Communication (Part I)

Amir H. Payberah
amir@sics.se

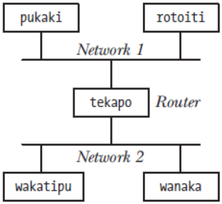
Amirkabir University of Technology
(Tehran Polytechnic)

AMIRKABIR UNIVERSITY
OF TECHNOLOGY

Basic Networking Model

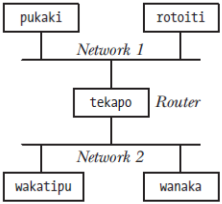
Internetworking

- An internetwork (internet (with a lowercase i)) is a network of computer networks.



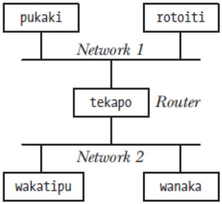
Internetworking

- An internetwork (internet (with a lowercase i)) is a network of computer networks.
- Subnetwork refers to one of the networks composing an internet.



Internetworking

- An internetwork (internet (with a lowercase i)) is a network of computer networks.
- Subnetwork refers to one of the networks composing an internet.
- An internet aims to hide the details of different physical networks, to present a unified network architecture.



The Internet

- TCP/IP has become the dominant protocol for the internetworking.

The Internet

- TCP/IP has become the dominant protocol for the internetworking.
- The Internet (with an uppercase I) refers to the TCP/IP internet that connects millions of computers globally.

The Internet

- TCP/IP has become the dominant protocol for the internetworking.
- The Internet (with an uppercase I) refers to the TCP/IP internet that connects millions of computers globally.
- The first widespread implementation of TCP/IP appeared with 4.2BSD in 1983.

Networking Protocols and Layers

- A networking protocol is a set of rules defining how information is to be transmitted across a network.

Networking Protocols and Layers

- A networking protocol is a set of rules defining how information is to be transmitted across a network.
- Networking protocols are generally organized as a series of layers.

Networking Protocols and Layers

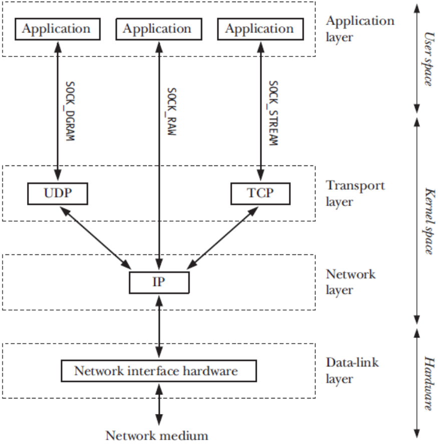
- A networking protocol is a set of rules defining how information is to be transmitted across a network.
- Networking protocols are generally organized as a series of layers.
- Each layer building on the layer below it to add features that are made available to higher layers.

Networking Protocols and Layers

- A networking protocol is a set of rules defining how information is to be transmitted across a network.
- Networking protocols are generally organized as a series of layers.
- Each layer building on the layer below it to add features that are made available to higher layers.
- Transparency: each protocol layer shields higher layers from the operation and complexity of lower layers.

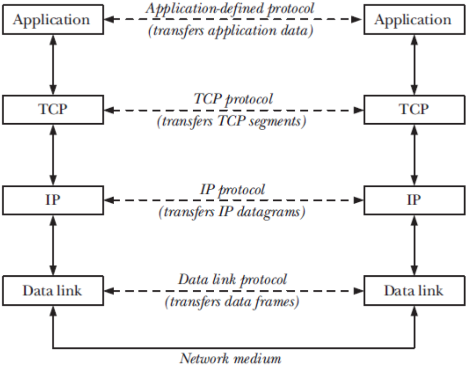
TCP/IP Protocol Suite

- The TCP/IP protocol suite is a layered networking protocol.



