Компьютерные сети

Василий Купоросов
Лекции: Владимир Михайлович Ицыксон
Практики: Артем Олегович Алексюк

3 февраля 2019 г.

Оглавление

Глава 1

Лекции

1. Введение

1.1.1. Сетевая модель ISO/OSI

Чтобы разные компьютерные системы, сделанные в разных странах, с разными протоколами, могли хорошо взаимодействовать, инженеры договорились, каким образом будут представляться уровни взаимодействия систем. На каждом уровне действуют свои протоколы, используются свои элементы информации, способы кодирования, шифрования, и каждый уровень выполняет свои функции. Чем ниже уровень, тем процессы, происходящие на нем, ближе к физической среде, чем выше – тем ближе к прикладным программам и к пользователю. В результате получилась семиуровневая модель, которую обычно представляют в виде таблицы.

№	Название	Тип данных	Функции
7	Прикладной Application	Данные	Доступ к сетевым службам
6	Представления Presentation	Данные	Представление и шифрование данных
5	Сеансовый Session	Данные	Управление сеансом связи
4	Транспортный Transport	Сегменты или датаграммы	Прямая связь между конечными пунктами и надёжность
3	Сетевой Network	Пакеты	Определение маршрута и логическая адресация
2	Канальный Link	Биты или кадры	Физическая адресация
1	Физический Physical	Биты	Работа со средой передачи, сигналами и двоичными данными

- Физический уровень отвечает за физическое кодирование битов. Этот уровень всегда реализуется аппаратно.
- На канальном уровне биты объединяются в массивы данных (кадры/фреймы). Этот уровень описывает, каким образом устройства, подключенные к одному сегменту сети, обмениваются между собой данными. В том числе здесь решается задача адресации в пределах одного сетевого сегмента с использованием MAC-адреса. Чаще всего канальный уровень реализуется аппаратно, но иногда может и программно.
- Сетевой уровень отвечает за передачу данных между сетями. Обеспечивает адресацию в глобальной сети с использованием логического адреса и передачу данных между двумя сетями, возможно через какие-то другие сети. При этом надежность передачи данных не обеспечивается. Начиная с этого уровня обычно всё реализовано программно, но здесь бывают и аппаратные реализации.
- Транспортный уровень отвечает за передачу данных между приложениями с заданным уровнем надежности. Здесь происходит адресация приложений, то есть доставляем информацию не просто на устройство, а передаем конкретной программе. В TCP/IP на транспортном уровне есть два протокола. TCP – более надежный и UDP – более быстрый.
- Сеансовый уровень отвечает за организацию сеанса связи между двумя узлами сети. Здесь мы создаем, поддерживаем и разрываем соединение.

- Уровень представления отвечает за согласование форматов информации на концах соединения. Сюда входит сжатие, шифрование информации, разные кодировки. Протоколы этого уровня нигде не реализованы, и задачи согласования решаются на прикладном уровне.
- Прикладной уровень отвечает за всякие прикладные функции, такие как передача файла, передача электронной почты, передача веб-страниц, итп.

Протоколы на разных уровнях ISO/OSI вкладываются друг в друга через механизм инкапсуляции, который называется стек протоколов.

Механизм следующий: в поле 'данные' протокола вкладываются данные протокола следующего уровня. В итоге данные устроены так:

заголовок 2 уровня	заголовок 3 уровня	...	данные	концовка (есть только в ethernet)

На практике разные стеки протоколов по-разному реализуют эту модель. Какие-то уровни пропускаются и объединяются с другими. В стеке TCP/IP четыре уровня – прикладной, транспортный, сетевой и канальный.

К архитектуре компьютерных сетей относятся только верхние пять уровней, и мы будем рассматривать только их.

1.1.2. Архитектуры компьютерных сетей

Существует много разных сетевых архитектур. Вот небольшой список:

- DNA (Digital Network Architecture) – архитектура, разработанная компанией DEC, которая раньше была одой из самых влиятельных на рынке сетей и цифровых устройств.
- SNA (Systems Network Architecture) – архитектура, разработанная компанией IBM для своих компьютеров. Используется до сих пор.
- DARPA (TCP/IP, Internet) – самая известная архитектура, разработанная министерством обороны США.
- IPv6 – разновидность TCP/IP. Это отдельная архитектура, во многом отличающаяся от оригинальной.
- Novell Netware – ранее самая популярная архитектура локальных сетей. Сейчас архитектура осталась, но протоколы почти не используются.
- SMB (Server Message Block) – совместная разработка Microsoft и IBM для ОС Windows и OS/2. Используется для доступа к разделяемым ресурсам – к серверам, принтерам, итп.
- AppleTalk – архитектура компании Apple для устройств Apple. Ни с чем остальным не совместима.
- XNS – архитектура компании Xerox. На основе нее была сделана архитектура TCP/IP и некоторые другие.

Мы подробно будем говорить только о TCP/IP и IPv6.

1.1.3. Характеристики архитектур К. С.

Все сетевые архитектуры обладают следующим характеристикам:

- Иерархия протоколов (в основном протоколов третьего уровня и выше)
- Соответствие модели ISO
- Адресация. В сети находится много сущностей, которые можно адресовать:
  - Узлы. Существует несколько видов адресации узлов:
    * Индивидуальная – один адрес соответствует одному узлу (например сетевой карте или маршрутизатору).
    * Групповая – один адрес соответствует группе узлов. Например когда хотим что-то вещать для фиксированной группы устройств.
    * Широковещательная – когда надо сообщить что-то всем объектам сети.

