В. А. Крищенко, А. О. Крючков

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

Учебное пособие

Черновик от 12 сентября 2013 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.

Содержание

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 виртуализаторов общего назначения затруднено из-за высоких требований к аппаратным средствам, необходимых для запуска десятка 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.

Данная технология называется User Mode Linux; для неё разработаны достаточно удобные средства управления виртуальными машинами и быстрого создания виртуальных сетей на их основе. 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 технологии» (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 **??** 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 **??** рассматривает канальный уровень и проблему построения его логической топологии с помощью протокола STP. В главе **??** рассмотрены основы транспортного уровня и протокол UDP. В главе **??** описано использование динамической маршрутизации на основе протокола RIP. Глава **??** посвящена основам протокола

и использованию сетевых сокетов, а глава **??** рассматривает подробно работу протокола TCP. В главе **??** рассмотрены вопросы работы локальных сетей, включая трансляцию сетевых адресов, фильтрацию сетевого трафика и создание виртуальных частных сетей. В главе **??** рассмотрена организация системы DNS, а в главе **??** — организация почтовой системы на основе протокола SMTP. Глава **??** рассматривает основы протокола HTTP.

Все главы заканчиваются вопросами для контроля изученного материала. В главах 3, **??**, **??**, **??** и **??** 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, 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 разных 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-сервер 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.

Физики ещё в XIX веке открыли несколько способов передачи информации, которые могут использоваться в компьютерной сети.

Сигнал — физический процесс, несущий информацию. 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 его область 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 типов 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, что спецификация программного интерфейса, через который реализация одного протокола взаимодействует с реализацией другого, не входит в спецификация протокола: в спецификации лишь указывается, какую информацию должен получить протокол «сверху». Поскольку две вполне соответствующие одинаковому спецификаций реализации сетевого стека могут корректно взаимодействовать даже при разных межуровневых интерфейсах, то их включение спецификацию протокола даже вредно. В главах **??** и **??** 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 **??** 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 **??** 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).

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

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

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

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

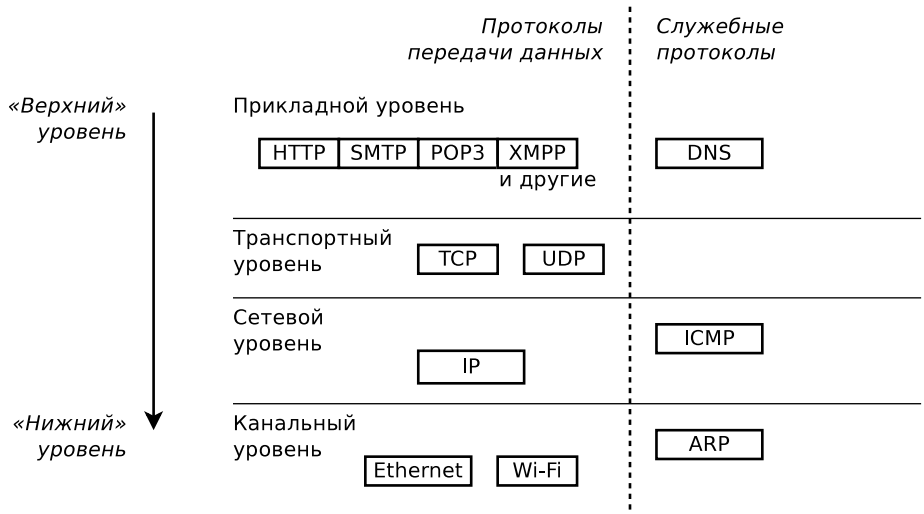


Рисунок 1.1 — Уровни и основные протоколы стека TCP/IP

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. Все протоколы стека TCP/IP (рисунок 1.1) можно разделить на две группы:

– *протоколы передачи данных*, передающие полезные данные между двумя сторонами;

– *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 терми-

нологии, поэтому в нём легче ответить на вопросы «для его 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 канальный 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 **??** 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-отправителем может быть, например, всё тот же веб-сервер, отправляющий HTTP-сообщение с веб-страницей внутри, а процессом-получателем — веб-браузер.

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

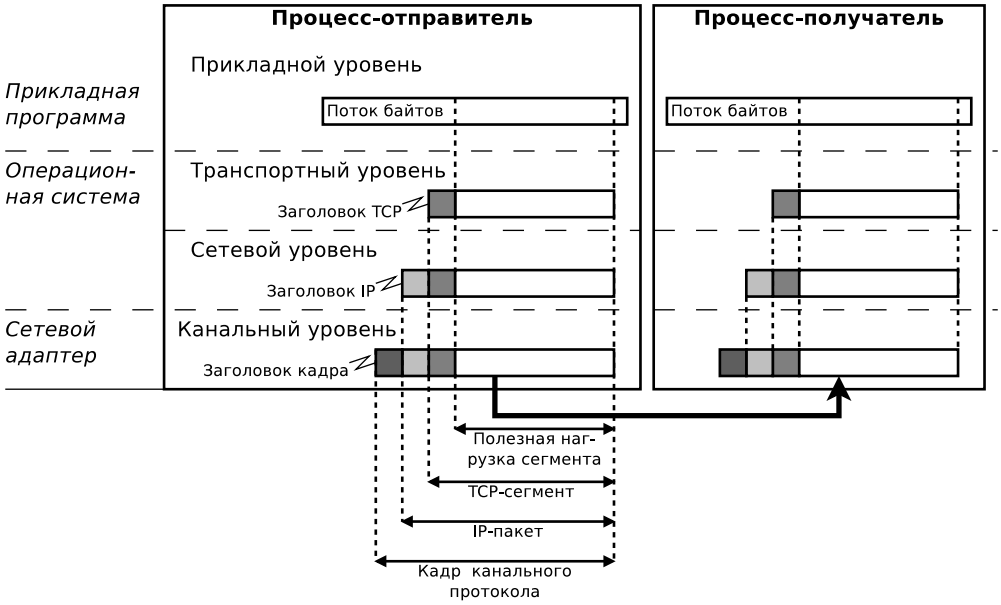


Рисунок 1.2 — Передача данных между процессами

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, относительно стороны отправителя, порядке, то стек сетевых протоколов действительно напоминает стек.

Здесь, конечно, мы очень упрощённо описали отправку информации HTTP-сервером. В главе **??** мы познакомимся с тем, как использующие TCP прикладные программы передают сообщения прикладного протокола операционной системе, а в главе **??** 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 двумя ЭВМ, соединённых с 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.

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 **??** мы увидим создание виртуального сегмента сети. Поэтому, с точки зрения вышестоящего сетевого уровня, определение сегмента сети не связано уже с разделяемой физической средой, и нам ничего не остаётся, как связать в единой целое понятия канального уровня и сегмента сети.

Сегмент сети — совокупность машин, которые могут передать друг другу сообщение канального уровня. 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.

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

В пределах сегмента (и только в них) 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 отправляемая 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) 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 **??** 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.

Для подсоединения компьютера к среде передачи используется устройство, называемое *сетевым адаптером* или сетевой картой. 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 сетевых портов стандарта Ethernet. Порт является точкой подключения сетевого кабеля к сетевому оборудованию канального уровня, в отличии от сетевого адаптера.

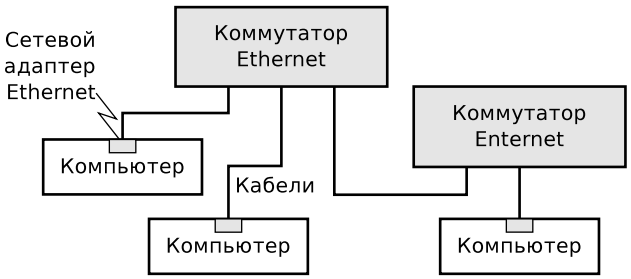


Рисунок 1.3 — Соединение коммутаторов и сетевых адаптеров в один сетевой сегмент

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 **??** мы разберёмся, как же коммутаторы связывают MAC-адреса получателей со своими портами.

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*.

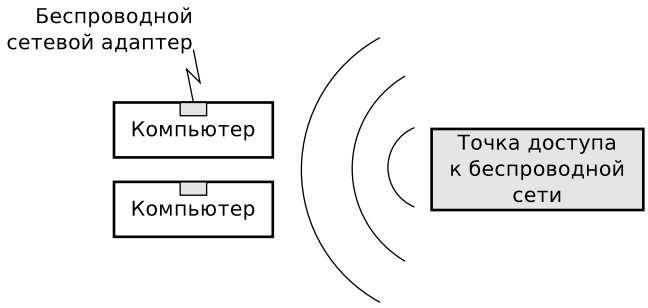


Рисунок 1.4 — Сегмент беспроводной сети

Мост — bridge.устройство, объединяющее несколько сетевых сегментов в один сегмент

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.

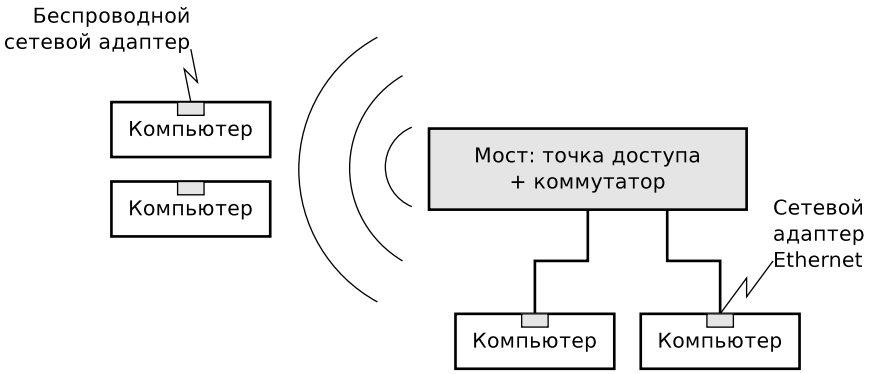


Рисунок 1.5 — Объединение в один сегмент проводной и беспроводной сети

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 **??**.

Поскольку к сегменту сети может быть подключено несколько компьютеров с сетевыми адаптерами, то для идентификации сетевого адаптера в пределах сегмента нужно использовать некоторые адреса. Исключением являются протоколы канального уровня, служащие для соединения ровно двух адаптеров в пределах сегмента: 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). Адреса всех сетевых адаптеров в пределах сегмента должны иметь уникальные MAC-адреса. MAC-адрес состоит из шести байт, которые принято записывать в шестнадцатеричной форме через двоеточие, например: aa:bb:c0:56:78:90. Отметим, что порты коммутаторов не имеют MAC-адресов — в этом нет нужды, поскольку коммутатор не является конечным получателем пакетов, а только пересылает их дальше.

