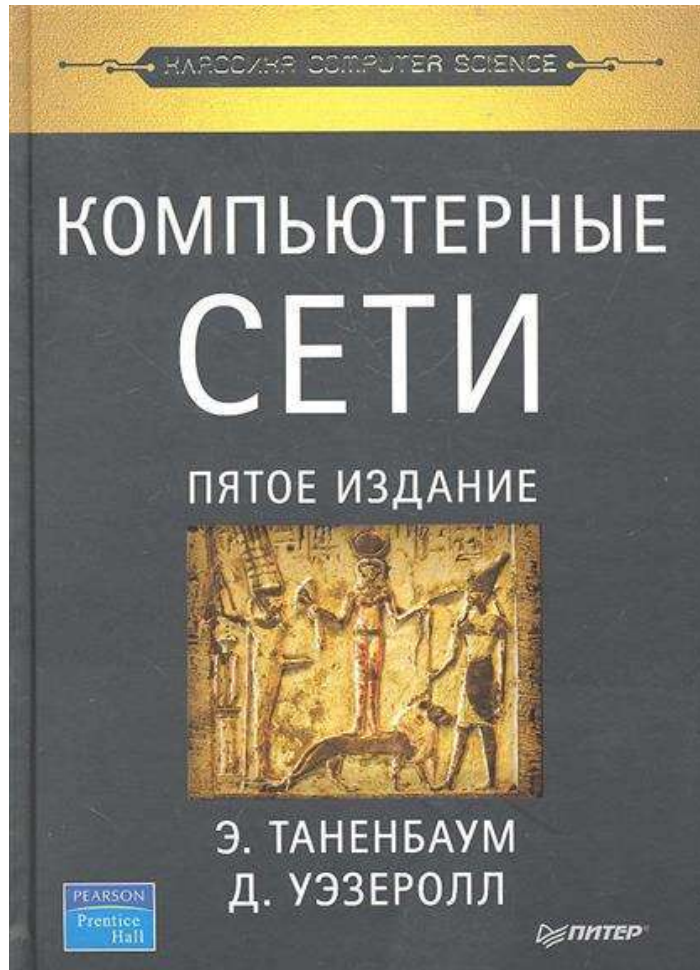


# Компьютерные сети

Курс читает Рогозин Н.О., кафедра ИУ-7

# Рекомендуемая литература




# Рекомендуемая литература

---

- ▶ **Столлинс В. Современные компьютерные сети** : пер. с англ. / Столлинс В. - 2-е изд. - СПб. : Питер, 2003. - 782 с. - (Классика computer science). - Библиогр.: с. 754-766. - ISBN 5-94723-327-4.
- ▶ **Стивенс У. Р. UNIX: разработка сетевых приложений** : пер. с англ. / Стивенс У. Р. ; пер. Колос А., Михайлова А. - СПб. : Питер, 2003. - 1085 с. - (Мастер-класс). - Библиогр.: с. 1027-1033. - ISBN 5-318-00535-7.
- ▶ **Уолтон Ш. Создание сетевых приложений в среде Linux = Linux Socket Programming : руководство разработчика** : пер. с англ. / Уолтон Ш. - М. : Вильямс, 2001. - 463 с. : ил. - ISBN 5-8459-0193-6.
- ▶ **Beej's Guide to Network Programming**  
(<http://beej.us/guide/bgnet/output/html/multipage/index.html>)





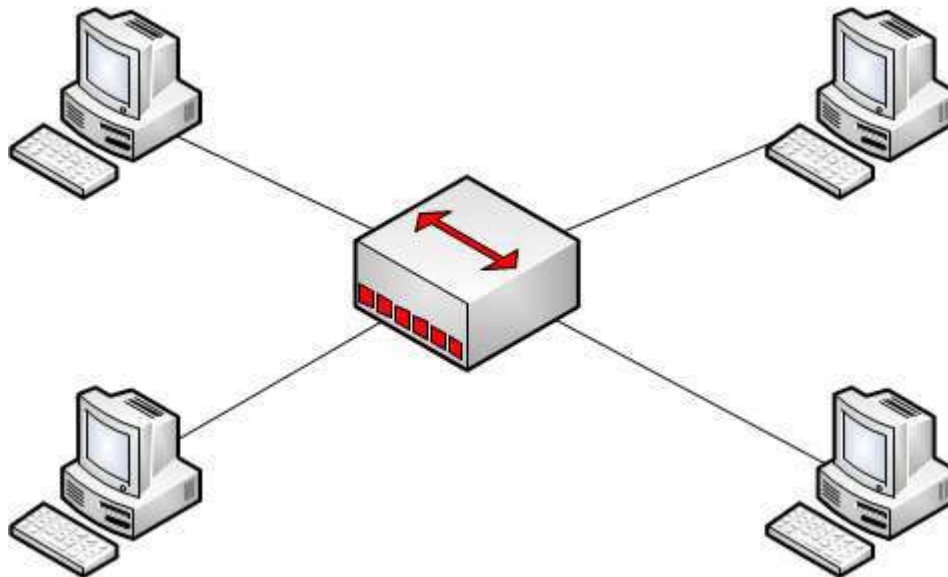
# Лекция I

## Классификация и обзор сетей

# Что такое компьютерная сеть?

---

- ▶ Совокупность компьютеров и других устройств, соединенных линиями связи и обменивающихся информацией между собой в соответствии с определенными правилами – протоколом.



# Назначение сети

---

- ▶ Предоставление конечным узлам возможности совместного использования ресурсов
- ▶ Ресурсы бывают трех типов: аппаратные, программные и информационные.
- ▶ Например, устройство печати (принтер) - это аппаратный ресурс. Емкости жестких дисков - тоже аппаратный ресурс. Когда все участники небольшой компьютерной сети пользуются одним общим принтером, это значит, что они разделяют общий аппаратный ресурс. То же можно сказать и о сети, имеющей один компьютер с увеличенной емкостью жесткого диска (файловый сервер), на котором все участники сети хранят свои архивы и результаты работы.



# Назначение сети

---

- ▶ Кроме аппаратных ресурсов компьютерные сети позволяют совместно использовать программные ресурсы. Так, например, для выполнения очень сложных и продолжительных расчетов можно подключиться к удаленной большой ЭВМ и отправить вычислительное задание на нее, а по окончании расчетов точно так же получить результат обратно.
- ▶ Данные, хранящиеся на удаленных компьютерах, образуют информационный ресурс. Роль этого ресурса сегодня видна наиболее ярко на примере Интернета, который воспринимается, прежде всего, как гигантская информационно-справочная система.
- ▶ При работе в компьютерной сети любого типа одновременно происходит совместное использование всех типов ресурсов.



# Протоколы

---

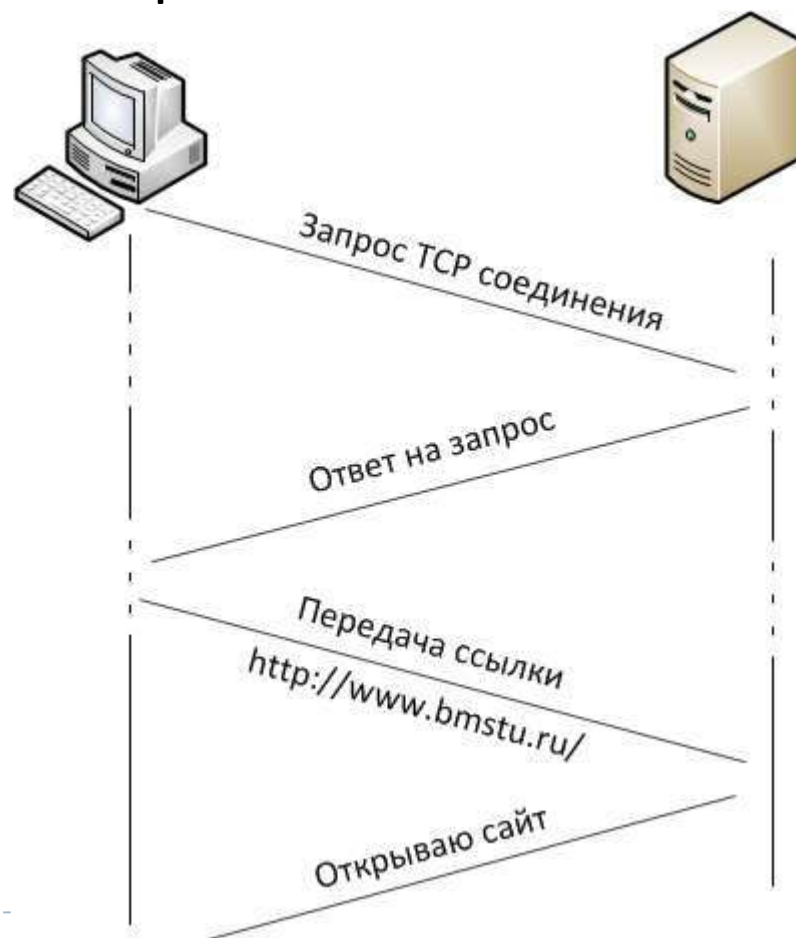
- ▶ Набор соглашений интерфейса *логического уровня*, которые определяют обмен данными между различными программами. Эти соглашения задают единообразный способ передачи сообщений и обработки ошибок при взаимодействии программного обеспечения разнесённой в пространстве аппаратуры, соединённой тем или иным интерфейсом.
- ▶ Сетевой протокол - набор правил и действий (очерёдности действий), позволяющий осуществлять соединение и обмен данными между двумя и более включёнными в сеть устройствами.





# Протоколы (cont.)

- ▶ Аналогия – человеческая беседа, проходящая по определенным правилам



# Сервер

---

- ▶ Специализированный компьютер и/или *специализированное оборудование* для выполнения на нём сервисного программного обеспечения (в том числе *серверов* тех или иных задач).



# Многотерминальные системы

---

- ▶ Компьютерные сети, называемые также сетями передачи данных, являются логическим результатом эволюции двух важнейших научно-технических отраслей современной цивилизации — компьютерных и телекоммуникационных технологий.
- ▶ В истоках компьютерного корня сетей - системы пакетной обработки данных. строились на базе мейнфрейма — мощного и надежного компьютера универсального назначения.
- ▶ Пользователи подготавливали перфокарты, содержащие данные и команды программ, и передавали их в ВЦ. Операторы вводили эти карты в компьютер, а распечатанные результаты пользователи получали обычно только на следующий день. Неверно набитая карта означала как минимум суточную задержку.



# История, ARPANET

---

- ▶ **ARPANET** (от англ. ***A**dvanced **R**esearch **P**rojects **A**gency **N**etwork*) — компьютерная сеть, созданная в 1969 году в США Агентством Министерства обороны США по перспективным исследованиям (DARPA) и явившаяся прототипом сети Интернет.



# История, ARPANET

---

- ▶ В августе 1962 г. Дж. К. Р. Ликлайдер пишет серию заметок, в которых обсуждает концепцию “Галактической сети”. В статье предсказывается появление глобальной взаимосвязанной сети компьютеров.
- ▶ Ликлайдер первым возглавил научно-исследовательскую компьютерную программу в агентстве DARPA, начиная с октября 1962 г.
- ▶ Работая в DARPA, он убедил своих последователей в DARPA Ивана Сазерленда, Боба Тейлора и ученого из MIT Лоренса Дж. Робертса в важности этой концепции сети.



# История, ARPANET

---

- ▶ Леонард Клейнрок в MIT опубликовал первую статью по теории пакетной коммутации в июле 1961 г. и первую книгу по данной теме в 1964 г. Клейнрок убедил Робертса в теоретической возможности связи с использованием пакетов вместо цепей, что стало важным шагом в области развития компьютерных сетей.
  - ▶ Другой важный шаг состоял в том, чтобы заставить компьютеры общаться друг с другом. Для изучения этого вопроса в 1965 г., работая вместе с Томасом Мерриллом, Робертс подключил компьютер TX-2, находящийся в штате Массачусетс, к компьютеру Q-32 в Калифорнии с использованием низкоскоростной телефонной линии.
  - ▶ В результате этого была создана первая (пусть и небольшая) широкомасштабная компьютерная сеть.
- 



# История, ARPANET

---

- ▶ В августе 1968 г. DARPA опубликовала заказ на разработку одного из главных компонентов, пакетных коммутаторов, которые назывались сопрягающими процессорами сообщений (IMP).
- ▶ В Калифорнийском Университете центр Network Measurement Center под руководством Клейнрока был выбран в качестве первого узла в сети ARPANET.
- ▶ В декабре 1970 г. Network Working Group (NWG), под руководством С. Крокера, завершила работу над созданием первоначального протокола связи между узлами сети ARPANET. Он назывался протоколом управления сетью (NCP).



# История, ARPANET

---

- ▶ Именно Крокер предложил формат **RFC** для сетевых стандартов и он же написал первый RFC.
- ▶ RFC – Request for Comments (“запрос на отзывы”).
- ▶ RFC 1 был опубликован 7 апреля 1969 г. и назывался «*Host Software*». Первые RFC распространялись в печатном виде на бумаге в виде обычных писем, но уже с декабря 1969 г., когда заработали первые сегменты ARPANET, документы начали распространяться в электронном виде.





# Сетевые приложения ARPANET

---

- ▶ **Электронная почта.** В 1971, Рэй Томлисон, из BBN переслал первый e-mail (RFC 524, RFC 561). К 1973, электронная почта составила 75 процентов трафика ARPANET.
- ▶ **Передача файлов.** К 1973 году, была определена и реализована спецификация для FTP (RFC 354) .
- ▶ **Сетевой Голосовой Протокол.** Спецификации были определены и реализованы в 1977 (RFC 741), однако из-за технических ограничений интернет-телефония в ARPANET так и не заработала в полную силу.



# История, создание Internet

---

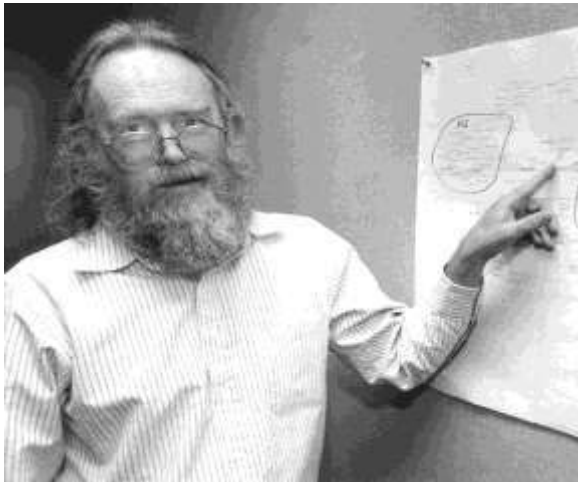
- ▶ В 1990 году сеть ARPANET прекратила своё существование, полностью проиграв конкуренцию NSFNet. В том же году было зафиксировано первое подключение к интернету по телефонной линии (т. н. «дозвóн», англ. *dialup access*).
- ▶ В 1991 году Всемирная паутина стала общедоступна в интернете, а в 1993 году появился знаменитый веб-браузер NCSA Mosaic.



# История, создание Internet

---

- ▶ Ведущую роль сыграли Джон Постел (автор стандартов множества ключевых протоколов, в т.ч. IP, ICMP, TCP, UDP, Telnet, FTP, DNS)
- ▶ И Тимоти Джон–Бернерс Ли, изобретатель URI, URL, HTTP, HTML, создатель Всемирной паутины.



# Классификация сетей

---

- ▶ По охвату (PAN, LAN, WAN и т.д.)
- ▶ В соотв. с технологическими признаками, обусловленными средой передачи (проводная и беспроводная)
- ▶ По топологии (полносвязная, шина, звезда и т.д.)
- ▶ По признаку первичности
- ▶ По способу коммутации



# PAN

---

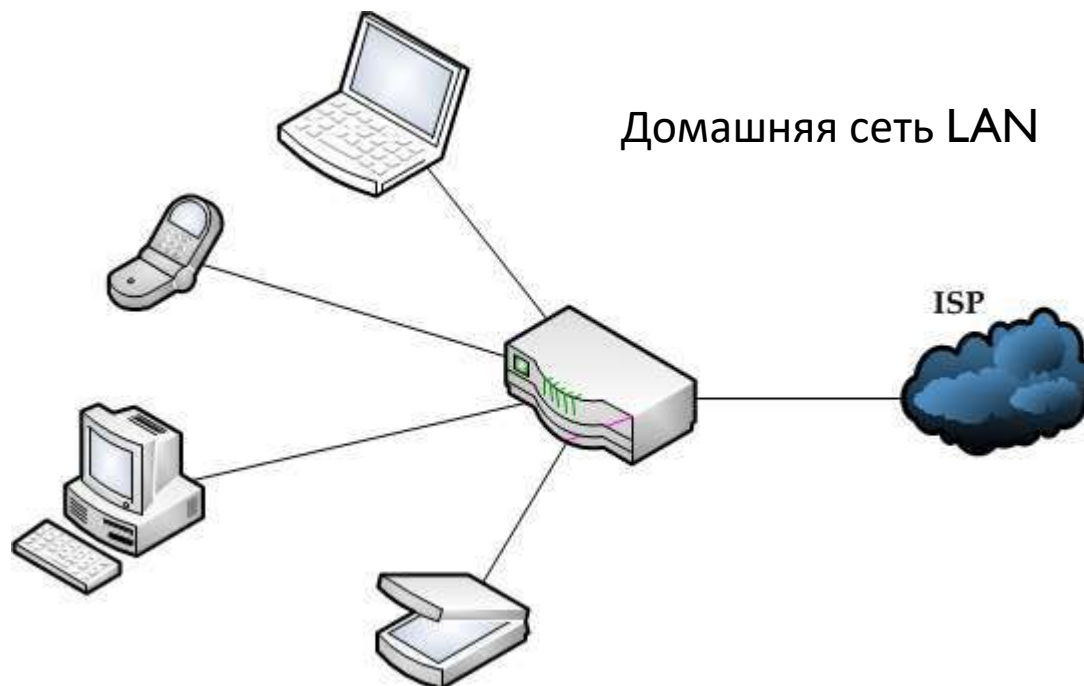
- ▶ Небольшой радиус (до 30 м)
- ▶ Малое число участников (до 8)
- ▶ Не критичность в отказоустойчивости
- ▶ Высокие требования к безопасности. Протоколы PAN должны обеспечивать разнообразные методы аутентификации устройств и шифрования данных в мобильной обстановке из-за частой смены окружения
- ▶ Основные технологии: Bluetooth, ZigBee и др.
- ▶ Предназначены для соединения устройств, принадлежащих, как правило, одному пользователю, на небольших расстояниях. Типичным примером PAN является беспроводное соединение компьютера с периферийными устройствами, такими как принтер, наушники, мышь, клавиатура и т. п. Мобильные телефоны также используют технологию PAN для соединения со своей периферией (чаще всего это наушники)



# LAN

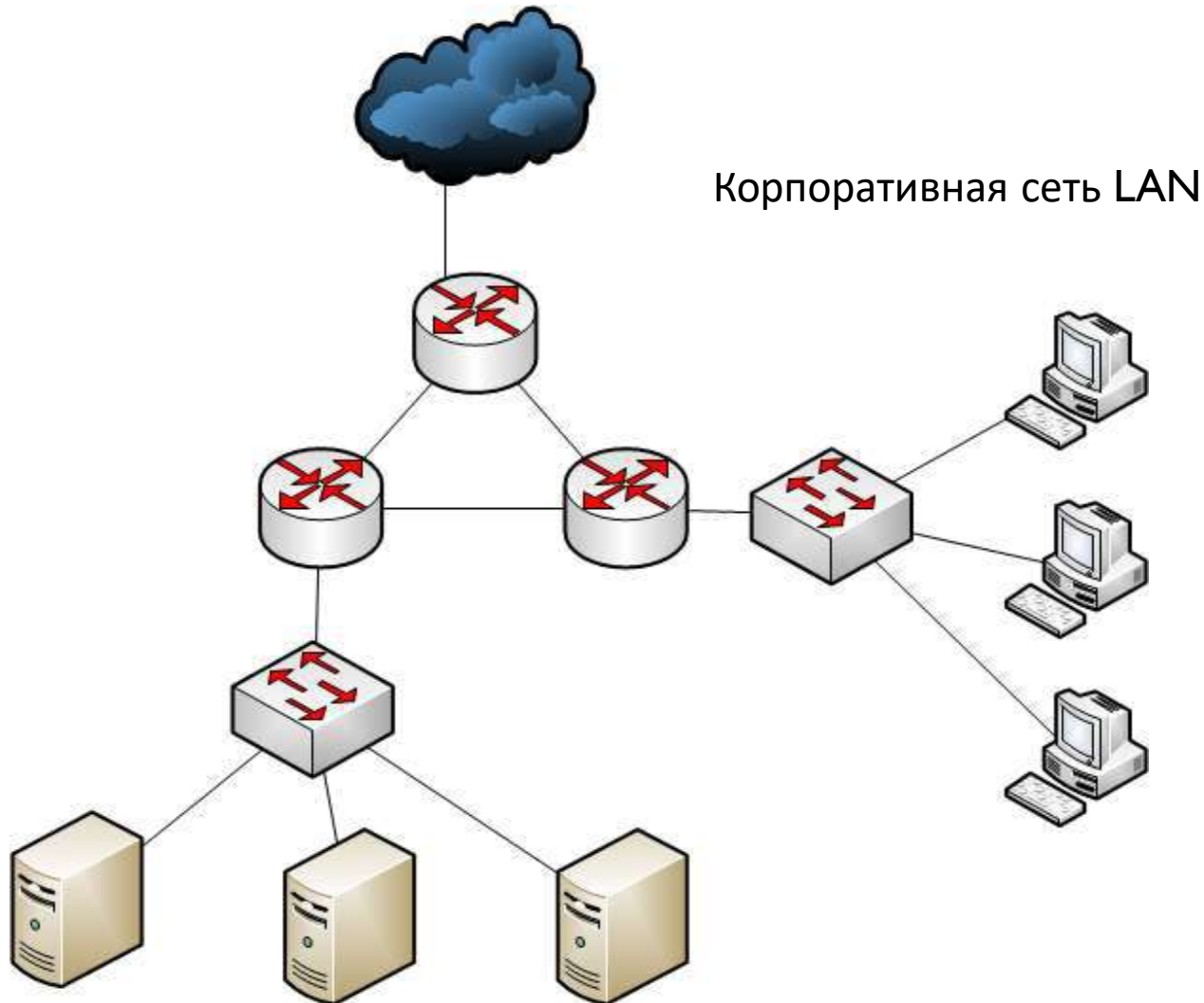
---

- ▶ Как небольшие домашние сети, так и крупные корпоративные
- ▶ Покрывают обычно небольшую территорию



# LAN (cont.)

---



# WAN

---

- ▶ Глобальная сеть, охватывающая большие расстояния
  - ▶ Соединяет несколько локальных сетей, географически удаленных друг от друга.
  - ▶ Крупнейшие примеры: Интернет и Фидонет, первый до сих пор остается самой распространенной сетью в мире, охватывая миллионы хостов
  - ▶ Основное отличие от локальных сетей – на физическом и канальном уровнях
  - ▶ На узлах локальной сети обычно используются службы доступа к файлам и принтерам, на узлах глобальной сети – маршрутизаторах и службы соответствующие: VPN, маршрутизация и т.п.
- 





# Сеть точка-точка

---

- ▶ Простейшая сеть, соединяет между собой только 2 компьютера
- ▶ Организуется для быстрой передачи данных, чаще всего когда другие способы сетевой коммуникации недоступны



# Клиент-серверная архитектура

---

- ▶ Сетевая архитектура, в которой устройства являются либо клиентами, либо серверами. Клиентом (front end) является запрашивающая машина (обычно ПК), сервером (back end) — машина, которая отвечает на запрос.
- ▶ **Сеть с выделенным сервером** (*Client/Server network*) — это локальная вычислительная сеть (LAN), в которой сетевые устройства централизованы и управляются одним или несколькими серверами. Индивидуальные рабочие станции или клиенты (такие, как ПК) должны обращаться к ресурсам сети через сервер(ы).



# Признак первичности

---

- ▶ Первичные сети - своего рода вспомогательные сети, которые нужны для того, чтобы гибко создавать постоянные физические двухточечные каналы для других компьютерных и телефонных сетей. В соответствии с семиуровневой моделью OSI первичные сети подобно простым кабелям выполняют функции физического уровня сетей. Однако в отличие от кабелей первичные сети включают дополнительное коммуникационное оборудование, которое путем соответствующего конфигурирования позволяет прокладывать новые физические каналы между конечными точками сети.
- ▶ Гибкая среда для создания физических каналов связи.
- ▶ Наложенные сети в этой классификации — это все остальные сети, которые предоставляют услуги конечным пользователям и строятся на основе каналов первичных сетей — «накладываются» поверх этих сетей. То есть и компьютерные, и телефонные, и телевизионные сети являются наложенными.



# Топология сети

---

- ▶ Конфигурация графа, вершинам которого соответствуют конечные узлы сети (например, компьютеры) и *коммуникационное оборудование* (например, маршрутизаторы), а *ребрам* — электрические и информационные связи между ними.
- ▶ Число возможных конфигураций резко возрастает при увеличении числа связываемых устройств. Так, если три компьютера мы можем связать двумя способами, то для четырех компьютеров можно предложить уже шесть топологически различных конфигураций (при условии неразличимости компьютеров).



# Топология сети

---

- ▶ От выбора топологии связей зависят многие характеристики сети. Например, наличие между узлами нескольких путей повышает надежность сети и делает возможной балансировку загрузки отдельных каналов.
- ▶ Простота присоединения новых узлов, свойственная некоторым топологиям, делает сеть легко расширяемой. Экономические соображения часто приводят к выбору топологий, для которых характерна минимальная суммарная длина линий связи.



# Топология сети

---

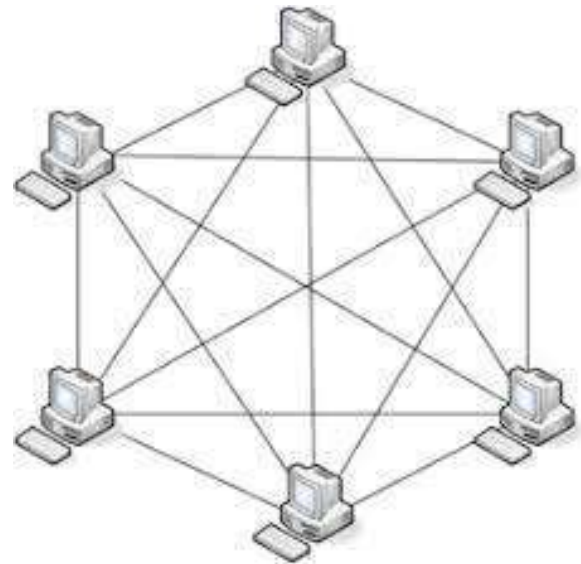
- ▶ **физическая** — описывает реальное расположение и связи между узлами сети.
- ▶ **логическая** — описывает хождение сигнала в рамках физической топологии.
- ▶ **информационная** — описывает направление потоков информации, передаваемых по сети.
- ▶ **управления обменом** — это принцип передачи права на пользование сетью.



# Топология “Полносвязная”

---

- ▶ Каждый компьютер связан со всеми остальными.  
Громоздкий и неэффективный вариант, т.к. каждый компьютер должен иметь большое кол-во коммуникационных портов.



# Топология “Полносвязная”

---

- ▶ В крупных сетях применяются редко, так как для связи  $N$  узлов требуется  $N(N-1)/2$  физических дуплексных линий связи, т.е. имеет место квадратичная зависимость.
- ▶ Чаще этот вид топологии используется в многомашинных комплексах или в сетях, объединяющих небольшое количество компьютеров.

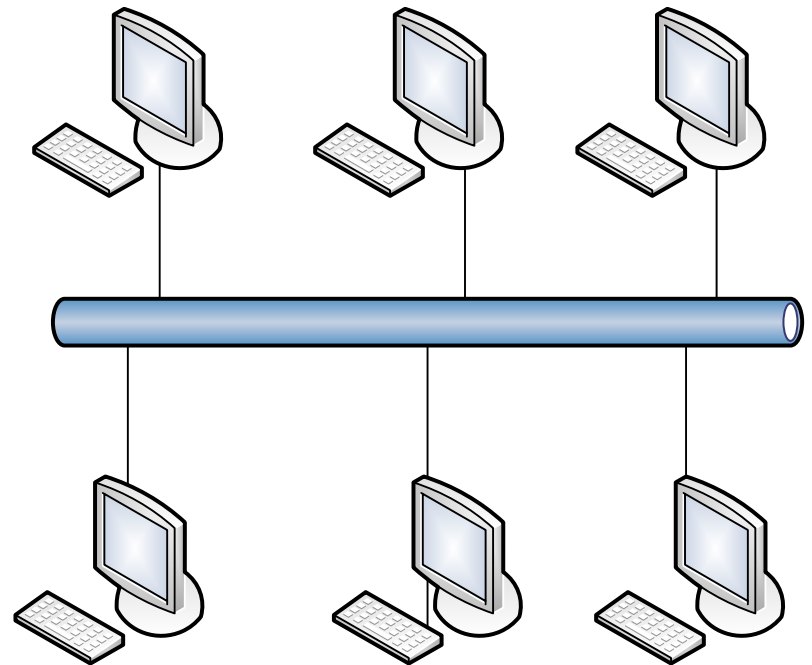




# Топология “Общая шина”

---

- ▶ Компьютеры подключаются к одному коаксиальному кабелю. Данные от передающего узла сети передаются по шине в обе стороны, отражаясь от оконечных терминаторов.
- ▶ Терминаторы предотвращают отражение сигналов, т.е. используются для гашения сигналов, которые достигают концов канала передачи данных.



# Топология “Общая шина”

---

## **Преимущества сетей шинной топологии:**

- ▶ отказ одного из узлов не влияет на работу сети в целом;
- ▶ сеть легко настраивать и конфигурировать;
- ▶ сеть устойчива к неисправностям отдельных узлов.

## **Недостатки сетей шинной топологии:**

- ▶ разрыв кабеля может повлиять на работу всей сети;
- ▶ ограниченная длина кабеля и количество рабочих станций;
- ▶ трудно определить дефекты соединений.



# Топология “Ячеистая”

---

- ▶ Получается из полносвязной путем удаления некоторых связей. Непосредственно связываются только те компьютеры, между которыми происходит интенсивный обмен данными. Даная топология характерна для глобальных сетей



# Топология “Ячеистая”

---

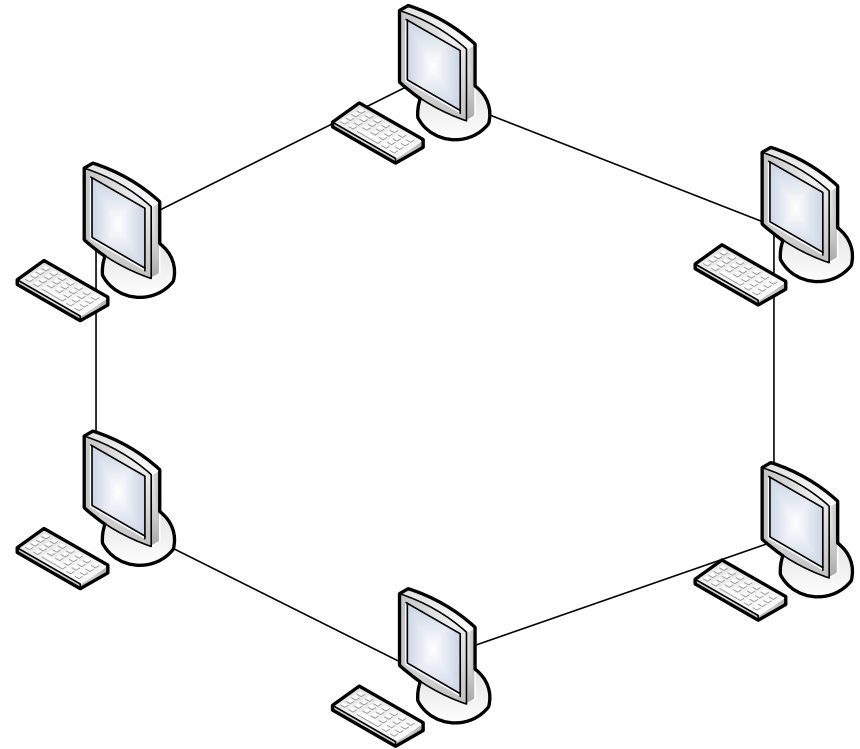
- ▶ Характеризуется высокой отказоустойчивостью, сложностью настройки и переизбыточным расходом кабеля.
- ▶ Каждый компьютер имеет множество возможных путей соединения с другими компьютерами. Обрыв кабеля не приведёт к потере соединения между двумя компьютерами.



# Топология “Кольцо”

---

- ▶ Данные передаются по кольцу от одного компьютера к другому, если компьютер распознает данные как свои, он копирует их себе во внутренний буфер.
- ▶ Данные в кольце всегда движутся в одном и том же направлении.



# Топология “Кольцо”

---

- ▶ Каждый компьютер ретранслирует (возобновляет) сигнал, то есть выступает в роли повторителя, потому затухание сигнала во всем кольце не имеет никакого значения, важно только затухание между соседними компьютерами кольца
- ▶ Четко выделенного центра в этом случае нет, все компьютеры могут быть одинаковыми.
- ▶ Однако достаточно часто в кольце выделяется специальный абонент, который управляет обменом или контролирует обмен. Понятно, что наличие такого управляющего абонента снижает надежность сети



# Топология “Кольцо”

---

## Достоинства:

- ▶ Простота установки;
- ▶ Практически полное отсутствие дополнительного оборудования;
- ▶ Возможность устойчивой работы без существенного падения скорости передачи данных при интенсивной загрузке сети, поскольку использование маркера исключает возможность возникновения коллизий.

## Недостатки:

- ▶ Выход из строя одной рабочей станции, и другие неполадки (обрыв кабеля), отражаются на работоспособности всей сети;
  - ▶ Сложность конфигурирования и настройки;
  - ▶ Сложность поиска неисправностей.
  - ▶ Необходимость иметь две сетевые платы, на каждой рабочей станции.
- 



# Топология “Кольцо”

---

- ▶ Подключить к сети новое устройство достаточно легко, хотя и требует обязательной остановки работы всей сети на время подключения.
- ▶ Как и в случае топологии «шина», максимальное количество абонентов в кольце может быть достаточно большое (1000 и больше).
- ▶ Кольцевая топология обычно является самой стойкой к перегрузкам, она обеспечивает уверенную работу с самыми большими потоками переданной по сети информации, потому что в ней, как правило, нет конфликтов (в отличие от шины), а также отсутствует центральный абонент (в отличие от звезды).
- ▶ В чистом виде топология редко применяется, на практике применяются различные ее модификации

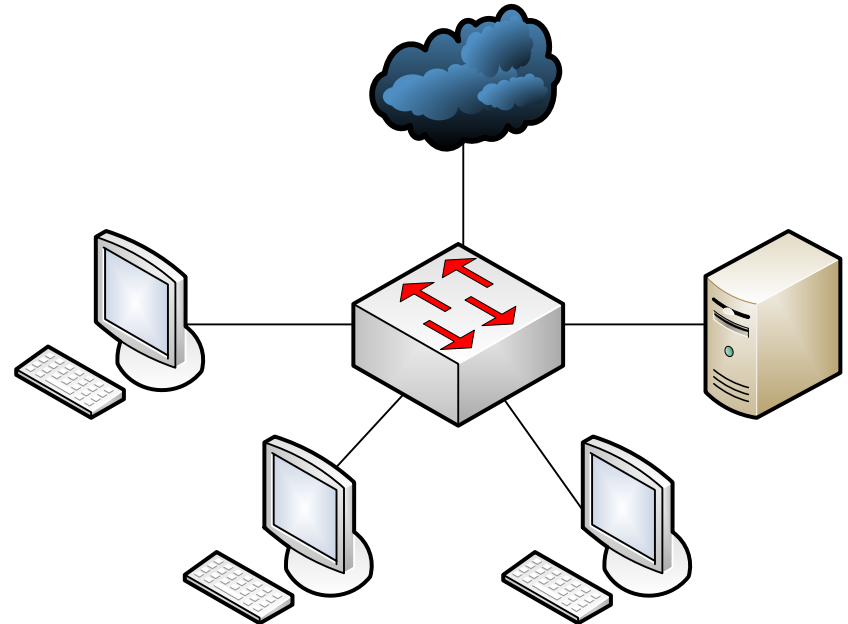




# Топология “Звезда”

---

- ▶ Каждый компьютер отдельным кабелем подключается к общему устройству – концентратору или коммутатору.
- ▶ При этом они образуют единый физический сегмент, способный работать автономно.



# Топология “Звезда”

---

## Достоинства:

- ▶ выход из строя одной рабочей станции не отражается на работе всей сети в целом;
- ▶ хорошая масштабируемость сети;
- ▶ лёгкий поиск неисправностей и обрывов в сети;
- ▶ высокая производительность сети (при условии правильного проектирования);
- ▶ гибкие возможности администрирования.

## Недостатки:

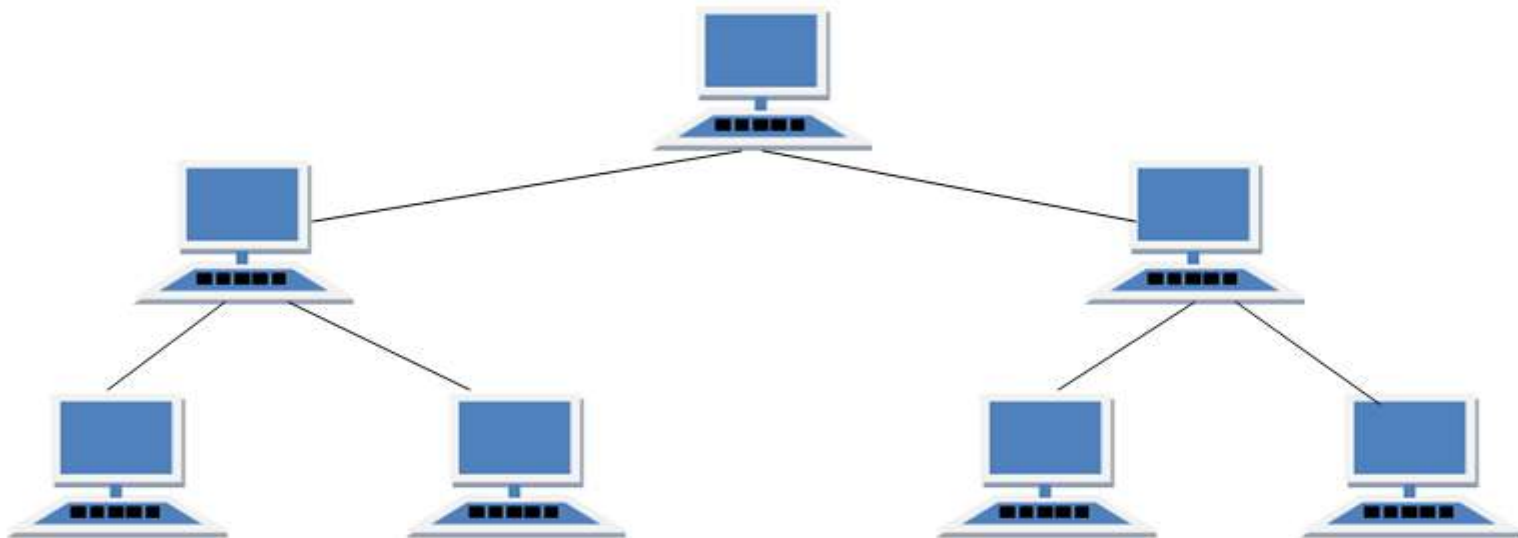
- ▶ выход из строя центрального концентратора обернётся неработоспособностью сети (или сегмента сети) в целом;
- ▶ для прокладки сети зачастую требуется больше кабеля, чем для большинства других топологий;
- ▶ конечное число рабочих станций в сети (или сегменте сети) ограничено количеством портов в центральном концентраторе.



# Топология “Иерархическая Звезда”

---

- ▶ Также “Древовидная” топология, “Снежинка” – топология типа звезды, но используется несколько концентраторов/коммутаторов, иерархически соединенных между собой связями типа звезда.
- ▶ Самый распространенный способ связей как в локальных сетях, так и в глобальных.