    В TCP/IP используются все эти виды.
  - Приложения. На транспортном уровне появляются приложения и сервисы, и их тоже надо адресовать.
- Связь с канальным уровнем. Хоть мы от него и абстрагируемся, нам надо как-то пользоваться его функциями. Механизм связи включает в себя:
  - Разрешение адресов, то есть сопоставление адресам сетевого уровня адресов канального уровня.
  - Фрагментация. Пакеты сетевого уровня могут быть большие, а кадры на канальном уровне маленькие. Поэтому сетевые пакеты приходится разбивать на кусочки. Когда мы передаем пакет между устройствами, он может проходить через несколько промежуточных узлов. При этом канальный уровень между ними может быть разный на разных участках маршрута. Поэтому фрагментация бывает двух видов:
    * Поузловая – каждый промежуточный узел сам обеспечивает фрагментацию. Используется в IPv4.
    * На источнике – делим сразу так, чтобы заведомо прошло весь путь. Используется в IPv6.
- Сетевые протоколы – это основа архитектуры, за счет протоколов всё и функционирует.
- Маршрутизация – одна из самых сложных функций. Она характеризуется различными атрибутами:
  - По типу маршрута:
    * Индивидуальная – строим маршрут для передачи информации одному узлу.
    * Групповая – строим маршрут для передачи информации группе узлов.
  - По адаптивности к изменениям в сети:
    * Статическая – не зависит от того, что происходит в сети, администратор сам определяет, как идут пакеты.
    * Динамическая – оптимальный маршрут строится автоматически с помощью динамических протоколов.
    * Предопределенная (”от источника”) – отправляемый пакет сам определяет, через какие узлы ему надо пройти.

  - По месту проведения маршрутных вычислений:
    * Централизованная – один узел рассчитывает маршруты для всей сети.
    * Децентрализованная – каждый узел решает, как проводить маршрутизацию.
    * Гибридная – часть вычислений происходит централизованно, часть на местах.
  - По числу возможных маршрутов:
    * Однопутевые – между двумя точками бывает одновременно только один маршрут.
    * Многопутевые – между двумя точками может быть одновременно несколько маршрутов.
  - По характеру информации, необходимой для маршрутизации:
    * Глобальные – необходимо знать информацию обо всей сети, чтобы построить оптимальный маршрут.
    * Локальные – достаточно только информации о соседях.
    * Смешанные – нужна локальная информация и часть глобальной.

- Транспортные механизмы – обеспечивают заданный уровень надежности передачи данных между приложениями. Транспортные механизмы бывают:
  - Датаграммные – менее надежные. Позволяют просто доставить пакет от одного приложения к другому.
  - Потоковые – обеспечивают уровень надежности и порядок доставки данных.
  - Многопоточные – когда необходимо поддерживать сразу несколько каналов связи между двумя приложениями.

- Именование ресурсов (в TCP/IP это DNS)
  - Одноуровневые.
  - Двухуровневые.
  - Иерархические – используется в TCP/IP.

- Прикладные протоколы – протоколы удаленного терминала, передачи файлов, почты, итп.
- Управление.
- Защита информации.

1.1.4. История создания интернета

- 1957 г. – создание DARPA (Defense Advances Research Projects Agency). Агентство занималось созданием единой сети для всех военных узлов США.
- 1968 г. – разработка сети ARPANET. Эта сеть не использовала протоколов TCP/IP, так как их тогда не существовало.
- 1969 г. – осуществление первой передачи данных между двумя университетами.
- 1974 г. – разработка TCP/IP.
- 1983 г. – переход ARPANET с NCP на TCP/IP.

- 1984 г. – создание системы DNS.
- 1989 г. – создание WWW – протокол HTTP, система серверов, использование браузеров.
- 1990 г. – переход на единый термин Internet.
- 1993 г. – первый графический веб-браузер Mosaic. До этого браузеры были текстовыми и использовались из консоли.

1.1.5. Стандартизующие организации Internet

В других стандартах, например Ethernet, обычно прописаны все детали до мелочей. Для Internet такого нет. Он развивался довольно демократично и как таковых стандартов для него не существует. Есть ряд организаций, которые занимаются его развитием:

- Internet Society (ISOC) – сообщество, занимающееся развитием интернета. В ее составе есть организация:
  - Internet Architecture Board (IAB) – координирует развитие TCP/IP. Эта организация состоит из департаментов:
    - Internet Engineering Steering Group (IESG) – рассматривает стандарты и технические работы для IETF.
    - Internet Engineering Task Force (IETF) – отвечает за разработку стандартов на протоколы и архитектуру Internet.
    - Internet Research Task Force (IRTF) – занимается развитием технологий, которые могут понадобиться для Internet в будущем.
    - Internet Corporation of Assigned Names and Numbers (ICANN) – занимается централизованным назначением IP-адресов и всем, что с этим связано.

1.1.6. Координирующие организации Internet

- Network Information Center (NIC) – организации, ответственные за распределение адресов:
  - InterNIC – на территории США.
  - Reseaux IP Europeens (RIPE) – в Европе.
  - African Network Information Centre (AfriNIC) – в Африке.
  - Asia Pacific Network Information Centre (APNIC) – в Азии и Океании.
  - Regional Latin-American and Caribbean IP Address Registry (LACNIC) – в Латинской Америке.
  - Russian Institute for Public Networks (RIPN) – в России.

