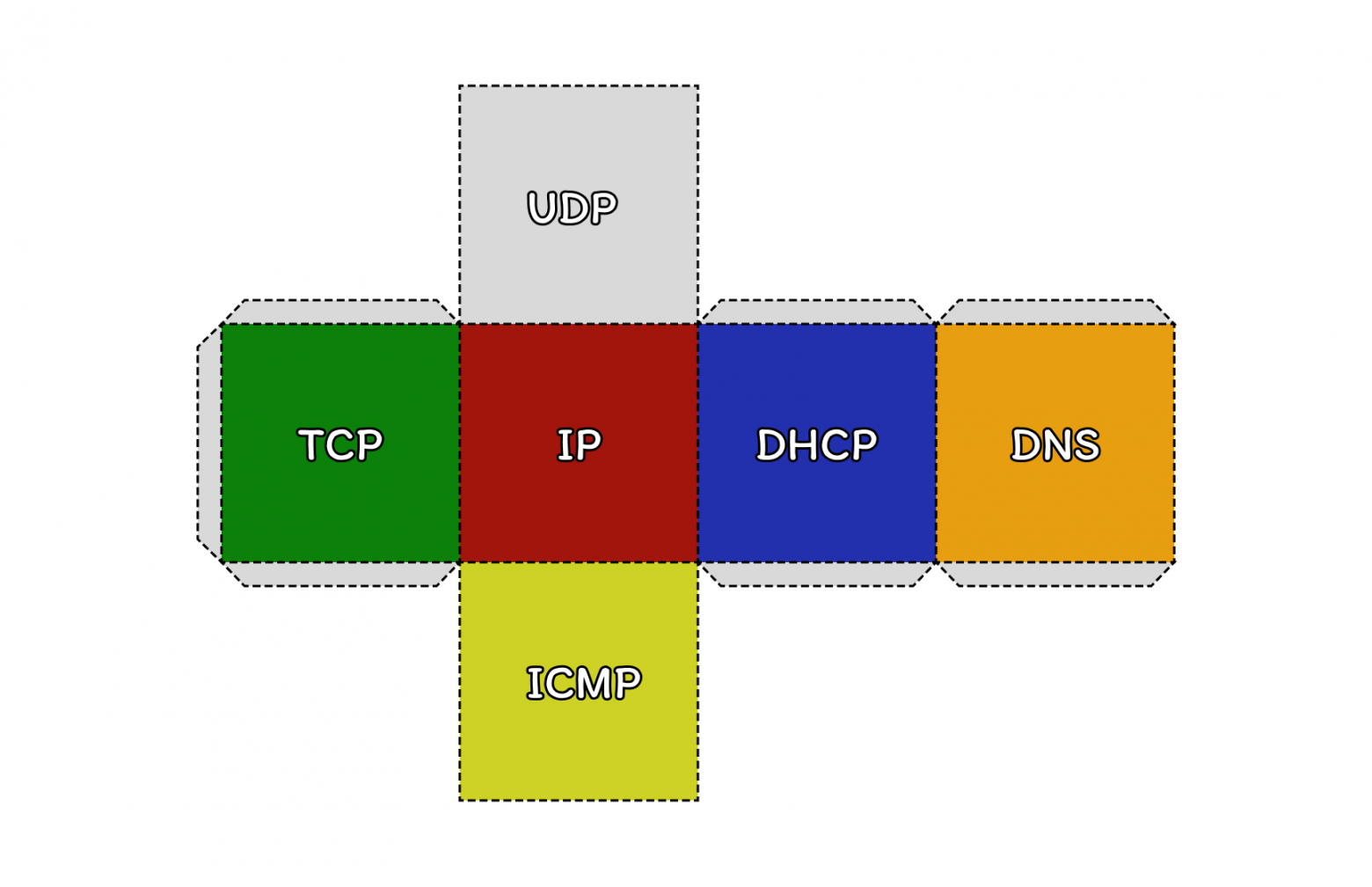
# Протоколы семейства TCP/IP. Теория и практика



В сети можно найти огромное количество материала о том, как функционируют сети на базе стека протоколов TCP/IP, а также как писать компьютерные программы с сетевыми возможностями. При рассмотрении компьютерных сетей часто углубляются в описание физических основ и структур данных, передаваемых по сети, а при рассмотрении сетевого программирования основное внимание уделяют интернет-сокетам.  
  
Но при изучении и исследовании хочется большего, например, поэкспериментировать с пакетами сетевых протоколов. Многие сетевые протоколы реализованы в ядре операционной системы, и что-либо изменить может оказаться сложной задачей, так как это требует навыков в написании драйверов для операционной системы. Но использование специализированных библиотек позволяет работать с протоколами на низком уровне из пространства пользователя.

## Кратко о компьютерных сетях

### ▍ Компьютерная сеть

Компьютерная сеть — это множество вычислительных устройств, взаимодействующих между собой и совместно использующих ресурсы. Понятие сеть близко по смыслу к понятию графа. Cеть также состоит из множества узлов (nodes) и множества звеньев (links). Отличие сети от графа в том, что узлы являются чем-то осмысленным, в данном случае — это вычислительные устройства, а звенья представляют связи этих устройств. В русскоязычной литературе компьютерную сеть иногда называют вычислительной сетью.

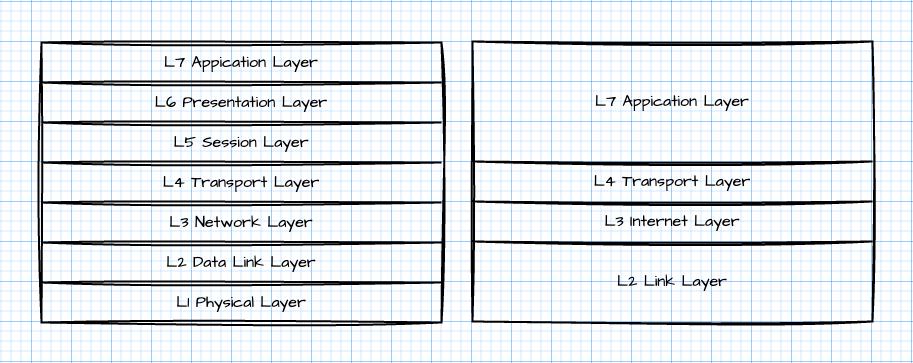
### ▍ Локальные и глобальные компьютерные сети

В зависимости от охвата территории компьютерные сети бывают:

1. Персональные — Personal Area Network (PAN).
2. Локальные — Local Area Network (LAN).
3. Городские — Metropolitan Area Network (MAN).
4. Глобальные — Wide Area Network (WAN).

Различные датчики, подключённые к смартфону, образуют сеть PAN. Компьютерная сеть из устройств, подключённых к вашему домашнему роутеру, является LAN-сетью, сеть из абонентов провайдера в городе — это MAN-сеть, а весь интернет, который вам предоставляет провайдер — WAN-сеть.

### ▍ Сетевые модели

Под сетевой моделью понимаются концептуальные основы, которые стандартизируют сетевое взаимодействие. Это основные термины, а также назначение и функции сетевых компонентов. Сетевая модель разделяет сетевые компоненты и их функции на уровни (слои (layers)). Каждый слой сетевой модели имеет определённое назначение и функции.  
  
Сейчас наиболее распространены две сетевых модели. Это семиуровневая OSI-модель и четырехуровневая TCP/IP-модель.  
  
Приведу схему моделей и как они соотносятся друг с другом.  
  
  
  
В большинстве случаев вы будете иметь дело с сетевой моделью TCP/IP, однако так исторически сложилось, что номера слоёв используются из сетевой модели OSI. Например, когда встречается термин Layer 2 или L2, подразумевается 1-й уровень (канальный уровень) из модели TCP/IP.  
  
Разбивка по слоям позволяет технологиям каждого слоя эволюционировать независимо от всех остальных. Например, благодаря повсеместному внедрению оптоволокна скорость интернета возросла в десятки или даже сотни раз, а интернет как базировался на протоколе IP, так и продолжает базироваться.

### ▍ Модель TCP/IP

Большинство действующих стандартов интернета и протоколов TCP/IP регламентируются документами Request For Comments (RFC). Учебники по компьютерным сетям ставят целью объяснить модель TCP/IP, но за точной трактовкой понятий лучше обращаться к RFC.  
  