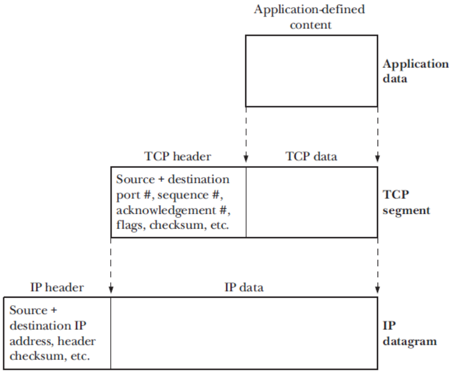
TCP/IP Protocol Layers

- Data-Link layer
- Network layer (IP)
- Transport layer (TCP, UDP)
- Application



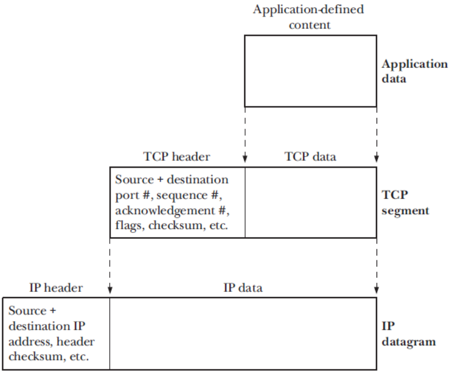
Encapsulation

- **Encapsulation**: the information passed from a higher layer to a lower layer is treated as opaque data by the lower layer.
  - The lower layer does not interpret information from the upper layer.



Encapsulation

- Encapsulation: the information passed from a higher layer to a lower layer is treated as opaque data by the lower layer.
  - The lower layer does not interpret information from the upper layer.
- When data is passed up from a lower layer to a higher layer, a converse unpacking process takes place.



Data-Link Layer (1/3)

- It is concerned with transferring data across a physical link in a network.

Data-Link Layer (1/3)

- It is concerned with transferring data across a physical link in a network.
- It consists of the device driver and the hardware interface (network card) to the underlying physical medium, e.g., fiber-optic cable.

Data-Link Layer (2/3)

- The data-link layer encapsulates datagrams from the network layer into units, called frames.

Data-Link Layer (2/3)

- The data-link layer encapsulates datagrams from the network layer into units, called frames.
- It also adds each frame a header containing the destination address and frame size.

Data-Link Layer (2/3)

- The data-link layer encapsulates datagrams from the network layer into units, called frames.
- It also adds each frame a header containing the destination address and frame size.
- The data-link layer transmits the frames across the physical link and handles acknowledgements from the receiver.

Data-Link Layer (3/3)

- From an application-programming point of view, we can generally ignore the data-link layer, since all communication details are handled in the driver and hardware.

Data-Link Layer (3/3)

- From an application-programming point of view, we can generally ignore the data-link layer, since all communication details are handled in the driver and hardware.

- Maximum Transmission Unit (MTU): the upper limit that the layer places on the size of a frame.
  - data-link layers have different MTUs.

```
netstat -i
```

Network Layer (1/4)

- It is concerned with delivering data from the source host to the destination host.

Network Layer (1/4)

- It is concerned with delivering data from the source host to the destination host.

- It tasks include:
  - Breaking data into fragments small enough for transmission via the data-link layer.
  - Routing data across the internet.
  - Providing services to the transport layer.

Network Layer (1/4)

- It is concerned with delivering data from the source host to the destination host.

- It tasks include:
  - Breaking data into fragments small enough for transmission via the data-link layer.
  - Routing data across the internet.
  - Providing services to the transport layer.

- In the TCP/IP protocol suite, the principal protocol in the network layer is IP.

Network Layer (2/4)

- IP transmits data in the form of packets.

Network Layer (2/4)

- IP transmits data in the form of packets.

- Each packet sent between two hosts travels independently across the network.

- IP transmits data in the form of packets.
- Each packet sent between two hosts travels independently across the network.
- An IP packet includes a header that contains the address of the source and target hosts.

Network Layer (3/4)

- IP is a connectionless protocol: it does not provide a virtual circuit connecting two hosts.

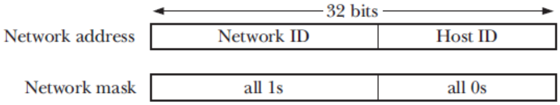
Network Layer (3/4)

- IP is a connectionless protocol: it does not provide a virtual circuit connecting two hosts.

- IP is an unreliable protocol: it makes a best effort to transmit datagrams from the sender to the receiver, but it does not guarantee:
  - that packets will arrive in the order they were transmitted,
  - that they will not be duplicated,
  - that they will arrive at all.

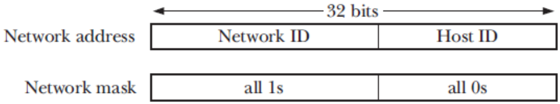
Network Layer (4/4)

- An IP address consists of two parts:
  - Network ID: specifies the network on which a host resides.
  - Host ID: identifies the host within that network.



