

The Application Layer: Sockets

Smith College, CSC 249
February 19, 2008

slides mostly from J.F Kurose and K.W. Ross,
copyright 1996-2007

Overview

- Socket Programming
 - ❖ Terminology
 - ❖ TCP and UDP
 - ❖ Programming examples
- Homework and Lab review
- Class project

2

Homework 1 Feedback

- Width / Size of a bit
 - ❖ Signal edges - rise and falltime
- Watch units
 - ❖ Bits v. bytes
 - ❖ Decimal v. binary values

3

Homework 2 Feedback

- R6: Complete transaction as fast as possible - TCP v. UDP?
 - ❖ TCP has handshaking - always
 - ❖ TCP uses flow and cong. control only sometimes
- R13: Web caching speed improvements
 - ❖ Can improve for *all* requests because the decreased traffic on the link relieves the bottlenecking
- R14: telnet with "If-modified-since:"
 - ❖ Some left off the "GMT" on specified time

4

Homework 3 Feedback

- ❑ P17 - Consider a distribution scheme for P2P to achieve specified times
 - ❖ Thought experiment
 - ❖ Parallel distribution will achieve results
- ❑ Overlay network
 - ❖ Routers are not part of the overlay - only edge hosts and their abstract links
 - ❖ Joining discussed on page 155
 - Compiles list of other peers through response (pong) messages
 - Must find starting point: bootstrap itself or use tracker site...

5

Chapter 2: Application layer

- ❑ 2.1 Principles of network applications
- ❑ 2.2 Web and HTTP
- ❑ 2.3 FTP
- ❑ 2.4 Electronic Mail
 - ❖ SMTP, POP3, IMAP
- ❑ 2.5 DNS
- ❑ 2.6 P2P file sharing
- ❑ 2.7 **Socket programming with TCP**
- ❑ 2.8 Socket programming with UDP

6

Sockets Analogy

- ❑ <Socket programming for Internet applications> ..is analogous to..
<_____ type of lines of code for single host applications...>
- ❑ Operating system calls in any standard language
- ❑ Operating systems to what for us?
 - ❖ File, device, memory... management
 - ❖ Print; read from file...

7

Socket programming

Goal: learn how to build client/server applications that communicate using sockets

Socket API

- ❑ client/server paradigm
- ❑ two types of transport service via socket API:
 - ❖ unreliable datagram
 - ❖ reliable, byte stream-oriented

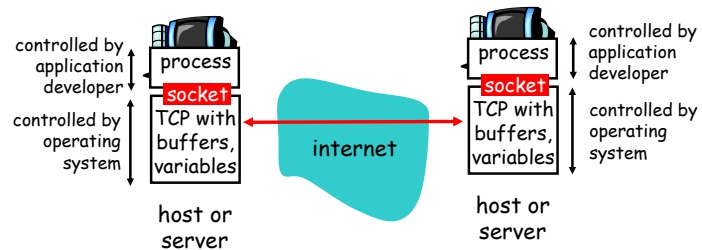
socket
a *local host, application-created, OS-controlled* interface into which an application process can **both send and receive** messages to/from another application process

8

Socket-programming using TCP

Socket: an interface between application process and end-end-transport protocol (UCP or TCP)

TCP service: reliable transfer of **bytes** from one process to another



9

Socket programming *with TCP*

Client must contact server, but first...

- ❑ server process must first be running
- ❑ server must have created socket (door) that welcomes client's contact

Client contacts server by:

- ❑ creating local client TCP socket
- ❑ specifying IP address, port number of server process
- ❑ When **client creates socket**: client TCP establishes connection to server TCP

10

Socket programming *with TCP*

- ❑ When contacted by client, **server TCP creates new socket** for server process to communicate with client
 - ❖ allows server to talk with multiple clients

application viewpoint
TCP provides reliable, in-order transfer of bytes ("pipe") between client and server

11

Socket programming *Activity*

- ❑ Handout on TCP Client-Server:
 - ❖ 3 pages with questions and code
 - ❖ Answer questions (identify lines of code)
 - ❖ Map the lines of code to the steps in the flowchart

12

Stream jargon

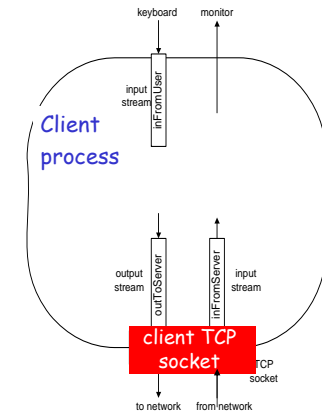
- A **stream** is a sequence of characters that flows into or out of a process.
- An **input stream** is attached to some input source for the process, e.g., keyboard or socket.
- An **output stream** is attached to an output source, e.g., monitor or socket.