Детально сетевая модель TCP/IP рассмотрена в [RFC 1122 (Requirements for Internet Hosts — Communication Layers )](https://datatracker.ietf.org/doc/html/rfc1122) и [RFC-1123 (Requirements for Internet Hosts — Application and Support)](https://datatracker.ietf.org/doc/html/rfc1123). Модель объясняется и расширяется другими RFC, но для понимания основ, я думаю, достаточно этих двух.  
  
Выделим базовые понятия из модели TCP/IP:

* хост (host);
* сообщение;
* IP-датаграмма;
* пакет;
* фрейм;
* IP-адрес;
* MAC-адрес;
* TCP-сегмент;
* UDP-датаграмма;
* MTU.

### ▍ Как работает сеть, построенная на базе TCP/IP

IP-сеть представляет собой множество связанных между собой хостов. Хосты связаны непосредственно или косвенно при помощи ретранслирующих устройств (маршрутизаторов и коммутаторов).  
  
Для приёма сообщений из сети и отправку их в сеть хост использует интерфейсы. Физический интерфейс отправляет и принимает фреймы, а логический интерфейс отправляет и принимает IP-пакеты. Физический интерфейс идентифицируется MAC-адресом, логический интерфейс — IP-адресом.  
  
Передаваемое сообщение представляет собой UDP-датаграмму или TCP-сегмент. Сообщение содержит заголовок и полезные данные. Чтобы передать сообщение внутри IP-сети оно помещается в IP-датаграмму. Конкретный физический интерфейс позволяет передавать данные порциями, которые имеют определённый максимально допустимый размер (MTU). Если размер IP-датаграммы превышает MTU, выполняется её фрагментация и создаётся несколько IP-пакетов, иначе создаётся только один IP-пакет для всей IP-датаграммы.  
  
IP-пакет в соответствии с таблицей маршрутизации хоста передаётся на выбранный логический интерфейс.  
  
Логический интерфейс сам непосредственно не может передать IP-пакет, он использует физический интерфейс. Физический интерфейс передаёт данные фреймами. Фрейм имеет заголовок и полезные данные (payload). В заголовке фрейма указывается MAC-адрес получателя, MAC-адрес отправителя и какому протоколу принадлежат данные в payload (Ethertype). Адрес отправителя известен, это МАС-адрес интерфейса отправляющего хоста. Для протокола IPv4 Ethertype=0x0800.  
  
Адрес физического интерфейса определяется путём посылки ARP-сообщения в широковещательный домен. ARP-сообщение инкапсулируется во фрейм, у которого EtherType = 0x0806 (ARP). В сообщении указывается MAC-адрес отправителя, широковещательный MAC-адрес получателя и интересующий IP-адрес. Хост с физическим интерфейсом, которому назначен этот IP-адрес в ответном сообщении, указывает MAC-адрес этого физического интерфейса. Чтобы не отсылать ARP-сообщение каждый раз, соответствие IP-адреса MAC-адресу сохраняется в кеше хоста.  
  
После передачи фрейма на другой сетевой интерфейс из него извлекается содержимое IP-пакета, и, если IP-адрес логического интерфейса хоста соответствует IP-адресу получателя, он собирается в IP-датаграмму. Из IP-датаграммы извлекается TCP-сегмент или UDP-датаграмма. Из них извлекаются сами данные и передаются процессу операционной системы, который уже понимает, что с ними делать дальше.  
  
Иначе IP-пакет или отвергается или пересылается далее в соответствии с таблицей маршрутизации хоста. При отсылке он опять передаётся на логический интерфейс. Там упаковывается во фрейм и отсылается.

### ▍ Адресация

Адресация позволяет указать источника и получателя данных. Для слоя L2 получатель и отправитель идентифицируются MAC-адресами, для L3 — IP-адресами, для L4 — портами.  
  
Что касается MAC-адресации, вам достаточно знать, что в большинстве случаев физический интерфейс имеет уникальный MAC-адрес, состоящий из 6 байт.  
  
Что касается IP-адресации, тут всё немного интереснее. Начнём с того, что всего возможно 2^32 IP-адресов, но количество допустимых IP-адресов хостов меньше, а глобальных (IP-адресов, видимых в интернете) ещё меньше.

Интернет проектировался как множество соединённых компьютерных сетей, хосты которых взаимодействуют между собой.  
  
Для идентификации сетей используется то же адресное пространство, что и для идентификации интерфейсов хостов. Как это реализуется?  
  
Каждый IP-адрес — это последовательность из 32 бит. Первые n-бит в IP-адресе несут информацию о том, к какой сети принадлежит IP-адрес, оставшиеся биты — это уникальный адрес внутри этой сети. Однако адреса, у которых все биты 0 или все биты 1 имеют особое значение. Если все биты 0 — это сеть, если 1 — это широковещательный адрес.  
  
Сколько первых бит в IP-адресе несут информацию о сети определяется маской сети. Если выполнить побитовое “И” IP-адреса с маской сети, то получится идентификатор сети. Если выполнить побитовое “И” с инвертированной маской сети, то мы получим уникальный адрес внутри сети. Чтобы можно было проще представить информацию об IP-адресе, и какая его часть используется для идентификации сети, используется CIDR-нотация.  
  
Например, запись 192.168.0.0/24 означает: сеть имеет идентификатор 192.168.0.0, 24 первых бита используются для идентификации сети. Для кодирования хостов используется 8 последних бит. Максимальное количество хостов в сети — 254 (0 — сеть, 255 — широковещательный адрес).  
Но это ещё не всё. Не все сети могут быть видимыми в сети и некоторые из них используются для специальных целей.  
  
Как мы видим, количество IP-адресов хостов в интернете не так уж и велико, поэтому выполняется переход на IPv6-адресацию, где количество IP-адресов гораздо больше.

### ▍ Сегмент сети, подсеть, широковещательный домен, виды адресов

Когда рассматривается сеть, нужно всегда иметь представление на каком уровне это выполняется. Если сеть рассматривается на канальном уровне, она состоит из устройств, имеющих MAC-адреса. Если она рассматривается на сетевом уровне, она состоит из хостов, имеющих IP-адреса.  
  
С целью упрощения управления, сети на канальном уровне разделяются на сегменты, а на сетевом уровне на подсети. И сегмент сети и подсеть являются широковещательным доменом. Иногда подсети называют также сегментами, однако нужно понимать, что подразумевается сегмент сети на уровне L3.  
  
Сетевое устройство или хост могут иметь следующие типы адресов:

* индивидуальный адрес (unicast address);
* широковещательный адрес (broadcast address);
* групповой адрес (multicast address).

Индивидуальный адрес — это уникальный адрес в сегменте сети, или локальной (глобальной) сети.  
  
Широковещательный адрес — общий для всех сетевых устройств, имеющих MAC-адрес или для всех хостов подсети. Сообщения, посылаемые на широковещательный адрес, будут получены всеми узлами.  
  
Можно сделать так, чтобы сообщения отсылались только тем узлам, которые в них заинтересованы. Для этого используются групповые адреса. Но это уже отдельная тема, требующая отдельного и более широкого объяснения.

### ▍ Маршрутизация в IP-сетях

Маршрутизация обычному пользователю сети выглядит следующим образом. Если необходимо отправить IP-пакет по определённому адресу, то каждый раз просматривается таблица маршрутизации хоста, и на основании её определяется, нужно его отправлять на хост внутри сети, которой принадлежит отправляющий хост, или нужно его перенаправить на особый хост (маршрутизатор, шлюз), который решит, что делать с ним дальше. В локальных домашних сетях часто таким шлюзом является WiFi-роутер.  
  
WiFi-роутер обладает рядом функций и фактически состоит из нескольких устройств. Одной из важных функций WiFi-роутера является функция трансляции сетевых устройств (Network Address Translation — NAT), позволяющая устройствам из локальной сети получить доступ в интернет. Доступ этот немного ограниченный, но большинству пользователей этого достаточно.  
  
Суть работы NAT в том, что хост, который находится за NAT, виден для глобальной сети как хост, у которого IP-адрес такой же, как и внешний адрес NAT, что позволяет более экономно расходовать глобальные IP-адреса. Подробное описание работы NAT потребует отдельной статьи, а то и книги.