# Гибридная топология

---

- ▶ В звездно-шинной (star-bus) топологии используется комбинация шины и пассивной звезды. В этом случае к концентратору подключаются как отдельные компьютеры, так и целые шинные сегменты, то есть на самом деле реализуется физическая топология «шина», включающая все компьютеры сети. В данной топологии может использоваться и несколько концентраторов, соединенных между собой и образующих так называемую магистральную, опорную шину. К каждому из концентраторов при этом подключаются отдельные компьютеры или шинные сегменты.
- ▶ Таким образом, пользователь получает возможность гибко комбинировать преимущества шинной и звездной топологий, а также легко изменять количество компьютеров, подключенных к сети.



# Гибридная топология

---

- ▶ В случае звездно-кольцевой (star-ring) топологии в кольцо объединяются не сами компьютеры, а специальные концентраторы, к которым в свою очередь подключаются компьютеры с помощью звездообразных двойных линий связи.
- ▶ В действительности все компьютеры сети включаются в замкнутое кольцо, так как внутри концентраторов все линии связи образуют замкнутый контур. Данная топология позволяет комбинировать преимущества звездной и кольцевой топологий.



# Сетевая модель и стек протоколов

Курс читает Рогозин Николай Олегович, каф. ИУ-7

# Модель OSI

- Эталонная модель OSI, иногда называемая стеком OSI представляет собой 7-уровневую сетевую иерархию разработанную Международной организацией по стандартам (International Standardization Organization - ISO).
- Содержит в себе по сути 2 различных модели:
  - **горизонтальную модель** на базе протоколов, обеспечивающую механизм взаимодействия программ и процессов на различных машинах
  - **вертикальную модель** на основе услуг, обеспечиваемых соседними уровнями друг другу на одной машине

# Модель OSI

- Каждый уровень компьютера-отправителя взаимодействует с таким же уровнем компьютера-получателя, как будто он связан напрямую. Такая связь называется логической или виртуальной связью. В действительности взаимодействие осуществляется между смежными уровнями одного компьютера.
- В горизонтальной модели двум программам требуется общий протокол для обмена данными.
- В вертикальной модели соседние уровни обмениваются данными с использованием интерфейсов прикладных программ API (Application Programming Interface).



# Развитие TCP

- Появился впервые в 1973
- В 1978 разделился на TCP и IP
- В 1983 заменил NCP (Network Control Protocol) для ARPANET
- Основной вклад принадлежит университету Беркли

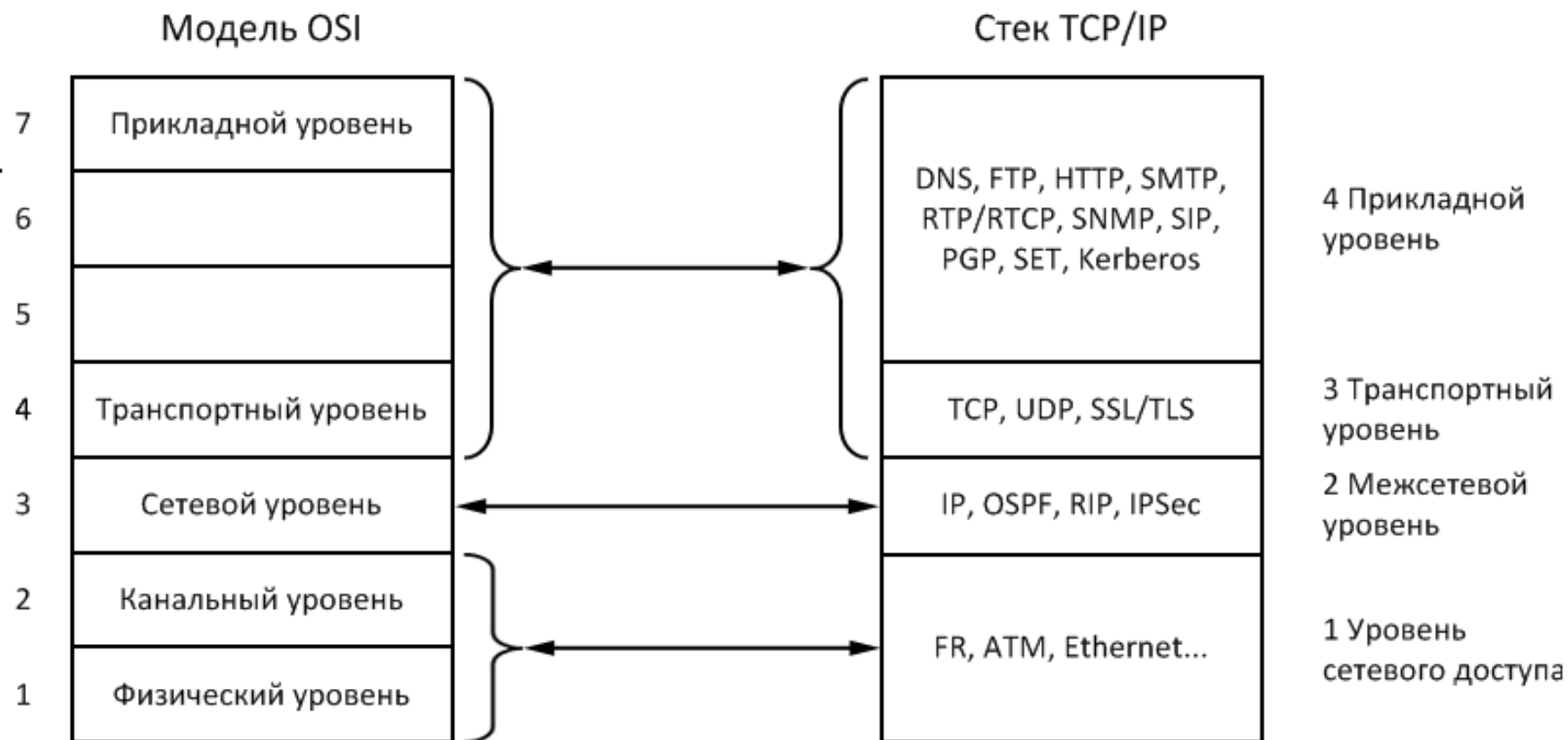
# TCP/IP

- Наиболее завершённый стандартный и в то же время популярный стек сетевых протоколов, имеющий многолетнюю историю.
- Почти все большие сети передают основную часть своего трафика с помощью протокола TCP/IP.
- Это метод получения доступа к сети Internet.
- Стек служит основой для создания intranet- корпоративной сети, использующей транспортные услуги Internet и гипертекстовую технологию WWW, разработанную в Internet.
- Все современные операционные системы поддерживают стек TCP/IP.
- Это гибкая технология для соединения разнородных систем как на уровне транспортных подсистем, так и на уровне прикладных сервисов.
- Это устойчивая масштабируемая межплатформенная среда для приложений клиент-сервер.

# Преимущества TCP/IP

- • **Независимость от сетевой технологии.** Стек протоколов TCP/IP не зависит от оборудования конечных пользователей, так как он только определяет элемент передачи - дейтаграмму - и описывает способ ее движения по сети.
- • **Всеобщая связанность.** Стек позволяет любой паре компьютеров, которые его поддерживают, взаимодействовать друг с другом. Каждому компьютеру назначается логический адрес, а каждая передаваемая дейтаграмма содержит логические адреса отправителя и получателя. Промежуточные маршрутизаторы используют адрес получателя для принятия решения о маршрутизации.
- • **Межконцевые подтверждения.** Протоколы стека TCP/IP обеспечивают подтверждение правильности прохождения информации при обмене между отправителем и получателем.
- • **Стандартные прикладные протоколы.** Протоколы TCP/IP включают в свой состав средства для поддержки наиболее часто встречающихся приложений, таких как электронная почта, передача файлов, удаленный доступ и т.д.

# Стек TCP/IP



# PDU

Уровень	Единица данных
Прикладной уровень ( I )	Поток
Основной уровень ( II )	Дейтаграмма(UDP)/ сегмент (TCP)
Межсетевой ( III )	Пакет или IP-дейтаграмма (IP)
Сетевых интерфейсов ( IV )	Кадр (фрейм)

# Взаимодействие стеков протоколов

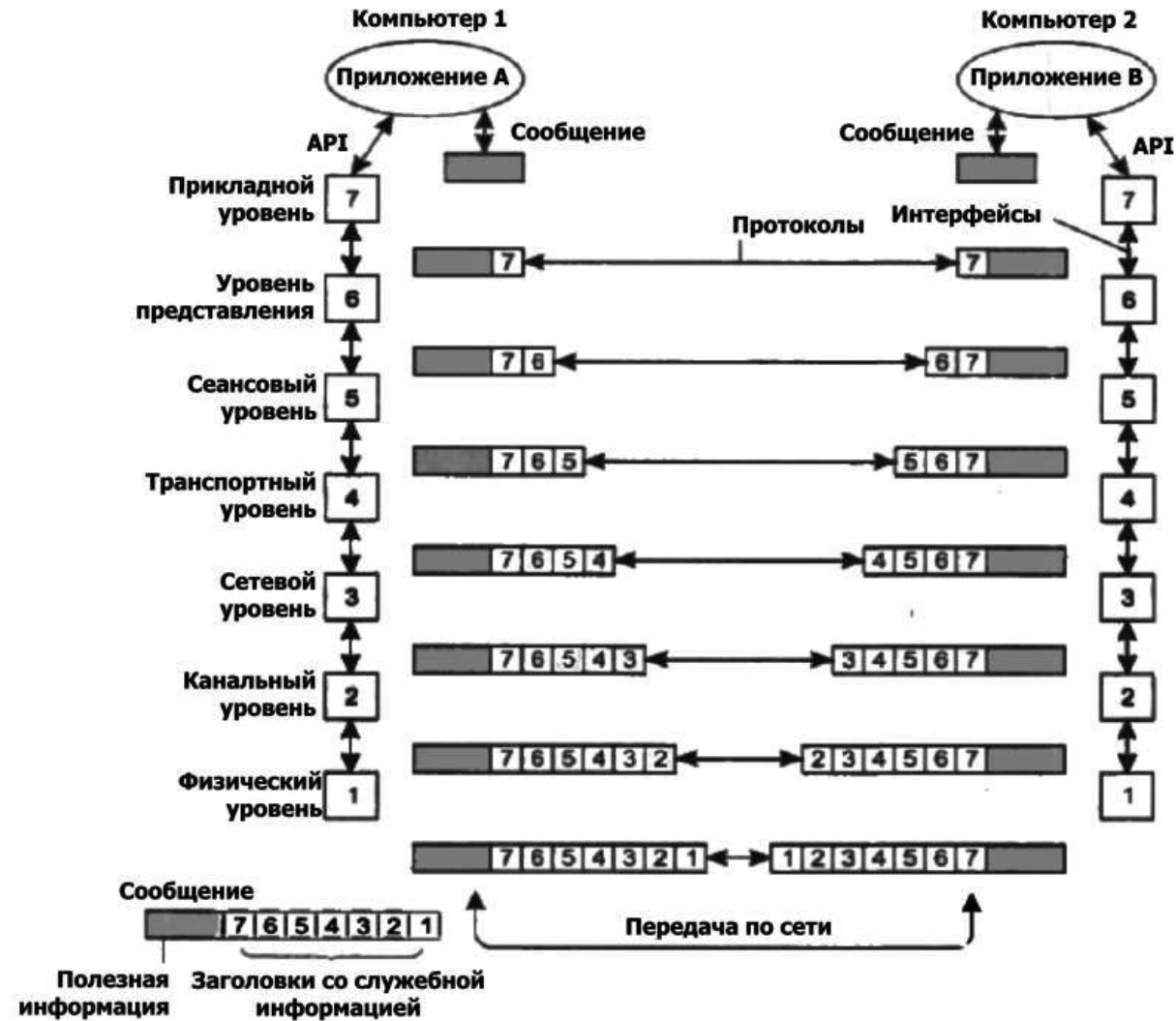


Рис. 4.6 Модель взаимодействия открытых систем ISO/OSI

# Действия модели OSI

- взаимодействие прикладных процессов;
- формы представления данных;
- единообразное хранение данных;
- управление сетевыми ресурсами;
- безопасность данных и защите информации;
- диагностика программ и технических средств.

# Прикладной уровень

- Обеспечивает прикладным процессам средства доступа к области взаимодействия
- Набор разнообразных протоколов, с помощью которых пользователи сети получают доступ к разделяемым ресурсам, таким как файлы, принтеры или гипертекстовые Web-страницы, а также организуют свою совместную работу, например с помощью протокола электронной почты.
- Одна из основных задач этого уровня – определить, как следует обрабатывать запрос прикладной программы, другими словами, какой вид должен принять данный запрос.



# Функции прикладного уровня

- Передача файлов, управление заданиями, управление системой и т. д;
- Идентификация пользователей по их паролям, адресам, электронным подписям;
- Определение функционирующих абонентов и возможности доступа к новым прикладным процессам;
- Определение достаточности имеющихся ресурсов;
- Организация запросов на соединение с другими прикладными процессами;
- Передача заявок представительскому уровню на необходимые методы описания информации;

# Функции прикладного уровня

- Выбор процедур планируемого диалога процессов;
- Управление данными, которыми обмениваются прикладные процессы и синхронизация взаимодействия прикладных процессов;
- Определение качества обслуживания (время доставки блоков данных, допустимой частоты ошибок);
- Соглашение об исправлении ошибок и определении достоверности данных;
- Согласование ограничений, накладываемых на синтаксис (наборы символов, структура данных).

# Протоколы прикладного уровня

- FTP (File Transfer Protocol) протокол передачи файлов;
- TFTP (Trivial File Transfer Protocol) простейший протокол пересылки файлов;
- X.400 электронная почта;
- Telnet работа с удаленным терминалом;
- SMTP (Simple Mail Transfer Protocol) простой протокол почтового обмена;
- CMIP (Common Management Information Protocol) общий протокол управления информацией;
- SLIP (Serial Line IP) IP для последовательных линий. Протокол последовательной посимвольной передачи данных;
- SNMP (Simple Network Management Protocol) простой протокол сетевого управления;
- FTAM (File Transfer, Access, and Management) протокол передачи, доступа и управления файлами.

# Уровень представления

- Обеспечивает то, что информация, передаваемая прикладным уровнем, будет понятна прикладному уровню в другой системе.
- В случаях необходимости уровень представления в момент передачи информации выполняет преобразование форматов данных в некоторый общий формат представления, а в момент приема, соответственно, выполняет обратное преобразование. Таким образом, прикладные уровни могут преодолеть, например, синтаксические различия в представлении данных.

# Уровень представления

- В основе система ASN.1, служащая для описания структуры файлов, а также позволяющая решить проблему шифрования данных.
- На этом уровне может выполняться шифрование и дешифрование данных, благодаря которым секретность обмена данными обеспечивается сразу для всех прикладных сервисов.
- Примером такого протокола является протокол Secure Socket Layer (SSL), который обеспечивает секретный обмен сообщениями для протоколов прикладного уровня стека TCP/IP.
- Этот уровень обеспечивает преобразование данных (кодирование, компрессия и т.п.) прикладного уровня в поток информации для транспортного уровня.

# Функции уровня представления

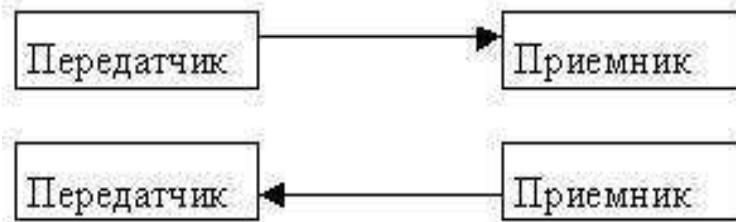
- Генерация запросов на установление сеансов взаимодействия прикладных процессов.
- Согласование представления данных между прикладными процессами.
- Реализация форм представления данных.
- Представление графического материала (чертежей, рисунков, схем).
- Засекречивание данных.
- Передача запросов на прекращение сеансов.

# Сеансовый уровень

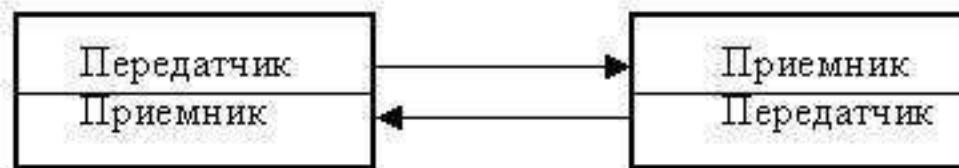
- Определяет процедуру проведения сеансов между пользователями или прикладными процессами.
- Обеспечивает управление диалогом для того, чтобы фиксировать, какая из сторон является активной в настоящий момент, а также предоставляет средства синхронизации.
- Последние позволяют вставлять контрольные точки в длинные передачи, чтобы в случае отказа можно было вернуться назад к последней контрольной точке, вместо того чтобы начинать все сначала.

# Режимы сеансового уровня

- На сеансовом уровне определяется, какой будет передача между двумя прикладными процессами:
  - полудуплексной (процессы будут передавать и принимать данные по очереди);



- дуплексной (процессы будут передавать данные, и принимать их одновременно).





# Функции сеансового уровня

- Установление и завершение на сеансовом уровне соединения между взаимодействующими системами.
- Выполнение нормального и срочного обмена данными между прикладными процессами.
- Управление взаимодействием прикладных процессов.
- Синхронизация сеансовых соединений.
- Извещение прикладных процессов об исключительных ситуациях.
- Установление в прикладном процессе меток, позволяющих после отказа либо ошибки восстановить его выполнение от ближайшей метки.
- Прерывание в нужных случаях прикладного процесса и его корректное возобновление.
- Прекращение сеанса без потери данных.
- Передача особых сообщений о ходе проведения сеанса.

# Транспортный уровень

- предназначен для передачи пакетов через коммуникационную сеть. На транспортном уровне пакеты разбиваются на блоки.
- Работа транспортного уровня заключается в том, чтобы обеспечить приложениям или верхним уровням модели (прикладному и сеансовому) передачу данных с той степенью надежности, которая им требуется, в связи с возможной потерей/искажением данных
- определяет адресацию физических устройств (систем, их частей) в сети. Этот уровень гарантирует доставку блоков информации адресатам и управляет этой доставкой. Его главной задачей является обеспечение эффективных, удобных и надежных форм передачи информации между системами.

# Основные протоколы транспортного уровня

- TCP (Transmission Control Protocol) протокол управления передачей стека TCP/IP;
- UDP (User Datagram Protocol) пользовательский протокол дейтаграмм стека TCP/IP;
- NCP (NetWare Core Protocol) базовый протокол сетей NetWare;
- SPX (Sequenced Packet eXchange) упорядоченный обмен пакетами стека Novell;
- TP4 (Transmission Protocol) – протокол передачи класса 4.

# Протокол TCP (Установление соединения)



# Протокол UDP (без установления соединения)



# Функции транспортного уровня

- Управление передачей по сети и обеспечение целостности блоков данных.
- Обнаружение ошибок, частичная их ликвидация и сообщение о неисправленных ошибках.
- Восстановление передачи после отказов и неисправностей.
- Укрупнение или разделение блоков данных.
- Предоставление приоритетов при передаче блоков (нормальная или срочная).
- Подтверждение передачи.
- Ликвидация блоков при тупиковых ситуациях в сети.

# Сетевой уровень

- Устанавливает связь в вычислительной сети между двумя системами и обеспечивает прокладку виртуальных каналов между ними
- Сообщает транспортному уровню о появляющихся ошибках.
- Отвечает за их адресацию и доставку сообщений – “пакетов”

# Сетевой уровень

- Прокладка наилучшего пути для передачи данных называется маршрутизацией, и ее решение является главной задачей сетевого уровня.
- Проблема осложняется тем, что самый короткий путь не всегда самый лучший. Часто критерием при выборе маршрута является время передачи данных по этому маршруту; оно зависит от пропускной способности каналов связи и интенсивности трафика, которая может изменяться с течением времени.
- Внутри сети доставка данных регулируется канальным уровнем, связанным с жестким ограничением по использованию в определенной топологии, а вот доставкой данных между сетями занимается сетевой уровень.
- При организации доставки пакетов на сетевом уровне используется понятие номер сети. В этом случае адрес получателя состоит из номера сети и номера компьютера в этой сети.



# Протоколы сетевого уровня

- IP (Internet Protocol) протокол Internet, сетевой протокол стека TCP/IP, который предоставляет адресную и маршрутную информацию;
- IPX (Internetwork Packet Exchange) протокол межсетевого обмена пакетами, предназначенный для адресации и маршрутизации пакетов в сетях Novell;
- X.25 международный стандарт для глобальных коммуникаций с коммутацией пакетов (частично этот протокол реализован на уровне 2);
- CLNP (Connection Less Network Protocol) сетевой протокол без организации соединений.

# Функции сетевого уровня

- Создание сетевых соединений и идентификация их портов.
- Обнаружение и исправление ошибок, возникающих при передаче через коммуникационную сеть.
- Управление потоками пакетов.
- Организация (упорядочение) последовательностей пакетов.
- Маршрутизация и коммутация.
- Сегментирование и объединение пакетов.

# Канальный уровень

- предназначен для обеспечения взаимодействия сетей на физическом уровне (в частности проверка доступности среды передачи) и реализация механизмов контроля за ошибками, которые могут возникнуть.
- Упаковывает данные в битах в кадры, проверяет их на целостность и, если нужно, исправляет ошибки (формирует повторный запрос поврежденного кадра) и отправляет на сетевой уровень
- обеспечивает корректность передачи каждого кадра, помещая специальную последовательность бит, в начало и конец каждого кадра, чтобы отметить его, а также вычисляет контрольную сумму, суммируя все байты кадра определенным способом и добавляя контрольную сумму к кадру.

# Протоколы канального уровня

- HDLC (High Level Data Link Control) протокол управления каналом передачи данных высокого уровня, для последовательных соединений;
- IEEE 802.2 LLC (тип I и тип II) обеспечивают MAC для сред 802.x;
- Ethernet сетевая технология по стандарту IEEE 802.3 для сетей, использующая шинную топологию и коллективный доступ с прослушиванием несущей частоты и обнаружением конфликтов;
- Token ring сетевая технология по стандарту IEEE 802.5, использующая кольцевую топологию и метод доступа к кольцу с передачей маркера;
- FDDI (Fiber Distributed Date Interface) сетевая технология по стандарту IEEE 802.6, использующая оптоволоконный носитель;
- X.25 международный стандарт для глобальных коммуникаций с коммутацией пакетов;
- Frame relay сеть, организованная из технологий X25 и ISDN.

# Канальный уровень, cont.

- MAC уровень – подуровень канального (второго) уровня модели OSI, согласно стандартам IEEE 802.

Выступает в качестве интерфейса между подуровнем LLC и физическим (первым) уровнем.

- LLC уровень - подуровень управления логической связью — по стандарту IEEE 802 — верхний подуровень канального уровня модели OSI, осуществляет:
  - 1) управление передачей данных;
  - 2) обеспечивает проверку и правильность передачи информации по соединению.

# Функции канального уровня

- Организация (установление, управление, расторжение) канальных соединений и идентификация их портов.
- Организация и передача кадров.
- Обнаружение и исправление ошибок.
- Управление потоками данных.
- Обеспечение прозрачности логических каналов (передачи по ним данных, закодированных любым способом).

# Физический уровень

- определяет метод передачи данных, представленных в двоичном виде, от одного устройства (компьютера) к другому.
- Задаёт то, как кабель присоединен к адаптеру, характеристики оборудования, топологию и дизайн сети.
- Передаёт данные в виде потока бит информации.

# Свойства физ. уровня

- Тип кабелей и разъемов
- Разводку контактов в разъемах
- Схему кодирования сигналов для значений 0 и 1



# Функции физ. уровня

- Установление и разъединение физических соединений.
- Передача сигналов в последовательном коде и прием.
- Прослушивание, в нужных случаях, каналов.
- Идентификация каналов.
- Оповещение о появлении неисправностей и отказов.

# Спецификации физ. уровня

- EIA-RS-232-C, CCITT V.24/V.28 - механические/электрические характеристики несбалансированного последовательного интерфейса.
- EIA-RS-422/449, CCITT V.10 - механические, электрические и оптические характеристики сбалансированного последовательного интерфейса.
- IEEE 802.3 -- Ethernet
- IEEE 802.5 -- Token ring

# IEEE 802

- IEEE (Institute of Electrical and Electronics Engineers) является профессиональной организацией (США), определяющей стандарты, связанные с сетями и другими аспектами электронных коммуникаций.
- Группа IEEE 802.X содержит описание сетевых спецификаций и содержит стандарты, рекомендации и информационные документы для сетей и телекоммуникаций.

# Стандарты IEEE 802.X

- **802.1** - задает стандарты управления сетью на MAC-уровне, включая алгоритм Spanning Tree. Этот алгоритм используется для обеспечения единственности пути (отсутствия петель) в многосвязных сетях на основе мостов и коммутаторов с возможностью его замены альтернативным путем в случае выхода из строя. Документы также содержат спецификации сетевого управления и межсетевого взаимодействия.
- **802.2** - определяет функционирование подуровня LLC на канальном уровне модели OSI. LLC обеспечивает интерфейс между методами доступа к среде и сетевым уровнем. Прозрачные для вышележащих уровней функции LLC включают кадрирование, адресацию, контроль ошибок. Этот подуровень используется в спецификации 802.3 Ethernet, но не включен в спецификацию Ethernet II.

# Стандарты IEEE 802.X

- **802.3** - описывает физический уровень и подуровень MAC для сетей с немодулированной передачей (baseband networks), использующих шинную топологию и метод доступа CSMA/CD. Этот стандарт был разработан совместно с компаниями Digital, Intel, Xerox и весьма близок к стандарту Ethernet. Однако стандарты Ethernet II и IEEE 802.3 не полностью идентичны и для обеспечения совместимости разнотипных узлов требуется применять специальные меры. 802.3 также включает технологии Fast Ethernet (100BaseTx, 100BaseFx, 100BaseFl).
- **802.5** - описывает физический уровень и подуровень MAC для сетей с кольцевой топологией и передачей маркеров. Этому стандарту соответствуют сети IBM Token Ring 4/16 Мбит/с.
- **802.8** - отчет TAG по оптическим сетям. Документ содержит обсуждение использования оптических кабелей в сетях 802.3 - 802.6, а также рекомендации по установке оптических кабельных систем.

# Стандарты IEEE 802.X

- **802.9** - отчет рабочей группы по интеграции голоса и данных (IVD). Документ задает архитектуру и интерфейсы устройств для одновременной передачи данных и голоса по одной линии. Стандарт 802.9, принятый в 1993 году, совместим с ISDN и использует подуровень LLC, определенный в 802.2, а также поддерживает кабельные системы UTP (неэкранированные кабели из скрученных пар).
- **802.10** - в этом отчете рабочей группы по безопасности ЛВС рассмотрены вопросы обмена данными, шифрования, управления сетями и безопасности в сетевых архитектурах, совместимых с моделью OSI.
- **802.11** - имя рабочей группы, занимающейся спецификацией 100BaseVG Ethernet 100BaseVG. Комитет 802.3, в свою очередь, также предложил спецификации для Ethernet 100 Мбит/с

# Стандарты IEEE 802.X

- Разные комитеты 802.X задают разный порядок битов при передаче.
- 802.3 (CSMA/CD) задает порядок LSB, при котором передается сначала наименее значимый бит (младший разряд)
- 802.5 (token ring) использует обратный порядок - MSB, как и ANSI X3T9.5 - комитет, отвечающий за архитектурные спецификации FDDI.
- Эти два варианта порядка передачи известны как "little-endian" (канонический) и "big-endian" (неканонический), соответственно. Эта разница в порядке передачи имеет существенное значение для мостов и маршрутизаторов, связывающих различные сети.

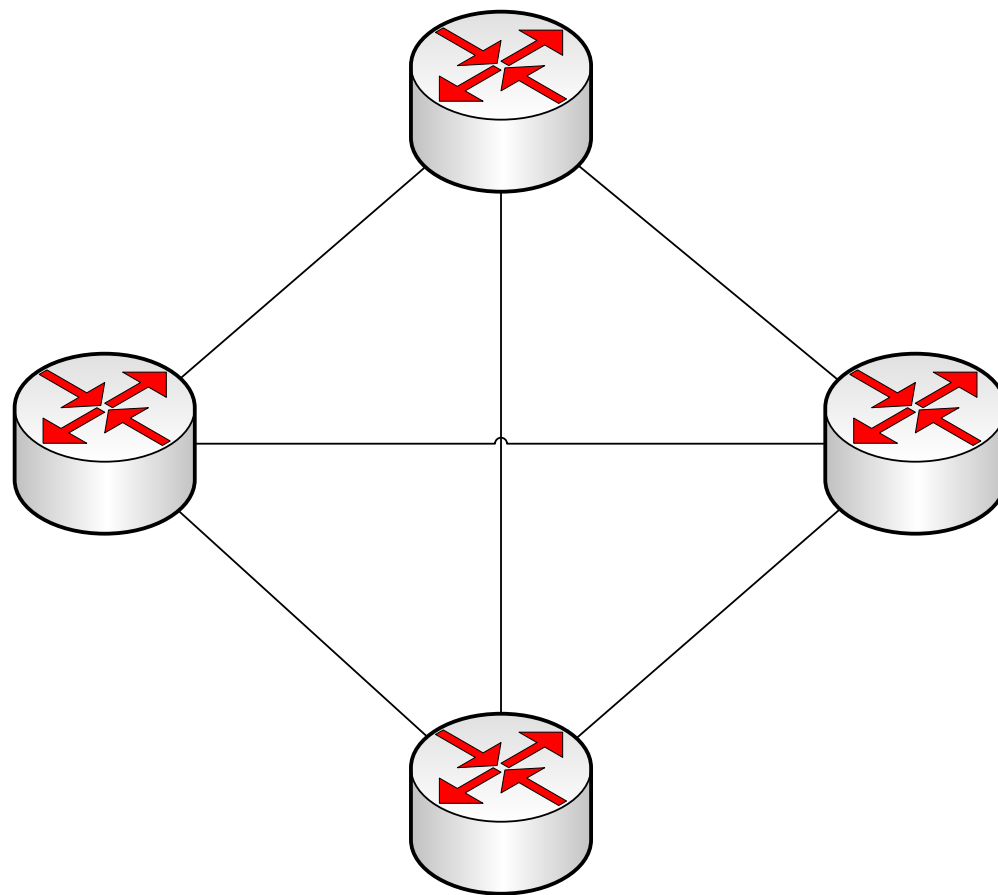
# Лекция III.

# Компоненты сети: узлы и линии связи

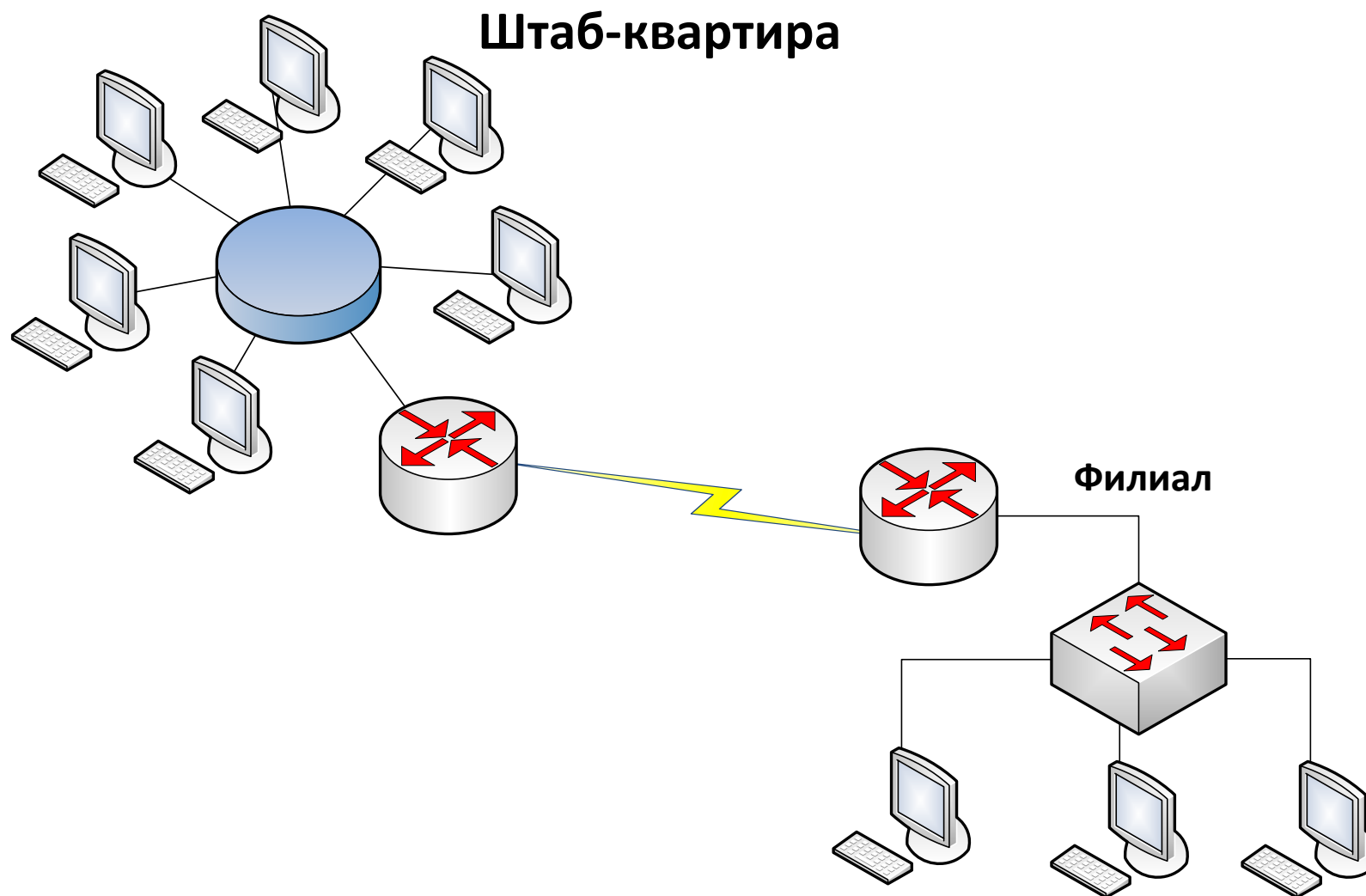
Курс читает: Рогозин Николай Олегович, кафедра ИУ-7



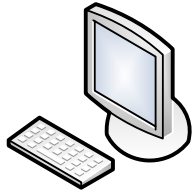
# Пример 1



# Пример 2



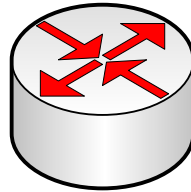
# Условные обозначения



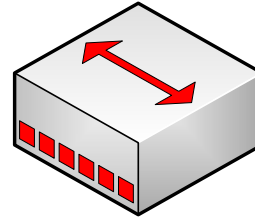
Хост



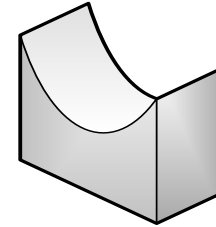
Сервер



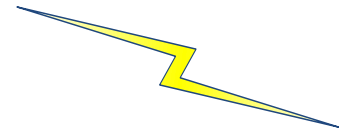
Маршрутизатор



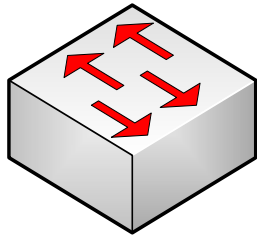
Концентратор



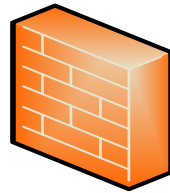
Мост



Серийный кабель



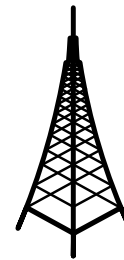
Коммутатор



Брандмауэр



Сеть



Вышка



Ethernet

# Интерфейс

- **Физический интерфейс** (порт) определяется набором электрических связей и характеристиками сигналов. Обычно разъем с набором контактов, каждый из которых имеет определенное назначение, например группа контактов для передачи данных, контакт синхронизации данных и т. п.
- **Логический интерфейс** (протокол) — это набор информационных сообщений определенного формата, которыми обмениваются два устройства или две программы, а также набор правил, определяющих логику обмена этими сообщениями.

# Интерфейс компьютер-компьютер

- аппаратный модуль, называемый сетевым адаптером, или сетевой интерфейсной картой (Network Interface Card, NIC);
- драйвер сетевой интерфейсной карты — специальной программой, управляющей работой сетевой интерфейсной карты.

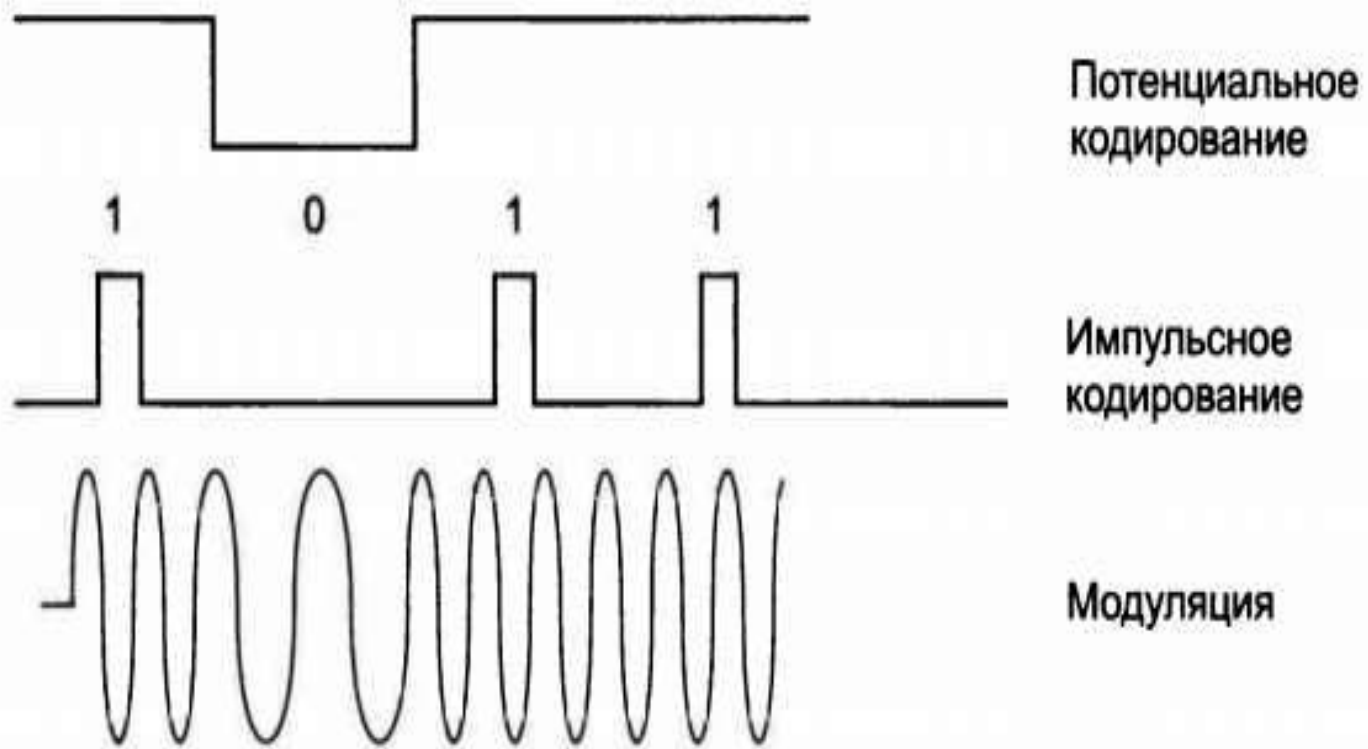
# Физическая среда передачи

- Среда, по которой возможно распространение информационных сигналов в виде электрических, световых и т.п. импульсов.
- На основе проводников, по которым передаются сигналы, строятся проводные (воздушные) или кабельные линии связи.  
**(Проводная среда)**
- В качестве среды также используется земная атмосфера или космическое пространство, через которое распространяются информационные сигналы.  
**(Беспроводная среда).**

# Кабельные линии связи

- Кабель состоит из проводников, заключенных в несколько слоев изоляции: электрической, электромагнитной, механической и, возможно, климатической.
- Может быть оснащен разъемами, позволяющими быстро выполнять присоединение к нему различного оборудования.
- Три основных типа кабеля: кабели на основе скрученных пар медных проводов — неэкранированная витая пара (Unshielded Twisted Pair, UTP) и экранированная витая пара (Shielded Twisted Pair, STP), коаксиальные кабели с медной жилой, волоконно-оптические кабели.

