IP-адрес.

Задание № 3: Получение сведений об IP-адресах интерфейсов

Выполните на любой виртуальной машине команду **ip addr** (сокращённо — **ip a**) для вывода информации об IP-адресах, присвоенных сетевым интерфейсам. Ограничим вывод адресами IPv4 параметром **-4**.

```
ip -4 a
```

Обратите внимание на IP-адреса включённых интерфейсов, т. е. имеющих состояние «UP».

Мы увидим, что с интерфейсом **lo** действительно связан сетевой адрес 127.0.0.1/8 (поле **inet**). Как можно заметить, значение и маска этого адреса отвечают классу «A»: сеть локальных IP-адресов напоминает о классовой маршрутизации. Действительно, для связи компьютера с самим собою стандартом [10] была выделена сеть класса «A» — явно расточительное решение, учитывая дальнейшее исчерпание множества адресов протокола IPv4. Локальному сетевому интерфейсу также присвоен адрес протокола IPv6, который не рассматривается нами.

Раз в нашей системе уже есть один сетевой интерфейс, то мы можем проверить его работу, хотя он и бесполезен для связи машин **ws1** и **ws2** друг с другом. Для посылки служебных сообщений между компьютерами используется служебный протокол сетевого уровня ICMP [11], работающий поверх протокола IP. Наиболее хорошо известным его применением является команда **ping**: она посылает ICMP-сообщения типа «эхо-запрос», в ответ на которые компьютер с поддержкой ICMP посылает сообщения типа «эхо-ответ», которые и выводятся этой программой на экран. Поддержка протокола ICMP встроена в ядра операционных системы, поэтому для посылки такого ответа нам не нужно запускать никакие дополнительные сетевые службы. Для каждого полученного эхо-ответа будет указано, на какой запрос он отвечает (поле **icmp_req**), время от посылки запроса до получения ответа и значение поля **ttl** в пришедшем IP-пакете. Поле TTL (англ. *Time To Live*, «время жизни») IP-пакета показывает, через сколько маршрутизаторов он сможет пройти, перед тем как будет уничтожен.

Задание № 4: Использование утилиты ping

Выполнив на любой из виртуальных машин следующую команду, мы увидим пронумерованную цепочку ответов.

```
ping 127.0.0.1
```

Для прекращения работы программы нужно нажать комбинацию «Ctrl+C»; мы не будем в дальнейшем вам напоминать про необходимость прерывания этой команды. Учитывая, что эхо-ответ не проходил через маршрутизаторы, мы видим здесь то значение поля **TTL**, с которым пакеты создаются в нашей системе по умолчанию — определите его значение

2.5 Настройка сетевых интерфейсов

Чтобы передавать данные по протоколу IP между машинами **ws1** и **ws2**, нам надо присвоить IP-адреса сетевым интерфейсам **eth0** обеих машин, как показано на рисунке 2.1. В наших машинах существует два способа задания IP-адреса и маски сети : «на лету», например с помощью команды **ip**, которая будет действовать до перезагрузки машины, и перманентно, с помощью файлов настроек (каких именно — не стандартизовано даже в пределах дистрибутивов Linux). В данной главе используется первый способ, в последующих — второй.

Задание № 5: Назначение интерфейсу IP-адреса и маски сети

Выполним на машине **ws1** следующие команды, первая из которых присвоит интерфейсу **eth0** нужный нам адрес 10.0.0.11 с маской /16, а вторая — включит (или «поднимет», неформально) интерфейс, сделав возможной передачу данных через него.

```
ip a add 10.0.0.11/16 dev eth0
ip l set eth0 up
```

Выполните аналогичные команды на машине **ws2**, не забыв сменить в первой команде присваиваемый адрес на 10.0.0.22. Командой **ip a** затем нужно убедиться, что адреса присвоены правильно.

После присвоения обоих IP-адресов на машине **ws1** нужно отдать команду **ping** с IP-адресом машины **ws2** — в случае правильной настройки вы получите эхо-ответы.

Отметим, что в случае ошибочного присвоения адреса дать затем верную команду недостаточно — у интерфейса тогда будут два сетевых адреса, и необходимо удалить неверный командой **ip addr del**. В данной работе для полного сброса неверно настроенных параметров можно просто перегрузить виртуальную машину командой **reboot**.

2.6 Поиск MAC-адреса получателя и протокол ARP

Нам следует разобраться, каким образом машина **ws1** в предыдущем опыте узнала MAC-адрес получателя, ведь для передачи IP-пакета с ICMP-сообщением в кадре канального уровня требуется именно он.

При передаче данных в пределах сети Ethernet компьютеры могут либо использовать широковещание, либо передачу кадра конкретному адресату. В первом случае единственный отправленный кадр распространяется по всему сегменту сети: например, каждый коммутатор будет отправлять его во все свои включённые порты. В последнем случае сначала обычно требуется узнать его MAC-адрес по известному IP-адресу, поскольку для идентификации компьютеров используется именно последний.

Для решения этой задачи используется служебный протокол канального уровня ARP, который посылает широковещательный запрос канального уровня, содержащий IP-адрес. Широковещательный запрос получают все компьютеры в рамках сегмента сети, но только обладатель искомого IP-адреса отвечает на него, сообщая свой MAC-адрес.

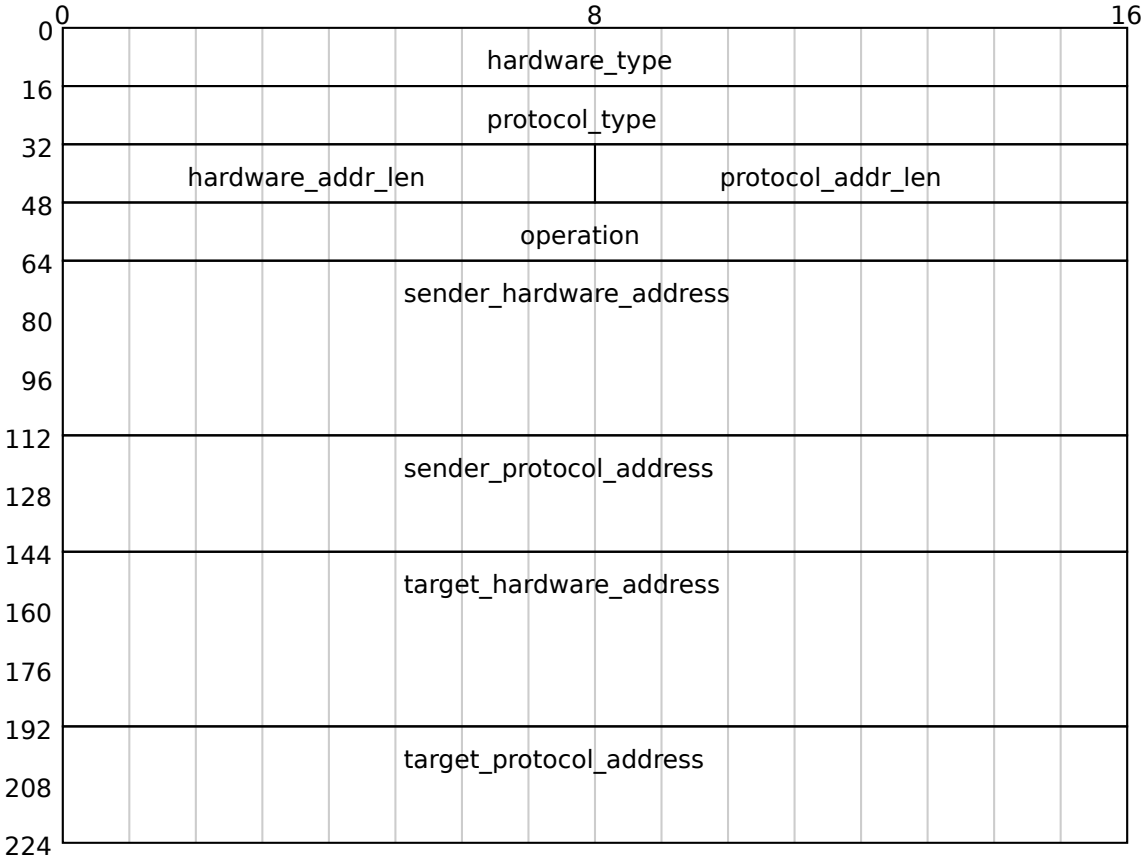


Рисунок 2.2 — Заголовок пакета ARP в случае протоколов Ethernet и IPv4

Протокол ARP создавался в расчете на использование для любой пары «протокол канального уровня + протокол сетевого уровня», что отражено в структуре его сообщения (рисунок 2.2; размеры полей указаны в битах). В заголовке указываются типы протоколов обоих уровней, называемых здесь **hardware** (протокол канального уровня) и **protocol** (протокол стевого уровня). Для случая протоколов Ethernet и IPv4 это числа 1 и 0x800 соответственно. Затем идут значения длин адресов обоих уровней в байтах, в нашем случае это будет 6 и 4. В поле **operation** указывается тип пакета: единица означает ARP-запрос («как выглядит MAC-адрес компьютера с таким-то IP-адресом?»), двойка — ARP-ответ («у меня такой-то MAC-адрес и такой-то IP-адрес»). Оставшуюся часть сообщения занимают собственно поля для MAC- и IP-адресов отправителя и получателя.

Отметим, что ARP является служебным протоколом и его сообщение помещается в кадр протокола канального уровня, где уже есть поля для MAC-адресов отправителя и получателя. В сообщении протокола ARP

эти поля могут использоваться в ряде случаев особым образом и не совпадать с такими же адресами в заголовке кадра.

Отображение IP → MAC запоминается в кеше известных MAC-адресов, чтобы не искать заново MAC-адрес получателя при каждом адресованном ему IP-пакете. Информация в этом кеше считается ненужной спустя некоторое время после нахождения MAC-адреса, если он более не использовался. В наших системах этот интервал равен по умолчанию одной минуте.

Для просмотра содержимого кеша ARP обычно используется команда **ip neighbour**[1](пакет iproute2, краткая форма — **ip n**) или команда **arp -n** (пакет net-tools).

Задание № 6: Вывод кеша протокола ARP

Выполните на любой из двух машин команду **ping** с IP-адресом другой машины в качестве параметра (для передвижения по списку выполненных команд можно использовать клавиши «Вверх» и «Вниз»). После её прерывания выведите содержимое кеша протокола ARP следующей командой.

```
ip n
```

Мы увидим в кеше ARP единственную запись (если вы её не видите — она уже удалена из ARP-кеша как малоиспользуемая и старая; повторите команду **ping**).

Теперь мы должны перехватить сетевой трафик, чтобы выяснить, какие кадры проходят по сети при работе протокола ARP. Для перехвата сетевого трафика здесь и далее мы будем использовать программу TCPdump. Команда **tcpdump** ожидает прихода кадров канального уровня на сетевую карту машины или посылки кадров через неё, в момент их появления она выведет на экран их содержимое в расшифрованном и достаточно удобном виде. В приложении В приведена дополнительная справка по всем нужным нам параметрам этой программы перехвата сетевого трафика.

Задание № 7: Перехват сообщений протокола ARP

Запустим на машине **ws2** перехватчик сетевых пакетов **tcpdump**, как показано ниже.

```
tcpdump -nt
```

В этой команде мы отключили вывод времени событий параметром **-t** и включили простой вывод всех адресов параметром **-n**: по стандартным соглашениям Unix-систем оба параметра можно записать как один.

[1]От англ. *neighbour* — «сосед».

На машине **ws1** при помощи команды **ip n flush** очистим кеш протокола ARP, чтобы начать последующие действия «с чистого листа».

```
ip n flush dev eth0
```

Затем пошлём два эхо-запроса — параметр **-c** у программы **ping** задает число передаваемых запросов.

```
ping 10.0.0.22 -c 2
```

На машине **ws2** мы увидим сетевой обмен по протоколам ARP и ICMP.

Программа tcpdump не только перехватывает сетевой трафик, но и «понимает» сообщения основных протоколов. Благодаря этому сообщения протоколов ARP и ICMP будут выведены в удобной для понимания форме.

На машине **ws2** будут перехвачены следующие кадры (в квадратных скобках ниже приведена информация, зависящая от вашей системы).

```
arp who-has 10.0.0.22 tell 10.0.0.11
arp reply 10.0.0.22 is-at [MAC-адрес интерфейса машины ws2]
IP 10.0.0.11 > 10.0.0.22: ICMP echo request, id [запрос], seq 1, length 64
IP 10.0.0.22 > 10.0.0.11: ICMP echo reply, [запрос], seq 1, length 64
IP 10.0.0.11 > 10.0.0.22: ICMP echo request, id [запрос], seq 2, length 64
IP 10.0.0.22 > 10.0.0.11: ICMP echo reply, id [запрос], seq 2, length 64
arp who-has 10.0.0.11 tell 10.0.0.22
arp reply 10.0.0.11 is-at [MAC-адрес интерфейса машины ws1]
```

Первая строка соответствует ARP-запросу IP-адреса 10.0.0.22, вторая — ответу на него. Затем следуют строки с эхо-запросами и эхо-ответами. Поле **seq** содержит порядковый номер запроса, поле **length** — длину пакета с запросом: в целях тестирования сети запросы можно делать и более длинными.

Поле **id** содержит идентификатор эхо-запроса; при посылке ответа в него помещается этот же идентификатор, по которому, вместе с номером последовательности, можно определить соответствие запроса и ответа. Отметим, что одного номера последовательности для этого недостаточно, ведь на одной машине может быть запущено одновременно несколько программ **ping**, опрашивающих один и тот же адрес.

После цепочки сообщений ICMP обычно видна пара из запроса и ответа ARP, в которых машина **ws2** запрашивает MAC-адрес по IP-адресу машины **ws1**. Связано это с тем, что операционная система регулярно проверяет записи в кеше ARP, если они были получены из пришедших IP-пакетов, а не ARP-опросом. Если пакеты с данного MAC и IP адреса не приходят в течение некоторого интервала (порядка нескольких секунд), то система отправляет ARP-запрос, желая проверить актуальность записи в ARP-кеше о даннном IP-адресе. Отметим, что если запись в кеше не ис-

пользуется в течение длительного периода времени (около минуты), то она удаляется с целью экономии ресурсов.

Прервать перехват трафика на машине **ws2** можно всё той же комбинацией «Ctrl+C». В дальнейшем мы не будем напоминать про необходимость прерывать эту программу: это всегда нужно будет делать, если на машине нужно будет выполнять какие-либо иные команды.

Задание № 8: Вывод MAC-адресов при перехвате сетевого трафика

Повторим описанный опыт, добавив к команде перехвата пакетов параметр **-e**. Это позволит увидеть в её выводе полную информацию о MAC-адресах получателя и отправителя каждого перехваченного кадра. Не забудьте сбросить ARP-кеш при повторе опыта (клавиши ««вверх»» и ««вниз»» помогут не набирать команды снова).

Не прерывайте перехват трафика, пока не увидите ARP-запрос IP-адреса 10.0.0.11, то есть вы должны перехватить две пары из запроса и ответа. Выясните, с какого и на какой MAC-адрес отправлялись запрос и ответ протокола ARP в обоих случаях.

В поле **ethertype** в выводе видна информация о том, какой протокол был вложен в кадр Ethernet. Например, строка **ethertype IPv4** означает, что внутри кадра находится IP-пакет. Определите, что вкладывается в кадр при посылке ARP-сообщения и при посылке сообщения ICMP.

Обратите внимание на MAC-адрес получателя ARP-запроса: это значение (ff:ff:ff:ff:ff:ff), очевидно, означает широковещательную рассылку кадра по всему сегменту сети. Отметим, что единичный младший бит старшего октета MAC-адреса означает, что этот адрес не соответствует некоторому конкретному адаптеру, а используется для групповой рассылки кадра, один из видов которой — широковещание — мы и видим здесь.

Отметим, что для посылки эхо-ответа нужно, разумеется, знать MAC-адрес отправителя. Машине **ws2** для этого не нужно отправлять ARP-сообщение, ей достаточно запомнить соответсвие IP-адреса и MAC-адреса отправителя ARP-запроса.

В проведенном опыте видно, что проверка актуальности кеша ARP не использует широковещательную рассылку — кадр отправляется на MAC-адрес, сохранённый в кеше ARP. Это решение позволяет сократить широковещательный сетевой трафик. Если ответ не будет получен и актуальность записи кеша не удастся подтвердить, то она будет удалена. В дальнейшем при необходимости послать пакет на этот IP-адрес будет использован уже широковещательный ARP-запрос.

Мы провели два опыта, когда протокол ARP позволил успешно найти MAC-адрес получателя. Представляет интерес и случай, когда такой адрес найти не удаётся.

Для лучшего понимания происходящего нам понадобится время перехвата сетевых кадров. При отсутствии ключа **-t** при перехвате трафика программой TCPdump будет выводится примерное время события в формате ЧЧ:ММ:СС.МММММММ, где ЧЧ — часы, ММ — минуты, СС — секунды, МММММММ — миллисекунды. Разумеется, это время определяется весьма приблизительно, с точностью примерно до сотых долей секунды. Наши виртуальные машины по умолчанию показывают время по Гринвичу, что нас не должно волновать, поскольку важны только интервалы времени между событиями, а не сами моменты их возникновения.

> **Задание № 9: Отправка пакета на отсутствующий в сети IP-адрес**
>
> Повторите опыт, попытавшись отправить запрос на адрес 10.0.0.55. Для перехвата трафика на машине **ws2** следует использовать следующую команду.
>
> ```
> tcpdump -n
> ```
>
> На машине **ws1** пошлём единственный эхо-запрос.
>
> ```
> ping 10.0.0.55 -c 1
> ```
>
> Протокол ARP, очевидно, не может рассчитывать на получение сообщения «а тут нет такого адреса», поэтому он использует таймер для повтора попытки обнаружения отсутствия адресата при отсутствии ответа. Определите его примерное значение и число попыток обнаружения адресата по сетевому трафику, перехваченному на машине **ws2**.

2.7 Маршрутная таблица протокола IP

Как отмечалось в главе 1, каждый компьютер с поддержкой протокола IP решает задачу маршрутизации. Для определения того, куда направить IP-пакет, используется *маршрутная таблица*. В типичных ОС эта таблица относится к данным ядра ОС, поскольку именно в ядре решается задача маршрутизации IP-пакетов.

Маршрутная таблица протокола IP — это, формально, отображение IP-адреса в пару из IP-адреса некоторого маршрутизатора и сетевого интерфейса. Из соображений компактности отображаемые IP-адреса не записываются в неё по отдельности, а представлены адресами и масками сетей.

С помощью этой таблицы можно решить, что следует делать с IP-пакетом, не адресованным данному компьютеру лично: по адресу получателя пакета компьютер с поддержкой протокола IP может определить, какому маршрутизатору и через какой сетевой интерфейс следует переслать IP-пакет. В строке таблицы адрес маршрутизатора также может быть пу-

стым, что означает, что искомый IP-адрес доступен непосредственно через указанный интерфейс, и пакет следует послать указанному в пакете получателю. Первый из этих случаев соответствует косвенной маршрутизации, второй — непосредственной маршрутизации. В данной главе мы увидим только второй случай, а в главе 3 будут рассмотрены оба из них.

Задание № 10: Вывод таблицы маршрутизации

Выполните команду **ip r** (полностью — **ip route**) для вывода таблицы маршрутизации протокола IP. Будет выведена следующая простейшая таблица маршрутизации (последний выведенный адрес будет зависеть от машины, где исполнена команда).

```
10.0.0.0/16 dev eth0 proto kernel scope link src 10.0.0.11
```

Как мы видим, в таблице маршрутизации приведена единственная строка, указывающая на доступность сети 10.0.0.0/16 через сетевой интерфейса **eth0**. После слова **src** указан источник этой записи в таблице: она появилась в силу наличия у машины адреса 10.0.0.11. После слова **scope** указано «область действия»: значение **link** означает, что эта запись верна только в пределах указанного в ней сетевого интерфейса, поскольку указанная сеть достижима именно через него. Локальный интерфейс **lo** должен быть всегда изолирован от всех остальных — иначе он не будет локальным — поэтому здесь в таблице маршрутизации он не показан.

Пометка **proto kernel** в таблице маршрутизации означает, что эта строка появилась в силу действий ядра ОС (которые, в свою очередь, были реакцией на нашу команды **ip addr add**). Позже, в главе 6 мы увидим и строки, созданные по указанию сетевых служб по обмену маршрутной информацией.

Задание № 11: Случай отсутствия адреса в таблице маршрутизации

Выполните команду **ping** с адресом, не попадающим в сети обоих сетевых интерфейсов (например, это адрес 10.10.10.10). Убедитесь, что сразу же будет выведено сообщения об отсутствии возможности отправки пакета. Объясните разницу во времени ответа и выводимых сообщениях по сравнению с попыткой послать эхо-запрос на адрес 10.0.0.55 в одном из предыдущих опытов.

Для демонстрации косвенной маршрутизации нам придётся создать компьютерную сеть как минимум с одним маршрутизатором, что и будет сделано в следующей главе.

2.8 Завершение работы

Проведите любые другие опыты, которые вам необходимы для ответов на контрольные вопросы в конце этой главы (раздел 2.9). После подготовки ответов на контрольные вопросы можно остановить виртуальные машины.

Отметим некоторые общие команды, которые нам пригодятся в дальнейшем. Для завершения работы виртуальной машины можно выполнить в ней команду **halt**, либо выполнить команду **vcrash <имена_машин>** в основной машине. Следует отметить, что команда **vcrash** может привести к потере данных в виртуальной машине, в т. ч. последних изменений в файлах конфигураций. Для перезагрузки виртуальной машины можно выполнить внутри неё команду **reboot**.

> **Задание № 12: Остановка виртуальных машин**
>
> Остановите обе виртуальные машины командой **halt**. После завершения работы машин вы можете удалить также образы их виртуальных дисков следующей командой, отданной в основной машине в том каталоге, откуда были запущены машины.
>
> ```
> rm ws1.* ws2.*
> ```

В дальнейших работах мы уже не будем вам предлагать завершать виртуальные машины и удалять образы их дисков, предполагая, что при необходимости вы сможете сделать это сами.

2.9 Контрольные вопросы

Для самоконтроля полученных знаний рекомендуется ответить на следующие вопросы.

1) Какие MAC-адреса имеют сетевые интерфейсы **eth0** запущенных в задании виртуальных машин?
2) Какое значение величины MTU установлено у интерфейса, связанного с сегментом Ethernet? У локального сетевого интерфейса?
3) Почему при настройке сетевого интерфейса, помимо IP-адреса, нужно указать и маску сети?
4) Чему в нашем случае равно значение TTL у созданных IP-пакетов?
5) Что выведет команда **ping 127.10.10.100**? Связано ли это с историческими классами сетей?
6) Для чего служит протокол ARP? Сколько видов ARP-сообщений существует?
7) С каким интервалом и сколько раз наша машина повторяет попытки найти MAC-адреса получателя при посылке одного IP-пакета?
8) На какие MAC-адреса отправляются запросы протокола ARP?

9) Попадают ли в кеш протокола ARP адреса из сети локального сетевого интерфейса?

10) Куда вкладывается ICMP-сообщение: в пакет IP или сразу в кадр Ethernet? Куда вкладывается ARP-сообщение?

11) Что такое эхо-запрос и эхо-ответ? Для чего нужен идентификатор эхо-запроса?

12) Как машина **ws1** узнаёт в проведенном опыте MAC-адрес машины **ws2**? Как машина **ws2** узнаёт MAC-адрес машины **ws1**?

13) В ARP-запросе есть, кроме MAC-адреса, и IP-адрес отправителя. Для чего он используется машинами в данной главе?

14) Для чего нужна маршрутная таблица протокола IP? На каких компьютерах она присутствует?

15) Почему в нашей системе команда **ping 10.0.0.33** выведет другое сообщения об ошибке, чем команда **ping 10.10.100.22**?

16) Как выглядит кеш известных MAC-адресов сразу после старта виртуальной машины?

3 Протокол IP и статическая маршрутизация

В данной работе мы настроим систему из двух маршрутизаторов, соединяющих три сетевых сегмента друг с другом. Каждому сегменту будет соответствовать одна IP-сеть (рисунок 3.1). В двух из этих сетей мы разместим рабочие станции, а третья только соединит два маршрутизатора друг с другом. Конечно, в реальных сетях в каждом сегменте обычно находятся десятки компьютеров, но поскольку их сетевые настройки различаются только адресом хоста, то мы обычно будем создавать в сегменте не более пары машин, не являющихся маршрутизаторами.

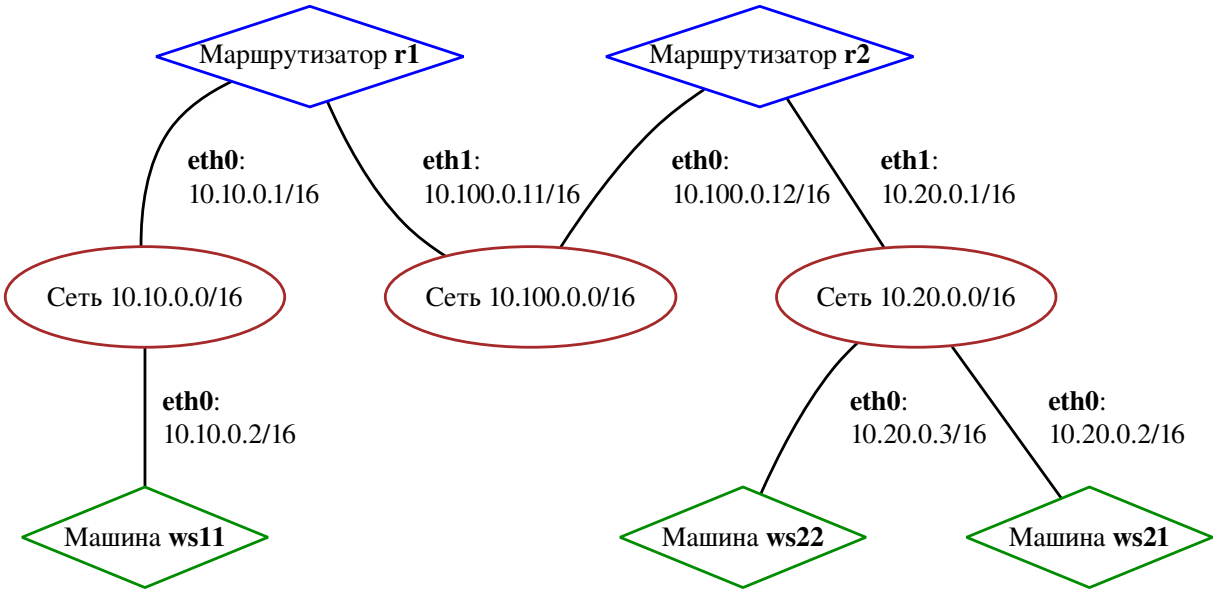


Рисунок 3.1 — Компьютерная сеть из соединённых двумя маршрутизаторами IP-сетей

Мы познакомимся с IP-маршрутизацией и с использованием протокола ICMP для информирования о возникающих в ходе маршрутизации ошибках. Также мы увидим фрагментацию IP-пакетов и один из способов борьбы с нею.

Для построения списка маршрутизаторов на пути до получателя мы используем программу **traceroute**. Также мы используем программу **ip** для добавления маршрутов и управлением величиной MTU. Для настройки параметров сети мы используем файл настройки сети **/etc/network/interfaces**, используемый в ОС Debian.

В конце этой главы (раздел 3.14) приведены возможные индивидуальные варианты заданий. В разделе 3.13 даны дополнительные указания по их выполнению.

3.1 Структура IP-пакета

Перед выполнением практических заданий рассмотрим структуру сообщения протокола IP. IP-пакет является наименьшей единицей инфор-

мационного обмена по протоколу IP. Он состоит из блока данных, перед которым размещается стандартный заголовок и дополнительные опции, причём последние на практике обычно отсутствуют. Размер заголовка без опций составляет 20 байт, или пять 32-битных слов; структура заголовка приведена на рисунке 3.2, размеры полей указаны в битах.

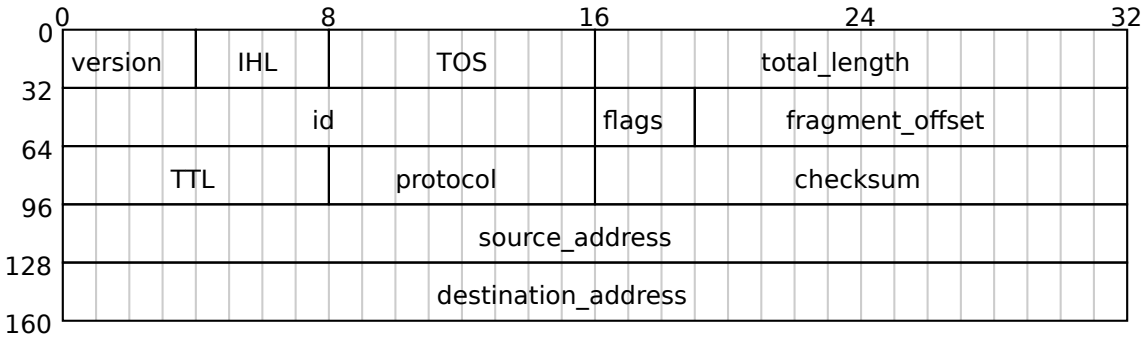


Рисунок 3.2 — Заголовок пакета IPv4

Опишем кратко присутствующие в заголовке IP-пакета поля. Поле **version** содержит номер версии протокола, для IPv4 это число 4. Поле **protocol** идентифицирует протокол транспортного или сетевого уровня, данные которого вложены в IP-пакет. Коды основных протоколов: ICMP — 1, TCP — 6, UDP — 17.

Полный размер заголовка в 32-битных словах, включая опции, указывается в поле **IHL**. Общий размер пакета в байтах хранится в 16-битном поле **total_length**. Таким образом, максимальный теоретический размер пакета IPv4 ограничен 0xFFFF байтами, а на практике размеры пакетов значительно меньше из-за ограничений канального уровня.

Последние, по порядку следования, поля заголовка содержат IP-адреса отправителя (поле **source_address**) и получателя (поле **destination_address**).

Поле **TOS** (англ. *Type of service*) задает требования к качеству передачи: низкие задержки или низкие потери, которыми могут руководствоваться маршрутизаторы при обработке пакета и выборе маршрута для него. В настоящее время маршрутизатор самостоятельно решает, что ему следует делать с пакетом, и поэтому может как принять во внимание это поле, так и нет.

Один IP-пакет может разбиваться на части и вкладываться в несколько кадров канального уровня, если его общий размер превышает величину MTU сегмента сети. Такое явление называется **фрагментацией**, а части пакета — фрагментами. Каждый фрагмент является также IP-пакетом и имеет поэтому свой собственный заголовок. Фрагменты одного и того же пакета имеют равное значение поля **id** в заголовке, а поле **fragment_offset** задаёт смещение фрагмента относительно начала пакета. Для управления фрагментацией также используется поле **flags**. Все

фрагменты, кроме последнего, имеют установленный флаг **MF** (англ. *more fragments*).

При использовании протоколов транспортного уровня с повторной передачей — например, TCP — потеря одного фрагмента ведёт к повторной отправке всех фрагментов пакета. Поэтому фрагментация обычно нежелательна, и её отключают. Для этого отправителем устанавливается флаг **DF** (англ. *don't fragment*). Мы увидим использование этого флага для борьбы с фрагментацией далее в этой главе.

Поле **ttl** содержит оставшееся «время жизни» пакета, обычно измеряемое в числе прыжков между маршрутизаторами. Каждый маршрутизатор уменьшает значение TTL пакета как минимум на единицу (на практике — ровно на единицу) и отбрасывает пакет, когда оно достигает нуля. Поле **checksum** содержит контрольную сумму заголовка пакета, причём тело пакета не участвует в расчете суммы — проверка контрольной суммы передаваемых данных не относится к функциям сетевого уровня в стеке TCP/IP.

Отметим, что ряд стандартов [12] определяют IP-датаграмму как синоним нефрагментированного IP-пакета. Данный термин тесно связан с датаграммой как сообщением протокола UDP: одно сообщение протокола UDP помещается в одну IP-датаграмму. Тем не менее, во избежание двусмысленности мы далее будем использовать термин только для сообщений протокола UDP.

3.2 Запуск машин и настройка сетевых интерфейсов

В этой и последующих работах мы будем использовать предварительно настроенную на канальном уровне сеть. Для этого понадобится архив с шаблоном, содержащий частично настроенные виртуальные машины и файл описания их соединения друг с другом.

Задание № 13: Запуск сети командой ltabstart

Скачайте и распакуйте архив с шаблоном сети следующими командами, выполненными в основной машине.

```
mkdir -p ~/tcp-ip; cd ~/tcp-ip
wget -r http://ftp.iu7.bmstu.ru/nets/lab-ip.tar.gz
rm -rf lab-ip
tar -xvf lab-ip.tar.gz
cd lab-ip
```

Запустить виртуальные машины можно следующей командой **ltabstart**, которая автоматически запустит виртуальные машины в различных вкладках эмулятора терминала.

```
ltabstart -d net
```

Последний параметр команды (после ключа **-d**) задает путь к файлу конфигурации виртуальной сети.

В предыдущей работе мы уже использовали для настройки IP-адреса «на лету» команду **ip**. Эта настройка сбрасывалась при перезагрузке машины, поэтому теперь мы будем использовать для настроек наших виртуальных машин файлы конфигурации. Отметим, что для перезапуска этой сети после, например, выключения компьютера, следует отдать следующие команды.

```
cd ~/tcp-ip/lab-ip
ltabstart -d net
```

После их выполнения сеть будет запущена с учётом изменённых файлов конфигурации. Это замечание касается как текущей, так и всех последующих глав — не забудьте сменить только каталог в команде (**cd**) выше на соответствующий.

К сожалению, соглашения о расположении и формате файлов сетевых настроек не стандартизированы, в силу чего дальнейшая информация о них распространяется только на Debian и родственные ему дистрибутивы GNU/Linux.

В системе Debian для настройки IP-адресов, MAC-адресов, масок и маршрутов используется файл **/etc/network/interfaces**. Для компьютера **ws11** ниже приведен фрагмент этого файла, соответствующий желаемой конфигурации сети (рисунок 3.1). Начиная с символа «решетки» идут комментарии, которые размещены ниже исключительно для пояснения содержимого файла.

```
# Интерфейс lo служит только для связи компьютера
# с самим собой (тип интерфейса: loopback)
auto lo
iface lo inet loopback
# Далее идёт описание сетевого интерфейса eth0.
auto eth0
# Ему будет назначен фиксированный IP-адрес.
iface eth0 inet static
    # А вот и его IP-адрес.
    address 10.10.0.2
    # Кроме адреса, нужно присвоить маску сети
    netmask 255.255.0.0
    # (использована длинная форма маски /16).
```

При настройке виртуальных машин далее используется стандартный подход для настройки unix-систем: сначала мы отредактируем файл конфигурации, а затем перезапустим соответствующую службу.

Для машин **ws21** и **ws22** этот файл будет отличаться только строкой с IP-адресом. Для редактирования этого файла можно воспользоваться

любым имеющимся текстовым редактором. Если у вас нет предпочтений, можно использовать **mcedit**, отдав следующую команду[1].

```
mcedit /etc/network/interfaces
```

В дальнейшем мы не будем более останавливаться на вопросе, каким редактором и как вам следует редактировать файлы конфигурации. Отметим также, что при желании использовать двухпанельный файловый менеджер можно запустить в виртуальной машине ПО Midnight Commander (команда **mc**).

Задание № 14: Настройка адресов рабочих станций

Вам нужно отредактировать на машинах **ws11**, **ws21**, **ws22** файл конфигурации сети **/etc/network/interfaces** в соответствии с рисунком 3.1. Для сокращения времени работы эти файлы в шаблоне сети уже почти готовы, в них нужно лишь удалить символ комментария перед нужными строчками, указать IP-адреса и проверить маски сетей. После изменения на любой машине файла **interfaces** здесь и в дальнейшем всегда нужно перезагрузить настройки сети на этой же машине следующей командой.

```
service networking restart
```

Прочитайте вывод команды выше — вполне возможно, при редактировании файла были допущены ошибки. Если это так, исправьте их и повторите попытку, если всё хорошо — проверьте командой **ip -4 a**, что адрес задан верно. В дальнейшем мы не будем больше напоминать про необходимость читать вывод команды **service** и исправлять найденные ошибки.

После этих действий с машины **ws21** можно будет отправить эхо запросы на машину **ws22** и наоборот — убедитесь в этом, используя команду **ping**.

Теперь нам нужно задать базовые настройки для маршрутизаторов **r1** и **r2** (для имён маршрутизаторов, мы будет использовать префикс «r»). Сетевая конфигурация каждого маршрутизатора в данной сети включает описание двух интерфейсов (не считая локального), пример для машины **r1** приведен ниже.

```
auto lo
iface lo inet loopback
auto eth0
iface eth0 inet static
    address 10.10.0.1
    netmask 255.255.0.0
auto eth1
iface eth1 inet static
```

[1]Краткая справка по редактору: F2 — сохранить, F9 — меню, F10 — выход.

```
address 10.100.0.11
netmask 255.255.0.0
```

Задание № 15: Настройка адресов маршрутизаторов

Отредактируйте на обоих маршрутизаторах файлы **interfaces** и затем перегрузите на них настройки сети. После обновления сетевых настроек у нас должны успешно отправляться эхо-запросы в пределах любого из трех сегментов сети — но не дальше. Проверьте это при помощи программы ping.

Таблицы маршрутизации на всех наших машинах все еще тривиальны и содержат только информацию о смежных сетях (убедитесь в этом командой **ip r**). В настоящий момент, например, попытка отправить эхо запрос с машины **ws11** на машину **ws21** будет неудачной. Для исправления ситуации следует добавить в маршрутные таблицы дополнительные строки. Строку в маршрутной таблице часто называют *маршрутом* (англ. *route*).

3.3 Информация о маршруте по умолчанию

Все три рабочие станции подключены к сегментам, в которых имеется единственный маршрутизатор. По этой причине необходимо и достаточно указать его IP-адрес в качестве маршрута по умолчанию. Для этого в случае машины **ws11** надо добавить в конец файла конфигурации сетевых интерфейсов (после маски сети интерфейса **eth0**) следующую строку, указывающую IP-адрес интерфейса маршрутизатора **r1** в нашей же сети.

```
gateway 10.10.0.1
```

Задание № 16: Установка маршрута по умолчанию

Добавьте в файл настройки сетевых адресов машины **ws11** маршрут по умолчанию и перезагрузите сетевые настройки. Команда вывода маршрутной таблицы (**ip r**) теперь должна показать маршрут по умолчанию через маршрутизатор 10.10.0.1.

Добавьте аналогичным образом маршрут по умолчанию для машин **ws21** и **ws22**, не забыв проверить затем их таблицы маршрутизации.

Правильная таблица маршрутизации на машине **ws11** будет выглядеть следующим образом.

```
10.10.0.0/16 dev eth0  proto kernel  scope link  src 10.10.0.2
default via 10.10.0.1 dev eth0
```

В этой таблице, кроме записи о подсоединенной через интерфейс **eth0** сети 10.10.0.0/16, присутствует и запись о маршруте по умолчанию через маршрутизатор с адресом 10.10.0.1, доступном через этот же сетевой интерфейс.

Отметим, что термин *шлюз* (англ. *gateway*) часто является синонимом маршрутизатора, указанного в маршруте до некоторой сети. Шлюз в маршруте по умолчанию также часто называют *маршрутизатором по умолчанию* или *шлюзом по умолчанию* (англ. *default gateway*). В файле конфигурации Debian этот маршрутизатор не вполне корректно назвали просто шлюзом. Поскольку *шлюзом* также называют устройство, соединяющее сети разных сетевых стеков, а также маршрутизатор, обычно соединяющий локальную сеть с интернетом, то данный термин мы не будем использовать по причине крайней многозначности.

Теперь с машины **ws11** можно отправлять эхо-запросы на любой IP-адрес маршрутизатора **r1**. Однако, уже при попытке послать запрос на адрес 10.100.0.12 мы не получим эхо-ответ, поскольку в таблице маршрутизатора **r2** нет сведений о сети 10.10.0.0/16.

> **Задание № 17: Проверка маршрута по умолчанию**
>
> Убедитесь, что эхо-запрос с **ws11** доходит до машины **r2**: перехват трафика поможет в этом. Для этого можно отдать следующую команду на машине **r2** (поскольку у маршрутизатора два сетевых интерфейса, в команде явно указан интерфейс для перехвата трафика после ключа **-i**).
>
> ```
> tcpdump -tn -i eth0
> ```
>
> Если теперь с машины **ws11** послать эхо-запрос на адрес 10.100.0.12, то вы увидите, что пакеты с ним дошли до маршрутизатора, но никаких эхо-ответов не было получено.

Этот простой пример показывает, что успешная доставка пакета по пути в одну сторону совершенно не гарантирует возможность доставки пакета в обратную сторону. Действительно, машина **r2** не может отправить ответ, поскольку ей не известен маршрут до сети с машиной **ws11**. Здесь же мы видим, что отсутствие эхо-ответов при диагностики сети совершенно не означает, что проблема связана именно с машиной-отправителем эхо-запросов, а из отсутствия ответа не следует, что потерян именно запрос.

3.4 Статический маршрут

После указания маршрутов по умолчанию рабочим станциям в нашей системе останется только одна проблема — каждый из маршрутизаторов «знает» только про две сети из трёх имеющихся. По этой причине попытка посылки с маршрутизатора пакета к рабочей станции в неизвестной сети приводит к неудаче — проверьте это командой **ping**.

Для создания статического маршрута, который будет корректно добавляться при включении сетевого интерфейса, можно воспользоваться тем же файлом **interfaces**. В описании сетевого интерфейса можно добавить команды, которые будут выполняться при его включении (они идут

после слов «up»), и команды, которые будут выполняться при его отключении (после слов «down»).

Нам нужно добавить в маршрутные таблицы обоих маршрутизаторах строки с информацией о неизвестной сети. Для машины **r1** в строке добавления маршрута указаны: адрес сети, IP-адрес маршрутизатора **r2** и сетевой интерфейс, через который доступен указанный маршрутизатор. В команде удаления маршрута указывается только адрес сети.

```
# Добавить в конец описания сетевого интерфейса eth1:
    up ip r add 10.20.0.0/16 via 10.100.0.12 dev eth1
    down ip r del 10.20.0.0/16
```

Как можно увидеть, после слов «up» и «down» здесь просто идет команда **ip route**, которая будет исполнена командной оболочкой. Этот способ задания маршрутов, к сожалению, нельзя назвать удобным, но лучше штатного способа в ОС Debian не предусмотрено.

Задание № 18: Добавление статических маршрутов

Добавьте приведенные выше команды добавления и удаления маршрута в файл сетевых настроек маршрутизатора **r1**. После перезагрузки его сетевых настроек проверьте маршрутную таблицу маршрутизатора командой **ip r** При необходимости исправьте ошибки и повторно перегрузите сетевые настройки. В случае некоторых ошибок иногда может быть проще перегрузить виртуальную машину.

После завершения настройки маршрутизатора **r1** можно переходить ко второму маршрутизатору. Для маршрутизатора **r2** в команде добавления маршрута надо указать сеть, находящуюся «за» маршрутизатором **r1**, IP-адрес **r1** в сети 10.100.0.0/16 и сетевой интерфейс, подключенный к этой сети — **eth0** (рисунок 3.1).

Обратите внимание: на маршрутизаторе **r2** команды добавления и удаления маршрута должны идти в конце описания интерфейса **eth0**: хотя сам маршрут от этого не меняется, но зато тогда он будет корректно удалён именно при отключении интерфейса **eth0**.

Перегрузите сетевые настройки маршрутизатора **r2** и проверьте его маршрутную таблицу. После завершения настроек обоих маршрутизаторов каждая машина сможет послать IP-пакет любой другой — проверьте это отправкой эхо-запроса с **ws11** на **ws21**.

Правильная таблица маршрутизации на машине **r1** будет выглядеть следующим образом. Помимо информации о двух подключенных к сетевым интерфейсам сетям, в ней видна и запись о доступности сети 10.20.0.0/16 через маршрутизатор 10.100.0.12.

```
10.100.0.0/16 dev eth1  proto kernel  scope link  src 10.100.0.11
10.20.0.0/16 via 10.100.0.12 dev eth1
10.10.0.0/16 dev eth0  proto kernel  scope link  src 10.10.0.1
```

Добавленные нами маршруты принято называть *статическими маршрутами* — они существуют с момента включения сетевого интерфейса и не меняются. Далее, в главе о динамической маршрутизации, мы увидим и динамические маршруты.

3.5 Использование маршрутных таблиц

Далее мы будем под *путём* (англ. *path*) понимать последовательность маршрутизаторов, которые некоторый IP-пакет проходит при следовании от отправителя к получателю. Поскольку задача маршрутизации решается независимо каждым маршрутизатором, а таблица маршрутизации может меняться в ходе его работы, то путь одного пакета может не совпадать с путём другого с теми же адресами получателя и отправителя. Возможность изменения пути между двумя получателями является важным достоинством сетей TCP/IP.

Как мы уже знаем, при наличии двух IP-адресов и маски сети при помощи побитового «И» можно понять, относятся ли адреса к одной сети. Поиск маршрута по таблице маршрутизации идёт аналогичным образом. Однако, в маршрутной таблице могут быть сети, включенные друг в друга. Выполним на машине **ws11** следующую команду, которая должна вывести маршрут до указанной в последнем параметре команды сети.

```
ip r list 10.10.0.0/16
```

Мы увидим ожидаемую запись из маршрутной таблицы.

```
10.10.0.0/16 dev eth0  proto kernel  scope link  src 10.10.0.2
```

Теперь мы попросим вывести маршрут до сети 0.0.0.0/0 следующей командой

```
ip r list 0.0.0.0/0
```

В ответ на неё мы увидим маршрут по умолчанию.

```
default via 10.10.0.1 dev eth0
```

Это означает, что маршрут по умолчанию с точки зрения маршрутизации эквивалентен маршруту до сети 0.0.0.0/0. Действительно, маршрут по умолчанию можно рассматривать просто как маршрут до сети 0.0.0.0/0, поскольку такая сеть благодаря нулевой маске включает все возможные адреса IPv4.

Таким образом, IP-адрес некоторого пакета может попадать в несколько сетей, указанных в маршрутной таблице маршрутизатора. Маршрутизатор тогда просто должен выбрать из них ту, которая имеет самую длинную маску, то есть взять самое «точное» решение. Согласно данному правилу «самой длинной маски» маршрут по умолчанию никогда

не будет выбран, если есть альтернативные варианты с маской ненулевой длины.

Отметим, что и в настройках маршрутизатора ничто не мешает использовать параметр сети **gateway**, когда маршруты во все сети, кроме некоторых, проходит через один и тот же маршрутизатор. Однако, такой маршрут обычно используется для указания пути к маршрутизатору, соединяющую нашу систему с остальными IP-сетями (т. е., с интернетом). Поскольку наша система не имеет такого соединения, использование маршрута по умолчанию в ней будет плохим примером его использования, за исключением, конечно же, настроек рабочих станций в тупиковых сетях.

3.6 Косвенная маршрутизация

В разделе 2.6 мы видели, как IP-пакет передается между двумя машинами в одном сегменте сети. В дальнейшим мы для краткости будем называть компьютеры *соседями*, если у них есть интерфейсы, подключенные к одному и тому же сегменту сети, и для передачи IP-пакетов между ними достаточно непосредственной IP-маршрутизации. Сегмент сети, к которому некоторый компьютер подключён непосредственно через свой сетевой интерфейс, мы будем для краткости называть *смежным*.Сейчас мы увидим, как пересылается IP-пакет, если на очередном шаге маршрутизации его получатель находится не в смежном сегменте.

Задание № 19: Наблюдение за косвенной маршрутизацией

Для наблюдения за косвенной маршрутизацией предварительно нужно выполнить ряд действий. Сначала мы сбросим на нескольких машинах кеш преобразования IP-адреса в MAC-адрес по пути от **ws11** к **ws21**, чтобы затем увидеть протокол ARP в действии. Для этого выполним на машине **ws11** следующую команду для сброса кеша протокола ARP.

```
ip n flush dev eth0
```

На маршрутизаторах **r1** и **r2** мы сбросим этот кеш на другом интерфейсе следующей командой.

```
ip n flush dev eth1
```

Затем следует подготовить перехват сетевого трафика, запустив перехват сетевых пакетов на машинах **r1**, **r2** и **ws21** следующим образом.

```
tcpdump -tne -i eth0
```

Теперь все готово в тому, чтобы мы смогли проследить, как передается один IP-пакет от **ws11** к **ws21** (и один — в обратном направлении). Отдадим на машине **ws11** команду **ping** с IP-адресом машины **ws21** в качестве аргумента.

```
ping 10.20.0.2 -c 1
```

Теперь изучите перехваченный на машинах **r1**, **r2** и **ws21** сетевой трафик.

На первом шаге машина-отправитель определяет по своей маршрутной таблице, что пакет нужно отправить не непосредственно получателю (его IP-адрес не принадлежит смежным сегментам), а маршрутизатору с адресом 10.10.0.1. Поскольку мы очистили кеш ARP, то для этого сначала нужно получить соответствующий MAC-адрес с помощью протокола ARP, и затем отправить сам IP-пакет. Перехваченный на маршрутизаторе **r1** сетевой трафик в сети 10.10.0.0/16 покажет запрос и ответ протокола ARP, а затем передаваемые ICMP-сообщения. В сокращенном варианте часть этого трафика показана ниже.

```
[MAC ws11:eth0] > ff:ff:ff:ff:ff:ff arp who-has 10.10.0.1 tell 10.10.0.2
[MAC r1:eth0] > [MAC ws11:eth0] arp reply 10.10.0.1 is-at [MAC r1:eth0]
[MAC ws11:eth0] > [MAC r1:eth0] 10.10.0.2 > 10.20.0.2: ICMP echo request
```

Из перехваченного трафика видно, что в качестве MAC-адреса получателя выступает адрес интерфейса **eth0** маршрутизатора **r1**, только что полученный машиной **ws11** по протоколу ARP. Напомним, что для вывода информации об адресах интерфейсов можно отдать команду **ip a**) Заметьте, что IP-адрес маршрутизатора (10.10.0.1) в самом маршрутизируемом пакете не фигурирует — он был использован только для получения его MAC-адреса.

Таким образом, получателем кадра, в который вложен отправляемый на адрес 10.20.0.2 IP-пакет, является очередной маршрутизатор, поэтому показанный процесс называется *косвенной маршрутизацией*.

Проходящий в нашем примере через маршрутизатор пакет называется *транзитным* — он получен извне, но ни один из IP-адресов маршрутизатора не фигурируют в поле IP-адреса получателя пакета.

Теперь давайте выясним, что происходит с этим пакетом при прохождении маршрутизатора. В перехваченном на машине **r2** трафике мы видим, что отправляет в сеть маршрутизатор **r1** через интерфейс **eth1**. По своей таблице маршрутизации **r1** определил, что пакет следует передать маршрутизатору с адресом 10.100.0.12. Чтобы передать ему кадр с этим IP-пакетом, он должен найти соответствующий MAC-адрес с помощью протокола ARP. Таким образом, кадр отправляется с MAC-адресом интерфейса **eth1** машины **r1** в качестве отправителя и MAC-адресом интерфейса **eth0** машины **r2** — в качестве получателя. Отметим, что IP-адреса отправителя и получателя в пересылаемом IP-пакете, конечно же, остаются без изменений.

На последнем шаге маршрута получатель кадра (MAC-адрес **ws21**) соответствует получателю IP-пакета — используется непосредственная

маршрутизация. В этом можно убедиться, просмотрев перехваченные на машине **ws21** пакеты.

3.7 Управление временем жизни IP-пакета

Как видно из перехваченной информации, в самом IP-пакете при прохождении его через маршрутизатор изменяется значение поля TTL, уменьшаясь на единицу. Это поле предотвращает появление «вечных» пакетов — при достижении им нулевого значения пакет будет уничтожен маршрутизатором, а получателю будет направлено особое ICMP-сообщение, извещающее об этом событии. Если бы этого поля не было, ошибочная информация в маршрутных таблицах могла бы привести к бесконечной пересылки пакета в случае маршрутного цикла, который может легко возникнуть даже при разовой ошибке с указанием адреса следующего маршрутизатора.

Давайте убедимся, что даже в случае возникновения маршрутного цикла IP-пакет не будет двигаться по сети до бесконечности. Для этого сначала нужно создать ошибочный маршрутный цикл.