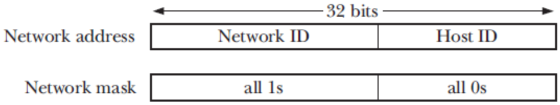
Network Layer (4/4)

- An IP address consists of two parts:
  - Network ID: specifies the network on which a host resides.
  - Host ID: identifies the host within that network.
- An IPv4 address consists of 32 bits: `204.152.189.0/24`
  - loopback `127.0.0.1` refers to system on which process is running.



Network Layer (4/4)

- An IP address consists of two parts:
  - Network ID: specifies the network on which a host resides.
  - Host ID: identifies the host within that network.
- An IPv4 address consists of 32 bits: `204.152.189.0/24`
  - loopback `127.0.0.1` refers to system on which process is running.
- Network mask: a sequence of 1s in the leftmost bits, followed by a sequence of 0s
  - The 1s indicate which part of the address contains the assigned network ID.
  - The 0s indicate which part of the address is available to assign as host IDs.



Transport Layer (1/5)

- Transport protocol provides an end-to-end communication service to applications residing on different hosts.

Transport Layer (1/5)

- Transport protocol provides an end-to-end communication service to applications residing on different hosts.

- Two widely used transport-layer protocols in the TCP/IP suite:
  - User Datagram Protocol (UDP): the protocol used for datagram sockets.
  - Transmission Control Protocol (TCP): the protocol used for stream sockets.

Transport Layer (2/5)

- Port: a method of differentiating the applications on a host.
  - 16-bit number

Transport Layer (2/5)

- Port: a method of differentiating the applications on a host.
  - 16-bit number
  - All ports below 1024 are well known, used for standard services, e.g., http: 80, ssh: 22.

Transport Layer (2/5)

- Port: a method of differentiating the applications on a host.
  - 16-bit number
  - All ports below 1024 are well known, used for standard services, e.g., http: 80, ssh: 22.
  - Shown as `192.168.1.1:8080`.

Transport Layer (3/5)

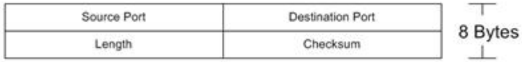
- UDP, like IP, is connectionless and unreliable.

Transport Layer (3/5)

- UDP, like IP, is connectionless and unreliable.
- If an application layered on top of UDP requires reliability, then this must be implemented within the application.

Transport Layer (3/5)

- UDP, like IP, is connectionless and unreliable.
- If an application layered on top of UDP requires reliability, then this must be implemented within the application.
- UDP adds just two features to IP:
  - Port number
  - Data checksum to allow the detection of errors in the transmitted data.



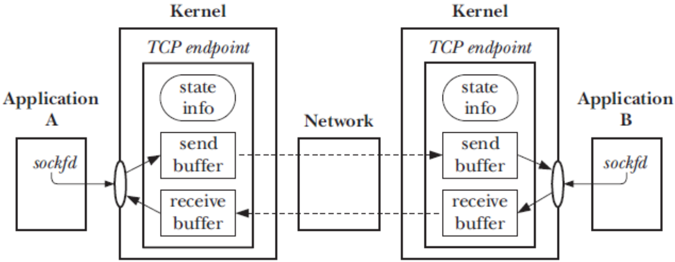
[http://www.tamos.net/~rhay/overhead/ip-packet-overhead.htm]

Transport Layer (4/5)

- TCP provides a reliable, connection-oriented, bidirectional, byte-stream communication channel between two endpoints.

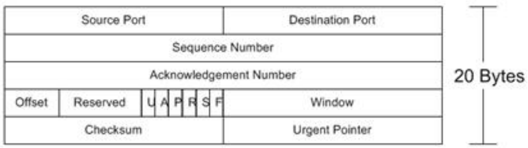
Transport Layer (4/5)

- TCP provides a reliable, connection-oriented, bidirectional, byte-stream communication channel between two endpoints.
- Before communication can commence, TCP establishes a communication channel between the two endpoints.



Transport Layer (5/5)

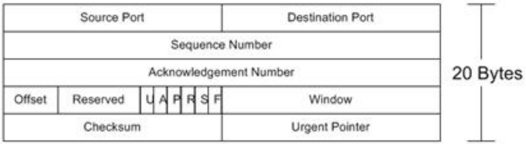
- In TCP, data is broken into segments: each is transmitted in a single IP packet.



