Network Applications: Network App Programming: TCP FTP

Y. Richard Yang

http://zoo.cs.yale.edu/classes/cs433/

2/8/2016

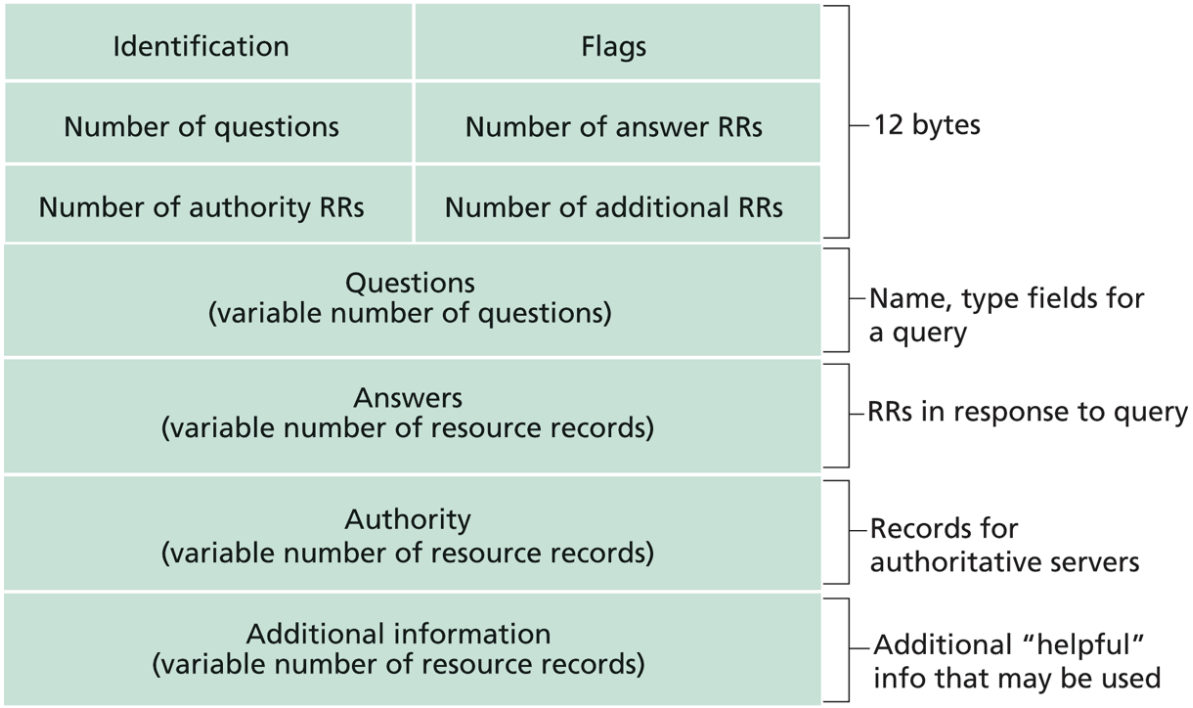
Outline

- Admin and recap
- Network app programming
- FTP

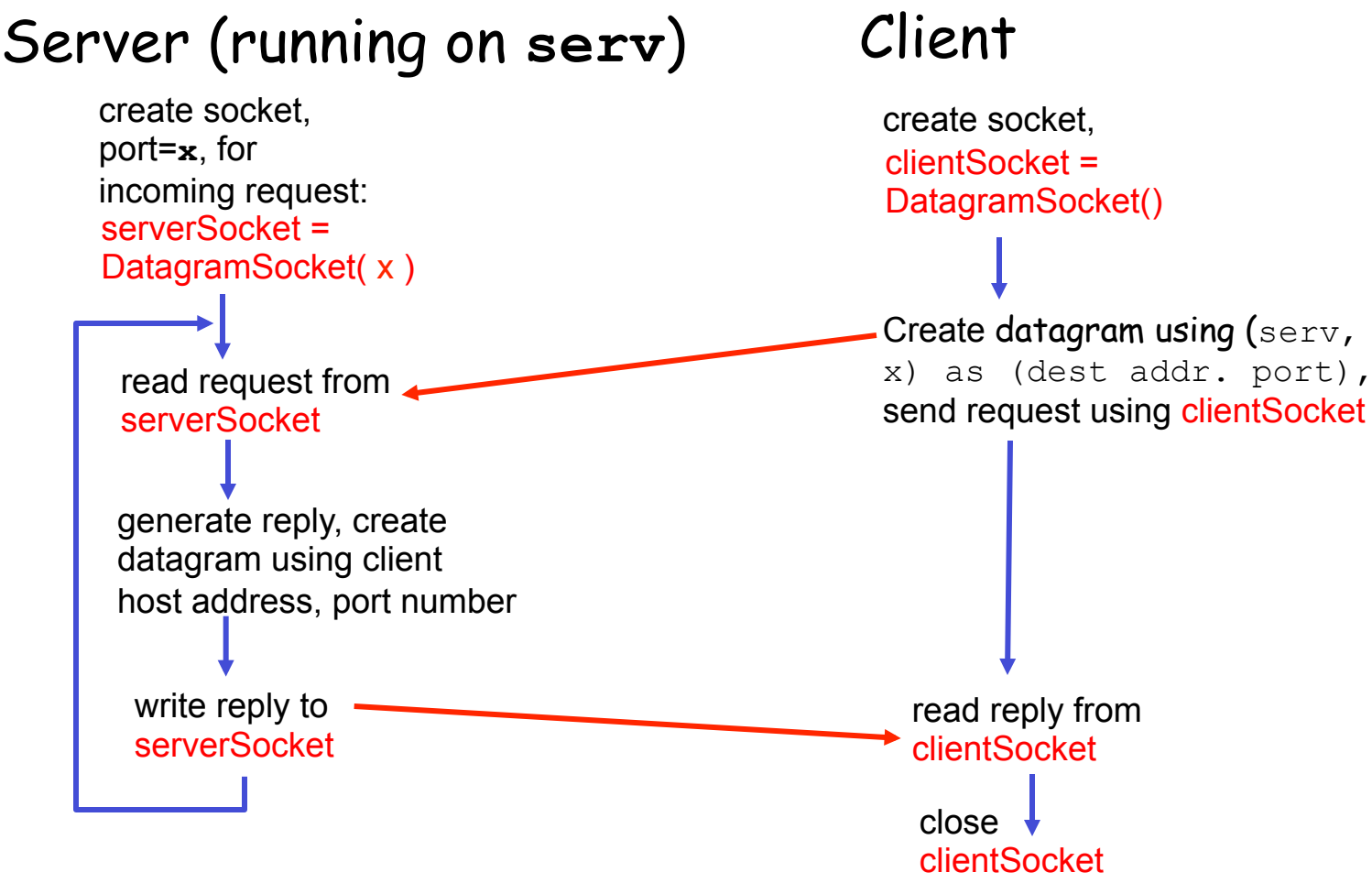
https://www.ietf.org/rfc/rfc1035.txt

Recap: DNS Protocol, Messages

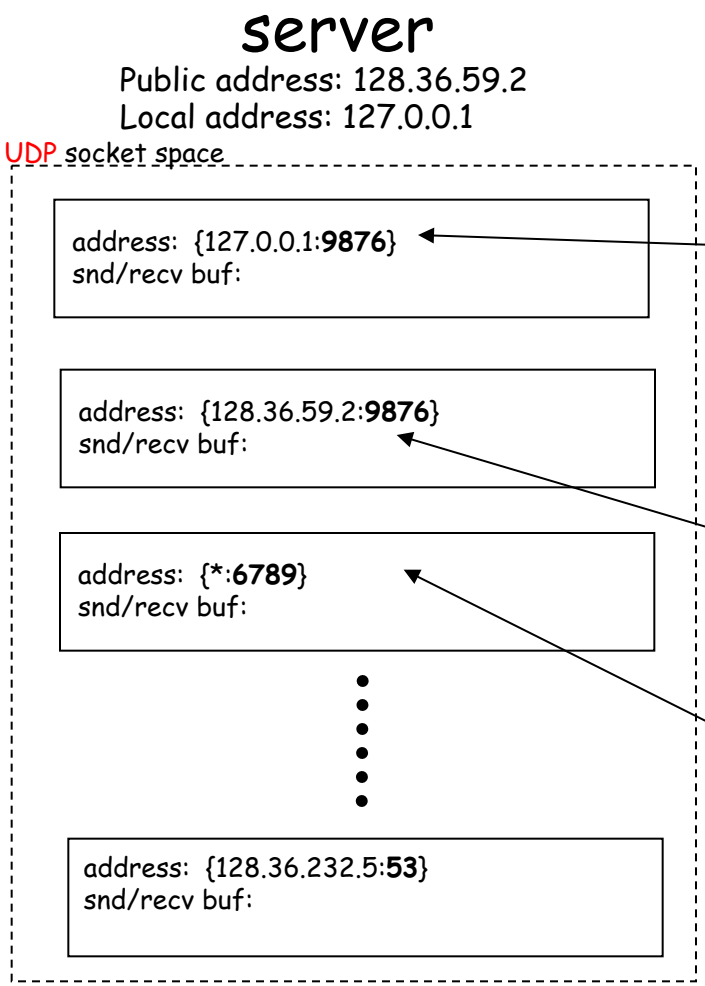
Many features: typically over UDP (can use TCP); *query* and *reply* messages with the same *message format*; *length/content encoding* of *names*; simple *compression*; *additional info* as *server push*



Recap: Connectionless UDP: Big Picture (Java version)



