1)Компьютерные сети. Классификация компьютерных сетей. Топология.

Компьютерная сеть (ComputerNetwork) – это множество компьютеров, соединенных линиями связи и работающих под управлением специального программного обеспечения.

Под **линией связи** обычно понимают совокупность технических устройств, и физической среды, обеспечивающих передачу сигналов от передатчика к приемнику. В реальной жизни примерами линий связи могут служить участки кабеля и усилители, обеспечивающие передачу сигналов между коммутаторами телефонной сети. На основе линий связи строятся каналы связи.

Каналом связи обычно называют систему технических устройств и линий связи, обеспечивающую передачу информации между абонентами. Соотношение между понятиями "канал" и "линия" описывается следующим образом: канал связи может включать в себя несколько разнородных линий связи, а одна линия связи может использоваться несколькими каналами

Главной целью объединения компьютеров в сеть является предоставление пользователям возможности доступа к различным информационным ресурсам (например, документам, программам, базам данных и т.д.), распределенным по этим компьютерам и их совместного использования.

Классификация

1)По территориальной распространенности

- **BAN** (BodyAreaNetwork — нательная компьютерная сеть) — сеть надеваемых или имплантированных компьютерных устройств.
- **PAN** (PersonalAreaNetwork) — персональная сеть, предназначенная для взаимодействия различных устройств, принадлежащих одному владельцу.
- **LAN** (**ЛВС,**LocalAreaNetwork) — локальные сети, имеющие замкнутую инфраструктуру до выхода на поставщиков услуг. Термин «LAN» может описывать и маленькую офисную сеть, и сеть уровня большого завода, занимающего несколько сотен гектаров. Зарубежные источники дают даже близкую оценку — около шести миль (10 км) в радиусе. Локальные сети являются сетями закрытого типа, доступ к ним разрешён только ограниченному кругу пользователей, для которых работа в такой сети непосредственно связана с их профессиональной деятельностью.
- **CAN** (CampusAreaNetwork) — кампусная сеть, объединяет локальные сети близко расположенных зданий.
- **MAN** (MetropolitanAreaNetwork) — городские сети между учреждениями в пределах одного или нескольких городов, связывающие много локальных вычислительных сетей.
- **WAN** (WideAreaNetwork) — глобальная сеть, покрывающая большие географические регионы, включающие в себя как локальные сети, так и прочие телекоммуникационные сети и устройства. Пример WAN — сети с коммутацией пакетов (Framerelay), через которую могут «разговаривать» между собой различные компьютерные сети. Глобальные сети являются открытыми и ориентированы на обслуживание любых пользователей.

- Термин «корпоративная сеть» также используется в литературе для обозначения объединения нескольких сетей, каждая из которых может быть построена на различных технических, программных и информационных принципах.

2)По архитектуре

- Клиент-сервер
- Одноранговая сеть (децентрализованная или пиринговая сеть)

3)По типу сетевой топологии

- Шина
- Кольцо
- Двойное кольцо
- Звезда
- Ячеистая
- Решётка
- Дерево
- Смешанная топология
- FatTree

4)По типу среды передачи

Проводные (телефонный провод, коаксиальный кабель, витая пара, волоконно-оптический кабель)
Беспроводные (передачей информации по радиоволнам в определенном частотном диапазоне)

5)По функциональному назначению

Сети хранения данных
Серверные фермы
Сети управления процессом
Сети SOHO, домовые сети

6)По скорости передачи

низкоскоростные (до 10 Мбит/с),
среднескоростные (до 100 Мбит/с),
высокоскоростные (свыше 100 Мбит/с);

7)По сетевым операционным системам

На основе Windows
На основе UNIX
На основе NetWare
На основе Cisco

8)По необходимости поддержания постоянного соединения

Пакетная сеть, например, Фидонет и UUCP
Онлайновая сеть, например, Интернет и GSM

ТОПОЛОГИЯ КОМПЬЮТЕРНЫХ СЕТЕЙ

Существует бесконечное число способов соединения компьютеров.

Топология сети – геометрическая форма и физическое расположение компьютеров по отношению к друг другу. Топология сети позволяет сравнивать и классифицировать различные сети. Различают три основных вида топологии:

1) Звезда;

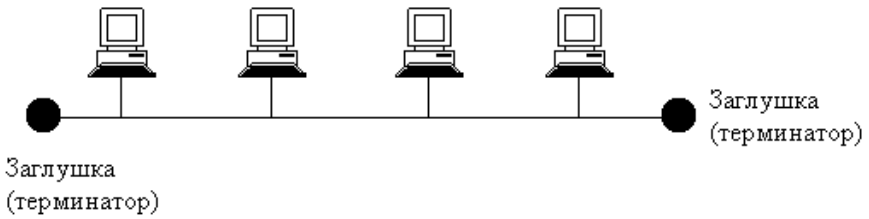
2) Кольцо;

3) Шина.

ШИННАЯ ТОПОЛОГИЯ

При построении сети по шинной схеме каждый компьютер присоединяется к общему кабелю, на концах которого устанавливаются терминаторы.

Сигнал проходит по сети через все компьютеры, отражаясь от конечных терминаторов.



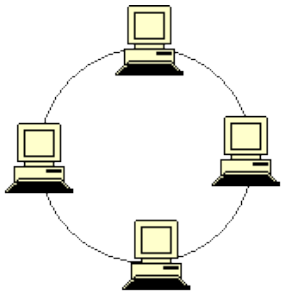
Шина проводит сигнал из одного конца сети к другому, при этом каждая рабочая станция проверяет адрес послания, и, если он совпадает с адресом рабочей станции, она его принимает. Если же адрес не совпадает, сигнал уходит по линии дальше. Если одна из подключённых машин не работает, это не сказывается на работе сети в целом, однако если соединения любой из подключенных машин нарушается из-за повреждения контакта в разъёме или обрыва кабеля, неисправности терминатора, то весь сегмент сети (участок кабеля между двумя терминаторами) теряет целостность, что приводит к нарушению функционирования всей сети.

Достоинства	Недостатки
1) Отказ любой из рабочих станций не влияет на работу всей сети. 2) Простота и гибкость соединений. 3) Недорогой кабель и разъемы. 4) Необходимо небольшое количество кабеля. 5) Прокладка кабеля не вызывает	1) Разрыв кабеля, или другие неполадки в соединении может исключить нормальную работу всей сети. 2) Ограниченная длина кабеля и количество рабочих станций. 3) Трудно обнаружить дефекты соединений. 4) Невысокая производительность.

особых сложностей.	5) При большом объеме передаваемых данных главный кабель может не справляться с потоком информации, что приводит к задержкам.

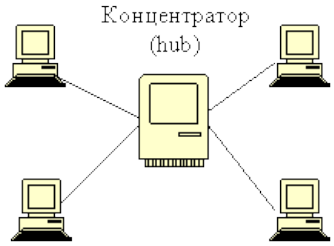
ТОПОЛОГИЯ «КОЛЬЦО»

Эта топология представляет собой последовательное соединение компьютеров, когда последний соединён с первым. Сигнал проходит по кольцу от компьютера к компьютеру в одном направлении. Каждый компьютер работает как повторитель, усиливая сигнал и передавая его дальше. Поскольку сигнал проходит через каждый компьютер, сбой одного из них приводит к нарушению работы всей сети.



ТОПОЛОГИЯ «ЗВЕЗДА»

Топология «Звезда» - схема соединения, при которой каждый компьютер подсоединяется к сети при помощи отдельного соединительного кабеля. Один конец кабеля соединяется с гнездом сетевого адаптера, другой подсоединяется к центральному устройству, называемому концентратором (hub).



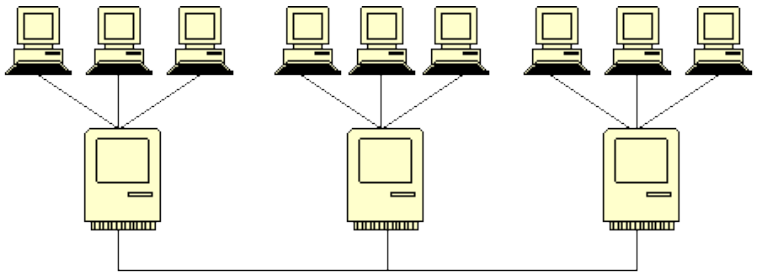
Устанавливать сеть топологии «Звезда» легко и недорого. Число узлов, которые можно подключить к концентратору, определяется возможным количеством портов самого концентратора, однако имеются ограничения по числу узлов (максимум 1024). Рабочая группа, созданная по данной схеме может функционировать независимо или может быть связана с другими рабочими группами.

Достоинства	Недостатки
1) Подключение новых рабочих станций не вызывает особых затруднений. 2) Возможность мониторинга сети и централизованного управления сетью 3) При использовании централизованного управления	1) Отказ концентратора приводит к отключению от сети всех рабочих станций, подключенных к ней. 2) Достаточно высокая стоимость реализации, т.к. требуется большое количество кабеля.

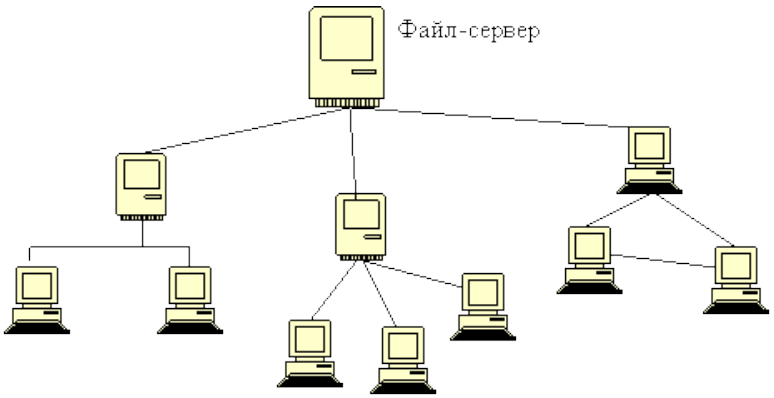
сетью локализация дефектов соединений максимально упрощается. 4)Хорошая расширяемость и модернизация.	