Задание № 20: Создание маршрутной петли

Создадим маршрутную петлю, отключив на маршрутизаторе **r2** интерфейс **eth1** и добавив неверный маршрут до сети 10.20.0.0. Выполним для этого на маршрутизаторе **r2** следующие команды.

```
ip link set eth1 down
ip route add 10.20.0.0/16 via 10.100.0.11 dev eth0
```

Как можно заметить, мы указали неверный маршрутизатор в качестве следующего на пути пакета. Убедимся командой **ip r**, что маршрутизатор **r2** будет пересылать пакеты с получателями в сети 10.20.0.0/16 маршрутизатору **r1**.

Теперь IP-пакеты до сети 10.20.0.0/16 будут ходить так, как показано на рисунке 3.3 и всё готово для опыта с «вечным» IP-пакетом. Таблица маршрутизации на машине **r2** сейчас должна выглядеть следующим образом.

```
10.100.0.0/16 dev eth0  proto kernel scope link src 10.100.0.12
10.20.0.0/16 via 10.100.0.11 dev eth0
10.10.0.0/16 via 10.100.0.11 dev eth0
```

Задание № 21: Истечение времени жизни пакета

Запустим на маршрутизаторе **r2** программу перехвата сетевого трафика.

```
tcpdump -tnve -i eth0
```

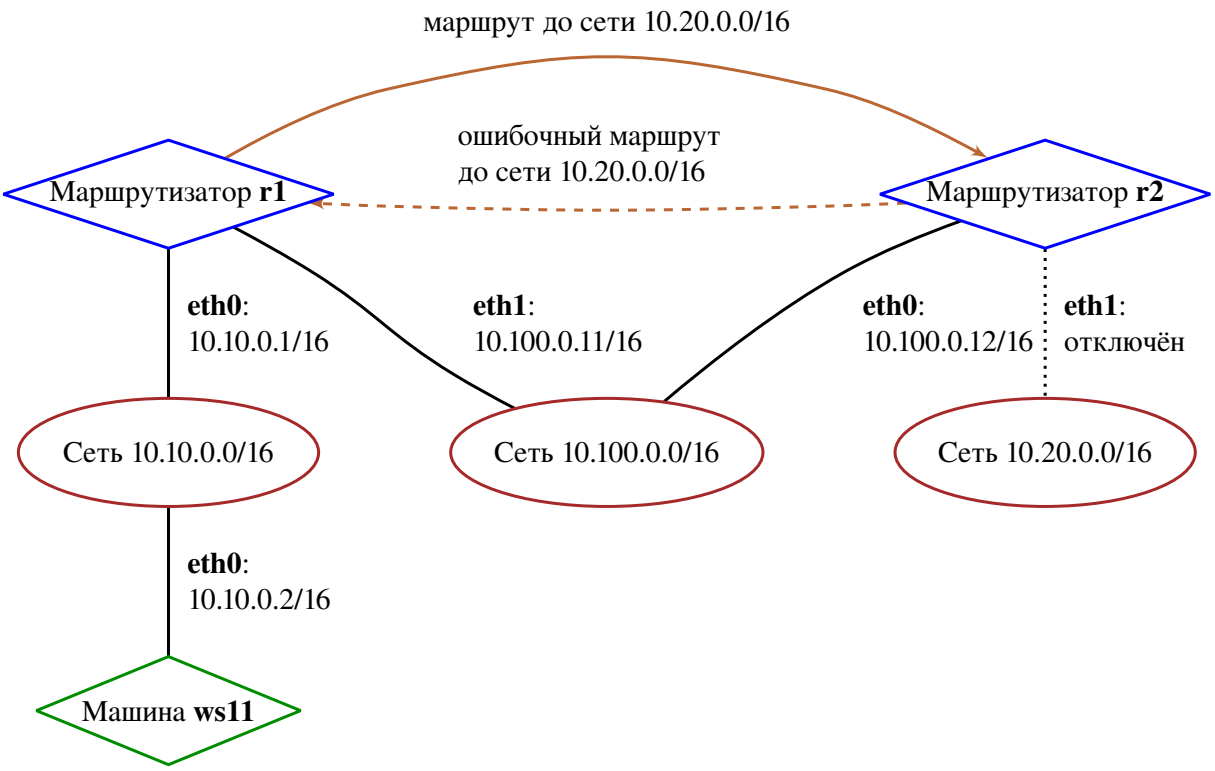


Рисунок 3.3 — Маршрутная петля

На маршрутизаторе **r1** запустим аналогичную команду — это позволит перехватить ICMP-сообщение об истечении времени жизни пакета, если его отправит маршрутизатор **r1**. С машины **ws11** пошлём эхо-запрос на адрес в «завернутой» сети.

```
ping 10.20.0.2 -c 1
```

В перехваченном на машине **r2** сетевом трафике видно, как IP-пакет с этим запросом будет многократно проходить через оба маршрутизатора со всё уменьшающимся значением поля **TTL**. В итоге маршрутизатор **r2** отправит ICMP-сообщение, извещающее отправителя пакета об истечении времени его существования.

После завершения опыта перезагрузим маршрутизатор **r2** командой **reboot** для того, чтобы вернуть его в правильное состояние.

Ниже в сокращенном виде приведены последние перемещения пакета с эхо-запросом по маршрутной петле и указанное ICMP-сообщение (последняя строка).

```
IP (ttl 2, ...) 10.10.0.2 > 10.20.0.2: ICMP echo request
IP (ttl 1, ...) 10.10.0.2 > 10.20.0.2: ICMP echo request
IP (ttl 64, ...) 10.100.0.12 > 10.10.0.2: ICMP time exceeded in-transit
```

По MAC-адресам в полном выводе должно быть видно, что пакет перемещается в цикле между интерфейсами машин **r1** и **r2**. Таким образом,

даже при ошибочных таблицах маршрутизации ограничение времени жизни пакета позволяет избавиться от бессмысленной нагрузки на сеть.

На всякий случай отметим, что ICMP-сообщения о проблемах при маршрутизации пакетов, рассмотренные здесь и далее, возникают при обработке любых IP-пакетов, а не только IP-пакетов с ICMP-сообщениями. Мы используем эхо-запросы просто как пример полезной нагрузки протокола IP, и в случае любой иной полезной нагрузки все сообщения типа «время жизни истекло» всё равно будут, разумеется, сообщениями служебного протокола ICMP. Это замечание может показаться некоторым читателям совершенно лишним, но авторы в ходе апробации пособия пришли к выводу о его полезности.

3.8 Построение списка маршрутизаторов

Как можно увидеть из предыдущего раздела, IP-пакет проходит максимум через столько маршрутизаторов, какого его начальное значение поля **TTL**. Тогда можно получить ICMP-ответы от всех маршрутизаторов на пути до получателя пакета, если посылать IP-пакеты с возрастающими значениями этого поля.

На этой идее основана работа программы **traceroute**[1]. Следует помнить, что маршруты в IP-сетях могут меняться с течением времени, поэтому совершенно достоверно построить список маршрутизаторов на некотором пути невозможно.

Задание № 22: Построение списка маршрутизаторов на пути

Запустим на маршрутизаторе **r1** перехват сетевого трафика.

```
tcpdump -tnv -i eth0
```

Затем при помощи команды **traceroute**, отданной с машины **ws11**, можно увидеть адреса всех маршрутизаторов на маршруте до указанного получателя. В качестве получателя можно взять адрес рабочей станции **ws21**, как показано в команде ниже.

```
traceroute -n 10.20.0.2
```

Вы увидите возможный путь пакета до указанного получателя.

Ожидаемый вывод последней команды приведен ниже. Как можно увидеть, программа **tcpdump** отправляет по три пакета на каждое значение TTL, и выводит время до получения ICMP-сообщения об истечении времени жизни пакета.

```
1  10.10.0.1  0 ms  1 ms  0 ms
2  10.100.0.12  1 ms  0 ms  1 ms
3  10.20.0.2  12 ms  1 ms  3 ms
```

[1] В ОС Windows она называется **tracert**.

Часто первая из трёх попыток занимает больше времени, чем две последующие (в приведенном примере это хорошо видно в третьей строке). Это связано с тем, что на данном этапе маршрутизации потребовалось использовать сначала протокол ARP. В остальных случаях требуемый MAC-адрес уже был известен.

На маршрутизаторе **r1** можно увидеть отправляемые пакеты и отправляемые маршрутизаторами ICMP-сообщения. Отметим, что отправляемые пакеты содержат UDP-датаграмму — именно их посылает программа **traceroute**.

3.9 Фрагментация IP-пакетов

Как упоминалось в главе 1, IP-пакет передаётся внутри ровно одного кадра протокола канального уровня. Если величина MTU сегмента сети не позволяет передать некоторый IP-пакет целиком, то необходимо либо разделить его на фрагменты, либо не пытаться посылать настолько большие пакеты. Первый подход называется IP-фрагментацией: при невозможности послать пакет целиком будет послано несколько IP-пакетов, являющихся его фрагментами. Каждый фрагмент является, с точки зрения маршрутизации, вполне самостоятельным пакетом. Для сборки фрагментов в единое целое (дефрагментации) в заголовке существует два поля: **id** и **fragment_offset**, а также флаг **MF**.

Отметим, что поле смещения фрагмента имеет длину всего 13 бит (три бита пришлось выделить на флаги (рисунок 3.2), а пакет в целом — 16 бит. Ясно, что смещение должно иметь возможность хранить примерно такие же значения, как и максимальная длина пакета, поэтому смещение в заголовке хранится не в байтах, а в октетах! Чтобы посчитать смещение в байтах, его следует сдвинуть на три разряда влево, дополнив до 16 бит (или умножить на 8). По этой причине фрагменты всегда начинаются со смещения, кратного 8 байтам, а длина предыдущего фрагмента также не может быть произвольной. Это ограничение учитывается при выборе длины фрагментов.

Как видно по заголовку пакета на рисунке 3.2, в нём нет поля длины исходного пакета до его фрагментации (поле **total_length** указывает на длину самого фрагмента вместе с заголовком). О размере исходного пакета можно узнать только в момент получения последнего фрагмента (признаком последнего фрагмента является отсутствие флага **MF**). Таким образом, если фрагменты приходят в типичном (хотя и негарантируемом) порядке от первого к последнему, то ядро ОС маршрутизатора не может сразу выделить буфер нужного размера для его дефрагментации. С другой стороны, динамическое увеличение такого буфера в ядре ОС является плохой идеей с точки зрения производительности. Связано такое решение было с тем, что предполагалась пересылка фрагментов до их получателя,

а не их дефрагментация на маршрутизаторе. В реальной жизни поведение маршрутизатора, как мы увидим далее, может быть как раз противоположным.

Задание № 23: Изменение величины MTU сегмента сети

Для изучения фрагментации IP-пакетов можно уменьшить величину MTU одной из сетей, например сети между маршрутизаторами. Это можно сделать как в файле конфигурации, так и «на лету» следующей командой на маршрутизаторе **r1**.

```
ip link set dev eth1 mtu 576
```

Поскольку все подключенные к одному сегменту сетевые интерфейсы должны иметь одинаковое представление о величине MTU сегмента, то изменим также величину MTU на машине **r2** следующей командой.

```
ip link set dev eth0 mtu 576
```

Командой **ip l** убедитесь, что величина MTU у обоих интерфейсов, подключенных к сети 10.100.0.0, одинакова (рисунок 3.1). Отметим, что мы всегда можем изменить представление о величине MTU всех подключённых к сегменту машин, пока это значение не превышает реальное MTU оборудования и пока он не меньше 576 байт (это минимальное MTU для сетей TCP/IP).

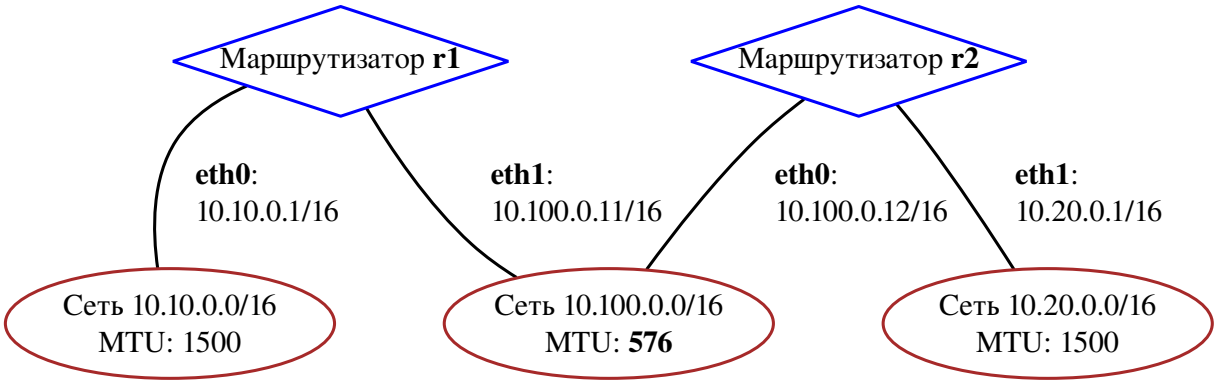


Рисунок 3.4 — Система с разными значениями MTU сетей

Величины MTU сетей сетей в нашей системе в настоящий момент показаны на рисунке 3.4. Вначале мы проведем опыт, показывающий IP-фрагментацию в том виде, как она предполагалась первоначальным стандартом [9]. Поскольку с тех пор фрагментация была признана вредной вещью, то для начала нам надо отключить метод борьбы с ней, основанный на отсылке IP-пакетов с флагом «don't fragment» и известный как PMTU (англ. *Path MTU*) — его мы рассмотрим позднее. Если этого не сделать, то отправляющая слишком большой пакет машина получит ICMP-сообщение о необходимости фрагментации, а пакет будет уничтожен. Отметим, что

механизм PMTU не является единственным способом борьбы с фрагментацией, как мы увидим в главе 8.

Задание № 24: Отключение механизма PMTU

Убедимся, что IP-пакеты сейчас отправляются с флагом **DF**.

Отключим на машине **ws11** механизм борьбы с фрагментацией следующей командой (кроме того она показывает, как можно менять параметры ядра Linux).

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

Борьба с фрагментацией теперь отключена и если сейчас с машины **ws11** послать машине **ws21** один ICMP-пакет с данными длиной 1000 байт, то до маршрутизатора **r1** он дойдёт как один IP-пакет, а до маршрутизатора **r2** — как два IP-пакета.

Задание № 25: Фрагментация IP-пакетов маршрутизатором

Убедимся, что маршрутизатор **r1** разобьёт пакет на два фрагмента, и выясним, дойдут ли они до машины **ws21** в виде собранного маршрутизатором **r2** пакета или же по отдельности. Начнём на маршрутизаторе **r1** перехват сетевого трафика на интерфейсе **eth0**.

```
tcpdump -tnv -i eth0 icmp
```

Точно такую же команду запустим на машинах **r2** и **ws21**. Теперь отправим с машины **ws11** единственный ICMP-пакет командой **ping**, указав в параметре **-s** размер дополнительных данных в эхо-запросе. Это позволит превысить величину MTU.

```
ping -c 1 -s 1000 10.20.0.2
```

Изучив результаты перехвата сетевого трафика на трёх машинах (**r1**, **r2**, **ws21**), мы увидим, что при проходе маршрутизатора **r1** эхо-запрос был фрагментирован: на **r2** перехвачены уже фрагменты. Первый фрагмент имел флаг **MF** (обозначается как **flags [+]**), а второй — ненулевое поле **offset**.

Затем запрос дефрагментирован на маршрутизаторе **r2**, поскольку на машину **ws21** пришел единственный собранный пакет. Эхо-ответ аналогичным образом был разбит на маршрутизаторе **r2** и был собран обратно на **r1**.

Обратите внимание — флаг **DF** не был использован, во всех пакетах указано отсутствие флагов или только флаг MF.

```
IP (... offset 0, flags [none], proto ICMP (1), length 1028)
```

Отметим, что полная информация об ICMP-сообщении выводится только рядом с первым фрагментом, но не со вторым. Хотя каждый фрагмент имеет собственный заголовок, передаваемое в нём сообщение вышестоящего протокола (например, ICMP, UDP или TCP) при фрагментации просто делится на части, в силу чего заголовок вышестоящего протокола оказывается в первом фрагменте. По этой причине перехватчик сетевого трафика для всех остальных фрагментов сообщает только вышестоящий протокол (он указан в поле IP-заголовка), но не выводит никакой дополнительной информации о нём.

В этом опыте мы наблюдали не вполне очевидную особенность ядра Linux, используемого нашими маршрутизаторами: ядро *всегда* собирает фрагменты при получении, а разбивает только при необходимости (недостаточное значение MTU интерфейса, через который пакет покидает маршрутизатор). Со стороны это выглядит так, как будто маршрутизаторы стремятся собрать фрагменты как можно раньше, избегая передачи их по сети (что, кстати, тоже разумная идея). Поведение маршрутизаторов, основанных не на ядре Linux, может отличаться от увиденного в опыте.

3.10 Определение величины MTU для пути

Фрагментация крайне вредна по целому ряду причин, которые будут более подробно описаны позднее. Выясним, как работает механизм нахождение минимального значения MTU для всего пути, известный как PMTU.

Задание № 26: Механизм PMTU

На машине **ws11** включим механизм PMTU следующей командой.

```
echo 0 > /proc/sys/net/ipv4/ip_no_pmtu_disc
```

Перехватим трафик на трёх машинах точно так же, как и в прошлом задании. С машины **ws11** пошлём эхо запрос.

```
ping -c 1 -s 1000 10.20.0.2
```

Как видно, посланный пакет не прошёл дальше маршрутизатора **r1**, который послал следующее сообщение о его кончине.

```
10.10.0.1 > 10.10.0.2: ICMP 10.20.0.2 unreachable - need to frag (mtu 576)
```

Как и ожидалось, проблема повтора такого пакета является проблемой транспортного уровня, причём протокол UDP она тоже не беспокоит, и только протокол TCP это сделает. В случае протокола ICMP нам остаётся только послать этот пакет еще раз — как можно будет увидеть на маршрутизаторе **r1**, два фрагмента уйдут сразу с машины **ws11**.

```
IP (... offset 0, flags [+], proto ICMP (1), length 572)
IP (... offset 552, flags [none], proto ICMP (1), length 476)
```

> Почему же на этот раз пакет был фрагментирован заранее? Как показывает вывод команды **ip r show cache**, отданной на машине **ws11**, ICMP-ответ о необходимости фрагментации был использован ядром ОС, и значение MTU было сохранено: в выводе параметров пути до адреса 10.20.0.2 указано **mtu 576**. В итоге пакет с эхо-запросом был фрагментирован ещё до выхода в сеть.
>
> Если же и при второй попытке был послан полный пакет, то это обычно означает,что время кеширования величины MTU маршрута истекло, поскольку между попытками прошло слишком много времени. В этом случае просто повторите попытку отправки ещё раз.

Отметим, что в этих фрагментах уже нет флага **DF** — его наличие у фрагмента, очевидно, является логическим противоречием. Таким образом, если на пути величина MTU падает дважды, то механизм PMTU в случае ICMP-пакетов учитывает только первое «понижение» и не определит фактический минимальный MTU всего пути. В главе 8, посвященной протоколу TCP, мы вернемся к механизму PMTU ещё раз — возможно, там он будет «рекурсивным».

Отметим, что в силу особенности ядра Linux маршрутизатор **r1** на самом деле не перешлет пакеты-фрагменты по одному. Сначала он проведет дефрагментацию, склеив оба полученных фрагмента, а потом разобьет снова. Таким образом, в проведенном опыте включение механизма PMTU по сути ничего не изменило.

Для восстановления исходного значения MTU при желании можно использовать те же команды, которыми оно был изменено (но со значением MTU, равным 1500), или перегрузить оба маршрутизатора. Для дальнейших опытов можно оставить и уменьшенное значение.

3.11 Использование протокола ICMP при маршрутизации

К настоящему моменту мы увидели, как используется протокол ICMP при истечении времени жизни IP-пакета на маршрутизаторе. Также был проведен опыт, показывающий использование протокола ICMP при определении величины MTU пути.

Использование протокола ICMP при маршрутизации не ограничивается двумя указанными применениями. Двумя часто встречающимися случаями является сообщение о недостижимости сети и о недостижимости узла. Первый из них можно наблюдать, если очередной маршрутизатор не нашел в своей таблице маршрутизации соответствующей записи и не смог найти таким образом IP-адрес следующего маршрутизатора, второй — при невозможности найти MAC-адрес следующего узла маршрута при известном IP-адресе.

Задание № 27: Отсутствие получателя в таблице маршрутизации

Запустим на маршрутизаторе **r1** перехват сетевого трафика, проходящего через интерфейс **eth0**.

```
tcpdump -n -i eth0 icmp
```

Поскольку мы не указали ключ **-t**, то вывод программы **tcpdump** будет начинаться с времени события.

Выполним следующую команду на машине **ws11**.

```
ping -c 1 10.30.0.111
```

Мы увидим ситуацию, когда маршрутизатор не может определить, куда дальше отправить полученный пакет, поскольку в его таблице маршрутизации нет подходящей записи. Как видно, маршрутизатор сразу отправляет следующее ICMP сообщение.

```
IP 10.10.0.1 > 10.10.0.2: ICMP net 10.30.0.111 unreachable, length 92
```

Следует отметить, что если бы наши маршрутизаторы имели маршруты по умолчанию, то сообщение об отсутствии сети вообще бы не было получено, а вместо него пришло бы сообщение об истечении времени жизни пакета. По этой причине использование таких маршрутов на маршрутизаторах в данной работе вредно. В самостоятельных заданиях в данной главе по этой же причине не рекомендуется использовать маршрут по умолчанию.

Для наблюдения сообщения о недостижимости узла нужно послать пакет через маршрутизатор в существующую сеть, но несуществующему IP-адресу.

Задание № 28: Отсутствие получателя в сегменте сети

Запустим на маршрутизаторе **r1** перехват сетевого трафика, как написано ранее. На маршрутизаторе **r2** нужно перехватывать трафик на интерфейсе **eth1**, что можно сделать следующей командой.

```
tcpdump -n -i eth1
```

Затем пошлем эхо-запрос несуществующему узлу в известной сети, выполнив следующую команду на машине **ws11**.

```
ping -c 1 10.20.0.111
```

Маршрутизатор **r2** не сможет получить MAC-адрес по известному адресу при непосредственной маршрутизации:по перехваченным на **r2** пакетам видно, что маршрутизатор не получил ответ на ARP-запрос. Поэтому отправил ICMP-сообщение о недоступности узла, оно будет выве-

дено при перехвате трафика на маршрутизаторе **r1**. Обратите внимание, кому оно адресовано.

```
IP 10.100.0.12 > 10.10.0.2: ICMP host 10.20.0.111 unreachable, length 92
```

Таким образом, мы увидели четыре наиболее частых случаев применения протокола ICMP маршрутизатором.

3.12 Контрольные вопросы

Для самоконтроля полученных знаний рекомендуется ответить на следующие вопросы.

1) Сколько кадров с запросами MAC-адреса отправляется через сетевой интерфейс за один сеанс протокола ARP: один единственный или по одному на каждую машину сегмента?

2) Проверяет ли протокол IP контрольную сумму передаваемых данных?

3) Какие столбцы есть в маршрутной таблице? Для чего и как она используется?

4) Что такое маршрут и путь в случае IP-маршрутизации?

5) Чем различаются прямая и косвенная IP-маршрутизация?

6) Как используется указанный в маршруте IP-адрес при косвенной маршрутизации?

7) Какой адрес (MAC, IP, или оба) маршрутизатора будет фигурировать в маршрутизируемом сообщении при косвенной маршрутизации?

8) Какие адреса (MAC/IP, получатель/отправитель) изменятся в транзитном пакете при прохождении через маршрутизатор?

9) Компьютер получил кадр, MAC-адрес отправителя которого находится в кеше ARP, но IP-адрес которого не совпадает с сохраненным в кеше. Следует ли изменить адрес в кеше, добавить в кеш новую запись, или кеш менять не следует? Зависит ли это от IP-адреса отправителя полученного пакета?

10) Может ли в маршрутной таблице некоторого компьютера быть указан адрес маршрутизатора, не являющегося его соседом,то есть не принадлежащий ни одной из непосредственно подключенных к компьютеру сетей?

11) Что происходит с таблицей маршрутизации при отключении сетевого интерфейса?

12) IP-пакет отправляется с **ws11** на **r2** через **r1** в сети на рисунке 3.1. MAC-адреса каких интерфейсов будут указаны в полях получателя и отправителя кадров Ethernet, в которые будет вложен этот пакет?

13) Для чего служит поле TTL в заголовке пакета? С каким начальным значением этого поля выходит пакет в нашем случае? При каком значении этого поля пакет не пересылается маршрутизатором далее?

14) Какой принцип работы программы **traceroute**?

15) Как при IP-маршрутизации используется протокол ICMP? Перечислите все случаи использования из данной главы.

16) В какой момент маршрутизатор понимает, что искомый IP-адрес отсутствует в сети?

17) Кому отправляются ICMP-сообщения о недоступности сети и недостижимости узла?

18) Для чего нужна фрагментация IP-пакетов и как она работает?

19) Как по заголовку IP-пакета можно понять, что он является фрагментом?

20) В опыте мы видим, что первый фрагмент получился чуть меньше, чем позволяет MTU (572 байта против 576 байтов). С чем это связано?

21) Чем плоха фрагментация IP-пакетов? На чём основывается борьба с нею в этой работе?

22) Кто должен повторить данные, уничтоженные вместе с не подлежащим фрагментации пакетом?

23) Пакет с одним ICMP-сообщением был разбит на два фрагмента размером около 500 байт. Какие из фрагментов содержат IP-заголовок? ICMP-заголовок?

24) Почему даже в случае использования PMTU в отправляемых пакетах-фрагментах не установлен флаг **DF**?

25) Может ли в сети TCP/IP путь между двумя узлами меняться с течением времени? Может ли величина MTU пути меняться также?

3.13 Выполнение самостоятельной работы

Для выполнения самостоятельных заданий необходимо настроить сеть с топологией, указанной в полученном варианте. В задании не указаны конкретные IP-адреса, только адреса и маски сетей. Присваивание IP-адресов вручную в соответствии с полученным заданием часто подвержено ошибкам. Поэтому мы сначала подготовим шаблон сети на основной машине, проверим его на корректность, и только затем запустим виртуальные машины.

Скачаем и распакуем архив с шаблоном работы следующими командами.

```
mkdir -p ~/tcp-ip; cd ~/tcp-ip
wget -r http://ftp.iu7.bmstu.ru/nets/lab-ip-hw.tar.gz
rm -rf lab-ip-hw
tar -xvf lab-ip-hw.tar.gz
cd lab-ip-hw
```

Как можно увидеть, что в каталоге **lab-ip2/net** имеются следующие файлы и каталоги:

– подкаталоги файлов виртуальных машин, имена которых совпадают с именами машин;

– файл топологии (**lab.conf**), который связывает интерфейсы машин с сегментами сети;

– файлы начальной инициализации виртуальных машин (имеют расширение **startup** и имя виртуальной машины).

Всего в шаблоне имеются заготовки для четырёх маршрутизаторов (**r1** – **r4**) и трёх рабочих станций (**ws1**, **ws2**, **ws3**). Вам следует удалить каталоги и startup-файлы машин, которые не нужны для вашего варианта задания.

Затем нужно задать в файле **lab.conf**, какие интерфейсы виртуальных машин связаны с какими сегментами сети. Имена сегментов должны начинаться с буквы и содержать буквы и цифры, в остальном же они могу быть произвольными. Например, следующие две строки в этом файле означают, что интерфейс **eth1** машины **r1** и интерфейс **eth0** машины **ws2** подключены к одному и тому же виртуальному сегменту сети с условным именем **lan30**.

```
r1[1] = lan30
ws2[0] = lan30
```

Рекомендуется давать условны имена сегментам в соответствии с адресами, которые предполагается присвоить подключённым к ним сетевым интерфейсам. Например, для сети 10.40.0.0/16 разумно выбрать имя сегмента **lan40**.

После редактирования файла **lab.conf** до запуска виртуальных машин нужно присвоить их сетевым адаптерам IP-адреса в файлах **interfaces**. Это позволит воспользоваться утилитой, которая проверит, что все IP-адреса в одном сегменте действительно относятся к одной IP-сети — в нашей простой системе это требование должно выполняться. Для присвоения IP-адреса нам нужно отредактировать файлы **etc/network/interfaces** во всех каталогах виртуальных машин.

Помимо IP-адресов и масок сетей можно сейчас же указать маршрутизаторы по умолчанию и правила добавления маршрутов — при построении отчёта они будут также проверены этой жде утилитой. При желании это можно сделать и после, внутри запущенных виртуальных машин.

После редактирования всех файлов **interfaces** можно запустить утилиту проверки настроек сети. Для этого перейдем в каталог с отчётом и выполним там команду **make** для сборки отчёта.

```
cd ~/tcp-ip/lab-ip-hw/report
make
```

На первом шаге сборки отчёта будет проверена конфигурация сети на уровне сегментов и адресов и, в случае успеха, построен рисунок с

её топологией. Вы наверняка увидите сообщения, свидетельствующие об ошибках в настройках интерфейсов или файле топологии сети. Вам следует исправлять выявленные ошибки и повторять попытки сборки отчёта до успеха.

В случае успеха при очередном запуске команды **make** в каталоге **report** появится файл **report.pdf**, в котором вы увидите изображение сети. Если эта картинка соответствует вашему варианту задания — можно переходить к запуску виртуальных машин. Воспользуемся для этого командой **ltabstart**.

```
ltabstart -d ~/tcp-ip/lab-ip-hw/net
```

После запуска виртуальных машин следует добавить в файл конфигурации статические маршруты и маршрут по умолчанию, как было показано в разделах 3.3 и 3.4, если вы не сделали этого ранее. Отметим, что теперь файл **/etc/network/interfaces** меняется уже внутри виртуальной машины и в этом случае следует перегрузить сетевые настройки.

После завершения настройки вы должны убедится с помощью команды **traceroute**, что пути до всех машин соответствуют ожидаемым, а если это не так — внести необходимые исправления в маршрутные таблицы через редактирование всё того же файла настроек сети.

Маршрутные таблицы должны быть заполнены так, чтобы путь до каждой сети проходил через наименьшее число маршрутизаторов. В частности, некоторые рабочие станции подключены к сети, в которой имеются более одного маршрутизатора (например, это машина **ws2** в первом варианте). Настройки такой машины не должны ограничиваться маршрутом по умолчанию, поскольку это нарушит требование минимизации длины путей.

После этого необходимо заполнить отчёт о выполнении работы (файл **report/tex/report.tex** на основной машине). Для этого нужно выполнить опыты, аналогичные указанным в этой главе, и включающие в себя наблюдение за :

- работой протокола ARP;
- прямой и косвенной маршрутизацией;
- временем жизни пакетов;
- фрагментацией пакетов;
- использованием протокола ICMP при маршрутизации.

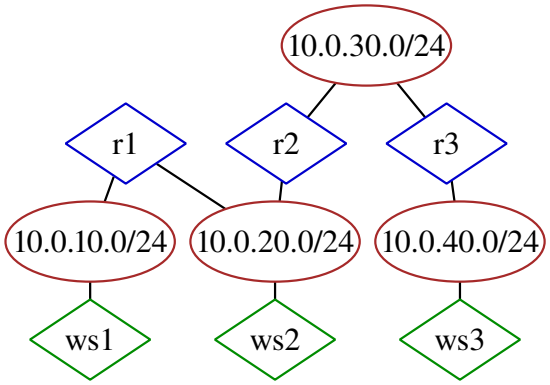
Для создания итогового pdf-файла с отчётом следует выполнить всё ту же команду **make** в каталоге **report**.

3.14 Варианты заданий для самостоятельной работы

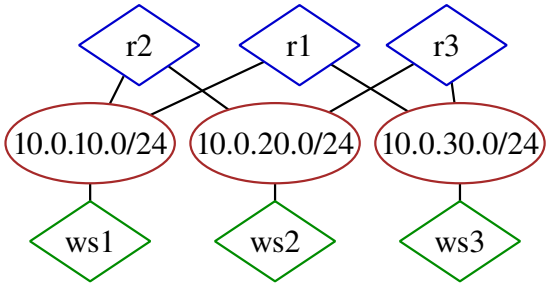
В вариантах указаны адреса и маски сетей, а также подключённые к ним машины. IP-адреса конкретных интерфейсов следует выбрать самосто-

ятельно с учетом заданных адресов сетей. По традиции следует выдавать маршрутизатору первый адрес хоста (например, 10.10.0.1/16) в сетях, где он является единственным маршрутизатором.

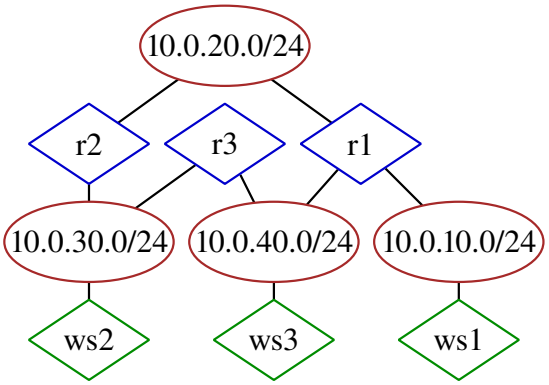
Вариант № 1



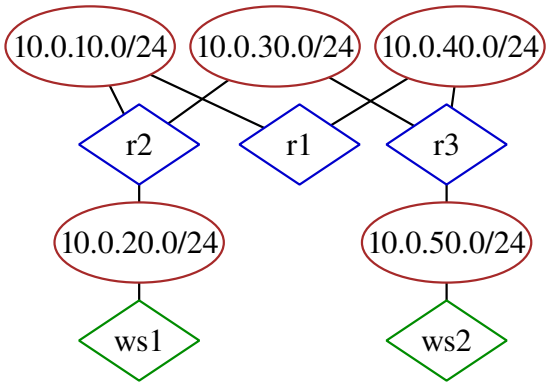
Вариант № 2



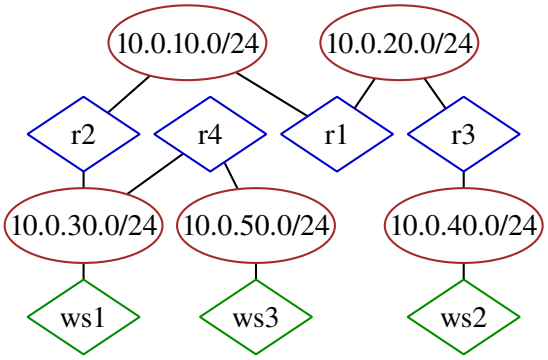
Вариант № 3



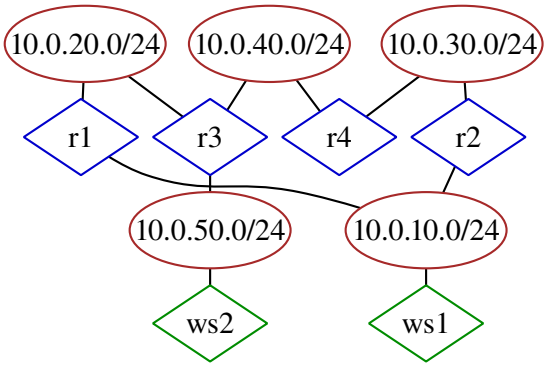
Вариант № 4



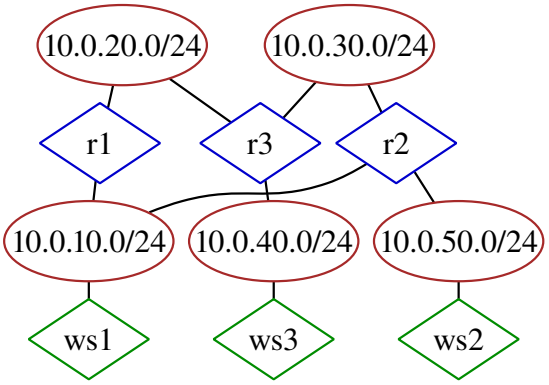
Вариант № 5



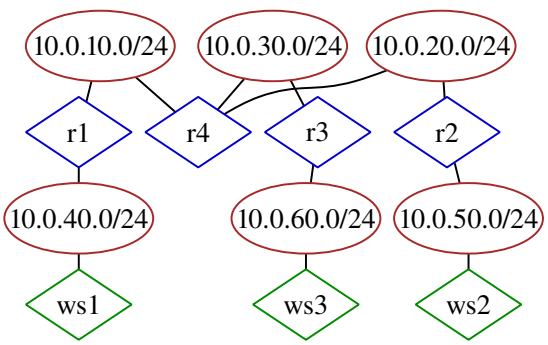
Вариант № 6



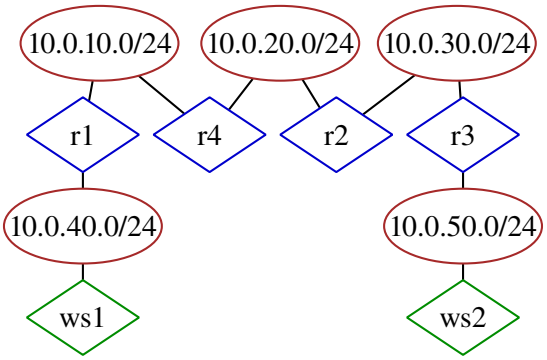
Вариант № 7



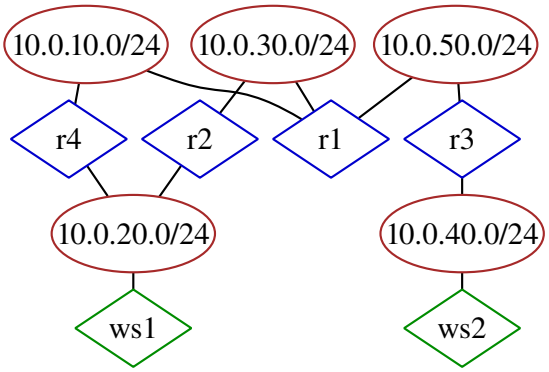
Вариант № 8



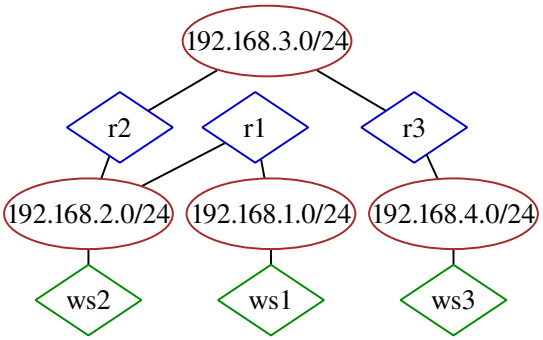
Вариант № 9



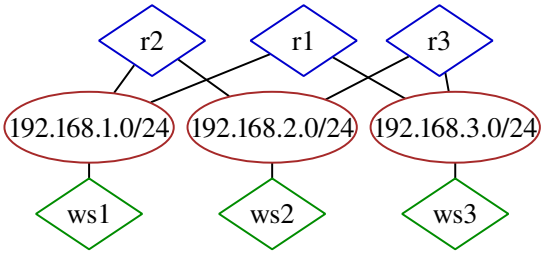
Вариант № 10



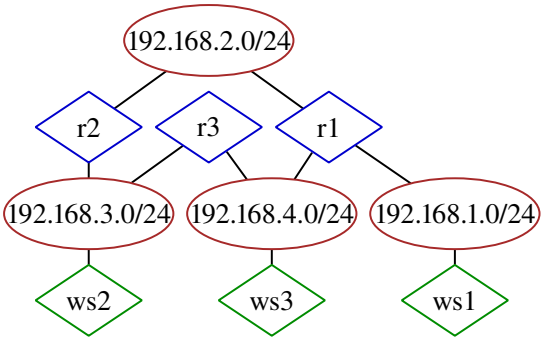
Вариант № 11



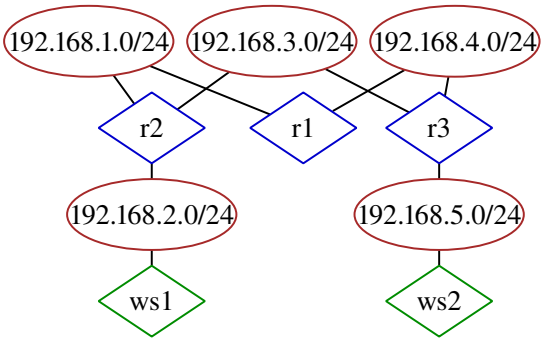
Вариант № 12



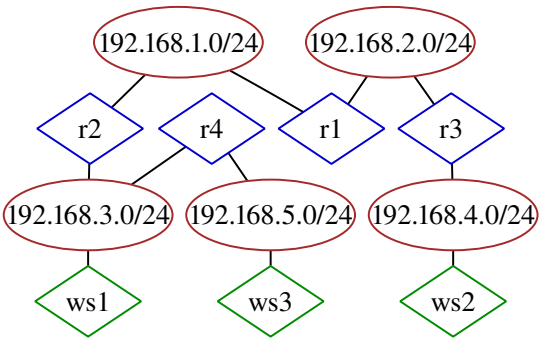
Вариант № 13



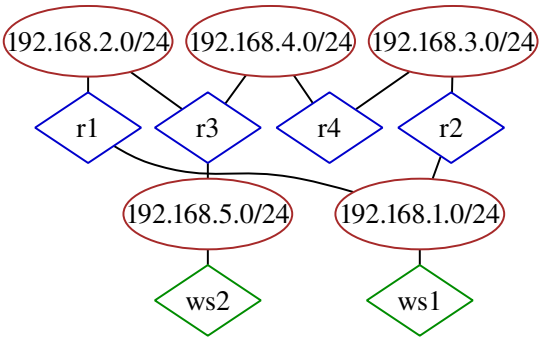
Вариант № 14



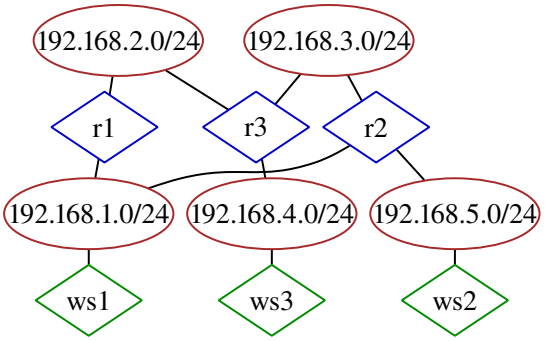
Вариант № 15



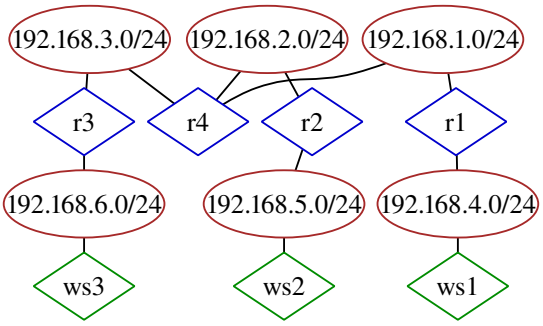
Вариант № 16



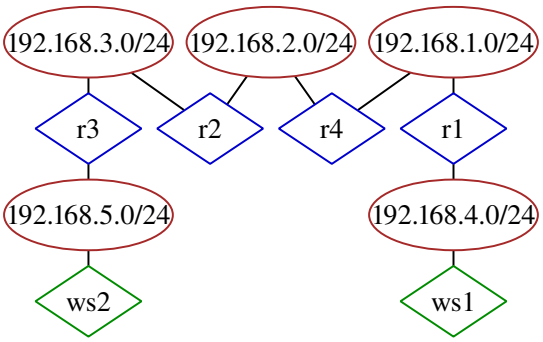
Вариант № 17



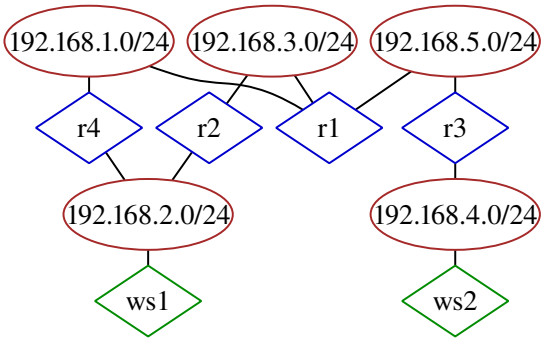
Вариант № 18



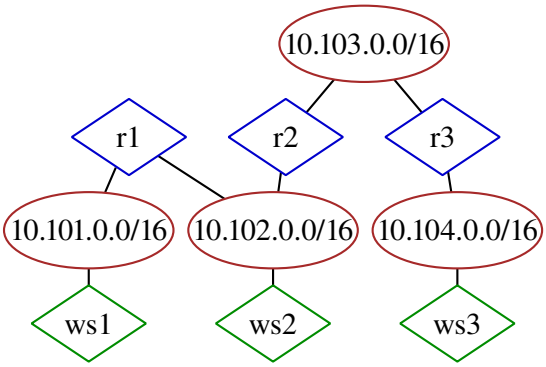
Вариант № 19



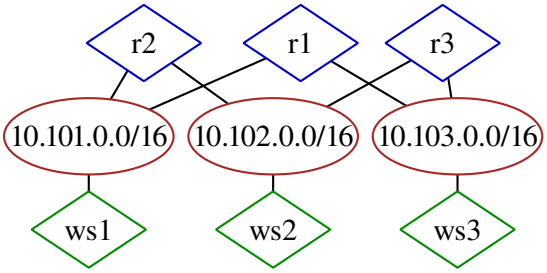
Вариант № 20



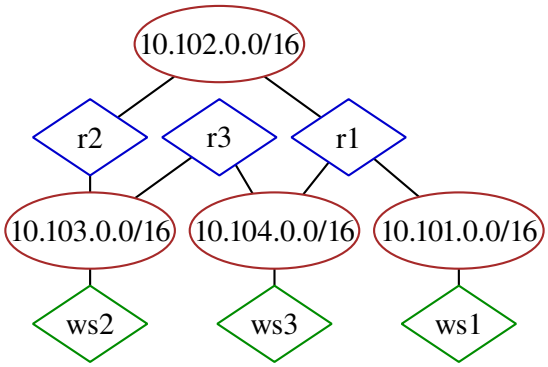
Вариант № 21



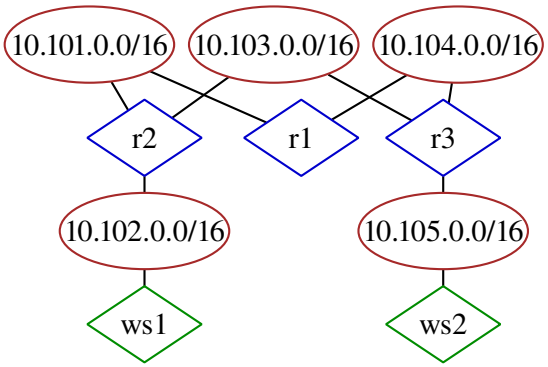
Вариант № 22



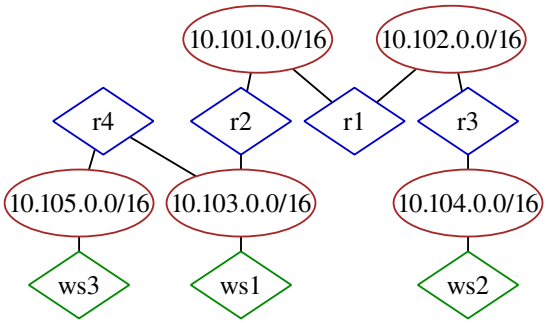
Вариант № 23



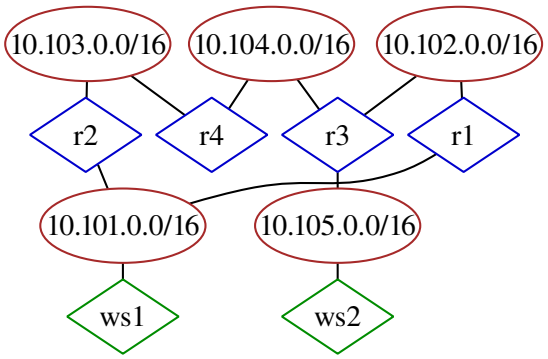
Вариант № 24



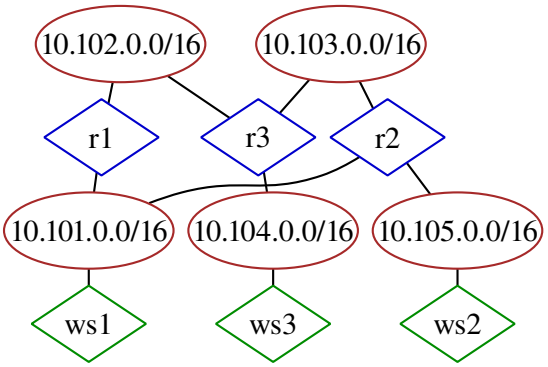
Вариант № 25



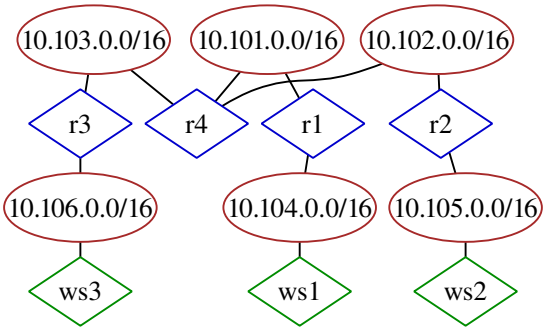
Вариант № 26



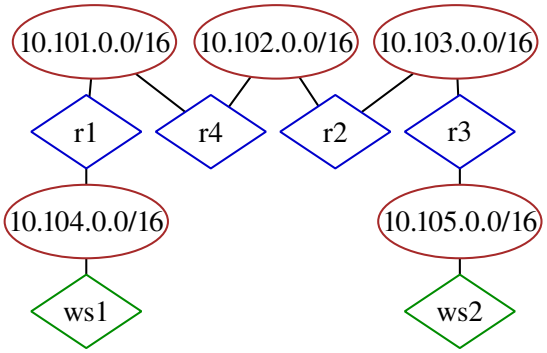
Вариант № 27



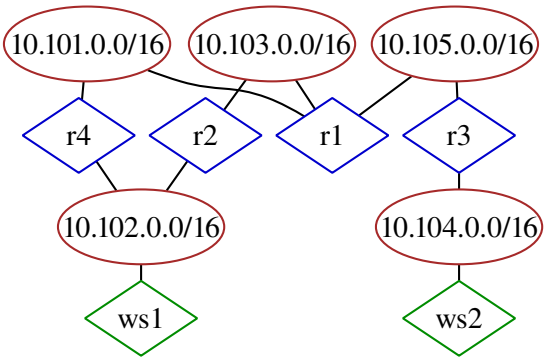
Вариант № 28



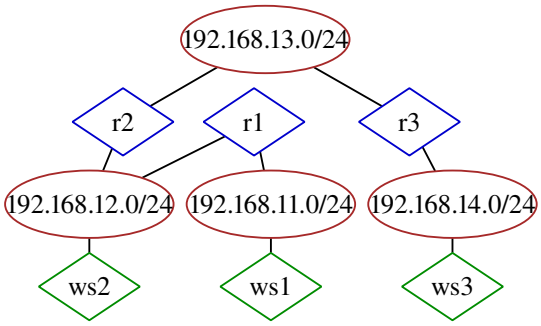
Вариант № 29



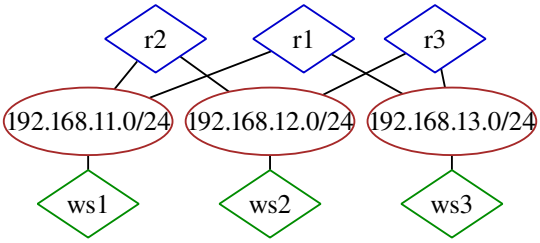
Вариант № 30



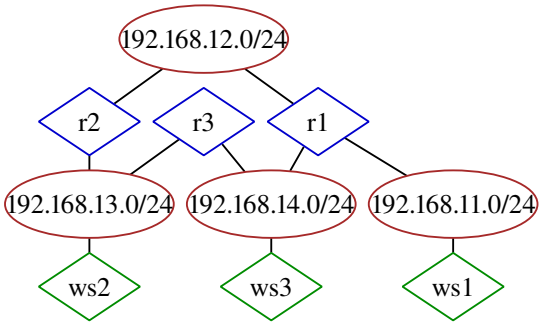
Вариант № 31



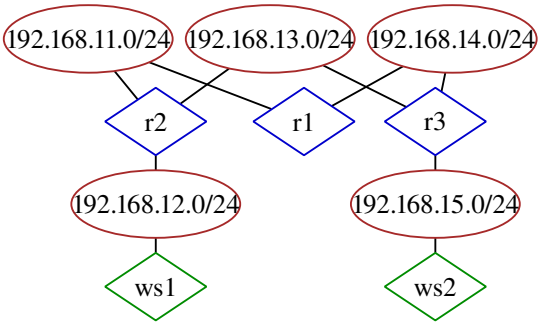
Вариант № 32



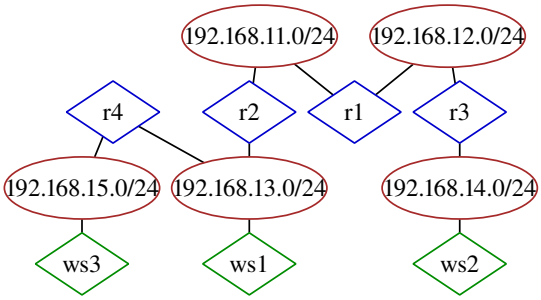
Вариант № 33



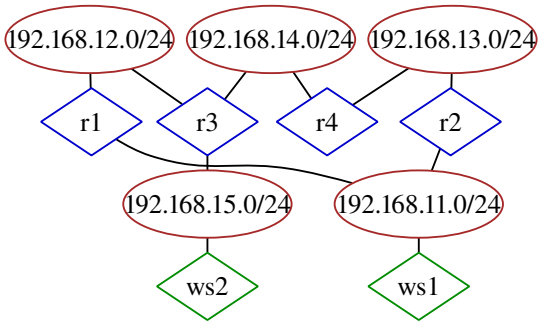
Вариант № 34



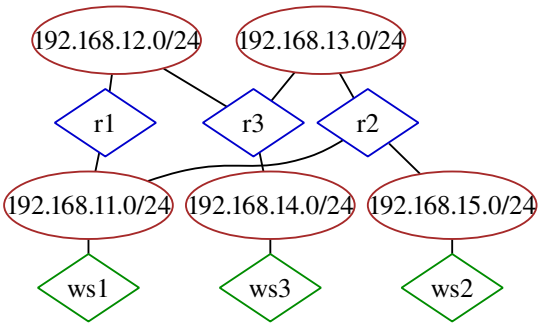
Вариант № 35



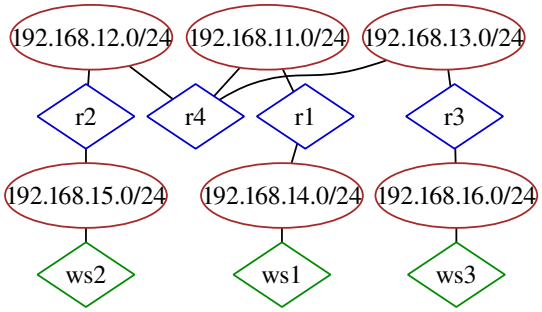
Вариант № 36



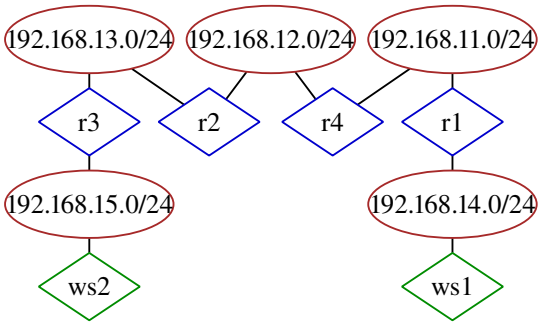
Вариант № 37



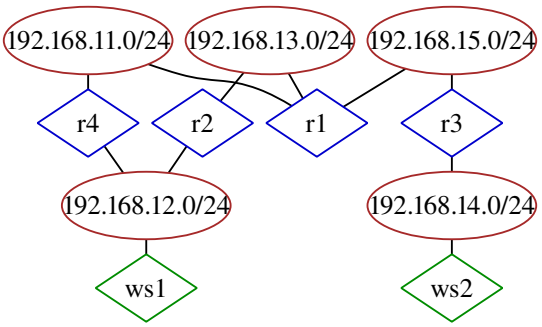
Вариант № 38



Вариант № 39



Вариант № 40



4 Коммутируемый канальный уровень и протокол STP

В предыдущих главах мы уже познакомились с функциями канального уровня с точки зрения вышестоящего сетевого протокола. В этой главе мы рассмотрим его более подробно, уделив основное внимание работе кабельного канального уровня при наличии избыточных связей между коммутаторами. За исключением последнего момента мы продолжим рассматривать канальный уровень как некоторый «чёрный ящик» и не будем вдаваться в подробности его реализации, вопросы кодирования двоичной информации и формирования физического сигнала. Эти вопросы, во-первых, решаются по-разному для различных протоколов канального уровня и, во-вторых, их конкретное решение не влияет на вышестоящие протоколы стека TCP/IP.