Recap: UDP Sockets

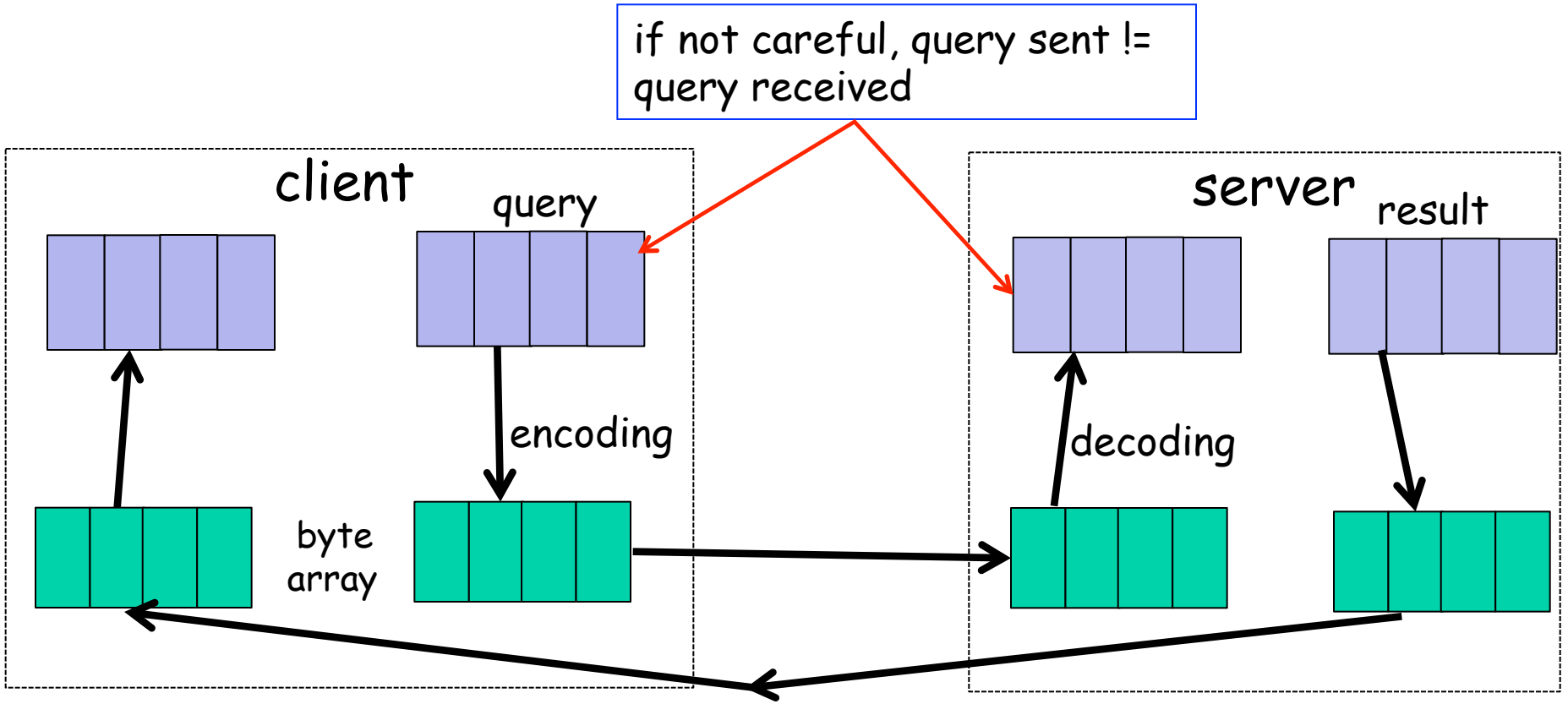



```
InetAddress sIP1 =
  InetAddress.getByName("localhost");
DatagramSocket ssock1 =
  new DatagramSocket(9876, sIP1);

InetAddress sIP2 =
 InetAddress.getByName("128.36.59.2");
DatagramSocket ssock2 =
  new DatagramSocket(9876,sIP2);

DatagramSocket serverSocket =
  new DatagramSocket(6789);
```

UDP demutiplexing is based on matching (dst address, dst port)

Recap: Data Encoding/Decoding

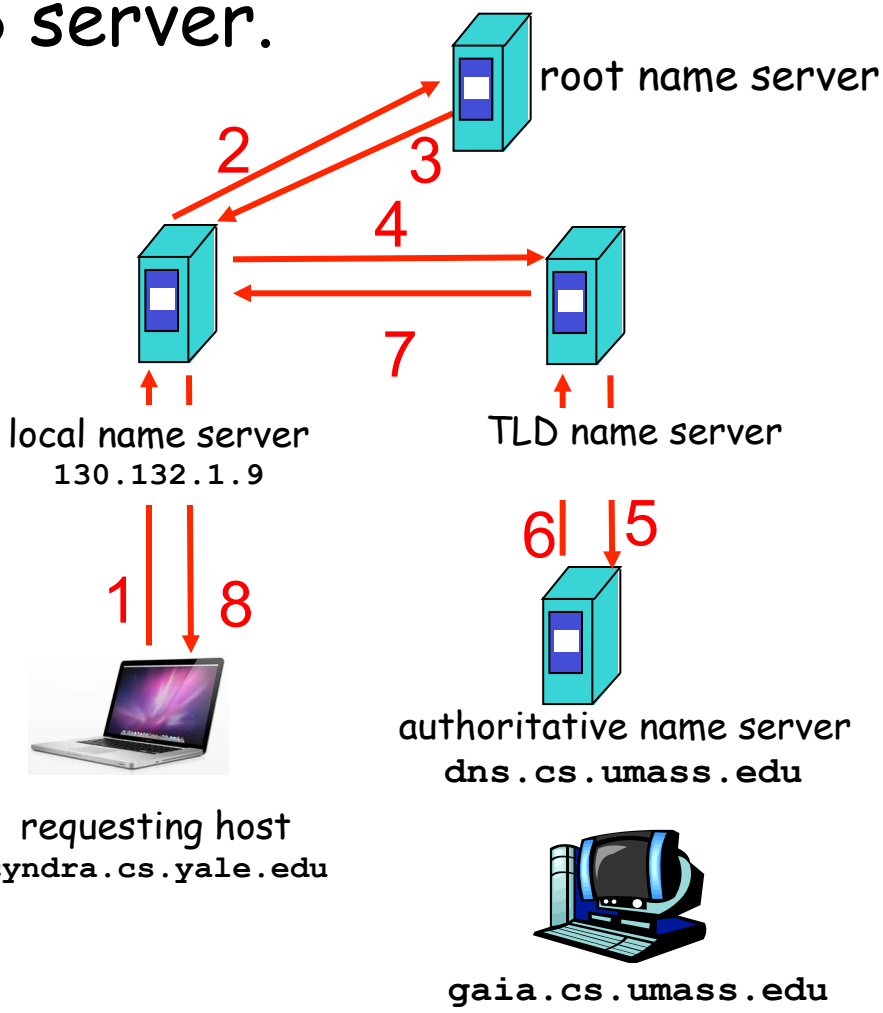
- Pay attention to encoding/decoding of data: transport layer handles only a sequence of bytes



Discussion: UDP/DNS Server Pseudocode

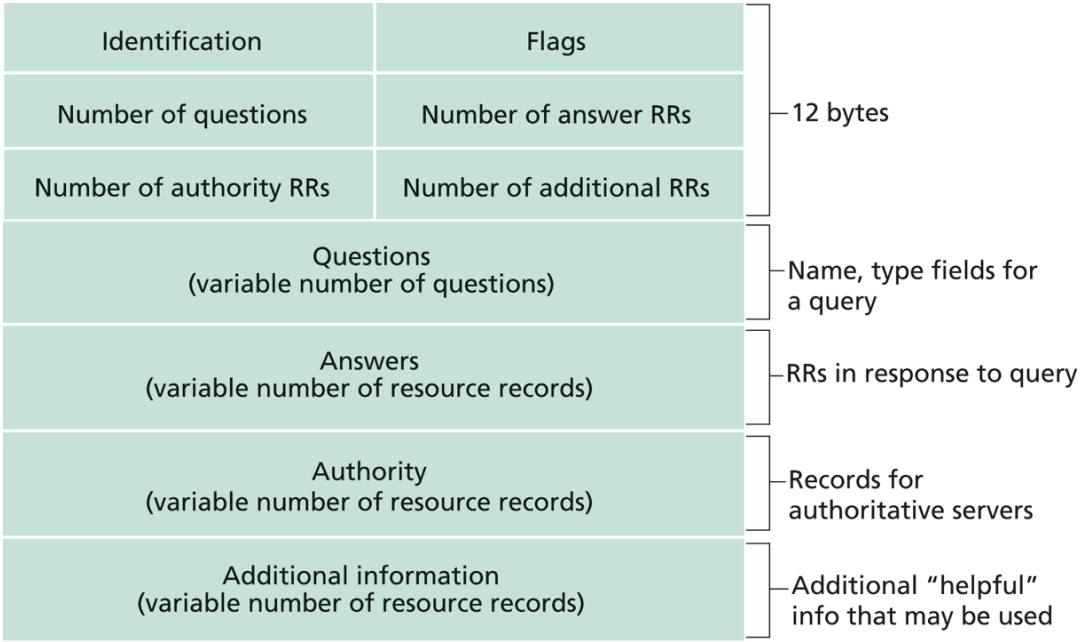
- Modify the example UDP server code to implement a local DNS server.

Identification	Flags	12 bytes
Number of questions	Number of answer RRs	
Number of authority RRs	Number of additional RRs	
Questions (variable number of questions)		Name, type fields for a query
Answers (variable number of resource records)		RRs in response to query
Authority (variable number of resource records)		Records for authoritative servers
Additional information (variable number of resource records)		Additional "helpful" info that may be used



