Оглавление

[1. Аппаратное обеспечение вычислительных сетей. 4](#_Toc74911137)

[1.Среда передачи данных вычислительных сетей 4](#_Toc74911138)

[2.Сетевое оборудование рабочих станций и серверов. 5](#_Toc74911139)

[3.Основное оборудование для организации ВС. 5](#_Toc74911140)

[2. Архитектура ВС 6](#_Toc74911141)

[4.Типы архитектур ВС 6](#_Toc74911142)

[5.Терминал / Главный компьютер 7](#_Toc74911143)

[6.Одноранговая сеть 8](#_Toc74911144)

[7.Клиент-сервер 10](#_Toc74911145)

[3.Топология ВС 12](#_Toc74911146)

[8. Виды топологий ВС. 12](#_Toc74911147)

[9.Топология «Шина». 12](#_Toc74911148)

[10. Топология «Кольцо». 14](#_Toc74911149)

[11.Топология «Звезда». 16](#_Toc74911150)

[12.Топология «Дерево». 17](#_Toc74911151)

[13.Топология «Сота». 18](#_Toc74911152)

[**Принцип действия** 19](#_Toc74911153)

[4.Модель ISO/OSI 20](#_Toc74911154)

[14.Описание модели ISO/OSI. 20](#_Toc74911155)

[15.Уровни модели ISO/OSI. 20](#_Toc74911156)

[5.Стек протоколов TCP/IP 25](#_Toc74911157)

[16.Основные понятия. 25](#_Toc74911158)

[Обмен данными TCP/IP 25](#_Toc74911159)

[Обмен данными на низком уровне 26](#_Toc74911160)

[17.Описание уровней. 26](#_Toc74911161)

[1.Прикладной уровень 27](#_Toc74911162)

[2.Транспортный уровень 27](#_Toc74911163)

[3.Сетевой уровень 27](#_Toc74911164)

[4. Уровень сетевого интерфейса 29](#_Toc74911165)

[18.Основные протоколы стека. 32](#_Toc74911166)

[6.Протокол TCP и UDP 34](#_Toc74911167)

[19.Основные понятия 34](#_Toc74911168)

[20.Алгоритм Установления соединения в TCP 37](#_Toc74911169)

[21.UDP 44](#_Toc74911170)

[7.Протокол IP. 46](#_Toc74911171)

[22. Основные понятия. 46](#_Toc74911172)

[23. Классы подсетей. 49](#_Toc74911173)

[8.Маршрутизация в IP сетях. 56](#_Toc74911174)

[24. Основные понятия. 56](#_Toc74911175)

[25. Таблицы маршрутизации 58](#_Toc74911176)

[26. Прямая/косвенная маршрутизация 63](#_Toc74911177)

[Практическая часть 66](#_Toc74911178)

[1.Socket 66](#_Toc74911179)

[2.Bind 66](#_Toc74911180)

[3.Connect 67](#_Toc74911181)

[4.Sendto 68](#_Toc74911182)

[5.Recvfrom 69](#_Toc74911183)

[6.Accept 71](#_Toc74911184)

[7.Listen 72](#_Toc74911185)

[8.Shutdown 72](#_Toc74911186)

[9.Close 73](#_Toc74911187)

[10.Алгоритм работы сервера 73](#_Toc74911188)

[11.Алгоритм работы клиента 75](#_Toc74911189)

[12.Блокирующие/неблокирующие сокеты 76](#_Toc74911190)

[13.Алгоритм работы с неблокирующими сокетами 77](#_Toc74911191)

# 1. Аппаратное обеспечение вычислительных сетей.

## 1.Среда передачи данных вычислительных сетей

**Среда передачи данных** - линии связи (или каналы связи), по которым производится обмен информацией между компьютерами. В подавляющем большинстве компьютерных сетей (особенно локальных) используются проводные или кабельные каналы связи, хотя существуют и беспроводные сети, которые сейчас находят все более широкое применение, особенно в портативных компьютерах.

**Компьютерная сеть** - система, обеспечивающая обмен данными между вычислительными устройствами (компьютеры, серверы, маршрутизаторы и другое оборудование). Для передачи информации могут быть использованы различные физические явления, как правило, — различные виды электрических сигналов, световых сигналов или электромагнитного излучения

## 2.Сетевое оборудование рабочих станций и серверов.

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.

**Абонент сети** (узел, хост, станция) — это устройство, подключенное к сети и активно участвующее в информационном обмене. Чаще всего абонентом (узлом) сети является компьютер, но абонентом также может быть, например, сетевой принтер или другое периферийное устройство, имеющее возможность напрямую подключаться к сети.

## 3.Основное оборудование для организации ВС.

**Канал связи –** совокупность средств двухстороннего обмена данными, включающая в себя линии связи и аппаратуру передачи (приема) данных. Каналы передачи данных связывают между собой источники информации и приемники информации.

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.

**Линия связи** - промежуточная аппаратура и физическая среда, по которой передаются информационные сигналы (данные).

# 2. Архитектура ВС

## 4.Типы архитектур ВС

Архитектура сетей - логическая организация абонентов при работе в сети.

## 5.Терминал / Главный компьютер



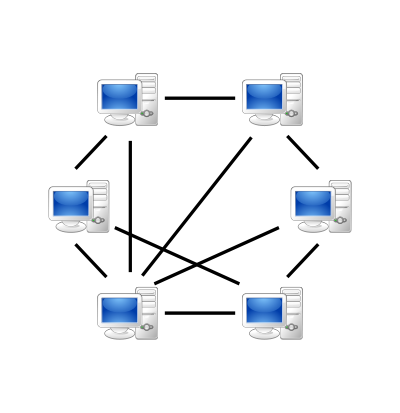
Архитектура терминал – главный компьютер (terminal – host computer architecture) – это концепция информационной сети, в которой вся обработка данных осуществляется одним или группой главных компьютеров.

Рассматриваемая архитектура предполагает два типа оборудования:

* Главный компьютер, где осуществляется управление сетью, хранение и обработка данных.
* Терминалы, предназначенные для передачи главному компьютеру команд на организацию сеансов и выполнения заданий, ввода данных для выполнения заданий и получения результатов.

**Основные** **достоинства** централизованной архитектуры «терминал-главный компьютер» обусловлены простотой администрирования и защиты информации. Все терминалы были однотипными, а, следовательно, устройства на рабочих местах пользователей вели себя предсказуемо и в любой момент могли быть заменены. Затраты на обслуживание терминалов и линий связи легко прогнозировались.

## 6.Одноранговая сеть



**Одноранговая архитектура** (peer-to-peer architecture) – это концепция информационной сети, в которой ее ресурсы рассредоточены по всем системам. Данная архитектура характеризуется тем, что в ней все системы равноправны.

К одноранговым сетям относятся малые сети, где любая рабочая станция может выполнять одновременно функции файлового сервера и рабочей станции. В одноранговых **ЛВС**(локально вычислительная сеть) дисковое пространство и файлы на любом компьютере могут быть общими. Чтобы ресурс стал общим, его необходимо отдать в общее пользование, используя службы удаленного доступа сетевых одноранговых операционных систем. В зависимости от того, как будет установлена защита данных, другие пользователи смогут пользоваться файлами сразу же после их создания.

ОдноранговыеЛВС достаточно хороши только для небольших рабочих групп.

Одноранговые ЛВС являются наиболее легким и дешевым типом сетей для установки. При соединении компьютеров, пользователи могут предоставлять ресурсы и информацию в совместное пользование.

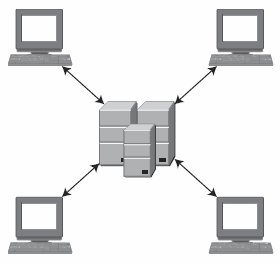
Одноранговые сети имеют следующие преимущества:

* они легки в установке и настройке;
* отдельные ПК не зависят от выделенного сервера;
* пользователи в состоянии контролировать свои ресурсы;
* малая стоимость и легкая эксплуатация;
* минимум оборудования и программного обеспечения;
* нет необходимости в администраторе;
* хорошо подходят для сетей с небольшим количеством пользователей (~10-15).

Также имеются недостатки:

* При отключении компьютеров от сети виды услуг, которые они предоставляли в сети также исчезают
* Отсутствие централизованного администирования
* При осуществлении доступа к разделяемому ресурсу ощущается падение производительности компьютеров.

## 7.Клиент-сервер



**Архитектура клиент – сервер** (client-server architecture) – это концепция информационной сети, в которой основная часть ее ресурсов сосредоточена в серверах, обслуживающих своих клиентов (рис. 1.4). Рассматриваемая архитектура определяет два типа компонентов: серверы и клиенты.

**Сервер** - абонент сети, который предоставляет свои ресурсы другим абонентам, но сам не использует их ресурсы. Таким образом, он обслуживает сеть. Серверов в сети может быть несколько, и совсем не обязательно, что сервер — самый мощный компьютер. **Выделенный** (dedicated) сервер — это сервер, занимающийся только сетевыми задачами. **Невыделенный** сервер может помимо обслуживания сети выполнять и другие задачи.

**Клиент** - Абонент сети, который только использует сетевые ресурсы, но сам свои ресурсы в сеть не отдает, то есть сеть его обслуживает, а он ей только пользуется. Компьютер-клиент также часто называют **рабочей станцией** . В принципе каждый компьютер может быть одновременно как клиентом, так и сервером.

**Абонент сети** - это устройство, подключенное к сети и активно участвующее в информационном обмене. Чаще всего абонентом (узлом) сети является компьютер, но абонентом также может быть, например, сетевой принтер или другое периферийное устройство, имеющее возможность напрямую подключаться к сети. Далее в курсе вместо термина "абонент" для простоты будет использоваться термин "компьютер".

Сети клиент – серверной архитектуры имеют следующие **преимущества**:

* позволяют организовывать сети с большим количеством рабочих станций;
* обеспечивают централизованное управление учетными записями пользователей, безопасностью и доступом, что упрощает сетевое администрирование;
* эффективный доступ к сетевым ресурсам;
* пользователю нужен один пароль для входа в сеть и для получения доступа ко всем ресурсам, на которые распространяются права пользователя.

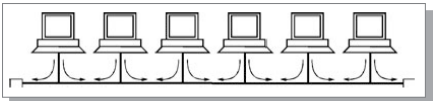
Наряду с преимуществами сети клиент – серверной архитектуры имеют и ряд **недостатков**:

* неисправность сервера может сделать сеть неработоспособной, как минимум потерю сетевых ресурсов;
* требуют квалифицированного персонала для администрирования;
* имеют более высокую стоимость сетей и сетевого оборудования.

# 3.Топология ВС

## 8. Виды топологий ВС.

## 9.Топология «Шина».



Все компьютеры параллельно подключаются к одной линии связи.

Информация от каждого компьютера одновременно передается всем остальным компьютерам

Используются одни кабеля, к которому подключаются все компьютеры сети. Отправляемое один абонентом сообщение распространяется на все абоненты сети. Каждый абонент проверяет кому адресовано сообщение, — если сообщение адресовано ему, то обрабатывает его.

Принимаются специальные меры для того, чтобы при работе с общим кабелем компьютеры не мешали друг другу передавать и принимать данные. Для того, чтобы исключить одновременную посылку данных, применяется либо «несущий» сигнал, либо один из компьютеров является главным и «даёт слово» остальным компьютерам такой сети. Например, в сетях Ethernet (IEEE 802.3) c шинной топологией станции прослушивают занятость среды и действуют по алгоритму CSMA/CD (англ. Carrier Sense Multiple Access with Collision Detection — множественный доступ с прослушиванием несущей и обнаружением столкновений).

Шина допускает идентичность сетевого оборудования компьютеров, а также равноправие всех абонентов. При таком соединении компьютеры могут передавать информацию только по очереди, — последовательно —потому что линия связи единственная. В противном случае пакеты передаваемой информации будут искажаться в результате взаимного наложения (то есть произойдет конфликт, коллизия). Таким образом, в шине реализуется режим полудуплексного(half duplex) обмена: данные могут передаваться в обоих направлениях, но лишь в различные моменты времени, а не одновременно (то есть последовательно, а непараллельно).

В топологии «шина» отсутствует центральный абонент, через которого передаётся вся информация, что увеличивает надёжность «шины». Добавление новых абонентов в «шину» достаточно простое и обычно возможно даже во время работы сети. В большинстве случаев при использовании «шины» нужно минимальное количество соединительного кабеля по сравнению с другой топологией. Правда, нужно учесть, что к каждому компьютеру (кроме двух крайних) подходят два кабеля, что не всегда удобно.

«Шине» не страшны отказы отдельных компьютеров. Но так как используется только один общий кабель, — в случае его обрыва нарушается работа всей сети. Может показаться, что «шине» обрыв кабеля не страшен, поскольку в этом случае остаются две полностью работоспособные «шины». Однако из-за особенности распространения электрических сигналов по длинным линиям связи необходимо предусматривать включение на концах шины специальных устройств —Терминаторов.

Без включения терминаторов в «шину» сигнал отражается от конца линии и искажается так, что связь по сети становится невозможной. Таким образом при разрыве или повреждении кабеля нарушается согласование линии связи, и прекращается обмен даже между теми компьютерами, которые остались физически соединёнными между собой. Короткое замыкание в любой точке кабеля «шины» выводит из строя всю сеть. Хотя в целом надёжность «шины» все же сравнительно высока, так как выход из строя отдельных компьютеров не нарушит работоспособность сети в целом, поиск неисправностей в «шине» затруднён. В частности: любой отказ сетевого оборудования в «шине» очень трудно локализовать, потому что все сетевые адаптеры включены параллельно, и понять, который из них вышел из строя, не так-то просто.

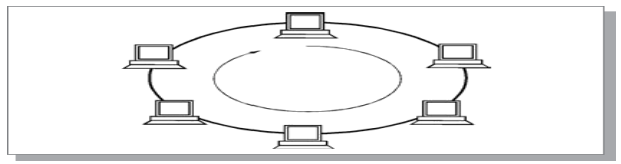
**Достоинства •**

Небольшое время установки сети; • Дешевизна (требуется кабель меньшей длины и меньше сетевых устройств); • Простота настройки; • Выход из строя одной рабочей станции не отражается на работе всей сети.

**Недостатки •**

Неполадки в сети, такие как обрыв кабеля или выход из строя терминатора, полностью блокируют работу всей сети; • Затрудненность выявления неисправностей; • С добавлением новых рабочих станций падает общая производительность сети.