В конце этой главы (раздел 4.10) приведены возможные индивидуальные варианты заданий. В разделе 4.9 даны дополнительные указания по их выполнению.

4.1 Задачи канального уровня

Канальный уровень занимается передачей данных между узлами сети, соединёнными одной физической средой передачи данных.

На этом уровне решаются следующие задачи:

1) контроль доступа к разделяемой физической среде;
2) выделение в потоке физических сигналов отдельных сообщений (*кадров*);
3) кодирование данных с целью минимизировать ошибки передачи;
4) адресация узлов в рамках одного сегмента сети.

На канальном уровне решаются также служебные задачи обслуживания самого канального уровня, такие как формирование логической топологии сети.

Ниже мы рассмотрим вопросы, связанные с адресацией и передачей кадров на канальном уровне. Вторая половина главы посвящена формированию логической топологии канального уровня.

Подуровни канального уровня. В стандартах группы IEEE 802 [3], к которой, в частности, относятся стандарты Ethernet (802.3) и Wi-Fi (802.11), канальный уровень подразделяется на два слоя: MAC (media access control) и LLC (logical link control).

Протоколы канального уровня можно разделить на две группы:

– транспортные (Ethernet, PPP, WLAN);
– служебные (ARP, STP).

Первые служат непосредственно для передачи полезной нагрузки, вторые — для управления сетью. Служебные протоколы канального уровня используют транспортные протоколы канального уровня для обмена сообщениями.

4.1.1 Контроль доступа к передающей среде

Если два узла, подключённые к одной передающей среде, одновременно начнут передачу, их сигналы наложатся друг на друга. Такая ситуация называется *коллизией*. Борьба с коллизиями — одна из задач канального уровня. Существуют три способа борьбы — обнаружение, избежание и недопущение коллизий.

Первый способ может быть кратко описан как «обнаружив коллизию, замолчать и подождать случайное время». Этот способ, имеющий название CSMA/CD, применялся в ранних версиях Ethernet, рассчитанных на топологии общей шины и/или использование концентраторов. Метод избежания коллизий (CSMA/CA) применяется в беспроводных сетях Wi-Fi.

Недопущение коллизий требует особой организации сети. Современные сети на основе Fast Ethernet и Gigabit Ethernet строятся с использованием коммутаторов, так что домен коллизий ограничен строго двумя сетевыми адаптерами; кроме того, кабель используется в полнодуплексном режиме, и на практике коллизии не возникают. В пакете Netkit эмулируется протокол Fast Ethernet без возникновения коллизий.

4.1.2 Адрес канального уровня

Большинство протоколов канального уровня предлагает два варианта рассылки кадров: широковещательную (англ. *broadcast*, все узлы в сегменте получают кадр) и однонаправленную (англ. *unicast*) рассылку. Для возможности осуществления однонаправленной рассылки вводится адресация. Адреса канального уровня обычно называются *MAC-адресами* (от Media Access Control) и назначаются сетевым адаптерам.

MAC-адреса обычно имеют длину 48 бит и записываются в шестнадцатеричной нотации (например, 01:2b:45:e7:89:ab). Такой формат называется MAC-48 и стандартизован IEEE[1].

В отличие от адресов сетевого и более высоких уровней, которые описывают место адресата в некой иерархии (например, IP-адрес отражает принадлежность узла IP-сети), MAC-адреса являются «плоскими» — по ним ничего нельзя сказать о местоположении узла. Поэтому иерархическая маршрутизация на основании адресов на канальном уровне невозможна, и коммутаторы используют описанную выше схему с обучением.

[1] Есть еще EUI-48 и EUI-64.

Производители прошивают в выпускаемом оборудовании MAC-адреса, которые, теоретически, являются глобально уникальными: первые три байта соответствуют фирме-производителю и назначаются регистрационными органами IEEE, а остальные производитель должен присваивать из соображений уникальности. Некоторые программные продукты пытаются использовать это свойство, например, для привязки копии ПО к одному компьютеру. Однако, известны случаи [13] выпуска партий оборудования с конфликтующими MAC-адресами. В сочетании с возможностью менять MAC-адрес вручную говорить о глобальной уникальности MAC-адресов не приходится. Глобальная уникальность, впрочем, и не является необходимым требованием к адресам, ведь их прямое назначение — идентифицировать оборудование в пределах одного сегмента сети. Поэтому в большинстве сетевых адаптеров MAC-адрес можно изменить программно.

При широковещательной рассылке используется MAC-адрес назначения, содержащий все единицы (FF:FF:FF:FF:FF:FF). Кадры с таким адресом принимаются всеми сетевыми адаптерами в сегменте. Адреса с установленным младшим битом старшего байта (например, 01:C0:12:34:56:78) используются при многоадресной рассылке. Указать, какие multicast-адреса интересуют сетевой адаптер, можно посредством драйвера; кадры с прочими multicast-адресами будут отбрасываться. Остальные адреса считаются адресами однонаправленной рассылки.

Сетевой адаптер по умолчанию отбрасывает все кадры, не адресованные ему. Это поведение можно изменить, переведя адаптер в «неразборчивый режим» (англ. *promiscuous mode*). В этом режиме адаптер перестанет отбрасывать кадры. «Неразборчивым режимом» пользуются сниффферы, поскольку он позволяет прослушивать весь трафик в одном сегменте сети.

В кадрах протокола канального уровня обычно указываются как адрес получателя, так и адрес отправителя.

4.1.3 Полезная нагрузка канального уровня. Величина MTU

В качестве полезной нагрузки (англ. *payload*) в кадрах канального уровня выступают пакеты протоколов более высокого уровня (один кадр — один пакет). Отдельное поле[1] в заголовке кадра указывает, к какому протоколу относится полезная нагрузка, чтобы узел-получатель знал, куда её передавать для обработки. Перечень возможных значений этого поля стандартизован организацией IANA [14].

Протоколы канального уровня ограничивают размер полезной нагрузки сверху. Это ограничение получило название Maximum Transmission Unit (MTU). Выбор значения MTU обычно является результатом компромисса: с одной стороны, высокое значение MTU позволяет передать боль-

[1] EtherType в Ethernet.

ший объем данных меньшим числом кадров и снижает накладные расходы (т.е. место под заголовки кадров); с другой стороны, передача больших кадров приводит к задержкам. Кроме того, ошибка при передаче большого кадра обходится дороже, чем ошибка в небольшом кадре — ведь приходится повторять передачу кадра целиком.

Величина MTU для современной версии Ethernet составляет 1500 октетов[1], для других протоколов она может быть больше или меньше (см. таблицу 4.1).

Таблица 4.1 — Значения MTU для различных протоколов канального уровня, в октетах

X.25	Ethernet (IEEE 802.3)	PPPoE	WLAN / Wi-Fi (IEEE 802.11)	Token Ring (IEEE 802.5)	FDDI
576	1500	1492	2272	4464	4352

Протоколы более высокого уровня, со своей стороны, указывают минимальное значение MTU, с которым они могут работать. Так, для IPv4 это значение составляет 68 байт, для IPv6 — 1280.

4.2 Топология сети

Топология компьютерной сети — графовая модель сети, отражающая связи между различными её элементами. Выделяют два вида топологий — физические и логические.

Физическая топология сети содержит материальные сущности — оборудование и провода. Любая конкретная компьютерная сеть имеет однозначно определяемую физическую топологию.

Обычно оборудование формализуется вершинами графа, а провода — рёбрами. Исключение составляет топология общей шины, когда к одному проводу (среде передачи) подключается произвольное число узлов; в этом случае среда передачи также может быть формализована как вершина графа.

Логическая топология сети описывает потоки данных через сеть.

Различным уровням сетевого взаимодействия в конкретной сети соответствуют различные логические топологии, абстрагирующие различные наборы деталей и придающие различный смысл вершинам и рёбрам графа.

1) Логическая топология канального уровня игнорирует концентраторы и другое оборудование физического уровня, а рёбра соответствуют сетевым адаптерам.

2) Логическая топология сетевого уровня скрывает мосты и коммутаторы за абстрактными сегментами сети.

[1] Октет — привычный для нас восьмибитный байт. Термин часто встречается в текстах сетевых стандартов, в частности в RFC.

3) При моделировании внешней маршрутизации игнорируются отдельные машины — вместо них вершины графа представляют целые «автономные системы».

4) На прикладном уровне в графе топологии зачастую достаточно наличия двух вершин — клиента и сервера, соединённых через «облако», а рёбрами здесь могут быть, например, соединения по протоколу TCP.

Таким образом, вид графовой модели определяется задачами, решаемыми конкретным уровнем или сетевым протоколом. Примеры приведены на рис. 4.1.

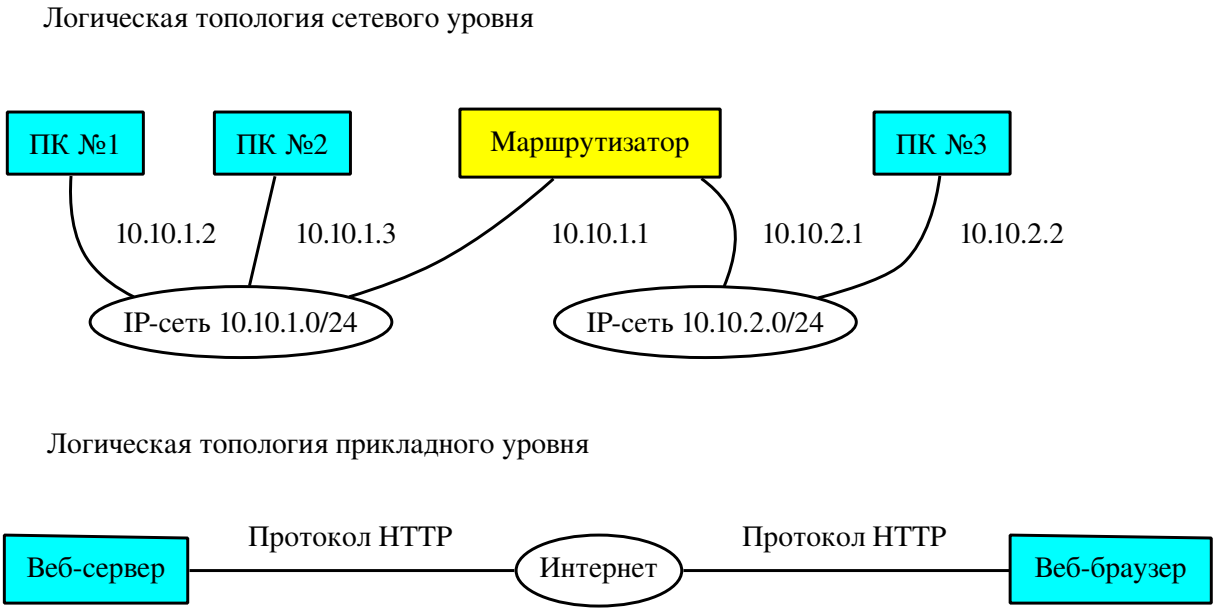


Рисунок 4.1 — Логические топологии различных уровней

4.3 Топология на канальном уровне

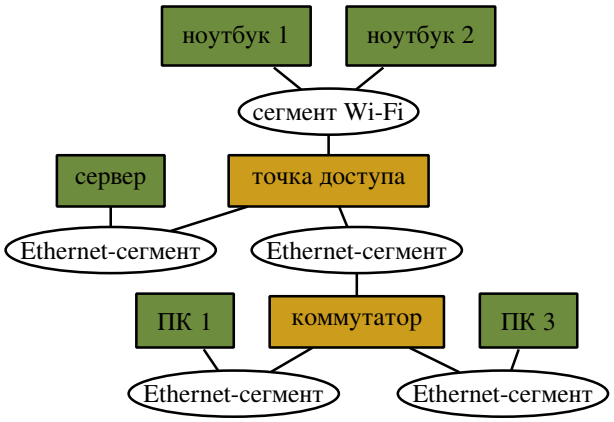


Рисунок 4.2 — Пример топологии канального уровня

При моделировании канального уровня нас интересуют отношения между узлами и сегментами сети:

– один узел может входить в несколько сегментов сети;

– один сегмент может содержать несколько узлов;

– связь «узел—сегмент» выражается в наличии у узла сетевого адаптера, подключённого к среде передачи этого сегмента.

Таким образом, вершинами графа являются узлы и сегменты сети, а рёбрами — сетевые адаптеры, соединяющие узлы с сегментами. Это означает, что топология канального уровня является двудольным графом[1].

В качестве узлов сети могут выступать:

– коммутаторы и мосты — оборудование канального уровня;

– маршрутизаторы, ПК, сетевые принтеры и т.п.

Оборудование физического уровня скрыто за сегментами сети.

Способность коммутаторов и мостов объединять различные сегменты не отражается в топологии канального уровня. Коммутаторы, мосты и объединяемые ими сегменты изображаются на графе отдельными вершинами.

Пример топологии канального уровня приведён на рис. 4.2.

4.3.1 Сетевые интерфейсы

Сетевые адаптеры в операционной системе отображаются в виде *сетевых интерфейсов* (в терминологии ОС Windows — сетевых подключений). Каждый сетевой интерфейс является либо физическим, связанный с сетевым адаптером, либо виртуальным, то есть не связанным напрямую с сетевым адаптером, а реализуемым некой программой. С сетевым интерфейсом связан стек протоколов, набор адресов и различные параметры.

В системах с ядром Linux сетевые интерфейсы имеют имена, состоящие из префикса и номера. Например, физические интерфейсы сети Ethernet называются **eth0**, **eth1** и т. д. Виртуальные интерфейсы IP-туннеля называют как **tun0**, **tun1** и т. д. Виртуальный локальный интерфейс имеет имя **lo** и с ним связан IP-адрес 127.0.0.1/8).

Пользователь может включать («поднимать») и отключать интерфейсы, назначать им сетевые адреса и задавать их параметры. Для управления сетевыми интерфейсами служат специальные утилиты. В POSIX-системах это утилиты **ifconfig** и **ip**. В Windows тоже имеется утилита командной строки (**ipconfig**).

На практике «сырыми» кадрами прикладные приложения не обмениваются, поскольку канальный уровень не имеет механизма доставки данных процессам (это механизм появляется лишь на транспортном уровне).

---

[1]Напомним, что **двудольным графом** называется граф $G(V,E)$, в котором множество вершин V является объединением непересекающихся множеств V_1 и V_2, а рёбра (дуги) этого графа соединяют пары вершин (a,b), в которых одна вершина принадлежит множеству V_1, а другая — множеству V_2 (то есть не существует ребра, соединяющего вершины из одного множества).

Задание состоит из трёх частей, выполняемых последовательно в одной и той же виртуальной сети:

– запуск сети и настройка сетевых интерфейсов;
– отправка и перехват Ethernet-кадров;
– объединение сегментов сети с помощью мостов.

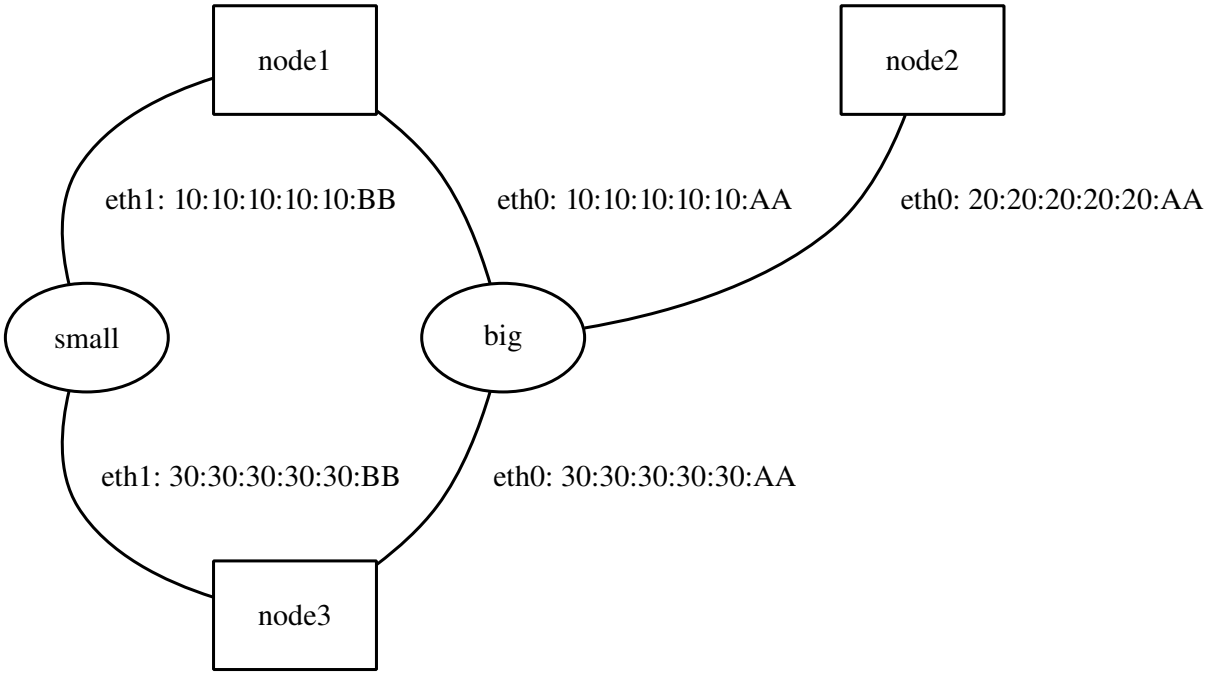


Рисунок 4.3 — Сеть для задания № 28.
Прямоугольники — рабочие станции, эллипсы — сегменты сети с одноимёнными коммутаторами. Рёбра — сетевые адаптеры машин со своими MAC-адресами.

Сеть, с которой мы будем работать в этом задании, показана на рисунке 4.3. Сеть состоит из двух сегментов — **big** и **small**; каждый из них обслуживается отдельным коммутатором. К сегменту сети **big** подключены все три машины **node1**, **node2**, **node3**, а к сегменту **small** — только **node1** и **node3**.

Запустите описанную выше сеть с использованием команд **vstart**, выполняя каждую команду в отдельном окне или вкладке эмулятора терминала:

```
$ vstart node1 --eth0=big --eth1=small
$ vstart node2 --eth0=big
$ vstart node3 --eth0=big --eth1=small
```

Дождитесь, пока все машины загрузятся.

Теперь надо включить все сетевые интерфейсы на каждой машине. Это делается командой **ifconfig <интерфейс> up** (обратите внимание на имена машин в приглашениях командной строки):

```
node1:~# ifconfig eth0 up && ifconfig eth1 up
node2:~# ifconfig eth0 up
node3:~# ifconfig eth0 up && ifconfig eth1 up
```

Чтобы просмотреть свойства работающих сетевых адаптеров, выполните на машинах сети команду **ifconfig** без параметров. Её вывод показывает в том числе и MAC-адреса соответствующих сетевых адаптеров (**HWaddr**).

```
node1:~# ifconfig
eth0      Link encap:Ethernet  HWaddr 42:28:4f:8e:8f:c3
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:12 errors:0 dropped:0 overruns:0 frame:0
          ...

eth1      Link encap:Ethernet  HWaddr c6:87:03:9a:d0:75
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:6 errors:0 dropped:0 overruns:0 frame:0
          ...
lo        Link encap:Local Loopback...
```

Сетевые адаптеры пока что имеют произвольные «фабрично прошитые» MAC-адреса. Поменяем их в соответствии с рисунком 4.3.

```
node1:~# ifconfig eth0 hw ether 10:10:10:10:10:AA
node1:~# ifconfig eth1 hw ether 10:10:10:10:10:BB
node2:~# ifconfig eth0 hw ether 20:20:20:20:20:AA
node3:~# ifconfig eth0 hw ether 30:30:30:30:30:AA
node3:~# ifconfig eth1 hw ether 30:30:30:30:30:BB
```

Теперь вы знаете, как обходить защиту программ, наивно пытающихся привязать свою копию к MAC-адресу.

4.4 Отправка и перехват сообщений.

Для просмотра передающихся Ethernet-кадров воспользуемся перехватчиком пакетов («сниффером») **tcpdump**. Эта программа прослушивает сетевые интерфейсы на машине, на которой запущена, и выводит в консоль информацию о проходящих пакетах. На узлах **node2** и **node3** на каждый сетевой интерфейс запустите по одному экземпляру **tcpdump**.

```
node2:~# tcpdump -Ae -i eth0
node3:~# tcpdump -Ae -i eth0 &
node3:~# tcpdump -Ae -i eth1 &
```

Параметр **-A** указывает, что полезную нагрузку пакетов следует выводить в текстовом виде; параметр **-e** означает расширенный вывод полей Ethernet.

Пока что обмена пакетами по сети не происходит, и снифферы будут молчать. Займемся отправкой Ethernet-кадров. Для этой цели воспользуемся пакетом **scapy** [15], запустив его на машине **node1**.

```
node1:~# scapy
Welcome to Scapy (2.1.0)
>>>
```

Вы видите перед собой интерактивную консоль языка Python, окружение которой пополнено функциями пакета **scapy**.

Сформируем Ethernet-кадр, указав в качестве полезной нагрузки строку с приветствием (прочие поля заполнять не будем):

```
>>> p = Ether() / "hello world!"
>>>
```

Используя функцию **sendp**, отправим кадр через интерфейс **eth0**:

```
>>> sendp(p, iface="eth0")
WARNING: Mac address to reach destination not found. Using broadcast.
.
Send 1 packets.
>>>
```

Нам сообщают, что будет использована широковещательная рассылка, раз мы не соизволили указать адрес назначения. Просмотрев вывод снифферов на машинах **node2** и **node3**, убедимся, что обе машины действительно получили наш кадр:

```
09:21:05.912122 00:00:00:00:00:00 (oui Ethernet) > Broadcast, 802.3, length 26:
LLC, dsap Unknown (0x68) Individual, ssap Unknown (0x64) Response, ctrl 0x6c6c:
Information, send seq 54, rcv seq 54, Flags [Response], length 12
hello world!
```

Отправив кадр через второй интерфейс (**sendp(p, iface="eth1")**), увидим, что его получила только машина **node3**. В самом деле, машина **node2** в нашей сети не подключена к сегменту сети **smalldomain**, а широковещание на канальном уровне действует только в пределах одного примитивного сегмента.

Отправим один кадр конкретному сетевому адаптеру. Например, **eth0** на машине **node3**, который имеет адрес **30:30:30:30:30:AA**:

```
>>> p = Ether(dst='30:30:30:30:30:AA') / "hi node3!"
>>> sendp(p, iface="eth0")
.
Sent 1 packets.
>>>
```

Снифферы выведут соответствующую строчку:

```
09:25:33.297936 00:00:00:00:00:00 (oui Ethernet) > 30:30:30:30:30:AA (oui Unknow
n), 802.3, ............
```

```
........length 9
hi node3!
```

Заметим, что хотя кадр предназначался машине **node3**, его получила и машина **node2**. Вспомним, что пересылкой кадров внутри сегмента сети занимается коммутатор, который прослушивает сеть и пытается определить, машины с какими MAC-адресами подключены к его портам. Так как машина **node3** до сих пор не отправила ни одного Ethernet-кадра, коммутатор не знает, к какому его порту она подключена, и рассылает кадры во все порты.

Дадим коммутатору знать, где следует искать машину **node3**. Для этого запустим на ней инструмент **scapy** и отправим один кадр, указав MAC-адрес своего интерфейса **eth0**:

```
node3:~# scapy
>>> my_addr = get_if_hwaddr('eth0')
>>> my_addr
'30:30:30:30:30:aa'
>>> sendp(Ether(src=my_addr) / "I'm here", iface="eth0")
WARNING: Mac address to reach destination not found. Using broadcast.
.
Sent 1 packets.
>>>
```

Вернувшись на машину **node1**, отправим приветствие узлу **node3** ещё раз:

```
>>> sendp(p, iface="eth0")
.
Sent 1 packets.
>>>
```

Просмотрев вывод снифферов, убедимся, что в этот раз пакет пришел только на машину **node3**. Это свидетельствует о том, что в базе фильтрации коммутатора, обслуживающего сегмент сети **big**, появилась запись с MAC-адресом машины **node3**, и направляемые на этот адрес кадры теперь пересылаются только в один порт коммутатора.

Коммутатор регулярно чистит свою базу фильтрации, удаляя давно не обновлявшиеся записи. Если подождать некоторое время (примерно полторы минуты) и повторить отправку приветствия на узел **node3**, кадр снова будет получен двумя машинами.

Попробуем теперь отправить кадр из одного сегмента сети в другой. В качестве адаптера-получателя выберем **eth1** машины **node3**, подключённый к сегменту сети **small** и имеющий MAC-адрес **30:30:30:30:30:BB**. Чтобы убедиться, что кадр придёт именно на этот интерфейс, завершим процессы-снифферы **tcpdump** и запустим **tcpdump** только на интерфейсе **eth1**:

```
>>> # нажмите Ctrl-D для выхода из scapy
node3:~# kill `pidof tcpdump`
node3:~# tcpdump -Ae -i eth1 &
```

Отправлять сообщения будем с машины **node2**. Остановим на ней **tcpdump** нажатием Ctrl-C, запустим **scapy** и отправим кадр:

```
node2:~# scapy
>>> p = Ether(dst='30:30:30:30:30:BB') / "hi there!"
>>> sendp(p, iface="eth0")
.
Sent 1 packets.
>>>
```

Сниффер на машине **node3** молчит, чего и следовало ожидать — ведь сегменты сети **big** и **small** никак не связаны. Как вы помните из теоретической части, для объединения сегментов сети используются коммутаторы и мосты. Мы можем объединить сегменты **big** и **small**, превратив машину **node1** в мост. Преобразованием протоколов наш мост заниматься не будет, так как в Netkit эмулируется только технология Fast Ethernet, но передавать кадры между сегментами сможет.

Для управления мостами в операционной системе Linux используется команда **brctl**. Перейдя в консоль машины **node1**, завершим ранее запущенный там инструмент **scapy** (Ctrl-D) и создадим мост, соединяющий сетевые адаптеры **eth0** и **eth1**:

```
node1:~# brctl addbr br0        ## создаём мост с именем br0
node1:~# brctl addif br0 eth0   ## добавляем интерфейсы
node1:~# brctl addif br0 eth1
node1:~# ifconfig br0 up        ## запускаем мост
```

Теперь машина **node1** будет пересылать Ethernet-кадры между сегментами **big** и **small**.

Убедимся в работоспособности моста, повторив отправку кадра с машины **node2**. Созданный нами мост отказывается пересылать кадры без адреса отправителя, поэтому **src** придется задать явно:

```
>>> sendp(Ether(src=get_if_hwaddr('eth0'), dst='30:30:30:30:30:BB') /
... "hi there!", iface="eth0")
.
Sent 1 packets.
>>>
```

Сниффер на машине **node3** сигнализирует об успешном получении кадра:

```
10:13:15.910474 20:20:20:20:20:aa (oui Unknown) > 30:30:30:30:30:bb (oui Unknown
), 802.3, length 23: LLC, dsap Unknown (0x68) Individual, ssap Unknown (0x68) Re
sponse, ctrl 0x7420: Information, send seq 16, rcv seq 58, Flags [Response], len
gth 9
hi there!
```

Это значит, что мост работает как надо.

В завершение заглянем в базу фильтрации нашего моста:

```
node1:~# brctl showmacs br0
port no mac addr        is local?  ageing timer
  1 10:10:10:10:10:aa   yes        0.00
  2 10:10:10:10:10:bb   yes        0.00
  1 20:20:20:20:20:aa   no         5.74
node1:~#
```

В базе содержатся три MAC-адреса. Два из них принадлежат сетевым адаптерам самого моста (**is local = yes**), и записи с ними не устаревают. Третий адрес был занесён в базу, когда мост получил отправленное нами «приветствие», предназначавшееся машине **node3**. Значение **ageing timer** показывает возраст этой записи. Когда возраст превысит пороговое значение, запись будет удалена. По умолчанию наш мост удаляет записи через пять минут после последнего обновления.

Закончив задание, завершите виртуальные машины командами **halt**.

4.5 Проблема динамической топологии

В реальной жизни сетевое оборудование имеет свойство выходить из строя: провода перегрызаются животными, коммутаторы лишаются питания или присваиваются посторонними лицами. Вполне естественно желание обеспечить надежность сети введением избыточных, запасных связей. К сожалению, такая мера приводит к появлению циклов в топологии, что недопустимо для сетей на канальном уровне: пересылаемые кадры будут бесконечно блуждать по такой сети, моментально сводя её производительность к нулю.

Решением данной проблемы является формирование в сети так называемой *логической топологии* поверх физической. Сеть настраивается так, чтобы между любыми двумя узлами в логической топологии существовал единственный путь. Избыточные связи, не вошедшие в логическую топологию, не отключаются физически, а помещаются в пассивное состояние, в котором только наблюдают за состоянием сети. Как только в сети фиксируется повреждение логической топологии, избыточные связи переводятся в активное состояние, восстанавливая целостность сети.

Стандартным методом формирования логической топологии в Ethernet-сетях является протокол Spanning Tree Protocol.

Протокол STP (англ. *Spanning Tree Protocol*) предназначен для формирования остовного дерева в топологии локальной сети с несколькими мостами[1].

[1]Заметим, что использование термина «мост» в данном случае не подразумевает соединения сетей с различной организацией канального уровня. В данном разделе используется терминология, принятая в стандарте IEEE 802.1d, где мостом фактически считается коммутатор, реализующий STP.

После завершения построения такого дерева разрешается пересылать кадры только по рёбрам этого дерева; это гарантирует отсутствие циклов при пересылке. Протокол также позволяет адаптировать дерево к изменениям топологии.

Остовное дерево строится множеством сетевых мостов, каждый из которых обладает неполной информацией о сети, поэтому привычный алгоритм Краскала здесь неприменим. Ниже приводится описание распределённого алгоритма и протокола STP, его реализующего. Полное описание можно найти в стандарте IEEE 802.1d в редакции 1998 года [16]; редакция 2004 года [17] содержит описание расширенной версии протокола под названием RSTP вместо «классического» STP. RSTP здесь не рассматривается.

Напомним пару определений из теории графов:

Двудольный граф — граф $G(V,E)$, в котором множество вершин V является объединением непересекающихся множеств V_1 и V_2, а рёбра (дуги) этого графа соединяют пары вершин (a,b), в которых одна вершина принадлежит множеству V_1, а другая — множеству V_2. То есть, в двудольном графе не существует ребра, соединяющего вершины из одного множества.

Двудольный граф является моделью топологии компьютерной сети на сетевом уровне и на коммутируемом канальном уровне. Топология сегмента сети может быть представлена в виде двудольного графа, вершинами которого являются мосты и сегменты сети (провода или концентраторы), а рёбрами — порты мостов, подключающие их к сегментам.

Остовное дерево графа $G(V,E)$ — граф $G_{st}(V',E')$, являющийся деревом, причем $V' = V$ и $E' \subseteq E$.

Для задачи построения остовного дерева сегмента любые узлы сети, не являющиеся мостами этого сегмента (т. е. все подключенные к сегменту компьютеры) никак не учитываются в графе топологии; порты мостов, ведущие к таким узлам, также не включаются в граф.

Рёбра графа имеют веса — стоимость передачи данных. В данном графе требуется найти некоторое (не обязательно минимальное) остовное дерево.

Для построения остовного дерева элементам сети должны быть назначены следующие параметры:

– приоритеты мостов (чем ниже численное значение, тем выше приоритет);

– приоритеты портов;

– стоимости портов (чем выше скорость передачи, тем ниже стоимость).

В качестве корня остовного дерева выбирается мост с наивысшим приоритетом. При равных приоритетах в протоколе STP предпочтение вез-

де отдается меньшему MAC-адресу. Для каждого из остальных мостов выбирается путь наименьшей стоимости, соединяющий этот мост и корень дерева. Порт, через который проходит этот путь, называется *корневым портом*. При наличии нескольких кратчайших путей предпочтение отдается тому, который проходит через мост с наивысшим приоритетом; если все пути проходят через один и тот же мост, выбирается порт с наивысшим приоритетом.

В каждом сегменте сети мост, расположенный «ближе всех» к корневому мосту, считается «назначенным» (англ. *designated*) мостом этого сегмента. Порт, через который «назначенный» мост подключен к данному сегменту сети, считается «назначенным» портом сегмента.

Все порты, не являющиеся корневыми или «назначенными», исключаются из логической топологии переводом в режим блокировки и остаются в этом режиме до тех пор, пока топология сети не изменится. Корневые и «назначенные» порты переключаются в режим пересылки кадров и образуют рёбра построенного остовного дерева.

Напомним, что речь идет только о портах, соединяющих мосты между собой и входящих в граф. Порты, к которым подсоединены рабочие станции, не регулируются протоколом STP и пересылают кадры всегда.

На рисунке 4.4 показаны два различных остовных дерева, получаемых в одной и той же топологии при различных приоритетах мостов. Мосты изображены прямоугольниками со значением приоритета; сегменты[1] сети — эллипсами. Корневой мост выделен серым цветом. Рёбра графа соответствуют портам. Корневые и «назначенные» порты помечены соответственно символами RP и DP. Заблокированные порты показаны штриховой линией. Приоритеты и стоимости портов в примере одинаковы и не влияют на построение деревьев.

Таким образом, в целом по сети получаемое остовное дерево определяется значениями приоритетов и стоимостей.

Формирование логической топологии осуществляется выбором корневого моста, корневых и «назначенных» портов. Этот выбор делают сами мосты на основании имеющейся у них информации. Протокол STP обеспечивает обмен этой информацией между мостами и сходимость алгоритма построения остовного дерева к результату.

4.6 Сообщения протокола STP

Мосты обмениваются между собой сообщениями, называемыми *Bridge Protocol Data Units* (BPDU). Сообщения помещаются в кадры протокола канального уровня и рассылаются по широковещательному MAC-

[1] Весь граф тоже, по сути, является *одним* сегментом, получающимся путем объединения примитивных сегментов мостами. Во избежание путаницы в этом разделе речь ведется только о примитивных сегментах.

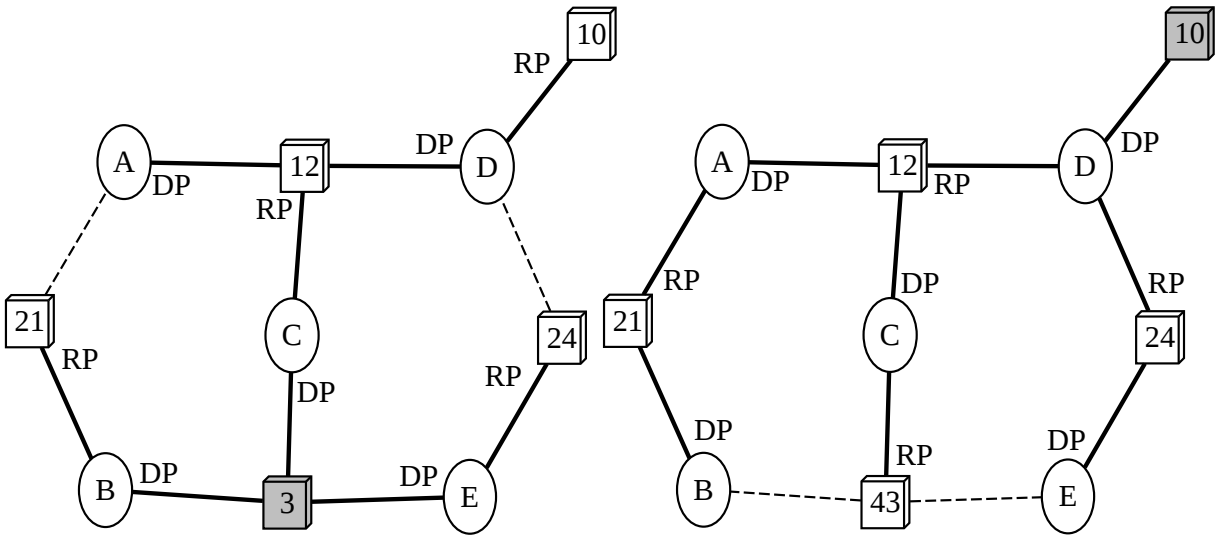


Рисунок 4.4 — Варианты остовного дерева в зависимости от приоритетов мостов

адресу (**01:80:C2:00:00:00**) в пределах сегмента сети. BPDU-сообщения несут следующую информацию:

– ID моста-отправителя (ID = приоритет + MAC-адрес);
– ID текущего корневого моста (по мнению моста-отправителя);
– ID порта (приоритет + порядковый номер), через который был выслан BPDU;
– стоимость пути от отправителя до корневого моста.

Распространение информации о сети осуществляется по следующим трём правилам.

1) Каждый мост, считающий себя корневым (а изначально все мосты считают себя таковыми), с интервалами в 2 секунды[1] рассылает BPDU, в котором указывает собственный ID в качестве корневого и нулевую стоимость пути.

2) Каждый не-корневой мост, получив BPDU с информацией, «лучшей», чем его собственная (порядок сравнения: более высокий приоритет корневого моста; меньшая стоимость пути; более высокий приоритет моста-отправителя; порта-отправителя), обновляет собственную информацию и пересылает обновленный BPDU через все свои «назначенные» порты (изменяя при этом ID отправителя и увеличивая стоимость пути). Аналогично пересылаются и BPDU с совпадающей информацией.

3) Каждый не-корневой мост, получив BPDU с «худшей» информацией, рассылает BPDU со своей собственной информацией через свои «назначенные» порты.

[1] Таймер **hello time**.

Отправляя и пересылая BPDU-сообщения, мост обновляет собственные представления о сети:

– в качестве корневого моста выбирается мост с наименьшим ID из «известных»;

– в качестве собственного корневого порта выбирается порт, получающий BPDU с наименьшей стоимостью пути до корня;

– «назначенным» портом для сегмента считается тот порт, через который в сегмент передаются самые «лучшие» BPDU-сообщения (мост определяет «назначенные» порты для каждого сегмента, к которому подключен, и может выбрать свой собственный порт, если исходящие BPDU «лучше» всех поступающих через этот порт).

После нескольких итераций обмена BPDU-сообщениями представления всех мостов о логической топологии сети совпадут. В этот момент алгоритм «сойдется» к результирующему остовному дереву, и портам будут назначены режимы блокирования или пересылки в зависимости от их принадлежности дереву.

Так как формальное определение момента сходимости затруднено, переключение портов в режим пересылки откладывается по таймеру, чтобы исключить возникновение в топологии временных циклов в переходный период. После включения моста порты находятся в состоянии прослушивания, передавая только кадры с BPDU-сообщениями. Таймер **forward delay** (значение по умолчанию — 15 с) устанавливается при последнем обновлении информации о логической топологии. Первое срабатывание таймера приводит к переключению порта из режима прослушивания в режим обучения; в этом режиме порт запоминает MAC-адреса устройств в сегменте, пополняя свою базу фильтрации. Второе срабатывание (через 30 с) переключает порт в режим пересылки. Порт также может в любой момент быть переведён в режим блокирования в соответствии с текущим остовным деревом. Состояния портов и переходы между ними показаны на рис. 4.5.

Чтобы обеспечить своевременную реакцию мостов на изменение топологии, вводится специальный таймер устаревания топологии (**max age**, по умолчанию 20 секунд). Этот таймер устанавливается заново при получении очередного BPDU-сообщения. Срабатывание таймера означает, что какой-то мост или сегмент на пути к корню дерева вышел из строя и не может пересылать исходящие от корня BPDU. В такой ситуации мост, обнаруживший неполадку, объявляет себя корневым мостом и начинает рассылать BPDU. Инициированный таким образом заново алгоритм сойдётся к новому остовному дереву, учитывающему изменения в топологии.

Подключение новых мостов к сети не требует введения дополнительных таймеров или другой специальной обработки. Как мы помним, только что включенные мосты считают себя корневыми, и, следовательно, рассылают BPDU-сообщения, оповещающие об этом. «Старожилы» сети

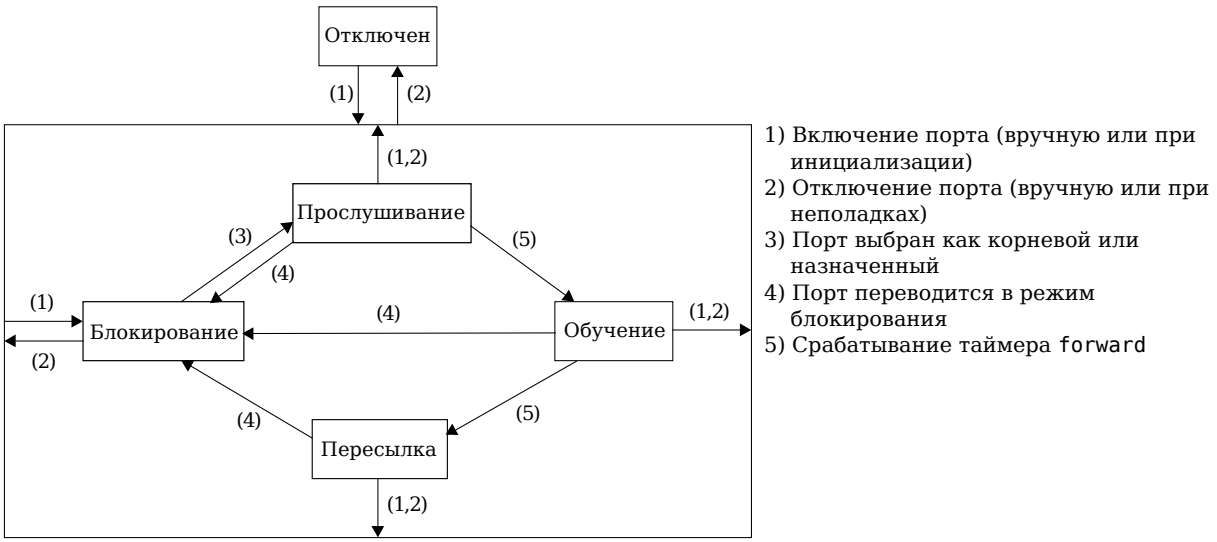


Рисунок 4.5 — Состояния порта согласно протоколу STP

обработают эти сообщения в обычном порядке и либо сообщат новому мосту о существующей логической топологии, либо выберут его новым «назначенным» или корневым мостом в зависимости от параметров новичка.

Обнаружив изменение топологии сети, мост начинает рассылку специальных BPDU-сообщений — уведомлений об изменении топологии (TCN). «Назначенный» мост отвечает на TCN-уведомления TCA-подтверждением (получив которое, мост-отправитель прекращает рассылку) и пересылает TCN-уведомление в направлении корневого моста. Корневой мост, получив уведомление, устанавливает в рассылаемых BPDU-сообщениях флаг изменения топологии. Получив BPDU с этим флагом, другие мосты ускоряют устаревание записей в своих базах фильтрации.

В ядре ОС Linux за поддержку сетевых мостов отвечает модуль ядра **bridge**, в котором, в частности, и реализован протокол STP. Для управления мостами используется утилита **brctl**. С этой утилитой вы уже познакомились, выполняя предыдущее задание. Кратко напомним, как с ней работать:

– **brctl addbr br0** — создание моста с именем br0,
– **brctl addif br0 eth0** — добавление порта,
– **ifconfig br0 up** — запуск моста.

Добавляемые порты получают последовательные номера, начиная с единицы.

Следующие команды используются для управления работой протокола STP:

– **brctl stp br0 on** — включение протокола STP,
– **brctl showstp br0** — вывод состояния STP,
– **brctl setbridgeprio br0 <число>** — задание приоритета моста,
– **brctl setportprio br0 eth0 <число>** — задание приоритета порта,

– **brctl setpathcost br0 eth0 <число>** — задание стоимости порта.

Утилита **brctl** не может управлять мостами, запущенными на других машинах.

Команда **brctl showstp <имя-моста>** выводит информацию о состоянии моста и его портов. Ниже приведён пример вывода этой команды с комментариями:

```
bridge2:~# brctl show br0
br0
 bridge id        8000.000000000601  <-- ID моста (приоритет.MAC)
 designated root  8000.000000000101  <-- ID корневого моста (по мнению данного)
 root port        3                  <-- номер корневого порта
 path cost        100                <-- стоимость пути до корня
 .....                               <-- значения различных таймеров (опущены)
eth0 (1) <-- имя интерфейса и номер соответствующего порта
 port id          8001 <-- ID порта (байт 1 -- приоритет, байт 2 -- номер)
 designated root   8000.000000000101
 designated bridge 8000.000000000804 <-- "назначенный" мост в сегменте,
                                         к которому подключен порт
 designated port   8003
 designated cost   500             <-- стоимость пути через "назначенный" мост
 state             blocking        <-- порт в состоянии блокирования кадров
 path cost         100             <-- стоимость порта
 .....      <-- различные таймеры и флаги (опущены)
eth1 (2)
 .....      <-- аналогичная, но различающаяся информация по второму порту
 port id           8002
 designated bridge 8000.000000000601 <-- мост считает сам себя "назначенным"
                                         в сегменте, к которому подключен порт
 designated port   8002              <-- совпадает с port id для "назначенного"
 state             forwarding      <-- порт пересылает кадры
 .....
.... <-- информация по остальным портам данного моста
```

Заметим, что значения приоритетов выводятся в шестнадцатеричной записи, а стоимостей — в десятичной. При задании этих значений в командной строке используется десятичная запись.

Для наблюдения за обменом BPDU-сообщениями между мостами можно использовать команду **tcpdump**, запуская её на интересующей машине. Указывать какие-либо дополнительные параметры не требуется.

Виртуальные концентраторы **uml_switch** в сетях на основе Netkit слишком «тупы», чтобы функционировать в качестве мостов, поэтому роль мостов возьмут на себя виртуальные машины.

Для выполнения данного задания подготовлена виртуальная сеть Netkit **net-stp**, состоящая из пяти виртуальных машин **b1**..**b5**. На каждой машине уже создан мост **br0** с параметрами по умолчанию (т.е. приоритеты всех мостов и портов и стоимости портов одинаковы). Топология сети показана на рисунке 4.6. Все машины в данной сети являются мостами; ра-

бочих станций, маршрутизаторов и т.п. тут нет. Таким образом, рисунок 4.6 полностью соответствует графу топологии, с которым имеет дело протокол STP.

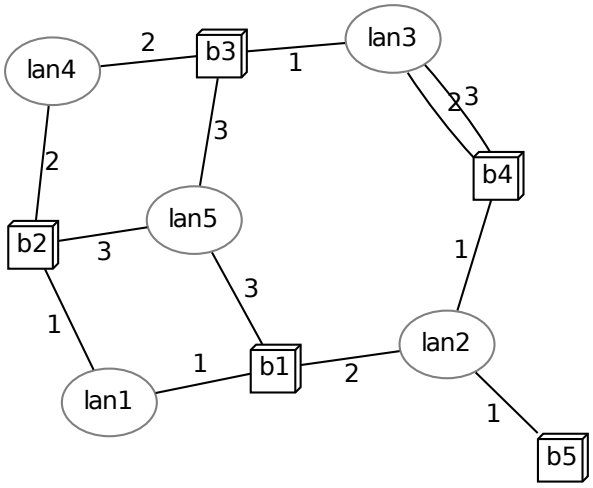


Рисунок 4.6 — Сеть **net-stp**, в которой выполняется задание. Метки рёбер — номера портов мостов.

Скачаем архив с шаблоном сети на рисунке 4.6.

```
mkdir -p ~/tcp-ip; cd ~/tcp-ip
wget -r http://ftp.iu7.bmstu.ru/nets/net-stp.tar.gz
rm -rf net-stp
tar -xvf net-stp.tar.gz
cd net-stp
```

В распакованной директории каталог **network** содержит заранее подготовленные файлы виртуальной сети. Эту виртуальную сеть следует запускать с помощью команды **ltabstart**. Подробнее такая операция описана в приложении Б.

В комплекте с сетью поставляется программа **stpvis**, позволяющая зафиксировать и представить графически текущее состояние остовного дерева. **stpvis** также используется для проверки индивидуальных заданий. Её применение будет описано по ходу выполнения.

Прежде всего просмотрите файл **shared.startup**. Помните, что этот сценарий выполняется на каждой машине при запуске. Обратите внимание на то, как создаются и настраиваются мосты.