### ▍ Расчёт контрольной суммы

При передаче данных по сети могут происходить различные ситуации, когда принятые данные могут отличаться от тех, которые были переданы. Чтобы определить такие случаи, с данными передаётся контрольная сумма, вычислив которую на принимающей стороне и сравнив с принятой, можно дать ответ, передались ли данные верно, или где-то произошла ошибка передачи данных. Как правило, сообщения с неверной контрольной суммой отвергаются, и сообщение считается потерявшимся. На различных уровнях стека используются различные варианты подсчёта контрольной суммы. Например, для Ethernet-фрейма используется CRC-32. Для вычисления контрольной суммы IP-заголовка, ICMP-сообщения, UDP-датаграммы и TCP-сегмента используется расчёт контрольной суммы с использованием обратного кода (one’s complement checksum). Чтобы не тратить процессорное время на вычисление контрольных сумм для передаваемых Ethernet-фреймов или IP-пакетов, используется механизм разгрузки (offloading), который заключается в том, что контрольные суммы вычисляются сетевым адаптером.

### ▍ Порядок байтов и битов

Данные в компьютере хранятся и обрабатываются в виде двоичного кода. Двоичный код подразумевает, что информация кодируется при помощи двух значений — 0 и 1. Один 0 или одна 1 содержат минимальное количество информации, которое называется бит. Процессор обычно оперирует не отдельными битами, а порциями по 8, 16, 32, 64 и т. д. Минимальная такая порция это 8 бит. 8 бит называются байтом. Можно сказать, что информация в памяти хранится в виде последовательности байт. Но байт может принимать только 256 значений, что явно недостаточно для большинства математических операций, поэтому байты группируются в слова (2 байта), двойное слово (4 байта) и т. д. Но хранить эти группы байт в памяти можно по-разному. Возьмём двойное слово 4 байта 0x12345678, оно состоит из 4 байт со значениями 0x12, 0x34, 0x56, 0x78. В памяти их можно разместить в следующем порядке 0x12, 0x34, 0x56, 0x78 или 0x78, 0x56, 0x34, 0x12. Как они будут размещены в памяти зависит от архитектуры процессора. Например в архитектуре x86 это будет порядок, при котором байт с менее значимыми битами будет храниться раньше байта с более значимыми битами.  
  
Так исторически сложилось, что данные в пакетах, передаваемых в IP-сетях, имеют сетевой порядок байтов, который подразумевает, что байт, содержащий старшие биты, хранится первый.  
Что касается порядка битов, то если не углубляться в вопросы кодирования передаваемых данных на уровне сигналов, порядок не важен, так как сетевой адаптер сделает всё без вмешательства программиста.

## Cтек протоколов TCP/IP

Существует ряд протоколов, на которых всё основывается:

* Ethernet II;
* IP — Internet Protocol;
* ICMP — Internet Control Management Protocol;
* UDP — User Datagram Protocol;
* TCP — Transmission Control Protocol;
* DHCP — Dynamic Host Configuration Protocol;
* DNS — Domain Name Service.

Рассмотрим их поподробнее. Данные передаются порциями, которые называются Protocol Data Unit (PDU). PDU состоит из заголовка (header) и полезных данных (payload). PDU одного протокола в полезных данных могут содержать PDU другого протокола. Это называется инкапсуляцией. В зависимости от уровня, на котором работает сетевой протокол, PDU могут называться по-разному:

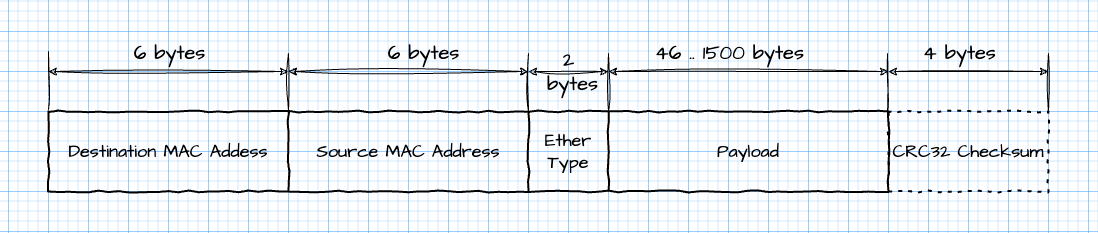
* на канальном уровне — фрейм;
* на сетевом уровне — пакет (IP, ICMP);
* на транспортном уровне — сегмент или датаграмма (TCP, UDP);
* на прикладном уровне — сообщение (DNS, DHCP).

Но в разной литературе могут не следовать правилу, например, часто можно встретить IP-датаграмму, TCP-пакеты или UDP-пакеты. А в программах по анализу трафика в сети все PDU называются пакетами. По мере приобретения опыта, вы будете лучше ориентироваться, что имелось в виду при употреблении того или иного термина.

## Протокол Ethernet

Обычно при описании протокола Ethernet опускаются чуть ли не до бит, а то и сигналов, передаваемых по сети. Я не буду углубляться в такие подробности, к тому же с большой долей вероятности вы используете WiFi-соединение, поэтому различные средства для перехвата сетевого трафика, как правило, будут показывать его как IP-пакеты, инкапсулированные в Ethernet II фреймы. Эти фреймы имеют мало чего общего с фреймами WiFi, но унифицируют работу с канальным уровнем.

Ethernet-фрейм, который передаётся или принимается драйвером сетевого адаптера, состоит из заголовка и полезных данных.



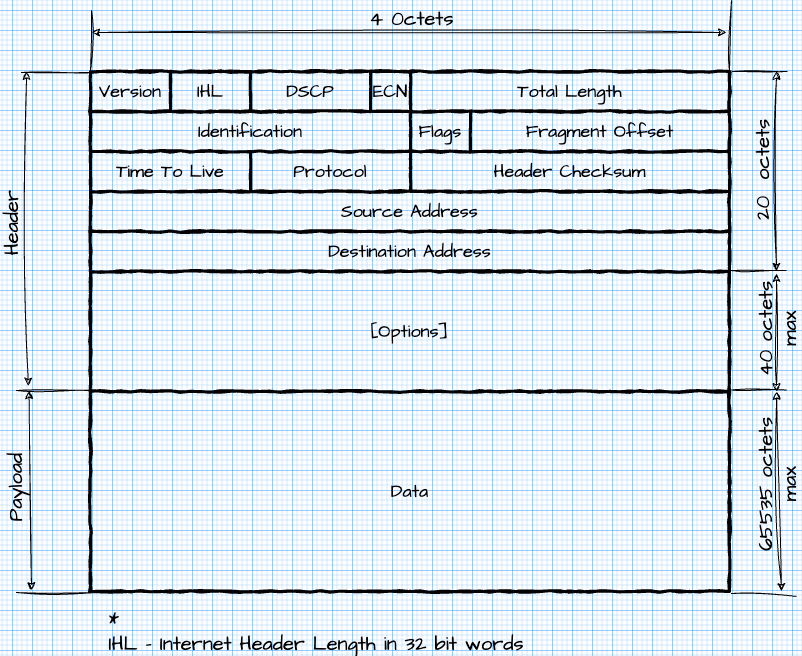
Часто можно встретить информацию о минимальном размере Ethernet-фрейма, преамбуле и контрольной сумме. Но это уровень реализации драйвера сетевого адаптера или аппаратной реализации самого адаптера.

Ethernet-фрейм содержит MAC-адрес получателя, MAC-адрес отправителя, тип фрейма и сами данные.

### ▍ Протокол IP

Оригинальное описание протокола находится в [RFC 791 Internet Protocol — DARPA Internet Program Protocol Specification](https://datatracker.ietf.org/doc/html/rfc791).

#### ▍ Структура пакетов

Структура IP-пакета приведена ниже.  
  


#### ▍ IP-пакеты и IP-датаграммы

Протокол IP позволяет передавать данные порциями. В литературе можно встретить два похожих термина: IP-датаграмма и IP-пакет.