## 10. Топология «Кольцо».

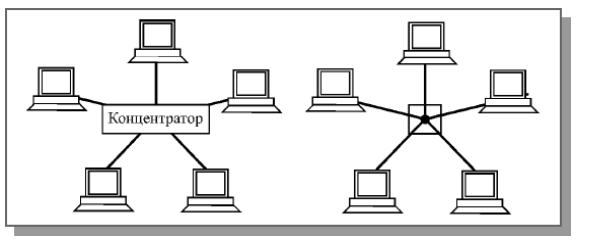


Компьютеры последовательно объединены в кольцо. Передача информации в кольце всегда производится только в одном направлении. Каждый из компьютеров передает информацию только одному компьютеру, следующему в цепочке за ним, а получает информацию только от п р е д ы д у щ е г о в ц е п о ч к е к о м п ь ю т е р а КольцоКольцо В кольце выделяется специальный абонент, который управляет обменом или контролирует обмен. Понятно, что наличие такого управляющего абонента снижает надёжность сети, потому что выход его из строя сразу же парализует весь обмен. Компьютеры в кольце не являются полностью равноправными (в отличие, например, от шинной топологии). Одни из них обязательно получают информацию от компьютера, который ведёт передачу в этот момент, раньше, а другие — позже. Именно на этой особенности топологии и строятся методы управления обменом по сети, специально рассчитанные на «кольцо». В этих методах право на следующую передачу (или, как ещё говорят, на захват сети) переходит последовательно к следующему по кругу компьютеру. 21Кольцо Подключение новых абонентов в «кольцо» обычно совсем без болезненно, 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). 22Кольцо В кольце не используется конкурентный метод посылки данных, компьютер в сети получает данные от стоящего предыдущим в списке адресатов и перенаправляет их далее, если они адресованы не ему. Список адресатов генерируется компьютером, являющимся генератором маркера. Сетевой модуль генерирует маркерный сигнал (обычно порядка 2—10 байт во избежание затухания) и передаёт его следующей системе (иногда по возрастанию MAC-адреса). Следующая система, приняв сигнал, не анализирует его, а просто передаёт дальше. Это так называемый нулевой цикл. Последующий алгоритм работы таков — пакет данных GR, передаваемый отправителем адресату, начинает следовать по пути, проложенному маркером. Пакет передаётся до тех пор, пока не доберётся до получателя. 23Кольцо **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 отражаются на работоспособности всей сети; • Сложность конфигурирования и настройки; • Сложность поиска неисправностей; • Необходимость иметь две сетевые платы на каждой рабочей станции; • Добавление/удаление станции требует временной остановки работы сети.

## 11.Топология «Звезда».



Топология сети с явно выделенным центром, к которому подключаются все остальные абоненты. Все компьютеры сети присоединены к центральному узлу (обычно коммутатор), образуя физический сегмент сети. Подобный сегмент сети может функционировать как отдельно, так и в составе сложной сетевой топологии(как правило, «дерево»). Весь обмен информацией идет исключительно через центральный компьютер, на который таким способом возлагается очень большая нагрузка, поэтому ничем другим, кроме сети, он заниматься не может. Как правило, именно центральный компьютер является самым мощным, и именно на него возлагаются все функции по управлению сетью. Рабочая станция, которой необходимо передать данные, отсылает их на концентратор. В определённый момент времени только одна машина в сети может пересылать данные, если на концентратор одновременно приходят два пакета, обе посылки оказываются не принятыми и отправителям нужно будет подождать случайный промежуток времени, чтобы возобновить передачу данных. Этот недостаток отсутствует на сетевом устройстве более высокого уровня —коммутаторе, который, в отличие от концентратора, подающего пакет на все порты, подает лишь на определенный порт — получателю. Одновременно может быть передано несколько пакетов. Сколько — зависит от коммутатора.

**Достоинства** • выход из строя одной рабочей станции не отражается на работе всей сети в целом; • 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.

## 12.Топология «Дерево».

Дерево — топология компьютерной сети, в которой каждый узел более высокого уровня связан с узлами более низкого уровня звездообразной связью, образуя комбинацию звезд. Также дерево называют иерархической звездой.

Название дерево пришло из теории графов. Первый узел дерева принято называть корнем, следующие узлы высокого уровня — родительскими, а узлы более низкого уровня — дочерними. Таким образом каждый дочерний узел, который имеет связь с более низкими узлами, является для этих узлов родительским.

По количеству дочерних узлов деревья делятся на двоичные (бинарные) и N-арные деревья. Топология двоичного дерева подразумевает, что у каждого родительского узла может быть не более двух дочерних узлов.

Также деревья могут быть как активными, так и пассивными. В активных деревьях в качестве узлов используют компьютеры, в пассивных — коммутаторы.

Таким образом эта топология объединяет в себе свойства двух других топологий: шина и звезда.

К достоинствам данной топологии можно отнести то, что сеть с данной топологией легко увеличить и легко её контролировать(поиск обрывов и неисправностей). Недостатками является то, что при выходе из строя родительского узла, выйдут из строя и все его дочерние узлы (выход из строя корня — выход из строя всей сети), и также ограничена пропускная способность (доступ к сети может быть затруднён). Последний недостаток, связанный с пропускной способностью, устраняется топологией «толстого» дерева.

## 13.Топология «Сота».

**Сотовая связь, сеть подвижной связи** — один из видов мобильной радиосвязи, в основе которого лежит сотовая сеть. Ключевая особенность заключается в том, что общая зона покрытия делится на ячейки (соты), определяющиеся зонами покрытия отдельных базовых станций (БС). Соты частично перекрываются и вместе образуют сеть. На идеальной (ровной и без застройки) поверхности зона покрытия одной БС представляет собой круг, поэтому составленная из них сеть имеет вид шестиугольных ячеек (сот).

Сеть составляют разнесённые в пространстве приёмопередатчики, работающие в одном и том же частотном диапазоне, и коммутирующее оборудование, позволяющее определять текущее местоположение подвижных абонентов и обеспечивать непрерывность связи при перемещении абонента из зоны действия одного приёмопередатчика в зону действия другого.

### **Принцип действия**

Основные составляющие сотовой сети — это сотовые телефоны и базовые станции, которые обычно располагают на крышах зданий и вышках. Будучи включённым, сотовый телефон прослушивает эфир, находя сигнал базовой станции. После этого телефон посылает станции свой уникальный идентификационный код. Телефон и станция поддерживают постоянный радиоконтакт, периодически обмениваясь пакетами. Связь телефона со станцией может идти по аналоговому протоколу (AMPS, NAMPS, NMT-450) или по цифровому (DAMPS, CDMA, GSM, UMTS). Если телефон выходит из поля действия базовой станции (или качество радиосигнала сервисной соты ухудшается), он налаживает связь с другой (англ. handover).

Сотовые сети могут состоять из базовых станций разного стандарта, что позволяет оптимизировать работу сети и улучшить её покрытие.

Сотовые сети разных операторов соединены друг с другом, а также со стационарной телефонной сетью. Это позволяет абонентам одного оператора делать звонки абонентам другого оператора, с мобильных телефонов на стационарные и со стационарных на мобильные.

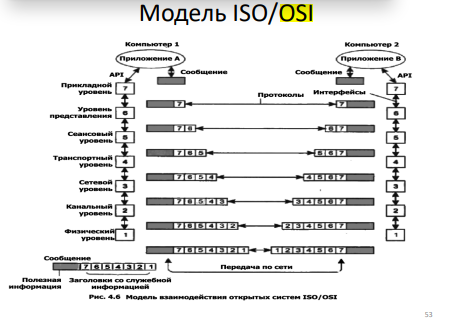
Операторы могут заключать между собой договоры роуминга. Благодаря таким договорам абонент, находясь вне зоны покрытия своей сети, может совершать и принимать звонки через сеть другого оператора. Как правило, это осуществляется по повышенным тарифам. Возможность автоматического роуминга появилась лишь в стандартах 2G и является одним из главных отличий от сетей 1G.[2]

Операторы могут совместно использовать инфраструктуру сети, сокращая затраты на развертывание сети и текущие издержки.

# 4.Модель ISO/OSI

## 14.Описание модели ISO/OSI.

Open Systems Interconnection Basic Reference Model (ISOBRM) — базовая эталонная модель взаимодействия открытых систем. Разрабатывалась очччень долго, поэтому в настоящее время не применяется.



## 15.Уровни модели ISO/OSI.

**Уровень 1, физический**

Физический уровень (англ. physical layer) — нижний уровень модели, который определяет метод передачи данных, представленных в двоичном виде, от одного устройства (компьютера) к другому. Осуществляет передачу электрических или оптических сигналов в кабель или в радиоэфир и, соответственно, их приём и преобразование в биты данных в соответствии с методами кодирования цифровых сигналов. На этом уровне также работают концентраторы, повторители сигнала и медиаконвертеры.

Функции физического уровня реализуются на всех устройствах, подключенных к сети. Со стороны компьютера функции физического уровня выполняются сетевым адаптером или последовательным портом. К физическому уровню относятся физические, электрические и механические интерфейсы между двумя системами. Физический уровень определяет такие виды сред передачи данных как оптоволокно, витая пара, коаксиальный кабель, спутниковый канал передач данных и т. п. Стандартными типами сетевых интерфейсов, относящимися к физическому уровню, являются: V.35, RS-232, RS-485, RJ-11, RJ-45, разъемы AUI и BNC. Протоколы физического уровня: IEEE 802.15 (Bluetooth), IRDA, EIA RS-232, EIA-422, EIA-423, RS-449, RS-485, DSL, ISDN, SONET/SDH, 802.11 Wi-Fi, Etherloop, GSM Um radio interface, ITU и ITU-T, TransferJet, ARINC 818, G.hn/G.9960.

**Уровень 2, канальный Канальный** уровень (англ. data link layer) предназначен для обеспечения взаимодействия сетей на физическом уровне и контроля за ошибками, которые могут возникнуть. Полученные с физического уровня данные, представленные в битах, он упаковывает в кадры, проверяет их на целостность и, если нужно, исправляет ошибки (формирует повторный запрос поврежденного кадра) и отправляет на сетевой уровень. Канальный уровень может взаимодействовать с одним или несколькими физическими уровнями, контролируя и управляя этим взаимодействием.

Протоколы канального уровня: ARCnet, ATM, Controller Area Network (CAN), Econet, IEEE 802.3 (Ethernet), Ethernet Automatic Protection Switching (EAPS), Fiber Distributed Data Interface (FDDI), Frame Relay, High-Level Data Link Control (HDLC), IEEE 802.2 (предоставляет функции LLC для подуровня IEEE 802 MAC), Link Access Procedures, D channel (LAPD), IEEE 802.11 wireless LAN, LocalTalk, Multiprotocol Label Switching (MPLS), Point-to\_Point Protocol (PPP), Point-to-Point Protocol over Ethernet (PPPoE), Serial Line Internet Protocol (SLIP, устарел), StarLan, Token ring, Unidirectional Link Detection (UDLD), x.25, ARP.

**Уровень 3, сетевой уровень**

Сетевой уровень (англ. network layer) модели предназначен для определения пути передачи данных. Отвечает за трансляцию логических адресов и имён в физические, определение кратчайших маршрутов, коммутацию и маршрутизацию, отслеживание неполадок и «заторов» в сети. Протоколы сетевого уровня маршрутизируют данные от источника к получателю. Работающие на этом уровне устройства (маршрутизаторы) условно называют устройствами третьего уровня (по номеру уровня в модели OSI).

Протоколы сетевого уровня: IP/IPv4/IPv6 (Internet Protocol), IPX (Internetwork Packet Exchange, протокол межсетевого обмена), X.25 (частично этот протокол реализован на уровне 2), CLNP (сетевой протокол без организации соединений), IPsec (Internet Protocol Security). Протоколы маршрутизации - RIP (Routing Information Protocol), OSPF (Open Shortest Path First).

**Уровень 4, транспортный Транспортный уровень** (англ. transport layer) модели предназначен для обеспечения надёжной передачи данных от отправителя к получателю. При этом уровень надёжности может варьироваться в широких пределах. Существует множество классов протоколов транспортного уровня, начиная от протоколов, предоставляющих только основные транспортные функции (например, функции передачи данных без подтверждения приема), и заканчивая протоколами, которые гарантируют доставку в пункт назначения нескольких пакетов данных в надлежащей последовательности, мультиплексируют несколько потоков данных, обеспечивают механизм управления потоками данных и гарантируют достоверность принятых данных.

Протоколы транспортного уровня: ATP (AppleTalk Transaction Protocol), CUDP (Cyclic UDP), DCCP (Datagram Congestion Control Protocol), FCP (Fiber Channel Protocol), IL (IL Protocol), NBF (NetBIOS Frames protocol), NCP (NetWare Core Protocol), SCTP (Stream Control Transmission Protocol), SPX (Sequenced Packet Exchange), SST (Structured Stream Transport), TCP (Transmission Control Protocol), UDP (User Datagram Protocol).

**Уровень 5, сеансовый**

Сеансовый уровень (англ. session layer) модели обеспечивает поддержание сеанса связи, позволяя приложениям взаимодействовать между собой длительное время. Уровень управляет созданием/завершением сеанса, обменом информацией, синхронизацией задач, определением права на передачу данных и поддержанием сеанса в периоды неактивности приложений. 62Уровень 5, сеансовый Протоколы сеансового уровня: ADSP (AppleTalk Data Stream Protocol), ASP (AppleTalk Session Protocol), H.245 (Call Control Protocol for Multimedia Communication), ISO-SP (OSI Session Layer Protocol (X.225, ISO 8327)), iSNS (Internet Storage Name Service), L2F (Layer 2 Forwarding Protocol), L2TP (Layer 2 Tunneling Protocol), NetBIOS (Network Basic Input Output System), PAP (Password Authentication Protocol), PPTP (Point-to-Point Tunneling Protocol), RPC (Remote Procedure Call Protocol), RTCP (Real-time Transport Control Protocol), SMPP (Short Message Peer-to-Peer), SCP (Session Control Protocol), ZIP (Zone Information Protocol), SDP (Sockets Direct Protocol)

**Уровень 6, представления**

уровень представления; англ. presentation layer) обеспечивает преобразование протоколов и кодирование/декодирование данных. Запросы приложений, полученные с прикладного уровня, на уровне представления преобразуются в формат для передачи по сети, а полученные из сети данные преобразуются в формат приложений. На этом уровне может осуществляться сжатие/распаковка или шифрование/дешифрование, а также перенаправление запросов другому сетевому ресурсу, если они не могут быть обработаны локально.

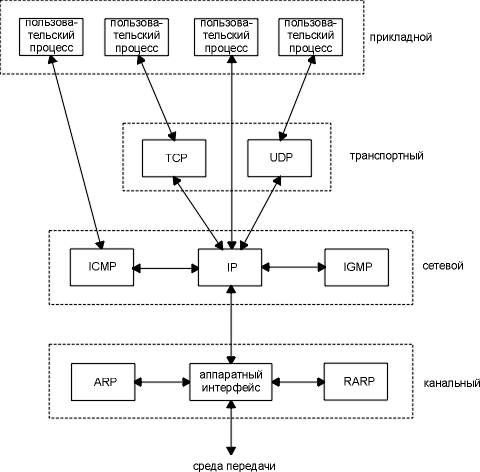
Уровень представлений обычно представляет собой промежуточный протокол для преобразования информации из соседних уровней. Это позволяет осуществлять обмен между приложениями на разнородных компьютерных системах прозрачным для приложений образом. Уровень представлений обеспечивает форматирование и преобразование кода. Форматирование кода используется для того, чтобы гарантировать приложению поступление информации для обработки, которая имела бы для него смысл. При необходимости этот уровень может выполнять перевод из одного формата данных в другой. Уровень представлений имеет дело не только с форматами и представлением данных, он также занимается структурами данных, которые используются программами. Таким образом, уровень 6 обеспечивает организацию данных при их пересылке.

**Уровень 7, прикладной** Прикладной уровень (уровень приложений; англ. application layer) — верхний уровень модели, обеспечивающий взаимодействие пользовательских приложений с сетью: • позволяет приложениям использовать сетевые службы: • удалённый доступ к файлам и базам данных, • пересылка электронной почты; • отвечает за передачу служебной информации; • предоставляет приложениям информацию об ошибках; • формирует запросы к уровню представления. Протоколы прикладного уровня: RDP, HTTP, SMTP, SNMP, POP3, FTP, XMPP, OSCAR, Modbus, SIP, TELNET и другие.