Запустите сеть командой **lstart** или **ltabstart**, как описано в приложении, и дождитесь загрузки всех виртуальных машин.

На каждой машине выполните команду **brctl showstp br0**. Изучите её вывод, обращая внимание на то, какие порты являются корневыми, а какие — назначенными. Убедитесь, что все мосты имеют одинаковый приоритет.

Определите, какая из виртуальных машин является корневым мостом. Объясните, почему была выбрана именно эта машина (приоритеты-то одинаковые).

Попробуйте изобразить логическую топологию графически (на бумаге), отметив порты, включённые в остовное дерево. Отметьте также, какой мост является «назначенным» для каждого сегмента сети.

Пришло время обратиться к программе **stpvis**. На основной машине перейдите в директорию **stpvis** (она является соседней с **network**) и выполните команду **./stpvis 1.svg**. Программа **stpvis** нарисует текущую конфигурацию остовного дерева[1] и сохранит её в указанном файле. Изучите созданный рисунок, сравните с тем, который составили самостоятельно. Если вы затрудняетесь определить, какие порты являются «назначенными», а какие — корневыми, выполните **./stpvis --roles 1.svg**. На получившемся рисунке роли портов будут отмечены символами DP и RP. Сохраните рисунок для включения в отчёт.

Используя команду **brctl setbridgeprio br0 <число>**, измените приоритет одного из мостов на более высокий (например, 0). Убедитесь в изменении корневого моста, запустив **stpvis** еще раз.

Изменяя приоритеты мостов, портов и стоимости портов, добейтесь того, чтобы результирующее остовное дерево соответствовало указанному в индивидуальном задании (см. ниже). Периодически вызывайте **stpvis**, чтобы наблюдать за реальными изменениями дерева. Указывайте другое имя файла, чтобы не перезаписать **1.svg**.

Обратите внимание, что в командах надо указывать не индексы портов (**1**, **2**, **3**), а имена интерфейсов (**eth0**, **eth1**, **eth2**).

Если вы зашли в тупик при настройке и желаете начать «с чистого листа» — перейдите в каталог **network** и введите команду **lcrash**. Виртуальные машины будут завершены, а их образы файловых систем (**<машина>.disk**) — удалены. При следующем запуске через **lstart** сеть вернётся к первоначальным настройкам.

Проверяйте себя командой **./stpvis --check <номер_вашего_варианта> 2.svg** (например, **./stpvis --check 4 2.svg**). Программа сообщит, правильно ли вы настроили сеть. Рисунок с индивидуальным заданием поместите в отчёт.

На текущем корневом мосте отсоедините один из портов (**brctl delif br0 <имя>**). Убедитесь, что через некоторое время сформировано новое остовное дерево. Получите его изображение с помощью программы **stpvis** и сохраните в файле **3.svg** (выполнять проверку варианта в этот раз не нужно).

[1] На получаемых рисунках не отражены промежуточные состояния портов; непрерывная линия означает, что порт либо уже находится в состоянии пересылки, либо неизбежно перейдет в него по срабатыванию таймера STP **forward time**, если топология не успеет измениться.

Подключите отсоединенный интерфейс обратно (**brctl addif**) и убедитесь, что остовное дерево опять изменилось.

Попытайтесь «насильно» добавить в дерево определенный путь, выставив на всей его протяжённости нулевые стоимости портов. Понаблюдайте за тем, как сеть пытается выработать решение, вызывая **stpvis** с интервалом в несколько секунд. Объясните, чем вызвано наблюдаемое поведение.

Попробуйте перевести один из портов корневого моста в блокирующий режим (корневой мост поменяться не должен). Убедитесь, что это возможно только с мостом **b4**. Поразмышляйте над результатами.

Закончив работу, составьте отчёт, включив в него полученные в процессе выполнения задания рисунки **1.svg**, **2.svg** и **3.svg**. Не забудьте подписать отчёт своим именем и номером учебной группы.

4.7 Выполнение самостоятельной работы

Требуется найти такие значения приоритетов и стоимостей, чтобы остовное дерево включало все порты, кроме четырёх указанных в задании, и корневой мост совпадал с указанным. Номер используемого варианта определяется как (№ студента в списке группы) mod (число вариантов). Для самопроверки используйте **./stpvis --check <номер>**, как описано выше.

№ варианта	Корневой мост	Блокируемые порты
0	b1	b2[1], b2[3], b4[1], b4[2]
1	b4	b1[2], b2[1], b2[2], b4[3]
2	b5	b1[2], b1[3], b2[3], b4[2]
3	b1	b2[3], b3[1], b3[2], b4[3]
4	b4	b1[2], b2[1], b2[3], b4[2]
5	b5	b2[3], b3[2], b3[3], b4[3]
6	b2	b1[1], b3[2], b3[3], b4[2]
7	b4	b2[2], b2[3], b3[1], b4[3]
8	b5	b1[3], b2[1], b2[3], b4[2]

При защите задания студент должен предоставить составленный отчёт, уметь ответить на контрольные вопросы (см. в конце главы), а так же уметь выполнить следующие манипуляции с запущенной виртуальной сетью по просьбе преподавателя:

– изменить настройки так, чтобы заданный порт или сегмент сети вошел в остовное дерево (или, наоборот, был исключен из него);

– запустив **tcpdump -vvv** на одной из машин, прокомментировать его вывод при изменении топологии сети.

4.8 Контрольные вопросы

Для самоконтроля полученных знаний рекомендуется ответить на следующие вопросы.

1) Какие задачи решаются на канальном уровне?
2) В чём заключаются отличия между коммутатором и мостом?
3) Являются ли MAC-адреса уникальными глобально?
4) Что хранится в базе фильтрации на коммутаторах или мостах? Как эта информация обновляется?

4.9 Выполнение самостоятельной работы

4.10 Варианты заданий для самостоятельной работы

5 Транспортный протокол UDP

Прикладные программы, реализующие протоколы прикладного уровня, не используют протокол IP напрямую для передачи данных по ряду причин. В частности, IP-адрес идентифицирует сетевой интерфейс, а не процесс, желающий передавать или принимать данные. Передача данных между процессами является задачей протоколов транспортного уровня, работающих поверх протокола IP.

Транспортный уровень в стеке TCP/IP работают поверх протокола IP и представлен двумя основными протоколами:

– протокол **UDP** (RFC 768 [18]) работает без установки соединения, и не гарантирует доставку сообщений;
– протокол **TCP** (базовое описание — RFC 793 [4]) работает с установкой соединения и пытается обеспечить надежную доставку.

В этой главе мы познакомимся с основами транспортного уровня, протоколом UDP и его использованием, основами сетевого программирования. Примерами использования протокола UDP являются службы DNS (глава 11), DHCP (глава 10) и RIP (глава 6), а также программа **traceroute** (глава 3).

5.1 Сетевые порты и сокеты

Как протокол TCP, так и протокол UDP используют *порты* для идентификации процессов, отправляющих и получающих данные. Сетевой порт — это число в диапазоне 0..65535 (16 бит без знака). Эти порты являются абстрактным понятием транспортного уровня стека TCP/IP и их не следует путать, например, с портами USB или портами сетевых коммутаторов. Номер порта сетевой службы обычно записывают после IP-адреса или доменного имени через двоеточие, например: 192.168.10.3:21 или **mail.google.com:110**. Пару из IP-адреса и порта можно считать адресом транспортного уровня.

Прикладные программы используют механизм *сокетов* для работы с сетевыми протоколами. Сокет рассматривается как абстракция стороны сетевого обмена транспортного протокола. Кроме того, сокет связаны с программным интерфейсом для использования протокола транспортного уровня прикладными программами, реализующими некоторый протокол прикладного уровня.

Каждый сокет связан с адресом и портом, используемыми данным процессом, а в случае протокола TCP — ещё и с адресом и портом другой стороны соединения. Реализация сетевого протокола в ядре ОС, получив из сети сообщение, определяет по указанному в нём номеру порта назначения, какому сокету (и, как следствие, процессу) необходимо доставить данные.

Процесс, ожидающий входящих сообщений (*сервер*), обычно использует какой-либо фиксированный номер порта. Стандартные номера портов различных протоколов прикладного уровня можно посмотреть в файле **/etc/services**. Большая их часть относится к диапазону 0..1023, известному как *хорошо известные порты* (англ. *well-known ports*). Например, серверы протокола HTTP обычно используют порт 80. Конечно, ничто не мешает запустить на одном компьютере HTTP-серверы на портах 81 и 82; главное, чтобы программы-клиенты каким-то образом узнали, с каким нестандартным портом им следует устанавливать соединение.

Процесс, инициирующий общение с сервером, соответственно, называется *клиентом*. Отправляющий сообщения серверу клиентский процесс обычно динамически получает свой номер порта. Такие номера обычно выделяются из числа свободных в верхнем диапазоне (номера портов 30000 и выше). В редких случаях клиент может использовать и фиксированный порт, как мы увидим в главе 10 в случае клиента DHCP.

5.2 Формат сообщения протокола UDP

Протокол UDP, упрощённо, добавляет к протоколу IP только понятие «порта» для идентификации процессов, а также контрольную сумму передаваемых данных. Это хорошо видно по заголовку сообщения UDP — датаграммы — на рисунке 5.1 (размеры полей указаны в битах). Напомним, что в протоколе IP контрольная сумма вычислялась только для заголовка пакета.

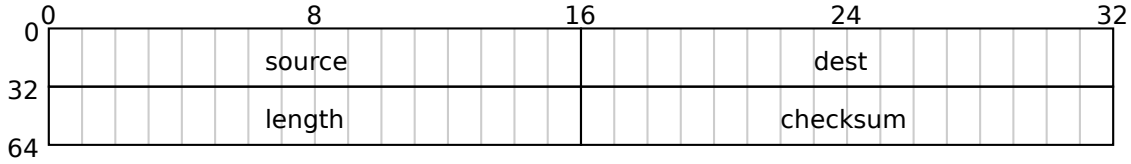


Рисунок 5.1 — Заголовок UDP-датаграммы

Поскольку каждое UDP-сообщение помещается ровно в один IP-пакет, то прикладные программы могут использовать протокол UDP для передачи небольших блоков данных. Разумную верхнюю граница размера полезных данных протокола UDP разумно выбирать так, чтобы IP-пакет с ними помещался в наихудший MTU, равный 576 байтам, во избежании возможной фрагментации. При отсутствии необязательных опций в заголовке IPv4 это составляет 548 байтов, а в качестве примерного значения можно взять полкилобайта.

Отметим, что всё сказанное в главе 3 про использование механизма PMTU для случая сообщений протокола ICMP верно и для UDP-датаграмм — пакет с датаграммой просто разбивается на фрагменты заранее исходя из первого понижения величины MTU на пути до получателя.

5.3 Применение протокола UDP

Протокол UDP был разработан примерно на шесть лет позднее протокола TCP: первым описанием TCP была ныне устаревшая спецификация RFC675, в то время как первой и последней спецификацией UDP является стандарт RFC768. Как уже говорилось в главе 1 TCP используется большинством прикладных протоколов, такими как HTTP, SMTP, POP3, SSH. Почему же после разработки сложного и достаточно надежного протокола TCP пришлось разработать и совершенно тривиальный протокол UDP, спецификация которого сводится по сути к формату его заголовка? Причина проста: недостатки протокола TCP являются, как обычно, следствием его достоинств. Хотя гарантия доставки и другие возможности протокола TCP выглядит привлекательно, существует ряд задач, для которых функциональность TCP избыточна или даже является помехой.

Важнейшими особенностями протокола TCP являются установка соединения и повторная доставка неподтверждённых сообщений. С ними обоими связаны накладные расходы: установка соединения требует передачи трёх IP-пакетов, а повторная доставка — подтверждений, часто на каждый отосланный сегмент. Кроме того, при потере сообщение передается повторно, что может быть не нужно в случае передачи данных, теряющих актуальность с течением времени.

В случае, если передаваемая информация помещается в единственный IP-пакет, протокол TCP имеет весьма низкий КПД. При его использовании будут переданы три пакета для установки соединения, один пакет данных, один пакет подтверждения, один пакет с ответом, три пакета для корректного закрытия соединения (подробнее работа протокола TCP будет рассмотрена в главе **??**). Когда процессу требуется отправить короткий запрос и максимально быстро получить столь же короткий ответ, подобные накладные расходы как минимум неприятны. В случае протокола UDP достаточно будет всего двух пакетов, и по указанной причине служба DNS используют именно протокол UDP. Задача гарантированной доставки в подобных случаях решается самим приложением или не решается вовсе (DNS-клиент может при отсутствии ответа опросить и другой сервер).

В некоторых случаях немедленное получение ответа вообще не предполагается: одному процессу просто нужно поделиться некоторой информацией с другим без ответа. При этом службам передачи потокового видео или аудио (например, IPTV) безразлична и потеря отдельных частей потока, но зато для них важна доставка потока без существенных задержек. Задержки, возникающие при повторной передаче потерянных сообщений, и подтверждения на каждое сообщение в этом случае только мешают, поэтому в таких службах используется протокол UDP. По аналогичной причине UDP применяется и в сетевых играх реального времени.

Ещё одной особенностью UDP является поддержка широковещательной и многоадресной рассылки, при которой число получателей данных точно неизвестно. Протокол TCP в принципе не предполагает наличия более чем одного получателя отправляемого потока данных, как минимум по причине установки соединения и подтверждений с повторами. Поэтому, например, для протокола DHCP использование UDP является единственным (из транспортных протоколов) возможным выбором в силу необходимости использования широковещания.

Многоадресная рассылка позволяет существенно снизить нагрузку на сеть, отправляя одно сообщение вместо отдельных сообщений для каждого получателя. Кроме того, отправителю даже не обязательно знать адреса каждого из получателей. Применение многоадресной рассылки — ещё одна причина популярности UDP в службах потокового видео и аудио и сетевых играх.

5.4 Использование протокола UDP службой доменных имён

Наша сеть включает кеширующий DNS-сервер на машине)s1 и машину **ws1**. Обе машины подключены к реальной сети (рисунок 5.2).

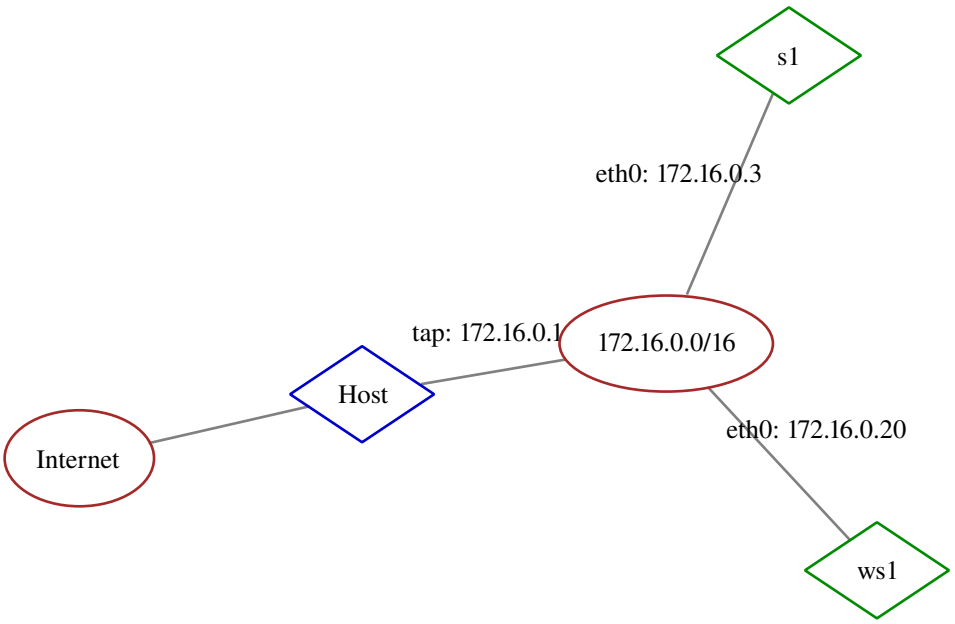


Рисунок 5.2 — Локальная сеть из двух машин, подключенная к интернет

> **Задание № 29: Работа кеширующего DNS-сервера**
>
> Скачаем шаблон сети и запустим её.

```
mkdir -p ~/tcp-ip; cd ~/tcp-ip
wget -r http://ftp.iu7.bmstu.ru/nets/lab-udp-dns.tar.gz
rm -rf lab-udp-dns
tar -xvf lab-udp-dns.tar.gz
cd lab-udp-dns
ltabstart -d net
```

На машине **ws1** запустим следующей командой перехват сообщений протокола UDP, в которых адрес 172.16.0.3 будет адрес получателя или отправителя (таким образом мы перехватим весь обмен по протоколу DNS, в котором участвует **s1**).

```
tcpdump -s 500 -tn ip host 172.16.0.3 and udp
```

На машине **ws1** пошлём эхо-запрос на доменное имя **bmstu.ru**.

```
ping -n bmstu.ru -c 1
```

Перед тем, как послать эхо-запрос, программе **ping** придётся получить через DNS IP-адрес для данного доменного имени. Обмен по протоколу DNS можно видеть на машине **s1**.

5.5 Сетевое программирование. Сокеты и протокол UDP

При выполнении данного задания мы изучим простейшие примеры программ, работающих с сокетами. Использованию сокетов протокола TCP посвящена глава 7.

Рассмотрим следующую ситуацию: пусть в нашей локальной сети имеется несколько однотипных сетевых серверов одного и того же прикладного протокола, который использует протокол UDP. Число серверов и их адреса неизвестны, кроме того факта, что они принадлежат образующим локальную сеть IP-сетям. Порт, на котором серверы ожидают поступления данных, известен, и один и тот же для все серверов. Приложению-клиенту требуется определить число и серверов в сети и получить некоторую информацию о каждом из них. Подобная задача может возникнуть, например, при программировании компьютерных игр, позволяющих пользователях в пределах локальной сети играть друг с другом.

Решением «в лоб» могла бы стать попытка соединиться с портом службы на каждом возможном компьютере в сети. Такой перебор адресов может занять значительное время, особенно если сеть большая: так, для сети типа 10.0.0.0/8 придется перебрать шестнадцать миллионов адресов.

Для эффективного решения поставленной задачи можно воспользоваться широковещательной рассылкой IP-пакетов, поддерживаемой протоколом UDP. Клиент отправит один широковещательный запрос и в течение короткого промежутка (известного как *таймаут*) времени будет ждать ответы от серверов. В пределах локальной сети такой интервал ожидания

может быть выбран фиксированным, но в главе 8 мы увидим, что выбор разумного времени ожидания ответа в общем случае является отдельной проблемой. Ответы, пришедшие после истечения этого интервала, мы не будем рассматривать — такое решение и является весьма грубым.

Код программы-сервера на языке Python с комментариями приведён ниже. Если вы не знакомы с языком Python, считайте, что это псевдокод.

```
#!/usr/bin/env python
# encoding: utf-8
import socket
import time
# отметим время запуска сервера
started = time.time()
# создаём UDP-сокет:
sk = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
# назначаем "свой" IP-адрес (в данном случае это "любой адрес") и порт
sk.bind(("0.0.0.0", 9966))
print "Listening --- press Ctrl-C to stop"
while True:
    # ждём получения датаграммы, выделив для неё буфер в 512 байт
    data, sender = sk.recvfrom(512)
    if data == 'Hello?':
        # формируем ответное сообщение
        response = 'My name is %s, and I am %5.1f seconds old' % (
            socket.gethostname(), time.time() - started)
        # отправляем тому, кто прислал запрос
        sk.sendto(response, sender)
```

Обратим внимание на следующие моменты.

Отметим, что при создании сокета имя протокола UDP не фигурирует — мы просим создать «сокет датаграммного типа (SOCK_DGRAM) для стека TCP/IP (AF_INET)», что фактически и означает использование протокола UDP[1].

Сервер должен связать свой сокет с адресом и портом, чтобы получать направляемые на них сообщения. Вместо адреса 0.0.0.0, который здесь означает «любой адрес данной машины», можно было указать адрес одного из сетевых интерфейсов или широковещательный адрес сети — в нашем случае это будет адрес 10.10.0.255. В таком случае сокет не смог бы получить пакеты, в которых указан отличающийся адрес назначения, даже притом, что номер порта назначения совпадает.

Функции **socket**, **bind**, **recvfrom**, **sendto** являются частью так называемого BSD socket API, или Berkeley sockets [19] — фактически стандартного программного интерфейса для работы с сокетами в большинстве операционных систем. Эти функции имеют одинаковые названия и семантику во всех реализациях, хотя структуры данных, которыми они оперируют, зависят от платформы и языка программирования. Впервые сокеты появи-

[1]Напомним, что сообщения протокола UDP называются датаграммами

лись в ОС BSD Unix, по этой причине данный программный интерфейс и получил такое имя.

Общим элементом всех сетевых серверов является бесконечный цикл, в котором они ждут очередного запроса или соединения. Приведённый выше сервер реализован по наиболее простой схеме и обслуживает поступающие запросы последовательно. На практике используются серверы с более сложной структурой, позволяющей обслужить множество запросов одновременно, используя параллельные потоки или мультиплексирование ввода-вывода. В главе 7 мы рассмотрим создание таких серверов.

После окончания работы сокеты следует закрывать функцией **close**. В языке Python это делается автоматически при сборке мусора.

Исходный код программы-клиента приведём по частям. Инициализация сокета выглядит так:

```
#!/usr/bin/env python
# encoding: utf-8
import socket
# создаём UDP-сокет:
sk = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
# разрешаем на нём широковещание
sk.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 1)
# отправляем запрос по широковещательному адресу для 10.10.0.0/24 на порт 9966
sk.sendto("Hello?", ("10.10.0.255", 9966))
# задаём таймаут для блокирующих операций (полсекунды)
sk.settimeout(0.5)
# начинаем принимать ответы
while True:
    try:
        data, addr = sk.recvfrom(512) # размер буфера для приёма
    except socket.timeout:
        # не дождались очередного ответа --- выходим
        break
    else:
        print "Response from", addr, ":"
        print "", data
print "No more servers."
```

В клиенте отсутствует явная привязка сокета к номеру порта. Вместо этого выделение номера порта производится динамически в момент отправки первого сообщения (**sendto**).

Функция **setsockopt**, входящая в BSD socket API, служит для задания различных параметров сокета и сетевых протоколов различных уровней. В примере она используется для разрешения широковещательной рассылки, которая по умолчанию запрещена.

Функция **settimeout** запрещает функции получения сообщения **recvfrom** блокировать процесс более чем на указанное время. Функция **settimeout** не является частью BSD socket API и специфична для языка Python; при программировании на языке Си таймаут задается по-другому.

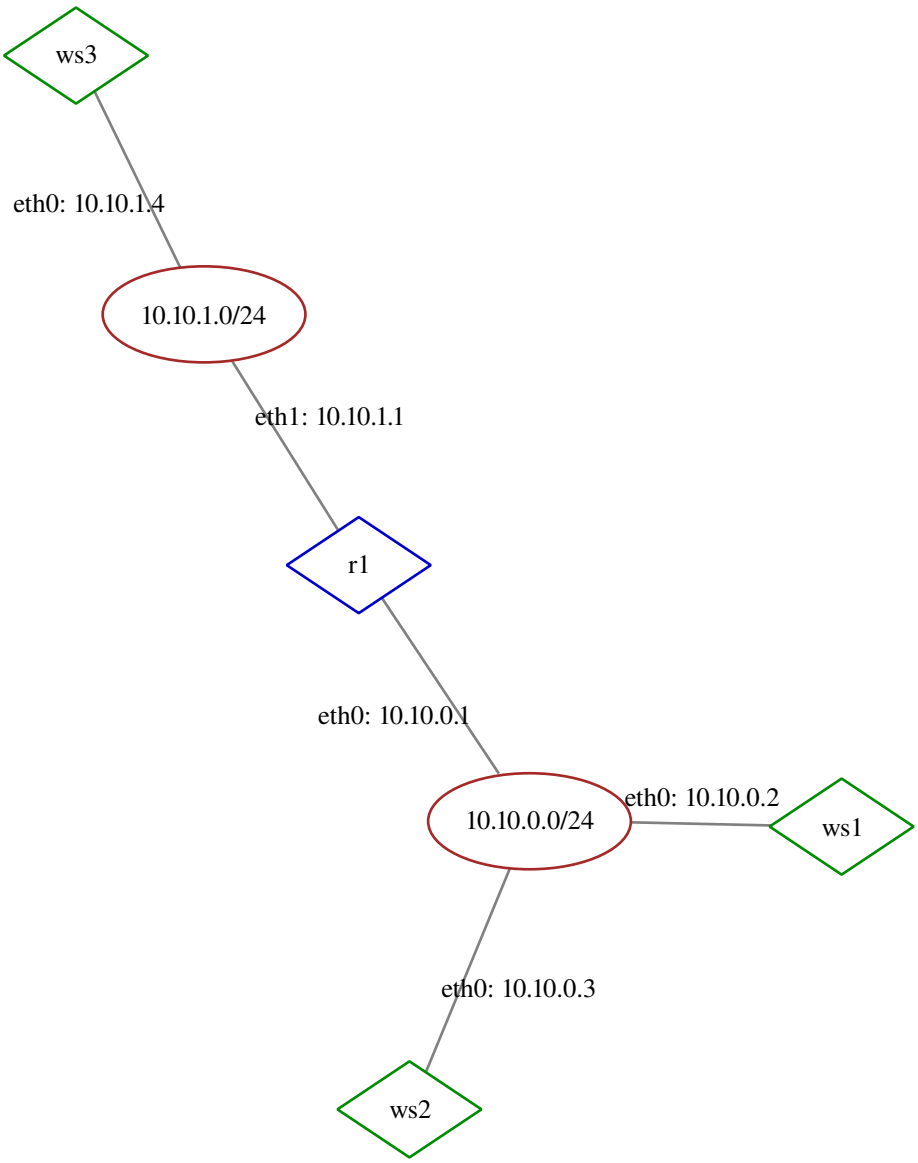


Рисунок 5.3 — Сеть для экспериментов с протоколом UDP

Задание № 30: Подготовка сети для экспериментов с протоколом UDP

Загрузите файлы виртуальной сети, в которой будет работать наше сетевое приложение, и разверните её на своём компьютере.

```
mkdir -p ~/tcp-ip; cd ~/tcp-ip
wget -r http://ftp.iu7.bmstu.ru/nets/lab-udp.tar.gz
rm -rf lab-udp
tar -xvf lab-udp.tar.gz
cd lab-udp
ltabstart -d net
```

Сеть состоит из четырёх машин (ws1, ws2, ws3 и маршрутизатора r1) и двух сетей: 10.10.0.0/24 и 10.10.1.0/24 (рисунке 5.3). Программа-клиент будет запускаться на машине **ws1**.

Запустите на машинах ws1, ws2, ws3, r1 программы-серверы следующей командой.

```
./udpserver.py &
```

Нам потребуется сделать некоторые действия уже при запущенных серверах, поэтому в конец команды добавлен амперсанд для её выполнения в фоне. На машине **ws2** запустите команду **tcpdump** для наблюдения за сообщениями.

```
tcpdump
```

Запустите на машине **ws1** приложение-клиент:

```
./udpclient.py
```

Изучите напечатанное в консоли, сравните с набором запущенных серверов. Загляните в вывод **tcpdump**.

Для остановки запущенных в фоне (с амперсандом) серверов дайте команду **fg**, которая вернёт сервер из фона на «передний план», после чего нажмите Ctrl-C.

```
fg
^C
```

Как можно заметить, сервер на машине **ws3** не появляется в списке, выводимом клиентом. Это объяснимо: **ws3** находится в подсети 10.10.1.0/24, а наш клиент использует широковещательный адрес подсети 10.10.0.0/24. Широковещание в IP ограничено одной непосредственно подключённой IP-сетью: даже если клиент отправит ещё одно сообщение на широковещательный адрес 10.10.1.255, оно не будет передано в сеть 10.10.0.0/24.

Задание № 31: Широковещательный UDP-ретранслятор Модифицируйте код сервера **udpserver.py** на маршрутизаторе **r1** так, чтобы тот транслировал широковещательные запросы для подсети 10.10.0.0/24 в подсеть 10.10.1.0/24, а ответы на них — обратно отправителю запроса. Не забудьте разрешить широковещание на сокете. Постарайтесь избежать зацикливания, при котором сервер отвечает на широковещательный запрос, который сам же только что отправил.

Для редактирования кода программы можно использовать консольные редакторы **emacs**, **vim**, **nano** на самой виртуальной машине. Программу-клиент и серверы на других машинах модифицировать не нужно.

После модификаций программа-клиент должна выводить список всех запущенных серверов, включая сервер на **ws3**.

5.6 Контрольные вопросы

Для самоконтроля полученных знаний рекомендуется ответить на следующие вопросы.

1) Какова основная задача протоколов транспортного уровня?
2) Перечислите основные различия между протоколами TCP и UDP.
3) Опишите различия в присвоении номеров портов серверам и клиентам.
4) В каких задачах предпочтительнее использовать протокол UDP, а не TCP?
5) Перечислите функции API, используемые при работе с UDP-сокетами.

6 Динамическая IP-маршрутизация с использованием протокола RIP

В главе 3 мы уже познакомились с ручной настройкой маршрутных таблиц в нетривиальном случае. Этот процесс явно можно автоматизировать, реализовав некоторый распределенный, т. е. работающий на нескольких компьютерах, алгоритм нахождения маршрута до IP-сети.

Существует два случая задачи автоматического построения маршрутных таблиц: в пределах одной организации, например, провайдера доступа в интернет или крупного предприятия, и в масштабах всей мировой сети. Первый случай сводится по сути к поиску минимальных путей в графе топологии сетевого уровня. Во втором случае только поиска минимальных путей недостаточно: нужно учитывать существующие договорённости между провайдерами об обмене сетевым трафиком, а также пытаться както сжимать маршрутные таблицы, иначе в них окажутся все существующие IP-сети.

В этой главе мы познакомимся с протоколом обмена маршрутной информацией RIP, который может использоваться для решения задачи построения маршрутных таблиц в первом из этих случаев.

Мы воспользуемся ПО Quagga, которое реализует различные протоколы обмена маршрутной информации, из которых нам понадобится только RIP, и настроим её файл конфигурации **/etc/quagga/ripd.conf**.

В конце этой главы (раздел 6.14) приведены возможные индивидуальные варианты заданий. В разделе 6.13 даны дополнительные указания по их выполнению.

6.1 Избыточная топология сетевого уровня

При статической IP-маршрутизации все записи в маршрутной таблице прописаны заранее администратором сети, как мы уже делали в главе 3. Наиболее очевидный недостаток такого подхода: если между двумя узлами сети передачи данных существует более одного пути, то при статической маршрутизации невозможно выбрать из них тот, который выглядит работоспособным в некоторый момент времени. Кроме того, трудоёмкость составления статических таблиц маршрутизации в системе растет примерно как квадрат числа IP-сетей, и выполнять это работу вручную с учетом желательности минимизации длины путей становится весьма сложно.

В этой главе мы будем использовать компьютерную сеть, показанную на рисунке 6.1. Как можно увидеть, эта сеть является избыточной с точки зрения передачи данных между машинами **ws1**, **ws2** и **wsp1**: между каждой парой этих машин существуют два возможных пути с точки зрения протокола IP. Кроме того, она является достаточно большой для того,

чтобы решение задач построения полных маршрутных таблиц с учётом минимизации путей было разумно переложить на плечи компьютера.

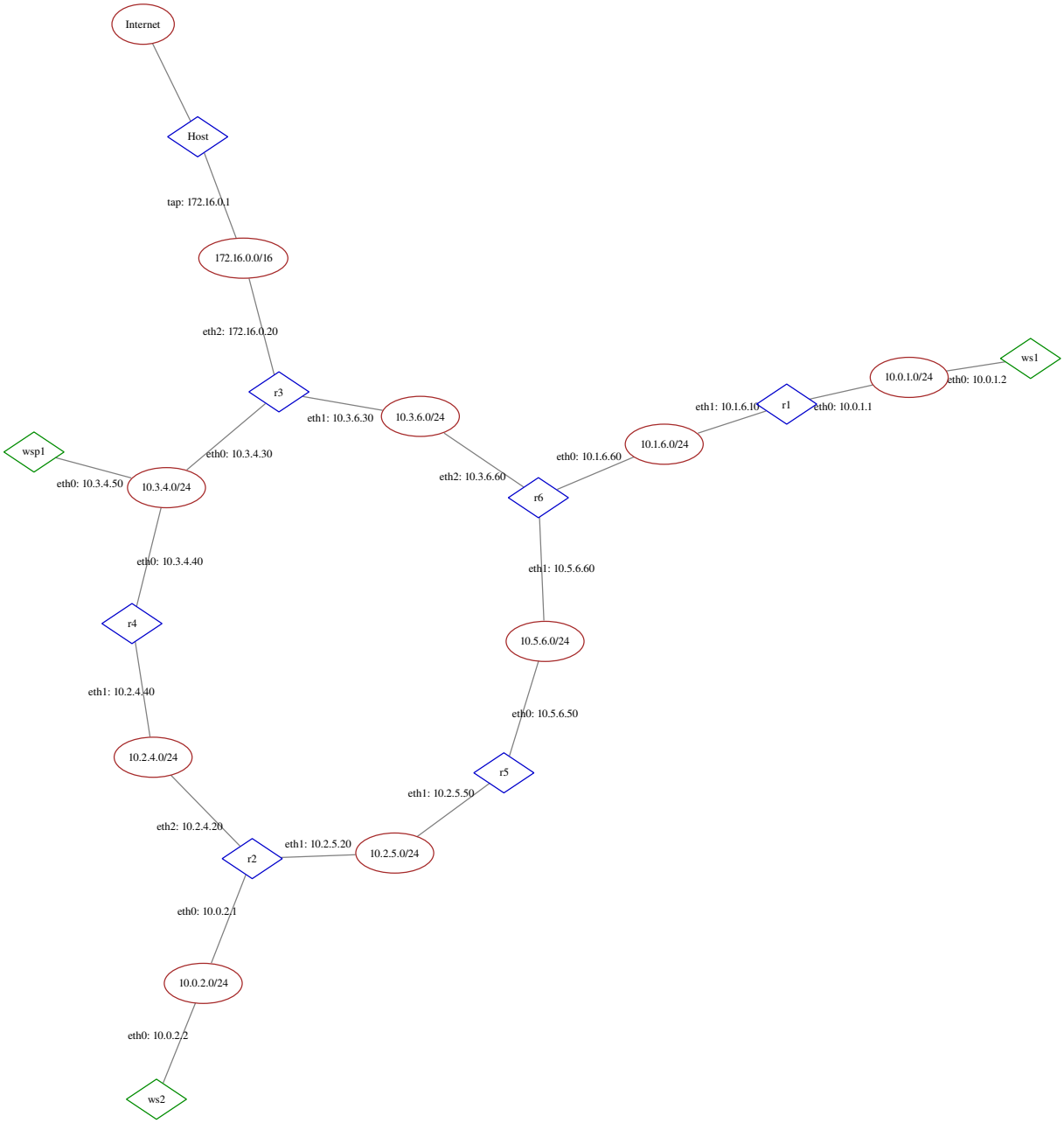


Рисунок 6.1 — Сеть с избыточной топологией сетевого уровня

Задание № 32: Запуск сети и проверка настроек IP-адресов

Загрузите и распакуйте файлы виртуальной сети, а затем запустите все виртуальные машины следующими командами.

```
mkdir -p ~/tcp-ip; cd ~/tcp-ip
wget -r http://ftp.iu7.bmstu.ru/nets/lab-rip.tar.gz
rm -rf lab-rip
tar -xvf lab-rip.tar.gz
cd lab-rip
ltabstart -d net
```

Как можно убедиться командами **ip a** и **ip r**, всем сетевым интерфейсам в нашей системе уже присвоены IP-адреса, но таблицы маршрутизации на всех машинах тривиальны, за исключением находящихся в тупиковых сетях машин **ws1** и **ws2**.Сейчас можно послать успешный эхозапрос с любой машины до любой соседней — но не далее.

У машин **ws1** и **ws2** уже указан маршрут по умолчанию: по рисунку 6.1 видно, что все пути от них до всех остальных машин всегда начинаются с одного и того же маршрутизатора. Остальные семь машины (**r1** – **r6** и **wsp1**) будут использовать *динамическую машрутизацию*.

6.2 Динамическая маршрутизация

При динамической маршрутизации содержимое маршрутных таблиц определяется некоторым специальным ПО, реализующим особый протокол и алгоритм для поиска кратчайших путей по некоторому критерию.

Динамическая маршрутизация — это подход к составлению маршрутных таблиц, при котором они заполняются автоматически на основе некоторого алгоритма в результатам информационного обмена между маршрутизаторами. Отметим, что сам процесс маршрутизации IP-пакетов одинаков и при статической, и при динамической маршрутизации — отличается лишь подход к заполнению маршрутных таблиц.

Для динамической маршрутизации требуется реализация отдельного протокола, который позволял бы маршрутизаторам обмениваться некоторой информацией о топологии системы. Отметим, что часто такие протоколы и называют протоколами динамической маршрутизации, что формально не совсем верно: самой маршрутизацией занимается реализация протокола IP. Более корректным термином является «протокол обмена маршрутной информацией».

Для определения минимизируемого при динамической маршрутизации параметра нам понадобится понятие *автономной системы*(англ. *autonomous system*) [20]. Можно выделить две группы протоколов обмена маршрутной информации:

– протоколы *внутренней маршрутизации* решают задачу определения маршрутов внутри автономной системы, её части или даже большой локальной сети и исходят из критерия минимизации расстояния до сети;

– протоколы *внешней маршрутизации* определяют маршруты между автономными системами в соответствии с договорённостями между владельцами этих систем и критерия минимизации расстояния.

Протокол RIP относится к протоколам внешней маршрутизации и может быть развернут в сети крупной организации или провайдера доступа в интернет. В настоящее время для этого чаще используют более современ-

ный протокол OSPF, но наблюдение за его работой весьма сложно, поэтому он плохо подходит для учебных целей. Для наблюдения за работой протокола RIP, как мы убедимся далее, часто достаточно перехватить любые отдельные сообщения маршрутизаторов.

6.3 Основные моменты протокола RIPv2

Протокол обмена маршрутной информацией RIPv2 (англ. *Routing Information Protocol version 2*) [21][22] работает поверх транспортного протокола UDP с использованием групповой IP-адресации (англ. *multicasting*).

Протокол RIP основан на использовании алгоритма Беллмана-Форда-Мура [**?**]. Однако его формальное изложение для случая протокола RIP чрезмерно сложно: как видно по рисунку 6.1, компьютерная сеть формально представляется двудольным графом, имеющих два вида вершин: сети и маршрутизаторы. При этом ребро от маршрутизатора к сети имеет единичный вес при таком прохождении, и нулевой — при обратном (что можно формализовать и двумя дугами). Кроме того, алгоритм предполагает, что все граф является известным, чего в случае использования протокола RIP не имеет место быть: каждый маршрутизатор рассылает своим соседям свой вектор расстояний до известных сетей, поэтому полный набор ребер в графе неизвестен маршрутизаторам, известно лишь множество вершин-сетей. По этой причине распределенный вариант алгоритма Беллмана-Форда-Мура, используемый службой RIP, лучше изложить, вообще не вводя формально граф сети [**?**].

Маршрутизатор, использующий протокол RIP, ведёт таблицу известных сетей. Каждая строка этой таблицы включает адрес и маску сети, расстояние до неё и IP-адрес следующего маршрутизатора (англ. *next-hop*, букв. «следующий прыжок») на пути до данной сети. Расстояние до сути измеряется в числе переходов между маршрутизаторами: сети, подключенные непосредственно, имеют метрику, равную единице, доступные через соседний маршрутизатор имеют метрику, равную двойке, и так далее. Адрес следующего маршрутизатора для непосредственно подключённых сетей условно равен 0.0.0.0. Кроме сети, расстояния до него и адреса следующего маршрутизатора со строкой таблицы связаны таймеры, которые будут рассмотрены далее. Отметим, что эта таблица протокола RIP не тождественна системной таблице маршрутизации, хотя и имеет близкую структуру.

Неприятной принципиальной особенностью распределенного алгоритма является то, что он не гарантирует корректной работы из-за возможных произвольных потерь сообщений RIP в сети и гонок между маршрутизаторами. Из-за этого, возможен, в частности, неограниченный рост метрики при появлении ложных циклов маршрутизации, аналогичных искусственно созданному нами в разделе 3.7 на стр. 61. В качестве грубого, но

радикального решения этой проблемы метрика сети в протоколе RIP принимает только значения $\langle 1, 2, ..., l_{inf} = 16 \rangle$, причём значение метрики, равное l_{inf}, означает, что сеть уже считается недоступной. Это значение было выбрано как компромисс между размерами системы, в которой можно развернуть протокол RIP (чем оно больше — чем больше будет максимально возможный путь в такой системе) и желанием обнаруживать бесконечно растущую метрику как можно быстрее.

6.4 Служба протокола RIP

До начала опытов настроить на всех машинах, которые должны использовать динамическую маршрутизацию, соответствующую службу — в наших системах это ПО Quagga, которое реализует основные протоколы обмена маршрутной информации, в том числе и RIP.

Нам нужно указать в файле конфигурации службы протокола RIP, через какие сетевые интерфейсы ей следует рассылать и принимать сообщения протокола RIP. Укажите на всех машинах, где должна быть включена служба RIP, через какие сетевые интерфейсы ей следует работать. Для этого в файле **/etc/quagga/ripd.conf** следует указать строки вида **network ethX** (в нашем случае просто достаточно убрать символ комментария у «нужных» строк). Следует указать те и только те интерфейсы, которые ведут к машинам, использующим протокол RIP. Эти интерфейсы легко увидеть по рисунку 6.1. Например, на маршрутизаторе **r1** в файле конфигурации должна быть строка **network eth1**, а на маршрутизаторе **r6** — строки для всех трёх его сетевых интерфейсов.

Отметим, что на маршрутизаторе **r3** нет нужды включать рассылку сообщений RIP на интерфейсе **eth2**: нашему виртуальному провайдеру доступа в интернет сообщения RIP решительно не интересны. На этом же маршрутизаторе желательно закомментировать в файле конфигурации строку **redistribute connected**. Это уберёт из рассылаемых им RIP-сообщений информацию о сети 172.16.0.0/12, соединяющий этот маршрутизатор с провайдером, поскольку в ней нет никакой необходимости.

> **Задание № 33: Настройка службы динамической маршрутизации**
>
> Отредактируйте файл конфигурации службы Quagga и перезапустить службу Quagga на всех машинах, которые должны использовать протокол RIP (то есть на всех, кроме **ws1** и **ws2**).
>
> После изменения файла конфигурации на очередной машине её службу динамической маршрутизации следует перегрузить следующей командой.
>
> ```
> service quagga restart
> ```

Если в файле конфигурации допущены синтаксические ошибки, то эта операция закончится неудачей и будет выведена вызвавшея ошибку строка. Отредактируйте файл конфигурации и после исправления ошибки опять перезапустите службу.

В файл журнала **/var/log/quagga/ripd.log** будут выводится важные сообщения о службе протокола RIP. Сейчас там вероятно есть только строка о её запуске, но в случае проблем рекомендуется просмотреть содержимое этого файла командами **cat**, **tail** и подобными [1], например так.

```
tail /var/log/quagga/ripd.log
```

Убедитесь, что в последняя строка в файле журнале сообщает об успешном запуске службы.

6.5 Алгоритм заполнение вектора расстояний

Сначала мы рассмотрим, как маршрутизатор RIP действует при получении сообщений от своих соседей. Пока условимся считать, что каждый маршрутизатор просто рассылает все известные сети и их метрики во все свои интерфейсы, и надо определить действия маршрутизатора при получении такого сообщения. Более подробно действия маршрутизатора при рассылки сообщений мы приведем несколько позже: далее мы увидим, что есть разные способы формирования сообщений протокола RIP.

Обозначим через N множество известных сетей, через V — вектор расстояния маршрутизатора, хранящий отображение сети в её метрики и следующий маршрутизатора на пути к ней, через C — множество подключенных непосредственно сетей. Изначально всех сетей, подключённых непосредственно к данному маршрутизатору, метрика равна единице: $\forall c \in C : c \in N, V(c) = \langle 1, r \rangle$. Метрики этих сетей не меняются в ходе работы протокола RIP.

При каждом получении информации о сети $\langle n, p, h \rangle$ есть следующие варианты событий в зависимости от полученной метрики, известности сети и совпадения маршрутизатора с текущим следующим в маршруте. Увеличим полученную метрику на единицу, если она ещё не равна бесконечности ($d = min(p + 1, l_{inf})$), и рассмотрим все возможные случаи.

1) Сеть n неизвестна, метрика равна «бесконечности». Обозначим это событие как $E_{inf} : n \notin N \vee d = l_{inf} \rightarrow E_{inf}$. Это событие нам безразлично.

2) Сеть n неизвестна, метрика меньше «бесконечности». Это событие обозначим как E_{new}: $n \notin N \vee d < l_{inf} \rightarrow E_{new}$.

3) Сеть n известна, но получена более короткая метрика от другого маршрутизатора. Это хорошие новости, обозначим событие как $E_{shorter}$: $\langle n, q, y \rangle \in V \vee y \neq h \vee d < q \rightarrow E_{shorter}$. Заметим, полученная метрика

должна быть именно меньше, а не меньше или равна, текущей, иначе мы будет постоянно переключать маршруты до сети в ряди случаев.

4) Сеть n известна, метрика меньше «бесконечности», информация от того же маршрутизатора. Обозначим это событие также как E_{same}: $d < l_{inf} \vee \langle n, x, h\rangle \in V \rightarrow E_{same}$. Не важно, как метрика меньше, текущая или полученная, поскольку это новое значение метрики от того маршрутизатора, который мы считаем следующим прыжком.

5) Сеть n известна, метрика равна «бесконечности», информация от другого маршрутизатора. Обозначим это событие как E_{ignore}, поскольку нам оно тоже безразлично :$n \in N \vee d = l_{inf} \vee \langle n, x, h\rangle \notin V \rightarrow E_{ignore}$.

6) Сеть n известна, информация от того же маршрутизатора, новая метрика равна «бесконечности». Это плохие новости: тот маршрутизатор, который мы полагали следующим, прислал нам плохую метрику. Обозначим это событие как E_{bad}: $d = l_{inf} \vee \langle n, x, h\rangle \in V \rightarrow E_{bad}$. Как и в прошлом случае, текущее значение метрики тут уже не важно.

В итоге мы получили следующие события, возникающие при получении информации о сети: $E_{ignore}, E_{inf}, E_{new}, E_{same}, E_{shorter}, E_{bad}$ При первых двух событиях делать маршрутизатору ничего не нужно, во всех остальных случаях необходимо обновить вектор расстояний: $E_{new} \wedge E_{same} \wedge E_{shorter} \wedge E_{bad} \rightarrow V(n) = \langle d, h\rangle$. Заметьте, при событии E_{bad} новая метрика в V будет равна l_{inf}, а при остальных она будет меньше l_{inf}.

Отметим, что если исходный алгоритм Беллмана-Форда-Мура сходился за некоторое число итераций, то для алгоритма протокола RIP сходимость доказана только для случая отсутствия или ограниченного возникновения проблем в сети. Динамическая маршрутизация, как уже было сказано, и должна решать задачу перестроения маршрутных таблиц при возникновения проблем в сети, но в случае протокола RIP можно только утверждать, что после любого сбоя маршрутные таблицы стремятся к правильным и даже сходятся к ним за конечное время, если в сети не происходит при этом нового сбоя.

После настройки и перезагрузки службы Quagga все маршрутизаторы регулярно рассылают свою таблицу известных сетей. Войдя на любом маршрутизаторе в консоль управления службой маршрутизации (команда **vtysh**), можно выполнить команду **show ip rip** для просмотра этой таблицы. В итоге в таблице протокола RIP появятся адреса всех сетей нашей системы, причём с наименьшими возможными метриками.

> **Задание № 34: Таблица протокола RIP**
>
> На маршрутизаторе **r6** зайдите в консоль управления и, повторяя команду **show ip rip**, вы увидите заполнение таблицы протокола. Если всё настроено правильно, то она в итоге будет выглядеть примерно следующим образом.

Строка, соответствующая сети 10.2.4.0/24, может содержать как адрес маршрутизатора...

6.6 Таймеры протокола RIP

Как видно, вектор расстояний отличается от таблицы маршрутизации тем, что запоминает сети с «бесконечными» метриками, которые не нужны системной таблице маршрутизации. Это нужно прежде всего для того, чтобы информировать другие маршрутизаторы о сетях с «бесконечной» метрикой, чтобы вызвать у них в свою очередь событие E_{bad}. В противном случае информация о недоступной сети не будет распространена по всей системе. Очевидно, что хранить информацию о такой сети не стоит сколь угодно долго и через некоторое время сеть n можно удалить из множества известных сетей N и вектора расстояний ($V(n) = \varnothing$).

Теперь все маршрутизаторы регулярно рассылают свою таблицу известных сетей путём групповой рассылки на IP-адрес 224.0.0.9. По стандарту они делают это раз в 30 сек., но в нашей системе этот интервал сокращён до 10 сек. для удобства наблюдения за рассылаемыми сообщениями. Войдя на любом маршрутизаторе в консоль управления службой маршрутизации (команда **vtysh**), можно регулярно повторять команду **show ip rip** (для повтора нажмите). В итоге в таблице протокола RIP появятся адреса всех сетей нашей системы, причём с наименьшими возможными метриками. На маршрутизаторе **r6** зайдите в консоль управления и, повторяя команду **show ip rip**, вы увидите заполнение таблицы протокола RIP. Если всё настроено правильно, то она в итоге будет выглядеть следующим образом.

Последняя колонка показывает текущее значение таймера устаревания. Такой таймер существует для каждой строки таблицы протокола RIP с метрикой сети, меньше «бесконечности». Его значение по стандарту равно 180 сек., но в нашей системе оно уменьшено до 60 сек. Если за данный промежуток маршрутизатор не получает сообщений с информацией о данном маршруте (то есть о пути до данной сети с тем же следующим маршрутизатором), то маршрут отмечается как недоступный и его метрика меняется на «бесконечность».

Выйдя из консоли управления (**exit**), командой **ip r** можно убедиться в том, что системная таблица маршрутизации также пополнилась: служба Quagga помещает в неё все известные ей сети, метрика которых меньше 16-ти. На маршрутизаторе **r1** в итоге будет такая таблица маршрутизации.

Поскольку в нашем случае все сети в таблице RIP имеют метрику, свидетельствующую об их доступности, то все в системной таблице маршрутизации все эти сети, как и ожидалось, присутствуют.

6.7 Сообщения протокола RIP

Теперь следует выяснить, как же выглядят сообщения, которые рассылает служба RIP и благодаря которым все маршуртизаторы (и машина **wsp1**) узнали про сети в нашей системе.

Пакет RIPv2 (рисунок 6.2) состоит из заголовка и строк маршрутной таблицы (до 25 штук). Всего есть два типа сообщений RIP, определяемых полем **command** в заголовке: запрос (1) и ответ (2).

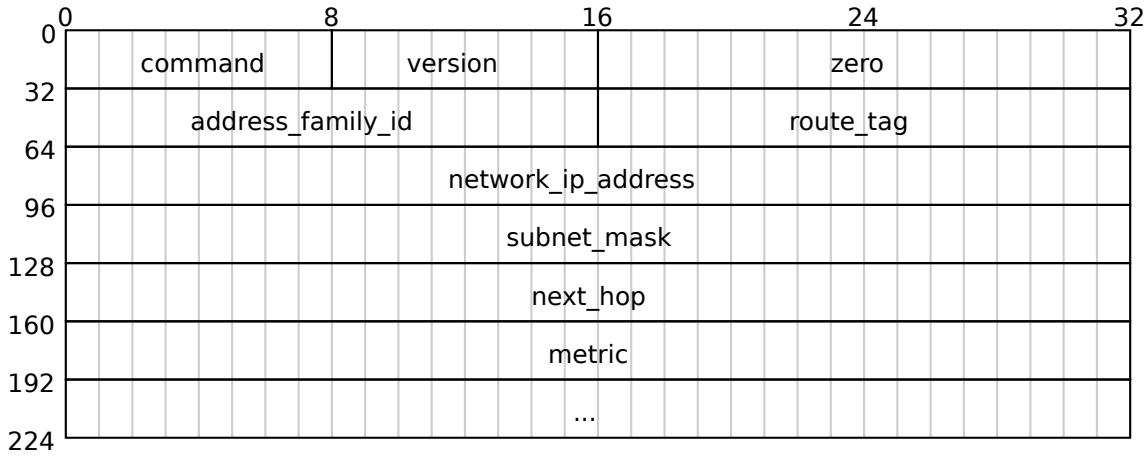


Рисунок 6.2 — Структура пакета RIPv2 (размеры полей указаны в битах)

Сообщение-запрос не содержит строк маршрутной таблицы и обычно отправляется маршрутизатором сразу после включения. Получив сообщение-запрос, другие маршрутизаторы рассылают сообщение-ответ с полным содержимым своей маршрутной таблицы. Если таблица содержит больше 25 записей, то отправляется несколько сообщений. Сообщения-ответы также рассылаются по таймеру; в таких случаях они не являются «ответом» на что-либо.