UDP/DNS Implementation

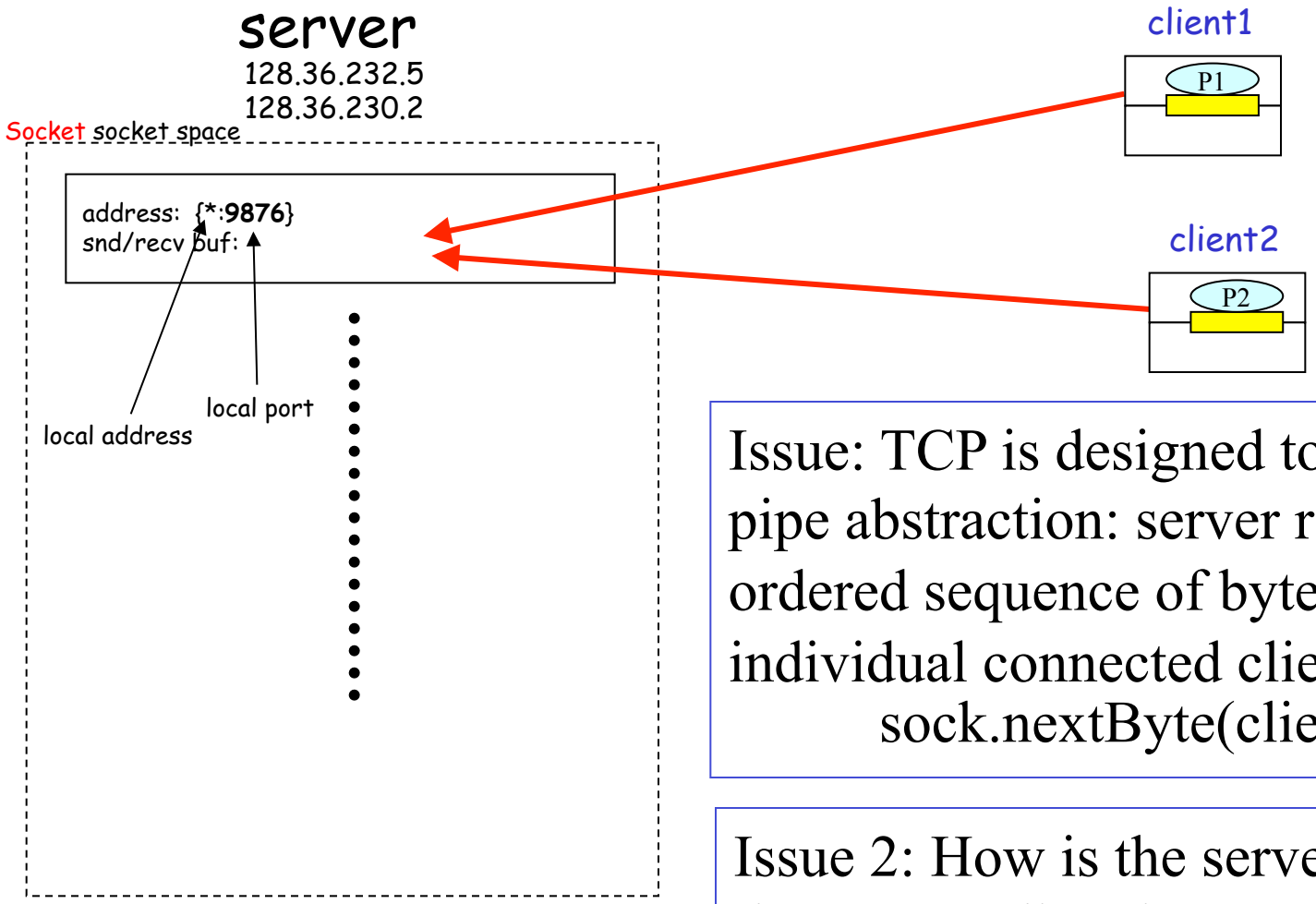
- Standard UDP demultiplexing (find out return address by src.addr/src.port of UDP packet) does not always work
- DNS solution: identification: remember the mapping



Outline

- Recap
- Network application programming
  - Overview
  - UDP
    - Basic TCP

TCP Socket Design: Starting w/ UDP



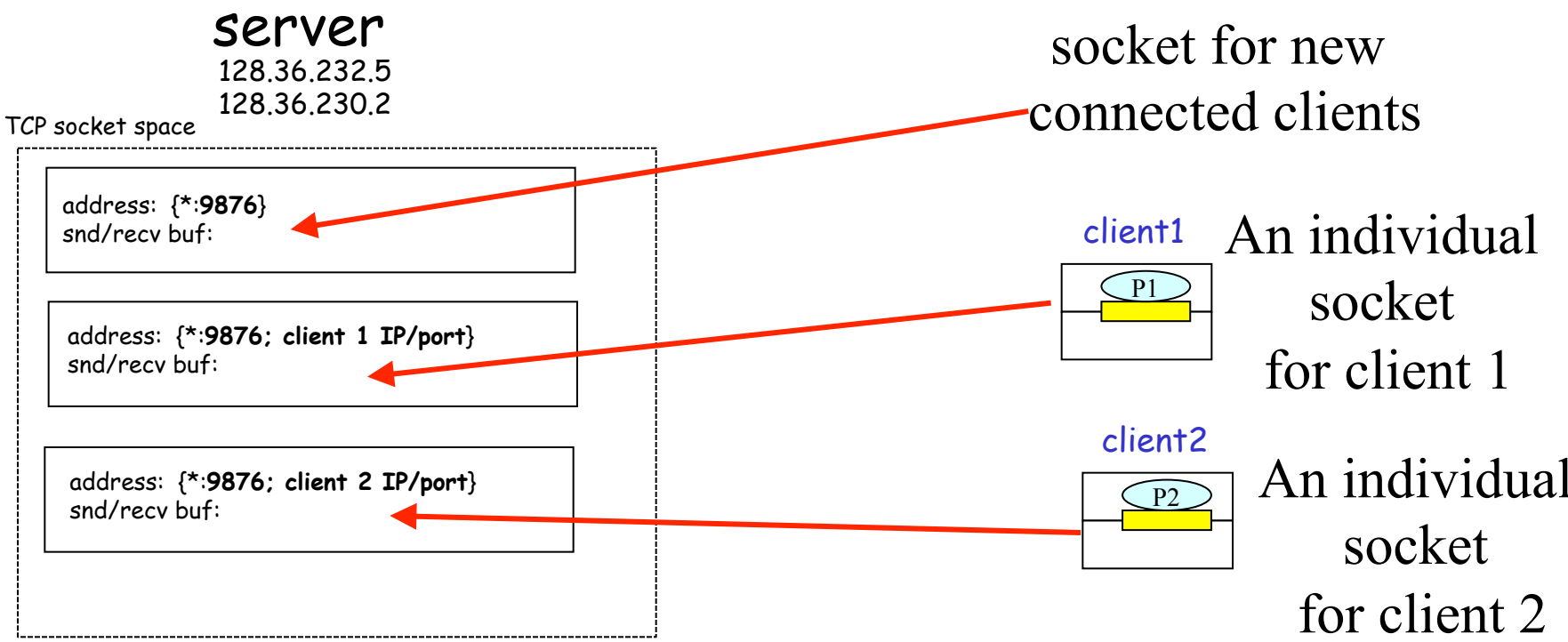
Issue: TCP is designed to provide a pipe abstraction: server reads an ordered sequence of bytes from each individual connected client

sock.nextByte(client1)?

Issue 2: How is the server notified that a new client is connected?

newClient = sock.getNewClient()?

BSD TCP Socket API Design



Q: How to decide where to put a new packet?

A: Packet demutiplexing is based on four tuples:
(dst addr, dst port, src addr, src port)

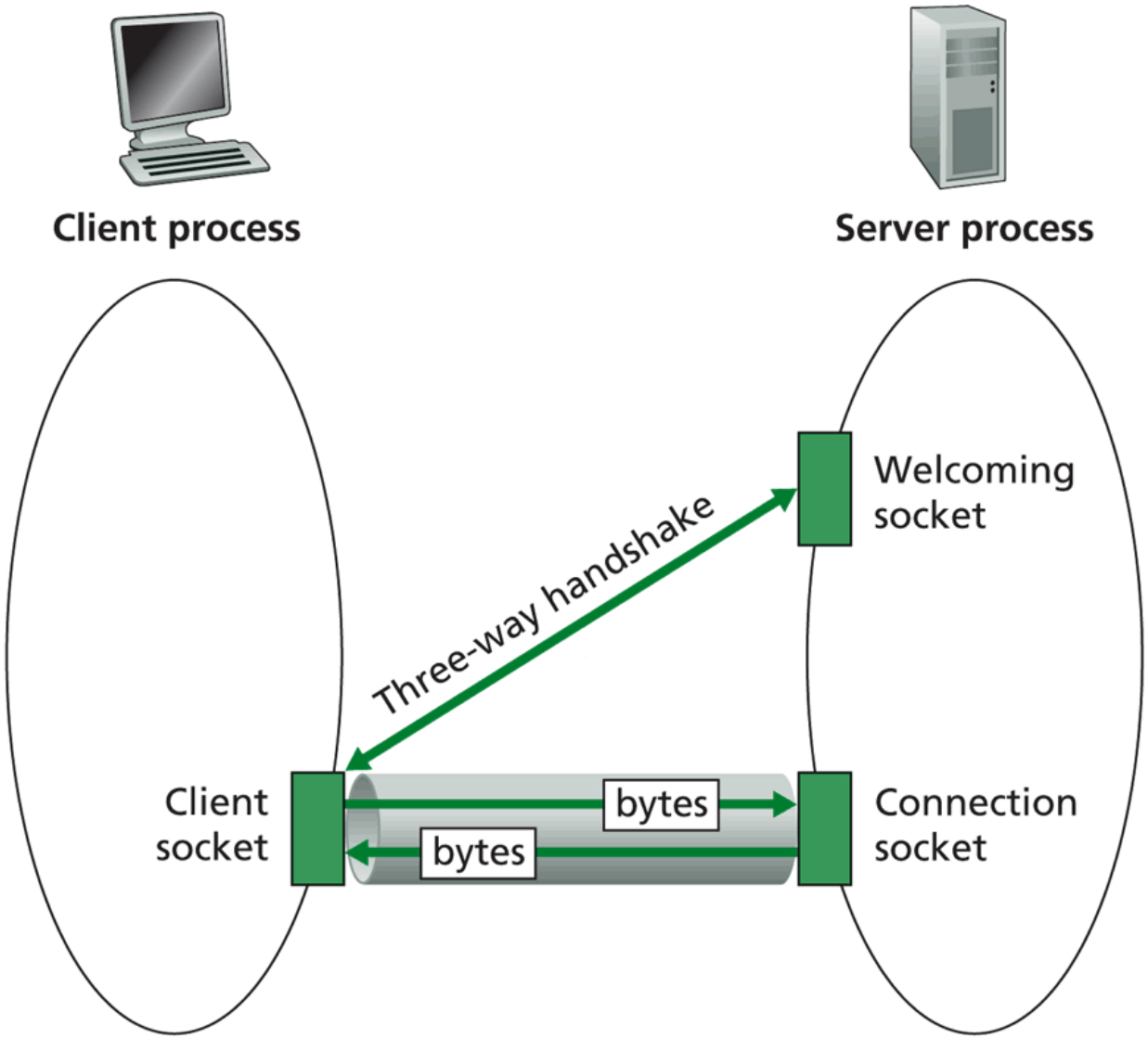
TCP Connection-Oriented Demux

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

- recv host uses all four values to direct segment to appropriate socket
  - different connections/sessions are automatically separated into different sockets

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

TCP Socket Big Picture



Client/server Socket Workflow: TCP

Server (running on `hostid`)

create socket,
port=**x**, for
incoming request:
welcomeSocket =
ServerSocket(x)

wait for incoming
connection request
connectionSocket =
welcomeSocket.accept()

read request from
connectionSocket

write reply to
connectionSocket

close
connectionSocket

TCP
connection setup

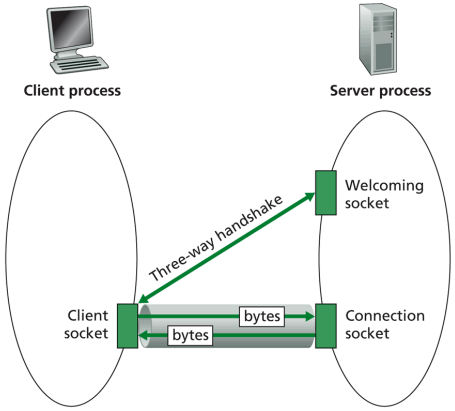
Client

create socket,
connect to hostid, port=**x**
clientSocket =
Socket()