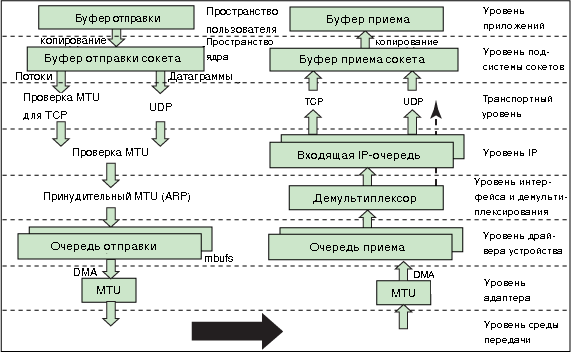
# 5.Стек протоколов TCP/IP

## 16.Основные понятия.

### Обмен данными TCP/IP

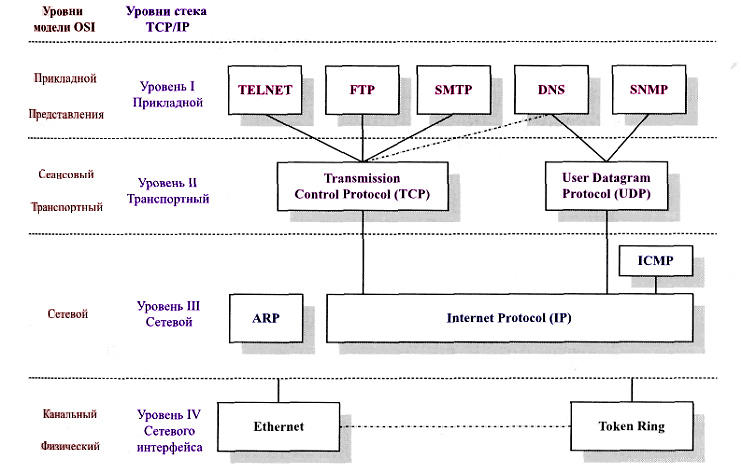


### Обмен данными на низком уровне



## 17.Описание уровней.

**Уровни модели TCP/IP**



### 1.Прикладной уровень

Множество протоколов, которые обеспечивают работу огромного числа сервисов. HTTP для веб-сайтов, FTP для файловых хранилищ и тд

### 2.Транспортный уровень

На этом уровне функционируют протокол управления передачей **TCP** (Transmission Control Protocol) и протокол дейтаграмм пользователя **UDP** (User Datagram Protocol). Протокол TCP обеспечивает надежную передачу сообщений между удаленными прикладными процессами за счет образования виртуальных соединений. Протокол UDP обеспечивает передачу прикладных пакетов дейтаграммным способом, как и IP, и выполняет только функции связующего звена между сетевым протоколом и многочисленными прикладными процессами.

### 3.Сетевой уровень

Протоколы сетевого уровня обеспечивают соединение между двумя компьютерами в сети, передачу пакетов с использованием различных транспортных технологий локальных сетей, территориальных сетей, линий специальной связи и т. п.

Протоколы принимают запросы на отправку пакетов (содержащие адрес получателя) от транспортного уровня, преобразуют пакеты в дейтаграммы и отправляют их на уровень сетевого интерфейса для дальнейшей обработки.

Базовый протокол – IP.

#### Протокол ARP

Базовый протокол сетевого уровня - это **Протокол преобразования адресов** (**ARP**). **ARP** выполняет динамическое преобразование IP-адресов в уникальные физические адреса хостов локальных сетей.

В качестве примера работы ARP рассмотрим два узла - *X* и *Y*. Если узел X хочет установить соединение с узлом Y, и при этом X и Y расположены в разных сетях , то *X* и *Y* взаимодействуют через мосты, маршрутизаторы или шлюзы, идентифицируя друг друга по IP-адресам. Внутри сети узлы обмениваются данными с помощью низкоуровневых аппаратных адресов.

Если узлы подключены к одному и тому же сегменту сети, то они определяют физические адреса партнеров по протоколу **ARP**. В этом случае узел X рассылает запрос **ARP** об аппаратном адресе узла Y всем узлам сети. В запросе **ARP** указывает аппаратный и **IP**-адрес узла X и **IP**-адрес узла Y. При получении запроса **ARP** узел Y помещает запись об узле X в свой кэш **ARP** (применяемый для быстрого преобразования IP-адресов в аппаратные адреса), а затем отправляет узлу X ответ **ARP**, содержащий аппаратный и **IP**-адрес узла Y. Когда узел X получает от узла Y ответ **ARP**, он помещает запись об узле Y в свой кэш **ARP**.

После того как в кэш **ARP** узла X добавлена запись об узле Y, узел X сможет отправлять пакеты непосредственно узлу Y без обращения к **ARP** (после удаления записи об узле Y из кэша **ARP** узел X будет вынужден снова обратиться к **ARP**).

В отличие от большинства протоколов, в протоколе **ARP** не зафиксирован формат заголовков. Сообщения ARP поддерживаются в различных локальных сетях:

* Адаптеры Ethernet LAN (поддерживает протоколы Ethernet и 802.3)
* Адаптеры Token-Ring
* Адаптеры FDDI (Оптоволоконного интерфейса распределенных данных)

#### Протокол ICMP

Вторым протоколом сетевого уровня является **Протокол управляющих сообщений Internet** (**ICMP**). **ICMP** - обязательная часть любой реализации **IP**. **ICMP** отправляет сообщения об ошибках и управляющие сообщения протоколу **IP**.

С помощью этого протокола шлюзы и хосты отправляют источнику пакетов отчеты о неполадках. **ICMP** выполняет следующие функции:

* Проверяет, что хост-получатель активен и доступен
* Сообщает об неправильных параметрах в заголовке дейтаграммы
* Синхронизует часы и определяет время передачи данных по маршруту
* Определяет IP-адреса и маски подсетей

Как и протоколы более высокого уровня, протокол **ICMP** использует базовые функции **IP**. Однако в действительности **ICMP**представляет собой часть протокола **IP** и должен быть реализован в каждом модуле **IP**.

Протокол **ICMP** - это всего лишь средство обмена информацией о неполадках в сети. Он не повышает надежность протокола **IP**. Таким образом, **ICMP** не гарантирует надежной доставки **IP**-пакета, а также доставки сообщения **ICMP** в случае, если **IP**-пакет не был получен или был получен в искаженном виде.

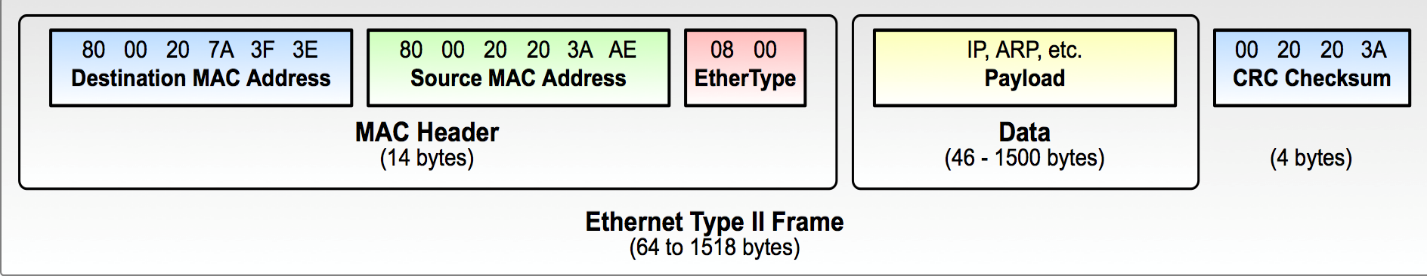
### 4. Уровень сетевого интерфейса

Соответствует физическому и канальному уровням модели OSI. Этот уровень в протоколах TCP/IP не регламентируется, но поддерживает все популярные стандарты физического и канального уровня: для локальных сетей это Ethernet, Token Ring, FDDI, Fast Ethernet, 100VG-AnyLAN, для глобальных сетей - протоколы соединений "точка-точка" SLIP и PPP, протоколы территориальных сетей с коммутацией пакетов X.25, frame relay.

Ethernet - Cемейство технологий пакетной передачи данных для компьютерных сетей.

Cтандарты Ethernet определяют проводные соединения и электрические сигналы на физическом уровне, формат кадров и протоколы управления доступом к среде — на канальном уровне модели OSI. Ethernet в основном описывается стандартами IEEE.

#### Формат Ethernet кадра



#### Структура MAC Адреса



**Передача данных TCP/IP**



## 18.Основные протоколы стека.

#### Протокол ARP

Базовый протокол сетевого уровня - это **Протокол преобразования адресов** (**ARP**). **ARP** выполняет динамическое преобразование IP-адресов в уникальные физические адреса хостов локальных сетей.

В качестве примера работы ARP рассмотрим два узла - *X* и *Y*. Если узел X хочет установить соединение с узлом Y, и при этом X и Y расположены в разных сетях , то *X* и *Y* взаимодействуют через мосты, маршрутизаторы или шлюзы, идентифицируя друг друга по IP-адресам. Внутри сети узлы обмениваются данными с помощью низкоуровневых аппаратных адресов.

Если узлы подключены к одному и тому же сегменту сети, то они определяют физические адреса партнеров по протоколу **ARP**. В этом случае узел X рассылает запрос **ARP** об аппаратном адресе узла Y всем узлам сети. В запросе **ARP** указывает аппаратный и **IP**-адрес узла X и **IP**-адрес узла Y. При получении запроса **ARP** узел Y помещает запись об узле X в свой кэш **ARP** (применяемый для быстрого преобразования IP-адресов в аппаратные адреса), а затем отправляет узлу X ответ **ARP**, содержащий аппаратный и **IP**-адрес узла Y. Когда узел X получает от узла Y ответ **ARP**, он помещает запись об узле Y в свой кэш **ARP**.

После того как в кэш **ARP** узла X добавлена запись об узле Y, узел X сможет отправлять пакеты непосредственно узлу Y без обращения к **ARP** (после удаления записи об узле Y из кэша **ARP** узел X будет вынужден снова обратиться к **ARP**).

В отличие от большинства протоколов, в протоколе **ARP** не зафиксирован формат заголовков. Сообщения ARP поддерживаются в различных локальных сетях:

* Адаптеры Ethernet LAN (поддерживает протоколы Ethernet и 802.3)
* Адаптеры Token-Ring
* Адаптеры FDDI (Оптоволоконного интерфейса распределенных данных)

#### Протокол ICMP

Вторым протоколом сетевого уровня является **Протокол управляющих сообщений Internet** (**ICMP**). **ICMP** - обязательная часть любой реализации **IP**. **ICMP** отправляет сообщения об ошибках и управляющие сообщения протоколу **IP**.

С помощью этого протокола шлюзы и хосты отправляют источнику пакетов отчеты о неполадках. **ICMP** выполняет следующие функции:

* Проверяет, что хост-получатель активен и доступен
* Сообщает об неправильных параметрах в заголовке дейтаграммы
* Синхронизует часы и определяет время передачи данных по маршруту
* Определяет IP-адреса и маски подсетей

Как и протоколы более высокого уровня, протокол **ICMP** использует базовые функции **IP**. Однако в действительности **ICMP**представляет собой часть протокола **IP** и должен быть реализован в каждом модуле **IP**.

Протокол **ICMP** - это всего лишь средство обмена информацией о неполадках в сети. Он не повышает надежность протокола **IP**. Таким образом, **ICMP** не гарантирует надежной доставки **IP**-пакета, а также доставки сообщения **ICMP** в случае, если **IP**-пакет не был получен или был получен в искаженном виде.

# 6.Протокол TCP и UDP

## 19.Основные понятия

**TCP** (Transmission Control Protocol) устанавливает соединение, данные передает потоком, осуществляет повторный запрос данных в случае потери данных и устраняет дублирование при получении двух копий одного пакета, **гарантируя целостность передаваемых данных и уведомление отправителя о результатах передачи** тем самым, в отличие от UDP.

* **Клиент** – сторона обмена, инициирующая обмен
* **Сервер** – сторона обмена данными, которая пассивно ожидает подключений для обмена
* **Соединение** (connection) – подключение клиента к серверу

Квитирование

Если простыми словами, то это “чек” о том, что пакет был доставлен.Один из традиционных методов обеспечения надежной связи.

Для того, чтобы можно было организовать повторную передачу искаженных данных отправитель нумерует отправляемые единицы передаваемых данных (далее для простоты называемые кадрами). Для каждого кадра *отправитель* ожидает от *приемника* так называемую положительную **квитанцию** - служебное сообщение, извещающее о том, что исходный кадр был получен и данные в нем оказались корректными. Время этого ожидания **ограничено** - при отправке каждого кадра отправитель запускает таймер, и если по его истечению положительная квитанция не получена, то кадр считается утерянным. В некоторых протоколах приемник, в случае получения кадра с искаженными данными должен отправить отрицательную квитанцию - явное указание того, что данный кадр нужно передать повторно.

Заголовок TCP

**Порядковый номер** выполняет две задачи:

* Если установлен флаг **SYN**, то это изначальный порядковый номер — ISN (Initial Sequence Number), и первый байт данных, которые будут переданы в следующем пакете, будет иметь номер, равный ISN + 1.
* В противном случае, если SYN не установлен, первый байт данных, передаваемый в данном пакете, имеет этот порядковый номер

Поскольку поток передаваемых TCP по данных в общем случае может быть длиннее, чем число различных состояний этого поля, то все операции с порядковым номером должны выполняться по модулю 2^32. Это накладывает практическое ограничение на использование TCP. Если скорость передачи коммуникационной системы такова, чтобы в течение MSL (максимального времени жизни сегмента) произошло переполнение порядкового номера, то в сети может появиться два сегмента с одинаковым номером, относящихся к разным частям потока, и приёмник получит некорректные данные.

**Порт источника** идентифицирует приложение клиента, с которого отправлены пакеты. Ответные данные передаются клиенту на основании этого номера.

**Порт назначения** идентифицирует порт, на который отправлен пакет.

**Номер подтверждения** если установлен флаг ***ACK***, то это поле содержит порядковый номер, ожидаемый получателем в следующий раз. Помечает этот сегмент как подтверждение получения.

**Размер окна** содержит число, определяющее в байтах размер данных, которые отправитель может отправить без получения подтверждения.

**Указатель срочности**16-битовое значение положительного смещения от порядкового номера в данном сегменте. Это поле указывает порядковый номер октета, которым заканчиваются важные (urgent) данные. Поле принимается во внимание только для пакетов с установленным флагом **URG**.

