Computer Networks

Adrian Sergiu DARABANT

Lecture 1

Introduction - Administrative

- Weekly:
  - Lectures
  - Labs - Each lab we practice things (programming and networking simulations).
  - You do not have individual scheduled assignments – but you need to be able to solve any problem from the proposed set
- Final grade (average of):
  - After each lab session (approx. 6 weeks) there will be a test counting towards your final grade
  - Final written examination / One Moodle test towards the end of semester
- Prerequisites
  - C/C++ system programming (Unix and Windows)
  - Operating systems
  - Python, PHP (interpreter)

Bibliography

1. **J. Kurose, K. Ross, Computer Networking: A Top Down Approach, Addison-Wesley, rev2,3,4 2002-2007**.
2. A.S. Tanenbaum – Computer Networks 4th ed., Prentice Hall, 2003
3. Douglas E. Comer, Internetworking with TCP/IP
    1. Vol 1- Principles, Protocols, and Architecture
    2. **Vol 3- Client-Server Programming and Applications**
4. G.R.Wright, R. Stevens, TCP/IP Illustrated – vol 1,2, Addison Wesley.
5. Matt Naugle, Illustrated TCP/IP – A Graphic Guide to protocol suite, John Willey & Sons, 1999.
6. W. Richard Stevens, Bill Fenner, Andrew M. Rudoff, UNIX® Network Programming Volume 1, Third Edition: The Sockets Networking API

Course Information

http://www.cs.ubbcluj.ro/~dadi/compnet

Required (?!) Tools/Materials

- Windows 32/64 +Linux – Development Env !
- VMware Player/ Virtual Box, etc
  - Install Linux / Windows !
  - Integration Tools
  - **Development Environment** Or **Vi** ?!?
- Set networking as <u>bridged</u> on VM!

You will thank me later :P

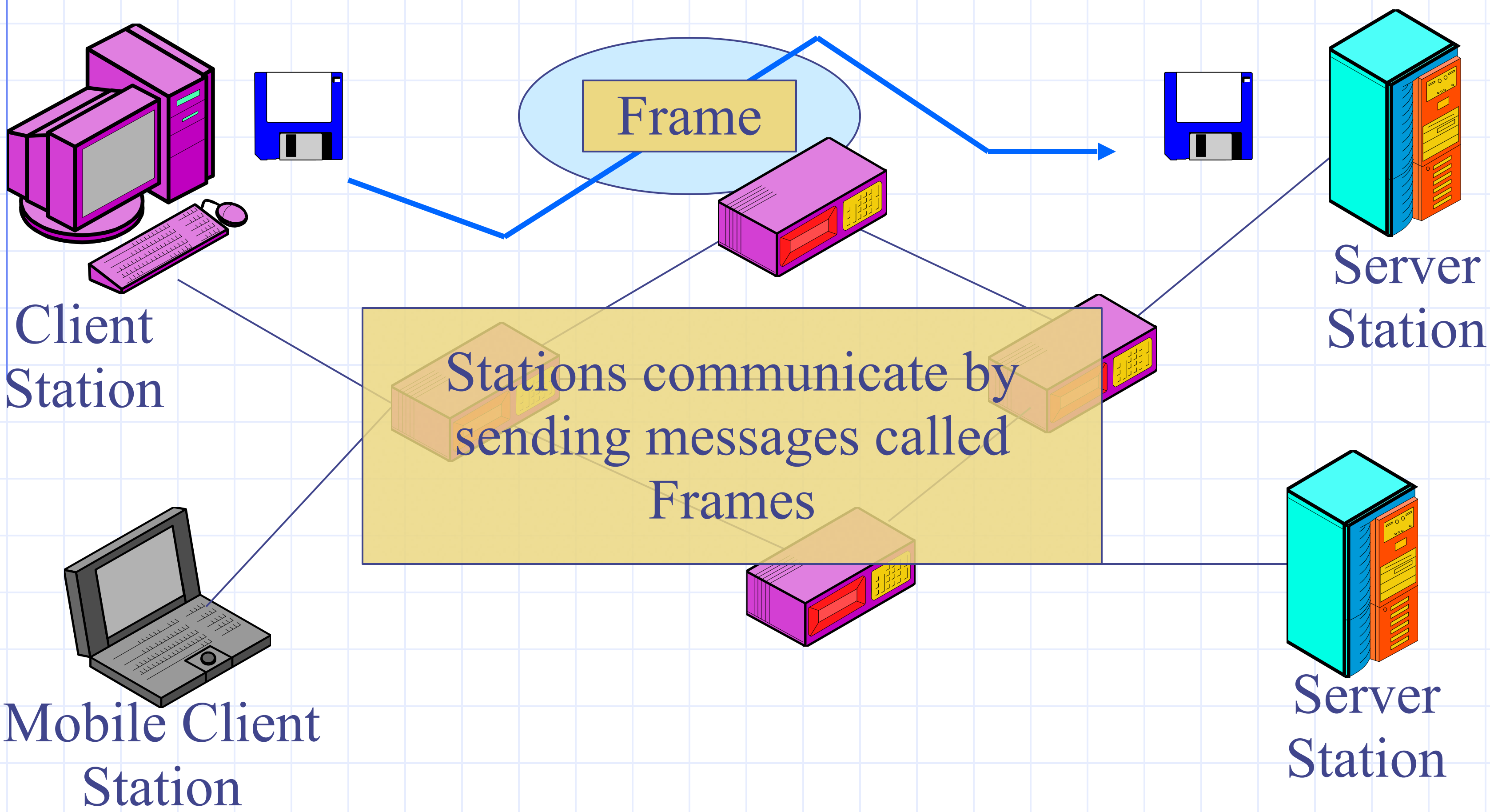
Syllabus – What are we going to study ?

1. TCP/IP Programming
   1. TCP sockets
   2. UDP sockets
2. Communication Protocols and Hierarchies
3. The OSI and TCP/IP layered architecture
4. The IP Protocol
   1. IP Addressing schemas
   2. Helper protocols: DHCP, ARP, DNS
5. TCP and UDP
6. Routing