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. 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.

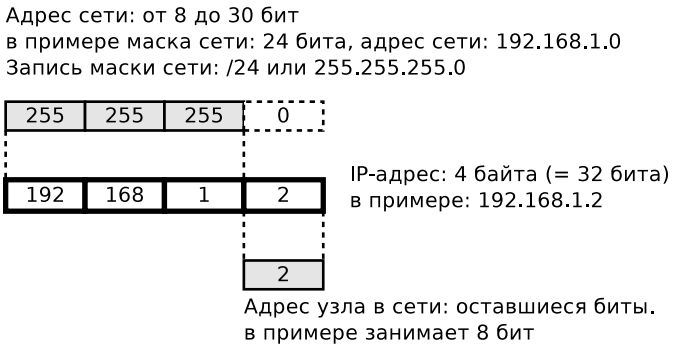


Рисунок 1.6 — Структура IP-адреса

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

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

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 конкретный не 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 — router.компьютер или устройство с двумя или более сетевыми адаптерами с IP-адресами в разных 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.

На пути от отправителя до получателя IP-пакета может находится множество маршрутизаторов и сегментов. Один шаг передачи IP-пакета по этому пути мы назовём *маршрутизацией*.

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

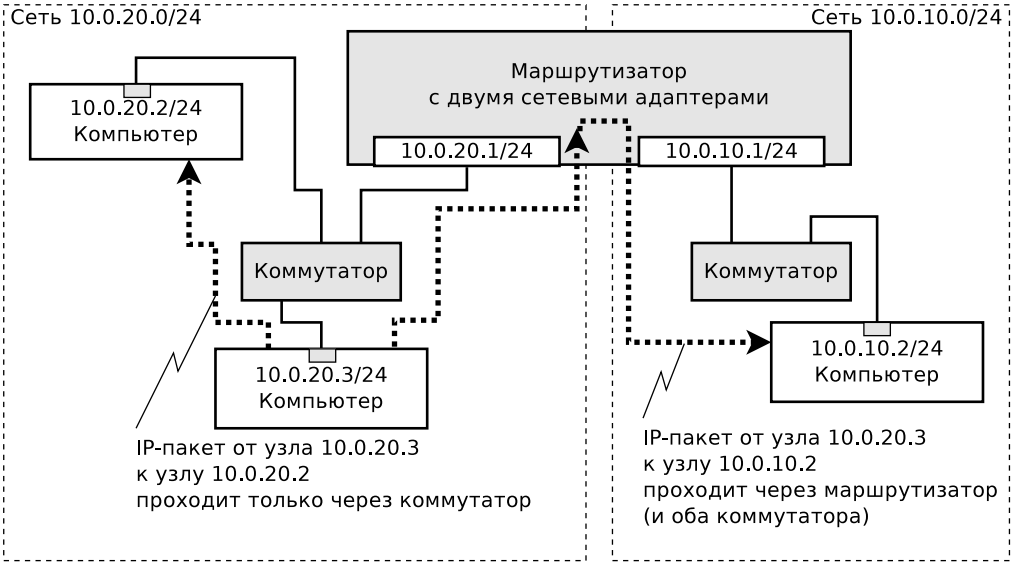


Рисунок 1.7 — Примеры пути IP-пакета

Маршрутизацией, по данному определению, занимаются все машины, поддерживающие протокол IP. Далее мы будем для краткости называть маршрутизацию IP-пакетов

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

просто «маршрутизацией». На пути от отправителя к получателю маршрутизацией занимается отправитель и все маршрутизаторы, через которые проходит пакет.

Можно выделить два разных случая маршрутизации, иного 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].

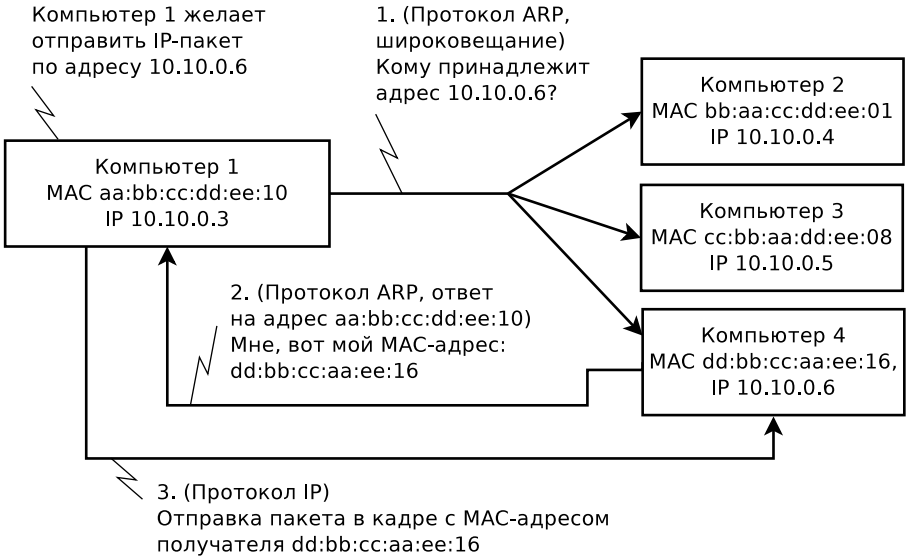


Рисунок 1.8 — Использование протокола ARP для поиска MAC-адреса

Для получения MAC-адреса по известному IP-адресу в пределах сегмента сети используется протокол разрешения адресов ARP [4] (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 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 **??**.

Как было отмечено ранее, величина MTU сегмента сети определяет максимальный размер IP-пакета, который может быть передан поверх канального уровня. Что же

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

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 **??**) уровней мы увидим и другие решения этой же проблемы, поскольку фрагментация имеет ряд крупных недостатков.

Отметим, что некоторые источники рассматривают 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*.

Порт транспортного уровня 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, 89 определяет лишь значение 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[1]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.

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-

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

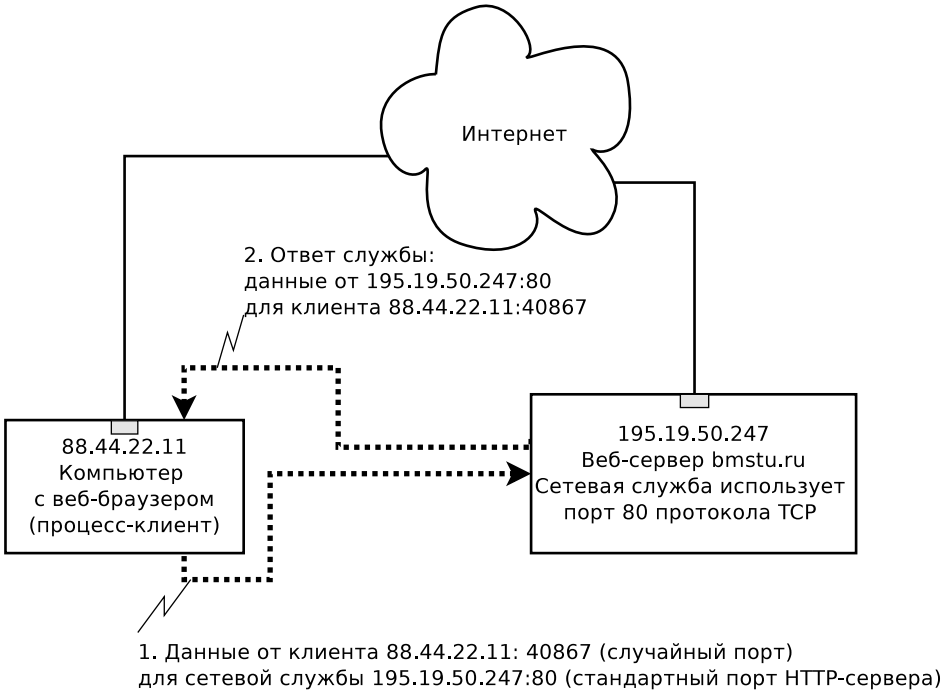


Рисунок 1.9 — Сетевая служба и её клиент

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 **??**.

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 **??** 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 **??** 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.

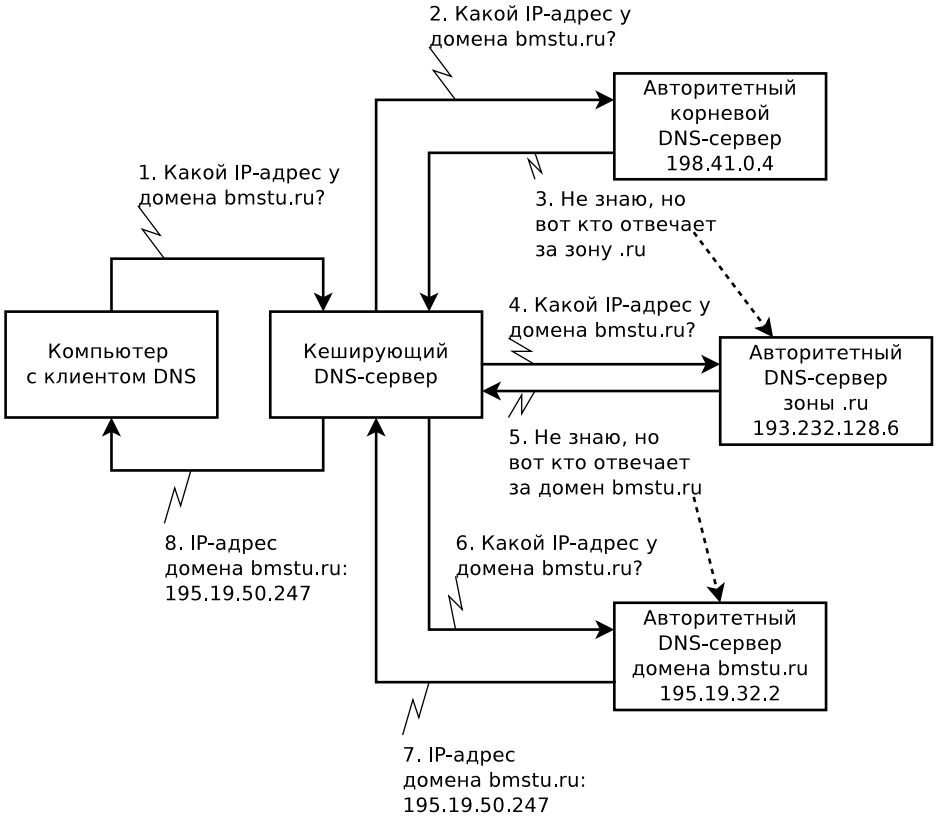


Рисунок 1.10 — Работа служба доменных имён

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.