КОМБИНИРОВАННЫЕ ТОПОЛОГИИ

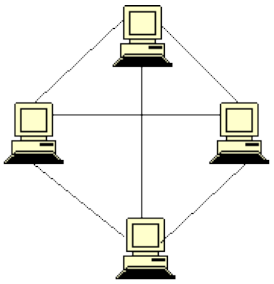
1. «Звезда-Шина» - несколько сетей с топологией звезда объединяются при помощи магистральной линейной шины.



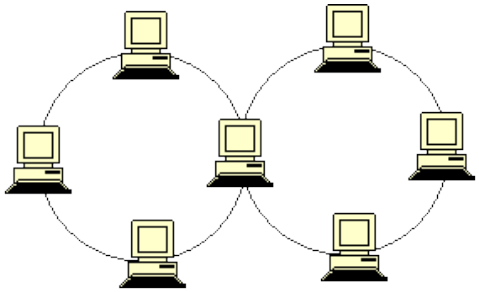
2. Древовидная структура.



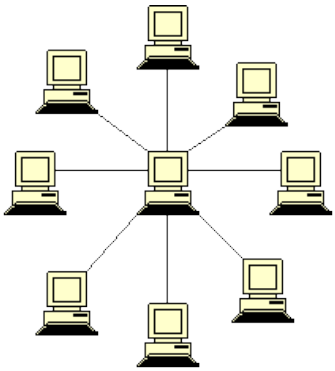
3. «Каждый с каждым»



4. Пересекающиеся кольца



5. «Снежинка»



Локальные сети при разработке, как правило, имеют симметричную топологию, глобальные— неправильную.

2) Протоколы. Модель ISO/OSI.

Протокол передачи данных — 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.

Стандартизированный*протокол передачи данных* также позволяет разрабатывать интерфейсы (уже на *физическом уровне*), не привязанные к конкретной аппаратной платформе и производителю (например, USB, Bluetooth).

Сигнальный протокол используется для управления соединением — например, установки, переадресации, разрыва связи. Примеры протоколов: RTSP, SIP. Для передачи данных используются такие протоколы как RTP.

Сетево́йпротоко́л — набор правил и действий (очерёдности действий), позволяющий осуществлять соединение и обмен данными между двумя и более включёнными в сеть устройствами.

Разные протоколы зачастую описывают лишь разные стороны одного типа связи. Названия «протокол» и «стек протоколов» также указывают на программное обеспечение, которым реализуется протокол.

Новые протоколы для Интернета определяются IETF, а прочие протоколы — IEEE или ISO. ITU-T занимается телекоммуникационными протоколами и форматами.

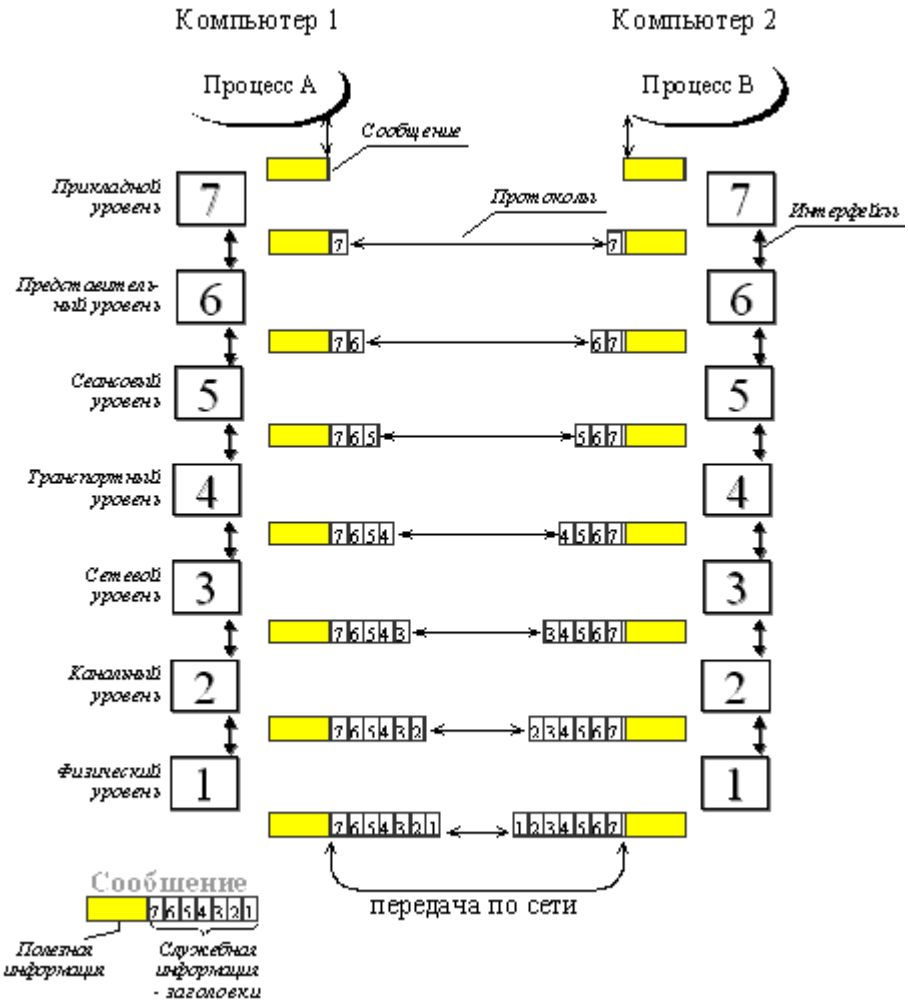
Наиболее распространённой системой классификации сетевых протоколов является так называемая модель OSI, в соответствии с которой протоколы делятся на 7 уровней по своему назначению — от физического (формирование и распознавание электрических или других сигналов) до прикладного (интерфейс программирования приложений для передачи информации приложениями).

Сетевые протоколы предписывают правила работы компьютерам, которые подключены к сети. Они строятся по многоуровневому принципу. Протокол некоторого уровня определяет одно из технических правил связи. В настоящее время для сетевых протоколов используется модель OSI (OpenSystemInterconnection — взаимодействие открытых систем, ВОС).

Модель ISO/OSI

Международная Организация по Стандартам (InternationalStandardsOrganization, ISO) разработала модель, которая четко определяет различные уровни взаимодействия систем, дает им стандартные имена и указывает, какую работу должен делать каждый уровень. Эта модель называется моделью взаимодействия открытых систем (OpenSystemInterconnection, OSI) или моделью ISO/OSI.

В модели OSI взаимодействие делится на семь уровней или слоев (рис. 1.1). Каждый уровень имеет дело с одним определенным аспектом взаимодействия. Таким образом, проблема взаимодействия декомпозирована на 7 частных проблем, каждая из которых может быть решена независимо от других. Каждыйуровеньподдерживаетинтерфейсы с выше- и нижележащимиуровнями.



Функции уровней модели ISO/OSI

1)*Физический уровень.* Этот уровень имеет дело с передачей битов по физическим каналам, таким, например, как коаксиальный кабель, витая пара или оптоволоконный кабель. К этому уровню имеют отношение характеристики физических сред передачи данных, такие как полоса пропускания, помехозащищенность, волновое сопротивление и другие. На этом же уровне определяются характеристики электрических сигналов, такие как требования к фронтам импульсов, уровням напряжения или тока передаваемого сигнала, тип кодирования, скорость передачи сигналов. Кроме этого, здесь стандартизуются типы разъемов и назначение каждого контакта.

2)*Канальный уровень.*Одной из задач канального уровня является проверка доступности среды передачи. Другой задачей канального уровня является реализация механизмов обнаружения и коррекции ошибок. Для этого на канальном уровне биты группируются в наборы, называемые кадрами (frames). Канальный 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. Когда кадр приходит, получатель снова вычисляет контрольную сумму полученных данных и сравнивает результат с контрольной суммой из кадра. Если они совпадают, кадр считается правильным и принимается. Если же контрольные суммы не совпадают, то фиксируется ошибка.

3)Сетевой уровень отвечает за адресацию и доставку сообщений. Сообщения сетевого уровня принято называть пакетами (packets). 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. Сети соединяются между собой специальными устройствами, называемыми маршрутизаторами. Маршрутизатор - это устройство, которое собирает информацию о топологии межсетевых соединений и на ее основании пересылает пакеты сетевого уровня в сеть назначения. Для того, чтобы передать сообщение от отправителя, находящегося в одной

сети, получателю, находящемуся в другой сети, нужно совершить некоторое количество транзитных передач (hops) между сетями, каждый раз выбирая подходящий маршрут. Таким образом, маршрут представляет собой последовательность маршрутизаторов, через которые проходит пакет.

4)*Транспортный уровень.* На пути от отправителя к получателю пакеты могут быть искажены или утеряны. Работа транспортного уровня заключается в том, чтобы обеспечить приложениям или верхним уровням стека - прикладному и сеансовому - передачу данных с той степенью надежности, которая им требуется. Модель OSI определяет пять классов сервиса, предоставляемых транспортным уровнем. Эти виды сервиса отличаются качеством предоставляемых услуг: срочностью, возможностью восстановления прерванной связи, наличием средств мультиплексирования нескольких соединений между различными прикладными протоколами через общий транспортный протокол, а главное - способностью к обнаружению и исправлению ошибок передачи, таких как искажение, потеря и дублирование пакетов.