Строка маршрутной таблицы представлена следующими полями:

– **address_family_id**: идентификатор семейства адресов, число 2 обозначает IP;

– **route_tag**: атрибут маршрута, указывающий на его происхождение (например, если маршрут был получен от протокола внешней маршрутизации, то здесь указывается номер автономной системы);

– **network_ip_address**: адрес сети;

– **subnet_mask**: маска сети;

– **next_hop**: адрес следующего маршрутизатора в маршруте;

– **metric**: метрика (число «прыжков»).

Существует расширение RIPv2, называемое Triggered RIP [23]. Это расширение предполагает рассылку маршрутов только при их изменении, вместо рассылки по таймеру. Так как UDP-пакет, несущий измененный маршрут, может пропасть из-за временных помех в сети, Triggered RIP добавляет механизм доставки сообщений с подтверждениями и три новых типа пакетов (запрос, ответ и подтверждение). Мы не будем рассматривать Triggered RIP, поскольку нововведения этого расширения имеют мало общего с интересующими нас принципами обмена маршрутной информацией.

Задание № 35: Перехват сообщений протокола RIP

Использовать команду, подобную следующей (интерфейс менять при необходимости).

```
tcpdump -tvn -i eth0 -s 1518 udp
```

После параметра **-s** указан размер данных, которые...

Обычно таблица рассылается с учётом правила расщеплённого горизонта (англ. *split horizon*): если информация о маршруте была получена через некоторую сеть, то при посылке сообщения в эту же сеть данный маршрут не упоминается. Существует расширение этого правила, называемое «испорченное обратное обновление» (англ. *poisoned reverse*), когда маршрут включается в сообщение, но его метрика в сообщении равна «бесконечности».

По умолчанию протокол рассылает сообщения с интервалом 30 с.

Задание № 36: Протокол RIP. Использование правила испорченного обратного маршрута

Для включения испорченных обратных обновлений на интерфейсе **eth0** (например) следует добавить следующие строки в конец файла настройки службы RIP (**ripd.conf**).

```
interface eth0
ip rip split-horizon poisoned-reverse
```

Теперь в сообщениях, отправляемых в этот интерфейс, будут включены все сети. Однако в информации тех сетях, которые не были бы разосланы при обычном расщеплённом горизонте, будет указана метрика 16. Таким образом маршрутизатор не «умалчивает» о них, а прямо сообщает: «нет у меня этих сетей». Это «подтолкнёт» его соседей использовать другой маршрутизатор для пути к ним, благо такой маршрутизатор и находится в этом же сегменте (иначе бы правило расщепленного горизонта не сработало).

Как можно заметить, теперь в сообщениях протокола поле **next-hop** у ряда сетей стало отличаться от самого отправляющего. В данном случае

это, впрочем, не важно, поскольку метрика сети равна 16 и до анализа этого поля получатель сообщения не должен дойти.

Задание № 37: Протокол RIP. Отключение правила расщеплённого горизонта

Пример отключения расщепленного горизонта на интерфейсе **eth0** всё в том же файле настроек службы RIP.

```
interface eth0
no ip rip split-horizon
```

Теперь для ряда сетей, которые были бы исключены при включённом правиле расщеплённого горизонта, поле **next-hop** указывает адрес не самого отправителя, а следующий маршрутизатор на пути к данной сети. Именно значение этого поля соседи и должны использовать в качестве следующего, если вдруг им «понравится» этот маршрут — что, впрочем, маловероятно, поскольку метрика здесь указана на единицу больше, чем в сообщениях, который рассылает сам указанный в этом поле маршрутизатор в эту же самую сеть.

Следует отметить, что поле **next-hop** появилось в протоколе RIP уже в ходе его развития (его не было в RIPv1), и стандарт допускает, что некоторые реализации будут его игнорировать. Если это поле игнорируется, то отключение расщеплённого горизонта может привезти к образованию ложной маршрутной петли всего из двух соседних маршрутизаторов (в обычном RIPv2 для такой петли нужно как минимум три маршрутизатора, соединённых тремя сетями как треугольник).

6.8 Состояние маршрута в протоколе RIP

На текущий момент мы описали следующие события, влияющие на состояние маршрута в протоколе RIP: $E = \{E_{new}, E_{inf}, E_{same}, E_{shorter}, E_{expired}, E_{garbage}\}$. Состояния маршрута до некоторой сети в протоколе RIP и переходы между этими состояниями можно отобразить в виде конечного автомата с выходом, таким как показан на рисунке 6.3 [**?**].

На рисунке 6.3 действие R_{remove} показано в соответствии со стандартом протокола: в используемой нами реализации существует переход $\frac{E_{expired}, E_{bad}}{R_{remove}}$.

6.9 Устаревание маршрутов

Когда служба RIP видит, что сеть стала недоступной (состояние на рисунке 6.3), соответствующая строка выводимой таблицы протокола RIP вместо таймера устаревания показывает таймер уборки мусора.

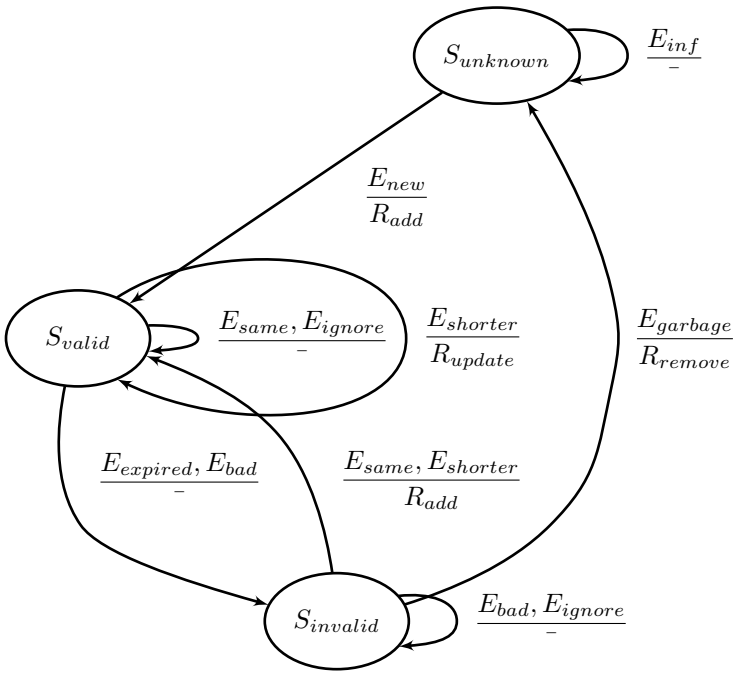


Рисунок 6.3 — Конечный автомат состояний маршрута в протоколе RIP

Задание № 38: Протокол RIP. Устаревшие маршрута

На маршрутизаторе **r6** выведем таблицу протокола RIP и убедимся, что путь до сетей 10.2.5.0/24 и 10.0.2.0/24 проходит через маршрутизатор **r5**. Отключим командой **halt** маршрутизатор **r5**, а на маршрутизаторе **r6** выведем несколько таблицу протокола RIP.

Мы увидим, как таймер устаревания у двух указанных сетей дойдёт до нуля, после чего сеть станет недоступной. Момент недоступности, возможно, не удастся заметить, поскольку маршруты до этих сетей начнут указывать на маршрутизатор **r3**, как только он пришлёт своё RIP-сообщение.

На машине **r6** маршрут до этой сети в таблице маршрутизации теперь ведёт к другом адресу маршрутизатора. Убедимся командой **traceroute**, что пакеты от **ws1** к **ws2** стали ходить через маршрутизаторы **r4** и **r3**.

Таким образом, RIP успешно переключился на более долгий, чем первоначальный, но вполне работающий маршрут. Теперь мы отключим второй маршрутизатор, что сделает ряд сетей недоступными.

Задание № 39: Протокол RIP. Сборка мусора

Отключим теперь маршрутизатор **r4** — это приведет к тому, что граф сети перестанет быть связанным. В течении двух минут после истечения таймера устаревания три недоступные сети на маршрутизаторе **r6** будут хранится в таблице протокола RIP с бесконечной метрикой. После истечения двухминутного таймера сборки мусора эти сети будут удале-

ны из неё. Убедитесь, что таблица маршрутизации на всех маршрутизаторах также их не содержит.

Для возврата системы в первоначальное состояние в соответствующих двух вкладках эмулятора терминала следует отдать команды **lstart -d net r4** и **lstart -d net r5**

6.10 Управление таблицей маршрутизации

Часть отмеченных событий протокола RIP изменяют системную таблицу маршрутизации.

1) При событии E_{new} сеть n добавляется в системную таблицу маршрутизации, в качестве адреса следующего маршрутизатора выступает h (обозначим это изменение как R_{add}).
2) При событиях E_{same}, E_{inf}, E_{ignore} таблица маршрутизации не меняется.
3) При событии $E_{shorter}$ в таблице маршрутизации для сети n маршрутизатор меняется на h, обозначим это изменение как R_{update}.
4) При событии $E_{garbage}$ нужно удалить запись про сеть n из таблицы маршрутизации, обозначим это изменение как R_{remove}.

Как видно, стандарт протокола RIP [22] предлагает удалять запись в маршрутной таблице как можно позже, при истечении таймера сборки мусора. Маршрутизатор будет до истечении этого таймера направлять пакеты на следующий маршрутизатор, несмотря на предположительную недоступность сети.

Используемая нами реализация протокола RIP отступает от стандарта: в ней запись из маршрутной таблицы происходит сразу при событии E_{bad}: в момент, когда метрика сети достигает бесконечности, она сразу удаляется из маршрутной таблицы, хотя стандарт оттягивает этот момент до сборки мусора. По этой причине в используемой нами реализации потерян сам смысл названия «сборка мусора» — в оригинале действительно имелась в виду в том числе сборка мусора из системной маршрутной таблицы, а не только из таблицы протокола RIP.

6.11 Программа перехвата сообщений RIP

Выше для перехвата и анализа сообщений протокола RIP использовалась команда **tcpdump**. Одним из её недостатков было то, что утилита показывала пакеты, отбрасываемые сетевым экраном ядра Linux. Это объясняется тем, что **tcpdump** находится до уровня сетевого экрана.

К счастью, у нас есть возможность узнать, какие пакеты будут действительно получены службой **ripd**. Для этого достаточно подключиться

к группе многоадресной рассылки, используемой протоколом RIP. Адрес группы известен: 224.0.0.9. Используемый UDP-порт — 520.

При подключении к группам многоадресной рассылки недостаточно просто связать сокет с адресом группы и портом. Необходимо явно указать информацию об интересующей нас группе с помощью функции **setsockopt**.

```
struct ip_mreqn membership; /* информация о multicast-группе */
...
/* вступаем в multicast-группу */
memset(&membership, 0, sizeof(membership));
membership.imr_multiaddr.s_addr = inet_addr("224.0.0.9"); /* адрес группы */
membership.imr_address.s_addr = INADDR_ANY;
membership.imr_ifindex = 3;  /* индекс сетевого интерфейса eth0 (см. `ip a`) */
if (setsockopt(sk, IPPROTO_IP, IP_ADD_MEMBERSHIP, (void *) &membership,
        sizeof(membership)) < 0)     perror("membership");    exit(1);
```

Эту операцию следует делать после открытия сокета и до связывания его с портом.

Поле **imr_ifindex** должно содержать индекс сетевого интерфейса, через который программа будет получать сообщения многоадресной рассылки. Индексы можно посмотреть, например, в выводе команды **ip a**. На наших виртуальных машинах индекс 3 соответствует интерфейу **eth0**, 4 — **eth1** и т.д.

Если программа будет запущена на машине, где уже работает служба **ripd**, при связывании сокета с адресом и портом может возникнуть ошибка. Чтобы её избежать, следует разрешить повторное использование адресов на сокете:

```
/* разрешаем повторное использование адресов */
int optvalue = 1;
if (setsockopt(sk, SOL_SOCKET, SO_REUSEADDR, (void *) &optvalue,
        sizeof(optvalue)) < 0)     perror("reuse addr");    exit(1);
```

На маршрутизаторе **r4** в каталоге **/root** лежит частично написанная программа-перехватчик **ripcatch.c**. Её необходимо закончить, вставив код для создания сокета, подключения к группе многоадресной рассылки и получения сообщений. При получении сообщения следует выводить IP-адрес отправителя. Отсутствующая часть исходного кода в целом аналогична содержанию программы **udpserver.c**, которая была рассмотрена в главе 5.

Задание № 40: Программа получения сообщений протокола RIP

Допишите программу **ripcatch.c**, как написано выше. Для компиляции следует использовать следующую команду — она использует компилятр стандарта C99, выводит расширенный список предупреждений и создаёт исполняемый файл **ripcatch**.

```
c99 -Wall ripcatch.c -o ripcatch
```

Крайне желательно, чтобы в результате компиляции вы не получили предупреждение (и, конечно же, не получили ошибок).

Теперь у нас есть наша программа для перехвата сообщений RIP и мы можем сравнить результаты работы с выводом перехватчика сетевых пакетов. Совпадение результатов позволит считать, что наша программа написано правильно.

Задание № 41: Проверка перехвата сообщений протокола RIP

Запустим команду **tcpdump** в фоне, как показано ниже (команда заканчивается символом амперсанда), а затем запустим нашу программу **ripcatch**.

```
tcpdump -tnv -i eth0 -s 1518 udp &
./ripcatch
```

Как известно, в UNIX-системах по соображениям безопасности для запуска программ из каталога вне переменной **PATH** необходимо указать путь к ним, а точка как раз означает текущий каталог — поэтому наща программа и запускается такой командой.

Поскольку сейчас запущены обе программы, то при приходе сообщений RIP их вывод программ будет перемежаться.

```
## это выводит tcpdump:
Out 4a:e4:d9:3b:f2:04 ethertype IPv4 (0x0800), length 128: (tos 0x0, ttl 1, id 0,
offset 0, flags [DF], proto UDP (17), length 112) 10.3.4.40.520 > 224.0.0.9.520:
        RIPv2, Response, length: 84, routes: 4
          AFI: IPv4:        10.0.1.0/24, tag 0x0000, metric: 3, next-hop: self
          AFI: IPv4:        10.0.2.0/24, tag 0x0000, metric: 2, next-hop: self
          AFI: IPv4:        10.1.2.0/24, tag 0x0000, metric: 2, next-hop: self
          AFI: IPv4:        10.2.4.0/24, tag 0x0000, metric: 1, next-hop: self
## а это уже вывод нашей программы:
from 10.3.4.40:
        AFI 2 addr 10.0.1.0/24, tag 0x0000, metric 3,  nexthop: 0.0.0.0
        AFI 2 addr 10.0.2.0/24, tag 0x0000, metric 2,  nexthop: 0.0.0.0
        AFI 2 addr 10.1.2.0/24, tag 0x0000, metric 2,  nexthop: 0.0.0.0
        AFI 2 addr 10.2.4.0/24, tag 0x0000, metric 1,  nexthop: 0.0.0.0
...
```

6.12 Контрольные вопросы

Для самоконтроля полученных знаний рекомендуется ответить на следующие вопросы.

1) Что такое динамическая маршрутизация?
2) Какие задачи решает протокол RIP?
3) При каких событиях известная сеть становится недоступной с точки зрения протокола RIP?

4) По каким причинам значение метрики сети в протоколе RIP так сильно ограничено сверху?

5) Должна ли служба RIP ухудшить метрику сети, если она получила худшую метрику от маршрутизатора, являющегося следующим?

6) В чём различия и что общего между таблицей протокола RIP и маршрутной таблицей?

7) Служба RIP обнаружила, что некоторая сеть стала недоступной. Будет ли она удалена сразу таблицы протокола RIP? Почему?

8) Почему после старта системы при работе только службы RIP таблица кеша ARP пуста?

9) Какой IP-адрес стоит в поле адресов назначения (IP и MAC) в сообщениях протокола RIP? Почему?

10) Что такое правило расщеплённого горизонта в RIP?

11) Как заполняется поле следующего маршрутизатора в сообщениях RIP? Рассмотрите случай включённого расщепленного горизонта и отключенного.

12) В чём плюсы и минусы испорченных обратных обновлений RIP?

13) За сколько времени в примере гарантированно происходит полный обмен маршрутной информацией, если данные в сети не теряются?

6.13 Выполнение самостоятельной работы

Для выполнения индивидуальных вариантов заданий надо скачать шаблон сети, созданный специально для этой цели.

```
mkdir -p ~/tcp-ip; cd ~/tcp-ip
wget -r http://ftp.iu7.bmstu.ru/nets/lab-rip-hw.tar.gz
rm -rf lab-rip-hw
tar -xvf lab-rip-hw.tar.gz
cd lab-rip-hw
```

Адреса сетевых интерфейсов можно выбрать любыми, для сетевого интерфейса маршрутизатора, выходящего в тупиковую сеть, рекомендуется выбирать адрес, оканчивающийся на «.1». Для контроля заданных адресов рекомендуется применять следующий порядок выполнения работы, примечания к нему приведены ниже.

1) Удалить из шаблона всё лишние файлы, как вы уже делали в главе 3.

2) Настроить файл **labs.conf** в соответствии с полученным заданием.

3) Настроить файлы сетевых настроек для всех виртуальных машин.

4) Собрать промежуточную версию отчета (командой **make** в каталоге **report**), визуально убедится, что сети заданы правильно.

5) Выбрать маршрутизатор, который будет отключён в ход опыта, некоторый другой маршрутизатор выбрать для соединения с интернетом и изменить его настройки (см. ниже).

6) Перестроить отчет для проверки правильности конфигурации сети.

7) Запустить виртуальные машины и приступить к выполнению основной части работы.

8) Провести опыты и заполнить отчет.

Для маршрутизатора, выбанного для соединения наше сети с интернетом, нужно указать в файле **labs.conf** следующую строку (вместо **rX** указать выбранный маршрутизатор).

```
rX[3]=tap,172.16.0.1,172.16.0.20
```

Как видно по ней, интерфейс **eth3** этого маршрутизатора будет подключён к реальной сети и получит IP-адрес 172.16.0.20 (адрес 172.16.0.1 будет присвоен реальной машине в этой сети, см. рисунок 6.1 на стр. 112). При запуске виртуальной машины Netkit автоматически соединит интерфейс **eth3** маршрутизатора с реальной сетью, указывать какие-либо дополнительные настройки для этого интерфейса не нужно. На этом же маршутизаторе следует включить трансляцию сетевых адресов, убрав в файле **r1/etc/rc.local** комментарий из строки включения трансляции («маскарада») сетевых адресов для пакетов, проходящих через интерфейс **eth3**.

```
iptables -t nat -A POSTROUTING -o eth3 -j MASQUERADE
```

Напомним, что файлы из конфигурации лабораторной работы копируются в виртуальную машину только единожды, при создании образа диска машины, так что возможно изменение в файле **/etc/rc.local** вам придётся сделать сразу в виртуальной машине.

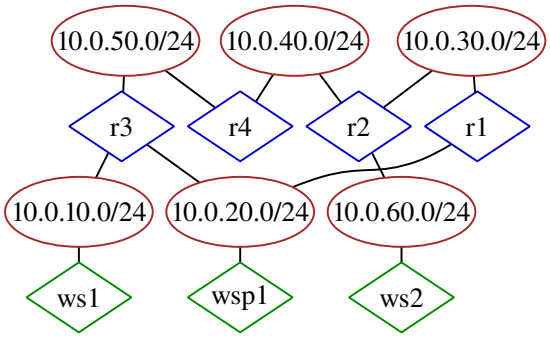
В качестве последней настройки данного маршрутизатора отключить распространение им по протоколу RIP информации о сети 172.16.0.0/12 — информацию об этой сети нет нужды распространять в нашей системе. Для этого нужно отключить в настройках распространение информации о подключенных сетях, закомментировав в файле **ripd.conf** строку **redistribute connected** и перезапустить службу Quagga.

6.14 Варианты заданий для самостоятельной работы

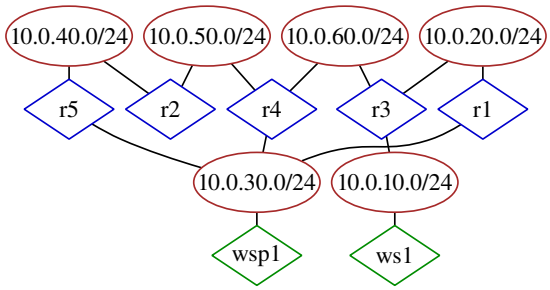
В индивидуальном задании на работу указываются только адреса сетей. При настройки сети идентификаторы виртуальных сетей можно выбирать любые. Для машин с единственным сетевым интерфейсом в файле **interfaces** следует указать, кроме адреса и маски, маршрутизатор по умолчанию.

В вариантах не указано, какой маршрутизатор будет иметь подключение к интернет — вам следует выбрать его самостоятельно так, чтобы он не совпадал с маршрутизатором, отключаемым при опыте с поиском новых маршрутов.

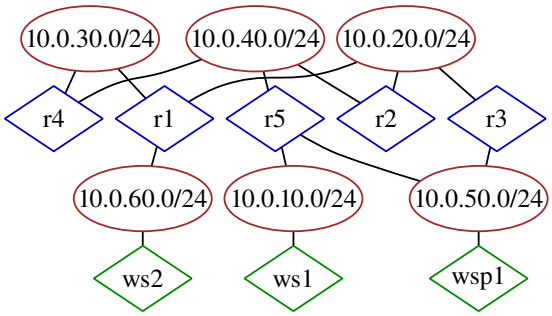
Вариант № 1



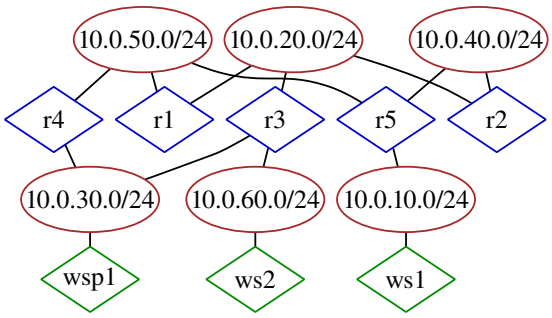
Вариант № 2



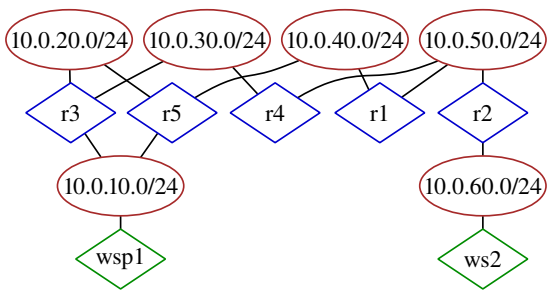
Вариант № 3



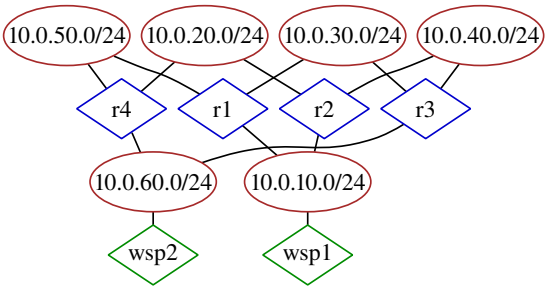
Вариант № 4



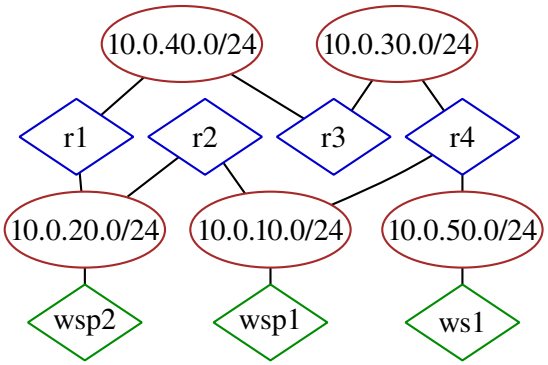
Вариант № 5



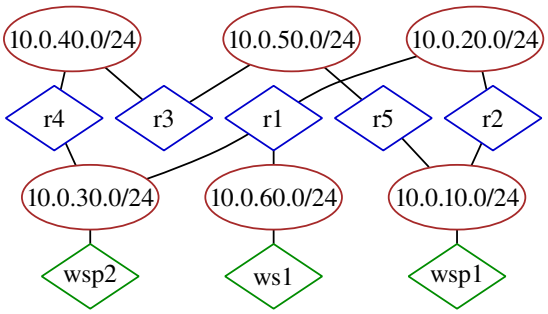
Вариант № 6



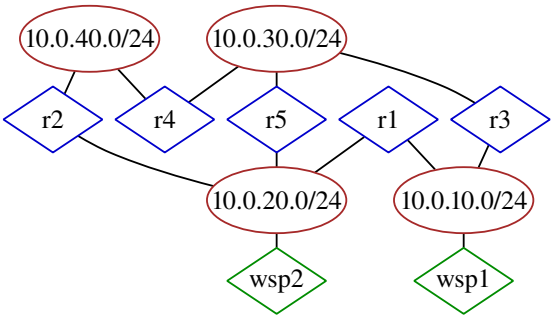
Вариант № 7



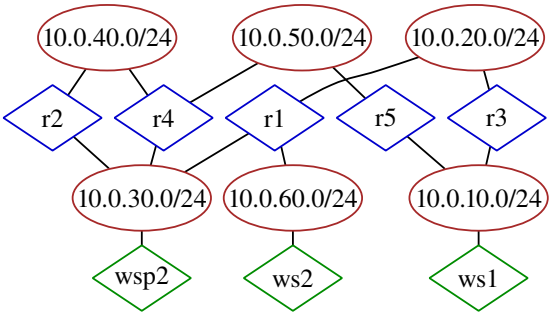
Вариант № 8



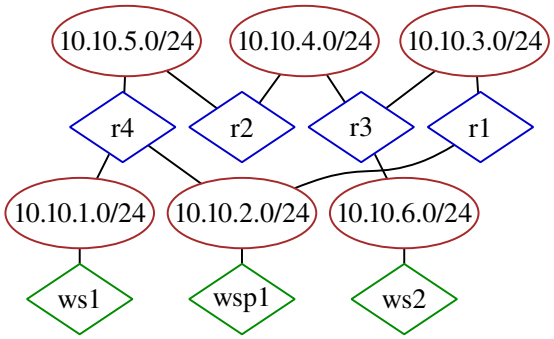
Вариант № 9



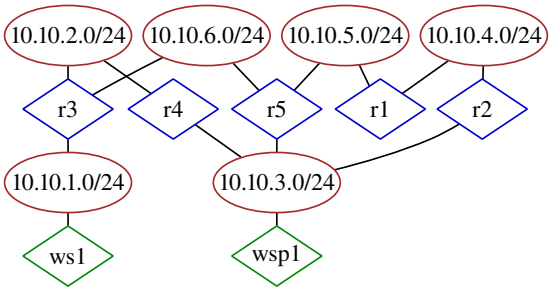
Вариант № 10



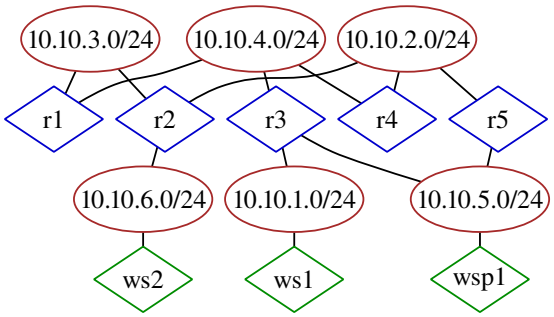
Вариант № 11



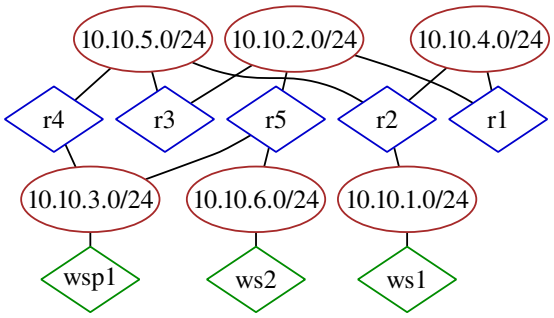
Вариант № 12



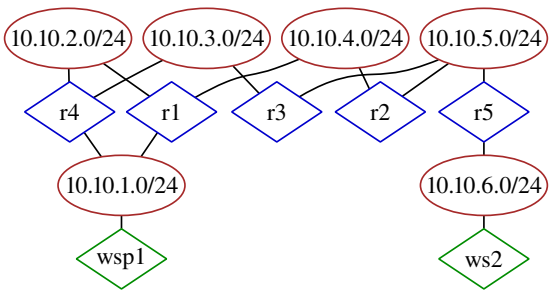
Вариант № 13



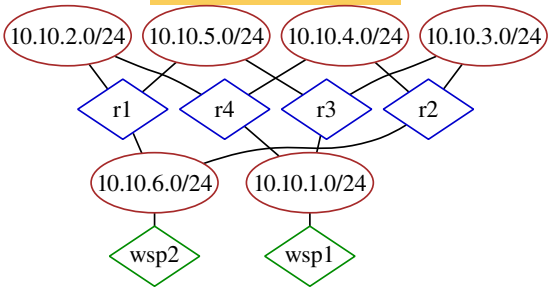
Вариант № 14



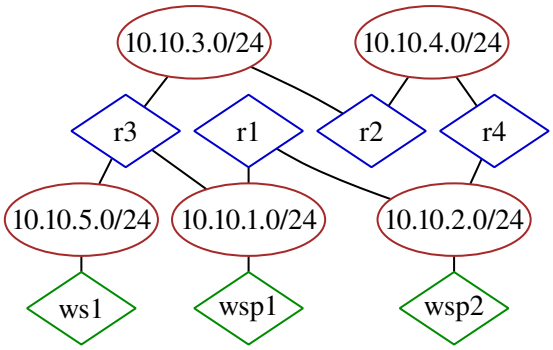
Вариант № 15



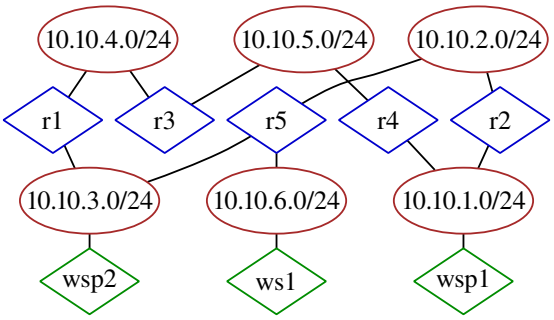
Вариант № 16



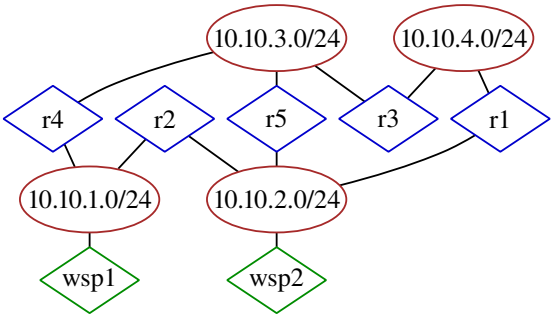
Вариант № 17



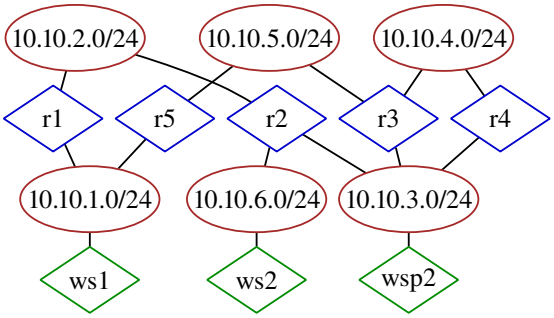
Вариант № 18



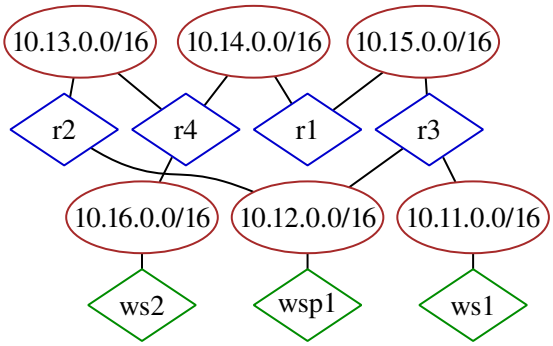
Вариант № 19



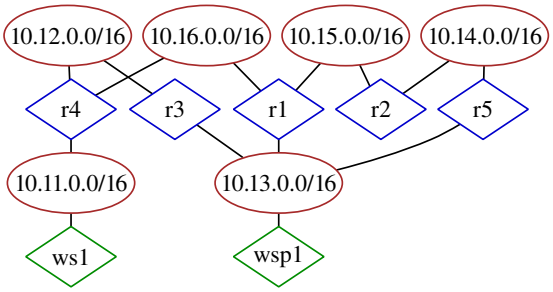
Вариант № 20



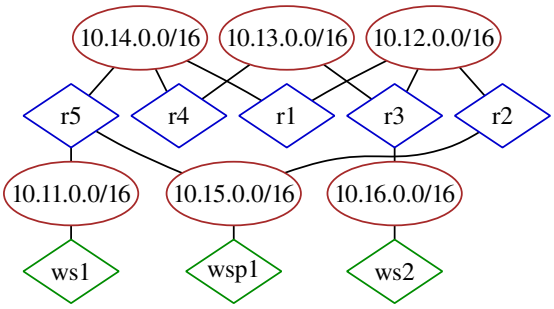
Вариант № 21



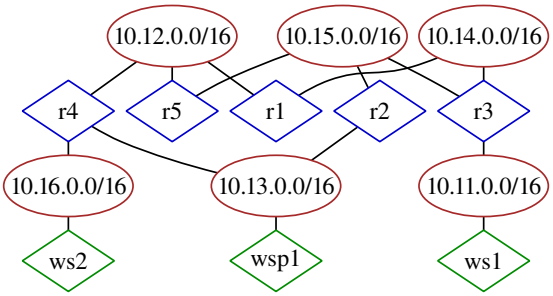
Вариант № 22



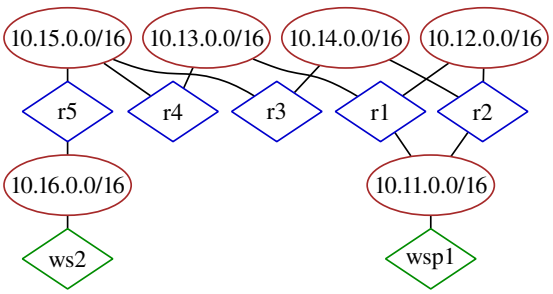
Вариант № 23



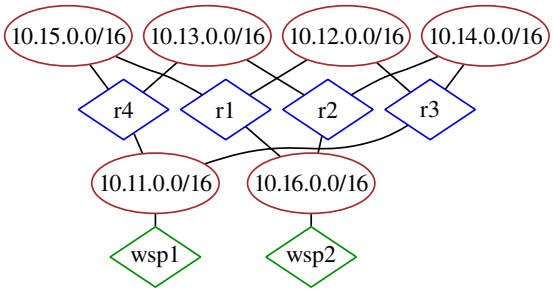
Вариант № 24



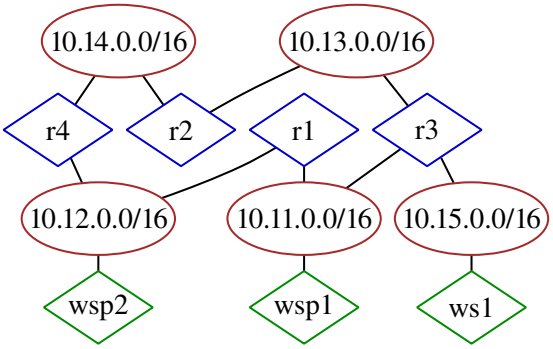
Вариант № 25



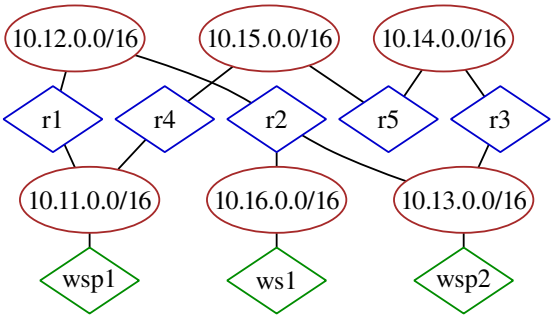
Вариант № 26



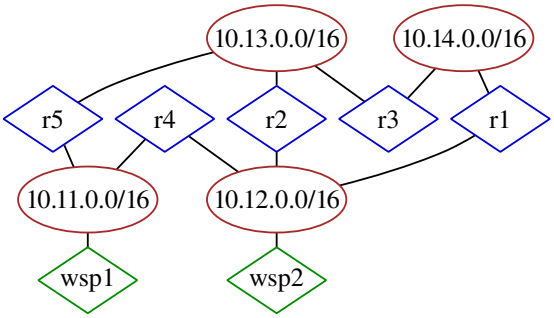
Вариант № 27



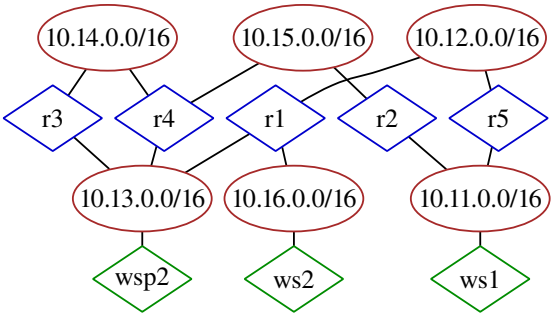
Вариант № 28



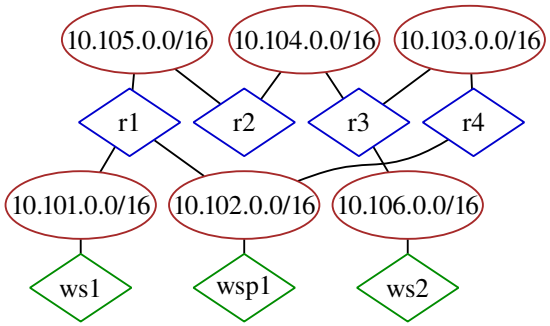
Вариант № 29



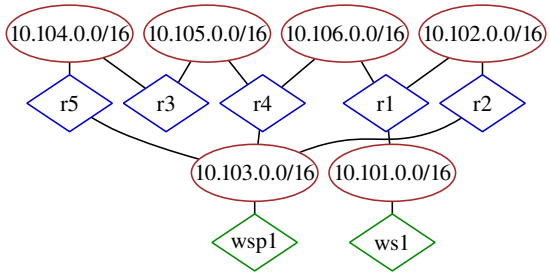
Вариант № 30



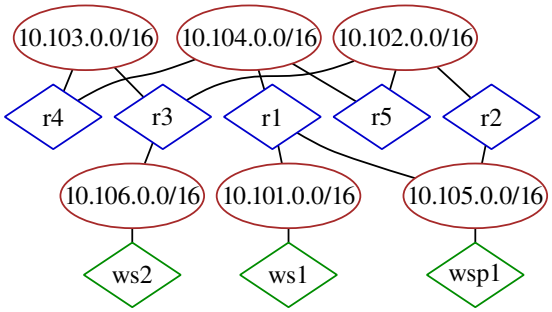
Вариант № 31



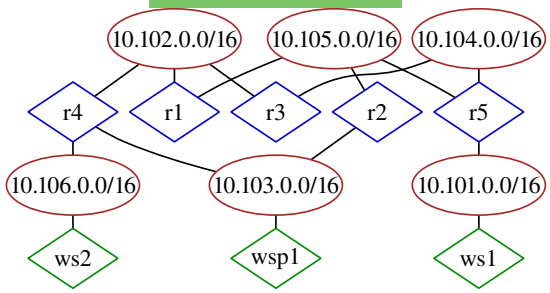
Вариант № 32



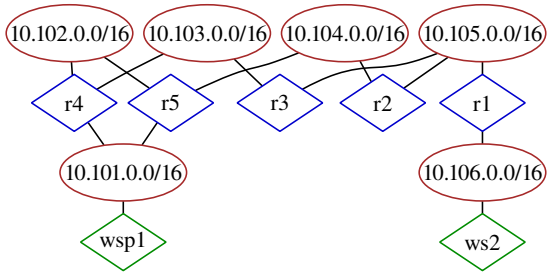
Вариант № 33



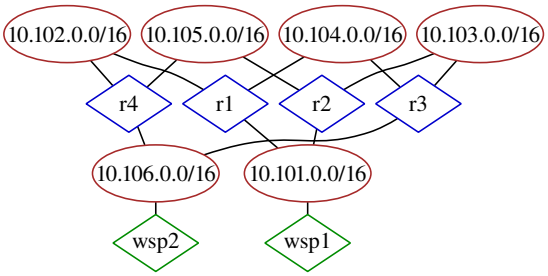
Вариант № 34



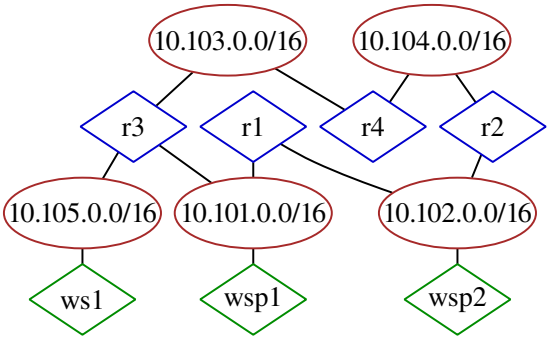
Вариант № 35



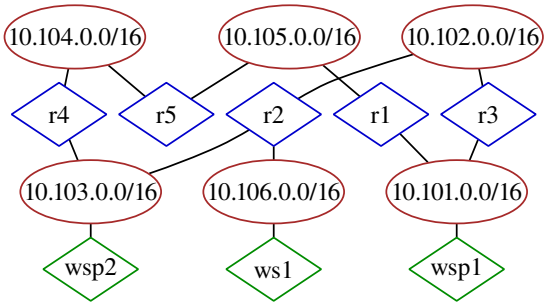
Вариант № 36



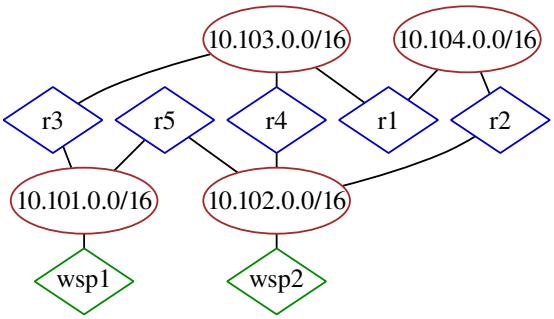
Вариант № 37



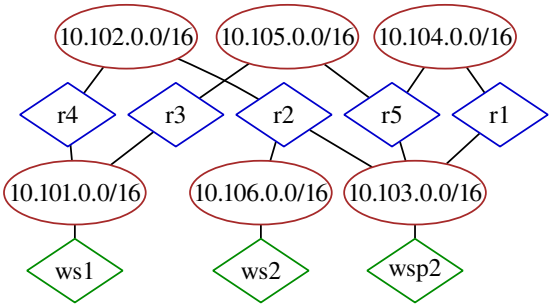
Вариант № 38



Вариант № 39



Вариант № 40



7 Транспортный протокол TCP и сетевые сокеты

Для передачи данных между процессами используются протоколы TCP и UDP, работающие поверх протокола IP. Протокол UDP, как мы видели в главе 5 представляет собой простейшую надстройку поверх IP: в его собственном заголовке добавляются только номера портов и контрольная сумма данных. Протокол TCP, наоборот, является весьма сложным, и пытается решить задачу надёжной и достаточно эффективной доставки потока данных произвольного размера.

Данная глава содержит общие сведения о протоколе TCP и рассматривает его использование в прикладных программах. В следующей главе подробно рассматриваются различные механизмы протокола TCP, которые служат для обеспечения надежной доставки и повышения эффективности передачи.

7.1 TCP-сегмент

Протокол TCP предназначен для передачи непрерывного потока данных между двумя процессами. Для этого поток разбивается на части, называемые *сегментами*. Каждый сегмент предваряется заголовком и помещается в один IP-пакет. Структура заголовка TCP-пакета показана на рисунке 7.1.

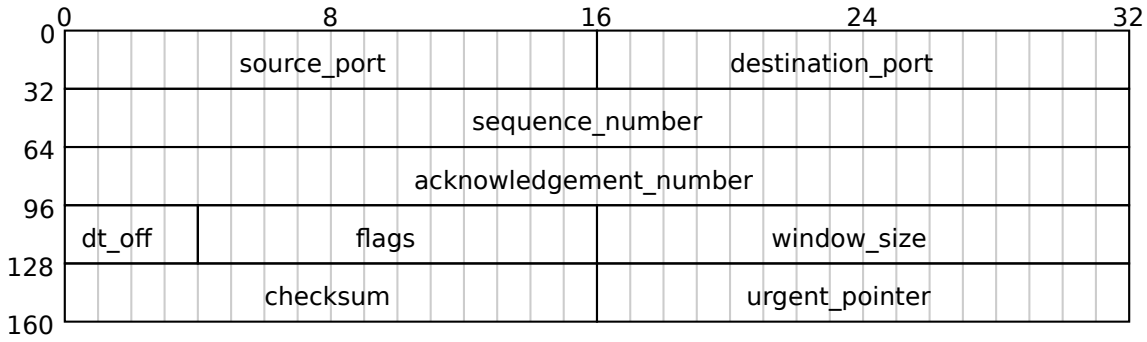


Рисунок 7.1 — Заголовок пакета TCP (размеры полей указаны в битах)

Заголовок содержит номера TCP-портов получателя и отправителя в полях **source_port** и **destination_port**. Эти порты имеют назначение, аналогичное портам протокола UDP из главы 5: четвёрка из двух портов и двух IP-адресов идентифицирует TCP-соединение.

Поскольку TCP должен передавать длинную последовательность байтов, то ему необходимо указывать место передаваемого сегмента в данной последовательности. Для этого существует поле номера последовательности (**sequence_number**) определяет позицию сегмента в потоке — оно содержит номер в потоке первого байта в передаваемом сегменте. Номера не считаются с нуля или единицы: при установке соединения стороны

выбирают случайное начальное значение номеров. Под номер последовательности выделено всего 32 бита, но это не накладывает ограничений на длину потока: при достижении максимального значения счетчик сбрасывается в 0 и продолжает увеличиваться.

Поскольку протокол TCP пытается передать все требуемые данные, то он использует механизм подтверждений. С помощью номера подтверждения (поле **acknowledgement_number**) получатель указывает, какая часть передеваемой последовательности успешно до него дошла В этом поле указывается номер первого байта после непрерывной области принятых данных. Подтверждение, таким образом, высылается не на сегмент, а на все полученные байты с начала потока и до указанной позиции. Отправитель, как мы увидим позднее, повторяет передачу потерянных сегментов, основываясь на значении **acknowledgement_number** в ответах получателя; так в TCP организовывается надёжная доставка.

Заметим, что по одному TCP-соединению передаются два независимых потока данных: от клиента к серверу и в обратном направлении. Пакеты, идущие от клиента к серверу, содержат номера последовательностей в «прямом» потоке и номера подтверждений на сегменты, идущие во «встречном» потоке от сервера к клиенту. Ситуация для обратного направления симметрична. Если одна из двух сторон не передаёт данные, а только принимает, то от неё идут пакеты, отличающиеся только номерами подтверждений и не содержащие тела.

Заголовок TCP-сегмента содержит набор флагов, наиболее важными из них являются следующие:

– **SYN** — флаг установки соединения;
– **FIN** — флаг завершение соединения (каждая сторона независимо закрывает соединение со своей стороны);
– **ACK** — признак того,что пакет содержит номер подтверждения.

Первый пакет, отправляемый клиентом при установке соединения, содержит флаг SYN; все последующие содержат ACK.

Из прочих полей, присутствующих в заголовке сегмента, можно отметить контрольную сумму данных сегмента (**checksum**) и поле **window_size**, используемое в механизме скользящего окна, о котором будет рассказано в следующей главе.

Поле **dt_off**) (от англ. *data offset*) определяет границу заголовка TCP, вместе со всеми опциями. В заголовке TCP-сегмента нет ни поля длины данных, ни поля общей длины сегмента: последнюю легко получить из заголовка IP, а размер полезной нагрузки сегмента сожно установить как разницу полезной нагрузки IP-пакета (после дефрагментации, конечно же) и значения поля **dt_offset**).

7.2 Установка и разрыв соединения

Протокол TCP предполагает установку двустороннего соединения между клиентом и сервером. На рисунке 7.2 показан упрощенный конечный автомат соединения протокола TCP на стороне сетевого клиента (программы, инициирующей общение по сети). На рисунке 7.3 показан аналогичный конечный автомат для стороны сервера — сетевой службы, ожидающей подключения клиентов. Для каждого состояния в скобках указано его «каноническое» название, которое можно встретить в выводе служебных программ типа netstat.

Показанные автоматы охватывают в основном установку и разрыв соединения, поэтому являются упрощенными. Полное формальное описание автоматов можно найти в [24].

Показанные специальные типы сообщений (SYN, FIN, ACK) устанавливаются соответствующими флагами в пакете TCP. Поскольку каждый флаг может быть установлен независимо, существуют невалидные их комбинации, что иногда используется злоумышленниками. Полный перечень флагов и их назначение можно посмотреть в [25, 26].

7.3 Сетевое программирование. Сокеты протокола TCP

Рассмотрим простейшие клиентские и серверные приложения, использующие этот протокол.

При работе с TCP в основном используются следующие функции API для работы с сокетами[1]:

– **socket()** — создание сокета (для TCP аргументы — AF_INET, SOCK_STREAM);
– **bind()** — привязка сокета к адресу и порту (на сервере);
– **listen()** — переход в состояние ожидания соединений (на сервере);
– **accept()** — получение клиентского сокета для входящего соединения (на сервере);
– **connect()** — создание соединения с сервером (на клиенте);
– **send(), recv()** — передача и приём данных;
– **close() или shutdown()** — закрытие сокета.

При возникновении ошибок эти функции возвращают отрицательное значение. Функция **recv** также вернёт ноль, если другая сторона завершила соединение.

Работа клиентской программы с сокетами достаточно проста (ниже приведен псевдокод):

[1]Напомним, что эти функции являются частью так называемого BSD socket API, присутствующего практически в неизменном виде в большинстве операционных систем.