1.1.7. Стандарты TCP/IP

Все стандарты, касающиеся интернета, формируются через RFC (Request for Comments). Все эти документы можно найти по по ссылке.

Часть этих документов являются информационными – FYI (For Your Information), а остальные стандартами – STD (Standard).

2. Архитектура TCP/IP

1.2.1. Иерархия протоколов TCP/IP

<table>
<tr><td>7</td><td colspan="2">SMTP, POP3, HTTP, FTP,
DNS, TELNET, SSH, ...</td><td>DNS, TFTP,
SNMP, ...</td></tr>
<tr><td>6</td><td colspan="3"></td></tr>
<tr><td>5</td><td colspan="3"></td></tr>
<tr><td>4</td><td colspan="2">TCP</td><td>UDP</td></tr>
<tr><td rowspan="2">3</td><td>ICMP,
IGMP</td><td colspan="2" rowspan="2">IP</td></tr>
<tr><td>ARP</td></tr>
<tr><td>2</td><td colspan="3">Ethernet, Token Ring, FDDI, ...</td></tr>
<tr><td>1</td><td colspan="3"></td></tr>
</table>

В таблице показана приблизительная иерархия протоколов в TCP/IP. Нижние два уровня нас не интересуют.

Самый главный – протокол сетевого уровня IP. Он обеспечивает функционирование всего интернета. Также на сетевом уровне есть протоколы ICMP и IGMP, они обеспечивают служебные и контрольные функции. Протокол ARP обеспечивает связь с канальным уровнем.

На транспортном уровне находятся протоколы TCP и UDP.

Пятого и шестого уровней нет, а на седьмом находятся такие протоколы, как:

- SMTP – электронная почта.
- POP3 – тоже электронная почта.
- HTTP – гипертекст и файлы.
- FTP – файлы.
- DNS – доменные имена.
- TELNET – работа с терминалом.
- SSH – защищенная работа с терминалом.

Вышеперечисленные протоколы используют TCP, а также есть протоколы, использующие UDP:

- DNS – доменные имена.
- TFTP – файлы. Используется в маршрутизаторах беспроводных устройств.
- SNMP – управляющий протокол.

1.2.2. Адресация в IP

Адресация в IP происходит с помощью IP-адресов. IP-адрес имеет 32 разряда и записывается в виде четырех десятичных чисел от 0 до 255 (например 195.19.212.13). Такой формат позволяет адресовать $2^{32} \approx 4 \cdot 10^9$ узлов. Благодаря такому большому объему адресов TCP/IP победила все остальные архитектуры, так как у них возможных адресов было сильно меньше. Тем не менее, сейчас такого количества адресов становится слишком мало для удовлетворения всех потребностей, и люди постепенно переходят на IPv6.

TCP/IP поддерживает индивидуальную, широковещательную и групповую адресацию.

При этом узлом считается не устройство, а сетевой интерфейс. Одно устройство может иметь несколько сетевых интерфейсов. Более того, одному сетевому интерфейсу может присваиваться несколько IP-адресов одновременно. Иногда, наоборот, один адрес может разделяться несколькими интерфейсами, но это тонкий момент, его пока рассматривать не будем.

Всё адресное пространство поделено на классы:

- Класс A – для сетей большого размера.
  - Формат адреса: `0nnnnnnn.hhhhhhhh.hhhhhhhh.hhhhhhhh`
    (`h` – разряды номера сети, `n` – разряды номера узла)
  - Количество сетей: $2^7 - 2 = 126$ (2 – зарезервированы)
  - Количество узлов: $2^{24} - 2 \approx 16 \cdot 10^6$
  - Диапазон адресов: `1.0.0.0` – `126.255.255.255`
  - Все адреса этого класса розданы.
- Класс B – для сетей среднего размера.
  - Формат адреса: `10nnnnnn.nnnnnnnn.hhhhhhhh.hhhhhhhh`
  - Количество сетей: $2^{14} - 2 \approx 16 \cdot 10^3$
  - Количество узлов: $2^{16} - 2 \approx 65 \cdot 10^3$
  - Диапазон адресов: `128.0.0.0` – `191.255.255.255`
  - Все адреса этого класса розданы.
- Класс C – для небольших сетей.
  - Формат адреса: `110nnnnn.nnnnnnnn.nnnnnnnn.hhhhhhhh`
  - Количество сетей: $2^{21} - 2 \approx 2 \cdot 10^6$
  - Количество узлов: $2^8 - 2 = 254$
  - Диапазон адресов: `192.0.0.0` – `223.255.255.255`
  - Все адреса этого класса розданы.
- Класс D – для групповых адресов.
  - Формат адреса: `1110xxxx.xxxxxxxx.xxxxxxxx.xxxxxxxx`
    (`x` – разряды адреса группы)
  - Количество адресов: $2^{28} - 2 \approx 256 \cdot 10^6$
  - Диапазон адресов: `224.0.0.0` – `239.255.255.255`

- Класс E – для экспериментов (зарезервировано).
  - Формат адреса: `1111xxxx.xxxxxxxx.xxxxxxxx.xxxxxxxx`
  - Количество адресов: $2^{28} - 2 \approx 256 \cdot 10^6$
  - Диапазон адресов: `240.0.0.0` – `254.255.255.255`
  - Так как это адреса для экспериментов, ни один валидный пакет, приходящий из сети, не может иметь адрес отправителя или получателя в этом диапазоне.

1.2.3. Зарезервированные IP-адреса

- Адрес `0.0.0.0`.

