



Программирование на Java

Лекция 16. Введение в сетевые протоколы

20 апреля 2003 года

Авторы документа:

Николай Вязовик (Центр Sun технологий МФТИ) <vyazovick@itc.mipt.ru>

Евгений Жилин (Центр Sun технологий МФТИ) <gene@itc.mipt.ru>

Copyright © 2003 года [Центр Sun технологий МФТИ, ЦОС и ВТ МФТИ](#)[®], Все права защищены.

Аннотация

Завершает курс лекция, в которой рассматриваются возможности построения сетевых приложений. Сначала дается краткое введение в сетевые протоколы, семиуровневую модель OSI, стек протоколов TCP/IP и описываются основные утилиты, предоставляемые операционной системой для мониторинга сети. Эти знания необходимы, поскольку библиотека `java.net` по сути является интерфейсом для работы с этими протоколами. Рассматриваются классы для соединений через высокоуровневые протоколы, протоколы TCP и UDP.

Оглавление

Лекция 16. Введение в сетевые протоколы.....	1
1. Основы модели OSI.....	2
2. Physical layer (layer 1).....	4
3. Data layer (layer 2).....	8
3.1. LLC sublayer.....	9
3.2. MAC sublayer.....	9
4. Network layer (layer 3).....	10
4.1. Class A.....	11
4.2. Class B.....	12
4.3. Class CClass DClass E.....	12
5. Transport layer (layer 4).....	13
5.1. TCP.....	14
5.2. UDP.....	14
6. Session layer (layer 5).....	15
7. Presentation layer (layer 6).....	15
8. Application layer (layer 7).....	15
9. Утилиты для работы с сетью.....	16
9.1. IPCONFIG (IFCONFIG).....	17
9.2. ARP.....	18
9.3. Ping.....	18
9.4. Traceroute.....	19
9.5. Route.....	21
9.6. Netstat.....	22
9.7. Задания для практического занятия.....	23
10. Пакет java.net.....	24
11. Заключение.....	32
12. Контрольные вопросы.....	32

Лекция 16. Введение в сетевые протоколы

Содержание лекции.

1. Основы модели OSI.....	2
2. Physical layer (layer 1).....	4
3. Data layer (layer 2).....	8
3.1. LLC sublayer.....	9
3.2. MAC sublayer.....	9
4. Network layer (layer 3).....	10
4.1. Class A.....	11
4.2. Class B.....	12
4.3. Class CClass DClass E.....	12
5. Transport layer (layer 4).....	13
5.1. TCP.....	14
5.2. UDP.....	14
6. Session layer (layer 5).....	15
7. Presentation layer (layer 6).....	15
8. Application layer (layer 7).....	15
9. Утилиты для работы с сетью.....	16
9.1. IPCONFIG (IFCONFIG).....	17
9.2. ARP.....	18
9.3. Ping.....	18
9.4. Traceroute.....	19
9.5. Route.....	21
9.6. Netstat.....	22
9.7. Задания для практического занятия.....	23
10. Пакет java.net.....	24
11. Заключение.....	32

12. Контрольные вопросы..... 32

1. Основы модели OSI

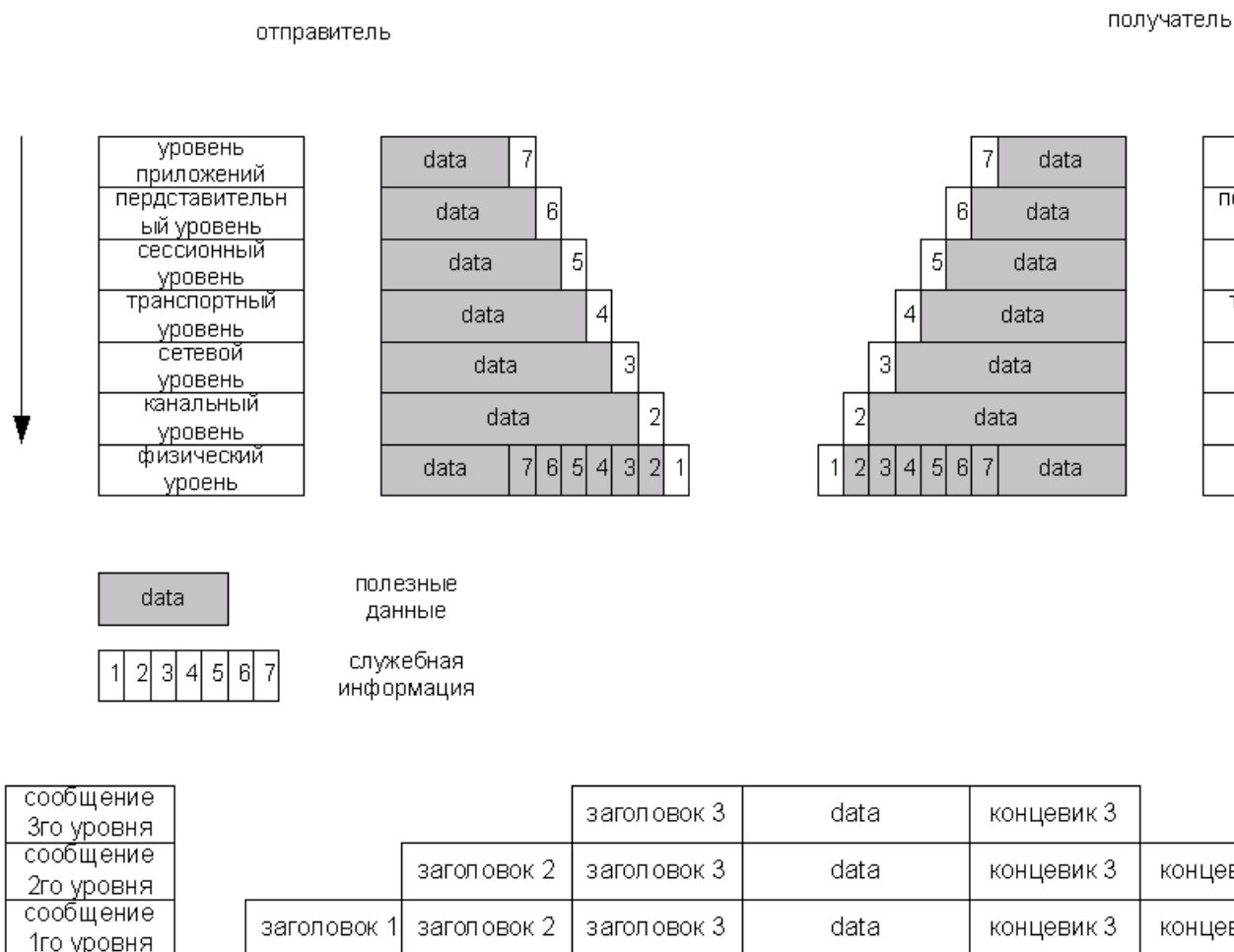
В течение последних нескольких десятилетий размеры и количество сетей значительно выросли. В 80-х годах имелось множество типов сетей. И практически каждая из них была построена на своем типе оборудования и программного обеспечения, зачастую не совместимых между собой. Это приводило к значительным трудностям при попытке соединить несколько различных типов сетей (например, различный тип адресации делал эти попытки практически безнадежными). Эта проблема была рассмотрена Всемирной Организацией по Стандартам (International Organization for Standardization, ISO), и было принято решение разработать модель сети, которая могла бы помочь разработчикам и производителям сетевого оборудования и программного обеспечения работать сообща. В результате в 1984 г. была разработана модель OSI - модель взаимодействия открытых систем (Open Systems Interconnected). Эта модель состоит из семи уровней. Схематично ее можно представить следующим образом:

Номер уровня	Название уровня	Единица информации
Layer 7	Уровень приложений	Данные (data)
Layer 6	Представительский уровень	Данные (data)
Layer 5	Сессионный уровень	Данные (data)
Layer 4	Транспортный уровень	Сегмент (segment)
Layer 3	Сетевой уровень	Пакет (packet)
Layer 2	Уровень передачи данных	Фрейм (frame)
Layer 1	Физический уровень	Бит (bit)

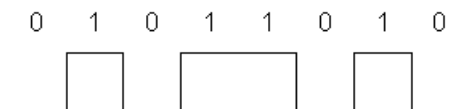
Хотя на сегодняшний день существуют разнообразные модели сети, большинство разработчиков придерживается именно этой общепризнанной схемы. OSI-модель позволяет разделить сетевые функции на уровни и понять, как происходит обмен данными по сети. Каждый уровень взаимодействует только с соседними уровнями, протокол взаимодействия стандартизован. Это позволяет использовать реализации сетевого и программного обеспечения от разных производителей в различных комбинациях. Например, протоколы высокого уровня (HTTP, FTP) не зависят от физических параметров используемой сети.

Рассмотрим процесс передачи информации между двумя компьютерами. Программное обеспечение формирует сообщение на уровне 7 (приложений), состоящее из заголовка и полезных данных. В заголовке содержится служебная информация, которая необходима уровню приложений адресата для обработки требуемой информации (например, это может быть информация о файле, который необходимо передать, или операции, которую необходимо выполнить). После того, как сообщение было сформировано, уровень приложений направляет его вниз на представительский уровень (layer 6). Полученное сообщение, состоящее из служебной информации уровня 7 и полезных данных, для уровня 6 представляется как одно целое сообщение (хотя уровень 6 может считывать служебную информацию уровня 7). Протокол представительского уровня выполняет требуемые действия на основании данных, полученных из заголовка уровня приложений, и добавляет заголовок своего уровня, в котором содержится необходимая информация для соответствующего (6-го) уровня адресата. Полученное в результате сообщение передается далее вниз сеансовому уровню, где также добавляется служебная информация. Дополненное сообщение передается на следующий транспортный уровень и т.д., на каждом последующем уровне (схематично это представлено на рис.1). При этом служебная

информация не обязательно добавляется в начало сообщения. Например, на 3-м уровне служебная информация добавляется к началу и концу сообщения (рис.2). В итоге получается сообщение, содержащее служебную информацию всех 7 уровней. Процесс добавления служебной информации называется инкапсуляцией (encapsulation).



Далее это сообщение передается через сеть в виде битов. Бит - это минимальная порция информации, которая может принимать значение 0 или 1. Таким образом, все сообщение кодируется в виде набора нулей и единиц, например 010110101. В простейшем случае на физическом уровне для передачи формируется электрический сигнал, состоящий из серии электрических импульсов (0 - нет сигнала, 1 - есть сигнал). Именно эта единица принята для измерения скорости передачи информации. Современные сети обычно предоставляют каналы с производительностью в десятки и сотни Кбит/с и Мбит/с.



Получатель на физическом уровне получает сообщение в виде электрического сигнала (рис.3). Далее происходит процесс обратный инкапсуляции - декапсуляция. На каждом

уровне происходит разбор служебной информации. После декапсуляции сообщения на первом уровне (считывания и обработки служебной информации 1го уровня), это сообщение, содержащее служебную информацию второго уровня и данные в виде полезных данных и служебной информации вышестоящих уровней, передается на следующий уровень. На канальном (2-м) уровне снова происходит анализ системной информации, и сообщение передается на следующий уровень. И так до тех пор, пока сообщение не дойдет до уровня приложений, где в виде конечных данных передается принимающему приложению. В качестве примера можно привести обращение браузера к веб-серверу. Приложение клиента - браузер, формирует запрос для получения веб-страницы. Этот запрос передается приложением на уровень 7 и далее последовательно на каждый уровень модели OSI. Достигнув физического уровня, наш первоначальный запрос "обрастает" служебной информацией каждого уровня. После этого он передается по физической сети (кабелям) на сервер в виде электрических импульсов. На сервере происходит разбор соответствующей системной информации на каждом уровне, в результате чего посланный запрос достигает процесса веб-сервера, где обрабатывается и после обработки клиенту отправляется ответ. Процесс отправки ответа аналогичен посылке запроса - за исключением того, что сообщение посылает сервер, а клиент его получает.

Вместе с названием сообщение (message) в стандартах ISO для обозначения единицы данных используют термин протокольный блок данных (Protocol Data Unit, PDU).

Для более глубокого понимания принципов работы сети рассмотрим каждый уровень по отдельности.

2. Physical layer (layer 1)

Как видно из общей схемы расположения уровней в модели OSI - физический уровень (physical layer) самый первый. Этот уровень описывает среду передачи данных. Стандартизируются физические устройства, отвечающие за передачу электрических сигналов (разъемы, кабели и т.д.) и правила формирования этих сигналов. Рассмотрим по порядку все составляющие этого уровня.

