Internetworking

Randal E. Bryant

CS347 Lecture 25

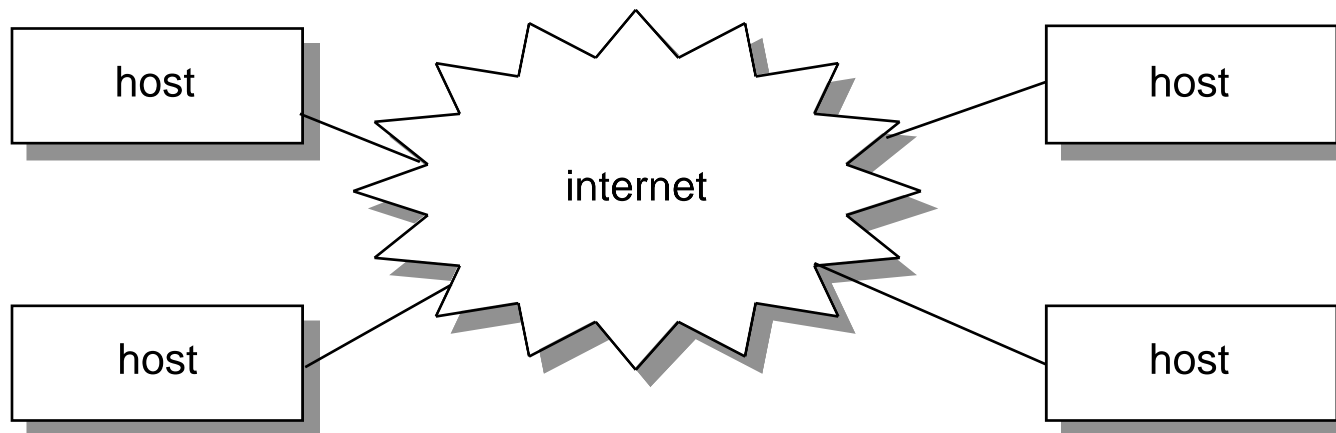
April 21, 1998

Topics

- Internetworks
- Internet protocol stack
- ARP
- TCP
- IP
- Sockets interface

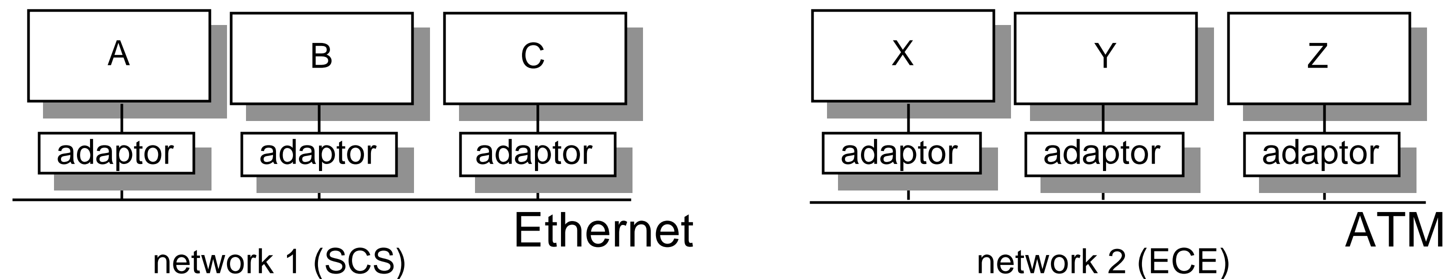
Internetworks

Def: An *internetwork* (internet for short) is an interconnected collection of networks



Building an internet

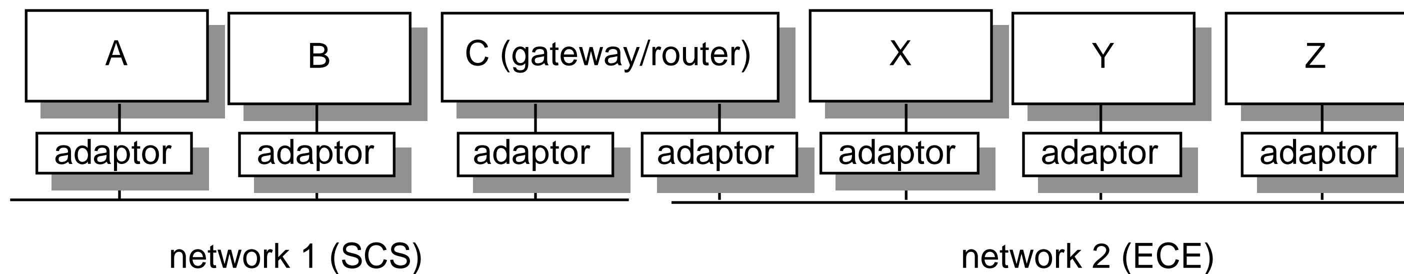
We start with two separate, unconnected computer networks (subnets), which are at different locations, and possibly built by different vendors.