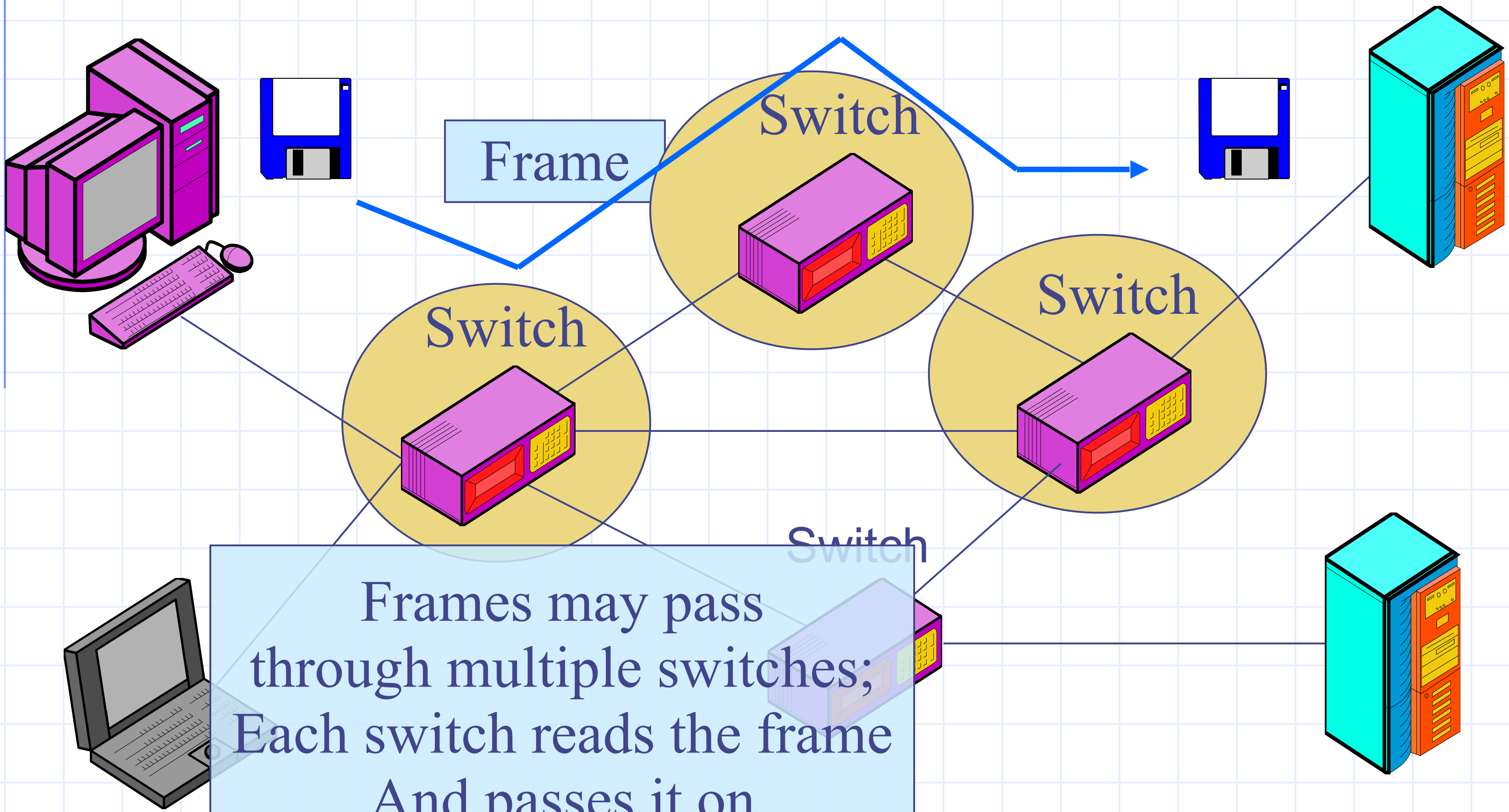
What is a Computer Network ?

- A collection of computers (PCs, Workstations) and other devices interconnected.
- **Components**:
  - Hosts (computers)
  - Links (coaxial cable, twisted pair, optical fiber, radio, satellite)
  - Switches/routers (intermediate systems)

What is a Computer Network/LAN?

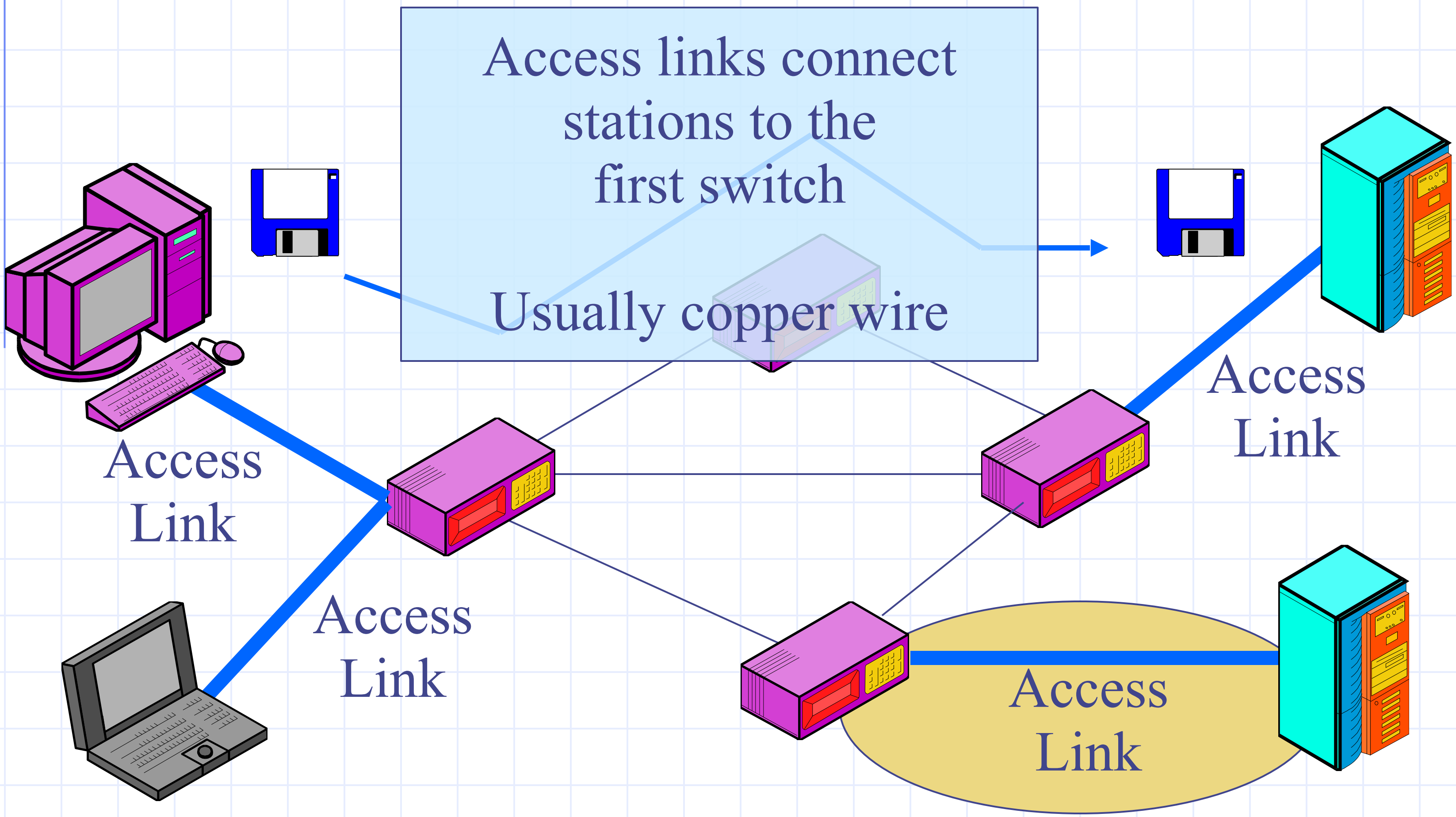


What is a Computer Network/LAN?