**Флаги пакета TCP**

Поле содержит 6 битовых флагов:

* **URG** — поле *«Указатель важности»* задействовано (англ. *Urgent pointer field is significant*)
* **ACK** — поле *«Номер подтверждения»* задействовано (англ. *Acknowledgement field is significant*)
* **PSH** — (англ. *Push function*) инструктирует получателя протолкнуть данные, накопившиеся в приемном буфере, в приложение пользователя
* **RST** — оборвать соединения, сбросить буфер (очистка буфера) (англ. *Reset the connection*)
* **SYN** — синхронизация номеров последовательности (англ. *Synchronize sequence numbers*)

**FIN** (англ. *final*, бит) — флаг, будучи установлен, указывает на завершение соединения (англ. *FIN bit used for connection termination*)

## 20.Алгоритм Установления соединения в TCP

В отличие от UDP, который может сразу же начать передачу пакетов, TCP устанавливает соединения, которые должны быть созданы перед передачей данных. TCP соединение можно разделить на 3 стадии:

Установка соединения --> Передача данных --> Завершение соединения

2.1 Состояния сеанса TCP

|  |  |
| --- | --- |
| CLOSED | Начальное состояние узла. Фактически фиктивное |
| LISTEN | Сервер ожидает запросов установления соединения от клиента |
| SYN-SENT | Клиент отправил запрос серверу на установление соединения и ожидает ответа |
| SYN-RECEIVED | Сервер получил запрос на соединение, отправил ответный запрос и ожидает подтверждения |
| ESTABLISHED | Соединение установлено, идёт передача данных |
| FIN-WAIT-1 | Одна из сторон (назовём её узел-1) завершает соединение, отправив сегмент с флагом FIN |
| CLOSE-WAIT | Другая сторона (узел-2) переходит в это состояние, отправив, в свою очередь сегмент ACK и продолжает одностороннюю передачу |
| FIN-WAIT-2 | Узел-1 получает ACK, продолжает чтение и ждёт получения сегмента с флагом FIN |
| LAST-ACK | Узел-2 заканчивает передачу и отправляет сегмент с флагом FIN |
| TIME-WAIT | Узел-1 получил сегмент с флагом FIN, отправил сегмент с флагом ACK и ждёт 2\*MSL секунд, перед окончательным закрытием соединения |
| CLOSING | Обе стороны инициировали закрытие соединения одновременно: после отправки сегмента с флагом FIN узел-1 также получает сегмент FIN, отправляет ACK и находится в ожидании сегмента ACK (подтверждения на свой запрос |

Процесс начала сеанса TCP состоит из трёх шагов:

1. Клиент, устанавливающий соединение, посылает серверу сегмент с номером последовательности и флагом SYN.

Сервер получает сегмент, запоминает номер последовательности и пытается создать сокет (буферы и управляющие структуры памяти) для обслуживания нового клиента.

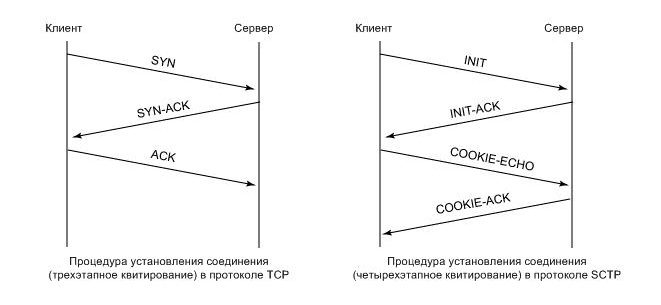
В случае успеха сервер посылает клиенту сегмент с номером последовательности и флагами SYN и ACK, и переходит в состояние SYN-RECEIVED.

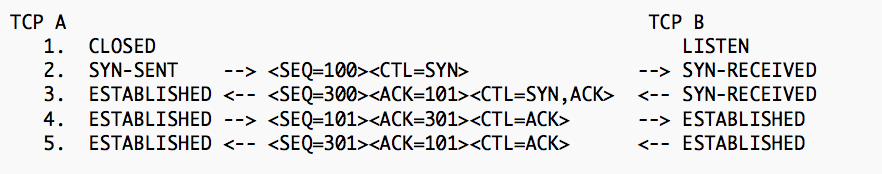
В случае неудачи сервер посылает клиенту сегмент с флагом RST.

1. Если клиент получает сегмент с флагом SYN, то он запоминает номер последовательности и посылает сегмент с флагом ACK.

* Если он одновременно получает и флаг ACK (что обычно и происходит), то он переходит в состояние ESTABLISHED.
* Если клиент получает сегмент с флагом RST, то он прекращает попытки соединиться.
* Если клиент не получает ответа в течение 10 секунд, то он повторяет процесс соединения заново.

1. Если сервер в состоянии SYN-RECEIVED получает сегмент с флагом ACK, то он переходит в состояние ESTABLISHED.

* В противном случае после тайм-аута он закрывает сокет и переходит в состояние CLOSED.

Пример базового 3-этапного согласования соединения

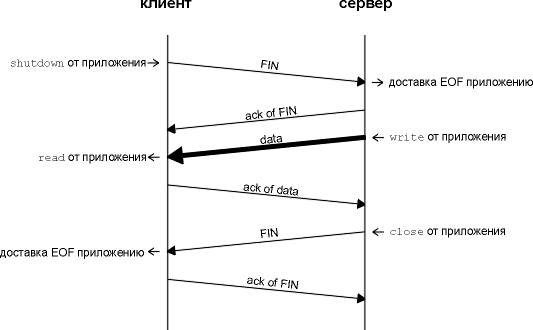
В строке 2 TCP A(клиент) начинает передачу сегмента SYN, говорящего об использовании номеров последовательности, начиная со 100. В строке 3 TCP B(сервер) передаёт SYN и подтверждение для принятого SYN в адрес TCP A. Надо отметить, что поле подтверждения показывает ожидание TCP B приёма номера последовательности 101, подтверждающего SYN с номером 100.

В строке 4 TCP A отвечает пустым сегментом с подтверждением ACK для сегмента SYN от TCP B; в строке 5 TCP B передаёт некоторые данные. Отметим, что номер подтверждения сегмента в строке 5 (ACK=101) совпадает с номером последовательности в строке 4 (SEQ=101), поскольку ACK не занимает пространства номеров последовательности (если это сделать, придётся подтверждать подтверждения — ACK для ACK).

Разрыв TCP соединения

Завершение соединения можно рассмотреть в три этапа:

* Посылка серверу от клиента флага FIN на завершение соединения.
* Сервер посылает клиенту флаги ответа ACK , FIN, что соединение закрыто.
* После получения этих флагов клиент закрывает соединение и в подтверждение отправляет серверу ACK , что соединение закрыто.



В отличие от UDP, который может сразу же начать передачу пакетов, TCP устанавливает соединения, которые должны быть созданы перед передачей данных. TCP соединение можно разделить на 3 стадии:

Установка соединения --> Передача данных --> Завершение соединения

2.1 Состояния сеанса TCP

|  |  |
| --- | --- |
| CLOSED | Начальное состояние узла. Фактически фиктивное |
| LISTEN | Сервер ожидает запросов установления соединения от клиента |
| SYN-SENT | Клиент отправил запрос серверу на установление соединения и ожидает ответа |
| SYN-RECEIVED | Сервер получил запрос на соединение, отправил ответный запрос и ожидает подтверждения |
| ESTABLISHED | Соединение установлено, идёт передача данных |
| FIN-WAIT-1 | Одна из сторон (назовём её узел-1) завершает соединение, отправив сегмент с флагом FIN |
| CLOSE-WAIT | Другая сторона (узел-2) переходит в это состояние, отправив, в свою очередь сегмент ACK и продолжает одностороннюю передачу |
| FIN-WAIT-2 | Узел-1 получает ACK, продолжает чтение и ждёт получения сегмента с флагом FIN |
| LAST-ACK | Узел-2 заканчивает передачу и отправляет сегмент с флагом FIN |
| TIME-WAIT | Узел-1 получил сегмент с флагом FIN, отправил сегмент с флагом ACK и ждёт 2\*MSL секунд, перед окончательным закрытием соединения |
| CLOSING | Обе стороны инициировали закрытие соединения одновременно: после отправки сегмента с флагом FIN узел-1 также получает сегмент FIN, отправляет ACK и находится в ожидании сегмента ACK (подтверждения на свой запрос |

Процесс начала сеанса TCP состоит из трёх шагов:

1. Клиент, устанавливающий соединение, посылает серверу сегмент с номером последовательности и флагом SYN.

Сервер получает сегмент, запоминает номер последовательности и пытается создать сокет (буферы и управляющие структуры памяти) для обслуживания нового клиента.

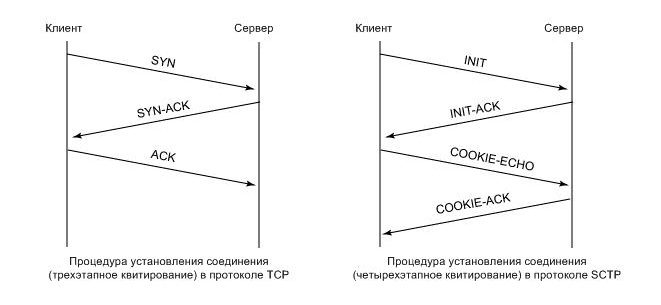
В случае успеха сервер посылает клиенту сегмент с номером последовательности и флагами SYN и ACK, и переходит в состояние SYN-RECEIVED.

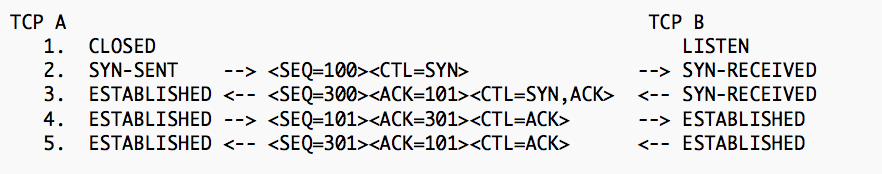
В случае неудачи сервер посылает клиенту сегмент с флагом RST.

1. Если клиент получает сегмент с флагом SYN, то он запоминает номер последовательности и посылает сегмент с флагом ACK.

* Если он одновременно получает и флаг ACK (что обычно и происходит), то он переходит в состояние ESTABLISHED.
* Если клиент получает сегмент с флагом RST, то он прекращает попытки соединиться.
* Если клиент не получает ответа в течение 10 секунд, то он повторяет процесс соединения заново.

1. Если сервер в состоянии SYN-RECEIVED получает сегмент с флагом ACK, то он переходит в состояние ESTABLISHED.

* В противном случае после тайм-аута он закрывает сокет и переходит в состояние CLOSED.

Пример базового 3-этапного согласования соединения

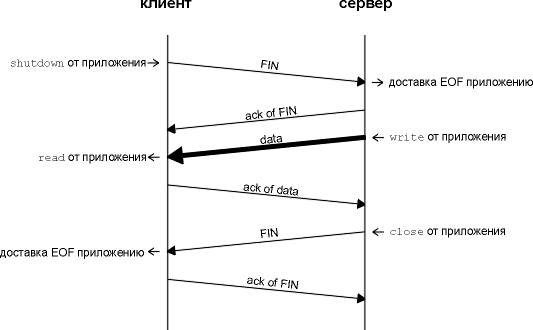
В строке 2 TCP A(клиент) начинает передачу сегмента SYN, говорящего об использовании номеров последовательности, начиная со 100. В строке 3 TCP B(сервер) передаёт SYN и подтверждение для принятого SYN в адрес TCP A. Надо отметить, что поле подтверждения показывает ожидание TCP B приёма номера последовательности 101, подтверждающего SYN с номером 100.

В строке 4 TCP A отвечает пустым сегментом с подтверждением ACK для сегмента SYN от TCP B; в строке 5 TCP B передаёт некоторые данные. Отметим, что номер подтверждения сегмента в строке 5 (ACK=101) совпадает с номером последовательности в строке 4 (SEQ=101), поскольку ACK не занимает пространства номеров последовательности (если это сделать, придётся подтверждать подтверждения — ACK для ACK).

Разрыв TCP соединения

Завершение соединения можно рассмотреть в три этапа:

* Посылка серверу от клиента флага FIN на завершение соединения.
* Сервер посылает клиенту флаги ответа ACK , FIN, что соединение закрыто.
* После получения этих флагов клиент закрывает соединение и в подтверждение отправляет серверу ACK , что соединение закрыто.



## 21.UDP

**ТУТ ПАРАША ТАК ЧТО ИЩИТЕ ИНФУ САМИ В ПРЕЗЕНТАХ!!!**

Протокол **UDP** (User Datagram Protocol) является одним из основных протоколов, расположенных непосредственно над IP. Он предоставляет прикладным процессам транспортные услуги, немногим отличающиеся от услуг протокола IP.

Протокол UDP обеспечивает доставку дейтограмм и, в отличии от TCP, не требует подтверждения их получения, не требует соединения с удаленным модулем UDP.

К заголовку IP-пакета UDP добавляет поля *порт отправителя* и *порт получателя*, которые обеспечивают мультиплексирование информации между различными прикладными процессами, а также поля *длина* UDP-дейтограммы и *контрольная сумма*, позволяющие поддерживать целостность данных.

Таким образом, если на уровне IP для определения места доставки пакета используется адрес, на уровне UDP - номер порта.

Малые накладные расходы, связанные с форматом UDP, а также отсутствие необходимости подтверждения получения пакета, делают этот протокол наиболее популярным при реализации приложений мультимедиа, но главное его место работы - локальные сети и мультимедиа.

Структура пакета UDP

Длина сообщения равна числу байт в UDP-дейтограмме, включая заголовок.

Поле UDP **контрольная сумма** содержит код, полученный в результате контрольного суммирования UDP-заголовка и поля данные. Не трудно видеть, что этот протокол использует заголовок минимального размера (8 байт).

# 7.Протокол IP.

## 22. Основные понятия.

IP объединяет сегменты сети в единую сеть, обеспечивая доставку пакетов данных между любыми узлами сети через произвольное число промежуточных узлов. IP не гарантирует надёжной доставки пакета до адресата — в частности, пакеты могут прийти не в том порядке, в котором были отправлены, могут появиться дубли, оказаться повреждёнными (обычно повреждённые пакеты уничтожаются) или не прийти вовсе.

**IP-адрес**

Уникальный сетевой адрес узла в компьютерной сети, построенной по протоколу IP. В сети Интернет требуется глобальная уникальность адреса; в случае работы в локальной сети требуется уникальность адреса в пределах сети. В версии протокола IPv4 IP-адрес имеет длину 4 байта, а в версии протокола IPv6 IP-адрес имеет длину 16 байт.

**IPv4**

В сети Интернет используется IP четвёртой версии, также известный как IPv4. В протоколе IP этой версии каждому узлу сети ставится в соответствие IP-адрес длиной 4 байта. При этом компьютеры в подсетях объединяются общими битами адреса. Количество этих бит, общее для данной подсети, называется маской подсети.

По состоянию на сентябрь 2015 года об исчерпании общего запаса свободных блоков IPv4 адресов и ограничениях на выдачу новых диапазонов адресов объявили все региональные регистраторы, кроме AfriNIC; ARIN объявил о полном исчерпании свободных IPv4 адресов, а для остальных регистраторов этот момент прогнозируется начиная с 2017 года. Выделение IPv4 адресов в Европе, Азии и Латинской Америке (регистраторы APNIC, RIPE NCC и LACNIC) продолжается блоками /22 (по 1024 адреса).

В 4-й версии IP-адрес представляет собой 32- битовое число. Удобной формой записи IPадреса (IPv4) является запись в виде четырёх десятичных чисел значением от 0 до 255, разделённых точками, например, 192.168.0.3.

