Компьютерные сети

С.Ю. Скоробогатов

Осень 2017

Список литературы по курсу

1. Слайды лекций.
   www.dropbox.com/s/7bmcieds2ynj7c2/networks.pdf
2. Дж. Куроуз, К. Росс. Компьютерные сети: нисходящий подход. – М.: Издательство «Э», 2016.
   www.dropbox.com/s/dyb5iqdkarqb0el/Kurouz.pdf
3. В. Олифер, Н. Олифер. Компьютерные сети. Принципы, технологии, протоколы: учебник для вузов. 5-е изд. – СПб.: Питер, 2016.
   www.dropbox.com/s/0ras57ozvx33ej5/Olifer.pdf
4. Э. Таненбаум, Д. Уэзеролл. Компьютерные сети. 5-е изд. – СПб.: Питер, 2012.
   www.dropbox.com/s/528ed67cb0q1e3b/Tanenbaum.pdf
5. Й. Снейдер. Эффективное программирование TCP/IP. – М.: ДМК Пресс, 2009.
   www.dropbox.com/s/998pbw27u2njzd9/Snader.pdf

Определение 1

Компьютерная сеть – *это совокупность конечных и транзитных узлов, соединённых линиями связи.*

Пример 1. *Конечные узлы – персональные компьютеры, смартфоны, планшеты, серверные компьютеры, устройства «интернета вещей».*

Пример 2. *Транзитные узлы – повторители, концентраторы, оптические терминалы, сетевые мосты, коммутаторы, маршрутизаторы.*

Пример 3. *Линии связи – проводные (медные провода в телефонных линиях), кабельные (витая пара, оптоволокно), наземные беспроводные (WiFi, 3G, LTE), спутниковые.*

Цель развёртывания компьютерной сети

Определение 2

Цель развёртывания компьютерной сети – *организация доступа одних конечных узлов к ресурсам других конечных узлов*, где ресурс – это:

- *данные, хранящиеся на узле;*
- *вычислительные мощности узла;*
- *периферийные устройства, подключенные к узлу.*

Пример 4. *Web-браузер на компьютере A скачивает HTML-страницу с Web-сервера B (доступ к данным).*

Пример 5. *Web-браузер на компьютере A инициирует генерацию динамической HTML-страницы на Web-сервере B (доступ к вычислительным мощностям и данным).*

Пример 6. *Текстовый редактор на компьютере A инициирует печать документа на принтере, подключённом к компьютеру B (доступ к периферийному устройству).*

Определение 3

Коммутация – *соединение двух конечных узлов, обеспечивающее передачу данных.*

Виды коммутации:

- **Коммутация каналов**.
  Между двумя конечными узлами резервируется отдельная линия связи, обеспечивающая фиксированную скорость передачи данных. Когда данные не передаются, линия простаивает.
- **Коммутация пакетов**.
  Данные разбиваются на фрагменты, называемые *пакетами*. Пакеты, относящиеся к различным парам взаимодействующих конечных узлов, передаются по общей линии связи. Скорость передачи зависит от загруженности линии.

Конечные узлы, как правило, напрямую не связаны линией связи, т.е. данные между ними передаются через транзитные узлы.

Определение 4

Маршрут – *последовательность транзитных узлов, через которые осуществляется передача данных между двумя конечными узлами.*

Между двумя конечными узлами часто можно проложить несколько маршрутов.

Определение 5

Маршрутизация – *выбор оптимального маршрута и оповещение сети об этом маршруте.*

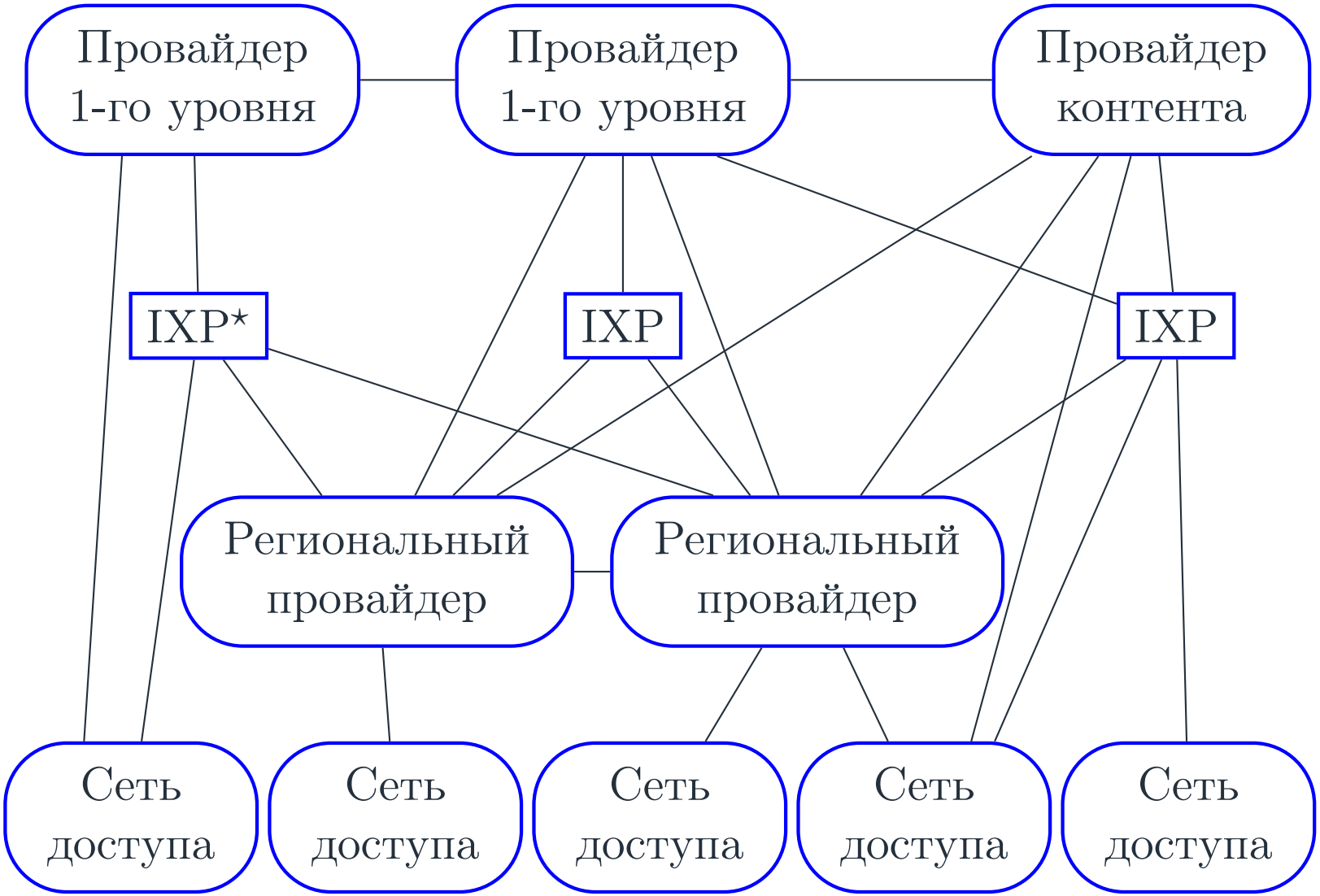
Выбор оптимального маршрута может осуществляться на основе информации о конфигурации сети и о загруженности линий связи.

Сеть Интернет

Интернет – это глобальная сеть, представляющая собой множество соединённых линиями связи компьютерных сетей, огранизованных в подобие иерархической структуры. Категории сетей, составляющих Интернет:

- **Сети доступа**.
  Сеть доступа физически соединяет конечный узел с первым маршрутизатором.
- **Региональные провайдеры**.
  Объединяет сети доступа, сосредоточенные в некоторой географической области (например, в городе).
- **Провайдеры 1-го уровня**.
  Сеть провайдера 1-го уровня выходит за рамки конкретной географической области.
- **Провайдеры контента**.
  Специализированные глобальные сети (например, сеть Google).

Схема взаимодействия компьютерных сетей, составляющих Интернет



*IXP – точка обмена трафиком.

Определение 6

Клиент – *это модуль, предназначенный для формирования и передачи сообщений-запросов к ресурсам удалённого конечного узла с последующим приёмом результатов из сети.*

Замечание. Говоря, что клиент – это модуль, мы имеем в виду, что он является программным или аппаратным компонентом конечного узла. Чаще всего, естественно, модули реализуются программно.

Пример 7. *Web-браузер – это клиент, реализованный в виде программного компонента.*

Пример 8. *Клиент доступа к среде передачи данных, осуществляющий оформление кадра данных MAC-уровня и выдачу сигналов в кабель, реализуется аппаратно в сетевой карте.*

Клиент как посредник между приложениями и ресурсами удалённых узлов

Замечание. Подразумевается, что клиент является посредником между приложениями, которые запущены на одном с ним конечном узле, и ресурсами удалённого узла.

Пример 9. *Клиент печати запущен на компьютере A. Также на этом компьютере запущены текстовый редактор, Web-браузер и графический редактор.*
Когда любому из этих трёх приложений требуется распечатать документ, они обращаются к клиенту печати, который отвечает за передачу документа на компьютер B, к которому подключён принтер.
Если бы не было клиента печати, его функциональность пришлось бы встраивать в каждое из приложений, поддерживающих печать документов.

Определение 7

Сервер – *это модуль, который постоянно ожидает прихода из сети сообщений-запросов от клиентов и, приняв запрос, пытается его выполнить и передать результат соответствующему клиенту.*

Замечание. Большинство серверных модулей реализуются программно. Аппаратная реализация характерна только для низкоуровневых серверных модулей.

Пример 10. *Сервер печати запущен на компьютере B. Он постоянно «слушает» сеть, ожидая запросы от клиентов печати, запущенных на удалённых компьютерах.*
Когда сервер печати получает запрос от компьютера A, он передаёт содержащийся в запросе документ драйверу принтера. Информация о завершении печати пересылается сервером печати на компьютер A.

Определение 8

Сетевая служба – *это пара клиент–сервер, обеспечивающая взаимодействие двух конечных узлов через сеть.*

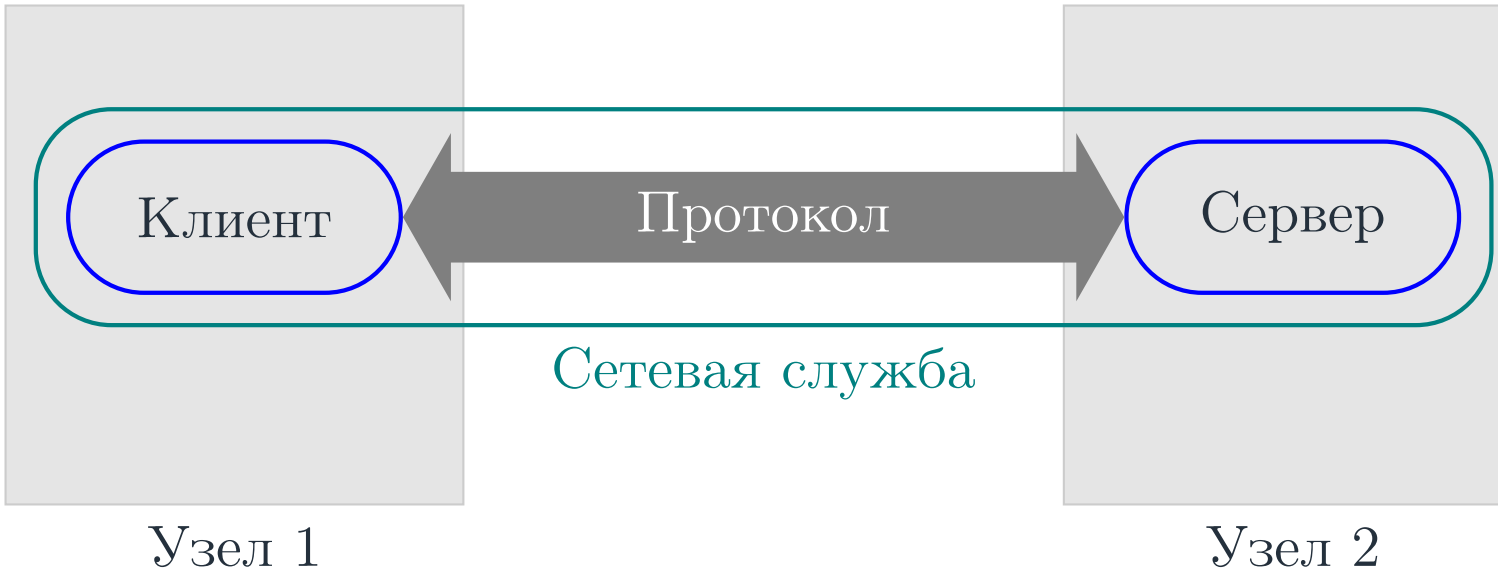
Замечание. В отличие от клиентов и серверов, являющихся реальными программными или аппаратными компонентами, сетевая служба – это абстрактное понятие, т.к.:

- на самом деле в сетевом взаимодействии могут участвовать одновременно несколько клиентов и серверов, а не «пара»;
- клиент и сервер, образующие сетевую службу, могут разрабатываться независимо и, следовательно, иметь несколько реализаций.

Пример 11. *Web-служба – это пара браузер–web-сервер. Один сервер обслуживает несколько браузеров. Существует множество реализаций браузеров и web-серверов.*

Определение 9

Протокол – *это набор правил и соглашений, определяющий форматы и порядок следования сообщений между клиентом и сервером в рамках одной сетевой службы.*



Замечание. Независимая разработка клиента и сервера, составляющих сетевую службу, возможна благодаря наличию протокола, определяющего их взаимодействие.