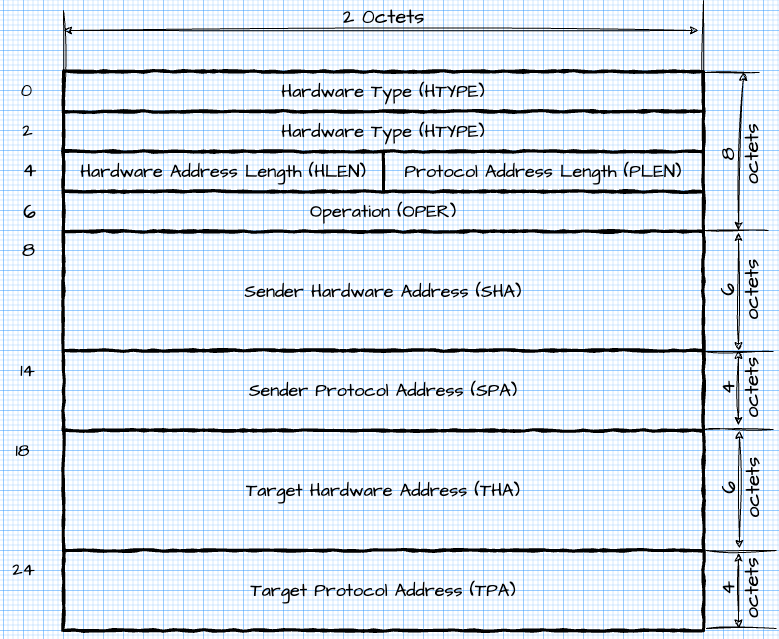
IP-датаграмма — это те данные, которые передаются на сетевой уровень, а IP-пакет — это те данные, которые передаются в IP-сети. Размер IP-датаграммы ограничивается максимальным значением в поле Total Length. Размер IP-пакета ограничивается MTU (Maximum Transmission Unit) — максимально возможное количество данных, которые могут быть переданы одним фреймом на канальном уровне. Чтобы передать IP-датаграмму, которая содержит полезных данных больше, чем может поместиться в один IP-пакет, используется фрагментация.

#### ▍ Фрагментация IP-датаграмм

Протокол IP поддерживает фрагментацию IP-датаграмм. Cуть фрагментации заключается в том, что максимальный размер передаваемых данных в одной IP-датаграмме составляет 65535 байт (октетов), а максимальный размер данных, который может поместиться в PDU канального уровня (MTU), гораздо меньше (обычно он составляет 1500 или около байт). Если размер IP-датаграммы больше MTU, она будет разбита на несколько IP-пакетов, каждый из которых можно будет передать в PDU канального уровня. При получении IP-пакетов они будут собраны в IP-датаграмму, и она будет разобрана на транспортном уровне.  
  
Все IP-пакеты одной IP-датаграммы имеют одинаковое значение в поле Identification, а поле Offset содержит смещение в payload IP-датаграммы.  
  
Хотя фрагментация позволяет соединять хосты в разнородных сетях, её желательно избегать, так как она усложняет передачу данных и увеличивает нагрузку на сеть из-за создания новых IP-пакетов и перерасчёта контрольных сумм. Поэтому желательно, чтобы TCP-сегмент или UDP-датаграмма имела размер не больше MTU по пути следования IP-пакетов.

### ▍ Протокол ARP

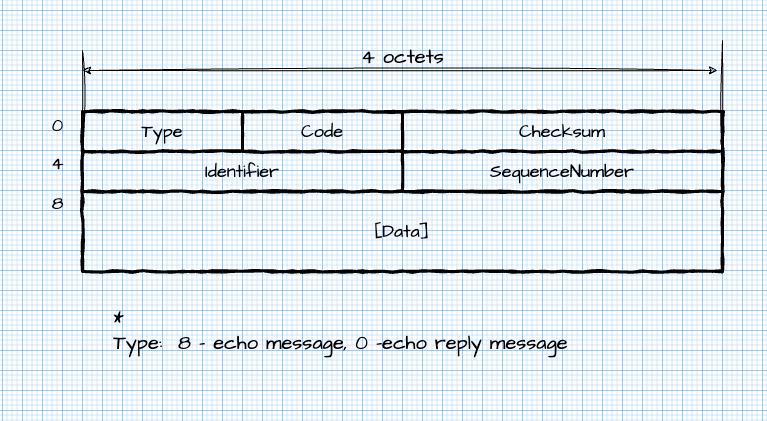
Протокол ARP используется для определения МАС-адреса физического интерфейса хоста по его IP-адресу. Описание протокола приведено в [RFC 826 — An Ethernet Address Resolution Protocol](https://datatracker.ietf.org/doc/html/rfc826). MAC-адрес не обязательно должен быть адресом в Ethernet-сети, структуру ARP-сообщения для Ethernet-сети.



### ▍ Протокол ICMP

Описание протокола приведено в [RFC 792 (Internet Protocol DARPA Internet Program Protocol Specification)](https://datatracker.ietf.org/doc/html/rfc792) .

#### ▍ Структура сообщений

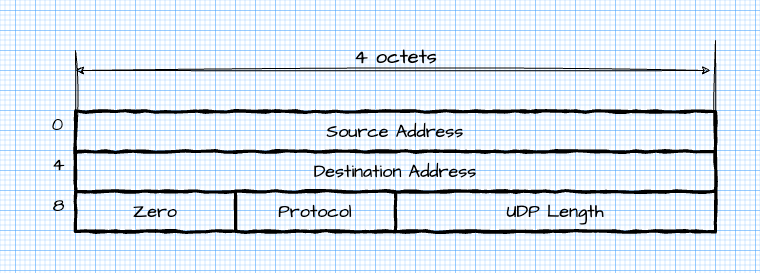
В описании протокола приведено много различных сообщений, но нам достаточно для начала разобраться с сообщениями Echo и Reply  
  
  
  
Хотя сообщения, используемые протоколом, инкапсулируются в IP-датаграммы, ICMP причисляют к тому же уровню, что и IP — сетевому.

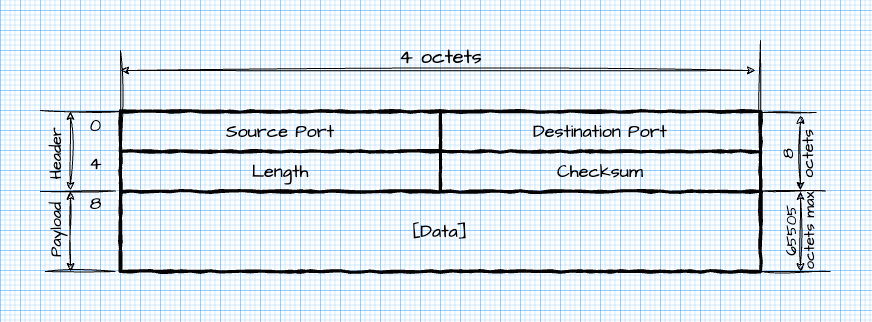
### ▍ Протокол UDP

Описание протокола приведено в [RFC 768 (User Datagram Protocol)](https://datatracker.ietf.org/doc/html/rfc768).  
  
Протокол позволяет двум процессам обмениваться UDP-датаграммами. Каждая UDP-датаграмма содержит в себе порт отправителя (Source Port), порт получателя (Destination Port), длину дейтаграммы (Length), контрольную сумму (Checksum) и собственно сами передаваемые данные.  
  
При расчёте контрольной суммы добавляется псевдозаголовок, который не передаётся, а только участвует в расчёте контрольной суммы.  
  
Протокол используется в качестве транспортного протокола там, где на транспортном уровне допускается дублирование получаемых данных, пропуск данных или не важен порядок, в котором данные будут доставлены.  
  
Как правило, обработка этих случаев возлагается на протоколы уровня приложений или не осуществляется вовсе. Например, в потоковом видео или аудио данные пропускаются, так как повторная передача данных является в этом случае бессмысленной. Но если вы хотите гарантированную доставку данных на транспортном уровне, то вам необходимо использовать протокол TCP.

### ▍ Структура сообщений

Структуры псевдозаголовка, используемого при вычислении контрольной суммы и UDP-датаграммы, приведены ниже.

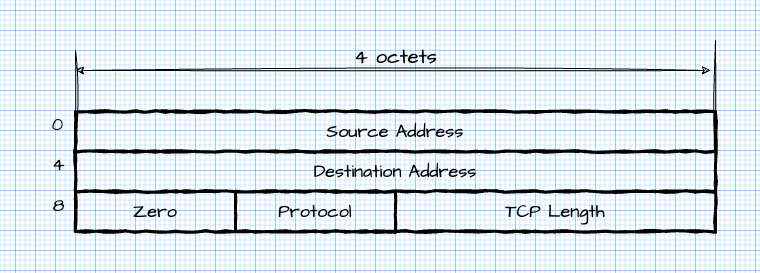
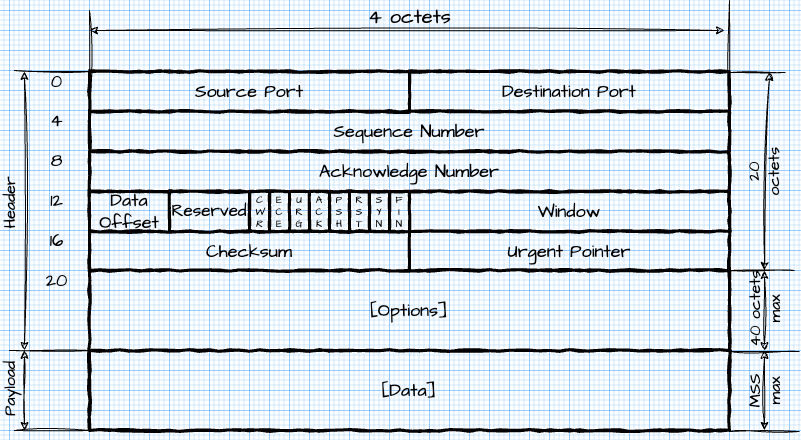




### ▍ Протокол TCP

Назначение протокола TCP — создать надёжное виртуальное полнодуплексное соединение между процессами. На данный момент самое свежее описание протокола приведено в [RFC 9293 — Transmission Control Protocol (TCP)](https://datatracker.ietf.org/doc/html/rfc9293).