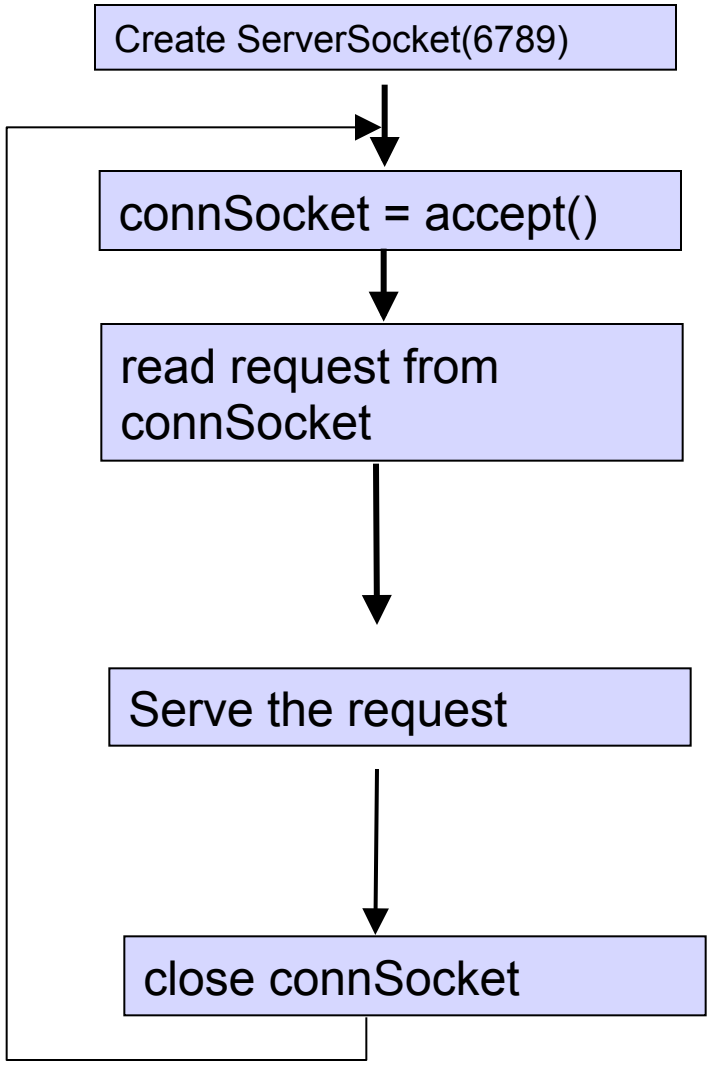
send request using
clientSocket

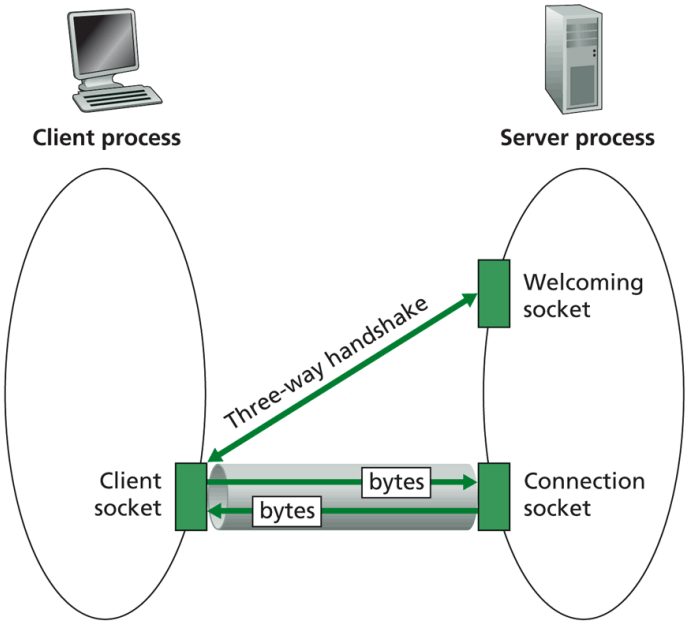
read reply from
clientSocket

close
clientSocket



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.

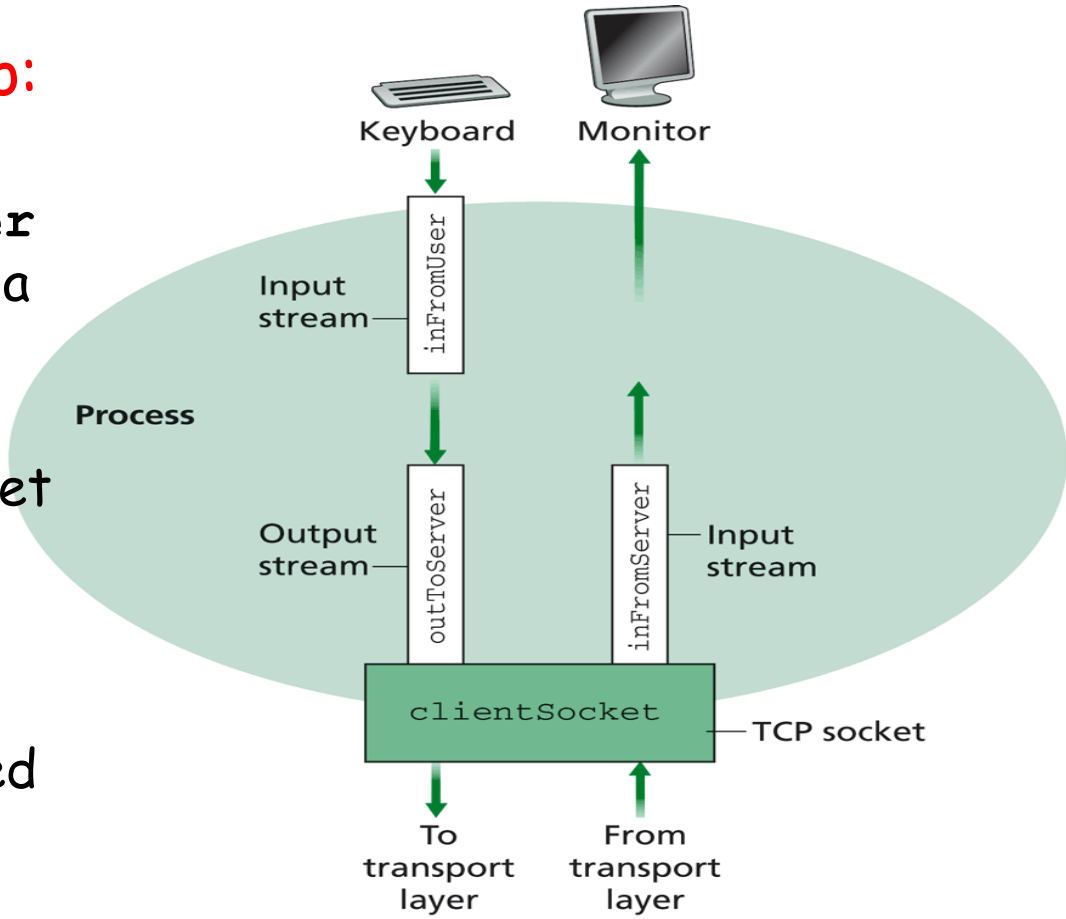
- InputStream**get InputStream**()
  returns an input stream for this socket.
- OutputStream**getOutputStream**()
  returns an output stream for this socket.

- **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.

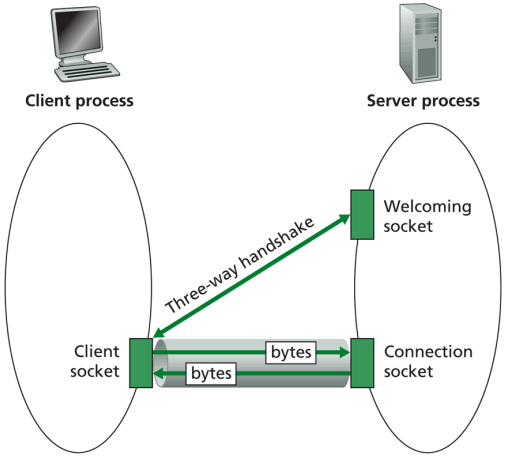
Send line to server

Create input stream attached to socket

Read line from server

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

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

modifiedSentence = inFromServer.readLine();

System.out.println("FROM SERVER: " + modifiedSentence);

clientSocket.close();

