Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
Национальный исследовательский университет «МЭИ»

ЭЛЕКТРОННЫЙ
УЧЕБНО-МЕТОДИЧЕСКИЙ КОМПЛЕКС

дисциплины по выбору студента вариативной части профессионального цикла Б.3

«Информационные сети и телекоммуникации»

Направление подготовки 220400 «Управление в технических системах»

Профиль «Управление и информатика в технических системах»

Авторский коллектив:

Старший преподаватель кафедры УиИ _____ В. В. Кузнецов

«____»_____ 2015 г.

Москва 2015 НИУ «МЭИ»

СОДЕРЖАНИЕ

1 ОСНОВЫ ИНФОРМАЦИОННЫХ СЕТЕЙ

1.1 Информационная сеть

*Информационной сетью (*или просто *сетью)* назовем систему, в которой:

1) присутствует несколько участников (узлов);
2) имеется возможность передать данные между любыми двумя узлами в любом направлении, и это является основной задачей сети.

Передача информации между узлами сети осуществляется, очевидно, с использованием некоторых сигналов. Сигнал — это изменение физической величины со временем. Физическая величина отражает количественно свойство некоторого объекта. Таким образом, для передачи информации в сети одному из узлов нужно изменить свойство некоторого объекта (например, напряжение проводника), а другому узлу — уловить это изменение. Говорят, что узлам необходима *общая среда передачи (media).* Примером такой среды может служить провод (телефонная линия), электромагнитное поле (радио, Wi-Fi) или воздух (при разговоре).

Для создания информационной сети недостаточно лишь наличия общей среды передачи — имеются и определенные требования к узлам.

Действительно, пусть некий узел может как улавливать изменения среды, так и не делать этого. Тогда гарантированно передать ему информацию нельзя: узел может не воспринять ни один предназначенный ему сигнал. Значит, *узел сети должен вести прием постоянно.* Это верно для всех узлов в любой момент времени.

Предположим теперь, что передачу ведут 2 узла одновременно. Совокупный сигнал, генерируемый ими, почти наверняка не будет полезным ни для какого узла. Более того, это может быть физически недопустимо (например, включение на электрической линии двух генераторов одновременно). Поэтому в информационных сетях в каждый момент времени передачу должен вести только один узел *(раздельный доступ к среде).*

На рис. 1 представлен пример элементарной сети, в которой общей средой передачи является электрическая линия с сигналом-напряжением, а каждый из трех узлов включает генератор-передатчик с ключом и вольтметр-приемник. Все вольтметры измеряют

напряжение в проводнике непрерывно, а генераторы должны подключаться к линии по одному во избежание короткого замыкания.

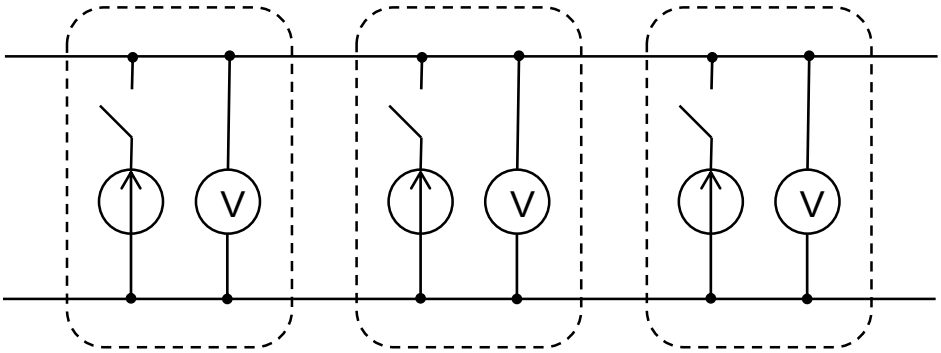


Рисунок 1 — Пример элементарной сети с тремя узлами

Можно представить ситуацию, когда в сети все узлы ведут прием, а один узел — непрерывную передачу (пример: лекция). Вышеприведенные требования не нарушаются при этом, однако такая система не удовлетворяет определению информационной сети. Значит, необходимо еще одно требование: *время непрерывной передачи одним узлом должно быть конечно.*

С другой стороны, пусть узлы ведут короткие передачи, но самих узлов — бесконечно много. В этом случае некоторые узлы могут никогда не получить возможность передать сигнал. Значит, *ограничено должно быть и количество узлов в сети.*

Время прохождения сигнала между любыми двумя узлами должно быть конечным, иначе сигнал, посланный источником, может никогда не быть получен. Скорость распространения сигналов всегда ограничена (хотя бы скоростью распространения электромагнитной волны в вакууме). Чтобы сигнал всегда мог дойти от источника до приемника, они не должны быть бесконечно далеко друг от друга, либо скорость их удаления не должна быть слишком высокой. Требование может показаться теоретическим, однако существует простой пример — почтовая служба. Если письмо будет утеряно («будет идти бесконечно долго»), переписка не состоится.

Передача данных ведется порциями, которые называются *пакетами.*

1.2 Модель OSI

Сетевое взаимодействие включает много подзадач, каждая из которых имеет различные решения. С целью разделить эту сложную задачу на несколько простых

(произвести декомпозицию) и упорядочить терминологию вводят модели сетевого взаимодействия. Наиболее употребительной моделью является *OSI (Open System Interconnection — взаимодействие открытых систем)* [1], предложенная в 1974 г. Международной организацией по стандартизации (ISO).

Ключевым понятием OSI является *уровень (layer)* — абстракция от некоторого аспекта сетевого взаимодействия. Например, при загрузке web-страницы приложение-обозреватель (прикладной уровень модели OSI) не заботится о том, будут ли переданы данные из соседнего помещения или из другой страны, по кабелю или по Wi-Fi, поскольку эти вопросы решаются на других уровнях. Одни уровни используются другими: обозреватель формирует запрос web-страницы, передает его на следующий уровень для шифрования, оттуда данные попадают на уровень ниже и т. д., пока не приходят на низший, физический уровень, задача которого — непосредственно передать данные по физическому каналу связи.

Уровни выделяются на каждом узле: формирование запроса, его шифрование, и, в конце концов, генерация физического сигнала передачи происходят на одной машине. Наборы средств (функций, форматов данных), при помощи которых уровни взаимодействуют друг с другом, называются *интерфейсами* уровней.

Каждый уровень модели OSI взаимодействует только с одноименным уровнем на другом узле. Так, операционная система обеспечивает физическую передачу данных, не заботясь о том, каковы они (запрос ли это страницы и корректен ли он), а сервер сайта обрабатывает только непосредственно запрос, и ему даже не предоставляется деталей того, как он был передан. Правила, по которым взаимодействуют одноименные уровни на разных узлах, называются *протоколами.*

Модель OSI выделяет 7 уровней и устанавливает решаемые ими задачи. Ниже перечислены названия уровней; каждый из них будет подробнее рассмотрен далее в курсе:

7) прикладной уровень (application layer);
6) представительский уровень (presentation layer);
5) сеансовый уровень (session layer);
4) транспортный уровень (transport layer);
3) сетевой уровень (network layer);
2) канальный уровень (data link layer);
1) физический уровень (physical layer).

Модель OSI является эталонной, реальные системы соответствуют ей приблизительно. Тем не менее, она активно используется на практике, поскольку указание уровня модели OSI ёмко описывает назначение части сетевой системы.

1.3 Физический уровень модели OSI

Физический уровень модели OSI определяет природу и параметры общей среды передачи, виды и уровни сигналов. К нему относятся выведенные ранее требования к узлам информационной сети. Примеры физического уровня: оптическое волокно, радиоволны (Wi-Fi и GSM), коаксиальный электрический кабель.

Выполнение требований постоянного приема, ограниченного числа узлов в сети и конечного времени распространения сигнала обеспечивается аппаратной частью. Обеспечение ведения передачи единственным узлом в каждый момент времени может решаться по-разному; как правило, используется явление интерференции и резонанса, на которых основан популярный метод CSMA/CD. Кроме того, Wi-Fi может использовать одновременно несколько диапазонов частот и мощностей сигнала; каждая такая область становится отдельной средой передачи.

Ограничение времени непрерывной передачи одним узлом физически не обеспечивается, а просто программируется в поведении каждого узла. Кроме того, передача зачастую из технических соображений ведется *кадрами (frames)* — пакетами физического уровня.

На физическом уровне любая передача ведется одним узлом для всех остальных.

1.4 Канальный уровень модели OSI

Канальный уровень модели OSI добавляет к возможностям физического уровня направленную передачу конкретному узлу в пределах единой среды.

Канальный уровень для непосредственной передачи данных использует физический уровень, где любой сигнал получают все узлы. Возникают три задачи:

1) различение узлов в пределах сети (адресация);
2) указание, что пакет предназначен некоторому узлу;
3) распознание узлом, что пакет предназначен именно ему.

1.4.1 Адресация на канальном уровне

Адрес на канальном уровне однозначно идентифицирует узел в сети: ни у каких двух узлов одной сети не должно быть одинакового адреса канального уровня.

На практике применяется *MAC-адрес (media access control)* [2, раздел 9], состоящий из шести байт и записываемый в шестнадцатеричном виде, например, `12:34:56:78:9A:BC`, или через дефисы: `12-34-56-78-9A-BC`.

Каждому узлу должен быть заранее известен собственный MAC-адрес. Желательно, чтобы при постройке сети из любых устройств MAC-адреса сразу оказались уникальными. С этой целью производителям сетевого оборудования выделяются диапазоны, MAC-адреса из которых присваиваются разъемам сетевых карт и другим устройствам, — адреса получаются уникальными глобально. Однако, MAC-адреса могут быть и *выданными локально (locally administered),* например, для сетевых карт виртуальных машин.

Специальный MAC-адрес `FF:FF:FF:FF:FF:FF` (все двоичные разряды — единицы) означает, что данный пакет предназначен всем узлам. Это называется *широковещательной рассылкой (broadcast)* на канальном уровне. Существует еще несколько MAC-адресов с тем же свойством, называемых групповыми адресами [3]. Они используются специальными протоколами канального уровня для обслуживания сети.

1.4.2 Структура пакета канального уровня

Каждый пакет канального уровня должен содержать указание, какому узлу он предназначен — адрес узла назначения (destination MAC, `MD`). Почти всегда необходим и адрес источника (source MAC, `MS`). В оставшейся части пакета находятся полезные данные (payload). Примерная структура пакета канального уровня приведена на рис. 2.

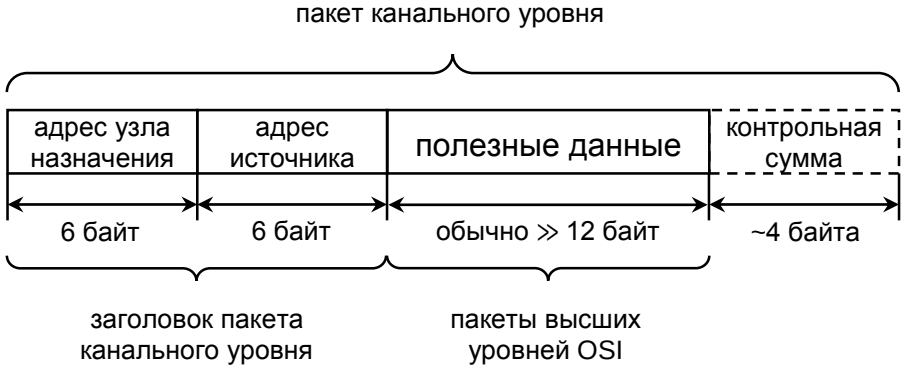


Рисунок 2 — Приблизительная структура пакета канального уровня

Каждый пакет канального уровня доставляется средствами физического уровня всем узлам сети. Узлы-получатели анализируют поле `MD`, и, если пакет предназначен данному узлу, передают содержимое `payload` для обработки вышестоящим уровням. Это включает два основных случая: когда `MD` равно адресу узла-получателя (адресная рассылка), и когда все двоичные разряды `MD` — единицы (широковещательная рассылка). Узел может принять пакет и тогда, когда `MD` является групповым адресом, если заинтересован в таких пакетах.

Часто канальный уровень дополнительно обеспечивает целостность кадров. Для этого в пакет включается *контрольная сумма (checksum)* — число, зависящее ото всех байт пакета. После приема контрольная сумма пакета рассчитывается получателем, и, если она не совпадает с указанной в пакете, считается, что пакет оказался поврежден при передаче.

1.4.3 Возможности, предоставляемые канальным уровнем

Канальный уровень добавляет к возможностям физического:

1) адресацию всех узлов сети;
2) возможность передачи данных конкретному узлу;
3) возможность передачи данных всем узлам данной сети.

В реальных системах канальный уровень плотно интегрирован с физическим — вплоть до аппаратной реализации. На канальном уровне может решаться задача обеспечения раздельного доступа к среде передачи, поскольку появляется удобная возможность учета узлов в сети по их адресам.

1.5 Литература к разделу 1

1. **ISO/IEC 7498-1:1994 «Information Technology — Open Systems Interconnection — Basic Reference Model: The Basic Model».**
2. IEEE 802-2001 «IEEE Standard for Local and Metropolitan Area Networks: Overview and Architecture».
3. Standard Group MAC Addresses: A Tutorial Guide [Электронный ресурс]. — Режим доступа: http://standards.ieee.org/develop/regauth/tut/macgrp.pdf, свободный.

2 ПЕРЕДАЧА ДАННЫХ МЕЖДУ СЕТЯМИ И ПРИЛОЖЕНИЯМИ

2.1 Недостатки сетей канального уровня

В своем развитии сети канального уровня сталкиваются с принципиальными проблемами.

В случае, если узлы сети географически сильно удалены, становится затруднительно создать единую среду передачи, так как дальняя радиосвязь подвержена помехам либо требует дорогостоящих и ненадежных спутников, а электрическая и волоконно-оптическая связь ограничены стоимостью проводников (а также их прокладки и обслуживания). Кроме того, единая среда передачи для N узлов требует $½ \cdot N^2$ связей, поэтому кабельные решения не подходят.

Еще одной проблемой при большом количестве узлов является возрастающее время ожидания узлом очереди на передачу. Это ограничение физического уровня не удалось бы решить никаким количественным улучшением (дешевой средой передачи или быстрыми устройствами узлов).

2.2 Сетевой уровень модели OSI

Поскольку небольшие сети канального уровня сегодня способны обеспечить желаемое быстродействие, естественное решение — разделить большую сеть на множество малых подсетей. Узлы каждой подсети работали бы в собственной среде передачи, компактной и дешевой. Узлы вели бы передачу по очереди, но одновременно в каждой из подсетей параллельно и независимо.

Описанная организация большой сети требует передачи данных между подсетями. Для этого требуется, чтобы некоторый участник принимал данные из одной сети и передавал в другую. Такая машина, подключенная одновременно более чем к одной сети, называется *маршрутизатором.* На рисунке 3 представлены три сети:

1) узлы 11, 12, 13, 14, 15; машины М1 и М2 (маршрутизаторы);
2) узел 21; машины М1 и М2 (маршрутизаторы);
3) узел 31; машина М2 (маршрутизатор).

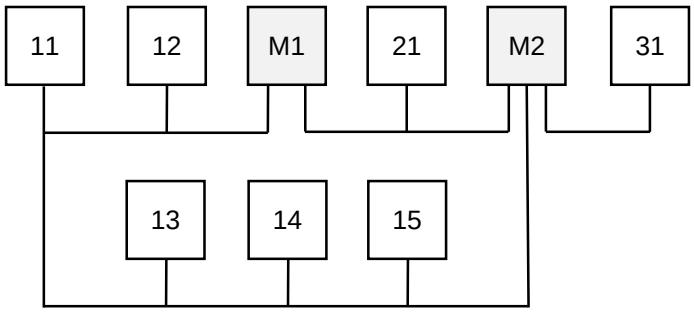


Рисунок 3 — Сети с маршрутизаторами

Узлом сети является *сетевой интерфейс (network interface controller, NIC):* разъем для кабеля, антенна и т. д. на сетевой плате. Машина-маршрутизатор содержит несколько узлов в различных сетях канального уровня. В случае, когда у машины сетевой интерфейс единственный, её часто называют просто узлом.

Маршрутизаторы обеспечивают передачу данных между различными локальными сетями (канального уровня), то есть решают задачу *сетевого уровня модели OSI.* Сетевой уровень расширяет возможности канального уровня передачей информации в сетях с произвольной топологией, то есть с любой схемой соединения узлов, а не только каждого с каждым.

На канальном уровне (в одной подсети) сигнал попадает в среду передачи и сразу же воспринимается целевым узлом. На сетевом уровне возникает понятие *маршрута* — последовательности узлов, через которые проходит сигнал от источника до назначения. Например, маршруты между узлами 11 и 31 — (11, M1, M2, 31) или (11, M2, 31). Важной задачей на сетевом уровне является маршрутизация — выбор «наилучшего» маршрута. Критерии выбора могут быть разными: скорость передачи, защищенность, цена и т. д.

Часто на сетевом уровне применяется протокол IP, адресация в сетях которого рассматривается в лекции 3, а маршрутизация — в лекции 4.

2.3 Транспортный уровень модели OSI

Вхождение узла в некоторую сеть означает, что к среде передачи сети подключен один из сетевых интерфейсов данного узла (например, порт Ethernet сетевой карты ПК). Однако, как правило, сетевое взаимодействие осуществляет не узел в целом, а приложение, выполняемое на нем, причем не одно, а несколько одновременно (например, загрузка web-страниц браузером и получение ОС обновлений). Только лишь адреса узла в сети

недостаточно для того, чтобы указать, что пакет данных предназначен конкретному приложению на этом узле. *Задача адресации приложений в пределах узла решается на транспортном уровне модели OSI.* С этой целью вводится номер порта. У каждого узла есть набор номеров (от 0 до 65535), каждый из которых отводится работающему приложению. Иллюстрация приведена на рис. 4.

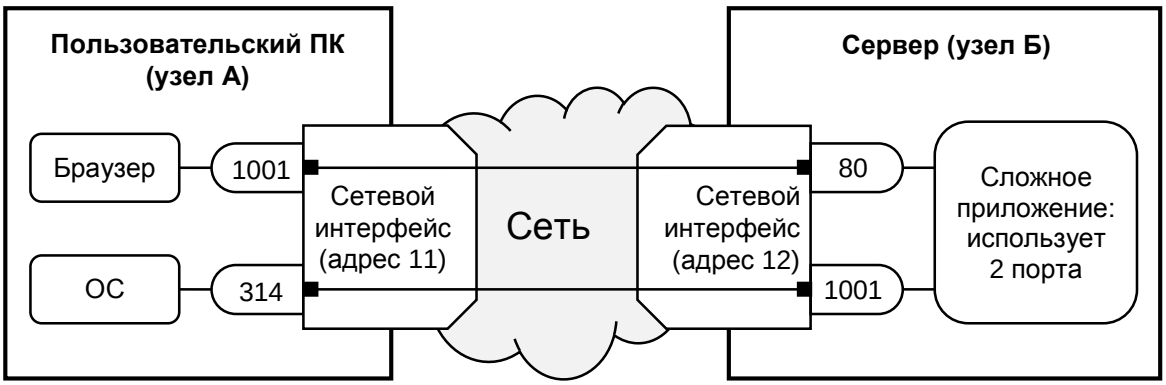


Рисунок 4 — Взаимодействие на транспортном уровне

Номера портов при сообщении двух узлов могут не совпадать: на рис. 4 браузер передает данные через порт 1001 узла А на порт 80 узла Б, на котором порт 1001 используется в других целях. Одно приложение может использовать несколько портов[1]. Через один порт приложение может одновременно сообщаться с несколькими другими приложениями, различая их пакеты по адресу и порту отправителя.

На транспортном уровне повторно (после канального уровня) возникает задача обеспечения целостности пакетов. Один пакет транспортного уровня может состоять из нескольких кадров (пакетов канального уровня), поэтому, даже в случае, если ни один из *принятых* кадров не поврежден, требуется проверять, что пакет транспортного уровня не испорчен *потерей* части кадров. Технически это выполняется так же, как и на канальном уровне — путем подсчета контрольной суммы всего пакета транспортного уровня.

2.4 Сеансовый уровень модели OSI

Сетевое взаимодействие между приложениями часто неравномерно во времени: есть сеансы активного обмена информацией (например, загрузка файла или звонок) и есть

[1] Один порт также может использоваться несколькими приложениями, однако, такой режим работы является исключением и по умолчанию запрещен.

периоды без активности узлов, хотя они продолжают находиться в сети. Во время сеанса обычно используется гораздо больше ресурсов, чем вне его, поэтому узлам важно знать, когда сеансы начинаются и заканчиваются. *Сеансовый уровень модели OSI и определяет промежуток времени сеанса связи между приложениями.* Вводятся процедуры установления, поддержания и разрыва соединения, что позволяет повысить эффективность передачи данных, а также диагностировать разрыв соединения.

В подразделе 5.2 рассмотрен подробнее широко распространенный протокол TCP, объединяющий транспортный и сеансовый уровни.

2.5 Представительский уровень модели OSI

Наивысший (7-й), прикладной уровень модели OSI только оперирует полезными данными и не участвует в их передаче. Напротив, уровни с физического по сетевой решают задачу обеспечения связи. *Представительский уровень модели OSI (6-й) выполняет сжатие, распаковку или шифрование данных.* Хотя эта работа не относится к обеспечению связи, важно, что она производится *прозрачно* для прикладного уровня. Например, web-страницы могут загружаться в сжатом виде, но это не отразится на работе прикладного уровня ни клиента, ни сервера, их не потребуется изменять или исправлять. Подходы, используемые на прикладном уровне, подробно рассмотрены в лекции 6.