Сетевые протоколы: примеры

Пример 12. *Internet Protocol (IP) – протокол, реализация которого обеспечивает доставку пакетов данных между любыми узлами сети через произвольное количество транзитных узлов, но не гарантирует надёжность доставки.*

Пример 13. *Transmission Control Protocol (TCP) – протокол, реализация которого позволяет надёжно передавать поток байтов от одной программы на некотором компьютере к другой программе на другом компьютере.*

Пример 14. *File Transfer Protocol (FTP) – протокол передачи файлов с возможностью аутентификации пользователей.*

Пример 15. *Hypertext Transfer Protocol (HTTP) – протокол передачи гипертекста, используемый в web-службе.*

Пример 16. *Internet Printing Protocol (IPP) – протокол службы печати.*

Категории приложений

С точки зрения использования сети приложения делятся на три категории:

- **Локальные приложения.** Целиком выполняются на данном компьютере и используют только локальные ресурсы (например, компилятор gcc).
- **Централизованные сетевые приложения.** Целиком выполняются на данном компьютере, но обращаются в процессе работы к ресурсам других компьютеров сети (например, среда разработки IntelliJ IDEA, способная загружать из сети обновления и плагины).
- **Распределённые приложения.** Состоят из нескольких взаимодействующих через сеть частей, каждая из которых отвечает за определённую функциональность. Каждая часть распределённого приложения может выполняться на отдельном компьютере. (Пример распределённого приложения – T-BMSTU.)

Модульно-уровневая декомпозиция

При проектировании больших программно-аппаратных систем (в том числе и компьютерных сетей) используют *модульно-уровневую декомпозицию*, при которой:

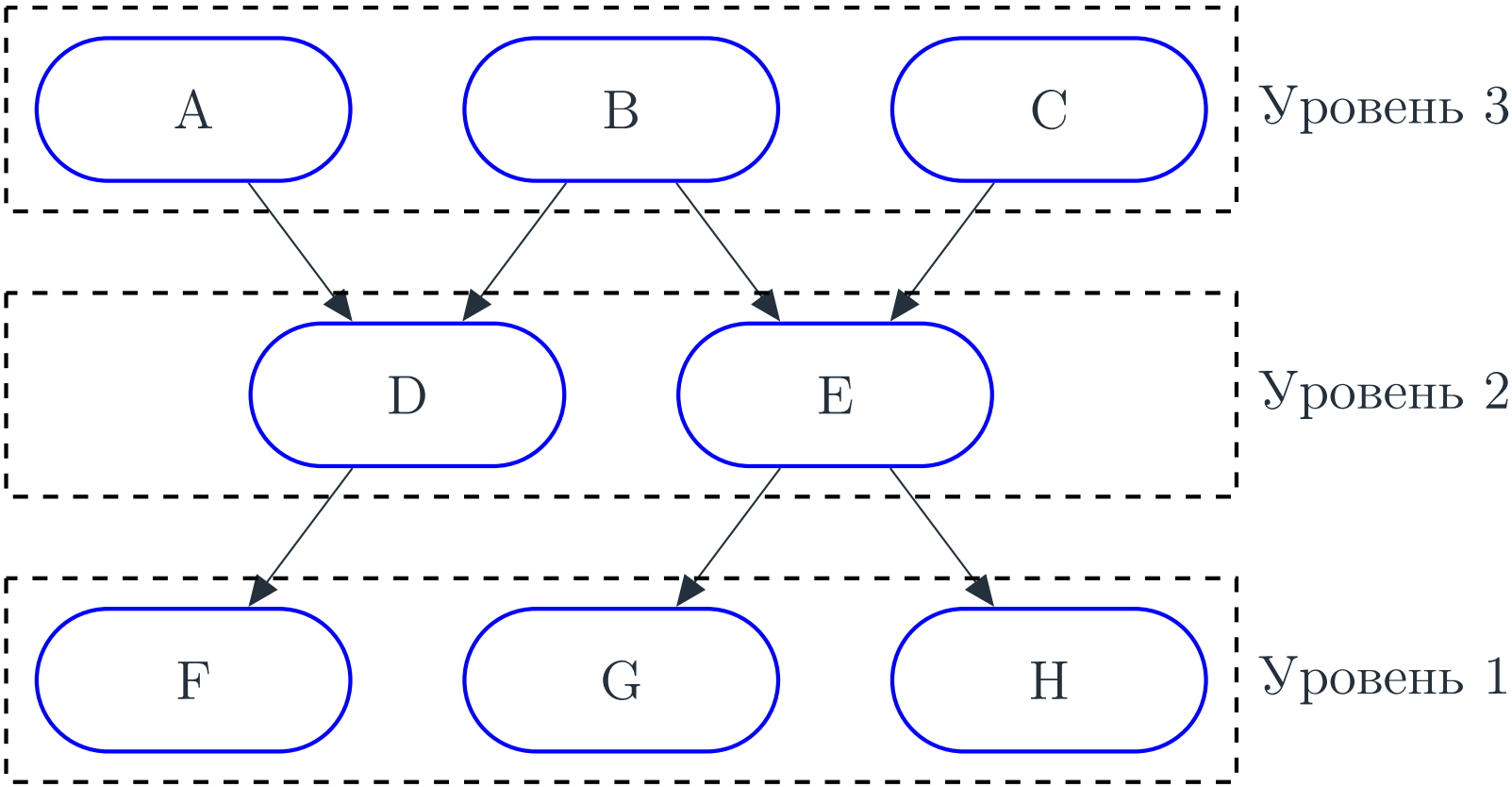
- вся система разбивается на *модули*;
- модули распределяются по иерархии *уровней*.

Замечание. Модуль – это программный или аппаратный компонент (клиент или сервер сетевой службы), предоставляющий доступ к реализуемой им функциональности через некоторый *интерфейс* (чаще всего, просто набор функций).

Замечание. Уровень – это, фактически, просто уникальное натуральное число – *номер уровня*.
Уровни образуют иерархию в том смысле, что уровень с бóльшим номером считается расположенным выше уровня с меньшим номером.
Каждому модулю ставится в соответствие номер уровня.
Несколько модулей могут принадлежать одному уровню.

Правило модульно-уровневой декомпозиции

Модульно-уровневая декомпозиция накладывает на проектирование системы следующее важное ограничение: **модулям i-го уровня разрешается обращаться только к модулям $(i-1)$-го уровня.**



Взаимодействие узлов как одновременная работа нескольких служб

Модуль сетевой службы реализует свою функциональность, делегируя часть действий модулям нижележащего уровня.

Пример 17. *Схема работы Web-браузера:*

1. *отправить Web-серверу HTTP-запрос;*
2. *принять ответ, содержащий HTML-страницу.*

Для этого Web-браузер делегирует клиенту службы TCP, расположенному уровнем ниже, следующие действия:

1. *установка соединения с Web-сервером;*
2. *передача Web-серверу потока байтов HTTP-запроса;*
3. *получение от Web-сервера потока байтов ответа.*

Клиент службы TCP, в свою очередь, для выполнения этих действий обращается к клиенту службы IP, расположенному ещё на один уровень ниже.

Тем самым, взаимодействие двух узлов означает одновременную работу нескольких служб.

Соответствие сетевых служб на взаимодействующих узлах

Клиентам сетевых служб, решающим задачу взаимодействия с удалённым узлом, должны соответствовать сервера этих же служб на удалённом узле.

Пример 18. *Схема работы Web-сервера:*

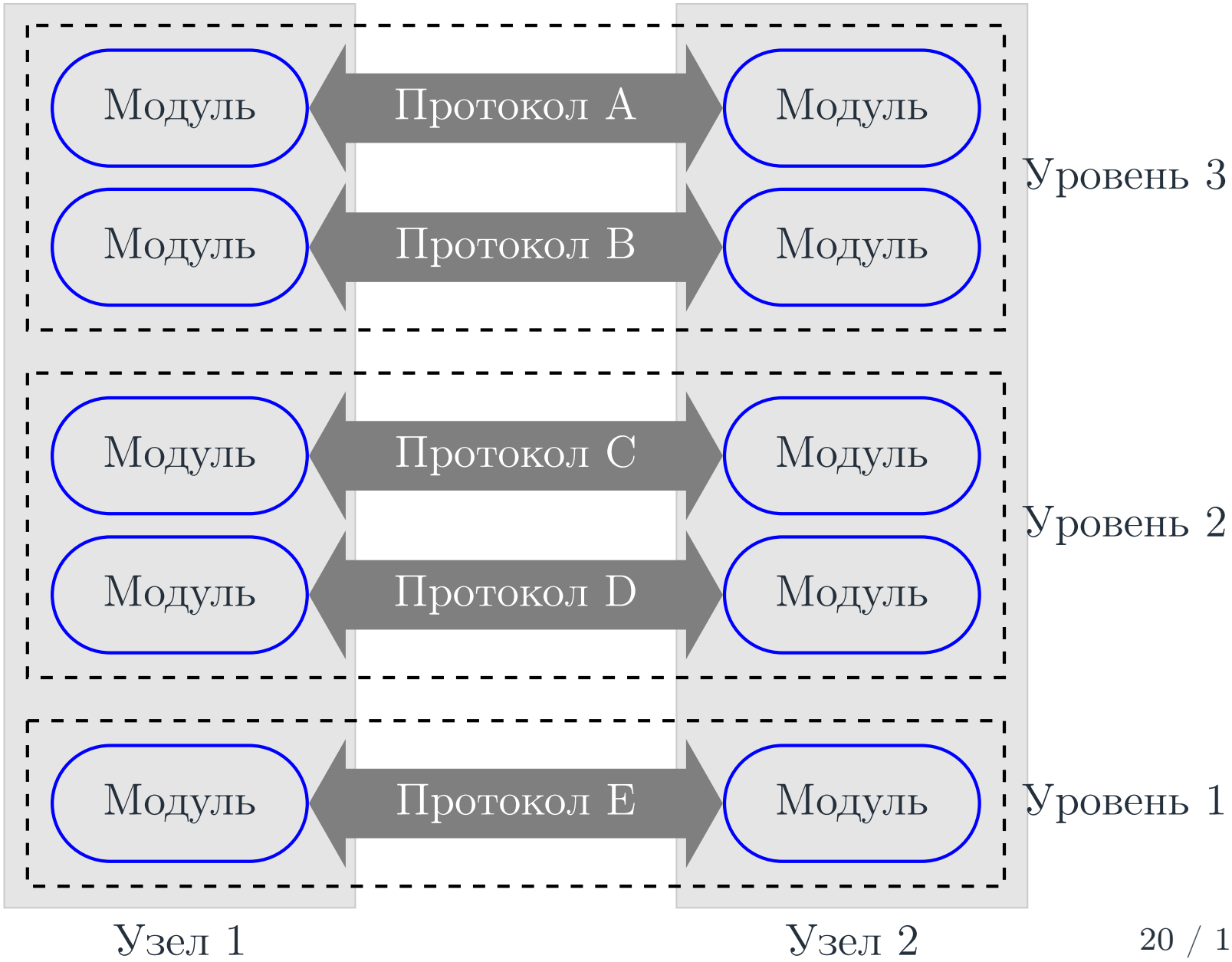
1. *принять HTTP-запрос от клиента;*
2. *выполнить запрос (сформировать HTML-страницу);*
3. *отправить ответ клиенту.*

Для этого Web-сервер делегирует серверу службы TCP следующие действия:

1. *обнаружение входящих соединений от клиентов;*
2. *приём входящего соединения;*
3. *получение потока байтов HTTP-запроса;*
4. *передача клиенту потока байтов ответа.*

Сервер службы TCP, в свою очередь, работает через сервер службы IP.

Схема взаимодействия двух узлов



Пусть служба A i-го уровня использует для передачи своих сообщений службу B $(i-1)$-го уровня.
Сообщение службы A выступает для службы B в роли данных, имеющих неизвестный формат и смысл. Для передачи этих данных служба B должна вставить их в своё собственное сообщение.

Определение 10

Инкапсуляция сообщения – *это заворачивание сообщения в одно или несколько сообщений нижележащего уровня.*

Замечание. Сообщение вышележащего уровня приходится разбивать на части, если формат сообщений службы нижележащего уровня накладывает ограничения на размер передаваемых данных.

Замечание. Заворачивание сообщения осуществляется, как правило, путём добавления к нему заголовка сообщения нижележащего уровня.

Т.к. модули сетевых служб распределены по уровням, то и их протоколы распределены по уровням.

Пример 19. *Web-служба реализована через TCP-службу, расположенную уровнем ниже. Это означает, что протокол HTTP реализован через протокол TCP, расположенный уровнем ниже.*

Определение 11

Стек протоколов – *иерархически организованный набор протоколов, достаточный для организации взаимодействия узлов в сети.*

Для взаимодействия двух узлов на них должны быть реализованы совместимые стеки протоколов.

Замечание. Современные сети основаны на стеке протоколов, называемом TCP/IP. Существует несколько независимых реализаций стека TCP/IP.

Стандартная модель взаимодействия открытых систем

В начале 80-х была разработана так называемая *стандартная модель взаимодействия открытых систем* (Open System Interconnection – OSI).

Цель разработки модели OSI – унификация существовавших на тот момент конкурирующих стеков протоколов (DECnet, TCP/IP, IBM SNA). Унификация была необходима, потому что устройства, разработанные для разных стеков, были несовместимы друг с другом.

Модель OSI мыслилась как первый этап проектирования единого универсального стека протоколов, объединяющего возможности всех существовавших стеков.

Замечание. Универсальный стек так и не был создан (вместо этого победил TCP/IP), но модель OSI оказалась полезной в качестве справочной модели, на которую ориентируются разработчики сетевых протоколов и другие специалисты по компьютерным сетям.

Содержание модели OSI

Модель OSI выделяет семь уровней взаимодействия в сетях с коммутацией пакетов:

1. физический уровень;
2. канальный уровень;
3. сетевой уровень;
4. транспортный уровень;
5. сеансовый уровень;
6. уровень представления;
7. прикладной уровень.