  Когда этот адрес записан в таблице маршрутизации, он означает маршрут по умолчанию, то есть маршрут, по которому будут посылаться данные, если никакой другой не задан.

  При адресации он означает всю текущую сеть. Этот вариант работает только в маршрутизаторах Cisco.

- Узел данной сети.

  Если мы знаем, что устройство находится в нашей локальной сети и не хотим писать его полный адрес, можно использовать один из следующих форматов, в зависимости от класса сети:

  - Класс A: `00000000.hhhhhhhh.hhhhhhhh.hhhhhhhh`
  - Класс B: `10000000.00000000.hhhhhhhh.hhhhhhhh`
  - Класс C: `11000000.00000000.00000000.hhhhhhhh`

  То есть вместо адреса сети – определенные зарезервированные значения. Такой формат есть в стандарте, но нигде не используется в IPv4.

- Конкретная сеть. Используется в таблице маршрутизации.

  Пишется только адрес сети, а вместо адреса узла – все нули:

  - Класс A: `0nnnnnnn.00000000.00000000.00000000`
  - Класс B: `10nnnnnn.nnnnnnnn.00000000.00000000`
  - Класс C: `110nnnnn.nnnnnnnn.nnnnnnnn.00000000`

- Конкретная сеть. Используется внутри пакета.

  Вместо адреса узла – все единицы. Означает широковещательный адрес IP сети:

  - Класс A: `0nnnnnnn.11111111.11111111.11111111`
  - Класс B: `10nnnnnn.nnnnnnnn.11111111.11111111`
  - Класс C: `110nnnnn.nnnnnnnn.nnnnnnnn.11111111`

- `255.255.255.255` – все узлы нашей локальной сети.

  Этот адрес отличается от предыдущего тем, что он действует строго в рамках локальной сети, то есть адресация происходит на канальном уровне. При этом устройства одной локальной сети могут находиться в различных IP сетях. IP сеть, в свою очередь, может состоять из нескольких локальных подсетей, поэтому широковещательная адресация происходит на сетевом уровне.

- Петля обратной связи (Loopback)

  Формат адреса: `01111111.xxxxxxxx.xxxxxxxx.xxxxxxxx`.

  Петля обратной связи – специальный адрес, который является виртуальным сетевым интерфейсом. Он живет внутри операционной системы и предоставляет способ обратиться к самому себе через сеть. Прекрасный способ тестирования и отладки сетевых приложений. Преимущество в том, что мы можем отлаживать приложения даже при отсутствии реальных сетевых интерфейсов. А также у него меньше накладных расходов и ошибок конфигурации.

- Адреса для внутреннего использования.

  – Класс A: `10.0.0.0 - 10.255.255.255`
  – Класс B: `172.16.0.0 - 172.31.255.255`
  – Класс C: `192.168.0.0 - 192.168.255.255`

  Эти адреса можно использовать для своих целей внутри своей сети. Другой человек при этом тоже может использовать те же адреса у себя. Конфликтов не происходит, потому что такие адреса не выходят в интернет. Перед выходом в интернет, пакеты проходят через маршрутизатор, который подменяет такой адрес на нормальный. При этом извне запрещено получать пакеты с таких адресов.

  Зачем это нужно? Если бы использовали обычные адреса из интернета в локальных сетях, маршрутизатор не мог бы понять, отправлять пакет в интернет или в локальную сеть.

- Адреса, связанные с групповой адресацией.

  Помимо индивидуальной и широковещательной адресации, существует групповая. Это когда данные отправляются не всем компьютерам сети, а определенной группе.

  – 224.0.0.1 – все узлы данной подсети, поддерживающие групповую адресацию
  – 224.0.0.2 – все маршрутизаторы данной подсети, поддерживающие групповую адресацию

- Адреса, зарезервированные для специальных протоколов.

  – 224.0.0.5 – все OSPF маршрутизаторы
  – 224.0.0.6 – все назначенные OSPF
  – 224.0.0.9 – все RIP-2 маршрутизаторы
  – 224.0.0.10 – все IGMP маршрутизаторы

1.2.4. Структуризация сетей IP

На практике оказалось, что крайне неудобно иметь только три класса сетей. Тогда решили разделить адрес сети на две части – сеть и подсеть. О том, как разделять данный адрес, говорит маска подсети.

Маска подсети – это 32-х разрядный вектор флагов, в котором ”1” на i-й позиции означает, что i-й разряд адреса относится к номеру сети или подсети, а ”0” – что разряд относится к номеру узла. Таким образом, зная IP адрес, класс сети и маску, мы можем восстановить адрес сети, подсети и узла.

Например, сети класса C можно делить на 2 подсети по 128 адресов, 4 подсети по 64 адреса, итд. При этом в каждой подсети два адреса (первый и последний) будут всё равно зарезервированы.

Рассмотрим пример. Пусть есть сеть класса C: `195.19.212.0 - 195.19.212.255`. Мы хотим ее разделить на 16 подсетей размера 16. Тогда маска будет `255.255.255.240`. Теперь посмотрим на последнюю подсеть. Она будет содержать адреса `195.19.212.240 - 195.19.212.255`. Получилось, что (последний) широковещательный адрес подсети совпал с широковещательным адресом всей сети. То же самое происходит с первым адресом первой подсети. В связи с этим в изначальном стандарте запретили использовать первую и последнюю подсети. Но сейчас почти все маршрутизаторы позволяют по маске понять, какой широковещательный адрес имеется в виду, и такие подсети используются.

