Network Applications: TCP Socket Programming; File Transfer Protocol; HTTP/1.0

Qiao Xiang

https://qiaoxiang.me/courses/cnns-xmuf22/index.shtml

10/04/2022

This deck of slides are heavily based on CPSC 433/533 at Yale University, by courtesy of Dr. Y. Richard Yang.

Outline

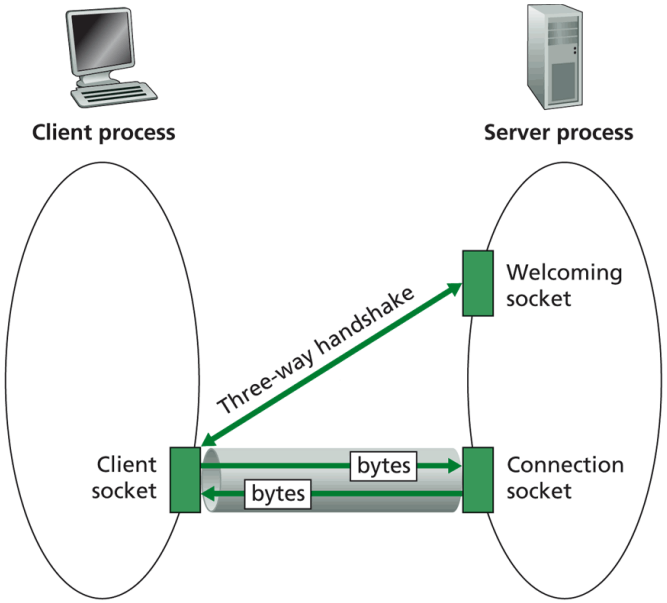
- Admin. and recap
- Network application programming
  - UDP sockets
  - TCP sockets
- Network applications (continue)
  - File transfer (FTP) and extension
  - HTTP
    - HTTP/1.0

Admin.

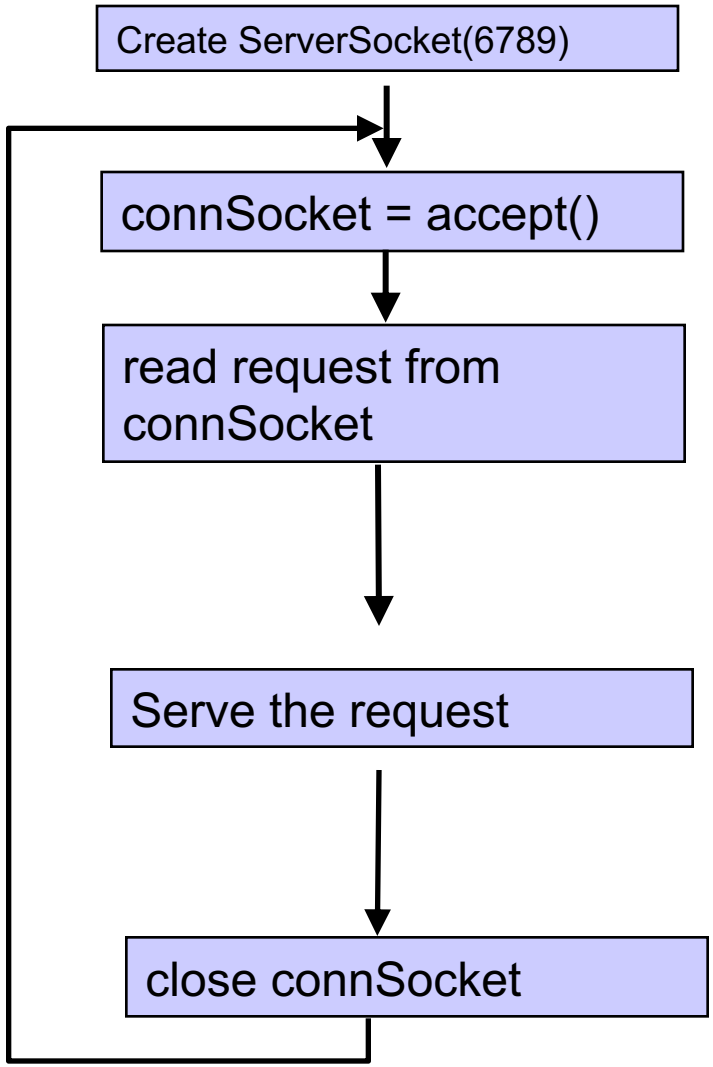
- Lab assignment 2 due Oct. 13

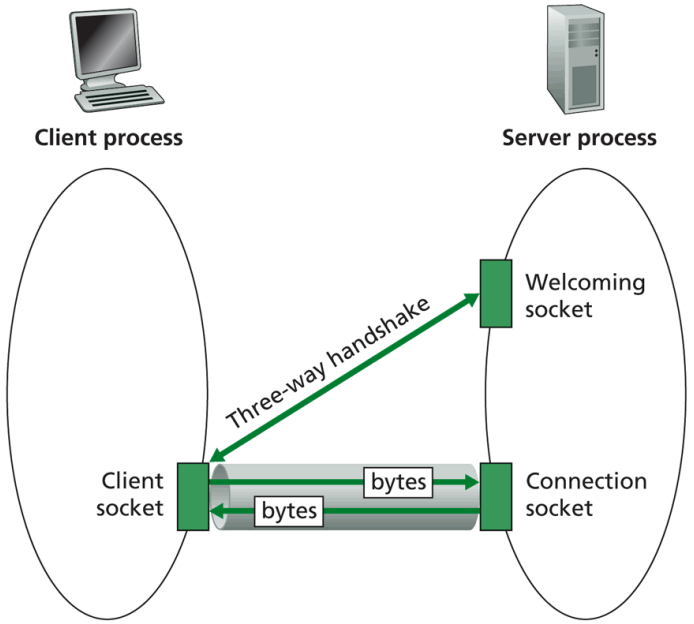
Recap: TCP Sockets

- TCP server socket demux by 4-tuple:
  - source IP address
  - source port number
  - dest IP address
  - dest port number



Server Flow





-Welcome socket: the waiting room
-connSocket: the operation room

ServerSocket

- **ServerSocket**()
  - creates an unbound server socket.
- **ServerSocket**(int port)
  - creates a server socket, bound to the specified port.
- **ServerSocket**(int port, int backlog)
  - creates a server socket and binds it to the specified local port number, with the specified backlog.
- **ServerSocket**(int port, int backlog, InetAddress bindAddr)
  - creates a server with the specified port, listen backlog, and local IP address to bind to.

- **bind**(SocketAddress endpoint)
  - binds the ServerSocket to a specific address (IP address and port number).
- **bind**(SocketAddress endpoint, int backlog)
  - binds the ServerSocket to a specific address (IP address and port number).

- Socket **accept**()
  - listens for a connection to be made to this socket and accepts it.

- **close**()

  closes this socket.

(Client)Socket

- **Socket**(InetAddress address, int port)
  - creates a stream socket and connects it to the specified port number at the specified IP address.
- **Socket**(InetAddress address, int port, InetAddress localAddr, int localPort)
  - creates a socket and connects it to the specified remote address on the specified remote port.
- **Socket**(String host, int port)
  - creates a stream socket and connects it to the specified port number on the named host.

- **bind**(SocketAddress bindpoint)
  - binds the socket to a local address.

- **connect**(SocketAddress endpoint)
  - connects this socket to the server.
- **connect**(SocketAddress endpoint, int timeout)
  - connects this socket to the server with a specified timeout value.