# Кодирование при передаче





# Характеристики линии связи

- **Предложенная нагрузка** — это поток данных, поступающий от пользователя на вход сети. (скорость поступления данных в сеть в битах в секунду)
- **Скорость передачи данных** (information rate, или throughput) — это фактическая скорость потока данных, прошедшего через сеть.
- **Емкость канала связи** (capacity), называемая также пропускной способностью, представляет собой максимально возможную скорость передачи информации по каналу.

# Пропускная способность

- 1) Характерика среды передачи - ширина полосы частот, которую линия передает без существенных искажений измеряется в герцах (Гц)
- 2) Синоним термина емкость канала связи. Измеряется в битах в секунду.

# Типы физических каналов

- **Дуплекс** - одновременная передача информации в обоих направлениях
- **Полудуплекс** - поочередная передача информации в обоих направлениях
- **Симплекс** - односторонняя передача информации

# Коаксиальный кабель (Coaxial cable)

- Два медных проводника, расположенных концентрически (коаксиально)
- Передача со скоростью до 10 Мбит/с
- Максимальная длина сегмента лежит в диапазоне от 185 до 500 м

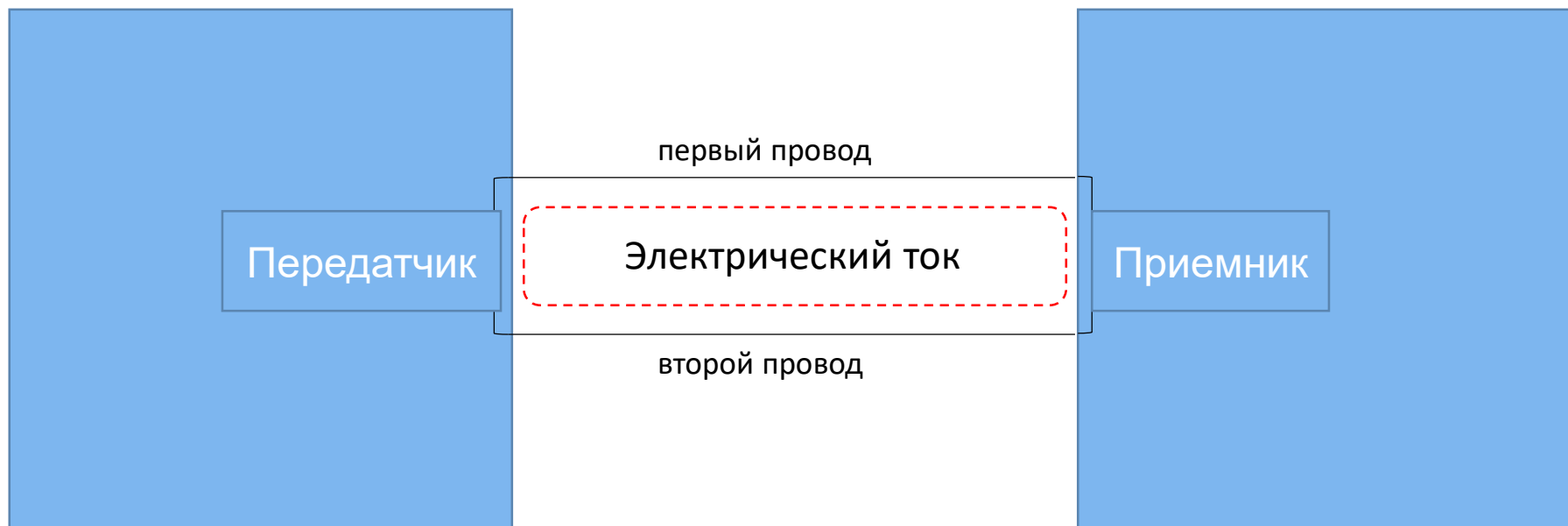
# “Толстый” коаксиальный кабель

- Внешний диаметр около 12 мм
- Волновое сопротивление 50 Ом
- Достаточно толстый внутренний проводник диаметром 2,17 мм, который обеспечивает хорошие механические и электрические характеристики (затухание на частоте 10 МГц — не хуже 18 дБ/км).

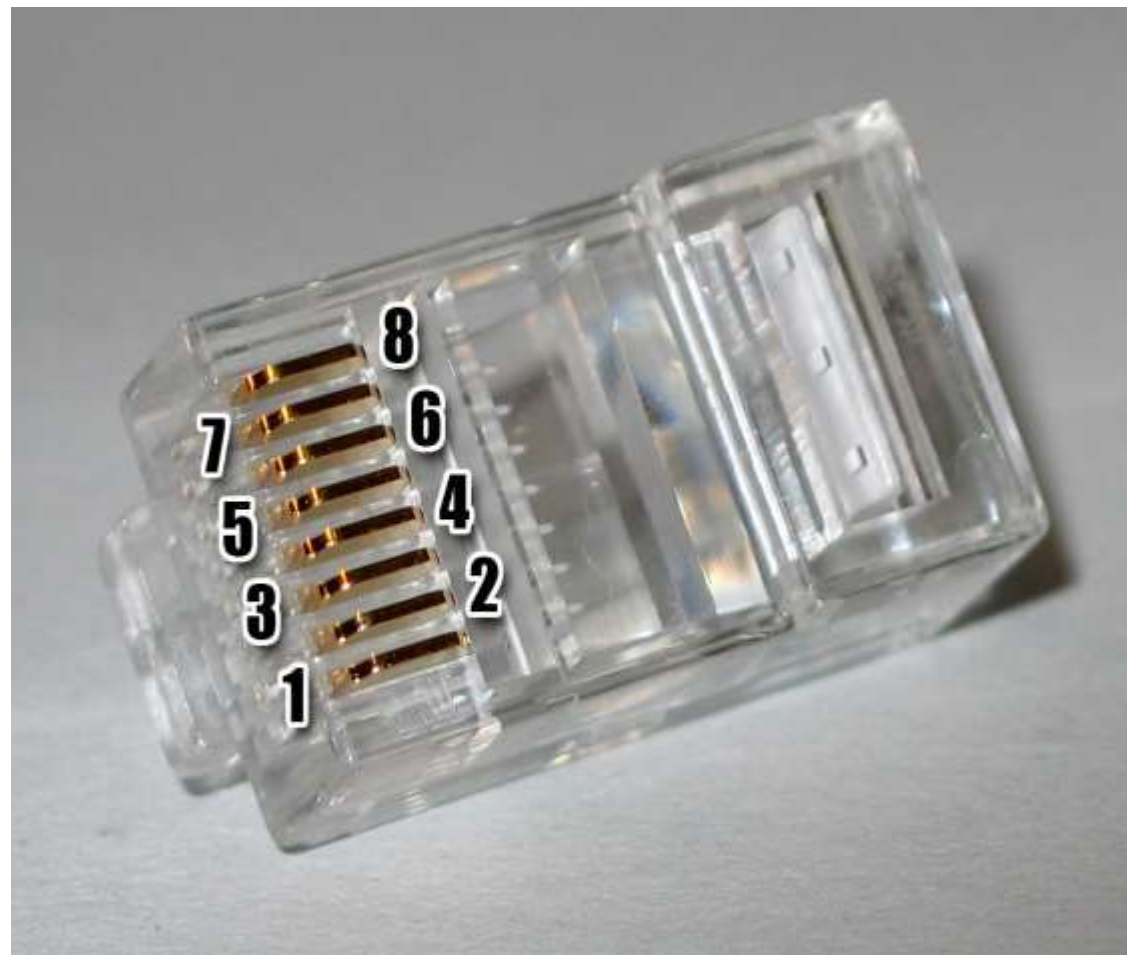
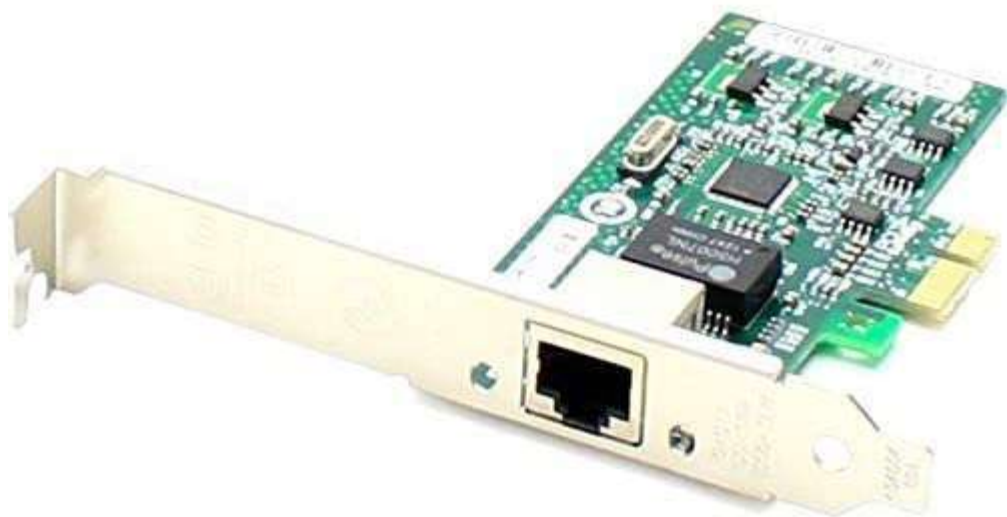
# “Тонкий” коаксиальный кабель

- Внешний диаметр около 50 мм
- Тонкий внутренний проводник 0,89 мм
- Не так прочен, как «толстый» коаксиал, зато обладает гораздо большей гибкостью, что удобно при монтаже.
- Волновое сопротивление 50 Ом, но его механические и электрические характеристики хуже, чем у «толстого» коаксиального кабеля.

# Передача данных витой парой



# Разъем и коннектор 8P8C





# Витая пара

- Несколько пар медных проводов, покрытых пластиковой оболочкой.
- Делится на экранированную (выше защита от интерференций) и неэкранированную
- Состоит из двух одинаковых в конструктивном отношении проводников
- Скорость от 10Мбит/с – 1Гбит/с

# Неэкранированная витая пара (Unshielded Twisted Pair)



# Экранированная витая пара (Shielded Twisted Pair)

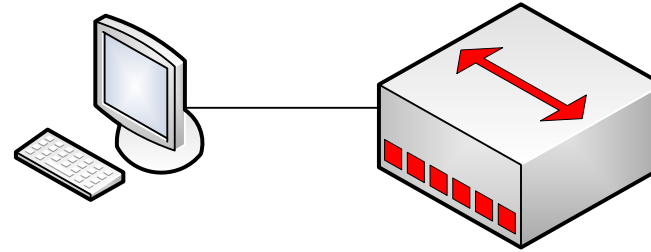


# Категории кабеля 8P8C (т.н. RJ-45)

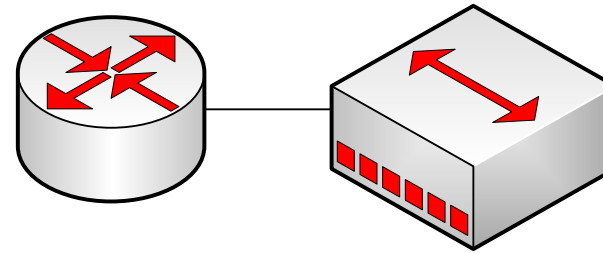
Категория	Скорость
CAT1	До 1 Mbps
CAT2	До 4 Mbps
CAT3	До 10 Mbps
CAT4	До 16 Mbps
CAT5	До 100 Mbps
CAT5-e	До 1 Gbps
CAT6	До 10 Gbps
CAT7	До 10 Gbps
CAT8	До 100 Gbps

# Прямой кабель (Straight-through, T586A)

- Хост к коммутатору или концентратору



- Маршрутизатор к коммутатору или концентратору



# Прямой кабель (Straight-through, T586A)

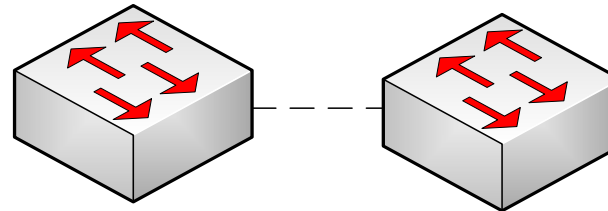
- также называют патч-кабелем, он используется как альтернатива беспроводному соединению, при котором один или более компьютеров связываются с маршрутизатором посредством беспроводного сигнала
- контакты проводов соответствуют контактам на другой стороне

# Кроссовый/ Перекрёстный кабель (Crossover cable, T586B)

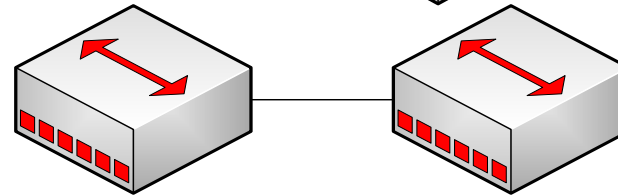
- Используется для прямого соединения компьютерных устройств.
- В отличие от прямого кабеля, использованы разные стандарты расположения контактов передачи: Чтобы получить этот тип соединения с UTP кабелем, один конец должен быть обжат согласно расположению контактов EIA/TIA T568A, а другой конец должен быть обжат согласно схеме T568B.
- часто используется для соединения устройств одного типа, например, двух компьютеров (через сетевой контроллер) или коммутаторов.

# Кроссовый/ Перекрёстный кабель (Crossover cable, 586B)

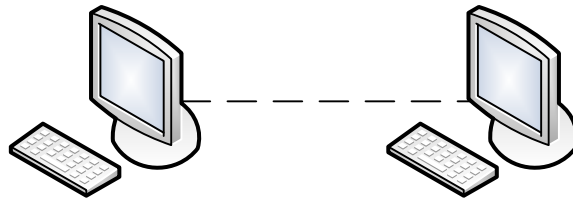
- Коммутатор к коммутатору



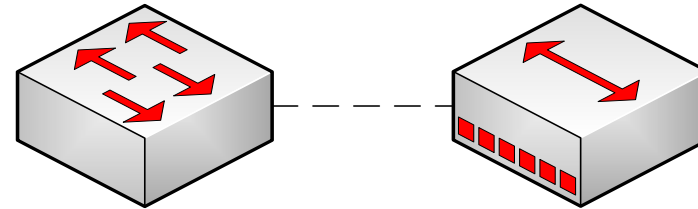
- Концентратор к концентратору



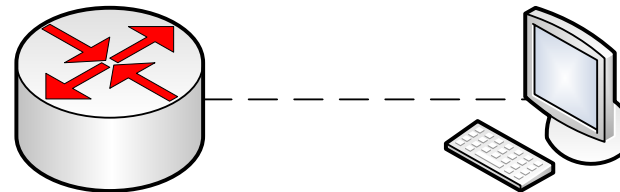
- Хост к хосту



- Концентратор к коммутатору

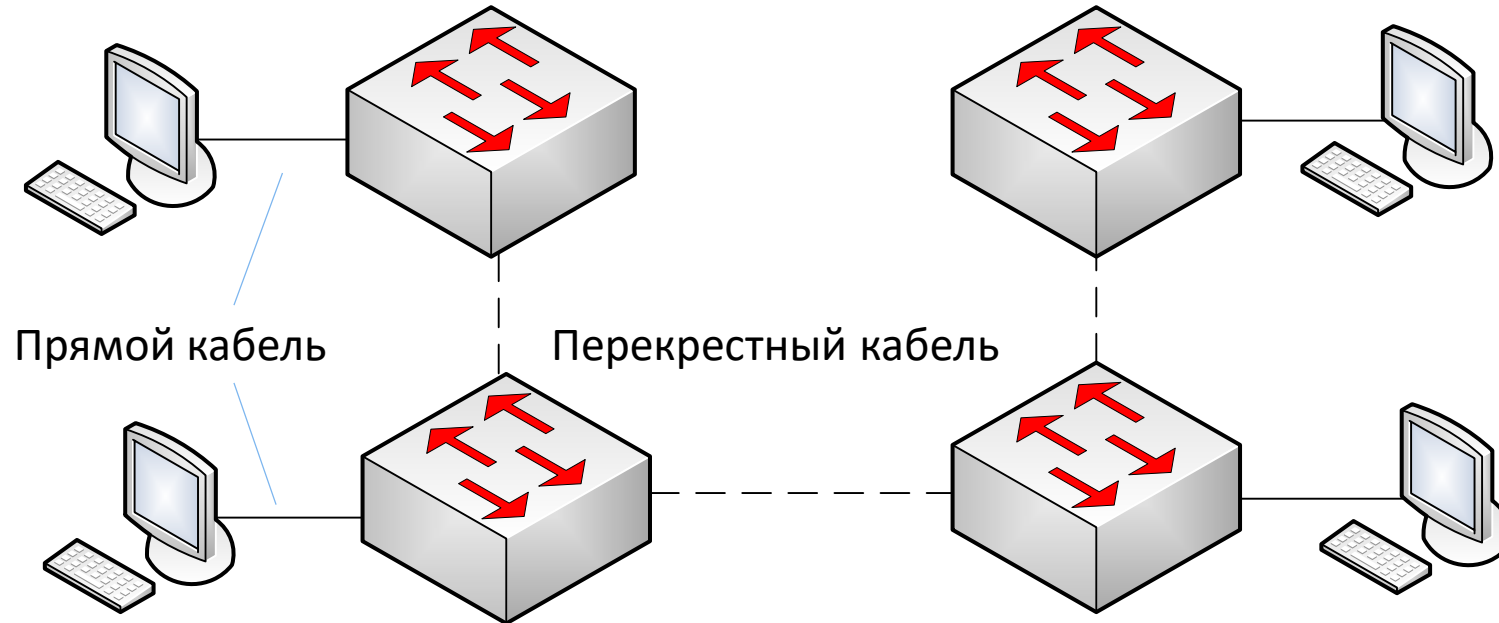


- Маршрутизатор к хосту





# Применение



# Скрученная витая пара/Консольный кабель (Rollover cable)

- Может соединять устр-ва, исп. RS-232 интерфейс без модема
- Подключается через консольный порт (COM)

# Опто-волоконный кабель (Fiber optic cable)

- Тонкий, гибкий кабель, по которому распространяются световые импульсы- биты информации.
- Состоит из центральной стеклянной нити толщиной в несколько микрон, покрытой сплошной стеклянной оболочкой, внутри дополнительной оболочки
- В качестве источников излучения света в волоконно-оптических кабелях применяются:
  - светодиоды, или светоизлучающие диоды (Light Emmited Diode, LED);
  - полупроводниковые лазеры, или лазерные диоды (Laser Diode).

# Опто-волоконный кабель (Fiber optic cable), достоинства

- Низкий уровень шумов/ более широкая полоса пропускания, достигаемая путем передачи сигналов без защиты с использованием различной модуляции и контроля правильности принятой информации только в оконечных терминалах.
- Защищенность от электромагнитных помех. Диэлектрический материал невосприимчив к помехам со стороны окружающих медных кабельных систем и электрического оборудования, способного индуцировать электромагнитное излучение
- Высокая безопасность от несанкционированного доступа. Практически не излучает в радиодиапазоне, передаваемую по нему информацию трудно подслушать, не нарушая приема/передачи.

# Опто-волоконный кабель (Fiber optic cable), достоинства

- Длительный срок эксплуатации (до 25 лет)
- Пожаробезопасность
- Экономичность (изготавливается из кварца)
- Малый вес и объем.
- Малое затухание светового сигнала в волокне. При допустимом затухании 20 дБ максимальное расстояние между усилителями или повторителями составляет около 100 км и более.

# Опто-волоконный кабель (Fiber optic cable), недостатки

- Высокая сложность монтажа (при установке разъемов необходима микронная точность, от точности скола стекловолокна и степени его полировки сильно зависит затухание в разьеме)
- Разветвления, несмотря на то, что технически допускаются, неизбежно сильно ослабляют световой сигнал, и если разветвлений будет много, то свет может просто не дойти до конца сети.
- Меньшая прочность. Чувствительность к перепадам температуры, механическим воздействиям (удары, ультразвук).
- Применяется только в сетях с топологией “звезда” и “кольцо”

# Одномодовое волокно (Single-Mode Fiber)

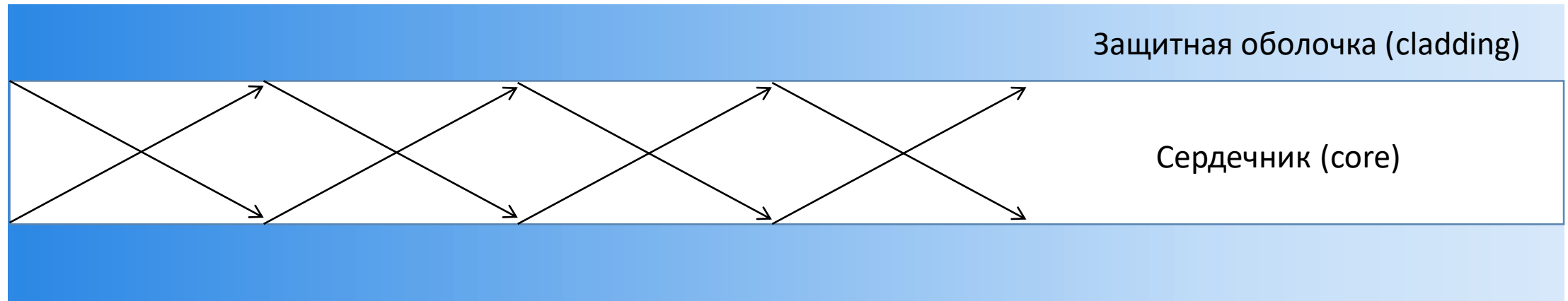


# Одномодовое волокно (Single-Mode Fiber)

- Центральный проводник очень малого диаметра, соизмеримого с длиной волны света, — от 5 до 10 мкм. Практически все лучи света распространяются вдоль оптической оси световода, не отражаясь от внешнего проводника.
- Изготовление сверхтонких качественных волокон для одномодового кабеля представляет собой сложный технологический процесс.
- Кроме того, в волокно такого маленького диаметра достаточно сложно направить пучок света, не потеряв при этом значительную часть его энергии.
- Обладает очень низким затуханием — примерно -0,2 дБ/км для окна прозрачности волны размером в 1550 нм.



# Многомодовое оптоволокно (Multi-mode fiber)



# Многомодовое оптоволокно (Multi-mode fiber)

- Во внутреннем проводнике одновременно существует несколько световых лучей, отражающихся от внешнего проводника под разными углами.
- Угол отражения луча называется модой
- Интерференция ухудшает качество передаваемого сигнала, что приводит к искажениям передаваемых импульсов

# Искажения сигнала в опто-волоконном кабеле

- Затухание
- Хроматическая дисперсия
- Поляризационная дисперсия

# Затухание сигнала

- Мощность сигнала уменьшается из-за поглощения света материалом волокна и примесями, рассеивания света из-за неоднородности плотности волокна, а также из-за кабельных искажений, обусловленных деформацией волокон при прокладке кабеля.
- Затухание измеряется в дБ/км, имеет типичные значения от -0,2 до -0,3 (диапазон 1550 нм), от -0,4 до -1 (диапазон 1310 нм) и от -2 до -3 (диапазон 880 нм).

# Хроматическая дисперсия

- Сигнал искажается из-за того, что волны различной длины распространяются вдоль волокна с различной скоростью.
- Так как прямоугольный импульс имеет спектр ненулевой ширины, из-за хроматической дисперсии составляющие его волны приходят на выход волокна с различной задержкой и фронты импульса оказываются «размытыми».

# Поляризационная дисперсия

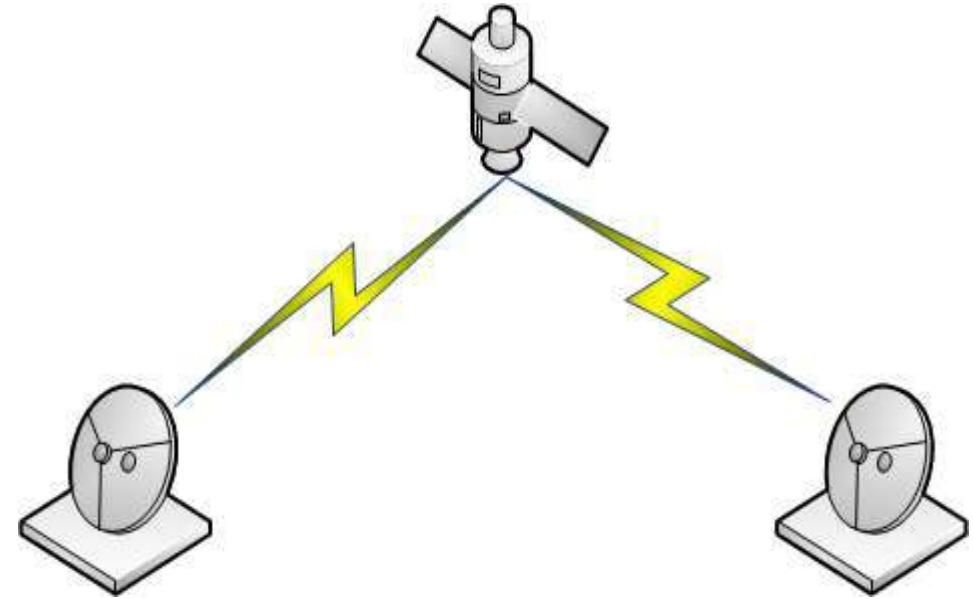
- Световая мода имеет две взаимно перпендикулярные поляризационные составляющие.
- В волноводе с идеальным поперечным сечением, то есть представляющим собой окружность, эти составляющие распространяются с одинаковой скоростью.
- Так как реальные волноводы всегда имеют некоторую овальность, то скорости составляющих отличаются, что приводит к поляризационной дисперсии.

# Наземные радиоканалы

- Сигнал передается электромагнитными волнами радиодиапазона
- Не требует физического носителя
- Обеспечивает соединение с мобильными пользователями
- Передача сигнала на значительное расстояние

# Спутниковые радиоканалы

- Спутниковая связь соединяет два или более наземных приемопередатчика сверхвысокочастотного (СВЧ) диапазона, известных как наземные станции.
- Спутник принимает сигнал на одной полосе частот, восстанавливает его с использованием ретранслятора и передает на другой частоте.





# Аппаратура передачи данных

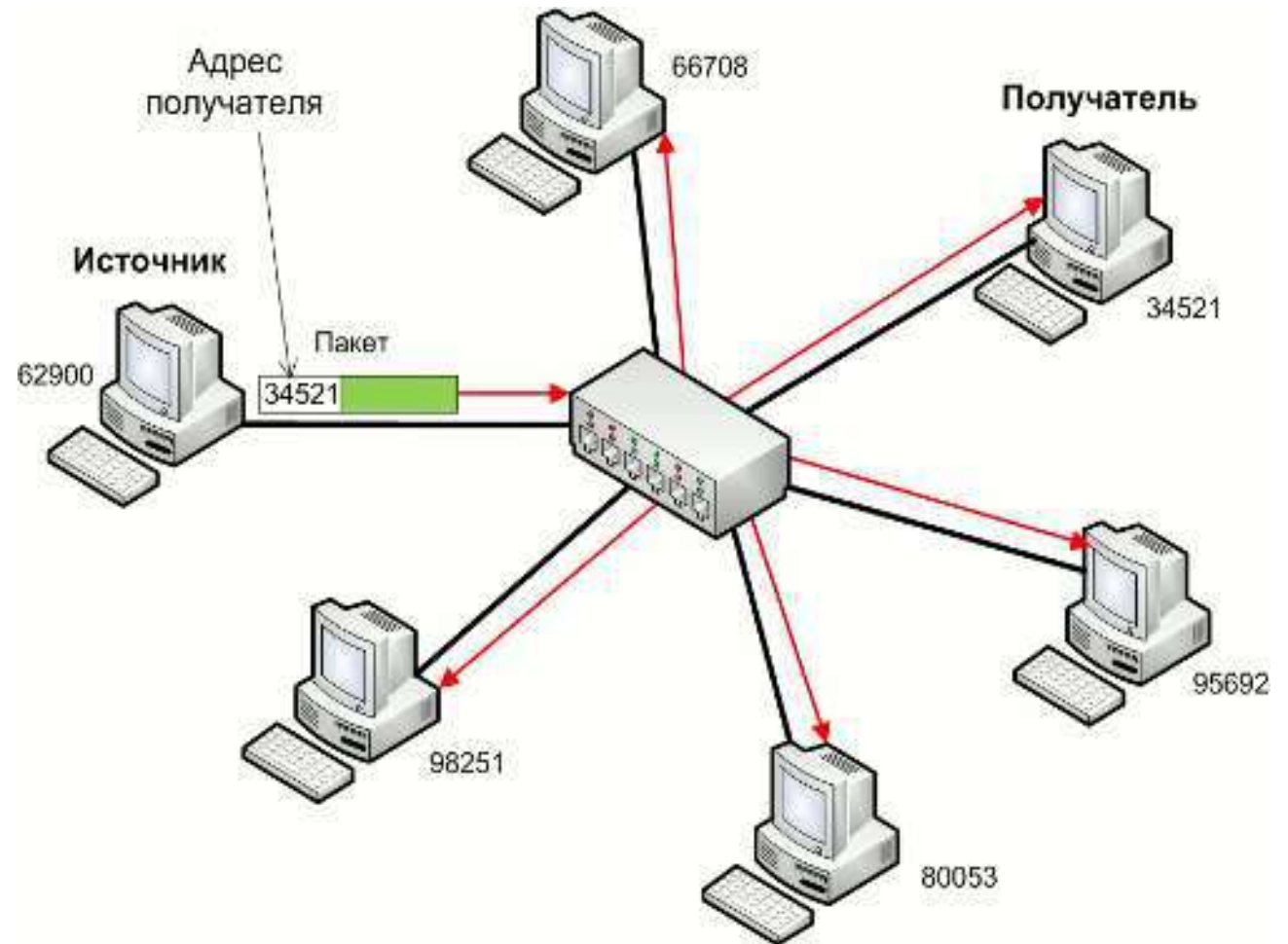
- Непосредственно присоединяет компьютеры или коммутаторы к линиям связи и является, таким образом, пограничным оборудованием.
- Примерами DCE являются модемы (для телефонных линий), терминальные адаптеры сетей ISDN, устройства для подключения к цифровым каналам первичных сетей DSU/CSU (Data Service Unit/Circuit Service Unit).
- Промежуточная аппаратура обычно используется на линиях связи большой протяженности. Она решает две основные задачи:
  - улучшение качества сигнала;
  - создание постоянного составного канала связи между двумя абонентами сети.

# Аппаратура передачи данных

- В локальных сетях промежуточная аппаратура может совсем не использоваться, если протяженность физической среды — кабелей или радиоэфира — позволяет одному сетевому адаптеру принимать сигналы непосредственно от другого сетевого адаптера без дополнительного усиления.
- В глобальных сетях необходимо обеспечить качественную передачу сигналов на расстояния в сотни и тысячи километров. Через определенное расстояние устанавливаются:
  - усилители для повышения мощности сигналов
  - регенераторы для повышения мощности и восстановления формы импульсных сигналов, исказившихся при передаче на большое расстояние

# Концентратор (hub)

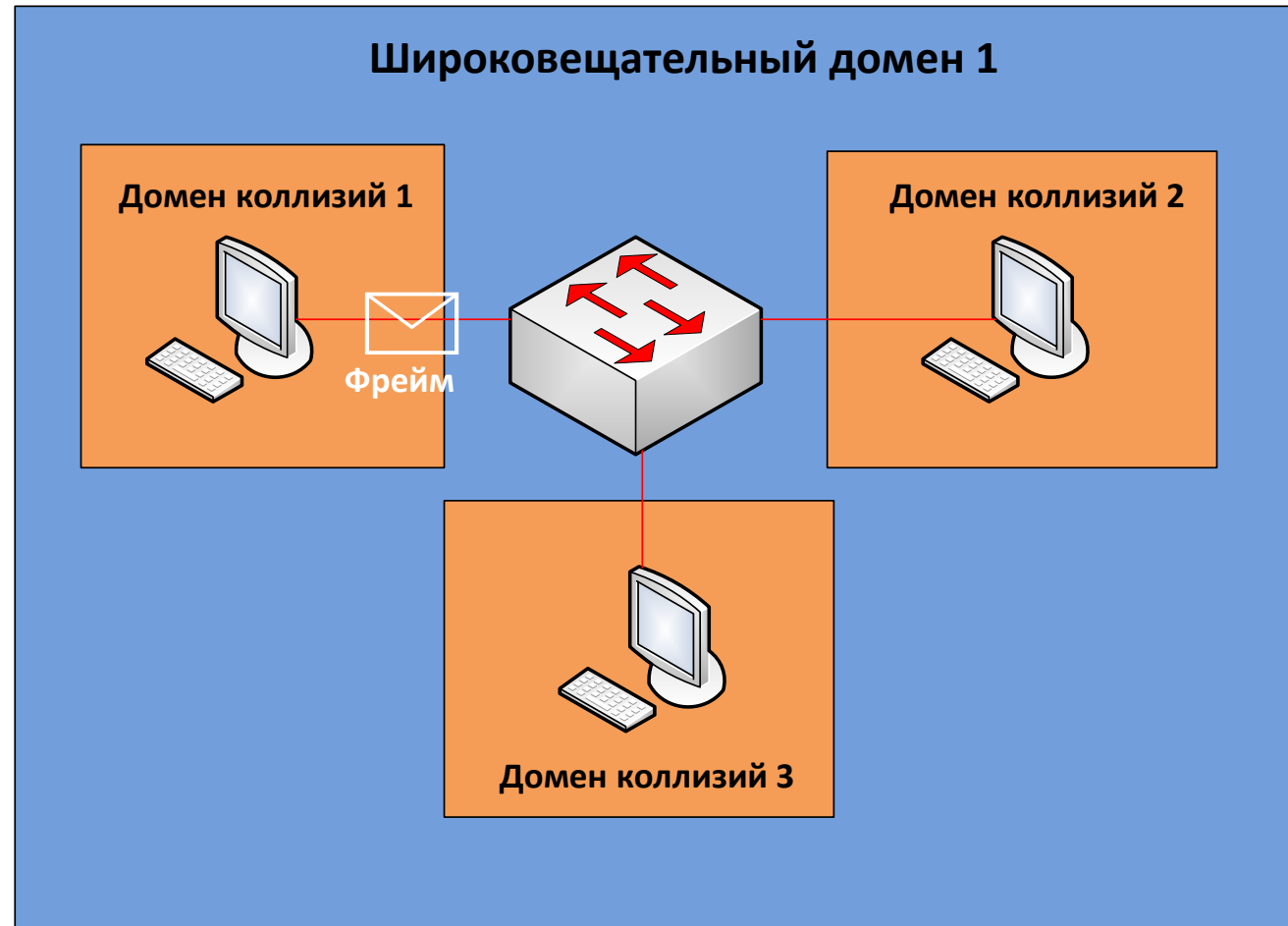
- Многопортовый повторитель
- Объединяет устр-ва в сегменты
- Трансляция пакетов, поступающих на один из его портов на все другие порты



# Коммутатор (Switch)

- устройство, предназначенное для соединения нескольких узлов компьютерной сети в пределах одного или нескольких сегментов сети
- работает на канальном (втором) уровне модели OSI.
- в отличие от концентратора коммутатор передаёт данные только непосредственно получателю

# Коммутатор (Switch)



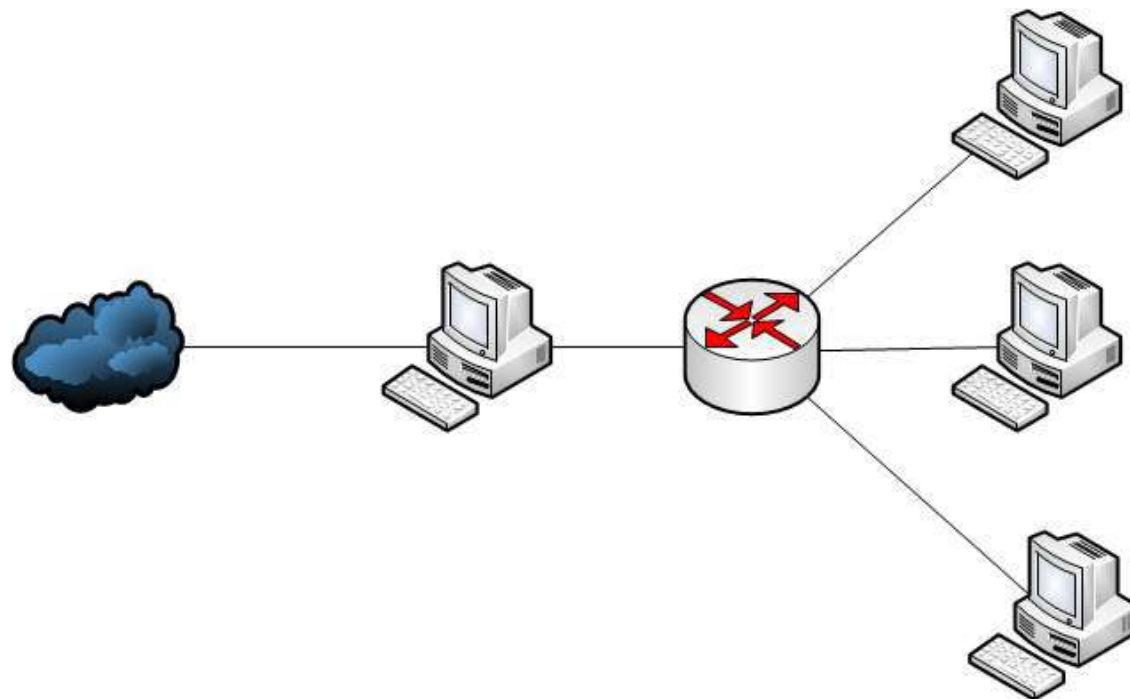
# Маршрутизатор

- Специализированный сетевой компьютер, имеющий два или более сетевых интерфейсов и пересылающий пакеты данных между различными сегментами сети.
- Может связывать разнородные сети различных архитектур.



# Шлюз (gate)

- Аппаратный маршрутизатор или программное обеспечение для сопряжения компьютерных сетей, использующих разные протоколы (например, локальной и глобальной).



# Используемая литература

- Куроуз, Росс “Компьютерные сети. Нисходящий подход.”
- Э. Таненбаум “Компьютерные сети”
- Н. Олифер, В. Олифер “Компьютерные сети. Принципы, технологии, протоколы.”



Спасибо за внимание

# Лекция IV, Основные принципы разработки сетевых приложений

Курс читает Рогозин Николай Олегович, кафедра ИУ-7

# Клиент-серверное приложение

- Типичное сетевое приложение состоит из двух частей — **клиентской** и **серверной** программ, работающих на двух различных конечных системах.
- Когда запускаются эти две программы, создаются два процесса: клиентский и серверный, которые взаимодействуют друг с другом, производя чтение и запись в сокеты.
- Таким образом, при создании сетевого приложения основная задача разработчика — написать коды как для клиентской, так и для серверной частей.

# Одноранговая архитектура

- применение серверов или центров обработки сведено до минимума или вообще до нуля.
- Вместо них приложения используют непосредственное взаимодействие между парой соединенных хостов, называемых пирами

# Открытые приложения

- Для открытых приложений функционирование описывается стандартами выбранного протокола, такими как RFC или другие документы: правила, определяющие операции в нем, известны всем.
- В такой реализации клиентская и серверная программы должны соответствовать правилам, продиктованным документами RFC.
- Например, клиентская программа может быть реализацией клиентской части протокола FTP, определенного в RFC 959; аналогично, серверная программа может быть частью реализации протокола FTP сервера, также описанного в документе RFC 959.

# Проприетарные приложения

- Другой тип сетевых приложений представляют проприетарные (закрытые) приложения. В этом случае клиентские и серверные программы используют протокол прикладного уровня, который открыто не опубликован ни в RFC, ни в каких-либо других документах.
- Разработчик (либо команда разработчиков) создает клиентские и серверные программы и имеет полный контроль над всем, что происходит в коде.
- Но так как в кодах программы не реализован открытый протокол, другие независимые разработчики не смогут написать код, который бы взаимодействовал с этим приложением.