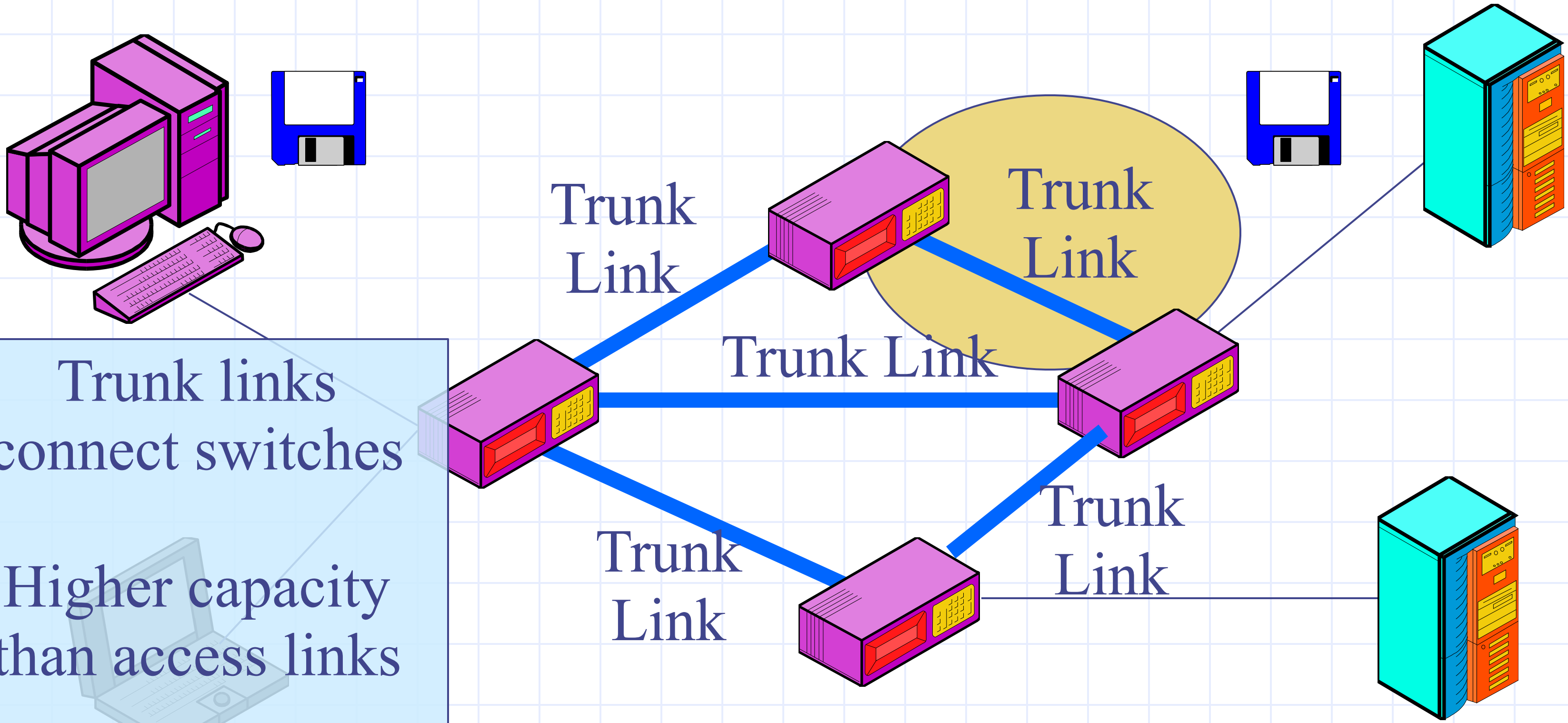


Frames may pass
through multiple switches;
Each switch reads the frame
And passes it on

What is a Computer Network/LAN?



What is a Computer Network/LAN?



Trunk links connect switches

Higher capacity than access links

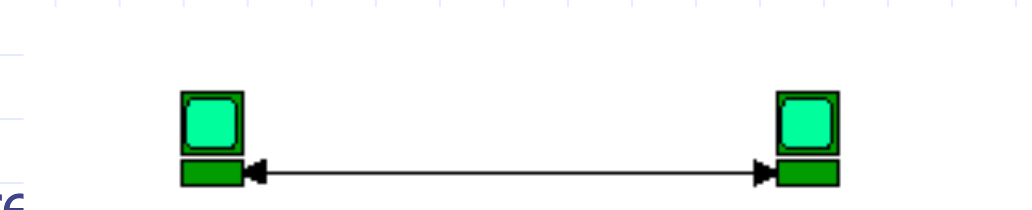
Often optical fiber

Classifications

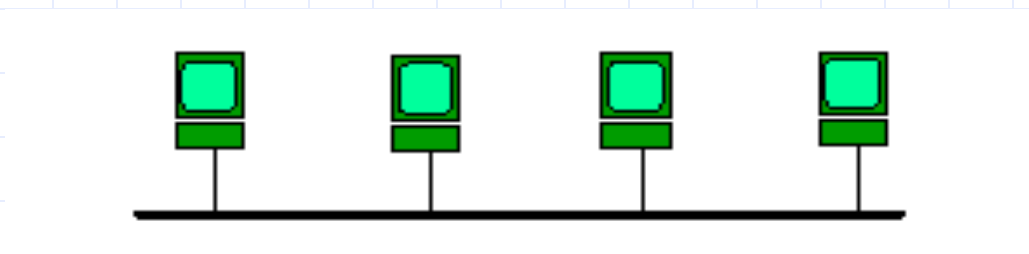
- Types of links
  - Direct links
  - Bus type links
- Type of transmission
  - Circuit switched networks
  - Packet switched networks
  - Frame Relay
  - Asynchronous Transfer Mode (ATM)