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 **??**.

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 **??** мы рассмотрим эти протоколы боле подробно.

Протокол SMTP используется для передачи электронной почты от почтового клиента к почтовому серверу и от одного почтового сервера к другому. В главе **??** 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 и IP-адрес, подключённые к локальной сети, принято называть *внутренними*, а подключёнными 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 работе 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 изме-

ряется 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.

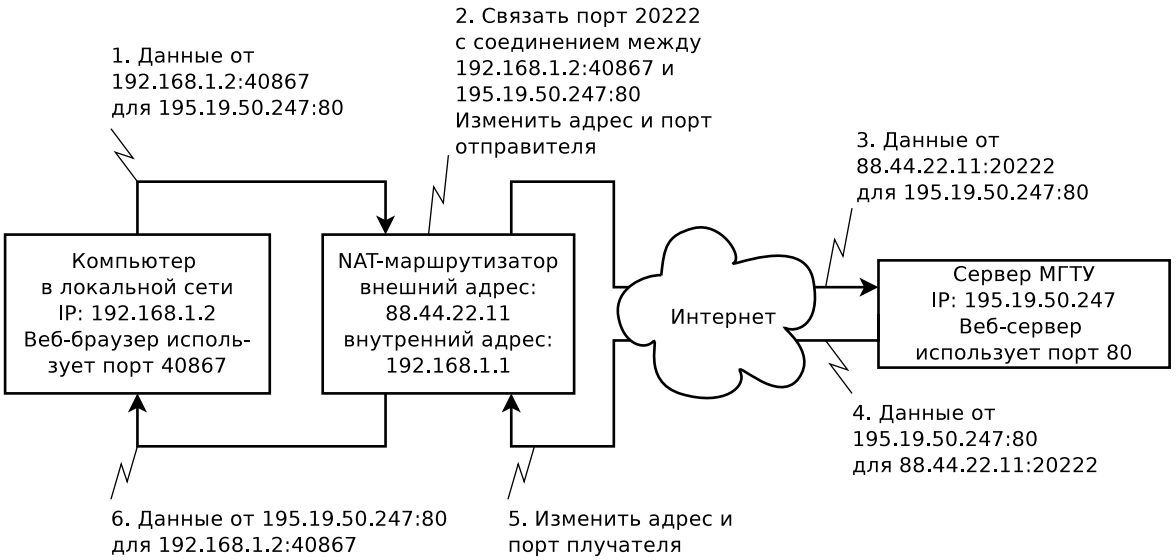


Рисунок 1.11 — Трансляция сетевых адресов

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 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 Ethernet кабель и 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);
– IP-адрес кеширующего DNS-сервера, в нашем случае он совпадает с адресом маршрутизатора.

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 ра-

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

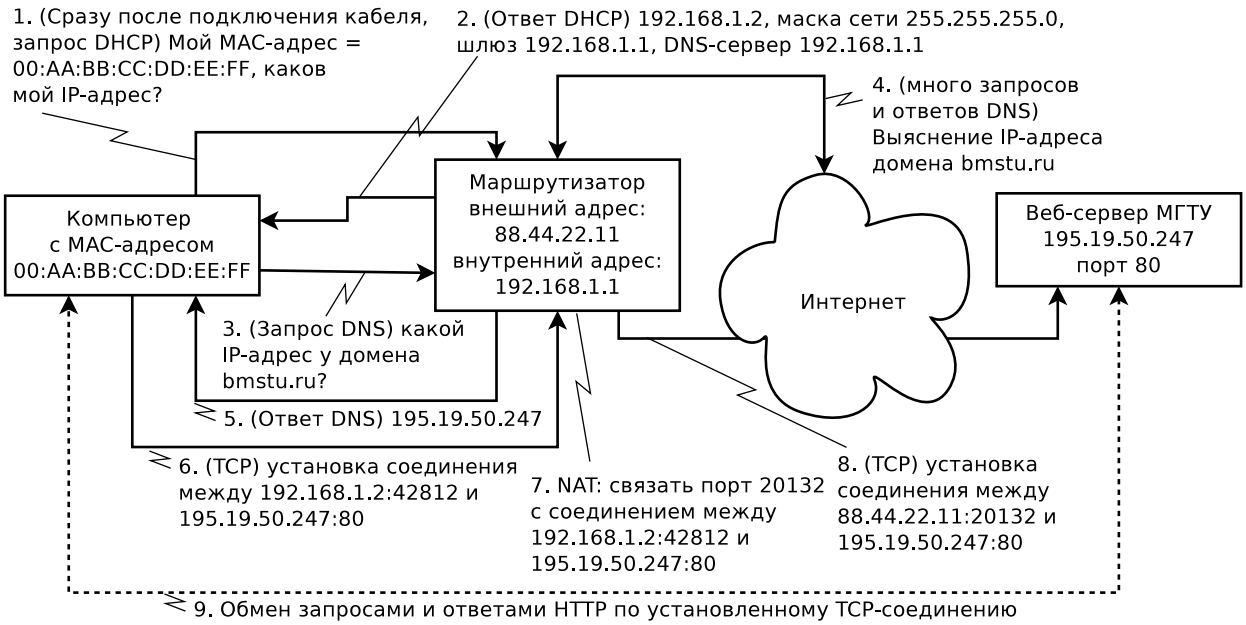


Рисунок 1.12 — Домашняя локальная сеть

ботает поверх протокола TCP использует 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. Заметьте, до этого момента через маршрутизатор не проходили такие транзитные пакеты, поэтому механизм NAT не был задействован.

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» [5].

Попытка создания сетевого стека была неудачной ([6], 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 както связать со стеком TCP/IP. Для понимания его ошибочности (отмеченной, например, в [6], стр. 130) достаточно изучить модель OSI/ISO подробнее и убедиться, что в ней нет места, например, простому транспортному протоколу типа UDP, поскольку в итоге сте-

ке 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 [7]. 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 [7], где даже не прямого упоминания проекта ARPANET и протоколов стека TCP/IP. В итоге сетевой стек ISO получился мертворожденным, но организация по стандартизации ISO так и не признала стек TCP/IP стандартом «де-юре». Это, 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, содержащие спецификации протоколов стека TCP/IP, стандарты протоколов OSI/ISO даже для ознакомления с ними необходимо приобретать за довольно значительную цену.

1.12 Контрольные вопросы

Для самоконтроля полученных знаний рекомендуется ответить на следующие вопросы.

1) Что такое протокол, сообщение протокола, стек протоколов?
2) Что такое сегмент сети? Для чего служат протоколы канального уровня?
3) В чем различия между MAC-адресом и IP-адресом?
4) Какие задачи решают протоколы Ethernet, ARP, IP?
5) Какие сообщения называют кадром, пакетом, сегментом?
6) Что такое маска сети и для чего она служит?
7) Перечислите все упомянутые в этой главе протоколы передачи данных и все служебные протоколы.
8) Какие основные различия между протоколами UDP и TCP?
9) Для чего служит протокол DNS и DNS-сервер?
10) Зачем были выделены диапазоны частных IP-адресов?
11) Какую проблему решает трансляция сетевых адресов?
12) Почему компьютеры пользователей обращаются к кеширующему DNS-серверу, а не обращаются напрямую к авторитетным серверам имён?

2 Простейшая сеть на основе протокола IP

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.

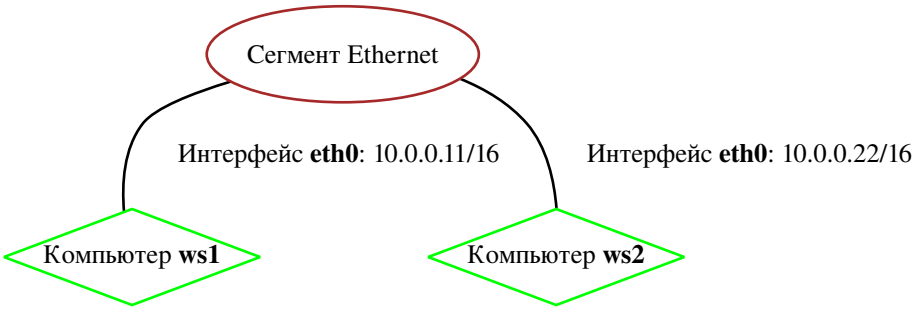


Рисунок 2.1 — Компьютерные сеть из двух машин, подключенных к одному сегменту

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 не обязательно: 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.

Сетевой интерфейс — это абстрактное сетевое устройство, за работу которого отвечает ядро операционной системы. 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.

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.

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.

```
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 терминале (т. е. в другой вкладке или в другом окне эмулятора терминала) запустите виртуальную машину **ws2**.

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

Как видно по приведенным выше командам, каждая машина будет иметь единственный сетевой интерфейс с именем **eth0**. Оба интерфейса 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 обоих машин завершится выводом приглашения (**ws1:~#**), и мы сможем работать с их интепретатором команд. 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 его главным недостатком. Таблица 2.1 описывает назначение и соответствие утилит данных пакетов.

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** «просит» 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.

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 стека TCP/IP процессами ОС	netstat	ss

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 машинах 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 обоих 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 интерфейсе. 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.

Интерефейс **eth0** связан с сетевым адаптертом в реальных машинах. В случае нашей виртуальной машины он связан с виртуальным адаптером, поделюченным к вируталь-ному сегменту, которые соединяет две запущенные виртуальные машины друг с другом.

2.4 Адреса протокола IPv4

Протокол сетевого уровня IPv4 (англ. *Internet Protocol, версия 4* [8]) позволяет передавать данные между компьютерами, находящихся 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 количество бит с начала IP-адреса (то, есть, со старшего байта левого октета), который отводятся под адрес сети. Маска может записываться, либо как число этих битов (например, /24), либо как четыре октета, где соответствующие маске биты равны единицы, а остальные — нулю (например, 255.255.255.0).

Допустим, что у нас есть некоторый IP-адрес и его маска сети, и мы хотим проверить, принадлежит ли некоторый другой IP-адрес этой сети. Если результат побитовой операции «И» длинного представления маски адреса и самого адреса равен результату побитового «И» маски и другого адреса, 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 определяалась 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 задавалсь 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. 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 команду **ip addr** (сокращённо — **ip a**) для вывода информации о IP-адресах, присвоенных сетевым интерфейсам. 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 [9] была выделена сеть класса «A» — явно расточительное решение, учитывая дальнейшее исчерпание множества адресов протокола IPv4. Локальному сетеавму 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 [10], 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** обоих 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») интерфейс, сделав возможной передачу данных через него.