3 АДРЕСАЦИЯ В СЕТЯХ IP

Главная задача сетевого уровня модели OSI — маршрутизация, то есть выбор последовательности узлов для передачи сообщения между подсетями канального уровня. Чтобы решить задачу, необходимо знать:

1) какой сети принадлежит узел-адресат;
2) в какие сети входят маршрутизаторы узла-отправителя.

MAC-адрес не содержит информацию о сети, поэтому задачу 1) можно решить только при наличии полного перечня узлов с указанием сетей, в которые они входят. Однако для глобальной сети это невозможно: узлов слишком много, структура сети постоянно меняется. Следовательно, MAC-адрес категорически непригоден на сетевом уровне.

Требуется адрес:

1) позволяющий определить сеть, которой принадлежит узел;
2) настраиваемый пользователем, чтобы оборудование можно было перемещать между сетями, добавлять и заменять.

В интернете получили распространение *адреса IP (internet protocol address)* версии 4 [1, подраздел 2.3]. Адрес IPv4 состоит из 4-х байт, записываемых через точку: «192.168.2.1». Адрес IP включает номер сети и номер узла в этой сети. Пусть узлы с адресами IP 192.168.2.1, 192.168.2.2, … 192.168.2.255 принадлежат одной сети; тогда 192.168.2 — номер сети, а 1 — 255 будут номерами узлов в ней. Способы отделить номер сети от номера узла в адресе IP называются *архитектурами адресации;* используется классовый и бесклассовый вариант.

3.1 Классовая архитектура адресации

Классовая архитектура адресации делит все множество адресов на фиксированное количество сетей разного размера (сети разных классов). Каждая сеть занимает непрерывный диапазон адресов. Нахождение адреса IP в той или иной сети можно легко проверить по страшим битам адреса. Классификация адресов IP приведена в таблице 1.

Таблица 1 — Классификация адресов IP ([1, с. 24], [2])

Класс	Старшие биты	Размер номера сети (бит)	Размер номера узла (бит)	Число сетей	Число узлов (примерно)	Диапазон адресов
A	0	8	24	128	16 777 216	0.0.0.0 — 127.255.255.255
B	10	16	16	16 384	65 536	128.0.0.0 — 191.255.255.255
C	110	24	8	2 097 152	256	192.168.0.0 — 223.255.255.255
D	1110	—	—	—	—	224.0.0.0 — 239.255.255.255
E	1111	—	—	—	—	240.0.0.0 — 255.255.255.255

Класс D применяется для технических нужд (многоадресная рассылка, multicast). Класс E зарезервирован и не используется, кроме нескольких специальных адресов.

Преимущество классовой архитектуры адресации — возможность по адресу IP определить номер сети. Основным недостатком классовой адресации является то, что размеры и диапазоны адресов сетей строго фиксированы, и крупная сеть не может быть разделена на меньшие. Вследствие этого существенная часть адресов в больших сетях (классы A и B) не используется, но и не может быть передана в другие сети.

3.2 Бесклассовая архитектура адресации

В *бесклассовой архитектуре адресации (Classless Inter-Domain Routing, CIDR)* предлагается отводить под номер сети столько бит, сколько реально необходимо. Чтобы указать количество бит в номере сети, помимо IP адреса используется т. н. *маска подсети (subnet mask).* Она представляет собой 4 байта, биты которых равны 1, если разряд отведен под номер сети, и 0 в противном случае [3].

Пример. Пусть требуется сеть на 10 узлов. На номер узла требуется 4 бита, следовательно, на номер сети отводится $32 - 4 = 28$ бит. Рассмотрим узел № 9, выбрав номер сети произвольно.

Узел номер 9:	10100110 00001111 01100101 01001001	(166.15.101.73)
Маска подсети:	11111111 11111111 11111111 11110000	(255.255.255.224)
Номер сети:	10100110 00001111 01100101 01000000	(166.15.101.64)

Номер сети можно вычислить побитовой конъюнкцией (логическое «И») между битами адреса узла и маской подсети. В данном случае, маска подсети — 255.255.255.224, тогда номер сети — 166.15.101.64. Сокращенно пишут количество единиц (28) в маске подсети: 166.15.101.64/28. Номер узла записывается полностью как 166.15.101.73/28.

Преимуществом бесклассовой архитектуры является возможность маской разделить большую сеть на несколько подсетей меньшего размера. Пусть, например, имеется сеть 192.168.0.0/16 с 254 узлами, то есть для номера узла достаточно младшего байта адреса. Зафиксировав второй (с конца) байт в 0, 1, …, 255, можно выделить 256 подсетей с маской /24: 192.168.1.0/24, 192.168.2.0/24 и т. д.

Бесклассовая архитектура адресации применяется совместно с классовой.

3.3 Специальные адреса IPv4

Некоторые адреса IPv4 имеют специальное значение, приведенное в таблице 2 [4].

Таблица 2 — Специальные адреса IPv4

Номер сети	**Номер узла**	**Смысл**
0…0	0…0	Данный узел данной сети.
0…0	x…x	Узел в данной сети.
0…0	1…1	Все узлы данной сети.
x…x	0…0	Данный узел в указанной сети.
x…x	x…x	Указанный узел в указанной сети.
x…x	1…1	Все узлы указанной сети.
1…1	0…0	Запрещен.
1…1	x…x	Запрещен.
1…1	1…1	Все узлы всех сетей.

Принцип составления таблицы 2 несложен: номер из 0 означает данный узел или данную сеть (используемую по умолчанию); номер из 1 означает все сети или все узлы.

Сообщение, отправленное адресату 255.255.255.255, формально должно быть передано всем узлам всех сетей. На практике это не только не нужно, но и потенциально опасно и вредно. Пакеты доставляются всем узлам всех сетей, в которые входит данный узел, но не маршрутизируются во внешние сети.

Специальная сеть 127.0.0.0/8 *(loopback, «локальная петля»)* предназначена для соединения узла самого с собой, даже если нет связи ни с одной физической сетью. Узел 127.0.0.1 называют обычно localhost.

Не всем узлам нужен адрес в глобальной сети. Существуют т. н. *частные сети (private networks):* 10.0.0.0/8 (1 сеть класса A), 172.16.0.0/12 (16 сетей класса B), 192.168.0.0/16 (256 сетей класса C). Их пользователи могут самовольно назначить себе адреса из этих сетей, но частные сети не входят в глобальную, по адресам узлов в них нельзя отправить данные из других сетей [5].

Как правило, маршрутизаторам стараются назначать номер узла 1. Например, домашние маршрутизаторы часто выбирают для себя адрес 192.168.1.1 и подключают устройства к частной сети 192.168.1.0/24, а для связи с интернетом используют отдельный интерфейс (в глобальной сети или в частной сети поставщика).

3.4 Адресация IPv6

Всего возможно около 4 млрд. адресов IPv4, в то время как количество узлов, которым требуется доступ к сети, значительно больше. Отчасти проблема решается частными сетями: их узлы не занимают адреса в интернете, но могут получать доступ к глобальной сети через маршрутизаторы-шлюзы. Однако, такой подход не всегда возможен, поэтому возникает проблема исчерпания доступных адресов IPv4.

Протокол IP версии 6 (IPv6) [6] призван заменить IPv4. Адрес IPv6 имеет размер 128 бит (16 байт), например, `2001:0db8:0000:0064:0000:0000:aa72:0004`. Тем самым проблемы исчерпания адресного пространства IPv6 в обозримом будущем не предвидится. Каждом узлу в интернете возможно выделить не только уникальный адрес, но даже сеть. По этой причине частные сети IPv6 ожидаются менее популярными. IPv6 не предусматривает широковещательной рассылки, предпочитая ей многоадресную.

С точки зрения программирования, сети IPv6 мало отличаются от IPv4. Например, в интерфейсе программирования сокетов вместо `AF_INET` используется `AF_INET6`, вместо структуры `sockaddr_in` — `sockaddr_in6`, а для перевода адреса в виде строки в 16 байт применяется функция `inet_pton()` вместо `inet_aton()`.

Переход на повсеместное использование IPv6 сдерживается большим количеством систем и оборудования, построенных на базе IPv4. В переходный период взаимодействие между сетями IPv4 и сетями IPv6 осуществляется через специальные шлюзы. Необходимость перехода на IPv6 деятельно признана крупными государствами (например, Китаем) и корпорациями, поэтому замена IPv4 на IPv6 — дело обозримого будущего.

3.5 Литература к разделу 3

1. RFC 791: Internet Protocol [Электронный ресурс]. — Страница в интернете. — Режим доступа: http://tools.ietf.org/html/rfc791, свободный.

2. M. Cotton, L. Vegoda, D. Meyer. *RFC 5771: IANA Guidelines for IPv4 Multicast Address Assignments* [Электронный ресурс]. — Страница в интернете. — Режим доступа: http://tools.ietf.org/html/rfc5771, свободный.

3. V. Fuller, T. Li. *RFC 4632: Classless Inter-domain Routing (CIDR): The Internet Address Assignment and Aggregation Plan* [Электронный ресурс]. — Страница в интернете. — Режим доступа: http://tools.ietf.org/html/rfc4632, свободный.

4. M. Cotton, L. Vegoda, et al. *RFC 6890: Special-Purpose IP Address Registries* [Электронный ресурс]. — Страница в интернете. — Режим доступа: http://tools.ietf.org/html/rfc6890, свободный.

5. Y. Rekhter, B. Moskowitz, et al. *RFC 1918: Address Allocation for Private Internets* [Электронный ресурс]. — Страница в интернете. — Режим доступа: http://tools.ietf.org/html/rfc1918, свободный.

6. R. Hinden, S. Deering. *RFC 4291: IP Version 6 Addressing Architecture* [Электронный ресурс]. — Страница в интернете. — Режим доступа: http://tools.ietf.org/html/rfc4291, свободный.

4 МАРШРУТИЗАЦИЯ В СЕТЯХ IP

В сетях IP каждому узлу известен собственный MAC-адрес, адрес IP и сеть IP, в которую входит данный узел (или, что то же самое, маска подсети). При отправке (или пересылке) данных известен адрес IP получателя. Маршрутизация требуется, если получатель находится в подсети, в которую отправитель не входит. В противном случае следует сформировать отправить сообщение в сеть получателя, сформировав пакет [1, подраздел 3.1], включающий заголовки сетевого и канального уровня, как показано на рис. 5.

Рисунок 5 — Примерная структура пакета сетевого уровня в сетях IP

В заголовке IP (сетевого уровня) указывается протокол вышестоящего уровня, данные которого передаются в пакете IP, — поле «Тип данных»; всем популярным протоколам присвоены коды [2]. Благодаря этому, получатель всегда может определить, пакет какого протокола доставлен. С другой стороны, маршрутизатор может анализировать тип данных с целью преобразования адресов и портов (см. лекцию 9), фильтрации и т. д.

4.1 Передача данных в пределах сети. Разрешение MAC-адреса по адресу IP

При передаче данных в пределах сети для формирования пакета недостает MAC-адреса получателя. В общем случае он известен только самому получателю. Однако, на канальном уровне доступна широковещательная рассылка. Можно отправить всем узлам запрос, на который должен ответить обладатель указанного в запросе адреса IP. Так действует *протокол разрешения адреса (Address Resolution Protocol, ARP),* приблизительная структура пакетов которого представлена на рис. 6 [3].

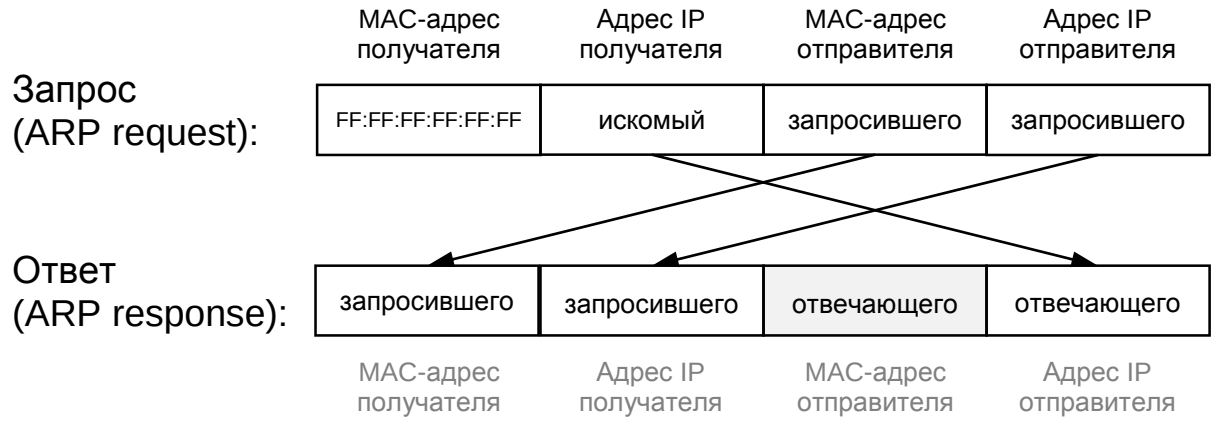


Рисунок 6 — Приблизительная структура пакетов ARP

Из рис. 6 видно, что узлу с искомым адресом IP доступны все необходимые данные для формирования ответа. В случае, если на запрос ARP ответа не приходит некоторое время, считается, что попытка разрешить адрес окончилась неудачно, и узла с нужным адресом IP в сети нет.

Таким образом, передача данных в пределах сети состоит из установления MAC-адреса получателя и отправки непосредственно сообщения. Выяснять MAC-адрес перед отправкой каждого пакета нерационально, поэтому каждый узел сохраняет результат для использования в будущем.

Существует и обратная процедура для установления адреса IP и соответствующий протокол — *Reverse ARP (RARP).* Узел запрашивает адреса IP всех остальных узлов сети и получает множество ответов, по которым может узнать, какие узлы и под какими адресами входят в сеть. Это может быть нужно маршрутизаторам. В настоящее время RARP считается устаревшим, поскольку его задачи решаются DHCP.

4.2 Передача данных между сетями

4.2.1 Таблица маршрутизации

В случае, если узел-получатель не входит в сеть, доступную напрямую узлу-отправителю, требуется направить пакет маршрутизатору, которому доступна такая сеть. Для этого требуется список известных маршрутизаторов в подключенных сетях и информация, какие сети им доступны. Кроме того, на случай, когда нет сведений о доступности искомой сети известным маршрутизаторам, следует предусмотреть т. н.

маршрутизатор по умолчанию (default gateway). Перечисленные данные составляют *таблицу маршрутизации (routing table).* На рис. 7 приведен пример «сложной» сети.

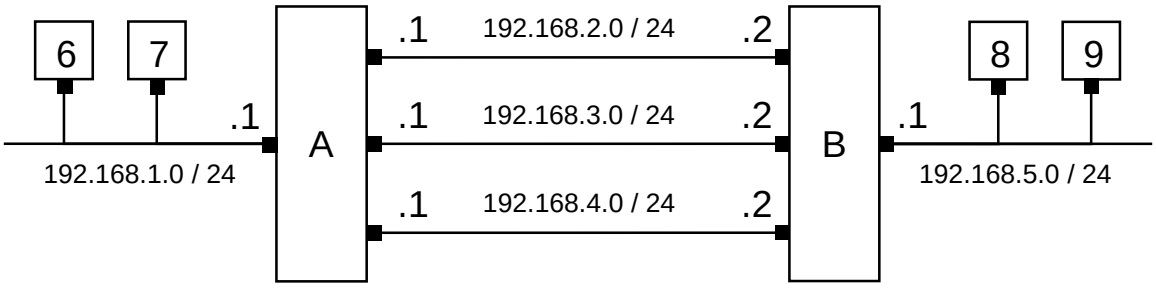


Рисунок 7 — Пример сети, в которой необходима маршрутизация

В сеть на рис. 7 входят 5 подсетей: 192.168.1.0/24, 192.168.2.0/24, …, 192.168.5.0/24 и два маршрутизатора: A и B, каждый с четырьмя интерфейсами в разных сетях. Ни один маршрутизатор не входит сразу во все подсети, следовательно, им необходимы таблицы маршрутизации; пример таковой приведен в таблице 3.

Таблица 3 — Таблица маршрутизации узла A для сети на рис. 7

Целевая подсеть	Маска подсети	Шлюз	Интерфейс
192.168.1.0	/24	192.168.1.1	192.168.1.1
192.168.2.0	/24	192.168.2.1	192.168.2.1
192.168.3.0	/24	192.168.3.1	192.168.3.1
192.168.4.0	/24	192.168.4.1	192.168.4.1
0.0.0.0	/32	192.168.3.2	192.168.3.1
192.168.5.0	/24	192.168.2.2	192.168.2.1
192.168.5.0	/24	192.168.3.2	192.168.3.1
192.168.5.0	/24	192.168.4.2	192.168.4.1

Смысл столбцов таблицы 3 таков: куда можно отправить пакет («Целевая подсеть» и «Маска подсети»), через какой узел («Шлюз»), и каким из собственных интерфейсов для этого следует воспользоваться («Интерфейс»). Строка с целевой подсетью 0.0.0.0/32 означает маршрут по умолчанию. По умолчанию в таблицу маршрутизации

заносятся сведения только о ближайших сетях, но могут помещаться любые, в данном случае — о доступе к сети 192.168.5.0 (серые строки таблицы 3).

Пусть узел 192.168.1.6 отправляет пакет узлу 192.168.5.9. Адресат находится вне текущей сети, поэтому пакет направляется на маршрут по умолчанию для 192.168.1.6 — на 192.168.1.1 (интерфейс маршрутизатора А). Маршрутизатор А определяет, что адресат находится в подсети 192.168.5.0/24. Из таблицы маршрутизации известно 4 способа передачи пакета в целевую подсеть: через узлы 192.168.2.2, 192.168.3.2 и 192.168.4.2 либо маршрутом по умолчанию через 192.168.3.2. В любом случае, пакет передается маршрутизатору Б, который обнаруживает (по аналогичной таблице), что его интерфейс 192.168.5.1 подключен к целевой подсети, и маршрутизация завершена. Далее будет получен MAC-адрес узла 192.168.5.9 и пакет будет передан внутри своей сети.

4.2.2 Алгоритмы маршрутизации

В примере встретилась ситуация, когда пакет мог быть отправлен по различным маршрутам, и возникает естественный вопрос, как делается выбор. Ответ таков, что одни маршруты предпочтительнее других, и маршрутизация должна быть оптимальной. Критерий оптимальности может быть различным (время или стоимость передачи, вид пакета, загрузка сети). В любом случае правилу в таблице маршрутизации сопоставляется некая *метрика (metric)* — показатель того, насколько «выгодно» отправлять пакет по данному маршруту, число от 0 до 255. Меньшая метрика является лучшей, метрика маршрута по умолчанию — наивысшая (худшая). Конкретные способы пересылки сообщений, учета и коррекции метрик маршрутов называются *алгоритмами маршрутизации* [4, гл. 6, раздел 3].

Неадаптивные алгоритмы не учитывают изменения в топологии и режиме работы сети, например, увеличение нагрузки, для коррекции метрики. Среди них:

1. *Лавинная рассылка, или затопление (flooding),* при которой маршрутизатор пересылает пакет во все доступные сети, кроме той, откуда пакет прибыл. Такой подход гарантирует доставку, но создает большую нагрузку на сеть.
2. *Пересылка в случайную сеть* (кроме сети отправителя). Доставка пакета не гарантируется, поэтому в общем случае алгоритм непригоден. Метод полезен при специальном построении сети, когда требуется распределить нагрузку равномерно между несколькими маршрутами.

Адаптивные алгоритмы маршрутизации учитывают изменения в топологии сети и режиме работы сети, производят специальные замеры и расчет параметров сети. Кроме того, маршрутизаторы могут обмениваться известной им информацией по специализированным протоколам. Выделяют алгоритмы *маршрутизации по состоянию канала (link-state routing, LSR)* и *дистанционно-векторной маршрутизации (distance vector routing, DVR).* В случае LSR каждый маршрутизатор строит граф-модель *всей* сети, вычисляет оптимальный маршрут *целиком* и передает пакет следующему узлу в рассчитанном маршруте. При использовании DVR маршрутизатор оперирует только условными расстояниями (метриками) от самого себя до известных узлов или подсетей без учета схемы всей сети. LSR почти оптимальна, но и более требовательна к ресурсам маршрутизатора, чем DVR, которая массово используется в небольших однородных сетях.

Никакой алгоритм маршрутизации не должен пересылать пакет в сеть, из которой пакет прибыл, иначе процесс передачи зациклится. Однако кольцевые маршруты все равно могут возникнуть. Во избежание бесконечной маршрутизации пакета в заголовке IP устанавливается наибольшее число пересылок — т. н. *время жизни пакета (time-to-live, TTL),* которое уменьшается с каждой пересылкой. Пакеты с нулевым TTL отбрасываются (reject, drop), и считается, что их не удалось доставить. Одна пересылка пакета, или шаг маршрутизации, называется *hop* (англ. «прыжок», не переводится). Иногда в них приближенно измеряют время передачи [1, подразделы 1.4 и 3.1].