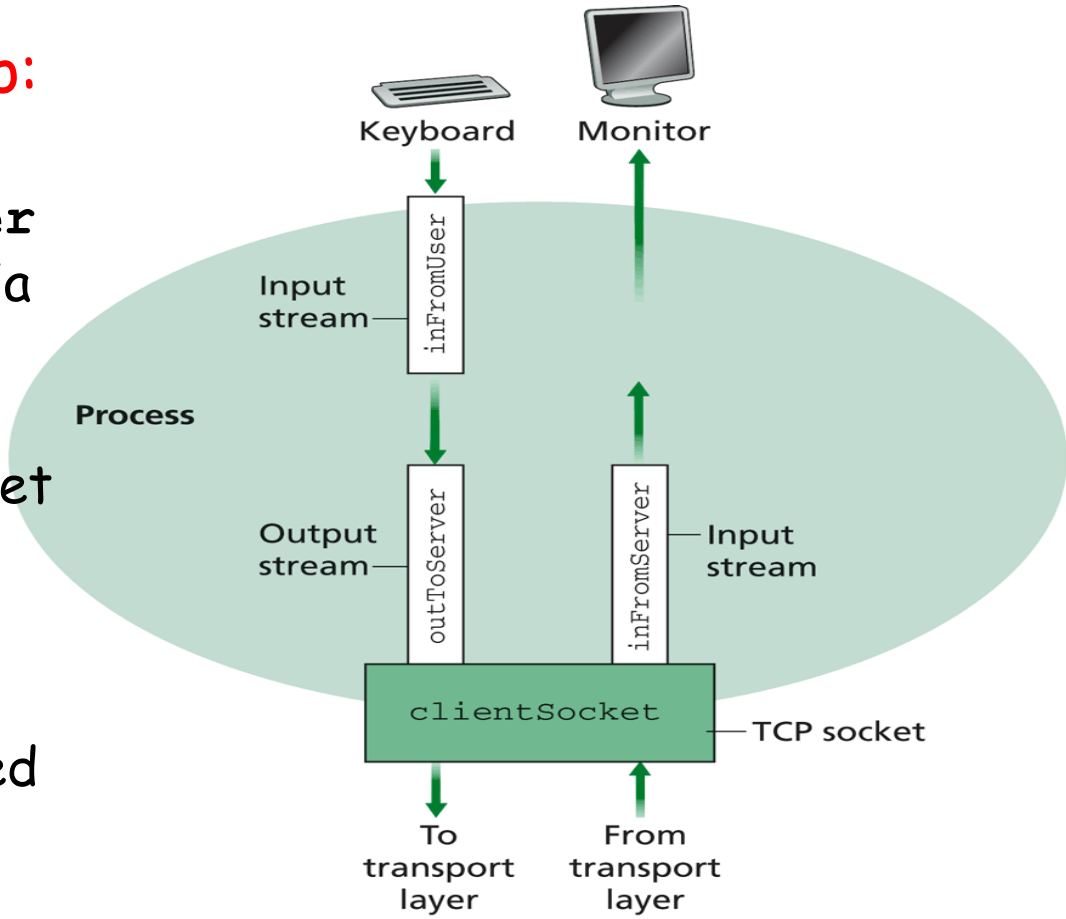
- InputStream **getInputStream**()
  - returns an input stream for this socket.
- OutputStream **getOutputStream**()
  - returns an output stream for this socket.

- **close**()
  closes this socket.

Simple TCP Example

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));
        sentence = inFromUser.readLine();

        Socket clientSocket = new Socket("server.name", 6789);

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

Create input stream

Create client socket, connect to server

Create output stream attached to socket

OutputStream

- public abstract class OutputStream
  - public abstract void write(int b) throws IOException
  - public void write(byte[] data) throws IOException
  - public void write(byte[] data, int offset, int length) throws IOException
  - public void flush( ) throws IOException
  - public void close( ) throws IOException

InputStream

- public abstract class InputStream
  - public abstract int read( ) throws IOException
  - public int read(byte[] input) throws IOException
  - public int read(byte[] input, int offset, int length) throws IOException
  - public long skip(long n) throws IOException
  - public int available( ) throws IOException
  - public void close( ) throws IOException

Example: Java client (TCP), cont.