```
ip addr add 10.0.0.11/16 dev eth0
ip link set eth0 up
```

Выполните аналогичные команды на машине **ws2**, не забыв сменить в первой 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** и получить 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.

Отображение IP $\rightarrow$ 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 устаревший спустя некоторое время после нахождения MAC-адреса. 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.

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 на сетевую карту машину или посылки кадров через неё, в момент их появления она выведет на экран их содержимое в расшифрованном и достаточно удобном виде. В приложении **??** приведена дополнительная справка по всем нужным нам параметрам этой программы перехвата сетевого трафика.

[1] От англ. *neighbour* — «сосед».

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.

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 запросов передаваемых.

```
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 могут запущено одновременно несколько программ **ping**, опрашивающих один и тот же адрес.

После цепочки сообщений ICMP обычно видна пара из запроса и ответа ARP, в которым машина **ws2** запрашивает MAC-адрес по IP-адресу машины **ws1**. Связано это с тем, что операционная система регулярно проверяет записи в кеше ARP. Если пакеты с данного MAC и IP адреса не приходят в течении некоторого времени (порядка нескольких секунд), то система отправляет ARP-запрос, желая проверить актуальность записи в кеше. Отметим, что если запись в кеше не используется в течении длительного времени (около минуты), то она удаляется даже в случае актуальности 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 про-

грамму: 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 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, 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 одного из видов групповой рассылки, один из которых мы 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 проведеном опыте видно, что проверка актульности кеша ARP не использует широковещательную рассылку — кадр отправляется на MAC из кеша. Это решение позволяет сократить широковещательный сетевой трафик. Если ответ не будет получен и актульность 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 — милисекунды. Разумеется, эта время определяется весьма приблизительно, с точностью примерно до сотых долей секунды. Наши виртуальные машины по-умолчанию показывают время по Гривиничу, что нам не должно волновать, поскольку далее нам только важны 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 сеетаыому 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. С целью компактности 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 может определить, какому маршрутизатору и через какой сетевой интерфейс следует переслать IP-пакет. В строке таблицы адрес маршрутизатора также может быть пустым, 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.

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).

```
10.0.0.0/24 dev eth0 proto kernel scope link src 10.0.0.11
```

Как мы видим, в таблице маршрутизации приведена единственная строка для интерфейса **eth0**. Локальный интерфейс **lo** должен быть всегда изолирован от всех остальных — иначе он не будет локальным — поэтому нет смысла включать его сеть в таблицу маршрутизации, хотя некоторые ОС так поступают.

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 **??** мы увидим и строки, созданные по указанию сетевых службами по обмену маршрутной информацией.

Задание № 11: Случай отсутствие 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**? Почему?

6) Для чего служит протокол ARP?

7) С каким интервалом и сколько раз наша машина повторяет попытки найти MAC-адреса получателя при посылке одного IP-пакета?

8) На какой MAC-адрес отправляется запрос 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 данном 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.

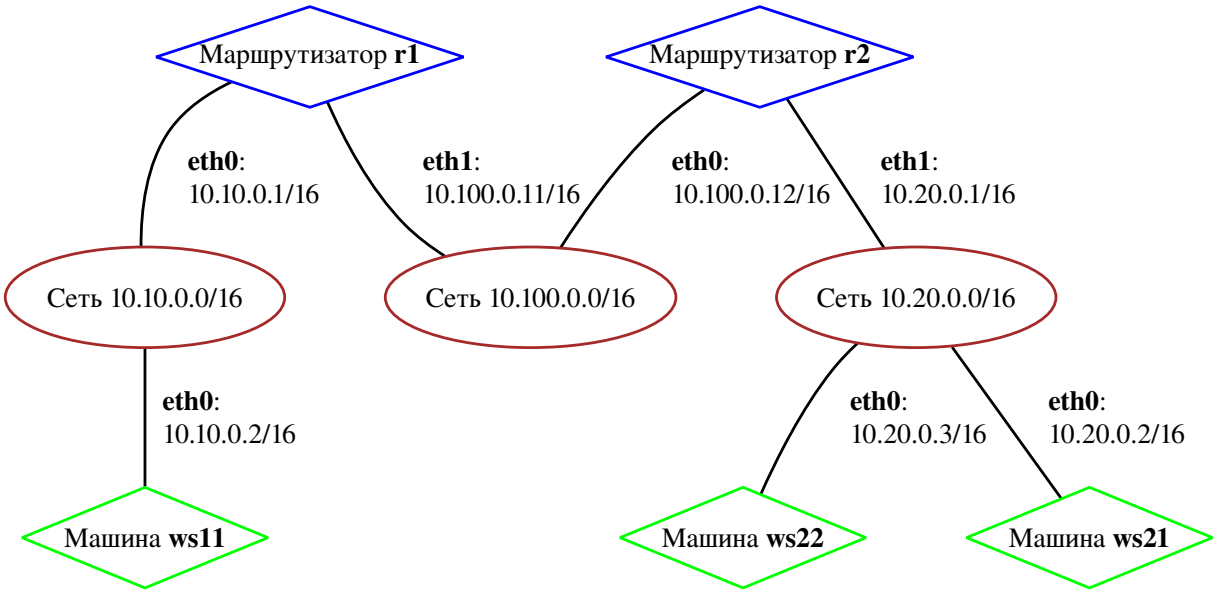


Рисунок 3.1 — Компьютерная сеть из соединённых двумя маршрутизаторами IP-сетей

Мы познакомимся с процессом IP-маршрутизации на основе статических маршрутов 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 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 3.13) приведены возможные индивидуальные варианты заданий. В разделе 3.12 даны дополнительные указания по их выполнению.

3.1 Структура IP-пакета

Перед выполнением практических заданий рассмотрим структуру сообщения протокола IP — IP пакета. IP-пакет является наименьшей единицей информационного 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.

Опишем кратко присутствующие в заголовке IP-пакета поля. Поле **version** содержит номер версии протокола, для IPv4 это число 4. Поле **protocol** идентифицирует протокол транспортного уровня, данные которого вложены в IP-пакет.

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

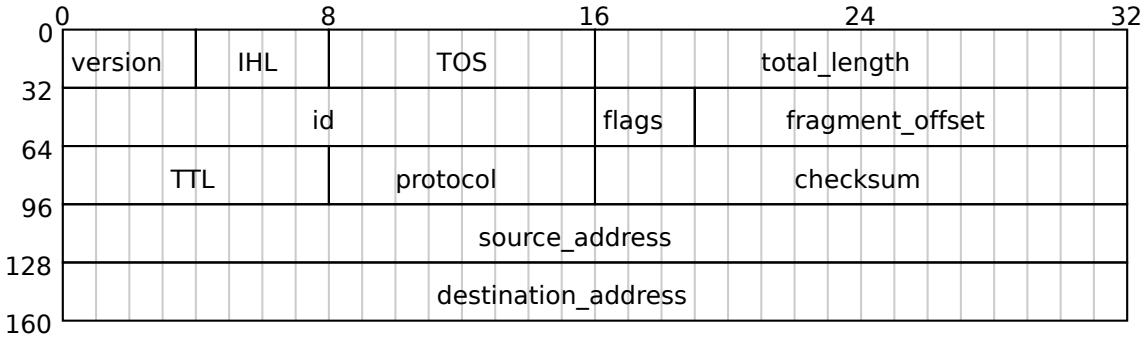


Рисунок 3.2 — Заголовок пакета IPv4 (размеры полей указаны в битах)

образом, максимальный теоретический размер пакета IPv4 ограничен 65535 байтами, а на практике размеры пакетов значительно меньше из-за ограничений канального уровня.

Последние, по порядку следования, поля заголовка — **source_address**, **destination_address** — содержат IP-адреса отправителя и получателя соответственно.

Поле **TOS** (англ. *Type of service*) задает требования к качеству передачи — низкие задержки, или низкие потери — которыми могут руководствоваться маршрутизаторы при выборе маршрута для пакета. На маршрутизатор сам решает, что ему делать с пакетом, и может как принимать во внимание это поле, так и не делать этого.

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*).

Поле TTL содержит оставшееся «время жизни» пакета, обычно измеряемое в числе прыжков между маршрутизаторами. Каждый маршрутизатор уменьшает значение TTL пакета на единицу и отбрасывает пакет, когда оно достигает нуля. Поле **checksum** содержит контрольную сумму заголовка пакета (тело пакета не участвует в расчете суммы).

3.2 Запуск машин и настройка сетевых интерфейсов

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

> **Задание № 13: Запуск сети командой ltabstart**
>
> Скачайте и распакуйте архив с этими файлами следующими командами, выполненными в основной машине.

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

Теперь нам нужно запустить виртуальные машины. Это можно сделать следующей командой, которая автоматически запустит виртуальные машины в различных вкладках эмулятора терминала. Последний параметр (после **-d**) задает путь к содержимому архива с преднастроенной сетью; точка, как известно, означает текущий каталог.

```
ltabstart -d net
```

В предыдущей работе мы уже использовали для настроки IP-адреса «на лету» команду **ip**. Эта настройка сбрасывалась при перезагрузке машины, поэтому теперь мы будем использовать для настроек параметры сети файлы конфигурации. К сожалению, соглашения о расположении и формате этих файлов не стандартизированы, в силу чего дальнейшая информация о них распространяется только на Debian и основанные на нем дистрибутивы GNU/Linux.

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

```
# Интерфейс lo служит только связи компьютера
# с самим собой (тип интерфейса: loopback)
auto lo
iface lo inet loopback
# Далее идёт описание сетевого интерфейса eth0.
auto eth0
# Ему будет статически 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

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

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 **/etc/network/interfaces** на всех машинах в соотвествии с рисунком 3.1. После изменения файла **interfaces** здесь и в дальнейшем всегда нужно перезагрузить настройки сети следующей командой.