**IPv6**

Новая версия протокола IP, призванная решить проблемы, с которыми столкнулась предыдущая версия (IPv4) при её использовании в Интернете, за счёт использования длины адреса 128 бит вместо 32. После того, как адресное пространство в IPv4 закончится, два стека протоколов — IPv6 и IPv4 — будут использоваться параллельно (англ. dual stack), с постепенным увеличением доли трафика IPv6, по сравнению с IPv4. Такая ситуация станет возможной из-за наличия огромного количества устройств, в том числе устаревших, не поддерживающих IPv6 и требующих специального преобразования для работы с устройствами, использующими только IPv6.

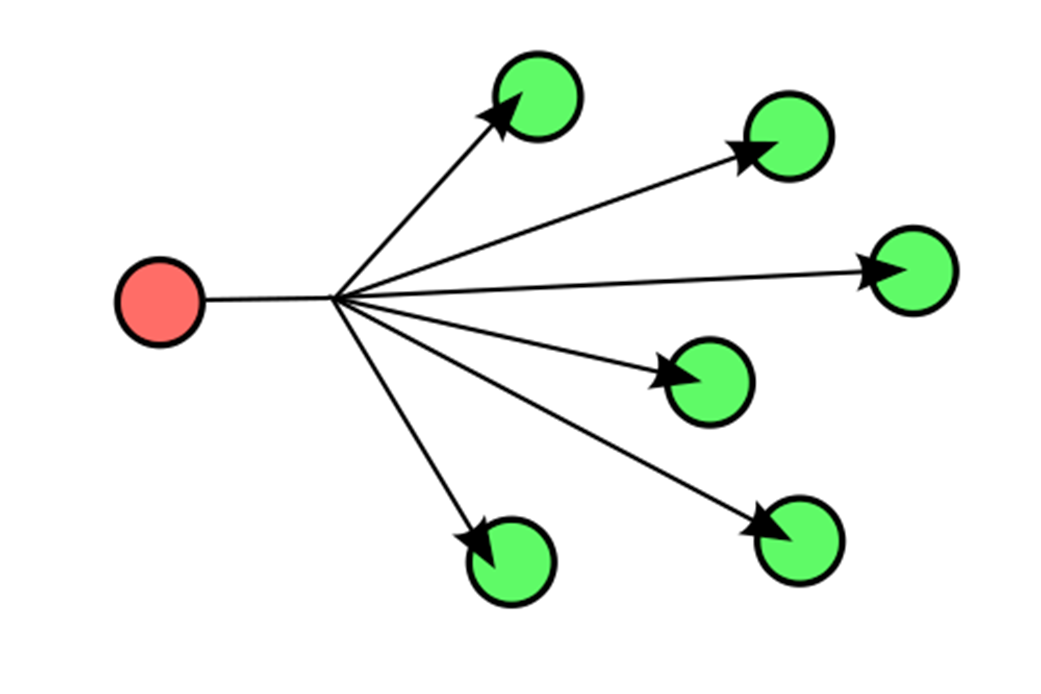
В 6-й версии IP-адрес является 128-битовым. Внутри адреса разделителем является двоеточие (напр. 2001:0db8:85a3:0000:0000:8a2e:0370:7334). Ведущие нули допускается в записи опускать. Нулевые группы, идущие подряд, могут быть опущены, вместо них ставится двойное двоеточие (fe80:0:0:0:0:0:0:1 можно записать как fe80::1). Более одного такого пропуска в адресе не допускается.

**Маска подсети**

Битовая маска, определяющая, какая часть IPадрес узла сети относится к адресу сети, а какая — к адресу самого узла в этой сети. Чтобы получить адрес сети, зная IP-адрес и маску подсети, необходимо применить к ним операцию Логическое И. При этом подходе маску подсети записывают вместе с IP-адресом в формате «IPадрес/количество единичных бит в маске»

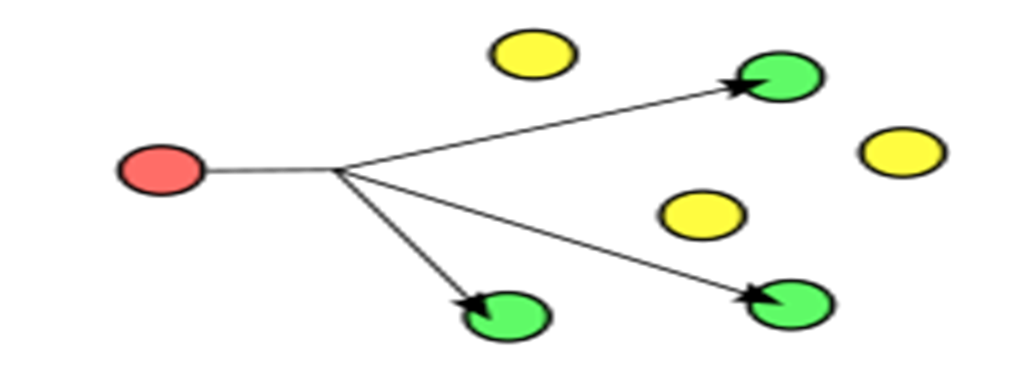
**Broadcast**

IP-сетях широковещательные адреса формируются следующим образом: к адресу подсети прибавляется побитовая инверсия маски подсети (то есть все биты адреса подсети, соответствующие нулям в маске, устанавливаются в «1»). Например, если адрес сети равен 192.168.0.0, маска подсети 255.255.255.0, то широковещательный адрес будет 192.168.0.255.



**Multicast**

Форма широковещания, при которой адресом назначения сетевого пакета является мультикастная группа (один ко многим). При этом данные получают все те, кто подписался к этой группе.



## 23. Классы подсетей.

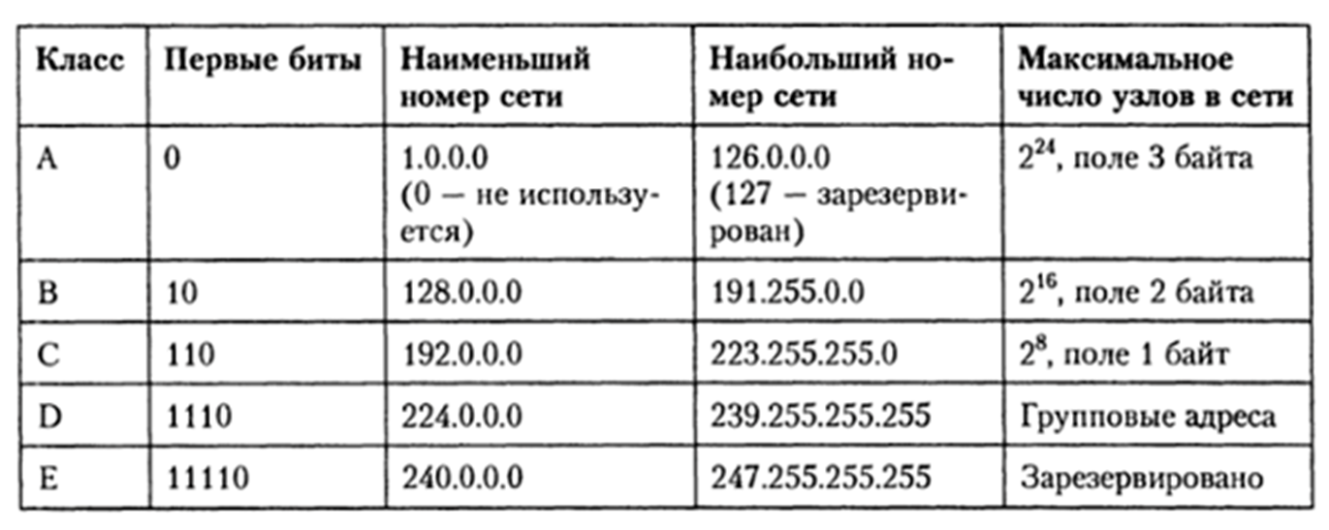
Типы адресации IPv4

INET – адресация, основанная на классах IP адресов. CIDR – бесклассовая адресация.

Классовая адресация сетей

IP адреса делятся на несколько классов. Метод не позволяет экономно использовать ограниченный ресурс IP-адресов, поскольку невозможно применение различных масок подсетей к различным подсетям.

Классы IP адресов



Классы IP адресов



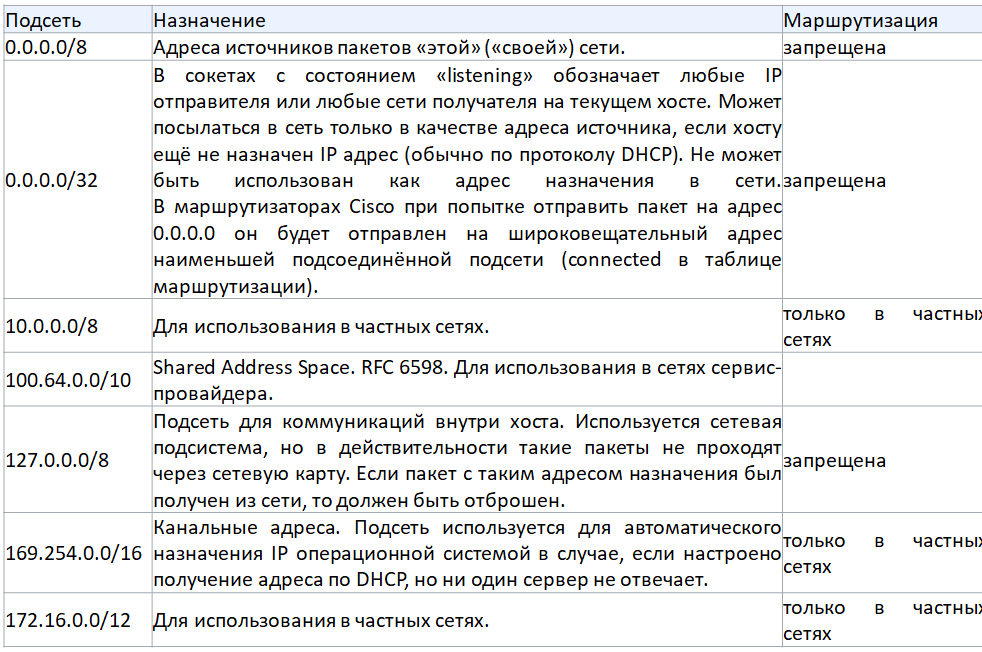
Бесклассовая адресация (англ. Classless Inter-Domain Routing, англ. CIDR)

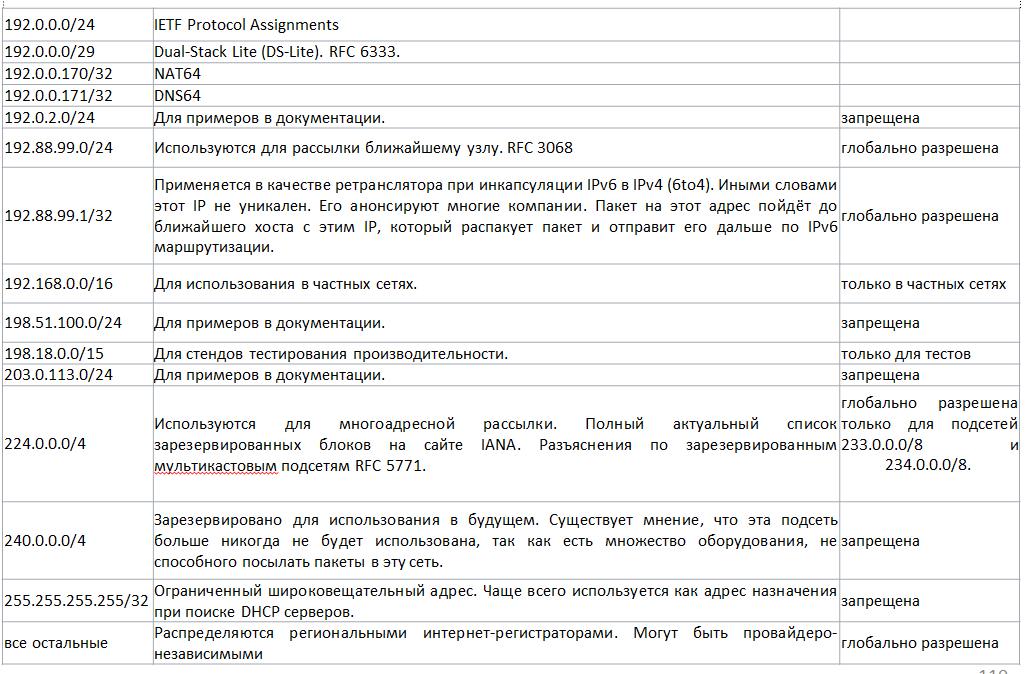
метод IP-адресации, позволяющий гибко управлять пространством IP-адресов, не используя жёсткие рамки классовой адресации. Использование этого метода позволяет экономно использовать ограниченный ресурс IP-адресов, поскольку возможно применение различных масок подсетей к различным подсетям.

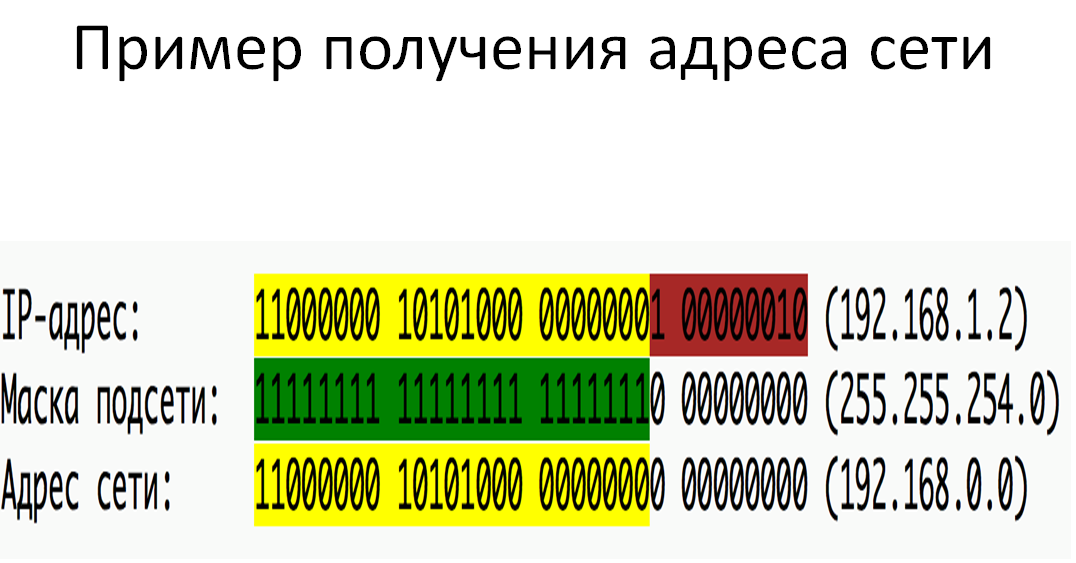
IP-адрес является массивом бит. Принцип IPадресации — выделение множества (диапазона, блока, подсети) IP-адресов, в котором некоторые битовые разряды имеют фиксированные значения, а остальные разряды пробегают все возможные значения. Блок адресов задаётся указанием начального адреса и маски подсети. Бесклассовая адресация основывается на переменной длине маски подсети.