4.2.3 Протокол межсетевых управляющих сообщений (ICMP)

В настоящее время практически повсеместно используются адаптивные алгоритмы маршрутизации, применяющие обмен информацией о состоянии сети по *протоколу межсетевых управляющих сообщений (Internet Control Message Protocol, ICMP).*

ICMP позволяет [5]:

1. Запрашивать и сообщать информацию о топологии и состоянии сети, об имеющихся маршрутизаторах (router advertisement), их настройках и возможностях.
2. Сообщать об ошибках передачи пакетов. В частности, если пакет был сброшен, по ICMP передается причина: узел отсутствует, истекло TTL и т. п.
3. Производить исследование и диагностику работы сети. Например, возможно проверить доступность узла или определить используемый маршрут до него.

4.2.3.1 Проверка доступности узлов

Проверка доступности некоторого узла основана на двух сообщениях ICMP: Echo Request (запрос отклика) и Echo Reply (отклик). При получении пакета ICMP Echo Request узел должен ответить пакетом ICMP Echo Reply. Если после отправки запроса отклик не получен некоторое время, считается, что удаленный узел недоступен. Возможно замерить время прохождения запроса и ответа (round-trip time, RTT) и рассчитать по серии запросов и ответов среднее время доступа к узлу. Этот процесс, а также программа для его проведения называется `ping`.

4.2.3.2 Отслеживание маршрута

Способ отслеживания маршрута до некоторого узла основан на сообщениях ICMP об ошибках передачи. Отправляется пакет ICMP Echo Request, назначением которого указан целевой узел, а TTL = 1. Первый же маршрутизатор, уменьшив TTL на 1 до 0, отбросит пакет и сообщит отправителю об этом пакетом ICMP Time Exceeded («время [жизни пакета] вышло»), который включает IP маршрутизатора. Отправляется следующий пакет на удаленный узел с TTL = 2, который будет аналогично отброшен вторым узлом маршрута. Процедура повторяется с увеличением TTL до тех пор, пока в ответ вместо ICMP Time Exceeded не будет получен пакет ICMP Echo Reply или сообщение о недоступности узла. Поскольку каждый пакет ICMP Time Exceeded содержит адрес IP отбросившего пакет узла, можно восстановить маршрут. Операция отслеживания маршрута и соответствующая ей программа называется `traceroute` (в ОС Windows доступна как `tracert`).

4.3 Литература к разделу 4

1. RFC 791: Internet Protocol.

2. Assigned Internet Protocol Numbers [Электронный ресурс]. — Страница в интернете. — Режим доступа: http://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml, свободный.

3. David C. Plummer. *RFC 826: An Ethernet Address Resolution Protocol* [Электронный ресурс]. — Режим доступа: http://tools.ietf.org/html/rfc826, свободный.

4. Internetworking Technologies Handbook, Fourth Ed. — Cisco Press, 2003. — 1128 с.

5. J. Postel. *RFC 792: Internet Control Message Protocol* [Электронный ресурс]. — Страница в интернете. — Режим доступа: http://tools.ietf.org/html/rfc792, свободный.

5 ТРАНСПОРТИРОВКА И НАДЕЖНАЯ ПЕРЕДАЧА ДАННЫХ

Транспортный уровень модели OSI расположен между сетевым и сеансовым уровнями: задача направления данных нужному узлу сети уже решена, а данные передаются или отдельными порциями, или в пределах сеанса связи. На транспортном уровне решаются следующие задачи:

1) адресация приложения в пределах узла, для чего вводится номер порта;
2) формирование целостных порций данных (datagrams);
3) обеспечение потоковой передачи (stream transmission).

Транспортный уровень тесно связан с сеансовым и часто неотделим от него.

5.1 Характеристики протоколов транспортного уровня

5.1.1 Надежность передачи

Протоколы с надежной (reliable) передачей данных гарантируют, что или будут доставлены все отправленные данные, или в случае неудачи будет обнаружена ошибка. *Ненадежная передача* допускает потерю части пакетов и не позволяет отследить, какие данные были доставлены, а какие были утеряны. Оба вида передачи разобраны подробно ниже.

5.1.2 Наличие соединения

С точки зрения модели OSI, вопрос о соединении относится к сеансовому уровню, однако на практике сеансовый и транспортный уровни часто совмещаются. Протоколы могут быть как *с наличием соединения (connection-oriented), так и без него (connectionless).* Соединение устанавливается между двумя узлами и позволяет каждому из них иметь уверенность, что другой остается доступен, пока соединение активно. Ясно, что при этом невозможна широковещательная или многоадресная рассылка, так как участников может быть сколь угодно много, а соединение устанавливается только между двумя из них.

5.1.3 Сохранение порядка данных

Сохранение порядка данных означает, что узел-получатель примет данные в том же порядке, в каком их выслал узел-отправитель. Доставка всех пакетов при этом может не быть гарантирована.

5.1.4 Сохранение границ сообщений

Размеры пакетов физического, канального и сетевого уровней могут значительно отличаться от размера порций данных, передаваемых с прикладного уровня. Несколько коротких сообщений могут быть переданы в одном пакете, а длинное сообщение может быть разбито на несколько пакетов. Протокол транспортного уровня, сохраняющий границы сообщений (message bounds), обеспечивает совпадение размеров принятого и отправленного сообщения. Это накладывает ограничение на максимальный размер сообщений, однако (в случае ненадежной доставки) гарантирует, что сообщение или не придет, или придет целиком. Если протокол не сохраняет границ сообщений, получатель принимает порции байт, которые могут быть частями или, наоборот, сцеплением нескольких сообщений.

5.2 Надежная передача данных. Протокол TCP[2]

Популярным протоколом транспортного и сеансового уровня для надежной передачи является *TCP (Transmission Control Protocol — англ. «протокол управления передачей»):* он требует соединения, сохраняет порядок сообщений, но не сохраняет границ между ними [1]. TCP является общепринятым решением для надежной передачи.

5.2.1 Квитирование

Квитирование — это подход к передаче данных, при котором на каждый пакет получатель высылает ответ-квитанцию об успешном приеме. Очевидно, что передача является надежной, если на каждый отправленный пакет получена квитанция о его приеме. В случае, если квитанция не приходит сразу, пакет считается утерянным, и выполняется его *повторная передача (retransmission).*

[2] Строго говоря, TCP означает «протокол управления передачей», и говорить «протокол протокол…» неправильно; однако, «протокол TCP» и т. п. является устоявшимся написанием в литературе.

Квитирование порождает т. н. «проблему двух генералов»: пакет-квитанция может быть утерян вне зависимости от того, успешна ли была передача пакета с данными. Теоретически, если утеряны все квитанции, узлы могут так и не узнать о том, состоялся ли вообще обмен данными. (Два генерала отправляют адъютантов с донесениями, но все гонцы могут пропасть, не добравшись до цели.) На практике задача решается ограничением на число повторных передач одного и того же пакета [2, подразделы 3.1 и 3.2].

Пример передачи с квитированием представлен на рис. 8. Процесс в нижней части рисунка — продолжение начатого в верхней; разрез показан штрихпунктирной линией.

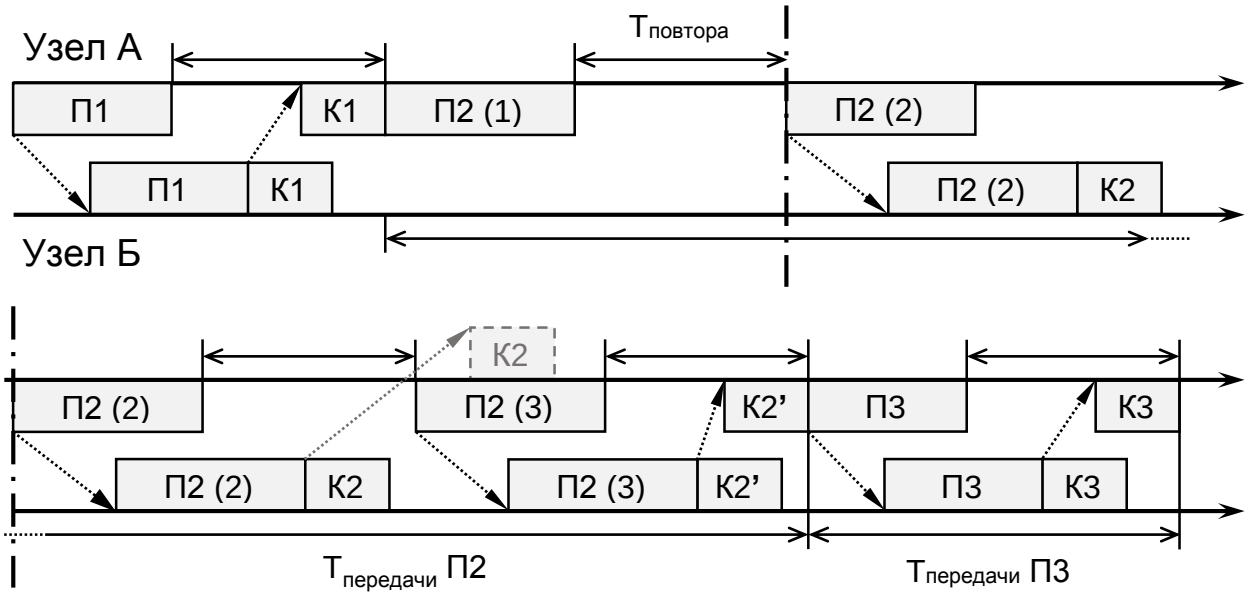


Рисунок 8 — Передача с квитированием

Узел А отправляет пакет П1, который успешно доставляется узлу Б. Узел Б отправляет квитанцию К1 о получении пакета П1, которая успешно доставляется узлу А. Передача пакета П1 окончена.

Затем узел А отправляет пакет П2, доставка которого не удается, поэтому узел Б квитанцию не отправляет. Не получив квитанции о доставке П2 в течение промежутка времени $T_{повтора}$, узел А отправляет пакет П2 заново, и тот успешно доставляется.

Узел Б отвечает квитанцией К2 о получении пакета П2, однако она теряется (или доставляется слишком поздно — см. «К2» пунктиром на рис. 8). Поскольку квитанция о получении П2 не приходит в течение заданного промежутка времени, узел А отправляет пакет П2 повторно, и он вновь доставляется успешно.

Узел Б повторно отправляет квитанцию о получении пакета П2, доставляемую успешно и вовремя. Передача пакета П2 окончена.

Наконец, узел А отправляет пакет П3, который доставляется с первой же попытки передачи, как и пакет П1 ранее. Время передачи для пакета П1 и П3 одинаково.

Время передачи пакета в представленной схеме работы — от начала его первой отправки до окончания получения квитанции на него. Если несколько попыток повторной передачи проваливаются, соединение считается разорванным (количество попыток определяется ОС). Время $T_{повтора}$ подбирается и динамически настраивается ОС, исходя из скорости доставки предыдущих пакетов.

5.2.2 Ускорение надежной передачи буферизацией

Из рис. 8 видно, что узел А ждет квитанции на каждый пакет прежде, чем передать следующий (подход «stop & wait»). Это не рационально: примерно половина времени тратится на ожидание квитанций. Можно было бы передавать пакеты непрерывно, не дожидаясь квитанций на предыдущие перед отправкой очередного и не отказываясь от квитирования (подход «pipelining»).

Однако, при таком методе порядок доставки пакетов может нарушиться. В одних случаях это происходит потому, что порядок доставки пакетов может отличаться от порядка отправки. Кроме того, после отправки нескольких пакетов может оказаться, что первый из них не был доставлен (не получено квитанции на него), и необходимо отправить этот пакет повторно.

Узел Б может решить проблему, игнорируя пакеты, номера которых нарушают непрерывно возрастающую последовательность. Однако, при этом неудача доставки всего одного пакета № N приведет к надобности повторной передачи пакетов с номерами $N + 1$, $N + 2$ и т. д., которые были отправлены, не дожидаясь квитанции на пакет № N. Поскольку эти передачи также могут не удаться, нагрузка на сеть лавинообразно возрастет.

Эффективным решением будет накопление всех принятых пакетов и передача их вышестоящим уровням только тогда, когда и все предыдущие пакеты успешно приняты. Если размер буфера приема — N пакетов, и один пакет не был доставлен сразу, можно принять $(N - 1)$ следующих пакетов, пока первый пересылается повторно. Узел А также вынужден держать буфер отправки и удалять из него пакеты только по приходе квитанций

на них. Временная диаграмма передачи потока данных с буферизацией представлена на рис. 9 (с. 31). Размеры буферов источника и приемника — 2 пакета.

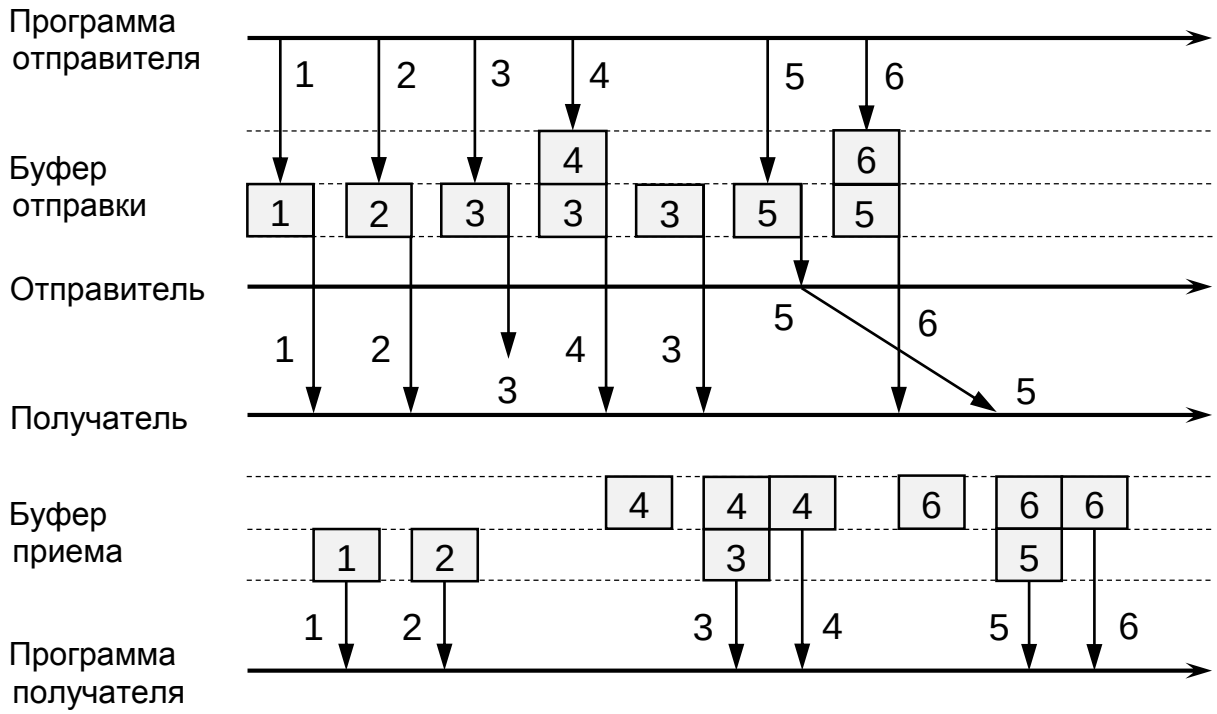


Рисунок 9 — Временная диаграмма приема потока данных с буферизацией

Пакеты № 1 и № 2 доставляются в порядке отправки и выбираются программой получателя из буфера приема сразу же. На сторону отправителя пакеты удаляются из буфера отправки по приходу квитанции (на рис. 9 квитанции не показаны).

Доставка пакета № 3 не удалась, квитанции не передано, поэтому пакет № 3 не удаляется из буфера отправки. Однако программа отправителя передает пакет № 4 сразу же за пакетом № 3. Не дожидаясь квитанции на пакет № 3, узел-отправитель передает пакет № 4, который доставляется успешно раньше пакета № 3 и удаляется из буфера отправки. Получатель помещает пакет № 4 в буфер приема; однако этот пакет не может быть передан программе получателя, так как предыдущий пакет еще не принят. Пакет № 3 удается доставить при повторной передаче, после чего он удаляется из буфера отправки. Получатель выдает программе сразу два пакета из буфера, № 3 и № 4.

Пакеты № 5 и № 6 отправляются в правильном порядке, но пакет № 6 оказывается доставлен раньше пакета № 5 (по причине того, что они, например, могли быть переданы по разным маршрутам). Получатель вновь сохраняет пакет № 6 в буфере приема,

но не может передать его программе, пока не принят пакет № 5. Когда это происходит, программе получателя передаются 2 пакета сразу.

Программе получателя пакеты передаются из буфера в порядке возрастания номеров без пропусков (потерь) вне зависимости от того, по какой причине была нарушена очередность передачи по сети.

Рассмотренный метод позволяет добиться для надежной передачи пропускной способности, близкой к максимальной для канала. *Пропускная способность (throughput)* — это количество информации, передаваемой в единицу времени [бит/с]. Важно, что это интегральная, или усредненная, характеристика: два пакета одного и того же размера могут быть доставлены за разное время. На примере рис. 9 видно, что, если отправка пакетов производится через равные промежутки времени, программе получателя они могут выдаваться неравномерно во времени. Пропускная способность ограничивается *задержкой передачи (latency)* — временем передачи пакета без учета времени его обработки и подтверждения. Источники задержки передачи — ожидание пакетом отправки в буфере, маршрутизация, а также операции на канальном и физическом уровне.

5.2.3 Скользящее окно

Большое влияние на передачу оказывает *размер скользящего окна (sliding window size)* — объем данных, которые источник может вслед отправить за первым пакетом, не дожидаясь подтверждения этого пакета. Понятие скользящего окна поясняется рис. 10.

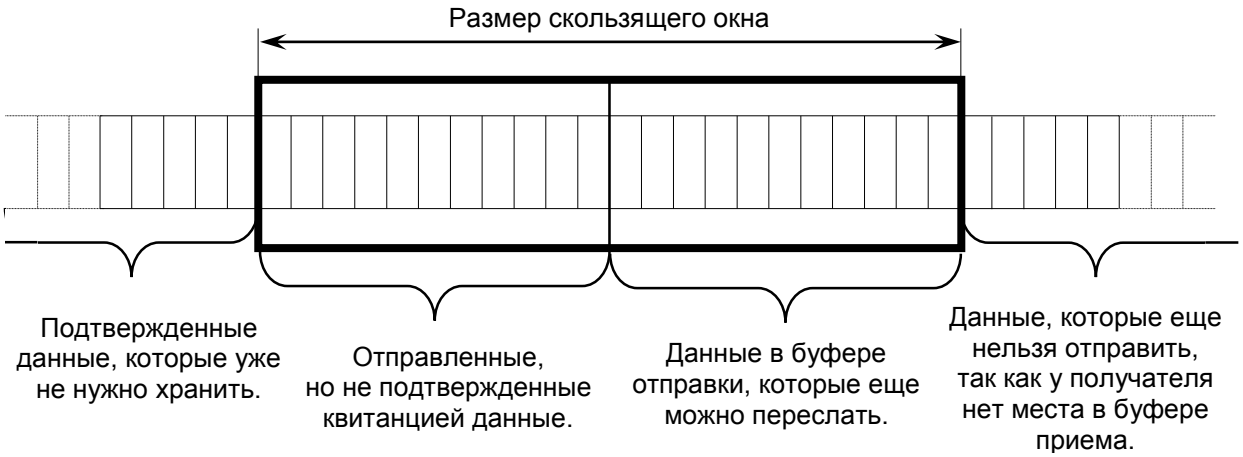


Рисунок 10 — Скользящее окно TCP с точки зрения отправителя

Скользящее окно не тождественно буферу приема или передачи: в первый попадают пакеты слева от окна, во втором хранятся находящиеся в пределах окна и справа от него.

И отправителю, и получателю необходимо хранить в буфере все пакеты, доставка которых не подтверждена, или доставленных вне очереди. Когда пакет у левой границы окна прибывает или подтверждается, окно сдвигается вправо. Размер окна ограничивает принимающая сторона: невозможно передавать данные быстрее, чем их принимают, поскольку «лишние» пакеты негде хранить, пока освободится часть буфера получателя. Рисунок 11 показывает, что наибольшая производительность достигается в случае, если к моменту заполнения буфера отправителя доставляется квитанция на самый ранний пакет окна, то есть, когда размер окна (по времени передачи всех пакетов в окне) равен времени прохождения запроса и ответа (round-trip time, RTT).