# Сокеты Беркли

- API для разработки сетевых приложений на языке C
- Первое появление – в ОС UNIX BSD 4.1 (1982), затем в UNIX BSD 4.2 (1983)
- Представляют де-факто стандарт абстракции для сетевых сокетов
- Относятся к транспортному уровню, используют протоколы TCP и UDP
- Наиболее широко используются в сетях на базе TCP/IP

# Ввод-вывод сетевых данных

- Поскольку сокеты представляют реализацию протокола TCP/IP в среде Unix, использует те же системные вызовы, что и остальные программы этой среды
- Системные вызовы выглядят как последовательный цикл, состоящий из операций типа открыть-считать-записать-закрыть
- Перед чтением файл всегда должен быть открыт. По окончании записи файл всегда закрывается.
- Одни и те же системные вызовы используются для работы с различной средой: принтером, модемом, дисплеем, файлами.



# Ввод-вывод сетевых данных

- На первых порах разработки интерфейса сокетов пытались заставить сетевой ввод-вывод функционировать так же, как и любой другой ввод-вывод UNIX. Т.е. считывать и записывать данные в цикле открыть-считать-записать-заккрыть.
- Однако возникла проблема: системная модель API не подходила для клиент-серверной архитектуры, т.к. не позволяла реализацию серверной части.

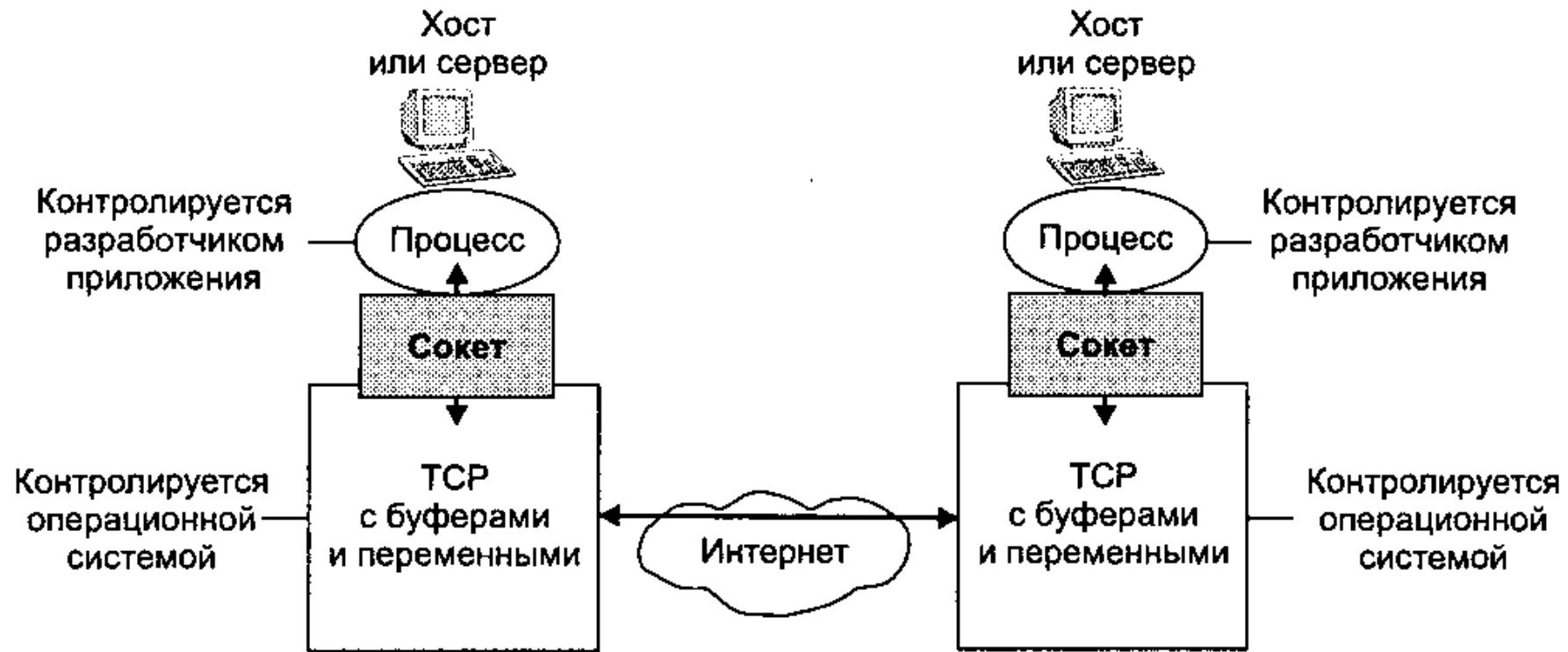
# Ввод-вывод сетевых данных

- Обычная система ввода-вывода Unix не умеет пассивно вводить и выводить данные
- Стандартные функции ввода-вывода UNIX также плохо умеют устанавливать соединения. Как правило, они пользуются фиксированным адресом файла и устройства для обращения к нему.
- Адрес файла или устройства для каждого компьютера — постоянная величина. Не подходит для датаграмм.
- Соединение (или путь) к файлу или устройству доступно на протяжении всего цикла запись-считывание — то есть до тех пор, пока программа не закроет соединение.

# Новый API

- От внесения изменений в существующую систему в итоге отказались в пользу разработки новой
- Тем не менее, ряд сходств остался: дескриптор сокетов в интерфейсе называется дескриптором файла, а информация о сокете хранится в той же таблице, что и дескрипторы файлов

# Работа веб-сокетов



**Рис. 2.18.** Взаимодействие процессов при помощи TCP-сокетов

# Адресация процессов

- Хост можно идентифицировать по его IP-адресу.
- IP-адрес представляет собой 32-разрядное число, которое однозначно определяет хост.
- Отправляющий процесс должен идентифицировать процесс принимающий (а точнее, принимающий сокет), запущенный на хосте-получателе.
- Наиболее популярным сетевым приложениям назначены определенные номера портов. Например, веб-сервер идентифицируется портом 80. Процесс почтового сервера, использующий протокол SMTP, использует порт 25.

# Службы транспортного уровня

- Надежная передача данных
- Пропускная способность
- Время доставки
- Безопасность

# Пропускная способность

- Мгновенная пропускная способность в любой момент времени — это скорость (в бит/с), с которой хост Б получает файл.
- Многие приложения, включая большинство систем в одноранговых сетях с разделением файлового доступа, отображают мгновенную пропускную способность в интерфейсе пользователя во время загрузки.
- Если файл содержит  $F$  бит, а передача занимает  $T$  секунд, то средняя пропускная способность для передачи файла равна  $F/T$  бит/с.

# Пропускная способность

- Транспортный протокол должен предоставлять гарантированную доступную пропускную способность, то есть доставку данных с определенной минимальной скоростью.
- Используя такую службу, приложение может запрашивать гарантированную пропускную способность  $r$  бит/с, и транспортный протокол должен будет заботиться о том, чтобы доступная скорость передачи данных была не меньше  $r$  бит/с.
- Например, если приложение IP-телефонии кодирует голосовые сообщения со скоростью 32 Кбит/с, то используется соотв. пропускная способность



# Время доставки

- Протокол транспортного уровня может также обеспечивать гарантии относительно времени доставки сообщений.
- Временные гарантии тоже предоставляются в различной форме.
- Например, протокол может гарантировать, что каждый бит, отправленный передающей стороной в сокет, приходит на сокет получателя не более чем через 100 мс.

# Безопасность

- Транспортный протокол может предоставлять приложениям одну или несколько служб, относящихся к безопасности.
- Транспортный протокол на передающем хосте способен шифровать все данные, отправленные процессом-источником, а затем на принимающем хосте расшифровывать их перед доставкой процессу-получателю.
- Это обеспечит конфиденциальность между двумя процессами.

<b>Приложение</b>	<b>Потери данных</b>	<b>Пропускная способность</b>	<b>Чувствительность к потере данных</b>
<b>Передача файлов/загрузка</b>	Не допускаются	Эластичное	Нет
<b>Электронная почта</b>	Не допускаются	Эластичное	Нет
<b>Веб-документы</b>	Не допускаются	Эластичное (несколько Кбит/с)	Нет
<b>IP-телефония/ Видео-конференции</b>	Допускаются	Аудио: От нескольких Кбит/с до 1 Мбит/с Видео: От 10 Кбит/с до 5 Мбит/с	Да, сотни миллисекунд
<b>Потоковый аудио и видеоконтент</b>	Допускаются	См. предыдущее	Да, несколько секунд
<b>Интерактивные игры</b>	Допускаются	От нескольких до 10 Кбит/с	Да, сотни миллисекунд
<b>Мгновенные сообщения</b>	Не допускаются	Эластичное	Да и нет

# Популярные приложения и используемые ими протоколы трансп. уровня

Приложение	Протокол прикладного уровня	Базовый транспортный протокол
Электронная почта	SMTP	TCP
Удаленный терминальный доступ	Telnet	TCP
Всемирная паутина	HTTP	TCP
Передача файлов	FTP	TCP
Потоковый мультимедийный контент	HTTP (например, YouTube)	TCP
IP-телефония	SIP , RTP или проприетарное (например, Skype)	UDP или TCP

# Абстракция сокетов

- Сетевое соединение — это процесс передачи данных по сети между двумя компьютерами или процессами.
- Сокет — конечный пункт передачи данных, абстракция, обозначающая одно из окончаний сетевого соединения.
- Каждая из программ, устанавливающих соединение, должна иметь собственный сокет.
- Связь может быть ориентирована на соединение, либо без соединения.

# Абстракция сокетов

- Когда программе нужен сокет, она формирует его характеристики и обращается к API, запрашивая у сетевого ПО его дескриптор.
- Структура таблицы с описанием параметров сокета весьма незначительно отличается от структуры таблицы с описанием параметров файла.

# Дескриптор сокета и дескриптор файла

- дескриптор файла указывает на определенный файл (уже существующий или только что созданный) или устройство
- дескриптор сокета не содержит каких-либо определенных адресов или пунктов назначения сетевого соединения, в отличие от дескрипторов в большинстве ОС
- Программы, работающие с сокетами, сначала образуют сокет и только потом соединяют его с точкой назначения на другом конце сетевого соединения.

# Операции сокетов Беркли

Операция	Назначение
Socket	Создать новый сокет
Bind	Связать сокет с IP-адресом и портом
Listen	Объявить о желании принимать соединения
Connect	Установить соединение
Accept	Принять запрос на установку соединения
Send	Отправить данные по сети
Receive	Получить данные из сети
Close	Закрыть соединение



# Создание сокета

- Интерфейс сокетов позволяет использовать два типа протоколов: с установлением соединения и без
- Процессы создания сокета и соединения происходят отдельно
- Для создания сокета вызывается команда `SOCKET`, возвращающая дескриптор сокета, в котором содержится описание свойств и структуры сокета (так же как и в случае с файлом)

# Создание сокета, cont.

- Пример: **`socket_handle = socket(protocol_family, socket_type, protocol);`**
- Первый параметр: группа или семейство, к которому принадлежит протокол, например семейство TCP/IP.
- Второй параметр, тип сокета, задает режим соединения: датаграммный или ориентированный на поток байтов.
- Третий параметр определяет протокол, с которым будет работать сокет, например TCP.

# Свойства протоколов и адресов

- Существует ряд символьных констант (макроопределений) для указания группы протоколов.
  - 1) **PF\_INET**, например, определяет семейство протоколов TCP/IP.
  - 2) **PF\_UNIX** определяет семейство внутренних протоколов ОС UNIX
- Подобные константы существуют для определения семейств адресов, начинающиеся на префикс “AF\_”
  - 1) **AF\_INET** обозначает семейство TCP/IP
  - 2) **AF\_UNIX** обозначает семейство адресов файловой системы UNIX

# Свойства протоколов и адресов, cont.

- Интерфейс для работы с адресами представляет собой обобщение в соответствии с различием форматов различных сетей
- Из-за тесной связи семейств протоколов и адресов существует заблуждение о том, что это одно и то же
- Однако на сегодняшний день применение AF\_INET и PF\_INET имеет одинаковый результат

# Тип соединения

- Соединение делится на два режима: ориентированные на установление постоянной связи и не требующие этого
- TCP использует для передачи непрерывный поток байт без деления на блоки, в то время как UDP – дейтаграммы, отдельные пакеты данных
- Вторым параметром вызова функции `socket` определяется тип требуемого соединения : **SOCK\_DGRAM** обозначает датаграммы, а **SOCK\_STREAM** — поток байтов.

# Выбор протокола

- Семейство TCP/IP состоит из нескольких протоколов, например IP, ICMP, TCP и UDP. Любое семейство состоит из набора протоколов, которыми пользуются сетевые программисты. Третий параметр функции `socket` позволяет выбрать тот протокол, который будет использоваться вместе с сокетом. Как и в случае остальных параметров, протокол задается символьной константой.
- В сетях TCP/IP все константы начинаются с префикса **IPPROTO\_**. Например, протокол TCP обозначается константой **IPPROTO\_TCP**. Символьная константа **IPPROTO\_UDP** обозначает протокол UDP.

# Выбор протокола, cont.

- Пример вызова функции `socket`:

```
socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
```

- Данный вызов сообщает интерфейсу сокетов о том, что программа желает использовать семейство протоколов Интернет (**PF\_INET**), протокол TCP (**IPPROTO\_TCP**) для соединения, ориентированного на поток байтов (**SOCK\_STREAM**).

# Структура данных сокета

- Семейство протоколов
- Тип сервиса
- Локальный IP адрес
- Удалённый IP адрес
- Локальный порт протокола
- Удалённый порт протокола



# Структура данных сокета

- Каждый раз, когда программа вызывает функцию-socket, реализация сокетов отводит машинную память для новой структуры данных, а затем размещает в ней семейство адресов, тип сокета и протокола.
- В таблице дескрипторов размещается указатель на эту структуру.
- Дескриптор, полученный вашей программой от функции socket, является индексом (порядковым номером) в таблице дескрипторов.

# Настройка сокета

- Каждая сетевая программа вначале создает сокет, вызывая функцию `socket`. При помощи других функций сокет конфигурируется или настраивается так, как это нужно сетевой программе.
- Один из двух типов сетевых служб: датаграммная или ориентированная на поток байтов.
- Один из двух типов поведения программы: клиент или сервер.
- Пять блоков информации, описывающей соединение: протокол, местный и удаленный IP-адреса, номера местного и удаленного портов.

# Соединение сокетов

- Ориентированная на соединение программа-клиент вызывает функцию `connect`, чтобы настроить сокет на сетевое соединение. Функция `connect` размещает информацию о локальной и удаленной конечных точках соединения в структуре данных сокета.
- Функция `connect` требует, чтобы были указаны: дескриптор сокета (указывающий на информацию об удаленном компьютере) и длина структуры адресных данных сокета.

# Функция connect

- **result = connect(socket\_handle, remote\_socket\_address, address\_length);**
- **Первый параметр** функции connect, дескриптор сокета, получен ранее от функции socket.
- **Второй параметр** функции connect — адрес удаленного сокета, является указателем на структуру данных адреса сокета специального вида. Информация об адресе, хранящаяся в структуре, зависит от конкретной сети, то есть от семейства протоколов, которое мы используем.
- **Третий параметр** функции connect, длина адреса, сообщает интерфейсу длину структуры данных адресов удаленного сокета (второй параметр), измеренную в байтах

# Функция bind

- Функция **bind** интерфейса сокетов позволяет программам связать локальный адрес (совокупность адресов локального компьютера и номера порта) с сокетом.
- Следующий оператор иллюстрирует вызов функции bind:

```
result = bind(socket_handle, local_socket_address,  
address_length) ;
```

# Передача данных через сокет

- После того как сокет сконфигурирован, через него можно установить сетевое соединение. Процесс сетевого соединения подразумевает посылку и прием информации. Интерфейс сокетов включает несколько функций для выполнения этих обеих задач.
- Интерфейс сокетов Беркли обеспечивает пять функций для передачи данных через сокет. Эти функции разделены на две группы.
- Трём из них требуется указывать адрес назначения в качестве аргумента, а двум остальным — нет. Основное различие между двумя группами состоит в их ориентированности на соединение.

# Функция write

- **result = write(socket\_handle, message\_buffer, buffer\_length);**
- **Первый параметр**, дескриптор сокета - он обозначает структуру в таблице дескрипторов, содержащую информацию о данном сокете.
- **Второй параметр** функции write, буфер сообщения, указывает на буфер, то есть область памяти, в которой расположены предназначенные для передачи данные.
- **Третий параметр** вызова функции обозначает длину буфера, то есть количество данных для передачи.

# Функция `writen`

- **`result = writen(socket_handle, io_vector, vector_length);`**
- **Первым параметром** требуется указывать дескриптор сокета, также как в функции `write`.
- **Второй параметр**, вектор ввода-вывода, указывает на массив указателей.
- **Третий параметр** функции `writen` определяет количество указателей в массиве указателей, заданном вектором ввода-вывода.



# Функция `writen`, cont.

- Предположим, что данные для передачи располагаются в различных областях памяти.
- В этом случае каждый член массива указателей представляет собой указатель на одну из областей памяти, содержащей данные для передачи.
- Когда функция `writen` передает данные, она находит их по указанным прикладной программой в массиве указателей адресам.
- Данные высылаются в том порядке, в каком их адреса указаны в массиве указателей.

# Функция send

- **result = send(socket\_handle, message\_buffer, buffer\_length, special\_flags) ;**
- **Первый параметр:** дескриптор, описывающий подключенный сокет
- **Второй параметр:** указатель на буфер передаваемых данных
- **Третий параметр:** длина, в байтах, данных в буфере на который указывает указатель во втором параметре
- **Четвертый параметр:** набор флагов, определяющих параметры вызова

# Функция sendto

- Следующий оператор демонстрирует вызов функции sendto:
- **result = sendto(socket\_handle, message\_buffer, buffer\_length, special\_flags, socket\_address\_structure, address\_structure\_length) ;**
- Первые четыре те же, что и в функции send.
- Пятый параметр, структура адреса сокета, определяет адрес назначения.
- Шестой параметр, длина структуры адреса сокета, - размер этой структуры в байтах.

# Функция `sendmsg`

- **`result = sendmsg(socket_handle, message_structure, special_flags);`**
- `sendmsg` позволяет использовать гибкую структуру данных вместо буфера, расположенного в непрерывной области памяти.
- Как и в функции `writen`, структура сообщения содержит указатель на массив адресов памяти.

# Прием данных через сокет

- В интерфейсе сокетов есть пять функций, предназначенных для приема информации. Они называются: `read`, `readv`, `recv`, `recvfrom`, `recvmsg` и соответствуют функциям, использующимся для передачи данных.
- Например, функции `recv` и `send` обладают одинаковым набором параметров.
- Точно так же одинаков набор параметров и у функций `writen` и `readv`. Функция `writen` передает данные, а `readv` — принимает. И та и другая позволяют задать массив адресов памяти, где располагаются данные.
- Функции `recvfrom` и `recvmsg` соответствуют функциям `sendto` и `sendmsg`.

# Соответствие функций приема и передачи

<b>Функция передачи данных</b>	<b>Соответствующая функция приема данных</b>
send	recv
write	read
writen	readv
sendto	recvfrom
sendmsg	recvmsg

# Функция listen

- **result = listen(socket\_handle, queue\_length);**
- переводит сокет в пассивный режим ожидания
- подготавливает его к обработке множества одновременно поступающих запросов, организуя их в виде очереди
- Первый параметр: дескриптор сокета
- Второй параметр: длина очереди
- В настоящее время максимальная длина очереди равна пяти, однако задание длины 1 или 2 также полезно и позволит серверу не отвергнуть запрос если он не справился с обработкой

# Функция ассерт

- позволяет серверу принять запрос на соединение, поступивший от клиента. После того как установлена входная очередь, программа-сервер вызывает функцию ассерт и переходит в режим ожидания (паузы), ожидая запросов.
- После того как установлена входная очередь, программа-сервер вызывает функцию ассерт и переходит в режим ожидания (паузы), ожидая запросов.



# Функция accept, cont.

- **result = accept(socket\_handle, socket\_address, address\_length);**
- Дескриптор сокета описывает сокет, который будет прослушиваться сервером.
- В момент появления запроса реализация сокетов заполняет структуру адреса (на которую указывает второй параметр) адресом клиента, от которого поступил запрос.
- Реализация сокетов заполняет также третий параметр, размещая в нем длину адреса.

# Функция select

- позволяет одиночному процессу следить за состоянием сразу нескольких сокетов.
- **result = select(number\_of\_sockets, readable\_sockets, writeable\_sockets, error\_sockets, max\_time);**
  - Первый параметр, количество сокетов (number of sockets), задает общее количество сокетов для наблюдения.
  - Параметры readable-sockets, writeable-sockets и error-sockets являются битовыми масками, задающими тип сокетов.

# Пример сокета-сервера

```
import socket

serv_sock = socket.socket(socket.AF_INET,    #Первый сокет - слушающий
                           socket.SOCK_STREAM,
                           proto=0)

serv_sock.bind(('127.0.0.1', 8700))          #Слушающий сокет привязывается к порту

serv_sock.listen(10)

client_sock, client_addr = serv_sock.accept() #Второй сокет отвечает за прием и передачу данных

while True:
    data = client_sock.recv(1024)
    if not data:
        break
    client_sock.sendall(data)

client_sock.close()
```



# Пример сокета-клиента

```
import socket
import struct

client_sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client_sock.connect(('127.0.0.1', 8700))

client_sock.sendall(bytes(str(123), 'utf8'))
data = client_sock.recv(1024)
strings = str(data, 'utf8')
num = int(strings)
client_sock.close()

print('Received: ' + str(num))
```

# Протоколы прикладного уровня

Курс читает Рогозин Н.О., каф. ИУ-7

# Telnet

- Сетевой протокол для реализации текстового интерфейса по сети
- Был одним из первых протоколов удаленного сообщения
- Позволяет обслуживающей машине рассматривать все удаленные терминалы как стандартные "сетевые виртуальные терминалы" строчного типа, работающие в коде ASCII, а также обеспечивает возможность согласования более сложных функций (например, локальный или удаленный эхо-контроль, страничный режим, высота и ширина экрана и т.д.)

# Telnet

- Протокол имеет симметричную структуру сообщения, позволяя двум терминалам обмениваться:
  - 1) прикладными данными
  - 2) командами протокола Telnet
- На прикладном уровне над TELNET находится либо программа поддержки реального терминала (на стороне пользователя), либо прикладной процесс в обсуживающей машине, к которому осуществляется доступ с терминала.

# Пример настройки доступа в Packet Tracer

- Router(config)#line vty 0 4
- Router(config-line)#password cisco
- Router(config-line)#login
- Router(config)#service password-encryption

либо:

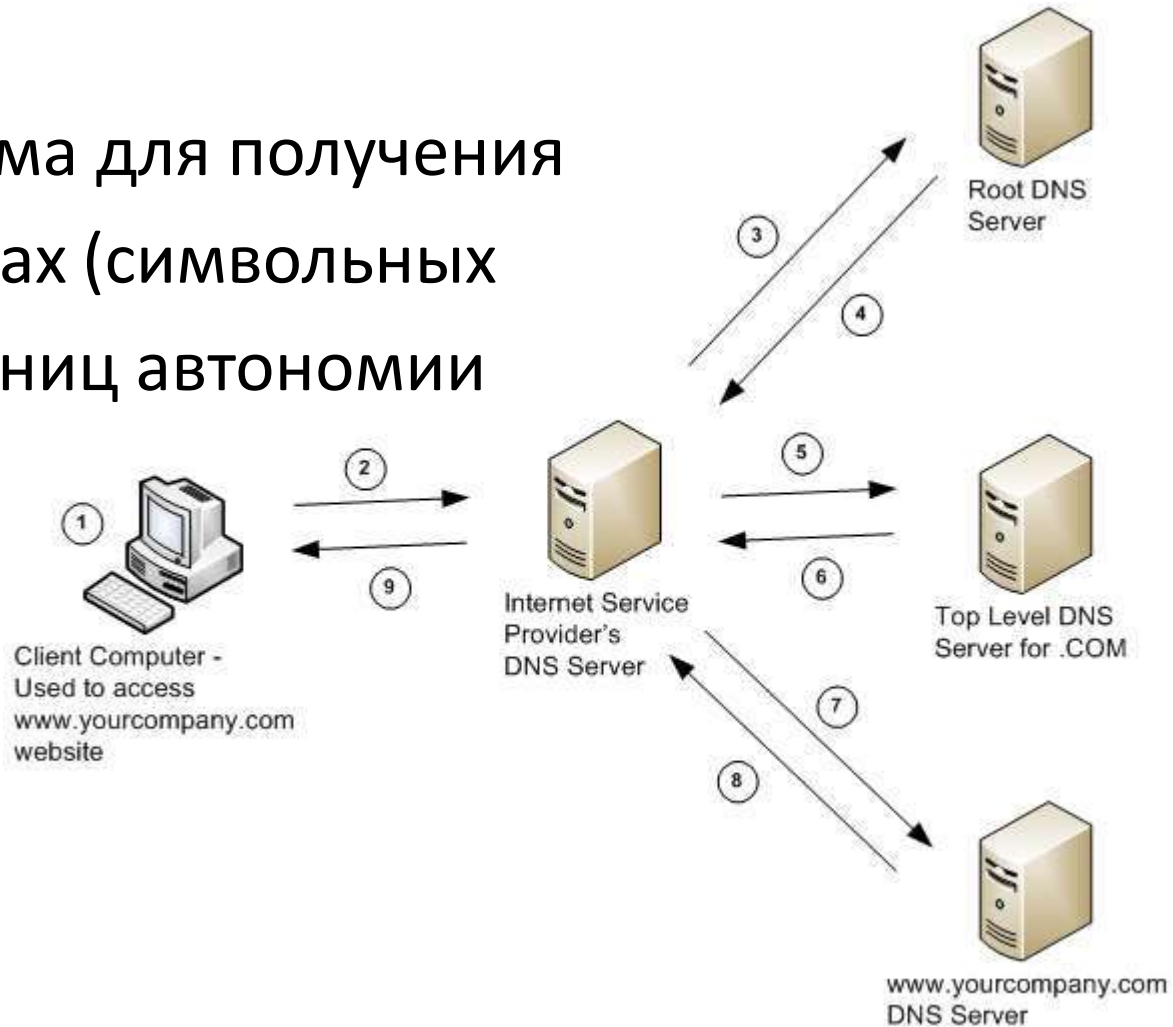
- Router(config)#aaa new-model
- Router(config)#username admin password 1234

(в рамках модели AAA (Authentication, Authorization, Accounting)).  
При этом появляется возможность использовать для аутентификации на устройстве RADIUS или TACACS сервер.



# DNS

- Распределенная система для получения информации о доменах (символьных идентификаторах единиц автономии в сети Интернет).



# DNS

- Протокол несимметричен - в нем определены DNS-серверы и DNS-клиенты. DNS-серверы хранят часть распределенной базы данных о соответствии символьных имен и IP-адресов.
- Эта база данных распределена по административным доменам сети Internet.
- Клиенты сервера DNS знают IP-адрес сервера DNS своего административного домена и по протоколу IP передают запрос, в котором сообщают известное символьное имя и просят вернуть соответствующий ему IP-адрес.

# Зона DNS и файл отображений

- Часть пространства доменных имен, для которых некоторый сервер DNS имеет информацию об их отображениях на основе соответствующего текстового файла, называется **зоной DNS** данного сервера, а сам текстовый файл — **файлом зоны**.

# Файл hosts (файл отображений)

```
1  # Copyright (c) 1993-2009 Microsoft Corp.
2  #
3  # This is a sample HOSTS file used by Microsoft TCP/IP for Windows.
4  #
5  # This file contains the mappings of IP addresses to host names. Each
6  # entry should be kept on an individual line. The IP address should
7  # be placed in the first column followed by the corresponding host name.
8  # The IP address and the host name should be separated by at least one
9  # space.
10 #
11 # Additionally, comments (such as these) may be inserted on individual
12 # lines or following the machine name denoted by a '#' symbol.
13 #
14 # For example:
15 #
16 #       102.54.94.97       rhino.acme.com          # source server
17 #       38.25.63.10       x.acme.com              # x client host
18
19 # localhost name resolution is handled within DNS itself.
20 #   127.0.0.1      localhost
21 #   ::1            localhost
```

# DNS

- Если данные о запрошенном соответствии хранятся в базе данного DNS-сервера, то он сразу посылает ответ клиенту, если же нет - то он посылает запрос DNS-серверу другого домена, который может сам обработать запрос, либо передать его другому DNS-серверу.
- Все DNS-серверы соединены иерархически, в соответствии с иерархией доменов сети Internet.
- Клиент опрашивает эти серверы имен, пока не найдет нужные отображения. Этот процесс ускоряется из-за того, что серверы имен постоянно кэшируют информацию, предоставляемую по запросам.
- Клиентские компьютеры могут использовать в своей работе IP-адреса нескольких DNS-серверов, для повышения надежности своей работы.

# Иерархические символьные имена

- База данных DNS имеет структуру дерева, называемого доменным пространством имен, в котором каждый домен (узел дерева) имеет имя и может содержать поддомены.
- Имя домена идентифицирует его положение в этой базе данных по отношению к родительскому домену, причем точки в имени отделяют части, соответствующие узлам домена.

# Иерархические символьные имена

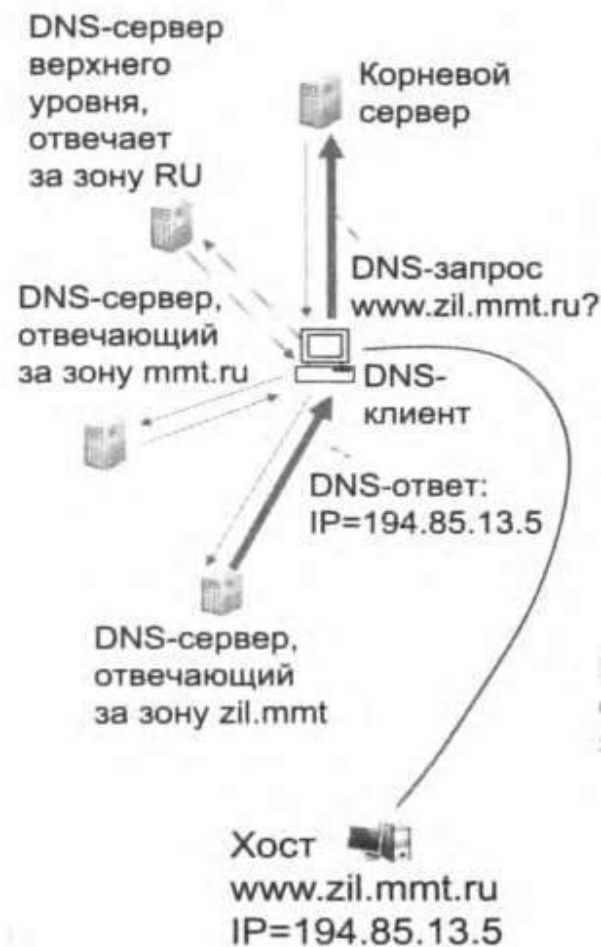
- Корень базы данных DNS управляется центром INIC (Internet Network Information Center).
- Домены верхнего уровня назначаются для каждой страны, а также на организационной основе. Для обозначения стран используются трехбуквенные и двухбуквенные аббревиатуры, а для различных типов организаций используются следующие аббревиатуры:
  - com - коммерческие организации (например, microsoft.com);
  - edu - образовательные (например, mit.edu);
  - gov - правительственные организации (например, nsf.gov);

# DNS

- Каждый домен DNS
  - Администрируется отдельной организацией, которая обычно разбивает свой домен на поддомены и передает функции администрирования этих поддоменов другим организациям.
  - Имеет уникальное имя, а каждый из поддоменов имеет уникальное имя внутри своего домена.
- Имя домена может содержать до 63 символов. Каждый хост в сети Internet однозначно определяется своим полным доменным именем - FQDN (fully qualified domain name), которое включает имена всех доменов по направлению от хоста к корню.
  - Например: **students.bmstu.ru**



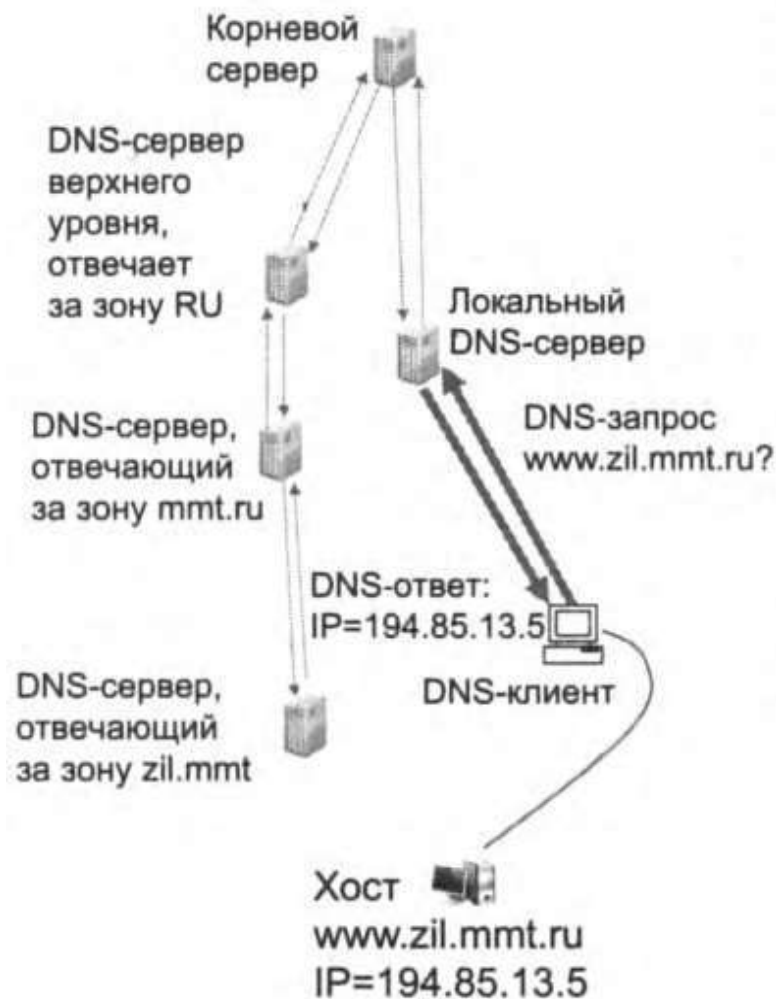
# Итеративная процедура разрешения имен DNS



# Итеративная процедура DNS

- DNS-клиент обращается к корневому DNS-серверу с указанием полного доменного имени `www.zil.mmt.ru` хоста, для которого он хочет найти IP-адрес.
- Корневой DNS-сервер отвечает клиенту, указывая адреса DNS-серверов верхнего уровня, обслуживающих домен, заданный в старшей части запрошенного имени, в данном случае — домен `ru`.
- DNS-клиент делает следующий запрос к одному из предложенных ему DNS-серверов верхнего уровня, который отсылает его к DNS-серверу нужного поддомена (в примере это сервер, отвечающий за зону `mmt.ru`), и так далее, пока не будет найден DNS-сервер, в котором хранится отображение запрошенного имени на IP-адрес. Этот сервер дает окончательный ответ клиенту, который теперь может установить связь с хостом по IP-адресу `194.85.13.5`.

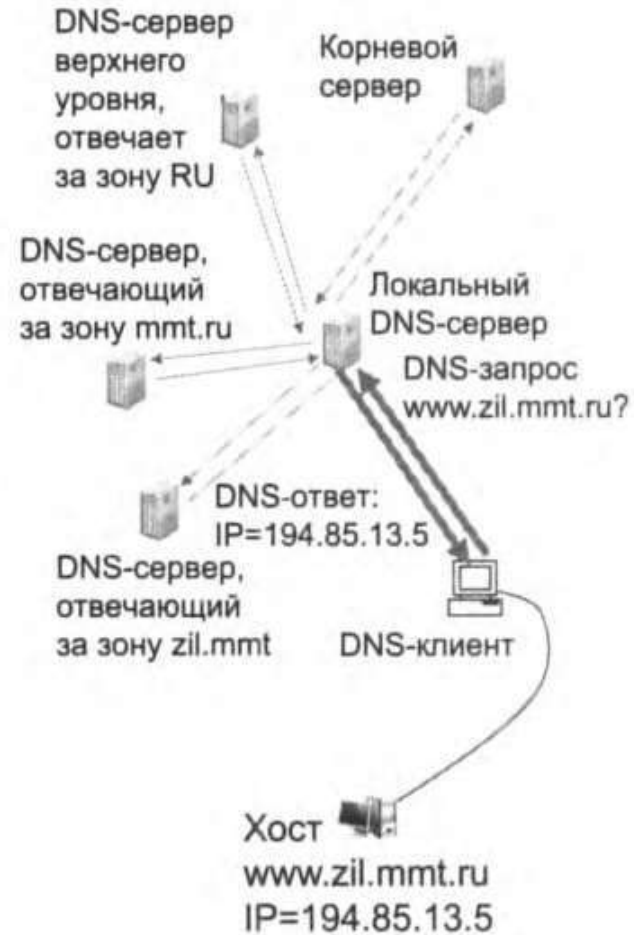
# Рекурсивная процедура разрешения имен DNS



# Рекурсивная процедура DNS

- DNS-клиент отправляет запрос к локальному DNS-серверу, то есть серверу, обслуживающему поддомен, которому принадлежит имя клиента.
- Если локальный DNS-сервер знает ответ, то он сразу же возвращает его клиенту. Это может быть полномочный ответ (запрошенное имя входит в тот же поддомен, что и имя клиента) или неполномочный ответ (сервер уже узнавал данное соответствие для другого клиента и сохранил его в своем кэше).
- Если локальный DNS-сервер не знает ответа, то он обращается к корневому серверу, который переправляет запрос к DNS-серверу верхнего уровня (отвечающему за зону RU), который в свою очередь запрашивает нижележащий сервер (зона mmt), и так далее, пока запрос не дойдет до полномочного сервера, имеющего в своем файле зоны запись о запрошенном имени.

# Смешанная процедура DNS



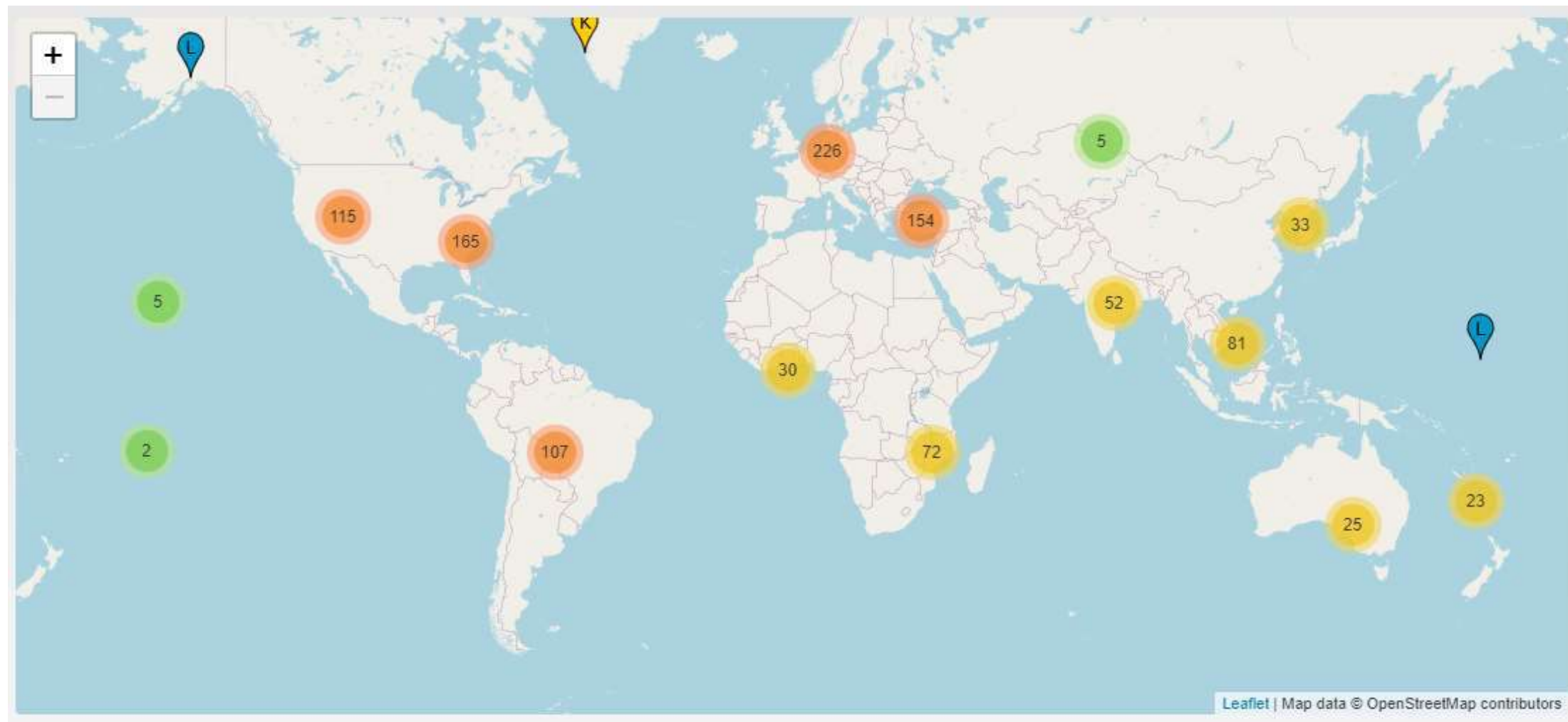
# Смешанная процедура разрешения имен DNS

- Начальная часть процедуры, когда DNS-клиент передает запрос локальному DNS- серверу и поручает ему действовать от его имени, является рекурсивной.
- Затем, если локальный DNS-сервер не знает ответ, то он последовательно выполняет итеративные запросы к иерархии серверов точно так же, как это делал DNS-клиент в первом варианте. Получив ответ, локальный DNS-сервер передает его клиенту.