```
service networking restart
```

Прочитайте вывод команды выше — возможно, при редактировании файла были допущены ошибки. Если это так, исправьте их и повторите попытку, если всё хорошо — проверьте командой **ip 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, как видно, мы будет использовать префикс «r»). Конфигурация каждого маршрутизатора в нашей 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
    address 10.20.0.1
    netmask 255.255.0.0
```

Задание № 15: Настройка адресов маршрутизаторов

Отредактируйте на обоих маршрутизаторах файлы **interfaces** и затем перегрузите 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 (строки в маршрутной таблице часто называются *маршрутами*).

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 **ip r** теперь должна показать маршрут по умолчанию через маршрутизатор 10.10.0.1.

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. Убедитесь, что эхо-запрос с **ws11** доходит до **r2**: запуск программы перехвата трафика на последнем поможет в этом.

```
tcpdump -tn -i eth0
```

Поскольку у маршрутизатора два сетевых интерфейса, выше явно указан интерфейс для перехвата трафика после ключа **-i**. Отдав необходимый эхо запрос, вы увидите, что пакеты с ним дошли до маршрутизатора, но он не может послать ответ на них, поскольку не знает маршрут до сети отправителя.

Добавьте аналогичным образом маршрут по-умолчанию для машин **ws21** и **ws22**.

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»).

Задание № 17: Добавление статических маршрутов

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

```
# Добавить в конец описания 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 ничего лучше в ОС Debian не предусмотрено.

Проверьте маршрутную таблицы маршрутизатора (команда **ip r**) после перезагрузки сетевых настроек. При необходимости исправьте ошибки и повторно перегрузите сетевые настройки. В случае некоторых ошибок иногда может быть проще перегрузить всю виртуальную машину.

После завершения настройки маршрутизатора **r1** можно переходить ко второму маршрутизатору. Для маршрутизатора **r2** в командах добавления маршрута надо указать сеть, находящуюся «за» **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**.

Отметим, что и в настройках маршрутизатора ничто не мешает использовать параметр **gateway**, когда путь во все сети, кроме некоторых, проходит через один маршрутизатор. В некоторых вариантах самостоятельных заданий его использование существенно облегчает настройку маршрутизаторов: этот параметр нужно использовать в настройках того интерфейса, который является соседом указываемого маршрутизатора по-умолчанию. Поскольку наша компьютерная сеть очень проста, то мы могли бы и в ней просто использовать другой маршрутизатор в качестве маршрута по умолчанию, но это не позволило бы показать использование статических маршрутов, которые понадобятся в самостоятельной работе.

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.

Далее мы будем под *путём* понимать последовательность маршрутизаторов, которые некоторый IP-пакет проходит пр следовании 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.

3.5 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 маршрутизации. Сейчас мы увидим, как двигается пакет, если адресат находится в другом сегменте.

Задание № 18: Косвенная маршрутизация

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 (в ней *neigh* — это сокращение от *neighbors*, то есть соседи по сегменту сети).

```
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 трафик, 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
```

На первом шаге машина-отправитель определяет по своей маршрутной таблице, что пакет нужно отправить на адрес 10.10.0.1. Теперь ей нужно получить соотвествующий 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**)

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*.

Теперь давайте выясним, что происходит с этим пакетом при прохождении маршрутизатора. В перехваченном на машине **r2** мы видим, что отправляет в сеть маршрутизатор **r1** через интерфейс **eth1**. По своей таблице маршрутизации **r1** определил, что MAC-адресом получателя должен соответствовать адресу 10.100.0.12. Таким образом, кадр отправляется с MAC-адреса интерфейса **eth1** машины **r1** на **eth0** машины **r2**.

На последнем шаге маршрута получатель кадра (MAC-адрес **ws21**) соответствует получателю IP-пакета — используется *непосредственная маршрутизация*. В этом можно убедится, просмотрев на перехваченные на **ws21** пакеты.

3.6 Время жизни IP-пакета

Как видно из перехваченной информации, в самом IP-пакете при прохождении его через маршрутизатор изменяется поле TTL (англ. *Time to Live* — время жизни), 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 при разовой ошибке с указанием адреса следующего маршрутизатора.

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

Задание № 19: 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 (как можно заметить, мы указали неверный маршрутизатор в качестве следующего на пути пакета).

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

Убедимся командой **ip r**, что маршрутизатор **r2** будет пересылать пакеты с получателями в сети 10.20.0.0/16 маршрутизатору **r1**. «Правильный» вывод этой команды приведен ниже.

```
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
```

Запустим на этом же маршрутизаторе программу перехвата сетевого трафика.

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

На маршрутизаторе **r1** запустим аналогичную команду, но с указанием перехватывать пакеты на интерфейсе **eth1**. С машины **ws11** пошлём эхо-запрос на адрес в «завернутой» сети.

```
ping 10.20.0.2 -c 1
```

Мы увидим, как IP-пакет с этим запросом будет многократно проходить через оба маршрутизатора, со всё уменьшающимся значением поля **TTL**.

В итоге маршрутизатор **r2** отправит отправителю ICMP-сообщение, извещающее об истечении времени его существования.

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 **r1** и **r2**. По завершении опыта перезагрузим маршрутизатор **r2** командой **reboot** для того, чтобы вернуть его в правильное состояние.

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

3.7 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 № 20: Построение списка маршрутизаторов на пути

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

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

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

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
```

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. Пример вывода приведён ниже.

```
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
```

Часто первая из трёх попыток занимает больше времени, чем две предыдущие (в приведенном примере это видно в третьей строке). Это связано с тем, что на данном этапе маршрутизации потребовалось использовать сначала протокол ARP.

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

3.8 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 (дефрагментации) в единое целое в заголовке есть два поля: **id** и **offset**, а также флаг **MF**.

Задание № 21: Изменение величины MTU

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», например 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).

Вначале мы проведем опыт, показывающий IP-фрагментацию в том виде, как она предполагалась первоначальным стандартом [8]. 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 — его мы

рассмотрим позднее. Если этого не сделать, то отправляющая слишком большой пакет машина получит ICMP-сообщение о необходимости фрагментации, а пакет будет уничтожен.

Задание № 22: Отключение механизма PMTU

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

Отключим на машине **ws1** механизм борьбы с фрагментацией следующей командой (она показывает, как можно менять параметры ядра Linux),

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

Теперь, как можно убедиться повторной посылкой эхо-запроса, IP-пакет отправляется без флага **DF**.

Борьба с фрагментацией теперь отключена и если сейчас с машины **ws1** послать машине **ws2** один ICMP-пакет с данными длиной 1000 байт, то до машрутизатора **r1** он дойдёт как один IP-пакет, а до **r2** и **ws2** — как два IP-пакета. Отметим, что PMTU не является единственным способом борьбы с фрагментацией, как мы увидим в главе **??**.

Задание № 23: Фрагментация IP-пакетов

Начнём на маршрутизаторах **r1** и **r2**, а также машине **ws21** перехват 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 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)
```

Здесь мы наблюдаем неочевидную 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.

3.9 Определение величины MTU для пути

Фрагментация крайне вредна по целому ряду причин, которые будут более подробно здесь позже. Пронаблюдаем теперь, как работает механизм нахождение минимального значения MTU для всего пути, известный как PMTU.

Задание № 24: Механизм PMTU

На машине **ws1** включим механизм PMTU следующей командой.

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

Перехватим трафик на трёх машинах точно также, как и в прошлом задании. С машины **ws1** 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**, ICMP-ответ о необходимости фрагментации был использован ядром ОС, и значение MTU было сохранено. В итоге пакет с эхо-запросом был фрагментирован ещё до выхода в сеть. Отметим, что в этих фрагментах уже нет (и не может быть) флага **DF**. Таим образом если на пути у отправителю величина MTU падает дважды, то механизм PMTU в случае ICMP-пакетов учитывает только первое «понижение». В лабе по TCP мы вернемся к нему ещё раз — возможно, там он будет «рекурсивным».

Для восстановления исходного MTU можно использовать те же команды, которыми оно был изменено (но со значением MTU, равным 1500), или перегрузить оба маршрутизатора.

3.10 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 маршрута.

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 № 25: Отсутствие получателя в таблице маршрутизации

Запустим на маршрутизаторе **r1** перехват сетевого трафика на интерфейса **eth0**.

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

Поскольку мы не указали ключ **-t**, то вывод программы tcpdump будет начинаться со 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
```

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

Задание № 26: Отсутствие получателя в сегменте сети

Запусим 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 выведено 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.

3.11 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) Что такое прямая и косвенная IP-маршрутизация?

3) 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?

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

5) IP-пакет отправляется с **ws1** на **r2** через **r1**. MAC-адреса каких интерфейсов будут указаны в полях получателя и отправителя кадров Ethernet, в которые будет вложен этот пакет?

6) 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?

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

8) Как при IP-маршрутизации используется протокол ICMP?

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

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

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

3.12 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.

```
cd ~/tcp-ip
wget 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 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. Это позволит воспользоваться утилитой, которая проверит, что все IP-адреса в одном сегменте сети относятся к одной IP-сети — в нашей простой сети это требование должно выполняться. Для присвоения IP-адреса нам нужно отредактировать файлы **etc/network/interfaces** Во всех каталогах виртуальных машин. Вы можете прямо сейчас указать, помимо IP-адресов и масок сетей, маршрутизаторы по-умолчанию и правила добавления маршрутов. но 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 отчетом 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 отчета 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
```

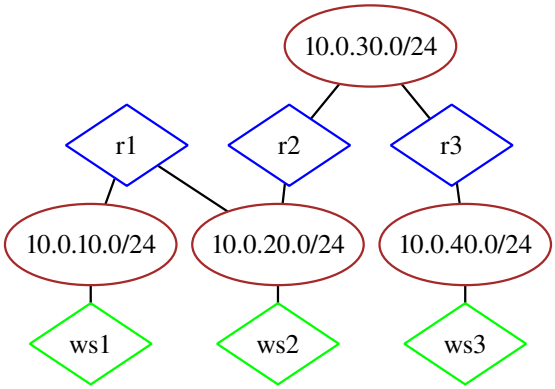
После запуска виртуальных машин следует добавить в файл конфигурации статические маршруты и маршрут по-умолчанию, как было показано в разделах 3.3 и 3.4. Отметим, что теперь файл **/etc/network/interfaces** меняется уже внутри виртуальной машины. После перезагрузки сетевых настроек вы должны убедится с помощью команды **traceroute**, что пути до всех машин соответствуют ожидаемым, а если это не так — внести необходимые исправления в маршрутные таблицы.

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.