В модели OSI отсутствуют:

- описание реализаций конкретного набора протоколов;
- описание уровня взаимодействия пользовательских приложений (определяются только уровни, реализуемые аппаратно или в рамках операционной системы и системных утилит).

Определение 12

Протоколы физического уровня *предназначены для передачи отдельных битов данных между соседними узлами сети.*

Замечание. Говоря, что узлы сети – соседние, мы подразумеваем, что они соединены общей физической средой (общим кабелем, одним радиодиапазоном и т.п.).

Пример 20. *1000Base-T – протокол физического уровня технологии Ethernet.*
Расчитан на использовании в линиях связи на базе неэкранированной витой пары категории 5 и разъёма RJ-45. Максимальная длина кабеля – 100 метров.
Данные в кабеле представлены в манчестерском коде, *который определяет, каким образом единицы и нули кодируются перепадами напряжения в кабеле.*

Определение 13

Протоколы канального уровня *обеспечивают передачу последовательностей байтов между соседними узлами сети.*

Терминология. Сообщения в протоколах канального уровня называются *кадрами.*

Функции протоколов канального уровня:

- разбиение передаваемых данных на кадры, снабженные контрольными суммами, позволяющими обнаружить и/или исправить ошибки;
- управление доступом к линии связи подключённых к ней узлов (чтобы они передавали в неё данные по очереди, а не все разом);
- синхронизация скоростей передатчика и приёмника;
- адресация узлов, подключённых к линии связи (чтобы было понятно, кто является получателем данных).

Определение 14

Протоколы сетевого уровня *обеспечивают прохождение данных через последовательность транзитных узлов.*

Терминология. Сообщения в протоколах сетевого уровня называются *дейтаграммами.*

Две категории протоколов сетевого уровня:

- **маршрутизируемые протоколы** – реализуют продвижение дейтаграмм по заранее определённому маршруту;
- **протоколы маршрутизации** – собирают информацию о топологии сети, на основании которой осуществляется выбор маршрутов продвижения дейтаграмм.

Замечание. Протоколы сетевого уровня используют собственную систему адресации узлов, отличную от адресации узлов канального уровня.

Определение 15

Протоколы транспортного уровня *передают данные между приложениями, работающими на различных узлах, с нужной степенью надёжности.*

Терминология. Сообщения в протоколах сетевого уровня называются *сегментами.*

Замечание. На одном узле могут быть запущены несколько сетевых приложений. Поэтому на транспортном уровне используется расширенная система адресации, позволяющая идентифицировать отдельное приложение.

Средства обеспечения надёжности могут включать:

- исправление ошибок передачи (искажение, потеря, дублирование и неправильный порядок сегментов);
- контроль переполнения буферов (регулирование скорости потока).

Определение 16

Протоколы сеансового уровня *должны обеспечивать управление диалогом, а именно: установление очерёдности обмена сообщениями и синхронизацию передачи сообщений.*

Синхронизация подразумевает, что при передаче длинного сообщения происходит вставка так называемых контрольных точек, позволяющих в случае отказа не повторять передачу сообщения с самого начала, а возобновлять передачу от последней контрольной точки.

Замечание. В стеке TCP/IP сеансовый уровень не реализован.

Уровень представления модели OSI

Определение 17

Протоколы уровня представления *предоставляют взаимодействующим приложениям средства для унификации данных, которыми они обмениваются.*

Взаимодействующие приложения могут быть написаны на разных языках программирования, могут работать на отличающихся аппаратных платформах под управлением различных ОС. Поэтому у них могут различаться стандарты кодирования текста, представление чисел и т.п.
Протоколы уровня представления должны обеспечивать преобразование между форматами данных, естественными для приложения, и некоторыми унифицированными форматами, в которых данные передаются по сети.

Замечание. В стеке TCP/IP уровень представления не реализован.

Прикладной уровень модели OSI

Определение 18

Протоколы прикладного уровня *предназначены для предоставления пользователю доступа к различным сетевым ресурсам (принтерам, файлам, web-страницам), а также для организации совместной работы пользователей (электронная почта и т.п.).*

Замечание. Собственно, протоколы уровней, лежащих ниже прикладного, абстрагируются от смысла передаваемых данных. Смысл проявляется только на прикладном уровне.

Определение 19

HTTP – *текстовый протокол прикладного уровня, обеспечивающий взаимодействие клиента с* ресурсами *web-сервера и поддерживающий кэширование ответов web-сервера.*

RFC 2616 определяет протокол HTTP версии 1.1:
`https://tools.ietf.org/html/rfc2616`

Замечание. *Ресурс* – это данные, в том или ином виде хранящиеся на сервере.

Замечание. Взаимодействие клиента с ресурсом подразумевает как получение содержимого ресурса, так и изменение ресурса на web-сервере.

Замечание. *Кэширование* – это запоминание пар «запрос–ответ» на клиенте или промежуточном сервере (*кэширующий прокси-сервер*) с тем, чтобы не передавать серверу одни и те же запросы повторно.

Общая схема взаимодействия клиента и web-сервера по протоколу HTTP

- Клиент устанавливает одно или несколько соединений с web-сервером по протоколу TCP.

  Замечание. Использование сразу нескольких соединений может ускорить передачу данных благодаря тому, что web-сервер получает возможность выполнять несколько запросов параллельно.

- В рамках каждого соединения происходит обмен текстовыми данными: клиент отправляет web-серверу HTTP-запросы и получает HTTP-ответы.

  Замечание. Инициатором передачи данных всегда является клиент. Web-сервер не может передавать данные клиенту без запроса.

- Соединения завершаются либо по инициативе клиента (когда клиент получил всё, что нужно), либо по инициативе сервера (когда через соединение не передаются данные в течение некоторого времени).

Адресация ресурсов в рамках HTTP

Адрес ресурса web-сервера указывается в HTTP-запросе и состоит из двух частей:

- доменное имя сайта (например, `www.refal.net`);

  Замечание. Доменное имя необходимо, т.к на одном web-сервере могут располагаться несколько сайтов (*виртуальный хостинг*).

- путь к ресурсу – последовательность имён каталогов, разделённых знаками «/», которая может заканчиваться именем файла и параметризовываться набором пар «ключ–значение».

  Пример 21. `forum/news.html?thread=50&msg=123`

  Замечание. В простейшем случае путь прямо соответствует пути в файловой системе web-сервера.

Формат HTTP-запроса

Стартовая строка	метод	’␣’	путь	’␣’	версия	’\r’	’\n’
Строки заголовка	имя поля	’:’	’␣’	значение		’\r’	’\n’
	...						
	имя поля	’:’	’␣’	значение		’\r’	’\n’
	’\r’	’\n’					
Тело запроса	...						

Пример 22. *В запросе стартовой страницы сайта* `www.refal.net` *используется метод* `GET`*, не подразумевающий наличия тела запроса:*

```
GET / HTTP/1.1
Host: www.refal.net
Connection: close
User-agent: Mozilla/5.0
```

Методы в HTTP-запросе

Метод в стартовой строке HTTP-запроса определяет, какое действие должно быть выполнено с ресурсом:

- `GET` – извлечение ресурса (в ответ web-сервер должен прислать метаинформацию о ресурсе и его содержимое);

  Замечание. *Метаинформация* – это способ кодирования содержимого ресурса, размер содержимого, дата последнего изменения и т.п.

- `HEAD` – извлечение метаинформации (то же, что и `GET`, но содержимое не пересылается);
- `POST` – дополнение ресурса новыми данными, передаваемыми через тело запроса (web-сервер должен прислать метаинформацию и содержимое обновлённого ресурса);

  Замечание. С помощью метода `POST` передаются данные HTML-форм, постинги в форумах и блогах и т.п.

- `PUT` – добавление нового ресурса;
- `DELETE` – удаление ресурса.

Поля заголовка HTTP-запроса

Поля, часто включаемые в заголовок HTTP-запроса:

- `Host` – задаёт доменное имя сайта;
- `Connection` – значение `close` этого поля сообщает web-серверу, что клиент не собирается поддерживать постоянное соединение;
- `User-agent` – информирует web-сервер о типе клиента (можно собирать на сервере статистику по браузерам, а также высылать разным браузерам разные версии ресурсов);
- `Accept-Charset` – задаёт предпочтительную для клиента кодировку символов текста в ответе (например, `utf8`);
- `Accept-Encoding` – задаёт предпочтительный способ кодирования данных в теле ответа (например, значение `gzip` означает сжатие данных).
- `Accept-Language` – задаёт предпочтительный для клиента язык текста ответа (например, `ru`).

Формат HTTP-ответа

Стартовая строка	версия	’␣’	код	’␣’	фраза	’\r’	’\n’
Строки заголовка	имя поля	’:’	’␣’	значение	’\r’	’\n’	
	...						
	имя поля	’:’	’␣’	значение	’\r’	’\n’	
	’\r’	’\n’					
Тело ответа	...						

Пример 23. *Ответ на запрос стартовой страницы сайта* `www.refal.net`*:*

```
HTTP/1.1 200 OK
Date: Thu, 05 Oct 2017 06:20:19 GMT
Server: Apache/2.4.27 (FreeBSD)
X-Powered-By: PHP/5.6.31
Connection: close
Transfer-Encoding: chunked
Content-Type: text/html; charset=windows-1251
...
```

Некоторые коды состояния в HTTP-ответе

Код состояния в стартовой строке HTTP-ответа определяет статус выполнения HTTP-запроса:

- `200 OK` – успешное выполнение запроса;
- `301 Moved Permanently` – запрашиваемый ресурс перемещён;
- `304 Not Modified` – запрашиваемый ресурс не изменился за указанный период времени (возможный ответ на *условный GET-запрос*, порождаемый кэширующим прокси-сервером);
- `400 Bad Request` – запрос не может быть понят;
- `404 Not Found` – запрашиваемый ресурс не существует;
- `505 HTTP Version Not Supported` – версия HTTP-протокола, прописанная в запросе, не поддерживается.

Замечание. Возле каждого кода в списке приведена соответствующая ему фраза, которая указывается после кода в стартовой строке HTTP-ответа.

Поля заголовка HTTP-ответа

Поля, которые часто включаются в заголовок HTTP-ответа:

- `Date` – указывает время и дату создания ответа;
- `Server` – информирует клиента о типе web-сервера;
- `Connection` – значение `close` этого поля означает, что web-сервер собирается закрыть соединение;
- `Last-Modified` – указывает время последниго изменения содержимого ресурса, передаваемого в ответе;
- `Location` – указывает новый адрес ресурса, который был перемещён (это поле добавляется в HTTP-ответ, если код состояния – `301 Moved Permanently`);

Поля заголовка HTTP-сообщения, описывающие передаваемые данные

Следующие поля могут присутствовать как в заголовке HTTP-запроса (в случае использования метода `POST`), так и в заголовке HTTP-ответа, и описывают данные, передаваемые в теле сообщения:

- `Content-Type` – задаёт тип данных, а также кодировку символов в случае текстовых данных;

  Пример 24. *Тип* `text/html` *означает HTML-страницу, а тип* `multipart/form-data` *– набор значений полей HTML-формы (отправляется в HTTP-запросе с методом* `POST`*).*

- `Content-Encoding` и `Transfer-Encoding` – задают способ кодирования данных (сжатие, разбиение на фрагменты и т.п.);
- `Content-Length` – указывает размер данных.

Определение 20

HTTP-прокси – *это web-сервер, обслуживающий клиентов путём перенаправления их запросов на другие web-сервера.*

HTTP-прокси задаётся в настройках клиента (т.е., web-браузера).
Получив HTTP-запрос от клиента, HTTP-прокси выполняет следующие действия:

1. определение web-сервера, которому нужно перенаправить запрос, по значению поля `Host` запроса;
2. передача запроса web-серверу;
3. получение ответа от web-сервера;
4. передача ответа клиенту.

Замечание. Существуют прокси-серверы уровня протокола TCP, которые способны перенаправлять запросы любого протокола, работающего поверх TCP.

Задачи, решаемые с помощью HTTP-прокси

- **Ограничение доступа к определённым ресурсам.**
  Вместо передачи запроса «запрещённому» web-серверу, HTTP-прокси может сразу передавать клиенту HTML-страничку, объясняющую запрет.
- **Очистка HTML-страниц от нежелательного контента.**
  HTTP-прокси может редактировать ответы web-сервера, удаляя из них всплывающие окна, рекламу и т.п.
- **Кэширование ответов web-серверов.**
  HTTP-прокси может сохранять пары «запрос–ответ» в базе данных, называемой *кэшем*.
  Если поступивший от клиента запрос присутствует в кэше, можно сразу отправить клиенту соответствующий ответ, не обращаясь к web-серверу.

  Замечание. Здесь имеется проблема: ресурс на web-сервере изменился, и в кэше хранится его старая версия.

Условные GET-запросы

Для изучения работы протокола HTTP удобно использовать утилиту `netcat`, которая:

- устанавливает TCP-соединение с удалённым узлом;
- передаёт этому узлу текст из стандартного потока ввода;
- выводит в стандартный поток вывода текст, получаемый от узла в ответ.

Замечание. Утилита `netcat` не является стандартной и в некоторых системах называется `nc`, `ncat` или `pnetcat`.

При работе с `netcat` удобно заранее подготовить текстовый файл с HTTP-запросом и связать стандартный поток ввода утилиты с этим файлом. Команда для вызова утилиты:

```
netcat узел порт <файл_запроса
```

Пример 25. *Передача запроса сайту* `www.refal.net`*:*

```
netcat refal.net 80 <query.txt
```

Определение 21

FTP – *протокол прикладного уровня, предназначенный для обеспечения доступа к файловой системе удалённого узла.*

Спецификация протокола FTP приведена в RFC 959:
`https://tools.ietf.org/html/rfc959`

Замечание. Протокол FTP поддерживает аутентификацию пользователей (клиентов).
Однако, аутентификация устроена так, что пароли передаются открытым текстом.
Поэтому в настоящее время используются, главным образом, FTP-серверы с анонимным входом, запрещающие внесение изменений в свою файловую систему.