13

Socket programming with TCP

Client-server application ex:

- 1) The client reads a line from standard input (`inFromUser` stream), sends it to server via socket (`outToServer` stream)
- 2) The server reads the line from its socket
- 3) The server converts the line to uppercase, and sends it back to the client
- 4) The client reads the modified line from its socket (`inFromServer` stream) and prints it to standard output

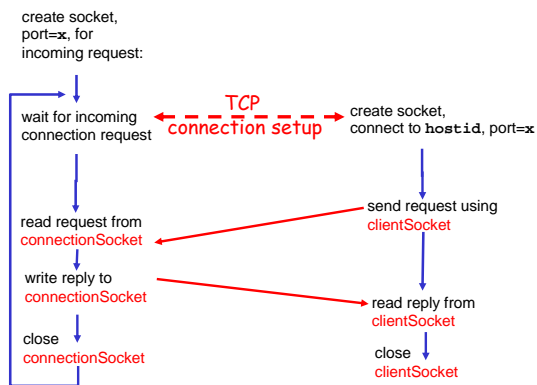


14

Client/server socket interaction: TCP

Server (running on `hostid`)

Client

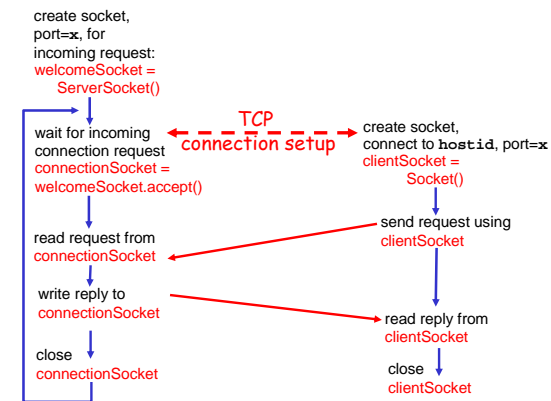


15

Client/server socket interaction: TCP

Server (running on `hostid`)

Client



16

Example: Java client (TCP)

```
import java.io.*;
import java.net.*;
class TCPClient {

    public static void main(String argv[]) throws Exception
    {
        String sentence;
        String modifiedSentence;

        Create input stream → BufferedReader inFromUser =
        Create client socket, initiate TCP connect to server (DNS lookup) → new BufferedReader(new InputStreamReader(System.in));
        Socket clientSocket = new Socket("hostname", 6789);
        Create output stream attached to socket → DataOutputStream outToServer =
        new DataOutputStream(clientSocket.getOutputStream());

    }
}
```

17

Example: Java client (TCP), cont.

```
BufferedReader inFromServer =
new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));

        sentence = inFromUser.readLine();

        Send line to server → outToServer.writeBytes(sentence + '\n');

        Read line from server → modifiedSentence = inFromServer.readLine();
        System.out.println("FROM SERVER: " + modifiedSentence);

        clientSocket.close();

    }
}
```

18

Example: Java server (TCP)

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

class TCPServer {

    public static void main(String argv[]) throws Exception
    {
        String clientSentence;
        String capitalizedSentence;

        Create welcoming socket at port 6789 → ServerSocket welcomeSocket = new ServerSocket(6789);

        Wait, on welcoming socket for contact by client → while(true) {
            Socket connectionSocket = welcomeSocket.accept();

            Create input stream, attached to socket → BufferedReader inFromClient =
            new BufferedReader(new
            InputStreamReader(connectionSocket.getInputStream()));

    }
}
```

19

Example: Java server (TCP), cont

```
        Create output stream, attached to socket → DataOutputStream outToClient =
        new DataOutputStream(connectionSocket.getOutputStream());

        Read in line from socket → clientSentence = inFromClient.readLine();

        capitalizedSentence = clientSentence.toUpperCase() + '\n';

        Write out line to socket → outToClient.writeBytes(capitalizedSentence);

