CPS221 Lecture: The Transport Layer

last revised 10/1/12

Objectives

1. To discuss the transport layer
2. To discuss Java facilities for using UDP and TCP

Materials:

1. Projectable showing place of transport layer in protocol stack
2. Projectable showing how a socket address is constructed from an IP + a port
3. Projectable of TCP state machine diagram
4. Projectables comparing connection-oriented and connectionless protocols among humans (C 6.1+6.2)
5. Projectable of common well-known port numbers
6. Executable and projectable of code for a datagram demo implemented with C++ sender on joshua (`datagramdemo.cc` and `datagramdemo`) and Java receiver (`DatagramDemo.java`).
7. "The wave" demonstration software to run on several student computers - projectable of source plus executable jar on Common volume. (Demo without protocol choice used in lab)
8. Distributed Dining Philosophers demonstration software to run on serveral student computers - projectable of source plus executable jar on Common volume. (Demo without protocol choice used in lab)

I. Introduction

A. Each of the lower layers in the network protocol stack is concerned with supporting reliable communication between systems, but at different levels.

1. The physical and data link layers support communication between hosts that are physically connected to one another.

   These layers identify hosts by physical addresses - e.g. the MAC addresses used by Ethernet

2. The network layer supports communication between hosts that may or may not be physically connected.

   a) This layer identifies hosts by IP number.

   b) Two major concerns at this layer:

      (1) Routing packets over (possibily multiple) physical links.

      (2) Mapping an IP number into the physical address of the destination system or an appropriate router along the way. (Address Resolution Protocol)

3. The transport layer supports communication between processes running on different computers. While the lower layers must exist at every node participating in a communication, this layer (and the layer(s) above it) exist only on the end-point machines.

   PROJECT: Place of transport layer in protocol stack

   a) The issue here is that a given system may be running several different processes that communicate over the network. But if the computer is using a single interface device, then packets associated with each process all use the same MAC address and ip number. So how does each packet get associated with the correct process?

      The issue here is sometimes called multiplexing - packets associated with several different processes are multiplexed over a single physical connection at the sending end and demultiplexed to the correct process at the receiving end.

   b) This layer identifies sources and destinations of message by an address consisting of the IP number of a host plus a port number that identifies a particular process running on that host.

      A port number is a 16 bit integer (and therefore in the range 1..65535)

      The combination of IP number plus port is sometimes called a socket address.

To access a web page on the college's web site:

199.97.45.22	+	80

(IP of Gordon's web server) (Default port used by web servers)

= | 199.97.45.22 80 |

(Socket address used by TCP)

PROJECT Socket Address structure

c) Four major concerns at this layer:

(1) The notion of a connection between processes (which may or may not be required)

(2) Flow control

(3) Error control

(4) Congestion control

B. Historically, TCP/IP provided two basic models of communication at this layer.

1. *Connectionless* communication using *datagrams*. (This is technically called UDP - the user datagram protocol).

a) In this model, a machine can send a message to another system (called a datagram) without first establishing a connection with that machine.

b) The network makes a good faith effort to deliver the message correctly, but there is no guarantee that the message will not be lost in transit, corrupted, or even delivered twice!

c) Further, if one machine sends several datagrams to another machine, there is no guarantee that the datagrams will arrive in the same order in which they were sent.

d) If the receiving machine wishes to reply, it may send a message back to the first machine. But this is treated as a separate message in its own right.

2. *Connection-oriented* communication using *streams*. (This is technically called TCP - transmission control protocol. Note that, in the acronym TCP/IP, IP stands for the underlying network protocol and TCP for the commonly-used stream protocol built on top of it. UDP is actually an alternative to TCP, also built on IP.)

a) In this model, a client first establishes a connection to a server.

b) While the connection remains open (i.e. until either machine closes it), either machine may send messages to the other machine.

c) If one machine sends several messages to another machine, they are guaranteed to appear to arrive in the same order as they were sent.

(1) The transport layer on the sender side includes a packet number in each packet.

(2) A "later" message that arrives before an "earlier" one is held by the TCP software at the receiving end until the "earlier" one arrives.

d) The network guarantees correct, accurate delivery of the messages.