    }
}
```

Example: Java server (TCP)



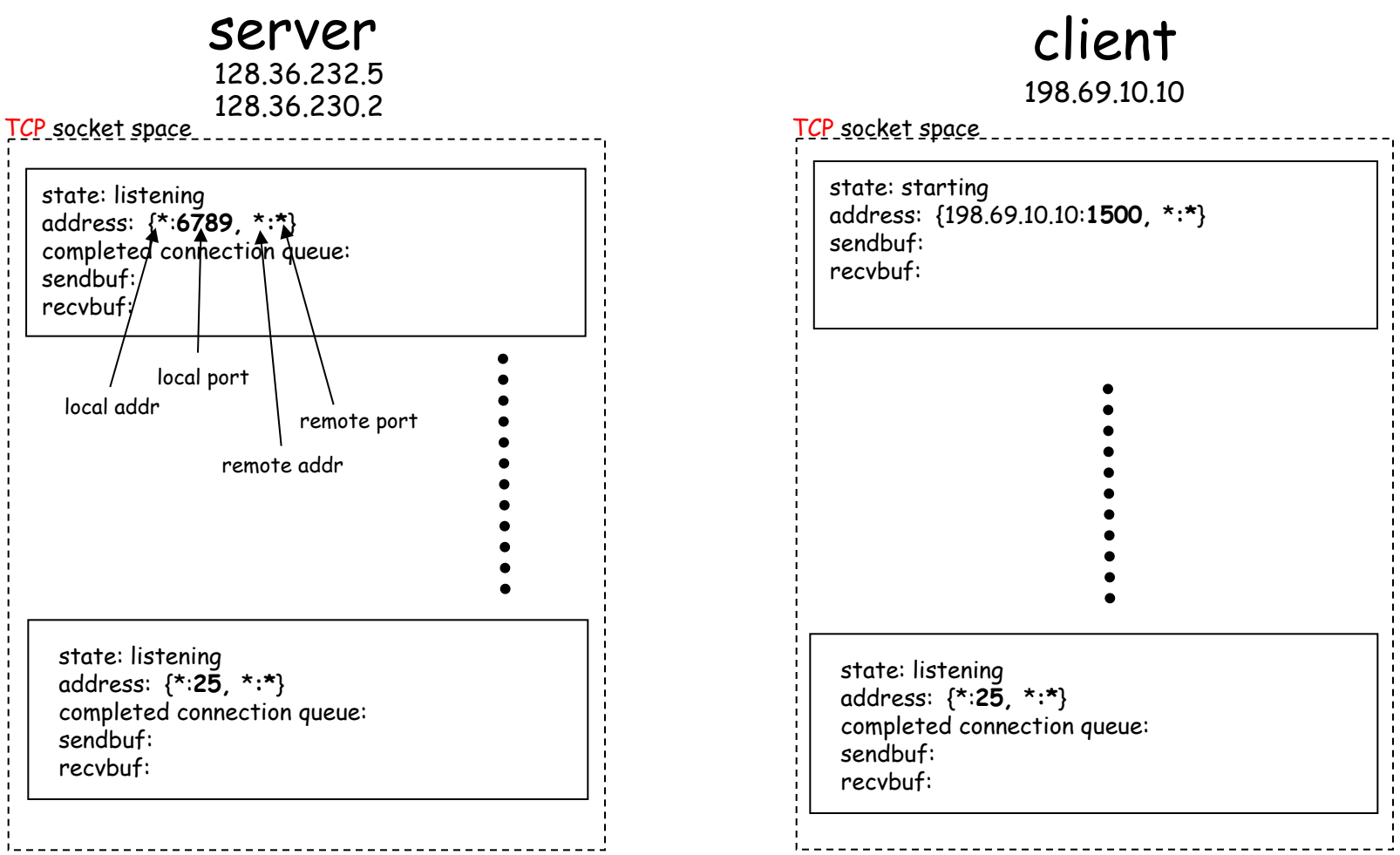

```
import java.io.*;
import java.net.*;

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

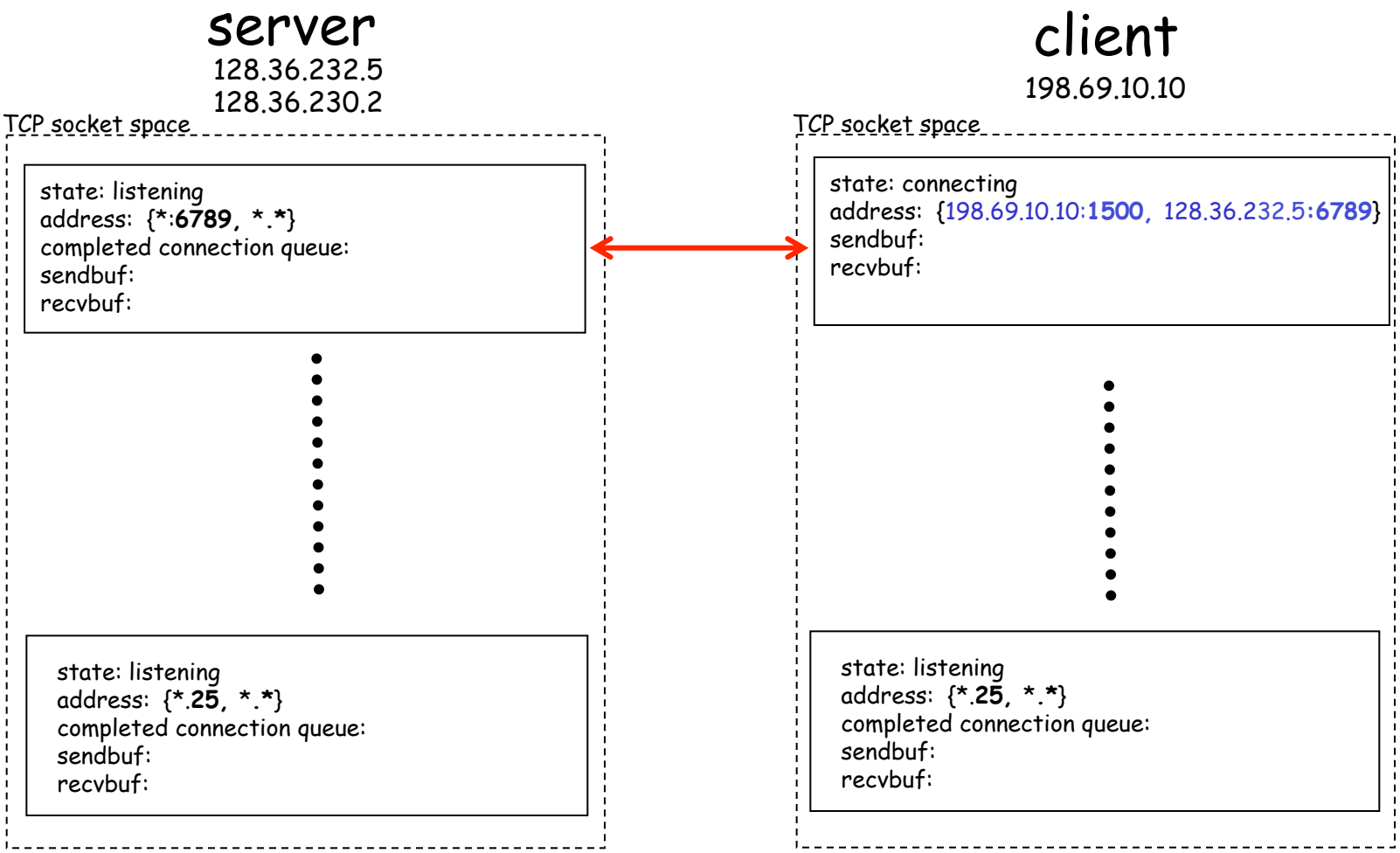
% wireshark to capture our TCP traffic
tcp.srcport==6789 or tcp.dstport==6789