#### ▍ Структура сообщений

Сообщения, используемые в протоколе TCP, называются TCP-сегментами. Просьба не путать с сегментами сети. Они с ними не имеют ничего общего. При расчёте контрольной суммы для TCP-сегмента как и в UPD используется псевдозаголовок. Но если для UDP расчёт контрольной суммы не является обязательным, то для TCP он таким не является.  
  
Структура псевдозаголовка и TCP-сегмента приведена ниже.  
  
  
  
  
  
Как видно из структуры заголовка, в протокол TCP, как и в протокол IP, заложены возможности для расширения и эволюции протокола при помощи поля Options.  
  
Ключевые понятия, необходимые для понимания TCP:

* Segment;
* Sequence Number;
* Acknowledge number;
* TCP Window;
* TCP Handshake;
* MSS — Maximum Segment Size;
* TCP Flags and TCP Options;
* Window Scaling;
* Selective Acknowledgement.

Для надёжной передачи данных используются Sequence Number, Acknowledge Number и Window Size. Рассмотрим, как они работают вместе. Передаваемые данные разбиваются на TCP-сегменты.  
  
За каждым TCP-сегментом закрепляется Sequence number.  
  
Sequence Number может принимать значения от 0x00000000 до 0xffffffff. Если каждому передаваемому байту присвоить номер и разбить на сегменты, то Sequence Number — это номер первого байта каждого сегмента.  
  
Sequence Number служит для упорядочивания сегментов, которые пришли не в порядке их отсылки (out of order), подтверждения полученных сегментов (acknowledgement), а также для повторной отсылки потерявшихся сегментов (retransmitting).  
  
Каждый передаваемый по сети TCP-сегмент содержит поля Sequence Number и Acknowledge Number. Sequence Number идентифицирует отправляемый сегмент, а Acknowledge Number указывает на то, какой сегмент ожидается.  
  
Значения Sequence Number и Acknowledge Number позволяют отслеживать прогресс передачи данных по TCP-соединению. Каждая сторона генерирует случайное число из диапазона от 0 до 2^32, которое называется ISN. Это число является началом для генерирования Sequence Numbers отсылаемых сегментов.  
  
Сегменты отсылаются только те, которые попадают в окно. Размер этого окна сообщается отправителю при установлении соединения, но может быть изменён принимающей стороной в дальнейшем.  
  
По мере получения подтверждений с принимающей стороны, окно сдвигается по кругу.  
Перед передачей данных, необходимо установить соединение. При установке соединения стороны обмениваются параметрами будущего соединения. Любая сторона может инициировать разрыв соединения.  
  
При установке соединения стороны обмениваются параметрами Sequence Number, Acknowledge Number, Window Size, а также параметрами, которые передаются в поле TCP Options (MSS, Window scale).

#### ▍ Флаги TCP

Для управления TCP-соединением используются флаги в отсылаемых TCP-сегментах. Наиболее важными являются:

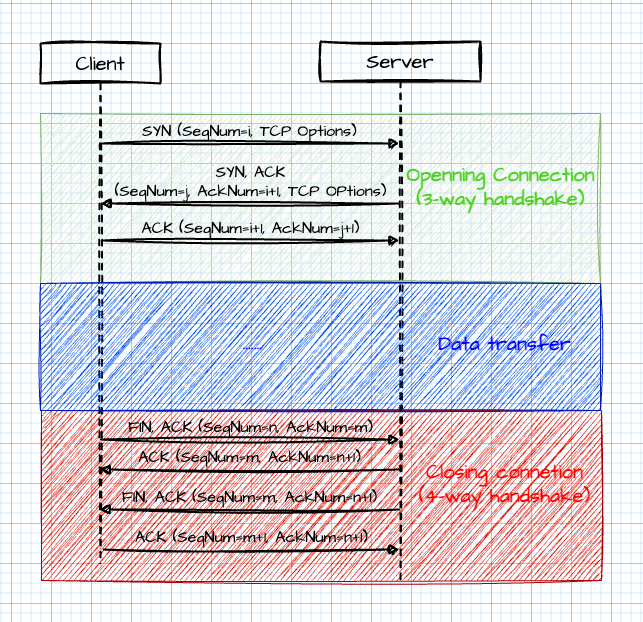
* SYN — используется при установлении TCP-соединения;
* ACK — означает, что сегмент был получен принимающей стороной;
* FIN — используется при нормальном (graceful) закрытии TCP-соединении;
* RST — используется при аварийном закрытии TCP-соединения.

#### ▍ Опции TCP

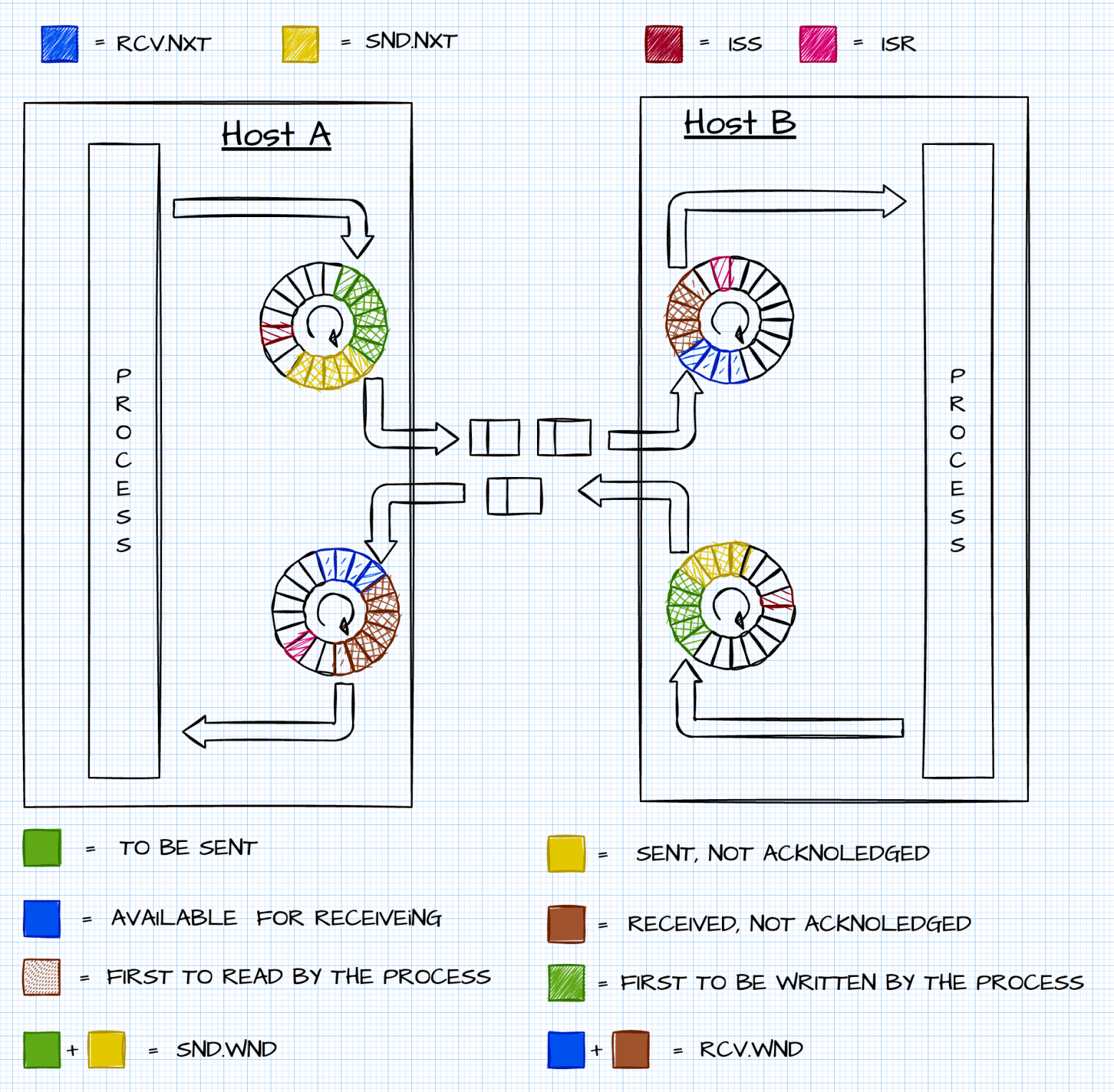
Протокол TCP разработан таким образом, что его можно расширять используя механизм опций. Опции это дополнительные поля, которые передаются в заголовке. Например, при установлении соединения стороны обмениваются опциями, Window Scale, MSS. В зависимости от настроек стека в TCP-сегменте может передаваться опция TCP timestamp. Если опция не поддерживается, она игнорируется стеком.