Send line to server

```
outToServer.writeBytes(sentence + '\n');
```

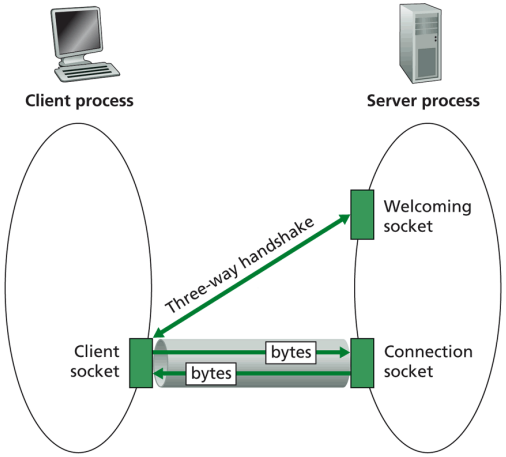
Create input stream attached to socket

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

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);
```

Create welcoming socket at port 6789

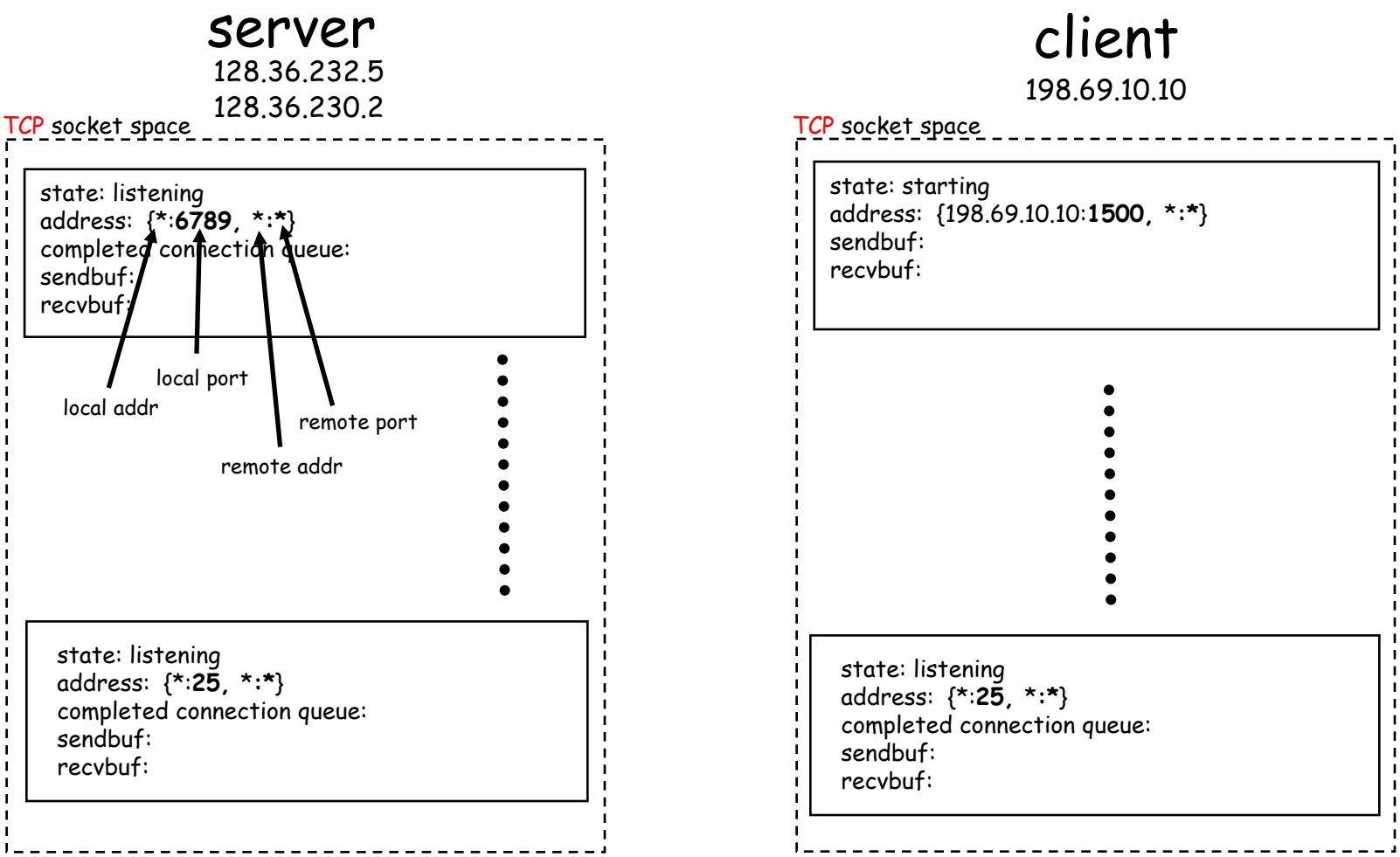
Demo

% on MAC

start TCPServer

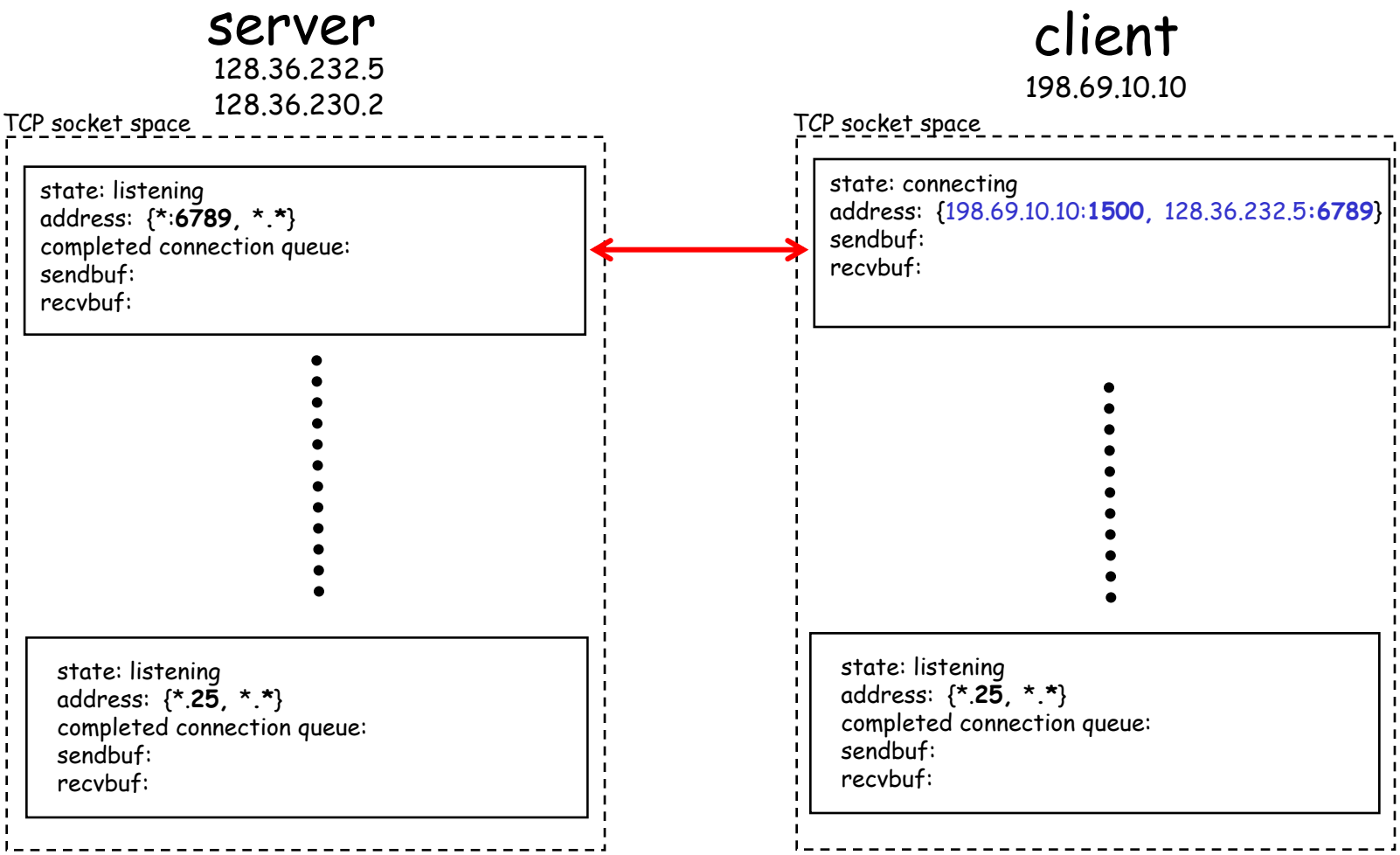
wireshark to capture our TCP traffic
tcp.srcport==6789 or tcp.dstport==6789

Under the Hood: After Welcome (Server) Socket



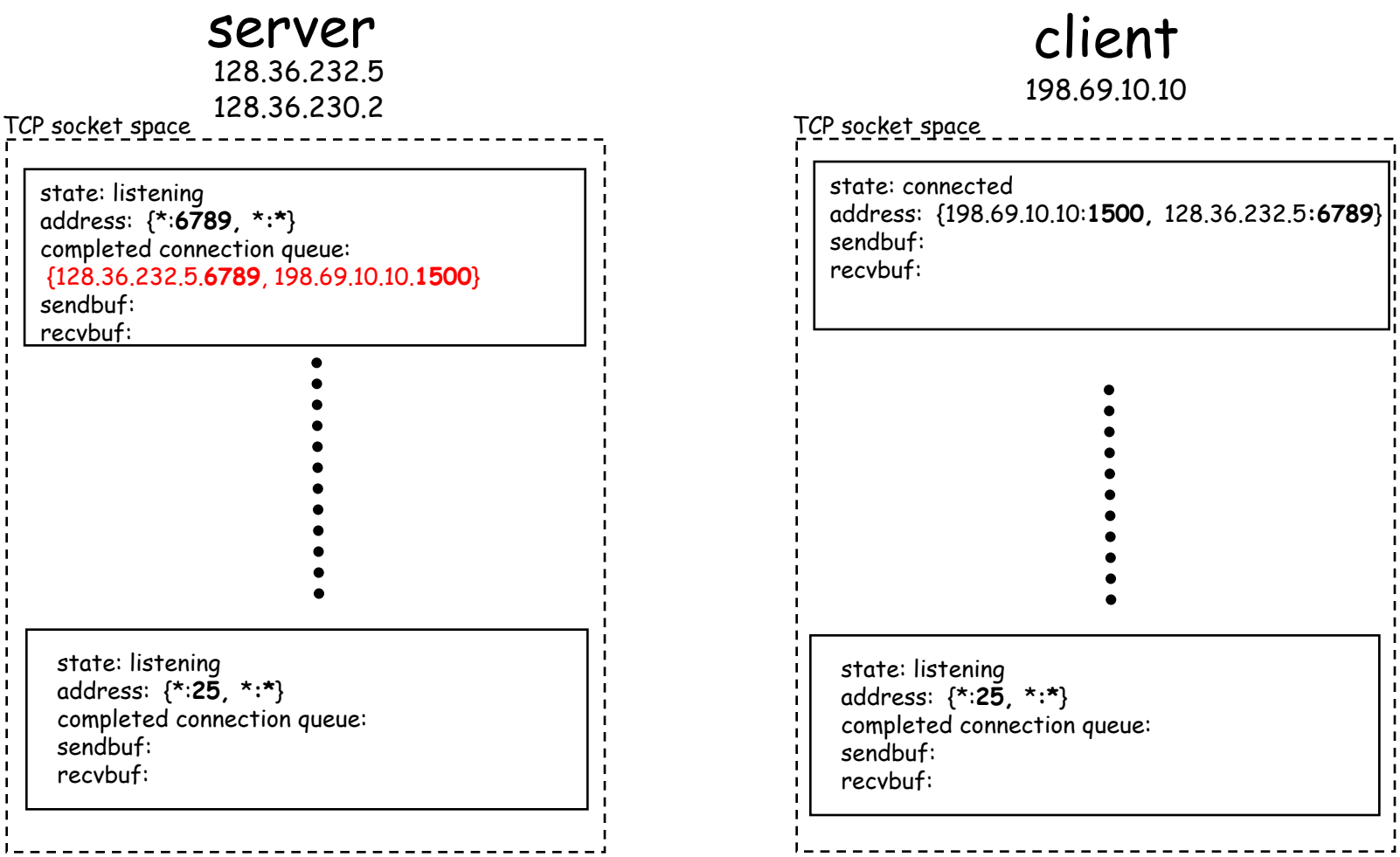
%netstat –p tcp –n -a

After Client Initiates Connection

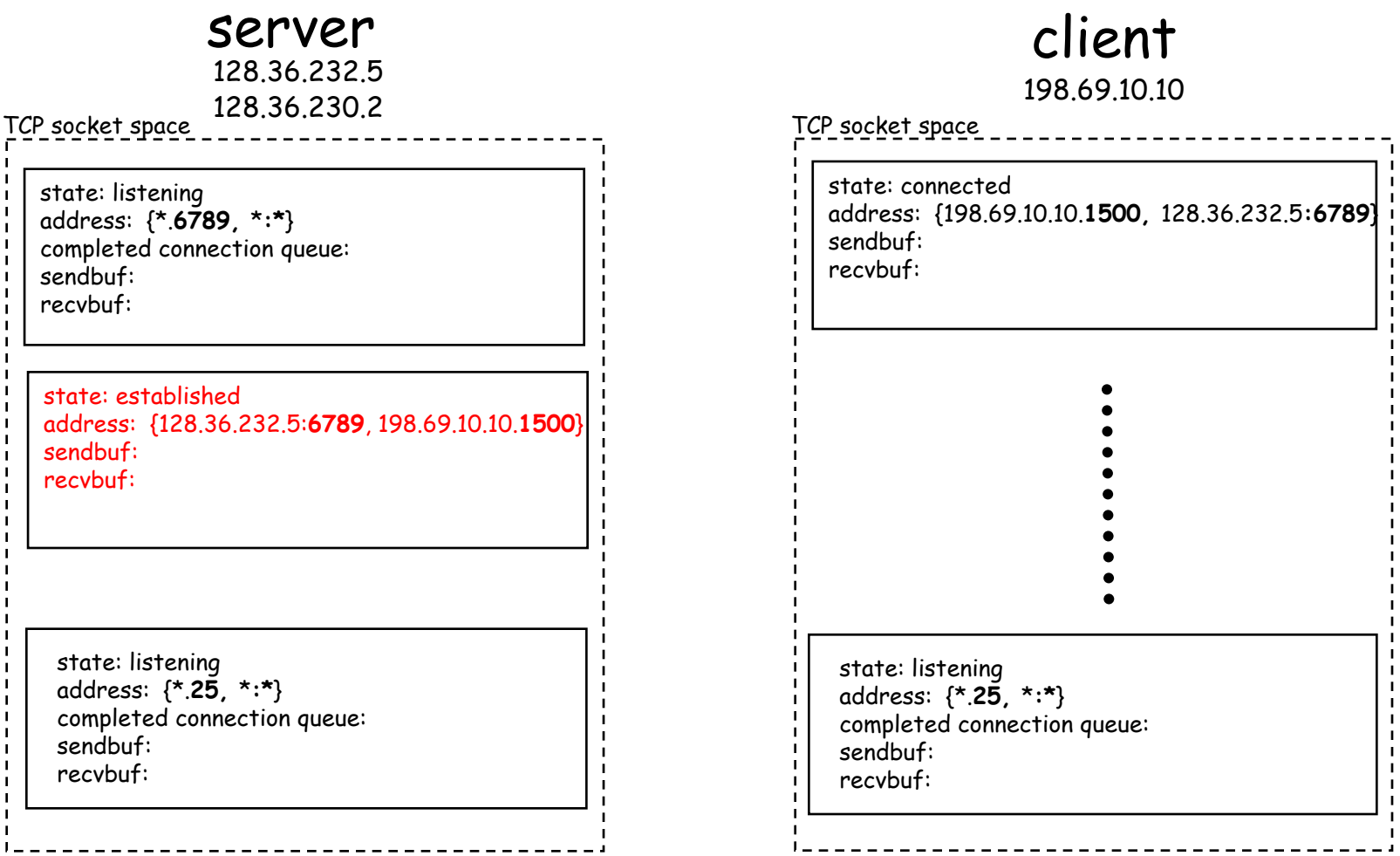


%ubuntu java TCPClient <server> 6789

Example: Client Connection Handshake Done



Example: Client Connection Handshake Done



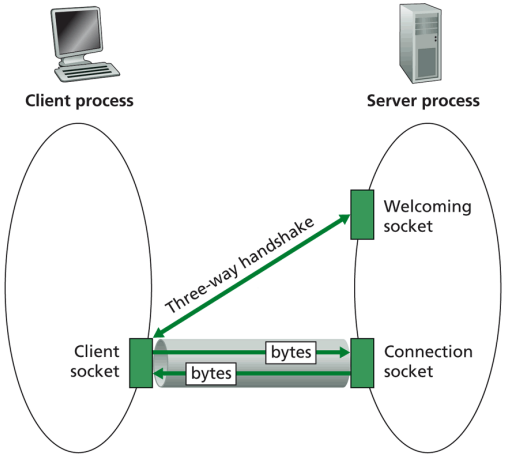
Packet demutiplexing is based on (dst addr, dst port, src addr, src port)

Packet sent to the socket with the best match!

Demo

- What if more client connections than backlog allowed?
  - We continue to start java TCPClient

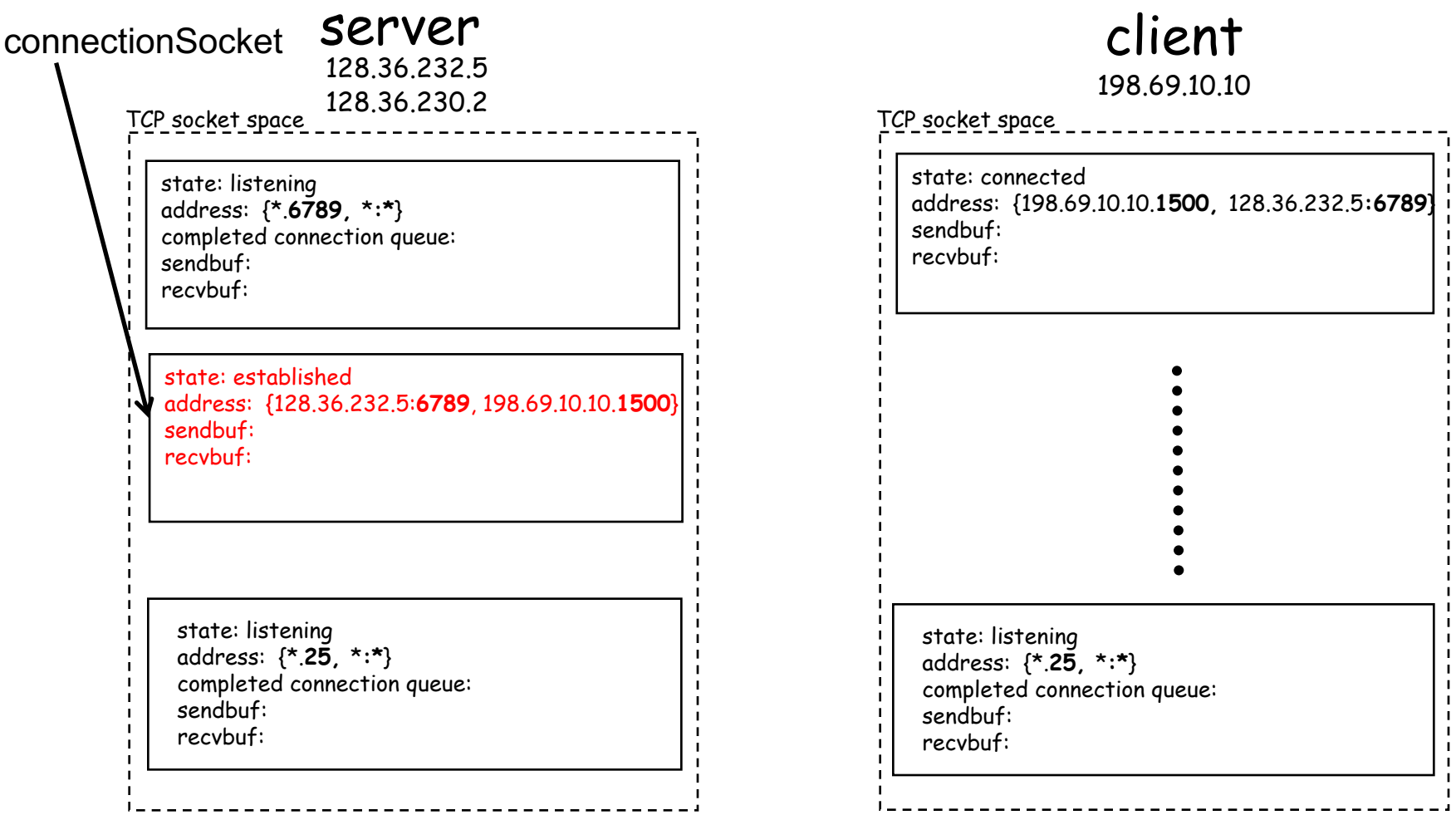
Example: Java server (TCP)