Under the Hood: After Welcome (Server) Socket

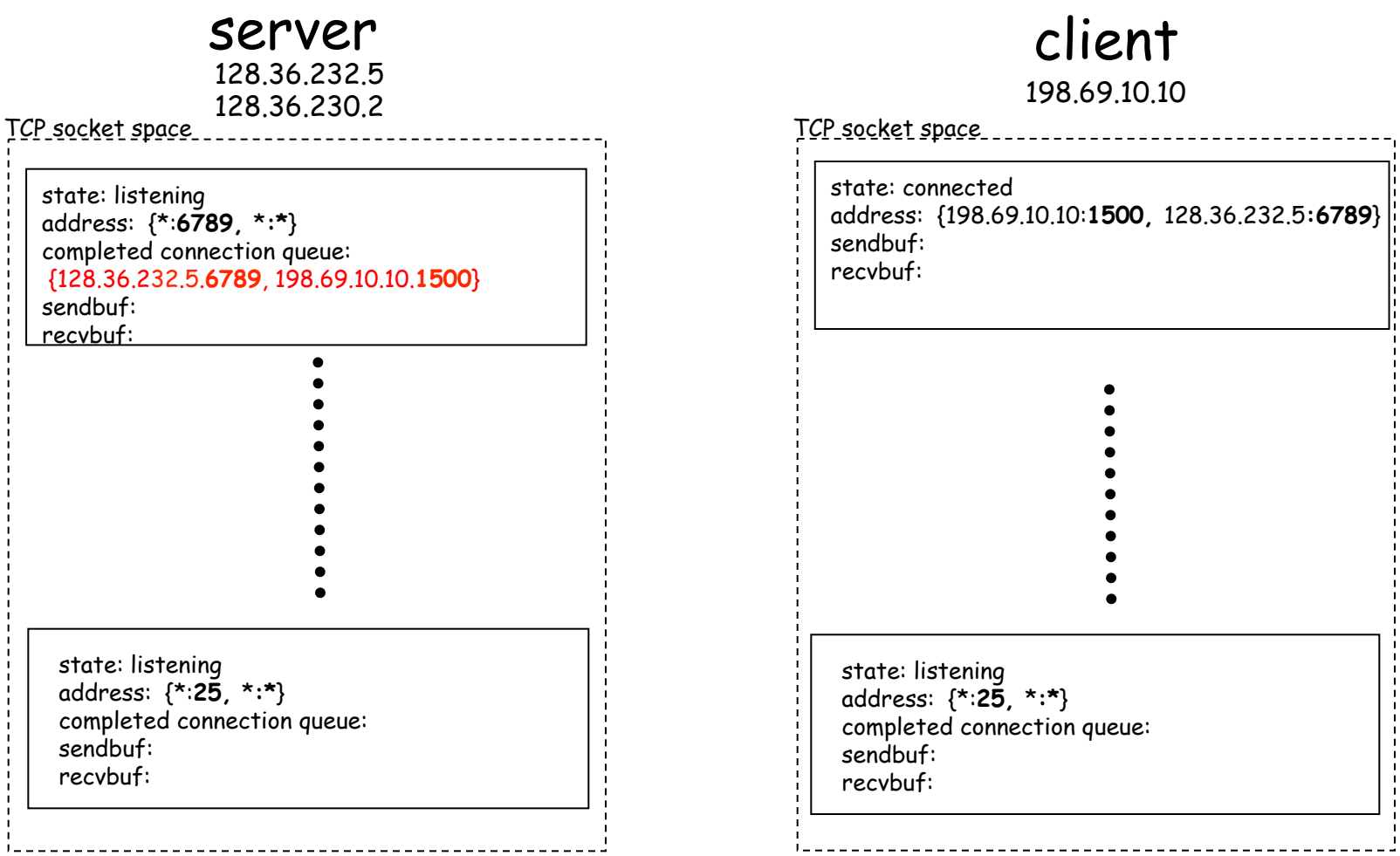


%netstat -p tcp -n -a

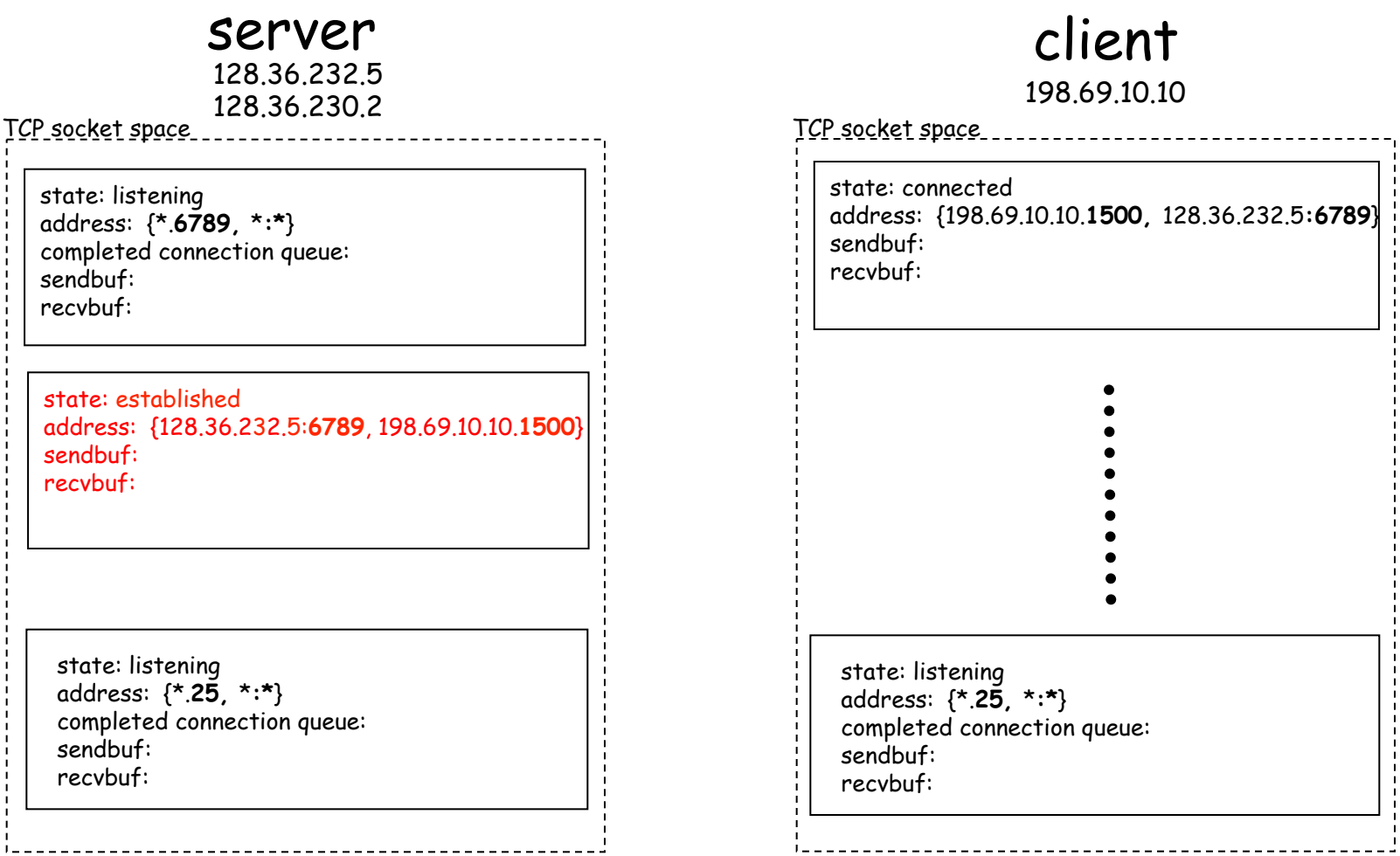
Client Initiates Connection



Example: Client Connection Handshake Done



Example: Client Connection Handshake Done



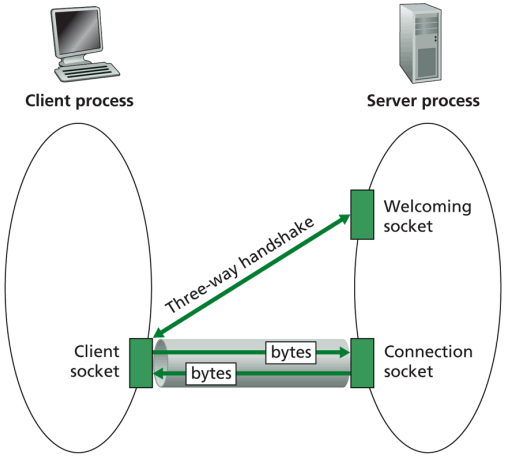
Packet demutiplexing is based on (dst addr, dst port, src addr, src port)

Packet sent to the socket with the best match!

Demo

- What if more client connections than backlog allowed?

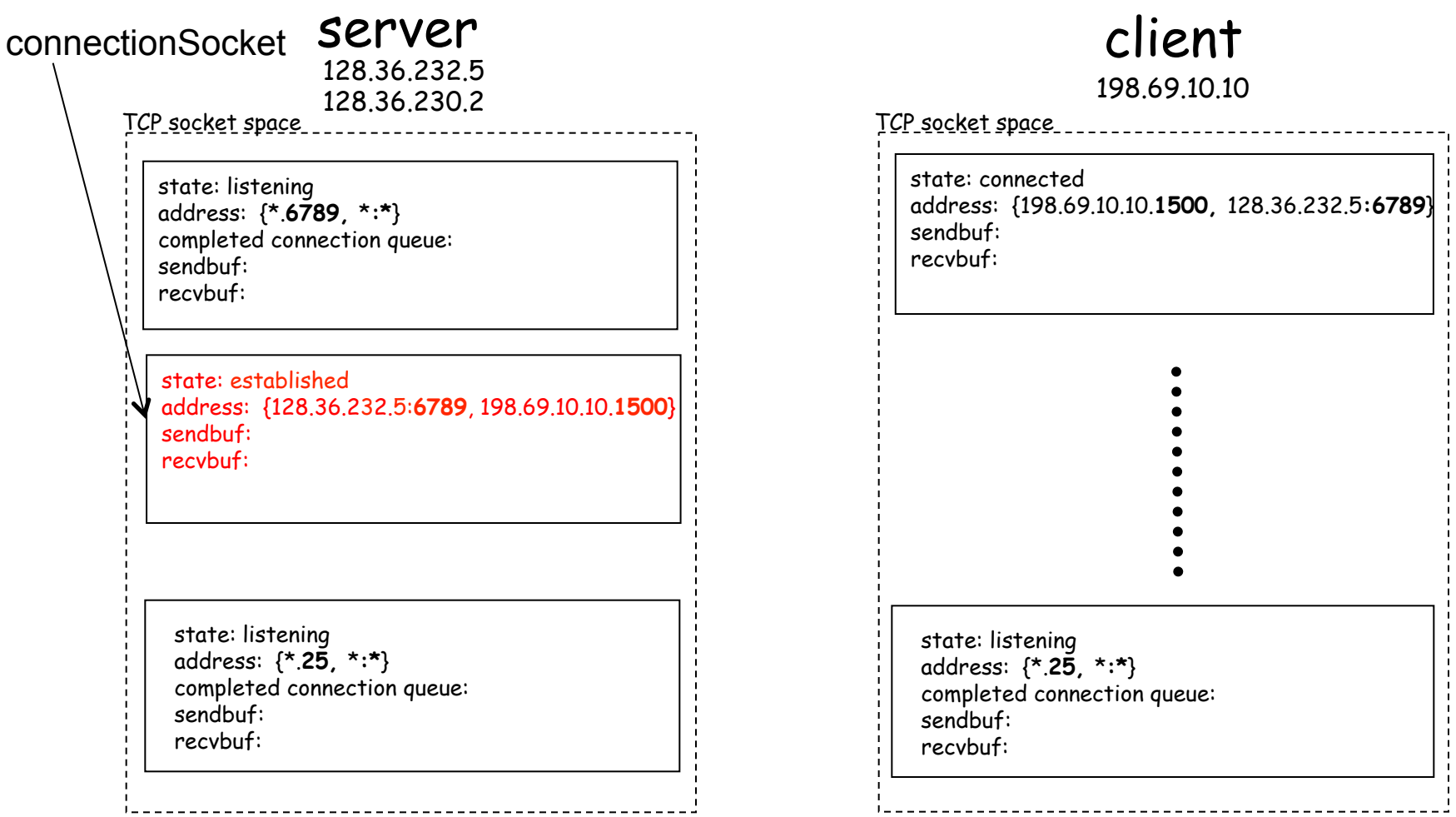
Example: Java server (TCP)




```
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());
```

