Washington University in St. Louis

Engineering

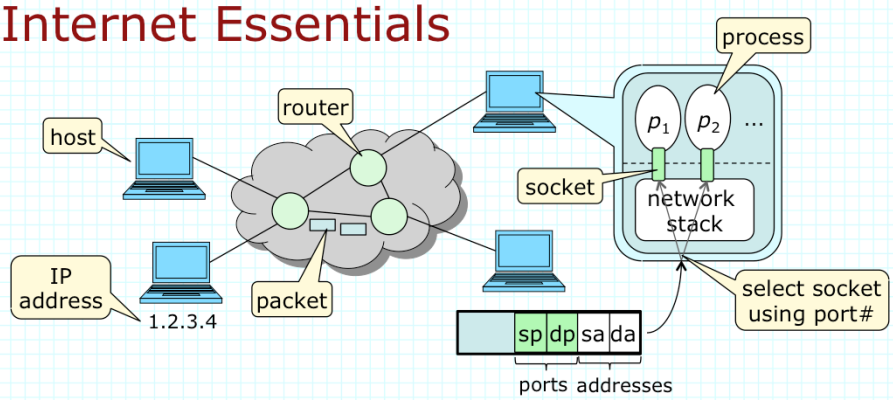
3. Introduction to the Transport Layer and Socket Programming

- Basic concepts
- The UDP transport protocol
- Building a UDP client and server in Java
- The TCP transport protocol
- Building a TCP client and server in Java

Jon Turner

Washington University in St. Louis — Engineering

Internet Essentials



- *Routers* forward *packets* among *hosts*
- *Internet Protocol* (IP) provides *best-effort delivery* of *datagram* packets based on *addresses* in packet *headers*
- *Processes* use *sockets* as interface to *network stack*
- *Transport protocols* define *port numbers* used to identify specific sockets

2

Transport Services and Protocols

- Enable communication between *application processes* running on different hosts
  - » most basic function is extending addressing beyond hosts
  - » in Internet, *port numbers* are used for this purpose
- Transport protocols run in end systems
  - » although in Internet, some network components (e.g. NATs) do examine and even modify transport fields
- Two principal transport protocols in Internet
  - » UDP – provides datagram interface, port numbers and some limited error detection
  - » TCP – provides byte-stream interface, port numbers, reliable data transport, flow control, congestion control
  - » neither provides bandwidth or delay guarantees
    - since neither does the underlying network

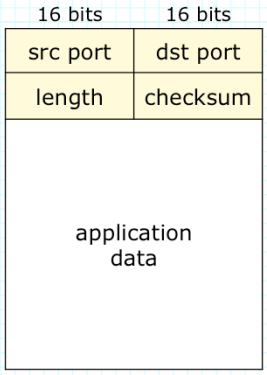
Transport Layer Multiplexing

- Multiplexing is fancy word for "sharing"
  - » here, we are sharing an IP address among multiple programs
  - » need mechanism to identify individual programs – port numbers
  - » running programs communicate through sockets and transport protocols map port numbers to sockets
- Connectionless multiplexing
  - » to send packet to a remote program, must know the port number it is using, in addition to its host's IP address
  - » when a UDP packet arrives at a host, the *destination port #* is used to identify the socket that is to receive the packet
    - the destination port is the only information used to identify socket
    - so, packets from different remote hosts can be delivered to the same socket
  - » the remote program uses the *source port #* of received packet when sending a reply back to the sender

4

UDP Details

- Port number are 16 bits long
- Length specifies number of bytes in UDP "segment", including header
- Checksum is used to detect errors in UDP segment
  - » sender computes checksum by adding 16 bit chunks
    - on arithmetic overflow, sum is incremented
  - » receiver checks for errors by re-computing sum and comparing to value in packet
  - » since IP runs over multiple different link layer protocols, it cannot rely on link layer for error detection
    - even though link layer routinely implements error detection



Exercises

1. What services does TCP provide that UDP does not?
2. Name three services that neither UDP nor TCP provide.
3. Suppose a host is running two programs *A* and *B* that both communicate over the internet using UDP. If the socket for program *A* is mapped to port number 6789, is it possible for the socket for program *B* to also be mapped to port number 6789?
4. When a UDP server receives a packet from a client program and replies to the client, how does it know where to send the packet?
5. Suppose a UDP payload contains the 32 ASCII characters that form the string "**happy daisies are here again**". Show how you can change just two bits in the payload in a way that leaves the UDP checksum unchanged. What string do you get in this case?