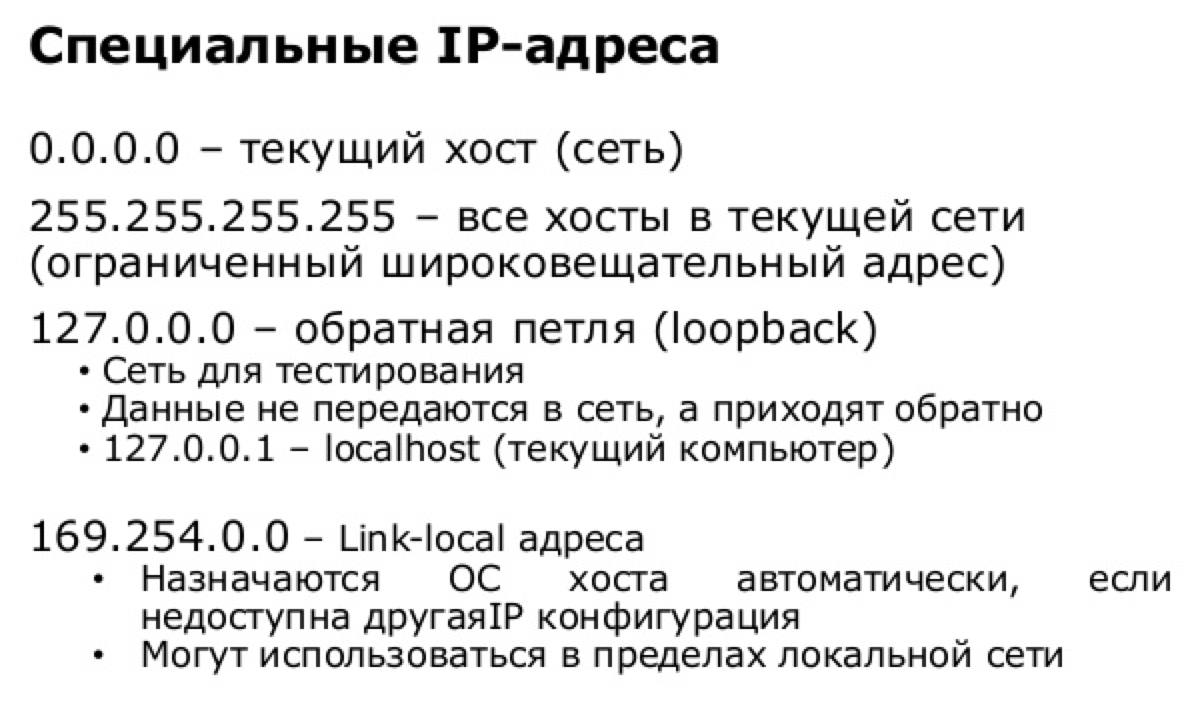
Специальные подсети IPv4

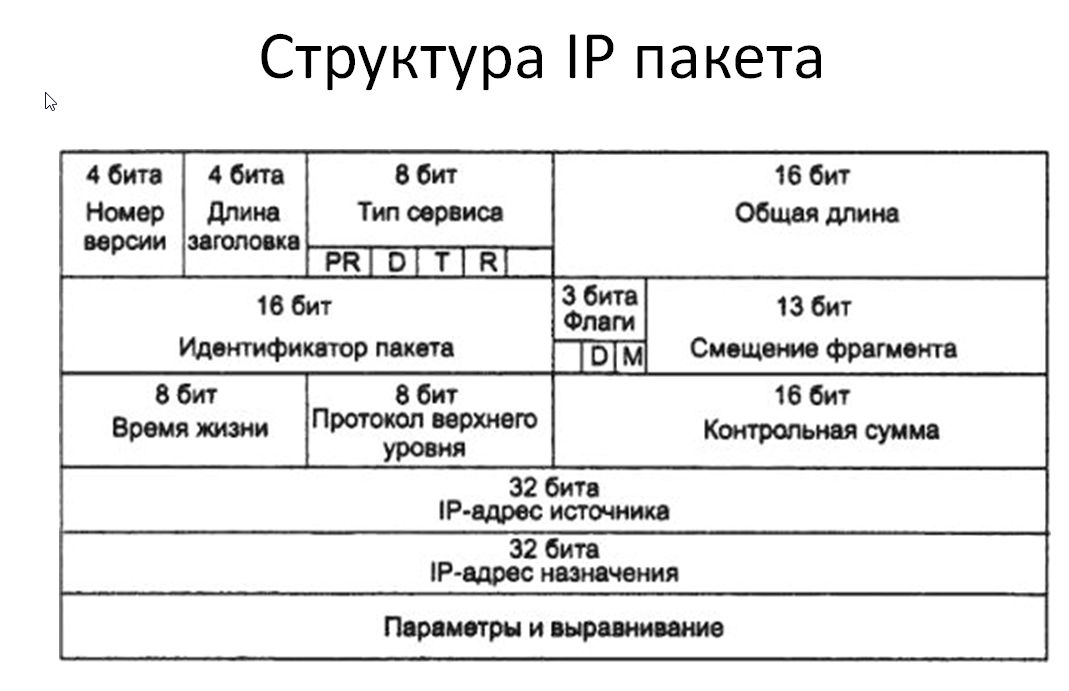
Некоторые адреса IPv4 зарезервированы для специальных целей и не предназначены для глобальной маршрутизации. Список подсетей специального назначения определён RFC 6890.

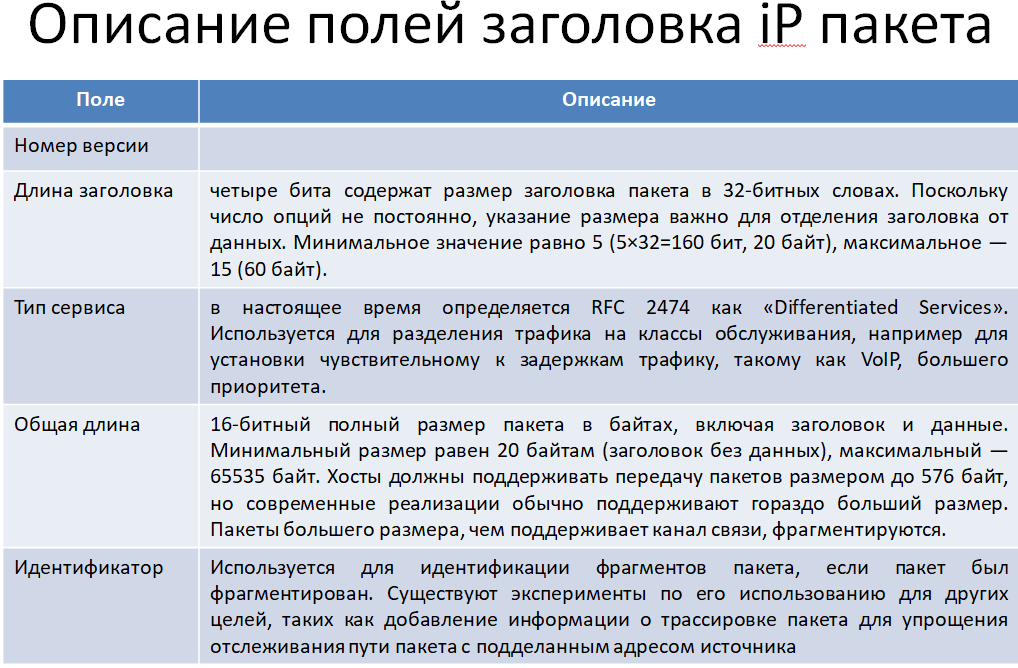












**Флаги**

Поле размером три бита (3 бита). Содержит флаги контроля над фрагментацией. Биты, от старшего к младшему, означают:

* 0: Зарезервирован, должен быть равен 0.
* 1: Не фрагментировать
* 2: У пакета ещё есть фрагменты

Если установлен флаг «не фрагментировать», то в случае необходимости фрагментации такой пакет будет уничтожен. Может использоваться для передачи данных хостам, не имеющим достаточных ресурсов для обработки фрагментированных пакетов.Флаг «есть фрагменты» должен быть установлен в 1 у всех фрагментов пакета, кроме последнего. У нефрагментированных устанавливается в 0 — такой пакет считается собственным последним фрагментом.

**Смещение фрагмента**

Поле размером в 13 бит, указывает смещение поля данных текущего фрагмента относительно начала поля данных первого фрагментированного пакета в блоках по 8 байт. Позволяет (213−1)×8=65528 байт смещения. При учёте размера заголовка итоговое смещение может превысить максимальный размер пакета (65528 + 20 = 65548 байт). Первый фрагмент в последовательности имеет нулевое смещение.

**«Время жизни» (Time to Live, TTL) пакета**

Определяет максимальное количество маршрутизаторов на пути следования пакета. Наличие этого параметра не позволяет пакету бесконечно ходить по сети. Каждый маршрутизатор при обработке пакета должен уменьшить значение TTL на единицу. Пакеты, время жизни которых стало равно нулю, уничтожаются, а отправителю посылается сообщение ICMP *Time Exceeded*. На отправке пакетов с разным временем жизни основана трассировка их пути прохождения (traceroute). Максимальное значение TTL=255. Обычное начальное значение TTL=64 (зависит от ОС).

**Протокол**

Указывает, данные какого протокола IP содержит пакет (например, TCP или ICMP). Присвоенные номера протоколов можно найти на сайте IANA

*Контрольная сумма заголовка*

16-битная контрольная сумма, используемая для проверки целостности заголовка. Каждый хост или маршрутизатор сравнивает контрольную сумму заголовка со значением этого поля и отбрасывает пакет, если они не совпадают. Целостность данных IP не проверяет — она проверяется протоколами более высоких уровней (такими, как TCP или UDP), которые тоже используют контрольные суммы.Поскольку TTL уменьшается на каждом шаге прохождения пакета, сумма тоже должна вычисляться на каждом шаге. Метод пересчёта контрольной суммы определён в RFC 1071.

**Адрес источника**

32-битный адрес отправителя пакета. Может не совпадать с настоящим адресом отправителя из-за трансляции адресов.

**Адрес назначения**

32-битный адрес получателя пакета. Также может меняться при трансляции адресов.

Опции

За адресом назначения может следовать поле дополнительных опций, но оно используется редко. Размер заголовка в этом случае должен быть достаточным, чтобы вместить все опции (с учётом дополнения до целого числа 32-битных слов). Присвоенные номера опций размещаются на сайте IANA. Если список опций не является концом заголовка, он должен оканчиваться опцией 0x00.

# 8.Маршрутизация в IP сетях.

## 24. Основные понятия.

**Маршрутизатором**, или **шлюзом**, называется узел сети с несколькими IP-интерфейсами (содержащими свой MAC-адрес и IP-адрес), подключенными к разным IP-сетям, осуществляющий на основе решения задачи маршрутизации перенаправление дейтаграмм из одной сети в другую для доставки от отправителя к получателю.

Маршрутизация служит для приема пакета от одного устройства и передачи его по сети другому устройству через другие сети. Если в сети нет маршрутизаторов, то не поддерживается маршрутизация. Маршрутизаторы направляют (перенаправляют) трафик во все сети, составляющие объединенную сеть.

**Динамическая маршрутизация** — это процесс протокола маршрутизации, определяющий взаимодействие устройства с соседними маршрутизаторами. Маршрутизатор будет обновлять сведения о каждой изученной им сети. Если в сети произойдет изменение, протокол динамической маршрутизации автоматически информирует об изменении все маршрутизаторы.

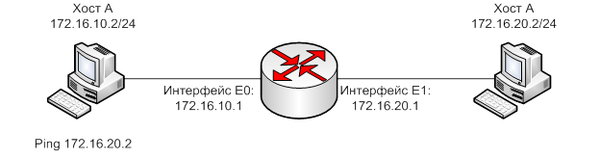
При **статической маршрутизации**, обновить таблицы маршрутизации на всех устройствах придется системному администратору.

Для маршрутизации пакета маршрутизатор должен владеть следующей информацией:

* Адрес назначения;
* Соседний(ие) маршрутизатор(ы), от которого(ых) он может узнать об удаленных сетях;
* Доступные пути ко всем удаленным сетям;
* Наилучший путь к каждой удаленной сети;
* Методы обслуживания и проверки информации о маршрутизации;

**IP-маршрутизация** простой процесс, который одинаков в сетях любого размера.

В примере пользователь хоста А запрашивает по ping IP-адрес хоста В.

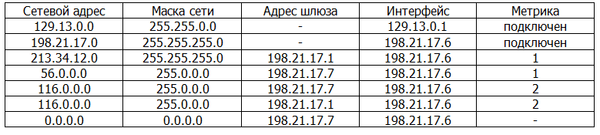


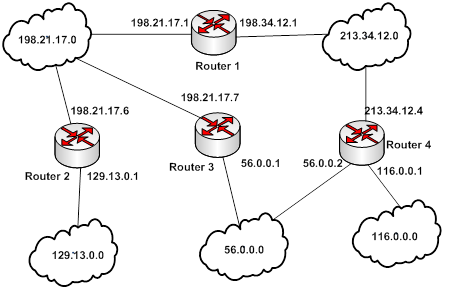
**Алгоритмы маршрутизации**

Основные требования к алгоритмам маршрутизации:

* точность;
* простота;
* надёжность;
* стабильность;
* справедливость;
* оптимальность.

## 25. Таблицы маршрутизации

В стеке TCP/IP маршрутизаторы и конечные узлы принимают решения о том, кому передавать пакет для его успешной доставки узлу назначения, на основании так называемых таблиц маршрутизации (routing tables). 

э

В таблице представлена таблица маршрутизации многомаршрутная, так как содержится два маршрута до сети 116.0.0.0. В случае построения одномаршрутной таблицы маршрутизации, необходимо указывать только один путь до сети 116.0.0.0 по наименьшему значению метрики.

В таблице определено несколько маршрутов с разными параметрами. Читать каждую такую запись в таблице маршрутизации нужно следующим образом:

*Чтобы доставить пакет в сеть с адресом из поля Сетевой адрес и маской из поля Маска сети, нужно с интерфейса с IP-адресом из поля Интерфейс послать пакет по IP-адресу из поля Адрес шлюза, а «стоимость» такой доставки будет равна числу из поля Метрика.*

Существуют различные алгоритмы построения таблиц для одношаговой маршрутизации. Их можно разделить на три класса:

* алгоритмы простой маршрутизации;
* алгоритмы фиксированной маршрутизации;
* алгоритмы адаптивной маршрутизации.

**Простая маршрутизация** – это способ маршрутизации, не изменяющийся при изменении топологии и состоянии сети передачи данных (СПД).

Простая маршрутизация обеспечивается различными алгоритмами, типичными из которых являются следующие:

**Случайная маршрутизация** – это передача сообщения из узла в любом случайно выбранном направлении, за исключением направлений по которым сообщение поступило узел.

**Лавинная маршрутизация** – это передача сообщения из узла во всех направлениях, кроме направления по которому сообщение поступило в узел. Такая маршрутизация гарантирует малое время доставки пакета, засчет ухудшения пропускной способности.

**Маршрутизация по предыдущему оп**ыту – каждый пакет имеет счетчик числа пройденных узлов, в каждом узле связи анализируется счетчик и запоминается тот маршрут, который соответствует минимальному значению счетчика. Такой алгоритм позволяет приспосабливаться к изменению топологии сети, но процесс адаптации протекает медленно и неэффективно.