(1) The recipient of a packet is required to acknowledge receipt of the packet in a future message to the sender. If the sender fails to receive an acknowledgement for a packet it has sent within a certain period of time, it knows that the packet was lost in transit, and can re-send it.

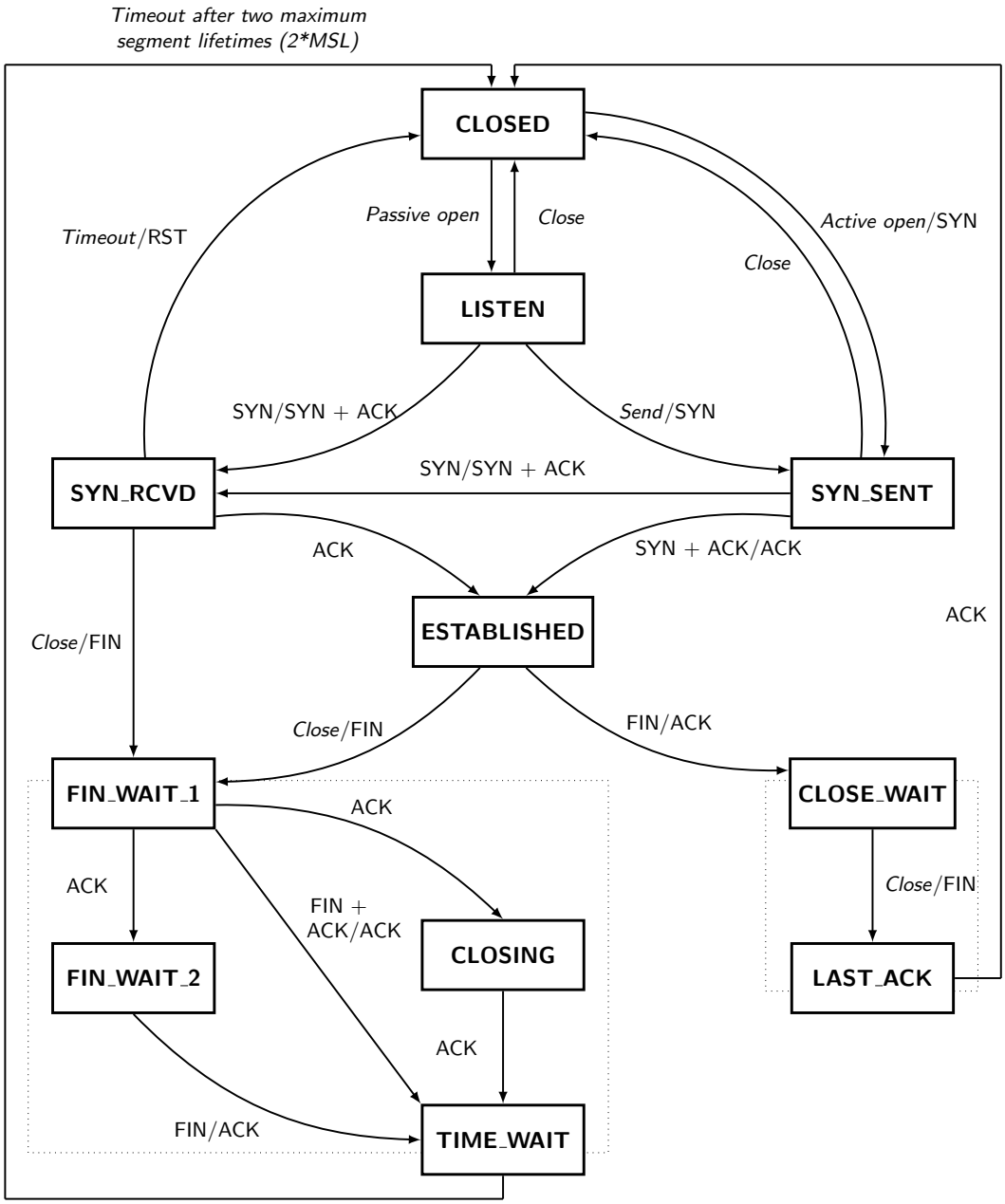
(2) The packet includes a checksum generated by the sender and checked by the receiver to verify that the packet has not been corrupted in transit. The recipient does not acknowledge a segment that it detects and error in - which will eventually lead to the sender re-sending it.