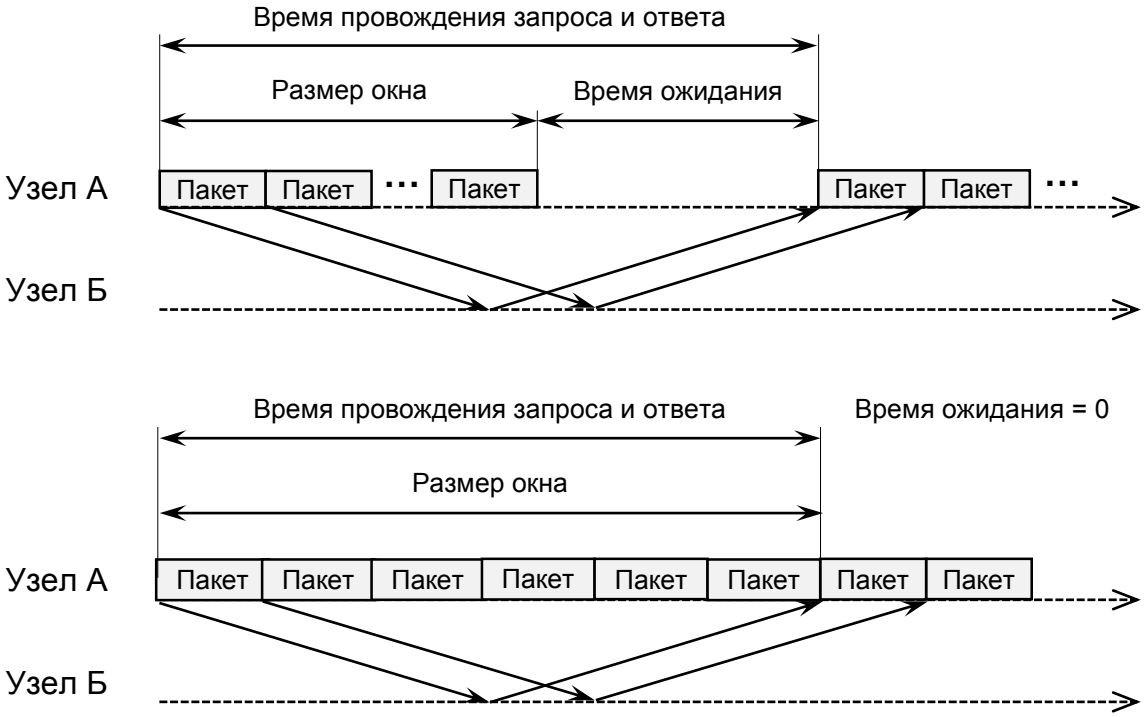


Рисунок 11 — Использование скользящего окна размером меньше RTT (сверху) и равного RTT (снизу)

Типовой размер окна — от единиц килобайт до единиц мегабайт, размер пакета — от единицы килобайт, в различных задачах эти величины могут существенно отличаться. Крупные пакеты или малая скорость связи (большое время передачи пакета) требуют повышения размера окна. Каждое соединение использует собственную пару буферов отправки и приема, поэтому при большом количестве подключений расходы памяти, связанные с буферизацией, существенны (пример: высоконагруженный сервер).

5.3 Ненадежная передача данных. Протокол UDP

Высокая пропускная способность выгодна при передаче больших объемов данных без потерь. Однако иногда важнее бывает *задержка (lag)* — наибольшее время передачи одного пакета при условии, что он доставлен. Например, при передаче потокового видео наиболее нежелательны задержки кадров. В этом случае разумно применить протокол с ненадежной доставкой: время будет тратиться только на передачу данных и никогда — на ожидание. Большую часть задержки составляет при этом время передачи, которое уже не может быть уменьшено протоколами транспортного уровня. Установка соединения при этом не требуется, а полезна возможность вещания сразу нескольким приемникам.

Наиболее популярным протоколом для описанных целей является *UDP (User Datagram Protocol).* Сообщения UDP называются *дейтаграммами (datagram).* UDP работает без соединения, не гарантирует доставки и порядка данных, но сохраняет границы между сообщениями [3]. UDP обычно используется:

1) для трансляции аудио и видео;
2) в играх и других приложениях, где важна задержка (lag);
3) при широковещательной или многоадресной рассылке.

Встречаются случаи, когда большая часть данных может быть передана по UDP, но некоторые порции следует доставить надежно. Одним выходом является использование соединения TCP параллельно с UDP. Другая возможность — реализация квитирования важных пакетов на прикладном уровне (используя UDP). Популярным решением второго рода является протокол и библиотека Reliable UDP (RUDP).

5.4 Литература к разделу 5

1. RFC 793: Transmission Control Protocol [Электронный ресурс]. — Страница в интернете. — Режим доступа: http://www.ietf.org/rfc/rfc793.txt, свободный.

2. E. A. Akkoyunlu, K. Ekanadham, R. V. Huber. *Some constraints and tradeoffs in the design of network communications* // Proceedings of the fifth ACM symposium on Operating systems principles. — New York, ACM. — 1975 г. — С. 67—74.

3. RFC 768: User Datagram Protocol [Электронный ресурс]. — Страница в интернете. — Режим доступа: http://tools.ietf.org/rfc/rfc768.txt, свободный.

6 ПОДХОДЫ ПРИКЛАДНОГО УРОВНЯ МОДЕЛИ OSI

На прикладном уровне модели OSI осуществляется обмен данными, специфичными для конкретных приложений. С точки зрения пользователя, вся полезная информация передается на этом уровне. Протоколы прикладного уровня настолько же многообразны, как и решаемые приложениями задачи.

6.1 Характеристики протоколов прикладного уровня

6.1.1 Формат данных

Формат данных может быть двоичным или текстовым. Двоичные протоколы используют такое же представление информации, какое используется в памяти ЭВМ — набор байт, удобный для работы ЦП. Текстовые протоколы представляют информацию в виде, пригодном для чтения человеком. Некоторые протоколы являются комбинированными: например, FTP (File Transfer Protocol) использует текстовый протокол для команд, а файлы передает «как есть» в виде двоичных данных. См. подробнее подраздел 6.2.

6.1.2 Наличие состояния

Протоколы с состоянием (stateful) предполагают, что каждая сторона имеет состояние, то есть хранит некие данные, от которых зависит обработка сообщений; также состояние может иметь сам сеанс связи. Например, в FTP большая часть команд может использоваться только после предъявления клиентом пароля (состояние сеанса — произведена ли аутентификация). В *протоколах без состояния (stateless),* например, в HTTP (см. пункт 6.3.1), смысл каждого сообщения целиком заключен в нем самом и не зависит от предшествующего взаимодействия (см. пункт 6.3.3).

6.1.3 Задача управления или передачи данных

Протоколы можно условно разделить на два вида. Задачи одних заключаются, по сути, в запросе и получении некоторой информации, других — в отправке команд и получении реакции на них. Разумеется, деление нестрогое: запрос информации тоже может рассматриваться как команда, а реакцией на команду может быть выдача полезной

информации. Важно то, что в первом случае одна сторона («клиент») обращается к другой («серверу») за некоторыми ресурсами (файлами, записями в БД и т. п.), а во втором случае клиент требует от сервера выполнения определенных действий. Подходы могут быть взаимодополняющими и не являются взаимоисключающими, а применяются сообразно логике задачи.

6.2 Двоичные и текстовые протоколы

Разница между двоичными и текстовыми протоколами напоминает отличия двоичных и текстовых файлов, однако отличаются приоритеты характеристик.

6.2.1 Характеристики

1. Компактность представления

Двоичное представление данных, как правило, короче текстового. Например, число 100000, записанное как строка, занимает 7 символов (не менее семи байт), а в двоичном виде хватило бы четырех байт. В двоичных протоколах часто применяют представления длиной менее байта (например, 1 бит на логическое значение).

2. Удобство приема и разбора (parsing)

В случае двоичного протокола удобно предварять сообщения их длиной, если сохранение границ сообщения не обеспечивается сетевым протоколом. Это позволяет просто и эффективно выделять память под принимаемые сообщения и считывать их. Разбор сообщения представляет собой простое копирование значений из пакета в переменные программы. Иногда при этом требуется делать несложные преобразования.

При работе с текстовыми протоколами, когда протокол сетевого уровня не сохраняет границ между сообщениями, приходится использовать маркер конца сообщения (например, перевод строки) и искать его в принятых данных. Из полученного текста необходимо выделять значения и преобразовывать их в двоичный формат, на что тратится время работы процессора.

3. Возможность чтения человеком и использования вручную

Это является преимуществом при диагностике, отладке или в отсутствие специального ПО. Последнее бывает актуально при решении задач на удаленных серверах.

6.2.2 Особенности двоичных протоколов

Двоичные протоколы более компактны и удобны для разбора программой, однако их разработка сложнее, а использование вручную практически невозможно.

6.2.2.1 Вопросы, решаемые при проектировании

При сетевом взаимодействии проблема совместимости представления типов данных является крайне актуальной: оборудование и ПО узлов разнообразно, тем более, что серверы используют специальные ОС и особые модели ЦП, ОЗУ и других компонентов.

Любой двоичный протокол должен точно определять:

1) перечень используемых элементарных типов (виды чисел, их точность);
2) размеры типов в байтах;
3) порядок байт в машинных словах (endianness, или byte order);
4) форматы используемых типов данных (например, представление целых чисел со знаковым битом или действительных чисел по стандарту IEEE 774).

6.2.2.2 Область применения

1. Передача информации «как есть», например, загрузка файлов.
2. Высокопроизводительные и высоконагруженные решения, в которых требуется обрабатывать большой объем данных в реальном времени. Примерами служат системы управления и диагностики, видеоигры.
3. Передача видео, звука и «сырых» данных, представление которых в понятном человеку виде бессмысленно или слишком громоздко.

6.2.3 Особенности текстовых протоколов

Сообщения текстовых протоколов объемнее, а их разбор программой сложнее, однако разработка текстовых протоколов проще, и существует возможность использовать их в «ручном» режиме, например, читая и набирая сообщения как простой текст.

6.2.3.1 Вопросы, решаемые при проектировании

1. Передача двоичных данных в текстовом виде

Бывает необходимо в преимущественно текстовых данных передавать двоичные фрагменты, например, изображения в электронной почте. Приходится использовать особые способы кодирования, например, base64, что приводит к увеличению размера сообщений.

2. Кодировка текста

Кодировка — это сопоставление символов и числовых кодов, которыми они представлены в памяти компьютера. Знание кодировки критично при обработке любого текста, в частности при разборе сообщения текстового протокола.

3. Форматы записи различных типов данных

Способы записи чисел и дат неоднозначны: «1,000» может означать как единицу, так и тысячу, «5/9/14» — как 5 сентября, так и 9 мая 2014 года. Стараются использовать как можно более жесткие, стандартные, универсальные и однозначные форматы, чтобы упростить разбор сообщений.

6.2.3.2 Область применения

1. Простые решения, не требовательные к скорости работы.
2. Передача преимущественно текстовой информации. Яркий пример — HTTP, используемый для web-страниц.

6.3 Открытые протоколы World-Wide Web

6.3.1 Протокол передачи гипертекста (HTTP)

Протокол передачи гипертекста (HyperText Transfer Protocol, HTTP) — текстовый протокол без состояния, предназначенный для доступа к ресурсам «всемирной паутины» (world-wide web) [1, 2]. HTTP разработан в 1989 командой из CERN (Швейцария) во главе с Тимом Бёрнерсом-Ли (Tim Berners-Lee) для проекта «всемирной паутины» с целью передачи web-страниц. Протокол успешно используется в таком качестве по настоящее времени, будучи модернизирован.

Протокол предполагает *запрос (request)* клиента к определенному ресурсу на сервере, используя для этого *универсальный идентификатор ресурса (Uniform Resource Identifier, URI).* Запросы могут быть нескольких видов — на совершение с ресурсом определенных действий (HTTP verb, или method), основные из которых [2, раздел 4]:

1) получение ресурса (`GET`);
2) размещение на сервере новых пользовательских данных (`POST`);
3) помещение данных на сервер по указанному адресу (`PUT`);
4) удаление ресурса (`DELETE`) и другие.

Например, получить титульную страницу сайта кафедры Управления и информатики можно, выполнив запрос с действием `GET` к ресурсу `/` («корень») на сервере `uii.mpei.ru`. Это и делает браузер при переходе по адресу `http://uii.mpei.ru` («`http`» указывает на используемый протокол).

Ответ (response) сервера предваряет код, характеризующий в целом результат запроса [2, раздел 6], в частности: 200 — успешное выполнение, 404 — ресурс не найден, 500 — произошла ошибка по вине сервера, и другие.

И запрос, и ответ могут включать, помимо полезных данных («тела», body), набор заголовков (headers) — именованных параметров для управления взаимодействием по HTTP. Например, заголовок «`Content-Length: 10000`» уведомляет клиента, что размер тела ответа — 10,000 байт [1, п. 3.3.2].

6.3.2 Единый интерфейс шлюза (CGI) и отдельные web-приложения

Программы, обслуживающие запросы по HTTP, называются *web-серверами.* На 2015 год самыми популярными являются nginx, Apache и MS Internet Information Services (IIS). В простейшем случае в ответ на запросы они выдают содержимое файлов на сервере — статические материалы. Однако очень часто ответ формируется динамически на основе запроса, информации в базе данных на сервере и т. п. Например, отобразить на странице IP-адрес клиента можно только таким образом.

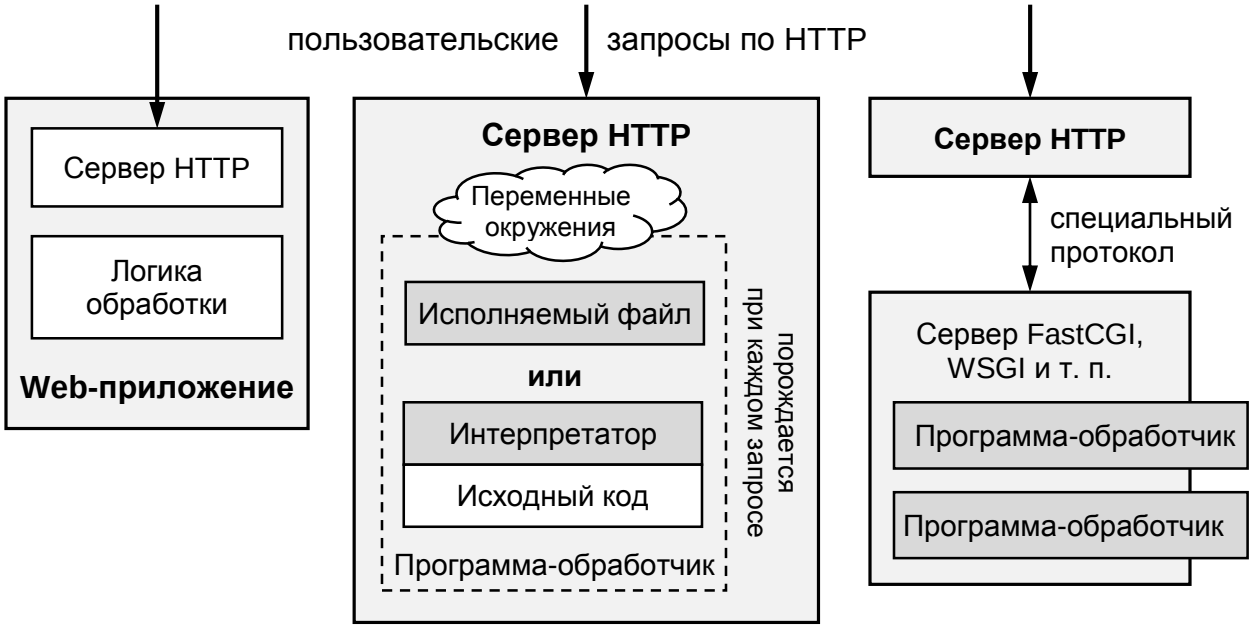
Web-сервер общего назначения не занимается формированием тела ответа; вместо этого он выполняет другую программу-обработчик, которой передается запрос, и от которой ожидается ответ (речь идет только о заголовках и теле). Интерфейс взаимодействия между web-сервером и программой, формирующей ответ, и называется *единым интерфейсом шлюза (Common Gateway Interface, CGI)* [1]. CGI позволяет программе-обработчику получить информацию о запросе. В свою очередь, обработчик может напечатать не только тело ответа, но и заголовки. Обработчик может быть любой программой: исполняемым файлом (EXE), shell-скриптом, сценарием на языках, предназначенных для web-страниц (PHP, Perl) и т. д.

Крупные или высоконагруженные web-приложения выполняют и в виде отдельных программ, исполняющих как роль web-сервера, так и роль обработчика. Это типичный подход в Java, Python и других языках, называемый *отдельными (standalone)*

web-приложениями. Метод позволяет добиться наилучшей производительности программы (не требуется создание процесса на каждый запрос), однако тратится больше ресурсов сервера (ОЗУ, время ЦП) и усложняется сама программа (в неё встраивается сервер HTTP). Так, web-сервер общего назначения может предоставлять доступ по CGI к десяткам и сотням сайтов, в то время как standalone-приложение будет обслуживать единицы (вероятно, крупных или популярных).

Получили популярность и гибридные решения, позволяющие использовать web-сервер общего назначения, но избавиться от запуска процесса или даже интерпретации программы при каждом запросе. Программы-обработчики запускаются служебным процессом на длительное время и могут обрабатывать много запросов, однако, не передают данные непосредственно по сети. Вместо этого служебный процесс по специальному протоколу транслирует запросы от web-сервера работающим программам-обработчикам и обратно — сформированные ответы. Единый стандарт для гибридных решений отсутствует; популярны реализации интерфейса ISAPI, протоколов FastCGI [4] и WSGI.

Рисунок 12 иллюстрирует описанные выше архитектуры web-приложений. Серым цветом изображены активные процессы, белым — элементы программ и данные.



а) отдельное б) с использованием CGI в) гибридное решение

Рисунок 12 — Виды web-приложений, доступных по HTTP

6.3.3 Архитектура REST

HTTP разрабатывался для передачи гипертекста (документа со ссылками), но является достаточно универсальным. В первые десятилетия XXI века приобрел популярность подход, где HTTP используется для доступа к любым ресурсам, даже не являющимся web-страницами или файлами — *representational state transfer (REST).*

Основой стандарта REST являются свойства стандартных методов HTTP: `GET` не изменяет ресурс и по умолчанию результат можно кэшировать (метод безопасный, safe), одинаковые запросы с `POST` не приводят к созданию дубликатов (метод идемпотентный, idempotent) и другие [2, подраздел 4.2]. Результатом является универсальный, удобный и расширяемый подход к управлению любыми ресурсами при помощи единого стандартного протокола.

Пример. В одном из популярных online-сервисов можно добавить к размещенному документу комментарий, выполнив запрос HTTP с методом `POST` («разместить») к ресурсу `/files/документ/comments` («комментарии к документу»). Применение к тому же ресурсу метода `GET` позволяет получить список комментариев [3].

6.3.4 Удаленный вызов процедур (RPC)

Удаленный вызов процедур (remote procedure call, RPC) — это подход, при котором клиент требует от сервера выполнить определенную процедуру, передав её наименование и аргументы. Логика работы подобна вызову функций в языках программирования, однако команда вызова передается по сети, и функция исполняется на другом узле. Ответ сервера — это результат вызова функции.

Как правило, протоколы RPC имеют состояние, изменяемое вызовами функций. Часто, но не всегда, обработчики RPC достаточно сложны, чтобы реализовывать их в виде standalone-серверов.

RPC — это подход, а не конкретный протокол. Техническая реализация прежде всего требует, чтобы вызов процедуры (имя действия, аргументы) был каком-либо образом оформлен. Представление логической или программной сущности для сохранения или передачи называется *сериализацией (serialization).* Протокол SOAP (Simple Object Access Protocol) предлагает способ сериализации, приведенный в листинге 1.

```
<m:GetStockPrice xmlns:m="http://example.org/stock">
      <m:StockName>IBM</m:StockName>
</m:GetStockPrice>
```

Листинг 1 — Фрагмент запроса RPC по протоколу SOAP

SOAP использует *расширяемый язык разметки (eXtensible markup language, XML).* Имя вызываемой функции — `GetStockPrice`, а значение её параметра под названием `StockName` — «IBM».

Задача SOAP — только сериализация; протокол не оговаривает порядка сетевого взаимодействия. Осуществить удаленный вызов можно, отправив сообщение SOAP по протоколу, задачей которого является только передача запроса, получение ответа и обнаружение ошибок (например, вызова неизвестной серверу функции). Такой протокол называется *транспортом (transport).* Им может выступать, например, HTTP: у него задана процедура обмена данными, есть средства управления взаимодействием (заголовки) и диагностики ошибок (код ответа); методом можно выбрать, например, `POST`.

Таким образом, прикладной уровень делится на 2 подуровня:

1) на верхнем выполняется RPC при помощи сообщения SOAP;
2) на нижнем к серверу делается HTTP-запрос с телом-сообщением SOAP.

Говорят, что работает «SOAP over HTTP», «SOAP поверх, или через, HTTP». При этом задействованы и протоколы всех нижележащих уровней: TCP, IP и другие.

6.4 Литература к разделу 6

1. R. Fielding (Ed.), J. Reschke (Ed.). *RFC 7230: Message Syntax and Routing* [Электронный ресурс]. — Режим доступа: http://tools.ietf.org/html/rfc7230, свободный.

2. R. Fielding (Ed.), J. Reschke (Ed.). *RFC 7231: Semantics and Content* [Электронный ресурс]. — Режим доступа: http://tools.ietf.org/html/rfc7231, свободный.

3. RFC 3875: The Common Gateway Interface (CGI) Version 1.1 [Электронный ресурс]. — Режим доступа: http://tools.ietf.org/html/rfc3875, свободный.

4. Mark R. Brown. *FastCGI Specification* [Электронный ресурс]. — 26 апреля 1996 г. — Режим доступа: http://www.fastcgi.com/drupal/node/6?q=node/22, свободный.

5. API Reference — Google Drive SDK — Google Developer [Электронный ресурс]. — Режим доступа: https://developers.google.com/drive/v2/reference/#Comments, свободный.

7 ЭТАПЫ ПРОЕКТИРОВАНИЯ СЕТЕВЫХ ПРОТОКОЛОВ НА ПРИМЕРЕ СИНХРОНИЗАЦИИ СПИСКА СТРОК

Решим задачу синхронизации списка строк между несколькими узлами сети. Участники могут добавлять строки в общий список и удалять их оттуда. На этом примере рассмотрим основные этапы и элементы проектирования протокола прикладного уровня.