Types of communication

1. **Types of links (connectivity)**



Direct Point to point communication

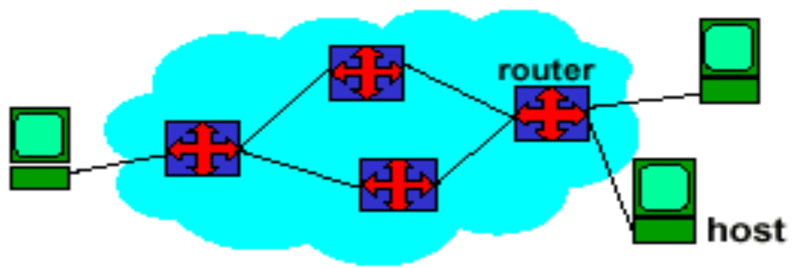


ss

Types of Communication

Switched Networks

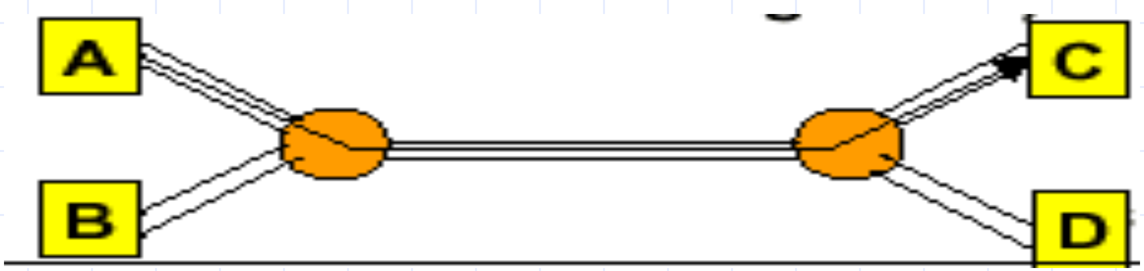
- Circuit - switched network: public telephone network



- Packet switched network: Internet (collection of networks)

Circuit-Switching

- Set up a connection path (circuit) between the source and the destination (permanent for the lifetime of the connection)
- network resources (e.g., bandwidth) divided into "pieces". Pieces allocated to calls
- Resource piece idle if not used by owning call (no sharing)
- All bytes follow the same dedicated path
- While A talks to C, B cannot talk to D on the same line.
- Piece Division ? TimeSlot vs frequency division



Packet Switching

each end-end data stream divided into packets

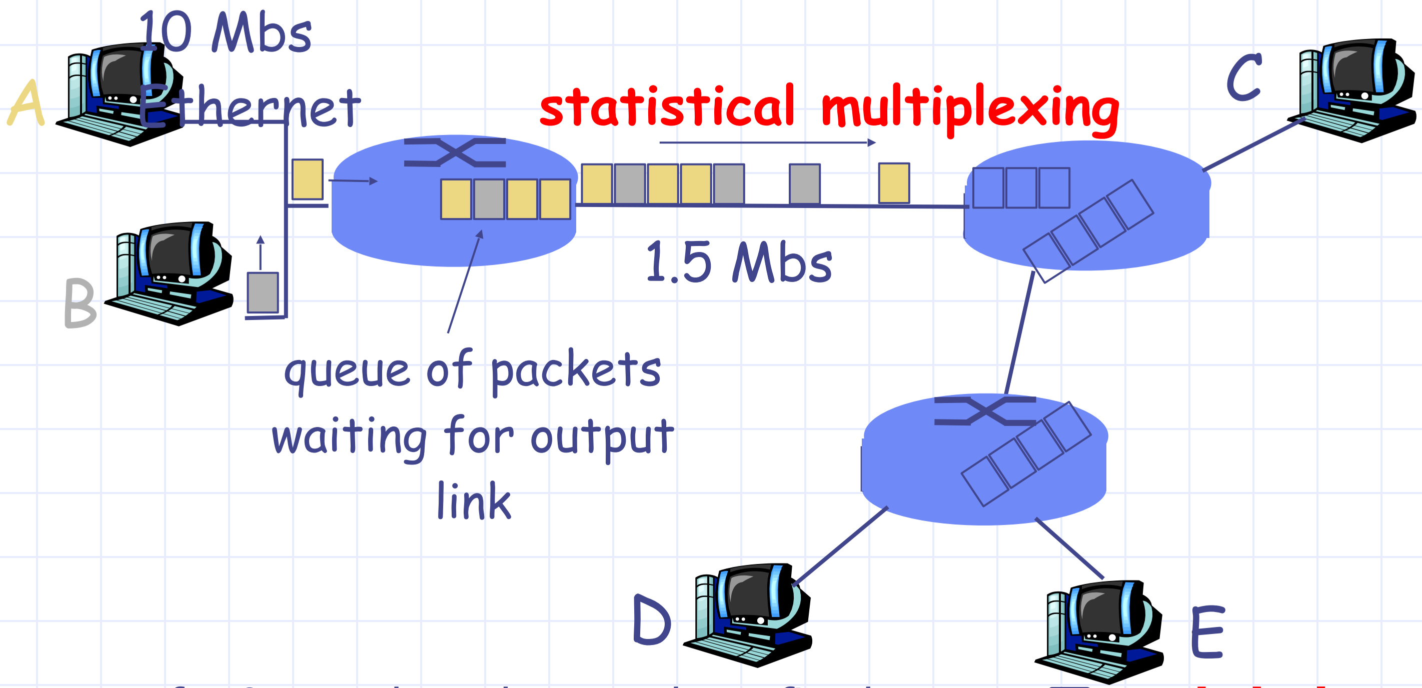
- user A, B packets share network resources
- each packet uses full link bandwidth
- resources used as needed

Bandwidth division into "pieces"
Dedicated allocation
Resource reservation

resource contention:

- aggregate resource demand can exceed amount available
- congestion: packets queue, wait for link use
- store and forward: packets move one hop at a time
  - transmit over link
  - wait turn at next link

Packet Switching: Statistical Multiplexing

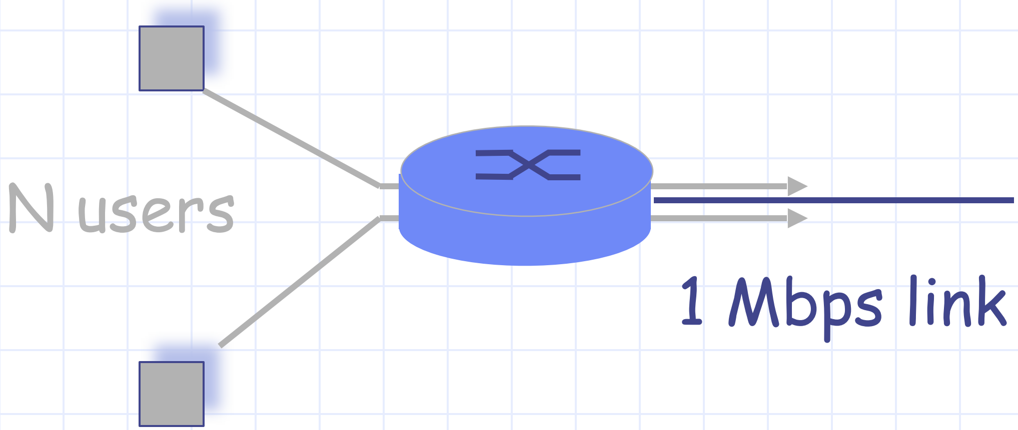


- Sequence of A & B packets does not have fixed pattern ? **statistical multiplexing**.
- Efficient use of resources: Nobody reserves a lane on a freeway!
- Can accommodate bursty traffic (as opposed to circuit-switching where transmission is at constant rate).