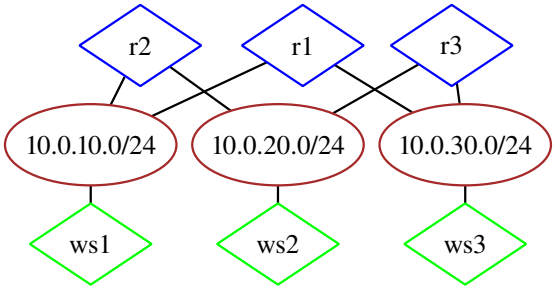
3.13 Варианты заданий для самостоятельной работы

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

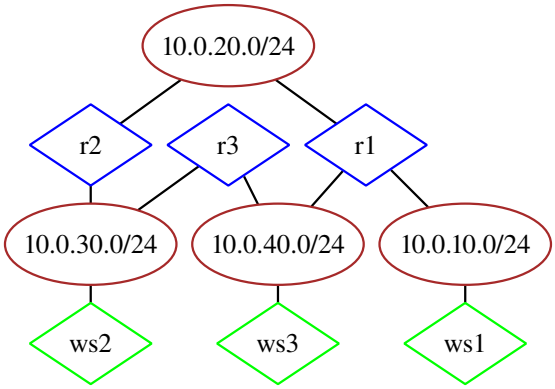
Вариант № 1



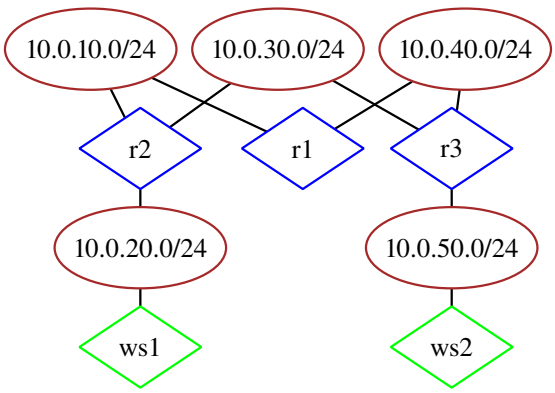
Вариант № 2



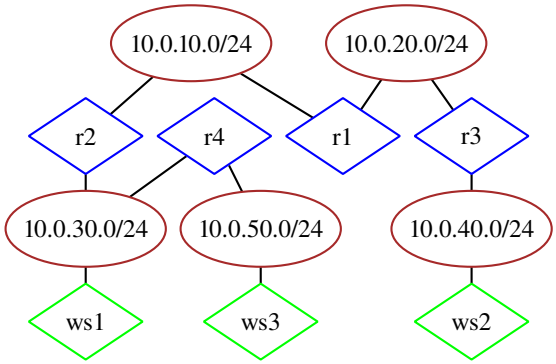
Вариант № 3



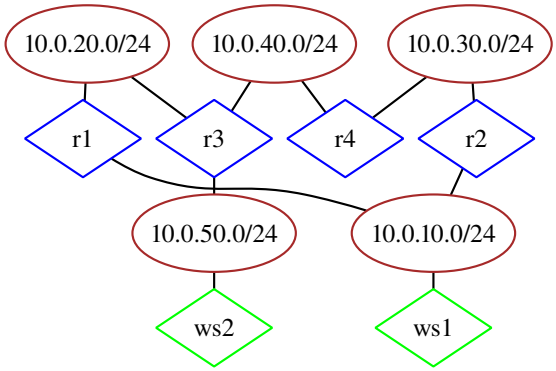
Вариант № 4



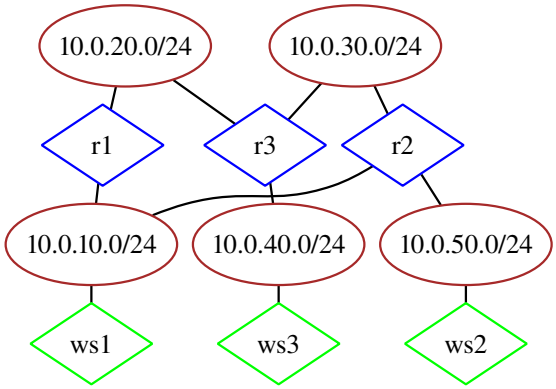
Вариант № 5



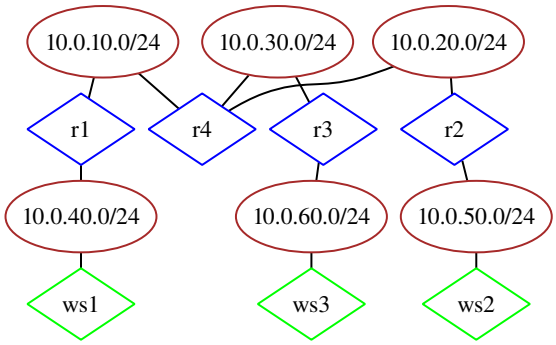
Вариант № 6



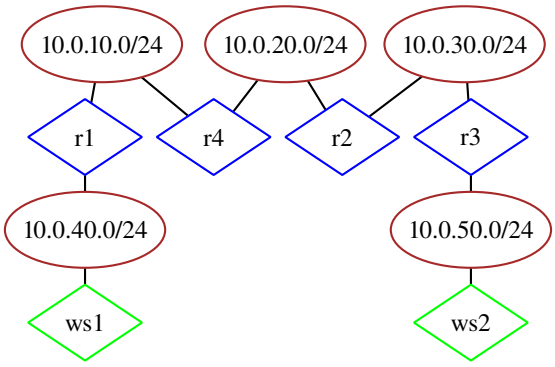
Вариант № 7



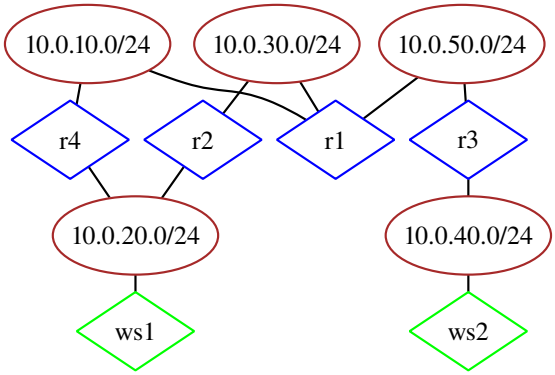
Вариант № 8



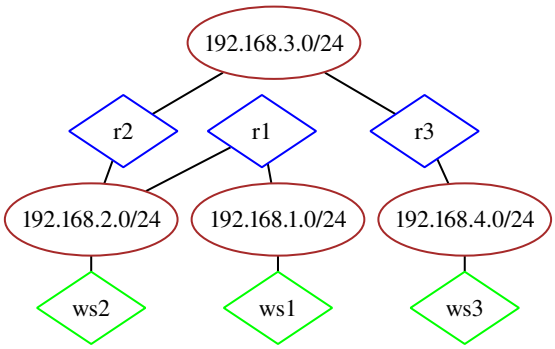
Вариант № 9



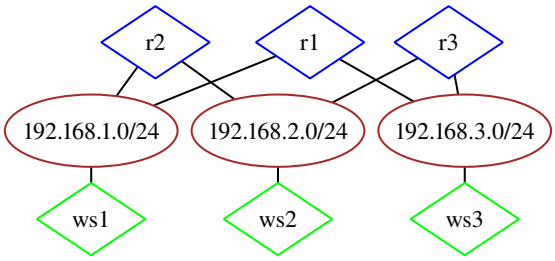
Вариант № 10



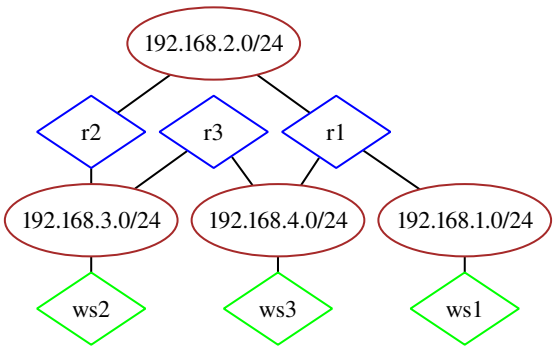
Вариант № 11



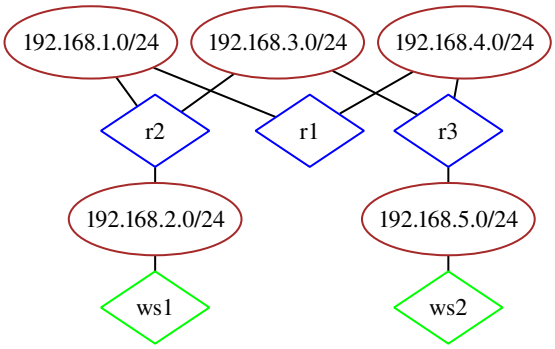
Вариант № 12



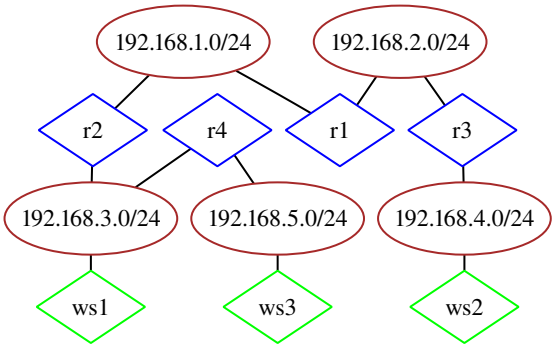
Вариант № 13



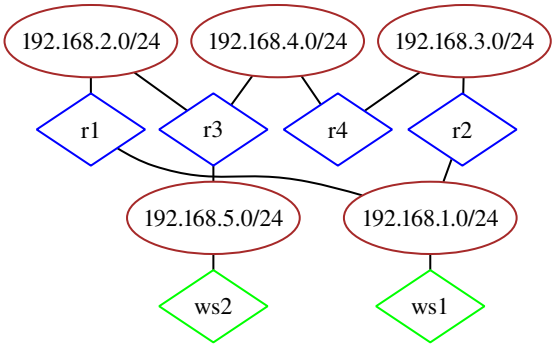
Вариант № 14



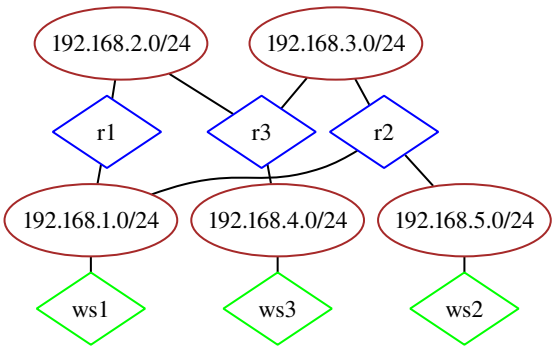
Вариант № 15



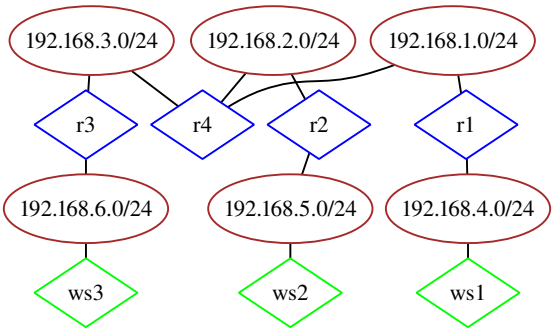
Вариант № 16



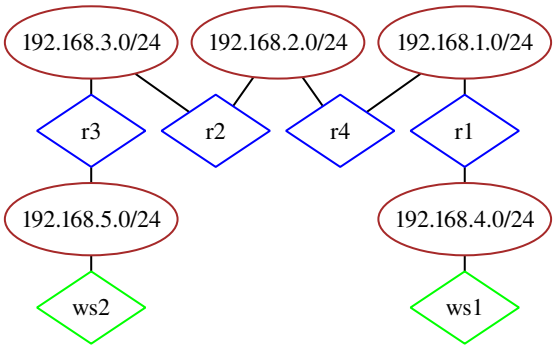
Вариант № 17



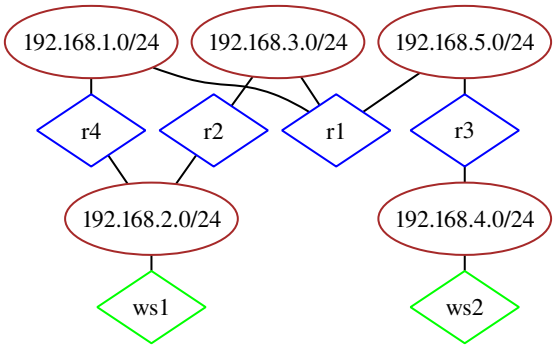
Вариант № 18



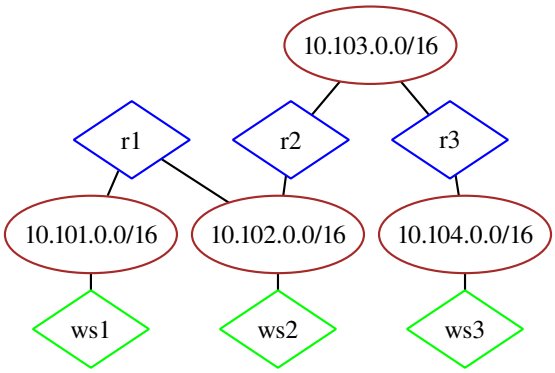
Вариант № 19



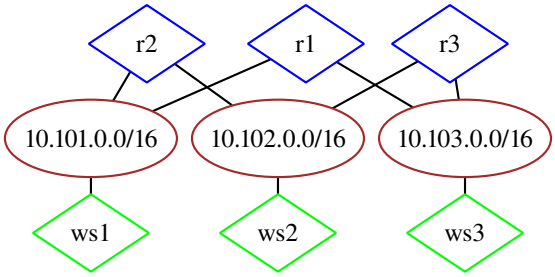
Вариант № 20



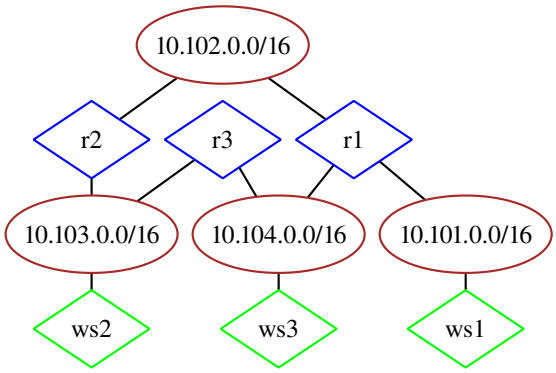
Вариант № 21



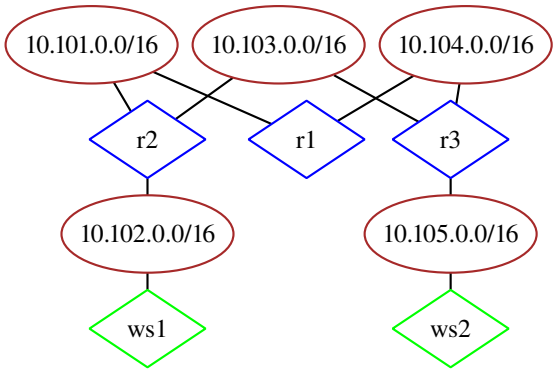
Вариант № 22



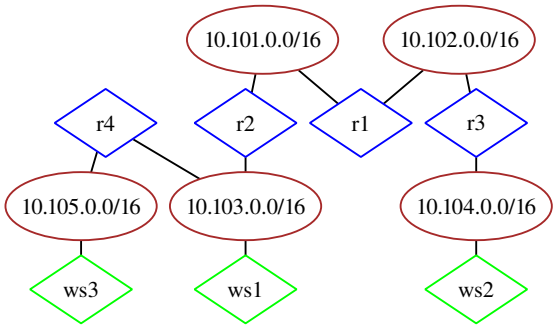
Вариант № 23



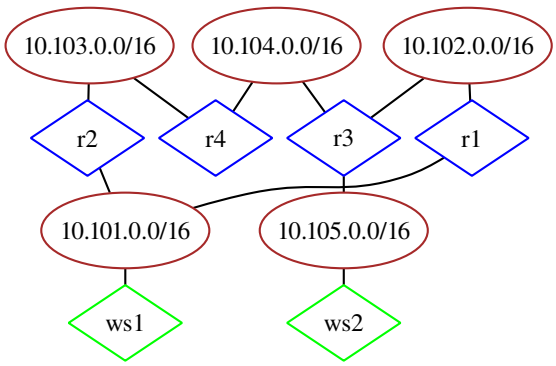
Вариант № 24



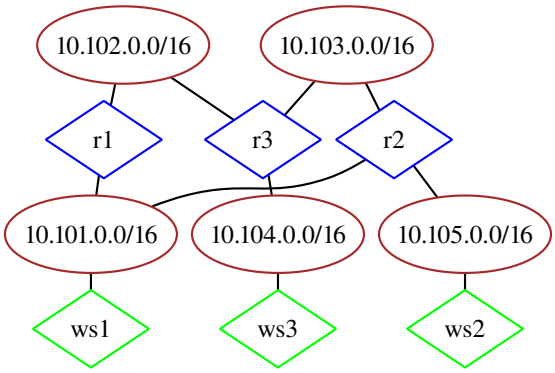
Вариант № 25



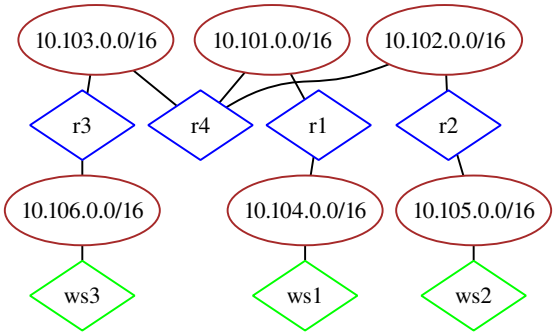
Вариант № 26



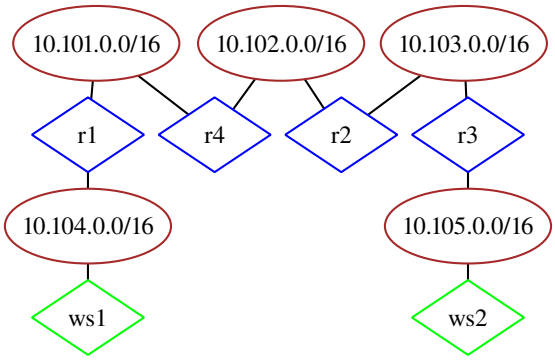
Вариант № 27



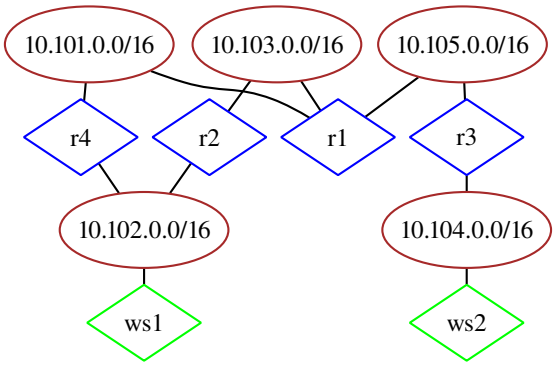
Вариант № 28



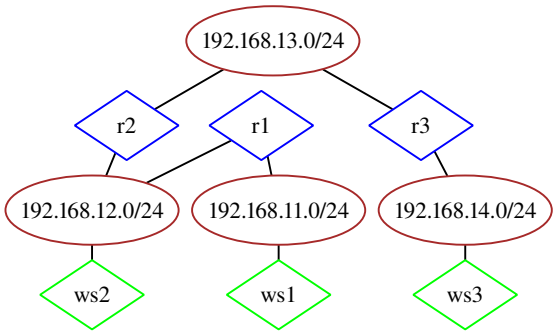
Вариант № 29



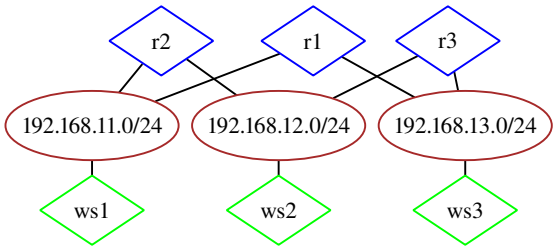
Вариант № 30



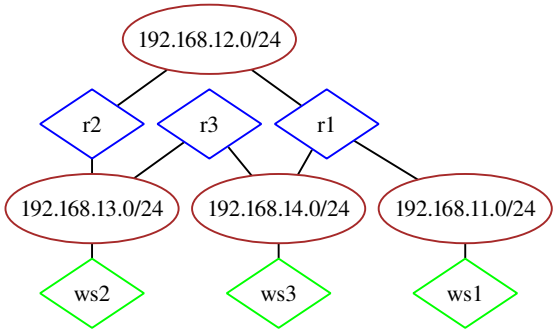
Вариант № 31



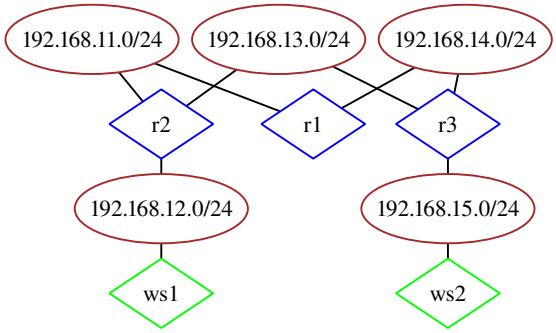
Вариант № 32



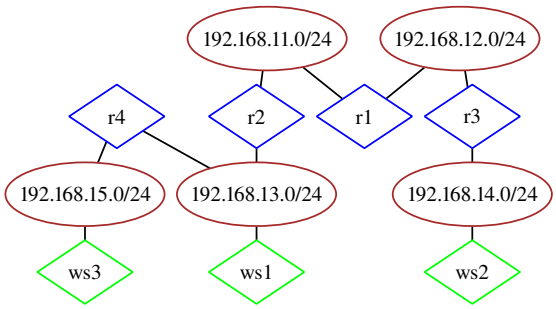
Вариант № 33



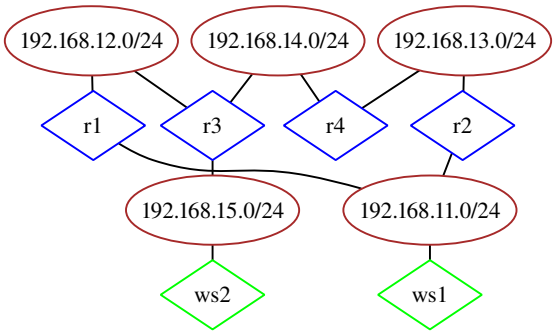
Вариант № 34



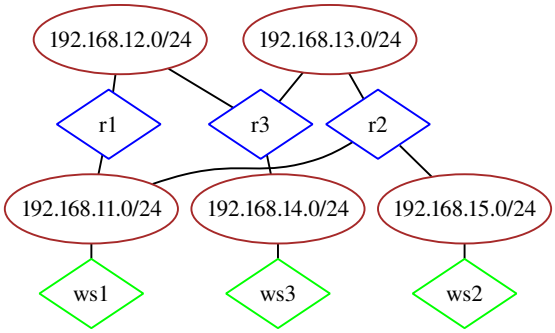
Вариант № 35



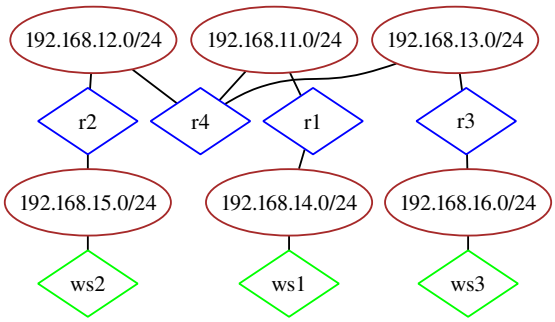
Вариант № 36



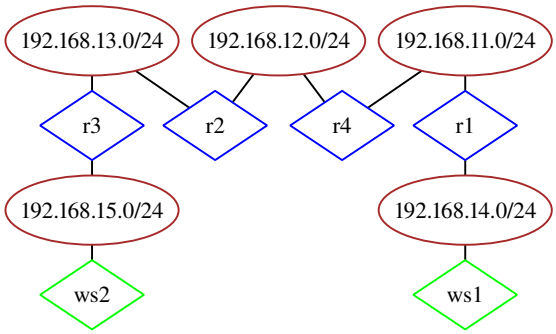
Вариант № 37



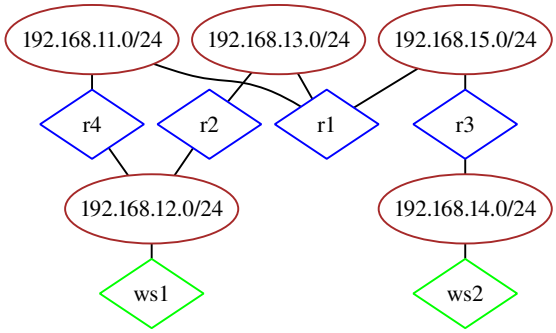
Вариант № 38



Вариант № 39



Вариант № 40



Заключение