Question: How to present the illusion of one network?

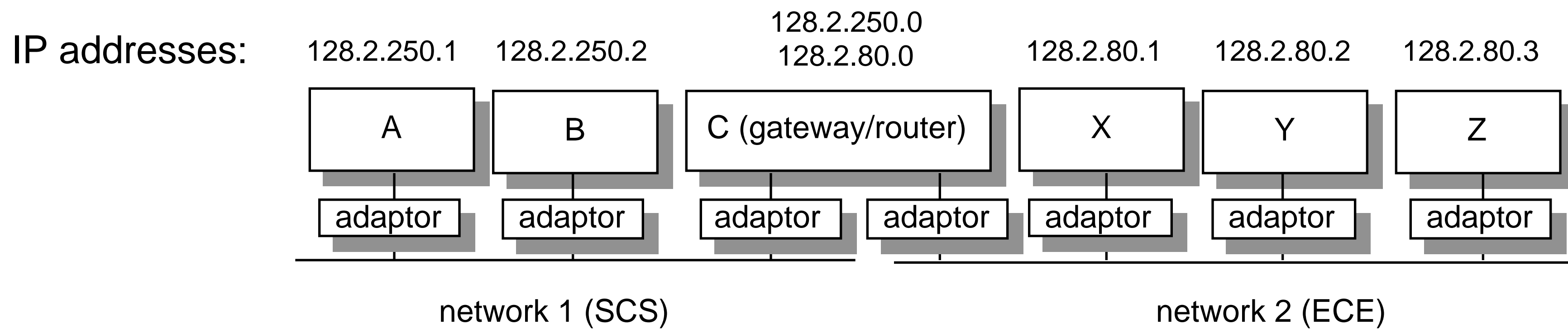
Building an internet (cont)

Next we connect one of the computers
(in this case computer C) to each of the networks.



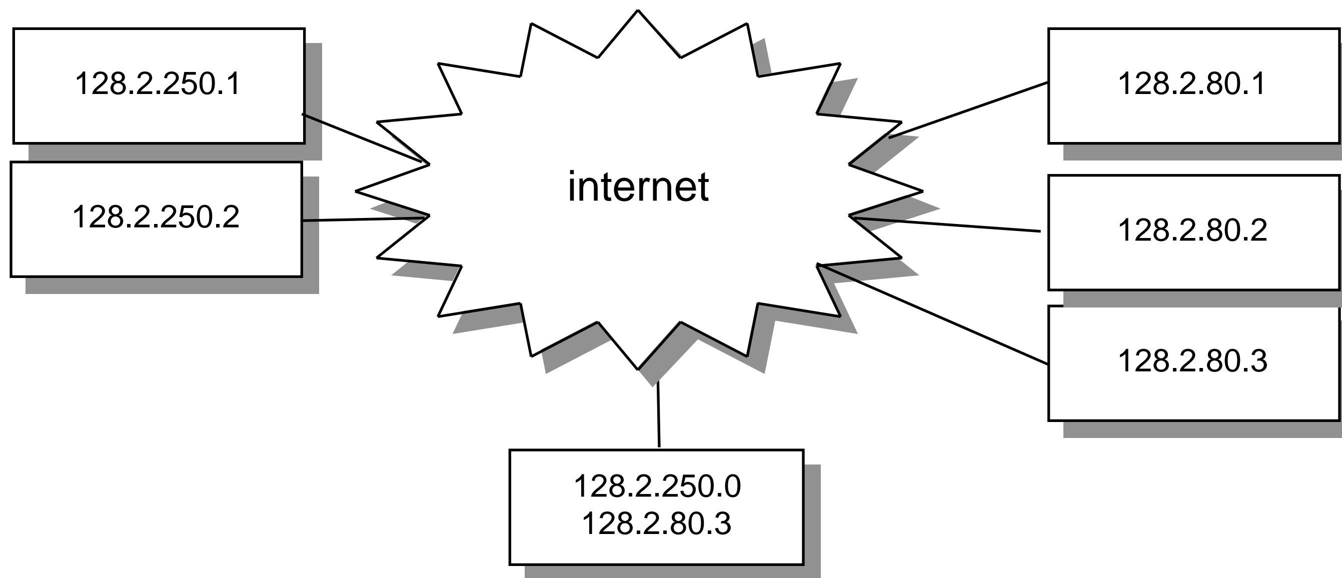
Building an internet (cont)

Finally, we run a software implementation of the Internet Protocol (IP) on each computer. IP provides a global name space for the hosts, routing messages between network1 and network 2 if necessary.