Packet switching versus circuit switching

Packet switching allows more users to use network!

- 1 Mbit link
- each user:
  - 100 kbps when “active”
  - active 10% of time
- circuit-switching:
  - 10 users
- packet switching:
  - with 35 users, probability > 10 active less than .0004

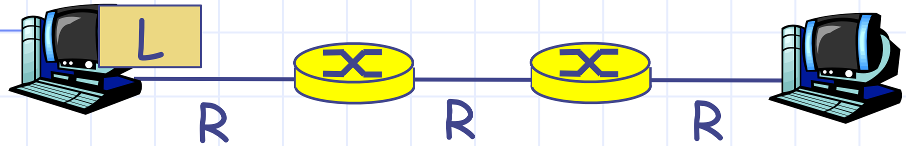


Packet switching versus circuit switching

Is packet switching a "slam dunk winner?"

- Great for bursty data
  - resource sharing
  - simpler, no call setup
- Excessive congestion: packet delay and loss
  - protocols needed for reliable data transfer, congestion control
- Q: How to provide circuit-like behavior?
  - bandwidth guarantees needed for audio/video apps
  - still an unsolved problem (chapter 6)

Packet-switching: store-and-forward

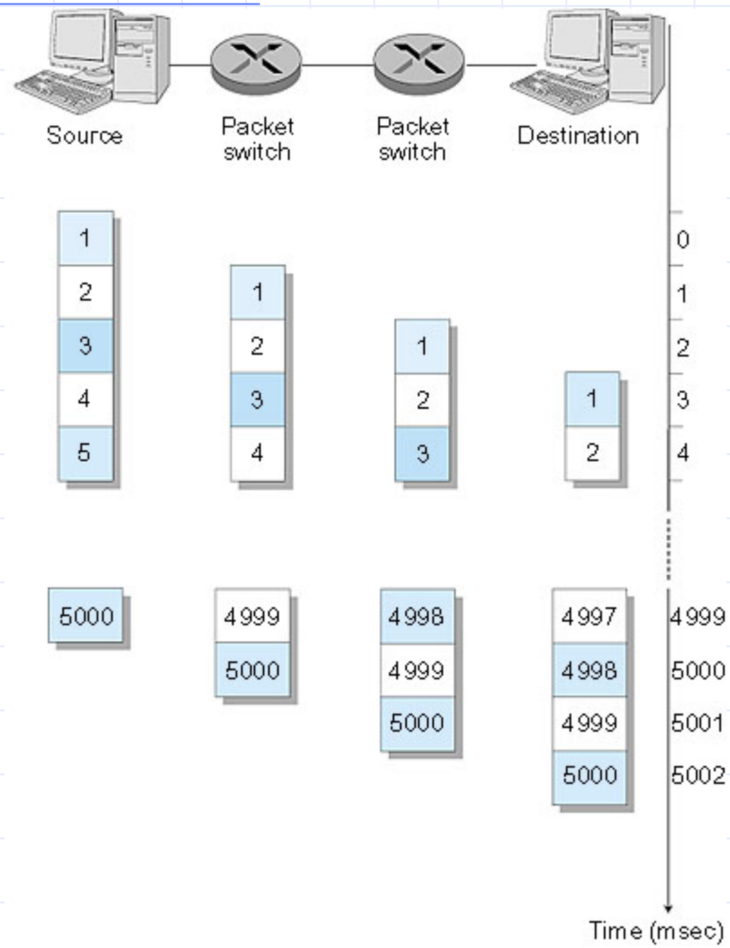


- Takes L/R seconds to transmit (push out) packet of L bits on to link or R bps
- Entire packet must arrive at router before it can be transmitted on next link: store and forward
- delay = 3L/R

Example:

- L = 7.5 Mbits
- R = 1.5 Mbps
- delay = 15 sec

Packet Switching: Message Segmenting



Now break up the message into 5000 packets

- Each packet 1,500 bits
- 1 msec to transmit packet on one link
- pipelining: each link works in parallel
- Delay reduced from 15 sec to 5.002 sec

App-layer protocol defines

- Types of messages exchanged, e.g., request & response messages
- Syntax of message types: what fields in messages & how fields are delineated
- Semantics of the fields, i.e., meaning of information in fields
- Rules for when and how processes send & respond to messages

Public-domain protocols:

- defined in RFCs
- allows for interoperability
- eg, HTTP, SMTP

Proprietary protocols:

- eg, KaZaA

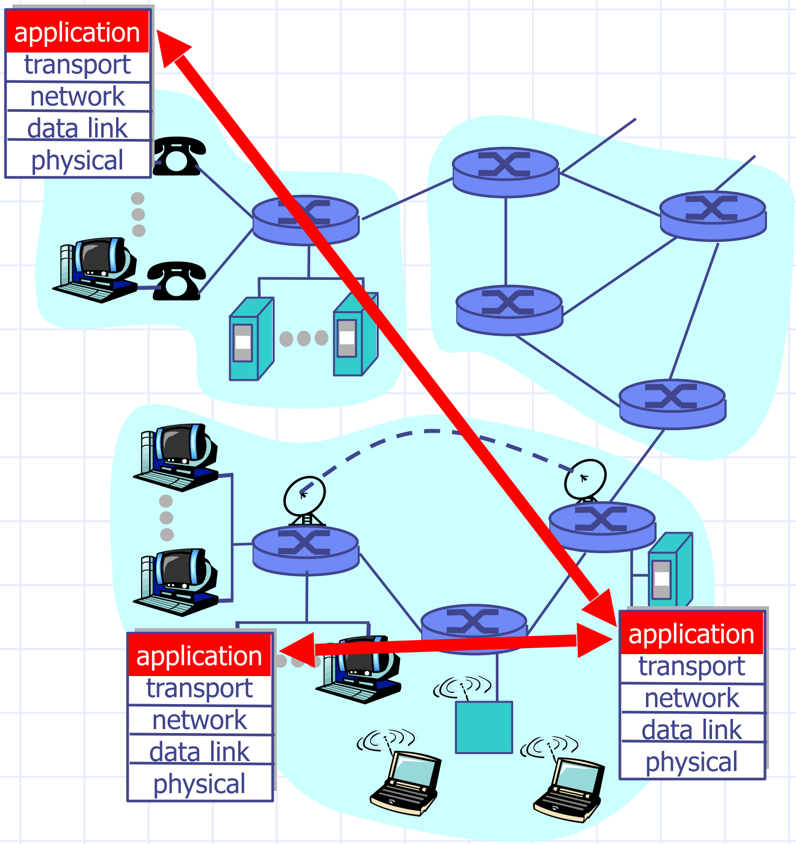
Applications and application-layer protocols