[http://www.tamos.net/∼rhay/overhead/ip-packet-overhead.htm]

Transport Layer (5/5)

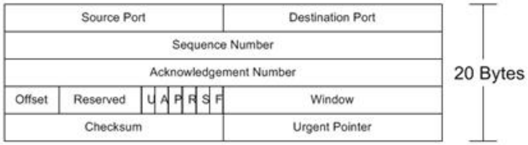
- In TCP, data is broken into segments: each is transmitted in a single IP packet.
- When a destination receives a TCP segment, it sends an ack. to the sender, informing weather it received the segment correctly or not.



[http://www.tamos.net/~rhay/overhead/ip-packet-overhead.htm]

Transport Layer (5/5)

- In TCP, data is broken into segments: each is transmitted in a single IP packet.
- When a destination receives a TCP segment, it sends an ack. to the sender, informing weather it received the segment correctly or not.
- Other features of TCP:
  - Sequencing
  - Flow control
  - Congestion control



[http://www.tamos.net/∼rhay/overhead/ip-packet-overhead.htm]

Socket Programming

Socket

- A socket is defined as an endpoint for communication.

Socket

- A socket is defined as an endpoint for communication.
- A typical client-server scenario:
  - Each process creates a socket: both processes require one.
  - The server binds its socket to a well-known address (name) so that clients can locate it.

TCP Socket

TCP Server

- Setting up a server process requires five steps:
  1. Create a `ServerSocket` object.
  2. Put the server into a waiting state.
  3. Set up input and output streams.
  4. Send and receive data.
  5. Close the connection.

Setting up a TCP Server Process (1/5)

- Create a ServerSocket object.
- The ServerSocket constructor requires a port number as an argument.

```
ServerSocket serverSocket = new ServerSocket(1234);
```

Setting up a TCP Server Process (2/5)

- Put the server into a waiting state.
- The server blocks for a client to connect, by calling `accept`.
- It returns a `Socket` object when a connection is made.

```
Socket link = serverSocket.accept();
```

Setting up a TCP Server Process (3/5)

- Set up input and output streams.
- Methods `getInputStream` and `getOutputStream` to get references to streams associated with the socket returned in step 2.
- These streams will be used for communication with the client that has just made connection.

```
Scanner input = new Scanner(link.getInputStream());

PrintWriter output = new PrintWriter(link.getOutputStream(),true);
```

Setting up a TCP Server Process (4/5)

- Send and receive data using the `Scanner` and `PrintWriter` objects.

```
Scanner input = new Scanner(link.getInputStream());
String input = input.nextLine();

PrintWriter output = new PrintWriter(link.getOutputStream(),true);
output.println("data");
```

Setting up a TCP Server Process (5/5)

- This is achieved via method `close` of class `Socket`.

```
link.close();
```

TCP Server Example (1/2)

```
public class TCPEchoServer {
  private static ServerSocket serverSocket;
  private static final int PORT = 1234;
  public static void main(String[] args) {
    try {
      serverSocket = new ServerSocket(PORT); //Step 1.
    } catch(IOException ioEx) { ... }

    do {
      handleClient();
    } while (true);
```

TCP Server Example (2/2)

```
  private static void handleClient() {
    Socket link = null; //Step 2.
    try {
      link = serverSocket.accept(); //Step 2.

      Scanner input = new Scanner(link.getInputStream()); //Step 3.
      PrintWriter output = new PrintWriter(link.getOutputStream(),true); //Step 3.

      String message = input.nextLine(); //Step 4.
      while (!message.equals("CLOSE")) {
        System.out.println("Message received.");
        output.println("Echo message: " + message); //Step 4.
        message = input.nextLine();
      }
    } catch(IOException ioEx) { ... }
    finally {
      try {
        link.close(); //Step 5.
      } catch(IOException ioEx) { ... }
    }
  }
}
```

TCP Client

- Setting up a client process requires four steps:
  1. Establish a connection to the server.
  2. Set up input and output streams.
  3. Send and receive data.
  4. Close the connection.

Setting up a TCP Client Process (1/4)

- Establish a connection to the server.
- Create a `Socket` object: supplying it with the server IP address and a port number for the service.

```
Socket link = new Socket(InetAddress.getLocalHost(), 1234);
```

Setting up a TCP Client Process (2/4)

- These are set up in exactly the same way as the server streams were set up.
- Calling methods `getInputStream` and `getOutputStream` of the `Socket` object.