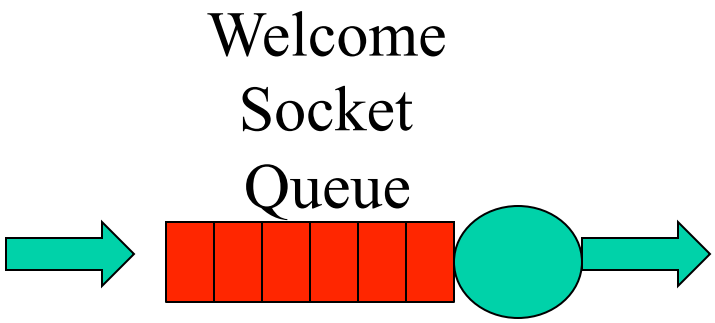
Write out line to socket

```
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
- Some basic questions
  - How big is the backlog (welcome queue)



Analysis

- Is there any interop issue in the sample program?

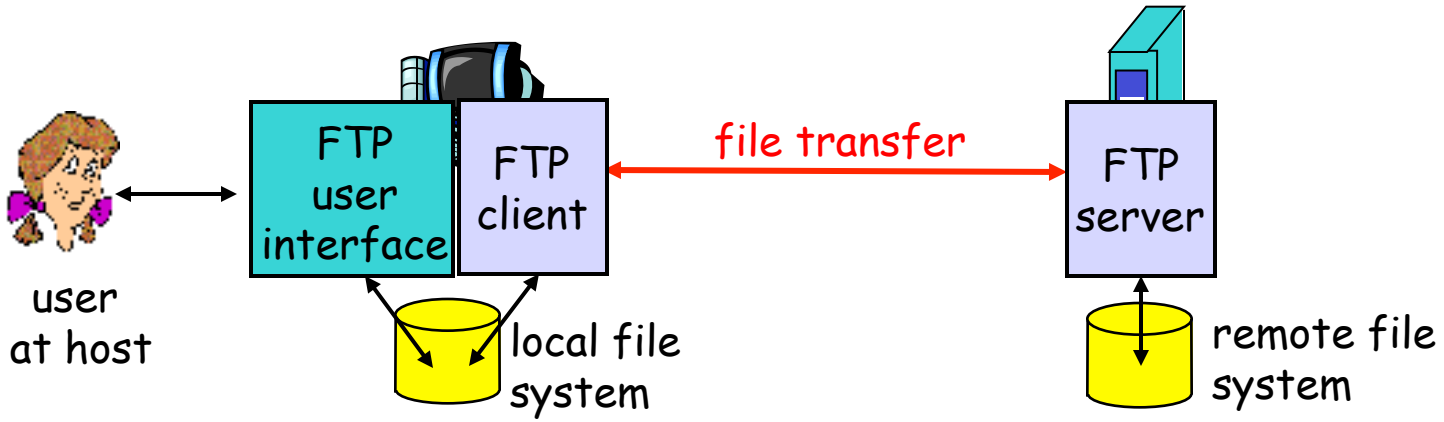
Analysis

- 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)

Outline

- Recap
- Network application programming
- FTP

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:

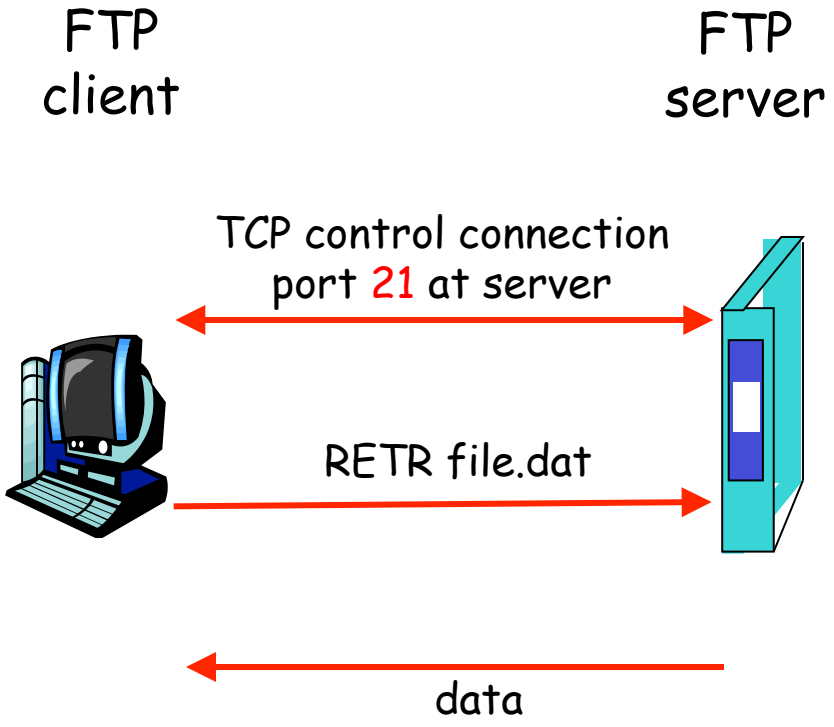
- sent as ASCII text over control channel
- **USER *username***
- **PASS *password***
- **PWD** returns current dir
- **STAT** shows server status
- **LIST** returns list of file in current directory
- **RETR filename** retrieves (gets) file
- **STOR filename** stores file

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?



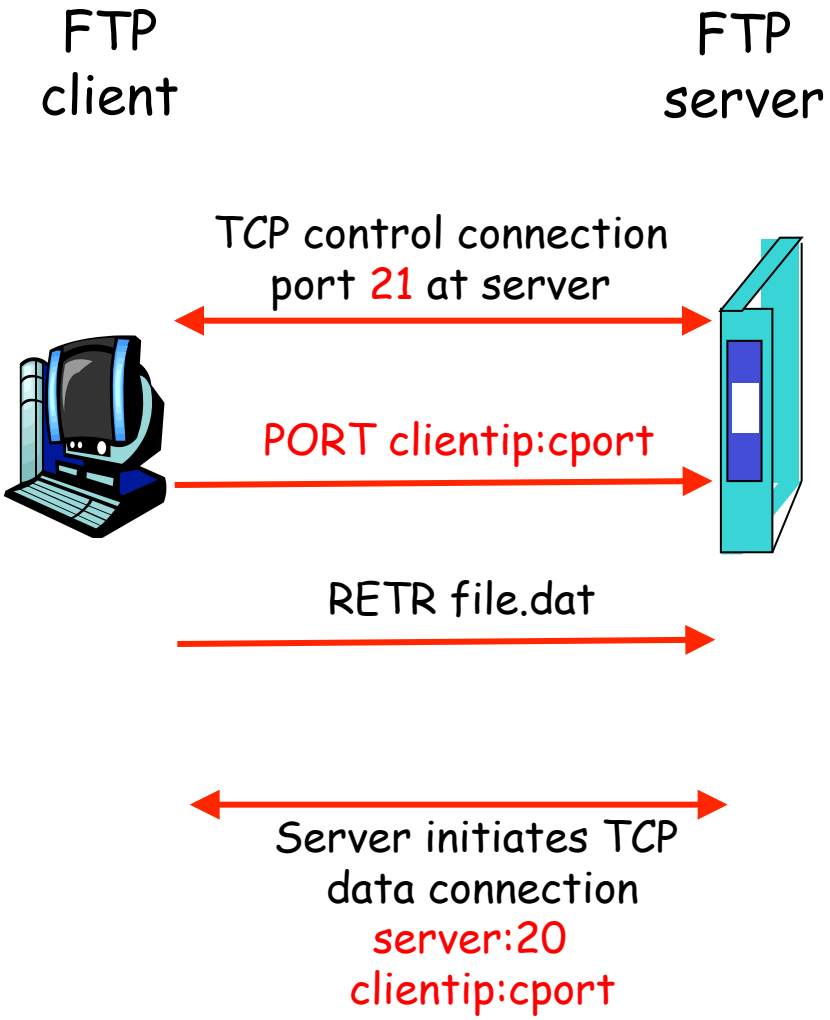
FTP: A Client-Server Application with Separate Control, Data Connections

- 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

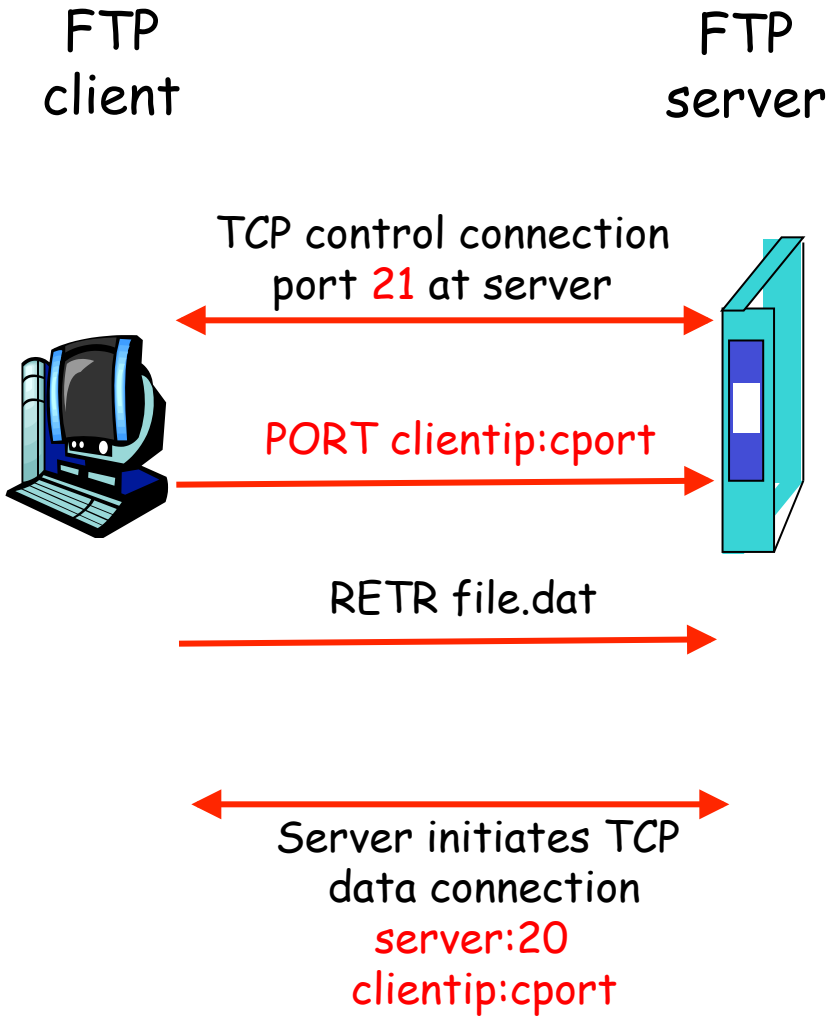


Example using telnet/nc

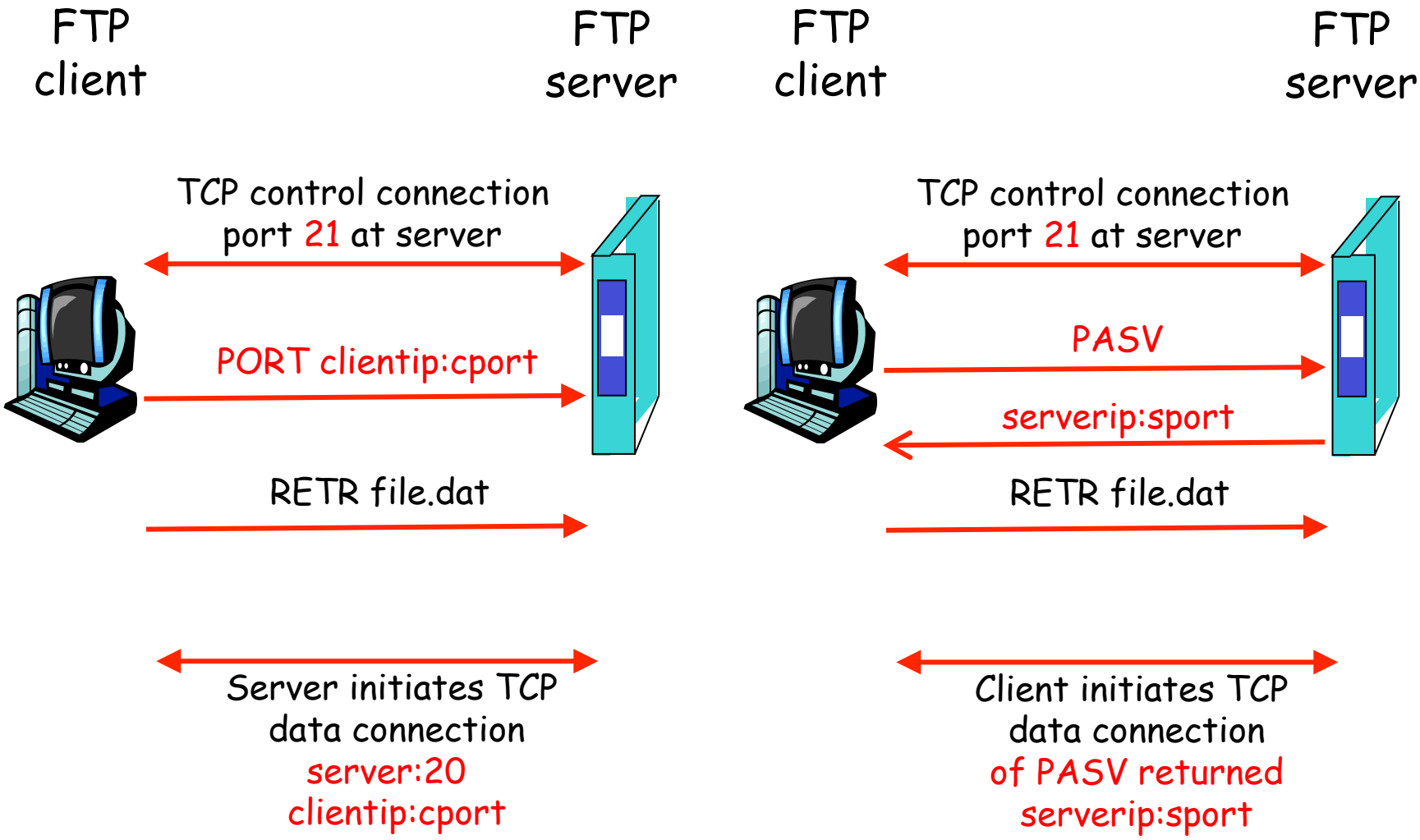
- Use telnet for the control channel
  - telnet ftp.freebsd.org 21
  - user, pass
  - port 172,27,5,145,4,1
  - list