```
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();
```

Wait, on welcoming socket for contact by client

Example: Server accept()



Example: Java server (TCP): Processing

Create input stream, attached to socket

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

Read in line from socket

```
clientSentence = inFromClient.readLine();

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

```
            }
        }
    }
```

Example: Java server (TCP): Output

Create output stream, attached to socket

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

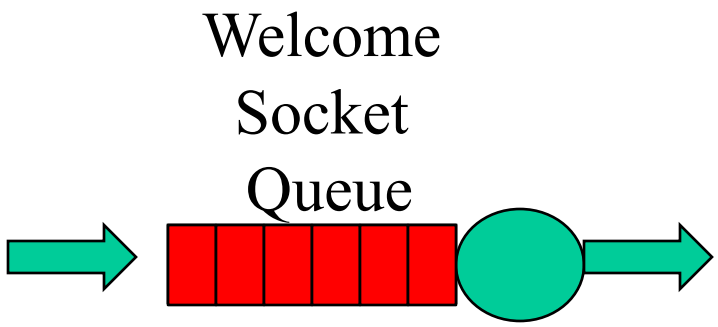
Write out line to socket

```
outToClient.writeBytes(capitalizedSentence);
            }
        }
    }
```

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

Analysis

- Assume that client requests arrive at a rate of lambda/second
- Assume that each request takes $1/\mu$ seconds
- A basic question
  - How big is the backlog (welcome queue)



Analysis

- Is there any interop issue in the sample program?