5)*Сеансовый уровень.* Сеансовый уровень 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 с начала. На практике немногие приложения используют сеансовый уровень, и он редко реализуется.

6)*Уровень представления.* Этот уровень обеспечивает гарантию того, что информация, передаваемая прикладным уровнем, будет понятна прикладному уровню в другой системе. При необходимости уровень представления выполняет преобразование форматов данных в некоторый общий формат представления, а на приеме, соответственно, выполняет обратное преобразование. Таким образом, прикладные уровни могут преодолеть, например, синтаксические различия в представлении данных. На этом уровне может выполняться шифрование и дешифрование данных, благодаря которому секретность обмена данными обеспечивается сразу для всех прикладных сервисов. Примером протокола, работающего на уровне представления, является протокол SecureSocketLayer (SSL), который обеспечивает секретный обмен сообщениями для протоколов прикладного уровня стека TCP/IP.

7)*Прикладной уровень.* Прикладной уровень - это в действительности просто 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. Единица данных, которой оперирует прикладной уровень, обычно называется *сообщением (message)*.

3)Протоколы прикладного уровня. Протокол HTTP

Протокол прикладного уровня (англ. *Applicationlayer*) — протокол верхнего (7-го) уровня сетевой модели OSI, обеспечивает взаимодействие сети и пользователя. Уровень разрешает приложениям пользователя иметь доступ к сетевым службам, таким, как обработчик запросов к базам данных, доступ к файлам, пересылке электронной почты. Также отвечает за передачу служебной информации, предоставляет приложениям информацию об ошибках и формирует запросы к уровню представления. Пример: HTTP, POP3, SMTP.

Протоколы

- 9P
- BitTorrent
- BOOTP
- DNS
- FTP
- HTTP
- NFS
- POP, POP3
- IMAP
- RTP
- SMTP
- SNMP
- SPDY
- Telnet

HTTP (HyperTextTransferProtocol — протокол передачи гипертекста) — символьно-ориентированный клиент-серверный протокол прикладного уровня без сохранения состояния, используемый сервисом WorldWideWeb.

Основным объектом манипуляции в HTTP является ресурс, на который указывает URI (UniformResourceIdentifier – уникальный идентификатор ресурса) в запросе клиента. Основными ресурсами являются хранящиеся на сервере файлы, но ими могут быть и другие логические (напр. каталог на сервере) или абстрактные объекты (напр. ISBN). Протокол HTTP позволяет указать способ представления (кодирования) одного и того же ресурса по различным параметрам: mime-типу, языку и т. д. Благодаря этой возможности клиент и веб-сервер могут обмениваться двоичными данными, хотя данный протокол является текстовым.**— протокол** передачи гипертекста) — символьно-ориентированный клиент-серверный протокол прикладного уровня без сохранения состояния, используемый сервисом WorldWideWeb.

Основным объектом манипуляции в HTTP является ресурс, на который указывает URI (*UniformResourceIdentifier* – уникальный идентификатор ресурса) в запросе клиента. Основными ресурсами являются хранящиеся на сервере файлы, но ими могут быть и другие логические (напр. каталог на сервере) или абстрактные объекты (напр. ISBN). Протокол HTTP позволяет указать способ представления (кодирования) одного и того же ресурса по различным параметрам: mime-типу, языку и т. д. Благодаря этой возможности клиент и веб-сервер могут обмениваться двоичными данными, хотя данный протокол является текстовым.

Структура протокола

Структура протокола определяет, что каждое HTTP-сообщение состоит из трёх частей которые передаются в следующем порядке:

1. Стартовая строка (англ. Startingline) — определяет тип сообщения;
2. Заголовки (англ. Headers) — характеризуют тело сообщения, параметры передачи и прочие сведения;
3. Тело сообщения (англ. MessageBody) — непосредственно данные сообщения. Обязательно должно отделяться от заголовков пустой строкой.

Стартовая строка HTTP

Стартовая строка является обязательным элементом, так как указывает на тип запроса/ответа, заголовки и тело сообщения могут отсутствовать.

Стартовые строки различаются для запроса и ответа. **Строка запроса** выглядит так:

```
Метод URI HTTP/Версия протокола
```

Примерзапроса:

```
GET /web-programming/index.html HTTP/1.1
```

Стартовая **строка ответа** сервера имеет следующий формат:

```
HTTP/Версия КодСостояния [Пояснение]
```

Например, на предыдущий наш запрос клиентом данной страницы сервер ответил строкой:

```
HTTP/1.1 200 Ok
```

Метод	Краткое описание
OPTIONS	Используется для определения возможностей веб-сервера или параметров соединения для конкретного ресурса. Предполагается, что запрос клиента может содержать тело сообщения для указания интересующих его сведений. Формат тела и порядок работы с ним в настоящий момент не определён. Сервер пока должен его игнорировать. Аналогичная ситуация и с телом в ответе сервера. Для того чтобы узнать возможности всего сервера, клиент должен указать в URI звёздочку — «*». Запросы «OPTIONS * HTTP/1.1» могут также применяться для проверки работоспособности сервера (аналогично «пингованию») и тестирования на предмет поддержки сервером протокола HTTP версии 1.1. Результат выполнения этого метода не кэшируется.
GET	Используется для запроса содержимого указанного ресурса. С помощью метода GET можно также начать какой-либо процесс. В этом случае в тело ответного сообщения следует включить информацию о ходе выполнения процесса. Клиент может передавать параметры выполнения запроса в URI целевого ресурса после символа «?»: GET /path/resource?param1=value1¶m2=value2 HTTP/1.1 Согласно стандарту HTTP, запросы типа GET считаются идемпотентными[4] — многократное повторение одного и того же запроса GET должно приводить к одинаковым результатам (при условии, что сам ресурс не изменился за время между запросами). Это позволяет кэшировать ответы на запросы GET.
HEAD	Аналогичен методу GET, за исключением того, что в ответе сервера отсутствует тело. Запрос HEAD обычно применяется для извлечения метаданных, проверки наличия ресурса (валидация URL) и чтобы узнать, не изменился ли он с момента последнего обращения. Заголовки ответа могут кэшироваться. При несовпадении метаданных ресурса с соответствующей информацией в кэше копия ресурса помечается как устаревшая.

POST	Применяется для передачи пользовательских данных заданному ресурсу. Например, в блогах посетители обычно могут вводить свои комментарии к записям в HTML-форму, после чего они передаются серверу методом POST и он помещает их на страницу. При этом передаваемые данные (в примере с блогами — текст комментария) включаются в тело запроса. Аналогично с помощью метода POST обычно загружаются файлы. В отличие от метода GET, метод POST не считается идемпотентным[4], то есть многократное повторение одних и тех же запросов POST может возвращать разные результаты (например, после каждой отправки комментария будет появляться одна копия этого комментария).
PUT	Применяется для загрузки содержимого запроса на указанный в запросе URI. Если по заданному URI не существовало ресурса, то сервер создаёт его и возвращает статус 201 (Created). Если же был изменён ресурс, то сервер возвращает 200 (Ok) или 204 (NoContent). Сервер не должен игнорировать некорректные заголовки Content-* передаваемые клиентом вместе с сообщением. Если какой-то из этих заголовков не может быть распознан или не допустим при текущих условиях, то необходимо вернуть код ошибки 501 (NotImplemented). Фундаментальное различие методов POST и PUT заключается в понимании предназначений URI ресурсов. Метод POST предполагает, что по указанному URI будет производиться обработка передаваемого клиентом содержимого. Используя PUT, клиент предполагает, что загружаемое содержимое соответствуют находящемуся по данному URI ресурсу.
PATCH	Аналогично PUT, но применяется только к фрагменту ресурса.
DELETE	Удаляет указанный ресурс.
TRACE	Возвращает полученный запрос так, что клиент может увидеть, что промежуточные сервера добавляют или изменяют в запросе.
LINK	Устанавливает связь указанного ресурса с другими.
UNLINK	Убирает связь указанного ресурса с другими.

Наиболее востребованными являются методы GET и POST — на человеко-ориентированных ресурсах, POST — роботами поисковых машин и оффлайн-браузерами.

Прокси - это транзитный сервер, перенаправляющий HTTP-трафик. Прокси-серверы используются для ускорения выполнения запросов путем кэширования веб-страниц. В локальной сети применяется как межсетевой экран и средство управления HTTP-трафиком (например, для блокирования доступа к некоторым ресурсам). В Интернете прокси часто используют для *анонимизации запросов* - в этом случае веб-сервер получает ip-адрес прокси-сервера, а не реального клиента.

Код состояния информирует клиента о результатах выполнения запроса и определяет его дальнейшее поведение. Набор кодов состояния является стандартом, и все они описаны в соответствующих документах RFC.

Каждый код представляется целым трехзначным числом. Первая цифра указывает на класс состояния, последующие - порядковый номер состояния. За кодом ответа обычно следует краткое описание на английском языке.