Большая часть сетей строится на кабельной структуре (хотя также существуют сети, основанные на передаче информации с помощью, например, радиоволн). На сегодняшний день существуют различные типы кабелей для передачи информации. Наиболее распространенные из них:

- телефонный провод;
- коаксиальный кабель;
- витая пара;
- оптоволокно.

Телефонный кабель начал использоваться для передачи данных со времен первых компьютеров. Главным преимуществом использования телефонных линий - использование существующих линий связи для передачи информации. При использовании телефонных линий можно передавать данные между компьютерами, находящихся на разных материках (также как и передача голоса между людьми, удаленных друг от друга на многие тысячи километров). На сегодняшний день использование телефонных линий также остается популярным. Большинство пользователей, которых устраивает небольшая скорость

передачи данных, могут получить доступ к Интернету со своих домашних компьютеров. Основными недостатками использования телефонного кабеля при передаче данных является небольшая скорость передачи, т.к. соединение происходит не напрямую, а через телефонные станции. При этом требование к качеству передаваемого сигнала при передаче данных значительно выше, чем при передаче "голоса". А т.к. большинство аналоговых АТС не справляется с этой задачей (уровень "шума" или помех и качество сигнала оставляет желать лучшего), то скорость передачи данных очень низкая. Хотя при подключении к современным цифровым АТС можно получить высокую и надежную скорость связи.

Коаксиальный кабель использовался в сетях несколько лет назад, но сегодня встретить сеть, использующую этот кабель, очень сложно. Этот тип кабеля по строению практически идентичен обычному телевизионному коаксиальному кабелю - центральная медная жила отделена слоем изоляции от оплетки, за исключением электрических характеристик (в телевизионном кабеле используется кабель с волновым сопротивлением 75 Ом, в сетевом коаксиальном кабеле - 50 Ом).

Основными недостатками этого кабеля является низкая скорость пропускания (до 10 Мбит/с), подверженность воздействиям внешних помех. Кроме того, подключение компьютеров в таких сетях происходит параллельно, а значит, скорость максимальная возможная скорость пропускания делится на всех пользователей. Но по сравнению с телефонным кабелем коаксиальный кабель позволяет объединять близко расположенные компьютеры, скорость передачи данных и качество связи намного лучше при использовании коаксиального кабеля вместо телефонного.

Витая пара ("twisted pair") - наиболее распространенное средство для передачи данных между компьютерами. В данном типе кабеля используется медный провод, попарно скрученный, что позволяет уменьшить количество помех и наводок при передаче сигнала по самому кабелю, так и при воздействии внешних помех. Существует несколько категорий этого кабеля. Основные из них: Cat 3 - был стандартизирован в 1991 г, электрические характеристики позволяли поддерживать частоты передачи до 16 МГц, использовался для передачи данных и голоса. Более высокая категория, Cat 5 была специально разработана для поддержки высокоскоростных протоколов. Поэтому электрические характеристики лежат в пределах до 100 МГц. На таком типе кабеля работают протоколы передачи данных 10, 100, 1000 Мбит/с. На сегодняшний день кабель Cat 5 практически вытеснил кабель Cat 3. Основным преимуществом витой пары перед телефонными и коаксиальными кабелями - более высокая скорость передачи данных при использовании одного и того же кабеля. Также использование кабеля Cat 5 в большинстве случаев позволяет, не меняя кабельную структуру, повысить производительность сети.

Оптическое волокно используется для соединения больших сегментов сети, которые располагаются далеко друг от друга, или в сетях, где требуется большая полоса пропускания, помехоустойчивость. Оптический кабель состоит из центрального проводника света (сердцевины) - стеклянного волокна, окруженного другим слоем стекла - оболочкой, обладающей меньшим показателем преломления, чем сердцевина. Распространяясь по сердцевине, лучи света не выходят за ее пределы, отражаясь от покрывающего слоя оболочки. Световой луч обычно формируется полупроводниковым или диодным лазером. В зависимости от распределения показателя преломления и от величины диаметра сердечника различают:

- одномодовое волокно;
- многомодовое волокно

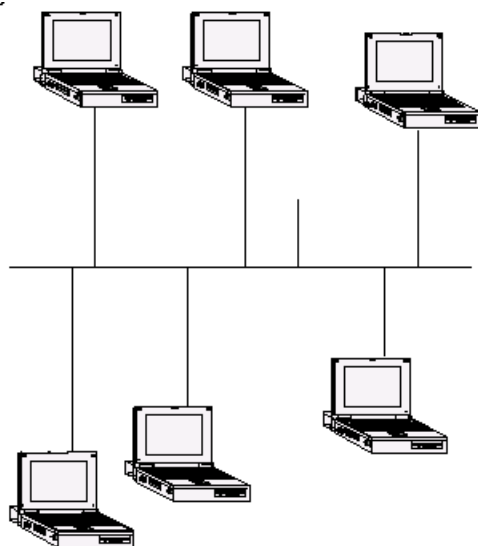
Понятие "мода" описывает режим распространения световых лучей в сердечнике кабеля. В одномодовом кабеле используется проводник очень малого диаметра, соизмеримого с длиной волны света. В многомодовом кабеле используются более широкие сердечники, которые легче изготовить технологически. В этих кабелях в сердечнике одновременно существуют несколько световых лучей, отражающихся от оболочки под разными углами. Угол отражения луча называется модой луча. Оптоволоконно обладает следующими преимуществами: устойчивы к электромагнитным помехам, высокие скоростные характеристики на больших расстояниях. Основным недостатком является как дороговизна самого кабеля, так и трудоемкость монтажных работ т.к. все работы выполняются на дорогостоящем высокоточном оборудовании.

Физический уровень также отвечает за преобразование сигналов между различными средами передачи данных. Например, при необходимости соединить сегмент сети, построенной на оптоволокне и витой паре применяют т.н. конверторы (в данном случае они преобразуют световой импульс в электрический).

Сетевой адаптер - устройство, позволяющее обмениваться наборами битов, представленными электрическими сигналами. Сетевой адаптер (сетевая карта, англ. Network adapter) обычно имеет шину для подключения в компьютер ISA или PCI, и соответствующий разъем для подключения к среде передачи данных (например, для витой пары, коаксиала и т.п.).

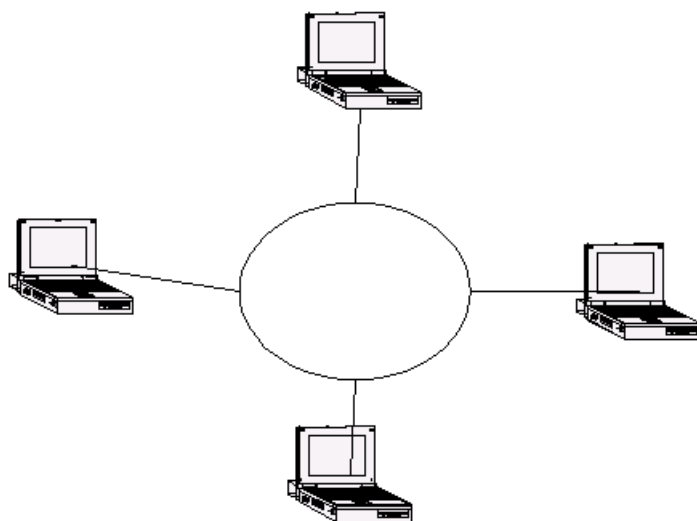
Теперь, когда известно с помощью чего происходит соединение компьютеров в одну сеть, рассмотрим физическую схему соединения компьютеров или другими словами - физическую топологию (структуру локальной сети).

Топология "шина"(bus):



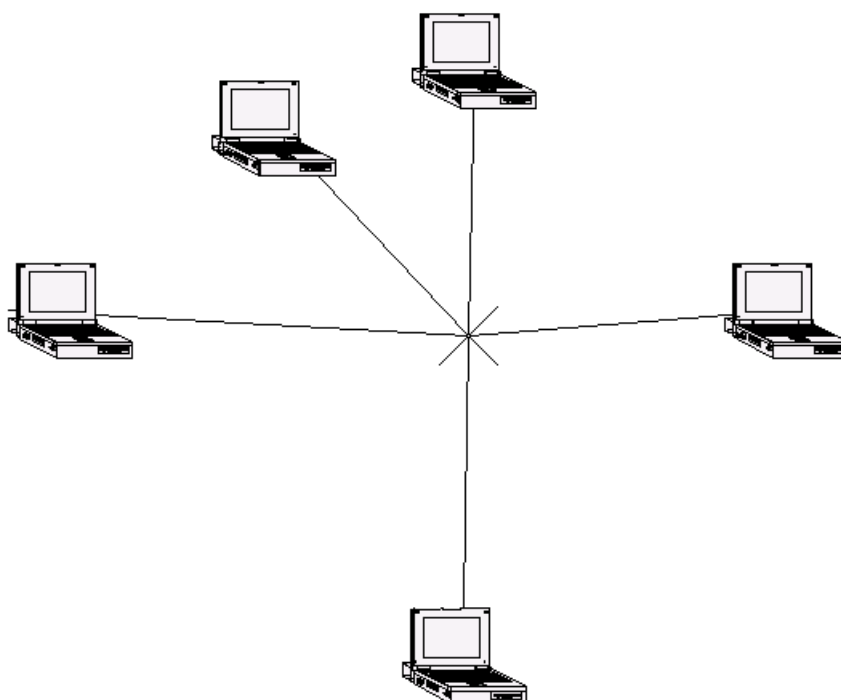
Все компьютеры и сетевые устройства подсоединены к одному проводу, и фактически они напрямую соединены между собой.

Топология "кольцо"(ring):



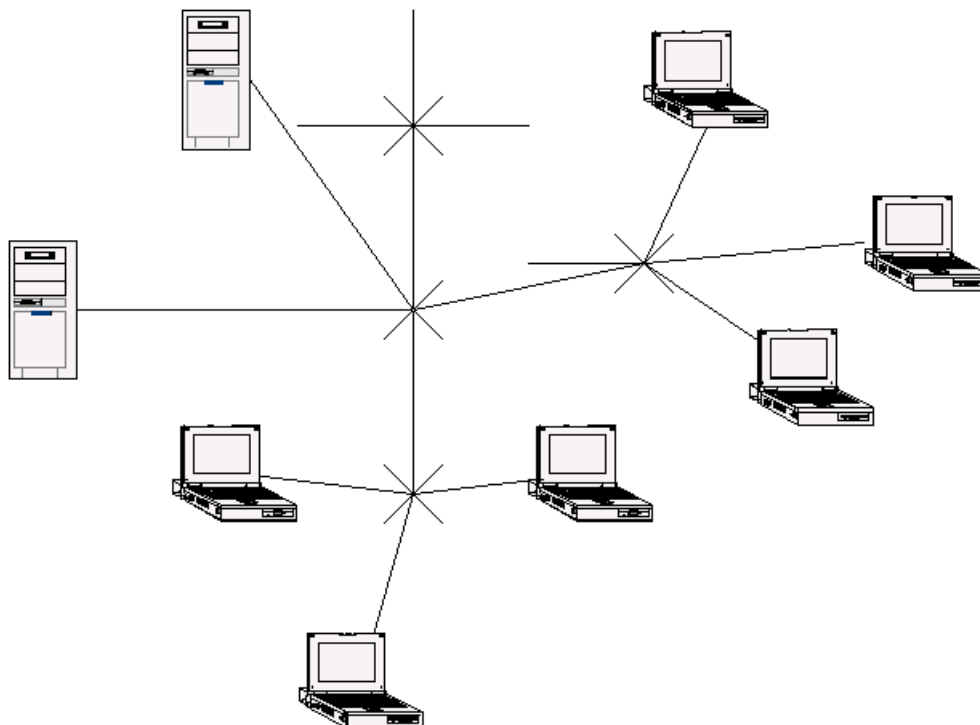
Кольцо состоит из сетевых устройств и кабелей между ними, образующих одно замкнутое кольцо.

Топология "звезда"(star):



Все компьютеры и сетевые устройства подключены к одному центральному устройству.

Топология "расширенная звезда"(extended star):



Такая схема практически аналогична топологии "звезда" за одним исключением. Каждое устройство соединено с локальным центральным устройством, а оно в свою очередь соединено с центром другой "звезды".

