Last time

- Fast retransmit
  - 3 duplicate ACKs
- Flow control
  - Receiver windows
- Connection management
  - SYN/SYNACK/ACK, FIN/ACK, TCP states
- Congestion control
  - General concepts
- TCP congestion control
  - AIMD, slow start, congestion avoidance

This time

- TCP
  - Throughput
  - Fairness
  - Delay modeling
- TCP socket programming

TCP sender congestion control

State	Event	TCP Sender Action	Commentary
Slow Start (SS)	ACK receipt for previously unacked data	CongWin = CongWin + MSS, If (CongWin > Threshold) set state to "Congestion Avoidance"	Resulting in a doubling of CongWin every RTT
Congestion Avoidance (CA)	ACK receipt for previously unacked data	CongWin = CongWin+MSS * (MSS/CongWin)	Additive increase, resulting in increase of CongWin by 1 MSS every RTT
SS or CA	Loss event detected by triple duplicate ACK	Threshold = CongWin/2, CongWin = Threshold, Set state to "Congestion Avoidance"	Fast recovery, implementing multiplicative decrease. CongWin will not drop below 1 MSS.
SS or CA	Timeout	Threshold = CongWin/2, CongWin = 1 MSS, Set state to "Slow Start"	Enter slow start
SS or CA	Duplicate ACK	Increment duplicate ACK count for segment being acked	CongWin and Threshold not changed

TCP Throughput

- What’s the average throughout of TCP as a function of window size and RTT?
  - Ignore slow start
- Let W be the window size when loss occurs.
- When window is W, throughput is W/RTT
- Just after loss, window drops to W/2, throughput to W/2RTT.
- Average throughout: .75 W/RTT

TCP Futures: TCP over "long, fat pipes"

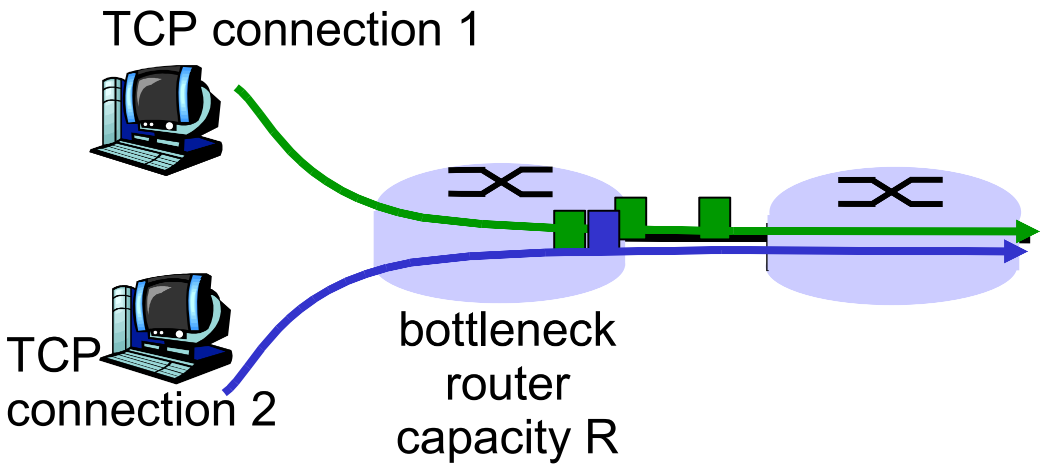
- Example: 1500 byte segments, 100ms RTT, want 10 Gbps throughput
- Requires window size W = 83,333 in-flight segments
- Throughput in terms of loss rate:

$$\frac{1.22 \cdot MSS}{RTT\sqrt{L}}$$

- ➜ L = $2 \cdot 10^{-10}$ *Wow*
- New versions of TCP for high-speed needed!