Допускается делить сеть на подсети разных размеров, тогда маска подсети будет переменной длины (VLSM, Variable Length Subnet Mask).

Вообще говоря, единицы в маске могут располагаться на любых местах. Однако есть негласная договоренность, что сначала идут все единицы, а потом – все нули.

В связи с этой договоренностью, всевозможных масок всего 32. Поэтому их часто записывают в виде одного числа, означающего количество единичных битов в маске. В рассмотренном примере IP адрес сети с маской будет выглядеть так: `195.19.212.0/28`.

С помощью масок можно также объединять сети. Например, есть две подряд идущие сети класса C: `192.168.0.0/24` и `192.168.1.0/24`. Они отличаются последним битом третьего числа. Убрав его из маски, получим одну сеть, составленную из двух имеющихся: `192.168.0.0/23`

Иногда объединить ровно две сети не получается. Например, сети `192.168.1.0/24` и `192.168.2.0/24` хоть и идут подряд, но их адреса отличаются в двух последних битах. Чтобы их объединить, придется добавить сети `192.168.0.0/24` и `192.168.3.0/24`, и получится надсеть в четыре раза больше: `192.168.0.0/22`

Маска используется при маршрутизации, а также в каждом сетевом интерфейсе, чтобы либо пытаться отправить пакет напрямую в локальную сеть, либо искать маршрутизатор, чтобы тот отправил его в другую сеть.

А теперь немного битовой магии. Пусть есть IP адрес узла (IP) и маска подсети (M). Выразим другие атрибуты сети:

- Адрес всей сети: `Net = IP & M`
- Широковещательный адрес сети: `Broad = IP | !M`
- Максимальное количество узлов в сети: `K = !M - 1`

1.2.5. Адресация сервисов (приложений)

В TCP/IP приложения адресуются с помощью идентификатора, который называется **порт**.

Порт – это 16-ти разрядное число, уникальный номер приложения на узле, использующего конкретный транспортный протокол. То есть номера различных TCP и UDP портов могут совпадать.

Таким образом, приложение идентифицируется IP-адресом узла, на котором оно запущено, типом транспортного протокола и номером порта этого протокола. Тройка этих параметров называется **сокет**.

1.2.6. Сетевой уровень TCP/IP

Основные протоколы сетевого уровня TCP/IP:

- IP – основной сетевой протокол, с помощью него осуществляется доставка большинства пакетов в сети
- ICMP – протокол управления и контроля
- IGMP – протокол работы с группами
- ARP – протокол разрешения адресов
- Также есть различные протоколы маршрутизации

1.2.7. Протокол IP

IP – Internet Protocol, стандартизирован в 1981 году RFC 791, текущая действующая версия – 4 (IPv4). IP пакет состоит из заголовка и тела, суммарная длина пакета – до 64 Кбайт.

Рассмотрим подробнее формат IP пакета:

<table>
<tr><td colspan="4">V (4b)</td><td colspan="4">HL (4b)</td><td colspan="8">TOS (8b)</td><td colspan="16">Length (16b)</td></tr>
<tr><td colspan="16">ID (16b)</td><td colspan="3">F (3b)</td><td colspan="13">Offset (13b)</td></tr>
<tr><td colspan="8">TTL (8b)</td><td colspan="8">Protocol (8b)</td><td colspan="16">HCRC (16b)</td></tr>
<tr><td colspan="32">Source Address (32b)</td></tr>
<tr><td colspan="32">Destination Address (32b)</td></tr>
<tr><td colspan="24">Options</td><td colspan="8">Pad</td></tr>
<tr><td colspan="32">Data</td></tr>
</table>

IP пакет содержит следующие поля:

- V (Version) – версия протокола
- HL (Header Length) – длина заголовка в машинных словах. Длина может быть разная из-за опций (поле 'Options'), которых может быть разное количество
- TOS (Type of Services) – позволяет управлять трафиком разного типа
- Length – длина всего пакета в байтах
- ID, F (Flags), Offset – идентификатор, флаги и смещение. Используются при фрагментации
- TTL (Time To Live)– время жизни пакета в сети. Изначально задумывалось хранить время в виде целого количества секунд. Каждый маршрутизатор должен был уменьшать TTL на количество секунд, сколько он этот пакет обрабатывал. Но современные маршрутизаторы обрабатывают пакеты быстрее чем за секунду, поэтому каждый уменьшает TTL ровно на 1. Считается, что любые два узла сети находятся друг от друга на расстоянии не больше 32, то есть ставить значения больше 32 не имеет смысла.
- Protocol – номер протокола, который вложен в поле 'Data'
- HCRC – контрольная сумма заголовка

- Source Address – адрес отправителя
- Destination Address – адрес получателя
- Options – опции. Это поле может занимать от 0 до 10 машинных слов
- Pad – выравнивание. Дополняет поле опций, чтобы число разрядов было кратно 32
- Data – данные (тело пакета)

Теперь подробнее про эти поля.

Поле Protocol

Protocol – это идентификатор вложенного протокола. Список всех протоколов, которые можно вложить в IP, перечислен в RFC 1700. В Linux есть выборка из этого списка в файле `/etc/protocols`. В современных Windows это файл `%systemroot%\system32\drivers\etc\protocol`. Основные типы: ICMP, IGMP, IP, TCP, UDP, OSPF.

Поле TOS

TOS – тип сервиса. Задает пожелания относительно способа отправки.

Priority			D	T	R	C	–
0	1	2	3	4	5	6	7

Поле занимает 8 разрядов. Первые три задают приоритет. Следующие четыре – желаемое качество обслуживания.