#### ▍ Открытие соединения, передача данных, закрытие соединения

В RFC 9293 приведено подробное описание и представлена диаграмма состояний для TCP-соединения. Ниже представлена диаграмма последовательностей для жизненного цикла TCP соединения.



Обратите внимание, как меняются значения в полях SequenceNumber и AcknowledgeNumber предаваемых TCP сегментов.  
  
Если при установлении TCP-соединения используется 3-way handshake, то с закрытием существует множество различных вариантов. На диаграмме приведен 4-way handshake, который будет в случае, если сервер не обработал все данные от клиента при получении TCP-сегмента FIN,ACK. В случае, если данные все обработанные, то выполняется 3-way handshake (сервер отсылает только сегмент FIN,ACK). Механизм передачи и приёма TCP-сегментов:



#### ▍ Настройка стека TCP/IP

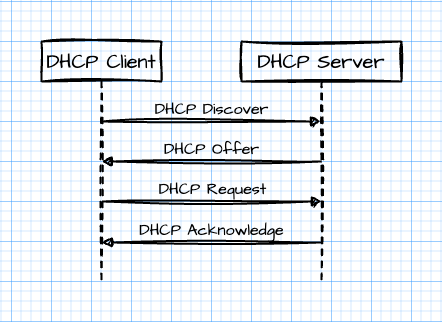
Работу стека протоколов TCP/IP можно настраивать в операционной системе, только делать это лучше в том случае, если вы точно осознаёте, что именно вы изменяете и зачем.  
  
В Windows стек можно настроить командой netsh, В Linux/MacOS — командой sysctl. В зависимости от операционной системы перечень и значения по умолчанию настраиваемых параметров могут отличаться. Например, в Windows tcp timestamps отключены по умолчанию. Чтобы включить, нужно выполнить команду:  
  
netsh int tcp set global timestamps=enabled  
В Linux, напротив, tcp timestamps включены, если вы хотите их отключить нужно выполнить команду:  
  
sysctl -w net.ipv4.tcp\_timestamps=0  
В MacOS включить/отключить tcp timestamps у вас не получится.

### ▍ Протокол DHCP

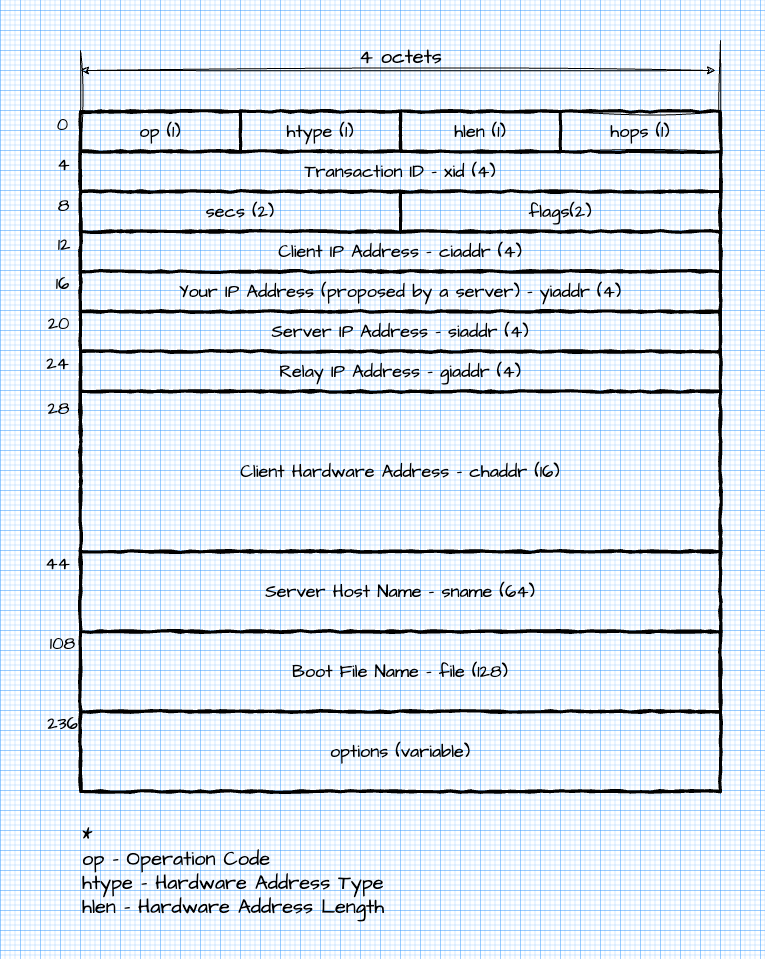
Для работы в сети TCP/IP хост необходимо настроить. Минимально необходимо указать его IP-адрес и маску подсети. Также может понадобиться указать адрес шлюза и адрес DNS-сервера. Протокол DHCP позволяет хосту получить эти данные автоматически из сети.  
  
Существуют различные варианты использования данного протокола, но мы рассмотрим основной успешный сценарий получения IP-адреса хостом, который состоит из обмена 4 сообщениями.

#### ▍ Получение конфигурации

1. Изначально хост не имеет IP-адреса и не знает, где расположен DHCP-сервер, который ему эту информацию может предоставить. Поэтому он посылает широковещательное сообщение DHCP Discover в свой сегмент сети.  
  
2. Если в сети присутствует DHCP-сервер, он отвечает unicast-сообщением DHCP Offer, в котором содержится предлагаемая конфигурация для хоста.  
  
3. Хост посылает unicast-сообщение DHCP Request, в котором указывает, назначенный ему IP-адрес  
  
4. Сервер отвечает unicast-сообщением DHCP Acknowledge, которое говорит о том, что конфигурация хосту назначена.  
  
Диаграмма последовательностей приведена ниже.



#### ▍ Структура сообщений

Описание протокола приведено в [RFC 2131 — Dynamic Host Configuration Protocol](https://datatracker.ietf.org/doc/html/rfc2131). DHCP-протокол является расширением более раннего протокола BOOTP ([RFC 951 Bootstrap Protocol](https://datatracker.ietf.org/doc/html/rfc951)). Поэтому заголовок DHCP-сообщения почти полностью совпадает с BOOTP-сообщением. Поле options всегда начинается с магического числа 0x62825363, за которым следуют DHCP-опции, описанные в [RFC 2132 DHCP Options and BOOTP Vendor Extensions](https://datatracker.ietf.org/doc/html/rfc2132) .  
  
Каждая опция состоит из кода, длины и одного или нескольких октетов сообщения. Исключение составляют опции с кодами 0x00 (заполнитель) и 0xff (конец опций). Размер DHCP-сообщения в октетах должен быть кратным четырём, поэтому после опции с кодом 0xff может быть одна или несколько опций с кодом 0x00.  
  
Как выглядит DHCP-сообщение, приведено ниже:  
  


### ▍ Протокол DNS

Протокол DNS регламентируется [RFC 1035 DOMAIN NAMES — IMPLEMENTATION AND SPECIFICATION](https://datatracker.ietf.org/doc/html/rfc1035).  
  