# Корневые сервера

- Разрешение всех запросов, ответы на которые не находятся в кэше или файле зоны какого-либо DNS-сервера нижнего уровня, начинаются с обращения к одному из корневых серверов.
- Разработчики системы DNS понимали это, поэтому изначально было решено обеспечить высокую степень резервирования: было установлено 13 корневых серверов с именами a.root-servers.net, b.root-servers.net, c.root-servers.net,... m.root-servers.net и тринадцатью IP-адресами.
- С тех пор организация корневых DNS-серверов изменилась.
- Более 300 серверов
- Большая отказоустойчивость и производительность современной службы DNS.
- Все корневые серверы по-прежнему разделяют те же 13 имен (от a.root-servers.org до rn.root-servers.org) и 13 IP-адресов.
- Но теперь каждому имени и адресу соответствует кластер серверов. Например, имени f.root-servers.net соответствует 56 серверов, а имени lroot-servers.net — 146.

# Карта корневых серверов (октябрь 2020)





# Обратная зона

- Система таблиц , которая хранит соответствие между IP-адресами и DNS-именами хостов в некоторой сети
- Находит нужное DNS-имя по IP адресу
- IP-адрес представляется в виде DNS -имени (в обратном порядке).
- Учитывая, что при записи IP-адреса старшая часть является самой левой частью адреса, а при записи DNS-имени — самой правой, составляющие в преобразованном адресе указываются в обратном порядке
- Например, для адреса 192.31. 106.0 — 106.31.192 .

# Актуальные проблемы DNS

- Denial of Service (DoS) атаки пропускной способности сети
- DoS атаки центрального процессора/потребление памяти
- Изменения в протоколах и системах безопасности
- Data Integrity
- Компрометация DNSKEY (ключ системы защиты DNSSEC)

# Режимы DHCP

- Ручное назначение статических адресов;
- Автоматическое назначение статических адресов;
- Автоматическое распределение динамических адресов.

# Ручной режим

- Администратор, помимо пула доступных адресов, снабжает DHCP-сервер информацией о жестком соответствии IP-адресов физическим адресам или другим идентификаторам клиентских узлов.
- DHCP-сервер, пользуясь этой информацией, всегда выдаст определенному DHCP-клиенту один и тот же назначенный ему администратором IP-адрес

# Автоматическое назначение

- DHCP-сервер самостоятельно, без вмешательства администратора произвольным образом выбирает клиенту IP-адрес из пула наличных IP-адресов.
- Адрес дается клиенту из пула в постоянное пользование, то есть между идентифицирующей информацией клиента и его IP-адресом по-прежнему, как и при ручном назначении, существует постоянное соответствие.
- Оно устанавливается в момент первого назначения DHCP-сервером IP-адреса клиенту.
- При последующих запросах сервер возвращает клиенту тот же самый IP-адрес.

# Динамическое распределение

- DHCP-сервер выдает адрес клиенту на ограниченное время, называемое сроком аренды.
- Когда компьютер, являющийся DHCP-клиентом, удаляется из подсети, назначенный ему IP-адрес автоматически освобождается.
- Когда компьютер подключается к другой подсети, то ему автоматически назначается новый адрес.
- Ни пользователь, ни сетевой администратор не вмешиваются в этот процесс. Это дает возможность впоследствии повторно использовать этот IP-адрес для назначения другому компьютеру.

# Свойства набора адресов DHCP

- Диапазон адресов + набор исключений
- Маска подсети
- Длительность аренды
- DNS-сервер
- Шлюз по умолчанию

# Пример использования

Сервер DHCP: ☒ Включить ☐ Отключить

DHCP начальн. IP адрес:

DHCP конечн. IP адрес:

Время аренды адреса DHCP:  :  :  :   
(Дней: Часов: Минут: Секунд)

DNS Servers:

MAX Bridge MTU:  (46~1954)

## Резервирование адресов

	#	IP адрес	Имя устройства	MAC адрес	Включить
<input checked="" type="radio"/>	1	192.168.1.65	MAIN-PC		1

[Подробнее](#)



# DHCP-агент

- Программное обеспечение, играющее роль посредника между DHCP-клиентами и DHCP-серверами (пример такого варианта — сеть 2).
- Связной агент переправляет запросы клиентов из сети 2 DHCP-серверу сети 3.
- Таким образом, один DHCP-сервер может обслуживать DHCP-клиентов нескольких разных сетей.

# Порядок работы DHCP

1. Когда компьютер включают, установленный на нем DHCP-клиент посылает ограниченное широковещательное сообщение DHCP-поиска (IP-пакет с адресом назначения, состоящим из одних единиц, который должен быть доставлен всем узлам данной IP сети).
2. Находящиеся в сети DHCP-серверы получают это сообщение. Если в сети DHCP-серверы отсутствуют, то сообщение DHCP-поиска получает связной DHCP-агент. Он пересылает это сообщение в другую, возможно, значительно отстоящую от него сеть DHCP-серверу, IP-адрес которого ему заранее известен.

# Порядок работы DHCP

3. Все DHCP-серверы, получившие сообщение DHCP-поиска, посылают DHCP-клиенту, обратившемуся с запросом, свои DHCP-предложения. Каждое предложение содержит IP-адрес и другую конфигурационную информацию. (DHCP-сервер, находящийся в другой сети, посылает ответ через агента.)
4. DHCP-клиент собирает конфигурационные DHCP-предложения от всех DHCP-серверов. Как правило, он выбирает первое из поступивших предложений и отправляет в сеть широковещательный DHCP-запрос. В этом запросе содержатся идентификационная информация о DHCP-сервере, предложение которого принято, а также значения принятых конфигурационных параметров.

# Порядок работы DHCP

5. Все DHCP-серверы получают DHCP-запрос и только один выбранный DHCP-сервер посылает положительную DHCP-квитанцию (подтверждение IP-адреса и параметров аренды), а остальные серверы аннулируют свои предложения, в частности, возвращают в свои пулы предложенные адреса.
6. DHCP-клиент получает положительную DHCP-квитанцию и переходит в рабочее состояние.

# Проблемы динамического назначения

- Возникают сложности при преобразовании символьного доменного имени в IP-адрес
- Трудно осуществлять удаленное управление и автоматический мониторинг интерфейса (например, сбор статистики), если в качестве его идентификатора выступает динамически изменяемый IP-адрес
- Усложняется фильтрация пакетов по IP-адресам

# HTTP

- Служит для передачи гипертекстовой информации
- Главный протокол всемирной паутины (www)
- Существует несколько версий этого протокола: HTTP 1.0, HTTP 1.1, HTTP/2 и HTTP/3
- Обмен сообщениями идет по обычной схеме «запрос-ответ». Клиент и сервер обмениваются текстовыми сообщениями стандартного формата, то есть каждое сообщение представляет собой несколько строк обычного текста в кодировке ASCII.
- Для транспортировки HTTP-сообщений служит протокол TCP.

HTTP определяет порядок того как веб-клиенты запрашивают веб-страницы с веб-сервера и как сервер передает эти страницы клиентам.



# Постоянные и непостоянные соединения

- При отправке каждой пары запрос-ответ через отдельное соединение говорят, что используются кратковременные, или **непостоянные соединения**;
- При отправке каждой пары через одно и то же TCP соединение — долговременные, или **постоянные соединения**.



# Непостоянное соединение

- Допустим, есть адрес: `http://www.bmstu.ru/home.index`
- 1) HTTP-клиент инициирует TCP-соединение с сервером `www.bmstu.ru` по порту 80 (порт по умолчанию для HTTP). Этому TCP-соединению выделяются сокеты на клиентской и серверной стороне.
  - 2) HTTP-клиент отправляет запрос серверу через свой сокет. Запрос включает путь к базовому файлу `/home.index`

# Непостоянное соединение

- 3) Процесс HTTP-сервера получает запрос через свой сокет, извлекает объект `/home.index` из своего места хранения (оперативной памяти или диска), помещает объект в ответное HTTP-сообщение и отправляет клиенту через свой сокет.
- 4) Процесс HTTP-сервера дает команду протоколу TCP закрыть соединение (на самом деле, TCP-соединение не разрывается до тех пор, пока сервер не получит информацию об успешном получении ответа клиентом).

# Непостоянное соединение

- 5) HTTP-клиент получает ответ от сервера, и TCP-соединение разрывается. Сообщение указывает, что полученный объект — это HTML-файл. Клиент извлекает файл из сообщения, обрабатывает его и находит ссылки на 10 объектов (файлов в формате JPEG).
- 6) Шаги с первого по четвертый повторяются для каждого из десяти JPEG-объектов.

# Непостоянное соединение

- Когда браузер получает веб-страницу, он отображает ее на экране.
- Два различных браузера могут интерпретировать веб- страницу по-разному.
- Спецификации протокола HTTP (RFC 1945 и RFC 2616) определяют только протокол взаимодействия между программой клиента и программой сервера, но ничего не говорят о том, как вебстраница должна интерпретироваться клиентом.

# Непостоянное соединение

- На самом деле большинство современных браузеров в режиме по умолчанию открывают от пяти до десяти параллельных TCP-соединений, и каждое из них обрабатывает одну транзакцию из запроса и ответа, а степень этого параллелизма может быть сконфигурирована пользователем.
- Если тот пожелает, число параллельных соединений можно установить равным единице, и в этом случае 10 соединений будут устанавливаться последовательно.

# Постоянное соединение

- В случае с постоянным соединением сервер после отправки ответа клиенту оставляет TCP-соединение открытым.
- Через одно и то же соединение можно отправить последовательность запросов и ответов между одним и тем же клиентом и сервером.
- В частности, одно постоянное TCP-соединение позволяет передать всю веб-страницу (в примере выше это базовый HTML-файл и десять изображений).

# Постоянное соединение

- Через одно постоянное соединение можно отправить одному и тому же клиенту много веб-страниц, размещенных на том же сервере.
- Эти запросы объектов могут быть сделаны один за другим, без ожидания ответов на обрабатываемый запрос (так называемая конвейеризация).
- Обычно HTTP-сервер закрывает соединение, когда оно не используется в течение определенного времени (настраиваемый интервал тайм-аута).
- Когда сервер получает последовательные запросы, он отправляет объекты также один за другим. По умолчанию HTTP использует постоянное соединение с конвейеризацией.

# HTTP 1.0

- Поддерживается только режим кратковременных соединений, когда после передачи одного запроса и получения ответа TCP-соединение закрывается.
- Такой режим полностью соответствует концепции сервера без сохранения состояния, а это, как уже отмечалось, приводит к замедлению работы браузера и увеличению трафика из-за частого выполнения процедуры трехэтапного установления TCP-соединения.



# HTTP1.1 (RFC 2616)

- По умолчанию применяются постоянные соединения и конвейерный режим.
- Соединение разрывается по инициативе либо браузера, либо сервера за счет отправки специального токена разрыва соединения в HTTP-пакете.
- Веб-сервер обычно использует таймер неактивности пользователя для того, чтобы разорвать соединение по тайм-ауту и не тратить ресурсы памяти на неактивные соединения.

# HTTP/2 (RFC 7540)

- Вместо использования отдельных TCP-соединений для передачи каждого запроса и ответа, приводившего к простоям из-за того, что новый запрос не может быть послан без получения ответа, теперь используется одно TCP-соединение для мультиплексирования нескольких запросов, которые могут быть посланы практически одновременно;
- Приоритезация запросов к веб-серверу, благодаря которой сервер знает, какой запрос более важен веб-браузеру для построения страницы
- Введение режима Server Push, при котором веб-сервер может передать веб-браузеру не только запрашиваемые ресурсы, но и те, которые, по мнению веб-сервера, скоро понадобятся веб-браузеру
- Компрессия заголовков сообщений HTTP, значительно сокращающая длину сообщения за счет компрессии таких потенциально длинных полей, как куки

# HTTP/3

- На текущий момент не стандартизирован
- Заменяет протокол TCP на новый протокол QUIC, являющийся транспортным протоколом, работающим поверх UDP, и более быстро, чем TCP, устанавливающая соединения и обрабатывающая потерю и искажения данных.
- Протокол QUIC первоначально был разработан компанией Google и уже применяется в браузере Chrome этой компании.

# Формат сообщения-запроса



# Формат сообщения-ответа



# Методы

- Метод HEAD аналогичен методу GET, но запрашиваются только метаданные заголовка HTML-страницы.
- Метод POST используется клиентом для отправки данных на сервер: сообщений электронной почты, ключевых слов в запросе поиска, веб-формы.
- Метод PUT используется клиентом для размещения некоторого объекта на сервере, на который указывает URL-адрес.

# Методы

- Метод DELETE указывает серверу на то, что некоторый объект на сервере, определяемый URL-адресом, необходимо удалить.
- Методы GET и HEAD считаются безопасными<sup>1</sup> для сервера, так как они только передают информацию клиенту, а методы POST, PUT и DELETE — опасными, поскольку передают информацию на сервер.
- Наибольшую угрозу представляют два последних метода, так как они непосредственно указывают на объект на сервере. Используя эти методы, злоумышленник может атаковать сервер, заменяя или удаляя некоторые его объекты.

# Форматы стартовых строк и заголовков

Обобщенная структура сообщения	HTTP-запрос	HTTP-ответ
Стартовая строка (всегда должна быть первой строкой сообщения; обязательный элемент)	Формат запроса Метод/ URL HTTP/1.x. Пример: GET /books/books.htm HTTP/1.1	Формат ответа: HTTP/1.x КодСостояния Фраза. Пример: HTTP/1.1 200 OK
Заголовки (следуют в произвольном порядке; могут отсутствовать)	Заголовок о DNS-имени компьютера, на котором расположен веб-сервер. Пример: Host: www.olifer.co.uk	Заголовок о времени отправления данного ответа. Пример: Date: 1 Jan 2009 14:00:30
	Заголовок об используемом браузере. Пример: User-agent: Mozilla/5.0	Заголовок об используемом веб-сервере. Пример: Server: Apache/1.3.0 (Unix)
	Заголовок о предпочтительном языке. Пример: Accept-language: ru	Заголовок о количестве байтов в теле сообщения. Пример: Content-Length: 1234
	Заголовок о режиме соединения. Пример: Connection: close	Заголовок о режиме соединения. Пример: Connection: close
Пустая строка		
Тело сообщения (может отсутствовать)	Здесь могут быть расположены ключевые слова для поисковой машины или страницы для передачи на сервер	Здесь может быть расположен текст запрашиваемой страницы



# Классы кодов состояний

- 1xx — информация о процессе передачи;
- 2xx — информация об успешном принятии и обработке запроса клиента (в таблице в примере стартовой строки ответа приведен код и соответствующая фраза 200 OK, сообщаящий клиенту, что его запрос успешно обработан);
- 3xx — информация о том, что для успешного выполнения операции нужно произвести следующий запрос по другому URL-адресу, указанному в дополнительном заголовке Location;

# Классы кодов состояний

- 4xx — информация об ошибках со стороны клиента (при указании адреса несуществующей страницы браузер выводит на экран сообщение 404 Not Found);
- 5xx — информация о неуспешном выполнении операции по вине сервера (например, сообщение 505 http Version Not Supported говорит о том, что сервер не поддерживает версию HTTP, предложенную клиентом).

# Cookie

- Определенный в документе RFC 6265562 механизм сохранения данных для идентификации пользователя
- Позволяет веб-сайтам отслеживать состояние пользовательского соединения.
- Подавляющее большинство коммерческих веб-сайтов используют данный механизм.

# Принцип работы

- Пользователь А посещает сайт, используя браузер
- Когда запрос приходит на веб-сервер Amazon, он создает уникальный идентификационный номер, а также запись в своей базе данных, которая индексируется этим идентификационным номером.
- Затем вебсервер Amazon посылает ответ браузеру Сьюзен, включающий в HTTPсообщение заголовок Set-cookie:, и в нем содержится соответствующий идентификационный номер.

# Принцип работы

- Когда браузер получает ответное HTTP-сообщение, он видит заголовок Set-cookie: и добавляет строку в специальный cookie-файл (имя сервера и идентификационный номер из заголовка Set-cookie)
- Каждый раз, когда пользователь запрашивает веб-страницу, браузер обращается к своему cookie-файлу, извлекает идентификационный номер этого сайта и помещает строку cookie-заголовка, включающую идентификационный номер, в HTTP запрос.
- Сайту точно известно, какие страницы посетил пользователь, в каком порядке и сколько раз

# Прокси-сервер



# Прокси-сервер

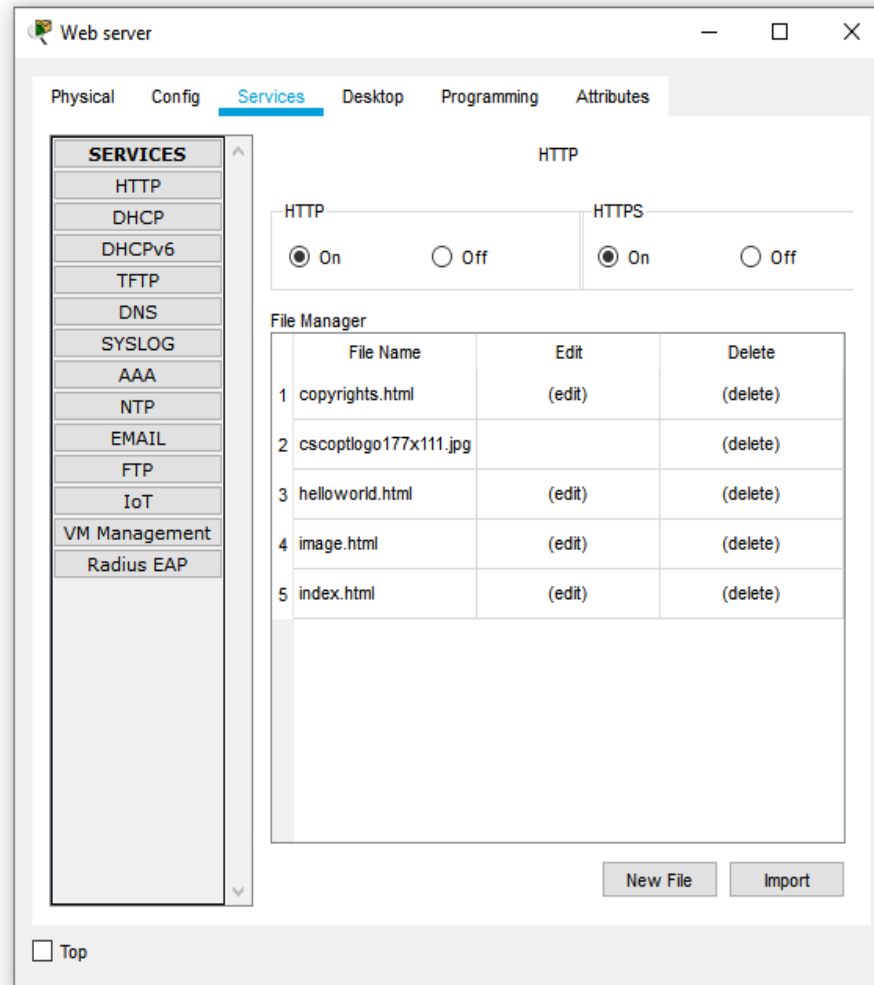
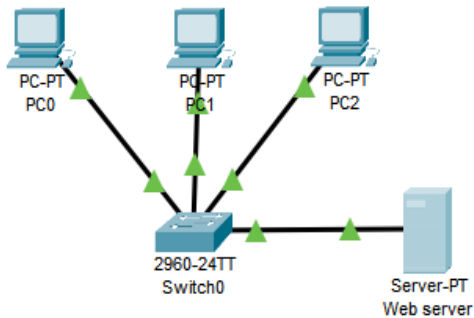
- Веб-кэш, также называемый прокси-сервером — это элемент сети, который обрабатывает HTTP-запрос в дополнение к «настоящему» вебсерверу.
- Для этого на прокси-сервере имеется собственное дисковое хранилище, куда помещаются копии недавно запрошенных объектов.

# Прокси-сервер

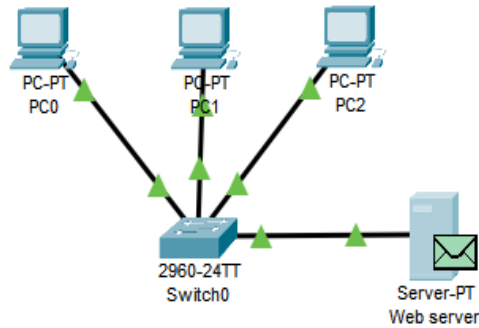
- позволяет уменьшить время ответа на запрос клиента, особенно если полоса пропускания между клиентом и вебсервером намного меньше, чем между клиентом и прокси-сервером.  
(высокоскоростное соединение, поэтому прокси-сервер способен доставить объект клиенту очень быстро)
- может уменьшить трафик в сети доступа организации, позволяет снизить расходы и положительно сказывается на производительности приложений, использующих сеть



# Пример настройки в Packet Tracer



# Пример настройки в Packet Tracer



PDU Information at Device: PC0

OSI Model   Outbound PDU Details

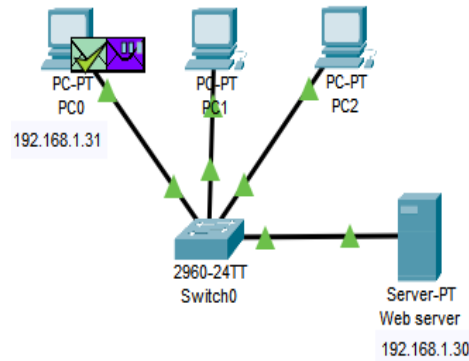
At Device: PC0  
Source: PC0  
Destination: 192.168.1.30

In Layers	Out Layers
Layer7	Layer 7:
Layer6	Layer6
Layer5	Layer5
Layer4	Layer 4: TCP Src Port: 1037, Dst Port: 80
Layer3	Layer 3: IP Header Src. IP: 192.168.1.31, Dest. IP: 192.168.1.30
Layer2	Layer 2: Ethernet II Header 0001.C998.3913 >> 0001.C728.72D4
Layer1	Layer 1: Port(s): FastEthernet0

1. The HTTP client makes a connection to the server.

Challenge Me   << Previous Layer   Next Layer >>

# Пример настройки в Packet Tracer



PDU Information at Device: PC0

OSI Model Outbound PDU Details

At Device: PC0  
Source: PC0  
Destination: HTTP CLIENT

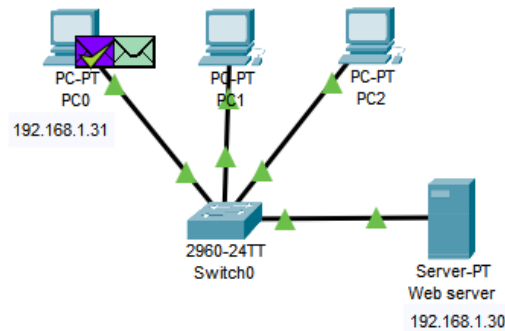
In Layers	Out Layers
Layer7	Layer 7:
Layer6	Layer6
Layer5	Layer5
Layer4	Layer 4: TCP Src Port: 1037, Dst Port: 80
Layer3	Layer 3: IP Header Src. IP: 192.168.1.31, Dst. IP: 192.168.1.30
Layer2	Layer 2: Ethernet II Header 0001.C998.3913 >> 0001.C728.72D4
Layer1	Layer 1: Port(s):

1. The HTTP client sends a HTTP request to the server.

Challenge Me << Previous Layer Next Layer >>

Vis.	Time(sec)	Last Device	At Device	Type
	0.000	--	PC0	TCP
	0.001	PC0	Switch0	TCP
	0.002	Switch0	Web server	TCP
	0.003	Web server	Switch0	TCP
	0.004	Switch0	PC0	TCP
	0.004	--	PC0	HTTP

# Пример настройки в Packet Tracer



PDU Information at Device: PC0

OSI Model [Inbound PDU Details](#)

PDU Formats

DST IP:192.168.1.31	
OPT:0x00000000	PADDING:0x00
DATA (VARIABLE LENGTH)	

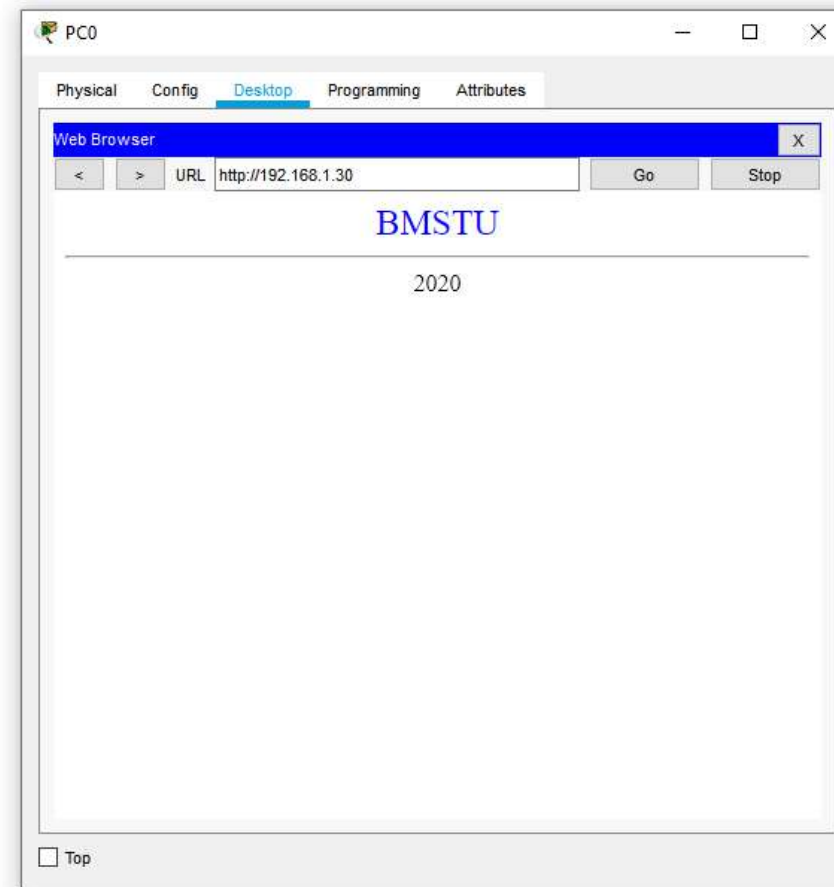
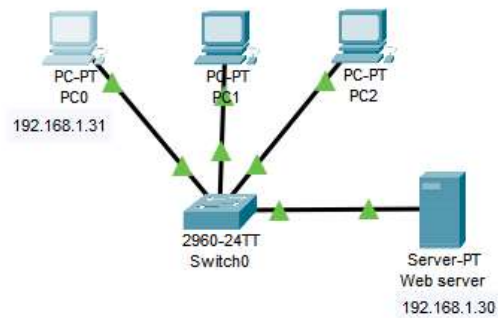
TCP

SOURCE PORT:80		DESTINATION PORT:1037	
SEQUENCE NUMBER:1			
ACKNOWLEDGEMENT NUMBER:102			
OFFS ET:0x	RESERVED : 0b000000	FLAGS:0b 011000	WINDOW:16384
CHECKSUM:0x0000		URGENT POINTER:0x0000	
OPTION		URGENT POINTER: 0x0000	
DATA (VARIABLE LENGTH)		PADDING: 0b000 ...000	

HTTP RESPONSE

HTTP Data:Connection: close Content-Length: 369	
--	--

# Пример настройки в Packet Tracer



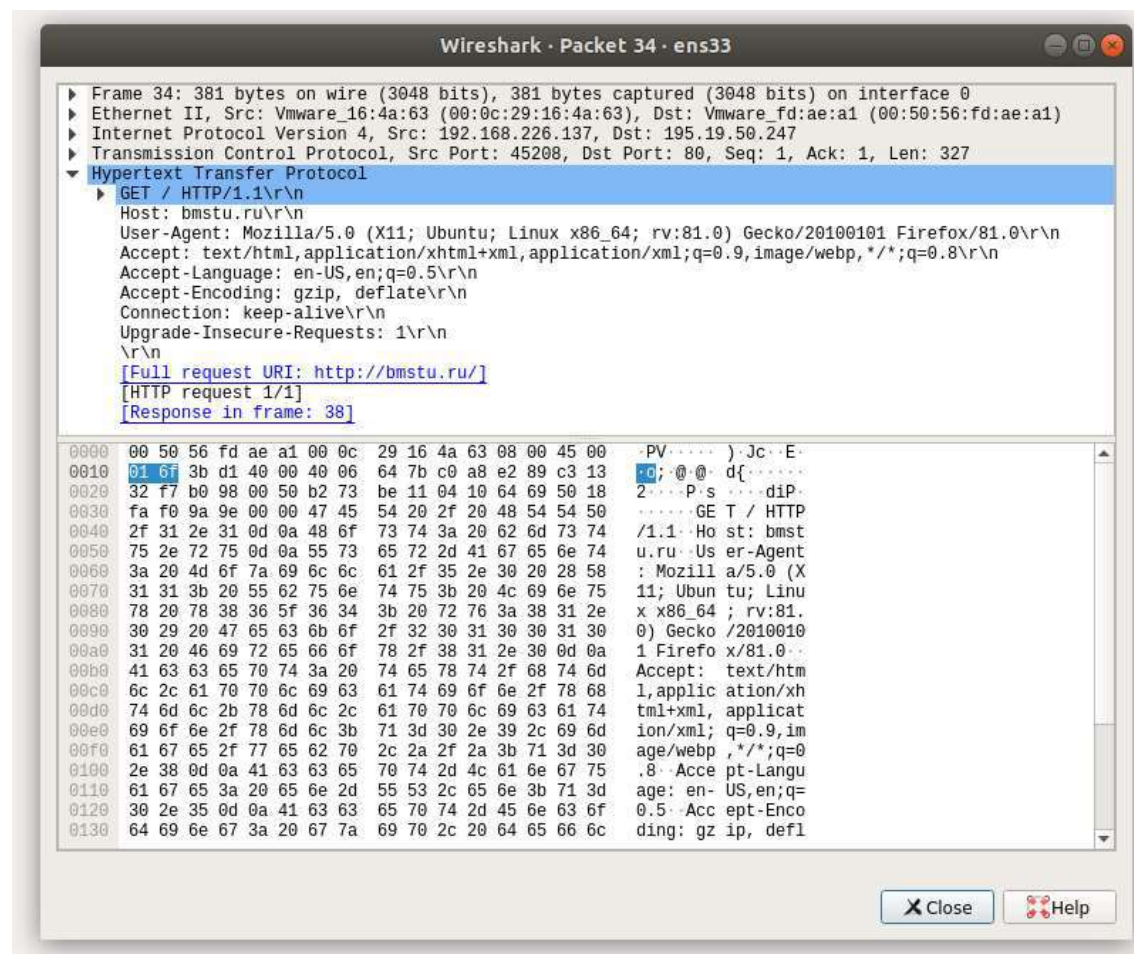
# Лекция VI. Протоколы прикладного уровня, часть II

Рогозин Н.О., каф. ИУ-7

# HTTPS (RFC 2818)

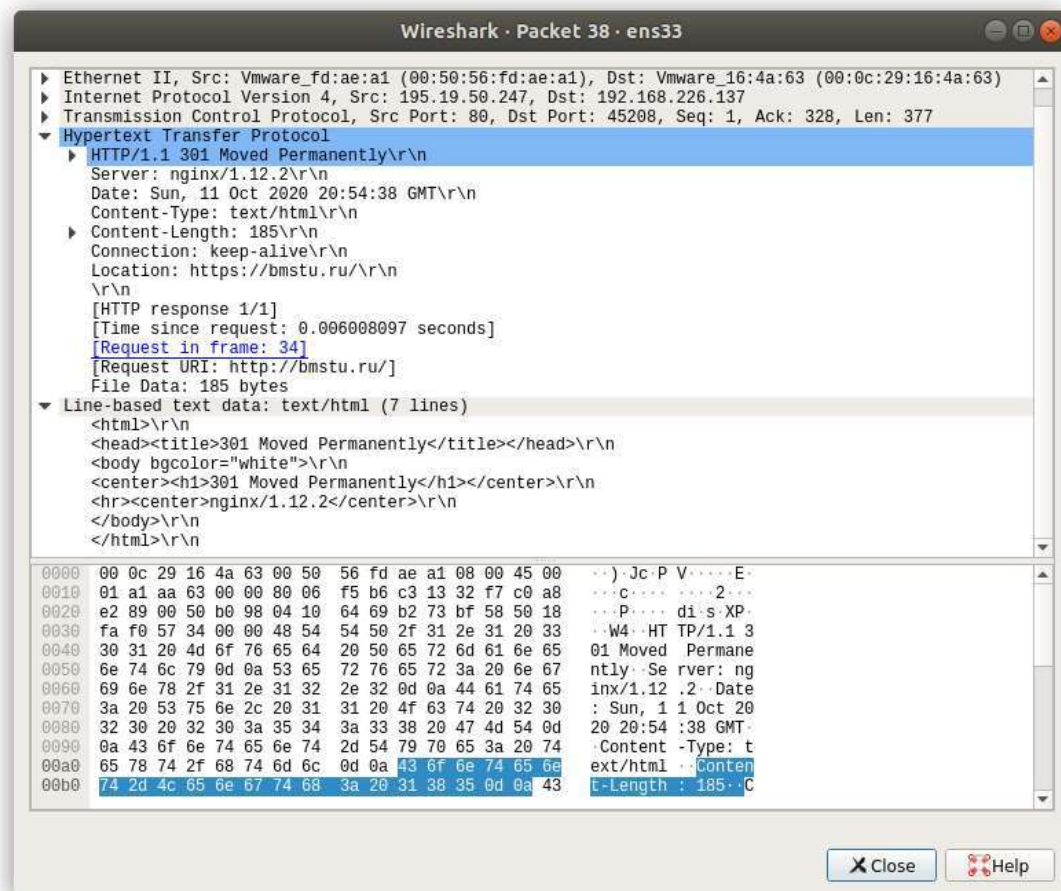
- По умолчанию HTTPS URL использует 443 TCP-порт (для незащищённого HTTP — 80).
- Чтобы подготовить веб-сервер для обработки https-соединений, администратор должен получить и установить в систему сертификат открытого и закрытого ключа для этого веб-сервера.
- В TLS используется как асимметричная схема шифрования (для выработки общего секретного ключа), так и симметричная (для обмена данными, зашифрованными общим ключом).

# Первоначальный http-запрос





# http-ответ



# Обмен сообщениями TLS

The image shows a Wireshark packet capture of a TLS handshake and an HTTP 301 redirect. The capture is filtered by `ip.dst == 195.19.50.247 || ip.src == 195.19.50.247`. The packet list shows 125 packets, with the 38th packet being the HTTP 301 redirect.