1xx Informational (Информационный)	Примеры ответов сервера: 100 Continue (Продолжать) 101 SwitchingProtocols (Переключение протоколов) 102 Processing (Идёт обработка)
2xx Success (Успешно)	200 OK (Успешно). 201 Created (Создано) 202 Accepted (Принято) 204 NoContent (Нет содержимого) 206 PartialContent (Частичное содержимое)
3xx Redirection (Перенаправление)	300 MultipleChoices (Множественный выбор) 301 MovedPermanently (Перемещено навсегда) 304 NotModified (Не изменялось)
4xx ClientError (Ошибка клиента)	401 Unauthorized (Неавторизован) 402 PaymentRequired (Требуется оплата) 403 Forbidden (Запрещено) 404 NotFound (Не найдено) 405 MethodNotAllowed (Метод не поддерживается) 406 NotAcceptable (Не приемлемо) 407 ProxyAuthenticationRequired (Требуется аутентификация прокси)
5xx ServerError (Ошибка сервера)	500 Internal Server Error (Внутренняяошибкасервера) 502 Bad Gateway (Плохойшлюз) 503 Service Unavailable (Сервиснедоступен) 504 GatewayTimeout (Шлюз не отвечает)

Заголовок	Группа	Краткое описание
Allow	Entity	Список методов, применимых к запрашиваемому ресурсу.
Content-Encoding	Entity	Применяется при необходимости перекодировки содержимого (например, gzip/deflated).
Content-Language	Entity	Локализация содержимого (язык(и))
Content-Length	Entity	Размер тела сообщения (в октетах)
Content-Range	Entity	Диапазон (используется для поддержания многопоточной загрузки или дозагрузки)
Content-Type	Entity	Указывает тип содержимого (mime-type, например text/html).Часто включает указание на таблицу символов локали (charset)
Expires	Entity	Дата/время, после которой ресурс считается устаревшим. Используется прокси-серверами
Last-Modified	Entity	Дата/время последней модификации сущности
Cache-Control	General	Определяет директивы управления механизмами кэширования. Для прокси-серверов.
Connection	General	Задает параметры, требуемые для конкретного соединения.
Date	General	Дата и время формирования сообщения
Pragma	General	Используется для специальных указаний, которые могут (опционально) применяется к любому получателю по всей цепочке запросов/ответов (например, pragma: no-cache).
Transfer-Encoding	General	Задает тип преобразования, применимого к телу сообщения. В отличие от Content-Encoding этот заголовок распространяется на все сообщение, а не только на сущность.
Via	General	Используется шлюзами и прокси для отображения промежуточных протоколов и узлов между клиентом и веб-сервером.
Warning	General	Дополнительная информация о текущем статусе, которая не может быть представлена в сообщении.
Accept	Request	Определяет применимые типы данных, ожидаемых в ответе.
Accept-Charset	Request	Определяет кодировку символов (charset) для данных, ожидаемых в ответе.
Accept-Encoding	Request	Определяет применимые форматы кодирования/декодирования содержимого (напр, gzip)
Accept-Language	Request	Применимые языки. Используется для согласования передачи.
Authorization	Request	Учетные данные клиента, запрашивающего ресурс.
From	Request	Электронный адрес отправителя
Host	Request	Имя/сетевой адрес [и порт] сервера. Если порт не указан, используется 80.
If-Modified-Since	Request	Используется для выполнения условных методов (Если-Изменился...). Если запрашиваемый ресурс изменился, то он передается с сервера, иначе - из кэша.
Max-Forwards	Request	Представляет механиз ограничения количества шлюзов и прокси при использовании методов TRACE и OPTIONS.
Proxy-Authorization	Request	Используется при запросах, проходящих через прокси, требующие авторизации
Referer	Request	Адрес, с которого выполняется запрос. Этот заголовок отсутствует, если переход выполняется из адресной строки или, например, по ссылке из js-скрипта.
User-Agent	Request	Информация о пользовательском агенте (клиенте)
Location	Response	Адрес перенаправления
Proxy-	Response	Сообщение о статусе с кодом 407.

Authenticate		
Server	Response	Информация о программном обеспечении сервера, отвечающего на запрос (это может быть как веб- так и прокси-сервер).

4)Язык разметки гипертекста. Теги и атрибуты.

HyperTextMarkupLanguage (HTML) — язык разметки гипертекста — предназначен для написания гипертекстовых документов, публикуемых в WorldWideWeb.

Гипертекстовый документ — это текстовый файл, имеющий специальные метки, называемые тегами, которые впоследствии опознаются браузером и используются им для отображения содержимого файла па экране компьютера.

С помощью этих меток можно выделять заголовки документа, изменять цвет, размер и начертание букв, вставлять графические изображения и таблицы. Но основным преимуществом гипертекста перед обычным текстом является возможность добавления к содержимому документа гиперссылок — специальных конструкций языка HTML, которые позволяют щелчком мыши перейти к просмотру другого документа.

HTML-документ состоит из двух частей: собственно текста, т. е. данных, составляющих содержимое документа, и *тегов* — специальных конструкций языка HTML, используемых для разметки документа и управляющих его отображением. Теги языка HTML определяют, в каком виде будет представлен текст, какие его компоненты будут исполнять роль гипертекстовых ссылок, какие графические или мультимедийные объекты должны быть включены в документ.

Графическая и звуковая информация, включаемая в HTML-документ, хранится в отдельных файлах. Программы просмотра HTML-документов (**браузеры**) интерпретируют флаги разметки и располагают текст и графику на экране соответствующим образом. Для файлов, содержащих HTML-документы приняты расширения **.htm** или **.html**.

В большинстве случаев теги используются парами. Пара состоит из открывающего <имя_тега> и закрывающего </имя_тега> тегов. Действие любого парного тега начинается с того места, где встретился открывающий тег, и заканчивается при встрече соответствующего закрывающего тега. Часто пару, состоящую из открывающего и закрывающего тегов, называют *контейнером*, а часть текста, окаймленную открывающим и закрывающим тегом, — *элементом*.

Последовательность символов, составляющая текст может состоять из пробелов, табуляций, символов перехода на новую строку, символов возврата каретки, букв, знаков препинания, цифр, и специальных символов (например #, +, $, @), за исключением следующих четырех символов, имеющих в HTML специальный смысл: < (меньше), > (больше), & (амперсанд) и " (двойная кавычка). Если необходимо включить в текст какой-либо из этих символов, то следует закодировать его особой последовательностью символов.

Структура HTML-документа

Самым главным из тегов HTML является одноименный тег *<html>*. Он всегда открывает документ, так же, как тег *</html>* должен непременно стоять в последней его строке. Эти теги обозначают, что находящиеся между ними строки представляют единый гипертекстовый документ. Без этих тегов браузер или другая программа просмотра не в состоянии идентифицировать формат документа и правильно его интерпретировать.

HTML-документ состоит из двух частей: заголовок (head) и тела (body), расположенных в следующем порядке:

```
<html>
<head> Заголовок документа </head>
<body> Тело документа </body>
</html>
```

Чаще всего в заголовок документа включают парный тег `<title>... </title>`, определяющий название документа. Многие программы просмотра используют его как заголовок окна, в котором выводят документ. Программы, индексирующие документы в сети Интернет, используют название для идентификации страницы. Хорошее название должно быть достаточно длинным для того, чтобы можно было корректно указать соответствующую страницу, и в то же время оно должно помещаться в заголовке окна. Название документа вписывается между открывающим и закрывающим тегами.

Тело документа является обязательным элементом, так как в нем располагается весь материал документа. Тело документа размещается между тегами `<body>` и `</body>`. Все, что размещено между этими тегами, интерпретируется браузером в соответствии с правилами языка HTML позволяющими корректно отображать страницу на экране монитора.

Текст в HTML разделяется на абзацы при помощи тега `<p>`. Он размещается в начале каждого абзаца, и программа просмотра, встречая его, отделяет абзацы друг от друга пустой строкой. Использование закрывающего тега `</p>` необязательно.

Если требуется «разорвать» текст, перенеся его остаток на новую строку, при этом, не выделяя нового абзаца, используется тег разрыва строки `<BR>`. Он заставляет программу просмотра выводить стоящие после него символы с новой строки. В отличие от тега абзаца, тег `<BR>` не добавляет пустую строку. У этого тега нет парного закрывающего тега.

Язык HTML поддерживает **логическое н физическое форматирование содержимого документа**. Логическое форматирование указывает на назначение данного фрагмента текста, а физическое форматирование задает его внешний вид.

При использовании *логического форматирования* текста браузером выделяются различные части текста в соответствии со структурой документа. Чтобы отобразить название, используется один из тегов заголовка. Заголовки в типичном документе разделяются по уровням. Язык HTML позволяет задать шесть уровней заголовков: h1 (заголовок первого уровня), h2, h3, h4, h5 и h6. Заголовок первого уровня имеет обычно больший размер и насыщенность по сравнению с заголовком второго уровня. Пример использования тегов заголовков:

```
<h1>l. Название главы</h1>
<h2>l.l. Название раздела</h2>
```

Теги *физического форматирования* непосредственно задают вид текста на экране браузера, например пара `<b></b>` выделяет текст полужирным начертанием, `<u></u>` задает подчеркивание текста, `<font></font>` управляет шрифтом текста.

Тег `<img>` вставляет изображение в документ, как если бы оно было просто одним большим символом. Пример применения тега:

```
<imgsrc = "picture.gif">
```

Для создания **гипертекстовой ссылки** используется пара тегов <a>... . Фрагмент текста, изображение или любой другой объект, расположенный между этими тегами, отображается в окне браузера как гипертекстовая ссылка. Активация такого объекта приводит к загрузке в окно браузера нового документа или к отображению другой части текущей Web-страницы. Гипертекстовая ссылка формируется с помощью выражения:

```
<a href = "document.html">ссылка на документ</a>
```

Href здесь является обязательным атрибутом, значение которого и есть URL-адрес запрашиваемого ресурса. Кавычки в задании значения атрибута href не обязательны. Если задается ссылка на документ на другом сервере, то вид гиперссылки такой:

```
<a href = "http://www.school.donetsk.ua/11.jpg">Фотография 11-А</a>
```

С помощью различных тегов можно рисовать таблицы, форматировать текст, вставлять в документ изображения, видео- , звуковые файлы и прочее.