3. Data layer (layer 2)

На физическом уровне пересылаются просто набор сигналов - битов. При этом не проверяется, что несколько компьютеров могут в одну среду передачи данных одновременно передавать информацию в виде битов. Поэтому одной из задач канального уровня является проверка доступности среды передачи. Также канальный уровень отвечает за доставку фреймов между источником и адресатом в пределах сети с одной топологией. Для обеспечения такой функциональности Data layer разделяют на два подуровня:

- LLC sublayer
- MAC sublayer

LLC отвечает за переход со второго уровня на более высший - 3й сетевой уровень.

MAC отвечает за передачу данных на более низкий уровень - physical layer.

Рассмотрим эти подуровни более подробно.

3.1. LLC sublayer.

Этот подуровень был создан для независимости от существующих технологий. Он обеспечивает передачу данных на сетевой (3-й) уровень вне зависимости от физической среды передачи данных. LLC получает данные с сетевого уровня, добавляет в них служебную информацию и передает пакет для последующей инкапсуляции и передачи для требуемой технологии. Например, это может быть Ethernet, Token Ring, Frame Relay.

3.2. MAC sublayer.

Этот подуровень обеспечивает доступ к физическому уровню. Как уже говорилось, data layer обеспечивает идентификацию компьютеров в сети. Т.е. у каждого компьютера на data layer есть уникальный адрес, который еще иногда называют физическим адресом или MAC-адресом.

Этот адрес прошит в энерго-независимую память сетевой карточки и задается производителем. Длина MAC-адреса 48 бит или 6 байт (каждый байт состоит из 8 бит), которые записываются в шестнадцатеричном формате. Первые 3 байта называются OUI. OUI - Organizational Unique Identifier, Организационный Уникальный Идентификатор - назначается IEEE (Institute of Electrical and Electronic Engineers, Институт инженеров по электротехнике и радиоэлектронике - международная организация, подготавливающая стандарты и спецификации) и обозначает производителя сетевой карты. Остальные 3 байта описывают идентификационный номер самой сетевой карты. Записываться физический адрес может в разных форматах, например: 00:00:B4:90:4C:8C, 00-00-B4-90-4C-8C, 0000.B490.4C8C - это зависит от производителя программного обеспечения.

Рассмотрим, например, адрес 0000.1c12.3456. Здесь 00001c - идентификатор производителя, а 12.3456 - идентификатор сетевой карты.

Такая система адресов гарантирует, что в сети не будет двух компьютеров с одинаковыми физическими адресами.

Рассмотрим более подробно процесс передачи данных на data layer в сети Ethernet. Один компьютер посылает данные другому, используя свой MAC-адрес и MAC-адрес получателя.

Каждый компьютер, получивший это сообщение, проверяет, кому он был адресован. Если MAC-адрес в фрейме и MAC-адрес получившего этот фрейм совпадают, то пакет принимается и передается на вышестоящий уровень для дальнейшей обработки. Если же адрес в пакете не совпадает с адресом сетевой карты, то такой пакет отбрасывается. Если отправитель хочет, чтобы его сообщение получили все узлы локальной сети, он отправляет пакет с MAC-адресом получателя в виде FF-FF-FF-FF-FF-FF. Этот адрес используется для широковещания (broadcast), которое примут все сетевые устройства и передадут на следующий уровень.

В сетях Ethernet используется метод разделения среды передачи данных - метод CSMA/CD (carrier sense multiply access/collision detect). Этот метод применяется в сетях с логической общей шиной. Все компьютеры такой сети имеют доступ к общей шине, поэтому она может быть использована для передачи данных между двумя любыми узлами сети. Одновременно все компьютеры сети имеют возможность практически немедленно получить данные, которые любой из компьютеров начал передавать на общую шину. Технология CSMA/CD так же позволяет уменьшать общее кол-во столкновений пакетов (collision - например, когда два компьютера начинают одновременно передавать данные в одну общую шину).

Рассмотрим устройства, используемые для построения сетей в разных топологиях.

При использовании топологии построения сети шина ("bus") все компьютеры посылают фреймы на физический уровень, а дальше по среде передачи данных (например, кабелю) другим компьютерам. Т.е. все устройства, подсоединенные к одной среде, получают все пакеты, которые проходят по сети. Специальных устройств для построения такой сети не используется.

При построении сети на основе топологии Token Ring используется другой принцип. Физически сеть представляет собой замкнутое кольцо. В отличие от сети, работающей на основе Ethernet, здесь используется передача данных по очереди. Т.е. вычисляется, сколько времени может передавать данные одна станция, по истечении этого времени данные начинает передавать другая рабочая станция и т.д.

При построении сети на основе технологии "звезда" нужно использовать кроме сетевых карт в компьютере дополнительное сетевое оборудование в центре, куда подключаются все "лучи звезды". Типичным примером является концентратор (hub). Подключение происходит с помощью кабеля "витая пара". Все компьютеры подключаются к концентратору. Используемая технология Ethernet позволяет снизить количество коллизий с помощью CSMA/CD. Подключенный компьютер посылает данные. Коммутатор транслирует этот фрейм на все порты. Дальше получатель проверяет, кому был послан фрейм и если MAC получателя совпадает с MAC-адресом, то пакет принимается, если нет - отбрасывается. Недостатком концентратора является то, что пользователи сети могут "прослушивать" чужой трафик (в том числе перехватить пароль, если он передается в открытом виде). Также общая максимальная скорость делится на всех подключенных в концентратор. И если скорость передачи данных используется 10 Мбит/с, то в среднем на каждого конкретного пользователя приходится около 2 Мбит/с.

Более дорогим, но и более производительным решением, является использование коммутатора (switch). Коммутатор в отличие от концентратора имеет таблицу MAC-address - Port. Он смотрит у всех фреймов адреса отправителя и получателя. Далее при прохождении фрейма коммутатор просматривает адрес получателя, и если он знает, какому порту соответствует адрес получателя, то он посылает его на этот порт. Если адрес получателя коммутатору не известен, то он отправляет фрейм на все порты, кроме того, с которого этот пакет пришел. Таким образом, получается, что если два компьютера обмениваются данными между собой, то они не забивают своими пакетами другие порты и соответственно их пакеты практически невозможно "подслушать".

4. Network layer (layer 3)

В предыдущей главе мы рассмотрели второй уровень в модели OSI. Одним из ограничений 2-го уровня является использование "плоской" модели адресации. При попытке построить большую сеть, используя для идентификации компьютеров MAC-адреса, мы получим огромное количество broadcast-трафика. Протокол, который поддерживается сетевым уровнем, использует иерархическую структуру для уникальной идентификации компьютеров.

Для примера представим себе телефонную сеть. Она также имеет иерархическую адресацию. Например, в номере +7-095-101-12-34 первая цифра обозначает код страны, далее идет код области/города (095), а затем указывается сам телефон (101-12-34). Последний номер также является составным. 101 - это код станции, куда подключен телефон, а 12-34 определяет местоположение телефона. Благодаря такой иерархической

структуре мы можем определить расположение требуемого абонента с наименьшими затратами. Иерархическая адресация для сети также должна позволять передавать данные между разрозненными и удаленными сетями. На сетевом уровне существует несколько протоколов, которые позволяют передавать данные между сетями - IP, IPX. Наиболее распространенным протоколом на сегодняшний день является IP. IPX же практически не используется в публичных сетях, но его можно найти в частных, закрытых сетях.

Устройства, работающие на 3-м уровне, называют роутерами (router). Они соединяют удаленные сети, объединяют территориальные сети (LAN) в глобальные (WAN). Роутер получает пакет с локального устройства или компьютера (в дальнейшем будем просто называть LAN) и смотрит заголовок 3го уровня. На основании полученной информации с 3го уровня роутер принимает решение, что делать с пакетом. Основная задача роутера - выбор пути, по которому нужно передать пакет. Т.к. у нас может существовать множество связей между двумя сетями - еще одна задача роутера выбрать наиболее оптимальный путь для прохождения пакета. Выбор роутером следующего узла сети (следующего hop'a) для доставки его получателю называется "routing the packet". Выбор "next hop", по которому роутер перешлет пакет, может зависеть от многих факторов - загрузки сети, наименьшего пути до получателя, стоимости трафика по различным маршрутам и т.д.

Адресация на сетевом уровне дает возможность роутеру определить путь для доставки пакета получателю через глобальные сети. Было разработано и сейчас существует несколько протоколов сетевого уровня. Один из основных и наиболее распространенный - протокол IP. Рассмотрим более реализацию этого протокола.

При прохождении данных с верхних уровней на нижние на сетевом уровне к ним добавляется дополнительные данные сетевого уровня. В заголовке IP-пакета содержится необходимая для дальнейшей передачи информация - адрес отправителя, адрес получателя (кроме адресов там также содержится служебная информация). Рассмотрим более подробно понятие IP-адрес. IP-адрес представляется 32-х битным бинарным числом, разбитым на 4 части по восемь бит каждая. Логически его разбивают на две части - network и host. Компьютеры с одинаковой частью network IP-адреса в сети могут легко передавать данные между собой, но если они имеют различные network-ID, то даже если они находятся в одном физическом сегменте, обычно они не могут "увидеть" друг друга.

Сетевая часть IP-адреса показывает принадлежность сетевого адреса - какой сети принадлежит адрес. Хост (host) идентифицирует сетевое устройство в этой сети. Т.к. сетевой адрес состоит из 4-х октетов, один, два или три первых октета могут использоваться для определения сетевого адреса. Подобным образом до 3-х октетов может быть использовано для определения host-части. Существует вообще пять классов IP-адресов. Три из них используются в открытых сетях по всему миру (class A, class B, class C).

Class A	N	H	H	H
Class B	N	N	H	H
Class C	N	N	N	H

4.1. Class A

В классе A используется для определения принадлежности адреса к сети первый октет, остальные для определения адреса хоста. Если перевести бинарное число сетевой части в десятичное - то мы получим что адрес сети класса A может быть в диапазоне 0-126(127)

адрес зарезервирован для специального использования). Остальные свободные три октета используются для задания адреса хоста в данной сети. В одной сети может быть использовано до 2^{24} адресов (за исключением двух) - получается 16 777 214 возможно использовать в одной сети класса А.

Диапазон адресов 10.0.0.0-10.255.255.255 не используется в публичных сетях. Эти адреса специально зарезервированы для использования в локальных сетях и не обрабатываются глобальными маршрутизаторами.

4.2. Class B

В сети класса В используются первые два октета для определения сети, последние два октета - для определения адреса хоста. Диапазон сети класса В может быть с 128 до 191 (исходя из первых двух октетов). В каждой сети класса В может быть не более 65534 адресов - 216 (за исключением двух адресов).

В этой подсети зарезервированными для локального использования являются следующие адреса: 172.16.0.0-172.31.0.0.

4.3. Class C Class D Class E

Диапазон сети класса С определяется первыми тремя октетами. И в десятичном виде эта сеть может начинаться с 192 по 223. Для определения адреса хоста используется последний октет. Таким образом, в сети класса С может быть использовано 28(без двух адресов) или 254 адреса

Зарезервированными для локального использования являются следующие адреса: 192.168.0.0-192.168.255.255.

Этот класс используется для multicast-группы. Диапазон адресов – 224.0.0.0-239.255.255.255.

Этот класс адресов зарезервирован для будущего использования. Диапазон адресов – 240.0.0.0-247.255.255.255.

Два адреса в каждой подсети являются зарезервированными. IP-адрес, оканчивающийся на бинарный ноль, используют для обозначения сетевого адреса. Например для сети класса А адрес сети - 112.0.0.0, и этой сети принадлежит сетевой адрес - 112.2.3.4. Адрес сети используется роутерами для определения маршрута. Второй зарезервированный адрес - бродкаст-адрес (broadcast). Этот адрес используется, когда источник хочет послать данные всем устройствам в сети. В отличие от адреса сети - в адресе бродкаста используется бинарная единица(в октетах, отвечающих за часть хоста). Например, для сети 171.10.0.0 последние 16 бит адреса используется для обозначения хоста, и бродкаст-адрес будет выглядеть как 171.10.255.255. И все устройства в сети 171.10.0.0 получат данные, которые были посланы по адресу 171.10.255.255.