Определение 22

Управляющее соединение *в протоколе FTP – это постоянное TCP-соединение между клиентом и сервером.*
Оно устанавливается по инициативе клиента и используется для передачи команд от клиента к серверу и ответов от сервера к клиенту.

Команды и ответы передаются в текстовом виде.
Формат команды:

Команда	'\r'	'\n'

или

Команда	'␣'	параметр	'\r'	'\n'

Формат ответа:

Код	'␣'	доп. информация	'\r'	'\n'

Замечание. По умолчанию FTP-сервер слушает входящие управляющие соединения на порту 21.

Определение 23

Передающее соединение *в протоколе FTP – это TCP-соединение между клиентом и сервером, по которому передаётся содержимое файла, описание файла или оглавление каталога.*
Передающее соединение устанавливается отдельно для каждой передачи и завершается сразу же после окончания передачи.

FTP-сервер может работать в двух режимах:

- **Активный режим.**
  В этом режиме FTP-сервер является инициатором передающих соединений. (Не подходит в случае, если клиент находится за firewall'ом.)
- **Пассивный режим.**
  В пассивном режиме передающие соединения устанавливаются клиентом.

Команды FTP, предназначенные для управления сеансом

- `USER логин` – передача идентификатора пользователя серверу (для анонимных FTP-серверов: `anonymous`);
- `PASS пароль` – передача пользовательского пароля серверу (для анонимных FTP-серверов: адрес e-mail);
- `PORT h1,h2,h3,h4,p1,p2` – оповещение сервера об IP-адресе и порте, на котором клиент собирается слушать входящие передающие соединения от сервера (используется при активном режиме работы сервера, `h1`–`h4` – байты IP-адреса, `p1`, `p2` – байты порта);
- `PASV` – переключение сервера в пассивный режим (в ответ сервер сообщает номер порта, на котором он будет слушать входящие передающие соединения от клиента);
- `QUIT` – прекращение сеанса взаимодействия с пользователем (новые команды от клиента больше не принимаются, однако управляющее соединение завершается только по окончании передач данных по передающим соединениям).

Режимы передачи данных в передающем соединении

- **Потоковый режим**.
  Данные передаются «как есть» и не содержат никакой дополнительной информации.
- **Блочный режим**.
  Данные разбиваются на блоки. При передаче каждый блок предваряется заголовком:

Дескриптор (8 битов)	Длина в байтах (16 битов)

  Дескриптор состоит из битовых флагов: бит 5 означает, что блок содержит имя *маркера перезапуска* и задаёт позицию маркера внутри данных, бит 6 – что блок последний в передаче.
  Маркер перезапуска может быть использован для повторной передачи следующих за ним данных в случае, если передача была прервана.
- **Сжатый режим**.
  В этом режиме длинные последовательности одинаковых байтов сжимаются с помощью *группового кодирования*.

Основные управляющие команды FTP

- `CWD путь` – установка текущего каталога;
- `PWD` – запрос текущего каталога (путь возвращается в ответе);
- `MODE S|B|C` – установка режима передачи (`S` – потоковый, `B` – блочный, `C` – сжатый);
- `RETR путь` – скачивание файла с сервера;
- `STOR путь` – отправка файла на сервер;
- `LIST [путь]` – запрос у сервера списка файлов в каталоге (текущем или указанном) или описания файла;
- `REST имя_маркера` – устанавливает позицию, начиная с которой будет осуществляться передача (после этой команды должна следовать `REST`, `STOR` или `LIST`);
- `DELE путь` и `RMD путь` – удаление файла или каталога;
- `MKD путь` – создание каталога;
- `RNFR путь` и `RNTO путь` – переименование файла (сначала старое имя файла отправляется через `RNFR`, а затем через `RNTO` задаётся новое имя файла).

Определение 24

SMTP – *протокол прикладного уровня, предназначенный для передачи электронной почты.*

Спецификация протокола SMTP приведена в RFC 5321: `https://tools.ietf.org/html/rfc5321`
В рамках протокола SMTP клиент устанавливает TCP-соединение с сервером и отправляет серверу электронные сообщения. Стандартный порт сервера – 25.

Замечание. Протокол SMTP – однонаправленный, т.е. для того, чтобы два узла обменивались сообщениями, необходимо, чтобы на каждом из них был запущен как клиент, так и сервер.
В реальности SMTP-серверы запускаются на выделенных почтовых узлах, которые сохраняют полученные сообщения и предоставляют к ним доступ по протоколам POP3 и IMAP.

Определение 25

Электронное сообщение *(письмо) – это информационная структура, состоящая из заголовка и тела.*
Заголовок *состоит из следующих частей:*

- *адреса отправителя и получателей;*
- *дата и тема;*
- *дополнительные отметки о применении шифрования, срочности доставки и т.п.*

Тело – *это данные, отправляемые получателям.*

Спецификация формата электронных сообщений приведена в RFC 5322:
`https://tools.ietf.org/html/rfc5322`

Замечание. Адреса отправителя и получателей имеют вид `пользователь@доменное_имя`.

Сценарий взаимодействия клиента с SMTP-сервером

1. Клиент устанавливает соединение с SMTP-сервером и получает сообщение с кодом 220, если сервер готов принимать почту.

   Сервер:

220	’␣’	приветствие	’\r’	’\n’

   Если сервер не готов (тогда от него приходит сообщение с кодом 554), клиент должен разорвать соединение и повторить попытку позже.

2. Клиент посылает серверу приветствие, которое начинается с `EHLO`, если клиент понимает расширенную версию протокола, или с `HELO` в противном случае.

   Клиент:

EHLO	’␣’	доменное имя клиента	’\r’	’\n’

   В случае `EHLO` сервер откликается перечислением своих расширений в сообщениях с кодом 250.

   Сервер:

250-расширение	’␣’	доп. данные	’\r’	’\n’
...				
250	’␣’	HELP	’\r’	’\n’

Сценарий взаимодействия клиента с SMTP-сервером: продолжение

3. Аутентификация (если сервер поддерживает соответствующее расширение).
4. Клиент выполняет первый этап транзакции, посылая серверу адрес отправителя.

   Клиент: | `MAIL` | `'␣'` | `FROM:<адрес>` | `'\r'` | `'\n'` |

   Сервер отвечает сообщением с кодом 250:

   Сервер: | `250` | `'␣'` | адрес `sender accepted` | `'\r'` | `'\n'` |
5. Клиент выполняет второй этап транзакции, посылая серверу адреса получателей (каждый адрес – отдельным сообщением).

   Клиент: | `RCPT` | `'␣'` | `TO:<адрес>` | `'\r'` | `'\n'` |

   Если сервер знает получателя, он откликается сообщением с кодом 250.

   Сервер: | `250` | `'␣'` | адрес `ok` | `'\r'` | `'\n'` |

   Если пользователь неизвестен, от сервера приходит сообщение с кодом 550.

Сценарий взаимодействия клиента с SMTP-сервером: продолжение

6. Клиент начинает третий этап транзакции.

   Клиент:

DATA	'\r'	'\n'

   Сервер отвечает предложением ввести электронное сообщение.

   Сервер:

354	'␣'	Enter mail, ...	'\r'	'\n'

   Клиент отправляет письмо. При этом конец письма отмечается точкой, расположенной в отдельной строке.

   Клиент:

...		
.	'\r'	'\n'

   На это сервер отвечает сообщением с кодом 250.

   Сервер:

250	'␣'	размер сообщения ...	'\r'	'\n'

7. Клиент может отправить следующее письмо, вернувшись к шагу 4, либо может завершить сеанс.

   Клиент:

QUIT	'\r'	'\n'

Post Office Protocol v.3 (POP3)

Передача письма от пользователя A к пользователю B осуществляется в три этапа:

1. почтовый клиент пользователя A по протоколу SMTP отправляет письмо почтовому серверу, на котором A имеет учётную запись (т.е., «почтовый ящик»);
2. почтовый сервер пользователя A по протоколу SMTP пересылает письмо почтовому серверу, на котором расположен «почтовый ящик» пользователя B;
3. почтовый клиент пользователя B скачивает письмо со своего почтового сервера по протоколу POP3 или IMAP.

Определение 26

POP3 – *протокол прикладного уровня, обеспечивающий скачивание электронных сообщений с почтового сервера, на котором они временно хранятся.*

Спецификация протокола POP3 приведена в RFC 1939: `https://tools.ietf.org/html/rfc1939`

Internet Message Access Protocol (IMAP)

Протокол POP3 расчитан на то, что архив писем хранится на компьютере пользователя, а почтовый сервер обеспечивает лишь временное хранение писем. Это неудобно, если пользователь использует несколько компьютеров.

Определение 27

IMAP – *протокол прикладного уровня, обеспечивающий доступ к электронным сообщениям, постоянно хранящимся на почтовом сервере.*

Спецификация протокола IMAP приведена в RFC 3501: `https://tools.ietf.org/html/rfc3501`

Domain Name System (DNS)

Для маршрутизации пакетов в сетях TCP/IP требуется, чтобы каждому узлу соответствовал IP-адрес.
Тем не менее, на прикладном уровне узлы адресуются с помощью *доменных имён*, т.к.:

- доменные имена более удобны для пользователя;
- они сохраняются при изменении IP-адресов (например, при смене провайдера);
- одно доменное имя может соответствовать нескольким узлам (например, в случае высоконагруженных сайтов).

Определение 28

DNS (система доменных имён) – *распределённая база данных, задающая соответствие доменных имён и IP-адресов.*

Определение 29

Протокол DNS – *протокол прикладного уровня, обеспечивающий перевод доменных имён в IP-адреса.*

Определение 30

Домен – *логическая группа узлов.*

Замечание. Домены не определяются физической организацией сети. Например, домену `com` принадлежат как сервера Apple в США, так и сервера Asus на Тайване.

Домены могут вкладываться друг в друга. Например, домен `apple` вложен в домен `com`. При этом говорят, что `apple` – *поддомен* домена `com`.
Каждый домен имеет имя, уникальное среди непосредственных поддоменов его родительского домена.

Определение 31

Пространство имён DNS – *ориентированное дерево, вершины которого соответствуют доменам и помечены их именами, а дуги задают вложенность доменов и направлены от поддоменов к родительским доменам.*
Корень дерева помечен пустым именем.

Управление доменами

Непосредственные поддомены корня пространства имён DNS называются *доменами первого уровня*.
Домены первого уровня делятся на *родовые* (`com`, `edu`, `net` и т.п.) и *национальные* (`ru`, `by`, `uk` и т.д.).

Замечание. Набор доменов первого уровня контролируется организацией под названием ICANN (Internet Corporation for Assigned Names and Numbers).

Поддомены доменов первого уровня называются *доменами второго уровня*, и т.д.

Замечание. Управление доменами второго уровня осуществляется различными организациями. Например, за поддомены доменов `ru` и `рф` отвечают регистраторы, аккредитованные «Координационным центром национального домена сети Интернет» (`https://cctld.ru`).

Домены третьего уровня, как правило, управляются владельцами соответствующих доменов второго уровня.

Определение 32

Доменное имя – *путь в пространства имён DNS. Доменные имена, заканчивающиеся корнем*, – абсолютные, *все остальные* – относительные.

Доменное имя однозначно идентифицирует домен, соответствующий начальной вершине пути, и записывается как последовательность имён вершин, разделённых точками.

Пример 26. `iu9.bmstu.ru.`

Точка, отделяющая имя домена первого уровня от имени корневого домена (пустой строки), часто не ставится.

Замечание. Следует различать понятия «имя домена» (это строка, которой помечена вершина пространства имён) и «доменное имя» (задаёт путь в пространстве имён).

Имена доменов нечувствительны к регистру символов, и их длина может достигать 63 символов. Длина доменного имени не должна превышать 255 символов.

Зоны пространства имён DNS

Система доменных имён – это распределённая база данных, т.к. всё пространство имён DNS разделено на непересекающиеся *зоны*, и информация о каждой зоне хранится на отдельной группе серверов.

Определение 33

Зона – это поддерево пространства имён DNS.

Замечание. Будем называть *именем зоны* доменное имя её корня.

Определение 34

Зона A называется дочерней *по отношению к зоне B, если непосредственным предком корня зоны A является какая-нибудь вершина зоны B.*

Замечание. Будем говорить, что *узел принадлежит зоне*, если его доменное имя начинается с вершины, принадлежащей этой зоне.

Определение 35

Сервер имён (DNS-сервер) – *это узел, содержащий часть данных системы доменных имён, относящуюся к определённой зоне, а именно:*

- *отображение доменных имён узлов, принадлежащих этой зоне, в их IP-адреса;*
- *доменные имена и IP-адреса серверов имён, обслуживающих дочерние зоны.*

Сервер имён является *уполномоченным* по отношению к узлам, принадлежащим его зоне. Кроме того, сервер имён может кэшировать IP-адреса узлов из других зон, и по отношению к этим узлам он является *неуполномоченным.*

Замечание. Обычно для каждой зоны существуют по крайней мере два сервера имён (первичный и вторичный), информация в которых дублируется. Это важно с точки зрения обеспечения отказоустойчивости работы системы.

Ресурсные записи

Данные на сервере имён представлены в виде множества так называемых *ресурсных записей.*
Ресурсные записи хранятся на сервере имён в текстовом виде в конфигурационном файле, заполняемым администратором зоны. Однако, в сообщениях протокола DNS ресурсные записи передаются в специальном двоичном формате.

Определение 36

Ресурсная запись – *базовый элемент системы доменных имён, привязывающий некоторую информацию (чаще всего, IP-адрес узла) к доменному имени.*

Замечание. Одному доменному имени может соответствовать несколько ресурсных записей. Например, одна запись может задавать IP-адрес соответствующего web-сервера, а другая – имя почтового сервера.