Приоритетов бывает восемь. Чем выше приоритет, тем быстрее пакет будет обрабатываться маршрутизатором. На практике используются только четыре приоритета.

- 0 – нормальный. Почти все пакеты в сети имеют такой приоритет
- 1 – приоритетный. Используется редко
- 2 – немедленный. Не используется
- 3 – мгновенный. Не используется
- 4 – срочный. Не используется
- 5 – критический. Не используется
- 6 – межсетевое управление. Используется редко
- 7 – сетевое управление. Используется редко

Чтобы система приоритизации трафика работала правильно, надо чтобы все узлы на пути пакета ее поддерживали. Обычно на это рассчитывать не стоит.

Следующие разряды отвечают за пожелания по качеству передачи.

- D (Delay) – просим отправлять пакет по каналу с минимальной задержкой

- T (Throughput) – просим отправлять по каналу с максимальной пропускной способностью
- R (Reliability) – по самому надежному каналу
- C (Cost) – по каналу с минимальной стоимостью

Всего может быть выбрано не более одной характеристики из этих четырех. В интернете это всё игнорируется. Но в локальных или корпоративных сетях может поддерживаться. Ниже приведены примеры выставления характеристик для стандартных приложений.

Протокол	**D**	**T**	**R**	**C**
FTP управление	1	0	0	0
FTP данные	0	1	0	0
TFTP	1	0	0	0
DNS UDP	1	0	0	0
DNS TCP	0	0	0	0
TELNET	1	0	0	0
SMTP команды	1	0	0	0
SMTP данные	0	1	0	0
SNTP	0	0	1	0
NNTP	0	0	0	1

Фрагментация пакетов. Поля ID, F, Offset

Хотим передать пакет по какому-то каналу от одного маршрутизатора к другому. У канала есть характеристика MTU (Maximum Transmission Unit) – это максимальная длина кадра канального уровня. Если эта величина меньше чем размер нашего IP пакета, его приходится разбивать на несколько частей. Это называется **фрагментация**. Чтобы потом корректно собрать пакет воедино, используются поля ID, F и Offset.

ID – это идентификатор исходного пакета. Фрагменты с одинаковым ID будут собираться в один пакет.

Offset – смещение фрагмента относительно начала исходного пакета в 8-байтовых словах.

F – флаги. Это поле состоит из 3-х разрядов. Нулевой разряд зарезервирован, первый – флаг разрешения фрагментации. Если необходима фрагментация, а флаг не установлен, то пакет уничтожается, а отправителю кидается сообщение об ошибке. Последний разряд – это признак последнего фрагмента. Он равен 0 у последнего фрагмента и 1 у всех остальных.

Как работает фрагментация. В какой-то момент пакет разбился на фрагменты. У последнего установлен флаг, что он последний. Если в дальнейшем какой-то фрагмент надо будет разбить еще сильнее, то для его последнего подфрагмента установится флаг только если фрагмент был последним. Таким образом, этот флаг всегда установлен ровно у одного фрагмента. Затем на последнем маршрутизаторе, где пора склеивать фрагменты, приходит какой-то из фрагментов (не обязательно первый). В этот момент выделяется буфер и запускается таймер. Если в течение этого таймера все фрагменты успеют дойти, они успешно склеятся и передадутся на уровень выше. В противном случае буфер сбросится, и отправится сообщение об ошибке.

У IP есть большая проблема. Все поля в пакетах передаются в открытом виде. То есть любой узел на пути пакета знает, кто кому и что отправляет. Более того, кто угодно может подменить пакет или изменить адрес отправителя. Это влечет за собой большие проблемы с безопасностью. Все средства защиты реализуются уже на прикладном уровне.

1.2.8. Поле Options

Options – необязательные опции. Они позволяют добавить протоколу IP некие свойства, которых изначально не было. Это либо новые, либо тестовые опции. Поле Pad служит для выравнивания опций до границы 32-битного слова.

Каждая опция имеет следующий формат:

Копировать	Класс		Номер опции					параметры
0	1	2	3	4	5	6	7	...

Флаг копирования говорит, распространяется ли эта опция на все фрагменты пакета или только на один. Классов опции есть всего два: 0 – управление дейтаграммами или сетью и 2 – отладка сети. Номер опции идентифицирует опцию внутри класса.

Примеры опций:

Класс	**Номер**	**Длина в байтах**	**Описание**
0	0	1	End of Options List
0	1	1	No operation
0	2	11	Security
0	3	переменная	Loose Source Routing
0	7	переменная	Record Route
0	8	4	Stream ID
0	9	переменная	Strict Source Routing
2	4	переменная	Internet timestemp

Strict Source Routing – строгая маршрутизация от источника. Содержит адреса маршрутизаторов, через которые должен пройти пакет (и только через них). Маршрутизатор обязан передать пакет напрямую следующему узлу в списке.

Loose Source Routing – нестрогая маршрутизация от источника. Содержит адреса маршрутизаторов, через которые должен пройти пакет. Маршрутизатор должен передать пакет следующему узлу в списке (возможно через какие-то промежуточные узлы).

Record Route – записывает маршрут, по которому прошел пакет. Каждый маршрутизатор записывает себя в список. Если место в списке закончилось, маршрутизатор ничего не дописывает.

Stream ID – это технология коммутации третьего уровня. Она работает так – первый пакет в потоке маршрутизируется, то есть обрабатывается, строится маршрут, а дальнейшие пакеты из этого потока идут по одному и тому же маршруту.