TCP Fairness

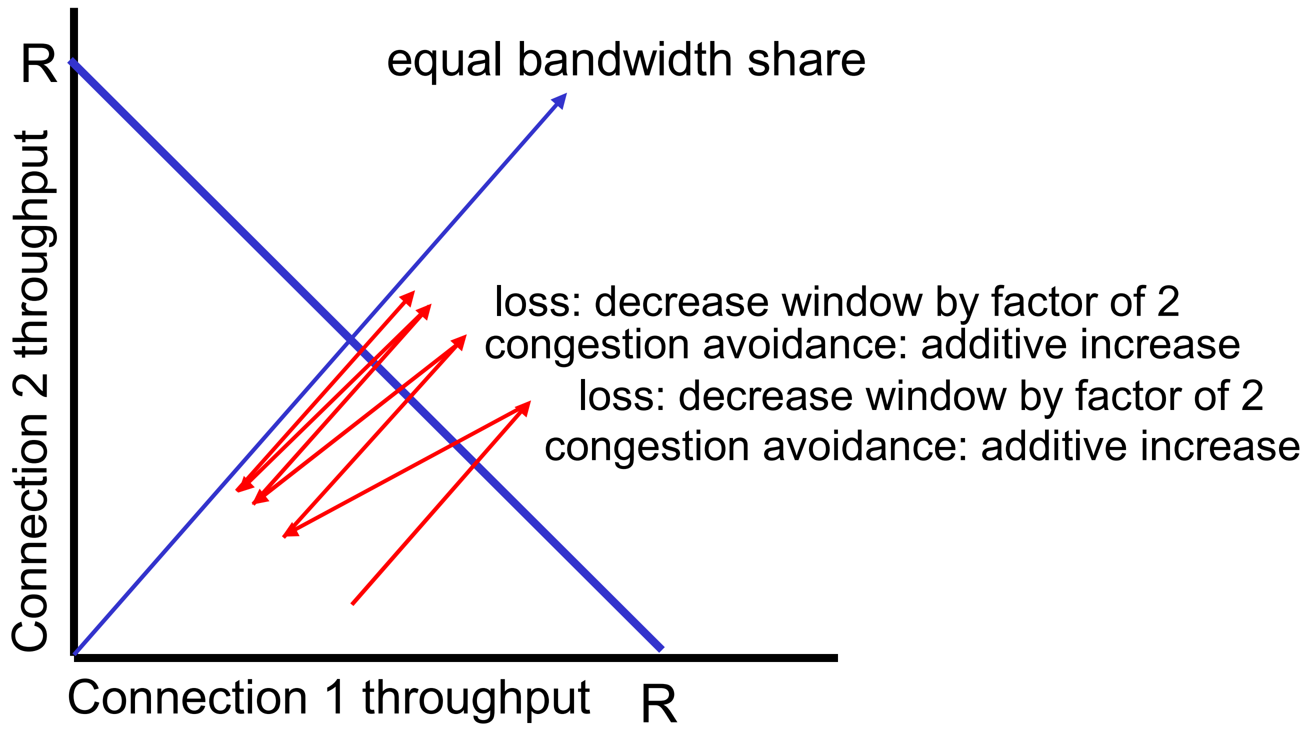
Fairness goal: if K TCP sessions share same bottleneck link of bandwidth R, each should have average rate of R/K



Why is TCP fair?

Two competing sessions:

- Additive increase gives slope of 1, as throughout increases
- multiplicative decrease decreases throughput proportionally


R
equal bandwidth share
loss: decrease window by factor of 2
congestion avoidance: additive increase
loss: decrease window by factor of 2
congestion avoidance: additive increase
Connection 2 throughput
Connection 1 throughput
R

Fairness (more)

Fairness and UDP

- Multimedia apps often do not use TCP
  - do not want rate throttled by congestion control
- Instead use UDP:
  - pump audio/video at constant rate, tolerate packet loss
- Research area: TCP friendly

Fairness and parallel TCP connections

- Nothing prevents app from opening parallel connections between 2 hosts.
- Web browsers do this
- Example: link of rate R supporting 9 connections;
  - new app asks for 1 TCP, gets rate R/10
  - new app asks for 11 TCPs, gets R/2 !

Delay modeling

Q: How long does it take to receive an object from a Web server after sending a request?

Ignoring congestion, delay is influenced by:

- TCP connection establishment
- data transmission delay
- slow start

Notation, assumptions:

- Assume one link between client and server of rate R
- S: MSS (bits)
- O: object size (bits)
- no retransmissions (no loss, no corruption)

Window size:

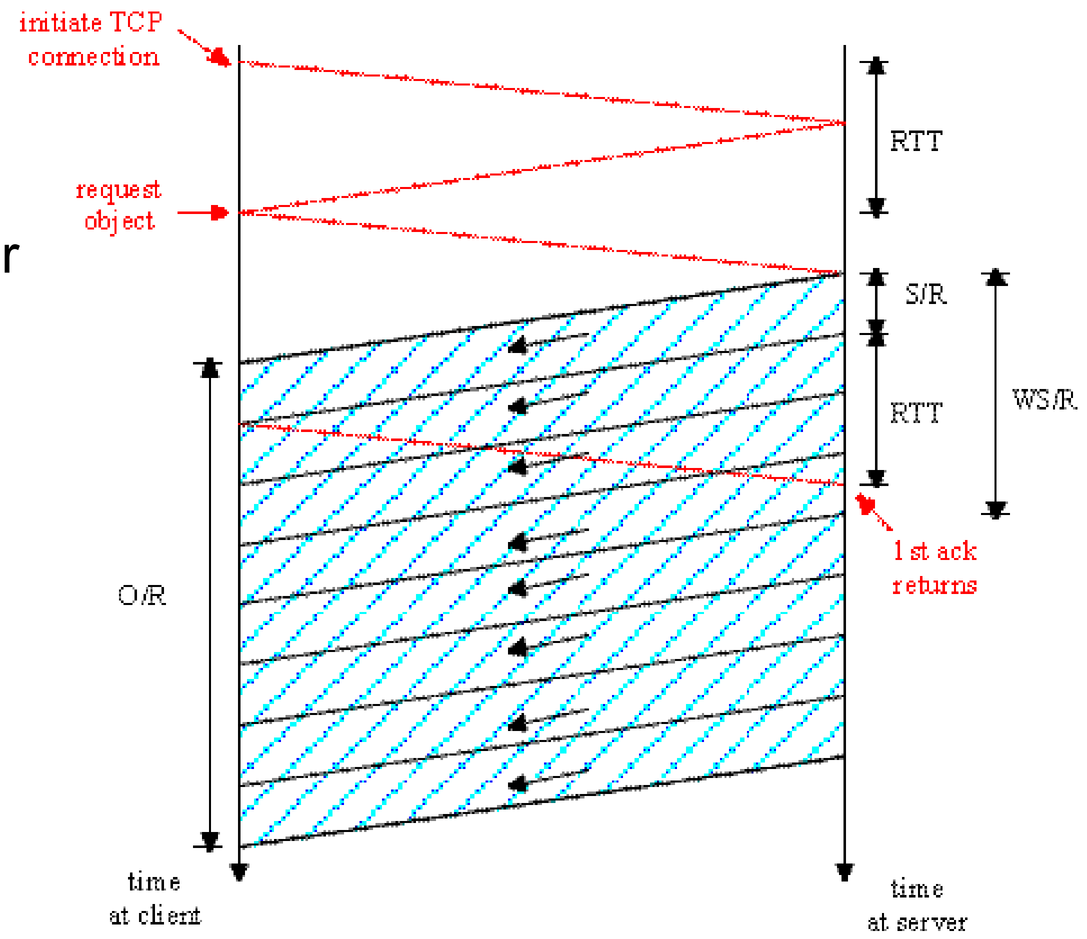
- First assume: fixed congestion window, W segments
- Then dynamic window, modeling slow start

Fixed congestion window (1)

First case:

WS/R > RTT + S/R: ACK for first segment in window returns before window's worth of data sent