5) Каскадные таблицы стилей CSS

Каскадные таблицы стили CSS (CascadingStyleSheets) – стандарт стилей, объявленный консорциумом W3C. Термин *каскадные* указывает на возможность слияния различных видов стилей и на наследование стилей внутренними тегами от внешних.

CSS – это язык, содержащий набор свойств для определения внешнего вида документа. Спецификация CSS определяет свойства и описательный язык для установления связи с HTML-элементами.

CSS – абстракция, в которой внешний вид Web-документа определяется отдельно от его содержания.

Некоторые стили поддерживаются не всеми браузерами. Однако можно назначать любые стили, потому что неподдерживаемые стили будут просто игнорироваться.

Внутренние стили

Внутренние стили определяются атрибутом **style** конкретных тегов. Внутренний стиль действует только на определенные такими тегами элементы. Этот метод мало отличается от традиционного HTML.

Пример

```
<pstyle="color:blue">Абзац с текстом синего цвета</p>
<pstyle="color:red">Абзац с текстом красного цвета</p>
```

Глобальные стили

Глобальные стили CSS располагаются в контейнере **<style>...</style>**, расположенном в свою очередь в контейнере **<head>...</head>**.

Атрибут **type="text/css"**, ранее обязательный для тега **<style>**, в стандарте HTML5 можно опускать.

Глобальные стили являются универсальным средством, позволяющим не только оперативно изменять внешний вид Web-страницы, но и и бороться с перегруженностью документа оформительскими тегами. Проблема в том, что такое стили надо прописывать на каждой странице сайта.

Пример

```
<html>
<head>
    .........
<styletype="text/css">
p{color:#808080;}
</style>
    .........
</head>
```

Внешние (связанные) стили определяются в отдельном файле с расширением **css**. Внешние стили позволяют всем страницам сайта выглядеть единообразно.

Для связи с файлом, в котором описаны стили, используется тег **<link>**, расположенный в контейнере **<head>...</head>**.

В этом теге должны быть заданы два атрибута: **rel="stylesheet"** и **href**, определяющиЙ адрес файла стилей.

Пример

```
<html>
<head>
    .........
<linkrel="stylesheet"href="style.css">
.........
</head>
<body>
    .........
</body>
</html>
```

Подключение стилей

Правило подключения глобальных и внешних стилей состоит из *селектора* и *объявлений стиля*.

Селектор, расположенный в левой части правила, определяет элемент (элементы), для которых установлено правило. Далее, в фигурных скобках перечисляются объявления стиля, разделенные точкой с запятой. Например:

```
p{
text-indent:30px;
font-size:14px;
color:#666;
}
```

Объявление стиля – это пара *свойство CSS: значение CSS*.

Например: **color: red**

color свойство CSS, определяющее цвет текста;

red значение CSS, определяющее красный цвет.

При внутреннем подключении стиля правило CSS, которое является значением атрибута **style**, состоит из объявлений стиля, разделенных точкой с запятой. Например:

```
<pstyle="text-indent: 30px; font-size: 14px; color: #666;">...</p>
```

Универсальный селектор

Универсальный селектор соответствует любому элементу html-документа.

Для обозначения универсального селектора применяется символ "звёздочка" (*).

Его используют, если надо установить одинаковый стиль для всех элементов Web-страницы. Например:

```
* {
margin:0;
padding:0;
}
```

В CSS-коде селектор идентификатора обозначается знаком "решетка" (#). Так как идентификатор **id** применяется только к уникальным элементам, название тега перед знаком "решетка" (#) обычно опускают:

```
#a1 {color: green;}
```

6) ФреймвокFlask. Его применение.

Веб-приложения на Python, основанные на WSGI, должны обладать центральным вызываемым объектом, реализующим приложение. Во Flask это экземпляр класса `Flask`. Каждое приложение Flask должно само создать экземпляр этого класса и передать ему имя модуля, но почему Flask не может сделать это сам?

Если бы не было явного объекта приложения, то следующий код:

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def index():
return 'HelloWorld!'
```

Выглядел бы так:

```
fromhypothetical_flask import route

@route('/')
def index():
return 'Hello World!'
```

Для этого есть три основных причины. Наиболее важная заключается в том, что неявный объект приложения может существовать одновременно только в одном экземпляре. Есть способы сымитировать множество приложений в одном объекте приложения, например, поддерживая пачку приложений, но это вызовет некоторые проблемы, в детали которых мы не станем вдаваться. Следующий вопрос: в каких случаях микрофреймворку может понадобиться создавать больше одного приложения одновременно? Хорошим примером является модульное тестирование. Когда вам нужно протестировать что-нибудь, может оказаться полезным создать минимальное приложение для тестирования определённого поведения. Когда объект приложения удалятся, занятые им ресурсы снова становятся свободными.

Другой возможной причиной, почему могут пригодиться явные объекты приложений, может быть необходимость обернуть объект в дополнительный код, что можно сделать создав подкласс базового класса (`Flask`) для изменения части его поведения. Это невозможно сделать без костылей, если объект был создан до этого, на основе класса, который вам не доступен.

Но существует и ещё одна очень важная причина, почему Flask зависит от явных экземпляров классов: имя пакета. Когда бы вы ни создавали экземпляр Flask, обычно вы передаёте ему *__name__* в качестве имени пакета. Flask использует эту информацию для правильной загрузки ресурсов относящихся к модулю. При помощи выдающихся

возможностей Python к рефлексии, вы можете получить доступ к пакету, чтобы узнать, где хранятся шаблоны и статические файлы (см. `open_resource()`). Очевидно, что фреймворкам не требуется какая-либо настройка и они по-прежнему смогут загружать шаблоны относительно модуля вашего приложения. Но они используют для этого текущий рабочий каталог, поэтому нет возможности надёжно узнать, где именно находится приложение. Текущий рабочий каталог одинаков в рамках процесса, поэтому если запустить несколько приложений в рамках одного процесса (а это может происходить в веб-сервере, даже если вы об этом не догадываетесь), путь может измениться. Даже хуже: многие веб-серверы устанавливают рабочим каталогом не каталог приложения, а корневой каталог документов, который обычно является другим каталогом.

Третья причина заключается в том, что “явное лучше неявного”. Этот объект - ваше приложение WSGI, вам больше не нужно помнить о чём-то ещё. Если вам нужно воспользоваться промежуточным модулем WSGI, просто оберните его этим модулем и всё (хотя есть способы лучше, чтобы сделать это, если вы не хотите потерять ссылки на объект приложения `wsgi_app()`).

Кроме того, такой дизайн делает возможным использование фабричных методов для создания приложения, которое может оказаться полезным для модульного тестирования и других подобных вещей (app-factories).

Система маршрутизации

Flask использует систему маршрутизации Werkzeug, которая была спроектирована для того, чтобы автоматически упорядочивать маршруты в зависимости от их сложности. Это означает, что вы можете объявлять маршруты в произвольном порядке и они всё равно будут работать ожидаемым образом. Это необходимо, если вы хотите, чтобы маршруты, основанные на декораторах, работали правильно, потому что декораторы могут активироваться в неопределённом порядке, если приложение разделено на несколько модулей.

Другое проектное решение системы маршрутизации Werkzeug заключается в том, что Werkzeug старается, чтобы URL’ы были уникальными. Werkzeug настолько последователен в этом, что автоматически выполнит переадресацию на канонический URL, если маршрут окажется неоднозначным.

Одна система шаблонизации

Flask остановился на одной системе шаблонизации: Jinja2. Почему Flask не имеет интерфейса для подключения систем шаблонизации? Очевидно, что вы можете использовать любые системы шаблонизации, но Flask всё равно настроит Jinja2. Хотя *всегда* настроенная Jinja2, возможно и не потребуется, скорее всего нет причин, чтобы не принять решение воспользоваться единой системой шаблонизации.

Системы шаблонизации похожи на языки программирования и каждая из них обладает собственным мировоззрением. Внешне они все работают одинаково: вы просите систему шаблонизации заполнить шаблон набором переменных и получаете его в виде строки.

7) Куки. Организация сессий.

Сессии и cookies предназначены для хранения сведений о пользователях при переходах между несколькими страницами. При использовании сессий данные сохраняются во временных файлах на сервере. Файлы с cookies хранятся на компьютере пользователя, и по запросу отсылаются броузером серверу.

Использование сессий и cookies очень удобно и оправдано в таких приложениях как Интернет-магазины, форумы, доски объявлений, когда, во-первых, необходимо сохранять

информацию о пользователях на протяжении нескольких станиц, а, во-вторых, своевременно предоставлять пользователю новую информацию.

Протокол HTTP является протоколом "без сохранения состояния". Это означает, что данный протокол не имеет встроенного способа сохранения состояния между двумя транзакциями. Т. е., когда пользователь открывает сначала одну страницу сайта, а затем переходит на другую страницу этого же сайта, то основываясь только на средствах, предоставляемых протоколом HTTP невозможно установить, что оба запроса относятся к одному пользователю. Т. о. необходим метод, при помощи которого было бы отслеживать информацию о пользователе в течение одного сеанса связи с Web-сайтов. Одним из таких методов является управление сеансами при помощи предназначенных для этого функций.

В Flaskесть два варианта для организации сессии:

1)С использование встроенных возможностей requestзапроса:

Для этого создается запрос в виде ответа(response) на который вешается render_templatec нужными параметра авторизации, затем переменной responseуказывается необходимый сессионный ключ(response.set_cookie('имя', session_key)). Этот сессионный ключ может иметь вид: session_key = str(uuid4()).encode("utf-8"), где uuid4() генерирует рандомную последовательность

2)С использование библиотеки session:

Этот способ удобнее так как работать с ним просто, как со списком:

Session['имя'] = session_key – Добавления ключа с заданным именем

session.get('имя', None) – получение сессионного ключа по имени

session.pop('имя', None) – удаление сессионного ключа по имени

8) Физический уровень. Передача данных по каналу с шумом.

Фотографии

9) Средства передачи данных в физических каналах. Скремблирование. Разделение доступа.

Для построения компьютерных сетей применяются линии связи, использующие различную физическую среду. В качестве физической среды в коммуникациях используются: металлы (в основном медь), сверхпрозрачное стекло (кварц) или пластик и эфир. Физическая среда передачи данных может представлять собой кабель "витая пара", коаксиальные кабель, волоконно-оптический кабель и окружающее пространство.

Линии связи или линии передачи данных - это промежуточная аппаратура и физическая среда, по которой передаются информационные сигналы (данные).

В одной линии связи можно образовать несколько каналов связи (виртуальных или логических каналов), например путем частотного или временного разделения каналов. Канал связи - это средство односторонней передачи данных. Если линия связи монопольно используется каналом связи, то в этом случае линию связи называют каналом связи.

Канал передачи данных - это средства двухстороннего обмена данными, которые включают в себя линии связи и аппаратуру передачи (приема) данных. Каналы передачи данных связывают между собой источники информации и приемники информации.

В зависимости от физической среды передачи данных линии связи можно разделить на:

- проводные линии связи без изолирующих и экранирующих оплеток;
- кабельные, где для передачи сигналов используются такие линии связи как кабели "витая пара", коаксиальные кабели или оптоволоконные кабели;
- беспроводные (радиоканалы наземной и спутниковой связи), использующие для передачи сигналов электромагнитные волны, которые распространяются по эфиру.

Скремблер — программное или аппаратное устройство (алгоритм), выполняющее **скремблирование** — обратимое преобразование цифрового потока без изменения скорости передачи с целью получения свойств случайнойпоследовательности. После скремблирования появление «1» и «0» в выходной последовательности равновероятны. Скремблирование — обратимый процесс, то есть исходное сообщение можно восстановить, применив обратный алгоритм

Цели скремблирования

Применительно к телекоммуникационным системам скремблирование повышает надежность синхронизации устройств, подключенных к линии связи (обеспечивает надежное выделение тактовой частоты непосредственно из принимаемого сигнала), и уменьшает уровень помех, излучаемых на соседние линии многожильного кабеля. Другая область применения скремблеров — защита передаваемой информации от несанкционированного доступа.

Для алгоритмов скремблирования исключительно важны скорость работы и случайный характер последовательности, чтобы его нельзя было восстановить в случае перехвата противником. Процесс скремблирования может включать в себя добавление определенных компонент к исходному сигналу либо изменение важных частей сигнала для того, чтобы усложнить восстановление вида исходного сигнала либо для придания сигналу определенных статистических свойств.

Скремблеры применяются в телефонных сетях общего пользования, спутниковой и радиорелейной связи, цифровом телевидении, а также для защиты лазерных дисков от копирования.

Обычно скремблирование осуществляется на последнем этапе цифровой обработки непосредственно перед модуляцией.

Коммутация — процесс соединения абонентов коммуникационной сети через транзитные узлы.

Коммуникационные сети должны обеспечивать связь своих абонентов между собой. Абонентами могут выступать ЭВМ, сегменты локальных сетей, факс-аппараты или телефонные собеседники. Как правило, в сетях общего доступа невозможно предоставить каждой паре абонентов собственную физическую линию связи, которой они могли бы монопольно «владеть» и использовать в любое время. Поэтому в сети всегда применяется какой-либо способ коммутации абонентов, который обеспечивает разделение имеющихся физических каналов между несколькими сеансами связи и между абонентами сети.

Каждый абонент соединен с коммутаторами индивидуальной линией связи, закрепленной за этим абонентом. Линии связи, протянутые между коммутаторами, разделяются несколькими абонентами, то есть используются совместно.

10) Уровень передачи данных. Управление потоком.

Фотки

PPP (англ. *Point-to-PointProtocol*) — двухточечный протокол канального уровня (DataLink) сетевой модели OSI. Обычно используется для установления прямой связи между двумя узлами сети, причём он может обеспечить аутентификацию соединения, шифрование (с использованием ECP, RFC 1968) и сжатие данных. Используется на многих типах физических сетей: нуль-модемный кабель, телефонная линия, сотовая связь и т. д.