Simple UDP Echo Server in Java

```
import java.io.*; import java.net.*;
public class UdpServer {
   public static void main(String args[]) throws Exception {
      // open datagram socket on port 9876
      DatagramSocket sock = new DatagramSocket(9876);
      // create two packets sharing a common buffer
      byte[] buf = new byte[1000];
      DatagramPacket inPkt = new DatagramPacket(buf, buf.length);
      DatagramPacket outPkt = new DatagramPacket(buf, buf.length);
      while (true) {
         // wait for incoming packet
         sock.receive(inPkt);
         // set address, port and length fields of outPkt
         // so as to return contents of inPkt to sender
         outPkt.setAddress(inPkt.getAddress());
         outPkt.setPort(inPkt.getPort());
         outPkt.setLength(inPkt.getLength());
         // and send it back
         sock.send(outPkt);
}  }  }
```

UDP Echo Client

```
import java.io.*; import java.net.*;
public class UdpClient {
   public static void main(String args[]) throws Exception {
      // get server address and opensocket
      InetAddress serverAdr = InetAddress.getByName(args[0]);
      DatagramSocket sock = new DatagramSocket();
      // build packet addressed to server, then send it
      byte[] outBuf = args[1].getBytes("US-ASCII");
      DatagramPacket outPkt = new
             DatagramPacket(outBuf,outBuf.length,serverAdr,9876);
      sock.send(outPkt);
      // create buffer and packet for reply, then receive it
      byte[] inBuf = new byte[1000];
      DatagramPacket inPkt = new DatagramPacket(inBuf,inBuf.length);
      sock.receive(inPkt);
      // print buffer contents and close socket
      String reply = new String(inBuf,0,inPkt.getLength(),"US-ASCII");
      System.out.println(reply);
      sock.close();
}  }
```

Recap of Classes/Methods Used

- DatagramSocket (defined in java.net)
  - » used to read/write packets
  - » opened by constructor, closed using close() method
- DatagramPacket (in java.net)
  - » defines packet with payload defined by array of bytes
    - in Java, a byte is not the same thing as a char
    - encode strings as bytes using String's getBytes method with a specified encoding
    - convert byte arrays to Strings using String constructor with specified encoding
  - » address and port fields specify peer address/port
  - » length field is # of bytes to send in packet, or # received
- InetAddress (in java.net)
  - » use static method *getByName*(addressString) to convert a destination host's name to its internet address

Exercises

- In UdpEchoServer, two packet objects are used. Do you think this is necessary? If so, explain why. If not, show how you can modify the code to use just one packet object.
- UdpEchoClient uses the US-ASCII method to encode Strings as bytes. If you run it using the command

```
java UdpEchoClient serverHostName "hi there"
```