End of Option List – конец списка опций. Используется для для выравнивания границы списка опций с границей заголовка IP.

No Operation – нет операции. Используется для выравнивания между опциями.

Глава 2

Практика

1. Простой TCP сервер

2.1.1. Напоминание о протоколах

Вспомним, что мы знаем про протоколы. Протокол – это соглашение, позволяющее двум сторонам понимать друг друга. Один протокол решает одну небольшую задачу. В реальных сетях решаются большие и сложные задачи, поэтому протоколы объединяются в стеки. Примеры задач, решаемых в сетях – адресация, надежная передача данных, шифрование, итп.

Мы будем говорить про стек протоколов TCP/IP. Есть и другие стеки, в основном они устаревшие. Как вариант стека TCP/IP, есть IPv6, про него мы тоже поговорим.

Какие протоколы есть в TCP/IP:

- IP – это протокол сетевого уровня, он отвечает за передачу данных между компьютерами в рамках всемирной паутины.
- TCP – протокол транспортного уровня, отвечающий за то, чтобы у нас были соединения, чтобы наши данные дошли до получателя, не потерялись и не испортились.
- UDP, ICMP, и другие – о них мы поговорим позже.

2.1.2. Напоминание о сокетах

Чтобы писать программы с использованием всех этих протоколов, нам нужны какие-то программные интерфейсы, с которыми мы будем взаимодействовать. Самый распространенный на данный момент интерфейс – BSD-сокеты. Они появились в BSD системах (разновидность UNIX), а затем их перенесли на другие UNIX-системы, на MacOS и в Windows. BSD-сокеты не привязаны к конкретным протоколам, и в этом их прелесть. Изначально программный интерфейс сокетов предназначался для языка си, а потом его сделали и для других распространенных языков.

2.1.3. Реализация TCP сервера на языке си

Рассмотрим простейший код TCP сервера на языке си и разберем интересующие нас места.

Сначала создается серверный сокет, который будет принимать входящие соединения.

```
1 sockfd = socket(AF_INET, SOCK_STREAM, 0);
2
3 if (sockfd < 0) {
4     perror("ERROR opening socket");
5     exit(1);
6 }
```

Функция `socket()` возвращает число – идентификатор соединения. В UNIX системах сетевой и файловый ввод-вывод объединены, то есть сокет также является и файловым дескриптором, и использовать его можно соответствующим образом.

У функции `socket()` три аргумента.

- `int domain` – определяет используемое семейство протоколов. `AF_INET` – протоколы IPv4.
- `int type` – определяет режим, в котором будет работать сокет и задает требования к конкретному протоколу, который будет использоваться. `SOCK_STREAM` говорит, что протокол должен поддерживать постоянное подключение. В случае IPv4 будет использоваться протокол TCP. Еще есть вариант `SOCK_DGRAM` – когда данные передаются пакетами. У нас это протокол UDP. Помимо этих, есть и другие режимы.
- `int protocol` – задает конкретный протокол. Обычно по значениям предыдущих аргументов протокол определяется однозначно. В этом случае сюда передается ноль. Если же есть несколько подходящих протоколов, то здесь задается, какой из них выбрать.

Далее обрабатываем возможные ошибки. Если сокет создать не удалось, функция `socket()` вернет -1. В этом случае мы вызываем функцию `perror()`, которая выводит код случившейся ошибки и текст, который мы передадим.

Теперь заполняем информацию о сетевом интерфейсе и порте, к которым будет привязан наш сокет и выполняем привязку, вызвав функцию `bind()`.

```
1  bzero((char *) &serv_addr, sizeof(serv_addr));
2  portno = 5001;
3
4  serv_addr.sin_family = AF_INET;
5  serv_addr.sin_addr.s_addr = INADDR_ANY;
6  serv_addr.sin_port = htons(portno);
7
8  if (bind(sockfd, (struct sockaddr *) &serv_addr, sizeof(serv_addr)) < 0) {
9      perror("ERROR on binding");
10     exit(1);
11 }
```

Переменная `serv_addr` типа `struct sockaddr_in` имеет следующие поля:

- `serv_addr.sin_family` – какому семейству протоколов соответствует адрес.
- `serv_addr.sin_addr.s_addr` – адрес сетевого интерфейса. Значение `INADDR_ANY` говорит, что мы хотим слушать все доступные интерфейсы. Привязываться к интерфейсу надо, например, для безопасности. Допустим, есть два интерфейса – один идет в локальную сеть, а другой – в интернет. И мы хотим получать только данные, приходящие по локальной сети.
- `serv_addr.sin_part` – номер порта, к которому привяжется сокет. Порт различает программы на конкретном компьютере. По нему мы можем понять, каким приложением обработать данные.

При передаче данных по сети всегда используется порядок байтов от старших к младшим (Big Endian). Но в некоторых процессорах используется обратный порядок (Little Endian). Если мы передали данные в одном порядке, а прочитали в другом, ничего хорошего мы не получим. Функции `htons()` и `htonl()` преобразуют порядок байтов из локального в сетевой для типов `short` и `long` соответственно. Функции `ntohs()` и `ntohl()` делают обратное преобразование.

Вызов `bind()` можно не делать, тогда ОС это сделает за нас и подставит дефолтные значения. Для интерфейса – `INADDR_ANY`, для порта – случайный свободный номер.