Analysis

- Is there any interop issue in the sample program?
  - DataOutputStream writeBytes(String) truncates
    - http://docs.oracle.com/javase/1.4.2/docs/api/java/io/DataOutputStream.html#writeBytes(java.lang.String)

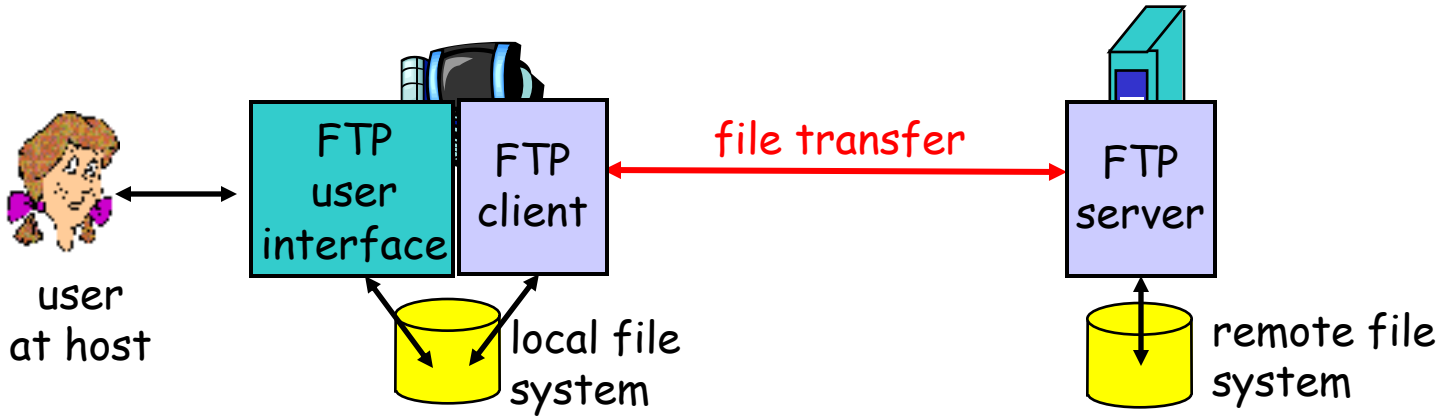
Summary: Basic Socket Programming

- They are relatively straightforward
  - UDP: DatagramSocket
  - TCP: ServerSocket, Socket
- The main function of socket is multiplexing/demultiplexing to application processes
  - UDP uses (dst IP, port)
  - TCP uses (src IP, src port, dst IP, dst port)
- Always pay attention to encoding/decoding

Outline

- Admin. and recap
- Network application programming
  - UDP sockets
  - TCP sockets
- Network applications (continue)
  - *File transfer (FTP) and extension*

FTP: the File Transfer Protocol



- Transfer files to/from remote host
- Client/server model
  - *client:* side that initiates transfer (either to/from remote)
  - *server:* remote host
- ftp: RFC 959
- ftp server: port 21/20 (smtp 25, http 80)

FTP Commands, Responses

Sample commands:

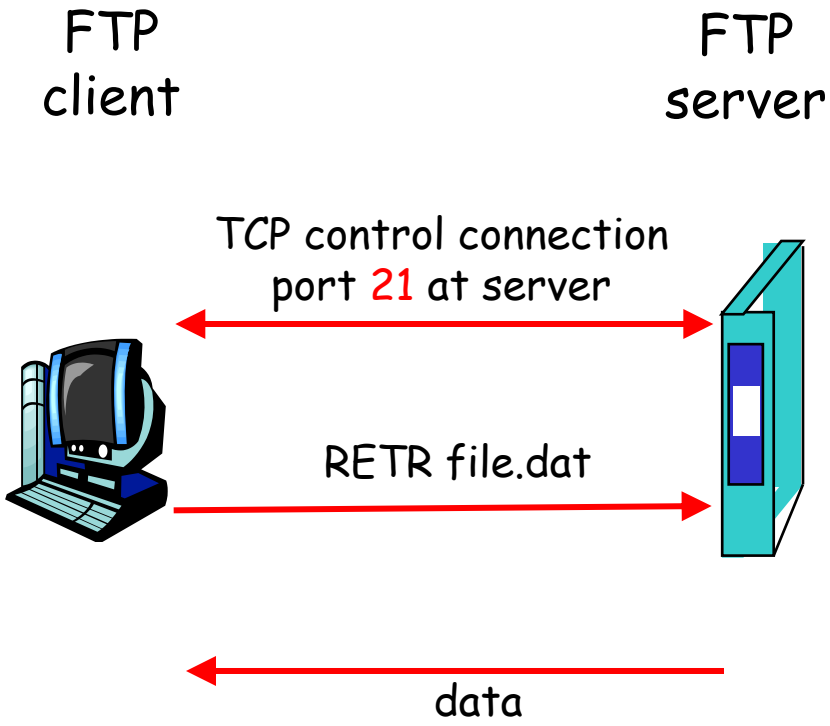
- sent as ASCII text over control channel
- **USER *username***
- **PASS *password***
- **PWD** returns current dir
- **STAT** shows server status
- **LIST** returns list of file in current directory
- **RETR filename** retrieves (gets) file
- **STOR filename** stores file

Sample return codes

- status code and phrase
- **331 Username OK, password required**
- **125 data connection already open; transfer starting**
- **425 Can't open data connection**
- **452 Error writing file**

FTP Protocol Design

- What is the simplest design of data transfer?



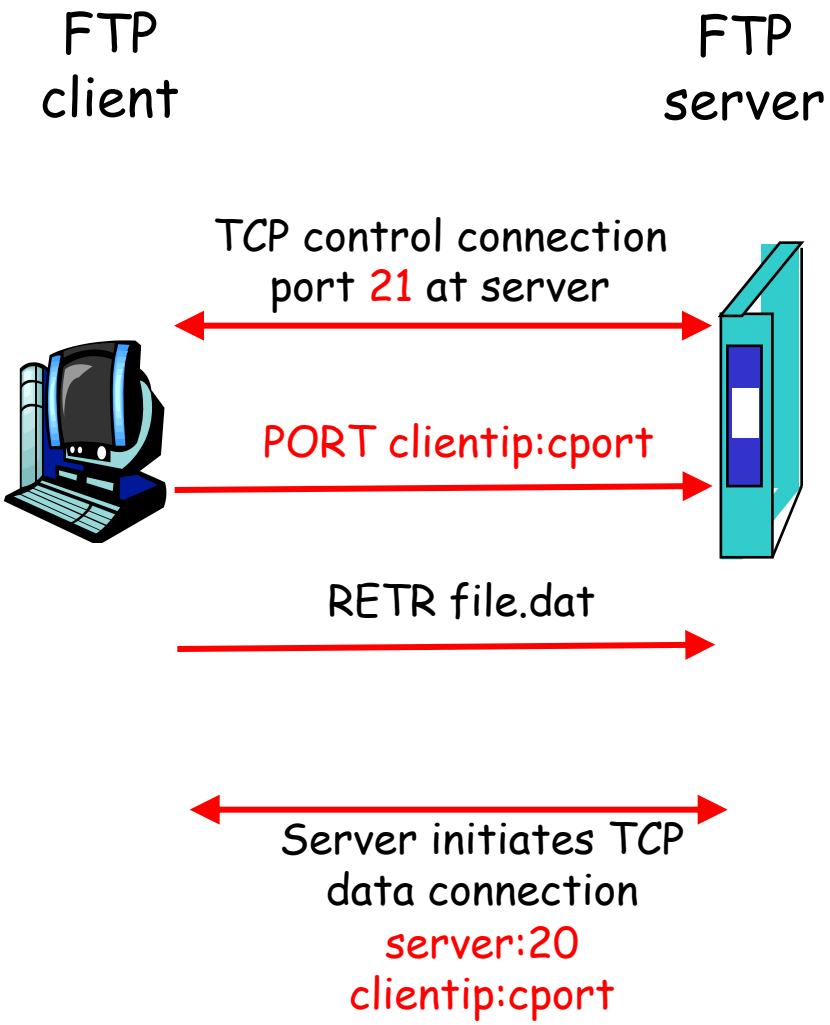
FTP: A Client-Server Application with Separate Control, Data Connections

- Two types of TCP connections opened:
  - A control connection: exchange commands, responses between client, server.
    "out of band control"
  - Data connections: each for file data to/from server

Discussion: why does FTP separate control/data connections?

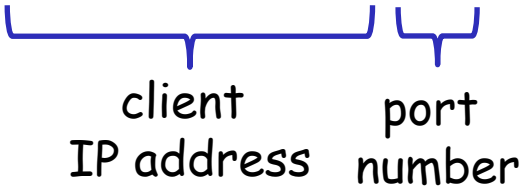
Q: How to create a new data connection?