delay = 2RTT + O/R

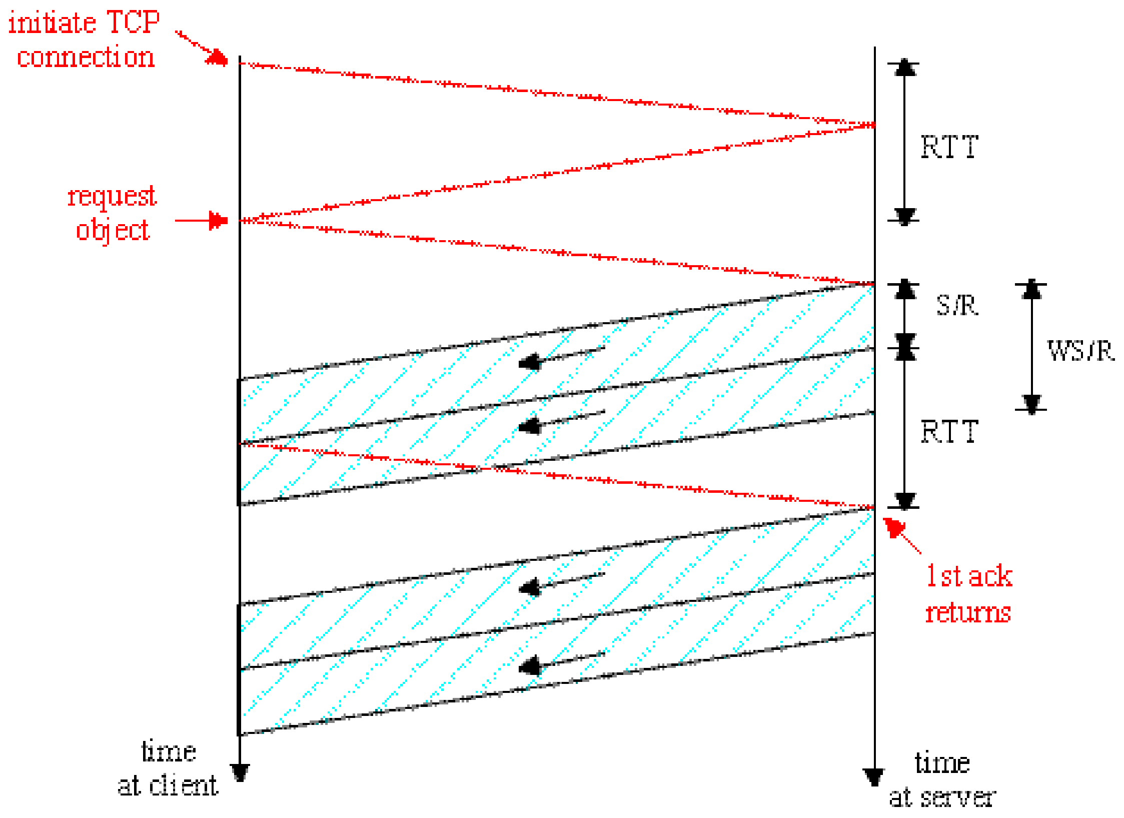


Fixed congestion window (2)

Second case:

- WS/R < RTT + S/R: wait for ACK after sending window's worth of data sent

delay = 2RTT + O/R
+ (K-1)[S/R + RTT - WS/R]



TCP Delay Modeling: Slow Start (1)

Now suppose window grows according to slow start

Will show that the delay for one object is:

$$Latency = 2\,RTT + \frac{O}{R} + P\left[RTT + \frac{S}{R}\right] - (2^P - 1)\frac{S}{R}$$

where *P* is the number of times TCP idles at server:

$$P = \min\{Q, K-1\}$$

- where Q is the number of times the server idles
  if the object were of infinite size.

- and K is the number of windows that cover the object.

TCP Delay Modeling: Slow Start (2)

Delay components:

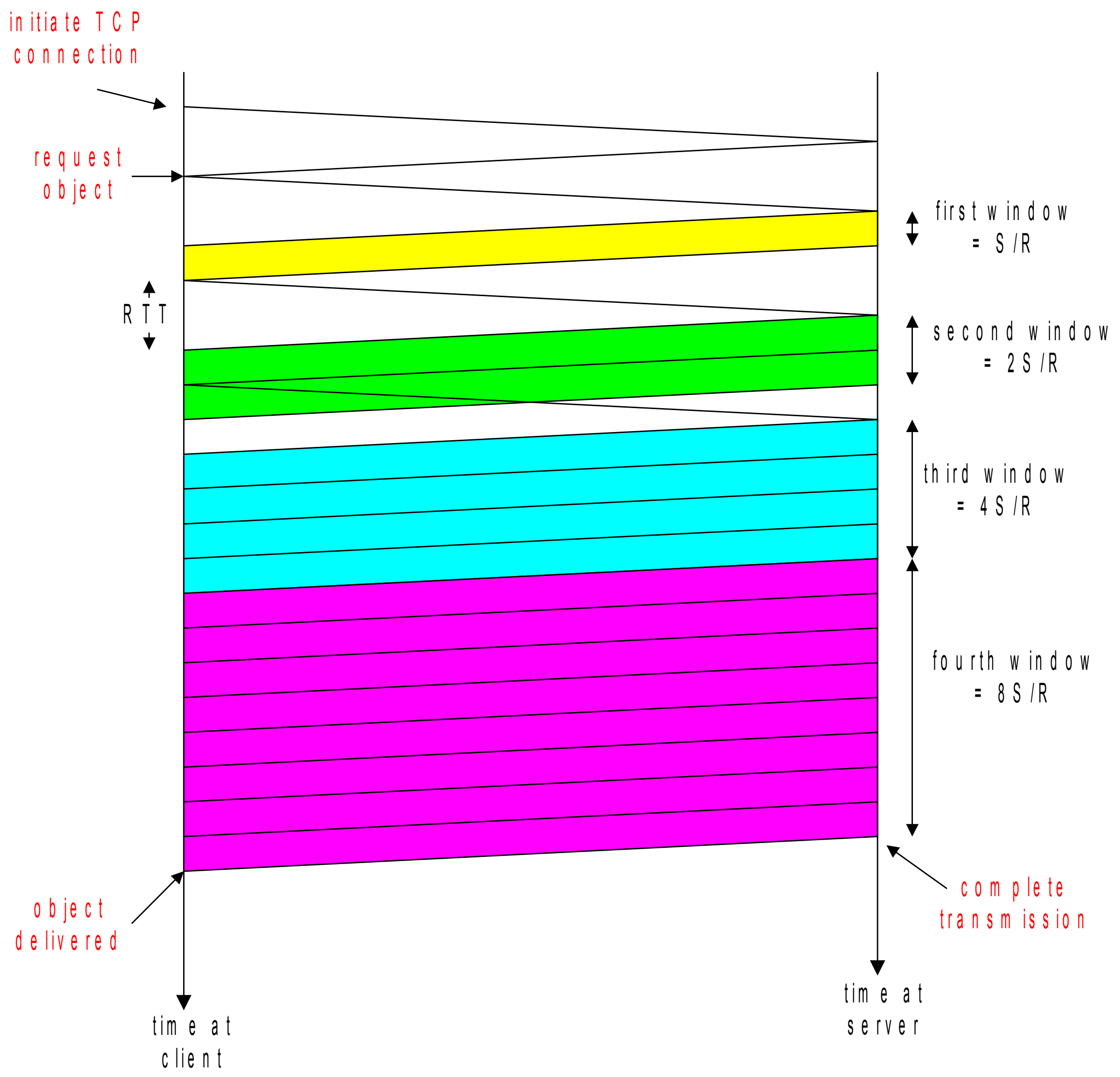
- 2 RTT for connection estab and request
- O/R to transmit object
- time server idles due to slow start