    }
}

End of while loop, loop back and wait for another client connection
```

20

Discussion Questions

- What is the role of each of the sockets?
- When is the TCP connection established?
 - ❖ What does it mean to establish a TCP connection?
- When is the DNS lookup performed?
- When is the welcome socket closed?

21

Discussion Questions

- Where is "the application" in the TCP client and server code segments?
 - ❖ Which lines are for the socket programming, that you could reuse for other applications?
 - ❖ Which lines are the application?
- What is a socket?

22

Chapter 2: Application layer

- 2.1 Principles of network applications
- 2.2 Web and HTTP
- 2.3 FTP
- 2.4 Electronic Mail
 - ❖ SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 P2P file sharing
- 2.7 Socket programming with TCP
- 2.8 Socket programming with UDP

23

Socket programming *with UDP*

UDP: no "connection" between client and server

- no handshaking
- sender explicitly attaches IP address and port of destination to each packet
- server must extract IP address, port of sender from received packet

UDP: transmitted data may be received out of order, or lost

application viewpoint

UDP provides unreliable transfer of groups of bytes ("datagrams") between client and server

24

Client/server socket interaction: UDP

Server (running on `hostid`)

```
create socket,
port=x, for
incoming request:
serverSocket =
DatagramSocket()
```

```
read request from
serverSocket
```

```
write reply to
serverSocket
specifying client
host address,
port number
```

Client

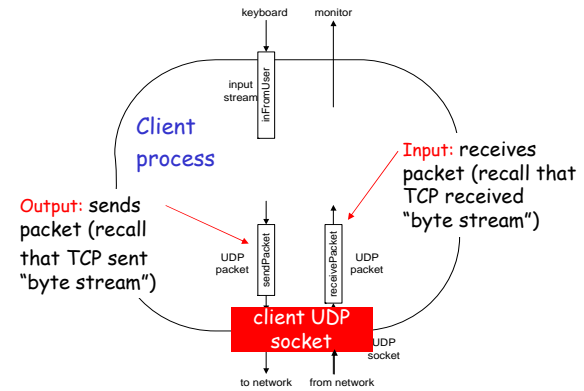
```
create socket,
clientSocket =
DatagramSocket()
```

```
Create, address (hostid, port=x,
send datagram request
using clientSocket
```

```
read reply from
clientSocket
```

```
close
clientSocket
```

Example: Java client (UDP)



25

26

Example: Java client (UDP)

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

class UDPClient {
    public static void main(String args[]) throws Exception
    {
        Create input stream → BufferedReader inFromUser =
        new BufferedReader(new InputStreamReader(System.in));
        Create client socket (no TCP connection) → DatagramSocket clientSocket = new DatagramSocket();
        Translate hostname to IP address using DNS → InetAddress IPAddress = InetAddress.getByName("hostname");
        byte[] sendData = new byte[1024];
        byte[] receiveData = new byte[1024];

        String sentence = inFromUser.readLine();
        sendData = sentence.getBytes();
```

27

Example: Java client (UDP), cont.

```
Create datagram with data-to-send, length, IP addr, port → DatagramPacket sendPacket =
new DatagramPacket(sendData, sendData.length, IPAddress, 9876);
Send datagram to server → clientSocket.send(sendPacket);
Read datagram from server, client idling → DatagramPacket receivePacket =
new DatagramPacket(receiveData, receiveData.length);
clientSocket.receive(receivePacket);
String modifiedSentence =
new String(receivePacket.getData());
System.out.println("FROM SERVER:" + modifiedSentence);
clientSocket.close();
}
```

28

Example: Java server (UDP)

```

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

class UDPServer {
    public static void main(String args[]) throws Exception
    {
        DatagramSocket serverSocket = new DatagramSocket(9876);

        byte[] receiveData = new byte[1024];
        byte[] sendData = new byte[1024];

        while(true)
        {
            DatagramPacket receivePacket =
                new DatagramPacket(receiveData, receiveData.length);
            serverSocket.receive(receivePacket);
        }
    }
}

```

Create datagram socket at port 9876 → DatagramSocket serverSocket = new DatagramSocket(9876);
 Placeholders for the packets → byte[] receiveData = new byte[1024];
 byte[] sendData = new byte[1024];
 Create space for received datagram → DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);
 Receive datagram Socket idling → serverSocket.receive(receivePacket);

29

Example: Java server (UDP), cont

```

String sentence = new String(receivePacket.getData());

InetAddress IPAddress = receivePacket.getAddress();
int port = receivePacket.getPort();

String capitalizedSentence = sentence.toUpperCase();

sendData = capitalizedSentence.getBytes();

DatagramPacket sendPacket =
    new DatagramPacket(sendData, sendData.length, IPAddress,
        port);

serverSocket.send(sendPacket);
}
}

```

Get IP addr port #, of sender → InetAddress IPAddress = receivePacket.getAddress();
 int port = receivePacket.getPort();
 Create datagram to send to client → DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length, IPAddress, port);
 Write out datagram to socket → serverSocket.send(sendPacket);
 End of while loop, loop back and wait for another datagram

30

Discussion Questions

- Differences between UDP and TCP?