В целом, простая маршрутизация не обеспечивает направленную передачу пакета и имеет низкую эффективности. Основным ее достоинством является обеспечение устойчивой работы сети при выходе из строя различных частей сети.

Алгоритм **фиксированной маршрутизации** применяется в сетях с простой топологией связей и основан на ручном составлении таблицы маршрутизации администратором сети. Алгоритм часто эффективно работает также для магистралей крупных сетей, так как сама магистраль может иметь простую структуру с очевидными наилучшими путями следования пакетов в подсети, присоединенные к магистрали, выделяют следующие алгоритмы:

**Однопутевая фиксированная маршрутизация** – это когда между двумя абонентами устанавливается единственный путь. Сеть с такой маршрутизацией неустойчива к отказам и перегрузкам.

**Многопутевая фиксированная маршрутизация** – может быть установлено несколько возможных путей и вводится правило выбора пути. Эффективность такой маршрутизации падает при увеличении нагрузки. При отказе какой-либо линии связи необходимо менять таблицу маршрутизации, для этого в каждом узле связи храниться несколько таблиц.

**Адаптивная маршрутизация** – это основной вид алгоритмов маршрутизации, применяющихся маршрутизаторами в современных сетях со сложной топологией. Адаптивная маршрутизация основана на том, что маршрутизаторы периодически обмениваются специальной топологической информацией об имеющихся в интерсети сетях, а также о связях между маршрутизаторами. Обычно учитывается не только топология связей, но и их пропускная способность и состояние.

Адаптивные алгоритмы позволяют всем маршрутизаторам собирать информацию о топологии связей в сети, оперативно отрабатывая все изменения конфигурации связей.

Алгоритмы имеют распределенный характер, который выражается в том, что в сети отсутствуют какие-либо выделенные маршрутизаторы, которые бы собирали и обобщали топологическую информацию: эта работа распределена между всеми маршрутизаторами, выделяют следующие алгоритмы:

**Локальная адаптивная маршрутизация** – каждый узел содержит информацию о состоянии линии связи, длины очереди и таблицу маршрутизации.

**Глобальная адаптивная маршр**утизация – основана на использовании информации, получаемой от соседних узлов. Для этого каждый узел содержит таблицу маршрутизации, в которой указано время прохождения сообщений. На основе информации, получаемой из соседних узлов, значение таблицы пересчитывается с учетом длины очереди в самом узле.

**Централизованная адаптивная маршрутизация** – существует некоторый центральный узел, который занимается сбором информации о состоянии сети. Этот центр формирует управляющие пакеты, содержащие таблицы маршрутизации и рассылает их в узлы связи.

**Гибридная адаптивная маршрутизац**ия – основана на использовании таблицы, периодически рассылаемой центром и на анализе длины очереди в самом узле.

**Показатели алгоритмов:**

В алгоритмах маршрутизации используется множество различных показателей. Сложные алгоритмы маршрутизации при выборе маршрута могут базироваться на множестве показателей, комбинируя их таким образом, что в результате получается один гибридный показатель. Ниже перечислены показатели, которые используются в алгоритмах маршрутизации:

* Длина маршрута.
* Надежность.
* Задержка.
* Ширина полосы пропускания.

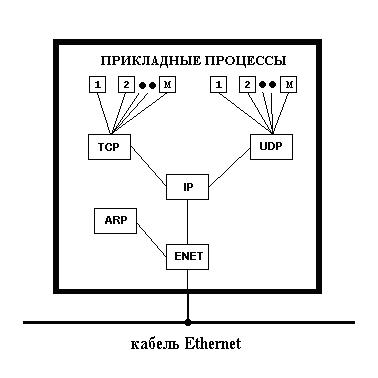
В зависимости от алгоритма маршрутизации **протоколы** делятся на два вида:

* дистанционно-векторные протоколы (основаны на алгоритме DVA — англ. *distance vector algorithm*);  
  RIP, IGRP, BGP, EIGRP, AODV
* протоколы состояния каналов связи (основаны на алгоритме LSA — англ. *link state algorithm*).  
  IS-IS, OSPF, NLSP, HSRP, CARP, OLSR, TBRPF

По области применения выделяют протоколы:

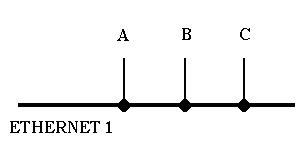
* для междоменной маршрутизации;  
  EGP, BGP, IDRP, IS-IS level 3
* для внутридоменной маршрутизации.  
  RIP, IS-IS level 1-2, OSPF IGRP, EIGRP

## 26. Прямая/косвенная маршрутизация



**Прямая маршрутизация.**

Рассмотрим IP-сеть, состоящую из 3 машин: A, B и C. Каждая машина имеет такой же стек протоколов TCP/IP как на рисунке выше. Каждый сетевой адаптер этих машин имеет свой Ethernet-адрес. Менеджер сети должен присвоить машинам уникальные IP-адреса.



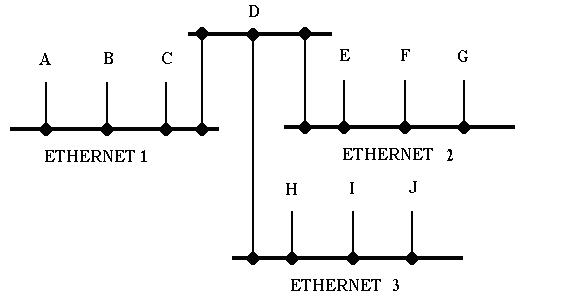
Простая IP-сеть, когда машина A посылает IP-пакет B, то заголовок IP-пакета содержит в поле отправителя IP-адрес узла A, а заголовок Ethernet-кадра содержит в поле отправителя Ethernet-адрес A. Кроме этого, IP-заголовок содержит в поле получателя IP-адрес узла B, а Ethernet-заголовок содержит в поле получателя Ethernet-адрес B.

|  |  |  |
| --- | --- | --- |
| **Адрес** | **Отправитель** | **Получатель** |
| IP-заголовок | A | B |
| Ethernet-заголовок | А | В |

В такой ситуации не возникает никаких проблем с передачей Ethernet - кадра адресату. Это прямая маршрутизация в пределах одной сети. Здесь двойная адресация является избыточной.

**Косвенная маршрутизация.**

Если в сети объединяется несколько локальных сетей, то используется косвенная маршрутизация. На рисунке представлена более реалистичная картина сети Интернет. В данном случае сеть состоит из трех сетей Ethernet, на базе которых работают три IP-сети, объединенные шлюзом D. Каждая IP-сеть включает четыре машины; каждая машина имеет свои собственные IP- и Ethernet- адреса.



Шлюз D соединяет все три сети и, следовательно, имеет три IP-адреса и три Ethernet адреса. Машина D имеет стек протоколов TCP/IP, похожий на тот, что показан на рисунке, но вместо двух модулей ARP и двух драйверов, он содержит три модуля ARP и три драйвера Ethernet. Обратим внимание на то, что машина D имеет только один модуль IP.   
  
Когда машина A взаимодействует с машинами, включенными в другую IP-сеть, то взаимодействие уже не будет прямым. Машина A должна использовать шлюз D для ретрансляции IP-пакетов в другую IP-сеть. Такое взаимодействие называется "косвенным".   
  
Если машина A посылает машине E IP-пакет, то IP-адрес и Ethernet- адрес отправителя соответствуют адресам A. IP-адрес места назначения является адресом E, но поскольку модуль IP в A посылает IP-пакет через D, Ethernet-адрес места назначения является адресом D.

|  |  |  |
| --- | --- | --- |
| Адрес | Oтправитель | Получатель |
| IP-заголовок | A | E |
| Ethernet-заголовок | А | D |

Адреса в Ethernet-кадре, содержащем IP-пакет от A к E (до шлюза D)

Модуль IP в машине D получает IP-пакет и проверяет IP-адрес места назначения. Определив, что это не его IP-адрес, шлюз D посылает этот IP-пакет прямо к E.

|  |  |  |
| --- | --- | --- |
| Адрес | Отправитель | Получатель |
| IP-заголовок | A | E |
| Ethernet-заголовок | А | E |

Адреса в Ethernet-кадре, содержащем IP-пакет от A к E (после шлюза D)

# Практическая часть

## 1.Socket

**socket()** – создать объект сокет

Эта функция используется для создания сокета. Прототип:

int socket (int domain, int type, int protocol);

Первый параметр - домен - накладывает определенные ограничения на формат используемых процессом адресов и их интерпретацию. При работе с одной ЭВМ используется UNIX-домен, где адреса интерпретируются как имена файлов в UNIX.

Domain указывает семейство протоколов создаваемого сокета:

* **AF\_INET** для сетевого протокола IPv4
* **AF\_INET6** для IPv6
* **AF\_UNIX** для локальных сокетов (используя файл)

Второй параметр определяет тип канала связи с сокетом, который должен быть использован.

Существует несколько типов каналов связи с сокетом, доступных при межпроцессном взаимодействии в UNIX, но обычно используются следующие два:

* SOCK\_STREAM - при этом типе связи поступающим в канал байтам информации гарантируется "доставка" в порядке их поступления; пока непрерывный поток байтов не прекратится, никакие другие данные приниматься каналом не будут (аналогом такой связи является pipe-механизм);
* SOCK\_DGRAM - этот тип связи используется для посылки отдельных пакетов информации, называемых datagrams; при этом не гарантируется, что пакеты будут доставлены на место назначения в порядке поступления, а в действительности не гарантируется, что они все вообще будут доставлены (пример такого типа связи - обычная почтовая связь).

Третий параметр позволяет программисту выбрать нужный протокол для канала связи. Если этот параметр равен нулю, ОС выберет нужный протокол автоматически.

Функция socket возвращает целое положительное число - номер сокет-дескриптора (который можно использовать, например, в функциях read и write аналогично файловому дескриптору). Если же сокет по каким-либо причинам не был создан (например, очень много открытых файлов), возвращается -1, а в переменную "errno" записывается причина неудачи.

Константы, используемые в качестве аргументов при вызове socket, определены во включаемых файлах sys/socket.h и sys/types.h.

## 2.Bind

**bind()** - привязка сокета к адресу на машине

Эта функция используется сервером для присваивания сокету имени. До выполнения функции bind (т.е. присваивания какого-либо имени, вид которого зависит от адресного домена) сокет недоступен программам-клиентам.

Прототип:

int bind(int s, char \* name, int namelen);

Первый параметр - сокет-дескриптор, который данная функция именует. Второй параметр - указатель на структуру имени сокета, тип которой зависит от домена. Для UNIX-домена этот тип называется sockaddr\_un, он определен во включаемом файле sys/un.h и выглядит таким образом:

struct sockaddr\_un

{

short sun\_family;

char sun\_path[108];

};

В качестве первого элемента структуры, обозначающего класс адресов, мы будем использовать константу AF\_UNIX, второй элемент - имя файла, который будет соответствовать используемому сокету.

Файл c именем, указанным в sun\_path, действительно создается, поэтому после окончания работы с данным сокетом надо выполнить функцию unlink, в противном случае другие программы, которые попытаются использовать данное имя, получат сообщение об ошибке.

recvfrom() – получение UDP датаграммы

sendto() – отправка UDP датаграммы

htons() – перевод числа из локального формата в сетевой

inet\_addr() - преобразует обычный вид IP-адреса *cp* (из номеров и точек) в двоичный код в сетевом порядке расположения байтов.

## 3.Connect

#include <sys/types.h>

#include <sys/socket.h>

int connect(int sockfd, struct sockaddr \*serv\_addr, int addrlen);

Используется для инициализации соединения.

Системный вызов connect () соединяет сокет, указанный дескриптором файла sockfd, с адресом, указанным в addr. Адрес и порт сервера указаны в addr.

Здесь **sockfd** - сокет, который будет использоваться для обмена данными с сервером, **serv\_addr** содержит указатель на структуру с адресом сервера, а **addrlen** - длину этой структуры. Обычно сокет не требуется предварительно привязывать к локальному адресу, так как функция **connect** сделает это за вас, подобрав подходящий свободный порт. Вы можете принудительно назначить клиентскому сокету некоторый номер порта, используя **bind** перед вызовом **connect**. Делать это следует в случае, когда сервер соединяется с только с клиентами, использующими определённый порт (примерами таких серверов являются rlogind и rshd). В остальных случаях проще и надёжнее предоставить системе выбрать порт за вас.

## 4.Sendto

**sendto()** – отправка UDP датаграммы (отправляет сообщения в сокет)

**ssize\_t sendto(int** *s***, const void \****msg***, size\_t** *len***,** **int** *flags***, const struct sockaddr \****to***, socklen\_t** *tolen***);**

**sendto** используется для пересылки сообщений в другой сокет. **sendto** можно использовать в любое время.

Этот системный вызов возвращает количество отправленных символов или -1, если произошла ошибка.

Адрес получателя задается параметром *to* длиной *tolen*. Длина сообщения задается параметром *len*. Если сообщение слишком длинное, чтобы быть отосланным протоколом нижнего уровня, возвращается ошибка **EMSGSIZE**, а сообщение не отсылается.

Параметр *flags* является битовой маской и может содержать такие флаги:

**MSG\_OOB**

Посылает *внепотоковые* данные, если сокет это поддерживает (как, например, сокеты типа **SOCK\_STREAM**); протокол более низкого уровня также должен поддерживать *внепотоковые* данные.

**MSG\_DONTROUTE**

Не использовать маршрутизацию при отправке пакета, а посылать его только на хосты в локальной сети. Обычно это используется в диагностических программах и программах маршрутизации. Этот флаг определен только для маршрутизируемых семейств протоколов; пакетные сокеты не используют маршрутизацию.

**MSG\_DONTWAIT**

Включает режим non-blocking; если операция должна была заблокировать программу, возвращается **EAGAIN** (этот режим также можно задать с помощью опции **O\_NONBLOCK**, команды **F\_SETFL** и функции [**fcntl**](https://www.opennet.ru/cgi-bin/opennet/man.cgi?topic=fcntl&category=2)(2)).

**MSG\_NOSIGNAL**

Требует не посылать сигнал **SIGPIPE**, если при работе с ориентированным на поток сокетом другая сторона обрывает соединение. Код ошибки **EPIPE** возвращается в любом случае.

**MSG\_CONFIRM** (только в Linux 2.3+)

Сообщает (на уровне связи), что процесс пересылки произошел: вы получаете успешный ответ с другой стороны. Если уровень связи не получает его, он регулярно перепроверяет сеть (например, посредством однонаправленной передачи ARP). Это работает с сокетами **SOCK\_DGRAM** и **SOCK\_RAW** и в настоящее время реализовано для IPv4 и IPv6.

htons() – перевод числа из локального формата в сетевой

inet\_addr() - преобразует обычный вид IP-адреса *cp* (из номеров и точек) в двоичный код в сетевом порядке расположения байтов.

## 5.Recvfrom

**recvfrom()** – получение UDP датаграммы (получить сообщение из сокета)

**int recvfrom(int** *s***, void \****buf***, size\_t** *len***,** **int** *flags***, struct sockaddr \****from***, socklen\_t \****fromlen***);**

Системный вызов **recvfrom** используется для получения сообщений из сокета, и может использоваться для получения данных, независимо от того, является ли сокет ориентированным на соединения или нет.

Этот системный вызов возвращает количество принятых байт или -1, если произошла ошибка.