7.1 Модель взаимодействия

Модель взаимодействия (communication model) — это логическое описание работы системы, основанной на протоколе. Технические подробности опускаются, хотя уже на данном этапе может подразумеваться надежная или потоковая передача данных или упоминаться сеанс связи. Обязательно определяются:

— виды и количество участников;

— задачи всех участников, их обязанности и гарантии;

— высокоуровневые операции и состояния участников (для stateful протоколов);

— этапы и процедуры взаимодействия.

Первый основополагающий выбор — будут ли все участники одинаковы по своим ролям или нет. Например, HTTP выделяет клиента и сервера один раз на все время соединения; SMTP позволяет клиенту и серверу в любой момент поменяться ролями; в протоколе BitTorrent все узлы равноправны, и каждый работает одновременно и как клиент, и как сервер. Число участников может фиксироваться (в HTTP и SMTP — строго две стороны) или ограничиваться (например, 5 клиентов на 1 сервер).

Распределение задач и обязанностей между участниками позволяет, во-первых, убедиться в выполнении всех необходимых задач, а во-вторых - установить, какой информацией располагают различные участники, когда и откуда она поступает.

Операции протокола, перечисляемые в модели взаимодействия, отражают логику работы участников; описание получается укрупненным. Например, в HTTP можно выделить 4 основных действия: установление и разрыв соединения, запрос и ответ;

при этом существует много видов запросов, а ответ на один запрос может состоять из нескольких пакетов (сообщений) HTTP.

Взаимодействие может состоять из нескольких этапов, на которых отличаются возможные операции и состояния, а также задачи и гарантии. Например, SMTP разрешает большинство действий только после процедуры авторизации.

Часто к модели взаимодействия добавляют концептуальную или историческую информацию: когда, где и зачем был разработан протокол, в каких случаях он рекомендован к применению, и другие сведения.

Пример.

На каждый синхронизируемый список приходится один сервер и до пяти клиентов. Все участники хранят собственные копии списка; версия сервера считается актуальной.

Клиенты отправляют серверу запросы-команды на добавление или удаление строк из общего списка; сервер выполняет команды над собственным списком. Далее все клиенты должны произвести те же манипуляции, поэтому сервер дублирует команды всем клиентам. Последние при получении команд изменяют собственные списки.

Алгоритм гарантирует, что сервер и каждый клиент получат одинаковые последовательности команд управления списком, следовательно, их списки будут синхронизированы. Некорректные команды игнорируются сервером.

7.2 Спецификация протокола

Спецификация протокола (specification, сокр. «spec») раскрывает техническое устройство протокола. Приводятся точные условия посылки сообщений и требования к реакции на них. Каждое сообщение описывается вплоть до отдельных полей или, для текстовых протоколов, до неделимых фрагментов. Фиксируются используемые типы данных, порядок байт, кодировки символов и т. п., чтобы обеспечить независимость протокола от программной и аппаратной платформы, где он будет реализован.

7.2.1 Типы данных

Типы данных, используемые протоколом синхронизации списка строк, приведены в таблице 4. Порядок байт в машинном слове — от старшего к младшему (big endian).

Таблица 4 — Типы данных, используемые протоколом

Тип	Описание	Размер	Примечания
`uint8`	беззнаковое целое	8 бит (1 байт)	
`uint32`	беззнаковое целое	32 бита (4 байта)	
`char`	символ ASCII	8 бит (1 байт)	код символа в кодировке ASCII; последний символ — с кодом 0.

7.2.2 Структура сообщения

Каждое сообщение начинается с числа (`uint32`) — своей длины в байтах, не включая сами 4 байта длины. Выбранный формат позволяет удобно разделять сообщения в потоке: после считывания длины известно, сколько байт требуется принять до конца пакета. Далее в описании длина не показывается в структурах сообщений.

После длины следует код команды (`uint8`), по которому участник может определить вид сообщения.

7.2.3 Команды клиентов

7.2.3.1 Добавить строку в общий список (add)

Смысл: клиент требует добавить строку в общий список.

Структура приведена в таблице 5.

Таблица 5 — Структура пакета с командой клиента «Добавить строку»

Поле	Тип данных	Назначение и примечания
`command`	`uint8`	Всегда 1.
`length`	`uint32`	Длина строки, не включая завершающий 0.
`string`	`char` (`length` + 1 штук)	Символы строки.

Сервер должен добавить строку `string` в конец общего списка и передать копию сообщения всем клиентам, включая приславшего.

7.2.3.2 Удалить строку из общего списка (remove)

Смысл: клиент требует удалить строку из общего списка.

Структура приведена в таблице 6.

Таблица 6 — Структура пакета с командой клиента «Удалить строку»

Поле	Тип данных	Назначение и примечания
`command`	`uint8`	Всегда 2.
`index`	`uint32`	Номер строки к удалению в общем списке.

Сервер должен удалить строку с номером `index` из общего списка и передать копию сообщения всем клиентам, включая приславшего.

7.2.4 Сообщения сервера

7.2.4.1 Очистить список (clear)

Смысл: сервер сообщает клиенту, что необходимо очистить список.

Структура приведена в таблице 7.

Таблица 7 — Структура пакета с командой сервера «Очистить список»

Поле	Тип данных	Назначение и примечания
`command`	`uint8`	Всегда 3.

Клиент должен очистить собственную копию списка строк.

7.2.4.2 Отклонение подключения (reject)

Смысл: сервер оповещает клиента, что его обслуживание производиться не будет.

Структура приведена в таблице 8.

Таблица 8 — Структура пакета с сообщением сервера об отклонении подключения

Поле	Тип данных	Назначение и примечания
`command`	`uint8`	Всегда 4.
`length`	`uint32`	Длина строки, не включая завершающий 0.
`string`	`char` (`length` + 1 штук)	Строка, содержащая причину отказа в обслуживании.

Клиент должен завершить соединение при получении, а сервер — сразу после отправки данного сообщения.

7.2.4.3 Добавить строку в общий список (add)

Смысл: сервер оповещает клиентов о добавлении строки в общий список.

Структура приведена в таблице 5 (выше).

Клиент должен добавить строку `string` в конец собственной копии списка.

7.2.4.4 Удалить строку из общего списка (remove)

Смысл: клиент требует удалить строку из общего списка.

Структура приведена в таблице 6 (выше).

Клиент должен удалить строку с номером `index` из собственного списка.

7.3 Моделирование и разбор примеров

Спецификация протокола не должна допускать различных толкований, в противном случае реализации протокола окажутся несовместимыми, а взаимодействие — невозможным. В этом причина формального и жесткого формата спецификации. Принято, однако, пояснять описание короткими примерами как типовых, так и граничных случаев. Пример работы протокола приведен на рис. 13 (с. 48).

Изначально списки строк на сервере и на узле клиента Б пусты (пустой прямоугольник), а в списке клиента А находится строка «А0» (возможно, сохранившаяся с предшествующего сеанса работы).

Клиент Б первым подключается к серверу, который немедленно отвечает командой очистить список, поскольку сервер не обладает сведениями о состоянии клиента Б. Далее клиент Б высылает серверу команду добавить в список строку «Б1». Сервер добавляет эту строку в собственный список и высылает всем подключенным клиентам (в данном случае, только Б) команду добавить строку «Б1» в свои списки. Только по её получении клиент Б заносит строку «Б1» в собственный список.

Клиент А подключается вторым. Сервер сразу же высылает клиенту А команду очистки списка, после чего список клиента А становится пуст. Далее сервер отправляет клиенту А команды на добавление всех имеющихся в общем списке элементов (в данном случае, только «Б1»), которые клиент А заносит в собственную копию списка.

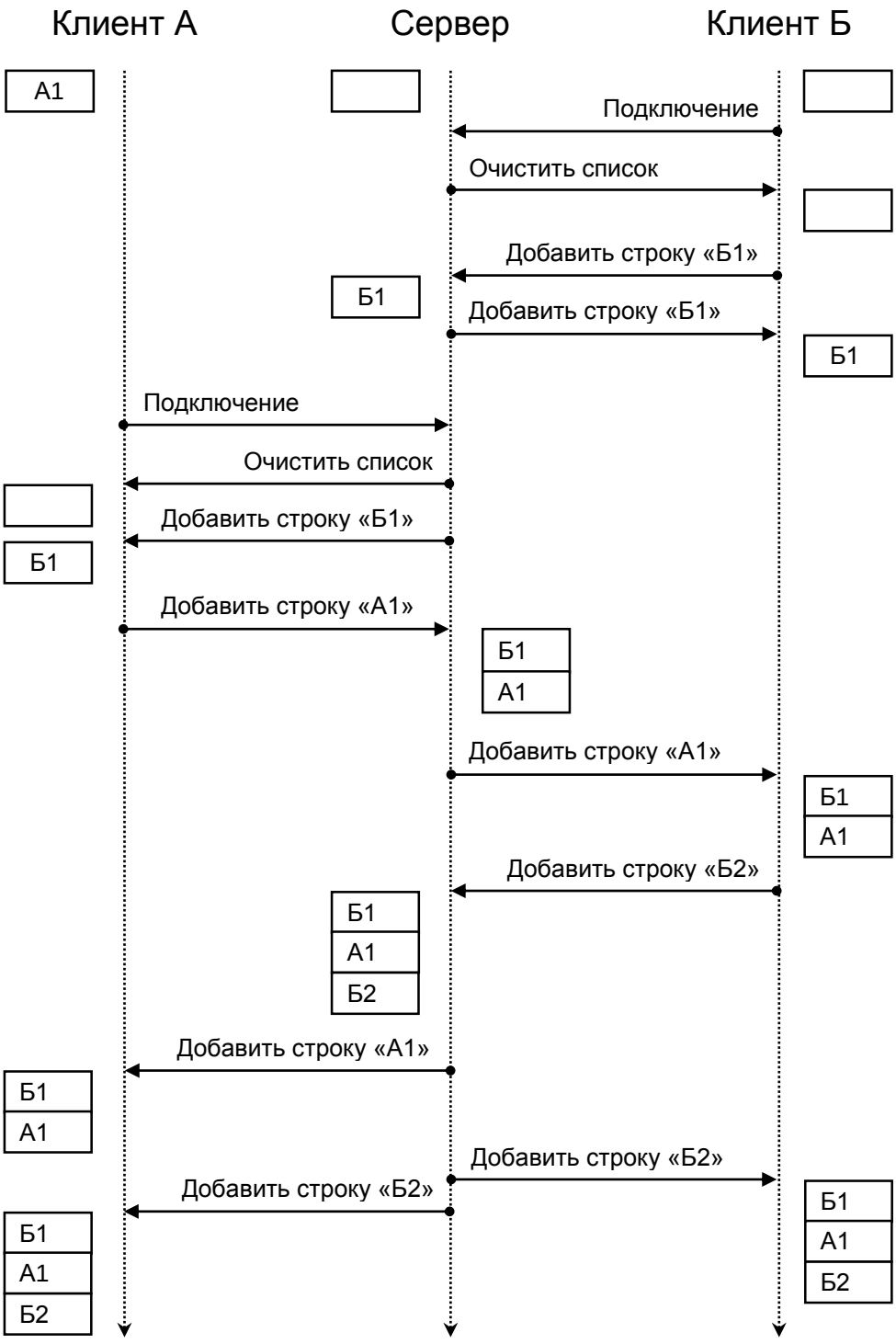


Рисунок 13 — Пример работы протокола синхронизации списка строк

Далее клиент А отправляет серверу команду добавить в список строку «А1». Сервер добавляет эту строку в собственный список и высылает копию команды всем подключенным клиентам: сначала Б, затем А. Прежде, чем сервер отправляет команду

на добавление строки «А1» клиенту А, клиент Б передает запрос на добавление строки «Б2». Сервер заносит строку «Б2» в собственную копию списка, однако, не производит рассылку команды на её добавление до тех пор, пока все клиенты не окажутся оповещены о добавлении предыдущей строки, «А1». Более обще: при получении команды сервер сначала оповещает всех клиентов копией команды, а затем приступает к обработке следующего сообщения. Благодаря этому, действия системы корректны вне зависимости от порядка оповещения клиентов, который зависит от работы ОС, от скорости передачи данных и других факторов.

Реакция сервера на команду удаления строки из списка была бы аналогична реакции на команду добавления. При отключении клиентов команда очистки списка им не высылается, поскольку узлам может быть нужна копия списка.

8 ПРИНЦИПЫ И МЕТОДЫ ЗАЩИЩЕННОЙ И ДОВЕРЕННОЙ СВЯЗИ

Защита информации (information security) заключается в обеспечении её конфиденциальности, целостности и доступности [1, с. 4]. К процедурам защиты информации относится шифрование (ограничение доступа), информационный аудит (проверка целостности), резервное копирование (предотвращение утраты) и другие. В случае сетевого взаимодействия пакет может быть:

1) потерян при маршрутизации;
2) поврежден при передаче (непреднамеренно, например, радиопомехами);
3) перехвачен злоумышленником;
4) изменен или заменен злоумышленником.

Угрозы 1) и 2) решаются на физическом, канальном, сетевом и транспортном уровнях. Рассмотрим процедуры защиты информации от угроз 3) и 4): обеспечение целостности, установление подлинности и шифрование.

8.1 Основы криптографии

Криптография (cryptography) — наука о математических методах обеспечения конфиденциальности и подлинности информации. Основным практическим результатом криптографии является шифрование — процесс преобразования *открытого текста (plaintext)* в *зашифрованный (cyphertext)* при помощи *ключа (key)* по определенному алгоритму — и обратный процесс, расшифровка [2, с. 25].

Исходные предположения криптографии таковы. Участники (Алиса и Боб) обмениваются сообщениями, а злоумышленник (Чарли или Ева) пытается получить доступ к открытому тексту. Считается, что злоумышленнику всегда известен алгоритм и зашифрованный текст. Шифрование должно в этих условиях обеспечивать невозможность расшифровки открытого текста злоумышленником. Невозможность означает такую же сложность, что и перебор всех ключей *(атака методом «грубой силы», brute force attack).*

Помимо знания алгоритма шифрования у злоумышленника может быть и другая информация или возможности. Различные случаи такого рода называются *моделями атак.*

8.1.1 Криптографические атаки и их модели

Криптографические системы подвергаются двум основным видам атак: на алгоритм и по сторонним каналам [2, подраздел 3.6]. Атака на алгоритм заключается в попытке преодолеть его математически. *Атаки по сторонним каналам (side channel attack)* используют тот факт, что практические реализации алгоритмов не идеальны, и анализируя их особенности, можно получить теоретически недоступную информацию для расшифровки. Последний вид атак наиболее продуктивен на практике и распространен в телекоммуникациях.

Модель *«человек посередине» (man-in-the-middle, MITM)* предполагает, что канал связи доступен злоумышленнику для наблюдения (пассивная атака) или для управления, включая изменение, замену и фабрикацию сообщений (активная атака).

Наиболее опасны для криптографических систем ошибки реализации и небрежность в соблюдении процедур. Например, часто некоторые данные требуется уничтожить («забыть»). Однако, если не предпринять специальных мер, информация может остаться в памяти ЭВМ и быть получена злоумышленником *(data remanence attack).*

8.1.2 Симметричная криптография

В симметричной криптографии для шифрования и расшифровки с обеих сторон алгоритм и ключ используются одни и те же.

Поскольку алгоритм шифрования считается известным злоумышленнику, применяться могут очень простые функции, например, «исключающее ИЛИ» (сложение по модулю 2, или XOR) между битами открытого текста и ключа. Эта операция, обозначаемая «⊕», удобна обратимостью (см. таблицу 9):

$$Cyphertext = Plaintext \oplus Key$$
$$Plaintext = Cyphertext \oplus Key,$$

Благодаря этому свойству, шифрование и расшифровка выполняются одинаково.

Таблица 9 — Таблица истинности операции «исключающее ИЛИ»

x	y	$x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0

Таблица 10 содержит пример симметричного шифрования (при чтении сверху-вниз) и расшифровки (при чтении снизу-вверх).

Таблица 10 — Пример симметричного шифрования

Элемент	Строка	Двоичные коды
Открытый текст	test	01110100 01100101 01110011 01110100
Ключ	1234	00110001 00110010 00110011 00110100
Зашифрованный текст	EW@@	01000101 01010111 01000000 01000000

Симметричной криптографической схеме необходимы процедуры, которые позволяют Алисе и Бобу получить ключи, не владея ими изначально, — алгоритмы *распространения ключей (key distribution).*

8.1.3 Протокол Диффи-Хеллмана

Популярным алгоритмом распространения ключей является т. н. протокол[3] Диффи-Хеллмана [3] для обмена ключами (Diffie-Hellman key exchange), приведенный в таблице 11.

Таблица 11 — Протокол Диффи-Хеллмана

Шаг	Алиса			Передача	Боб		
	скрыто	открыто	вычисления		вычисления	открыто	скрыто
1	a	p, g		$p, g \rightarrow$			b
2	a	p, g, A	$A = g^a \bmod p$	$A \rightarrow$		p, g	b
3	a	p, g, A		$\leftarrow B$	$B = g^b \bmod p$	p, g, A, B	b
4	a, s	p, g, A, B	$s = B^a \bmod p$ $= g^{ab} \bmod p$		$s = A^b \bmod p$ $= g^{ab} \bmod p$	p, g, A, B	b, s

Запись вида $c = a + b \mod N$ означает, что значение c равно остатку от деления $a + b$ на N, например, $(7 + 4 \mod 5) = 1$, так как $7 + 4 = 11 = 5 \cdot 2 + \mathbf{1}$.

1. Алиса и Боб заранее договариваются, что $p = 23$, $g = 5$ (простые числа). Они не представляют секрета и могут использоваться не только Алисой и Бобом.
2. Алиса выбирает число $a = 6$, после чего отправляет Бобу значение $A = g^{\boldsymbol{a}} \bmod p = 5^{\mathbf{6}} \bmod 23 = 8$.
3. Аналогично, Боб выбирает число $b = 15$ и отправляет Алисе значение $B = g^b \bmod p = 5^{15} \bmod 23 = 19$.
4. Алиса может вычислить $S = B^a \bmod p = 19^6 \bmod 23 = 2$. Боб может вычислить то же значение иным путем: $S = A^b \bmod p = 8^{15} \bmod 23 = 2$.

[3]«Протокол» здесь означает, что процедура описывает обмен информацией, хотя и выполняется по строгому алгоритму. Новых форматов данных не вводится.

Рисунок 13 иллюстрирует протокол Диффи-Хеллмана в упрощенном виде — со смешиванием красок вместо операций.

Результат: у Алисы и Боба есть общий ключ *s* (common secret), который никогда не передавался и неизвестен Чарли. При разумно подобранных константах вычислить *s* для Чарли практически невозможно. Здесь «разумно» означает соблюдение некоторых требований, обоснование которых выходит за рамки курса; как правило, требуются очень большие *простые*[4] числа или их произведения.

Константы *a* и *b* называются *закрытыми ключами (private keys)* Алисы и Боба соответственно, числа же *A* и *B* называются *открытыми ключами (public keys)*. На основе публичного и своих закрытых ключей участники формируют общий секрет, который затем используется как ключ для шифрования.

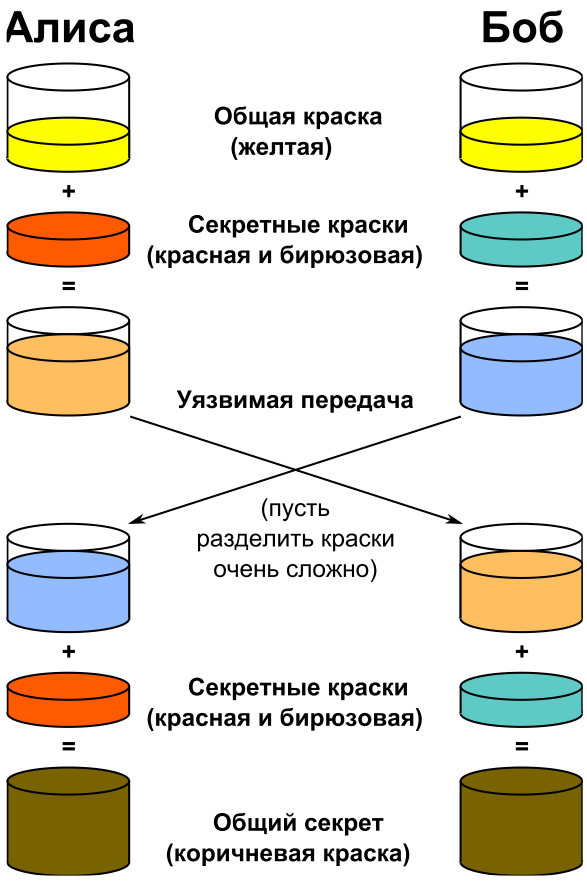


Рисунок 14 — Упрощенная форма протокола Диффи-Хеллмана

Предложенная процедура устойчива к пассивной атаке вида «человек посередине», при которой злоумышленник способен читать, но не изменять сообщения (eavesdropping). На практике это соответствует, например, приему всех передач в сети Wi-Fi или прослушиванию трафика в системе программой-сниффером (sniffer).

При активной атаке считается, что Чарли способен заменять сообщения собственными так, что Алиса и Боб не могут этого распознать, то есть, с их точки зрения, взаимодействие происходит напрямую. Такую атаку на практике можно осуществить, контролируя сетевое оборудование, например, установив программу на маршрутизатор или будучи поставщиком интернета. Протокол Диффи-Хеллмана неустойчив к активной

[4] Напомним: это натуральные числа, имеющие делителями только самих себя и единицу.