Порядок ресурсных записей, соответствующих одному доменному имени, не играет роли.

Поля ресурсной записи

Ресурсная запись состоит из пяти полей:

- `Доменное_имя` – задаёт домен, к которому относится запись (имя может быть как абсолютным, так и относительным);
- `Время_жизни` – количество секунд, в течение которого запись предположительно не поменяется (в протоколе DNS для ускорения обработки запросов активно применяется кэширование, и это поле задаёт время, в течение которого можно верить ресурсной записи, сохранённой в кэше);
- `Класс` – можно считать, что значение этого поля всегда равно `IN`, что означает «Internet»;
- `Тип` – показывает, как интерпретировать информацию, содержащуюся в поле `Значение`;
- `Значение` – информация, приввзываемая данной ресурсной записью к доменному имени.

Некоторые типы ресурсных записей

- `A` – отображает доменное имя узла в его IP-адрес.
  Пример 27. *На сервере имён зоны «*`bmstu.ru.`*» адрес web-сервера* `iu9.bmstu.ru` *может задаваться как*
  ```
  iu9 86400 IN A 195.19.40.181
  ```
- `NS` – отображает имя зоны в доменное имя сервера имён, обслуживающего эту зону.
  Пример 28. *Сервер имён* `ns.bmstu.ru`*, обслуживающий зону «*`bmstu.ru.`*», может задаваться как*
  ```
  bmstu.ru. 86400 IN NS ns
  ns 86400 IN A 195.19.32.2
  ```
- `MX` – отображает доменное имя узла в доменное имя почтового сервера.
  Пример 29. *Почтовый сервер* `mail.bmstu.ru` *для зоны «*`bmstu.ru.`*» может определяться записями*
  ```
  bmstu.ru. 86400 IN MX mail
  mail 86400 IN A 195.19.32.40
  ```

Ссылки на дочерние зоны

Ресурсные записи типа NS могут использоваться для задания ссылок на сервера имён, обслуживающие дочерние зоны.
Такие ссылки позволяют серверу родительской зоны *делегировать* хранение информации об IP-адресах дочерней зоны серверу дочерней зоны.

Пример 30. *Сервер имён, который обслуживает зону «*`example.com.`*», делегирует обслуживание дочерней зоны «*`zone.example.com.`*» серверу* `ns.zone.example.com`*:*

```
zone.example.com. 86400 IN NS ns.zone.example.com.
```

Ссылки на сервера дочерних зон всегда сопровождаются *добавочными* записями типа `A`, содержащими их IP-адреса.

Пример 31. *Добавочнаая запись:*

```
ns.zone.example.com. 86400 IN A 192.168.70.1
```

Определение 37

Разрешение имени – *процесс получения IP-адреса узла по его доменному имени.*

Взаимодействие с серверами имён, необходимое для разрешения имени, осуществляется по протоколу DNS, описанному в RFC 1035:
`https://tools.ietf.org/html/rfc1035`

Протокол DNS имеет единый двоичный формат запросов и ответов, которые чаще всего передаются по протоколу UDP (на сервере используется порт 53).

Замечание. UDP – протокол транспортного уровня, который отличается от TCP тем, что не подразумевает установку соединения, а также не гарантирует доставку сообщений и порядок, в котором они приходят.

Алгоритм разрешения имён

Разрешение имени можно представить себе в виде следующего простого алгоритма, получающего на вход доменное имя и список IP-адресов серверов имён:

1. послать DNS-запрос, содержащий обрабатываемое доменное имя, очередному серверу имён из списка;
2. подождать, пока от сервера не придёт ответ или пока не пройдёт определённое время ожидания;
3. если ответ не пришёл, то переход к 1;
4. если сервер перенаправляет нас к другим серверам имён, то заменить список IP-адресов серверов имён списком, взятым из ответа, и перейти к 1;
5. вернуть IP-адрес узла, если он содержится в ответе, или сообщение об ошибке.

Замечание. Алгоритм начинает разрешение имён со списка из 13 так называемых *корневых серверов имён.*

Определение 38

Одноранговая сетевая служба – *это служба, в которой клиенты и серверы равноправны.*

Замечание. Клиент одноранговой сетевой службы может выступать в роли сервера, и наоборот.

Пример 32. *Пиринговые файлообменные сети, децентрализованные платёжные системы (криптовалюты), службы луковой маршрутизации (анонимные сети).*

Определение 39

BitTorrent – *протокол прикладного уровня, на основе которого работает одноимённая одноранговая сетевая служба обмена файлами.*

Замечание. Строго говоря, служба BitTorrent является *гибридной*, потому что в ней подразумевается наличие серверов – так называемых *трекеров*.

Определение 40

Торрент – *объект файлообмена, представляющий собой конкатенацию содержимого набора файлов, разбитую на* сегменты *равного размера.*

Замечание. В вырожденном случае торрент содержит всего один файл.

Размер сегмента выбирается автором торрента, исходя из его представлений о желаемом балансе между размером описания торрента и эффективностью передачи торрента:

- маленькие сегменты повышают надёжность передачи при слабой пропускной способности каналов связи;
- большое количество сегментов увеличивают размер описания торрента.

Замечание. Если размер торрента не кратен выбранному размеру сегмента, то последний сегмент короче остальных.

Определение 41

Пир – *узел, который полностью или частично содержит торрент и взаимодействует с другими пирами, получая от них недостающие и отправляя им имеющиеся сегменты.* Рой – *совокупность пиров, работающих с одним торрентом.*

Замечание. Между парами пиров, участвующих в файлообмене, установлены TCP-соединения, по которым в обе стороны передаются сегменты. Пиры – равноправны, т.к.:

- любой пир может выступать в роли сервера, отправляющего сегменты, которые запрашивают другие пиры;
- любой пир может выступать в роли клиента, запрашивающего и получающего сегменты от других пиров.

Определение 42

Сиды *и* личи – *это пиры, которые, соответственно, содержат все или не все сегменты торрента.*

Чтобы установить TCP-соединение, пир должен узнать IP-адрес и порт другого пира.

Определение 43

Трекер – *сервер, позволяющий пирам, которые принадлежат одному рою, найти друг друга.*

Служба BitTorrent может работать вообще без трекеров. В этом случае поиск пиров осуществляется децентрализованно по тому или иному варианту протокола DHT (Distributed Hash Table).

Замечание. Трекер не содержит ни самих торрентов, ни даже имён файлов, содержащихся в торрентах. Его задача – отвечать на запросы пиров, отправляя им информацию о других пирах, принадлежащих требуемому рою. При этом рой идентифицируется хешем, вычисленным на основе *метаданных* торрента.

Определение 44

Метаданные – *информация, позволяющая идентифицировать торрент:*

- *адрес трекера;*
- *имена и размеры файлов;*
- *контрольные суммы сегментов.*

Контрольные суммы сегментов – 160-битные числа (*дайджесты*), вычисленные по содержимому сегментов с помощью алгоритма криптографического хеширования SHA1 (Secure Hash Algorithm 1).

Замечание. Метаданные кодируются в формате Bencode и распространяются среди участников файлового обмена в виде файлов с расширением «`.torrent`». Подразумевается, что участники могут скачивать их с web-серверов, получать по электронной почте и т.п.

Формат Bencode: атомарные значения

Определение 45

Bencode – *формат представления древовидных данных, допускающий эффективную обработку.*

Bencode поддерживает два типа атомарных значений:

- **строка** – представляет собой последовательность байтов и предваряется десятичным числом, задающим размер последовательноси, за которым ставится двоеточие;

  Пример 33. `3:iu9` *и* `5:иу9` – *строки* «`iu9`» *и* «`иу9`».

- **целое число** – записывается в десятичной системе и обрамляется символами `i` и `e`, причём ненулевое число не может начинаться с цифры 0.

  Пример 34. `i-50e` – *число* -50, `i0e` – *число* 0.

Замечание. Строки могут содержать произвольные двоичные данные (не обязательно текст).

Формат Bencode: составные значения

Промежуточные вершины древовидных данных (т.е., составные значения) также бывают двух типов:

- **список** – последовательность закодированных в Bencode элементов, обрамлённая буквами `l` и `e`;

  Пример 35. *Список из трёх целых чисел* 10, 5, −100 *записывается как* `li10ei5ei-100ee`.

- **словарь** – обрамлённая символами `d` и `e` и отсортированная по ключу последовательность пар «ключ–значение», в которой ключи и значения представлены в Bencode и, кроме того, ключи являются строками.

  Пример 36. *Словарь из двух словарных пар* $\langle alpha, 1\rangle$ *и* $\langle beta, 2\rangle$ *записывается как* `d5:alphai1e4:betai2ee`.

Представление метаданных в Bencode

На верхнем уровне метаданные – это словарь с ключами:

- `announce` – адрес трекера, на котором раздаётся торрент;
- `info` – словарь с ключами:
  - `name` – рекомендуемое имя файла или каталога;
  - `piece length` – размер сегмента в байтах;
  - `pieces` – строка, представляющая собой конкатенацию дайджестов SHA1, соответствующих сегментам;
  - `length` – длина файла в байтах (присутствует в случае, если торрент содержит единственных файл);
  - `files` – список описателей файлов (присутствует в случае, если торрент содержит несколько файлов), причём каждый описатель – это словарь с ключами:
    - `length` – длина файла в байтах;
    - `path` – список закодированных в UTF-8 строк, последняя из которых соответствует имени файла, а остальные задают путь к файлу.

Взаимодействие пира и трекера

Протокол BitTorrent расчитан на то, что пир может в любой момент присоединяться к рою или покидать рой. Соответственно, взаимодействие пира и трекера решает две задачи:

- извещение трекера о событиях, происходящих с пиром (таких, как присоединение и выход из роя);
- предоставление пиру списка других пиров, в данный момент принадлежащих рою.

Пир обращается к трекеру по протоколу HTTP: он отправляет HTTP-запрос, в стартовой строке которого указывается метод `GET` и путь, в котором в виде набора «ключ–значение» закодированы параметры запроса.

Замечание. Существует отдельный протокол взаимодействия пира и трекера, реализованный поверх UDP. Он более эффективен, т.к не требует установления TCP-соединения. Этот протокол описан в `http://www.bittorrent.org/beps/bep_0015.html`.

Основные ключи в запросе пира к трекеру

- `info_hash` – 20-байтовый SHA1-дайджест, вычисленный по значению ключа `info` метаданных торрента.
- `peer_id` – случайная 20-байтовая строка, которая идентифицирует пир в рое (генерируется самим пиром);
- `port` – номер порта, на котором пир обещает ожидать входящие соединения от других пиров роя;
- `event` – может не присутствовать или содержать одно из трёх значений:
  - `started` – означает, что пир начинает загрузку торрента (присоединение к рою);
  - `completed` – отправляется, когда пир полностью получил торрент;
  - `stopped` – означает, что пир прекращает работу с торрентом (выход из роя).

Замечание. Ключ `event` отсутствует в запросах к трекеру, которые периодически отправляются пиром, чтобы подтвердить, что пир ещё «жив».

Ответ трекера на запрос от пира

Ответ трекера, закодированный в Bencode, приходит в теле HTTP-ответа в виде словаря с ключами:

- `failure reason` – если присутствует, содержит описание ошибки (др. ключи ответа в этом случае отсутствуют);
- `interval` – время в секундах, в течение которого пир должен отправить трекеру следующий запрос (если не отправит, трекер будет считать, что пир выбыл из роя);
- `peers` – либо список пиров, каждый из которых описывается словарём с ключами
  - `peer id` – идентификатор пира;
  - `ip` и `port` – IP-адрес и порт пира;

  либо строка, в которой каждому пиру соответствует 6 байтов: 4 на IP-адрес и 2 на порт.

  Замечание. Трекер отправляет данные не о всех пирах роя, а о случайно выбранных N пирах, где N, как правило, равно 50.

Состояние пира в рое

В каждый момент времени пир в рое обладает следующей информацией:

- частичный или полный набор сегментов торрента;
- полученный от трекера список из N других пиров, принадлежащих рою (*соседей*);
- списки сегментов, имеющихся у каждого из соседей.

Кроме того, в каждый момент времени пир поддерживает набор параллельных TCP-соединений с соседями, часть из которых инициировал он сам, а часть были установлены по инициативе соседей.
Состояние пира постоянно обновляется, т.к.:

- к пиру приходят от соседей недостающие сегменты;
- каждый ответ от трекера содержит обновлённый список соседей;
- пир получает от соседей обновлённые списки имеющихся у них сегментов.

Действия пира в рое

Во время нахождения в рое пир может выполнять следующие действия:

- установка TCP-соединений с новыми соседями;
- блокировка и разблокировка соседей;
- отправка запросов к трекеру и приём от него ответов;
- извещение соседей о только что полученных сегментах;
- запрос и получение недостающих сегментов у соседей;
- обработка запросов от соседей.

Замечание. **Действия пира в рое недетерминированы**. Поэтому он вынужден в реальном времени решать нетривиальную оптимизационную задачу, которая в основном заключается в поиске ответа на два вопроса:

1. какие недостающие сегменты нужно в первую очередь запросить у соседей?
2. удовлетворение запросов каких соседей наиболее приоритетно?

Стратегия пира в рое

Все пиры в рое в идеальном случае решают общую задачу: **наиболее быстрое распространение сегментов торрента среди пиров роя**. Соответственно, стратегия пира определяется двумя эвристиками.

1. **Эвристика «сначала редкие»**. Пир в первую очередь запрашивает у соседей те недостающие сегменты, которые наиболее редки, т.е. имеются у минимального количества соседей.
2. **Эвристика «торговли»**. Пир приоритетным образом удовлетворяет запросы тех соседей, которые поставляют ему сегменты на самой высокой скорости.
   Пир постоянно измеряет скорость передачи данных от соседей и в каждый момент времени обрабатывает запросы от 4 соседей с наиболее высокой скоростью и от одного случайного соседа, который меняется каждые 30 секунд (даём ему «продегустировать» нашу скорость). Остальные соседи *блокируются*.