Далее надо перевести сокет в слушающее состояние и принять входящее соединение.

```
listen(sockfd, 5);

newsockfd = accept(sockfd, (struct sockaddr *) &cli_addr, &clilen);

if (newsockfd < 0) {
    perror("ERROR on accept");
    exit(1);
}
```

Функция `listen()` принимает сокет и максимальное количество входящих соединений, которые одновременно могут находиться в очереди. Функция `accept()` ждет следующего клиента и возвращает новый сокет, через который будет происходить непосредственно общение с ним. Последним аргументом `accept()` передается указатель на переменную, куда запишется размер структуры переменной `cli_addr`.

В переменную `clilen` записывается длина структуры данных `cli_addr`. Это своего рода имитация наследования. Так как сокет не привязан к конкретному стеку протоколов, в структуре `struct sockaddr` приходится хранить разный набор полей. Значение `clilen` показывает, сколько памяти занимает конкретный экземпляр этой структуры.

Затем мы читаем данные из клиентского сокета функцией `read()`.

```
bzero(buffer, 256);
n = read(newsockfd, buffer, 255); // recv on Windows
if (n < 0) {
    perror("ERROR reading from socket");
    exit(1);
}
```

Функция `read()` принимает файловый дескриптор сокета, указатель на буфер и количество байтов, которые надо прочитать.

После обработки данных мы отправляем ответ функцией `write()`.

```
n = write(newsockfd, "I got your message", 18); // send on Windows
if (n < 0) {
    perror("ERROR writing to socket");
    exit(1);
}
```

Код клиента оставляется для самостоятельного изучения, там всё примерно так же, за небольшим исключением. Чтобы подключиться к серверу нам надо узнать его IP адрес. Но чаще всего мы знаем только доменное имя. Чтобы узнать адрес домена, мы вызываем функцию `gethostbyname()`.

```
struct hostent *server;
// получаем хост через параметры командной строки:
server = gethostbyname(argv[1]);
bzero((char *) &serv_addr, sizeof(serv_addr));
serv_addr.sin_family = AF_INET;
// копируем IP адрес хоста в соответствующее поле serv_addr:
bcopy(server->h_addr, (char *)&serv_addr.sin_addr.s_addr, (size_t)server->h_length);
serv_addr.sin_port = htons(portno);
```

Проблема `gethostbyname()` в том, что на многих операционных системах она не работает с IPv6. Альтернативный способ получения адреса – функция `getaddrinfo()`.

2.1.4. Проблемы предложенной реализации

Во-первых, наш сервер обрабатывает только одно сообщение от одного клиента и сразу выключается. Чтобы поддержать несколько клиентов и несколько сообщений, надо для каждого клиента создавать поток, который в цикле принимает от него сообщения и отправляет ответ.

Во-вторых, здесь мы не закрываем сокет, а надо бы.

Следующая проблема в том, что мы работаем с TCP, который передает данные в виде потока, не указывая границы пакетов. В результате, когда мы пытаемся прочитать данные из сокета, мы можем либо что-то не дочитать, либо попытаться прочитать что-то лишнее и зависнуть в ожидании. Самый яркий пример – это передача строки, длину которой мы заранее не знаем.

Для решения этой проблемы есть два основных подхода. Первый – это сначала передать размер данных, а потом сами данные. Но размер данных может сильно меняться, из-за чего может меняться и размер переменной, содержащий размер этих данных. Приходится заранее ограничивать максимальный размер сообщения, что тоже не всегда хорошо. Второй подход – использовать терминирующий символ, который заведомо не может встретиться в самом сообщении.

Еще одна проблема. Может быть такое, что программа, работающая с сокетом, некорректно завершилась, сокет при этом какое-то время остается в памяти операционной системы. Мы перезапускаем программу и получаем ошибку вида "Socket error: Address already in use". Никакого способа залезть в ядро и удалить оттуда сокет нет. Однако можно заранее предусмотреть такую ситуацию и использовать флаг `SO_REUSEADDR`. Чтобы этот флаг поставить, надо вызвать функцию `setsockopt()`. После этого система будет корректно работать с несколькими сокетами, слушающими один и тот же адрес.

Также есть флаг `SO_REUSEPORT`. Он нужен, чтобы запускать несколько серверных приложений параллельно. В этом случае ОС старается раскидывать запросы на установление соединения равномерно между этими приложениями.

Чтобы корректно закрыть сокет, надо вызвать `close()` или `shutdown()`. При вызове `close()` другая сторона может не узнать о том, что соединение закрылось, поэтому лучше использовать `shutdown()`. К тому же `shutdown()` позволяет закрыть соединение как в обе стороны, так и в одну (только чтение или только запись).

2.1.5. Бинарные vs Текстовые протоколы

Различные структуры данных можно передавать в сериализованном виде (в виде байтов) или в виде текстового описания. В чем их преимущества и недостатки?

Преимущества бинарных протоколов – простота и небольшой объем. Мы просто берем структуру в том виде, как она хранится в памяти, и отправляем. Но есть и недостатки. Во-первых, при хранении может применяться выравнивание, а может не применяться. Во-вторых, если структура содержит какие-то указатели, то нам недостаточно будет просто их передать, потому что нас интересуют не адреса, а значения, которые там расположены.

2.1.6. Блокирующая vs неблокирующая архитектура сервера

Если серверу приходится выполнять какие-то сложные вычисления, лучше делать его блокирующим, неблокирующий вариант лучше подходит когда задания не очень большие. Но в общем случае нельзя заранее сказать, что выбрать, можно только реализовать и сравнить.

Объяснение на паре было не очень внятное, см. конспект по Java.