Application: communicating, distributed processes

- e.g., e-mail, Web, P2P file sharing, instant messaging
- running in end systems (hosts)
- exchange messages to implement application

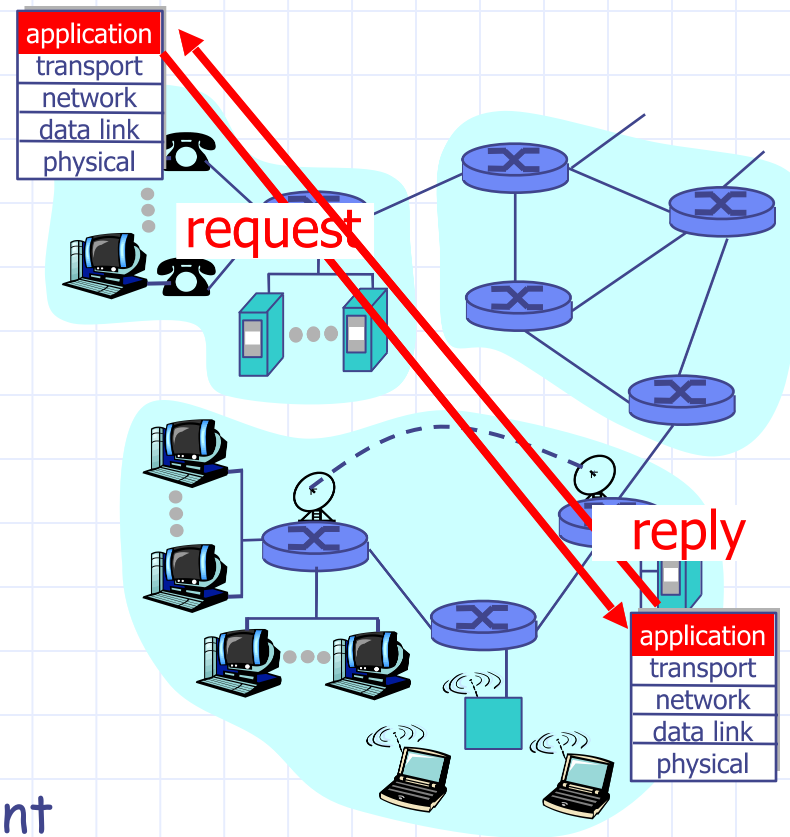
Application-layer protocols

- one "piece" of an app
- define messages exchanged by apps and actions taken
- use communication services provided by lower layer protocols (TCP, UDP)



Client-server paradigm

Typical network app has two pieces: client and server



Client:

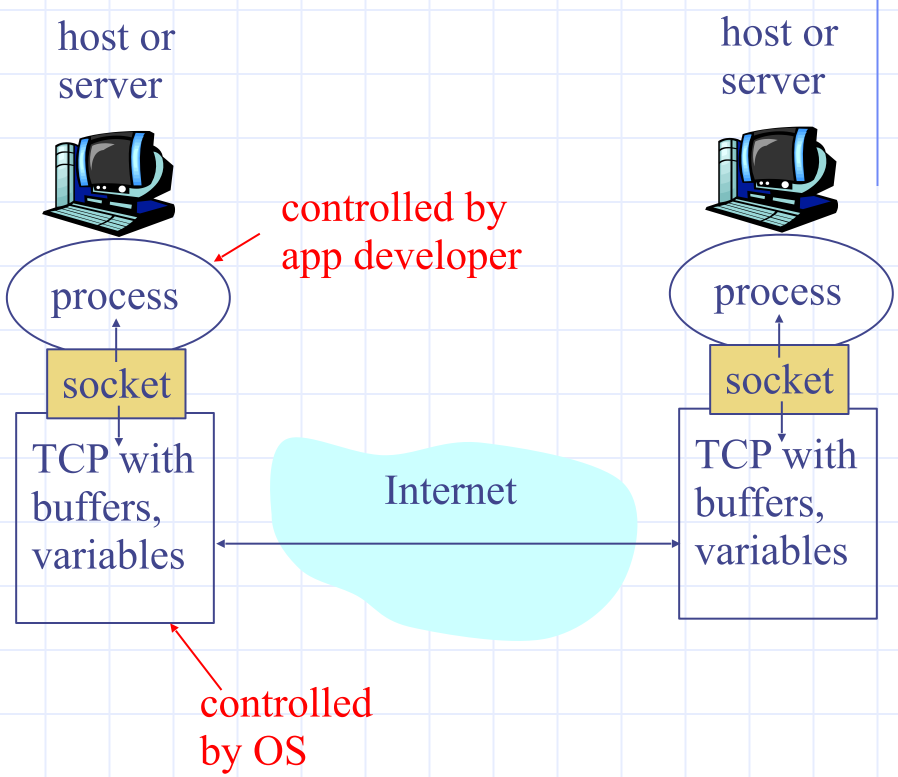
- initiates contact with server ("speaks first")
- typically requests service from server,
- Web: client implemented in browser; e-mail: in mail reader

Server:

- provides requested service to client
- e.g., Web server sends requested Web page, mail server delivers e-mail

Processes communicating across network

- process sends/receives messages to/from its socket
- socket analogous to door
  - sending process shoves message out door
  - sending process assumes transport infrastructure on other side of door which brings message to socket at receiving process



- API: (1) choice of transport protocol; (2) ability to fix a few parameters (lots more on this later)

Addressing processes:

- For a process to receive messages, it must have an identifier
- Every host has a unique 32-bit IP address
- Q: does the IP address of the host on which the process runs suffice for identifying the process?
- Answer: No, many processes can be running on same host
- Identifier includes both the IP address and port numbers associated with the process on the host.
- Example port numbers:
  - HTTP server: 80
  - Mail server: 25
- More on this later

What transport service does an app need?

Data loss

- some apps (e.g., audio) can tolerate some loss
- other apps (e.g., file transfer, telnet,ssh) require 100% reliable data transfer

Timing

- some apps (e.g., Internet telephony, interactive games) require low delay to be "effective"

Bandwidth

- some apps (e.g., multimedia) require minimum amount of bandwidth to be "effective"
- other apps ("elastic apps") make use of whatever bandwidth they get

Transport service requirements of common apps

Application	Data loss	Bandwidth	Time Sensitive
file transfer	no loss	elastic	no
e-mail	no loss	elastic	no
Web documents	no loss	elastic	no
real-time audio/video	loss-tolerant	audio: 5kbps-1Mbps video:10kbps-5Mbps	yes, 100's msec
stored audio/video	loss-tolerant	same as above	yes, few secs
interactive games	loss-tolerant	few kbps up	yes, 100's msec
instant messaging	no loss	elastic	yes and no

Internet transport protocols services

TCP service:

- connection-oriented: setup required between client and server processes
- reliable transport between sending and receiving process
- flow control: sender won't overwhelm receiver
- congestion control: throttle sender when network overloaded
- does not provide: timing, minimum bandwidth guarantees

UDP service:

- unreliable data transfer between sending and receiving process
- does not provide: connection setup, reliability, flow control, congestion control, timing, or bandwidth guarantee

Q: why bother? Why is there a UDP?