атаке. Действительно, в таком случае Чарли, с точки зрения Алисы, ничем не отличается от Боба, и они могут сформировать общий секрет. Аналогично, общий секрет (другой) могут сформировать Чарли (в роли Алисы) и Боб. Чарли сможет расшифровывать сообщения и от Алисы, и от Боба, чего и требовалось достичь.

Причина проблемы в том, что Алиса не проверяет, является ли собеседник Бобом на самом деле (и наоборот), то есть не выполняется проверка и подтверждение подлинности источника данных, а главное, самих данных — открытых ключей *A* и *B*.

8.1.4 Асимметричная криптография

В асимметричной криптографии для шифрования применяется один ключ (обычно открытый, public key), а для расшифровки — другой (обычно закрытый, private key). По одному ключу из пары определить второй практически невозможно. Ключи в паре математически связаны и подходят только один к другому, то есть, зашифровав данные ключом из одной пары, нельзя расшифровать их ключом из другой.

Алгоритм шифрования в асимметричной криптографии усложняется. «Исключающее ИЛИ» не подходит, поскольку должно быть невозможно получить из шифрованного текста исходный тем же ключом, который применялся для шифрования:

$$cyphertext = encrypt(plaintext, public\ key)$$
$$plaintext = decrypt(cyphertext, private\ key)$$

Открытые ключи публикуются свободно. Зашифровав сообщение открытым ключом Боба, Алиса может быть уверена, что расшифровку сможет осуществить только обладатель соответствующего закрытого ключа, то есть Боб; со стороны Боба процесс аналогичен. Секретные данные (закрытые ключи) никак не входят в состав передаваемых данных.

Чарли может осуществлять атаку двумя способами. Первый — подменить открытые ключи на собственные в момент их публикации, поскольку обмен ими криптографически не защищен (но может быть защищен организационно). Второй способ — отправлять Бобу сообщения, зашифрованные открытым ключом Алисы, выдавая их за написанные ею. Вновь проблемой является установление подлинности данных: ключей или сообщений.

Хотя асимметричная криптография и не решает проблему активной атаки «человек посередине», она часто применяется в случаях, когда безопасность обмена открытыми ключами обеспечена организационно. Главное же свое применение асимметричная криптография находит в задаче подтверждения подлинности (см. подраздел 8.3).

8.2 Необратимые функции и хэширование

Необратимой (one-way) называется такая функция $f(x)$, по значению y которой невозможно определить аргумент x, то есть неизвестна обратная функция $f^{-1}(y)$. Функция $f(x) = 7x$ является обратимой, поскольку $f^{-1}(y) = \frac{y}{7}$. Однако, необратима функция $g(x) = 7x \pmod 5$, так как $g(1) = g(6) = g(11) =_{...} = 2$, — по значению нельзя определить, был ли аргумент 1, 6, 11 или другим.

При защите информации широкое применение находят *хэш-функции (hash functions)* — необратимые функции, преобразующие данные любого объема в значения фиксированной длины, называемые *хэш-кодами (hash-code),* или просто *хэшами* [4, п. п. 3.1.13 и 3.1.14]. Пример (алгоритм SHA1, размер хэша — 20 байт):

```
SHA1("password") = 5baa61e4c9b93f3f0682250b6cf8331b7ee68fd8
SHA1("Password") = 8be3c943b1609fffbfc51aad666d0a04adf83c9d
```

Хэш-функции составляются так, чтобы малые изменения в данных приводили к сильному изменению хэша (т. н. *лавинный эффект,* как в примере). Это позволяет применять подвид хэш-функций, контрольные суммы, для защиты целостности данных.

Множество «любых данных» бесконечно, а множество значений фиксированной длины конечно, следовательно, некоторые различные входные данные будут иметь одинаковый хэш. Это называется *коллизией (collision)* и принципиально неустранимо. Если вычислительно сложно подобрать набор входных данных, дающий коллизию с известным хэшем, такая хэш-функция называется *криптографической.*

Пример. Пароли в БД обычно не хранят, а хранят их хэши. Когда выполняется проверка, достаточно вычислить хэш введенного пароля и сравнить с хранимым. В случае же, когда злоумышленник получает доступ к БД, узнать пароли ему не удастся.

В приведенном примере хэш-функция должна быть криптографической, иначе злоумышленник сможет подобрать пароль (не обязательно совпадающий с исходным), который даст тот же хэш и позволит аутентификацию. Понятие «вычислительно сложно» на практике условно: алгоритм MD5 был крайне популярен до середины 2000-х; о том, что он не является криптографическим, было известно с 1996 года (когда результат сочли теоретическим), а в 2005 году были разработаны практические методы нахождения коллизий для заданного хэша [5].

8.3 Подтверждение подлинности

Распространенной задачей является подтверждение *подлинности данных (authenticity).* Подлинность заключается в целостности данных и гарантированном авторстве. Например, желательно иметь уверенность, что страница сайта интернет-магазина сформирована именно сервером компании, и цены не искажены при передаче по сети.

8.3.1 Электронная цифровая подпись

Электронная цифровая подпись (ЭЦП, digital signature) — это способ подтверждения подлинности, основанный на шифровании. В настоящее время применяется асимметричная криптографическая схема, в которой шифрование производится закрытым ключом, а расшифровка — открытым. Процедура нанесения цифровой подписи и её проверки приведены на рис. 15 ([4, раздел 4], [6]).

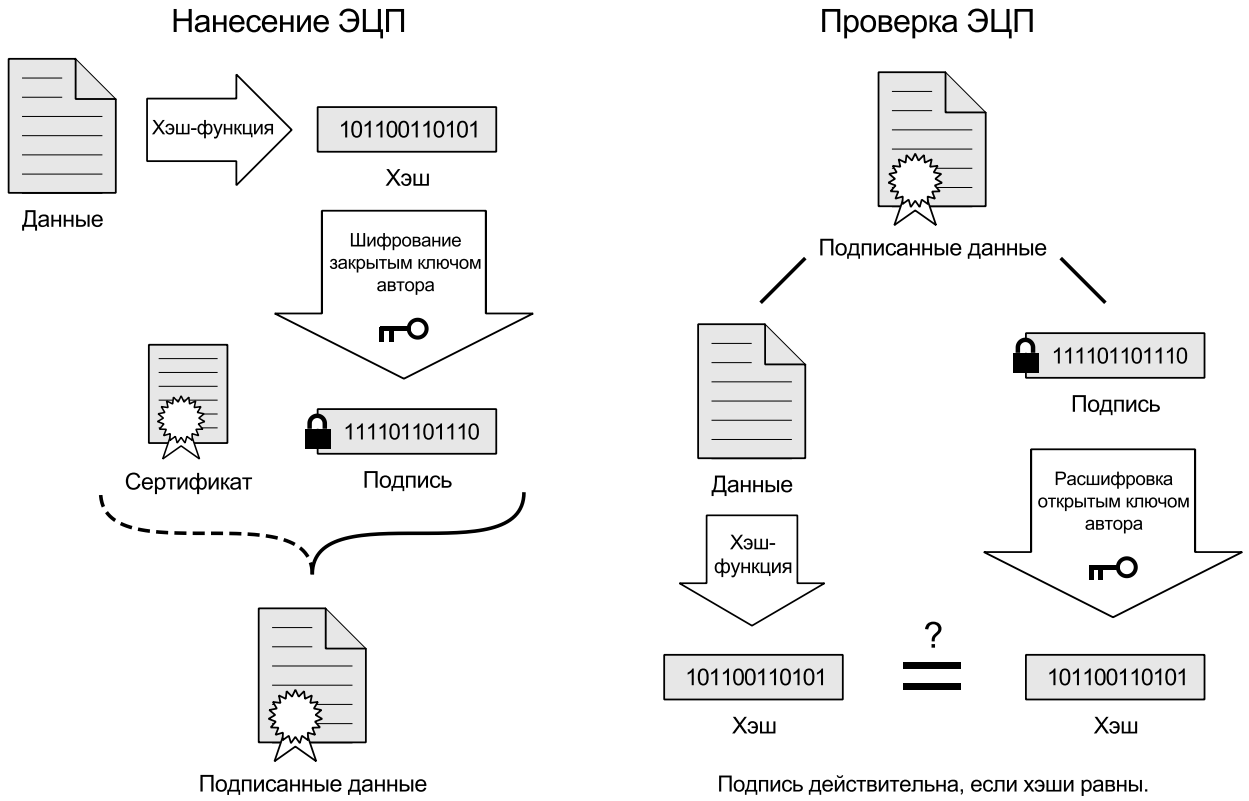


Рисунок 15 — Процедура нанесения и проверки цифровой подписи

Хэш-функция обязательно должна быть криптографической, иначе злоумышленник сможет подобрать такое искажение данных, что хэш не изменится.

При использовании ЭЦП закрытый ключ известен только подписывающему. Более того, критически важно сохранять закрытый ключ секретным. В случае, если он будет раскрыт (скомпрометирован), окажется возможным подделать все когда-либо подписанные данные и подписать любые другие, а законный владелец закрытого ключа не сможет отрицать свое авторство. Компрометация закрытого ключа для ЭЦП называется *полным взломом (total break).*

8.3.2 Электронные сертификаты

Проверка ЭЦП позволяет при помощи открытого ключа гарантировать, что подпись нанесена владельцем закрытого ключа той же пары. Однако необходима уверенность в подлинности самого открытого ключа. Это достигается комбинацией организационных и технических методов с использованием электронных сертификатов.

Электронный сертификат (digital certificate) связывает личность владельца — человека или организации — с открытым ключом пары при помощи цифровой подписи. Электронный сертификат представляет собой документ с ЭЦП специального формата (обычно X.509 [7]), содержащий сведения о владельце, период действия и открытый ключ. Подлинность сертификата заверяется ЭЦП, то есть сертификат — это подписанные данные.

Каким закрытым ключом подписывается сертификат?

Можно использовать закрытый ключ *той же* ключевой пары, что и открытый ключ в составе сертификата — такой сертификат называется *самоподписанным (self-signed).* Однако это позволяет удостовериться лишь в том, что сертификат подписан владельцем ключевой пары, и открытый ключ действителен. Невозможно проверить, что в владелец ключевой пары и владелец сертификата — одно лицо. В некоторых случаях гарантировать это удается организационно: некое лицо объявляется владельцем открытого ключа самоподписанного сертификата, и нет сомнений, что закрытый ключ надежно защищен. Такое лицо называется *удостоверяющим центром,* или *центром сертификации (certificate authority, CA).*

Сертификат может быть подписан и закрытым ключом из *другой* ключевой пары, нежели открытый ключ в составе сертификата. Её владелец называется *issuer* — обычно это центр сертификации (ЦС). Открытый ключ этой пары необходим для проверки подписи, поэтому ссылка на сертификат владельца этого ключа добавляется в состав подписанного сертификата (в виде URL, названия и т. п). В общем случае вышестоящий сертификат может

быть подписан аналогичным образом, то есть образуется цепочка сертификатов (обычно короткая). Выполняется проверка одного сертификата за другим, пока не будет найден конец цепочки — самоподписанный сертификат (или недействительная подпись).

Важно понимать, что даже если подтверждена подлинность сертификата, вопрос доверия его владельцу остается за проверяющим. Например, директор компании может заверить сертификат помощника, но очевидно, что подпись помощника не становится при этом равносильна подписи директора.

8.3.3 Инфраструктуры открытых ключей

Причины доверия и недоверия удостоверяющему центру могут быть различными. В зависимости от того, какие стороны считаются УЦ, выделяют *инфраструктуры открытых ключей (public key infrastructure, PKI):* централизованные (иерархические) и децентрализованные (сетевые).

8.3.3.1 Иерархические инфраструктуры открытых ключей

В иерархических инфраструктурах имеется несколько главных удостоверяющих центров, владеющих самоподписанными *корневыми сертификатами (root certificate)* и публикующие их открытые ключи. При проверке подлинности отдельно взятого сертификата выясняется, каким сертификатом он подписан, затем — каким сертификатом подписан последний и т. д. вверх по иерархии до тех пор, пока очередной сертификат не будет подписан одним из корневых. В этом случае можно доверять всем сертификатам в рассмотренной цепи, иначе — ни одному из них.

Центрам сертификации доверяют по организационным причинам: политическим, юридическим или коммерческим. Реальные удостоверяющие центры для глобальной инфраструктуры — правительственные учреждения и несколько крупных компаний. Список из нескольких десятков глобальных корневых сертификатов, используемых публично, распространяется в составе ОС. Возможно и построение локальных иерархий: например, владелец компании может создать сертификат, сделать его самоподписанным и использовать в пределах фирмы, где владельцу доверяют, но не глобально.

На практике сертификаты выдаются на ограниченный период времени, причем срок действия и реквизиты удостоверяющего центра включаются в выданный сертификат.

8.3.3.2 Сетевые инфраструктуры открытых ключей

Сетевые, или *распределенные, инфраструктуры открытых ключей (Web of Trust, англ. «сеть доверия»)* предлагают подход без единого центра сертификации. Изначально каждый участник владеет самоподписанным сертификатом. В ходе функционирования инфраструктуры владелец приобретает информацию о том, каким участникам и насколько (с его точки зрения) можно доверять. В частности, можно указать сертификаты, которым доверять не следует. При проверке подлинности требуется выяснить у известных уже участников их вердикт относительно проверяемого сертификата. Суждения могут оказаться различными, поэтому результат часто оказывается оценочным.

Сети доверия используются в открытых системах GNU Privacy Guard (GPG) и OpenPGP [8], для обмена информацией о репутации сайтов и в других целях. Преимуществом сетей доверия является хорошая защита от компрометации корневых сертификатов. Критическим недостатком сетей доверия является «мягкость» суждений о подлинности, непригодная в юридических и коммерческих целях.

Реальные сети доверия строятся на безвозмездной основе и удобны для больших децентрализованных систем, где ответственность всех участников сомнительна и существует риск компрометации сертификатов.

8.3.4 Отзыв сертификатов

Время от времени сертификатам оказываются нельзя доверять, навсегда (в случае компрометации) или временно. На этот случай предусмотрена процедура *отзыва сертификата (certificate revocation):* доверенной стороной публикуется и подписывается бюллетень, что сертификат не действителен [7, подраздел 3.3]. В распределенной инфраструктуре пользователи указывают, что не доверяют сертификату. Заметим, что истечение срока действия сертификата не является поводом для его отзыва, вместо этого проверяющая сторона вольна не доверять ему [8, гл. 1].

Иногда доверие утрачивает не отдельный сертификат, а весь удостоверяющий центр; в этом случае процедура называется *authority revocation.*

8.4 Протоколы безопасности сетевого взаимодействия

Шифрование данных относится к представительскому уровню модели ISO. Важным его свойством является *прозрачность (transparency)* — незаметность для протоколов прикладного уровня.

Наиболее широко в глобальной сети распространен *протокол безопасности транспортного уровня (Transport Layer Security, TLS)* [9], сменивший долгое время применявшийся *уровень безопасных сокетов (Secure Sockets Layer, SSL).* Несмотря на названия, эти протоколы действуют в основном на представительском и частично на сеансовом уровне. Отличия между ними состоят в технических деталях.

Сеанс безопасной связи по протоколам SSL или TLS состоит из двух фаз: установления безопасного соединения *(handshake, «рукопожатия»)* и зашифрованного взаимодействия. Диаграмма процедуры установления безопасного соединения представлена на рис. 16 (с. 61).

SSL и TLS используют сертификаты стандарта X.509, поэтому требуют для функционирования соответствующей налаженной инфраструктуры открытых ключей.

На практике SSL и TLS массово используются на web-сайтах, при этом на прикладном уровне используется протокол HTTP, а комбинация протоколов называется Secure HTTP (HTTPS).

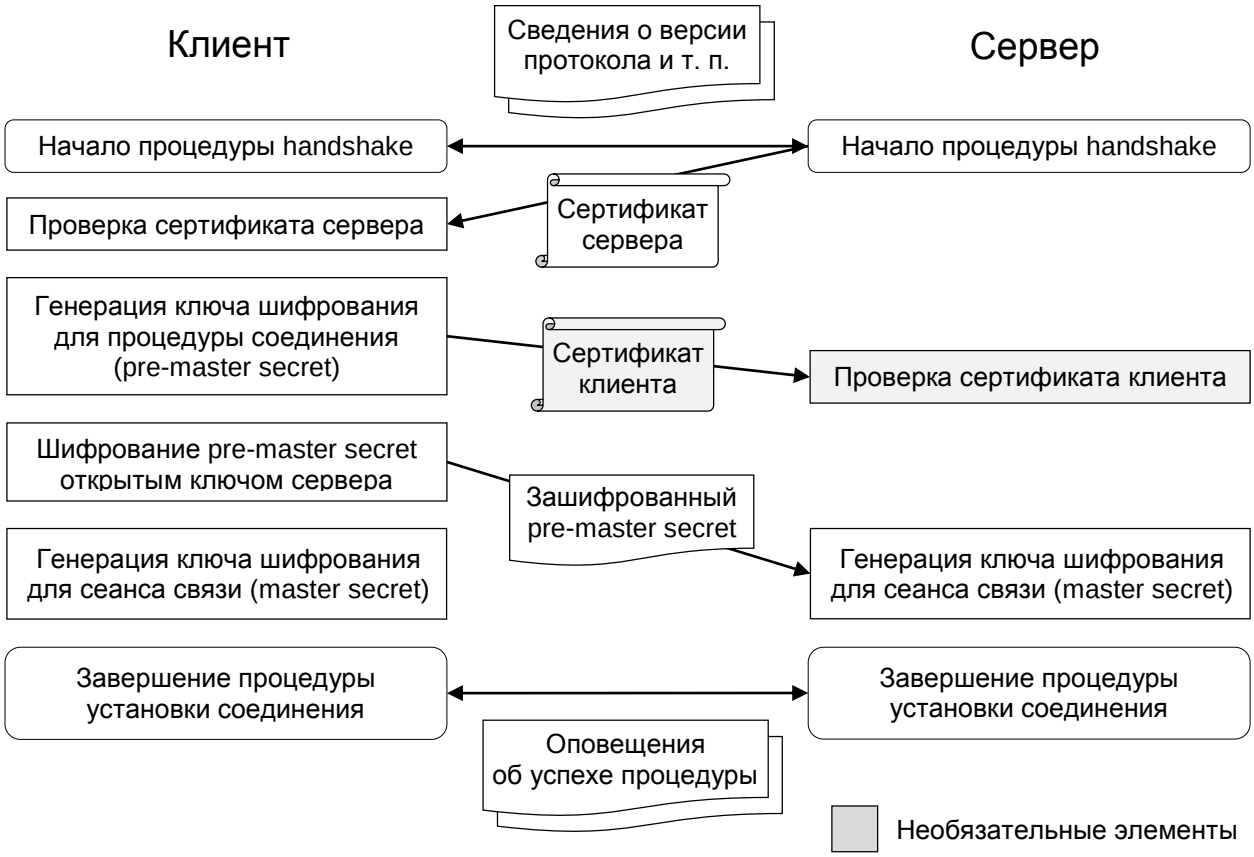


Рисунок 16 — Диаграмма процедуры установления соединения TLS

Как видно из диаграммы на рис. 16, шифрование данных во время взаимодействия будет осуществляться по симметричной схеме ключом master secret, который генерируется заново для каждого сеанса по схеме наподобие протокола Диффи-Хеллмана. Ключ для шифрования никогда не передается, а данные для его генерации (pre-master secret) сообщаются клиентом серверу зашифрованным по ассиметричной схеме. При этом для защиты от атаки «человек посередине» применяется обязательная проверка подлинности открытого ключа сервера. Проверка сертификата клиента необязательна; она может применяться, если важна подлинность клиента: при доступе к банковским счетам, при передаче налоговой отчетности и т. п.

8.5 Литература к разделу 8

1. ISO/IEC 27000:2014 «Information technology — Security techniques — Information security management systems — Overview and vocabulary», third Edition.

2. Н. Фергюсон, Б. Шнайер. *Практическая криптография.* — СПб.: Вильямс. — 2005 г. — 416 с.

3. E. Rescorla. *RFC 2631: Diffie-Hellman Key Agreement Method* [Электронный ресурс]. — Режим доступа: http://tools.ietf.org/html/rfc2631, свободный.

4. ГОСТ Р 34.10—2012 «Информационная технология. Криптографическая защита информации. Процессы формирования и проверки электронной цифровой подписи».

5. P. Hoffman, B. Schneier. *RFC 4270: Attacks on Cryptographic Hashes in Internet Protocols* [Электронный ресурс]. — Режим доступа: http://tools.ietf.org/html/rfc4270, свободный.

6. File: Digital_Signature_diagram.svg [Электронный ресурс]. — Режим доступа: http://commons.wikimedia.org/wiki/File:Digital_Signature_diagram.svg, свободный.

7. D. Cooper, S. Santesson, et al. *RFC 5280: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile* [Электронный ресурс]. — Режим доступа: http://tools.ietf.org/html/rfc5280, свободный.

8. The GNU Privacy Handbook [Электронный ресурс] // The Free Software Foundation, 1999. — Режим доступа: https://www.gnupg.org/gph/en/manual/book1.html, свободный.

9. T. Dierks, E. Rescorla. *RFC 5246: The Transport Layer Security (TLS) Protocol Version 1.2* [Электронный ресурс]. — Страница в интернете. — Режим доступа: http://tools.ietf.org/html/rfc5246, свободный.