No.	Time	Source	Destination	Protocol	Length	Info
30	11.756531879	192.168.226.137	195.19.50.247	TCP	74	45208 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=2504779527 TSecr=0 WS=128
31	11.756574871	192.168.226.137	195.19.50.247	TCP	74	45210 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=2504779527 TSecr=0 WS=128
32	11.761754857	195.19.50.247	192.168.226.137	TCP	60	80 → 45208 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
33	11.761771685	192.168.226.137	195.19.50.247	TCP	54	45208 → 80 [ACK] Seq=1 Ack=1 Win=64240 Len=0
34	11.762143400	192.168.226.137	195.19.50.247	HTTP	381	GET / HTTP/1.1
35	11.762250540	195.19.50.247	192.168.226.137	TCP	60	80 → 45208 [ACK] Seq=1 Ack=328 Win=64240 Len=0
36	11.762468452	195.19.50.247	192.168.226.137	TCP	60	80 → 45210 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
37	11.762477385	192.168.226.137	195.19.50.247	TCP	54	45210 → 80 [ACK] Seq=1 Ack=1 Win=64240 Len=0
38	11.768151497	195.19.50.247	192.168.226.137	HTTP	431	HTTP/1.1 301 Moved Permanently (text/html)
39	11.768164913	192.168.226.137	195.19.50.247	TCP	54	45208 → 80 [ACK] Seq=328 Ack=378 Win=63863 Len=0
40	11.774084982	192.168.226.137	195.19.50.247	TCP	74	53682 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=2504779545 TSecr=0 WS=128
41	11.779079102	195.19.50.247	192.168.226.137	TCP	60	443 → 53682 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
42	11.779098707	192.168.226.137	195.19.50.247	TCP	54	53682 → 443 [ACK] Seq=1 Ack=1 Win=64240 Len=0
43	11.780246949	192.168.226.137	195.19.50.247	TLSv1.2	571	Client Hello
44	11.780382225	195.19.50.247	192.168.226.137	TCP	60	443 → 53682 [ACK] Seq=1 Ack=518 Win=64240 Len=0
45	11.787879673	195.19.50.247	192.168.226.137	TLSv1.2	3903	Server Hello, Certificate, Server Key Exchange, Server Hello Done
46	11.787896026	192.168.226.137	195.19.50.247	TCP	54	53682 → 443 [ACK] Seq=518 Ack=3850 Win=61320 Len=0
47	11.789869069	192.168.226.137	195.19.50.247	TLSv1.2	180	Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
48	11.790009768	195.19.50.247	192.168.226.137	TCP	60	443 → 53682 [ACK] Seq=3850 Ack=644 Win=64240 Len=0
53	11.796056167	195.19.50.247	192.168.226.137	TLSv1.2	312	New Session Ticket, Change Cipher Spec, Encrypted Handshake Message
54	11.796076746	192.168.226.137	195.19.50.247	TCP	54	53682 → 443 [ACK] Seq=644 Ack=4108 Win=62780 Len=0
66	11.861080848	192.168.226.137	195.19.50.247	TLSv1.2	414	Application Data
67	11.861221597	195.19.50.247	192.168.226.137	TCP	60	443 → 53682 [ACK] Seq=4108 Ack=1004 Win=64240 Len=0
68	11.875876511	195.19.50.247	192.168.226.137	TLSv1.2	14284	Application Data
69	11.875897074	192.168.226.137	195.19.50.247	TCP	54	53682 → 443 [ACK] Seq=1004 Ack=18338 Win=55480 Len=0
119	12.155774014	192.168.226.137	195.19.50.247	TLSv1.2	401	Application Data
120	12.155914924	195.19.50.247	192.168.226.137	TCP	60	443 → 53682 [ACK] Seq=18338 Ack=1351 Win=64240 Len=0
121	12.156077766	192.168.226.137	195.19.50.247	TCP	74	53690 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=2504779927 TSecr=0 WS=128
122	12.156259786	192.168.226.137	195.19.50.247	TCP	74	53692 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=2504779927 TSecr=0 WS=128
123	12.156445971	192.168.226.137	195.19.50.247	TCP	74	53694 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=2504779927 TSecr=0 WS=128
124	12.156640062	192.168.226.137	195.19.50.247	TCP	74	53696 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=2504779927 TSecr=0 WS=128
125	12.156819925	192.168.226.137	195.19.50.247	TCP	74	53698 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=2504779928 TSecr=0 WS=128

Frame 38: 431 bytes on wire (3448 bits), 431 bytes captured (3448 bits) on interface 0  
Ethernet II, Src: Vmware fd:ae:a1 (00:50:56:fd:ae:a1), Dst: Vmware 16:4a:63 (00:0c:29:16:4a:63)  
Internet Protocol Version 4, Src: 195.19.50.247, Dst: 192.168.226.137  
Transmission Control Protocol, Src Port: 80, Dst Port: 45208, Seq: 1, Ack: 328, Len: 377  
Hypertext Transfer Protocol

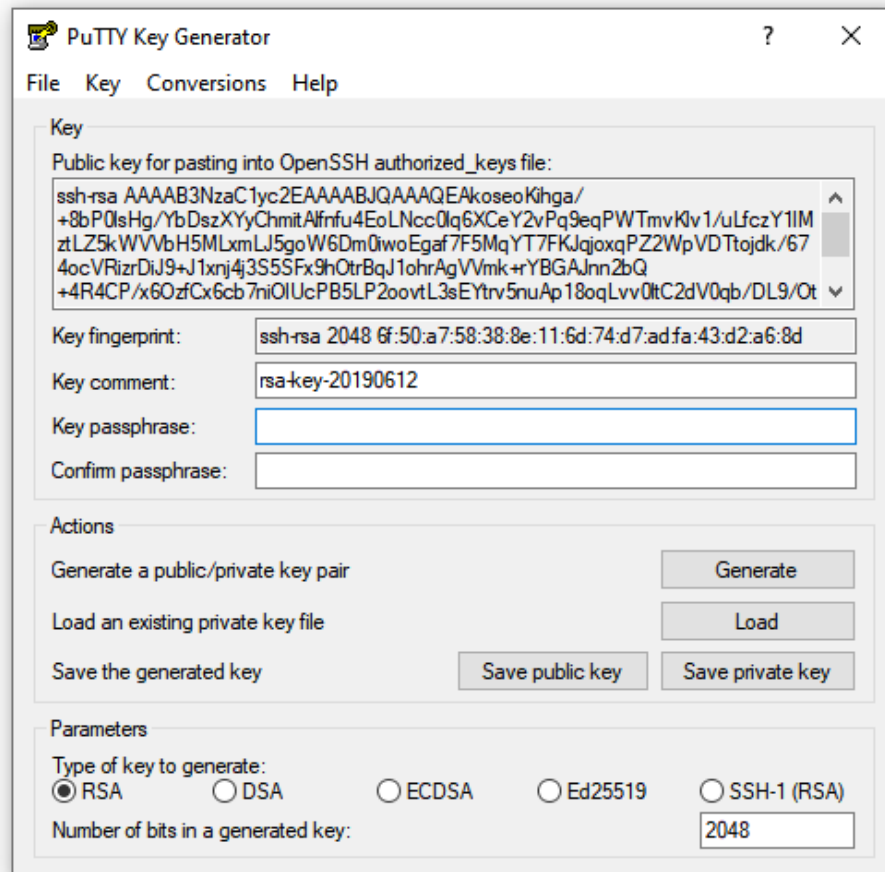
0000 00 0c 29 16 4a 63 00 50 56 fd ae a1 08 00 45 00 ... Jc P V ... E  
0010 01 a1 aa 63 00 00 80 06 f5 b6 c3 13 32 f7 c0 a8 ... c ... 2 ...  
0020 e2 89 00 50 b0 98 04 10 64 69 b2 73 bf 58 50 18 ... P ... di s XP  
0030 fa f0 57 34 00 00 48 54 54 50 2f 31 2e 31 20 33 ... W4 HT TP/1.1 3  
0040 30 31 20 4d 6f 76 65 64 20 50 65 72 6d 61 6e 65 01 Moved Permanently  
0050 6e 74 6c 79 0d 0a 53 65 72 76 65 72 3a 20 6e 67 ntly Se rver: ng  
0060 69 6e 78 2f 31 2e 31 32 2e 32 0d 0a 44 61 74 65 inx/1.12 .2 Date  
0070 3a 20 53 75 6e 2c 20 31 31 20 4f 63 74 20 32 60 : Sun, 1 1 Oct 20

wireshark\_ens33\_20201011135425\_u0a2Md.pcapng Packets: 2883 · Displayed: 1840 (63.8%) · Dropped: 0 (0.0%) Profile: Default

# SSH

- Как и telnet, передает набираемые на терминале пользователя символы на удаленный узел без интерпретации их содержания.
- Предусматривает меры по защите передаваемых аутентификационных и пользовательских данных.
- Поддерживает симметричное, асимметричное шифрование и хеширование (обсуждается в лекции, посвященной безопасности сетевых соединений)

# Генерация ключа в PuttyGen



The screenshot shows the PuTTY Key Generator window. The 'Key' section displays a public key for pasting into an OpenSSH authorized\_keys file. The key fingerprint is shown as 'ssh-rsa 2048 6f:50:a7:58:38:8e:11:6d:74:d7:ad:fa:43:d2:a6:8d'. The key comment is 'rsa-key-20190612'. The 'Actions' section includes buttons for 'Generate', 'Load', 'Save public key', and 'Save private key'. The 'Parameters' section shows 'Type of key to generate' set to 'RSA' and 'Number of bits in a generated key' set to '2048'.

PuTTY Key Generator

File Key Conversions Help

Key

Public key for pasting into OpenSSH authorized\_keys file:

```
ssh-rsa AAAAB3NzaC1yc2EAAAABJQAAQEAkoseoKihga/  
+8bP0lsHg/YbDszXYyChmitAlfnfu4EoLNcc0lq6XCeY2vPq9eqPWtmvKlv1/uLfczY1IM  
ztLZ5kVVVbH5MLxmLJ5goW6Dm0iwoEgaf7F5MqYT7FKJqioxqPZ2WpVDTtojdk/67  
4ocVRizrDiJ9+J1xnj4j3S5SFx9hOtrBqJ1ohrAgVVmk+rYBGAJnn2bQ  
+4R4CP/x6OzfCx6cb7niOIUcPB5LP2oovtL3sEYtrv5nuAp18oqLvv0ltC2dV0qb/DL9/Ot
```

Key fingerprint: ssh-rsa 2048 6f:50:a7:58:38:8e:11:6d:74:d7:ad:fa:43:d2:a6:8d

Key comment: rsa-key-20190612

Key passphrase:

Confirm passphrase:

Actions

Generate a public/private key pair Generate

Load an existing private key file Load

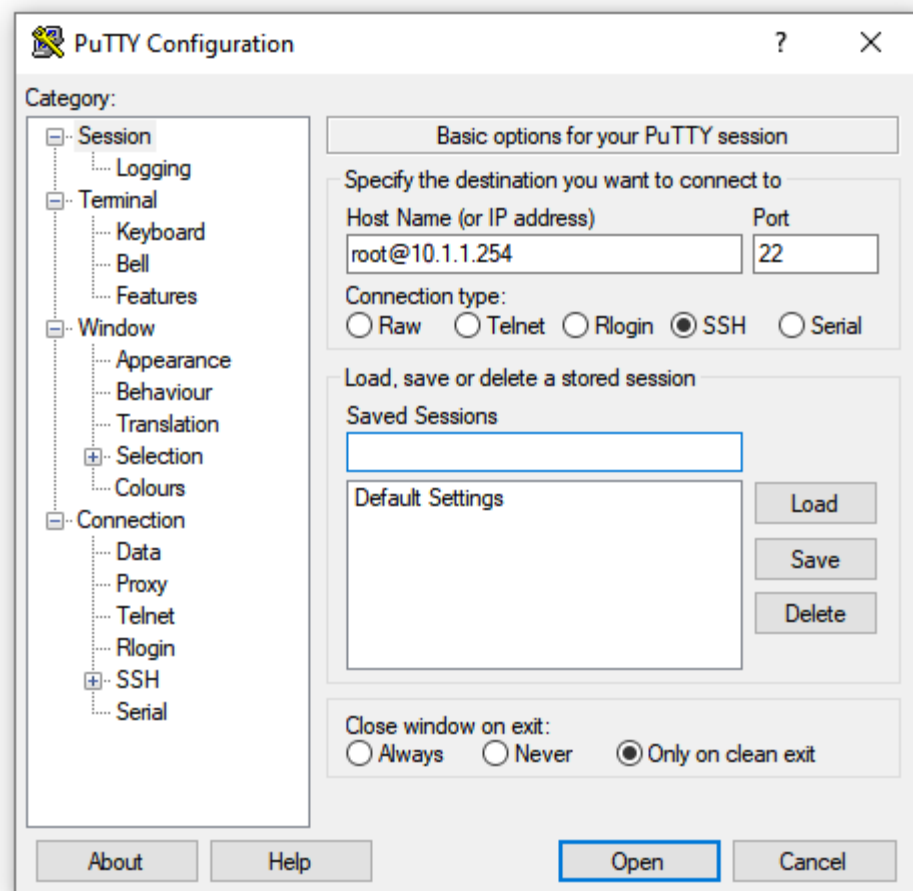
Save the generated key Save public key Save private key

Parameters

Type of key to generate:  
☒ RSA ☐ DSA ☐ ECDSA ☐ Ed25519 ☐ SSH-1 (RSA)

Number of bits in a generated key: 2048

# Выполнение доступа



# FTP

- Один из старейших протоколов в Internet и входит в его стандарты.
- Обмен данными в FTP проходит по TCP-каналу.
- Построен обмен по технологии "клиент-сервер".
- Пользователь FTP может вызывать несколько команд, которые позволяют ему посмотреть каталог удаленной машины, перейти из одного каталога в другой, а также скопировать один или несколько файлов.



# FTP

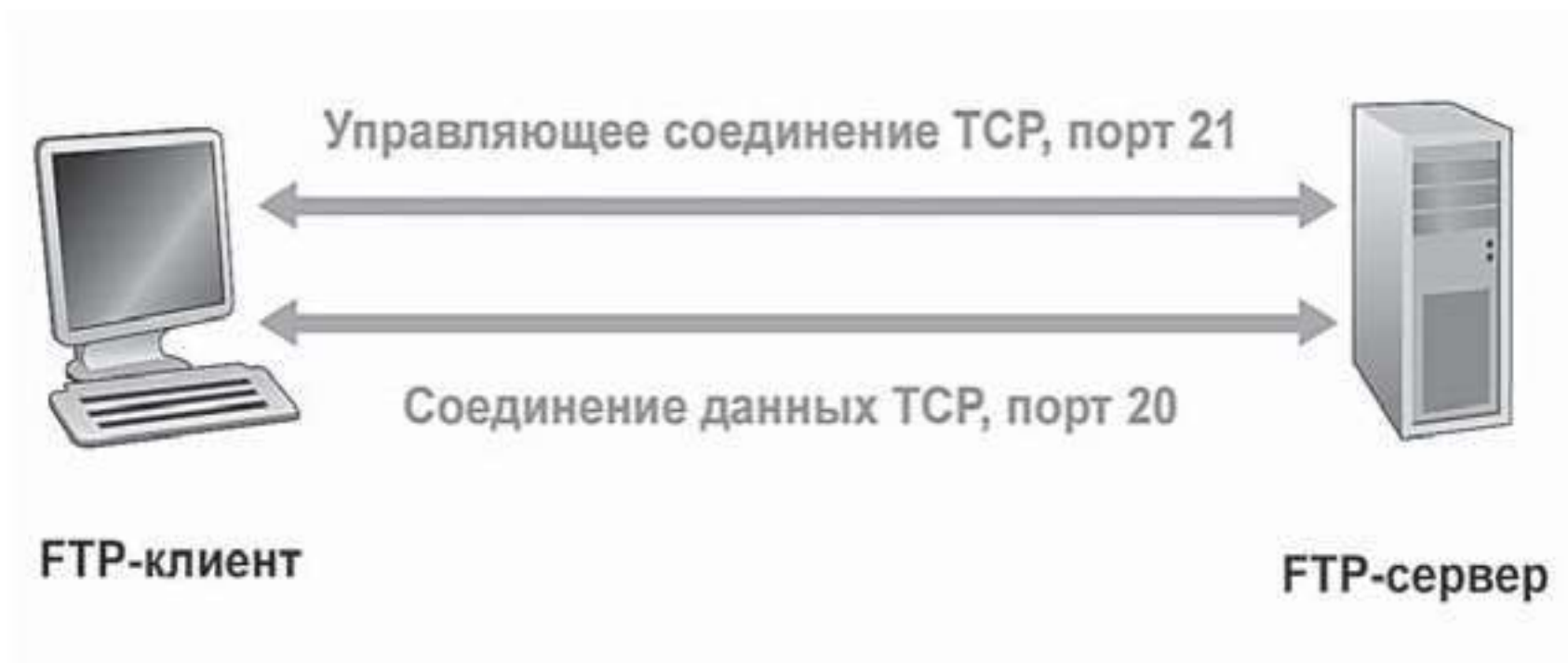
- FTP использует два TCP соединения для передачи файла.
- Управляющее соединение устанавливается как обычное соединение клиент-сервер.
- Сервер осуществляет пассивное открытие на заранее известный порт FTP (21) и ожидает запроса на соединение от клиента.
- Клиент осуществляет активное открытие на TCP порт 21, чтобы установить управляющее соединение.
- Управляющее соединение существует все время, пока клиент общается с сервером.

# FTP

- Это соединение используется для передачи команд от клиента к серверу и для передачи откликов от сервера.
- Тип IP сервиса для управляющего соединения устанавливается для получения "минимальной задержки", так как команды обычно вводятся пользователем .
- Соединение данных открывается каждый раз, когда осуществляется передача файла между клиентом и сервером.
- Тип сервиса IP для соединения данных должен быть "максимальная пропускная способность", так как это соединение используется для передачи файлов.



# FTP



# FTP, Представление данных

- **ASCII файлы.** (По умолчанию) Текстовый файл передается по соединению данных как NVT ASCII.
- При этом требуется, чтобы отправитель конвертировал локальный текстовый файл в NVT ASCII, а получатель конвертировал NVT ASCII в текстовый файл.
- Конец каждой строки передается в виде NVT ASCII символа возврата каретки, после чего следует перевод строки. Это означает, что получатель должен просматривать каждый байт в поисках пары символов CR, LF.

# FTP, Представление данных

- **EBCDIC файлы.**

Альтернативный способ передачи текстовых файлов, когда на обоих концах системы EBCDIC.

- **Двоичные или бинарные файлы.**

Данные передаются как непрерывный поток битов.

- **Локальный тип файлов.**

Способ передачи бинарных файлов между хостами, которые имеют различный размер байта. Количество битов в байте определяется отправителем. Для систем, которые используют 8-битные байты, локальный тип файла с размером байта равным 8 эквивалентен бинарному типу файла.

# FTR, управление форматом

- **Nonprint. (По умолчанию)**  
Файл не содержит информацию вертикального формата.
- **Telnet format control.**  
Файл содержит управляющие символы вертикального формата Telnet, которые интерпретируются принтером.
- **Fortran carriage control.**  
Первый символ каждой строки это Fortran символ управления формата.

# FTP, структура

- **Структура файла.**  
(По умолчанию) Файл воспринимается в виде непрерывного потока байтов. Файл не имеет внутренней структуры.
- **Структура записи.**  
Эта структура используется только в случае текстовых файлов (ASCII или EBCDIC).
- **Структура страницы.**  
Каждая страница передается с номером страницы, что позволяет получателю хранить страницы в случайном порядке. Предоставляется операционной системой TOPS-20. (Требование к хостам Host Requirements RFC не рекомендует использовать эту структуру.)

# FTP, режим передачи

- **Режим потока.**

(По умолчанию) Файл передается как поток байтов. Для файловой структуры конец файла указывает на то, что отправитель закрывает соединение данных. Для структуры записи специальная 2-байтовая последовательность обозначает конец записи и конец файла.

- **Режим блоков.**

Файл передается как последовательность блоков, перед каждым из них стоит один или несколько байт заголовков.

- **Сжатый режим.**

Простое кодирование неоднократно встречающихся повторяющихся байт. В текстовых файлах обычно сжимаются пустые строки или строки из пробелов, а в бинарных строки из нулевых байт. (Этот режим поддерживается редко. Существуют более оптимальные способы сжатия файлов для FTP.)

# Команды FTP

Команда	Описание
ABOR	прервать предыдущую команду FTP и любую передачу данных
LIST список файлов	список файлов или директорий
PASS пароль	пароль на сервере
PORT n1,n2,n3,n4,n5,n6	IP адрес клиента (n1.n2.n3.n4) и порт (n5 x 256 + n6)
QUIT	закрывает соединение на сервере
RETR имя файла	получить (get) файл
STOR имя файла	положить (put) файл
SYST	сервер возвращает тип системы
TYPE тип	указать тип файла: A для ASCII, I для двоичного
USER имя пользователя	имя пользователя на сервере

# Отклики FTP (первая цифра)

Отклик

Описание

- |     |  |
|-----|--|
| 1yz | Положительный предварительный отклик. Действие началось, однако необходимо дождаться еще одного отклика перед отправкой следующей команды.         |
| 2yz | Положительный отклик о завершении. Может быть отправлена новая команда.  |
| 3yz | Положительный промежуточный отклик. Команда принята, однако необходимо отправить еще одну команду.   |
| 4yz | Временный отрицательный отклик о завершении. Требуемое действие не произошло, однако ошибка временная, поэтому команду необходимо повторить позже. |
| 5yz | Постоянный отрицательный отклик о завершении. Команда не была воспринята и повторять ее не стоит.  |



# Отклики FTP (вторая цифра)

x0z	Синтаксическая ошибка.
x1z	Информация.
x2z	Соединения. Отклики имеют отношение либо к управляющему, либо к соединению данных.
x3z	Аутентификация и бюджет. Отклик имеет отношение к логированию или командам, связанным с бюджетом.
x4z	Не определено.
x5z	Состояние файловой системы.

# Пример ftp-клиента

```
from ftplib import FTP
```

```
HOST = 'localhost'
```

```
ftp = FTP(HOST, 'user', 'mypassword')
```

```
ftp.retrlines('LIST')
```

```
with open('README', 'wb') as fp:
```

```
    ftp.retrbinary('RETR README', fp.write)
```

```
ftp.quit()
```

# TFTP

- Простой протокол передачи файлов.
- Как правило, используется при загрузке бездисковых систем (рабочие станции или X терминалы).
- В отличие от протокола передачи файлов FTP, который использует TCP, TFTP использует UDP.
- Это сделано для того, чтобы протокол был как можно проще и меньше. Реализации TFTP (и необходимого UDP, IP и драйвера устройства) могут поместиться в постоянной памяти (ПЗУ).

# TFTP

- Обмен между клиентом и сервером начинается с того, что клиент запрашивает сервер либо прочитать, либо записать файл для клиента. В стандартном варианте загрузки бездисковой системы первый запрос - это запрос на чтение (RRQ).
- Первые 2 байта TFTP сообщения это код операции (opcode). В запросе на чтение (RRQ) и в запросе на запись (WRQ) имя файла (filename) указывает файл на сервере, который клиент хочет либо считать, либо записать.

# TFTP

- Если файл может быть прочитан клиентом, сервер отвечает пакетом данных с номером блока равным 1. Клиент посылает подтверждение (ACK) на номер блока 1.
- Сервер отвечает следующим пакетом данных с номером блока равным 2.
- Клиент подтверждает номер блока 2.
- Это продолжается до тех пор, пока файл не будет передан. Каждый пакет данных содержит 512 байт данных, за исключением последнего пакета, который содержит от 0 до 511 байт данных.
- Когда клиент получает пакет данных, который содержит меньше чем 512 байт, он считает, что получил последний пакет.

# TFTP

- В случае запроса на запись (WRQ) клиент посылает WRQ, указывая имя файла и режим. Если файл может быть записан клиентом, сервер отвечает подтверждением (ACK) с номером блока равным 0. Клиент посылает первые 512 байт файла с номером блока равным 1, сервер отвечает ACK с номером блока равным 1.
- Так как TFTP использует UDP, то именно от TFTP зависит, как будут обработаны потерянные и дублированные пакеты.
- В случае потери пакета, отправитель отрабатывает тайм-аут и осуществляет повторную передачу. (Возможно появление проблемы, называемой "синдромом новичка" которая может возникнуть, если с обеих сторон будет отработан тайм-аут и осуществлена повторная передача.

# Сетевая почтовая служба

- Распределенное приложение, главной функцией которого является предоставление пользователям сети возможности обмениваться электронными сообщениями.
- Обмен почтой с использованием ТСП осуществляется посредством агентов передачи сообщений (MTA - message transfer agent).

# Почтовый клиент / Mail User Agent

- программа, предназначенная для поддержания пользовательского интерфейса (обычно графического), а также для предоставления пользователю широкого набора услуг по подготовке электронных сообщений.
- Outlook, Thunderbird и т.д.



# Программа передачи сообщений / Mail Transfer Agent

- Наиболее распространенные MTA для Unix систем это Sendmail.
- При общении между двумя MTA используется NVT ASCII. Команды посылаются клиентом серверу, а сервер отвечает с помощью цифровых кодов и опциональных текстовых строк (для чтения человеком).

# MIME

- Multipurpose Internet Mail Extensions — многоцелевые расширения почты Интернета
- В заголовке каждой части сообщения имеется также информация о том, каким образом почтовый клиент должен обрабатывать тело части — отображать ее немедленно при открытии сообщения (например, встраивая изображение в текст) или считать это тело вложением (attachment), которое пользователь будет обрабатывать сам.

# Типы данных

- ASCII
- текст в 8-битном формате
- текст не в формате ASCII, преобразованный в ASCII-код
- гипертекст (HTML)
- изображение
- видеоклип
- звуковой файл

# S/MIME

- RFC 1847
- Цифровая подпись (Mutipart/Signed);
- Шифрованное тело (Multipart/Encrypted).

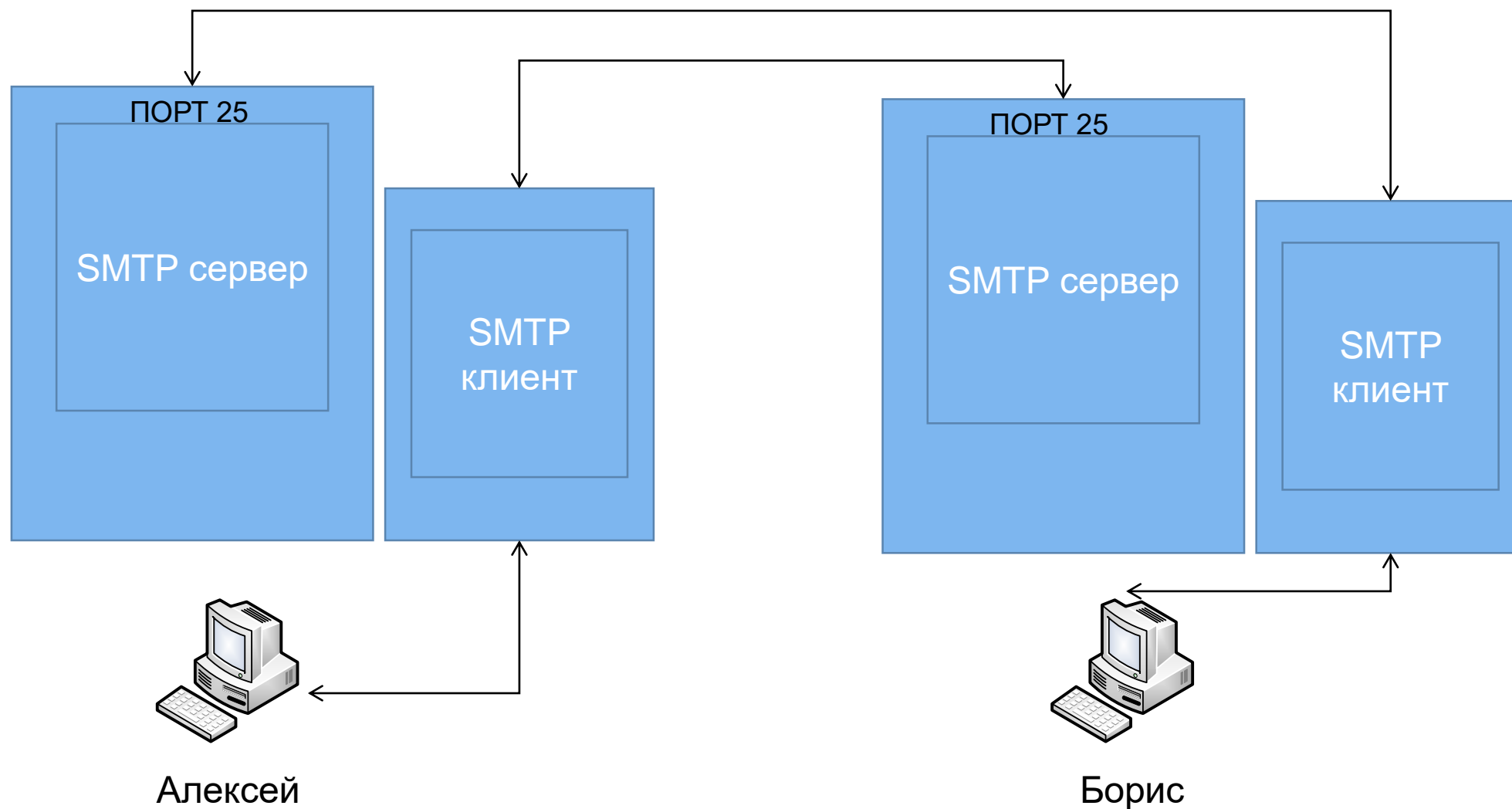
# Состав электронной почты

- **Конверт** используется МТА для доставки.
- Например:
  - MAIL From: <staff@bmstu.ru>
  - RCPT To: <student@bmstu.ru>
- RFC 821 определяет содержимое и интерпретацию конверта, а также протокол, который используется для обмена почтой по TCP соединению.
- **Заголовки** используются пользовательскими агентами. Например: Received, Message-Id, From, Date, Reply-To, X-Phone, X-Mailer, To и Subject. Каждое поле заголовка содержит имя, после которого следует двоеточие, а затем следует значение этого поля. RFC 822 определяет формат и интерпретацию полей заголовка. (Заголовки, начинающиеся с X-, это поля, определяемые пользователем.)
- **Тело** это содержимое сообщения - текстовые строки в формате NVT ASCII.

# SMTP

- Simple Mail Transfer Protocol — простой протокол передачи почты
- Является одним из первых стандартизованных протоколов прикладного уровня -1982
- Реализуется несимметричными взаимодействующими частями: SMTP-клиентом, работающим на стороне отправителя, и SMTP-сервером, работающим на стороне получателя. SMTP-сервер должен постоянно быть в режиме подключения, ожидая запросов со стороны SMTP-клиента.

# Непосредственное взаимодействие



# Процесс передачи сообщения

- Алексей, используя графический интерфейс своего почтового клиента, вызывает функцию создания сообщения, в результате чего на экране появляется стандартная незаполненная форма сообщения
- В поля которой Алексей вписывает свой адрес, адрес Бориса и тему письма, а затем набирает текст письма



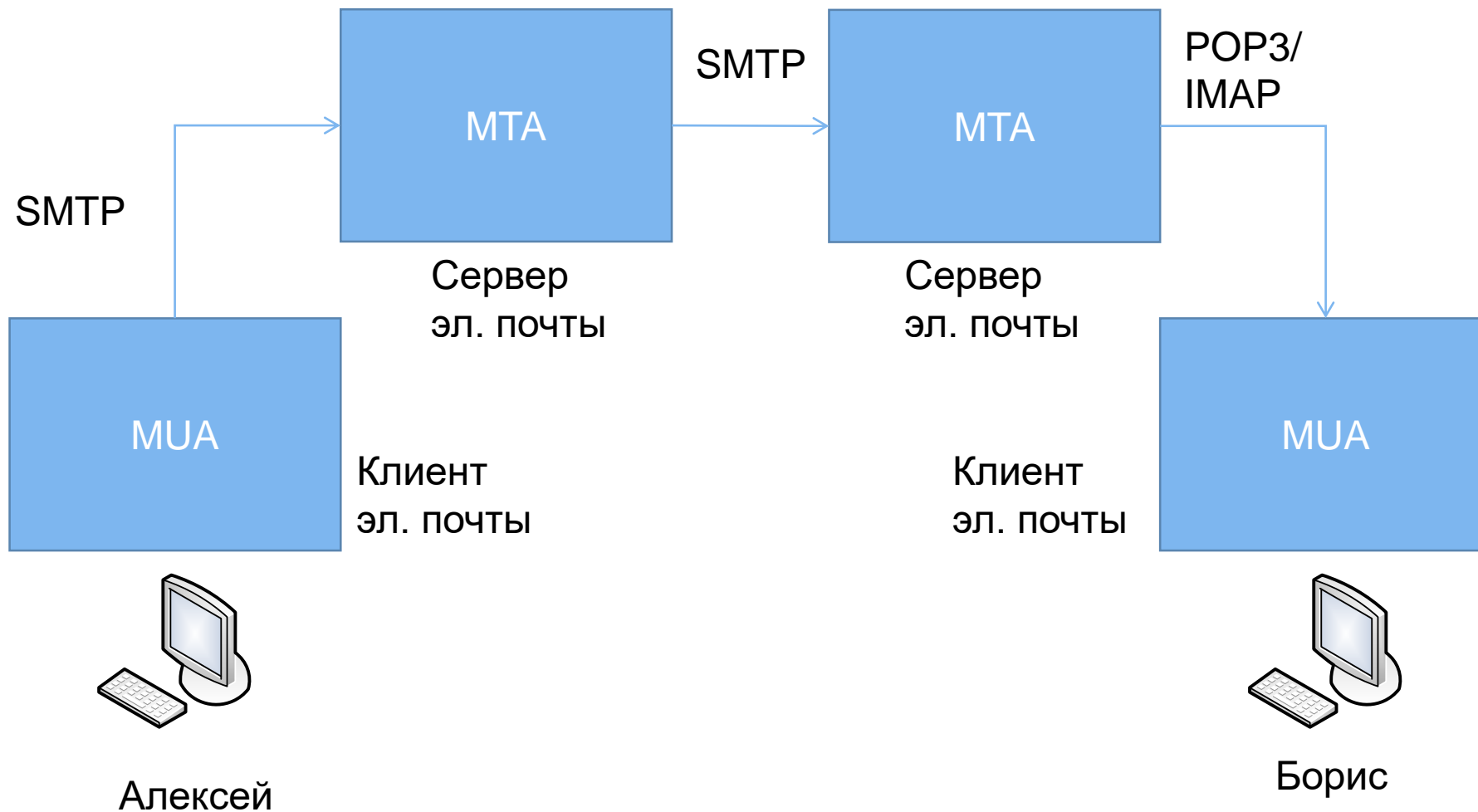
# Процесс передачи сообщения

- Когда письмо готово, Алексей вызывает функцию отправки сообщения, и встроенный SMTP-клиент посылает запрос на установление связи SMTP- серверу на компьютере Бориса.
- В результате устанавливаются SMTP и TCP-соединения, после чего сообщение передается через сеть.
- Почтовый сервер Бориса сохраняет письмо в памяти его компьютера, а почтовый клиент по команде выводит его на экран, при необходимости выполняя преобразование формата.

# Выделенный почтовый сервер

- Достаточно мощный и надежный компьютер, способный круглосуточно передавать почтовые сообщения от многих отправителей ко многим получателям (предоставляется организацией)
- Для каждого домена имен система DNS создает записи типа **MX**, хранящие DNS-имена почтовых серверов, обслуживающих пользователей, относящихся к этому домену.

# Использование промежуточных серверов



# Порядок обмена данными

- Он пишет текст сообщения, указывает необходимую сопроводительную информацию, в частности адрес получателя `boris@bmstu.ru`.
- Поскольку готовое сообщение должно быть направлено совершенно определенному почтовому серверу, клиент обращается к системе DNS, чтобы определить имя почтового сервера, обслуживающего домен `bmstu.ru`.
- Получив от DNS ответ `mail.bmstu.ru`, SMTP-клиент еще раз обращается к DNS — на этот раз чтобы узнать IP-адрес почтового сервера `mail.bmstu.ru`.

# Порядок обмена данными

- SMTP-клиент посылает по данному IP-адресу запрос на установление TCP-соединения через порт 25 (SMTP-сервер).
- Начинается диалог между клиентом и сервером по протоколу SMTP.
- Направление передачи запроса от клиента на установление SMTP-соединения совпадает с направлением передачи сообщения. Если сервер оказывается готовым, то после установления TCP-соединения сообщение Алексея передается.

# Порядок обмена данными

- Письмо сохраняется в буфере почтового сервера, а затем направляется в индивидуальный буфер, отведенный системой для хранения корреспонденции Бориса (т.е. почтовый ящик)
- В какой-то момент Борис запускает свою почтовую программу и выполняет команду проверки почты.
- После этой команды почтовый клиент должен запустить протокол доступа к почтовому серверу. Однако это будет не SMTP.

# Порядок обмена данными

- Инициатором передачи сообщений от почтового сервера почтовому клиенту по протоколу POP3 или IMAP является клиент.
- Почтовый сервер ожидает запрос на установление TCP-соединения по протоколу POP3 через порт 110, а по протоколу IMAP — через порт 143.
- В результате работы любого из них письмо Алексея передается в память компьютера Бориса.

# Пример smtp-клиента

```
from email.mime.multipart import MIMEMultipart
from email.mime.text import MIMEText
import smtplib
```

```
msg = MIMEMultipart()
message = "Sample message"
```

```
password = "mypassword"
msg['From'] = "sender@gmail.com"
msg['To'] = "receiver@gmail.com"
msg['Subject'] = "Subscription"
```

```
msg.attach(MIMEText(message, 'plain'))
```

```
server = smtplib.SMTP('smtp.gmail.com: 587')
```

```
server.starttls()
```

```
server.login(msg['From'], password)
```

```
server.sendmail(msg['From'], msg['To'], msg.as_string())
server.quit()
```

```
print
"successfully sent email to %s:" % (msg['To'])
```



# SMTP команды

- **MAIL** запускает пользовательского агента. Затем необходимо ввести тему сообщения, после чего можно печатать тело сообщения. Ввод точки в начале строки завершает сообщение, и пользовательский агент передает почту в MTA для доставки.
- Клиент осуществляет активное открытие на TCP порт 25, после чего ожидает приветственного сообщения (отклик с кодом 220) от сервера.
- **HELO** позволяет клиенту идентифицировать себя.
- **MAIL** идентифицирует автора сообщения (или отправителя). Следующая команда, RCPT, идентифицирует получателя. Если сообщение предназначено нескольким получателям, может быть исполнено несколько команд RCPT.
- **DATA** отправляет содержимое почтового сообщения. Строка, содержащая только точку, указывает на конец сообщения.
- **QUIT** прекращает обмен почтой.

# SMTP команды

- **RSET** прекращает текущую передачу почты и заставляет оба конца "сброситься". Любая сохраненная информация об отправителе, получателе или содержимое почты уничтожается.
- **VRFY** позволяет клиенту попросить отправителя проверить адрес получателя, не отправляя ему почту. Этим часто пользуются системные администраторы, чтобы вручную определить проблемы с доставкой почты. **NOOP** не делает ничего, однако заставляет сервер ответить, что все нормально, а именно откликом с кодом 200.
- Дополнительно: **EXPN** расширяет список почты и часто используется системными администраторами, так же как и **VRFY**. Более того, большинство версий Sendmail обрабатывают эти две команды одинаково.

# SMTP команды

- **TURN** позволяет клиенту и серверу поменяться ролями, чтобы послать почту в обратном направлении, не разрывая TCP соединение и не создавая новое. (Sendmail не поддерживает)
- **SEND**, **SOML** и **SAML** редко реализуются и призваны заменить собой команду MAIL. Позволяют доставлять почту непосредственно на пользовательский терминал (если пользователь находится терминалом в системе) или складывать ее в почтовый ящик получателя.

# Интервалы между ретрансляциями

- Когда пользовательский агент передает новое почтовое сообщение своему МТА, попытка доставить сообщение обычно осуществляется немедленно. Если доставить сообщение не удалось, МТА поставит сообщение в очередь и повторит попытку позже.
- Требования к хостам Host Requirements RFC рекомендует устанавливать первоначальный тайм-аут по крайней мере в 30 минут.
- Отправитель должен повторять свои попытки по меньшей мере 4-5 дней.
- Если сбои в доставке происходят часто (получатель вышел из строя или произошла временная потеря сетевого соединения), имеет смысл делать две попытки установить соединение в течение первого часа, когда сообщение находится в очереди.

# POP3

- Предназначен для получения сообщений, находящихся в почтовом ящике пользователя на удаленном сервере электронной почты.
- Сервер **SMTP** должен быть доступен постоянно, а рабочие станции обычно включают только на время работы пользователя, соединение с сервером они нередко устанавливают по коммутируемым линиям только для того, чтобы забрать накопившуюся почту.
- Почта доставляется только в хранилище сообщений, откуда пользователь может ее забрать в удобное для него время.

# Алгоритм работы POP3

- После установления соединения сервер посылает клиенту строку приветствия, свидетельствующую о готовности к диалогу, и сеанс переходит в состояние авторизации (AUTHORIZATION State). На этом этапе выясняется, доступ к какому именно почтовому ящику запрашивает клиент и имеет ли он соответствующие права. Успешное прохождение авторизации необходимо для продолжения работы.
- Если авторизация проходит успешно, то сеанс переходит в состояние транзакции (TRANSACTION State). На этом этапе клиент может проделывать все необходимые манипуляции с почтовым ящиком: он может просмотреть информацию о состоянии ящика и отдельных сообщений, получить выбранные сообщения и пометить письма, подлежащие удалению.

# Алгоритм работы POP3

- По окончании всех операций, клиент сообщает об окончании связи, и сеанс переходит в состояние обновления (UPDATE State). На этом этапе сервер стирает из ящика сообщения, помеченные на предыдущем этапе как подлежащие удалению, и закрывает соединение.
- Переход в состояние обновления в принципе возможен, только если клиент выходит из состояния транзакции по команде QUIT . Ни при каких других обстоятельствах, например, если сеанс связи прерывается по таймауту или из-за обрыва связи, переход в состояние обновления происходить не должен. То есть, если состояние транзакции прерывается не по команде QUIT , никакие удаления не должны производиться, пометки для удаления должны быть аннулированы. К сожалению, как показывает практика, это требование выполняется не всег

# Алгоритм работы POP3

- В ходе сеанса клиент посылает серверу команды, а сервер сообщает о результате выполнения каждой из них.
  - Ответ состоит из индикатора состояния (status indicator) и, если нужно, дополнительной информации, отделенной пробелом.
  - Строка ответа может содержать до 512 символов, включая последовательность **CRLF**, обозначающую конец строки.
- Предусмотрено два индикатора состояния:
  - "+OK" – успешное завершение и
  - "-ERR" – неуспешное завершение.

Если строка ответа не содержит дополнительной информации, то после индикатора состояния сразу должна идти последовательность **CRLF**. Однако некоторые клиенты ожидают пробела после индикатора состояния.



# Алгоритм работы POP3

- Если команда предусматривает многострочный ответ, то индикатор состояния передается только в первой строке, а последняя строка ответа должна состоять из одной точки. Эта строка не является частью ответа, а только обозначает его завершение.
- Чтобы сделать возможным использование строк, состоящих из одной точки, в ответах сервера, ко всем строкам ответа, начинающимся с точки, добавляется еще одна точка.

# Основные команды POP3

- В ответ на команду **STAT** сервер возвращает количество сообщений в почтовом ящике и общий размер ящика в октетах. Сообщения, помеченные для удаления, при этом не учитываются. Например, ответ "+OK 4 223718" означает, что в почтовом ящике имеется 4 сообщения общим объемом 223718 октет.
- Ответ на команду **LIST** без аргумента: список сообщений в почтовом ящике, содержащий их порядковые номера и размеры в октетах.
- Команда **RETR** Требуется в качестве аргумента номер существующего и не помеченного для удаления сообщения. В ответ сервер присылает запрошенное сообщение.