Server idles:
P = min{K-1,Q} times

Example:

- O/S = 15 segments
- K = 4 windows
- Q = 2
- P = min{K-1,Q} = 2

Server idles P=2 times



TCP Delay Modeling (3)

$$\frac{S}{R} + RTT = \text{time from when server starts to send segment until server receives acknowledgement}$$

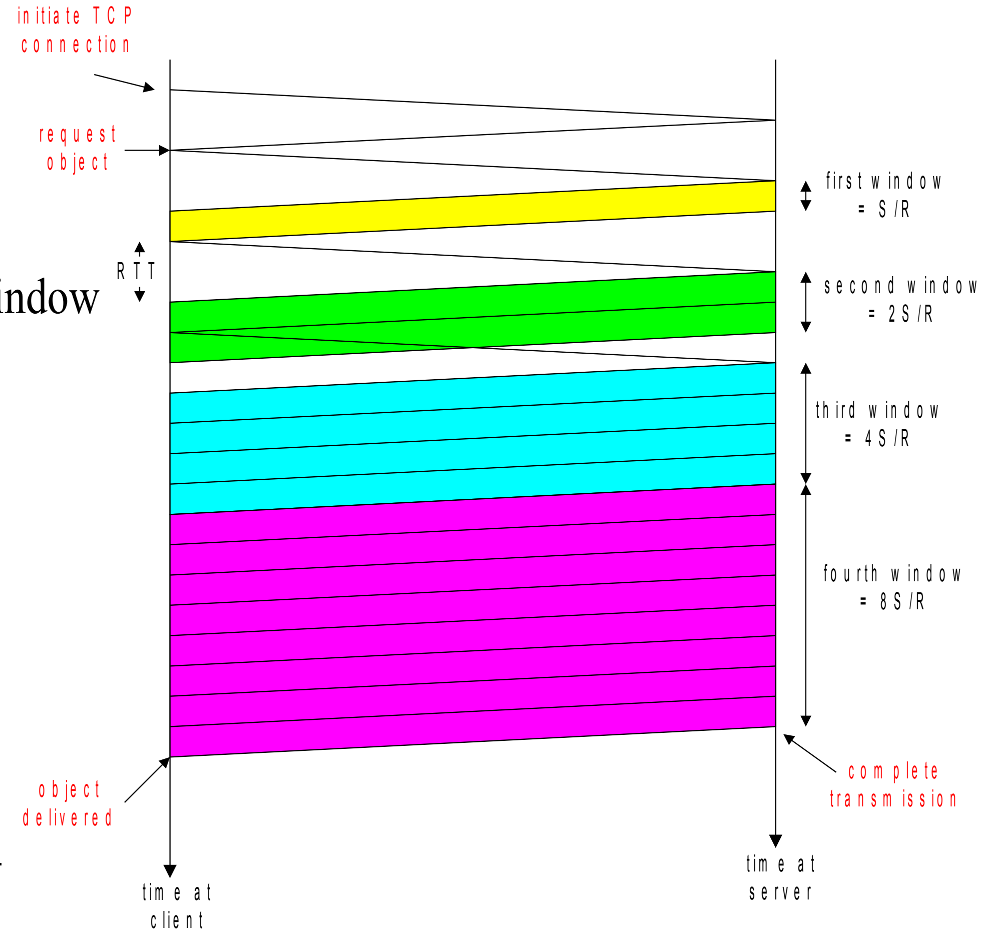
$$2^{k-1}\frac{S}{R} = \text{time to transmit the kth window}$$