Traditional FTP: Client Specifies Port for Data Connection



Example using telnet/nc

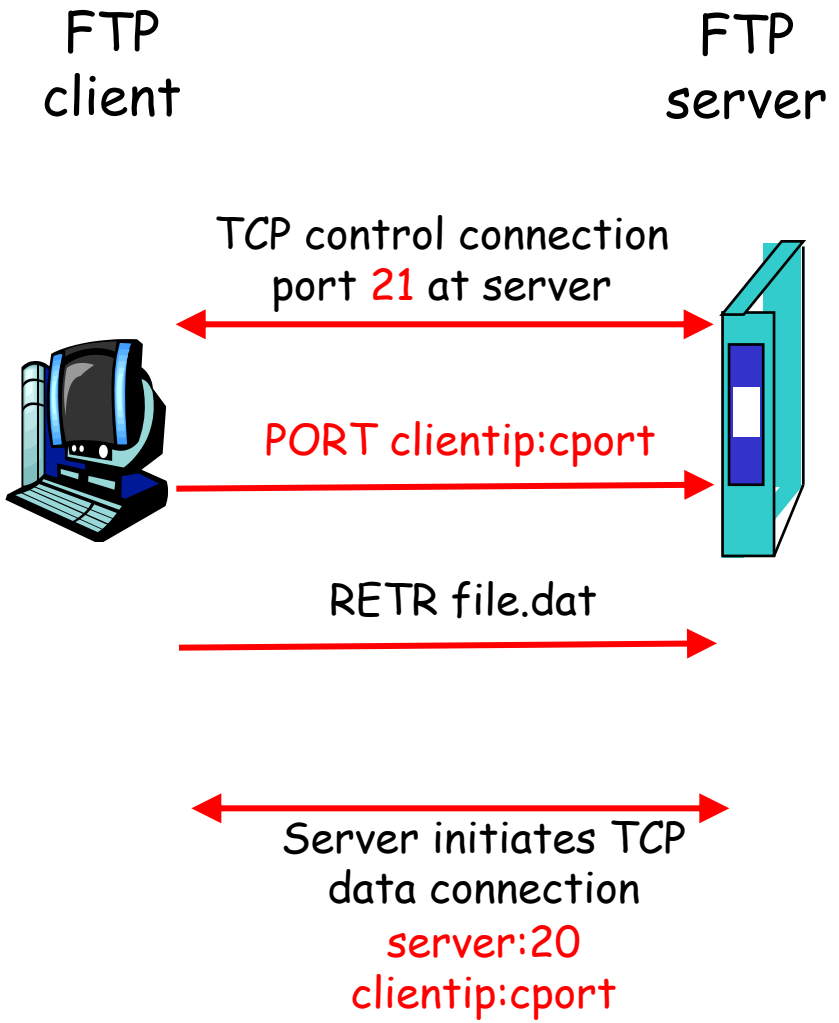
- Use telnet for the control channel
  - telnet ftp.ietf.org 21
  - user anonymous
  - pass your_email
  - port 10,90,61,172,4,1
  - list



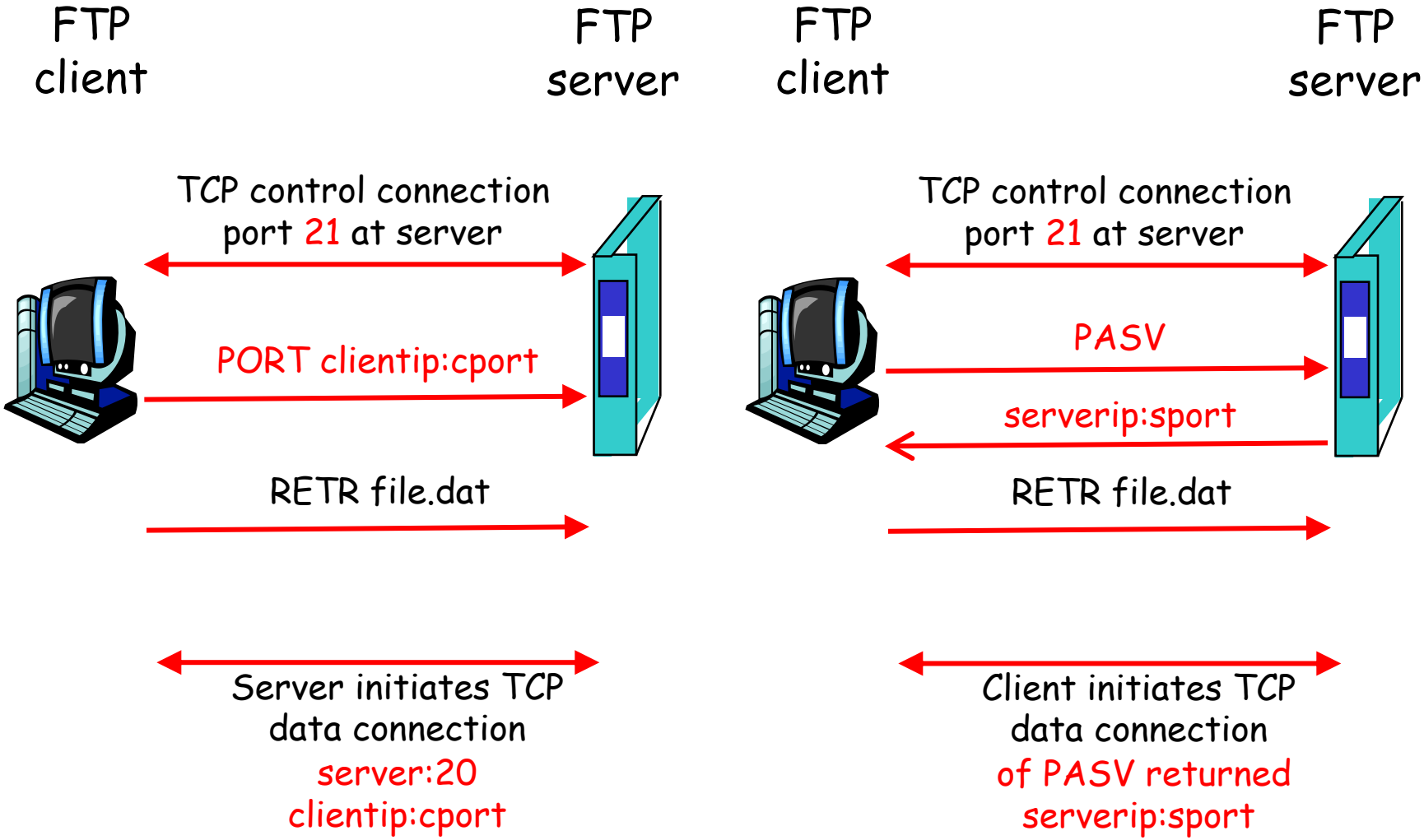
- use nc (NetCat) to receive/send data with server
  - nc –v –l 1025

Problem of the Client PORT Approach

- Many Internet hosts are behind NAT/firewalls that block connections initiated from outside



FTP PASV: Server Specifies Data Port, Client Initiates Connection

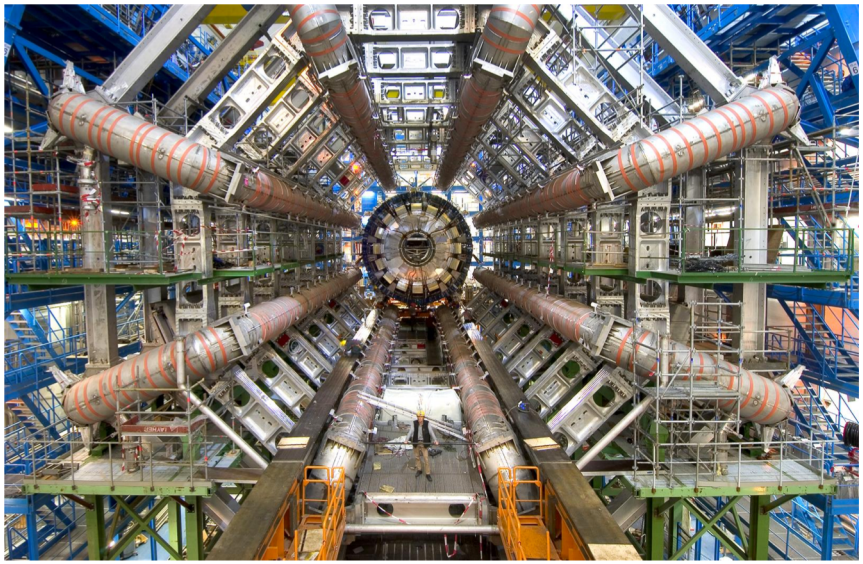


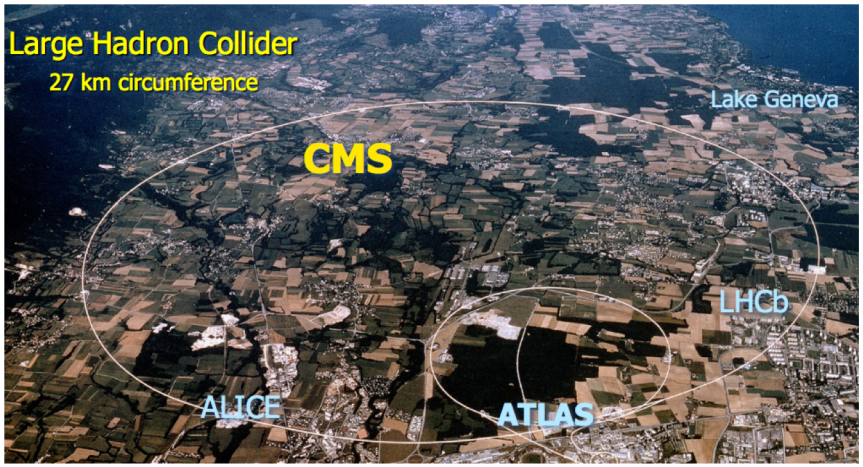
Example

- Use Wireshark to capture FTP traffic
  - Using chrome to visit
    ftp://ftp.freebsd.org