Если параметр *from* не равен **NULL**, а сокет не является ориентированным на соединения, то адрес отправителя в сообщении не заполняется. Аргумент *fromlen* передается по ссылке, в начале инициализируется размером буфера, связанного с *from*, а при возврате из функции содержит действительный размер адреса.

Функция возвращает длину сообщения при успешном завершении. Если сообщение слишком длинное и не поместилось в предоставленный буфер, лишние байты могут быть отброшены, в зависимости от типа сокета, на котором принимаются сообщения.

Если на сокете не доступно ни одного сообщения, то обсуждаемые функции ожидают их прибытия, если сокет не помечен как неблокирующий, в противном случае возвращается значение -1, а внешняя переменная *errno* устанавливается в значение **EAGAIN**. Все эти функции обычно возвращают уже доступные данные вплоть до запрошенного объема, и не ждут, пока появятся данные полной запрошенной длины.

Аргумент *flags* системного вызова **recv** формируется с помощью объединения логической операцией *ИЛИ* одного или более нижеследующих значений:

**MSG\_OOB**

Этот флаг запрашивает прием внепотоковых данных, которые в противном случае не были бы получены в обычном потоке данных. Некоторые протоколы помещают данные повышенной срочности в начало обычной очереди данных, и поэтому этот флаг не может использоваться с такими протоколами.

**MSG\_PEEK**

Этот флаг заставляет выбрать данные из начала очереди, но не удалять их оттуда. Таким образом, последующий вызов функции вернет те же самые данные.

**MSG\_WAITALL**

Этот флаг просит подождать, пока не придет полное запрошенное количество данных. Однако, этот вызов все равно может вернуть меньше данных, чем было запрошено, если был пойман сигнал, произошла ошибка или разрыв соединения, или если начали поступать данные другого типа, не того, который был сначала.

**MSG\_TRUNC**

Возвращает реальную длину пакета, даже если она была больше, чем предоставленный буфер. Этот флаг можно использовать только с пакетными протоколами.

**MSG\_ERRQUEUE**

Получить пакет из очереди ошибок.

**MSG\_NOSIGNAL**

Этот флаг отключает возникновение сигнала **SIGPIPE** на потоковых сокетах, если другая сторона вдруг исчезает.

**MSG\_ERRQUEUE**

Указание этого флага позволяет получить из очереди ошибок сокета накопившиеся ошибки. Каждая ошибка передается во вспомогательном сообщении, чей тип зависит от протокола (для IPv4 этим типом является **IP\_RECVERR**). Пользователь должен предоставить буфер достаточной длины, где приведена дополнительная информация.

## 6.Accept

#include <sys/socket.h>

int accept(int sockfd, void \*addr, int \*addrlen);

Функция **accept** создаёт для общения с клиентом *новый* сокет и возвращает его дескриптор. Параметр **sockfd** задаёт слушающий сокет. После вызова он остаётся в слушающем состоянии и может принимать другие соединения. В структуру, на которую ссылается **addr**, записывается адрес сокета клиента, который установил соединение с сервером. В переменную, адресуемую указателем **addrlen**, изначально записывается размер структуры; функция **accept** записывает туда длину, которая реально была использована. Если вас не интересует адрес клиента, вы можете просто передать NULL в качестве второго и третьего параметров.

Обратите внимание, что полученный от **accept** новый сокет связан с тем же самым адресом, что и слушающий сокет. Сначала это может показаться странным. Но дело в том, что адрес TCP-сокета не обязан быть уникальным в Internet-домене. Уникальными должны быть только *соединения*, для идентификации которых используются *два* адреса сокетов, между которыми происходит обмен данными.

## 7.Listen

int listen(int sockfd, int backlog);

Функция listen используется сервером, чтобы информировать ОС, что он ожидает ("слушает") запросы связи на данном сокете. Без такой функции всякое требование связи с этим сокетом будет отвергнуто.

Первый аргумент - сокет для прослушивания, второй аргумент (backlog) - целое положительное число, определяющее, как много запросов связи может быть принято на сокет одновременно. В большинстве систем это значение должно быть не больше пяти. Заметим, что это число не имеет отношения к числу соединений, которое может поддерживаться сервером. Аргумент backlog имеет отношение только к числу запросов на соединение, которые приходят одновременно. Число установленных соединений может превышать это число

## 8.Shutdown

int shutdown(int sockfd, int how);

Параметр **how** может принимать одно из следующих значений:

* 0 - запретить чтение из сокета
* 1 - запретить запись в сокет
* 2 - запретить и то и другое

Хотя после вызова **shutdown** с параметром **how**, равным 2, вы больше не сможете использовать сокет для обмена данными, вам всё равно потребуется вызвать **close**, чтобы освободить связанные с ним системные ресурсы.

## 9.Close

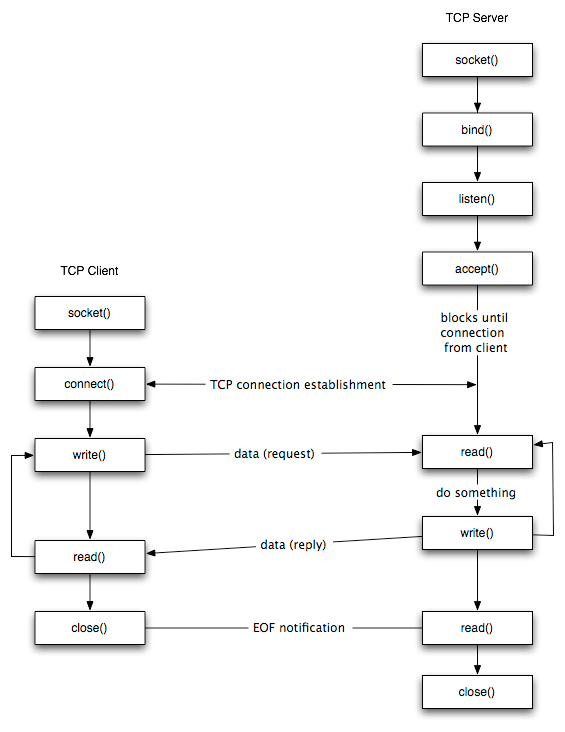
Эта функция закрывает сокет и разрывает все соединения с этим сокетом. В отличие от функции shutdown функция close.может дожидаться окончания всех операций с сокетом, обеспечивая "нормальное", а не аварийное закрытие соединений.

Прототип:

int close (int s);

Аргумент функции - закрываемый сокет-дескриптор.

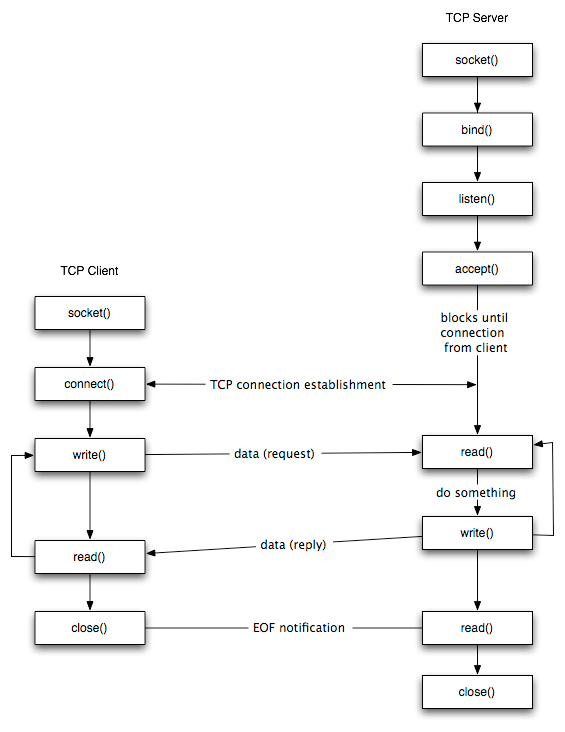
## 10.Алгоритм работы сервера



Создание простейшего TCP-сервера состоит из следующих шагов:

* Создание TCP-сокетов вызовом функции socket().
* Привязывание сокета к прослушиваемому порту вызовом функции bind(). Перед вызовом bind() программист должен объявить структуру sockaddr\_in, очистить её (при помощи memset()), затем sin\_family (PF\_INET или PF\_INET6) и заполнить поля sin\_port (прослушиваемый порт, указать в виде [последовательности байтов](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D1%80%D1%8F%D0%B4%D0%BE%D0%BA_%D0%B1%D0%B0%D0%B9%D1%82%D0%BE%D0%B2)). Преобразование short int в порядок байтов может быть выполнено при помощи вызова функции htons() (сокращение от «от хоста в сеть»).
* Подготовка сокета к прослушиванию на предмет соединений (создание прослушиваемого сокета) при помощи вызова listen().
* Принятие входящих соединений через вызов accept(). Это блокирует сокет до получения входящего соединения, после чего возвращает дескриптор сокета для принятого соединения. Первоначальный дескриптор остаётся прослушиваемым дескриптором, а accept() может быть вызван вновь для этого сокета в любое время (пока он открыт).
* Соединение с удаленным хостом, которое может быть создано при помощи send() и recv() или write() и read().
* Итоговое закрытие каждого открытого сокета, который больше не нужен, происходит при помощи close(). Необходимо отметить, что если были любые вызовы fork(), то каждый процесс должен закрыть известные ему сокеты (ядро отслеживает количество процессов, имеющих открытый дескриптор), а кроме того, два процесса не должны использовать один и тот же сокет в одно время.

## 11.Алгоритм работы клиента



Создание TCP-клиента происходит следующим образом:

* Создание TCP-сокета вызовом socket().
* Соединение с сервером при помощи connect(), передача структуры sockaddr\_in с sin\_family с указанными PF\_INET или PF\_INET6, sin\_port для указания порта прослушивания (в байтовом порядке), и sin\_addr для указания IPv4 или IPv6 адреса прослушиваемого сервера (также в байтовом порядке).
* Взаимодействие с сервером при помощи send() и recv() или write() и read().
* Завершение соединения и сброс информации при вызове close(). Аналогично, если были какие-либо вызовы fork(), каждый процесс должен закрыть (close()) сокет.

## 12.Блокирующие/неблокирующие сокеты

Блокирующий сокет не возвращает контроль, пока не отошлёт (или пока не получит) все данные, указанные для операции. Это верно лишь для Linux-систем. В других системах, например во FreeBSD, вполне естественно для блокирующего сокета посылать не все данные (но можно поставить в send() или recv() флаг MSG\_WAITALL). Приложение должно проверять возвращаемое значение для отслеживания того, сколько байт было послано/получено и, соответственно, перепосылать необработанную на данный момент информацию[2]. Это может привести к проблемам, если сокет продолжает «слушать»: программа может повиснуть из-за того, что сокет ждет данных, которые могут никогда не прибыть.

Сокет обычно указывается блокирующим или неблокирующим при помощи функций fcntl() или ioctl().

## 13.Алгоритм работы с неблокирующими сокетами

Обслуживание каждого соединения в отдельном потоке при помощи *блокирующих* сокетов крайне просто для программирования, однако этот подход к созданию асинхронных сетевых приложений имеет принципиальные ограничения.

* Как известно, использования потоков влечет накладные расходы ОС на их переключение. Де-факто они становятся критическими при сотнях активных потоках на одно ядро ЦП или при тысячах потоков на всю систему. Современные нагрузки предполагают десятки тысяч активных соединений одновременно или миллионы малоактивных, поэтому для них порождать поток на каждое соединение неприемлемо.
* Блокирующие сокеты неудобны тем, что позволяют либо принимать данные из сокета, либо отправлять их. Это приемлемо, если клиент и сервер обмениваются запросами в строгой очередности. Однако, если любая сторона может как принимать, так и отправлять данные по своему усмотрению (например, мессенджер), требуется или два соединения (расход сокетов), или два потока (накладные расходы, описанные выше).

Необходим инструмент для работы из одного потока с несколькими сокетами сразу или с несколькими операциями над одним сокетом сразу. Таковым является **неблокирующий** режим работы сокетов.

Стандартные операции над сокетом, переведенным в неблокирующий режим, никогда не приводят к блокировке. Вместо этого они завершаются со специальным кодом ошибки (EWOULDBLOCK или EAGAIN), который означает, что операция не может быть выполнена в данный момент (не пришли данные в случае recv(), нет места в буфере отправки в случае send() и прочее). Таким образом, программа не лишается управления при временной невозможности выполнить операцию, а лишь информируется об этом, и может повторить ее успешно позже.

Как дождаться момента, когда потенциально блокирующую функцию можно будет вызвать успешно (говорят: когда сокет будет готов к работе)? Постоянно вызывать ее, пока результат — EWSAWOULDBLOCK, неэффективно с точки зрения как производительности — данные могут прибыть для одного сокета, пока совершается опрос других, — так и энергоэффективности - процесс все время работает, в отличие от состояния блокировки. Вместо этого применяется вызов специальной функции, называемой *мультиплексором.*

*Мультиплексор* в программировании сокетов — это специальная операция, которая заключается в ожидании событий, которые могут произойти на некоторых сокетах из группы. Например, мультиплексор одновременно ожидает, когда одному из сокетов-передатчиков поступит запрос клиента или сокету-слушателю придет новый запрос на подключение. Мультиплексор — единственная блокирующая операция, связанная с неблокирующими сокетами.

Два стандартных мультиплексора — select() и poll().В современных приложениях рекомендуется использовать poll().

Эта функция принимает набор сокетов и блокирует выполнение, пока хотя бы один из них не будет готов к работе, а по ее завершении можно узнать, какой именно (их может быть несколько). При этом для каждого из сокетов можно указать, что для него считается готовностью к работе: возможность приема, отправки или любая из них. Сокет-слушатель также может быть неблокирующим, для него актуальна возможность приема, но для вызова не recv(), а accept().

Таким образом, работа асинхронного приложения на основе неблокирующих сокетов строится в таком цикле:

1. Для каждого сокета выбираются события, которые должны наступить, чтобы считать сокет готовым к работе, и он добавляется в набор сокетов.
2. Вызывается функция-мультиплексор, выполнение блокируется.
3. Для каждого сокета из набора проверяется, какие события для него наступили (готовность к приему, готовность к отправке, ошибка).
4. Для сокетов, готовых к работе, соответствующая функция (например, recv() при готовности к приему) вызывается, пока не завершится с ошибкой WSAEWOULDBLOCK.

Недостаток неблокирующего режима — сложность программирования. При использовании потоков данные, связанные с каждым клиентом, хранятся в локальных переменных. При работе со всеми клиентами из одного потока требуется самостоятельно вести массив данных всех клиентов. Поскольку после любого вызова потенциально блокирующей функции (например, recv()) может быть нужно вернуться к мультиплексору, требуется также для каждого клиента хранить состояние, например, сколько байт осталось принять. При использовании потоков состояние определялось местом программы.

Для клиентов неблокирующий режим обычно менее актуален, но также возможен. Функция connect() в нем возвращает EWOULDBLOCK при первом вызове, затем нужно дожидаться возможности записи в сокет, что означает успешное завершение соединения. Неблокирующий режим доступен и для дейтаграммных сокетов, и соответственно, recvfrom() и sendto().

Неблокирующий режим переключается для каждого сокета отдельно. Даже если сокет-слушатель работает в неблокирующем режиме, сокеты-передатчики, порождаемые accept(), начинают работать в блокирующем режиме. Для переключения используется функция ioctl().