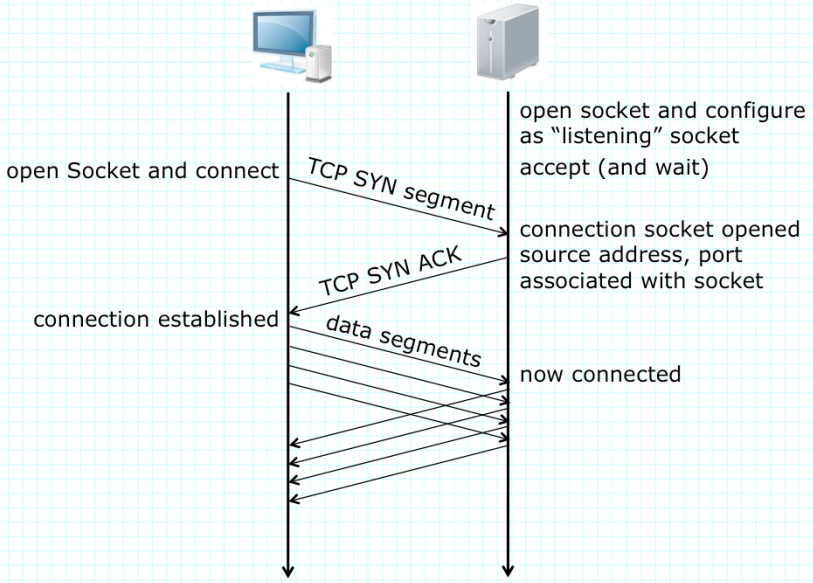
how many bytes are carried in the payload of the UDP packet? Suppose you modified the program to use UTF-16, in place of US-ASCII, how many bytes would be carried in the UDP payload in this case? Try it.

Key TCP Concepts

- To provide reliable communication, the TCP networking software at both communicating hosts must cooperate
  - » requires information that must be maintained consistently at both ends
  - » to allow the network software to initialize this information, communicating processes must first "establish a connection"
- Two kinds of sockets in TCP
  - » *listening socket* used by server to make itself available for connection requests
    - any host in the internet can send a connection request to a listening socket (assuming it knows the IP address and port#)
  - » *connection socket* used by a client-server pair to exchange data
    - identified by "4-tuple": src address/port#, dest address/port#
    - server can have multiple connection sockets using same port#
    - typically, each connection socket handled by separate thread

Washington University in St. Louis — Engineering

TCP Connection Establishment



12

TCP Echo Server

```
import java.io.*; import java.net.*;
public class TcpServer {
  public static void main(String args[]) throws Exception {
      // create listen socket and buffer
      ServerSocket listenSock = new ServerSocket(6789);
      byte[] buf = new byte[1000];
      while (true) {
            // wait for connection request; create connection socket
            Socket connSock = listenSock.accept();
            // create buffered versions of socket's in/out streams
            BufferedInputStream   in = new BufferedInputStream(
                                       connSock.getInputStream());
            BufferedOutputStream out = new BufferedOutputStream(
                                       connSock.getOutputStream());
            while (true) { // read and echo until peer closes
                  int nbytes = in.read(buf, 0, buf.length);
                  if (nbytes < 0) break;
                  out.write(buf,0,nbytes); out.flush();
            }
            connSock.close();
} }   }
```

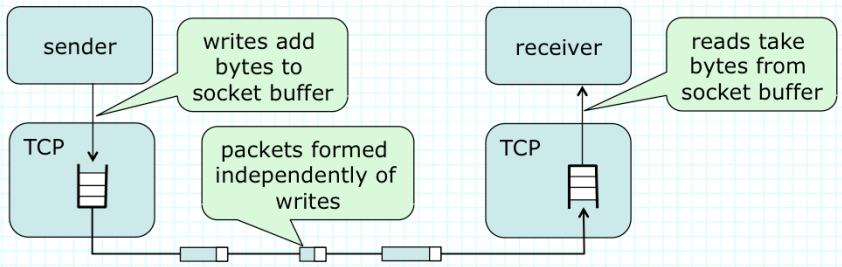
TCP Echo Client

```
import java.io.*; import java.net.*;
public class TcpClient {
  public static void main(String args[]) throws Exception {
    // open socket and connect to server
    Socket sock = new Socket(args[0], 6789);

    // create buffered reader & writer for socket's in/out streams
    BufferedReader  in = new BufferedReader(new InputStreamReader(
               sock.getInputStream(),"US-ASCII"));
    BufferedWriter out = new BufferedWriter(new OutputStreamWriter(
               sock.getOutputStream(),"US-ASCII"));

    // output second argument to server, adding a newline
    // as delimiter; flush all data from buffer to network
    out.write(args[1]); out.newLine(); out.flush();

    // read data up to end of line and close connection
    System.out.println(in.readLine());
    sock.close();
} }
```