9 СЕТЕВЫЕ ТОПОЛОГИИ

Сетевая топология – это способ соединения узлов сети, то есть конфигурация графа, вершины которого являются узлами сети, а дуги — связями между ними. От топологии зависят свойства сети: надежность, быстродействие, расширяемость, стоимость. Надежность тем выше, чем больше связей между узлами, однако такую сеть сложнее (дороже) обслуживать. Наилучшее быстродействие достигается прямой связью между узлами, однако в случае, если машины физически удалены, повышается стоимость линий связи и оборудования. Конкретные особенности топологий сильно зависят от используемых технологий, оборудования и программного обеспечения.

9.1 Классификация сетевых топологий

Наиболее общим является деление топологий на *полносвязные (fully connected),* в которых каждый узел имеет отдельную связь с каждым другим, и *неполносвязные (partially connected).* Полносвязные топологии для N узлов требуют порядка $\frac{1}{2} \cdot N^2$ связей и потому непрактичны, кроме случая двух узлов. Неполносвязные топологии получаются удалением из полносвязной части линий (то есть прокладываются только оставшиеся).

В сетях с *кольцевой конфигурацией (ring)* узлы соединяются в кольцо, и от источника пакет передается узлам поочередно, пока не достигнет адресата. Кольцевая топология обладает свойством резервирования: при разрыве связи в любой точке кольца остается маршрут пакета, связанный с обходом кольца в обратную сторону. Рисунок 17 иллюстрирует последнее свойство, как и саму топологию «кольцо».

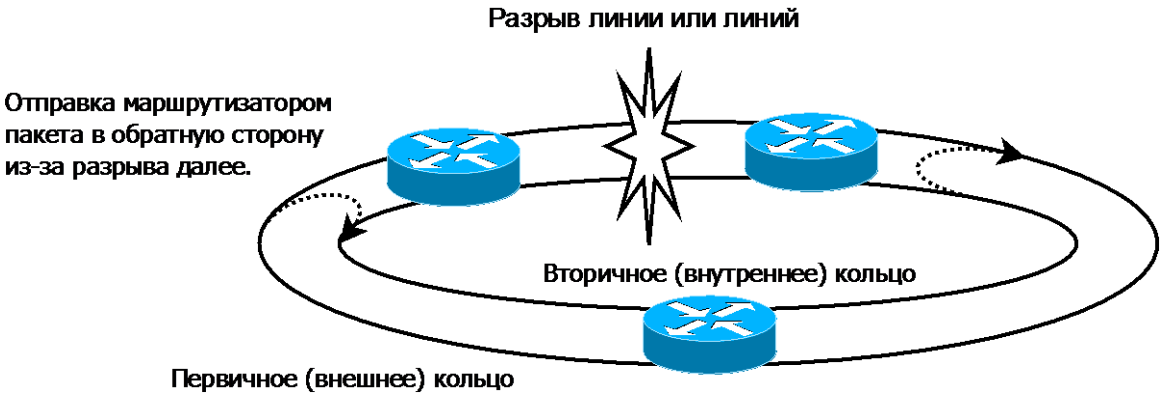


Рисунок 17 — Топология «кольцо» и резервирование маршрута при разрыве

Внутреннее и внешнее кольцо — это физически одна линия связи. Конструктивные особенности позволяют маршрутизаторам диагностировать разрыв и начать отправлять пакеты не далее по первичному кольцу, а в обратную сторону по вторичному кольцу.

Топология «звезда» (star) образуется в случае, когда каждый узел с помощью отдельной связи подключается к общему центральному узлу-маршрутизатору, называемому *концентратором*. В функции концентратора входит направление передаваемой компьютером информации остальным компьютерам сети. К недостаткам топологии типа «звезда» относится более высокая стоимость сетевого оборудования: в качестве концентратора необходимо использовать специальное устройство с большим количеством сетевых интерфейсов, числом которых ограничен размер сети.

Преодолеть ограничение на размер сети с топологией «звезда» можно, присоединяя к концентратору верхнего уровня другие концентраторы в качестве узлов. Получающаяся иерархическая топология называется *деревом (tree)* и представлена на рис. 18 наряду со «звездой». В настоящее время дерево является самым распространенным типом топологии связей, как в локальных, так и в глобальных сетях.

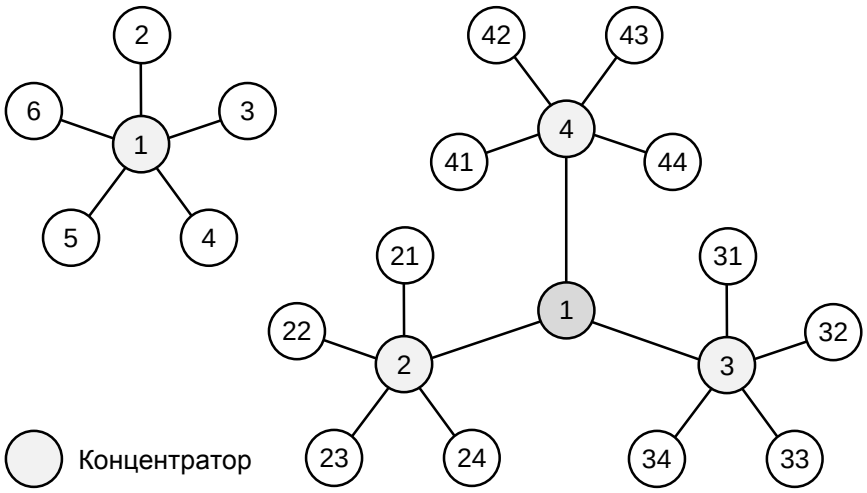


Рисунок 18 — Топология «звезда» (вверху слева) и дерево (справа)

В топологии «общая шина» (bus) узлы присоединяются к одному кабелю, который и служит единой средой передачи. Организация такой топологии очень проста и дешева, поскольку стоимость шины невелика. Однако низка надежность: повреждение шины приводит к неработоспособности всей сети, даже если между двумя узлами линия цела.

На практике встречаются случаи, когда крупная сеть создается постепенным наращиванием небольшой без предварительного проектирования. Топология на ранних этапах близка к полносвязной, однако, на конечных этапах плотность связей (отношение числа линий к числу узлов) снижается. Получается *ячеистая топология (mesh-сеть),* схема которой представлена на рис. 19.

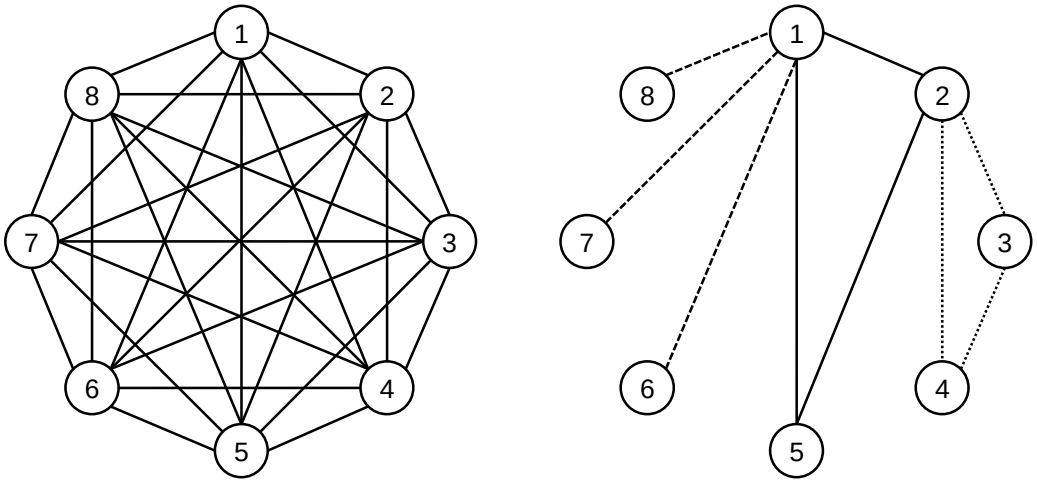


Рисунок 19 — Полносвязная (слева) и ячеистая (справа) топологии

Очевидно, любая неполносвязная топология есть частный случай ячеистой. С другой стороны, ячеистая топология часто получается объединением «колец», «звезд», деревьев и т. д.: на рис. 19 можно выделить «звезду» (узлы 6, 7, 8 и концентратор 1) и два «кольца» (узлы 1, 2, 5 и узлы 2, 3, 4), как это обозначено линиями различных типов.

9.2 Физическая и логическая топология. Оверлейные сети

Различают физическую и логическую топологию сети. *Физическая топология* — это способ соединения устройств на физическом и канальном уровне; она диктуется техническими соображениями. *Логическая топология* — это топология сетевого уровня; она выбирается, исходя из административных надобностей. Логическая топология может не совпадать с физической. Пример приведен на рис. 20 (с. 66).

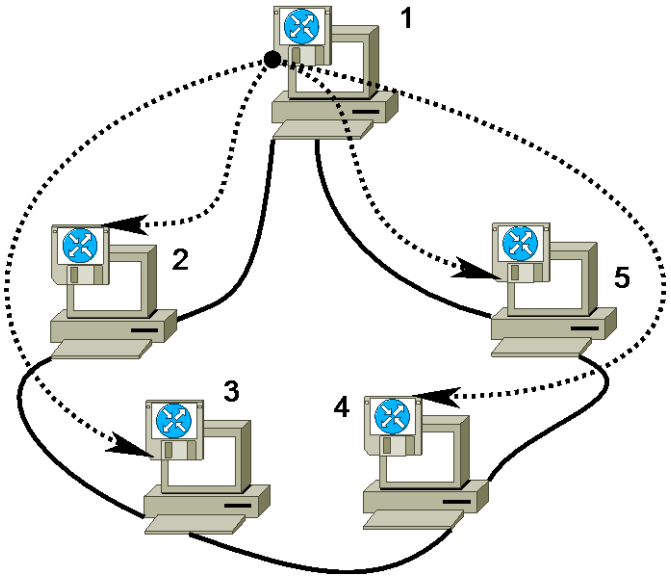


Рисунок 20 — Сеть с различающейся физической и логической топологией

На рис. 20 машины соединены физическим кабелем (сплошные линии) в кольцо, однако на сетевом уровне считается (пунктирные линии), что каждый узел находится в отдельной подсети, а узел 1 выступает в роли маршрутизатора. Такую схему целесообразно применять, если узел 1 выполняет фильтрацию, кэширование и т. п. функции. Дополнительным преимуществом будет то, что при повреждении одного из физических кабелей логическая связь-маршрут будет перестроена через уцелевшую дугу кольца, хотя топология «звезда» логической сети свойством резервирования и не обладает.

Логическая топология может строиться на базе не только физической, но и другой логической топологии. В этом случае транспортный уровень нижележащей (базовой) сети используется вышестоящей в качестве канального, но без присущих реальному канальному уровню недостатков. Такие логические сети называются *оверлейными (overlay network, англ. и жарг. «сеть поверх сети»).* К ним относятся виртуальные частные сети, децентрализованные глобальные сети (см. лекцию 10) и другие, которым требуется строго определенная топология без перенастройки сетей IP.

9.3 Виртуальные частные сети

Виртуальные частные сети (virtual private networks, VPN) предназначены для объединения удаленных друг от друга узлов в единую сеть IP с высоким уровнем безопасности [1]. Например, компания может обеспечить работу всех сотрудников в единой

сети с доступом через интернет из любой точки мира почти с такой же безопасностью, как если бы работа велась из офиса. На рис. 21 представлен пример топологии базовой логической и виртуальной частной сети для решения вышеизложенной задачи.

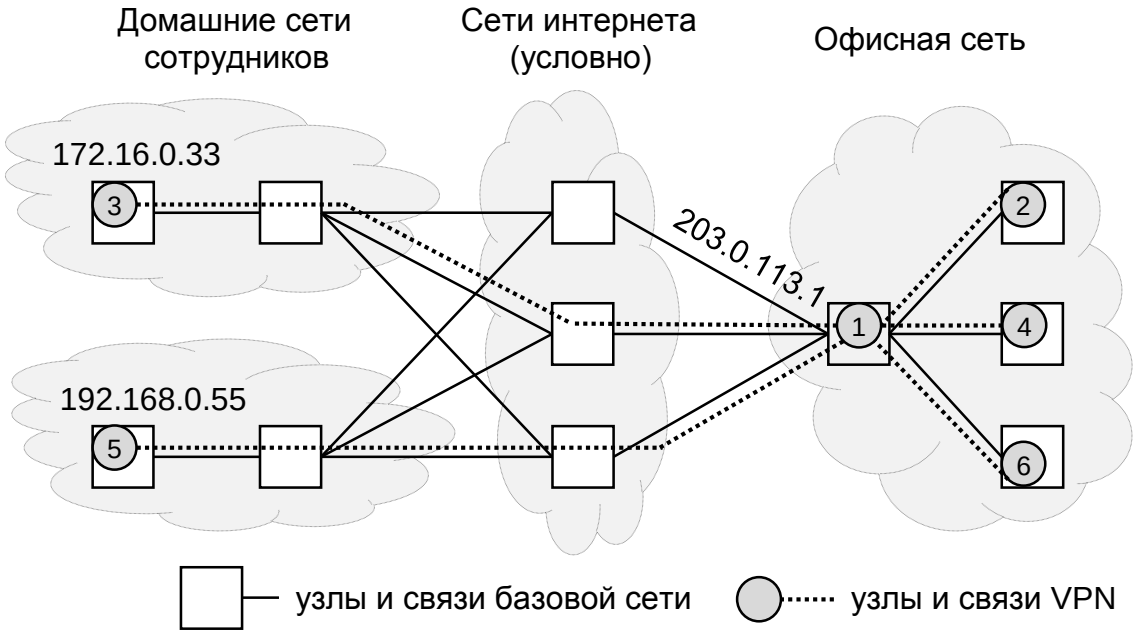


Рисунок 21 — Пример топологии базовой и виртуальной частной сети

Пусть офисная сеть — 10.0.0.0/24, маршрутизатор в офисе имеет адрес 10.0.0.1, а 3 машины в офисе — 10.0.0.2, 10.0.0.4 и 10.0.0.6 (топология «звезда»). Создание такой структуры от физического до сетевого уровня тривиально, поскольку все узлы доступны для прямого подключения и настройки. Офисная сеть связана с интернетом через маршрутизатор, один интерфейс которого (10.0.0.1) предназначен для внутренней сети, а другой (203.0.113.1) — для внешней. Специальное ПО маршрутизатора и его настройки ограничивают доступ к офисной сети, поэтому она считается защищенной.

Устройство домашних сетей сотрудников и сетей интернета неизвестно заранее и может меняться; кроме конечных, узлы этих сетей не могут быть настроены. Базовые сети вне офиса, таким образом, не могут считаться защищенными.

Технология VPN позволяет настроить сеть таким образом, что узлы в домашних сетях будут включены в оверлейную сеть 10.0.0.0/24, став её узлами 10.0.0.3 и 10.0.0.5. Эти адреса присваиваются не физическим, а виртуальным (программным) сетевым интерфейсам. При этом данные будут передаваться через интернет в зашифрованном виде прозрачно, то есть, с точки зрения узлов в 10.0.0.0/24, все они сообщаются напрямую,

как будто соединены сетью канального уровня. Узлу 10.0.0.1 достаточно обрабатывать пакеты по безопасного протокола, используемого для маршрутизации в VPN.

Принципиальная структура пакета при использовании VPN приведена на рис. 22.

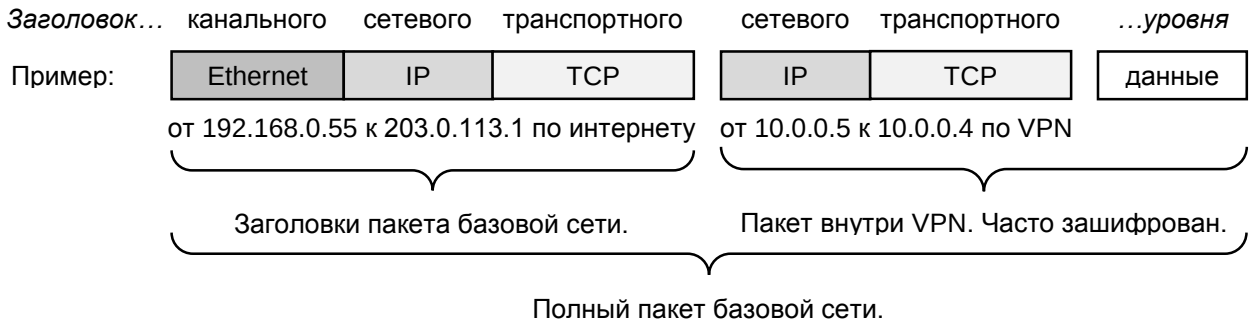


Рисунок 22 — Принципиальная структура пакета при использовании VPN

Сначала пакет маршрутизируется в базовой сети по заголовкам её пакета (например, на рис. 22 — от 192.168.0.55 до 203.0.113.1). Будучи доставлен на маршрутизатор, связывающий VPN с базовой сетью, пакет освобождается от заголовков базовой сети, и производится его маршрутизация внутри VPN (от 10.0.0.5 до 10.0.0.4 на рис. 22). При отправке пакета из VPN происходит обратный процесс: пакет VPN направляется в туннель, где дополняется заголовком базовой сети (говорят: заключается, или инкапсулируется, в пакет базовой сети).

Инкапсуляция пакетов VPN выполняется по различным протоколам, то есть пакеты VPN в составе пакета базовой сети — не обязательно пакеты IP, даже если такова базовая сеть. Обычно пакеты VPN защищены шифрованием: SSL/TLS, IPsec и т. п.

Примечание. Не следует путать VPN и частные сети IPv4 (192.168.0.0/16 и другие): не всякая частная сеть является VPN. Виртуальная частная сеть — это комплекс оборудования, настроек и процедур, а частная сеть IP — диапазон адресов.

9.4 Преобразование сетевых адресов и портов

Распространена ситуация, когда ряд узлов (организация, дом, квартира) объединены в частную сеть и связаны с внешней сетью через маршрутизатор. Адреса частной сети не маршрутизируются за её пределами, таким образом, единственный пригодный для использования в интернете (жарг. «реальный», «белый») адрес IP назначен одному

из сетевых интерфейсов маршрутизатора. Необходимо всем узлам частной сети обеспечить взаимодействие с узлами во внешней сети.

9.4.1 Преобразование сетевых адресов

Преобразование сетевых адресов (network address translation, NAT) предназначено для случаев, когда соединение инициируется узлом частной сети или им же отправляется сообщение (datagram) без установки соединения. При получении маршрутизатором пакета, направленного из частной сети во внешнюю, адрес отправителя в заголовке пакета заменяется на адрес интерфейса маршрутизатора во внешней сети. Таким образом, удаленный узел получит сообщение от маршрутизатора и сможет отправить ответ ему же. Описанный подход не единственный, но самый популярный, и называется *маскарадом (masquerading):* все узлы частной сети представляются во внешней как один. Пример NAT приведен на рис. 23; далее адреса из иллюстрации используются для примера.

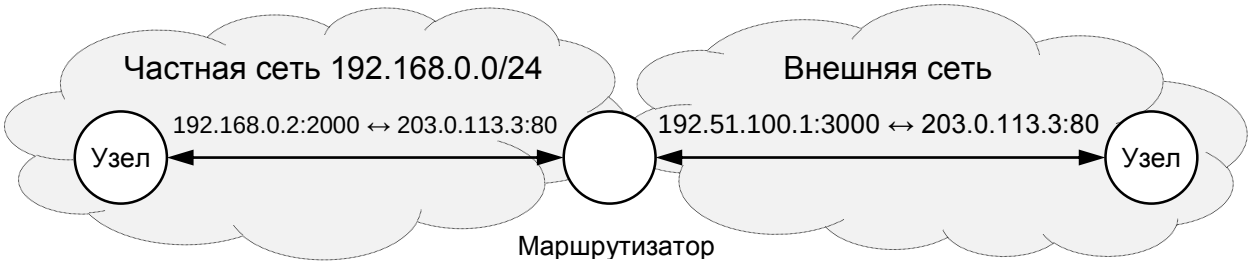


Рисунок 23 — Преобразование сетевых адресов и портов

При получении пакета из внешней сети маршрутизатору требуется определить, что этот пакет предназначен одному из узлов частной сети и переслать его. Узлы во внешней сети никак такие пакеты не отмечают, поскольку не осведомлены о NAT.

В случае, когда используется сеансовый уровень, при установлении узлом частной сети соединения с узлом внешней маршрутизатор может сохранить следующие данные:

1) адрес в частной сети, с которого инициируется соединение (192.168.0.2);
2) порт на узле частной сети, с которого инициируется соединение[5] (2000);
3) порт, использованный для пересылки пакета во внешнюю сеть (3000).

Из описанных троек *(исходный адрес, исходный порт, преобразованный порт)* на маршрутизаторе формируется таблица. При получении пакета-ответа на порт

[5] В прикладных программах разработчик обычно не указывает этот порт явно, и он выбирается ОС.

интерфейса во внешней сети можно проверить, не использовался ли этот порт (3000) при установлении соединения. В таком случае следует заменить порт назначения в заголовке пакета на исходный (из таблицы, 2000) и переслать пакет по исходному адресу в частной сети (192.168.0.2). Количество одновременных соединений между *всей* частной сетью и внешней ограничено числом портов (менее 65536). Запись хранится в таблице, как правило, не менее 4-х минут [2, подраздел 2.2].

В случае использования протокола без установления соединения (UDP и других) может быть использован аналогичный подход. Однако тогда потребовалось бы хранение записи в таблице на каждый исходящий из частной сети пакет. Поэтому маршрутизатор единственный раз выделяет порт под все сообщения одного узла частной сети.