Выше мы рассмотрели разбиение на классы сетей. Но не всегда имеет смысл использовать например сеть класса С когда в ней реально будут использоваться только половина адресов. Для более рационального распределения адресов используются подсети. Адрес подсети включает в себя сетевую часть сети класса А,В или С и т.н. subnet field и часть хоста. Для subnet field выделяется значение из октетов, принадлежащих хосту (т.е. для адреса подсети может быть использовано до 3-х октетов из сети класса А, до 2х из сети класса В, и 1 для С соответственно). Создавая адрес подсети, несколько бит из части

адреса хоста переназначаются в адрес сети. Минимальное значение, которое может быть таким образом занято из адреса хоста - 2 бита. Если занять один бит, то мы получим подсеть, состоящую из двух адресов - адреса подсети и бродкаст-адреса. Максимальное число, которое может быть занято под подсеть из адреса хоста - такое, чтобы в последнем октете для хоста осталось два бита. Разбиение на подсети уменьшает также размеры бродкаст-доменов - т.к. в сетях класса А в бродкаст-домеене будет порядка 16 миллионов компьютеров. И если каждый пошлет хотя бы по одному бродкаст-адресу то нагрузка на сеть будет очень большой. А т.к. бродкасты не пересылаются роутерами - происходит ограничение бродкаст-домена и сохранения полосы пропускания от ненужного трафика.

Для определения размерности подсети используется маска подсети. Маска подсети определяет, какая часть IP-адреса используется для задания сетевой части, а какая часть для хоста. Маску подсети можно определить следующим образом. Запишем IP-адрес подсети в бинарном виде. Все значения, относящиеся к network-part и subnet-part заменим на 1, все значения, относящиеся к host-part заменим на 0. В результате получим маску подсети.

Например маска подсети для сети класса А будет выглядеть следующим образом: 255.0.0.0, для сети класса В: 255.255.0.0, для сети класса С: 255.255.255.0. Если у нас используется подсеть - то как было сказано выше все значения, относящиеся к сети и подсети будут 1, хосту - 0. Например маска 255.255.255.192 определяет подсеть класса С, кол-во хостов в данной подсети будет равно 64.

Для передачи данных кроме IP-адреса также нужно знать MAC-адрес. Для определения соответствия IP-адресу MAC-адреса существует ARP-протокол(Address Resolution Protocol, протокол определения адресов). ARP-таблица находится в оперативной памяти и периодически обновляется. В этой таблице находятся IP-адреса только локальной сети. Когда источник определяет адрес получателя, источник также смотрит MAC адрес получателя и если он его не находит у себя в ARP-таблице, то он делает запрос для получения MAC-адреса получателя. Протокол RARP (Reverse ARP - обратный ARP) действует наоборот - он известному MAC-адресу сопоставляет IP-адрес. Это необходимо, например, для работы таких протоколов, как BOOTP, DHCP. При загрузке по локальной сети посылается broadcast-запрос - противоположный ARP-запросу. Если в ARP-запросе идет опрос "IP-получателя известен, MAC-получателя - ???", то в RARP-запросе "MAC-получателя известен, IP - ???". Поэтому если сервер знает например какому маку должен соответствовать IP-адрес, он отвечает на RARP-запрос и не придется вручную вводить IP-адрес на компьютере (пример такого сервиса - DHCP - особенно это эффективно при большом кол-ве компьютеров).

5. Transport layer (layer 4)

Рассмотрим TCP/IP протокол 4го транспортного уровня модели OSI. TCP/IP имеет два протокола - TCP и UDP. TCP обеспечивает виртуальные соединения между пользовательскими приложениями.

Основные характеристики TCP и UDP

TCP	UDP
Для работы устанавливает соединение	Работает без соединений
Гарантированная доставка данных	Гарантий доставки нет

TCP	UDP
Разбивает исходное сообщение на сегменты	Передает сообщения целиком в виде датаграмм
На стороне получателя сообщение заново собирается из сегментов	Принимаемые сообщения не объединяются
Пересылает заново потерянные сегменты	Подтверждений о доставке нет
Контролирует поток сегментов	Никакого контроля потока датаграмм нет

5.1. TCP

TCP/IP представляет собой комбинацию двух уровней TCP и IP. IP - протокол 3го уровня - не обеспечивает гарантированной доставки данных, но обеспечивающий наилучшую доставку через сеть. TCP - протокол 4го уровня - позволяет гарантировать доставку данных. Поэтому совместно они могут предоставить большее количество сервисов. Работа с TCP-соединением состоит из трех фаз. Инициации соединения - инициатор посылает пакет с порядковым номером (sequence number - x). Второй хост, который получает тот пакет, запоминает его и отправляет подтверждение(sequence number + 1 - $x+1$) и свое собственное sequence number(y). Первый хост снова получает свой sequence number, увеличенный на 1. Если данные дошли корректно (полученный sequence number равен, тому который он отправил+1($x+1$)), то первый хост отправляет sequence number $y+1$ и второй хост проверяет его верность. Если во всех трех фазах было получено верное значение для sequence number, то соединение считается успешно установленным.

Понятие окна (window) в TCP. При передаче пакета принимающий хост посылает подтверждение о получении пакета. Источник, получив подтверждение, посылает следующий пакет. Более эффективным способом передачи пакетов является использование "окна". В этом случае отправляется подряд несколько пакетов, затем ожидается подтверждение, если подтверждение пришло, посылает следующую часть пакетов подряд. При этом размер окна может динамически меняться - например размер "окна" равен 5, но хост загружен и размер буфера на текущий момент не позволяет одновременно принять 5 пакетов, об этом он сообщает источник и размер окна уменьшается. Одним из недостатков TCP является то, что установка "виртуального соединения" требует постоянного маршрута между источником и получателем. Это приводит при изменении маршрутной таблицы в сети к необходимости заново устанавливать TCP-соединение.

5.2. UDP

В отличие от TCP - UDP не гарантирует доставку данных. UDP не устанавливает виртуального соединения, источник просто шлет user datagram получателю. Если данные были некорректно доставлены, или вообще часть пакетов потерялась - UDP не позволяет их восстановить. Запрос на получение данных должен будет выполнен заново. Казалось бы, недостатков у данного протокола довольно много что ставит под сомнение его эффективность. Но есть сервисы, где UDP незаменимо. Например, при передаче потокового аудио-видео если бы мы использовали TCP, то при потере одного пакета у нас будет приостановлена передача данных для передачи потерянного пакета. При использовании UDP потерянный пакет - всего лишь незначительное ухудшение изображения/звука, при этом передача данных не прерывается. Также при использовании UDP нам не обязательно установление виртуального соединения - нам не важен путь, по которому пройдет пакет - например при недоступности самого оптимального маршрута, пакет может быть доставлен

через другой запасной маршрут - при этом последовательность UDP-датаграмм будет сохранена.

Оба протокола TCP и UDP используют порты(port) для прохождения информации на вышестоящие уровни. Использование номера порта позволяет передавать разные данные одновременно. Приложения используют эти порты, часть которых зарезервировано под стандартные приложения. Например, для FTP зарезервирован порт 21. Далее приведен список распределения портов: порты меньше 255 - используются для публичных сервисов, порты из диапазона 255-1023 - назначаются компаниями-разработчиками для их приложений, номера выше 1023 - не регулируемые.

6. Session layer (layer 5)

После того как пакет, разобранный на сетевом уровне, пройдет транспортный уровень - он поступит на session layer. 5й уровень обеспечивает установку, контроль и окончание сессии между приложениями. Уровень сессий координирует приложения, когда они взаимодействуют между двумя хостами. Соединения между двумя компьютерами включает в себя множество мини-"переговоров", обеспечивающих более эффективное соединение двух компьютеров.

7. Presentation layer (layer 6)

Уровень представлений отвечает за представление данных в форме, что бы получатель их смог понять. Приведем пример - два человека общаются на разных языках. Что бы друг друга понять, им нужно использовать человека, который понимает оба этих языка. Также уровень представлений - сервис для трансляции данных, которые необходимо передать через сеть. 6-й уровень обеспечивает следующую функциональность: data formatting(presentation), data encryption, data compression. После получения данных с уровня приложений, уровень представлений выполняет одну или все эти функции перед передачей данных на session layer. Приведем пример использования уровня представлений. Например, первый хост использует Extended Binary Coded Decimal Interchange Code (EBCDIC) для представления данных. Второй хост для представления данных использует American Standard Code for Information Interchange (ASCII). Presentation layer обеспечивает взаимодействие двух этих систем с разными типами представления данных. Уровень представлений также отвечает за шифрование данных - для сохранения частной информации при передаче через публичные сети. За компрессию данных также отвечает представительский уровень. Используя математические алгоритмы для уменьшения объема передаваемых данных. Но это эффективно при использовании каналов связи с маленькой пропускной способностью т.к. использование компрессии может потребовать значительных вычислительных мощностей при больших объемах трафика.

8. Application layer (layer 7)

Уровень приложений определяет, какие ресурсы существуют для связи между хостами. Этот уровень не обеспечивает связь со всеми уровнями модели OSI. Он передает данные только на presentation layer. Большинство сетевых приложений можно классифицировать как клиент-серверные приложения. Клиент находится на локальном компьютере и

запрашивает необходимый сервис. Сервер находится на удаленном компьютере и обеспечивает необходимый клиенту сервис. Клиент-серверное приложение работает по следующей схеме: client-request, server-response, client-request, server-response и т.д. Так работает, например браузер - клиент запрашивает URL, а сервер в ответ на этот запрос выдает соответствующую веб-страничку. Примером приложений application layer могут служить:

- telnet - удаленный клиент для работы с сетью
- dns - domain name system
- e-mail - электронные сообщения

Остановимся немного подробнее на DNS. Как известно каждому компьютеру соответствует IP-адрес. И, например, у нас есть web-сервер. И что бы получить к нему доступ мы должны послать запрос на этот сервер. Запрос можно сделать по IP-адресу, например 194.87.0.50 - но запомнить каждый IP-адрес сервера очень затруднительно. Да и сам IP-адрес мало информативен. Для решения этой задачи существует система соответствия имени сервера и IP-адреса. Т.е. например адресу 194.87.0.50 соответствует доменное имя www.ru. ДНС - это иерархическая система. В этой иерархии домены делятся на уровни. Первый уровень обозначает принадлежность домена. Существует много доменов первого уровня - классификация их может быть разная: например .ru, .us, .uk - обозначают принадлежность к стране, .edu - .edu - educational sites, .com - commercial sites, .gov - government sites, .org - non-profit sites, .net - network service. В каждом домене первого уровня может быть множество доменов второго уровня. У домена третьего уровня может быть множество доменов третьего уровня и т.д. За каждую зону отвечает ДНС-сервер. При запросе на сервер у него либо содержится требуемая информация, или он знает какой сервер отвечает за эту ДНС-зону (например сервер test.ru знает про все имена в зоне test.ru, но например у нас есть домен следующего уровня node1.host1.test.ru - основной сервер не обязан знать все имена в домене host1.test.ru, он знает какой сервер отвечает за эту зону и на этот сервер пере направляет запрос)

9. Утилиты для работы с сетью

Рассмотрим основные программы, позволяющие читать и изменять сетевые параметры, диагностировать и выявлять ошибки при работе сети. В различных ОС существует свои наборы утилит. Сравним их для 2 систем, например Microsoft Windows NT и Sun Solaris. Какими бы разными не были эти ОС, в каждой из них реализована модель OSI. Естественно программная и аппаратная реализация стека модели OSI у них различается, но формирование, обработка данных на всех уровнях осуществляет по установленному стандарту.