$$\left[\frac{S}{R} + RTT - 2^{k-1}\frac{S}{R}\right]^{+} = \text{idle time after the } k \text{ th window}$$

$$\text{delay} = \frac{O}{R} + 2\,RTT + \sum_{p=1}^{P} idleTime_p$$

$$¿\,\frac{O}{R} + 2\,RTT + \sum_{k=1}^{P}\left[\frac{S}{R} + RTT - 2^{k-1}\frac{S}{R}\right]$$

$$¿\,\frac{O}{R} + 2\,RTT + P\left[RTT + \frac{S}{R}\right] - (2^{P} - 1)\frac{S}{R}$$



TCP Delay Modeling (4)

Recall K = number of windows that cover object

How do we calculate K ?

$$K = \min\{k : 2^0 S + 2^1 S + \cdots + 2^{k-1} S \geq O\}$$

$$\text{¿} \min\{k : 2^0 + 2^1 + \cdots + 2^{k-1} \geq O/S\}$$

$$\text{¿} \min\{k : 2^k - 1 \geq \frac{O}{S}\}$$

$$\text{¿} \min\{k : k \geq \log_2(\frac{O}{S} + 1)\}$$

$$\text{¿} \left[\log_2(\frac{O}{S} + 1)\right]$$

Calculation of Q, number of idles for infinite-size object, is similar (see text).

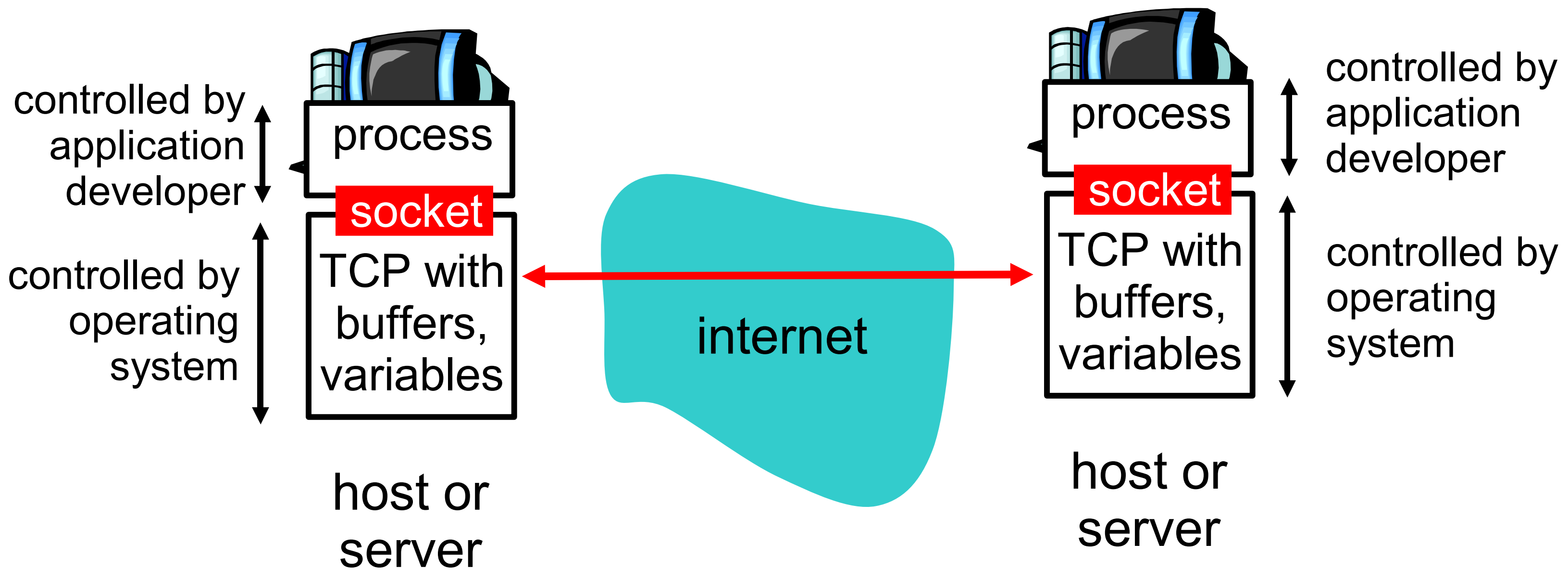
Chapter 2: Application layer

- 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
- 2.9 Building a Web server

Socket-programming using TCP

Socket: a door between application process and end-end-transport protocol (UCP or TCP)