Setting up a TCP Client Process (3/4)

- Send and receive data.
- The `Scanner` object at the client end will receive messages sent by the `PrintWriter` object at the server end.
- The `PrintWriter` object at the client end will send messages that are received by the `Scanner` object at the server end.

Setting up a TCP Client Process (4/4)

- Close the connection.
- This is exactly the same as for the server process.

TCP Client Example (1/2)

```
public class TCPEchoClient {
  private static InetAddress host;
  private static final int PORT = 1234;
  public static void main(String[] args) {
    try {
      host = InetAddress.getLocalHost();
    } catch(UnknownHostException uhEx) { ... }
    accessServer();
  }
```

TCP Client Example (2/2)

```
private static void accessServer() {
  Socket link = null; //Step 1.
  try {
    link = new Socket(host,PORT); //Step 1.

    Scanner input = new Scanner(link.getInputStream()); //Step 2.
    PrintWriter output = new PrintWriter(link.getOutputStream(),true); //Step 2.

    Scanner userEntry = new Scanner(System.in);
    String message, response;
    do {
      message = userEntry.nextLine();
      output.println(message); //Step 3.
      response = input.nextLine(); //Step 3.
      System.out.println("\nSERVER> " + response);
    } while (!message.equals("CLOSE"));
  } catch(IOException ioEx) { ... }
  finally {
    try {
      link.close(); //Step 4.
    } catch(IOException ioEx) { ... }
  }
}
```

UDP Socket

UDP Server

- Setting up a server process requires nine steps:
  1. Create a `DatagramSocket` object.
  2. Create a buffer for incoming datagrams.
  3. Create a `DatagramPacket` object for the incoming datagrams.
  4. Accept an incoming datagram.
  5. Accept the sender's address and port from the packet.
  6. Retrieve the data from the buffer.
  7. Create the response `datagram_socket`.
  8. Send the response datagram.
  9. Close the `DatagramSocket`.

Setting up a UDP Server Process (1/9)

- Create a DatagramSocket object.
- Supplying the object's constructor with the port number.

```
DatagramSocket datagramSocket = new DatagramSocket(1234);
```

Setting up a UDP Server Process (2/9)

- Create a buffer for incoming datagrams.
- This is achieved by creating an array of bytes.

```
byte[] buffer = new byte[256];
```

Setting up a UDP Server Process (3/9)

- Create a DatagramPacket object for the incoming datagrams.
- The constructor for this object requires the previously-created byte array and its size.

```
DatagramPacket inPacket = new DatagramPacket(buffer, buffer.length);
```

Setting up a UDP Server Process (4/9)

- Accept an incoming datagram.
- This is effected via the `receive` method.

```
datagramSocket.receive(inPacket);
```

Setting up a UDP Server Process (5/9)

- Accept the sender's address and port from the packet.
- Methods `getAddress` and `getPort` are used for this.

```
InetAddress clientAddress = inPacket.getAddress();
int clientPort = inPacket.getPort();
```

Setting up a UDP Server Process (6/9)

- Retrieve the data from the buffer.
- The data will be retrieved as a strin, using a `String` constructor that takes three arguments:
  - A byte array
  - The start position within the array
  - The number of byte

```
String message = new String(inPacket.getData(), 0, inPacket.getLength());
```

Setting up a UDP Server Process (7/9)

- Create the response datagram.
- Create a DatagramPacket object, using the constructor that takes four arguments:
  - Te byte array containing the response message
  - The size of the response
  - The client's address
  - The client's port number

```
DatagramPacket outPacket = new DatagramPacket(response.getBytes(),
  response.length(), clientAddress, clientPort);
```

Setting up a UDP Server Process (8/9)

- Send the response datagram.
- By calling method send.

```
datagramSocket.send(outPacket);
```

Setting up a UDP Server Process (9/9)

- Close the DatagramSocket.
- By calling method close.

```
datagramSocket.close();
```

UDP Server Example (1/2)

```
public class UDPEchoServer {
  private static fi nal int PORT = 1234;
  private static DatagramSocket datagramSocket;
  private static DatagramPacket inPacket, outPacket;
  private static byte[] buffer;
  public static void main(String[] args) {
    try {
      datagramSocket = new DatagramSocket(PORT); //Step 1.
    } catch(SocketException sockEx) { ... }
    handleClient();
  }
```