Протокол взаимодействия двух пиров

После установления TCP-соединения два пира обмениваются «рукопожатиями», а затем в обе стороны следуют потоки байт, состоящие из сообщений.
«Рукопожатие» – это специальное сообщение, имеющее следующий формат:

- строка `BitTorrent protocol`, предваряемая байтом со значением 19;
- 8 нулевых зарезервированных байтов;
- 20-байтовый SHA1-дайджест, идентифицирующий торрент;
- 20-байтовый идентификатор пира.

Перед каждым сообщением указывается его длина в байтах, представляющая собой 4-байтовое беззнковое целое число в big-endian.

Замечание. Вообще, все целые числа передаются в рамках протокола в таком виде.

Типы сообщений

Каждое сообщение начинается с байта, задающего его тип:

0. `choke` – сообщает о блокировке пира;
1. `unchoke` – сообщает о разблокировке пира;
2. `interested` – сообщает о том, что отправитель заинтересован в получении сегментов;
3. `not interested` – обратное к interested;
4. `have` – сообщение состоит из номера только что полученного сегмента;
5. `bitfield` – сообщение состоит из битовой маски, показывающей наличие сегментов у отправителя (отправляется один раз в самом начале);
6. `request` – сообщение состоит из индекса сегмента, а также смещения и длины запрашиваемого фрагмента сегмента;
7. `piece` – сообщение содержит индекс сегмента, смещение в нём и последовательность байтов;
8. `cancel` – отменяет запрос на сегмент.

На одном узле могут одновременно работать несколько сетевых приложений.
Сетевые службы транспортного уровня распредяют получаемые из сети данные по приложениям, а также собирают данные, формируемые приложениями, для отправки в сеть.
Приложения принимают и отправляют данные через *порты*.

Определение 46

Порт – *совокупность двух очередей, привязанных к запущенному на узле приложению и обслуживаемых определённой сетевой службой транспортного уровня:*

- *очередь, в которую служба помещает приходящие из сети данные, предназначенные этому приложению;*
- *очередь, из которой служба забирает данные, сформированные приложением для отправки в сеть.*

Замечание. К одному сетевому приложению при желании можно привязать несколько портов.

Определение 47

Номер порта – *16-разрядное беззнаковое число, которое уникально в рамках, задаваемых сетевым интерфейсом узла и протоколом транспортного уровня, и используется для идентификации порта.*

Замечание. Узел может иметь несколько сетевых интерфейсов (т.е., сетевых карт). Через один порт приложение может взаимодействовать с сетью, используя только один из доступных интерфейсов, поэтому один и тот же номер можно использовать для двух портов, если они привязаны к разным интерфейсам.

Замечание. Каждый порт обслуживается определённой сетевой службой, поэтому один и тот же номер можно использовать для двух портов, если они обслуживаются разными службами.

Категории номеров портов

Номера портов делятся на три категории:

- **системные (или общеизвестные)** $(0 \ldots 1023)$ – идентифицируют порты, привязанные к системным процессам, которые обеспечивают работу популярных сетевых служб (HTTP, FTP, SMTP, DNS и т.п.);

  Замечание. Присвоение порту системного номера в большинстве ОС требует прав администратора.

- **зарегистрированные (или пользовательские)** $(1024 \ldots 49151)$ – используются сетевыми службами, зарегистрированными в Internet Assigned Numbers Authority (IANA);

  Замечание. Регистрация номеров портов снижает вероятность конфликтов между сетевыми приложениями.

- **динамические (или частные)** $(49152 \ldots 65535)$ – используются частными сетевыми службами, а также динамически назначаются *эфемерным* портам.

Определение 48

Эфемерный порт – *временный порт, создаваемый, как правило, на стороне клиента сетевой службы, чтобы клиент мог взаимодействовать через него с сервером.*

Пример 37. *При взаимодействии web-браузера и web-сервера на стороне web-браузера – эфемерный порт, а на стороне web-сервера – порт с системным номером 80.*

Пример 38. *В передающем TCP-соединении, которое устанавливает работающий в активном режиме FTP-сервер с FTP-клиентом, эфемерный порт, вопреки обыкновению, создаётся на стороне сервера.*

Для эфемерного порта автоматически выбирается один из незадействованных динамических номеров портов.

Определение 49

UDP – *протокол транспортного уровня, предназначенный для передачи дейтаграмм между приложениями и работающий без установки соединения.*

Спецификация протокола UDP приведена в RFC 768: `https://tools.ietf.org/html/rfc768`

Замечание. UDP по сути является примитивной обёрткой вокруг протокола сетевого уровня IP, добавляющей поддержку портов. За доставку дейтаграмм между узлами сети целиком отвечает IP, и поэтому они могут теряться по пути, дублироваться и приходить в неправильном порядке.

Замечание. UDP может работать как поверх традиционного IPv4 (4-байтовые IP-адреса), так и поверх IPv6 (16-байтовые IP-адреса). Версия нижележащего протокола IP влияет на вычисление контрольных сумм дейтаграмм.

Формат дейтаграммы UDP

- `порт отправителя` – 0 или номер порта, на который предполагается получить ответную дейтаграмму;
- `порт получателя` – номер порта на узле, которому адресована дейтаграмма;
- `длина дейтаграммы` – длина в байтах, включающая 8 байтов заголовка;
- `контрольная сумма` – целое число, позволяющее получателю проверять целостность дейтаграммы (может не использоваться в случае работы поверх IPv4);
- `данные` – массив байтов (тело дейтаграммы).

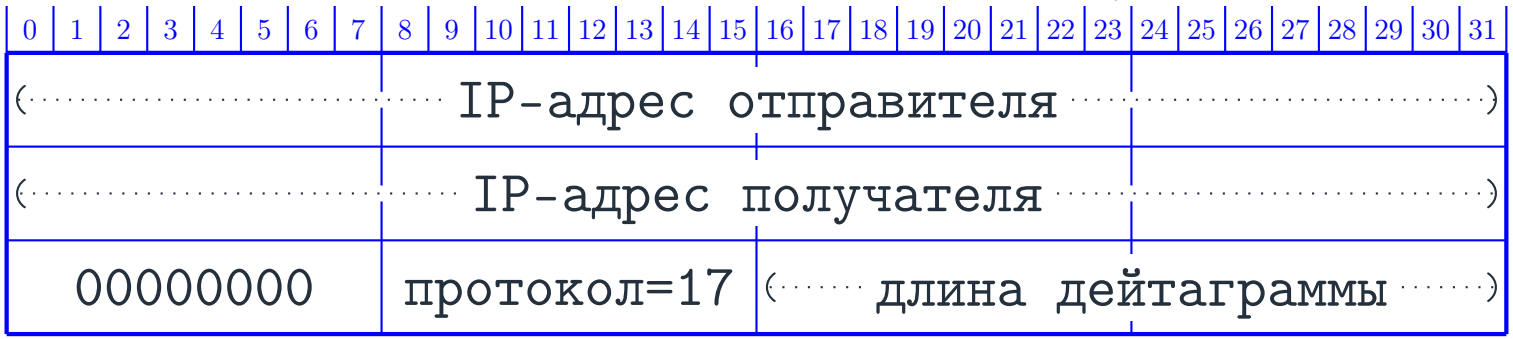
Замечание. Целочисленные поля имеют порядок байтов big-endian, хотя это и не отражено в RFC.

Подготовка к вычислению контрольной суммы

Формирование данных для вычисления контрольной суммы:

- в поле «`контрольная сумма`» записывается 0;
- если длина дейтаграммы нечётна, в конец добавляется нулевой байт;
- к дейтаграмме временно добавляется псевдозаголовок.

Псевдозаголовок «мимикрирует» под заголовок пакета IP, в который будет вложена UDP-дейтаграмма для передачи по сети. Поэтому фактически контрольная сумма будет вычисляться по пакету IP. В случае использования IPv4 псевдозаголовок имеет следующий вид:



Замечание. Добавление псевдозаголовка даёт защиту от неправильно маршрутизированных дейтаграмм.

Алгоритм вычисления контрольной суммы

Подготовленные данные, размер которых всегда чётный, рассматриваются как массив 16-разрядных беззнаковых целых чисел.

Алгоритм вычисления контрольной суммы не особенно надёжен, но зато допускает эффективную программную реализацию:

1. элементы массива суммируются, причём в качестве аккумулятора выступает 32-разрядная беззнаковая целочисленная переменная `sum`;
2. старшее и младшее 16-разрядные слова переменной `sum` складываются, а результат записывается в 16-разрядную беззнаковую целочисленную переменную `x`;
3. если при сложении на шаге 2 возникло переполнение, то к `x` прибавляется 1;
4. контрольной суммой будет являться побитовое НЕ от `x`, если `x` не равно `0xFFFF` (в противном случае контрольная сумма – число `0xFFFF`).

Алгоритм вычисления контрольной суммы: реализация на C


```
1  unsigned short checksum(unsigned short *buf, int n)
2  {
3      unsigned int sum = 0;
4      for (int i = 0; i < n; i++) {
5          sum += buf[i];
6      }
7
8      while (sum >> 16) {
9          sum = (sum & 0xFFFF) + (sum >> 16);
10     }
11
12     return sum == 0xFFFF ? 0xFFFF : ~sum;
13 }
```

Замечание. Интересным свойством алгоритма является его независимость от порядка байт в слове.
Если порядок байт – little-endian, мы получим контрольную сумму в little-endian.
Если порядок байт – big-endian, мы получим ту же самую контрольную сумму, но уже в big-endian.

Проверка контрольной суммы

Пусть $s \neq$ `0xFFFF` – контрольная сумма UDP-дейтаграммы, которая, как мы знаем, вычислялась при нулевом значении поля «`контрольная сумма`».

Обозначение. Пусть $\overline{s}$ – побитовое НЕ от s.

Так как UDP-дейтаграмма, полученная из сети, содержит в поле «`контрольная сумма`» значение s, то, натравив на неё алгоритм вычисления контрольной суммы, мы получим в переменной `sum` значение

$$\overline{s} + s \equiv (-s - 1) + s \equiv -1 \equiv \texttt{0xFFFF} \pmod{2^{16}}.$$

Отсюда следует, что алгоритм вернёт число `0xFFFF`.

Замечание. Нетрудно убедиться, что тот же результат получится при $s =$ `0xFFFF`.

Тем самым, критерий правильности полученной UDP-дейтаграммы: посчитанная для неё контрольная сумма должна равняться `0xFFFF`.

Ещё о контрольной сумме

Замечание. Легко убедиться, что вычисленная приведённым алгоритмом контрольная сумма не может принимать значение 0.

Когда нижележащим протоколом для UDP является IPv4, разрешается отправлять UDP-дейтаграммы без контрольной суммы. Для этого в соответствующее поле заголовка записывается 0.

Замечание. Вариант алгоритма вычисления контрольной суммы, используемый в TCP, отличается на шаге 4:

4. контрольной суммой будет являться побитовое НЕ от `x`, ~~если `x` не равно `0xFFFF` (в противном случае контрольная сумма — число `0xFFFF`)~~.

Тем самым, в TCP не применяется фокус, обеспечивающий неравенство контрольной суммы нулю.

Определение 50

TCP – *протокол транспортного уровня, предназначенный для надёжной передачи потоков байтов между приложениями и работающий на основе логического соединения.*

Спецификация протокола TCP приведена в RFC 793: `https://tools.ietf.org/html/rfc793`

Замечание. Служба TCP разбивает поток данных на *сегменты* и передаёт их по протоколу IP, стараясь, чтобы каждый сегмент помещался в один пакет IP.

Определение 51

Логическое соединение – *это метод обмена данными между двумя приложениями, позволяющий участникам обмена следить за тем, чтобы данные не были потеряны или продублированы, а также чтобы они пришли к получателю в том порядке, в каком были отправлены.*

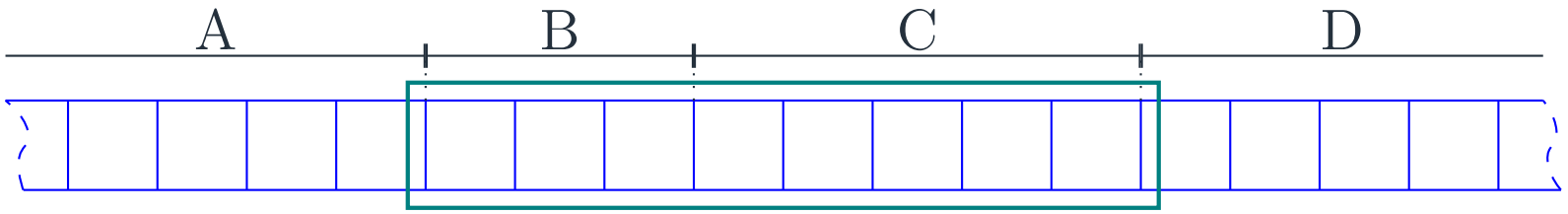
Схема надёжной передачи данных в TCP

Надёжность в TCP обеспечивается подтверждением доставки сегментов, которое работает следующим образом:

- байты потока данных, следующие от отправителя к получателю, имеют *последовательные номера*;
- каждый сегмент, посылаемый отправителем, содержит последовательный номер байта, с которого этот сегмент начинается;
- получатель периодически посылает отправителю *подтверждающий номер*, сообщающий об успешном приёме всех байтов, предшествующих байту с этим номером;
- отправитель повторно отправляет сегменты, подтверждение на содержимое которых не пришло в течение определённого промежутка времени.

Замечание. Соединение в TCP – *дуплексное*, т.е. в его рамках существуют два потока данных, и каждое из взаимодействующих приложений может выступать как в роли отправителя, так и в роли получателя.