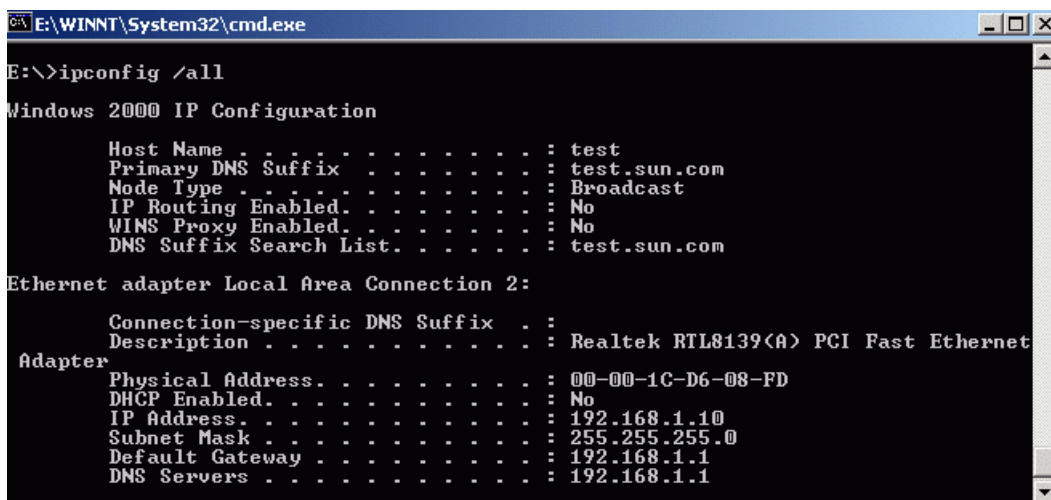
Например, рассмотрим работу файл-сервера под управлением 2 выбранных ОС. Пусть передается файл с файл-сервера Windows на сервер Solaris (будем рассматривать только процесс передачи данных с одного сервера на другой). ОС Windows передает данные на уровень приложения модели OSI. Далее происходит процесс инкапсуляции – на каждом уровне ОС добавляет служебную информацию. Полученное в результате сообщение на физическом уровне в виде электрических импульсов передается по сети получателю – файл-серверу, под управлением ОС Solaris. На этом сервере происходит процесс декапсуляции. Полученное сообщение постепенно “разворачивается”, на каждом уровне считывается соответствующая служебная информация. Т.к. каждый уровень модели OSI

стандартизирован, для потребителей есть возможность использовать совместно оборудование и программное обеспечение различных производителей. В результате файл-сервер под управлением ОС Solaris может получить файл, посланный под управлением ОС другого производителя.

Если бы на серверах применялся различный стандарт передачи данных на физическом уровне (один сервер формирует передачу данных в виде светового импульса, а другой в виде электрических сигналов), то взаимодействие без использования дополнительного оборудования было бы невозможно. Поэтому вводят понятие сете-независимых и сете-зависимых уровней. Три нижних уровня – физический, канальный и сетевой уровни являются сете-зависимыми. Т.е. например смена Ethernet на ATM влечет полную смену протокола физического и канального уровней. Три верхних уровня – приложений, представительский и сессионный - ориентированы на приложения и практически не зависят от физической технологии построения сети. Так переход от Ethernet на FDDI не требует изменений в перечисленных уровнях. Транспортный уровень является “прослойкой” между сете-зависимыми и сете-независимыми уровнями. Он скрывает все детали функционирования нижних уровней от верхних. Это позволяет разработчику приложений не задумываться о технических средствах реализации транспортировки сообщений.

9.1. IPCONFIG (IFCONFIG)

Рассмотрим утилиты, которые позволяют просматривать, проверять и изменять сетевые настройки. Обычно сетевые настройки включают информацию 3-го уровня (сетевого) – IP адрес, маску подсети и т.д. Для просмотра сетевых настроек в ОС Windows можно использовать команду `ipconfig`. Она выдает информацию об IP-адресе, маски подсети (`netmask`), роутера по умолчанию(`default gateway`). Задав дополнительный параметр команде `ipconfig – all`, можно получить более подробную информацию – имя компьютера, имя домена, тип сетевой карты, MAC-адрес и т.д.



```
E:\WINNT\System32\cmd.exe
E:\>ipconfig /all

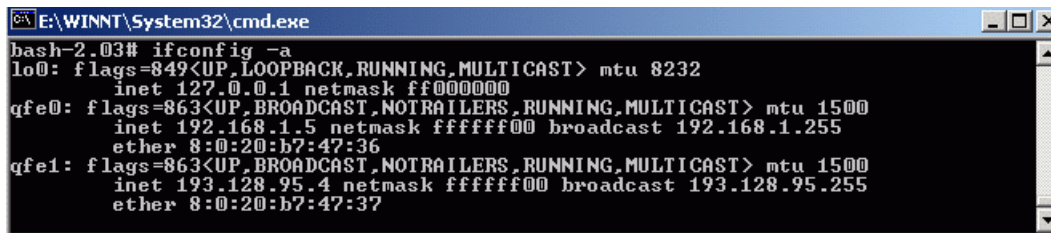
Windows 2000 IP Configuration

    Host Name . . . . . : test
    Primary DNS Suffix . . . . . : test.sun.com
    Node Type . . . . . : Broadcast
    IP Routing Enabled. . . . . : No
    WINS Proxy Enabled. . . . . : No
    DNS Suffix Search List. . . . . : test.sun.com

Ethernet adapter Local Area Connection 2:

    Connection-specific DNS Suffix . : 
    Description . . . . . : Realtek RTL8139(A) PCI Fast Ethernet
    Adapter Physical Address. . . . . : 00-00-1C-D6-08-FD
    DHCP Enabled. . . . . : No
    IP Address. . . . . : 192.168.1.10
    Subnet Mask . . . . . : 255.255.255.0
    Default Gateway . . . . . : 192.168.1.1
    DNS Servers . . . . . : 192.168.1.1
```

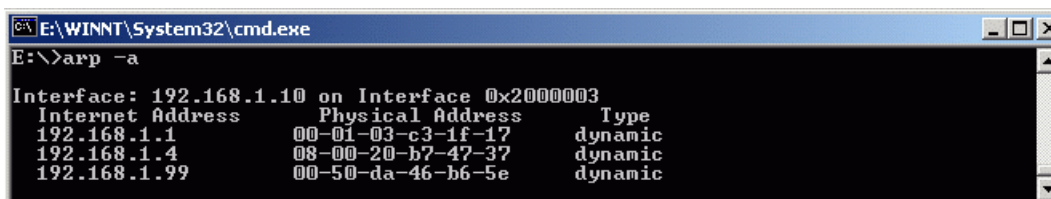
В ОС Solaris для получения IP-адреса и прочих сетевых настроек используется команда – `ifconfig`. Она также показывает название интерфейса, IP-адреса, маску подсети, MAC-адрес.



```
E:\WINNT\System32\cmd.exe
bash-2.03# ifconfig -a
lo0: flags=849<UP,LOOPBACK,RUNNING,MULTICAST> mtu 8232
    inet 127.0.0.1 netmask ffffffff
qfe0: flags=863<UP,BROADCAST,NOTRAILERS,RUNNING,MULTICAST> mtu 1500
    inet 192.168.1.5 netmask ffffffff broadcast 192.168.1.255
    ether 8:0:20:b7:47:36
qfe1: flags=863<UP,BROADCAST,NOTRAILERS,RUNNING,MULTICAST> mtu 1500
    inet 193.128.95.4 netmask ffffffff broadcast 193.128.95.255
    ether 8:0:20:b7:47:37
```

9.2. ARP

Как уже было сказано ранее, в оперативной памяти компьютера находится arp-таблица. В ней содержится MAC-адрес сетевого устройства и соответствующий ему IP-адрес. Для просмотра этой таблички используется команда arp. Например, arp -a выводит все известные MAC-адреса.



```
E:\WINNT\System32\cmd.exe
E:\>arp -a

Interface: 192.168.1.10 on Interface 0x20000003
Internet Address      Physical Address      Type
192.168.1.1           00-01-03-c3-1f-17     dynamic
192.168.1.4           08-00-20-b7-47-37     dynamic
192.168.1.99          00-50-da-46-b6-5e     dynamic
```

Таблица MAC-адресов хостов хранится в памяти не постоянно. После определенного времени записи из таблицы автоматически удаляются, если к данному IP-адресу не было обращений. Существует вообще два типа записей в ARP-таблице – статический и динамический. Динамическая запись периодически обновляется, при появлении новой пары MAC-IP автоматически добавляется. Статическая запись вносится вручную и существует до тех пор, пока вручную эта запись не будет удалена или компьютер (маршрутизатор) не будет перезагружен.

Если компьютер посылает сообщение на IP-адрес, который неизвестен на транспортном и канальном уровне формируется broadcast-frame – т.н. ARP-запрос. Каждый узел локальной сети получает ARP-запрос и сравнивает IP-адрес указанный в запросе и свой. При совпадении адреса в запросе и получателя формируется ARP-ответ, в котором указывается свой IP-адрес и MAC-адрес. Этот ответ получает машина, посылавшая ARP-запрос и добавляет запись в свою ARP-таблицу. Задержка на получение MAC-адреса составляет порядка нескольких миллисекунд, поэтому для пользователя это будет практически незаметно и задержки при передаче данных не возникнет (в большинстве реализаций сетевой уровень ставит пакет в очередь если неизвестен MAC-адрес, но бывают системы когда формируется ARP-запрос, но сам пакет отбрасывается, и на транспортный уровень возлагаются задачи по повторной передаче пакета).

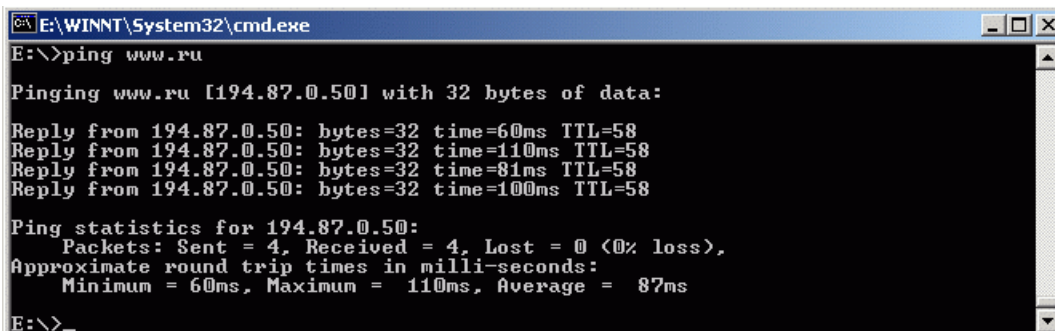
9.3. Ping

Для выявления различных неполадок в сети существует несколько утилит, которые позволяют определить, на каком уровне модели OSI произошел сбой или неверная конфигурация сетевых протоколов. Одна из таких утилит – ping. Эта утилита позволяет определить ошибки на сетевом уровне (layer 3), используя протокол ICMP (Internet Control Message Protocol) – протокол межсетевых управляющих сообщений. Формат использования этой утилиты довольно прост: ping 194.87.0.50 (где 194.87.0.50 – IP-адрес удаленного

компьютера). Если все работает верно – в результате выводится время задержки прихода ответа от удаленного компьютера и время жизни пакета.

Протокол ICMP находится на стыке двух уровней – сетевого и транспортного. Основным принцип действия этого протокола – формирования ICMP эха-запроса (echo-request) и эха-ответа (echo-reply). Запрос эха и ответ на него может использоваться для проверки достижимости хоста-получателя и его способности отвечать на запросы. Также прохождение эхо-запроса и эхо-ответа проверяет работоспособность основной части транспортной системы, маршрутизацию на машине источника, проверяет работоспособность и корректную маршрутизацию на роутерах между источником и получателем, а также работоспособность и правильность маршрутизации получателя.