# Основные команды POP3

- Команда **DELE** требует в качестве аргумента номер существующего и не помеченного для удаления сообщения. Указанное сообщение помечается для удаления. До конца сеанса обращаться к нему становится невозможно. После окончания диалога, когда сеанс переходит в состояние обновления, сообщение удаляется окончательно.
- На команду **NOOP** команду сервер должен дать положительный ответ. Никаких других действий не производится.
- Команда **RSET** сервер снимает все установленные ранее пометки для удаления.
- Команда **QUIT** завершает сеанса. Если в ходе сеанса какие-то сообщения были помечены для удаления, то после выполнения команды QUIT они удаляются из ящика.

# Дополнительные возможности POP3

- Кроме обязательных команд, перечисленных выше, программное обеспечение, реализующее взаимодействие по протоколу **POP3**, поддерживает дополнительные возможности (capabilities), вводящие новые команды, влияющие на исполнение основных команд, облегчающие взаимодействие клиента и сервера, информирующие об особенностях реализации сервера и хранилища сообщений.
- В число дополнительных возможностей входят, например, команды авторизации. Хотя бы один механизм авторизации должен быть реализован, так как доступ к почтовому ящику предоставляется только после аутентификации. Но, поскольку таких механизмов несколько, и их выбор оставляется на усмотрение разработчиков и администраторов, соответствующие команды не входят в число обязательных.
- Предусмотрена команда **CAPA**, позволяющая клиенту получить информацию о дополнительных возможностях, реализованных на сервере, и их параметрах.

# Протокол IMAP

- Используется на участке между **MUA** получателя и хранилищем сообщений.
- предоставляет более широкие возможности работы с почтовыми ящиками, чем **POP3**: он позволяет работать с несколькими почтовыми ящиками на одном или нескольких серверах **IMAP** как с файлами и каталогами на собственной машине пользователя.
- Сервер **IMAP** способен анализировать сообщение: выделять заданные поля заголовка и разбирать структуру тела сообщения.

# Состояния сеанса IMAP

- **Неаутентифицированное состояние** (Not Authenticated State): клиент должен пройти процедуру аутентификации прежде, чем сможет выполнять большинство команд;
- **Аутентифицированное состояние** (Authenticated State): клиент аутентифицирован и должен выбрать почтовый ящик, прежде чем сможет работать с отдельными сообщениями;
- **Выбранное состояние** (Selected State): почтовый ящик выбран;
- **Состояние выхода** (Logout State): сеанс завершается.

# Команды любого состояния сеанса

- **CAPABILITY.** В ответ на эту команду сервер присылает непомеченную строку с ключевым словом CAPABILITY, содержащую список поддерживаемых возможностей (расширений) и их параметров. В число возможностей входит в частности поддерживаемая версия протокола **IMAP** – IMAP4rev1 и механизмы аутентификации ( AUTH =механизм\_аутентификации), описанные в RFC 2595 .
- **NOOP.** Не выполняет никаких действий. Однако эта команда сбрасывает таймер неактивности, что позволяет избежать разрыва соединения по таймауту. Кроме того, при определенных обстоятельствах эта или другая команда служит неявным запросом информации об обновлениях, произошедших на сервере. Таким образом, с помощью команды NOOP можно периодически проверять, не появились ли новые сообщения или не изменился ли статус старых.
- **LOGOUT.** Конец сеанса.

# Команды неаутентифицированного состояния

- Информация о поддерживаемых способах аутентификации передается сервером клиенту в ответе на команду CAPABILITY.
- Так запись STARTTLS свидетельствует о поддержке одноименной команды, описанной в RFC 2595 , LOGINDISABLED – расширение, исключающее аутентификацию с использованием незашифрованных имени и пароля, параметры AUTH указывают, какие механизмы аутентификации с использованием **SASL** поддерживает сервер.



# Команды неаутентифицированного состояния

- **STARTTLS.** Переводит сеанс в защищенный режим. После получения сервером команды STARTTLS клиент и сервер согласовывают параметры дальнейшего взаимодействия. Все данные, которыми обмениваются клиент и сервер после успешного завершения этой команды, передаются в зашифрованном виде. Однако аутентификация при помощи этой команды не производится, сеанс остается в неаутентифицированном состоянии.
- **LOGIN регистрационное\_имя\_пользователя пароль** Аутентификация при помощи регистрационного имени и пароля, передаваемых открытым текстом.
- **AUTHENTICATE механизм** - передача зашифрованных аутентификационных данных с использованием **SASL** .

# Команды аутентифицированного состояния

- **SELECT имя\_ящика**
- Открывает доступ к указанному почтовому ящику. Сеанс переходит в состояние выбора, после этого клиент может работать с отдельными сообщениями в ящике.
- В ответ на эту команду сервер присылает ряд непомеченных ответов, содержащих информацию о почтовом ящике: количество сообщений, список допустимых флагов (см. описание команды APPEND), количество новых сообщений, номер первого непрочитанного сообщения, идентификатор почтового ящика.

# Команды аутентифицированного состояния

- **EXAMINE имя\_ящика** - аналогично команде SELECT , но почтовый ящик открывается только для чтения.
- **CREATE имя\_объекта** Создает новый почтовый ящик или каталог. Если объект создается не в корневом каталоге, то надо указать путь к нему.
- Если на конце указанного имени стоит символ, используемый в качестве иерархического разделителя, создается каталог.
- **DELETE имя\_ящика** - удаляет указанный почтовый ящик. Эта же команда удаляет также и каталоги, если они не содержат почтовые ящики.

# Команды аутентифицированного состояния

- **RENAME** имя\_ящика новое\_имя\_ящика
- Переименование почтового ящика.
- **SUBSCRIBE** имя\_ящика
- Почтовый ящик помечается как "активный". Эта пометка используется для вывода списка почтовых ящиков при помощи команды LSUB.
- **UNSUBSCRIBE** имя\_ящика
- Снимает с почтового ящика пометку "активный". Эта пометка может быть снята с почтового ящика только при помощи команды UNSUBSCRIBE. Даже если ящик больше не существует, это не может само по себе стать причиной снятия пометки "активный".

# Команды аутентифицированного состояния

- **LIST** *путь\_к\_ящику имя\_ящика* возвращает список каталогов и почтовых ящиков, соответствующих указанным аргументам.
- **LSUB** *путь\_к\_ящику имя\_ящика* команда LSUB аналогична команде LIST , но она возвращает только имена почтовых ящиков с пометкой "активный".
- **APPEND** *имя\_ящика (флаги\_сообщения) метка\_времени сообщение*
- Добавляет сообщение в конец указанного почтового ящика. В качестве аргументов указываются имя ящика, флаги сообщения (не обязательно), метка времени (не обязательно) и само сообщение – заголовок и тело.

# Команды аутентифицированного состояния

- **STATUS имя\_ящика (имена\_элементов)**
- Возвращает запрошенные элементы информации об указанном почтовом ящике. Имена элементов информации разделяются пробелами и все вместе заключаются в скобки. Предусмотрены следующие имена элементов информации:
  - **MESSAGES** – общее количество сообщений в ящике;
  - **RECENT** – количество новых сообщений;
  - **UIDNEXT** –уникальный идентификатор, который изменяется всякий раз, когда в почтовый ящик помещается новое сообщение, используется для того, чтобы определить, появились ли в ящике новые сообщения за время, прошедшее после предыдущей проверки;
  - **UIDVALIDITY** – уникальный идентификатор почтового ящика;
  - **UNSEEN** – количество сообщений, не помеченных как прочитанные.

# Команды выбранного состояния

- **CHECK**

- Команда производит проверку выбранного почтового ящика, характер которой зависит от реализации программного обеспечения сервера.

- **CLOSE**

- Выбранный почтовый ящик закрывается. При этом, если почтовый ящик был открыт для чтения и записи, все помеченные для удаления сообщения в ящике удаляются. Сеанс возвращается в аутентифицированное состояние.

# Команды выбранного состояния

- **EXPUNGE**
- Из выбранного почтового ящика удаляются все помеченные для удаления сообщения. Для каждого удаляемого сообщения посылается непомеченный ответ, содержащий номер сообщения и ключевое слово EXPUNGE .
- **SEARCH кодировка\_символов критерии\_поиска**
- Поиск в выбранном почтовом ящике сообщений, отвечающих указанным критериям поиска.
- **FETCH x:y имя\_элемента\_сообщения\_или\_макрос**
- Сервер возвращает информацию, относящуюся к сообщениям, обозначенным первым аргументом команды. Это может быть либо число, обозначающее номер сообщения, либо интервал от номера x до номера y, записанный в формате x:y.
- Во втором аргументе перечисляются запрашиваемые информационные элементы.



# Команды выбранного состояния

- **STORE x:y имя\_элемента\_данных (список\_флагов)**
- Команда STORE изменяет значения флагов для указанного в первом аргументе сообщения или сообщений. В ответ сервер возвращает непомеченный ответ с ключевым словом FETCH, содержащий значения флагов, если в имени элемента данных не используется суффикс .SILENT.
- **COPY x:y имя\_ящика**
- Копирует сообщения с номерами от x до y из текущего почтового ящика в указанный почтовый ящик.

# Команды выбранного состояния

- **UID** команда аргументы
- Команда **UID** принимает в качестве аргументов команды COPY, FETCH или STORE с их аргументами, но наряду с номерами сообщений в ответах указываются уникальные идентификаторы сообщений.
- Также команду **UID** можно использовать совместно с командой SEARCH. В этом случае интерпретация аргументов команды SEARCH не изменяется, но в ответах на эту команду будут приведены уникальные

# Список использованных источников

- В. Олифер, Н. Олифер “Компьютерные сети. Принципы, технологии, протоколы”
- Куроуз, Росс “Компьютерные сети. Нисходящий подход.”
- <https://docs.python.org/>

**Спасибо за внимание!**

# Лекция VII. Протоколы транспортного уровня

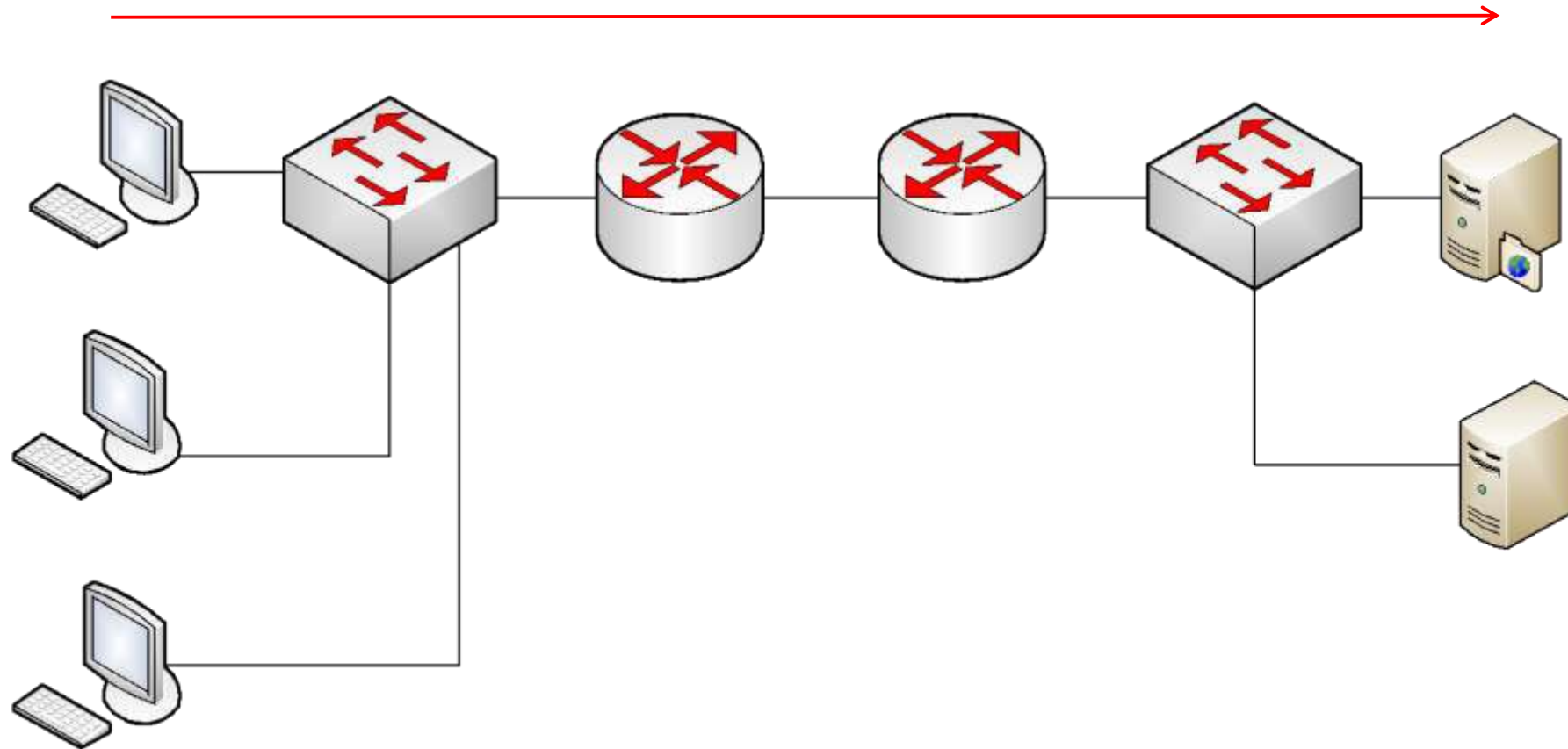
Рогозин Н.О., кафедра ИУ-7

# Введение

- Транспортный уровень – **центральная часть** сетевой архитектуры
- Связывает службы доставки сетевого уровня, работающие между двумя конечными системами — с одной стороны, и службы доставки, действующими между двумя процессами прикладного уровня, запущенными на конечных системах — с другой.
- Определяет адресацию физических устройств (систем, их частей) в сети.
- Гарантирует доставку блоков информации адресатам и управляет этой доставкой.
- Когда в процессе обработки находится более одного пакета, транспортный уровень контролирует очередность прохождения пакетов. Если проходит дубликат принятого ранее сообщения, то данный уровень опознает это и игнорирует сообщение.

# Введение

логическое соединение трансп. уровня



# Сегмент

- Пакет транспортного уровня, используемый для передачи информации
- Получается разбиением сообщения прикладного уровня на фрагменты с последующим добавлением заголовка (TCP/UDP)
- Далее инкапсулируется в пакет сетевого уровня (дейтаграмму) и отсылается
- На принимающей стороне сетевой уровень извлекает сегмент и передает транспортному
- Транспортный уровень выполняет обработку



# Сегмент транспортного уровня

Номер порта отправителя	Номер порта получателя
Другие поля заголовка	
Прикладные данные (сообщение/payload)	

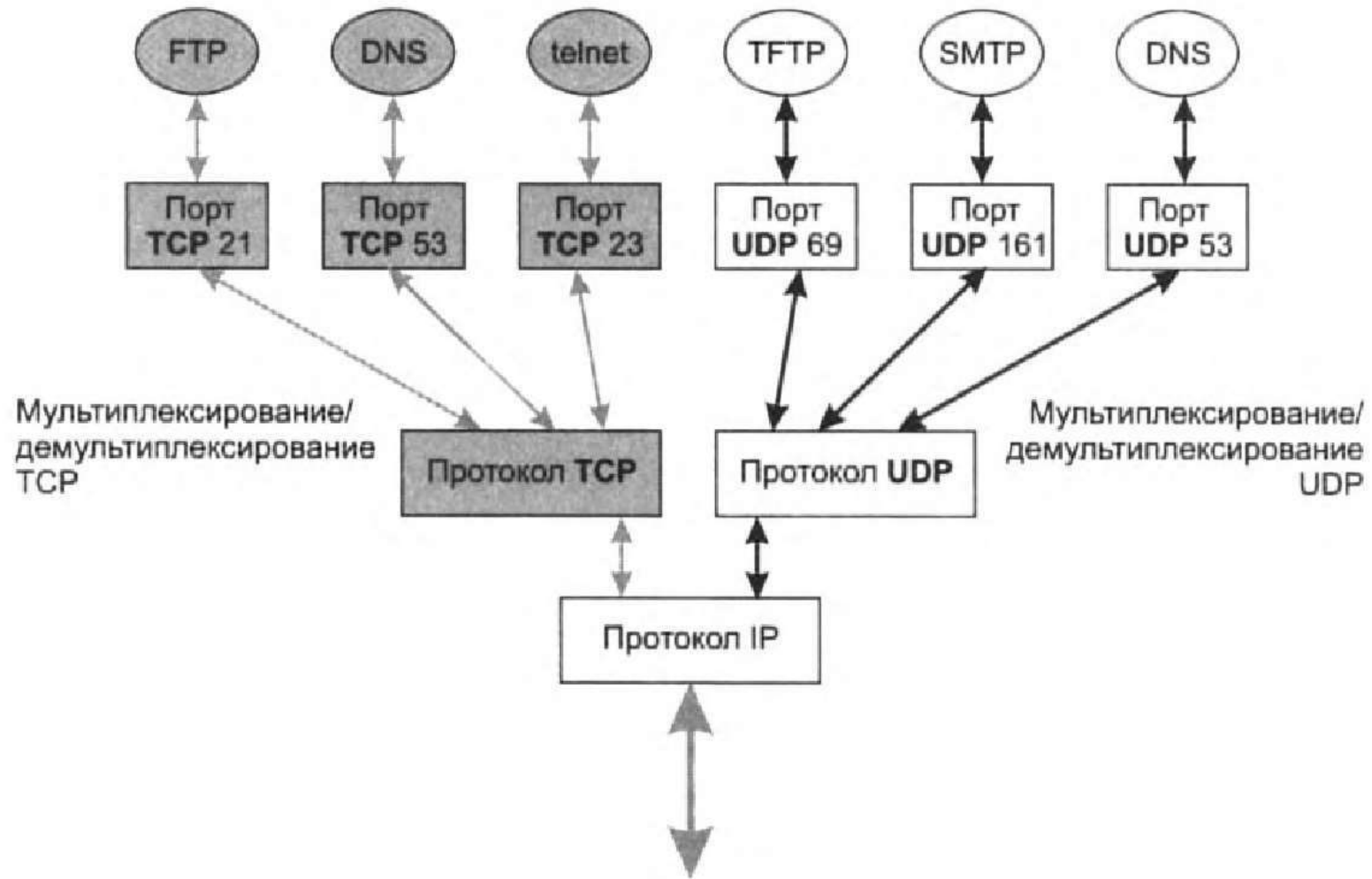
# Мультиплексирование/ демультиплексирование

- Работа по доставке данных сегмента транспортного уровня нужному, соответствующему сокету называется **демультиплексированием**.
- Сбор фрагментов данных, поступающих на транспортный уровень хоста-отправителя из различных сокетов, создание сегментов путем присоединения заголовка (который используется при демультиплексировании) к каждому фрагменту и передача сегментов сетевому уровню называется **мультиплексированием**

# Мультиплексирование/ демультиплексирование

- Протоколы TCP и UDP ведут для каждого приложения две системные очереди: очередь данных, поступающих к приложению из сети, и очередь данных, отправляемых этим приложением в сеть.
- Такие системные очереди и называются портами, причем входная и выходная очереди одного приложения рассматриваются как один порт.
- Для идентификации портов им присваивают номера.

# Мультиплексирование/ демультиплексирование



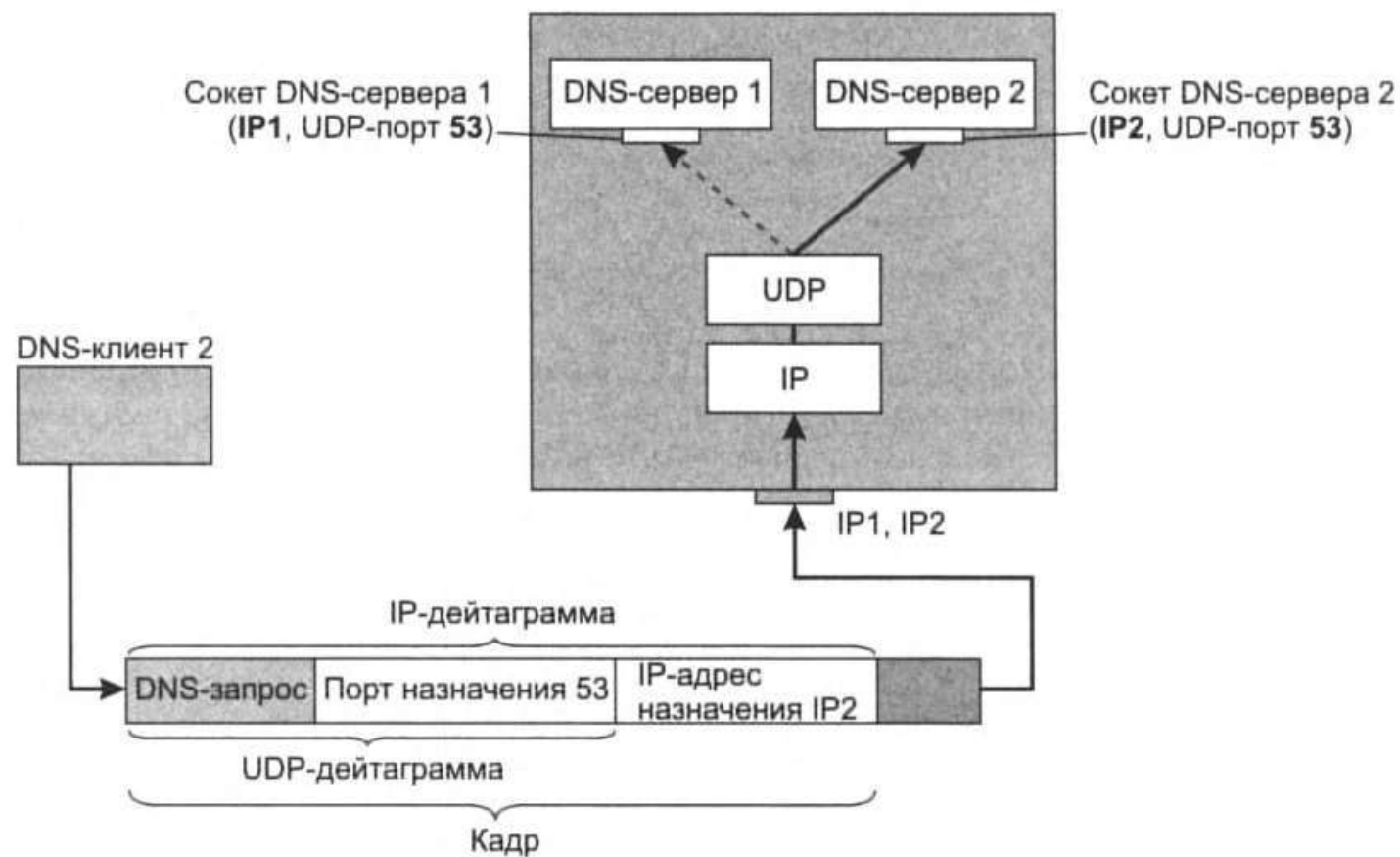
# Номера портов

- **Хорошо известные** - для популярных системных служб, таких как FTP, telnet, HTTP, TFTP, DNS (1 - 1024)
- **Зарегистрированные** - предоставляются IANA (1025 - 49151)
- **Динамические** - назначаются локально разработчиками этих приложений или ОС в ответ на поступление запроса от приложения.
- На каждом компьютере ОС ведет список занятых и свободных номеров портов. При поступлении запроса от приложения, выполняемого на данном компьютере, ОС выделяет ему первый свободный номер.

# UDP и TCP порты

- Нет никакой зависимости между назначением номеров портов для приложений, использующих протокол TCP, и приложений, работающих с протоколом UDP. Приложения, передающие данные на уровень IP по протоколу UDP, получают номера, называемые UDP-портами.
- Аналогично приложениям, обращающимся к протоколу TCP, выделяются TCP-порты.
- В том и другом случаях это могут быть как назначенные, так и динамические номера.
- Диапазоны чисел, из которых выделяются номера TCP- и UDP-портов, совпадают: **от 0 до 1023** для назначенных и **от 1024 до 65 535** для динамических.

# Демультимплексирование UDP по порту

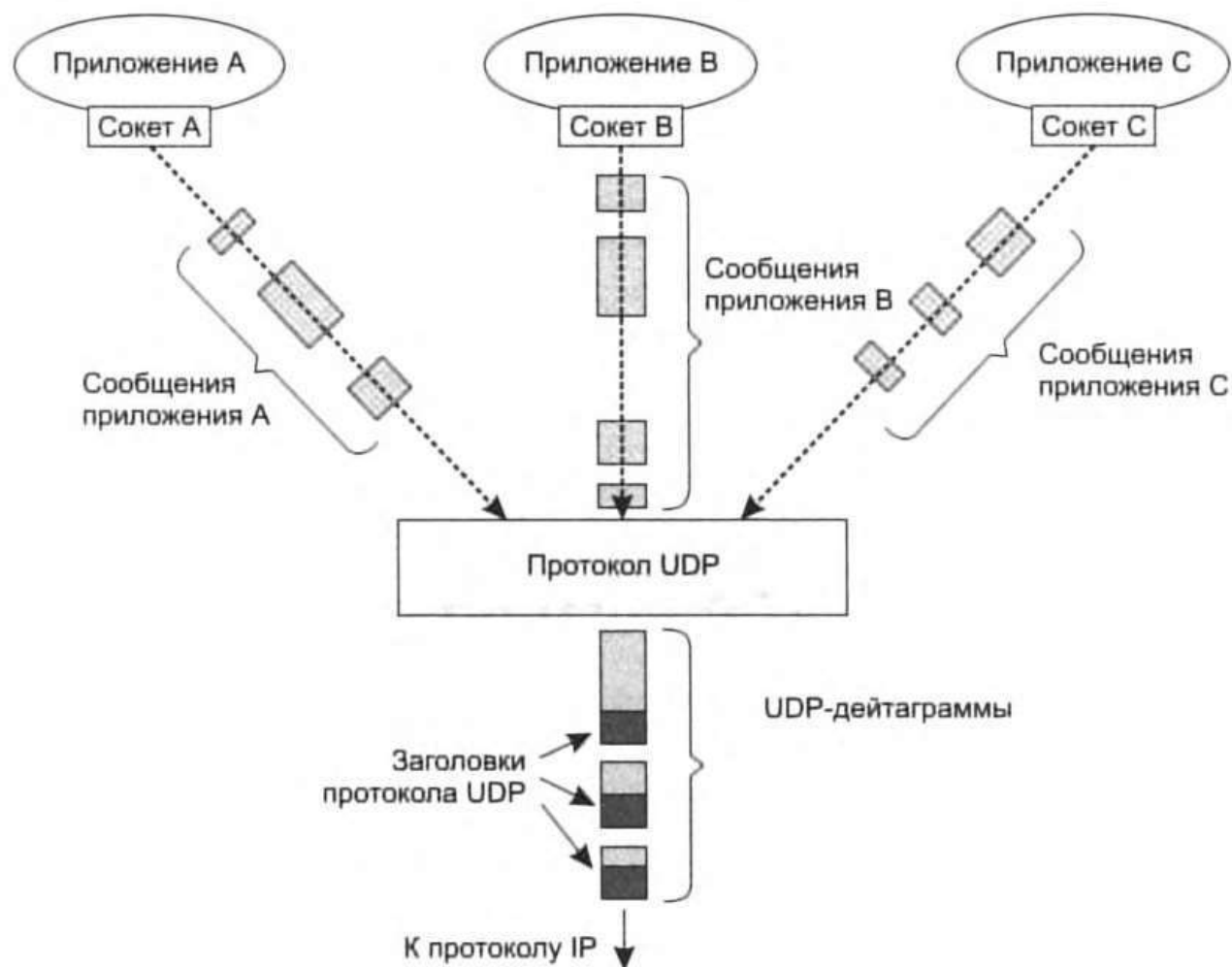


# Ассоциации IP-порт

- Чтобы снять неоднозначность в идентификации приложений, разные копии связываются с разными IP-адресами.
- Для этого сетевой интерфейс компьютера, на котором выполняется несколько копий приложения, должен иметь соответствующее число IP-адресов — на рисунке это IP1 и IP2. Во всех IP-пакетах, направляемых DNS-серверу 1, в качестве IP-адреса указывается IP1, а DNS-серверу 2 — адрес IP2.
- Поэтому показанный на рисунке пакет, в поле данных которого содержится UDP-дейтаграмма с указанным номером порта 53, а в поле заголовка задан адрес IP2, однозначно будет направлен заданному адресату — DNS-серверу 2.



# UDP(RFC 0768)



# UDP

- Поля портов состоят из 16 -битных целых чисел, представляющих номера портов приложений.
- Поле «порт источника» содержит номер порта, которым пользуется приложение - источник данных.
- Поле «порт - получатель» соответственно указывает на номер порта приложения - получателя данных.
- Поле «длина сообщения» определяет длину (в байтах) UDP - дейтаграммы, включая UDP - заголовок.
- Поле «контрольная сумма», в отличие от контрольной суммы IP - заголовка, содержит результат суммирования всей UDP -дейтаграммы, включая ее данные, область которых начинается сразу после заголовка.
- Модуль UDP отслеживает появление вновь прибывших дейтаграмм, сортирует их и распределяет в соответствии с портами назначения.

# Расчет контрольной суммы

- Если длина UDP-сообщения в байтах нечётна, то UDP-сообщение дополняется в конце нулевым байтом (псевдозаголовок и добавочный нулевой байт не отправляются вместе с сообщением, они используются только при расчёте контрольной суммы).
- Поле контрольной суммы в UDP-заголовке во время расчёта контрольной суммы принимается нулевым.
- Псевдозаголовок и UDP-сообщение разбивается на двухбайтные слова. Затем рассчитывается сумма всех слов в арифметике обратного кода (т. е. кода, в котором отрицательное число получается из положительного инверсией всех разрядов числа и существует два нуля: 0x0000 (обозначается +0) и 0xffff(обозначается -0)).
- Результат записывается в соответствующее поле в UDP-заголовке.

# Расчет контрольной суммы

- Значение контрольной суммы, равное 0x0000 (+0 в обратном коде), зарезервировано и означает, что для посылки контрольная сумма не вычислялась. В случае, если контрольная сумма вычислялась и получилась равной 0x0000, то в поле контрольной суммы заносят значение 0xffff (-0 в обратном коде).
- При получении сообщения получатель считает контрольную сумму заново (уже учитывая поле контрольной суммы), и, если в результате получится -0 (то есть 0xffff), то контрольная сумма считается сошедшейся. Если сумма не сходится (данные были повреждены при передаче, либо контрольная сумма неверно посчитана на передающей стороне), то решение о дальнейших действиях принимает принимающая сторона.
- Как правило, в большинстве современных устройств, работающих с UDP/IP-пакетами имеются настройки, позволяющие либо игнорировать такие пакеты, либо пропускать их на дальнейшую обработку, невзирая на неправильность контрольной суммы.

# Особенности UDP

- **Ненадёжный** — когда сообщение посылается, неизвестно, достигнет ли оно своего назначения — оно может потеряться по пути. Нет таких понятий, как подтверждение, повторная передача, тайм-аут.
- **Неупорядоченность** — если два сообщения отправлены одному получателю, то порядок их достижения цели не может быть предугадан.
- **Легковесность** — никакого упорядочивания сообщений, никакого отслеживания соединений и т. д. Это небольшой транспортный уровень, разработанный на IP.
- **Датаграммы** — пакеты посылаются по отдельности и проверяются на целостность только если они прибыли. Пакеты имеют определенные границы, которые соблюдаются после получения, то есть операция чтения на сокете-получателе выдаст сообщение таким, каким оно было изначально послано.
- **Нет контроля перегрузок** — UDP сам по себе не избегает перегрузок. Для приложений с большой пропускной способностью возможно вызвать коллапс перегрузок, если только они не реализуют меры контроля на прикладном уровне.

# Особенности UDP

- Из-за недостатка надёжности приложения UDP должны быть готовы к некоторым потерям, ошибкам и дублированиям. Некоторые из них (например, TFTP) могут при необходимости добавить элементарные механизмы обеспечения надёжности на прикладном уровне.
- В отличие от TCP, основанные на UDP приложения не обязательно имеют хорошие механизмы контроля и избегания перегрузок.
- Чувствительные к перегрузкам UDP-приложения, которые потребляют значительную часть доступной пропускной способности, могут поставить под угрозу стабильность в Интернете.

# Приложения UDP

- Многочисленные ключевые Интернет-приложения используют UDP, в их числе — **DNS** (где запросы должны быть быстрыми и состоять только из одного запроса, за которым следует один пакет ответа), **Простой Протокол Управления Сетями (SNMP)**, **Протокол Маршрутной Информации (RIP)**, **Протокол Динамической Конфигурации Узла (DHCP)**.
- **Голосовой и видеотрафик.** Протоколы потокового видео в реальном времени и аудио разработаны для обработки случайных потерь пакетов так, что качество лишь незначительно уменьшается вместо больших задержек при повторной передаче потерянных пакетов. Поскольку и TCP, и UDP работают с одной и той же сетью, многие компании замечают, что недавнее увеличение UDP-трафика из-за этих приложений реального времени мешает производительности TCP-приложений вроде систем БД.

# TCP

- Служит для передачи данных между сетевыми и прикладными уровнями сетевой модели.
- Для обеспечения надежной доставки и правильной последовательности данных в общем потоке, протокол TCP использует подтверждения. Каждый раз при передаче сообщения модуль TCP запускает таймер. По истечении заданного в нем времени и не получения подтверждения, протокол TCP повторяет попытку передать свое сообщение.



# Логическое соединение

- Основным отличием TCP от UDP является то, что на протокол TCP возложена дополнительная задача — обеспечить надежную доставку сообщений, используя в качестве основы ненадежный дейтаграммный протокол IP.
- **Логическое соединение** дает возможность участникам обмена следить за тем, чтобы данные не были потеряны, искажены или продублированы, а также чтобы они пришли к получателю в том порядке, в котором были отправлены.
- Протокол TCP устанавливает логические соединения между прикладными процессами, причем в каждом соединении участвуют только два процесса. TCP-соединение является дуплексным, то есть каждый из участников этого соединения может одновременно получать и отправлять данные.

# Параметры логического соединения

Каждая сторона сообщает:

- максимальный размер сегмента, который она готова принимать;
- максимальный объем данных (возможно несколько сегментов), которые она разрешает другой стороне передавать в свою сторону, даже если та еще не получила квитанцию на предыдущую порцию данных (размер окна);
- начальный порядковый номер байта, с которого она начинает отсчет потока данных в рамках данного соединения.

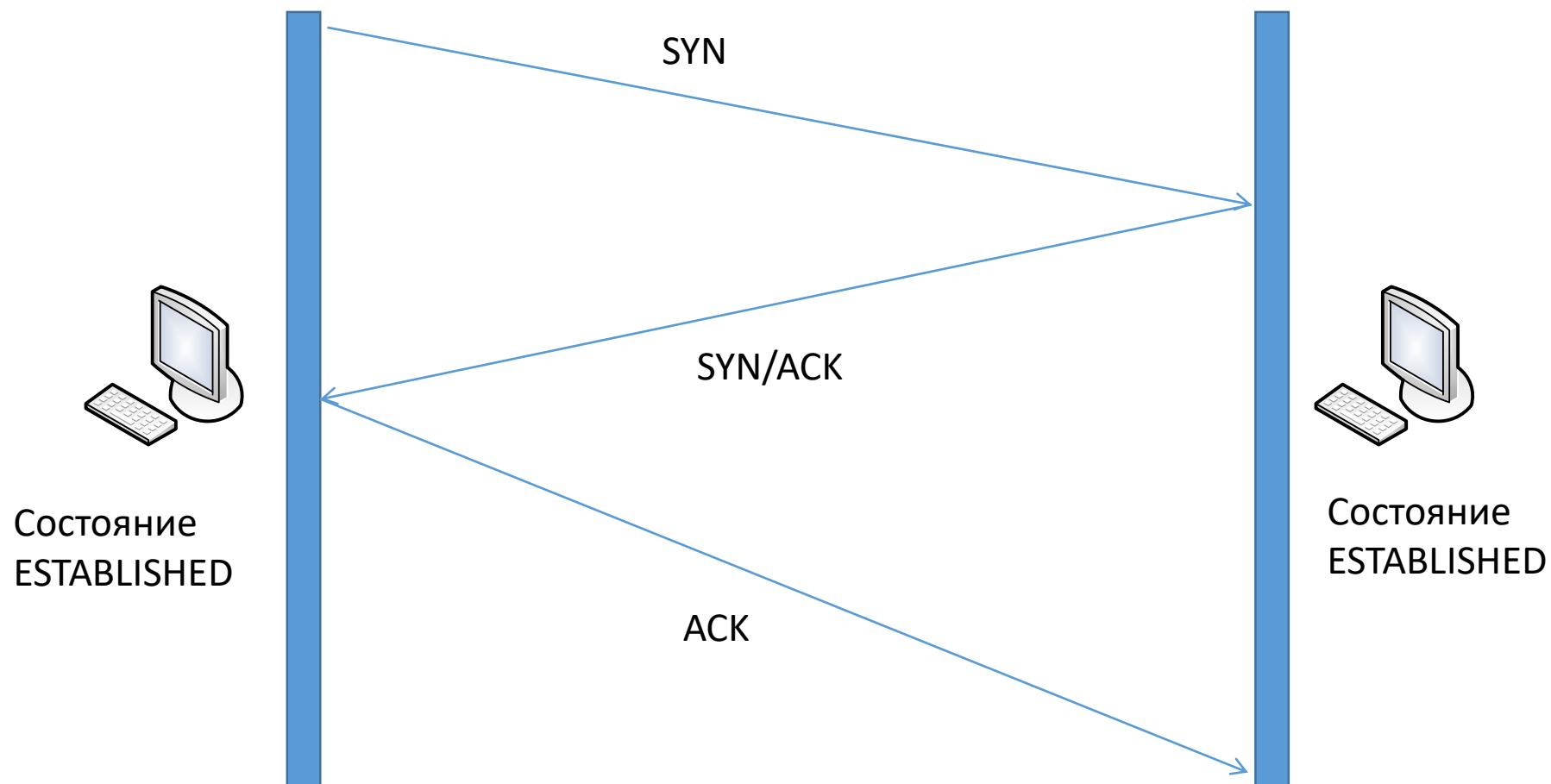
# Особенности TCP

- **Надёжность** — TCP управляет подтверждением, повторной передачей и тайм-аутом сообщений. Производятся многочисленные попытки доставить сообщение. Если оно потеряется на пути, сервер вновь запросит потерянную часть. В TCP нет ни пропавших данных, ни (в случае многочисленных тайм-аутов) разорванных соединений.
- **Упорядоченность** — если два сообщения последовательно отправлены, первое сообщение достигнет приложения-получателя первым. Если участки данных прибывают в неверном порядке, TCP отправляет неупорядоченные данные в буфер до тех пор, пока все данные не могут быть упорядочены и переданы приложению.
- **Тяжеловесность** — TCP необходимо три пакета для установки сокет-соединения перед тем, как отправить данные. TCP следит за надёжностью и перегрузками.
- **Потоковость** — данные читаются как поток байтов, не передается никаких особых обозначений для границ сообщения или сегментов.

# Установление сеанса

- Процесс установления сеанса TCP между клиентом и сервером (называемый также трехэтапным квитированием) происходит следующим образом. Клиент инициирует сеанс, передавая сегмент с установленным битом синхронизации (SYNchronization — SYN).
- Этот сегмент содержит данные о размере окна клиента и его текущем порядковом номере. Сервер отвечает на запрос SYN клиента сегментом ACK и также включает в передаваемый им сегмент бит SYN, данные о размере окна и начальном порядковом номере.
- Наконец, клиент отвечает на сегмент SYN сервера подтверждением ACK.