Механизм управления потоком в TCP



Окно управления потоком

С точки зрения отправителя последовательные номера потока делятся на 4 категории:

- A – старые последовательные номера, на которые получено подтверждение;
- B – последовательные номера отправленных, но ещё неподтверждённых данных;
- C – последовательные номера, разрешённые для передачи новых данных;
- D – будущие последовательные номера, которые ещё не разрешено использовать.

Номера категорий B и C образуют *окно управления потоком*, размер которого согласуется с получателем, чтобы синхронизировать скорости отправителя и получателя.

Механизм управления перегрузкой в TCP

Скорость передачи данных определяется не только возможностями отправителя и получателя, но и состоянием транзитных узлов, расположенных между ними.
Протокол TCP определяет механизм управления перегрузкой, работающий по следующей схеме:

- отправитель оценивает пропускную способность сети по количеству сегментов, которые ему приходится повторно отправлять;
- на основании оценки пропускной способности сети отправитель изменяет размер так называемого *окна перегрузки* (руководствуясь принципом **аддитивное увеличение/мультипликативное уменьшение**);
- число байт, которые отправитель может передать в сеть, не может превышать размера окна перегрузки, даже если размер окна управления потоком позволяет передавать больше данных.

Формат сегмента TCP

Сегмент TCP состоит из заголовка и следующего сразу за ним тела. *Тело* – массив байтов, передаваемых в сегменте. Заголовок сегмента имеет переменный размер:

Поля заголовка, обеспечивающие целостность сегмента:

- `длина заголовка` – смещение тела относительно начала сегмента (задаётся в 32-разрядных словах);
- `контрольная сумма` – вычисляется по сегменту с приписанным к нему псевдозаголовком (аналогично UDP).

Формат сегмента TCP: продолжение

Поля заголовка, обеспечивающие адресацию:

- `порт отправителя` – номер порта на узле-отправителе;
- `порт получателя` – номер порта на узле-получателе.

Передача данных обеспечивается следующими полями:

- `последовательный номер (посл#)` – последовательный номер первого байта тела сегмента в потоке от отправителя к получателю;
- `подтверждающий номер (подтв#)` – последовательный номер первого ещё не полученного байта в потоке от получателя к отправителю;
- `размер окна` – количество байтов, которое отправитель в данный момент готов принять от получателя;
- `указатель срочности` – указывает, какое количество байтов, расположенных в начале сегмента, являются *срочными данными*.

Замечание. В современных протоколах прикладного уровня указатель срочности не используется.

Формат сегмента TCP: продолжение

Смысл битов в поле «`флаги`» (при установке бита в 1):

0. `URG` – поле «`указатель срочности`» несёт значимую информацию;
1. `ACK` – поле «`подтверждающий номер`» несёт значимую информацию;
2. `PSH` – пересылаемые данные должны быть немедленно выданы получателю, даже если буфер, в котором получатель накапливает данные, ещё не заполнен;
3. `RST` – сброс соединения;
4. `SYN` – поле «`последовательный номер`» содержит номер байта, с которого начинается поток от отправителя к получателю (подготовка к установлению соединения);

   Замечание. Байты в потоках нумеруются не с 0, а с произвольных значений, выбираемых отправителями.

5. `FIN` – отправитель больше не будет передавать данные (подготовка к завершению соединения);

Формат сегмента TCP: продолжение

Поле «опции» имеет переменный размер и состоит из последовательности опций.
Опция может кодироваться двумя способами:

- единственный байт, содержащий код опции;
- байт кода опции, байт длины опции, и последовательность байтов тела опции.

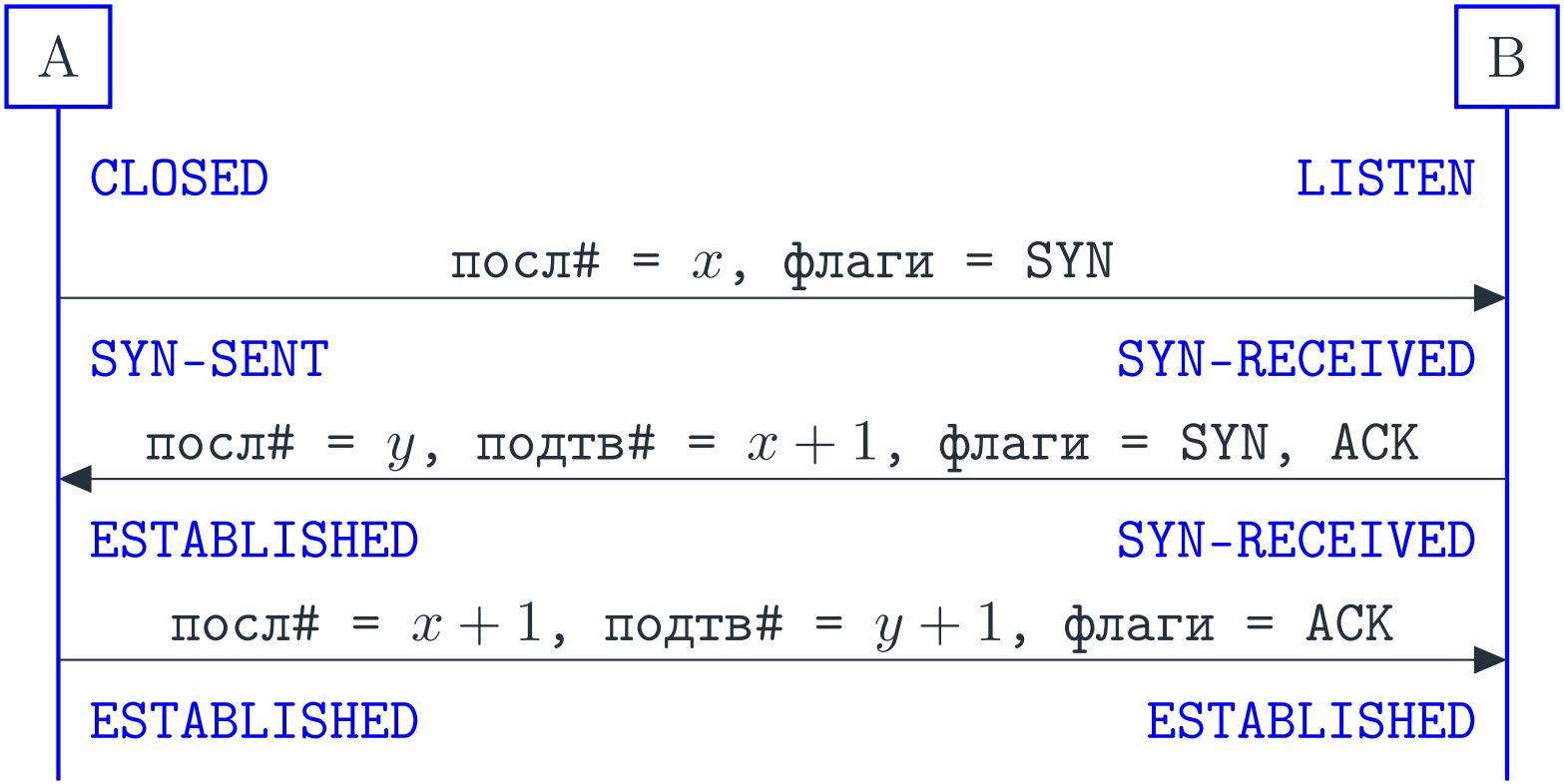
Опции, определённые в RFC 793:

Код	Длина	Значение
0	–	Последняя опция в последовательности
1	–	Заполнитель (в целях выравнивания)
2	4	Максимальный размер сегмента (16 бит, используется при установлении соединения)

Замечание. Опции позволяют расширять протокол TCP без изменения формата заголовка сегмента.

Установка соединения

Для установки соединения между процессом A, находящимся в состоянии `CLOSED`, и процессом B, находящимся в состоянии `LISTEN`, процессы обмениваются тремя сегментами (процедура *тройного рукопожатия*).



Замечание. Как видно из диаграммы, флаг `SYN` занимает один последовательный номер.

Задачи, решаемые тройным рукопожатием

- **Синхронизация последовательных номеров**
  Обе стороны в процессе установки соединения сообщают друг другу последовательные номера первых байтов в потоках данных, которыми они будут обмениваться.

  Замечание. Номера выбираются таким образом, чтобы не спутать сегменты нового соединения с «заблудившимися» сегментами, относящимися к старым соединениям.

- **Восстановление от дублирующих SYN-сегментов**
  Процесс B, находящийся в состоянии `LISTEN`, может получить «заблудившийся» уже не актуальный IP-пакет, содержащий сегмент с флагом `SYN`.
  Процессу B придётся ответить на этот пакет отправкой сегмента с флагами `SYN+ACK` процессу A.
  Благодаря тройному рукопожатию процесс A имеет возможность сообщить процессу B, что он на самом деле не хочет устанавливать соединение: для этого он отправляет процессу B сегмент с флагом `RST`.

Особенности передачи данных в TCP

- **Отложенные подтверждения.** Получатель не обязан подтверждать каждый полученный сегмент по отдельности: он может в течение некоторого времени накапливать сегменты, а потом подтвердить их все разом.
- **Отправка подтверждений вместе с данными.** Для отправки подтверждения не обязательно создавать отдельный пустой сегмент – подтверждения, как правило, передаются в сегментах с данными встречного потока.
- **Пробные сегменты.** Даже если получатель в сегменте с подтвержденим получения данных установил 0 в поле «`размер окна`», разрешается передавать ему 1-байтовые *пробные сегменты*.
  Пробный сегмент передаётся в случае, если получатель в течение долгого времени не сообщает об увеличении размера окна. Предполагается, что сегмент с этим сообщением может потеряться.

Завершение соединения

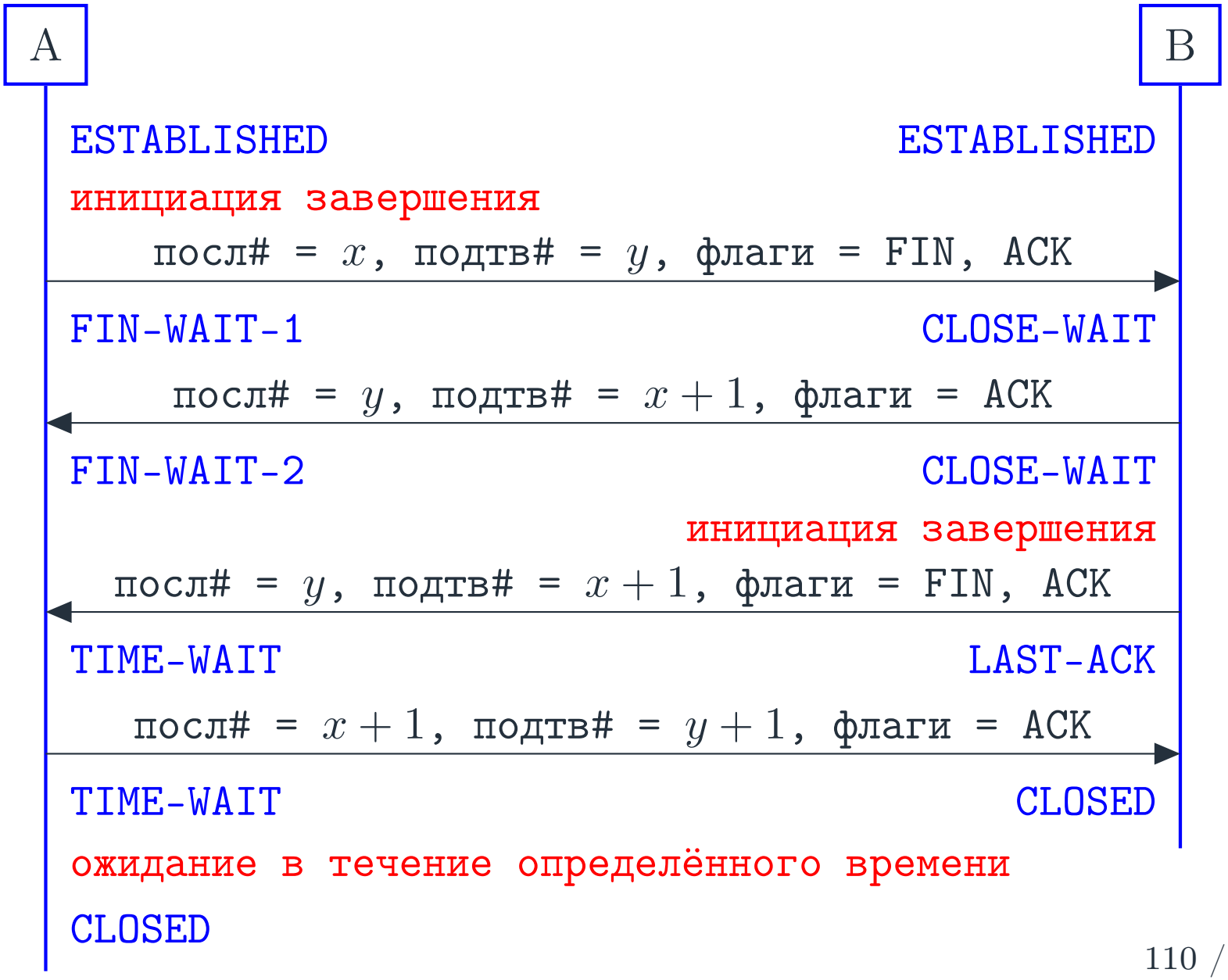
Для *корректного завершения соединения* требуется согласие обеих сторон, потому что каждая сторона ответственна за закрытие потока данных, для которого она выступает в роли отправителя.

Замечание. *Некорректное завершение соединения* происходит по прошествии определённого времени после того, как одна из сторон перестанет высылать подтверждения на передаваемые ей данные.