Например, если на посланный echo-request возвращается корректный echo-reply от машины, которой был послан запрос – можно сказать что транспортная система работает корректно. И если браузер не может отобразить веб-страницу, то проблема, скорее всего, не в первых трех уровнях модели OSI.



```
E:\WINNT\System32\cmd.exe
E:\>ping www.ru

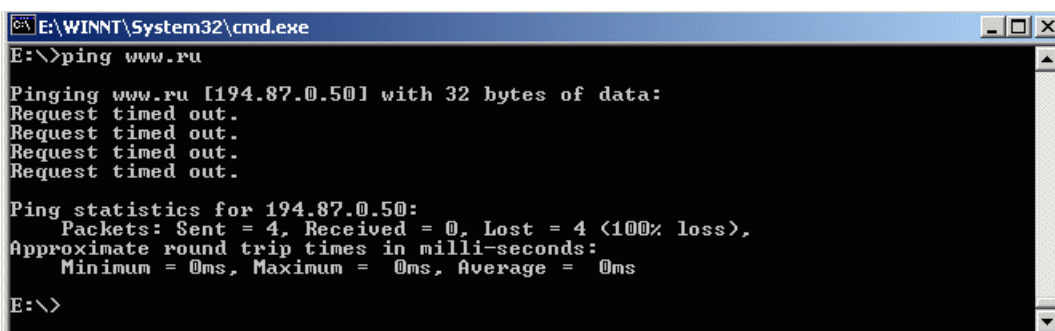
Pinging www.ru [194.87.0.50] with 32 bytes of data:

Reply from 194.87.0.50: bytes=32 time=60ms TTL=58
Reply from 194.87.0.50: bytes=32 time=110ms TTL=58
Reply from 194.87.0.50: bytes=32 time=81ms TTL=58
Reply from 194.87.0.50: bytes=32 time=100ms TTL=58

Ping statistics for 194.87.0.50:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 60ms, Maximum = 110ms, Average = 87ms

E:\>
```

Из примера видно, что по умолчанию размер посылаемого пакета 32 байта, далее вычисляется время задержки ответа, и TTL (time to live – время жизни пакета). В приведенном выше примере показано успешное выполнение команды ping. В случаях, когда запросы echo request посылаются, но echo reply не возвращаются, выводится сообщение об истечении времени запроса.



```
E:\WINNT\System32\cmd.exe
E:\>ping www.ru

Pinging www.ru [194.87.0.50] with 32 bytes of data:
Request timed out.
Request timed out.
Request timed out.
Request timed out.

Ping statistics for 194.87.0.50:
    Packets: Sent = 4, Received = 0, Lost = 4 (100% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 0ms, Maximum = 0ms, Average = 0ms

E:\>
```

9.4. Traceroute

Утилита traceroute также использует протокол ICMP для определения маршрута прохождения пакета. При отсылке traceroute устанавливает значение TTL последовательно от 1 до 30. Каждый маршрутизатор, через который проходит пакет на пути к назначенному хосту, увеличивает значение TTL на единицу. С помощью TTL происходит предотвращение заикливание пакета в “петлях” маршрутизации. Например, при выходе маршрутизатора

или линии связи из строя требуется некоторое время для определения, что данный маршрут потерян, и его необходимо обойти.

В это время существует вероятность, что датаграмма будет уничтожена в петле маршрутизации. Чтобы предотвратить потерю датаграммы, поле TTL устанавливается в максимальную величину.

Когда маршрутизатор получает IP датаграмму с TTL равным либо 0, либо 1, он не должен отправлять эту датаграмму дальше. (Принимающий хост должен доставить подобную датаграмму в приложение, так как датаграмма не может быть обработана маршрутизатором. Как правило, системы не должны получать датаграммы с TTL равным 0). Если такую датаграмму получает маршрутизатор, он уничтожает ее и посылает хосту, который ее отправил, ICMP сообщение "время истекло" (time exceeded). Принцип работы traceroute заключается в том, что IP датаграмма, содержащая это ICMP сообщение, имеет в качестве адреса источника IP адрес маршрутизатора.

Теперь мы можем понять, как работает Traceroute. На хост назначения отправляется IP датаграмма с TTL, равным единице. Первый маршрутизатор, который должен обработать датаграмму, уничтожает ее (так как TTL равно 1) и отправляет ICMP сообщение об истечении времени (time exceeded). Таким образом, определяется первый маршрутизатор в маршруте. Затем Traceroute отправляет датаграмму с TTL, равным 2, что позволяет получить IP адрес второго маршрутизатора. Это продолжается до тех пор, пока датаграмма не достигнет хоста назначения. Однако если датаграмма прибыла именно на хост назначения, он не уничтожит ее и не сгенерирует ICMP сообщение об истечении времени, так как датаграмма достигла своего конечного назначения. В ответ генерируется UDP-датаграмма с номером порта, который заведомо не будет обработан приложением (порт выше 30000) и с сообщением port unreachable. При получении такой датаграммы хостом, с которого выполнялась программа traceroute, делается вывод, что или удаленный хост работает корректно, или значение TTL было превышено значения по умолчанию (при выполнении утилиты traceroute TTL по умолчанию равно 30).

Рассмотрим пример выполнения утилиты traceroute

```
traceroute to netserv1.chg.ru (193.233.46.3), 30 hops max, 38 byte packets
 1  n3-core.mipt.ru (194.85.80.1)  1.508 ms  0.617 ms  0.798 ms
 2  mipt-gw-eth0.mipt.ru (193.125.142.177)  2.362 ms  2.666 ms  1.449 ms
 3  msu-mipt-atm0.mipt.ru (212.16.1.1)  5.536 ms  5.993 ms  10.431 ms
 4  M9-LYNX.ATM6-0.11.M9-R2.msu.net (193.232.127.229)  12.994 ms  7.830 ms
 6.816 ms
 5  Moscow-BNS045-ATM4-0-3.free.net (147.45.20.37)  12.228 ms  7.041 ms  8.731
ms
 6  ChgNet-gw.free.net (147.45.20.222)  77.103 ms  75.234 ms  92.334 ms
 7  netserv1.chg.ru (193.233.46.3)  96.627 ms  94.714 ms  134.676 ms
```

Первая строка содержит имя и IP-адрес хоста назначения, величина TTL может быть не более 30 и размер посылаемого пакета 38 байт. Последующие строки начинаются с TTL, после чего следует имя хоста или маршрутизатора и его IP адрес. Для каждого значения TTL отправляет 3 датаграммы. Для каждой возвращенной датаграммы вычисляется и выводится время возврата. Если в течение 3-х секунд на каждую из 3-х датаграмм не был получен ответ, то посылается следующая датаграмма, а вместо значения времени выводится звездочка. Время возврата – это время прохождения датаграммы от источника (хоста, выполняющего программу traceroute) до маршрутизатора. Если нас интересует

время, потраченное на каждую пересылку, то необходимо вычесть из значения времени TTL N время TTL N+1. В каждой из операционных систем сетевая часть утилиты реализована практически одинаково, но реализация на уровне приложений различается.

В ОС Solaris используется утилита traceroute. В качестве параметра задается IP-адрес или доменное имя удаленного хоста, связь до которого требуется проверить. В примере, приведенном выше, видно успешное выполнение traceroute и корректную работу сетезависимых уровней (физический, канальный, сетевой).

В ОС – Windows утилита называется tracert. Используется также как и в ОС Solaris (tracert net.serv1.chg.ru). Принципиального различия между утилитами tracert и traceroute нет. Особым отличием traceroute является наличие большей функциональности (например, можно указать, с какого TTL выводить информацию).

В случаях какой-либо неполадки выводится соответствующее сообщение. Например, при недоступности сети на маршрутизаторе выдается сообщение net unreachable:

```
Moscow-BNS045-ATM4-0-3.free.net (147.45.20.37) 947.327 ms !N 996.548 ms !N
995.257 ms
```

!N – где 147.45.20.37 – маршрутизатор, на котором последующий маршрут недоступен. Если недоступен сам хост: msu-mipt-atm0.mipt.ru (212.16.1.1) 5.536 ms !H 5.993 ms !H 10.431 ms !H. Если в качестве ошибки мы получаем IP - protocol unreachable.

9.5. Route

Для просмотра и редактирования таблицы маршрутов используется утилита – route. Типичный пример таблицы маршрутизации на персональном компьютере:

Для ОС Windows:

route print

```
E:\>route print
=====
Interface List
0x1 ..... MS TCP Loopback interface
0x1000003 ...00 00 1c d6 08 fd ..... rt181398 NDIS 5.0 driver
=====
Active Routes:
Network Destination    Netmask          Gateway         Interface      Metric
0.0.0.0                0.0.0.0          192.168.1.1     192.168.1.10   1
127.0.0.0              255.0.0.0        127.0.0.1       127.0.0.1      1
192.168.1.0            255.255.255.0    192.168.1.10   192.168.1.10   1
192.168.1.10          255.255.255.255  127.0.0.1       127.0.0.1      1
192.168.1.255          255.255.255.255  192.168.1.10   192.168.1.10   1
224.0.0.0              224.0.0.0        192.168.1.10   192.168.1.10   1
255.255.255.255        255.255.255.255  192.168.1.10   192.168.1.10   1
Default Gateway:       192.168.1.1
=====
```

В таблице маршрутизации указывается сеть, маска сети, маршрутизатор, через который доступна эта сеть, интерфейс и метрика маршрута. Из приведенной таблицы видно, что маршрут по умолчанию доступен через маршрутизатор 192.168.1.1. Сеть 192.168.1.0 netmask 255.255.255.0 – является локальной сетью.

При добавлении маршрута можно использовать следующую команду.

```
route ADD 157.0.0.0 MASK 255.0.0.0 157.55.80.1
```

157.0.0.0 – удаленная сеть, 255.0.0.0 – маска удаленной сети, 157.55.80.1 – маршрутизатор, через который доступна эта сеть. Примерно такой же синтаксис используется при удалении маршрута: `route DELETE 157.0.0.0`

В ОС Solaris для просмотра таблицы маршрутизации используется немного другая команда – `netstat -r`.

```

E:\WINNT\System32\cmd.exe
bash-2.03$ netstat -rn

Routing Table:
  Destination      Gateway            Flags   Ref       Use    Interface
-----
192.168.1.0        192.168.1.4        U        2          0      qfe1
default            192.168.1.1        UG       0        81231   qfe1
127.0.0.1          127.0.0.1          UH       0       7446936  lo0
  
```

Добавление и удаление маршрутов происходит командой `route`:

`route add -net 157.6 157.6.1.20`, где 157.6 – сокращенный адрес подсети, а 157.6.1.20 – маршрут, по которому эта сеть доступна. Также удаление маршрутов в таблице маршрутизации: `route del -net 157.6`

9.6. Netstat

Утилита `netstat` позволяет определить, какие порты открыты и по каким портам происходит передача данных между узлами сети. Например, если запустить веб-браузер и открыть для просмотра web-страницу, то, запустив `netstat`, можно увидеть следующую строку:

```
TCP      java:3687                www.ru:http           ESTABLISHED
```

В проведенном примере первое значение – TCP – тип протокола (может быть tcp,udp), далее идет имя локальной машины и локальный порт, `www.ru:http` - имя удаленного хоста и порта, к которому производится обращение, ESTABLISHED – показывает, что tcp-соединение установлено.

В ОС Windows командой `netstat -an` можно получить список всех открытых портов (параметр `-n` не определяет DNS-имя, а выводит только IP-адрес). Из примера ниже видно, что установленных соединений нет, а все открытые порты находятся в состоянии “прослушивания”, т.е. к этому порту можно обратиться для установки соединения. TCP-порт 139 отвечает за установку Netbios-сессий (например для передачи данных через “сетевое окружение”).


```

E:\WINNT\System32\cmd.exe
E:\>netstat -an

Active Connections

Proto Local Address          Foreign Address         State
TCP    0.0.0.0:135              0.0.0.0:0               LISTENING
TCP    0.0.0.0:445              0.0.0.0:0               LISTENING
TCP    0.0.0.0:1087             0.0.0.0:0               LISTENING
TCP    0.0.0.0:3759             0.0.0.0:0               LISTENING
TCP    192.168.1.10:139         0.0.0.0:0               LISTENING
UDP    0.0.0.0:135              *:*:                     *:*:
UDP    0.0.0.0:445              *:*:                     *:*:
UDP    0.0.0.0:500              *:*:                     *:*:
UDP    0.0.0.0:1053             *:*:                     *:*:
UDP    0.0.0.0:1059             *:*:                     *:*:
UDP    127.0.0.1:1191           *:*:                     *:*:
UDP    127.0.0.1:62515          *:*:                     *:*:
UDP    127.0.0.1:62517          *:*:                     *:*:
UDP    127.0.0.1:62519          *:*:                     *:*:
UDP    127.0.0.1:62521          *:*:                     *:*:
UDP    127.0.0.1:62523          *:*:                     *:*:
UDP    127.0.0.1:62524          *:*:                     *:*:
UDP    192.168.1.10:137        *:*:                     *:*:
UDP    192.168.1.10:138        *:*:                     *:*:

E:\>

```

В ОС Solaris для получения информации об используемых портах также используется утилита netstat. Формат вывода практически одинаков.

```

E:\WINNT\System32\cmd.exe
bash-2.03$ netstat -an

UDP
  Local Address      Remote Address      State
-----
  *.137              Idle
  *.138              Idle
192.168.113.5.137    Idle
192.168.113.5.138    Idle
192.168.113.4.137    Idle
192.168.113.4.138    Idle
  *.3130             Idle

TCP
  Local Address      Remote Address      Swind Send-Q Rwind Recv-Q State
-----
  *.*                *.*                0      0      0      0 IDLE
  *.111              *.*                0      0      0      0 LISTEN
  *.*                *.*                0      0      0      0 IDLE
  *.389              *.*                0      0      0      0 LISTEN
  *.32775            *.*                0      0      0      0 LISTEN
  *.8888             *.*                0      0      0      0 LISTEN
  *.4800             *.*                0      0      0      0 LISTEN
  *.22               *.*                0      0      0      0 LISTEN
  *.6000             *.*                0      0      0      0 LISTEN
  *.7001             *.*                0      0      0      0 LISTEN
  *.7002             *.*                0      0      0      0 LISTEN
  *.32782            *.*                0      0      0      0 LISTEN
  *.139              *.*                0      0      0      0 LISTEN
  *.25               *.*                0      0      0      0 LISTEN
  *.21               *.*                0      0      0      0 LISTEN
  *.80               *.*                0      0      0      0 LISTEN
  *.443              *.*                0      0      0      0 LISTEN
  *.3128             *.*                0      0      0      0 LISTEN
  *.1234             *.*                0      0      0      0 LISTEN
192.168.1.4.7000     192.168.113.3.3167 8760    0      8760    0 ESTABLISHED

bash-2.03$

```

9.7. Задания для практического занятия

1. Выведите информацию об IP-адресе, маске подсети и маршрутизаторе по умолчанию.
2. Выведите arp-таблицу.