Additional Classes/Methods Used

- ServerSocket (in java.net)
  - » used to listen for incoming connection requests
- Socket (in java.net)
  - » used for connection sockets
  - » associated input and output streams for transferring *bytes*
- BufferedInputStream/BufferedOutputStream (in java.io)
  - » buffers *bytes* in user space for more efficient data transfers
- InputStreamReader/OutputStreamWriter (in java.io)
  - » for converting between bytes and chars/Strings
  - » can use variety of encoding methods
- BufferedReader/BufferedWriter (in java.io)
  - » buffers *chars* in user space for more efficient transfers
  - » *readLine*() method simplifies parsing of input

TCP Provides a Byte Stream Interface



- Bytes that are written together may not be transferred together, so don't expect single read to get them all
- If receiver is slow to read bytes
  - » receiver's socket buffer fills, causing it to flow-control sender
  - » when sender's socket buffer fills, additional writes block, suspending application program
  - » bytes are transferred as space becomes available at receiver

Formatting a Byte Stream

- Since no visible packet boundaries (as with datagrams), sending program must format data to assist receiver
- For character data, common technique is to use newline character (or CR,LF pair) as delimiter
  - » receiver reads complete lines, then parses contents of each line
- For binary data, simplest approach is fixed format
  - » receiver relies on pre-arranged format when reading data
  - » can send a variable length array by first sending a length field
    - receiver first reads length, then reads as many additional items as specified by the length field
- Many other possibilities, but there must be some well-defined format that receiver can depend on

Network and Host Byte Order

- Two common conventions for representing multi-byte values in a computer's memory
  - » little-endian – low order byte comes first in memory
    - so 32 bit hex value 0xabcdef12 is stored 12, ef, cd, ab
  - » big-endian – high order byte comes first
- In the internet, high-order byte always sent first
  - » to ensure this, may need to reformat data before sending a packet (and after receiving it)
  - » Java provides mechanisms to handle this automatically
    - DataInputStream/DataOutputStream (in java.io) for TCP sockets

      declare by "wrapping" BufferedInputStream/OutputStream

```
DataInputStream in = new DataInputStream(
  new BufferedInputStream(connSock.getInputStream()));
```

      use provided methods to write/read binary data

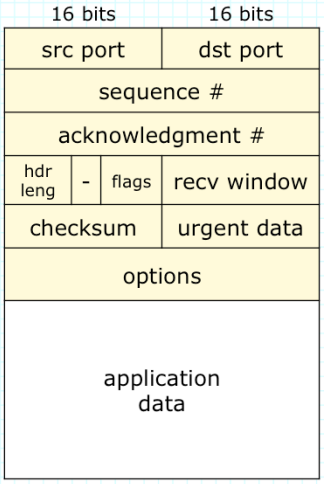
```
int i = in.readInt(); float x; out.writeFloat(x);
```

    - ByteBuffers (in java.nio) for UDP sockets

18

TCP Segment Format

- Port numbers are 16 bits long
- Length specifies number of 32 bit words in TCP header (typically 5)
- Flags include
  - » SYN and FIN bits used to setup and teardown connections
- Checksum is used to detect errors in TCP segment
  - » sender computes checksum by adding 16 bit chunks
    - on arithmetic overflow, sum is incremented
  - » receiver re-computes sum and compares to value in packet
- Defer discussion of other fields



19

Exercises

1. How many sockets does a UDP server need to communicate with 100 remote clients concurrently?
   How many port numbers does it need?
2. How many sockets does a TCP server need to communicate with 100 remote clients concurrently?
   How many port numbers does it need?
3. How does the network stack identify the socket to which an incoming TCP packet is to be delivered?