Скорее всего, вы имеете представление о службе DNS, так она используется для преобразования доменного имени хоста в его IP-адрес. Вроде бы всё просто, и для программиста всё скрывается за простым API, но этот факт затрудняет понимание сути DNS. IP-адрес хоста — это лишь часть той информации, которую может хранить DNS.  
  
Программы типа nslookup и функции в Winsock или glibc запутывают в понимании DNS. Я бы советовал начинать изучение DNS с экспериментов с утилитой dig и анализа трафика. DNS нужно рассматривать, не привязываясь к IP. DNS — это распределённая иерархическая база данных доменов.  
  
Чтобы убедиться, что это действительно база данных можете зайти [на сайт](https://trendoceans.com/dns-toys/) и увидеть тому подтверждение.

#### ▍ Домен

Что такое домен? Домен можно определить как именованную сущность, которая содержит метаинформацию о себе и находится в ведении организации или частного лица. К такой информации относятся IP-адрес, имя используемого почтового сервера, имена серверов имён, обслуживающих данный домен и др. Для упрощения управления доменами они организовываются в иерархию.  
  
Домены не существуют сами по себе, они хранятся серверами имён. Информация, связанная с доменом, объединяется в зону, которая обслуживается конкретным сервером имён.

#### ▍ Зона

Зона — это информация о доменах, размещённая на DNS-сервере.  
  
Корневую зону доменов интернета можно посмотреть [здесь](https://www.internic.net/domain/root.zone).  
  
Корневая зона доменов обслуживается [13 корневыми доменными серверами](https://www.iana.org/domains/root/servers).

#### ▍ Ресурсная запись

Система доменных имён позволяет структурировано хранить информацию. Доменные имена организуются в древовидную структуру, листьями которой являются ресурсные записи (Resource Record (RR)). Каждая RR имеет класс и тип. Как правило, классы могут принимать следующие значения:

* IN (Internet);
* CH (Chaos);
* HS (Hesiod).

Хотя можно встретить классы CH и HS, применение их специфическое, и с большой долей вероятности вы с ними не столкнётесь.  
  
А вот типов RR гораздо больше, и часть из них знать обязательно:

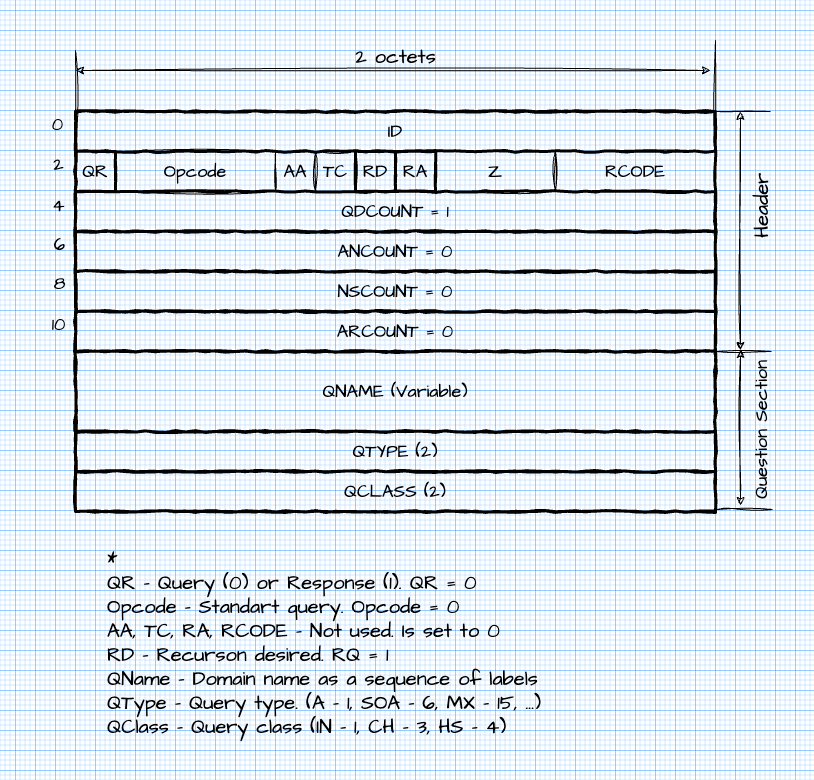
* A (Address) — IP-адрес, закреплённый за доменным именем;
* AAAA (IPv6 Address) — IPv6-адрес, закреплённый за доменным именем;
* CNAME (Canonical Name);
* MX (Mail Exchanger);
* NS (Name Server);
* PTR (Pointer);
* SOA (Start of Authority);
* TXT (Text);
* SRV (Service).

#### ▍ Резолвер

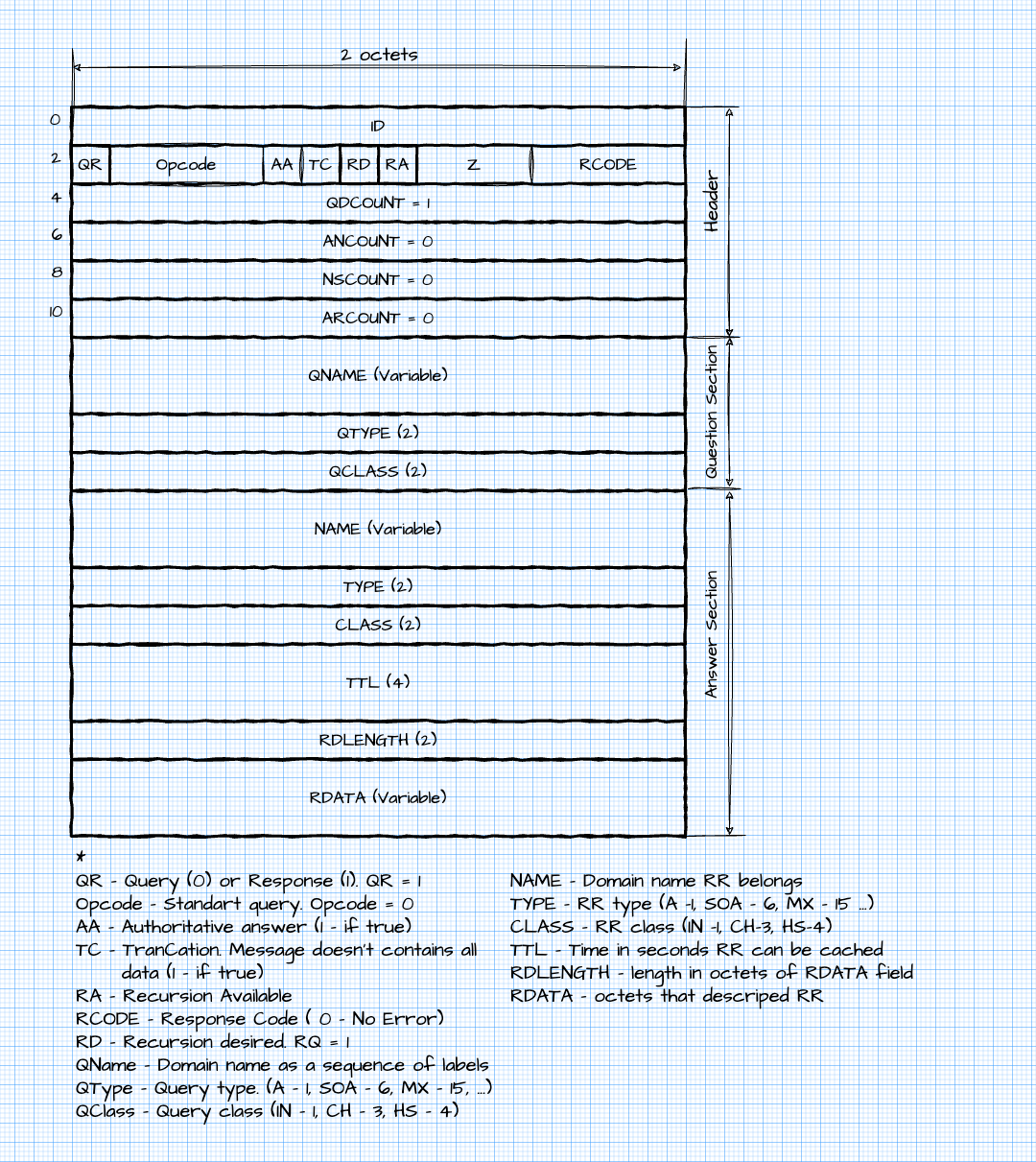
Domain Name Service позволяет по имени хоста получить его IP-адрес. Реализуется это при помощи распределённой базы данных, работающих на множестве хостов. Хост, как правило, взаимодействует с локальным компонентом, называемым резолвером (resolver). К резолверу можно обратиться через API операционной системы или библиотеки языка программирования.  
  