- use nc (NetCat) to receive/send data with server
  - nc –v –l 1025

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



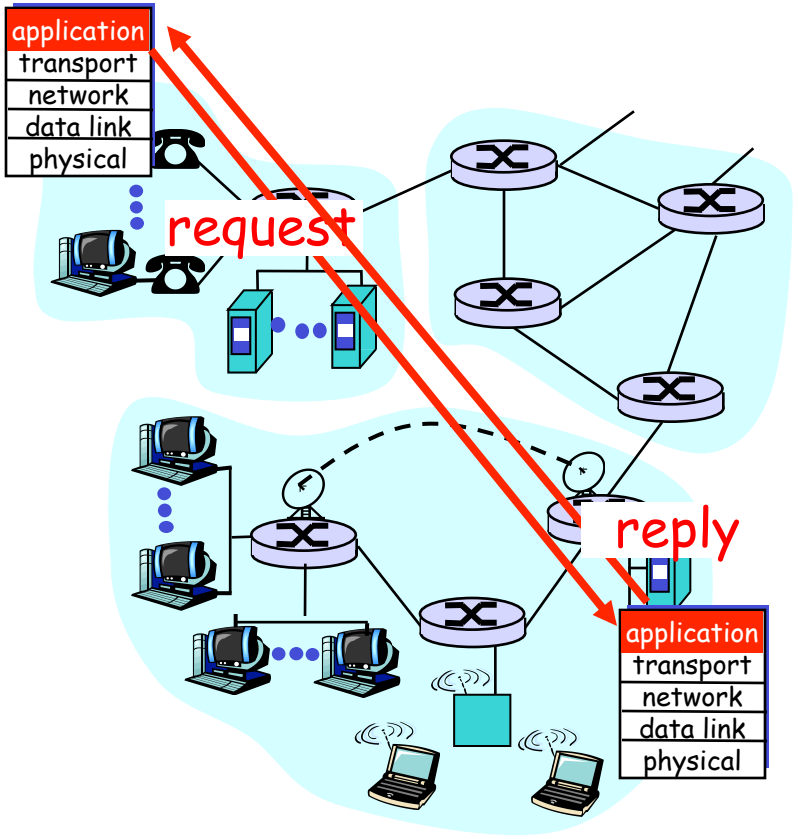
Example

- Use Wireshark to capture traffic
  - Using chrome to visit ftp://ftp.freebsd.org/pub/FreeBSD/README.TXT

FTP Evaluation

Key questions to ask about a C-S application

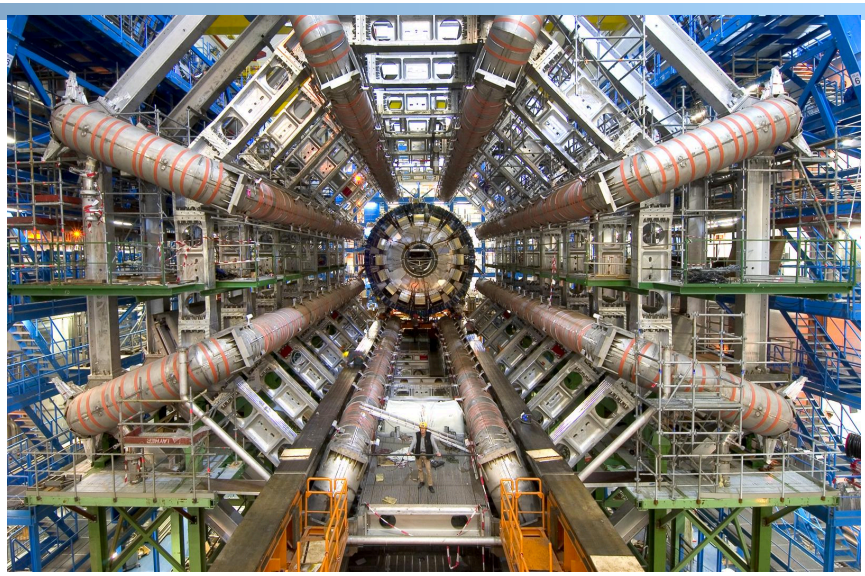
- **Is the application extensible?**
- **Is the application scalable?**
- **How does the application handle server failures (being robust)?**
- **How does the application provide security?**



What are some interesting design features of the FTP protocol?

FTP Extensions

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



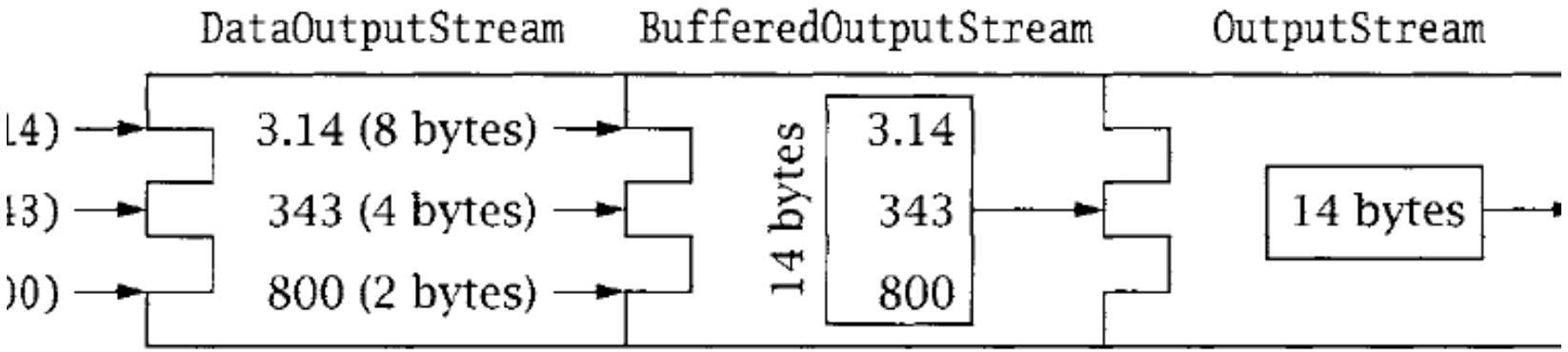
- See GridFTP to FTP extensions
  - https://www.ogf.org/documents/GFD.20.pdf

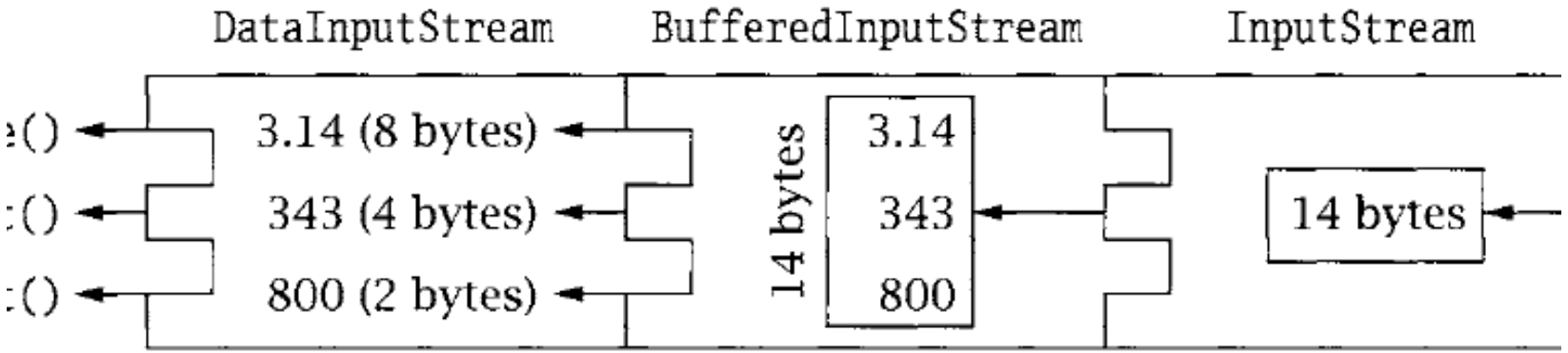
Outline

- Recap
- Network application programming
- FTP
- HTTP

Backup Slides

DataStream


DataOutputStream
BufferedOutputStream
OutputStream
3.14 (8 bytes)
343 (4 bytes)
800 (2 bytes)
14 bytes
3.14
343
800
14 bytes


DataInputStream
BufferedInputStream
InputStream
3.14 (8 bytes)
343 (4 bytes)
800 (2 bytes)
14 bytes
3.14
343
800
14 bytes

Demo

% wireshark to capture traffic
tcp.srcport==6789 or tcp.dstport==6789