В ходе выполнения заданий мы познакомились с функционированием компьютерных сетей, использующих протоколы сетевого стека TCP/IP. Были рассмотрены основные моменты функционирования канального, сетевого и транспортного уровней, а так же служебные протоколы обмена маршрутной информацией, поиска адресов, диагностики и другие. Приведены основы создания виртуальных частных сетей и виртуальных сегментов. Были изложены протоколы прикладного уровня, служащие для передачи почты, поддержки мнемонических имён и удалённого администрирования систем. Проблемы обеспечения сетевой безопасности так же нашли отражения в ряде заданий, касающихся виртуальных сетей, перехвата трафика, удалённого администрирования.

В ходе практических работ были также рассмотрены основы сетевого программирования и сетевого администрирования, изучены основные утилиты для диагностики сети и перехвата сетевого трафика. Рассмотрение протоколов транспортного уровня охватывало моменты, важные для осмысленного использования данных протоколов прикладными программистами в своей работе.

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. *Wikipedia*. Address Resolution Protocol — Wikipedia, The Free Encyclopedia. — 2011. `http://en.wikipedia.org/wiki/Address_Resolution_Protocol`.
5. *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.
6. *Олифер, В.Г.* Компьютерные сети. Принципы, технологии, протоколы / В.Г. Олифер, Н.А. Олифер. — СПб: Питер, 2010. — 944 с.
7. *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.
8. *Wikipedia*. Internet Protocol — Wikipedia, The Free Encyclopedia. — 2011. `http://en.wikipedia.org/wiki/Internet_Protocol`.
9. *Reynolds, J.* Assigned Numbers. — RFC 1700 (Historic). — 1994. — Obsoleted by RFC 3232. `http://www.ietf.org/rfc/rfc1700.txt`.
10. *Wikipedia*. Internet Control Message Protocol — Wikipedia, The Free Encyclopedia. — 2011. `http://en.wikipedia.org/wiki/Internet_Control_Message_Protocol`.

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
```

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 тестовую машину (всю работу мы будем вести в каталоге **■/nets**).

```
mkdir -p ~/nets; cd ~/nets
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.

[1] Также это можно сделать командой `source ~/.bashrc`.

```
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 программу Konsole (или Gnome Terminal), затем скачем и разархивируем файл с описанием сети из двух виртуальных машин.

```
cd ~/tcp-ip
wget 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. Прервём выполнение сценария **ltabstart**, нажав Ctrl-C, затем остановим машины и удалим файлы их образов дисков следующей командой.

```
lcrash -d .
```

Право на подключение к реальной сети

Выполнение заданий из главы **??** 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 Справочная информация о пакете 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
```

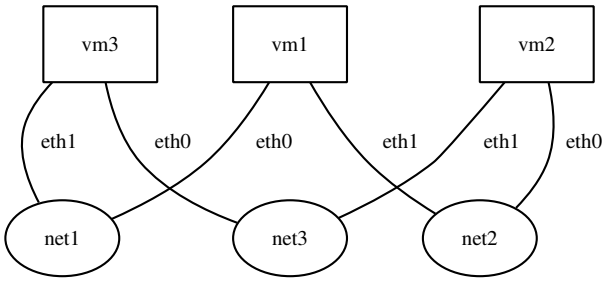


Рисунок Б.1 — Виртуальная сеть из трёх машин

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).