e) The receiving machine can reply to a message over the same connection on which it was sent. To keep network traffic down, acknowlegements can by "piggy-backed" on other messages, and several successive packets can be acknowledged by a single message, rather than each having to be acknowledged individually.

f) Another issue TCP deals with is flow control. Due to variations in the speed of computers and network connections, it is easily possible for a sender to send data faster than a recipient can handle it. To prevent this, TCP packets include a window size indication that lets the recipient know how much data the sender of the packet can handle from the recipient; a "sliding window" approach is used so that the recipient can indicate willingness to receive more after processing some.

g) The overall structure of a TCP session can be represented by a state machine. (This diagram shows the overall structure of a session - not the details of sending/receiving packets comprising the actual session data.

PROJECT TCP state diagram



3. The differences between these two approaches can be understood in light of different ways humans communicate.

a) Connectionless

Figure 6.2 shows how the same data would be sent using a connectionless protocol.

FIGURE 6.2
Connectionless communication.

PROJECT

b) Connection-oriented

FIGURE 6.1
Connection-oriented communication.

PROJECT

4. There are significant analogies between these two models and more familiar communication mechanisms:

    a) Connectionless networking using datagrams is similar to postal mail. When you mail a letter to another person, the Postal Service makes a good-faith effort to deliver it intact. However, there is no guarantee that the letter will not be lost or damaged in transit, and if you send several letters to the same destination they may arrive in a different order than the order in which you sent them. (One thing that the US Postal Service cannot do that the network can is to duplicate messages!).

    b) Connection-based networking streams is similar to the use of a telephone. Before you can communicate with someone else via telephone, you must first establish a connection by placing a call and having it answered. Absent serious problems with the phone system, what you say is heard accurately and completely by the other person, and in the order in which you say it. The other person can reply to you over the same connection. (The network improves on the phone system in that, if there are problems with the network, the message will still get through in the proper sequence - but maybe after some delay.)

    c) At the implementation level, connection-based networking is actually implemented by using the datagram mechanism to transmit information and acknowledgments that ensure that message delivery is done reliably.

5. A third model - SCTP (Stream Control Transfer Protocol) has recently been added to TCP/IP, though it is not yet widely supported (e.g. limited support was added to Java 1.7) It incorporates key features of the other two models.

C. We said that port numbers are 16 bit integers that lie in the range 1..65535.

1. Port numbers in the range of 1..1023 are reserved. These numbers are used by servers - that is, the server for a given service normally expects to be contacted on its well-known port.

PROJECT: Table showing some well-known port numbers

a) When using UDP, a client wishing to use a particular service simply sends a message to the well known port for that service on a machine that is thought to be running a server for it, requesting the service.

   (1) If there is a server listening on that port, it performs the requested service.

   (2) If there is no server, the message ends up being ignored and the requesting application typically times out.

b) When using TCP, a client wishing to use a particular service simply sends a message to the well known port for that service on a machine that is thought to be running a server for it, requesting the establishment of a connection.

   (1) If there is a server listening on that port, it accepts the connection and notifies the client of this fact.

   (2) If there is no server, the target system refuses the connection.

2. Ports with numbers in the range 1024 .. 49,151 are called registered ports. A company that produces a server application may request a registered port number in this range from ICANN (Internet Corporation for Assigned Names and Numbers), which will only give the number out to one user. Such a number can then be used like a well-known port number for that kind of server.

   Example: mysql uses port number 3306

3. Ports with numbers from 49,152 on up are called ephemeral ports. These numbers can be assigned temporarily by the transport layer on a host - though of course the same number cannot be used by the same host for two different things at the same time.

A typical use of an ephemeral port number is the following: A client wishes to download a web page. It sends a connection request to the appropriate web server on port 80 (the well known port for http). The connection request will specify an ephemeral port assigned by the client that will be used for the client side of the connection.