Internet apps: application, transport protocols

Application	Application layer protocol	Underlying transport protocol
e-mail	SMTP [RFC 2821]	TCP
remote terminal access	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
file transfer	FTP [RFC 959]	TCP
streaming multimedia	proprietary (e.g. RealNetworks)	TCP or UDP
Internet telephony	proprietary (e.g., Dialpad)	typically UDP

Network programming

- Programmer does not need to understand the hardware part of network technologies.
- Network facilities accessed through an *Application Program Interface* - API

Socket programming

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

Socket API

- introduced in BSD4.1 UNIX, 1981
- explicitly created, used, released by apps
- client/server paradigm
- two types of transport service via socket API:
  - unreliable datagram
  - reliable, byte stream-oriented

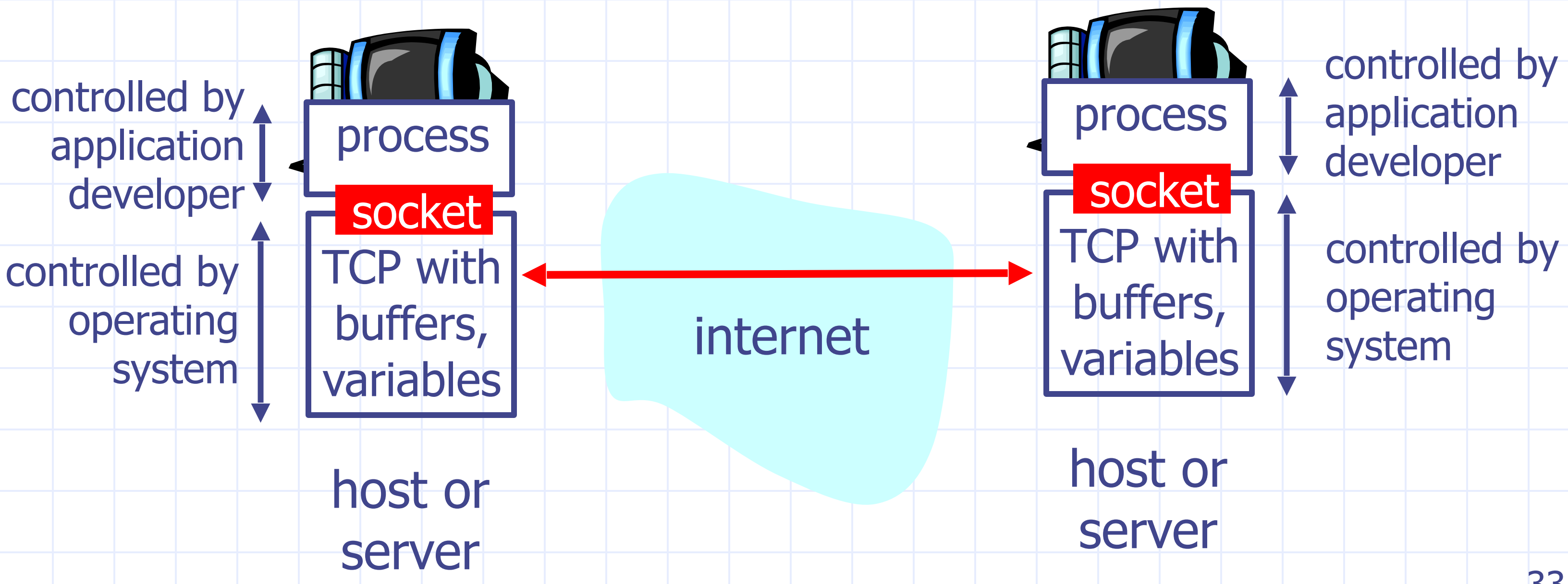
socket

a host-local, application-created, OS-controlled interface (a "door") into which application process can both send and receive messages to/from another application process

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



Socket programming with TCP

Client must contact server

- server process must first be running
- server must have created socket (door) 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 (more in Chap 3)

application viewpoint

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

Socket programming with TCP

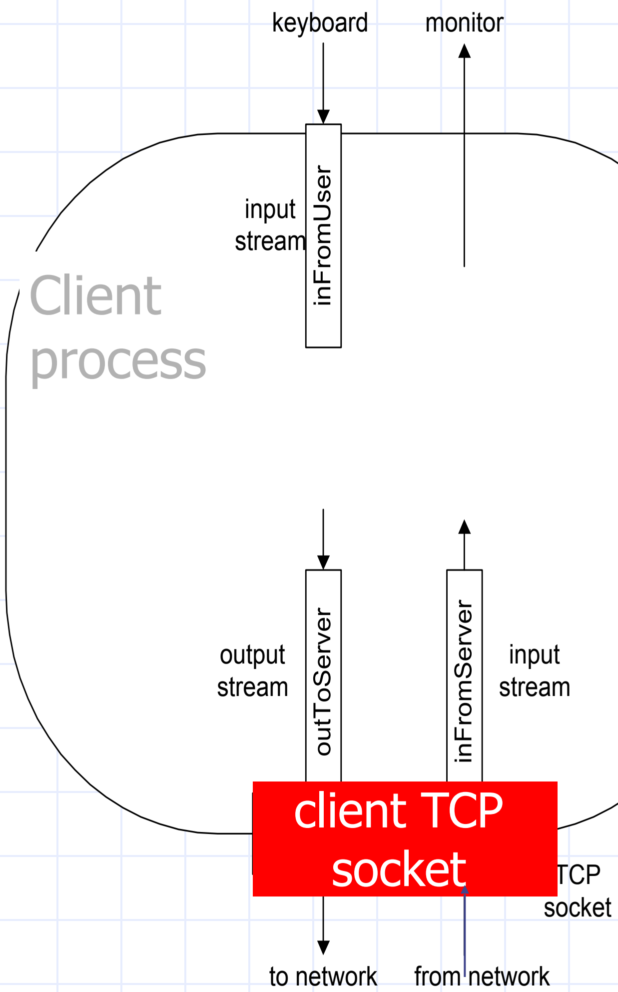
Example client-server app:

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

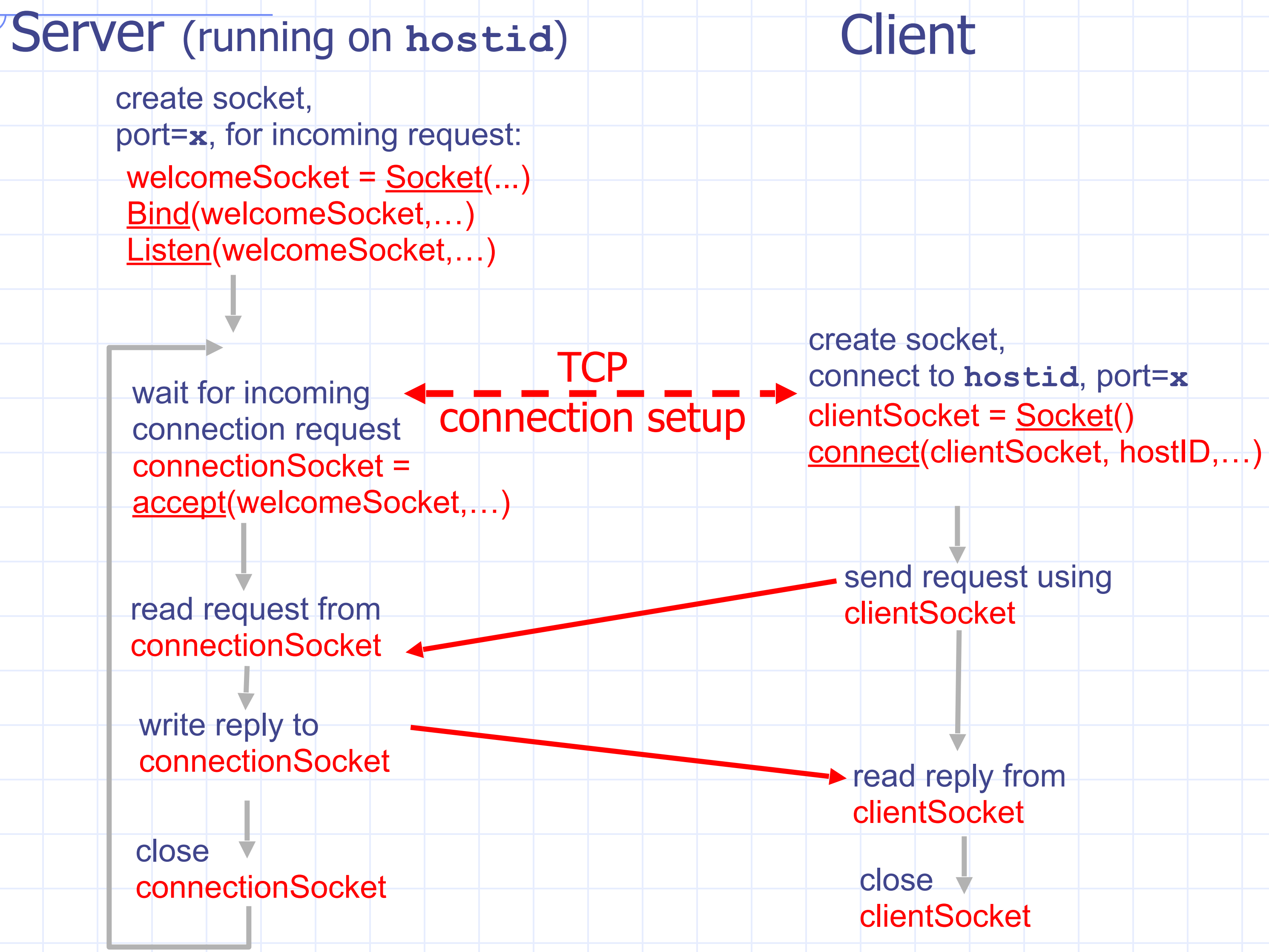
2) server reads line from socket

3) server sends back to client (echo) the same data

4) client reads from socket and prints line (**inFromServer** stream)



Client/server socket interaction: TCP



Connection oriented-API

- The BSD socket library
  - Socket
  - Bind
  - Listen, Accept
  - Connect
  - Read, Write, Recv, Send
  - Close, Shutdown
- Where do we get info on these ?
  - man, msdn

Socket Example

Server.c

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <stdio.h>
#include <unistd.h> /* close */

#define SERVER_PORT 1500
```

```
int main (int argc, char *argv[]) {
  int sd, newSd, cliLen;
  struct sockaddr_in cliAddr, servAddr;
  char line[MAX_MSG];
  int len;

  sd = socket(AF_INET, SOCK_STREAM, 0);
   if(sd<0) {
    perror("cannot open socket ");
    return ERROR;
  }

  /* bind server port */
  servAddr.sin_family = AF_INET;
  servAddr.sin_addr.s_addr = htonl(INADDR_ANY);
  servAddr.sin_port = htons(SERVER_PORT);
```

```
if (bind(sd, (struct sockaddr *)
    &servAddr, sizeof(servAddr))<0) {
  perror("cannot bind port ");
  return ERROR;
 }
 listen(sd,5);
 while(1) {
   printf("%s: waiting for data on port
    TCP %u\n",argv[0],SERVER_PORT);

  cliLen = sizeof(cliAddr);
  newSd = accept(sd, (struct sockaddr
    *) &cliAddr, &cliLen);
  if(newSd<0) {
    perror("cannot accept connection ");
    return ERROR;
   } // end if
```

```
  /* init line */
   memset(line,0,MAX_MSG);

   /* receive segments */
  if ( (len=read(newSd,line,MAX_MSG))> 0) {
    printf("%s: received from %s:TCP%d :
    %s\n", argv[0],
         inet_ntoa(cliAddr.sin_addr),
         ntohs(cliAddr.sin_port), line);

     write(newSd,line,len);
   } else
     printf("Error receiving data\n");
   close(newSd);
  } //end if
} //end while
```

CLIENT.C

```
include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <stdio.h>
#include <unistd.h> /* close */

#define SERVER_PORT 1500
#define MAX_MSG 100

int main (int argc, char *argv[]) {

  int sd, rc, i;
  struct sockaddr_in servAddr;
  struct hostent *h;
  char msg[300];

 if(argc < 3) {
  printf("usage: %s <server> <text>\n",argv[0]);
  exit(1);
}

h = gethostbyname(argv[1]);
if (h==NULL) {
  printf("%s: unknown host
   '%s'\n",argv[0],argv[1]);
  exit(1);
}

servAddr.sin_family = h->h_addrtype;
memcpy((char *) &servAddr.sin_addr.s_addr,
    h->h_addr_list[0], h->h_length);
servAddr.sin_port = htons(SERVER_PORT);
```

```
/* create socket */
  sd = socket(AF_INET, SOCK_STREAM, 0);
  if(sd<0) {
    perror("cannot open socket ");
    exit(1);
  }
  /* connect to server */
  rc = connect(sd, (struct sockaddr *) &servAddr, sizeof(servAddr));
  if(rc<0) {
    perror("cannot connect ");
    exit(1);
  }
  write(rc, argv[1],strlen(argv[1]+1) );
  read(rc, msg, 300);
  printf(”Received back: %s\n”, msg);
  close(rc);
  return 0;
}
```