При выполнении API-функции резолвер проверяет в своём локальном кэше IP-адрес для имени хоста. Если не находит, то пытается сделать запрос DNS-серверу, адрес которого прописан в конфигурации. Что может происходить дальше опустим, для упрощения описания. В конечном итоге DNS-сервер возвращает IP-адрес для имени хоста или ошибку, если такой хост отсутствует. Резолвер помещает эту информацию в кэш и возвращает значение вызвавшему коду. В Linux API к резолверу находится в библиотеке glibc, в Windows — в библиотеке Winsock.  
  
Обычно в примерах по сетевому программированию приводится именно работа с API резолвера. Я же в практической части покажу, как можно послать запрос DNS-серверу на низком уровне, сформировав IP-пакет, содержащий DNS-запрос.

#### ▍ Структура пакетов

Простой DNS-запрос выглядит следующим образом:



Пример DNS-ответа приведён ниже:



## Как правило, запросы и ответы отсылаются с использованием UDP в качестве транспортного протокола. Однако если ответ слишком большой, сервер вернёт флаг TC. Это означает, что для получения полного ответа нужно использовать TCP в качестве транспорта. Сетевое программирование и анализ сетевого трафика

### ▍ Стеки протоколов TCP/IP и программные интерфейсы

Практически любая современная операционная система имеет поддержку работы с семейством протоколов TCP/IP. Набор компонентов операционной системы, которые обеспечивают коммуникацию посредством семейства протоколов TCP/IP, называют стеком протоколов. Разные операционные системы предоставляют доступ к стеку, используя различные программные интерфейсы. Наиболее распространённым является интерфейс сокетов.  
  
Хотя и существуют различия в этом интерфейсе для различных операционных систем, большинство функций схожи. Работа с сокетами подразумевает программирование на языке С. Однако для различных языков написаны обёртки, которые позволяют кроссплатформенно работать с сокетами. В зависимости от языка, обёртки могут предоставлять больше или меньше функций. Например обёртка в Python больше зависит от платформы, на которой исполняется, чем обёртка в Java.

### ▍ Анализ пакетов в сети

Обычно в литературе по сетевым технологиям рассматриваются основы компьютерных сетей, потом сокеты. Но то, как именно можно создать и послать пакет, структура которого подробно расписана, не приводится. Я хочу восполнить этот пробел.  
  
Инструкции процессора в операционной системе могут исполняться в режиме ядра или режиме пользователя. Большинств кода, который пишет программист — это инструкции процессору, которые выполняются в режиме пользователя. В режиме ядра выполняется код драйверов и ядра операционной системы.  
  
Стек TCP/IP выполняется в режиме ядра, а из режима пользователя он, как правило, доступен только через вызов АPI-сокетов. Поэтому программист только может выполнить высокоуровневые операции, такие как открыть TCP-соединение, передать данные по TCP-соединению или передать данные как UDP-датаграмму. Доступа к формируемым пакетам он не имеет.  
  
Чтобы получить доступ к формируемым пакетам, используются так называемые Raw-сокеты. Замечу, что в Windows Raw-сокеты имеют ограниченный функционал. Например, невозможно создать Raw-сокет, который бы позволял работать с Ethernet-фреймами. Поэтому для получения доступа к формированию Ethernet-фреймов используют специальный NDIS-драйвер и библиотеку npcap. В Linux же достаточно же просто создать AF\_PACKET Raw-сокет.

### ▍ libpcap

Библиотека предоставляет высокоуровневый, если так можно сказать, API для формирования, фильтрации и перехвата пакетов, который прячет детали реализации для разных платформ.

### ▍ Scapy

Для Windows существует библиотека npcap, для Unix-подобных систем — libpcap. Python-библиотека Scapy позволяет писать платформо-независимые приложения, игнорируя этот факт. Также она поддерживает многие сетевые протоколы. И содержит средства для разбора, генерации и отсылки пакетов из пространства пользователя. Например, расчёт контрольной суммы для заголовка IP-пакета, указание правильного Ethertype или Protocol при инкапсуляции пакетов. Библиотека удобна для экспериментирования и исследования сетевых протоколов.

### ▍ Wireshark

Программа Wireshark, можно сказать, самая распространённая программа для анализа сетевого трафика. При помощи графического интерфейса, можно записывать сетевой трафик на диск, фильтровать пакеты, рассматривать их структуру, отслеживать работу TCP-сессий и многое другое. Совместно с Scapy, это, наверное, лучший набор для изучения основ сетевых протоколов и проведения различных экспериментов.

## Примеры программного кода

Далее я хочу привести фрагменты из моего приложения для упрощения понимания основ TCP/IP.

### ▍ Создание ARP-запроса

**def** **create\_arp\_request**(ip):  
 **return** Ether(dst="ff:ff:ff:ff:ff:ff")/ARP(pdst=ip)

### ▍ Создание сообщения Echo, используемого в протоколе ICMP

**def** **create\_ping\_request**(ip):  
 **return** IP(dst=ip)/ICMP()

### ▍ Создание сообщений DHCP Discover и DHCP Request

**def** **create\_dhcp\_discover**(mac):  
 **return** (Ether(src=mac, dst='ff:ff:ff:ff:ff:ff')  
 / IP(src='0.0.0.0', dst='255.255.255.255')  
 / UDP(dport=67, sport=68)  
 / BOOTP(op=1, chaddr=mac\_to\_bytes(mac))  
 / DHCP(options=[('message-type', 'discover'), 'end']))  
  
def **create\_dhcp\_request**(mac, ip):  
 **return** (Ether(src=mac, dst='ff:ff:ff:ff:ff:ff')  
 / IP(src='0.0.0.0', dst='255.255.255.255')  
 / UDP(dport=67, sport=68)  
 / BOOTP(op=1, chaddr=mac\_to\_bytes(mac), ciaddr=ip)  
 / DHCP(options=[('message-type', 'request'), 'end']))

### ▍ Создание сообщения DNS на получение Resource Record типа AA

**return** IP(dst=dns\_server)/UDP()/DNS(rd=1, qd=DNSQR(qname=host\_name))

### ▍ Организация TCP соединения

**def** **create\_tcp\_syn**(src, sport, dst, dport, seq):  
 **return** IP(src=src, dst=dst)/TCP(seq=seq,sport=sport, dport=dport, flags="S")  
  
def **create\_tcp\_ack**(src, sport, dst, dport, ack, seq):  
 **return** IP(src=src, dst=dst)/TCP(ack=ack, seq=seq, sport=sport, dport=dport, flags="A")  
  
  
def **create\_tcp\_fin\_ack**(src, sport, dst, dport, ack, seq):  
 **return** IP(src=src, dst=dst)/TCP(ack=ack, seq=seq, sport=sport, dport=dport, flags="F")  
  
def **create\_and\_close\_tcp\_connection**(host, dport):  
 dst = socket.gethostbyname(host)  
 src = get\_default\_interface\_ip()  
 sport = 12360  
 seq = 1000  
 syn = packets.create\_tcp\_syn(src=src, sport=sport, dst=dst, dport=dport, seq=seq)  
 response = send\_receive\_l3(syn)  
 ack = packets.create\_tcp\_ack(src=src, sport=sport, dst=dst, dport=dport, seq=response[TCP].ack, ack=response[TCP].seq+1)  
 send\_l3(ack)  
  
 fin\_ack = packets.create\_tcp\_fin\_ack(src=src, sport=sport, dst=dst, dport=dport, seq=response[TCP].ack, ack=response[TCP].seq+1)  
 response = send\_receive\_l3(fin\_ack)  
 ack = packets.create\_tcp\_ack(src=src, sport=sport, dst=dst, dport=dport, seq=response[TCP].ack, ack=response[TCP].seq+1)  
 send\_l3(ack)  
  
  
Так как мы вмешиваемся в стандартную работу стека протоколов TCP/IP, и стек не догадывается о нашем вмешательстве, то он может отсылать RST-сегменты на сегменты, которые он не ожидает. Чтобы такого не было, нам придётся временно запретить отсылку RST-сегментов. В MacOS это выполняется при помощи команды pfctl. В Linux можно использовать iptables.  
  
echo "block drop out proto tcp from any to any flags R/R" | cat /etc/pf.conf - | sudo /sbin/pfctl -Ef -