3. Выполните утилиту `ping`. В качестве удаленного компьютера используйте IP-адрес компьютера из вашей локальной подсети; произвольного IP-адреса – например 194.87.0.50. Посмотрите `arp`-таблицу и обратите внимание на изменения в ней (если они есть).
4. Выполните утилиту `tracert`. В качестве удаленного хоста используйте DNS-имя или IP-адрес удаленного сервера или компьютера.
5. Выполните утилиту `netstat`. Запустите веб-браузер и откройте произвольную интернет-страничку и практически одновременно выполните `netstat` заново.

10. Пакет java.net

Классы для работы с сетью в Java располагаются в пакете `java.net`, и простейшим из них является класс `URL`. С его помощью можно сконструировать `uniform resource locator` (URL), который имеет следующий формат:

```
protocol://host:port/resource
```

Здесь `protocol` - название протокола, используемого для связи; `host` - IP-адрес или DNS-имя сервера, к которому производится обращение; `port` - номер порта сервера (если порт не указан, то используется значение по умолчанию для указанного протокола); `resource` - имя запрашиваемого ресурса, причем оно может быть составным, например:

```
ftp://myserver.ru/pub/docs/Java/JavaCourse.txt
```

Затем можно воспользоваться методом `openStream()`, который возвращает `InputStream`, что позволяет считать содержимое ресурса. Например, следующая программа при помощи `LineNumberReader` считывает первую страницу сайта `http://www.ru` и выводит ее на консоль.

```
import java.io.*;
import java.net.*;

public class Net {
    public static void main(String args[]) {
        try {
            URL url = new URL("http://www.ru");
            LineNumberReader r = new LineNumberReader(new
InputStreamReader(url.openStream()));
            String s = r.readLine();
            while (s!=null) {
                System.out.println(s);
                s = r.readLine();
            }
            System.out.println(r.getLineNumber());
            r.close();
        } catch (MalformedURLException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

```
}  
}
```

Из примера можно видеть, что работа с сетью, как и работа с потоками, требует дополнительной работы с исключительными ситуациями. Ошибка `MalformedURLException` появляется в случае, если строка с URL содержит ошибки.

Более функциональным классом является `URLConnection`, который можно получить с помощью метода класса `URL.openConnection()`. У этого класса есть два метода - `getInputStream()` (именно с его помощью работает `URL.openStream()`) и `getOutputStream()`, который можно использовать для передачи данных на сервер, если он поддерживает такую операцию (многие публичные web-сервера закрыты для таких действий).

Класс `URLConnection` является абстрактным. Виртуальная машина предоставляет реализации этого класса для каждого протокола, например, в том же пакете `java.net` определен класс `HttpURLConnection`. Понятно, что классы `URL` и `URLConnection` предоставляют возможности работы через сеть на прикладном уровне с помощью высокоуровневых протоколов.

Пакет `java.net` также предоставляет доступ к протоколам более низкого уровня - TCP и UDP. Для этого сначала надо ознакомиться с классом `InetAddress`, который является интернет-адресом, или IP. Экземпляры этого класса создаются не с помощью конструкторов, а с помощью статических методов:

```
InetAddress getLocalHost()  
InetAddress getByName(String name)  
InetAddress[] getAllByName(String name)
```

Первый метод возвращает IP-адрес машины, на которой выполняется Java-программа. Второй метод возвращает адрес сервера, чье имя передается в качестве параметра. Это может быть как DNS-имя, так и числовой IP, записанный в виде текста, например, "67.11.12.101". Наконец третий метод определяет все IP-адреса указанного сервера.

Для работы с TCP-протоколом используются классы `Socket` и `ServerSocket`. Первым создается `ServerSocket` - сокет на стороне сервера. Его простейший конструктор имеет только один параметр - номер порта, на котором будут приниматься входящие запросы. После создания вызывается метод `accept()`, который приостанавливает выполнение программы и ожидает, пока какой-нибудь клиент не инициализирует соединение. В этом случае работа сервера возобновляется, а метод возвращает экземпляр класса `Socket` для взаимодействия с клиентом:

```
try {  
    ServerSocket ss = new ServerSocket(3456);  
    Socket client=ss.accept(); // Метод не возвращает управление, пока не  
    подключится клиент  
} catch (IOException e) {  
    e.printStackTrace();  
}
```

Клиент для подключения к серверу также используется класс `Socket`. Его простейший конструктор принимает два параметра - адрес сервера (в виде строки или экземпляра `InetAddress`) и номер порта. Если сервер принял запрос, то сокет конструируется успешно, и далее можно воспользоваться методами `getInputStream()` или `getOutputStream()`.

```
try {
    Socket s = new Socket("localhost", 3456);
    InputStream is = s.getInputStream();
    is.read();
} catch (UnknownHostException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
}
```

Обратите внимание на обработку исключительной ситуации `UnknownHostException`, которая будет генерироваться, если виртуальная машина с помощью операционной системы не сможет распознать указанный адрес сервера в случае, если он задан строкой. Если же он задан экземпляром `InetAddress`, то эту ошибку надо обрабатывать при вызове статических методов этого класса.

На стороне сервера класс `Socket` используется точно таким же образом - через методы `getInputStream()` и `getOutputStream()`. Приведем более полный пример:

```
import java.io.*;
import java.net.*;

public class Server {
    public static void main(String args[]) {
        try {
            ServerSocket ss = new ServerSocket(3456);
            System.out.println("Waiting...");
            Socket client=ss.accept();
            System.out.println("Connected");
            client.getOutputStream().write(10);
            client.close();
            ss.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Сервер по запросу клиента отправляет число 10 и завершает работу. Обратите внимание, что при завершении вызываются методы `close()` для открытых сокетов.

Класс клиента:

```
import java.io.*;
import java.net.*;

public class Client {
    public static void main(String args[]) {
        try {
            Socket s = new Socket("localhost", 3456);
            InputStream is = s.getInputStream();
            System.out.println("Read: "+is.read());
        }
    }
}
```

```
s.close();
} catch (UnknownHostException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
}
}
}
```

После запуска сервера, а затем клиента, можно увидеть результат - полученное число 10, после чего обе программы закроются.

Рассмотрим эти классы более подробно. Во-первых, класс `ServerSocket` имеет конструктор, в который передается кроме номера порта еще и адрес машины. Это может показаться странным, ведь сервер открывается на той же машине, где работает программа, зачем специально указывать ее адрес? Однако если компьютер имеет несколько сетевых интерфейсов (сетевых карточек), то он имеет и несколько сетевых адресов. С помощью такого детализированного конструктора можно указать, по какому именно адресу ожидать подключение. Это должен быть именно локальный адрес машины, иначе возникнет ошибка.

Аналогично класс `Socket` имеет расширенный конструктор для указания как локального адреса, с которого будет устанавливаться соединение, так и локального порта (иначе операционная система выделяет произвольный свободный порт).

Во-вторых, можно воспользоваться методом `setSoTimeout(int timeout)` класса `ServerSocket`, чтобы указать время в миллисекундах, на протяжении которого нужно ожидать подключение клиента. Это позволяет не "зависать" серверу, если никто не пытается начать с ним работать. Таймаут задается в миллисекундах, нулевое значение означает бесконечное время ожидания.

Важно подчеркнуть, что после установления соединения с клиентом сервер выходит из метода `accept()`, то есть перестает быть готов принимать новые запросы. Однако как правило желательно, чтобы сервер мог работать с несколькими клиентами одновременно. Для этого необходимо при подключении очередного пользователя создавать новый поток исполнения, который будет обслуживать его, а основной поток снова войдет в метод `accept()`. Приведем пример такого решения:

```
import java.io.*;
import java.net.*;

public class NetServer {
    public static final int PORT = 2500;
    private static final int TIME_SEND_SLEEP = 100;
    private static final int COUNT_TO_SEND = 10;
    private ServerSocket servSocket;

    public static void main(String[] args) {
        NetServer server = new NetServer();
        server.go();
    }

    public NetServer() {
```

```
try{
    servSocket = new ServerSocket(PORT);
}catch(IOException e){
    System.err.println("Unable to open Server Socket : " + e.toString());
}
}

public void go() {

    // Класс-поток для работы с подключившимся клиентом
    class Listener implements Runnable{
        Socket socket;
        public Listener(Socket aSocket){
            socket = aSocket;
        }
        public void run(){
            try{
                System.out.println("Listener started");
                int count = 0;
                OutputStream out = socket.getOutputStream();
                OutputStreamWriter writer = new OutputStreamWriter(out);
                PrintWriter pWriter = new PrintWriter(writer);
                while(count<COUNT_TO_SEND){
                    count++;
                    pWriter.print(((count>1)?",":"")+ "Say" + count);
                    sleeps(TIME_SEND_SLEEP);
                }
                pWriter.close();
            }catch(IOException e){
                System.err.println("Exception : " + e.toString());
            }
        }
    }

    // Основной поток, циклически выполняющий метод accept()
    System.out.println("Server started");
    while(true){
        try{
            Socket socket = servSocket.accept();
            Listener listener = new Listener(socket);
            Thread thread = new Thread(listener);
            thread.start();
        }catch(IOException e){
            System.err.println("IOException : " + e.toString());
        }
    }
}

public void sleeps(long time) {
```

```
try{
    Thread.sleep(time);
}catch(InterruptedException e){
}
}
}
```

Теперь объявим клиента. Эта программа будет запускать несколько потоков, каждый из которых независимо подключается к серверу, считывает его ответ и выводит на консоль.

```
import java.io.*;
import java.net.*;

public class NetClient implements Runnable{
    public static final int PORT = 2500;
    public static final String HOST = "localhost";
    public static final int CLIENTS_COUNT = 5;
    public static final int READ_BUFFER_SIZE = 10;

    private String name = null;

    public static void main(String[] args) {
        String name = "name";
        for(int i=1; i<=CLIENTS_COUNT; i++){
            NetClient client = new NetClient(name+i);
            Thread thread = new Thread(client);
            thread.start();
        }
    }

    public NetClient(String name) {
        this.name = name;
    }

    public void run() {
        char[] readed = new char[READ_BUFFER_SIZE];
        StringBuffer strBuff = new StringBuffer();
        try{
            Socket socket = new Socket(HOST, PORT);
            InputStream in = socket.getInputStream();
            InputStreamReader reader = new InputStreamReader(in);
            while(true){
                int count = reader.read(readed, 0, READ_BUFFER_SIZE);
                if(count==-1)break;
                strBuff.append(readed, 0, count);
                Thread.yield();
            }
        } catch (UnknownHostException e) {
            e.printStackTrace();
        }
    }
}
```

```
    } catch (IOException e) {  
        e.printStackTrace();  
    }  
    System.out.println("client " + name + "    read : " + strBuff.toString());  
}  
}
```

Теперь рассмотрим UDP. Для работы с этим протоколом и на стороне клиента, и на стороне сервера используется класс `DatagramSocket`. У него есть следующие конструкторы:

```
DatagramSocket()  
DatagramSocket(int port)  
DatagramSocket(int port, InetAddress laddr)
```

При вызове первого конструктора сокет открывается на произвольном доступном порту, что уместно для клиента. Конструктор с одним параметром, задающим порт, как правило применяется на серверах, чтобы клиенты знали, на каком порту им нужно пытаться устанавливать соединение. Наконец, последний конструктор необходим для машин, у которых присутствует несколько сетевых интерфейсов.