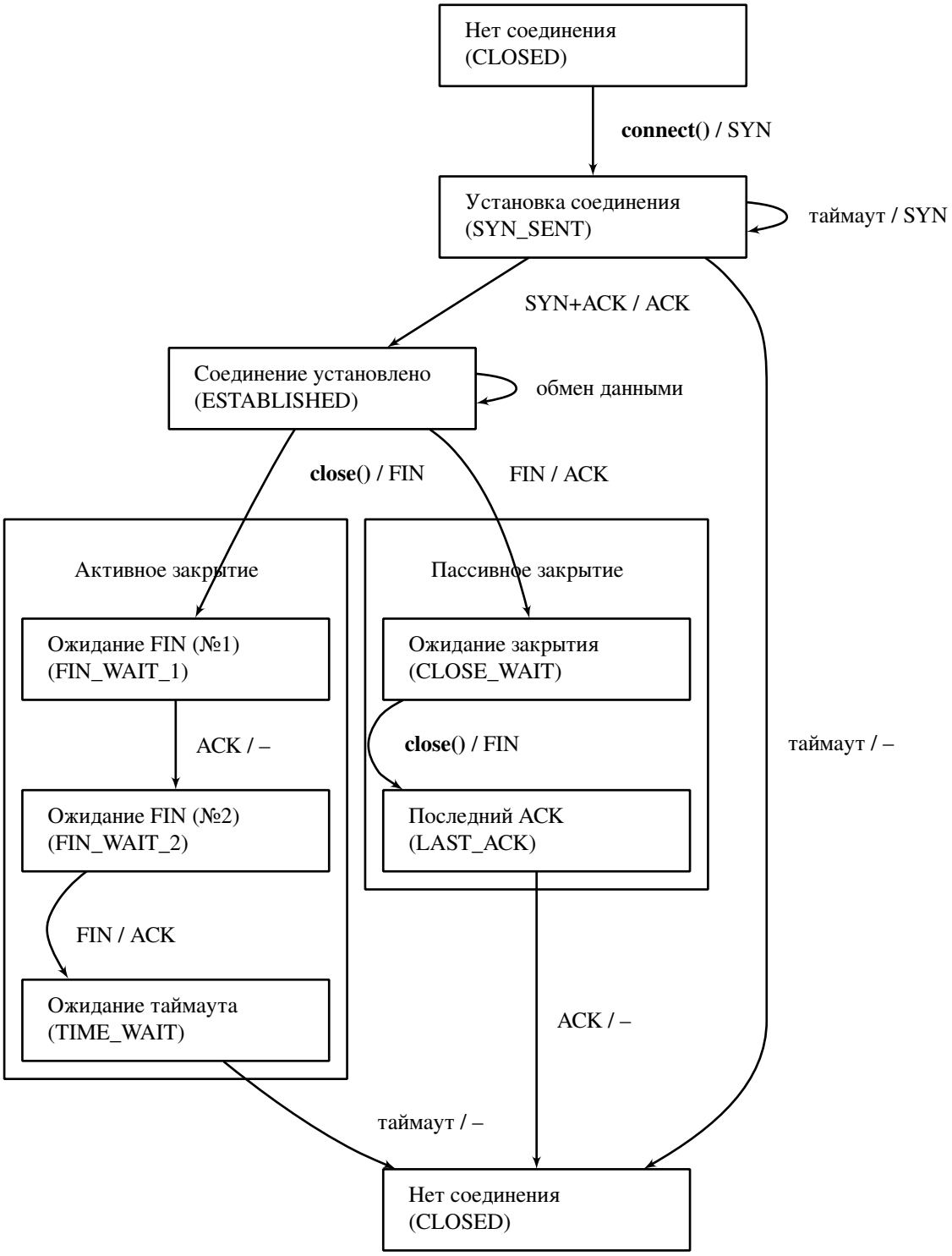


Рисунок 7.2 — Упрощённая диаграмма состояний TCP-клиента

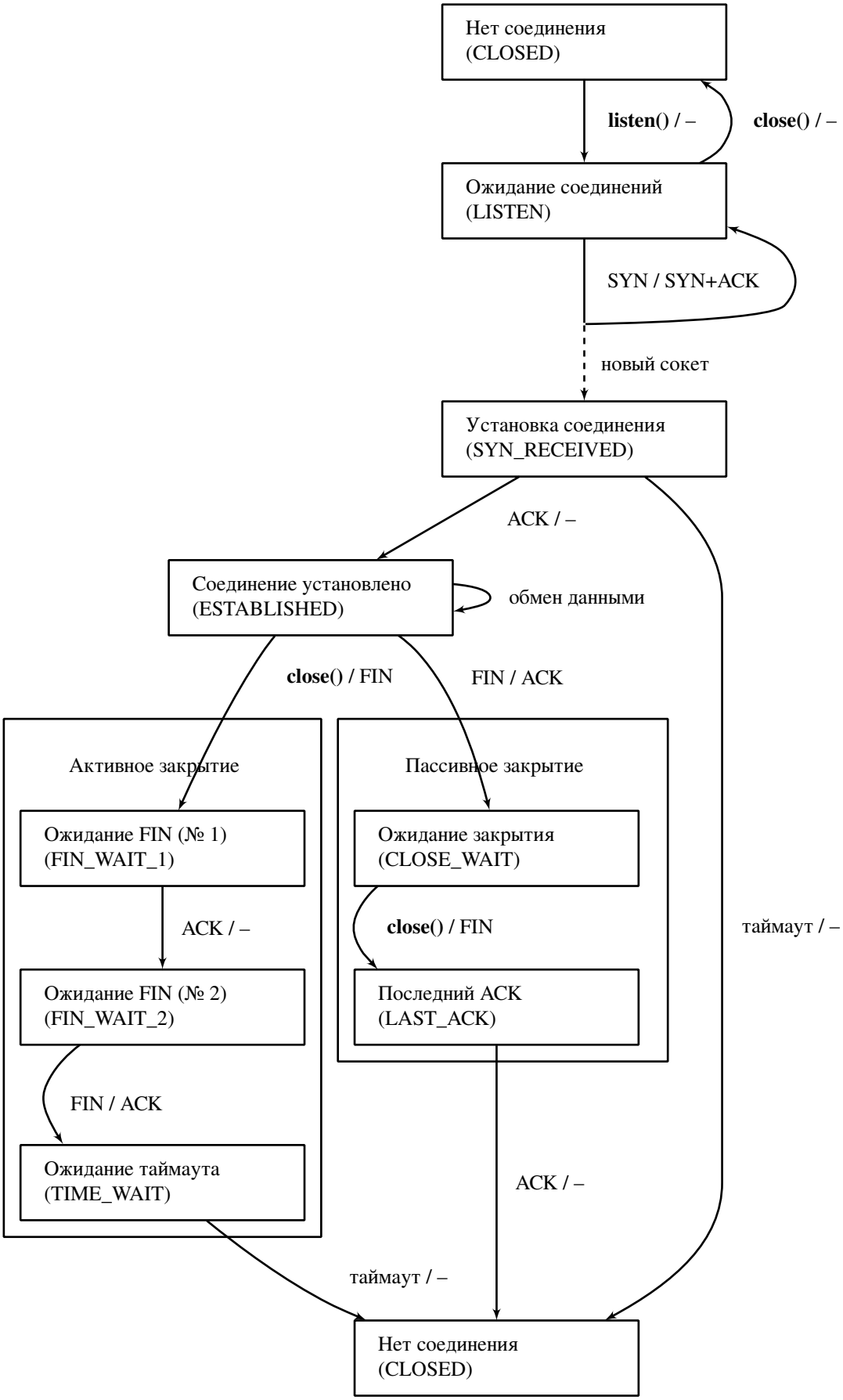


Рисунок 7.3 — Упрощённая диаграмма состояний TCP-сервера

```
int sock = socket(AF_INET, SOCK_STREAM);
connect(sock, "10.30.0.2", 1996); // адрес и порт сервера
/* ...передача данных с использованием send и recv... */
close(sock);
```

Серверная программа должна выполнять две задачи: приём входящих соединений от клиентов и обслуживание отдельных клиентов. Простейший сервер выглядит следующим образом.

```
int sock = socket(AF_INET, SOCK_STREAM);
bind(sock, INADDR_ANY, 1996);
listen(sock, SOMAXCONN);
while (1)     int client_sock = accept(sock);    /* обслуживание клиента (send, re
```

Заметим, что в коде фигурируют два сокета, а не один. Первый (**sock**) помещается в состояние ожидания соединений функцией **listen**. Любая попытка установить соединение с сервером приводит к созданию нового, «клиентского» сокета. Новый сокет переводится в состояние «Установка соединения» (рисунок 7.3) и помещается в так называемую *backlog*-очередь. Находясь в этой очереди, «клиентский» сокет может перейти в состояние «Соединение установлено». Функция **accept** извлекает[1] «клиентский» сокет из очереди и передаёт его дескриптор приложению. Из этого, кстати, следует, что в приложении-клиенте функция **connect()** может вернуть успешный результат задолго до того, как сервер вызовет **accept**.

Размер *backlog*-очереди задаётся вторым аргументом функции **listen**. Максимальный размер ограничен системной переменной SOMAXCONN — в Debian / Ubuntu её значение по умолчанию равно 128.

Работоспособные примеры простейших клиента и сервера можно найти в файлах виртуальной сети: **net-tcp/c1/root/echocl.c** и **net-tcp/c4/root/echosrv.c** соответственно.

7.4 Схемы обслуживания клиентов

Обычно от сервера требуется способность работать со множеством подключившихся клиентов. Приведённый выше простейший пример сервера не позволяет обслуживать их одновременно: подключившиеся к серверу клиенты должны ждать, пока текущий клиент не будет полностью обслужен (т.е. до **close(client_sock)**). У нас есть несколько возможностей это исправить.

[1] Документация по **accept** обычно утверждает, что как раз вызов этой функции создаёт новый сокет для соединения с клиентом. Это верно, если называть сокетом структуру с буферами приёма и передачи, связанную с файловым дескриптором. В то же время, сокет как сущность, являющаяся конечной точкой соединения и обладающая состоянием, появляется в момент прихода SYN-пакета и проходит через состояния «Установка соединения» и «Соединение установлено» *независимо* от серверного сокета (**sock** в примере). Функция **accept** лишь оборачивает такую сущность в новый экземпляр структуры-сокета.

Один клиент — один поток (процесс).. Одним из решений задачи обслуживания нескольких клиентов является использованием мультипроцессных или многопоточных возможностей операционной системы: можно создать отдельный процесс или поток для обслуживания одного клиента. В случае создания процесса на каждого клиента в псевдокоде это будет выглядеть следующим образом.

```
while (1)     client_sock = accept(sock);    pid = fork();    if (pid == 0)
```

Заметим, что дочерний процесс здесь закрывает клиентский сокет после завершения обслуживания, в то время как родительский делает это немедленно. Вызов **fork()** приводит к копированию таблицы дескрипторов родительского процесса, содержащей в том числе и дескриптор сокета. Родительскому процессу этот дескриптор больше не нужен, и его необходимо освободить. После того, как функция **close()** будет вызвана дочерним процессом, дескрипторов, ссылающихся на сокет, больше не останется, и соединение будет закрыто.

Многопоточный подход в описанном виде, имеет ряд недостатков. Создание нового процесса или потока — сравнительно затратная операция для операционной системы. В периоды пиковой нагрузки на сервер большое число одновременно запущенных процессов может исчерпать ресурсы системы, а непрерывное создание, переключение и завершение процессов только усугубит ситуацию.

Практика показывает, что традиционный подход «один клиент — один поток» перестаёт работать, когда число одновременных запросов к серверу достигает порядка нескольких тысяч. Эта ситуация известна как «Проблема C10K» (проблема десяти тысяч клиентов) — в [27] подробно описаны её возможные решений. Ниже рассмотрены некоторые из них.

Пул потоков.. Часто применяют пул заранее созданных процессов или потоков ограниченного размера. Если все процессы в настоящий момент заняты, новые соединения остаются в *backlog*-очереди, пока какой-либо процесс не освободится. Псевдокод:

```
pool = create_process_pool(32);
while (1)     pid = get_idle_process(pool); /* блокируется, пока процесс не освобо
```

Мультиплексирование при работе с сокетами: select и poll.. Если процессы / потоки сервера большую часть времени проводят в ожидании сообщений от клиента или вспомогательных служб, а не занимаются сложными вычислениями, пользы от многопоточности мало. Альтернативой в этом случае является ожидание событий на множестве сокетов в одном потоке выполнения. Для этого используются функции **select()** или **poll()**.

Общая идея такова: в функцию select / poll передается массив дескрипторов сокетов. Функция блокирует выполнение до тех пор, пока один

из сокетов в массиве не станет доступен для неблокирующего чтения (т.е. поступили новые данные или новое соединение) или записи (освободился буфер отправки), либо пока не истечёт указанный таймаут. Программист может указать, какие события его интересуют на каждом из сокетов. Всё взаимодействие с сокетами осуществляется в одном потоке выполнения, что снимает необходимость синхронизации доступа к разделяемым ресурсам.

Программный код, написанный с использованием вызовов select / poll, не содержит конструкций типа «отправили сообщение клиенту и ждем, пока он ответит» и пишется по схеме «отправили сообщение и ищем, кого ещё можно обслужить». О таком коде говорят, что он написан в асинхронном стиле.

Функции **select()** и **poll()** различаются способом передачи параметров; **poll()** можно считать более продвинутой версией **select()**. Рассмотрим поподробнее использование **select()**. Эта функция работает со множествами файловых дескрипторов (тип данных **fd_set**, по сути — битовая маска). Программа-сервер формирует множество сокетов, события на которых её интересуют, с использованием макросов **FD_SET** и **FD_CLR**. Функция **select()** принимает указатели на три множества: для проверки доступности чтения, записи и исключительных ситуаций. После возврата из **select()** множества будут очищены от всех дескрипторов, кроме тех, которые сейчас доступны для чтения или записи.

Следующий листинг демонстрирует использование **select()** для ожидания входящих соединений или данных на наборе сокетов. Проверки ошибок и другие несущественные детали опущены.

```
int server_sk, client_sk, sk;
fd_set socket_set, ready_set;
char buf[BUFSIZE];
int len;
...
FD_ZERO(&socket_set);
server_sk = socket(AF_INET, SOCK_STREAM, 0);
FD_SET(server_sk, &socket_set);
bind(sk, ...);
listen(sk, SOMAXCONN);

while (1)     ready_set = socket_set;    select(FD_SETSIZE, &ready_set, NULL, NULL
```

На что следует обратить внимание в данном примере:

– серверный сокет **server_sk** также добавляется в **socket_set**. Входящие соединения делают его доступным для «чтения» (т.е. вызова **accept()**;

– закрытие клиентских сокетов сопровождается их удалением из **socket_set**;

– пассивное закрытие соединения приводит к тому, что сокет становится доступным для чтения, но **recv** возвращает ноль.

В приведённом коде цикл **for** проверяет доступность для каждого файлового дескриптора в диапазоне от 0 до FD_SETSIZE. Значение константы FD_SETSIZE обычно равно 1024; это максимальное число битов, которое может вместить тип **fd_set**. Поэтому **select()** неприменима, если процесс имеет более 1024 открытых файлов или сокетов. Функция **poll()** лишена такого ограничения, но использовать её сложнее. Если число открытых сокетов меньше максимума, возможно, стоит указать меньший диапазон (первый параметр **select()** и граница цикла **for**) — например, самый большой дескриптор из открытых на данный момент, плюс единица.

Заметим, что select и poll не являются единственными средствами организации асинхронной работы с сокетами. Можно отметить такие механизмы, как **epoll** — расширенную версию poll в Linux, и **kqueue** в FreeBSD. Однако, эти вызовы не являются переносимыми.

Неблокирующие сокеты.. В заключение отметим, что операции с сокетами могут выполняться в неблокирующем режиме. Например, если сокет помещён в неблокирующий режим, то функция **recv** немедленно вернёт код ошибки EWOULDBLOCK[1] вместо того, чтобы заблокировать выполнение в ожидании поступления данных. Следующий код позволяет поместить сокет **sock** в неблокирующий режим:

```
fnctl(sock, F_SETFL, fnctl(sock, F_GETFL, 0) | O_NONBLOCK);
```

Работа с неблокирующими сокетами предполагает их опрос в цикле, как показано ниже.

```
while (1)     for (i = 0; i < n_sockets; i++)         len = recv(sockets[i], buf,
```

Опрос в большинстве случаев является пустой тратой процессорного времени, и использование **select()** гораздо более эффективно. Тем не менее, в некоторых ситуациях нужно немедленно получить ответ, доступен ли сокет для чтения / записи или нет — например, чтобы отразить состояние в пользовательском интерфейсе. Неблокирующие вызовы можно комбинировать с **select()**, который блокируется всегда.

Подводя итоги, перечислим рассмотренные выше схемы обслуживания клиентов:

1) обслуживание по очереди в одном потоке;
2) многопоточный сервер (по одному потоку на клиента);
3) вариация с пулом потоков или процессов;
4) использование **select()** или **poll()** в одном потоке;
5) опрос неблокирующих сокетов.

[1]Функция, разумеется, вернёт -1, а глобальная переменная **errno** будет установлена в EWOULDBLOCK.

За дополнительными сведениями отсылаем читателя к статье [27].

7.5 Выполнение задания

Загрузим и распакуем файлы виртуальной сети для выполнения задания.

```
mkdir -p ~/tcp-ip; cd ~/tcp-ip
wget -r http://ftp.iu7.bmstu.ru/nets/net-tcp.tar.gz
rm -rf net-tcp
tar -xvf net-tcp.tar.gz
cd net-tcp
```

На машинах виртуальной сети **net-tcp** есть исходные тексты простейших клиента и сервера:

– клиент **echocl.c** — на машинах **c1** и **c5**;
– сервер **echosrv.c** — на машине **c4**.

Клиент считывает строки из консоли, отправляет серверу и выводит ответ сервера в консоль. Сервер просто возвращает полученные данные отправителю (эхо). Сервер работает по простейшей однопоточной схеме.

Изучите исходные тексты программ, обращая внимание на обработку ошибок. Поэкспериментируйте с одновременным подключением двух клиентов к серверу и с аварийным (Ctrl-C) завершением каждой из сторон.

Задание.. На основе предоставленных программ напишите простейший чат:

– пользователь вводит сообщения в клиентской программе и нажимает Enter для отправки;
– сервер рассылает полученное от одного клиента сообщение всем остальным подключённым клиентам;
– клиент, получив сообщение, немедленно выводит его на экран.

И клиент, и сервер должны быть однопоточными программами и использовать **select()**. Подсказка: у потока стандартного ввода с консоли тоже есть файловый дескриптор.

Код обоих программ включите в отчёт.

7.6 Контрольные вопросы

Для самоконтроля полученных знаний рекомендуется ответить на следующие вопросы.

1) Сколько байт занимает основной заголовок TCP-сегмента?
2) Как обеспечивается надежность передачи данных в протоколе TCP?

3) Почему в заголовке TCP-сегмента отсутствует такое поле, как размер сегмента? Откуда ещё можно получить эту информацию?

4) Какие комбинации флагов TCP могу быть нормальными, а какие — заведомо неверными?

5) Сколько существует различных вариантов разрыва TCP-соединения?

6) Как выглядит типичная последовательность вызова функций Socket API при работе с TCP в программе-сервере? В клиенте?

7) Что такое backlog-очередь? На что влияет её размер?

8) Какими способами можно организовать обслуживание клиентов сервером?

9) В чём преимущества асинхронной работы с сокетами перед схемой «один клиент — один поток»?

8 Механизмы протокола TCP

Протокол TCP содержит ряд механизмов для обеспечения надежной и эффективной доставки данных по сети. Эти механизмы можно условно разделить на основные, присутствующие в нём изначально и образующие костяк его функциональности, и дополнительные, добавленные во время его развития для решения каких-либо обнаруженных проблем или улучшения проиводительности. Число дополнительных механизмов росло со временем и им посвящены отдельные стандарты RFC, такие как [28], [29], [30] и другие.

В предыдущей главе мы описали установку и разрыв TCP-соединения, а так же упрощённые конечные автоматы протокола. В этой главе мы изучим различные механизмы протокола TCP, предназначенные для повышения надёжности передачи информации и повышения эффективности использования каналов связи. К ним относятся установление и разрыв соединения, окна отправителя и получателя, определение величины MSS соединения, повторная передача и буферизация данных.

Прежде чем приступать к дальнейшему чтению и выполнению задания, нужно хотя бы поверхностно ознакомиться с описанием TCP в [25].

За детальной информацией о протоколе TCP следует обращаться либо к стандарту RFC 793 [4], либо к иллюстрированным пособиям [26], [31].

8.1 Борьба с фрагментацией пакетов

Сетевой протокол IP поддерживает фрагментацию пакетов, но на практике этого стараются избежать. Более того, по умолчанию многие операционные системы отправляют IP-пакеты с флагом, запрещающим их фрагментацию. Этот подход, называемый MTU Discovery, позволял бы узнать MTU всего маршрута как наименьшую величину MTU входящих в него сетей, если бы маршрутизаторы отправляли ICMP-сообщения с информацией о невозможности передавать далее не подлежащий фрагментации пакет. Отправитель уменьшал бы MTU маршрута до получателя, пока не удалось бы отправить IP-пакет без получения ICMP-пакета с сообщением об ошибке.

На практике этот механизм не работает, поскольку его разрешение на маршрутизаторах может легко использоваться для DDoS (во всяком случае, так считают параноики). В силу этого в случае использования протокола UDP разумно предполагать наименьший возможный MTU (576 байт) и отсылать пакеты длиной не более пол-килобайта, однако для протокола TCP это слишком неэффективно. Поэтому для определения величины MTU маршрута используется поле MSS в пакете TCP (максимальный размер поля данных, необязательное поле в options). Поскольку в ходе установки соединения происходит обмен как минимум двумя пакетами, то

это может легко использоваться для установки минимального MSS маршрута, если это поле будет корректно изменяться всеми маршрутизаторами, образующими маршрут. Таким образом, к моменту установки соединения обе стороны будет иметь информацию о его MSS. Эта опция включается на маршрутизаторах и называется MSS clamping. Фактически она является «костылём» — как мы увидим позднее, MSS определяется буквально «не в ту сторону».

8.2 Механизм окна

Протокол TCP использует механизм скользящего окна, представленный на рис. 8.1. В [32] можно посмотреть на теоретическое описание алгоритма (не слишком относящееся к TCP), а в [33] — на демонстрационный ролик. Отметим, что на демонстрации [33] размер окна в пакетах, а на самом деле он в байтах и может быть не кратен величине MSS, даже если её превышает. Величина размера окна в заголовке TCP-пакета сообщает второй стороне, сколько еще байт другая сторона готова принять на данный момент. Это значение может достигать нуля, когда приложение не принимает данные, а буфер ядра, связанный с соединением, заполнился.

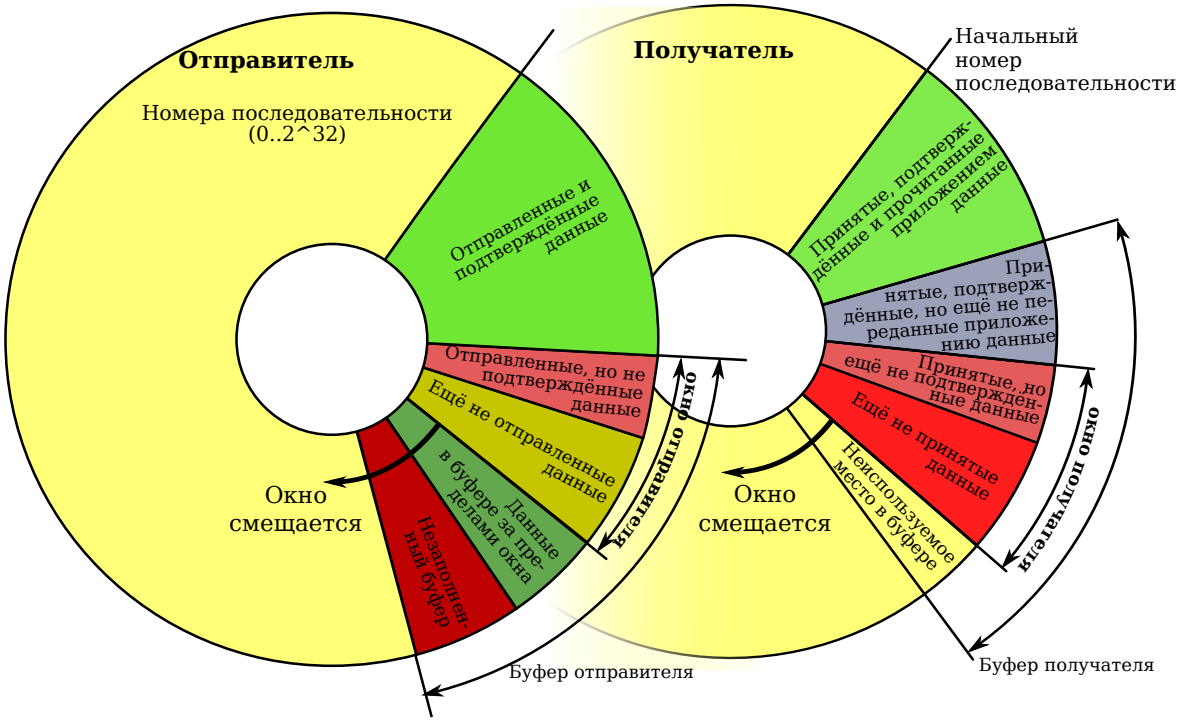


Рисунок 8.1 — Скользящее окно TCP

Получатель отсылает пакет с подтверждением в ответ на все полученные данные. Номер подтверждения соответствует номеру последнего непрерывного участка данных, увеличенного на единицу. Сегменты, образующие «раздробленные» участки, хранятся в буфере соединения.

Предположим, что TCP-отправителю надо переслать с помощью скользящего окна последовательность байт (пронумерованных от 1 до 3000) получателю с размером окна 1000 и с величиной MSS, равной 500[1].

Отправитель помещает в два пакета первые 1000 байт, передает их и ждёт подтверждения некоторое время. После того, как отправитель получит пакет с установленным флагом ACK и номером подтверждения, равным 501, то он передаст следующий пакет с байтами с 1001 по 1500 (предполагается, что величина размера окна не меняется), а при получении подтверждения с номером 1001 он отправит и четвёртый сегмент. Таким образом, алгоритм окна позволяет отправлять несколько пакетов, не ожидая подтверждения первых из них.

Пусть при приёме второго пакета получатель отправляет ACK с номером подтверждения равным 2001, и указывает в заголовке размер окна, равный нулю (например из-за того, что клиентское приложение не считало данные). В этом случае отправитель будет ожидать пакета ACK с номером подтверждения 2001 и ненулевым размером окна перед началом передачи следующей порции байт.

Размер таймера повтора отсылки (англ. *Retransmit timer*) в протоколе TCP выбирается достаточно сложным образом. Этот таймер зависит от значения RTT (англ. *Round-trip time*), которое в свою очередь рассчитывается на основе данных о времени задержки подтверждения сегмента. Однако, таймер отсылки по стандарту не может быть меньше 200 мс — видимо, чтобы не повторять данные слишком быстро даже при малых RTT. Кроме того, если причиной потери пакета является перегрузка некоторого маршрутизатора, то повторять пакеты слишком рано нет смысла.

Таймер повтора отсылки используется для обнаружения потери данных. Если подтверждение отправленных данных не поступает за время работы этого таймера, то отправитель повторяет потерянный сегмент и все последующие (если размер окна это позволяет, конечно же).

Следует отметить, что протокол TCP параллельно передаёт данные в обе стороны, поэтому приведённый пример является сильным упрощением.

8.3 Буферизация и алгоритм Нейгла

Описанный выше механизм не запрещает пересылать данные от приложения в сеть *сразу при их поступлении*, если размер окна это позволяет — то есть, в окне отправителя есть «еще не отправленные данные», а в окне получателя «еще не принятые» (см. рис. 8.1). Приложение, злонамеренно или по неусмотрительности пишущее в TCP-сокет данные по одному байту, может таким образом породить поток IP-пакетов, в которых

[1]Такого MSS быть не может, но допустим это для примера. Кроме того, мы предполагаем, что дополнительные опции в заголовке TCP

на 40 байт заголовков IP и TCP приходится 1 байт данных. Очевидно, что это приводит к падению эффективности передачи. «Защититься» от таких приложений можно, реализовав буферизацию на уровне TCP.

Алгоритм Нейгла. Для увеличения эффективности передачи в протоколе TCP по умолчанию используется алгоритм Нейгла [28, 26]. Этот алгоритм не является простой буферизацией: если сторона уже получила подтверждение приёма всех отправленных данных, то новые данные отправляются сразу, а иначе они накапливаются, пока их размер не достигнет MSS (в случае размера окна, превышающего величину MSS).

Согласно алгоритму Нейгла, следующие проверки выполняются каждый раз при поступлении данных от приложения:

```
Если размер окна == 0
    задержать данные в буфере
Иначе
    Если размер накопленных данных >= MSS и окно >= MSS
        послать сегмент размером в MSS байт
    Иначе
        Если все отправленные до этого данные были подтверждены
            послать сегмент с накопленными данными
        Иначе
            задержать данные в буфере
```

Задержанные данные могут быть впоследствии отправлены, когда

1) придет подтверждение на отправленные ранее данные, или
2) в буфере накопится достаточно данных для отправки полного сегмента (размером в MSS).

В ядре Linux реализован также вариант «агрессивной буферизации» (опция **TCP_CORK**). В этом случае присутствует тайм-аут буферизации, который зависит от величины RTT и составляет минимум 200 мсек. В отличие от алгоритма Нейгла, здесь получение подтверждения не приводит к ускорению отсылки данных, то есть мы имеем дело с простой буферизацией.

Логика разработчиков, в соответствии с которой тайм-ауты буферизации растут вместе с RTT, понятна: при высокой задержке в сети обеспечить интерактивность всё равно невозможно, и лучше попытаться поднять эффективность передачи данных с помощью буферизации.

Дополнение Миншаля к Нейглу. Как оказалось, алгоритм Нейгла в чистом виде неадекватно взаимодействует с другим элементом протокола TCP под названием «задержанное подтверждение» (англ. *delayed ACK*, [29]), о сути которого можно догадаться по названию: принимающая сторона отправляет подтверждения на полученные данные не сразу, а с задержкой, в надежде на то, что сможет подтвердить несколько сегментов

одним пакетом (в идеале — пакетом с данными, идущими в обратную сторону). По сути, это тоже буферизация, только на стороне получателя.

Для части прикладных задач, таких как передача больших файлов в одном направлении (протокол FTP), буферизация не представляет каких-либо проблем и даже является желательной. Страдают прежде всего протоколы класса «вопрос–ответ» (HTTP, SMTP и прочие).

Рассмотрим типичную ситуацию: приложение-клиент выдает в сокет запрос и ждет ответа сервера. Размер запроса, в общем случае, не кратен MSS, и поэтому последний сегмент является неполным. В соответствии с алгоритмом Нейгла, этот сегмент высылается только после подтверждения предыдущих отправленных; согласно политике «задержанного подтверждения», подтверждение на сегменты с запросом также отправляется с задержкой. Далее ситуация зеркально повторяется, когда приложение-сервер выдает ответ (тоже не кратный MSS по длине, вообще говоря). В итоге в простейшем обмене парой сообщений мы получаем неприятный эффект в виде сложения нескольких задержек, во время которых простаивает как сеть, так и оба приложения.

Грэг Миншаль предложил[34, 35] внести поправку к алгоритму Нейгла и проверять только подтверждения на неполные сегменты[1]:

```
Если размер окна == 0
    задержать данные в буфере
Иначе
    Если размер накопленных данных >= MSS и окно >= MSS
        послать сегмент размером в MSS байт
    Иначе
        Если все отправленные до этого неполные сегменты были подтверждены
            послать сегмент с накопленными данными
        Иначе
            задержать данные в буфере
```

Такая поправка позволяет сразу после полного сегмента размером в MSS байт отправить еще один неполный, не дожидаясь подтверждения. Легко убедиться, что это позволит устранить лишнюю задержку из ситуации «вопрос–ответ».

В ядре Linux реализован именно такой вариант алгоритма Нейгла.

Отключение буферизации. Приложения могут полностью отключить алгоритм Нейгла опцией **TCP_NODELAY**. Но и в этом случае можно будет наблюдать буферизацию отправляемых данных при заполнении окна отправителя.

[1] TCP, разумеется, подтверждает байты, а не сегменты. Но концептуально проще выразить суть поправки в терминах сегментов. Реализация в терминах байтов описана в [34].

8.4 Ранний повтор

Для улучшения работы протокола TCP в дальнейшем был создан ряд дополнительных механизмов. Мы рассмотрим один[1] из них — ранний повтор (англ. *fast retransmit*) [30].

Механизм повтора TCP не очень хорошо работает в случае разовой потери отдельного сегмента Поступающие отправителю одинаковые номера подтверждений наводят на мысль о потери пакета, однако TCP будет ждать срабатывания таймера, при этом вероятно исчерпание окна получателя и простой соединения по этой причине. Для решения указанной проблемы был предложен **быстрый повтор** — ..

Ранний повтор работает следующим образом. Допустим, отправитель получает три подтверждения приёма с одним и тем же номером последовательности, полученные после отправки пакета с этим же номером (таким образом, всего отправитель получил четыре подтверждения с этим номером, одно из которых пришло до его отправки). Тогда он может резонно предположить, что сегмент с данным номером потерян, а три последующих сегмента — дошли нормально. Поэтому отправитель повторяет предположительно потерянный сегмент сразу же, до того, как сработает таймер RTT. Этот алгоритм работает, когда в окно «влезает» минимум четыре сегмента.

8.5 Проблема корректного завершения соединения

Когда приложение закрывает соединение вызовом **close()** в момент, когда в сокете ещё есть непрочитанные приложением данные, то в современных реализациях TCP/IP-стека вместо стандартного алгоритма закрытия сокета через FIN-пакет посылается пакет с флагом RST, сигнализирующий о ненормальном закрытии соединения.

Проблема заключается в том, что, согласно стандарту, сторона, получившая пакет RST, должна отбросить всё содержимое своего буфера, ещё не прочтённого приложением. Таким образом, одна сторона, неверно закрыв сокет, может тем самым вынудить другую сторону потерять часть принятых данных. К счастью, в реальной жизни приложение все равно получает несчитанные данные, даже после получения пакета с флагом RST — видимо, эта проблема известна разработчикам стека TCP/IP, и он реализован не в полном соответствии с RFC. Однако, определённый риск все равно остаётся: например, часть пакетов (теоретически) может придти после RST, и тогда они точно потеряются.

[1] Рассмотреть медленный старт и быстрое восстановление не удастся из-за проблем с программой **tc** в эмуляторе Netkit.

8.6 Утилита screen

В некоторых случаях хотелось бы запустить несколько экземпляров командного интерпретатора **bash** в одном терминале. Это легко сделать при помощи программы **screen**. При запуске ей можно указать число запоминаемых строчек вывода.

```
screen -h 2000
```

Главные команды:

Ctrl+A D — отключение от screen (сессия остается запущенной);
Ctrl+A [— переход в режим прокрутки (выход: Esc). Базовые клавиши перемещения: h, j, k, l.

Для входа в запущенную сессию **screen** выполните следующую команду.

```
screen -r <a тут можно нажать TAB>
```

8.7 Создание и запуск виртуальных машин

Приступая к выполнению работы, необходимо загрузить и распаковать файлы виртуальной сети **lab-tcp** и затем запустить виртуальные машины командой **ltabstart**, как показано ниже.

```
mkdir -p ~/tcp-ip; cd ~/tcp-ip
wget -r http://ftp.iu7.bmstu.ru/nets/lab-tcp.tar.gz
rm -rf lab-tcp
tar -xvf lab-tcp.tar.gz
cd lab-tcp
ltabstart -d net
```

На машинах **c1** и **c4** при первом запуске будут автоматически скомпилированы необходимые для опытов вспомогательные программы.

На рисунке 8.2 показана конфигурация сети. Машина с адресом 10.40.0.2 используется для демонстрации алгоритма Нейгла, её линия связи имеет искусственную задержку, см. файл **/etc/delay**. Задержку можно менять через параметр этого сценария, например следующим образом.

```
/etc/delay 100
```

8.8 Процесс установки и разрыва TCP-соединения и определение величины MSS

Мы будем устанавливать соединение клиента на машине **c1** с сервером на машине **c6**. На участке между ними есть сеть 10.20.0.0/16 с величиной MTU, равной 1492 байтам, и сеть 10.30.0.0/16, для которой величина MTU равно 576 байтам.

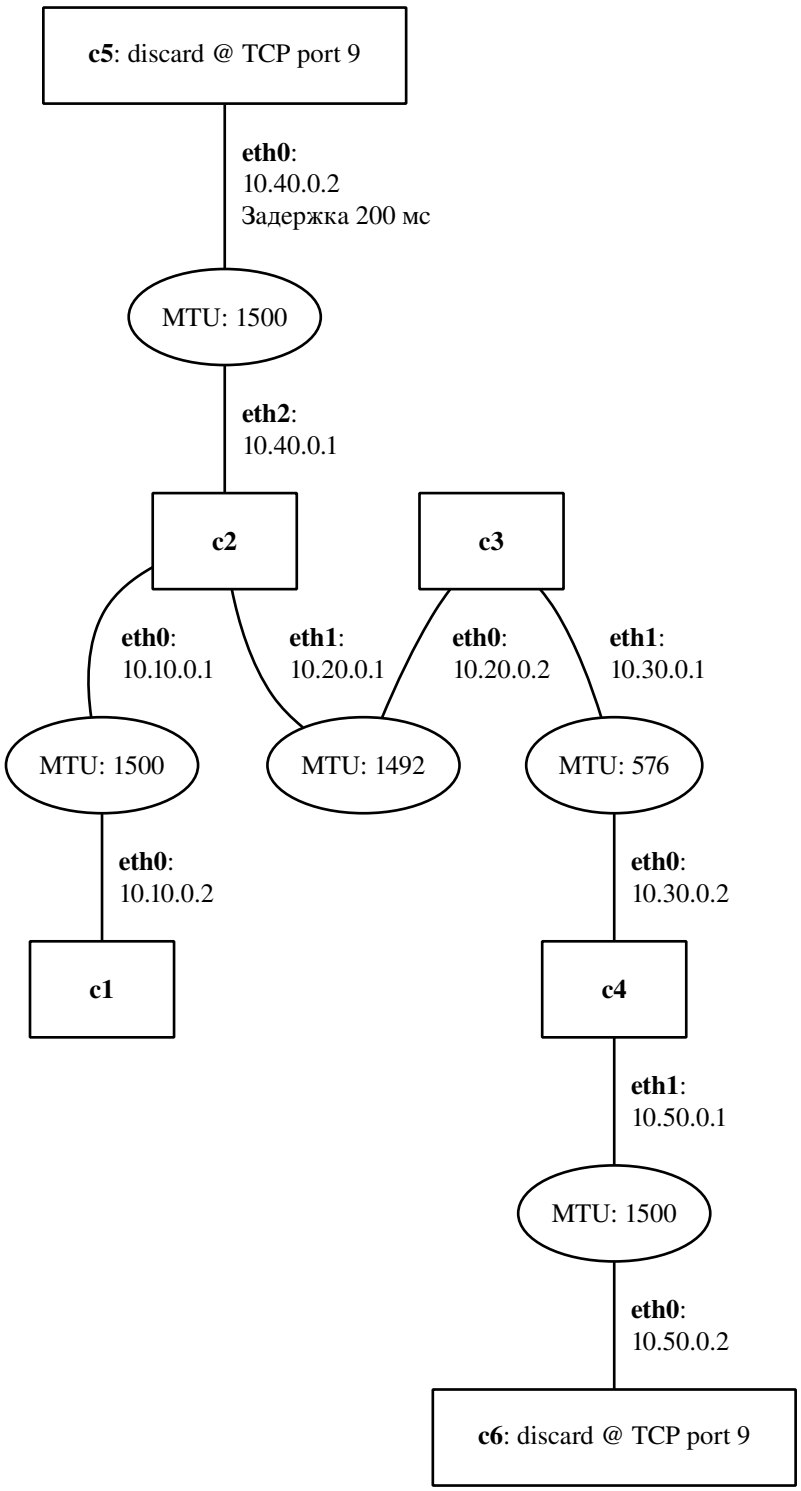


Рисунок 8.2 — Сеть передачи данных

В качестве сервера будем использовать сервер **discard** на 9-м порту машины **c6**. Этот сервер, являясь своеобразным эквивалентом **/dev/null**, выбрасывает все приходящие данные.

Для просмотра пакетов запустите программу **tcpdump** на машине **c2** (интерфейс **eth0**) и на машине **c3** (интерфейс **eth0**).

```
tcpdump -n -i eth0 tcp
```

Соединитесь с сервером следующей командой:

```
telnet 10.50.0.2 9
```

Напишите пару строчек, затем разорвите соединение (Ctrl+], q).

Прокомментируйте вывод **tcpdump**. Обратите внимание на величину поля MSS и номера полей Seq и Ack при установке соединения.

Отметим, что механизм MSS clamping включается в сценарии **/etc/firewall** на маршрутизаторах **c3** и **c4**.

8.9 Основы механизма окна

Для изучения основ механизма окна можно использовать программу **win**, запустив её на машине **c4**.

```
./win 3002 2 10
```

Здесь 3002 — номер порта, который она будет слушать, 2 — величина задержки в секундах, 10 — число принимаемых (после паузы) пакетов. Алгоритм работы программы следует изучить по её исходникам (**lab-tcp/c4/root/win.c**).

Затем запустим на машине **c3** программу **tcpdump** (или воспользуемся программой **screen**). Теперь запустим клиента на машине **c1**.

```
netcat 10.30.0.2 3002 </dev/zero
```

Понаблюдаем за изменением размера окна.

Теперь проведём такой же эксперимент на **c4**, запустив сервер **win** внутри программы **screen** или же в фоновом режиме (используя знак амперсанда).

```
./win 3002 2 10 &
```

Затем воспользуемся **screen** для запуска программы **tcpdump** на интерфейсе **lo**. После завершения подготовки запустим клиента на всё той же машине **c4**.

```
netcat localhost 3002 </dev/zero
```

8.10 Окно отправителя

На машине **c5** установите задержку канала 100мс.

```
/etc/delay 100
```

На машине **c2** запустите перехват пакетов.

```
tcpdump -n -i eth0 tcp
```

На машине **c1** проверьте командой **ping**, что время ответа узла 10.40.0.2 действительно составляет 100мс. Затем выполните на ней по одной следующие следующие команды.

```
./nagle nodelay 10.40.0.2 9 1 20
./nagle nodelay 10.40.0.2 9 1 50
./nagle nodelay 10.40.0.2 9 1 100
```

Окно отправителя в любой момент времени можно оценить, вычисляя разницу между полученным подтверждением (переведенным в номер сегмента) и номером отправляемого сегмента. (Номеров сегментов у нас формально нет, но их можно просто подсчитать по отправке). При расчёте, конечно, предполагается, что данных для отсылки у нас много, чтобы не принять за механизм окна отправителя следствия работы изложенного ниже алгортима Нейгла, например.

После выполнения каждой команды найдите по логам на машине **c2** определите, с какого значения начинается, как увеличивается и до какого максимального размера доходит окно отправителя (напоминаем, что оно измеряется в неподтверждённых сегментах). Попытайтесь объяснить наблюдаемый рост окна.

8.11 Алгоритм Нейгла и буферизация

Для изучения работы алгоритма Нейгла на машине **c1** есть программа **nagle**. Она отправляет в сокет буфер размером в 100 байт с заданной задержкой и использует один из трёх режимов отправки: с алгоритмом Нейгла, без него, с агрессивной буферизацией.

Первый её параметр (**cork|nodelay|default**) указывает на опции TCP_CORK («агрессивный» Нейгл, без отсылки по получению ACK) и TCP_NODELAY (отключение Нейгла) [36]. По адресу 10.40.0.2 (машина c5) на порту 9 имеется TCP-сервер (discard), причём линия связи до неё искусственно замедленна.

Запустите **tcpdump** на **c2**.

```
tcpdump -n -i eth0 tcp
```

Изучите исходный текст утилиты (**nagle.c**). Возможные варианты запуска этой утилиты приведены ниже. Изучите значения задержки 0 и -1.

1) Почти без задержки, алгоритм Нейгла работает (он включен по умолчанию).

```
./nagle default 10.40.0.2 9 0 30
```

2) Маленькая задержка, алгоритм Нейгла работает частично (если он совсем не работает — подберите сами задержку сами экспериментально).

```
./nagle default 10.40.0.2 9 3 30
```

3) Буферизация (**TCP_CORK**) работает частично, успевает накопиться часть данных. Установите, при каком значении задержки буферизация исчезает.

```
./nagle cork 10.40.0.2 9 400 30
```

4) Алгоритм Нейгла отключён, данные обычно отсылаются сразу (но иногда всё-таки буферизируются при заполнении cwnd).

```
./nagle nodelay 10.40.0.2 9 0 30
```

5) Если процесс не будет блокироваться (на функции **usleep**), то данные буферизуются и при параметре **TCP_NODELAY**.

```
./nagle nodelay 10.40.0.2 9 -1 30
```

Проведите аналогичные эксперименты и с сервером на узле **c4** и клиентом на том же узле **c1**. В этом случае у вас будет близкая к нулю задержка в сети, и вы наверняка увидите несколько отличающуюся картину с точки зрения значений тайм-аутов.

8.12 Повтор сегментов и ранний повтор

Для знакомства с механизмом раннего повтора надо запустить на машине **c3** сценарий, по-умолчанию отбрасывающий каждый 10-ый пакет, а затем запустить перехват трафика командой **tcpdump**. Внимание: сценарий **/etc/drop** лучше перезапускать перед каждым экспериментом, поскольку он выбрасывает не каждый 10-ый пакет отдельного соединения, а вообще каждый 10-ый (нумерация начинается с первого). Вы можете сменить номер пакета: для эксперимента нам нужно потерять пакет с данными от клиента к серверу, а не его подтверждение. Ниже показана имитация потери 11-го пакета.

```
/etc/drop 11
tcpdump -n -i eth0 tcp
```

Для отключения потери пакетов достаточно запустить **/etc/firewall** на машине **c3** (если потеря не нужна — так и сделайте).

В качестве сервера используем тот же сервер на машине **c4** и 9-ом порту.

На машине **c1** запустим **tcpdump** (при помощи **screen**), а затем клиента, например **nagle**.

```
./nagle nodelay 10.30.0.2 9 1 30
```

Наблюдая вывод **tcpdump**, понаблюдайте за действием раннего повтора. Убедитесь, что теряется именно пакет с данными, а не пакет подтверждения от **c4**. Если это не так, измените номер выбрасываемого пакета в сценарии **/etc/drop** и перезапустите его. Не забудьте перезапустить сценарий отбрасывания пакетов при повторе эксперимента.

В ядре Linux невозможно отключить быстрый повтор, поэтому для демонстрации «медленного повтора» можно пойти одним из двух путей.

Первый путь — терять один из последних передаваемых пакетов, чтобы не получить в ответах четыре одинаковых подтверждения. Второй — уже в ходе передачи данных включать большую задержку на промежуточном узле, **c3** следующим образом.

```
/etc/delay 300
```

Задержка должна несколько превышать значение таймера повтора.

Измените запуск команды **nagle** на машине **c1** соответствующим образом (или поменяйте сценарий **drop**).

Определите по выводу **tcpdump** значение таймера повтора в данном случае. Не забывайте, что значение таймера следует выяснить именно по логу на машине **c1**. Если вам неудобно работать с утилитой **screen**, то перенаправьте вывод утилиты **tcpdump** в файл. После эксперимента отключите потерю пакетов.

8.13 Использование механизма PMTU

В нашей системе на пути от **c1** до **c6** величина MTU падает дважды. Именно на этом пути и следует проверить, как работает механизм PMTU в случае передачи данных по TCP.

К этому опыту нужно немного подготовится. Во-первых, нужно отключить коррекцию величины MSS на маршрутизаторах **c3** и **c4**. Для этого на них следует сбросить эти правила следующей командой.

```
iptables -F
```

Во-вторых, следует убедится, что на машине **c1** не была уже каклибо определена величина MSS (или MTU) пути до машины **c6**. Для этого на первой машине можно выполнить команду **ip r show cache** — в выводе не должны быть адреса второй упомянутой машины.

Теперь мы можем соединится с машины **c1** с discard-сервером на машине **c6**, как это делали выше. При перехвате трафика на машине **c2** (интерфейс **eth0**) и **c6** мы увидим, что величина MSS при установке соединения осталась равна 1500 байтам.

Для дальнейшего опыта нам нужно отправить сегменты с полностью заполненным MSS. Это можно сделать, используя интенсивную буферизацию посылаемых данных (опация **cork** программы **nagle**). Убедитесь, что машина **c1** дважды получит извещения о невозможности передачи пакета без фрагментации. Выясните, как протокол TCP начинает себя в этом случае. Отметим, что при перехвате трафика нам надо будет захватить сообщения протоколов TCP и ICMP. Для этого в конце команды запуска программы Tcpdump следует указать перехват их обоих: **tcp or icmp**.

Для того, чтобы включить механизм изменения MSS обратно, можно выпонить следующую команду на тех маршрутизаторах, где он был от-

ключен в начале этого раздела (содержимое запускаемого командного файла при желании можно изучить).

```
/etc/firewall
```

8.14 Контрольные вопросы

Для самоконтроля полученных знаний рекомендуется ответить на следующие вопросы.

1) Чему будет равно значение MSS если на маршрутизаторах включён механизм расчёта MSS? А если не включен?

2) Чему равно поле **seq** в первом SYN-пакете? Почему?

3) Является ли найденная при установлении TCP-соединения величина MSS полностью достоверной? В каких случаях она будет установлена неверно?

4) Какая опция при создании сокета включает алгоритм Нейгла?

5) Для чего служит алгоритм Нейгла? Какой цели служат дополнения Мишналя?

6) Какой в нашем случае примерный размер имеет TCP-буфера ядра для приёма данных? Зависит ли эта величина от интерфейса (локальный / Ethernet)?

7) Какое минимальное значение таймера повтора в TCP?

8) Что даёт механизм быстрого повтора и когда он срабатывает?

9) Что ограничивает отсылку данных в случае использования опции сокета **TCP_NODELAY**, если программа хочет послать сразу большой буфер (больше величины MTU на два, например, порядка)?

10) Что происходит, если окно получателя достигает нуля?

11) Какое минимальное значение окна отправителя нужно для срабатывания быстрого повтора (в случае потери одного сегмента)?

12) Что происходит после получения системой сообщения о невозможности послать IP-пакет дальше без его фрагментации в случае, когда этот пакет передавал TCP-сегмент?

9 Протокол HTTP

9.1 Контрольные вопросы

Для самоконтроля полученных знаний рекомендуется ответить на следующие вопросы.

10 Локальные компьютерные сети и фильтрация пакетов

В этой главе мы познакомимся с управления IP-трафиком: фильтрацией, трансляцией сетевых адресов, тунеллированием.Кроме того, мы увидим динамическую настройку IP-адресов машин в локальной сети.

В конце этой главы (раздел 10.15) приведены возможные индивидуальные варианты заданий. В разделе 10.14 даны дополнительные указания по их выполнению.

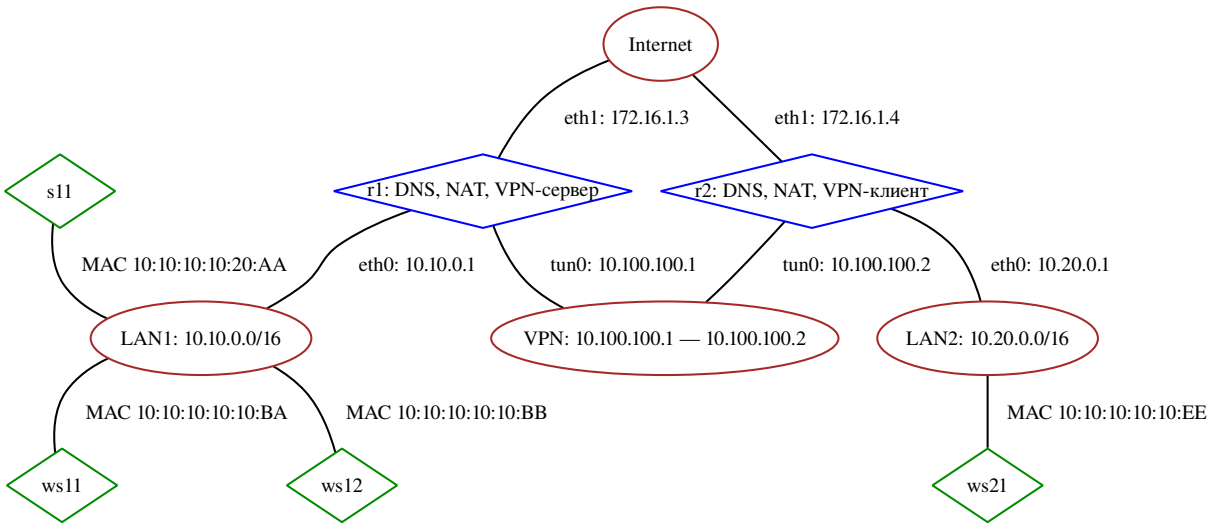


Рисунок 10.1 — Топология сети

Скачаем и запустим сеть, показанную на рисунке 10.1.

```
mkdir -p ~/tcp-ip; cd ~/tcp-ip
wget -r http://ftp.iu7.bmstu.ru/nets/lab-lan.tar.gz
rm -rf lab-lan
tar -xvf lab-lan.tar.gz
cd lab-lan
ltabstart -d net
```

Отметим, что загрузка всех машин, кроме **r1** и **r2**, «зависла» на попытках DHCP-клиента получить IP-адрес. Эта ситуация в нашем случае является нормальной.

Убедимся, что оба маршрутизатора успешно подключились к реальной сети. Если мы видим, что команда **ping bmstu.ru** работает нормально, то всё в порядке (для проверки можно отправить эхо-запросы и на любой другой внешний адрес).