Часто встречаются подвиды протокола PPP такие, как Point-to-PointProtocoloverEthernet (PPPoE), используемый для подключения по Ethernet, и иногда через DSL; и Point-to-PointProtocoloverATM (PPPoA), который используется для подключения по ATM AdaptationLayer 5 (AAL5), который является основной альтернативой PPPoE для DSL.

PPP представляет собой целое семейство протоколов: протокол управления линией связи (LCP), протокол управления сетью (NCP), протоколы аутентификации (PAP, CHAP), многоканальный протокол PPP (MLPPP).

11) Сетевой уровень. Виртуальные каналы и дейтаграммы. Маршрутизация. QoS.

Фотки

12)Стек протоколов TCP/IP. Уровни протоколов стека

Стек протоколов TCP/IP — набор сетевых протоколов передачи данных, используемых в сетях, включая сеть Интернет. Название TCP/IP происходит из двух наиважнейших протоколов семейства — TransmissionControlProtocol (TCP) и InternetProtocol (IP), которые были разработаны и описаны первыми в данном стандарте. Также изредка упоминается как модель DOD в связи с историческим происхождением от сети ARPANET из 1970 годов (под управлением DARPA, Министерства обороны США).

Протоколы работают друг с другом в стеке — это означает, что протокол, располагающийся на уровне выше, работает «поверх» нижнего, используя механизмы инкапсуляции. Например, протокол TCP работает поверх протокола IP.

Стек протоколов TCP/IP включает в себя четыре уровня:

- прикладной уровень (applicationlayer),
- транспортный уровень (transportlayer),

- сетевой уровень (Internetlayer),
- канальный уровень (linklayer).

Протоколы этих уровней полностью реализуют функциональные возможности модели OSI. На стеке протоколов TCP/IP построено всё взаимодействие пользователей в IP-сетях. Стек является независимым от физической среды передачи данных.

Прикладной уровень

На прикладном уровне (Applicationlayer) работает большинство сетевых приложений.

Эти программы имеют свои собственные протоколы обмена информацией, например, HTTP для WWW, FTP (передача файлов), SMTP (электронная почта), SSH (безопасное соединение с удалённой машиной), DNS (преобразование символьных имён в IP-адреса) и многие другие.

Транспортный уровень

Протоколы транспортного уровня (Transportlayer) могут решать проблему негарантированной доставки сообщений («дошло ли сообщение до адресата?»), а также гарантировать правильную последовательность прихода данных. В стеке TCP/IP транспортные протоколы определяют, для какого именно приложения предназначены эти данные.

Протоколы автоматической маршрутизации, логически представленные на этом уровне (поскольку работают поверх IP), на самом деле являются частью протоколов сетевого уровня; напримерOSPF (IP идентификатор 89).

TCP (IP идентификатор 6) — «гарантированный» транспортный механизм с предварительным установлением соединения, предоставляющий приложению надёжный поток данных, дающий уверенность в безошибочности получаемых данных, перезапрашивающий данные в случае потери и устраняющий дублирование данных.

UDP (IP идентификатор 17) протокол передачи датаграмм без установления соединения. Также его называют протоколом «ненадёжной» передачи, в смысле невозможности удостовериться в доставке сообщения адресату, а также возможного перемешивания пакетов. В приложениях, требующих гарантированной передачи данных, используется протокол TCP.

Сетевой уровень

Сетевой уровень (Internetlayer) изначально разработан для передачи данных из одной (под)сети в другую.

С развитием концепции глобальной сети в уровень были внесены дополнительные возможности по передаче из любой сети в любую сеть, независимо от протоколов нижнего уровня, а также возможность запрашивать данные от удалённой стороны, например в протоколе ICMP (используется для передачи диагностической информации IP-соединения) и IGMP (используется для управления multicast-потоками).

ICMP и IGMP расположены над IP и должны попасть на следующий — транспортный — уровень, но функционально являются протоколами сетевого уровня, и поэтому их невозможно вписать в модель OSI.

Канальный уровень

Канальный уровень (Linklayer) описывает, каким образом передаются пакеты данных через физический уровень, включая кодирование (то есть специальные последовательности бит, определяющих начало и конец пакета данных). Ethernet, например, в полях заголовка пакета содержит указание того, какой машине или машинам в сети предназначен этот пакет.

13)Протокол IPv4. Протокол ICMPv4.

IPv4 (англ. *InternetProtocolversion 4*) — четвёртая версия интернет протокола (IP). Первая широко используемая версия.

Фотка

ICMPv4 (ang. InternetControlMessageProtocol) - это один из протоколов сетевого уровня в модели ISO/OSI. Его задачей является обслуживание функции контроля правильности работы сети. С его помощью передаются всякого рода, низкоуровневые сводки, с раскроенными неправильностями во время сетевых связей. Практически целая коммуникация между данными компьютерами или другими устройствами при употреблении протокола ICMP происходит незаметным для конечнего пользователя образом. Единичными исключениями являются здесь инструменты ping и traceroute. Коммуникация, ицпользующая протокол ICMP, состоит в пересылке подходящих информаций об ошибках, раскроенных во время связи между двумя устройствами. Одиночная информация существует в виде пакета, сформированного надлежащим образом (ang. Datagram), который следовательно будет подвергнутый инкапсуляции в рамке протокола IP. Протокол ICMP, вопреки всеобщему мнению, не использует в своей работе протоколы TCP, ни UDP. Итак, не пользуется никакими сетевыми портами. Устройство пакета ICMP следующее:

• Заголовок в 4 байтов - первый байт определяет тип пакета, второй - код операции, третий и четвёртый представляют собой контрольную сумму.
• Поле данных с долготой зависимой от типа пакета и его функции. В некоторых случаях может быть установленным с уровня инструмента, напр. догадливая команда ping в Виндоуз устанавливает размер данных пакета ECHO_REQUEST на 32 байта, а версия, встречающаяся в системах Юникс на 56 байтов.

Примеры работы протокола ICMP:

• Ping - один из инструментов, выступающих практически в каждой операционной системе, обслуживающей протокол TCP/IP. С его помощью пакеты ICMP ECHO_REQUEST отправляются в целевой компьютер. Дистанционная машина, после получения такого сообщения должна ответить при помощи ECHO_REPLY. Поэтому можно определить следующее: Конфигурация сети делает возможной связь с дистанционной машиной, и оценку её нагрузки на основании информаций, касающихся количества потерянных пакетов и времени ответа.
•Traceroute – инструмент, делающий возможным определение, через какие маршрутизаторы проходит пакет по дороге к дистанционному компьютеру. Сначала, локальный компьютер посылает пакет ECHO_REQUEST в дистанционное устройство, с параметром ТТЛ (TTL -TimetoLive), установленным на 1. Первый рутер уменьшает ТТЛ на один, значит до нуля, удаляет пакет и отсылает адресату сообщение ICMP TIME_EXCEEDED. Целевой компьютер, после получения такой информации, возобновляет высылку ECHO_REQUEST, но ц ТТЛ установленным на стоимость 2. Первый рутер уменьшает ТТЛ на 1, второй сделает то же самое, устанавливая 0, и вновь удалит пакет, и отошлёт сообщение TIME_EXCEEDED. Такая ситуация повтаряется так долго, что даже пакет доберётся до дистанционного комньютера, который тогда отошлёт отправителю сообщение ECHO_REPLY.

14)Адреса IPv4. CIDR.

IPv4 использует 32-битные (четырёхбайтные) адреса, ограничивающие адресное пространство 4 294 967 296 (2^{32}) возможными уникальными адресами.

Традиционной формой записи IPv4 адреса является запись в виде четырёх десятичных чисел (от 0 до 255), разделённых точками. Через дробь указывается длина маски подсети.

Форма записи	***Пример***	***Преобразование из десятичной нотации с точками***
Десятичная с точками (англ.)	`192.0.2.235`	—
Шестнадцатеричная с точками	`0xC0.0x00.0x02.0xEB`	*Каждый октет преобразуется в шестнадцатеричную форму*

Восьмеричная с точками	*0300.0000.0002.0353*	*Каждый октет преобразуется в восьмеричную форму*
Шестнадцатеричная	*0xC00002EB*	*Конкатенация октетов из шестнадцатеричной нотации с точками*
Десятичная	*3221226219*	*32-битное число в десятичной форме*
Восьмеричная	*030000001353*	*32-битное число в восьмеричной форме*

Бесклассоваяадресация (англ. *Classless Inter-Domain Routing*, англ. *CIDR*) — метод IP-адресации, позволяющий гибко управлять пространством IP-адресов, не используя жёсткие рамки классовой адресации. Использование этого метода позволяет экономно использовать ограниченный ресурс IP-адресов, поскольку возможно применение различных масок подсетей к различным подсетям.

+ фотка

15)Протокол IPv6. Протокол ICMPv6.

IPv6 (англ. *InternetProtocolversion 6*) — новая версия протоколаIP, призванная решить проблемы, с которыми столкнулась предыдущая версия (IPv4) при её использовании в Интернете, за счёт использования длины адреса 128 бит вместо 32. Протокол был разработан IETF.

Версия 6 (IPv6)

Позиция в октетах		0								1								2								3							
	Позиция в битах	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Версия				Класс трафика								Метка потока																			
4	32	Длина полезной нагрузки																След. заголовок								Число переходов							
8	64	IP-адрес отправителя																															
12	96																																
16	128																																
20	160																																
24	192	IP-адрес получателя																															
28	224																																
32	256																																
36	288																																

1)Класс трафика — определяет приоритет трафика для QoS

2)Метка потока — уникальное число, одинаковое для однородного потока пакетов.

Преимущество IPv6

-Больше адресов

-Маршрутизатором не требуется фрагментировать пакет

-Исключена контрольная сумма пакета, проверка данных возлагается на другие уровни

-Появились метки потоков и классы трафика

16)Адреса IPv6.

Существуют различные типы адресов IPv6: одноадресные (Unicast), групповые (Anycast) и многоадресные (Multicast).

Адреса типа Unicast хорошо всем известны. Пакет, посланный на такой адрес, достигает в точности интерфейса, который этому адресу соответствует.

Адреса типа Anycast синтаксически неотличимы от адресов Unicast, но они адресуют группу интерфейсов. Пакет, направленный такому адресу, попадёт в ближайший (согласно метрике маршрутизатора) интерфейс. Адреса Anycast могут использоваться только маршрутизаторами.

Адреса типа Multicast идентифицируют группу интерфейсов. Пакет, посланный на такой адрес, достигнет всех интерфейсов, привязанных к группе многоадресного вещания.

Широковещательные адреса IPv4 (обычно xxx.xxx.xxx.255) выражаются адресами многоадресного вещания IPv6.