II. **Java support for the Transport Layer Services**

A. We now look at some of the support available in Java for the UDP and the TCP/IP transport layer protocols. We will look at these by looking at some example systems.
(At the present time, Java does not provide support for SCTP, though this is planned for inclusion in a forthcoming release)

B. TCP/IP implementations (including the Java implementation) are typically built around a basic abstraction known as a "socket", which represents an endpoint for communication. The Java implementations of UDP and TCP are built around different types of socket.

C. UDP

1. The primary support for UDP is found in the classes java.net.DatagramPacket and java.net.DatagramSocket

   a) Javadoc for DatagramPacket

      (1) Constructor

         (a) Buffer

         (b) host address and port only meaningful when sending

      (2) Getter and setter methods

   b) Javadoc for DatagramSocket

(1) Constructor - note options for specifying port and host address to receive messages on. (The port and host address for messages that are sent are specified in the packet that is sent.)

A reasonable question is why is there a provision for specifying a host address - since a socket is always created on a specific machine?

(a) A given machine actually has more than one IP address

i) localhost

ii) At least one IP address

iii) Sometimes more than one IP address if multihomed

(b) It is also possible to listen for broadcast messages on the broadcast address for the network or subnet

(2) receive() - waits for a message to arrive on the address/port specified when the socket was created (or after it was created by the bind() method)

(3) send() - sends a message to another system at the address/ port specified in the packet.

2. A simple example using UDP

To illustrate the language-independent nature of IP networking, we will use a program written in C++ on one machine to send a message to another program written in Java on another machine, which will simply display the message, along with information about where it came from.

a) DEMO: `datagramdemo.cc` (client) on joshua; `DatagramDemo.java` (server) on laptop. (Note: must dismiss dialog before message is sent)

(1) First demo with no one listening - note message is ignored

(2) Then demo with server listening

b) PROJECT: Code excerpts

3. A more complicated example using UDP: A distributed version of "the wave".

a) DEMO on single machine. (Specify localhost as neighbor)

b) DEMO using student computers - have several load onto machine from common volume and run.

c) Show source code for class DistributedWaveUDP, walking through constructor, doWave(), listenForMessages()

D. TCP

1. Implementations of this protocol - including the Java implementation - are built on the notion of a socket pair - a pair of connected sockets on two different machines.

a) A stream of bytes written into one member of a socket pair can be read from the other member and vice versa. (A pair of sockets supports two-way communication).

(1) A "tin can telephone" of the type you may have used as a kid is a good model for a pair of connected sockets.

(2) Note that sockets support communication at the level of streams of bytes. More complex data (numbers, objects, etc) must be reduced down to a stream of bytes in order to be sent, and must be converted back to the appropriate type of entity at the other end. This process is known as "marshaling" and "unmarshaling".

b) Connections are set up by using a “server socket”, which is a special kind of socket that waits for connection requests from another system.

(1) When an appropriate connection request is received, a socket is created on each system, and the two sockets are connected to each other.

(2) The only role of the server socket is to initiate the connection; (so the same server socket can be used to initiate any number of connections.)

(3) Typically, server programs create a single server socket which is used to receive incoming connection requests. When a connection request is received, the resulting socket is given to a separate thread to carry out the requested service while the server socket (and its thread) remain available to handle new requests.)

2. The primary Java support for TCP is found in the classes java.net.ServerSocket and java.net.Socket

a) Javadoc for ServerSocket

(1) Constructor

(2) accept() method

b) Javadoc for Socket

(1) connect() method

(2) The other member of the pair is created on the other system by the accept() method of its server socket

(3) getInputStream() + read() method of this stream

(4) getOutputStream() + write() method of this stream

3. An example using TCP: A distributed version of the Dining Philosophers

   a) DEMO on single machine. (Specify localhost as neighbor)

   b) DEMO using student computers - have several load onto machine from common volume and run.

   c) Show source code for class TCPChopstick, walking through pickup(), putdown(), constructor, listenForConnections(), listenForMessages(), sendMessage()