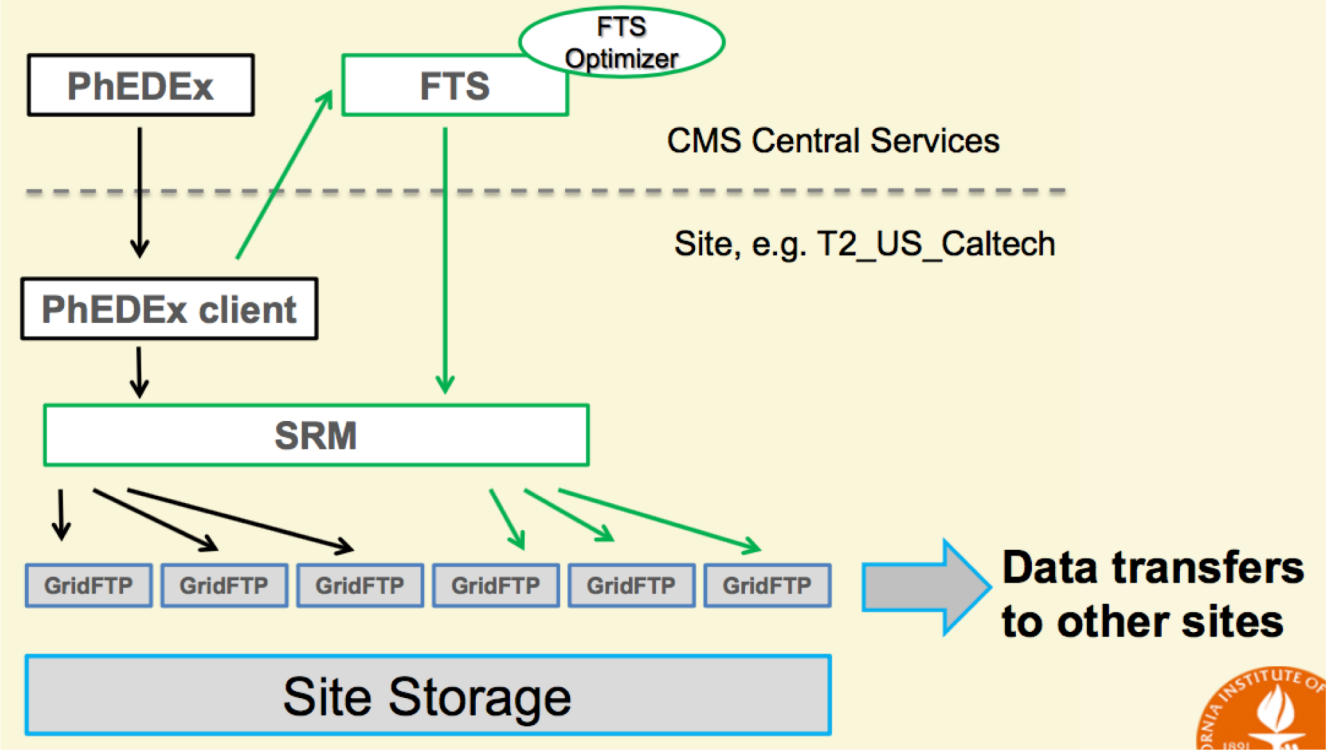
FTP Extensions

- FTP with extensions are being used extensively in large data set transfers (e.g., LHC)





Data Transfer Structure

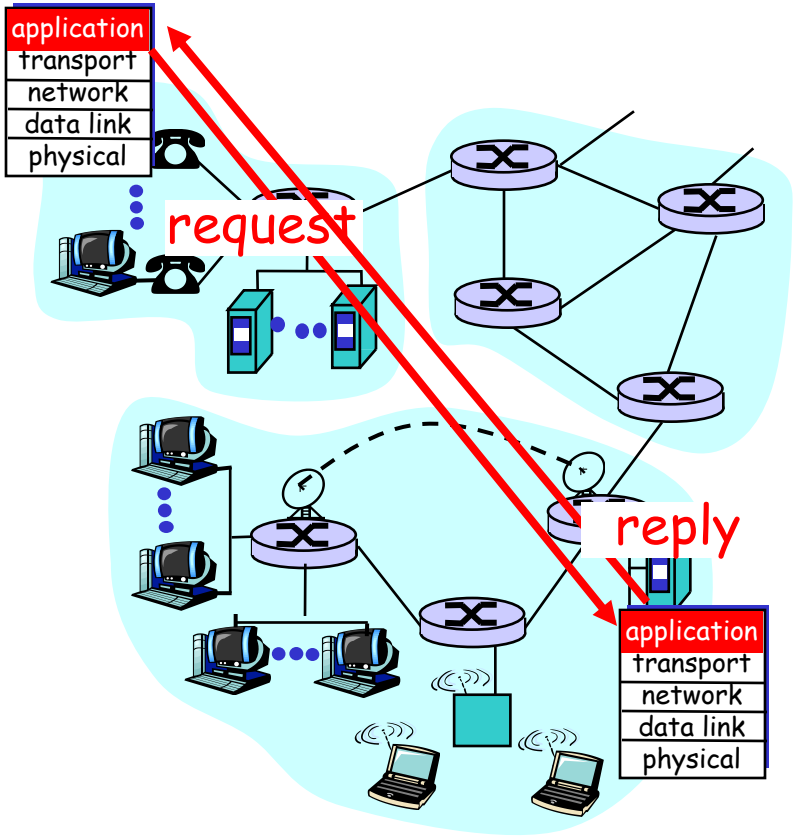


- See GridFTP to FTP extensions
  - https://www.ogf.org/documents/GFD.20.pd
- Goal of GridFTP: allow parallel, high-throughput data transfer
  - Discussion: What features do we need to add to FTP to allow parallel transfers?

FTP Evaluation

Key questions to ask about a C-S application

- **Is the application extensible?**
- **Is the application scalable?**
- **How does the application handle server failures (being robust)?**
- **How does the application provide security?**



What are some interesting design features of the FTP protocol?

Outline

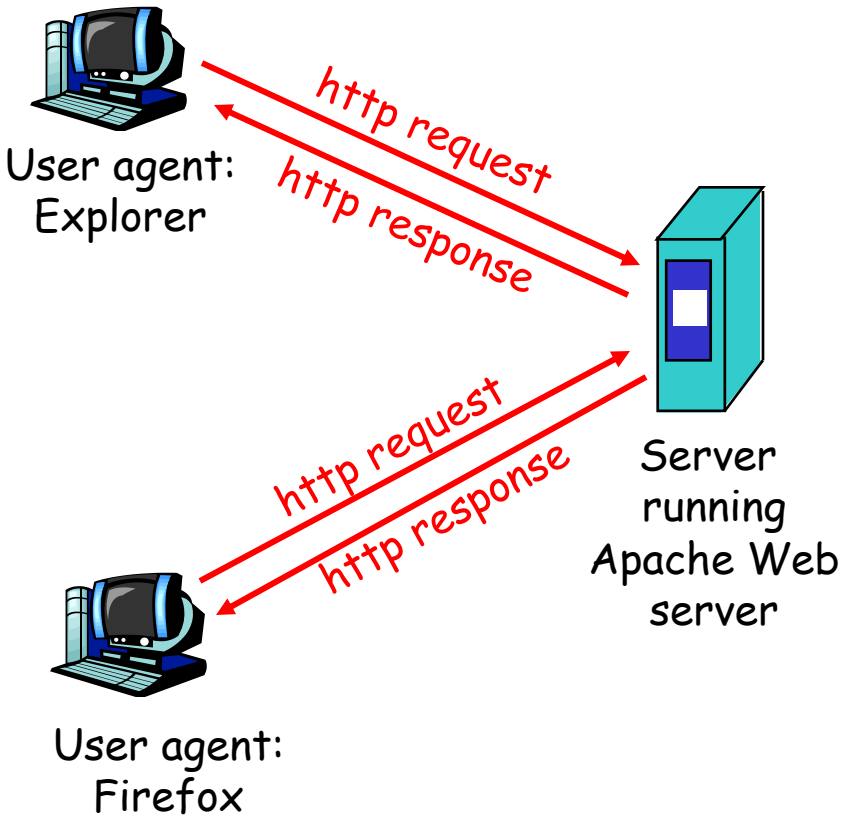
- Admin. and recap
- Network application programming
  - UDP sockets
  - TCP sockets
- Network applications (continue)
  - File transfer (FTP) and extension
  - *HTTP*