Адреса разделяются двоеточиями (напр. fe80:0:0:0:200:f8ff: fe21:67cf). Большое количество нулевых групп может быть пропущено с помощью двойного двоеточия (fe80::200:f8ff: fe21:67cf). Такой пропуск должен быть единственным в адресе.

Типы Unicast адресов

- Глобальные

Соответствуют публичным IPv4 адресам. Могут находиться в любом не занятом диапазоне. В настоящее время региональные интернет-регистраторы распределяют блок адресов 2000::/3 (с 2000:: по 3FFF:FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FFFF).

- Link-Local

Соответствуют автосконфигурированным с помощью протокола APIPA IPv4 адресам. Начинаются с FE80.
Используется:

1. В качестве исходного адреса для RouterSolicitation(RS) и RouterAdvertisement(RA) сообщений, для обнаружения маршрутизаторов
2. Для обнаружения соседей (эквивалент ARP для IPv4)
3. Как next-hop адрес для маршрутов

- Unique-Local

RFC 4193, соответствуют внутренним IP адресам, которыми в версии IPv4 являлись 10.0.0.0/8, 172.16.0.0/12 и 192.168.0.0/16. Начинаются с цифр FC00 и FD00.

Типы Multicast адресов

Адреса мультикаст бывают двух типов:

- Назначенные (*Assignedmulticast*) — специальные адреса, назначение которых предопределено. Это зарезервированные для определённых групп устройств мультикастовые адреса. Отправляемый на такой адрес пакет будет получен всеми устройствами, входящими в группу.

- Запрошенные (*Solicitedmulticast*) — остальные адреса, которые устройства могут использовать для прикладных задач. Адрес этого типа автоматически появляется, когда на некотором интерфейсе появляется юникастовый адрес. Адрес формируется из сети FF02:0:0:0:0:1:FF00::/104, оставшиеся 24 бита — такие же как у настроенного юникастового адреса.

+фотка

17)Трансляция сетевых адресов (NAT).

Трансляция сетевых адресов (NAT, RFC 1631) функционирует посредством отображения (маппинга) частных IP-адресов в уникальные глобальные IP-адреса, требуемые для связи с хостами других сетей. NAT подменяет первоначальный IP-адрес источника и номера TCP- или UDP-портов источника перед тем как переслать пакет в Интернет, делая их такими, как если бы исходили от самой системы NAT (например, от маршрутизатора). Таблица NAT хранит исходные адреса и номера портов, чтобы восстановить их первоначальные значения в ответном пакете.

Как правило, Multi-NAT имеет следующие типы отображения (маппинга) IP-адресов.
1. Один к одному (One-to-One)

В режиме "один к одному" таблица NAT выполняет маппинг одного локального IP-адреса в один глобальный IP-адрес.

2. Многие к одному (Many-to-One)

В режиме "многие к одному" таблица NAT выполняет маппинг множества локальных IP-адресов в один глобальный IP-адрес.
"Многие к одному" это тип NAT, который в маршрутизаторах ZyXEL называется SUA (SingleUserAccount). Фактически данный режим представляет собой PortForwarding.

3. Многие ко многим с перегрузкой (Many-to-ManyOverload)

В режиме "многие ко многим с перегрузкой" таблица NAT выполняет маппинг множества локальных IP-адресов к общим глобальным IP-адресам.

4. Многие ко многим без перегрузки (Many-to-ManyNoOverload)
В режиме "многие ко многим без перегрузки" таблица NAT выполняет маппинг каждого локального IP-адреса к уникальному глобальному IP-адресу.

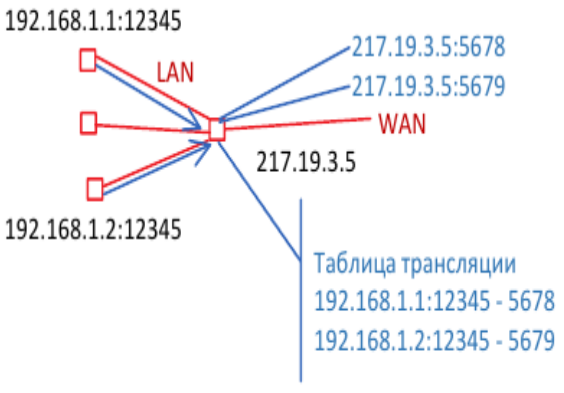
5. Сервер (Server)
Этот режим позволяет указывать внутренние серверы различных служб в NAT, которые должны быть доступными для внешнего мира.

Модель NAT

Недостатки:

- Необходимо вести таблицу трансляции
- Внешний компьютер не может инициировать передачу
- Нарушается модель ISO/OS

Для различения пакетов, исходящих из локальной сети, используется номер порта, у пакетов адрес заменяется на внешний адрес маршрутизатора, а номер порта на случайный, который сохраняется в таблице. У входящих пакетов адрес заменяется обратно на сохранный.



18)Транспортный уровень. Протокол TCP. Порты.

Транспортный уровень

На транспортном уровне обеспечивается надёжная передача данных через создаваемое сетевое соединение. Участник обмена данными(аппаратура или программа) называются транспортной сущностью или транспортным объектом. Протоколы транспортного уровня могут быть как ориентированы на создания соединения(TCP), так и не создавать его(UDP). Несмотря на то что UDP не создаёт соединение он выше уровнем чем IP, IP - передаёт данные от компьютера к компьютеру, а UDP от транспортной сущности к транспортной сущности(от программы к программе). Распространённый набор примитивов транспортного уровня — это так называемые сокеты Беркли.
Примитивы сокетов (для TCP):

- Socket
- Bind - связывает локальный адрес с сокетом.
- Listen - объявление о готовности принять соединение.
- Accept - блокировка до получения соединения.
- Connect - активно установить соединение.
- Send - отправить данные.
- Receive - получить данные.
- Close - разорвать соединение.

Распространённые протоколы транспортного уровня это TCP (обеспечивает надёжную передачу информации через создаваемое соединение) и (UDP не гарантирует надёжность передачи), так же используются служебные протоколы ACMP. Установка соединения требует дополнительных действий. Это связано с тем что пакеты в реальной сети могут дублироваться приходить в неверном порядке или не приходить вовсе. Могут потребоваться выполнить запросы на повторную отправку. Перед началом передачи данных узлы должны договориться о параметрах создаваемого соединения. Один из алгоритмов установки соединения — это тройное рукопожатие. Соединение необходимо так же корректно закрыть.

Структура TCP

TCP (*transmission control protocol* — протокол управления передачей) — один из основных протокол передачи данных интернета, предназначенный для управления передачи данных. Сети и подсети, в которых совместно используются протоколы TCP и IP называются сетями TCP/IP.

Структура заголовка

Бит	0 — 3	4 — 9	10 — 15	16 — 31
0	Порт источника			Порт назначения
32	Порядковый номер			
64	Номер подтверждения			
96	Длина заголовка	Зарезервировано	Флаги	Размер Окна
128	Контрольная сумма			Указатель важности
160	Опции (необязательное, но используется практически всегда)			
160/192+	Данные			

Флаги (управляющие биты)

Это поле содержит 6 битовых флагов:

- **URG** — поле *«Указатель важности»* задействовано
- **ACK** — поле *«Номер подтверждения»* задействовано
- **PSH** — инструктирует получателя протолкнуть данные, накопившиеся в приемном буфере, в приложение пользователя
- **RST** — оборвать соединения, сбросить буфер (очистка буфера)
- **SYN** — синхронизация номеров последовательности
- **FIN** — флаг, будучи установлен, указывает на завершение соединения.

Структура UDP

Биты	0 - 15	16 - 31
0-31	Порт отправителя (Source port)	Порт получателя (Destination port)
32-63	Длина датаграммы (Length)	Контрольная сумма (Checksum)
64-...	Данные (Data)	

UDP (англ. *UserDatagramProtocol* — протокол пользовательских датаграмм) — один из ключевых элементов TCP/IP, набора сетевых протоколов для Интернета. С UDP компьютерные приложения могут посылать сообщения (в данном случае называемые датаграммами) другим хостам по IP-сети без необходимости предварительного сообщения для установки специальных каналов передачи или путей данных.

UDP использует простую модель передачи, без неявных «рукопожатий» для обеспечения надёжности, упорядочивания или целостности данных. Таким образом, UDP предоставляет ненадёжный сервис, и датаграммы могут прийти не по порядку, дублироваться или вовсе исчезнуть без следа. UDP подразумевает, что проверка ошибок и исправление либо не нужны, либо должны исполняться в приложении. Чувствительные ко времени приложения часто используют UDP, так как предпочтительнее сбросить пакеты, чем ждать задержавшиеся пакеты, что может оказаться невозможным в системах реального времени. При необходимости исправления ошибок на сетевом уровне интерфейса приложение может задействовать TCP или SCTP, разработанные для этой цели.

Датаграмма - блок информации, передаваемый протоколом без предварительного установления соединения и создания виртуального канала. Любой протокол, не устанавливающий предварительное соединение (а также обычно не контролирующий порядок приёмо-передачи и дублирование пакетов), называется датаграммным протоколом. Таковы, например, протоколы Ethernet, IP, UDP и др.

Порт— натуральное число, записываемое в заголовках протоколовтранспортного уровнямодели OSI (TCP, UDP, SCTP, DCCP). Используется для определения процесса-получателя пакета в пределах одного хоста.

SCTP— протоколтранспортного уровня в компьютерных сетях.

DCCP — протокол транспортного уровнямодели OSI, разрабатываемый IETF

18)Процедура соединения в протоколе TCP (трехстороннее рукопожатие).

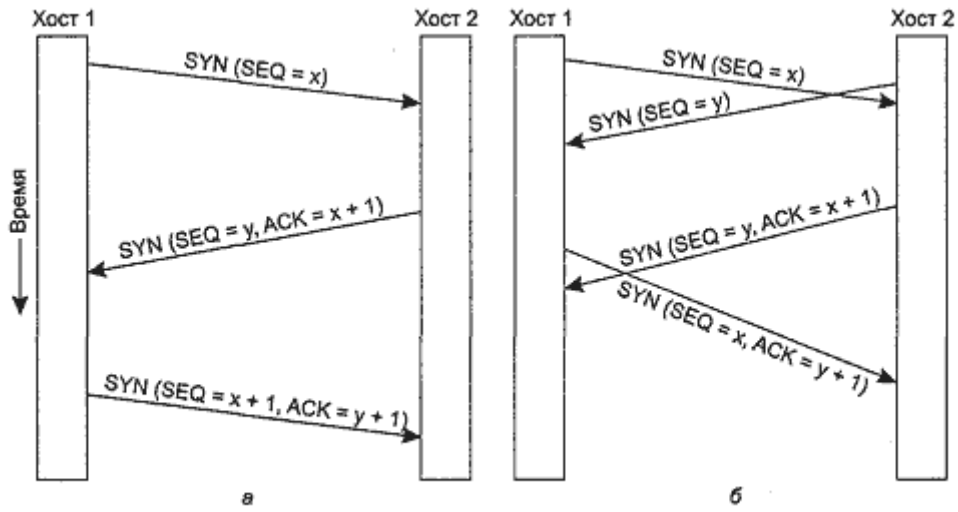
Установка TCP-соединения

В протоколе TCP-соединения устанавливаются с помощью «тройного рукопожатия», описанного в разделе «Установка соединения». Чтобы установить соединение, одна сторона (например, сервер) пассивно ожидает входящего соединения, выполняя примитивы LISTEN и ACCEPT, либо указывая конкретный источник, либо не указывая его.

Другая сторона (например, клиент) выполняет примитив CONNECT, указывая IP-адрес и порт, с которым он хочет установить соединение, максимальный размер TCP-сегмента и, по желанию, некоторые данные пользователя (например, пароль). Примитив CONNECT

посылает TCP-сегмент с установленным битом SYN и сброшенным битом ACK и ждет ответа.

Когда этот сегмент прибывает в пункт назначения, TCP-сущность проверяет, выполнил ли какой-нибудь процесс примитив LISTEN, указав в качестве параметра тот же порт, который содержится в поле Порт получателя. Если такого процесса нет, она отвечает отправкой сегмента с установленным битом RST для отказа от соединения.



Установка TCP-соединения в нормальном случае (*а*); столкновение вызовов (*б*)

Если какой-либо процесс прослушивает какой-либо порт, то входящий TCP-сегмент передается этому процессу. Последний может принять соединение или отказаться от него. Если процесс принимает соединение, он отсылает в ответ подтверждение. Последовательность TCP-сегментов, посылаемых в нормальном случае, (рис. а) Обратите внимание на то, что сегмент с установленным битом SYN занимает 1 байт пространства порядковых номеров, что позволяет избежать неоднозначности в их подтверждениях.

Если два хоста одновременно попытаются установить соединение друг с другом, то последовательность происходящих при этом событий будет соответствовать рис. б. В результате будет установлено только одно соединение, а не два, так как пара конечных точек однозначно определяет соединение. То есть если оба соединения пытаются идентифицировать себя с помощью пары (x, y), делается всего одна табличная запись для (x, y).

Начальное значение порядкового номера соединения не равно нулю по обсуждавшимся выше причинам. Используется схема, основанная на таймере, изменяющем свое состояние каждые 4 мкс. Для большей надежности хосту после сбоя запрещается перезагружаться ранее чем по прошествии максимального времени жизни пакета. Это позволяет гарантировать, что ни один пакет от прежних соединений не бродит где-нибудь в Интернете.