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


controlled by application developer
controlled by operating system
process
socket
TCP with buffers, variables
host or server
internet
process
socket
TCP with buffers, variables
host or server
controlled by application developer
controlled by operating system

Socket programming *with TCP*

Client must contact server

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

Client contacts server by:

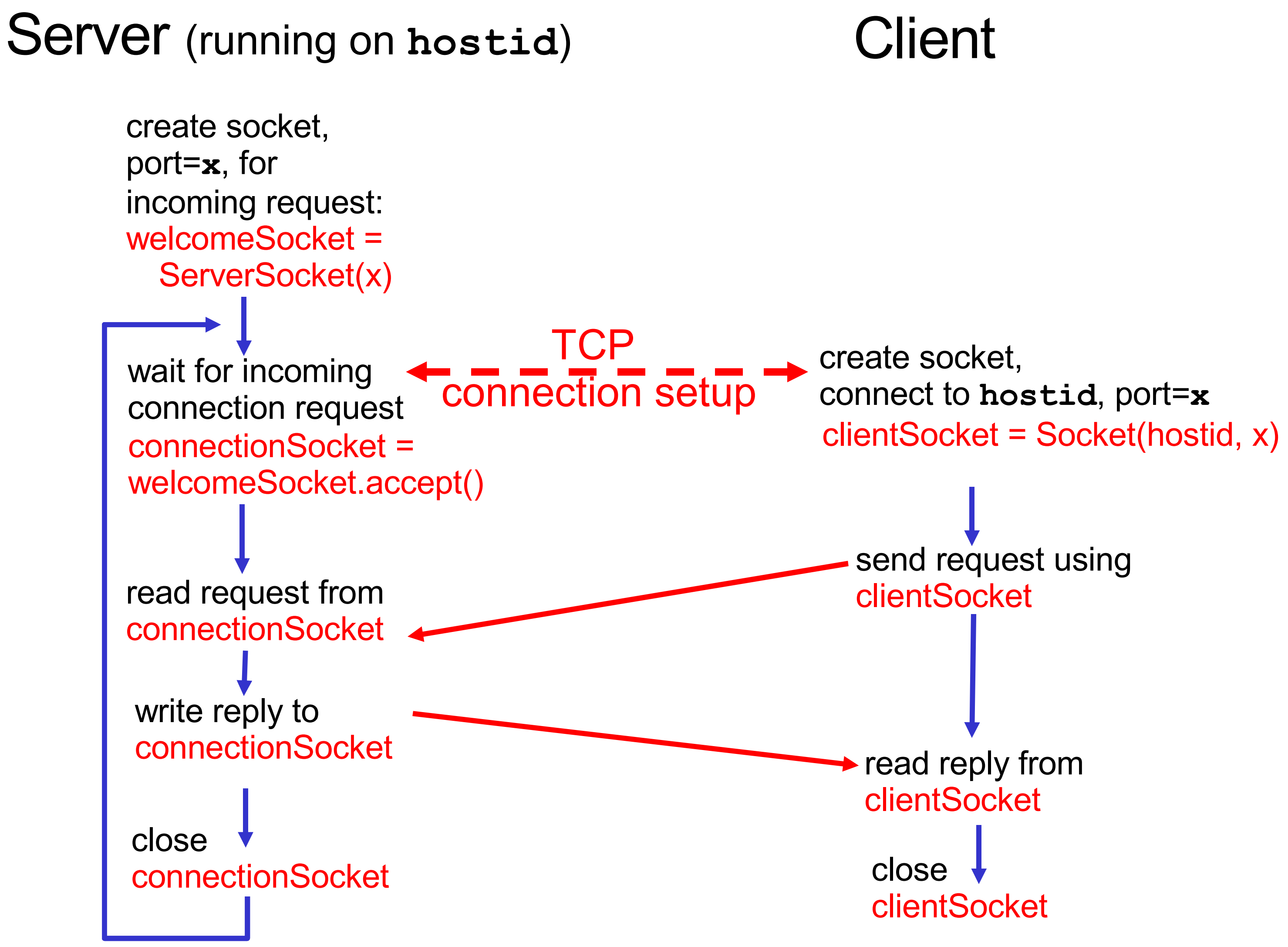
- creating client-local TCP socket
- specifying IP address, port number of server process
- When client creates socket: client TCP establishes connection to server TCP