9.4.2 Перенаправление портов

Перенаправление портов (англ. «port forwarding», жарг. «проброс портов») предназначено для случаев, когда необходимо предоставить узлам внешней сети возможность направлять сообщения и устанавливать соединения с узлами частной сети. Требуется пакеты, направленные маршрутизатору из внешней сети на т. н. исходный порт перенаправить на определенный (целевой) порт заданного узла частной сети. Правило относится как к отдельным сообщениям, так и к пакетам, инициирующим соединение. Набор троек (исходный порт, целевой узел, целевой порт) хранится на маршрутизаторе в виде таблицы, которую заполняет и изменяет только администратор сети.

Пусть, например, для рис. 23 настроено перенаправление исходного порта 4000 на целевой узел и порт 192.168.0.2:21. При попытке соединения с 198.51.100.1:4000 оно будет перенаправлено к 192.168.0.2:21. Таким образом может быть организован доступ из интернета к файловому серверу на машине в частной сети (порт 21 — FTP).

9.4.3 Особенности программирования

Преобразование сетевых адресов и портов создает особые условия, которые необходимо учитывать при программировании. Многие протоколы имеют специальное устройство или расширения для работы в условиях NAT.

Преобразованный адрес, указываемый в заголовке пакета сетевого уровня, не совпадает с адресом узла в частной сети (жарг. «скрытого NAT»). С другой стороны, действительный адрес в частной сети бесполезен вне её. Пусть, например, узел определяет

свой адрес и отправляет его узлу во внешней сети. Полученный адрес будет из частной сети, поэтому удаленному узлу не удастся впоследствии установить подключение к нему.

Аналогичная проблема возникает и с номером порта. Решения *NAT Traversal (англ. «по преодолению NAT»)* в совокупности основаны на том, что узел из частной сети по специальному протоколу сообщает маршрутизатору о необходимости перенаправления порта. В частности, так проблема решается при передаче файлов по FTP [2, подраздел 4.4].

Один и тот же преобразованный адрес может соответствовать нескольким узлам в частной сети (которые могут быть крупными). Поэтому из одинаковости адресов не следует, что пакеты отправлены одним и тем же узлом. На практике часто ограничивают доступ для узлов с определенным адресом: для защиты серверов от перегрузок, блокировки на интернет-форумах и т. п., — что может привести к ограничению для *всех* узлов частной сети, адреса которых преобразуются тем же маршрутизатором.

9.5 Литература к разделу 9

1. Virtual Private Networking: An Overview / Microsoft Corp [Электронный ресурс]. — Страница в интернете. — Режим доступа: http://technet.microsoft.com/en-us/library/bb742566.aspx, свободный.

2. P. Srisuresh, K. Egevang. *RFC 3022: Traditional IP Network Address Translator (Traditional NAT)* [Электронный ресурс]. — Режим доступа: http://tools.ietf.org/html/rfc3022, свободный.

10 ДЕЦЕНТРАЛИЗОВАННОЕ СЕТЕВОЕ ВЗАИМОДЕЙСТВИЕ

10.1 Децентрализованные системы и сети

Децентрализованной (decentralized) называют систему, в которой общая цель достигается при решении элементами нижнего уровня частных задач без координации со стороны элементов верхнего уровня. В централизованных сетях некоторые узлы (серверы), в основном, поставляют ресурсы — данные или работу, а другие узлы (клиенты) их потребляют. Децентрализованные сети (ДС), в которых все узлы выступают в равной мере потребителям и поставщиками ресурсов, называются *одноранговыми (peer-to-peer, P2P).* Часто децентрализованные системы называют просто распределенными (distributed).

Децентрализованным системам и сетям присущ ряд особенностей:

— *Субсидиарность (subsidiarity)* — принцип, согласно которому задача должна решаться на наиболее низком уровне, где это может быть сделано эффективно. Подход относится к системе в целом: вместо отправки всех задач на сервер и получения результата для решения каждой задачи выбирается узел, способный их решить (в принципе или наиболее эффективно).

— *Разнообразие узлов (diversity)* по их функциональности и ресурсам. Множество клиентов с однотипными запросами к небольшому количеству мощных серверов в централизованных системах заменяется множеством клиентов с различными задачами, для решения которых требуются разнообразные возможности других узлов. Например, в файлообменных сетях у каждого из 10-и узлов может быть по одной (уникальной) части файла, и это означает, что 10 различных запросов (по всей сети) может быть удовлетворено без создания чрезмерной нагрузки на единственный узел.

Децентрализованные сети обладают следующими преимуществами:

— *Отказоустойчивость (fault tolerance).* Поскольку ни один узел не является «особенным» в том или ином смысле, при нарушениях в работе одного узла его задачи почти всегда может решить какой-либо другой. Кроме того, отсутствует опасность компрометации центральных узлов: технического сбоя, несанкционированного доступа, юридических претензий.

— *Масштабируемость (scalability).* При увеличении размера системы не требуется ни изменения её структуры, ни усложнения центральных управляющих узлов (которых нет). Раскрытие преимущества требует грамотного проектирования сети.

— *Конфиденциальность (privacy).* В отличие от централизованных сетей, в ДС обычно нет необходимости доверия центральному узлу и передачи ему большого объема пользовательских данных. Отметим, что децентрализация сама по себе не защищает данные от перехвата на обычных узлах, а лишь устраняет вектор атаки на центральные узлы.

Невозможность выхода из строя отсутствующих центральных узлов вместе с конфиденциальностью называют иногда *автономностью* децентрализованных сетей.

Децентрализованным сетям присущи и недостатки [1, раздел 2]:

— *Сложность управления.* Узлы ДС действуют независимо и единообразно, поэтому нет возможности радикально влиять на сеть ресурсами отдельного узла. Борьба с этим недостатком обычно снижает конфиденциальность.

— *Издержки взаимодействия.* Любая сеть требует обслуживания (по меньшей мере, маршрутизации), и в отсутствии центральных узлов эта задача распределяется между всеми. Ни один узел не владеет сведениями о сети в целом, поэтому распределение задач выполняется в условиях недостатка информации. С другой стороны, задачи назначаются узлам, способным решать их эффективнее.

10.2 Задача маршрутизации в децентрализованных сетях

В централизованных сетях, как правило, интерес представляют специфические ресурсы, предоставляемые известным участником. Например, узел может быть сервером СУБД, FTP и HTTP, и его адрес известен заранее. По этой причине удобна традиционная схема маршрутизации, в которой путь доставки пакета прокладывается до узла с заданным адресом, а конкретный ресурс-приложение определяется портом. Децентрализованные сети отличаются тем, что состав участников велик и непостоянен, а набор предоставляемых каждым узлом ресурсов широк, но не слишком специфичен. Например, у 10-и участников файлообменной сети может быть по 91 части файла из 100, причем у одного узла — части с 1-й по 91-ю, у другого — со 2-й по 92-ю и т. д. По этой причине в децентрализованных сетях требуется прокладывать маршрут не до узла с заданным адресом, а до узла с необходимым ресурсом.

Будучи оверлейными, децентрализованные сети используют базовые сети как канальный уровень, а у узлов на сетевом уровне оказывается по множеству адресов-ресурсов. Тогда алгоритм поиска участника может быть организован подобно работе ARP: следует отправить всем узлам запрос, на который должны ответить обладатели нужного ресурса. Проблема заключается в том, что число участников и масштаб сети очень велики. Продолжая аналогию, следует выделить подсети, чтобы вести поиск иерархически. Однако, при этом каждый участник оказывается одновременно в большом количестве подсетей (оценочно, по числу ресурсов), то есть становится маршрутизатором ДС.

Задачи маршрутизации в децентрализованных сетях:

1) компактное представление каждого ресурса участника для быстрого поиска;
2) определение адреса в базовой сети участника с определенным ресурсом;
3) разбиение адресного пространства на подсети так, чтобы обеспечить наибольшую скорость поиска (наименьшее число ретрансляций запроса).

10.3 Распределенные хэш-таблицы (DHT)

Распределенные хэш-таблицы (distributed hash table, DHT) являются самым популярным подходом к исполнению маршрутизации в децентрализованных сетях. Хэш-таблица — это структура данных, хранящая пары *(имя, значение)* и позволяющая получить доступ к значению по имени. Значения размещаются в ячейках в зависимости от хэша имени (ключа), что позволяет находить нужные ячейки очень быстро.

Примечание. В литературе по структурам данных ключом чаще называется *имя* в паре *(имя, значение)*. Поскольку в DHT *имя* из пары практически не используется, ключом называют хэш *имени.*

Аналогом ячейки простой хэш-таблицы в DHT служит узел: каждый участник ответственен за хранение значений с ключами из некоторого подмножества. Конкретные способы распределения называются схемами *разбиения пространства ключей (keyspace partitioning).* Распространено резервирование, когда одна и та же пара *(ключ, значение)* хранится несколькими узлами.

В простой хэш-таблице хэширование применяется для быстрого поиска ячеек. DHT используют хэши для простой и уникальной адресации данных. Производительность операций над DHT зависит от схемы разбиения пространства ключей [2].

10.3.1Сохранение и поиск в распределенных хэш-таблицах

Рисунок 24 иллюстрирует принципиальный процесс помещения данных в DHT.

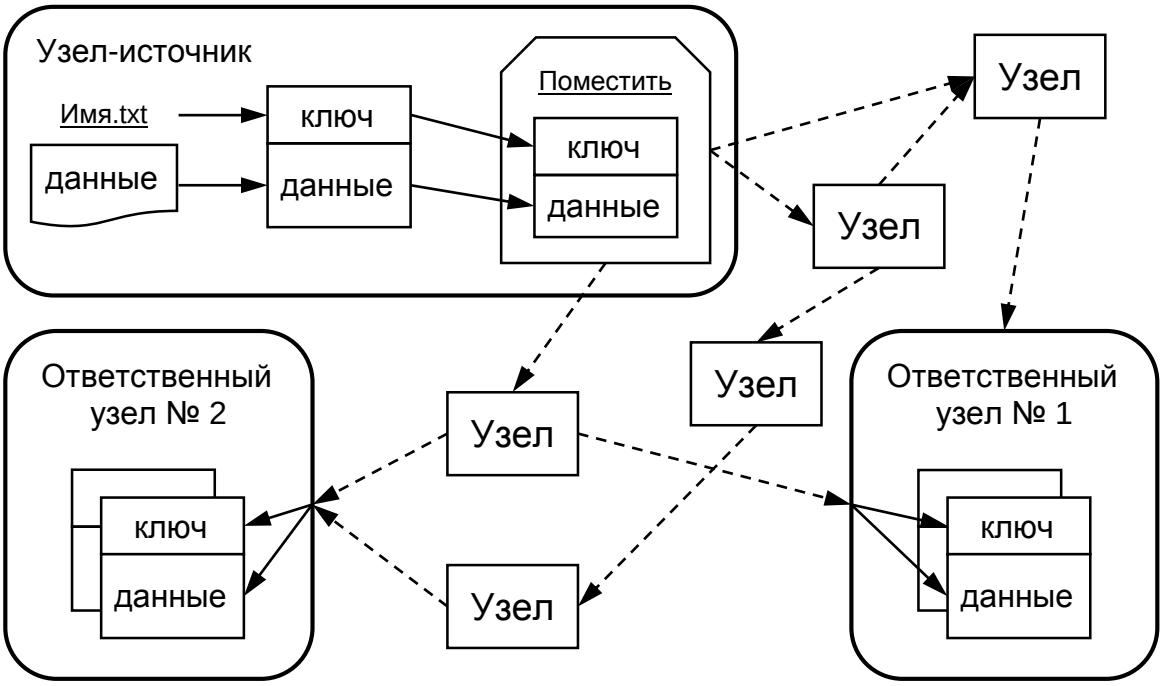


Рисунок 24 —Процесс помещения данных в распределенную хэш-таблицу

Пусть необходимо поместить в DHT содержимое файла `Имя.txt`. Строка «имя.txt» преобразуется в ключ (хэш), а данные — это содержимое файла. Формируется сообщение-команда поместить в DHT данные под вычисленным ключом. Сообщение рассылается участникам, известным узлу-источнику, которые пересылают команду известным им узлам. Так продолжается до тех пор, пока сообщение не будет доставлено одному из узлов, ответственных за хранение данных с соответствующим ключом. Информация сохраняется в простой хэш-таблице небольшого размера, расположенной на ответственном узле.

Поиск выполняется аналогично: сообщение-запрос будет содержать ключ искомых данных и адрес узла, которому данные требуются, в базовой сети. При получении запроса узлом, хранящем данные под соответствующим ключом, будет отправлен ответ.

Описанная схема очевидна и проста, но нерациональна и не всегда практична. Совокупный трафик сети в несколько раз превышает полезный, поскольку пакеты копируются на всех узлах, через которые проходят, а ценность представляют лишь те пакеты, которые доставлены ответственным узлам (лавинный алгоритм маршрутизации). Проблема усугубляется при большом объеме данных в одном сообщении; если же делить

данные на фрагменты, маршрутизацию каждого фрагмента приходится выполнять отдельно, что замедляет работу сети. Кроме того, приходится хранить логически единые данные на нескольких узлах, что часто не соответствует задачам участников: например, в файлообменной сети файлы нужны целиком.

На практике для DHT применяются двухэтапные схемы: сначала разыскиваются адреса участников, ответственных за хранение определенного ключа, затем высылаются сообщения с данными только им. На первом этапе размер сообщений существенно меньше, чем в наивной схеме, на втором этапе объемные данные не копируются многократно, а передаются адресно. Иногда применяются две параллельно работающие DHT: одна для адресов узлов (в базовой или оверлейной сети), другая для фрагментов данных.

10.3.2Топологии и схемы разбиения пространства ключей в DHT

На маршрутизацию в DHT влияет способ связи узлов в базовой сети, называемый топологией децентрализованной сети. Любая разумная топология должна обеспечивать фундаментальное свойство: сообщение передается либо ответственному узлу, либо узлу, расположенному ближе к ответственному (по количеству передач). Удобство применения той или иной топологии для N узлов определяется необходимым количеством связей (соседей) у каждого узла и средней длиной маршрута. Чем ближе топология к полносвязной, тем короче маршруты, но тем больше связей необходимо поддерживать каждому узлу. В случае, если количество связей у каждого из N узлов постоянно (например, m = 1), топология кольцевая, и средняя длина маршрута, очевидно, $\mathcal{O}(N)$. При $\mathcal{O}(N)$ связях у каждого узла (полносвязная топология) длина любого маршрута составляет $\mathcal{O}(1)$. Поскольку в децентрализованных сетях N очень велико, оба случая неприемлемы.

Наибольшее распространение получила *хордовая топология (chord)* [3], пример которой приведен на рис. 25 (с. 77). Соединение всех узлов в логическое кольцо обязательно (жирные линии). Некоторые дополнительные связи по хордам кольца могут отсутствовать (тонкие сплошные и пунктирные линии на рис. 25). Поддерживают количество связей у каждого узла порядка $m = \mathcal{O}(\log N)$, тогда средняя длина маршрута составляет $l = \mathcal{O}(\log N)$. Отношение l/m постоянно и не зависит от N. На рис. 25 узлов $N = 8$, и если у каждого узла имеются все обозначенные связи (по $m = \log_2 8 = 3$ штуки), длина маршрута не превышает двух передач ($\lceil \log_m 8 \rceil = 2$).

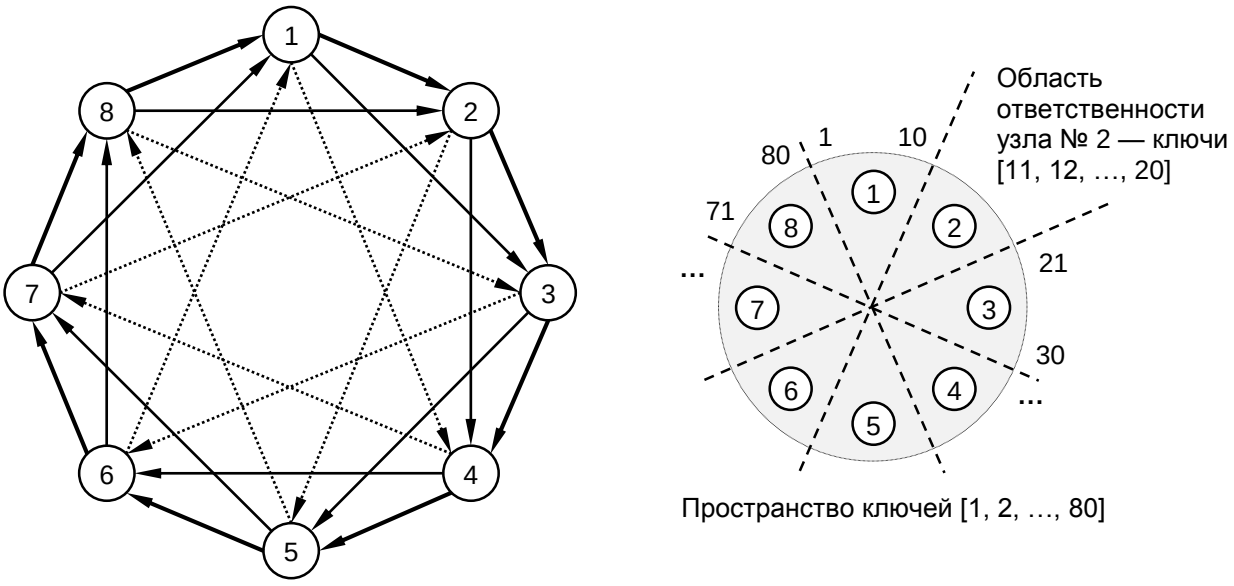


Рисунок 25 — Сеть с хордовой топологией и разбиение пространства ключей

При разбиении пространства ключей считается, что всевозможные ключи расположены на окружности по порядку, и каждый узел отвечает за ключи в некотором секторе. На практике с целью резервирования сектора перекрываются, и за один диапазон назначаются ответственными несколько узлов. Например, на рис. 25 узел 2 может хранить данные для ключей от 11 до 25, узел 3 — от 21 до 35 и т. д.

Существует модификация хордовой топологии — Koorde [4], в которой за счет усложненного разбиения пространства ключей достигается большее (следовательно, лучшее) соотношение количества связей у узла к средней длине маршрута — $\mathcal{O}(\log N)$ к $\mathcal{O}\left({\log N}/{\log\log N}\right)$.

10.4 Обнаружение участников

Важная задача узлов децентрализованной сети — *обнаружение других участников в базовой сети (peer discovery).* Существует два основных способа: обмен информацией и массовые запросы.

Обмен информацией заключается в том, что узлы децентрализованной сети передают друг другу сведения об известных им участниках, чтобы при отключении одного из узлов оставшиеся могли обнаружить его соседей. Обновление информации происходит либо естественным образом при запросах к DHT, либо регулярно принудительно [5].

Обнаружение участников массовыми запросами осуществляется отправкой пакетов на адреса широковещательной (broadcast) или многоадресной (multicast) рассылки на некоторый порт. Узел, входящий в децентрализованную сеть, может ответить на такой запрос отправителю, обнаружив себя. Широковещательная рассылка проще в реализации, но невозможна в сетях IPv6. Многоадресная рассылка предпочтительна еще и потому, что не создается излишней нагрузки на сеть. На практике многоадресную рассылку использует популярный протокол BitTorrent (расширение Local Peer Discovery [6]).

Первый раз задача обнаружения участников возникает в начале работы любого узла децентрализованной сети, подобно *заводу* автомобиля *(bootstrapping).* Если ни в локальной сети, ни в пределах досягаемости многоадресной рассылки других узлов нет, задача чисто децентрализованным образом решена быть не может. Поэтому в реальных ДС вводятся несколько особых узлов, основная задача которых — предоставить адреса других узлов, с которыми уже осуществляется взаимодействие: трекеры в BitTorrent, supernodes в Skype, публичные узлы Tox (аналог Skype) и т. п. Адреса таких специальных узлов в базовой сети могут быть универсальны для всей ДС (как в Skype и Tox, в которых распространяются с ПО) или быть специфичными ресурса в DHT (как в BitTorrent, где включаются в torrent-файл [5] или Magnet-ссылку).

10.5 Литература к разделу 10

1. E. A. Akkoyunlu, K. Ekanadham, R. V. Huber. *Some constraints and tradeoffs in the design of network communications //* Proceedings of the fifth ACM symposium on Operating systems principles. — New York, ACM. — 1975 г. — С. 68—70.

2. Eric Rescorla. Introduction to Distributed Hash Tables // Proceedings of the Sixty-Fifth Internet Engineering Task Force [Электронный ресурс]. — Режим доступа: http://www.ietf.org/proceedings/65/slides/plenaryt-2.pdf, свободный.

3. Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, Hari Balakrishnan. *Chord: A scalable peer-to-peer lookup service for internet applications //* ACM SIGCOMM Computer Communication Review, vol. 31, № 4, pp. 149—160. — ACM, 2001.

4. Kaashoek M. Frans, David R. Karger. *Koorde: A simple degree-optimal distributed hash table //* Peer-to-peer systems II. — Springer Berlin Heidelberg, 2003. — С. 98—107.

5. Andrew Loewenstern, Arvid Norberg. *DHT Protocol* [Электронный ресурс]. — Страница в интернете. — Режим доступа: http://www.bittorrent.org/beps/bep_0005.html, свободный.

6. Robin Perkins. *Zeroconf Peer Advertising and Discovery* [Электронный ресурс]. — Страница в интернете. — Режим доступа: http://bittorrent.org/beps/bep_0026.html, свободный.