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/
        etc/
            network/
                interfaces
            openvpn/
                server.conf
                ...
            resolv.conf
    r2/
        ...
    pc11/
        ...
    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**. Копирование файлов из каталога «лабораторной работы» выполняется только при создании образа (при первом запуске машины). Поэтому, если вы изменили у себя конфигурационные файлы и хотите, чтобы изменения вступили в силу, выполните на основной машине следующие команды (при этом буду удалены все изменения внутри виртуальных машин, об их сохранении написано далее).

```
lhalt         # завершает все виртуальные машины
rm *.disk     # удаляет все файлы образом машин
ltabstart -d .   # снова запускает лабораторную работу
```

Можно удалить только образ отдельной машины и затем запустить только её с помощью команды **lstart**. В силу специфики реализации файла образа удалять образ его либо командой **rm**, либо без перемещения её в корзину (типичное клавиатурное сочетание в графическом файловом менеджере — Shift+Del).

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.

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/
```

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.

Приложение В Список практических заданий

1) Запуск виртуальных машин командой vstart (стр. 25).
2) Получение сведений о сетевых интерфейсах (стр. 26).
3) Получение сведений об IP-адресах интерфейсов (стр. 27).
4) Использование утилиты ping (стр. 28).
5) Назначение интерфейсу IP-адреса и маски сети (стр. 28).
6) Вывод кеша протокола ARP (стр. 29).
7) Перехват сообщений протокола ARP (стр. 30).
8) Вывод MAC-адресов при перехвате сетевого трафика (стр. 31).
9) Отправка пакета на отсутствующий в сети IP-адрес (стр. 31).
10) Вывод таблицы маршрутизации (стр. 32).
11) Случай отсутствие адреса в таблице маршрутизации (стр. 32).
12) Остановка виртуальных машин (стр. 33).
13) Запуск сети командой ltabstart (стр. 36).
14) Настройка адресов рабочих станций (стр. 38).
15) Настройка адресов маршрутизаторов (стр. 38).
16) Установка маршрута по умолчанию (стр. 39).
17) Добавление статических маршрутов (стр. 39).
18) Косвенная маршрутизация (стр. 40).
19) Создание маршрутной петли (стр. 41).
20) Построение списка маршрутизаторов на пути (стр. 42).
21) Изменение величины MTU (стр. 43).
22) Отключение механизма PMTU (стр. 44).
23) Фрагментация IP-пакетов (стр. 44).
24) Механизм PMTU (стр. 45).
25) Отсутствие получателя в таблице маршрутизации (стр. 45).
26) Отсутствие получателя в сегменте сети (стр. 46).

Приложение Г Предметный указатель