После открытия сокетов начинается обмен датаграммами. Они представляются экземплярами класса `DatagramPacket`. При отсылке сообщения применяется следующий конструктор:

```
DatagramPacket(byte[] buf, int length, InetAddress address, int port)
```

Массив содержит данные для отправки (созданный пакет будет иметь длину равную `length`), а адрес и порт указывают получателя пакета. После этого вызывается метод `send()` класса `DatagramSocket`.

```
try {  
    DatagramSocket s = new DatagramSocket();  
    byte data[]={1, 2, 3};  
    InetAddress addr = InetAddress.getByName("localhost");  
    DatagramPacket p = new DatagramPacket(data, 3, addr, 3456);  
    s.send(p);  
    System.out.println("Datagram sent");  
    s.close();  
} catch (SocketException e) {  
    e.printStackTrace();  
} catch (UnknownHostException e) {  
    e.printStackTrace();  
} catch (IOException e) {  
    e.printStackTrace();  
}
```

Для получения датаграммы также создается экземпляр класса `DatagramPacket`, но в конструктор передается лишь массив, в который будут записаны данные по получении (также указывается ожидаемая длина пакета). Сокет необходимо создать с указанием порта, иначе скорее всего сообщение просто не дойдет до адресата. Используется метод `receive()` класса `DatagramSocket` (аналогично методу `ServerSocket.accept()` этот метод также

прерывает выполнение потока, пока не придет запрос от клиента). Пример реализации получателя:

```
try {
    DatagramSocket s = new DatagramSocket(3456);
    byte data[]=new byte[3];
    DatagramPacket p = new DatagramPacket(data, 3);
    System.out.println("Waiting...");
    s.receive(p);
    System.out.println("Datagram received: "+data[0]+", "+data[1]+", "+data[2]);
    s.close();
} catch (SocketException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
}
```

Если запустить сначала получателя, а затем отправителя, то можно увидеть, что первый напечатает содержимое полученной датаграммы, а потом программы завершат свою работу.

В заключение приведем пример сервера, который получает датаграммы и отправляет их обратно, дописав к ним слово `received`.

```
import java.io.*;
import java.net.*;

public class DatagramDemoServer {
    public static final int PORT = 2000;
    private static final int LENGTH_RECEIVE = 1;
    private static final byte[] answer = ("received").getBytes();

    private DatagramSocket servSocket = null;
    private boolean keepRunning = true;

    public static void main(String[] args) {
        DatagramDemoServer server = new DatagramDemoServer();
        server.service();
    }

    public DatagramDemoServer() {
        try{
            servSocket = new DatagramSocket(PORT);
        }catch(SocketException e){
            System.err.println("Unable to open socket : " + e.toString());
        }
    }

    protected void service() {
        DatagramPacket datagram;
        InetAddress clientAddr;
```

```
int clientPort;
byte[] data;
while(keepRunning){
    try{
        data = new byte[LENGTH_RECEIVE];
        datagram = new DatagramPacket(data, data.length);
        servSocket.receive(datagram);
        clientAddr = datagram.getAddress();
        clientPort = datagram.getPort();
        data = getSendData(datagram.getData());
        datagram = new DatagramPacket(data, data.length, clientAddr, clientPort);
        servSocket.send(datagram);
    }catch(IOException e){
        System.err.println("I/O Exception : " + e.toString());
    }
}

protected byte[] getSendData(byte b[]) {
    byte[] result = new byte[b.length+answer.length];
    System.arraycopy(b, 0, result, 0, b.length);
    System.arraycopy(answer, 0, result, b.length, answer.length);
    return result;
}
}
```

11. Заключение

В данном разделе были рассмотрены теоретические моменты сети как одной большой взаимодействующей системы. Были рассмотрены все уровни модели OSI и их функциональные назначения. Также были рассмотрены основные утилиты, используемые для настройки и обнаружения неисправностей в сети.

12. Контрольные вопросы

16-1. Назовите уровни модели OSI.

а.) OSI модель состоит из семи уровней:

7 - уровень приложений

6 - уровень представлений

5- уровень сессий

4- транспортный уровень

3 - сетевой уровень

2 – уровень передачи данных

1 – физический уровень

16-2. Что характеризует физический уровень? Приведите основные типы физической среды передачи данных.

а.) Основная функция физического уровня – передача данных в виде сигнала (электрического, электромагнитного, светового). В простейшем случае конечное сообщение на физическом уровне формируется в виде битов (набор из 0 и 1, например, 010110101). В виде электрического сигнала оно выглядит как “пила”.

Основные типы физической среды передачи данных:

- телефонный провод;
- коаксиальный провод;
- витая пара;
- оптоволокно.

16-3. Опишите основные функции MAC-адреса и LLC-подуровня.

а.) MAC-адрес обеспечивает идентификацию компьютеров в сети. Адресаций является “плоской” – в пределах одного логического сегмента. MAC-sublayer отвечает за связь канального уровня с физическим уровнем. LLC-sublayer отвечает за связь канального уровня с сетевым уровнем. LLC-подуровень обеспечивает независимость передачи данных на сетевой уровень от физической среды, которая используется для передачи данных.

16-4. К какому классу сети относятся следующие IP-адреса?

- 64.12.8.130
- 224.180.224.5
- 172.16.0.1
- 194.86.87.256
- 195.149.20.130

а.) Ответ:

- A;
- D;
- B, private;
- не существует;
- C.

16-5. Каковы основные функции ARP-протокола и RARP-протокола?

а.) Для передачи данных отправителю кроме IP-адреса также нужно знать MAC-адрес получателя. Для определения соответствия IP-адресу MAC-адреса существует ARP-протокол (Address Resolution Protocol). ARP-таблица находится в оперативной памяти и периодически обновляется. В этой таблице находятся IP-адреса компьютеров только из той же

локальной сети. Когда источник определяет MAC-адрес получателя, сначала он обращается к своей arp-таблице, и если не находит его там, то делает широковещательный запрос для определения требуемого адреса.

Протокол RARP действует наоборот – известному MAC-адресу он сопоставляет IP-адрес. Это необходимо, например, для работы таких протоколов, как BOOTP, DHCP. При загрузке по локальной сети посылается broadcast запрос – противоположный arp-запросу. Если в ARP-запросе идет опрос “IP-получателя известен, MAC-получателя - ???”, то в RARP-запросе “MAC-получателя известен, IP - ???”. Например, сервис DHCP занимается динамическим выделением IP-адресов для компьютеров в локальной сети. Новая машина отправляет broadcast запрос со своим MAC-адресом. На сервере хранится таблица соответствия MAC и IP-адресов, поэтому он отвечает на гагр-запрос, и таким образом пользователю не приходится вручную вводить IP-адрес на компьютере (особенно это эффективно при большом количестве машин).

16-6. Что такое маска сети, маска подсети и как она вычисляется?

- а.) Для определения размерности подсети используется маска подсети. Она определяет, какие биты в IP-адресе являются сетевой частью (network part), то есть адресуют сеть, а какие – host part (часть, задающая адрес компьютера в сети). Чтобы определить маску сети, нужно записать IP-адрес подсети в двоичном виде. Все биты, относящиеся к network-part заменим на 1, а все значения, относящиеся к host-part, заменим на 0. В результате получим маску сети.

Например маска сети класса А будет выглядеть следующим образом: 255.0.0.0, для сети класса В: 255.255.0.0, для сети класса С: 255.255.255.0. Если используется подсеть, то (по описанному алгоритму) все биты, относящиеся к сети и подсети, будут равны 1, к хосту – 0. Например маска 255.255.255.192 определяет подсеть класса С, в которой кол-во хостов будет равно 64.

16-7. Какие основные различия между протоколами TCP и UDP?

- а.) TCP- протокол с образованием постоянного соединения гарантирует доставку каждого пакета. UDP не использует постоянного соединения и не гарантирует доставку сообщения. TCP-протокол используется в сервисах, где важна гарантированная доставка информации – электронная почта, передача HTML-страниц, FTP.

UDP используется в сервисах, где важна своевременная доставка пакета с минимальными затратами: например при транслировании видеоинформации.

16-8. Перечислите основные функции уровня приложений, уровня представлений, уровня приложений.

- а.) Уровень приложений определяет, какие ресурсы существуют для связи между сетевыми узлами. Этот уровень не обеспечивает связь со всеми уровнями модели OSI.

Уровень представлений отвечает за представление данных в форме, понятной получателю. Например, для представления данных могут быть использованы такие кодировки, как Extended Binary Coded Decimal Interchange Code (EBCDIC) или American Standard Code for Information Interchange (ASCII). Если компьютеры, между которыми установлено сетевое соединение, используют различные протоколы, то presentation layer обеспечивает их корректное взаимодействие.

Уровень сессий обеспечивает управление диалогом: фиксирует, какая из сторон является активной в настоящий момент, и предоставляет средства синхронизации. Последние позволяют вставлять контрольные точки в длинные передачи, чтобы в случае отказа можно было вернуться назад к последней контрольной точке, а не начинать все сначала. На практике немногие приложения используют сеансовый уровень, и он редко реализуется в виде отдельных протоколов, хотя функции этого уровня часто объединяют с функциями прикладного уровня и реализуют в одном протоколе.

16-9. Ping и traceroute – назначение и различия между этими утилитами.

- а.) Основное назначение утилит ping и traceroute – выявление неисправностей в сети. Эти утилиты используют протокол ICMP. Ping определяет работоспособность сети между двумя хостами. Во время работы утилита ping посылает echo-запрос на IP-адрес получателя. Если echo-запрос был корректно доставлен, то в ответ присылается echo-ответ. При корректном получении echo-ответа принимается решение, что удаленный хост доступен. Т.к. echo-ответ был корректно доставлен отправителю, можно сделать вывод что физические, канальные и сетевые уровни на обоих хостах работают корректно, и роутеры корректно пересылают пакеты.

Traceroute использует протокол ICMP для определения маршрута прохождения пакета. Он используется для выявления ошибок в таблицах маршрутизации, доступности подсетей и хостов.

16-10. Как происходит выбор маршрута для передачи данных? Какая утилита позволяет изменять маршрут передачи данных?

- а.) Маршрут, по которому необходимо переслать пакет, определяется на основе данных маршрутной таблицы, которая хранится в каждом компьютере. В простейшем случае она содержит только одну запись для работы с удаленными сетями – маршрут по умолчанию. В этом случае все данные, посылаемые в удаленную сеть, отправляются на маршрутизатор по умолчанию, и далее он определяет дальнейший маршрут пересылки пакетов.

Если в маршрутной таблице есть несколько записей, то определяется, к какой подсети принадлежит IP-адрес получателя, и в соответствии с выбранной записью о маршруте происходит пересылка данных. Если в маршрутной таблице не найдена требуемая подсеть, то данные отправляются на “default gateway” – маршрутизатор по умолчанию.

Для просмотра и изменения таблицы маршрутизации используется утилита route.

16-11. Какие действия необходимо предпринять для установления TCP соединения между двумя Java-приложениями? ?

- a.) Во-первых, на стороне сервера надо создать экземпляр класса `ServerSocket` с указанием порта, и затем вызвать у этого объекта метод `accept()`. При входе в этот метод поток исполнения приостанавливает свою работу в ожидании подключения клиента.

Клиенту необходимо создать экземпляр класса `Socket` с указанием IP-адреса и порта сервера. После успешного выполнения конструктора на стороне сервера метод `accept()` вернет экземпляр класса `Socket` для взаимодействия двух приложений.

16-12. Какие действия необходимо предпринять для обмена данными по UDP протоколу?

- a.) Во-первых, отправляющая сторона должна создать экземпляр класса `DatagramSocket`. Затем поместить в объект `DatagramPacket` данные, предназначенные для отсылки, а также IP-адрес и порт получателя.

Принимающая стороны также создает `DatagramSocket` с указанием порта, на котором ожидается получение датаграммы. Создается объект `DatagramPacket` для сохранения полученных данных, и вызывается метод `DatagramSocket.receive()`. Теперь отправитель может вызывать метод `DatagramSocket.send()`

16-13. Можно ли с помощью класса `URL` пересылать данные на сервер?

- a.) Нет

- b.) Да

- ✓с.) Да, если сервер позволяет закидывание (upload) данных.