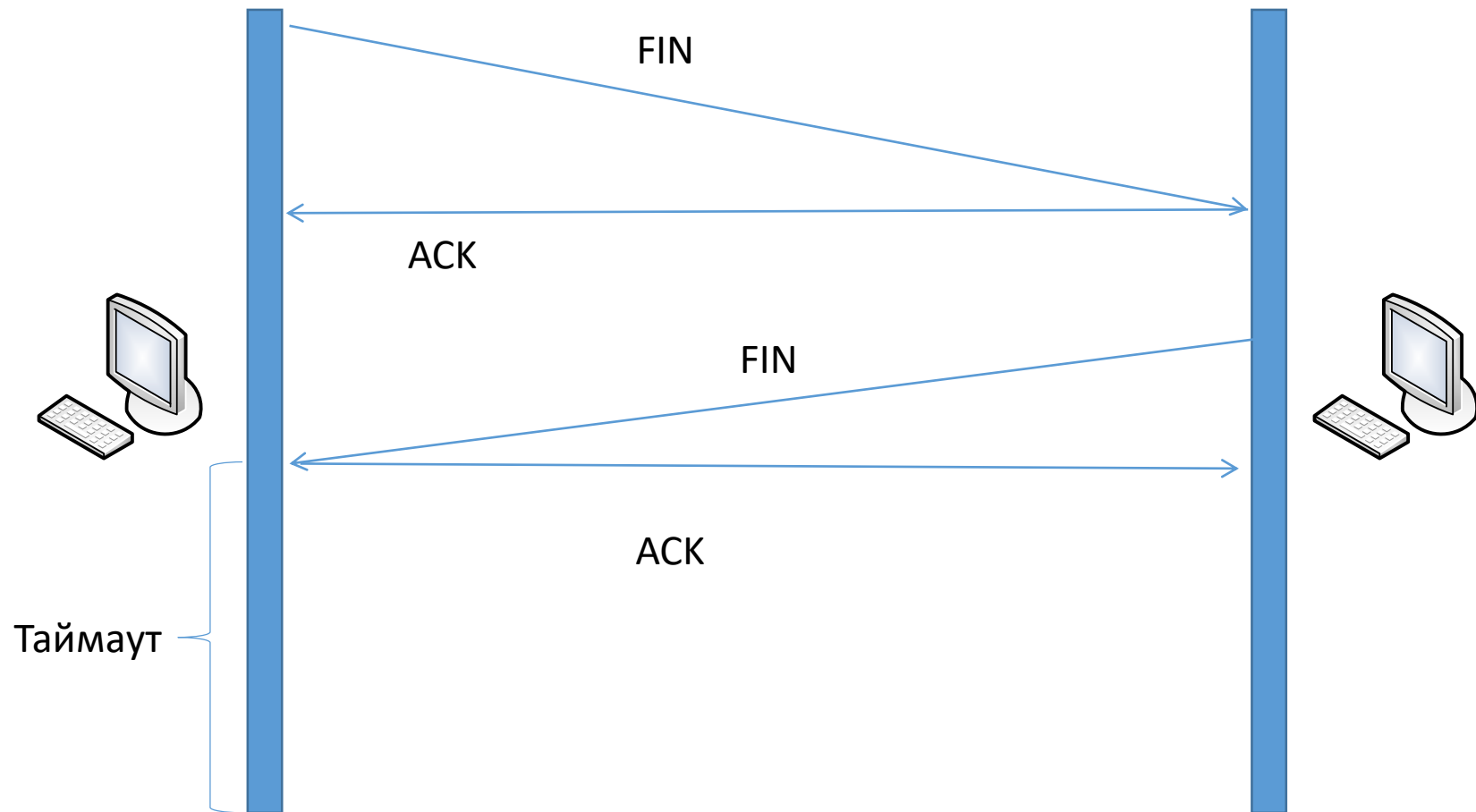
# Установление сеанса



# Разрыв сеанса

- аналогичен процессу его установления. Вначале участник соединения, желающий закрыть сеанс (предположим, что в данном примере это — клиент), инициализирует процесс завершения сеанса, отправляя сегмент с установленным битом завершения (FINish — FIN).
- Сервер отвечает, передавая в ответ на сегмент FIN клиента подтверждение ACK. Затем сервер передает собственный сегмент с установленным битом FIN. После этого клиент передает в ответ на сегмент FIN сервера подтверждение ACK и сеанс закрывается.
- Каждый FIN/ACK закрывает одно направление передачи

# Разрыв сеанса



# ARQ

- Должен ли отправитель ждать, пока не придет квитанция на отправленный пакет, прежде чем отсылать следующий?
- Должна ли принимающая сторона подтверждать приход каждого или сразу нескольких пакетов?
- Какое время ожидания квитанции источником является предельно допустимым?
- Что, если квитанция потеряется и отправитель еще раз пошлет тот же пакет? Каким образом приемник должен распознавать дубликаты пакетов, а источник — квитанций?



# Методы решений

- методы простоя источника (Stop-and-Wait);
- методы скользящего окна, которые свою очередь тоже подразделяются на два класса:
  - методы, использующие **окно передачи** — к ним, в частности, относится **метод передачи с возвратом на N пакетов (Go-Back-N)**;
  - методы, использующие окно передачи и окно приема — примером является **метод передачи с выборочным повторением (Selective Repeat)**.

# Общие условия для всех методов

- Отправитель и получатель, в общем случае работающие асинхронно, осуществляют передачу пакетов по **ненадежной линии связи**, в которой возможны искажения, большие задержки и потери пакетов.
- Отправитель принимает данные от протокола верхнего уровня (приложения), получатель передает полученные данные на верхний уровень (приложению).
- Получатель располагает механизмом определения искаженных пакетов, например, по **контрольной сумме**.
- После успешного получения пакета получатель посылает отправителю **квитанции (acknowledgment, ACK)**.
- Для отслеживания задержек пакетов используется таймер, тайм-аут которого устанавливается равным предельному времени ожидания квитанции.

# Окно перегрузки

- Принцип скользящего окна позволяет послать несколько сообщений и только потом ожидать подтверждения. Модуль TCP регулирует полосу пропускания сети, договариваясь с удаленным узлом о параметрах потока данных.
- При этом процесс регулировки происходит на протяжении всего соединения TCP. Регулировка заключается в изменении размеров скользящего окна.
- При низкой загрузке сети, размер скользящего окна увеличивается, скорость выдачи данных на канале связи увеличивается. Если загрузка сети достаточна велика, модуль TCP уменьшает размер скользящего окна.

# Окно перегрузки

- Метод передачи с применением окон, предусмотренный в протоколе ТСР, не только представляет собой динамический процесс, но и осуществляется в дуплексном режиме. Это означает, что на каждом хосте предусмотрена возможность и передавать, и принимать данные о размере и положении окна.
- Каждое окно применяется независимо от другого и может увеличиваться или уменьшаться с учетом условий передачи в соответствующем направлении.

# Окно управления

- Для идентификации пакетам присваиваются уникальные последовательные номера, размещаемые в заголовках пакетов.
- Разрядность поля «номер пакета» определяет диапазон возможных номеров. Когда этот диапазон исчерпывается, нумерация пакетов снова начинается с нуля.
- Таким образом, нельзя абсолютно исключить ситуацию, когда в сети существуют пакеты с одинаковыми номерами. Окно определяется на последовательности пронумерованных пакетов.
- Окно всегда имеет нижнюю границу, называемую также базой окна, и верхнюю границу.
- Количество номеров, попадающих в пределы окна, называют размером окна.
- Очевидно, что окно всегда перемещается в сторону больших номеров.

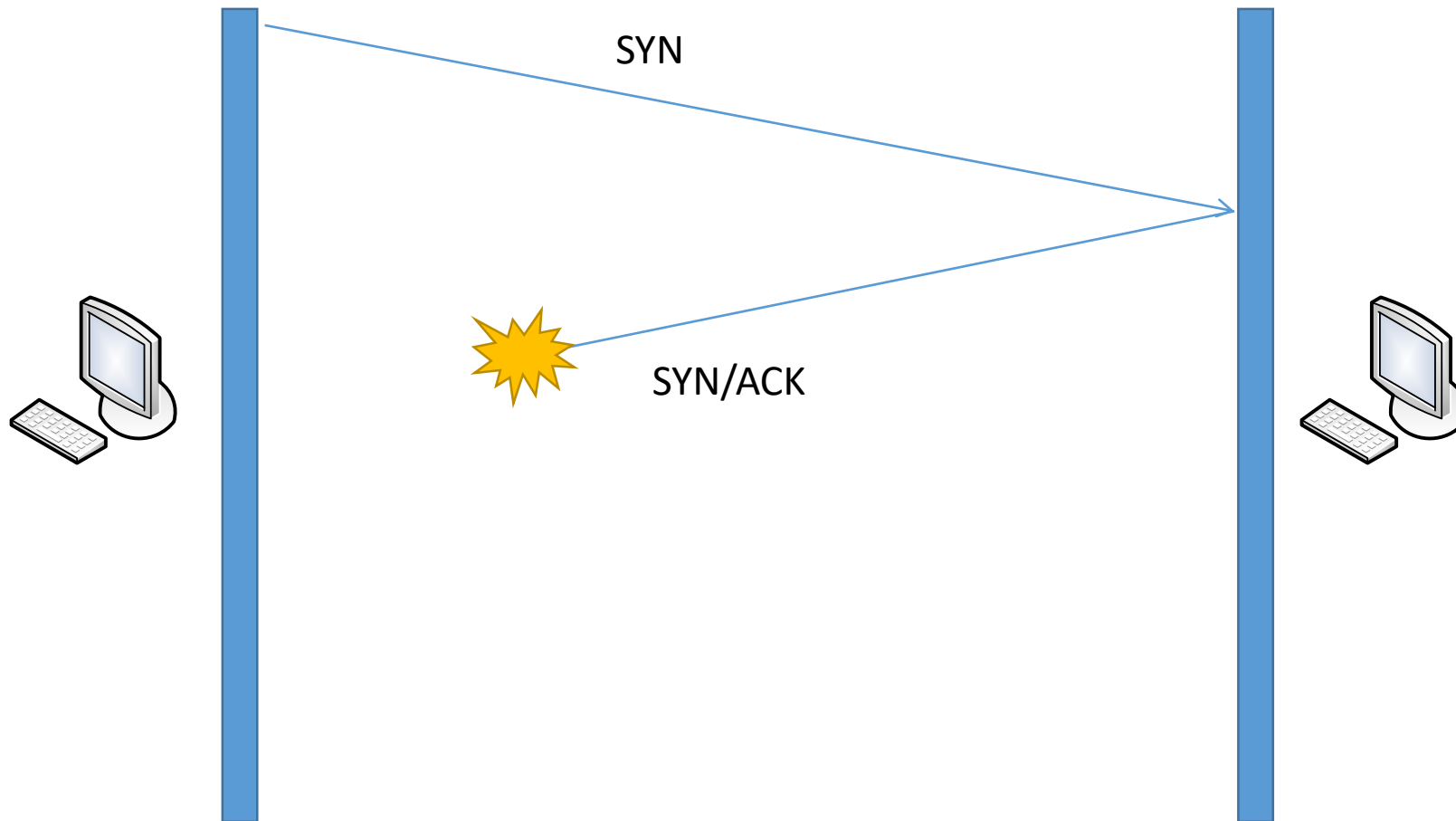
# Скольльзящее окно



# Скользящее окно

- В идеальном случае пакеты и квитанции не теряются и приходят в том же порядке, в котором были отправлены.
- При этом интервалы между пакетами и квитанциями являются неравномерными.
- Движение окна определяется, во-первых, поступлением квитанций — подтверждение успешного приема очередного пакета позволяет переместить окно вперед, во-вторых, исчерпанием окна — окно приостанавливается, когда отправитель передал все пакеты из окна, но не получил ни на один из них квитанции.

# Дублирование сообщения из-за потери





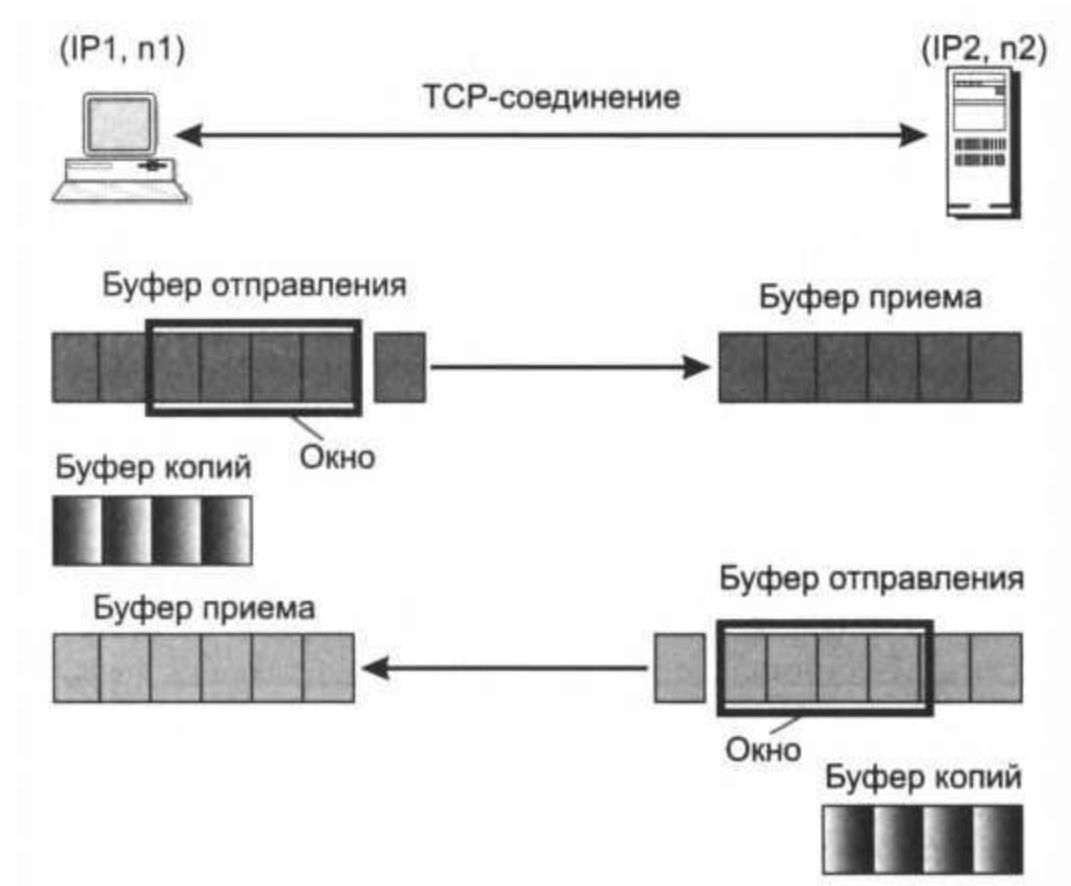
# Сегменты и поток байтов

- Одним из параметров, регулируемых в процессе обмена данными является начальный номер байта, с которого будет вестись отсчет в течение всего функционирования данного соединения. У каждой стороны — свой начальный номер.
- Нумерация байтов в пределах сегмента осуществляется, начиная от заголовка
- Когда отправитель посылает ТСР-сегмент, он помещает в поле последовательного номера номер первого байта данного сегмента, который служит идентификатором сегмента.

# Сегменты и поток байтов



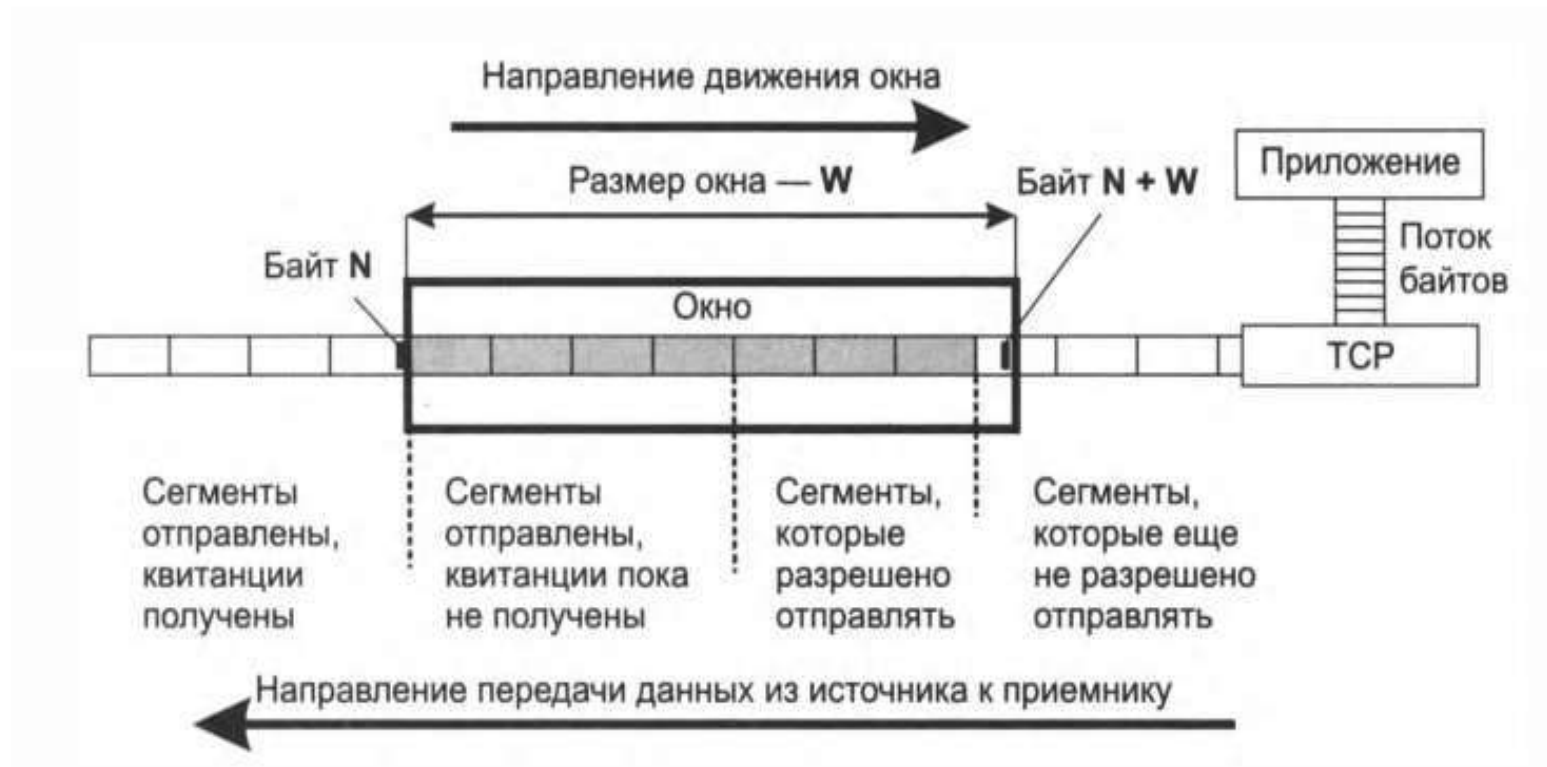
# Система окон в TCP



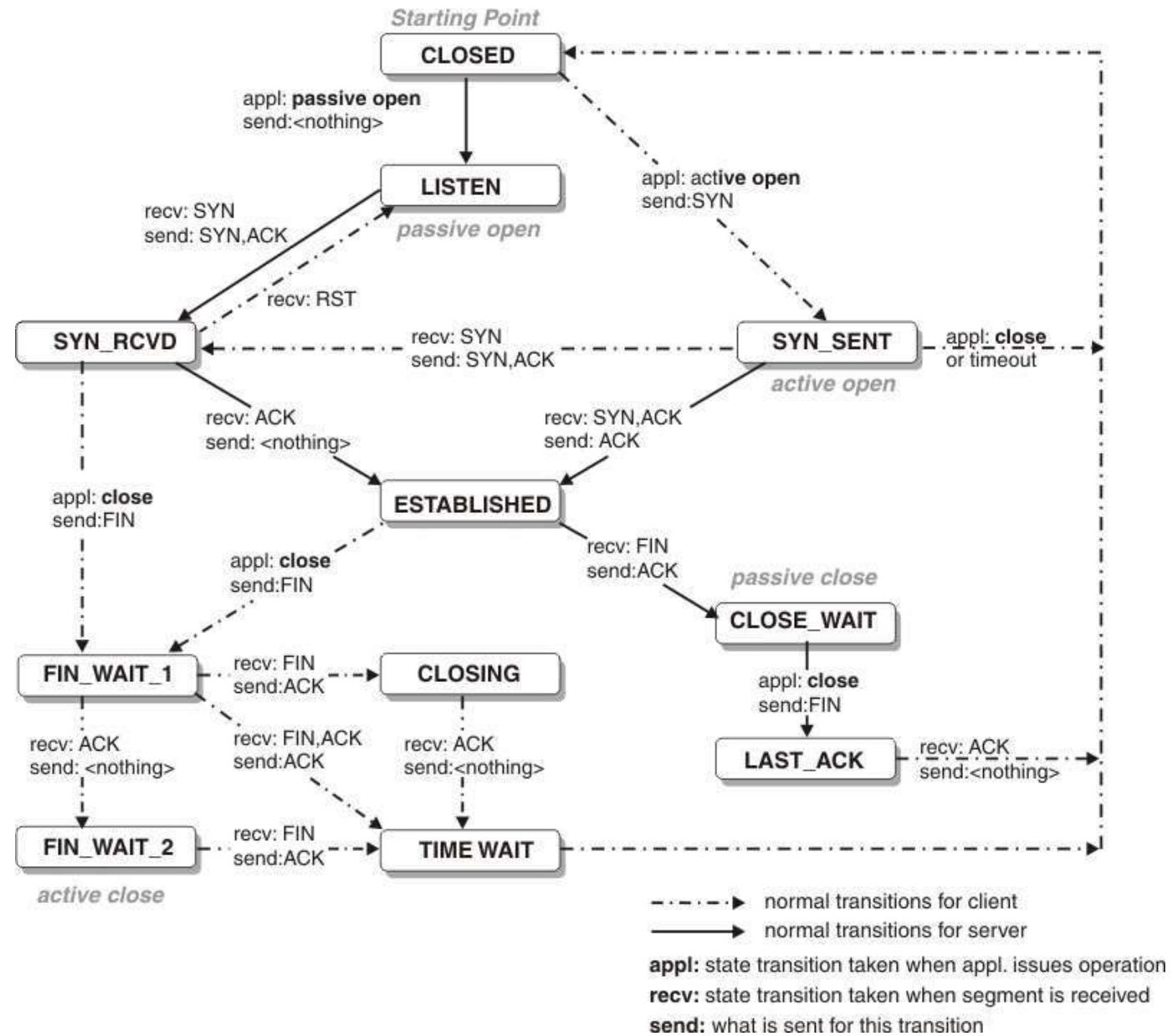
# Система окон в ТСР

- Первая граница отделяет сегменты, которые уже были отправлены и на которые уже пришли квитанции. Последняя квитанция пришла на байт с номером  $N$ .
- По другую сторону этой границы располагается окно размером  $W$  байт. Часть байтов, входящих в окно, составляют сегменты, которые также уже отправлены, но квитанции на которые пока не получены.
- Оставшаяся часть окна — это сегменты, которые пока не отправлены, но могут быть отправлены, так как входят в пределы окна.
- И наконец, последняя граница указывает на начало последовательности сегментов, ни один из которых не может быть отправлен до тех пор, пока не придет очередная квитанция и окно не будет сдвинуто вправо.

# Система окон в ТСР



# TCP FSM



# TCP, формат сегмента

- Состоит из поля данных и нескольких полей заголовка.
- *Поле данных* содержит фрагмент данных, передаваемых между процессами.
- Размер поля данных ограничивается величиной **MSS** (максимальный размер сегмента). Большой файл, как правило, разбивается на фрагменты размером MSS (кроме последнего фрагмента, который обычно имеет меньший размер).
- Интерактивные приложения, напротив, часто обмениваются данными, объем которых значительно меньше MSS. Например, приложения удаленного доступа к сети, подобные Telnet, могут передать транспортному уровню 1 байт данных.
- Поскольку обычно длина заголовка TCP-сегмента составляет 20 байт (что на 12 байт больше, чем в UDP), полный размер сегмента в этом случае равен 21 байт.

# ТСР, формат сегмента

<b>Порт источника (sours port)</b>				<b>Порт приемника (destination port)</b>			
<b>Последовательный номер (sequence number)</b> — номер первого байта данных в сегменте, определяет смещение сегмента относительно потока отправляемых данных							
<b>Подтвержденный номер (acknowledgement number)</b> — максимальный номер байта в полученном сегменте, увеличенный на единицу							
<b>Длина заголовка (hlen)</b>	<b>Резерв (reserved)</b>	<b>URG</b>	<b>ACK</b>	<b>PSH</b>	<b>RST</b>	<b>SYN</b>	<b>FIN</b>
							<b>Окно (window)</b> — количество байтов данных, ожидаемых отправителем данного сегмента, начиная с байта, номер которого указан в поле подтвержденного номера
<b>Контрольная сумма (checksum)</b>							<b>Указатель срочности (urgent pointer)</b> — указывает на конец данных, которые необходимо срочно принять, несмотря на переполнение буфера
<b>Параметры (options)</b> — это поле имеет переменную длину и может вообще отсутствовать, используется для решения вспомогательных задач, например для согласования максимального размера сегмента							
<b>Заполнитель (padding)</b> — это фиктивное поле может иметь переменную длину, используется для доведения размера заголовков до целого числа 32-битовых слов							



# Сегмент TCP, структура

- **Порт источника и порт получателя.** 16 - битные поля источника и получателя однозначно определяют посылающие и принимающие данные приложения или прикладные протоколы (программы).
- Номера портов источника и получателя в совокупности с IP - адресами сетевых компьютеров (в IP - заголовке) однозначно идентифицируют любое TCP - соединение и представляют сокет.

# Сегмент TSP, структура

- **Номер последовательности.** 32 - битное поле номера последовательности обозначает первый байт данных из области данных сегмента TSP.
- Оно соответствует смещению этого байта относительно начала потока данных.
- Каждый байт в потоке данных может быть идентифицирован при помощи номера последовательности.

# Сегмент TCP, структура

- Флаги. *Заголовок TCP содержит шесть однобитных полей флагов.*
- **Флаг UR** сообщает принимающему модулю TCP о наличии данных, требующих немедленной обработки
- **Флаг ACK** подтверждает правильность номера подтверждения в заголовке TCP
- **Флаг PSH** требует от принимающего модуля немедленно передать принятый сегмент данных приложению -получателю
- **Флаг RST** запрашивает у принимающего модуля TCP сброс соединения (прекращения работы приложений)
- **Флаг SYN** указывает принимающему модулю TCP необходимость синхронизации последовательности
- **Флаг FIN** сообщает принимающему модулю TCP о том, что источник закончил передачу данных.

# Сегмент ТСР, структура

- **Размер окна.** 16 - битное поле «размер окна» сообщает принимающему модулю ТСР количество байтов, которое собирается принять передатчик.
- Значение данного поля определяет размер скользящего окна. Как правило, оно равняется нескольким тысячам байтов.
- **Контрольная сумма ТСР.** 16 - битное поле контрольной суммы ТСР содержит сумму, вычисленную по всему сегменту ТСР, включая данные. Протокол ТСР требует от передатчика, чтобы он включил вычисленную контрольную сумму в поле, а от приемника - чтобы он вычислил ее повторно и сравнил с принятой.

# Мультиплексирование с установлением соединения

- Когда ТСР- сегмент поступает из сети на хост, тот использует IP-адрес отправителя, номер порта отправителя, IP-адрес получателя и номер порта получателя, чтобы направить (демультиплексировать) сегмент в соответствующий сокет.
- Два полученных ТСР сегмента будут иметь различные IP-адреса отправителя или номера портов отправителя, (исключая случай, когда ТСР-сегмент содержит первичный запрос на установление соединения) и окажутся направлены в два разных сокета

# Пример работы ТСР приложения

- На серверном приложении запущен “впускающий сокет”, ожидающий запрос на соединение
- ТСР-клиент создает сокет и отправляет сегмент с запросом на создание соединения, для этого используются функции `socket` и `connect`
- Запрос на установление логического соединения – ТСР-сегмент с портом назначения и комбинацией битов в заголовке ТСР-сегмента, создающего соединение.

# Пример работы TSP приложения

- Когда операционная система на хосте с запущенным серверным процессом получает входящий сегмент, который содержит запрос соединения с номером порта получателя 12 000, она направляет сегмент серверному процессу, ожидающему его, чтобы принять соединение на порт с номером 12 000.
- Затем серверный процесс создает новый сокет, с исп. функции `accept`.
- При обработке сегмента с запросом на установление логического соединения сервер использует четыре параметра: номер порта отправителя сегмента, IP-адрес хоста отправителя, номер порта получателя сегмента и собственный IP-адрес.

# Соотношение протоколов

Приложение	Прикладной протокол	Транспортный протокол
Электронная почта	SMTP	TCP
Доступ с удаленного терминала	Telnet	TCP
Web	HTTP	TCP
Передача файлов	FTP	TCP
Удаленный файловый сервер	NFS	UDP или TCP
Потоковое мультимедиа	Нестандартный	UDP или TCP
Интернет-телефония	Нестандартный	Как правило, UDP
Сетевое администрирование	SNMP	Как правило, UDP
Протокол маршрутизации	RIP	Как правило, UDP
Трансляция имен	DNS	Как правило, UDP



# SCTP

- первый стандарт - RFC 2960, затем дополнен в RFC 4960
- надежный транспортный протокол, который обеспечивает стабильную, упорядоченную (с сохранением порядка следования пакетов) передачу данных между двумя конечными точками (подобно TCP).
- Кроме того, протокол обеспечивает сохранение границ отдельных сообщений (подобно UDP).
- Однако в отличие от протоколов TCP и UDP протокол SCTP имеет дополнительные преимущества, такие как поддержка множественной адресации (multihoming) и многопоточности (multi-streaming) - каждая из этих возможностей увеличивает доступность узла передачи данных.

# Основные функции

- unicast-протокол, который обеспечивает обмен данными между двумя конечными точками.
- каждая из этих точек может быть представлена более чем 1 адресом IP.
- обеспечивает гарантию доставки, детектирование случаев отбрасывания данных, изменения порядка доставки, повреждения или дублирования данных, а также повторную передачу пакетов при возникновении необходимости.
- Обмен данными по протоколу SCTP происходит в полнодуплексном режиме.

# Основные функции

- Работа SCTP основана на обмене сообщениями и протокол поддерживает кадрирование границ отдельных сообщений. Протокол TCP ориентирован на обмен байтами и не хранит никаких неявных структур в передаваемых потоках данных.
- Скорость передачи по протоколу SCTP может адаптироваться подобно тому, как это происходит в TCP, – в зависимости от загрузки сети.
- может использоваться одновременно с TCP на общих каналах передачи данных.

# Поддержка многопоточности

- Поддержка множества одновременных потоков позволяет распределить между этими потоками передаваемую информацию так, чтобы каждый из потоков обеспечивал независимую упорядоченную доставку данных. При потере сообщения в любом из потоков это оказывает влияние лишь на данный поток, не затрагивая работу других потоков данных.
- Протокол TCP работает с одним потоком данных и обеспечивает для такого потока сохранение порядка доставки байтов из потока. Такой подход удобен для доставки файлов или записей, но он может приводить к дополнительным задержкам при потере информации в сети или нарушении порядка доставки пакетов. При возникновении таких ситуаций протокол TCP должен дожидаться доставки всех данных, требуемых для восстановления порядка.

# Поддержка многопоточности

- Для многих приложений строгое сохранение порядка доставки не является обязательным.
- В системах передачи телефонной сигнализации достаточно поддерживать порядок передачи для последовательности сообщений, которые воздействуют на один ресурс (например, для одного вызова или одного канала).
- Остальные сообщения слабо коррелируют между собой и могут доставляться с нарушением порядка.

# Поддержка многопоточности

- Другим примером является возможность использования множества потоков для доставки multimedia-документов (например, web- страниц) в рамках одной сессии.
- Поскольку такие документы состоят из разнотипных объектов разных размеров, многопотоковая доставка таких компонент не требует строгой упорядоченности. Однако использование множества потоков при доставке существенно повышает для потребителей привлекательность сервиса.

# Поддержка многопоточности

- Многопоточная передача поддерживается за счет устранения зависимости между передачей и доставкой данных. В частности, каждый блок полезной информации типа DATA (данные) использует два набора порядковых номеров. Номер TSN управляет передачей сообщений и детектированием их потери, а пара “идентификатор потока Stream ID – номер SSN” используется для управления порядком доставки потребителю полученных данных.
- Такая независимость механизмов нумерации позволяет получателю незамедлительно обнаруживать пропуски данных (например, в результате потери сообщения), а также видеть влияние потерянных данных на поток.
- Утрата сообщения вызывает появление пропуска в порядковых номерах SSN для потока, на который это сообщение оказывает влияние и не вызывает такого пропуска для других потоков. Следовательно, получатель может продолжать доставку незатронутых потоков, не дожидаясь повтора передачи утраченного сообщения

# Многодомность

- механизм предназначен для того, чтобы увеличить уровень устойчивости сети к выходам из строя интерфейсов на хосте и ускорить восстановление в случае сбоя в сети.
- Однако эффективность этого механизма падает, когда путь взаимодействия внутри ассоциации проходит через единую точку сбоя сети, к примеру, единственный канал или маршрутизатор, через которые должен проходить весь трафик ассоциации, или хост, обладающий всего одним интерфейсом.



# Механизм Cookie

- обеспечивает защиту от атак вслепую, когда атакующий генерирует множество блоков INIT с целью перегрузки сервера SCTP за счет расхода памяти и иных ресурсов, требуемых для обработки новых запросов INIT.
- Взамен выделения памяти для TCB (Transport Control Block) сервер создает параметр Cookie с информацией TCB, корректным временем жизни и сигнатурой для аутентификации. Этот параметр передается клиенту в сообщении INIT ACK. Поскольку сообщения INIT ACK всегда возвращаются по адресу отправителя запросов INIT, атакующий вслепую не будет получать Cookie. Легитимный клиент SCTP получит Cookie и вернет серверу блок COOKIE ECHO, по которому сервер SCTP может проверить корректность Cookie и использовать это значение для создания TCB.
- Поскольку параметр Cookie создается сервером и на сервере же проверяется, формат и секретный ключ Cookie знает только сервер и не возникает необходимости в передаче их через сеть клиенту.

# INIT Collision Resolution

- Поддержка многодомных хостов может приводить к изменению порядка доставки сообщений, для которых использовались разные пути.
- Эта проблема возникает на этапе создания ассоциации, когда работа без процедуры разрешения конфликтов может с легкостью привести к созданию множества параллельных ассоциаций между парой конечных точек.
- Для предотвращения этого SCTP включает множество процедур связывания параллельных попыток создания ассоциации с одной ассоциацией SCTP.

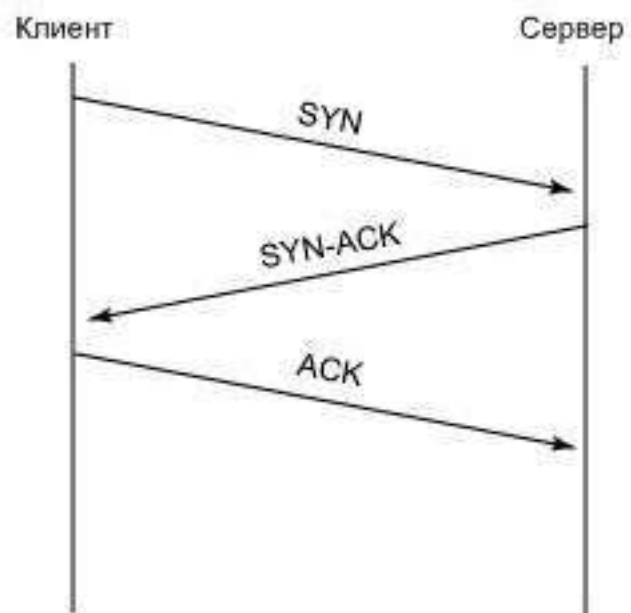
# Инициация

- Прежде, чем начнется обмен данными, два SCTP-хоста должны передать друг другу информацию о состоянии соединений (в том числе задействованные IP-адреса) с помощью четырехэтапной процедуры установки соединения
- Процедура, предусмотренная протоколом SCTP, позволяет защититься от DoS-атак. Получателю сообщения о намерении установить контакт INIT в четырехэтапной процедуре установки соединения не требуется сохранять никакую информацию о состоянии или резервировать какие-либо ресурсы.
- Вместо этого он посылает в ответ сообщение INIT-ACK, которое включает в себя специальную запись (cookie) состояния, содержащую всю информацию необходимую отправителю INIT-ACK для того, чтобы сформировать свое состояние.

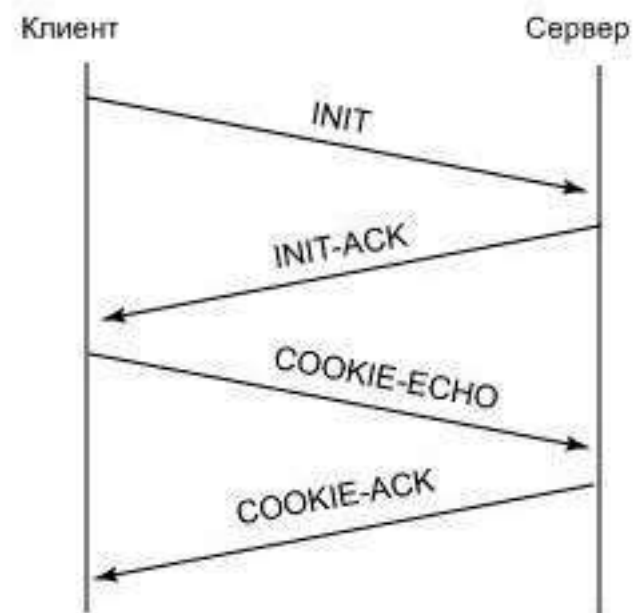
# Инициация

- INIT и INIT-ACK содержат несколько параметров:
  - список всех IP-адресов, которые станут частью ассоциации;
  - номер транспортной последовательности, который используется для надежной передачи данных;
  - тег инициации, который должен быть включен в каждый входящий пакет SCTP;
  - число выходящих потоков, запрашиваемых каждой из сторон;
  - число входящих потоков, которые способна поддерживать каждая из сторон.

# Инициация



Процедура установления соединения  
(трехэтапное квитирование) в протоколе TCP



Процедура установления соединения  
(четырёхэтапное квитирование) в протоколе SCTP

# Структура пакета

Биты	Биты 0-7	8-15	16-23	24-31
+0	Порт источника		Порт назначения	
32	Тег проверки			
64	Контрольная сумма			
96	Тип 1 блока	Флаги 1 блока	Длина 1 блока	
128	Данные 1 блока			
...	...			
...	Тип N блока	Флаги N блока	Длина N блока	
...	Данные N блока			

# Структура пакета

- Любой пакет SCTP в ассоциации, не содержащий специальный тег, при получении будет удален. Тег проверки защищает от старых, неактуальных пакетов, «отставших» от предыдущей ассоциации, а также от различных вторжений, позволяет избежать характерного для TCP состояния ожидания, при котором расходуются ресурсы и ограничивается общее число соединений, которые может поддерживать хост.
- Каждый из типов фрагмента включает в себя информацию заголовка TLV, который содержит тип фрагмента, флаги обработки доставки и длину поля.

# Структура пакета

- Кроме того, перед фрагментом DATA будет размещаться пользовательская информация о полезной нагрузке, состоящей из номера транспортной последовательности (TSN — transport sequence number), идентификатора потока, номера последовательности потока (SSN — stream sequence number).
- TSN и SSN — два разных номера последовательности для каждого фрагмента DATA. TSN используется для обеспечения надежности каждой ассоциации, а SSN для упорядочивания по потокам. Идентификатор потока отмечает отдельные сообщения в каждом потоке.



# Передача данных

- Хост SCTP посылает избранные подтверждения (фрагменты SACK) в ответ на каждый пакет SCTP, сопровождающий фрагменты DATA.
- Сообщение SACK полностью описывает состояние получателя так, что отправитель может принимать решение о повторной передаче в зависимости от того, что ему уже удалось получить.
- SCTP поддерживает алгоритмы быстрой повторной передачи и повторной передачи с тайм-аутом, аналогичные тем, которые применяются в TCP.

# Завершение

- SCTP использует трехэтапную процедуру установки соединения, которая имеет важное отличие от процедуры, применяемой в TCP: конечная точка TCP может инициировать процедуру отключения, сохраняя открытым соединение и получая новые данные от другого хоста.
- SCTP не поддерживает такого «наполовину закрытого» состояния, т. е. обе стороны не могут передавать новые данные на свой более высокий уровень, если инициирована последовательность постепенного отключения.

# Завершение соединения



# Источники

- В. Олифер, Н. Олифер “Компьютерные сети. Принципы, технологии, протоколы”
- Куроуз, Росс “Компьютерные сети. Нисходящий подход”
- Richard Stevens “TCP/IP Illustrated, vol. I”

Спасибо за внимание!