10.1 Динамическая настройка IP-адресов

Для динамической настройки IP-адреса и всех его параметров может использоваться протокол DHCP [37]. Он позволяет выдавать компью-

терам в одном сегменте широковешания IP-адреса из некоторого указанного диапазона. Кроме адреса, клиент DHCP получает от сервера маску сети, адрес (или адреса) маршрутизаторов, адреса DNS-серверов. Протокол DHCP использует широковещательную рассылку поверх протокола UDP для обмена данными между клиентом и сервером. Поскольку у клиента ещё нет IP-адреса, в качестве него используется 0.0.0.0, а IP-адресом получателя при широковещательном обмене выступает 255.255.255.255.

Сервер может привязать выдаваемый IP-адрес к MAC-адресу клиента, а клиент — попросить у сервера некоторый конкретный IP (например, полученный в последний раз).

На машинах **r1** и **r2** нужно настроить DHCP-серверы. Кроме IP-адреса и маски сети, оба DHCP-сервера должны сообщать клиентам адрес маршрутизатора по умолчанию и адрес dns-сервера: в качестве первого выступает внутренней IP-адрес соответствующего маршрутизатора, а в качестве второго придётся взять адрес DNS кафедры (192.168.0.1) или ваш домашний (если вы делаете дома).

> **Задание № 42: Настройка DHCP-службы**
>
> На маршрутизаторе **r2** настроим службу DHCP: в файле **/etc/dhcp3/dhcpd.conf** следует указать адрес маршрутизатора по-умолчанию, DNS-сервера (для кафедры это 192.168.0.1) и адрес внутренней сети. Затем перезагрузим службу DHCP.
>
> ```
> service dhcp3-server restart
> ```
>
> На машине **ws21** отдадим команду **reboot**. После перезагрузки проверьте командой **ip a**, что она получила адрес.

DHCP-сервер на машине **r2** выдает адреса клиентам произвольным образом.

В отличие от него, DHCP-сервер на машине **r1** будет выдавать адреса с жесткой привязкой к MAC-адресу клиента. Это позволит выдавать всем машинам сети фиксированные IP-адреса централизованным образом. Мы затем сможем использовать эти адреса при фильтрации сетевого трафика.

> **Задание № 43: Привязка IP-адреса к MAC-адресу клиента**
>
> В настройках службы DHCP на r1 присвойте адреса машин в соответствии с рисунком 10.1 и перезагрузите службу. Убедитесь, что все внутренние машины в сети получили после их перезагрузки ожидаемые IP-адреса.

10.2 Трансляция сетевых адресов и портов

Трансляция сетевых адресов (NAT) представляет собой решение двух проблем [38]:

– недостатка сетевых адресов протокола IPv4 для всего мира[1];

– желания изолировать внутреннюю локальную сеть от внешнего мира.

Более точно данная трансляция называется трансляцией сетевых адресов и портов (NAPT), поскольку затрагивает не только адреса, но и порты протоколов TCP/UDP.

Трансляция представляет собой отображение пары «адрес во внутренней сети — порт» в пару «внешний адрес маршрутизатора — порт». Отображение задаётся маршрутизатором в момент, например, обнаружение исходящего из локальной сети соединения. Это преобразование так же называют «маскарадом» сетевых адресов.

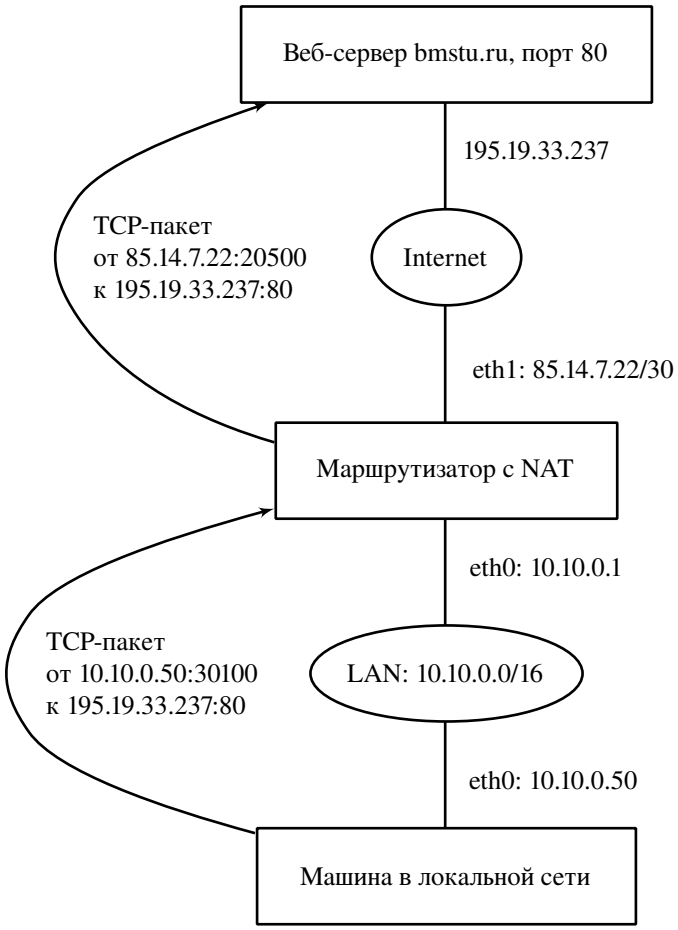


Рисунок 10.2 — Подключение внутренней сети к сети общего пользования с использованием NAT

[1]В протоколе IPv6 данная проблема отсутствует, однако пока что сети IPv4 очень широко используются

Рассмотрим трансляцию на примере сети, изображённой на рис. 10.2. При попытке соединиться по TCP с внешним веб-сервером с компьютера локальной сети с адресом 10.10.0.50 и динамически назначанным браузеру порта 30200 маршрутизатор сопоставит пару 10.10.0.50:30100 с каким-либо динамическим портом (например, 20500) на своём внешнем сетевом интерфейсе (например, 85.14.7.22). После этого пакеты будут отправляться в интернет с подменённым на 85.14.7.22:20500 адресом и портом источника, а пакеты, приходящие от веб-сервера на порт маршрутизатора 85.14.7.22:20500, будут перенаправляться на внутренний порт 10.10.0.50:30100.

В реальной жизни в основном используется трансляция с ограничением по портам: не все внешние пакеты транслируются вовнутрь, а только с того адреса и порта, на который до этого в течении недавнего времени высылались пакеты (в случае протокола UDP) или с которым установлено/устанавливается соединение (в случае протокола TCP).

Поскольку NAT использует порты протоколов транспортного уровня (TCP и UDP), то для функционирования поверх него протокола ICMP используется несколько другой подход: вместо номера порта выступает поле идентификатора пакета ICMP в его заголовке.

Трансляцию сетевых адресов для поддержки исходящих из сети соединений так же иногда называют SNAT (от англ. *source NAT*).

10.3 Переброска портов

Трансляция адресов естественным образом блокирует доступ извне ко всем компьютерам внутренней сети, что выглядит преимуществом в случае домашней иди офисной сети. Однако, часто нужно организовать доступ в внутреннему серверу при попытке соединения из-вне. Переброска портов (англ. *port forwarding*, [39]) позволяет обеспечить доступ к внутреннему серверу с «серым» ip-адресом [1] из интернет. Типичные её применения: доступ к почтовому или веб-серверу в офисе, доступ к p2p-серверу дома. Поскольку решение о переброске входящих пакетов принимается на основании адреса порта назначения, оно называется DNAT (от англ. *Destination NAT*).

Переброска портов является частным случаем NAT, когда соотношение внешних и внутренних портов и адресов прописывается заранее и оно не является ограниченным: обычно все входящие пакеты на внешний интерфейс и порт (например, 80) перенаправляются на сервер (например, 10.10.0.20:80). На практике часто перенаправляются не все пакеты, поскольку внутренний сервер может подвергаться атакам отказа обслуживания.

[1] Серые или частные адреса: 10.0.0.0/8, 192.168.0.0/16. 172.16.0.0/12

Стоит отметить, что современные «домашние» маршрутизаторы поддерживают протокол UPnP, благодаря которому многие программы, работающие на компьютерах во внутренней сети, могут сообщать маршрутизатору желаемые номера пробрасываемых портов. Это избавляет домашнего пользователя от ручной настройки своего маршрутизатора с NAT.

Трансляцию сетевых адресов для поддержки перебрасываемых на внутренний сервер входящих соединений так же иногда называют DNAT (от англ. *destination NAT*).

10.4 Фильтрация сетевого трафика

Фильтрация сетевого трафика — это набор правил, которые принимает одно их решений «пропустить пакет» или « выбросить пакет» для каждого IP-пакета. При этом принимаются во внимание следующие параметры пакета:

– протокол, пакет которого вложен в данный IP-пакет (TCP, UDP, ICMP, etc);
– адрес и порт отправителя;
– адрес и порт получателя;
– входящий и исходящий сетевой интерфейсы;
– тип пакета: транзитный, входящий или исходящий;
– состояние TCP-соединения;
– и другие.

На практике при фильтрации трафика в офисе хотелось бы привязываться не к адресу машины, а к учётной записи пользователя, с правами которого выполняется работающая с сетью программа. Поскольку это выходит за рамки TCP/IP, в данной работе расматривается фильтрация на основе IP-адреса рабочей станции.

Маршрутизатор в первой локальной сети будет выдавать адреса по MAC-адресу, поскольку организует фильтрацию трафика по IP-адресам. Для фильтрации адреса внутренних компьютеров можно использовать подсети, например такие:

– первая (например, 10.10.10.0/24) — внутренние серверы, неограниченный доступ во внешюю сеть;
– вторая (например, 10.10.1.0/24) — слабо ограниченные пользователи;
– третья (например, 10.10.2.0/24) — сильно ограченные пользователи;
– остальные адреса считаются полностью ограничеными и имеют доступ только в пределах локальной сети организации.

Если DHCP-сервер выдаёт адреса в соответствии с такими требованиями, то в разрешающих правилах межсетевого экрана можно использовать указаные подсети.

10.5 Межсетевые экраны в ОС GNU/Linux

В ядро Linux включён межсетевой экран Netfilter, являющийся частью ядра. Для управления его правилами существует программа **iptables** [40].

Нам понадобятся изменить две таблицы системы обработки IP-пакетов в ядре:

– таблица фильтрации (filter) содержит правила фильтрации пакетов, и предполагается по-умолчанию, при отсутстви параметра **-t** при запуске программы **iptables**;

– таблица трансляции (nat) содержит правила трансляции портов и адресов, для её изменения нужно явно указывать параметр **-t nat** при запуске **iptables**.

В таблице фильтрации могут быть правила для каждого из трёх видов пакетов: исходящих с данного компьютера (OUTPUT), входящих в него (INPUT), и транзитных, маршрутизируемых данным компьютером (FORWARD). В работе с целью упрощения рассматриваются только последние. Поэтому следует отметить, что это не работа по сетевому администрированию.

В таблице трансляции нам нужно будет изменить цепочки правил, использующиеся при приходе маршрутизируемого пакета (PREROUTING) и перед отправлением маршрутизируемого пакета дальше (POSTROUTING).

10.6 Создание безопасного IP-туннеля

При соединение двух локальных сетей через интернет необходимо создать виртуальную частную сеть [41]. Для этого следует создать IP-туннель с шифрованием трафика между двумя компьютерами в данных сетях (обычно это их маршрутизаторы). На каждом из маршрутизаторов заводится виртуальный[1] сетевой интерфейс с адресом в частной сети. Данные, передаваемые через этот интерфейс, зашифровываются и инкапсулируются в TCP или UDP пакеты, передаваемые через публичную сеть. Часто в качестве транспорта используется UDP, поскольку инкапсулирование TCP в TCP может показаться избыточным. С другой стороны, UDP-датаграммы имеют меньший, чем TCP-пакеты, размер, что уменьшает эффективность передачи данных.

[1] Здесь предполагается, что сами маршрутизаторы вполне реальны.

Для обеспечения безопасности передаваемых по туннелю данных может использоваться криптографический протокол SSL [42], использующий шифрование на основе пары из закрытого и открытого ключей на начальном этапе и симметричное шифрование по временному ключу при передаче данных, или простой симметричный ключ, разделяемый двумя маршрутизаторами, между которыми создаётся *туннель* (виртуальный сегмент, соединяющий ровно две машины). Мы будем использовать второй вариант при всех его очевидных недостатках, поскольку он позволит (не спрашивайте, почему) использовать уже знакомый нам протокол RIP для обмена маршрутами поверх VPN.

Мы воспользуемся следующими ранее не рассмотренными программами:

– программное обеспечение для фильтрации сетевого трафика (программа управления: **iptables**, сама фильтрация работает в ядре ОС);
– служебная программа **dig** для опроса DNS-серверов;
– служба для динамической настройки параметров IP с использованием протокола DHCP;
– служба для создания безопасного IP-туннеля (OpenVPN).

Всего в работе могут быть запущены следующие машины (рисунок 10.1):

– **r1** и **r2**: маршрутизаторы первой и второй локальной сети соответственно;
– ws11, ws12 и ws21: две рабочие станции в первой локальной сети и одна во второй;
– s11 — сервер в первой локальной сети.

Маршрутизаторы имеют два сетевых интерфейса: **eth0** подключён в локальную сеть маршрутизатора, **eth1** — подключён к интернет через ваш компьютер, поэтому на **eth1** используется «серый» адрес). На маршрутизаторах в файле **/etc/resolv.conf** указан используемый DNS-сервер.

На всех внутренних компьютерах настроен сетевой интерфейс с DCHP, но DHCP-сервер на маршрутизаторах ещё не настроен полностью. У внутренних компьютеров с помощью файла настроек **/etc/network/interfaces** уже заданы фиксированные MAC-адреса (см. рисунок 10.1).

На маршрутизаторах настроено соединения с интернетом (интерфейс **eth1**). Внутренний интерфейс **eth0** также настроен. Команды **ip a** и **ip r** позволят узнать настройки интерфейса **eth1**.

При первоначальном запуске всех машин одновременно может случиться так, что внутренние компьютеры локальных сетей запустятся раньше маршрутизаторов, и не дождутся получения адреса по протоколу DHCP. Это можно будет исправить, перезапустив на них настройки сети

(**service networking restart**) при уже настроенном DHCP или перегрузив их командой **reboot**.

10.7 Настройка сетевого интерфейса и DHCP-сервера на маршрутизаторе локальной сети

В соответствии с заданием необходимо настроить внутренний сетевой интерфейс маршрутизаторов в файле **/etc/network/interfaces**. Интерфейс, подключённый к реальной сети, в этом файле не описывается: при запуске виртуальной машины в этой лабораторной автоматически выполняется команда **ip** для добавления внешнего адреса и маршрутизатора по-умолчанию. Пример файла сетевых настроек маршрутизатора приведен ниже.

```
auto eth0
# Статическое назначение адреса (static)
iface eth0 inet static
# IP-адрес:
address 10.10.0.1
# Маска сети:
netmask 255.255.0.0
```

После изменения этого файла следует перегрузить настройки для интерфейса **eth0**.

На внутренних компьютерах сетевые настройки не меняются, поскольку они уже настроены на получения адреса от DHCP-сервера. Следует помнить, что до настройки и запуска DHCP-сервера на маршрутизаторах внутренний компьютеры не получат правильный IP-адрес, поэтому их запуск лучше отложить.

Для включения на маршрутизаторе DHCP-сервера нужно:

– отредактировать файл конфигурации сервера **/etc/dhcp3/dhcpd.conf** (см. ниже);

– запустить службу DHCP: **service dhcp3-server restart**.

Файл конфигурации DHCP-сервера уже частично настроен. Пример заполненного файла конфигурации для маршрутизатора **r1**.

```
# Внутренняя сеть.
subnet 10.10.0.0 netmask 255.255.0.0
  range 10.10.0.2 10.10.10.200; # диапазон адресов  option routers 10.10.0.1; # ад
# Все IP-адреса станциям выделяются по MAC
# Имя хоста здесь сугубо мнемоническое.
host ws11     hardware ethernet 10:10:10:10:10:BA;    fixed-address 10.10.1.1;
host ws12     hardware ethernet 10:10:10:10:10:BB;    fixed-address 10.10.2.1;
host s11     hardware ethernet 10:10:10:10:20:AA;    fixed-address 10.10.10.1;
```

Адреса внутренним станциям следует назначать так: серверы (**s1** и **s2** в одной подсети с маской /24, остальные компьютеры — каждый в

своей подсети с маской /24). В пример выше серверы будут в подсети 10.10.10.0/24.

На втором маршрутизаторе (**r2**) делать привязку IP-адреса к MAC-адресу не нужно.

При использовании в реальных сетях на маршрутизаторе локальной сети обычно настраивается кеширующий DNS-репликатор, и в качестве DNS-сервера для станций указывается также адрес маршрутизатора. Если вы хотите сделать так же, то следует настроить службу **pnds-recursor** (см. далее), и указать внутренний адрес маршрутизатора в качестве выдаваемого сервера DNS в настройках DHCP-сервера.

10.8 Проверка работы DHCP в локальной сети

Сообщения об ошибках при запуске сервера DHCP добавляются в файл **/var/log/syslog**. В случае ошибок при запуске этой службы следует выполнить примерно такую команду.

```
tail /var/log/syslog
```

После настройки DHCP-серверов на обоих маршрутизаторах следует проверить настройки сети. Внутренние машины должны получить сетевой адрес от DHCP-сервера и иметь доступ друг к другу и к внутреннему интерфейсу маршрутизатора, используйте команду **ping** для проверки. Сам маршрутизатор должен иметь доступ в интернет. Доступа в интернет у внутренних компьютеров пока нет: созданная локальная сеть используют частные адреса, которые не маршрутизируются в интернете.

Для изучения работы протокола DHCP следует начать анализ пакетов на DHCP-сервере следующей командой.

```
tcpdump -tnv -s 200 -i eth0 udp
```

После чего можно перезагрузить компьютер клиента (команда **reboot**).

На сервере будут виден обмен командами DHCP между клиентом и сервером.

10.9 Создание виртуальной частной сети

После того, как мы настроили две локальные сети, следует настроить передачу данных между ними через защищённый от прослушки и изменения третьими лицами безопасный канал передачи данных.

Для создания виртуальной частной сети используется служба OpenVPN [43]. При её использовании один из маршрутизаторов (в наше примере — **r1**) выступает в роли сервера, ожидающего подключение, а второй — в роли клиента. Сервер управляет настройками VPN.

Виртуальная сеть в нашем случае огранизуется с помощью виртуальных сетевых интерфейсов, которые начинаются с префикса «tun» (от англ. *tunnel*): **tun0** и т.д.

Для настройки OpenVPN в виртуальной машние NetKit нужно:

– настроить конфигурацию openvpn (см. ниже), файл конфигурации должен иметь расширение **.conf**;

– запустить службу: **service openvpn restart**.

На первом маршрутизаторе следует создать файл **/etc/openvpn/tun0.conf** примерно следующего вида (конкретный вид файла зависит от задания).

```
# Внешний адрес:
local 172.16.1.3
# Используемый нижестоящий протокол и его порт.
proto  udp
port 1194
# Использовать туннель (возможен так же виртуальный мост)
dev tun
# Первый адрес - локальный, второй - удалённый.
ifconfig 10.100.100.1 10.100.100.2
# Ключ симметричного шифрования
secret /etc/openvpn/keys/somesecret.key
# Файл статуса сервера
status /var/log/openvpn/tun0.status
# Файл журнала
log /var/log/openvpn/tun0.log
```

Каталог **/etc/openvpn/keys/** с (не слишком то и секретным) ключом доступен только суперпользователю. Для того, чтобы каждый из маршрутизаторов узнал всё о топологии сети, на них развёрнута служба RIP.

На втором маршрутизаторе файл конфигурации сети отличается только отсутствием параметра **local**, вместо него должен присутствовать параметр **remote**, и «перевёрнутыми» адресами в тунеле.

```
# IP-адрес и порт OpenVPN-сервера
remote 172.16.1.3 1194
...
# Первый адрес - локальный, второй - удалённый.
ifconfig 10.100.100.2 10.100.100.1
```

Как можно заметить, наш VPN использует протокол UDP. Это решение имеет как плюсы (вариант «реальный TCP поверх виртуального TCP» явно может приводить к неясным проблемам из-за двойного TCP), так и минусы. Например, OpenVPN полностью полагается на работающий механизм PMTU для поиска MTU пути. Если PMTU не работает, то в конфигурации OpenVPN следует явно указать величину MSS в параметре **mssfix** (при использовании протокола UDP эа величина рассматривается как его

максимальная полезная нагрузка) или хотя бы добавить параметр **mtu-test** для проверки работоспособности PMTU при запуске клиента.

10.10 Проверка работы виртуальной частной сети

После запуска ПО OpenVPN создаст сетевой интерфейс **tun0**. Для проверки получения настроек клиентом нужно изучить вывод следующих команд.

```
ip a
ip r
```

В таблицу маршрутов должны добавится строчки для доступа к виртуальной частной сети и для сети за сервером OpenVPN. Просмотр файлов журналов даёт достаточно информации для поиска неисправностей и некорректных настроек.

Следует отметить, что в силу использования виртуальных машин здесь адреса внешних интерфейсов обоих маршрутизаторов находятся в одной сети. В реальных сетях, конечно же, они могут принадлежать различным ISP, что не мешает использованию OpenVPN.

10.11 Настройка межсетевого экрана

Межсетевой экран управляет IP-трафиком и будет выполнять следующие функции:

– трансляцию сетевых адресов для доступа внутренних компьютеров в интернет и доступа снаружи к серверам внутри сети;
– ограничение доступа рабочих компьютеров к ресурам внешней сети;

В данной учебной работе сценарий установки правил помещается в **/etc/firewall**, а его запуск прописывается в **/etc/rc.local**. Экран настраивается только на первом маршрутизаторе. Данный сценарий можно запускать многократно при тестировании, не перегружая систему. Напоминаем, что для запуска файл должен иметь соотвествующий аттрибут (**x**).

При использовании программы **iptables** в работе используются следующие параметры:

– сеть отправителя: **-s** (или **–source**);
– сеть получателя: **-d** (или **–destination**);
– порт отправителя: **–sport**, порт получателя: **–dport**;
– входящий интерфейс **-i**, исходящий интерфейс **-o**;
– протокол: **-p** или **–protocol**;
– отбрасывание пакета: **-j DROP**;
– прохождение пакета: **-j ACCEPT**;

– указание используемой таблицы правил (**-t**) и цепочки для добавления правила (**-A**).

Для просмотра цепочек полезны следующие команды. Первая из них показывает правила из таблицы **filter**, вторая — из таблицы **nat**.

```
iptables -L -nv
iptables -L -nv -t nat
```

Цепочка правил просматривается сверху вниз последовательно, пока не будет найдено правило, все условия которого удовлетворяются. По-умолчанию все исходящие (цепочка **OUTPUT**), входящие (цепочка **INPUT**) и транзитные (цепочка **FORWARD**) пакеты проходят через межсетевой экран.

В нижеследующем примере примере политика по-умолчанию для транзитных (цепочка FORWARD) пакетов меняется на их отбрасывание, после чего разрешаются некоторые виды пакетов.

```
#!/bin/sh
LAN=eth0
INET=eth1
VPN=tun0
# Удаление всех правил в таблице "filter" (по-умолчанию).
iptables -F
# Удаление правил в таблице "nat" (её надо указать явно).
iptables -F -t nat
# По-умолчанию все маршрутизируемые пакеты выбрасываются.
iptables --policy FORWARD DROP
# ICMP разрешим
iptables -A FORWARD -p icmp -j ACCEPT

# Разрешаем любую маршрутизацию для интерфейса VPN
iptables -A FORWARD  -i $VPN -j ACCEPT
iptables -A FORWARD  -o $VPN -j ACCEPT
# Включение SNAT для маршрутизируемых пакетов, выходящих
# через eth1. Это правило выполняется после самой маршрутизации
# (POSTROUTING) и помещается в таблицу правил "nat".
iptables -t nat -A POSTROUTING -o $INET -j MASQUERADE
# Разрешение пакетов-ответов (они отслеживаются как
# -- state ESTABLISHED)
iptables -A FORWARD -m state --state ESTABLISHED           -i $INET -j ACCEPT
# Осталось разрешить исходящие пакеты в соответствии с заданием.
```

Обычно в сценарий включают также команду для включения маршрутизации IPv4, но в ядре NetKit маршрутизация уже включёна.

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

Пример выше можно использовать за основу для создания межсетевого экрана в работе. Однако, сейчас он разрешает все исходящие из локальной сети соединения TCP. Это можно изменить двумя способами:

– добавить нужные правила запрещения (с действием **-j DROP**) выше разрешения всех маршрутизируемых пакетов, тогда они будут иметь более высокий приоритет; при необходимости до запрещающих правил можно разместить и некоторые разрешающие правила;

– убрать разрешение новых соединений из правила с **–state NEW,ESTABLISHED**, а затем разрешить все нужные соединения явным образом.

Пример правила переброски портов (DNAT) с внешнего интерфейса на внутренний сервер сети с адресом 10.10.10.1.

```
iptables -t nat -A PREROUTING -p tcp --dport 25  -i $INET
             -j DNAT --to 10.10.10.1:25
```

Как видно, перенаправление на внутренний адрес должно происходить до принятия решения о маршрутизации пакета. Обратите внимание: приведённая выше команда не включает разрешение каких-либо пакетов на этот сервер (или от этого сервера), это следует сделать отдельными командами.

В файл следует добавить команды разрешения трафика от компьютеров внутренней сети в соответствии с заданием. Примеры правил фильтрации исходящего трафика:

– разрешение всех пакетов для указанной внутренней подсети:

```
iptables -A FORWARD -s 10.x.x.x/24 -j ACCEPT
```

– разрешение для доступа к указанной удалённой сети:

```
iptables -A FORWARD -d 213.88.5.0/24 -j ACCEPT
```

– разрешение для локальной подсети доступа к порту 80 протола TCP на внешних серверах.

```
iptables -A FORWARD -s 10.10.5.x/24 -p tcp           --dport 80 -j ACCEPT
```

При реальном использовании HTTP-трафик обычно пропускается через локальный HTTP-прокси для реализации тонкой фильтрации.

Для разрешения протокола DNS можно использовать один из двух вариантов: либо настроить локальный кеширующий DNS-сервер (мы используем этот вариант в этой работа), либо разрешить обращение к внешнему серверу DNS (протокол UDP, порт 53) в сценарии фильтрации пакетов, как показано ниже. Во втором случае в настройках DHCP нужно указать некоторый доступный внешний кеширующий DNS-сервер нужно.

```
iptables -A FORWARD -p UDP --dport 53 -o $INET -j ACCEPT
```

Для выяснения адресов сетей типа ICQ можно использовать утилиту **dig**, многократно опрашивая сервер имён сервер (NS-сервер) домена примерно следующей командой (здесь опрашивается NS-сервер AOL для выяснения адресов поддомена aol.com).

```
$ dig @dns-02.ns.aol.com  login.messaging.aol.com
```

Сервер имен конкретного домена можно узнать, опросив командой **dig** DNS-сервер по-умолчанию командой **dig -t NS домейн**, подробности см. в [44]. DNS-сервер системы (указанный в /etc/resolv.conf) обычно является кеширующим, поэтому его опрос не рекомендуется.

После внесения изменений в файл правил межсетевого экрана его следует запустить.

```
/etc/firewall
```

10.12 Проверка межсетевого экрана

После корректной настройки файла с правилами фильтрации у внутренних компьютеров появится доступ во внешнюю сеть с заданными ограничениями. Используйте команду

```
$ telnet адрес порт
```

для проверки соединений по протоколу TCP (разрыв связи: Ctrl+]).

Для тестирования переброски портов на внутренний сервер нужно запустить на нём соответствующую заданию службу:

– веб-сервер Apache для проверки протокола HTTP, порт 80 (запуск: **service apache2 start**);
– SMTP-сервер Exim для проверки протокола SMTP, порт 25 (запуск: **service exim4 start**);

Следует отметить, что ваш набор правил может и не разрешать протокол ICMP или, наоборот, разрешать его всегда. Тогда использовать программу **ping** даже для тестирования доступа по адресу совершенно бесполезно.

Последнее замечание: во внутренней сети МГТУ гарантировано доступен единственный 25-ый порт на SMTP-сервере **mx.bmstu.ru**. Соединения с внешними SMTP-серверами через 25-ый порт блокируются.

10.13 Контрольные вопросы

Для самоконтроля полученных знаний рекомендуется ответить на следующие вопросы.

1) Какие задачи решает NAT?
2) Какие задачи решает VPN?
3) Что такое межсетевой экран?
4) Прокомментируйте вывод **ip r** на маршрутизаторах **r1** и **r2**.
5) Какая программа занимается самой фильтрацией IP-пакетов?
6) Используется ли здесь NAT для передачи данных из одной локальной сети в другую через VPN?
7) Прокомментируйте обмен сообщениями DHCP.

8) Используется ли при обмене по протоколу DHCP протокол ARP?

9) Когда происходит SNAT-преобразование адресов в исходящих из локальной сети, до или после принятия решения о его маршрутизации? Когда происходит DNAT-преобразование входящего в локальную пакета?

10) Изменяет ли SNAT пакеты, входящие из интернета? Изменяет ли DNAT пакеты, исходящие в интернет?

10.14 Выполнение самостоятельной работы

Скачаем и распакуем архив с шаблоном работы следующими командами.

```
mkdir -p ~/tcp-ip; cd ~/tcp-ip
wget -r http://ftp.iu7.bmstu.ru/nets/lab-lan-hw.tar.gz
rm -rf lab-lan-hw
tar -xvf lab-lan-hw.tar.gz
cd lab-lan-hw
```

Всего будут запущены следующие машины (рисунок 10.1):

– **r1** и **r2**: маршрутизаторы первой и второй локальной сети соответственно;

– ws11, ws12, ws13, ws14 —три рабочих станции в первой локальной сети;

– s11, s12 и s13 — серверы в первой локальной сети, к указанной в задании службе которых требуется организовать доступ извне;

– рабочая станция ws21 во второй локальной сети.

Если в вашем задании требуются не все из машин в первой локальной сети, удалите информацию о них из файла **lab.conf**, а также их каталоги и startup-файлы.

Не забываем заполнить отчёт. Отметим, что в файле **/etc/resolv.conf** на машинах **r1** и **r2** должен быть указан адрес доступного в реальной сети кеширующего DNS-сервера. В нашем случае он выглядит следующим образом — если ваш DNS-сервер имеет иной адрес, отредактируйте этот файл (никакие службы после изменения этого файла перезапускать не нужно).

```
nameserver 192.168.0.1
```

10.15 Варианты заданий для самостоятельной работы

Пока ещё выдаются индивидуально по электронной почте.

11 Служба доменных имён DNS

В конце этой главы (раздел 11.11) приведены возможные индивидуальные варианты заданий. В разделе 11.10 даны дополнительные указания по их выполнению.

11.1 Протокол DNS и служба имен

Изначально в Unix для получения IP-адреса по имени использовался файл **/etc/hosts**. Однако, его использование для получения адреса по имени компьютера в быстро растущем в 80-ые годы интернете быстро стало технически невозможно.

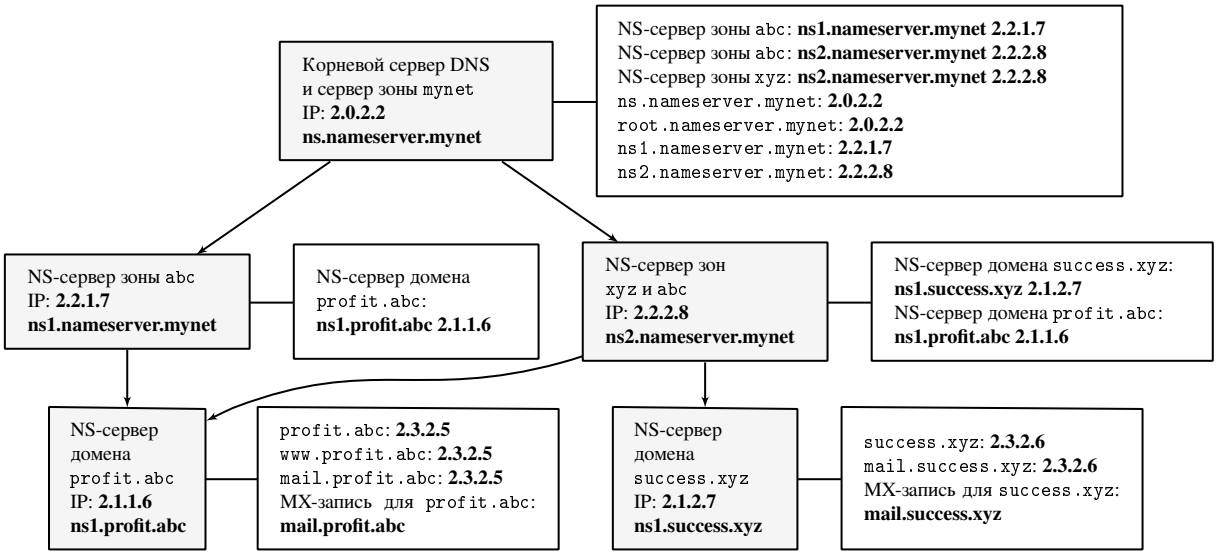


Рисунок 11.1 — Система DNS как иерархическая база данных

Система DNS реализует хорошо масштабируемый способ хранения отображения мнемонических имён в IP-адреса («прямая зона DNS») и обратного отображения IP-адресов в имена («обратная зона DNS»). Каждый сервер имён (NS-сервер) имеет свою область ответственности, которая известна вышестоящим серверам. На рис. 11.1 показана иерархическая организация базы данных прямого преобразования DNS с двумя зонами, в каждой из которых есть только одно доменное имя второго уровня.

Для повышения надежности каждый участник системы DNS должен как минимум дублироваться. Например, в интернете по состоянию на 2010 год существует 13 корневых серверов, за зону **.ru** отвечают 6 серверов и так далее. При этом каждый из них шести серверов имён зоны **.ru** имеет всю информацию о зоне (в случае корректной работы и с учетом времени на репликацию). В нашей лабораторной работе мы, тем не менее, ограничимся в основном единственным сервером имён для каждой зоны и домена, за исключением зоны **abc**, за которую будут отвечать два сервера (рисунок 11.1).

Корневой сервер DNS в нашем примере так же полностью ответственен за служебную зону **mynet**, в которой выделены имена для трёх серверов имён.

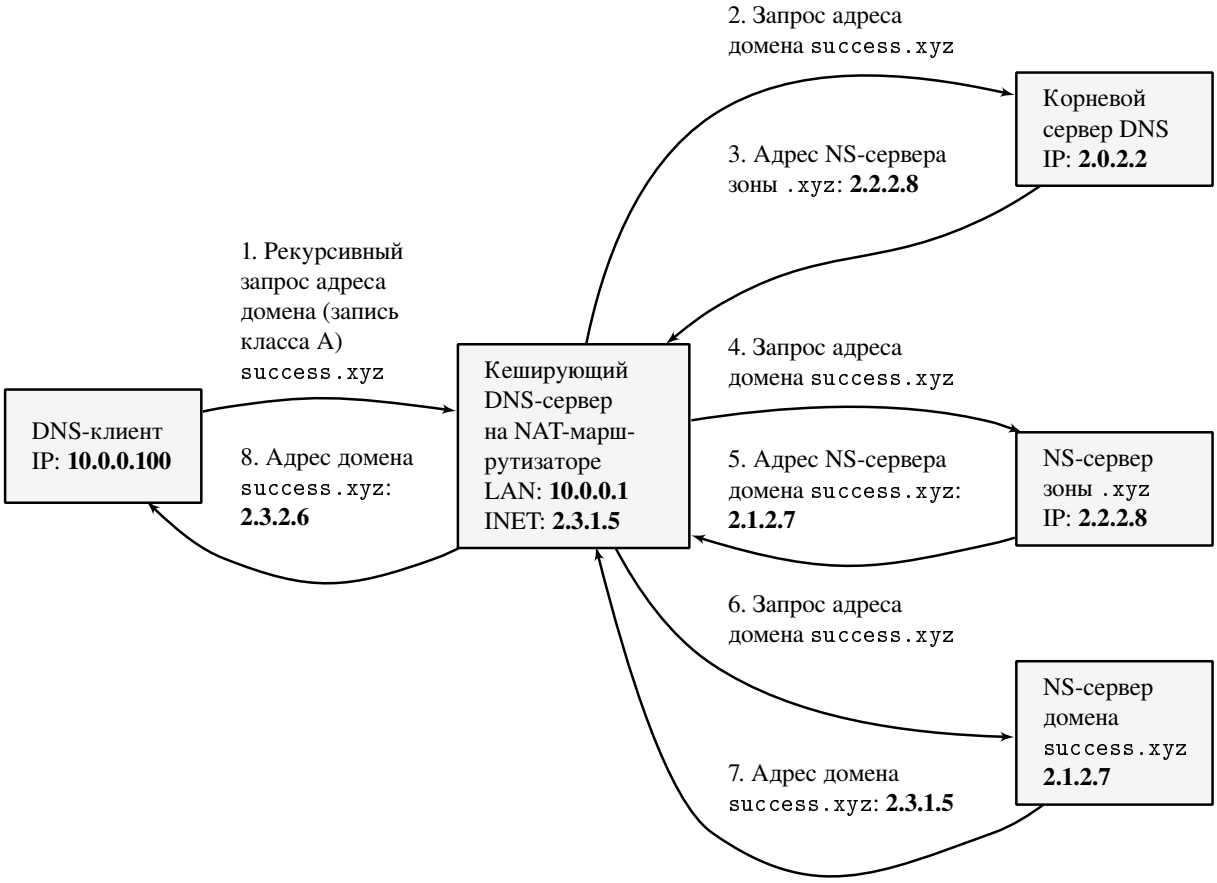


Рисунок 11.2 — Работа системы DNS

Каждая запись в БД DNS имеет ряд атрибутов:

1) исходное доменное имя (с точкой на конце), например: **ru.**, **bmstu.ru.**, **iu7.bmstu.ru.**;
2) тип записи:
   - тип NS: запись о сервере имён для данного имени,
   - тип A: запись о IPv4-адресе для данного имени,
   - тип AAAA: запись о IPv6-адресе для данного имени,
   - тип MX: запись о почтовом сервере (*Mail eXchange*) для данного имени;
3) результирующее доменное имя или адрес;
4) прочие атрибуты, которые лень описывать.

На рисунке 11.2 показана работа системы DNS, включающей несколько авторитетных серверов имён, кеширующий сервер и клиента. Клиент знает адрес кеширующего сервера имён, в случае unix-подобных систем она находится в файле **/etc/resolv.conf**. Кеширующий сервер изначально знает адреса корневых серверов. Первые два сервера имён, получив

запрос от кеширующего сервера, выдают только информацию о нижестоящем сервере имён, а кеширующий сервер уже сам должен опрашивать всю цепочку NS-серверов. Таким образом, кеширующий сервер выполняет рекурсивный запрос клиента, опрашивая столько NS-серверов, сколько потребуются, и сохраняя в своём кеше все промежуточные результаты.

11.2 Работа DNS для систем во внутренней сети

Как видно из рисунка 11.4 и рисунка 12.1, у доменного имени **mail.profit.abc** особое положение. С точки зрения машин вне частной сети оно должно отображаться во внешний адрес NAT-маршрутизатора **r1**, то есть в адрес 2.3.1.5. С точки же зрения машин в частной сети (машины **r1**, **pc1**, **mail1**) оно должно отображаться в частный адрес машины адрес **mail1**, то есть в адрес 10.0.0.2.

Из этой ситуации возможно несколько выходов. Самый очевидный — развернуть отдельный домен (скажем, **profit.local**) для частных адресов и использовать его адреса (скажем, **mail.profit.local**) в локальной сети. Такой домен не нужно подключать в единую систему DNS, о нем достаточно знать машине **r1**, поскольку на ней развёрнут кеширующий DNS-сервер, используемый машинами локальной сети.

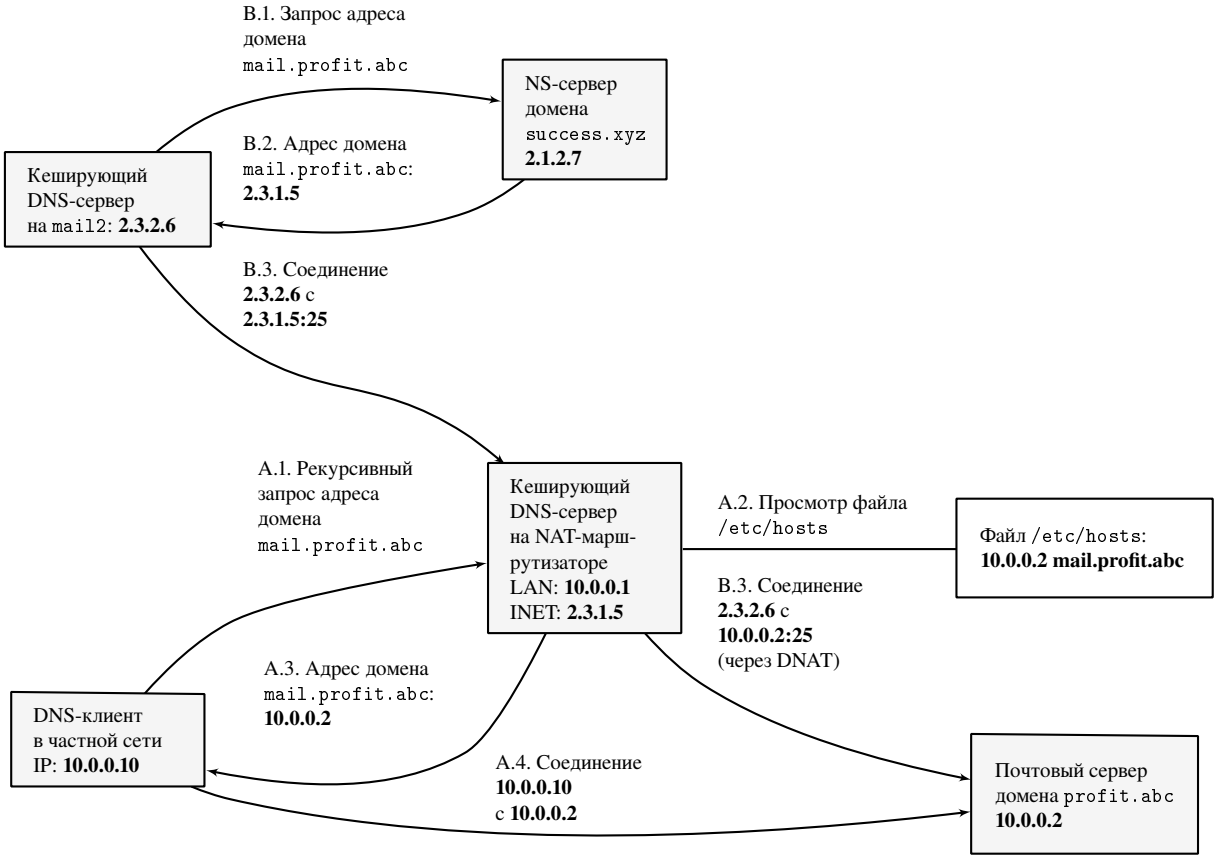


Рисунок 11.3 — Имя DNS для машины во внутренней сети

Второй выход так же сводится к настройке кеширующего сервера на машине **r1**. Он «обучен» обрабатывать запрос адреса имени **mail.profit.abc** особым образом. Как показано на рис. 11.3, кеширующий сервер должен сначала проверить, нет ли записи о таком имени в файле **/etc/hosts**. Следует отметить, что для клиента содержимое файла **/etc/hosts** имеет по-умолчанию наиболее высокий приоритет, а вот наш кеширующий сервер по-умолчанию его не использует.

11.3 Настройка DNS-сервера BIND

BIND (Berkeley Internet Name Domain) — открытая и наиболее распространённая реализация DNS-сервера, обеспечивающая выполнение преобразования DNS-имени в IP-адрес и наоборот.

Все его файлы конфигурации на машинах с ОС Debian находятся в каталоге **/etc/bind**. Например, для корневого DNS-сервера, ответственного, также, за зону **france**, в нём будут находится следующие файлы:

1) db.0
2) db.127
3) db.255
4) db.empty
5) db.france
6) db.local
7) db.root
8) named.conf
9) named.conf.local
10) rndc.key

При запуске сервер BIND считывает файл **named.conf**. Его изменение не предполагается в данной работе, но для лучшего понимания работы сервера, кратко рассмотрим его. В этом файле мы увидим приметно следующее:

```
options     directory "/var/cache/bind";    allow-recursion "none";;    recursion

// Be authoritative for the localhost forward and reverse zones, and for
// broadcast zones as per RFC 1912
zone "localhost"     type master;    file "/etc/bind/db.local";;
zone "127.in-addr.arpa"     type master;    file "/etc/bind/db.127";;
zone "0.in-addr.arpa"     type master;    file "/etc/bind/db.0";;
zone "255.in-addr.arpa"     type master;    file "/etc/bind/db.255";;

// If you are just adding zones, please do that in /etc/bind/named.conf.local
include "/etc/bind/named.conf.local";
```

Здесь определяются прямая и обратная локальные зоны (и ещё какие-то), их типы и файлы, в которых описываются dns-записи. Последней

строчкой включается файл **named.conf.local**, в который нас и просят добавлять свои зоны, а не «мусорить» здесь. Давайте последуем их совету.

Для нашего случая (машина одновременно является и корневым dns-сервером и отвечает за зону Франции) файл **named.conf.local** будет следующим:

```
zone "."     type master;    file "/etc/bind/db.root";    allow-query  any; ;

zone "france"     type master;    file "/etc/bind/db.france";    allow-query  any;
```

Теперь рассмотрим поподробнее. Здесь указаны 2 зоны: «.» - корневая, «france» — соответственно Франция. Первый параметр — тип зоны, у обоих он master (определяет зону, для которой данный сервер является первичным сервером). Также бывают slave, hint и stub, если интересно с этими параметрами можно ознакомиться с ними в доке к BIND.

Далее указан файл откуда брать информацию о данной зоне. В общем-то их можно именовать как угодно, но чтобы не запутаться лучше именовать db.zonename.

Параметр allow-query — хосты, которым разрешено выполнять запросы к DNS, в нашем случае — любые.

Посмотрим что у нас в файлах db.*. Для примера возьмём db.root:

```
$TTL    1D
.           IN SOA       root.ns.france. admin.paris.france. (
                                        1
                                        4H
                                        1H
                                        1W
                                        1D )

                        IN NS   root.ns.france.

root.ns.italy.       IN A        2.20.1.1
ns.italy.        IN A        2.20.20.1
root.ns.france.      IN A         2.10.1.1
ns.france.           IN A         2.10.10.1

italy.                      IN NS   ns.italy.
italy.                      IN NS   root.ns.italy.
france.                     IN NS   ns.france.
france.                     IN NS   root.ns.france.
```

Инструкция $TTL задает время жизни зоны (в сек.). Это значение определяет, как долго другие серверы будут кэшировать информацию об этой зоне.

Следующая запись — загадочная Start of Authority (SOA, начало авторитетности). Тут представлено краткое описание зоны — каково ее поведение и как серверам следует себя с ней вести. Два имени после слова SOA означают следущее: первое — основной NS-сервер (у зоны должен

быть минимум один сервер имён), второе — почтовый адрес админа, где символ «@» заменен на первую по порядку точку. Запись после закрывающей скобки также относится к "."(о NS далее).

Записи с "IN A"можно провести аналогию с #define name value, т.е. если мы dig'ом будем допытываться от этого сервера кто такой ns.italy, он нам сразу ответит что его IP-адрес — 2.20.20.1.

Записи с «IN NS» — это сервера имён, т.е. если мы, опять-таки, будем спрашивать у этого сервера какой адрес, скажем о sevik.italy, то он сначала посмотрит нет ли такого имени среди A-записей, и в случае неудачи выдаст все NS сервера ответственные за зону .italy, а именно ns.italy. (2.20.20.1) и root.ns.italy (2.20.1.1)

Напомним что зона должна знать о всех нижестоящих NS-серверах, именно по-этому здесь и прописаны и Италия, и Франция.

Обратите внимание, что в конце имени стоит точка. Точка ставится для того, чтобы в конце имени не добавлялась запись домена. Например, если мы не поставим точку в конце имени ns.mydomain.local, то фактическое имя компьютера станет: ns.mydomain.local.mydomain.local

P.S. в принципе в строчках IN NS можно ставить непосредственно IP-адрес, но по ряду причин это делать не рекомендуется.

По аналогии делается и для остальных зон.

Настройка MX-записей. . Для MX записей настройка аналогична NS-серверам:

```
mail.rome.italy.          IN A          2.20.25.10
rome.italy.       IN MX  10    mail.rome.italy.
```

У MX-записей есть понятие приоритета. Насколько я понял приоритеты нужны для нескольких почтовых серверов на одной машине (но на самом деле - хз).

Важно: имя хоста, указанного в записи MX, должно содержать IP-адрес, определённый с помощью записи IN A.

11.4 Настройка кэширующего dns-сервера

Для кэширующего dns-сервера используется программа Power-dns. Её настройка крайне проста. В файле /etc/powerdns/recursor.conf нас интересует строчка

```
hint-file=/etc/bind/db.root
```

Которая задаёт файл с укаанными корневыми dns-серверами.

Его синтаксис аналогичен описанному выше. Отличия только в том, <вставить сюда что такое 1D> (какое время хранить кэш с данной зоны?). Пример для 2-х корневых dns:

```
.                    1D IN NS ns22011.italy
ns22011.italy. 1D IN A 2.20.1.1
.                    1D IN NS ns21011.france
ns21011.france. 1D IN A 2.10.1.1
```

11.5 Настройка почтового сервера

Все настройки почтового сервиса Exim4 хранятся в /etc/exim4/update-exim4.conf.conf

```
dc_eximconfig_configtype='internet'
dc_other_hostnames='paris.france'
dc_local_interfaces='0.0.0.0'
dc_readhost=''
dc_relay_domains=''
dc_minimaldns='false'
dc_relay_nets='127.0.0.1/8:2.10.0.0/16'
dc_smarthost=''
CFILEMODE='644'
dc_use_split_config='false'
dc_hide_mailname=''
dc_mailname_in_oh='true'
dc_localdelivery='mail_spool'
```

Где dc_other_hostnames=’paris.france’ - имя вашего домена, dc_relay_nets=’127.0.0.1/8:2.0.0.0/8’ - список сетей, для которых будет открыт SMTP open relay.

Также нужно будет поменять доменное имя машины, которое находится здесь: /etc/mailname

Для окончательной проверки работы почты лучше использовать nc и напрямую общаться с SMTP сервером, но для отладки можно воспользоваться следующей командой:

```
echo "mail data" | mail -s "subject" username@post.domen
```

11.6 Развёртывание сети

Исходными данными к работе являются девять заранее сконфигурированных виртуальных машин (рис. 11.4).

Данная система не подключается к реальной сети и является самодостаточной. В состав системы входят:

- корневой DNS-сервер и два DNS-сервера, отвечающих за зоны имён;
- два DNS-сервера, отвечающих за домены;
- один NAT-маршрутизатор;
- один клиентский компьютер.