UDP Server Example (2/2)

```
  private static void handleClient() {
    try {
      String messageIn, messageOut;
      InetAddress clientAddress = null;
      int clientPort;
      do {
        buffer = new byte[256]; //Step 2.
        inPacket = new DatagramPacket(buffer, buffer.length); //Step 3.
        datagramSocket.receive(inPacket); //Step 4.
        clientAddress = inPacket.getAddress(); //Step 5.
        clientPort = inPacket.getPort(); //Step 5.
        messageIn = new String(inPacket.getData(), 0,
          inPacket.getLength()); //Step 6.
        messageOut = "Message";
        outPacket = new DatagramPacket(messageOut.getBytes(),
          messageOut.length(), clientAddress, clientPort); //Step 7.
        datagramSocket.send(outPacket); //Step 8.
      } while (true);
    } catch(IOException ioEx) { ... }
    finally {
      datagramSocket.close(); //Step 9.
    }
  }
}
```

UDP Client

- Setting up a server process requires eight steps:
  1. Create a `DatagramSocket` object.
  2. Create the outgoing datagram.
  3. Send the datagram message.
  4. Create a buffer for incoming datagrams.
  5. Create a `DatagramPacket` object for the incoming datagrams.
  6. Accept an incoming datagram.
  7. Retrieve the data from the buffer.
  8. Close the `DatagramSocket`.

Setting up a UDP Client Process (1/8)

- Create a DatagramSocket object.
- Similar to the creation of a DatagramSocket object in the server program, but the constructor here requires no argument.

```
DatagramSocket datagramSocket = new DatagramSocket();
```

Setting up a UDP Client Process (2/8)

- Create the outgoing datagram.
- This step is exactly as for step 7 of the server program.

```
DatagramPacket outPacket = new DatagramPacket(message.getBytes(),
  message.length(), host, PORT);
```

Setting up a UDP Client Process (3/8)

- Send the datagram message.
- By calling method send.

```
datagramSocket.send(outPacket);
```

Setting up a UDP Client Process (4-6/8)

- Exactly the same as steps 2-4 of the server procedure.
  - Create a buffer for incoming datagrams.
  - Create a `DatagramPacket` object for the incoming datagrams.
  - Accept an incoming datagram.

Setting up a UDP Client Process (7/8)

- Retrieve the data from the buffer.
- This is the same as step 6 in the server program.

```
String response = new String(inPacket.getData(), 0,
  inPacket.getLength());
```

Setting up a UDP Client Process (8/8)

- Close the `DatagramSocket`.
- By calling method `close`.

```
datagramSocket.close();
```

UDP Client Example (1/2)

```
public class UDPEchoClient {
  private static InetAddress host;
  private static fi nal int PORT = 1234;
  private static DatagramSocket datagramSocket;
  private static DatagramPacket inPacket, outPacket;
  private static byte[] buffer;
  public static void main(String[] args) {
    try {
      host = InetAddress.getLocalHost();
    } catch(UnknownHostException uhEx) { ... }

    accessServer();
  }
```

UDP Client Example (2/2)

```java
  private static void accessServer() {
    try {
      datagramSocket = new DatagramSocket(); //Step 1
      Scanner userEntry = new Scanner(System.in);
      String message = "", response = "";
      do {
        message = userEntry.nextLine();
        if (!message.equals("CLOSE")) {
          outPacket = new DatagramPacket(message.getBytes(),
            message.length(), host, PORT); //Step 2
          datagramSocket.send(outPacket); //Step 3
          buffer = new byte[256]; //Step 4
          inPacket = new DatagramPacket(buffer, buffer.length); //Step 5.
          datagramSocket.receive(inPacket); //Step 6
          response = new String(inPacket.getData(), 0,
            inPacket.getLength()); //Step 7.
        }
      } while (!message.equals("CLOSE"));
    } catch(IOException ioEx) { ... }
    finally {
      datagramSocket.close(); //Step 8.
    }
  }
}
```

Summary

Summary

- TCP-IP protocol layers: data-link, network, transport, application
- Data-link: network card
- Network layer: routing, IP, 32-bit address, 16-bit port
- Transport layer: TCP (stream, connection-oriented), UDP (datagram, connectionless)
- Sockets

Questions?