Процесс, решивший завершить соединение, должен отправить сегмент с установленным флагом `FIN` и дождаться подтверждения получения этого сегмента.

Замечание. Ситуация, когда процесс A закрыл свой поток данных, а процесс B на другом конце соединения свой поток не закрыл и продолжает передавать по нему данные, является нормальной. Предполагается, что процесс A должен продолжать принимать данные и дожидаться закрытия потока со стороны процесса B.

Диаграмма завершения соединения



Любая современная сетевая операционная система предоставляет стандартный набор системных вызовов, составляющий классический *сокетный API*.

Замечание. Сокетный API предоставляет доступ к службам транспортного и сетевого уровня.

Определение 52

Сокет – *абстрактный объект, представляющий конечную точку сетевого взаимодействия.*

Любое сетевое взаимодействие определяется и осуществляется парой сокетов.

Замечание. В UNIX-системах сокеты рассматриваются как файловые объекты и, как и файлы, идентифицируются уникальными в рамках процесса номерами – *файловыми дескрипторами.*

Замечание. В Windows дескрипторы сокетов представлены переменными типа `SOCKET`.

Создание сокета (socket)

Функция `socket` создаёт сокет и возвращает его дескриптор (в случае неуспеха возвращает `-1`).

```
#include <sys/socket.h>
int socket(int domain, int type, int protocol);
```

- `domain` – область использования сокета (`AF_INET` – Интернет, `AF_UNIX` – UNIX-сокет для межпроцессного взаимодействия в рамках одного узла);
- `type` – тип сокета:
  - `SOCK_STREAM` – надёжный протокол на основе установления логического соединения (TCP);
  - `SOCK_DGRAM` – ненадёжный дейтаграммный протокол (UDP);
  - `SOCK_RAW` – низкоуровневый доступ (IP);
- `protocol` – имеет смысл, когда `type == SOCK_RAW`, в остальных случаях передаётся `0`.

Установка соединения (connect)

Функция `connect` устанавливает соединение в случае применения к сокету типа `SOCK_STREAM`, или же просто приписывает сокету адрес удалённого узла в случае `SOCK_DGRAM` и `SOCK_RAW`. Возвращаемое значение `0` означает успех, а `-1` – неудачу.

```
#include <sys/socket.h>
int connect(int s, const struct sockaddr *peer,
            int peer_len);
```

- `s` – дескриптор сокета, через который будет осуществляться передача данных;
- `peer` – указатель на структуру, в которой записан адрес удалённого узла;

  Замечание. Тип структуры определяется областью использования сокета. Для `AF_INET` – это `sockaddr_in`.
- `peer_len` – размер в байтах структуры, на которую указывает `peer`.

Структура адреса (sockaddr_in)

Структура `sockaddr_in` используется для хранения адресов в области `AF_INET`.

```
#include <netinet/in.h>

struct sockaddr_in {
    sa_family_t     sin_family;
    in_port_t       sin_port;
    struct in_addr  sin_addr;
};

struct in_addr {
    uint32_t        s_addr;
};
```

- `sin_family` – должно быть равно `AF_INET`;
- `sin_port` – номер порта (в big-endian) для сокетов типа `SOCK_STREAM` и `SOCK_DGRAM`, либо версия протокола IP для сокетов типа `SOCK_RAW`;
- `s_addr` – IP-адрес (в big-endian).

Заполнение структуры адреса

Для преобразования номера порта в big-endian удобно воспользоваться функцией `htons`.

```
#include <netinet/in.h>
uint16_t htons(uint16_t hostshort);
```

Функция `inet_addr` преобразует IP-адрес из строкового представления в 32-разрядное число в big-endian.

```
#include <arpa/inet.h>
uint32_t inet_addr(const char *cp);
```

Пример 39. *Фрагмент программы, в котором в переменную* `peer` *помещается адрес 127.0.0.1:6060:*

```
1 struct sockaddr_in peer;
2 peer.sin_family = AF_INET;
3 peer.sin_port = htons(6060);
4 peer.sin_addr.s_addr = inet_addr("127.0.0.1");
```

Получение и передача данных (recv и send)

Функции `recv` и `send` выполняют получение и передачу данных через сокет, для которого уже была вызвана функция `connect`. Функции возвращают размер полученных / переданных данных или `-1` в случае ошибки.

```
#include <sys/socket.h>
int recv(int s, void *buf,
         size_t len, int flags);
int send(int s, const void *buf,
         size_t len, int flags);
```

- `s` – дескриптор сокета;
- `buf` и `len` – указатель на буфер для записи полученных данных / чтения передаваемых данных и его размер;
- `flags` – флаги:
  - `MSG_OOB` – получение / передача срочных данных;
  - `MSG_PEEK` – не удалять полученные данные из приёмного буфера (т.е. их можно будет ещё раз получить при последующем вызове `recv`).

Получение и передача данных через дейтаграммные сокеты (recvfrom и sendto)

Функции `recvfrom` и `sendto` выполняют получение и передачу данных через дейтаграммные сокеты, для которых функция `connect` не вызывалась. Возвращают размер полученных / переданных данных или `-1` в случае ошибки.

```
#include <sys/socket.h>
int recvfrom(int s, void *buf, size_t len,
    int flags, struct sockaddr *from, int *fromlen);
int sendto(int s, const void *buf, size_t len,
    int flags, const struct sockaddr *to, int tolen);
```

- `s`, `buf`, `len`, `flags` – то же, что и для `recv` и `send`;
- `from` – указатель на структуру, в которую `recvfrom` запишет адрес отправителя дейтаграммы;
- `fromlen` – указатель на переменную с размером адреса отправителя (`recvfrom` может переписать её значение);
- `to` – указатель на структуру, в которую перед вызовом `sendto` нужно записать адрес получателя дейтаграммы;
- `tolen` – размер адреса получателя.

Завершение TCP-соединения (shutdown)

Для корректного завершения TCP-соединения стороны соединения должны обменяться `FIN`-сегментами.
Если этого не произойдёт, соединение будет разорвано только при истечение некоторого временного интервала.
Отправку `FIN`-сегмента осуществляет функция **`shutdown`**.
Она возвращает `0` в случае успеха и `-1` – в случае неудачи.

```
#include <sys/socket.h>
int shutdown(int s, int how);
```

- `s` – дескриптор сокета;
- `how` – определяет, какой из двух потоков данных нужно закрыть (0 – входящий, 1 – исходящий, 2 – оба).

  Замечание. Отправку `FIN`-сегмента вызывает `how=1`. Значения 0 и 2 по-разному реализованы в различных операционных системах, и поэтому их использование не рекомендуется.

Закрытие сокета (close)

Закрытие дескриптора сокета, как и всякого файлового дескриптора, осуществляется в UNIX-системах функцией `close`, которая возвращает `0` в случае успеха и `-1` в случае неудачи.

```
#include <unistd.h>
int close(int fd);
```

- `fd` – дескриптор сокета.

Функция `close` освобождает структуры данных, выделенные для поддержки сокета. Для корректного завершения соединения следует использовать функцию `shutdown`.

Замечание. В Windows для закрытия сокета используется функция `closesocket`.

Пример: простейший TCP-клиент

Пример 40. *Простейший TCP-клиент устанавливает соединение, после чего передаёт и получает 1 байт данных.*

```
1  #include <sys/socket.h>
2  #include <netinet/in.h>
3  #include <arpa/inet.h>
4  #include <stdio.h>
5
6  int main()
7  {
8      struct sockaddr_in peer = {
9          AF_INET,
10         htons(6060),
11         { inet_addr("127.0.0.1") }
12     };
13
14     int s = socket(AF_INET, SOCK_STREAM, 0);
15     if (s < 0) {
16         fprintf(stderr, "socket␣вернула␣ошибку");
17         return 1;
18     }
19 ...
```

Пример: простейший TCP-клиент (продолжение)


```
19  ...
20      int res = connect(s, (struct sockaddr*)&peer,
21                        sizeof(peer));
22      if (res < 0) {
23          fprintf(stderr, "connect вернула ошибку");
24          return 1;
25      }
26
27      res = send(s, "A", 1, 0);
28      if (res <= 0) {
29          fprintf(stderr, "ошибка передачи данных");
30          return 1;
31      }
32
33      res = shutdown(s, 1);
34      if (res < 0) {
35          fprintf(stderr, "shutdown вернула ошибку");
36          return 1;
37      }
38  ...
```

Пример: простейший TCP-клиент (продолжение)


```
38  ...
39      char buf[1];
40      res = recv(s, buf, 1, 0);
41      if (res <= 0) {
42          fprintf(stderr, "ошибка␣получения␣данных");
43          return 1;
44      }
45
46      printf("%c\n", buf[0]);
47
48      res = close(s);
49      if (res < 0) {
50          fprintf(stderr, "close␣вернула␣ошибку");
51          return 1;
52      }
53      return 0;
54  }
```

Привязка адреса к сокету (bind)

Замечание. По умолчанию для сокета типа `SOCK_STREAM` или `SOCK_DGRAM` выбирается адрес с эфемерным портом.

Функция `bind` позволяет привязать к сокету нужный адрес. Она возвращает `0` в случае успеха и `-1` – в случае неудачи.

```
#include <sys/socket.h>
int bind(int s, const struct sockaddr *name,
         int namelen);
```

- `s` – дескриптор сокета;
- `name` – указатель на структуру, содержащую привязываемый адрес;
- `namelen` – размер структуры, содержащей адрес.

Замечание. Как правило, функция `bind` вызывается перед переключением сокета типа `SOCK_STREAM` в режим прослушивания для приёма входящих соединений. Тем не менее, привязка адреса может использоваться и перед вызовом функций `connect`, `recvfrom` или `sendto`.

Перевод сокета в режим прослушивания (listen)

Функция `listen` переводит сокет типа `SOCK_STREAM` в режим прослушивания для приёма входящих соединений. Она возвращает `0` в случае успеха и `-1` – в случае неудачи.

```
#include <sys/socket.h>
int listen(int s, int backlog);
```

- `s` – дескриптор сокета;
- `backlog` – длина очереди входящих соединений.

  Замечание. Если процесс не успевает принимать соединения с той скоростью, с какой приходят `SYN`-сегменты, то эти `SYN`-сегменты накапливаются в очереди указанной длины. При полностью заполненной очереди на новые входящие `SYN`-сегменты следует ответ с флагом `RST`, т.е. они отвергаются.
  Нормальным значением для параметра `backlog` является число 32, максимальным в большинстве современных операционных систем – 128.

Приём входящего соединения (accept)

Приём входящего соединения, ожидающего в очереди, осуществляется функцией `accept`. Она возвращает дескриптор нового сокета в случае успеха и `-1` – в случае неудачи.

```
#include <sys/socket.h>
int accept(int s, struct sockaddr *addr,
           int *addrlen);
```

- `s` – дескриптор сокета, находящегося в режиме прослушивания;
- `addr` – указатель на структуру, в которую `accept` поместит адрес удалённого процесса;
- `addrlen` – указатель на переменную с размером адреса (`accept` может переписать значение переменной).

Замечание. Новый сокет, возвращаемый функцией `accept`, предназначен для взаимодействия с удалённым процессом. При этом адрес, к которому он привязан, совпадает с адресом, к которому привязан соответствующий слушающий сокет.

Пример: простейший TCP-сервер

Пример 41. *Простейший TCP-сервер принимает соединение, после чего получает и передаёт 1 байт данных.*

```
1  #include <sys/socket.h>
2  #include <netinet/in.h>
3  #include <arpa/inet.h>
4  #include <stdio.h>
5
6  int main()
7  {
8      struct sockaddr_in addr = {
9          AF_INET,
10         htons(6060),
11         { inet_addr("127.0.0.1") }
12     };
13
14     int s = socket(AF_INET, SOCK_STREAM, 0);
15     if (s < 0) {
16         fprintf(stderr, "socket␣вернула␣ошибку");
17         return 1;
18     }
19 ...
```

Пример: простейший TCP-сервер (продолжение)


```
20      int res = bind(s, (struct sockaddr*)&addr,
21                  sizeof(addr));
22      if (res < 0) {
23          fprintf(stderr, "bind␣вернула␣ошибку");
24          return 1;
25      }
26
27      res = listen(s, 32);
28      if (res < 0) {
29          fprintf(stderr, "listen␣вернула␣ошибку");
30          return 1;
31      }
32
33      struct sockaddr_in client_addr;
34      int client_addr_size = sizeof(client_addr);
35      int s2 = accept(s,(struct sockaddr*)&client_addr,
36                      &client_addr_size);
37      if (s2 < 0) {
38          fprintf(stderr, "accept␣вернула␣ошибку");
39          return 1;
40      }
41  ...
```

Пример: простейший TCP-сервер (продолжение)


```
42      char buf[1];
43      res = recv(s2, buf, 1, 0);
44      if (res <= 0) {
45          fprintf(stderr, "ошибка␣получения␣данных");
46          return 1;
47      }
48      printf("%c␣from␣%s:%hu\n", buf[0],
49          inet_ntoa(client_addr.sin_addr),
50          ntohs(client_addr.sin_port));
51
52      res = send(s2, "B", 1, 0);
53      if (res <= 0) {
54          fprintf(stderr, "ошибка␣передачи␣данных");
55          return 1;
56      }
57
58      res = shutdown(s2, 1);
59      if (res < 0) {
60          fprintf(stderr, "shutdown␣вернула␣ошибку");
61          return 1;
62      }
63  ...
```

Пример: простейший TCP-сервер (продолжение)


```
64          res = close(s2);
65          if (res < 0) {
66              fprintf(stderr, "close␣вернула␣ошибку");
67              return 1;
68          }
69
70          res = close(s);
71          if (res < 0) {
72              fprintf(stderr, "close␣вернула␣ошибку");
73              return 1;
74          }
75
76          return 0;
77      }
```