- When contacted by client, server TCP creates new socket for server process to communicate with client
    - allows server to talk with multiple clients
    - source port numbers used to distinguish clients

application viewpoint

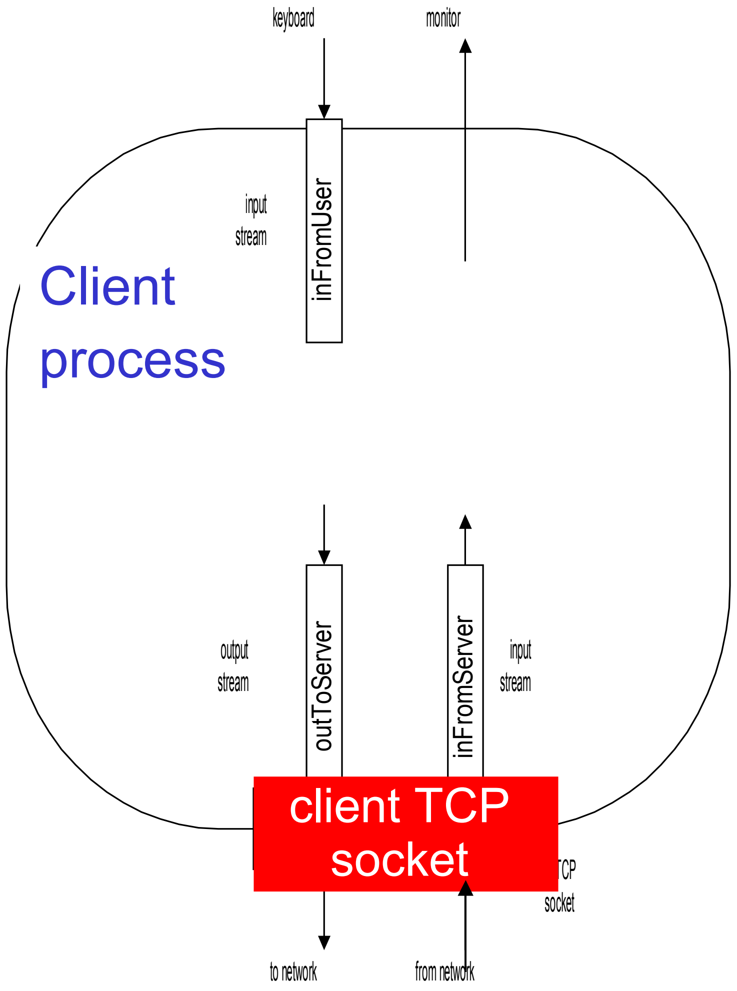
TCP provides reliable, in-order transfer of bytes ("pipe") between client and server

Client/server socket interaction: TCP


Server (running on **hostid**)
Client
create socket, port=**x**, for incoming request:
welcomeSocket = ServerSocket(x)
wait for incoming connection request
connectionSocket = welcomeSocket.accept()
TCP connection setup
create socket, connect to **hostid**, port=**x**
clientSocket = Socket(hostid, x)
send request using clientSocket
read request from connectionSocket
write reply to connectionSocket
read reply from clientSocket
close connectionSocket
close clientSocket

Stream jargon

- A stream is a sequence of characters that flow 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.



Socket programming with TCP

Example client-server app:

1) client reads line from standard input (**`inFromUser`** stream) , sends to server via socket (**`outToServer`** stream)

2) server reads line from socket

3) server converts line to uppercase, sends back to client

4) client reads, prints modified line from socket (**`inFromServer`** stream)

Example: Java client (TCP)

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

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

        BufferedReader inFromUser =
          new BufferedReader(new InputStreamReader(System.in));

        Socket clientSocket = new Socket("hostname", 6789);

        DataOutputStream outToServer =
          new DataOutputStream(clientSocket.getOutputStream());
```

Create input stream

Create client socket, connect to server

Create output stream attached to socket

Example: Java client (TCP), cont.

Create input stream attached to socket

```
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();

    }
}
```

Example: Java server (TCP)

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

class TCPServer {

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

     ServerSocket welcomeSocket = new ServerSocket(6789);

     while(true) {

         Socket connectionSocket = welcomeSocket.accept();

        BufferedReader inFromClient =
          new BufferedReader(new
          InputStreamReader(connectionSocket.getInputStream()));
```

Create welcoming socket at port 6789

Wait, on welcoming socket for contact by client

Create input stream, attached to socket

Example: Java server (TCP), cont

Create output stream, attached to socket

Read in line from socket

Write out line to socket

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

```
            DataOutputStream  outToClient =
              new DataOutputStream(connectionSocket.getOutputStream());

            clientSentence = inFromClient.readLine();

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

            outToClient.writeBytes(capitalizedSentence);

            connectionSocket.close();
        }
    }
}
```

Recap

- TCP
  - Throughput
  - Fairness
  - Delay modeling
- TCP socket programming

Next time

- NAT
- Application layer
    - Intro
    - Web / HTTP