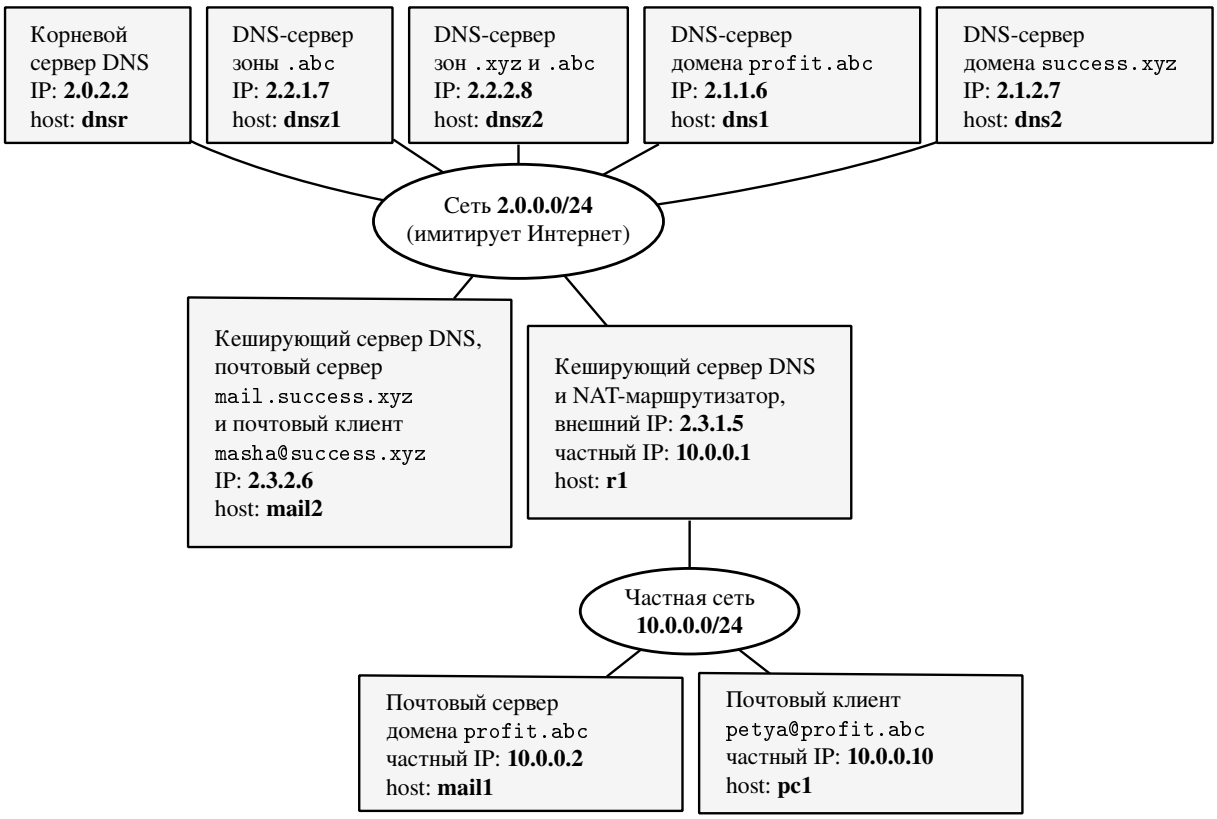


Рисунок 11.4 — Сеть передачи данных

Чтобы избежать необходимости настройки IP-маршрутизации, в модели имеется всего две IP-сети, одна из которых — частная, за NAT-маршрутизатором[1]:

– сеть 2.0.0.0/24 имитирует глобальную сеть;
– частная сеть 10.0.0.0/24 за NAT-маршрутизатором содержит один из почтовых серверов.

11.7 Настройка системы DNS

Изучите содержимое файла **/etc/bind/named.conf.local** на NS-серверах. Так же просмотрите содержимое файлов **db.root**, **db.abc**, **db.xyz**, **db.profit.abc**, **db.profit.xyz**. В этих файлах задаётся:

1) отображение имён в адреса (строчки с **IN A**);
2) записи о NS-серверах (строчки с **IN NS**);
3) MX-записи о SMTP-серверах доменов (строчки с **IN MX**, число указывает приоритет);
4) записи о зоне (**SOA**) которая существует, когда данный сервер отвечает («авторитетен») за указанную зону.

Авторитет сервера проистекает из того, что «вышестоящий» сервер сообщает о нём как об сервере имён некоторый зоны. Авторитет корневого

[1] В индивидуальных заданиях частной сети может и не быть.

сервера заключается только в вере в него клиентов DNS, поэтому атаки на корневые сервера DNS — довольно частое явление (пока, к счастью, все они были неудачными).

Слово **IN** является сокращением от *Internet* (видимо когда-то система могла DNS работать не только с протоколом DNS в IP-сетях). Цифры в SOA-записи управляют временем кеширования записей на кеширующих серверах. Два имени после слова SOA означают следующее: первое — основной NS-сервер (у зоны должен быть минимум один сервер имён), второе — почтовый адрес админа, где символ «@» заменен на первую по порядку точку.

Заметьте, что мы нахально задали в качестве адреса администратора **admin@nameserver.mynet**, но не настроили MX-запись и почтовую службу для этого домена, поскольку это предполагается сделать самостоятельным заданием в перспективе.

11.8 Работа системы DNS

Давайте проследим за ходом работы системы DNS. Запустим утилиту **tcpdump**, как показано ниже, на следующих машинах: **dnsr**, **dnsz1**, **dns1**

```
tcpdump -tnvv -i eth0 -s 1518 udp
```

Теперь перейдём на машину **mail2** и проимитируем вручную работу кеширующего сервера по поиску MX-записи домена **profit.abc**. Поскольку мы «знаем» только адрес корневого сервера, то начнём с команды обращения к корневому серверу.

```
dig @2.0.2.2 profit.abc MX
```

Проанализируйте вывод этой команды и продолжите поиск MX записи самостоятельно, «спускаясь» по иерархии DNS-серверов. Отметим, что именно таким последовательным обходом от корня при помощи команды **dig** и нужно проверять настроенную структуру DNS во время выполнения индивидуальных заданий. Крайне малопродуктивно использовать для этой цели команды типа **ping**, поскольку ими невозможно диагностировать, какой именно из DNS-серверов не выдал ожидаемый ответ.

Автоматизировать процесс «спуска» поможет опция **+trace**, заставляющая команду показать все шаги поиска:

```
dig @2.0.2.2 profit.abc MX +trace

; <<>> DiG 9.5.0-P2 <<>> @2.0.2.2 profit.abc MX +trace
; (1 server found)
;; global options:  printcmd
.           86400   IN  NS  ns.nameserver.mynet.
;; Received 65 bytes from 2.0.2.2#53(2.0.2.2) in 24 ms
```

```
abc.            86400   IN  NS  ns2.nameserver.mynet.
abc.            86400   IN  NS  ns1.nameserver.mynet.
;; Received 112 bytes from 2.0.2.2#53(ns.nameserver.mynet) in 4 ms

profit.abc.     86400   IN  NS  ns1.profit.abc.
;; Received 62 bytes from 2.2.2.8#53(ns2.nameserver.mynet) in 6 ms

profit.abc.     86400   IN  MX  10 mail.profit.abc.
profit.abc.     86400   IN  NS  ns1.profit.abc.
;; Received 99 bytes from 2.1.1.6#53(ns1.profit.abc) in 3 ms
```

После успешного нахождения искомой DNS-записи вручную давайте найдём эту запись при помощи нашего кеширующего сервера. Отдадим на машине **mail2** следующую комманду.

```
dig @127.0.0.1 profit.abc MX
```

Как видно, кеширующий сервер сразу выполнил этот рекурсивный запрос, в то время как наши авторитетные серверы настроены так, чтобы выполнять только один шаг. В результате в запущенной системе с машины **mail2** сразу должен пинговаться хост **mail.profit.abc**, а с машин **pc1** и **mail1** — хост **mail.success.xyz**.

В нащеми случае для сброса кеша кеширующего сервера, если понадобится, на его машине надо выполнить команду.

```
rec_control wipe-cache <имена доменов>
```

11.9 Контрольные вопросы

Для самоконтроля полученных знаний рекомендуется ответить на следующие вопросы.

1) Какой транспортный протокол и порт используется системой DNS в нашем примере?

2) На каких компьютера корневая зона DNS прописана как «master»? Почему?

3) Как кеширующие сервера узнают про корневые DNS-серверы?

4) Каким образом в настройках авторитетного DNS-сервера указывается информация о «нижестоящих» авторитетных серверах?

5) В какой адрес в примере преобразуется имя **nameserver.mynet**? Почему?

6) Для чего используется A-запись? MX-запись?

7) Вы набрали в адресе броузера **http://www.bmstu.ru** и нажали «ввод». Запись какого типа какого домена будет получена клиентом?

8) Может ли MX запись для домена совпадать по имени с самим доменом?

9) Злонамеренные программы для Windows часто меняют файл **System32/drivers/etc/hosts** на зараженном компьютере. Как вы думаете, для чего они это делают?

10) Будет ли выполнен запрос к DNS при команде **ping localhost**?

11.10 Выполнение самостоятельной работы

Скачаем и распакуем архив с шаблоном работы следующими командами.

```
mkdir -p ~/tcp-ip; cd ~/tcp-ip
wget -r http://ftp.iu7.bmstu.ru/nets/lab-dns-hw.tar.gz
rm -rf lab-dns-hw
tar -xvf lab-dns-hw.tar.gz
cd lab-dns-hw
```

Отметим, что в примере имеется служебная зона **.mynet**, в которой заведены записи для самих серверов имён, включая корневые. При выполнении индивидуальных заданий вы можете как настраивать, так и не настраивать эту зону — по вашему желанию.

С целью уменьшения времени выполнения индивидуальных заданий работу следует вести следующим образом:

1) настроить файлы конфигурации DNS-сервера bind до запуска виртуальных машин;

2) запустить построение первой версии отчета, в котором появятся картинки со структурой вашей сети и структурой системы DNS (или сообщение об ошибке);

3) повторять пп. 1-2, пока структура DNS не станет соответствовать ожидаемой;

4) запустить виртуальные машины и выполнить остальную часть работы.

11.11 Варианты заданий для самостоятельной работы

Варианты высылаются по почте.

12 Почтовая система и протокол SMTP

В этой главе мы рассмотрим работу почтовой системы интернета.

Мы будем использовать сеть из предыдущей главы (рисунок 11.1), поскольку работа почтовой системы тесно связана с работой системы DNS.

12.1 Развёртывание сети

После успешного пинга соединимся с 25-м портом указанных машин командой следующего вида.

```
nc <mail.домен.зона> 25
```

Разъединимся после успешного соединения (вы увидите приглашение в случае успеха, выход по Ctrl+C).

12.2 Почтовая система, протоколы SMTP и POP3

На рис. 12.1 показана общая схема передачи почты в нашей системе, а на рис. 12.2 — процесс передачи одного письма от отправителя получателю.

Стандартный порт протокола SMTP — 25, протокола POP3 — 110. Оба протокола работают поверх протокола TCP.

12.2.1 Запуск системы

Для запуска можно использовать команду **ltabstart**, находясь в каталоге с файлами виртуальной сети.

12.3 Настройка почтовых ящиков

В системе уже настроены и запущены необходимые почтовые службы: SMTP-сервер exim4 на машинах **mail1** и **mail2**, POP3-сервер popa3d на машине **mail1**. Изучите содержимое файла конфигурации **update-exim4.conf.conf** в каталоге **/etc/exim4/**. Обратите внимание на список сетей, с которых сервер принимает транзитную почту, и список его доменов.

После того, как вы убедились в работоспособности системы, давайте добавим почтовые ящики, по одному на каждый домен. Мы пойдём по самому простому (и обычно не используемому на практике) пути, заведя в системе обычного системного пользователя.

На машине **mail1** выполните следующие команды.

```
adduser petya
```

Введите дважды пароль пользователя, пароль придумайте сами. В ответ на все остальные вопросы достаточно нажать «ввод».

На машине **mail2** выполните следующие команды.

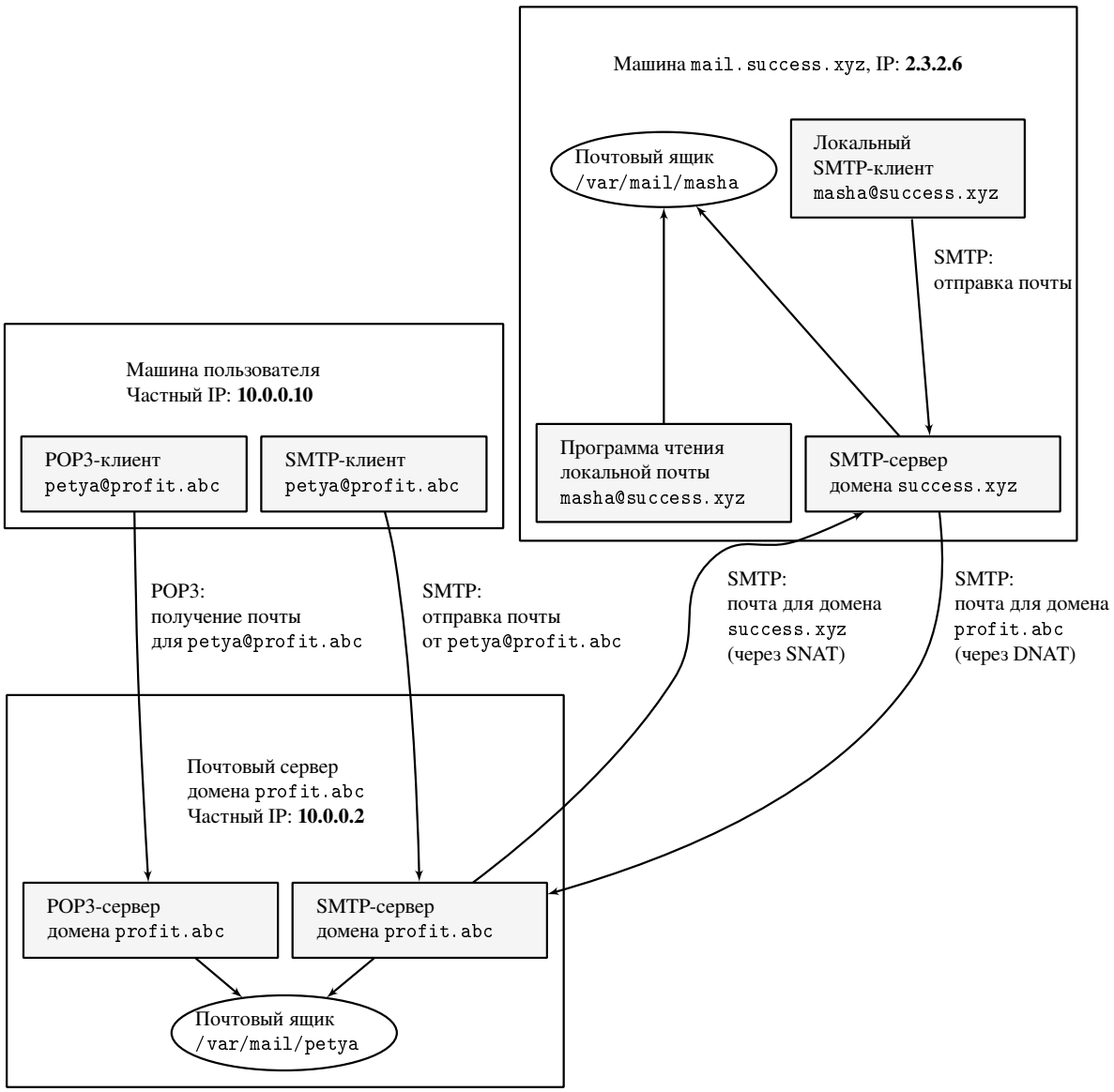


Рисунок 12.1 — Почтовая система

```
adduser masha
```

Серверы SMTP и POP3 будут использовать заданное нами имя пользователя и пароль. Почта такого пользователя будет хранится в каталоге **/var/mail**.

12.4 Сеанс протокола SMTP

После заведения пользователя пошлем ему почту.

Сначала мы пошлём почту напрямую на сервер, где находится его почтовый адрес. Для этого запустим на машине **pc1** следующую команду (после её запуска вы войдёте в режим общения с удалённым сервером, не надо нажимать Ctrl+C и выходить из него).

```
nc mail.success.xyz 25
```

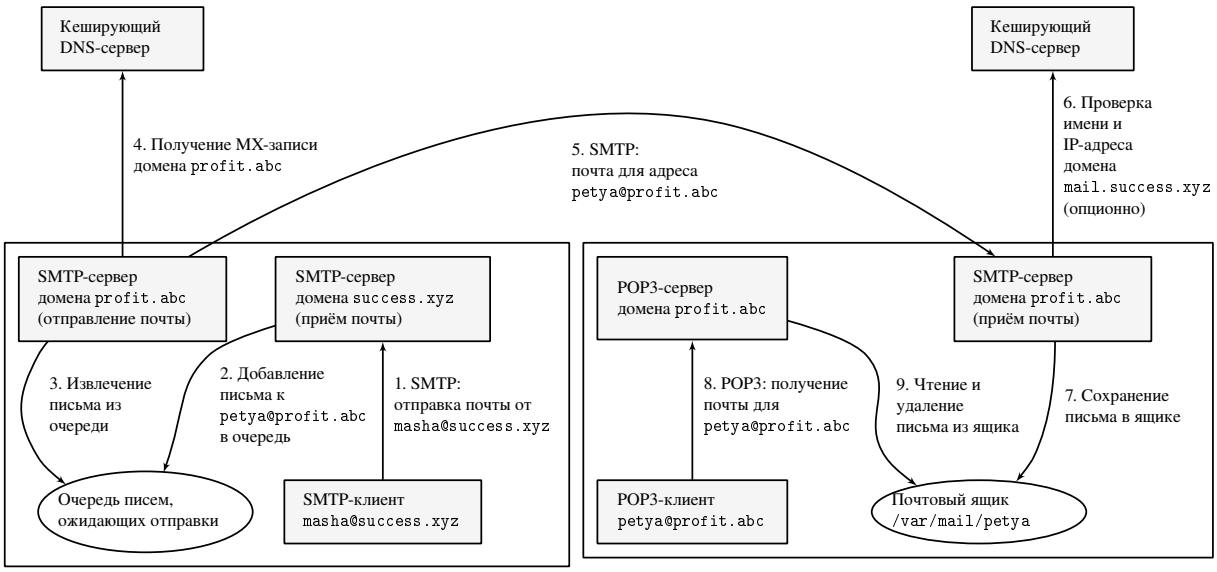


Рисунок 12.2 — Передача почты

Мы соединились с SMTP-сервером и можем написать письмо для пользователя **masha.success.xyz**. Для этого передадим серверу несколько команд (внимательно читайте ответы сервера после выполнения каждой команды). Сначала представимся, нагло указав **localhost** в качестве нашего имени, поскольку проверки этого имени сервером в лабе нет, и сообщим наш адрес.

```
HELO localhost
MAIL FROM:petya@profit.abc
```

Затем укажем получателя письма

```
RCPT TO:masha@success.xyz
```

Теперь перейдём к воду текста письма и введём само письмо.

```
DATA
```

Вы перешли в режим ввода письма и теперь сервер не будет давать ответы до завершения ввода тела письма. Ввод письма завершается строкой, состоящей из единственной точки.

```
Subject: Greetings from profit.abc
Hi, Masha!
.
```

При нажатии клавиши «ввод» после последней строки (состоящей из единственной точки) письмо должно быть принято сервером (код операции в ответе — 250). Завершим сеанс командой **QUIT**.

Все команды можно было писать и маленькими буквами, но традиционно их пишут большими.

Посмотрим на вывод tcpdump на машине **mail2** чтобы понять, что же передавалось в IP-пакетах.

Попытайтесь повторить сеанс и передать сообщение несуществующему пользователю в домене (например, **ivan@success.xyz**). Проведите также опыт с отсылкой почты, указав неверный домен (например, **masha@suces.xyz**). В обоих случаях изучите содержимое журнала почтового сервера, с которрым вы общались, следующей командой.

```
cat /var/log/exim4
```

Убедимся, что в каталоге **/var/mail** на машине **mail2** появилось сообщение, просмотрим его командой **cat**.

Теперь мы сымитируем «нормальную» передачу почты, послав транзитное сообщение «нашему» почтовому серверу в надежде, что он сам найдёт нужный SMTP-сервер по MX-записи и передаст ему почту.

Это сеанс будет отличаться только соединения командой (мы по-прежнему отдаём её на машине **pc1**).

```
nc mail.profit.abc 25
```

Кроме того, лучше задать чуть другую тему письма, чтобы их потом мы смогли различить при получении.

После успешного сеанса посмотрим журнал сервера exim4 на машине **mail1**. Журналы этого сервера находятся в каталоге **/var/log/exim4**, найдите нужный при помощи команды **ls** и выведите его командой **cat** или **less**. В случае успеха вы обнаружите в журнале сообщение об успешной передаче письма на адрес **masha@success.xyz** серверу **mail.success.xyz**.

12.5 Чтение локальной почты

После просмотра содержимого почтового ящщика командой **cat /var/mail/masha** прочитаем полученные сообщения простейшим почтовым клиентом. Поскольку на машине **mail2** «по легенде» находится и рабочее место пользователя masha, то поступим самым простым образом: сменим пользователя и прочитаем его локальную прочту командой **mail**

```
su masha
mail
```

В случае успеха мы увидим в ящике два сообщения. Ответим на второе из них и напишем что-нибудь в ответ (ответ можно написать, как мы делали ранее, или нажав 'r' в программе **mail**). Просмотрим логи серверов, чтобы убедиться, что почта прошла.

12.6 Получение почты через POP3-сервер

Пользователь **petya@profit.abc** работает на машине **pc1**, поэтому он не может воспользоваться командой **mail** для чтения своей почты, которая находится на машине **mail1**. Для приёмы почты ему нужно прибегнуть к

протоколу POP3. Мы проведём сессию протокола POP3 «вручную» при помощи телнет-клиента. Для этого запустим на машине **pc1** клиента телнет.

```
nc mail.profit.abc 110
```

После её запуска мы должны подключиться к POP3-серверу. Сначала нам нужно авторизоваться.

```
USER petya
PASS <пароль>
```

В случае успеха авторизации мы можем получать список сообщений, текст отдельного сообщения, а также удалять их из почтового ящика. Просмотрим список сообщений (нам должен придти ответ) и прочитаем полученное.

```
LIST
RETR 1
```

Команда **DELE** помечает сообщение как удалённое, а команда **QUIT** — закрывает соединение и физически удаляет сообщения.

12.7 Борьба с открытыми почтовыми релеями

Открытый релей — это почтовый сервер, готовый получать почту от кого угодно для кого угодно и передающий её далее на SMTP-сервер получателя по MX-записи. Очевидно, что открытый релей сразу станет мишенью спамеров. Отметим, что «правильный» SMTP-сервер может принимать почту для чужих пользователей только от компьютеров, относящихся к его так называемой доверенной сети (а совсем правильный — еще и требовать при этом авторизации). Изучите список доверенных сетей, для которых разрешен релей, у обоих SMTP-серверов. Это можно сделать, просмотрев файл **/etc/exim4/update-exim4.conf.conf**.

> **Задание № 44: Проверка почтового релея**
>
> Убедимся, что наши почтовые серверы не являются открытыми релеями. Для проверки с компьютера, не относящегося к доверенной сети SMTP-сервера, следует напрямую послать письмо, указав адресатом «чужого» для сервера пользователя. Например, попытайтесь отправить с машины **mail2** письмо для **masha@success.xyz** напрямую на сервер **mail.profit.abc** (то есть Маша отсылает письмо самой себе, но через «чужой» сервер). В результате вы должны получить ошибку после команды **RCPT TO**. В журнале SMTP-сервера также появится сообщение о заблокированной попытке использовать сервер как релей.

12.8 Контрольные вопросы

Для самоконтроля полученных знаний рекомендуется ответить на следующие вопросы. Дайте ответы на следующие контрольные вопросы.

1) Какие сети являются доверенными для SMTP-серверов в этой работе?

2) Какой код ответа SMTP-сервера говорит об успешном приёме письма?

3) Какие коды ошибок SMTP-сервера и в каких ситуациях вы наблюдали в этой работе?

4) Пусть пользователь почтового сервера **mail.ru** посылает письмо пользователю сервера **yandex.ru**, причём оба пользователя читают почту через веб-интерфейс. Как вы думаете, используется (и где) в этом случае протокол SMTP? Протокол POP3?

5) Требует ли протокол SMTP явной идентификации отправителя (логина-пароля, например) для передачи письма пользователю, ящик которого находится на этом сервере? Почему?

6) Требует ли протокол POP3 идентификации клиента?

12.9 Выполнение самостоятельной работы

Список использованных источников

1. *Робачевский, Андрей*. Операционная система UNIX / Андрей Робачевский, Сергей Немнюгин, Ольга Стесик. — СПб: БХВ-Петербург, 2010. — 656 с.
2. *Stevens, W. Richard*. TCP/IP illustrated (vol. 1): the protocols / W. Richard Stevens. — Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1993.
3. LAN/MAN Standards Committee (IEEE 802) of the IEEE Computer Society.
   `http://www.ieee802.org/`.
4. *Postel, J.* Transmission Control Protocol. — RFC 793 (Standard). — 1981. — Updated by RFCs 1122, 3168, 6093.
   `http://www.ietf.org/rfc/rfc793.txt`.
5. *Wikipedia*. Address Resolution Protocol — Wikipedia, The Free Encyclopedia. — 2011.
   `http://en.wikipedia.org/wiki/Address_Resolution_Protocol`.
6. *ISO/IEC 7498-1:1994*. Information technology — Open Systems Interconnection — Basic Reference Model: The Basic Model / ISO/IEC 7498-1:1994. — Geneva, Switzerland: ISO, 1994.
7. *Олифер, В.Г.* Компьютерные сети. Принципы, технологии, протоколы / В.Г. Олифер, Н.А. Олифер. — СПб: Питер, 2010. — 944 с.
8. *Zimmermann, Hubert*. OSI Reference Model — The ISO Model of Architecture for Open Systems Interconnection / Hubert Zimmermann // *IEEE Transactions on Communications*. — 1980. — Vol. 28, no. 4. — Pp. 425–432.
9. *Wikipedia*. Internet Protocol — Wikipedia, The Free Encyclopedia. — 2011.
   `http://en.wikipedia.org/wiki/Internet_Protocol`.
10. *Reynolds, J.* Assigned Numbers. — RFC 1700 (Historic). — 1994. — Obsoleted by RFC 3232.
    `http://www.ietf.org/rfc/rfc1700.txt`.
11. *Wikipedia*. Internet Control Message Protocol — Wikipedia, The Free Encyclopedia. — 2011.
    `http://en.wikipedia.org/wiki/Internet_Control_Message_Protocol`.
12. *Baker, F.* Requirements for IP Version 4 Routers. — RFC 1812 (Proposed Standard). — 1995. — Updated by RFC 2644.
    `http://www.ietf.org/rfc/rfc1812.txt`.
13. Duplicate MAC Addresses on Cisco 3600 Series.
    `http://www.cisco.com/en/US/ts/fn/misc/7.html`.
14. *IANA*. Assigned Internet Protocol Numbers. — 2011.
    `http://www.iana.org/assignments/protocol-numbers/protocol-numbers.x`
15. Scapy.
    `http://www.secdev.org/projects/scapy/`.

16. *ANSI/IEEE Std 802.1D, 1998 edition (ISO/IEC 15802-3:1998)*. Media Access Control (MAC) Bridges / 1998 edition (ISO/IEC 15802-3:1998) ANSI/IEEE Std 802.1D. — IEEE, 1998.
http://ftp.iu7.bmstu.ru/papers/802.1D-1998.pdf.
17. *ANSI/IEEE Std 802.1D, 2004 edition*. Media Access Control (MAC) Bridges. — 2004.
http://standards.ieee.org/getieee802/download/802.1D-2004.pdf.
18. *Postel, J.* User Datagram Protocol. — RFC 768 (Standard). — 1980.
http://www.ietf.org/rfc/rfc768.txt.
19. *Wikipedia*. Berkeley sockets — Wikipedia, The Free Encyclopedia. — 2011.
http://en.wikipedia.org/wiki/Berkeley_sockets.
20. *Wikipedia*. Autonomous system (Internet) — Wikipedia, The Free Encyclopedia. — 2011.
http://en.wikipedia.org/wiki/Autonomous_system_(Internet).
21. *Malkin, G.* RIP Version 2. — RFC 2453 (Standard). — 1998. — Updated by RFC 4822.
http://www.ietf.org/rfc/rfc2453.txt.
22. *Семёнов, Юрий Алексеевич*. Внутренний протокол маршрутизации RIP. — 2008.
http://book.itep.ru/4/44/rip44111.htm.
23. *Meyer, G.* Triggered Extensions to RIP to Support Demand Circuits. — RFC 2091 (Proposed Standard). — 1997.
http://www.ietf.org/rfc/rfc2091.txt.
24. *Zaghal, Raid Y.* EFSM/SDL modeling of the original TCP standard (RFC793) and the Congestion Control Mechanism of TCP Reno / Raid Y. Zaghal, Javed I. Khan.
http://www.medianet.kent.edu/techreports/TR2005-07-22-tcp-EFSM.pdf.
25. *Wikipedia*. Transmission Control Protocol — Wikipedia, The Free Encyclopedia. — 2011.
http://en.wikipedia.org/wiki/Transmission_Control_Protocol.
26. *Семёнов, Юрий Алексеевич*. Протокол TCP. — 2008.
http://book.itep.ru/4/44/tcp_443.htm.
27. *Kegel, Dan*. The C10K problem / Dan Kegel. — 2011.
http://www.kegel.com/c10k.html.
28. *Nagle, J.* Congestion Control in IP/TCP Internetworks. — RFC 896. — 1984.
http://www.ietf.org/rfc/rfc896.txt.
29. *Braden, R.* Requirements for Internet Hosts - Communication Layers. — RFC 1122 (Standard). — 1989. — Updated by RFCs 1349, 4379, 5884, 6093.
http://www.ietf.org/rfc/rfc1122.txt.
30. *Floyd, S.* The NewReno Modification to TCP's Fast Recovery Algorithm. — RFC 3782 (Proposed Standard). — 2004.

http://www.ietf.org/rfc/rfc3782.txt.
31. *Firewall.cx*. Transmission Control Protocol (TCP) Introduction. — 2003.
http://www.firewall.cx/tcp-intro.php.
32. *Wikipedia*. Sliding Window Protocol — Wikipedia, The Free Encyclopedia. — 2011.
http://en.wikipedia.org/wiki/Sliding_Window_Protocol.
33. TCP Sliding Window — Interactive Networking Tutorial.
http://www.osischool.com/protocol/Tcp/slidingWindow/index.php.
34. *Minshall, G.* A Proposed Modification to Nagle's Algorithm / G. Minshall. — 1999.
http://tools.ietf.org/html/draft-minshall-nagle-01.
35. *Mogul, J. C.* Rethinking the TCP Nagle algorithm / J. C. Mogul, G. Minshall. — 2001.
http://dl.acm.org/citation.cfm?id=382177.
36. tcp(7) - Linux man page.
http://linux.die.net/man/7/tcp.
37. *Wikipedia*. Dynamic Host Configuration Protocol — Wikipedia, The Free Encyclopedia. — 2011.
http://en.wikipedia.org/wiki/DHCP.
38. *Wikipedia*. Network address translation — Wikipedia, The Free Encyclopedia. — 2011.
http://en.wikipedia.org/wiki/Network_address_translation.
39. *Wikipedia*. Port forwarding — Wikipedia, The Free Encyclopedia. — 2011.
http://en.wikipedia.org/wiki/Port_forwarding.
40. *Andreasson, Oskar*. Iptables Tutorial 1.1.19. — 2003.
http://www.opennet.ru/docs/RUS/iptables/.
41. *Wikipedia*. Virtual private network — Wikipedia, The Free Encyclopedia. — 2011.
http://en.wikipedia.org/wiki/VPN.
42. *Wikipedia*. Transport Level Security — Wikipedia, The Free Encyclopedia. — 2011.
http://en.wikipedia.org/wiki/Transport_Layer_Security.
43. OpenVPN.
http://openvpn.net.
44. Сетевые сервисы и настройка сети. Преобразование имён и IP-адресов: практика // Пользовательские и технические аспекты ПСПО (курс лекций). — 2008.
http://uneex.ru/PspoClasses/080703/02DNSPractice.

Приложение А Настройка рабочего места

Общие требования

Для работы нам понадобится любая система на основе GNU/Linux (например, Ubuntu), работающая на компьютере архитектуры x86 или amd64. Для развёртывания Netkit нужно примерно 2 Гб свободного дискового пространства, которые он займёт в каталоге **/usr/local**. Далее предполагается, что в качестве интерпретатора команд используется программа GNU Bash, а для получения прав суперпользователя используется команда **sudo** и текущий пользователь имеет такую возможность.

Помимо пакета Netkit, установка которого рассмотрена ниже, для запуска лабораторных работ потребуются утилиты для создания мостов между сетевыми интерфейами и манипулирования виртуальными машинами, использующими технологию User Mode Linux. В дистрибутивах Debian и Ubuntu эти пакеты устанавливаются следующей командой.

```
sudo apt-get install bridge-utils uml-utilities
```

Отметим, что ядро Netkit собрано как исполняемый файл для x86-32. В системах x86-64 для его запуска должен быть установлен набор базовых 32-битных библиотек. В современных версиях ОС Debian (7.0 и старше) это делается следующими командами, в более старых можно установить пакет **ia32-libs**.

```
sudo dpkg --add-architecture i386
sudo apt-get update
sudo apt-get install libc6:i386
```

Для автоматического запуска множества виртуальных машин понадобится эмулятор терминала Konsole или Gnome Terminal. В случае использования сред KDE, Gnome или Unity один из этих эмуляторов терминала уже наверняка стоит в системе, иначе его следует поставить.

Для создания отчётов к лабораторным работам понадобится также следующее ПО:

– интерпретатор языка Python 2.7 с библиотеками Netaddr и PyParsing;
– пакет визуализации графов Graphviz;
– система компьютерной вёрстки Latex, например, пакет TeX Live, с пакетом pgf/tikz и стилем предпросмотра;
– система сборки make;
– пакет шрифтов scalable-cyrfonts-tex (не обязателен).

В современных дистрибутивах Debian 7.0 и Ubuntu 12.04 эти пакеты устанавливаются следующими командами.

```
sudo apt-get install make graphviz python2.7 python-netaddr python-pyparsing
sudo apt-get install texlive texlive-lang-cyrillic texlive-latex-extra pgf
sudo apt-get install preview-latex-style scalable-cyrfonts-tex
```

Также нам понадобится любой текстовый редактор, желательно с подсветкой синтаксиса файлов Latex и распространённых файлов конфигураций.

Установка эмулятора Netkit

Откроем терминал и выполним следующие команды для скачивания пакета Netkit (в ходе выполнения этих команд будет скачано около 0.5 Гб данных).

```
mkdir ~/tmp; cd ~/tmp
wget -c http://ftp.iu7.bmstu.ru/netkit/netkit-current.tar.bz2
wget -c http://ftp.iu7.bmstu.ru/netkit/netkit-filesystem-i386-current.tar.bz2
wget -c http://ftp.iu7.bmstu.ru/netkit/netkit-kernel-i386-current.tar.bz2
```

Разархивируем скаченные архивы, после выполнения следующих команд пакет Netkit будет установлен в каталоге **/usr/local/netkit**.

```
cd /usr/local/
sudo tar xvf ~/tmp/netkit-current.tar.bz2
sudo tar xvf ~/tmp/netkit-filesystem-i386-current.tar.bz2
sudo tar xvf ~/tmp/netkit-kernel-i386-current.tar.bz2
```

При необходимости можно удалить скачееыне архивы (**rm ~/tmp/netkit***). Для удаления самого ПО Netkit, в случае необходимости, достаточно удалить его каталог: **sudo rm -rf /usr/local/netkit**.

Для корректной работы ПО Netkit переменная окружения **NETKIT_HOME** должна указывать на расположение пакета Netkit, а в переменные **MANPATH** и **PATH** нужно добавить каталоги с его документацией и исполняемыми файлами соответственно. Для этого допишем в самый конец файла **~/.bashrc** три следующие команды для изменения переменных окружения.

```
export NETKIT_HOME=/usr/local/netkit
export MANPATH=$NETKIT_HOME/man:$MANPATH
export PATH=$NETKIT_HOME/bin:$PATH
```

Это можно сделать в текстовом редакторе или просто выполнив нижеследующие команды.

```
echo 'export NETKIT_HOME=/usr/local/netkit' >> ~/.bashrc
echo 'export MANPATH=:$NETKIT_HOME/man' >> ~/.bashrc
echo 'export PATH=$NETKIT_HOME/bin:$PATH' >> ~/.bashrc
```

Проверка работы системы

Откроем теперь новый терминал, чтобы описанные выше изменения в файле **.bashrc** вступили в силу[1]. Запустим проверку конфигурации пакета Netkit и убедимся, что всё в порядке.

```
cd /usr/local/netkit
./check_configuration.sh
```

Вы можете увидеть предупреждения об отсутствии в системе программы **xterm**, что не имеет значения для пособия. Кроме того, нам достаточно наличия одной из программ **gnome-terminal** и **konsole**.

Выполним теперь команду **vstart**: если вы увидите сообщение о необходимости задать имя виртуальной машины, то наше рабочее место почти готово.

Создадим рабочий каталог (**~/tcp-ip**) и тестовую машину следующими командами.

```
mkdir -p ~/tcp-ip; cd ~/tcp-ip
vstart test
```

Вы увидите запуск виртуальной машины, и через некоторое время в ней появится приглашение командной оболочки Bash. Завершите работу машины, отдав в ней команду **halt**, и дождитесь завершения её работы.

Как мы можем увидеть, после завершения работы в каталоге остался файл образа диска виртуальной машины **test.disk**. Благодаря тому, что технология виртуализации User Mode Linux использует метод копирования при записи, этот образ на диске физически занимает около одного мегабайта, хотя выглядит как занимающий около 10 Гб. В этом вы можете убедиться, выполнив следующие команды.

```
ls -l test.disk
ls -hs test.disk
```

Таким образом, файл образа каждой машины занимает ровно столько места, сколько секторов в нём изменилось по сравнению с оригинальным образом виртуальной машины, находящейся в каталоге Netkit.

Поскольку с точки зрения файловых менеджеров файл образа занимает 10 Гб, то не стоит удалять его перемещением в «корзину». Используйте для файлов образа необратимое удаление или команду **rm**.

Запуск виртуальной сети сценарием ltabstart

Для удобства запуска мы используем сценарий **ltabstart**, который выделит каждой виртуальной машине по отдельной закладке в эмулятора терминала. Проверим его работу на сети из двух машин. Сначала запустим

[1] Также это можно сделать командой `source ~/.bashrc`.

программу Konsole (или Gnome Terminal), затем скачем и разархивируем файл с описанием сети из двух виртуальных машин.

```
mkdir -p ~/tcp-ip; cd ~/tcp-ip
wget -r http://ftp.iu7.bmstu.ru/nets/test.tar.gz
rm -rf test
tar -xvf test.tar.gz
cd test
```

Теперь запустим сценарий **ltabstart**: после параметра **-d** идёт имя директории, в нижеследующей команде это текущая директория.

```
ltabstart -d .
```

В случае успеха откроются две новые вкладки с именами «m1» и «m2», в которых начнут выполнятся одноимённые машины. Остановим машины и удалим файлы их образов дисков следующей командой.

```
lcrash -d .
```

Право на подключение к реальной сети

Выполнение заданий из главы 10 требует подключения виртуальной сети к реальной. Для этого в момент старта виртальной машины, подключаемой к реальной сети, будет запущена программу **manage_tuntap** с правами администратора, для чего используется программа **sudo**. Настоятельно рекомендуется дать тому пользователю, который будет выполнять практические задания, возможность выполнять это действия без ввода пароля. Для этого можно включить в файл **/etc/sudoers** строку следующего вида (в нашем случае разрешение даётся учётной записи **student**, вам следует указать имя того пользователя, который выполняет практические задания).

```
student ALL=NOPASSWD: /usr/local/netkit/bin/manage_tuntap
```

На этом подготовку рабочего места можно считать завершённой. В заключении отметим, что работоспособность примеров в пособии проверялась при выделении каждой виртуальной машине 128 Мб виртуальной памяти. Объём выделенной физической памяти, разумеется, будет зависеть от реальной потребности машины и наличия свободной памяти, но обычно на компьютере с 2 Гб оперативной памяти может нормально работать до полутора десятков виртуальных машин. При необходимости вы можете изменить объём выделяемой машинам виртуальной памяти в файле **/usr/local/netkit/netkit.conf** (параметр **VM_MEMORY**).

Приложение Б Справка по пакету Netkit

Пакет Netkit фактически представляет собой набор утилит для запуска виртуальных машин на основе технологии виртуализации User Mode Linux. Эта технология позволяет запустить достаточно полноценную виртуальную машину на основе ядра Linux как обычный процесс в основной Linux-системе. Такая виртуальная машина имеет собственную файловую систему, список процессов и сетевые виртуальные интерфейсы, при этом на типичном компьютере может быть легко запущено несколько десятков таких машин. Преимуществом данной технологии является возможность быстрого запуска множества машин при довольно скромных объёмах используемой памяти. В пакет Netkit также входит образ виртуальной машины на основе дистрибутива Debian GNU/Linux.

Подробную справку о командах Netkit можно получить из **man**-страниц или их онлайновой версии `http://wiki.netkit.org/man/man7/netkit.7.html`. Ниже приведена информация, достаточная для выполнения заданий в настоящем пособии.

Запуск отдельных виртуальных машин

Для запуска виртуальной машины используется команда **vstart**, основным параметром которой выступает идентификатор машины. При отсутствии виртуальной машины с указанным идентификатором она будет создана. Кроме того, следует указать идентификаторы виртуальных сегментов сети (*доменов коллизий* по терминологии Netkit), подключённых к сетевым интерфейсам виртуальной машины. Каждый сегмент обслуживается одним виртуальным коммутатором.

Следующие команды создадут сеть, показанную на рисунке Б.1 (эллипсы — сегменты сети; прямоугольники — виртуальные машины).

```
vstart vm1 --eth0=net1 --eth1=net2
vstart vm2 --eth0=net2 --eth1=net3
vstart vm3 --eth0=net3 --eth1=net1
```

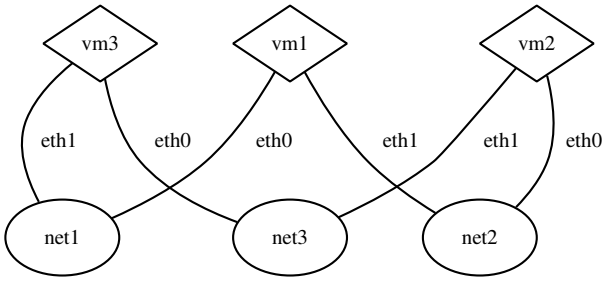


Рисунок Б.1 — Виртуальная сеть из трёх машин

Так как консоль запускаемой машины связывается с текущим терминалом, следует выполнять эти команды в различных терминальных эмуляторах или вкладках.

Параметр **–con0** позволяет сделать это автоматически. При выполнении следующей команды откроется новый терминальный эмулятор.

```
vstart vm1 --con0=xterm --eth0=net1 --eth1=net2
```

Обратите внимание: в виртуальных машинах сразу после загрузки начинается сеанс вы пользователя **root**. Убедиться в том, что сетевые интерфейсы соответствует ожидаемым, можно с помощью команды **ip l**. Дальнейшая процедура присвоения интерфейсам IP-адресов уже не относится к командам Netkit, в главах 2 и 3 показано, как это можно сделать разными способами.

Виртуальной машине можно назначить произвольное число сетевых интерфейсов, но не более 40 (при необходимости можно увеличить параметр **MAX_INTERFACES** в файле **netkit.conf**).

Запуск «лабораторных работ»

Большие виртуальные сети неудобно запускать отдельными командами **vstart**. В этом случае используют заранее подготовленные «лабораторные работы» (англ. *labs*). «Лабораторная работа» Netkit — это каталог, содержащий следующие элементы:

1) файл **lab.conf**, описывающий назначение сетевых интерфейсов машинам (вместо назначения через **–eth0=lan1** в командной строке);

2) набор подкаталогов, по одному на каждую виртуальную машину в составе сети. Их содержимое скопируется в корень файловой системы соответствующей машины при запуске;

3) файл **lab.dep**, описывающий в стиле **Makefile** зависимости между машинами при параллельном запуске;

4) файл **shared.startup** — сценарий языка shell, выполняющийся на каждой машине в момент запуска;

5) файлы **<имя_машины>.startup** — сценарии для индивидуальных машин, выполняющиеся после **shared.startup**.

Отдельные элементы могут отсутствовать: например, **lab.dep** не нужен, если не предполагается возможность параллельного запуска машин, а в сценарии **shared.startup** нет необходимости, если машины не выполняют одинаковых действий при загрузке.

Файл **lab.conf**, соответствующий сети на рисунке Б.1, выглядит следующим образом.

```
vm1[eth0]=net1
vm1[eth1]=net2
```

```
vm2[eth0]=net2
vm2[eth1]=net3

vm3[eth0]=net3
vm3[eth1]=net1
```

Помимо назначения интерфейсов файл **lab.conf** может содержать информацию об авторах «лабораторной работы».

Для запуска такой «лабораторной работы» можно использовать команду **ltabstart**. Её следует вызывать из эмулятора терминала, передавая ей имя каталога с лабораторной работой как параметр (в примере ниже это каталог **net** в текущем каталоге).

```
ltabstart -d net
```

Виртуальные машины будут запущены в отдельных вкладках эмулятора терминала. Этот вариант запуска наиболее удобен. Сценарий **ltabstart** сначала пытается использовать программу Gnome Terminal, а затем, если она не найдена, — программу Konsole. Вы можете также использовать при желании сценарии **gtabstart** (всегда использует Gnome Terminal) и **ktabstart** (всегда использует Konsole).

К сожалению, иногда при использовании программы Konsole сценарий ltabstart не работает (новые вкладки не открываются). Для исправления ситуации обычно следует закрыт все запущенные экземпляры Konsole, запустить Konsole снова и попробовать ещё раз запустить лабораторную работу.

Если по каким-то причинам команда **ltabstart** не работает, воспользуйтесь командой **lstart**, вызвав её следующим образом.

```
lstart -o "--con0=xterm" -d net
```

Машины в составе виртуальной сети будут последовательно запущены в отдельных эмуляторах терминалов. Опция **-o** служит для передачи дополнительных параметров команде **vstart**, запускающей отдельные машины. В данном случае использование **–con0=xterm** приведёт к открытию каждой виртуальной машины в отдельном терминальном эмуляторе (**xterm**, **gnome-terminal** или **konsole** — какой обнаружится на компьютере).

При необходимости можно, конечно, отрывать вкладки эмулятора терминала вручную и запускать виртуальные машины по одной командами **lstart**.

```
cd net
#  (в первой вкладке)
lstart vm1
#  (во второй вкладке)
```

```
lstart vm2
#  (и т.д.)
```

При этом **lstart** всё равно будет использовать настройки из **lab.conf**, сохраняя все свои преимущества над запуском через **vstart**.

Копирование файлов в виртуальную машину

Как было отмечено выше, каталог лабораторной работы может содержать набор подкаталогов, названных по именам виртуальных машин. Возможна, например, следующая структура.

```
net/  <--- каталог "лабораторной работы"
    r1/ <--- каталог файлов машины r1
        etc/ <--- будет скопировано в её каталог /etc
            network/
                interfaces
            resolv.conf
    r2/ <--- каталог файлов машины r2
        ...
    lab.conf
```

После запуска лабораторной работы содержимое каталога **r1** будет автоматически скопировано в корень файловой системы одноименной машины. Это позволяет подготовлять различные конфигурационные файлы заранее, а не редактировать их на самой виртуальной машине после каждого запуска.

Образ файловой системы виртуальной машины хранится в файле **<имя_машины>.disk**. Учтите, что копирование файлов из каталога «лабораторной работы» выполняется только при создании образа её диска, т.е. только при первом запуске машины! Поэтому, если вы изменили на основной машине конфигурационные файлы и хотите, чтобы изменения вступили в силу, то проще всего скопировать эти файлы вручную. Для это в виртуальной машине существует специальный путь **/hostlab**, отображённый в каталог лабораторной работы. Обычно для копирования настроек в виртуальной машине нужно просто выполнить следующую команду, которая скопирует всё содержимое каталога **etc**.

```
cp -r /hostlab/$HOSTNAME/etc /
```

Альтернативным вариантом, разумеется, будет редактирование файлов конфигурации сразу в виртуальной машине. После изменения файлов конфигурации нам, как обычно, понадобится перезапустить соответствующие службы

Сохранение файлов конфигурации виртуальной машины

В силу специфики организации образа виртуальных машин UML их сложно унести из компьютерного зала на съёмном накопителе. Для сохранения в основной машине измененных файлов конфигурации виртуальных машин их рекомендуется сохранить на основной машине через специальный путь **/hostlab**, отображённый в каталог лабораторной работы. Например, для сохранения всех файлов настройки службы маршрутизации Quagga нужно выполнить следующую команду на виртуальной машине.

```
cp -r /etc/quagga /hostlab/$HOSTNAME/etc/
```

Сохранять весь каталог **/etc** обычно нет нужды, но при желании можно сделать и это следующей командой.

```
cp -r /etc /hostlab/$HOSTNAME/
```

После остановки всех виртуальных машин и удаления образов дисков (без перемещения их в корзину) каталог с лабораторной работой можно заархивировать и скопировать на съёмный накопитель или послать по почте. Для архивирования используйте формат с сохранением прав доступа, например **tar.gz**, поскольку в ряде лабораторных работ для подготовки отчётов используются исполняемые сценарии.

Файлы, скопированные описанным образом, при повторном запуске лабораторной работы автоматически будут скопированы в виртуальную машину.

Кроме каталога **/hostlab** в виртуальной машине существует аналогичный каталог **/hosthome**, отображённый в домашний каталог пользователя на основной машине. Оба этих каталога можно использовать для перемещения файлов между реальной и виртуальной машиной по собственному усмотрению.

Остановка виртуальной сети

Отдельные виртуальные машины можно остановить, выполнив в них команду **halt**. Сделать это извне виртуальной машины можно командой **vhalt <имя_машины>**. Если виртуальная машина зависла, следует применить более жёсткий вариант — команду **vcrash <имя_машины>**.

Завершить группу машин в составе «лабораторной работы» можно командами **lhalt** и **lcrash** без параметров, а отдельную машину — указав её имя в качестве параметра. Внимание: команда **lcrash** удаляет образ диска машины (файл с расширением **.disk**).

Если аварийно завершённая виртуальная машина зависает при повторном запуске, попробуйте удалить директорию **/.netkit**. Стоит также убедиться, что образ файловой системы этой машины (файл с расширением **.disk**) удалён.

В силу специфики реализации файла образа диска виртуальной машины удалять его либо командой **rm**, либо без перемещения её в корзину (типичное клавиатурное сочетание в графическом файловом менеджере — Shift+Del).

Приложение В Справка по перехвату сетевого трафика

Перехват UDP-трафика с полным анализом пакета IPv4 (полезно для сообщений RIP).

```
tcpdump -tn -i eth0 -s 1518 udp
```

Приложение Г Список практических заданий

1) Запуск виртуальных машин командой vstart (стр. 36).
2) Получение сведений о сетевых интерфейсах (стр. 38).
3) Получение сведений об IP-адресах интерфейсов (стр. 40).
4) Использование утилиты ping (стр. 40).
5) Назначение интерфейсу IP-адреса и маски сети (стр. 41).
6) Вывод кеша протокола ARP (стр. 43).
7) Перехват сообщений протокола ARP (стр. 43).
8) Вывод MAC-адресов при перехвате сетевого трафика (стр. 45).
9) Отправка пакета на отсутствующий в сети IP-адрес (стр. 46).
10) Вывод таблицы маршрутизации (стр. 47).
11) Случай отсутствия адреса в таблице маршрутизации (стр. 47).
12) Остановка виртуальных машин (стр. 48).
13) Запуск сети командой ltabstart (стр. 52).
14) Настройка адресов рабочих станций (стр. 54).
15) Настройка адресов маршрутизаторов (стр. 55).
16) Установка маршрута по умолчанию (стр. 55).
17) Проверка маршрута по умолчанию (стр. 56).
18) Добавление статических маршрутов (стр. 57).
19) Наблюдение за косвенной маршрутизацией (стр. 59).
20) Создание маршрутной петли (стр. 61).
21) Истечение времени жизни пакета (стр. 61).
22) Построение списка маршрутизаторов на пути (стр. 63).
23) Изменение величины MTU сегмента сети (стр. 65).
24) Отключение механизма PMTU (стр. 66).
25) Фрагментация IP-пакетов маршрутизатором (стр. 66).
26) Механизм PMTU (стр. 67).
27) Отсутствие получателя в таблице маршрутизации (стр. 69).
28) Отсутствие получателя в сегменте сети (стр. 69).
29) Работа кеширующего DNS-сервера (стр. 104).
30) Подготовка сети для экспериментов с протоколом UDP (стр. 108).
31) Широковещательный UDP-ретранслятор (стр. 109).
32) Запуск сети и проверка настроек IP-адресов (стр. 112).
33) Настройка службы динамической маршрутизации (стр. 115).
34) Таблица протокола RIP (стр. 117).
35) Перехват сообщений протокола RIP (стр. 120).
36) Протокол RIP. Использование правила испорченного обратного маршрута (стр. 120).
37) Протокол RIP. Отключение правила расщеплённого горизонта (стр. 121).
38) Протокол RIP. Устаревшие маршрута (стр. 122).
39) Протокол RIP. Сборка мусора (стр. 122).

40) Программа получения сообщений протокола RIP (стр. 124).
41) Проверка перехвата сообщений протокола RIP (стр. 125).
42) Настройка DHCP-службы (стр. 158).
43) Привязка IP-адреса к MAC-адресу клиента (стр. 158).
44) Проверка почтового релея (стр. 187).

Приложение Д Предметный указатель