From Opaque Files to Web Pages

- Web page:
  - authored in HTML
  - addressed by a URL
    - URL has two components:
      - host name, port number and
      - path name

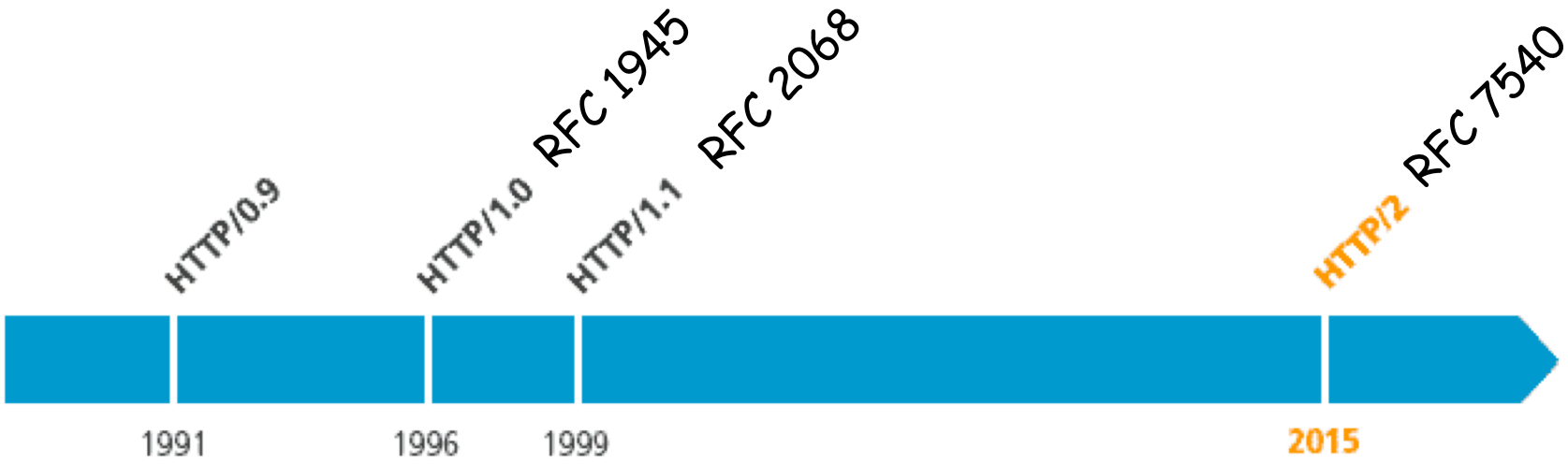
`http://qiaoxiang.me:80/index.html`

- Most Web pages consist of:
  - base HTML page, and
  - several referenced objects
    - E.g., image



The Web pages are requested through HTTP: hypertext transfer protocol

HTTP is Still Evolving



HTTP 1.0 Message Flow

- Server waits for requests from clients

- Client initiates TCP connection (creates socket) to server, port 80
- Client sends request for a document
- Web server sends back the document
- TCP connection closed

- Client parses the document to find embedded objects (images)
  - repeat above for each image

HTTP 1.0 Message Flow (more detail)

Suppose user enters URL
qiaoxiang.me/index.html

0. http server at host qiaoxiang.me waiting for TCP connection at port 80.

1a. http client initiates TCP connection to http server (process) at qiaoxiang.me. Port 80 is default for http server.

1b. server "accepts" connection, ack. client

2. http client sends http *request message* (containing URL) into TCP connection socket

3. http server receives request message, forms *response message* containing requested object (index.html), sends message into socket (the sending speed increases slowly, which is called slow-start)

time

HTTP 1.0 Message Flow (cont.)

4. http server closes TCP connection.

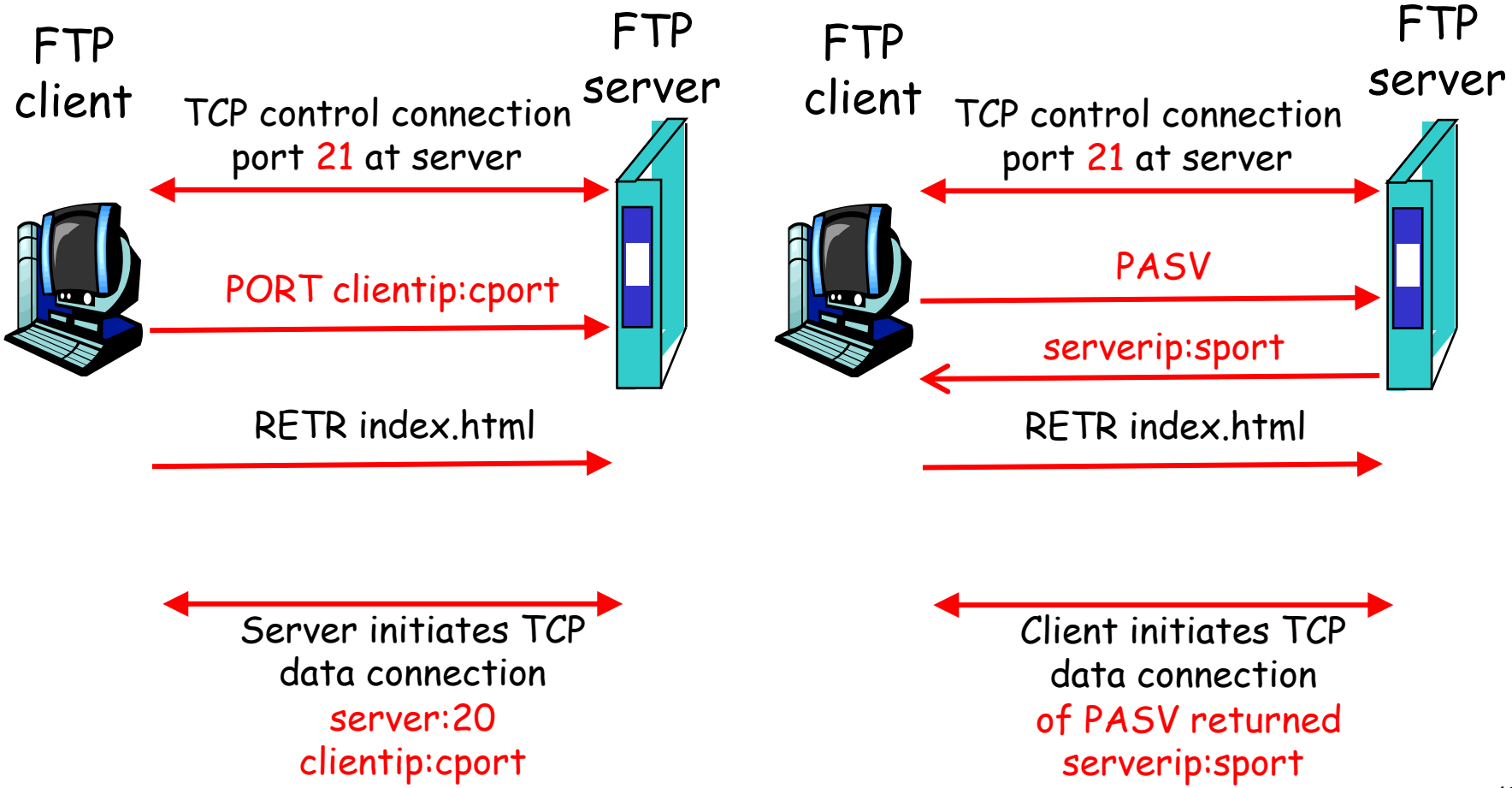
5. http client receives response message containing html file, parses html file, finds embedded image

time

6. Steps 1-5 repeated for each of the embedded images

Discussion

- How about we use FTP as HTTP?



HTTP1.0 Message Flow

- HTTP1.0 servers are stateless servers: each request is self-contained