31

Discussion Questions

- Because UDP is connectionless, there is no socket to perpetually listen
 - ❖ We may begin the client process first (it does not immediately contact the server)
- There is no initial handshaking with UDP
- UDP client creates a datagram socket
- No "streams" are attached to the sockets with UDP
- Sending hosts create packets by attaching IP dest. addr and port number to each batch of bytes it sends
 - ❖ The receiving process must parse each packet to obtain the packet's information bytes

32

Socket Summary

- Sockets defined
 - ❖ Through analogy to a "door"
 - An interface between the application and the Internet
 - ❖ Through examples with socket-API programming
 - ❖ Through understanding the differences between TCP and UDP sockets

33

Wireshark Labs Feedback

- HTTP Lab #17: Parallel v. Series
 - ❖ Checking timestamp on messages
 - Parallel vs. Pipelining concepts
 - Requests sent serially, so are downloaded serially
 - Small difference in timestamps is not significant in 'human time' but is significant for timing on the Internet
 - ❖ Check if there are distinct TCP ports
 - 2 TCP connections indicates serial download
 - vs. 1 connection with parallel paths

35

Chapter 2: Summary

Protocols

- Typical request/reply message exchange:
 - ❖ client requests info or service
 - ❖ server responds with data, status code
 - Message formats:
 - ❖ headers: fields giving info about data
 - ❖ data: info being communicated
- Important themes:*
- control vs. data msgs
 - ❖ in-band, out-of-band
 - centralized vs. decentralized
 - stateless vs. stateful
 - reliable vs. unreliable message transfer
 - "complexity at network edge"

34

Wireshark Labs Feedback

- DNS Lab
 - ❖ Third requested nslookup can be done via
 - nslookup mail.yahoo.com <your-DNS-server>
 - ❖ Type A "Class" field
 - "Class: IN " simply indicates the class is the INternet

36

Project Overview

- ❑ The objective of the course project is to learn and then teach the class about a selected application,
- ❑ ...In terms of the network layer model
- ❑ ...Including details of how the application functions,
- ❑ ...And issues of security, privacy and future developments

37

Project Elements

- ❑ Select an application that we have not discussed in class, and that is not discussed in the text
- ❑ Research this application so that you can describe how it works.
 - ❖ Explain what the application is, how and why people would want to use it,
 - ❖ Specify/identify the protocol(s) used
 - ❖ Include details about the protocols - pros, cons and unique/interesting features,
 - Note that many protocols are proprietary...
 - ❖ Explain the details of how messages between end hosts are transmitted through the internet (chapters 3, 4, 5, ...)
 - ❖ Highlight any privacy, security and/or legal issues surrounding this application
- ❑ Include a bibliography, that has at least two technical, peer reviewed references
- ❑ Present your findings to the class at the end of the semester in a short (5 – 10 minute) presentation

39

Project Topic Possibilities

- ❑ a specific peer-to-peer application,
- ❑ an emerging instant messaging application,
- ❑ chats,
- ❑ an advance in voice over IP,
- ❑ Ambient Devices "orb" that glows for stock prices,
- ❑ IP-addressable home appliances,
- ❑ sensor networks for public spaces or for endangered habitats,
- ❑ video streaming...
- ❑ Be creative and think about hunting down an application that uses the internet in unexpected and unforeseen ways.

38

Project Elements - for Friday

- ❑ Select an application that we have not discussed in class, and that is not discussed in the text
- ❑ Research this application so that you can describe how it works.
 - ❖ Explain what the application is, how and why people would want to use it,
 - ❖ Specify/identify the protocol(s) used
 - ❖ Include details about the protocols - pros, cons and unique/interesting features,
 - Note that many protocols are proprietary...
 - ❖ Explain the details of how messages between end hosts are transmitted through the internet (chapters 3, 4, 5, ...)
 - ❖ Highlight any privacy, security and/or legal issues surrounding this application
- ❑ Include a bibliography, that has at least two technical, peer reviewed references - One reference for Friday
- ❑ Present your findings to the class at the end of the semester in a short (5 – 10 minute) presentation + a ~5 page paper

40