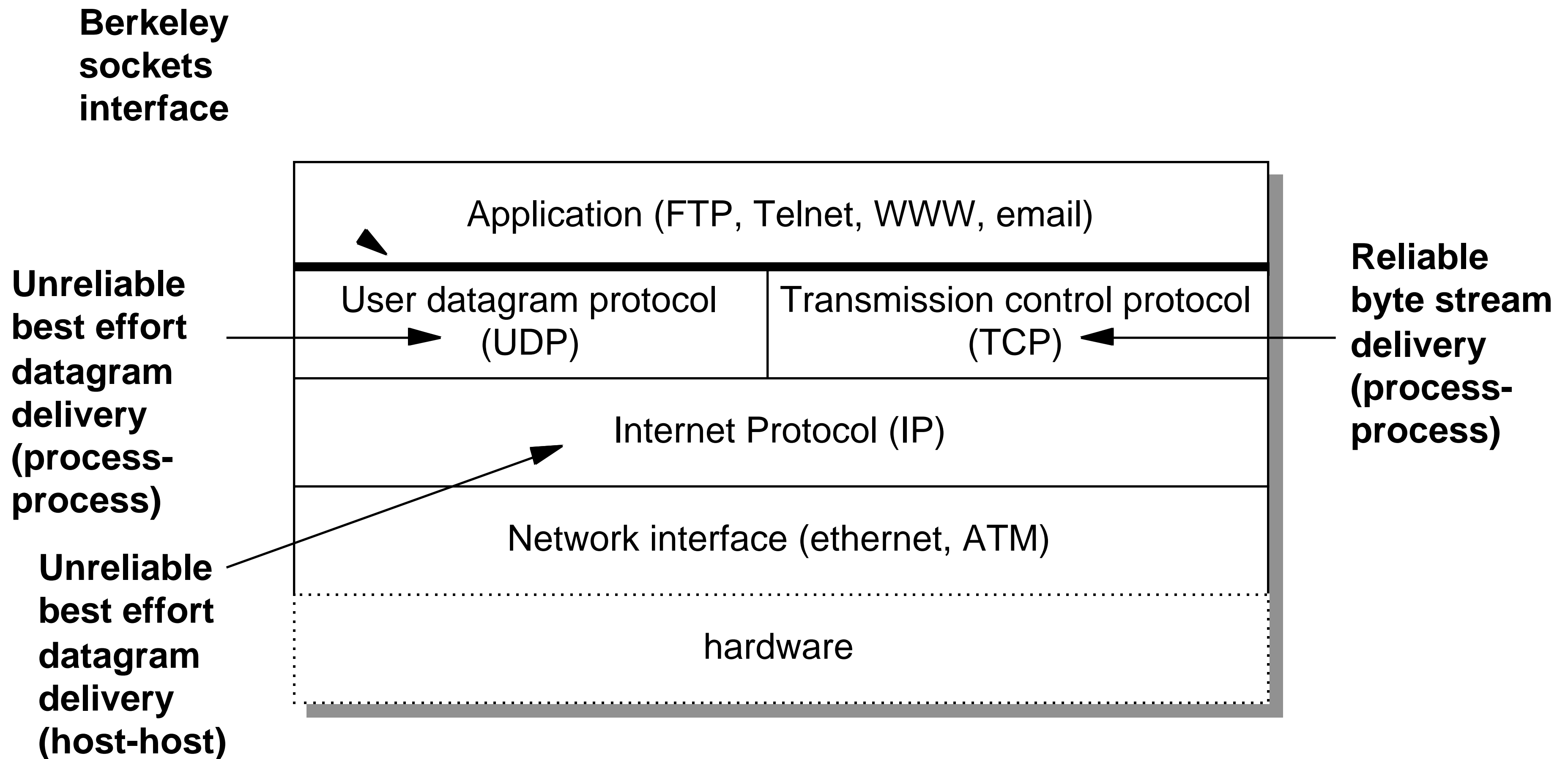


Building an internet (cont)

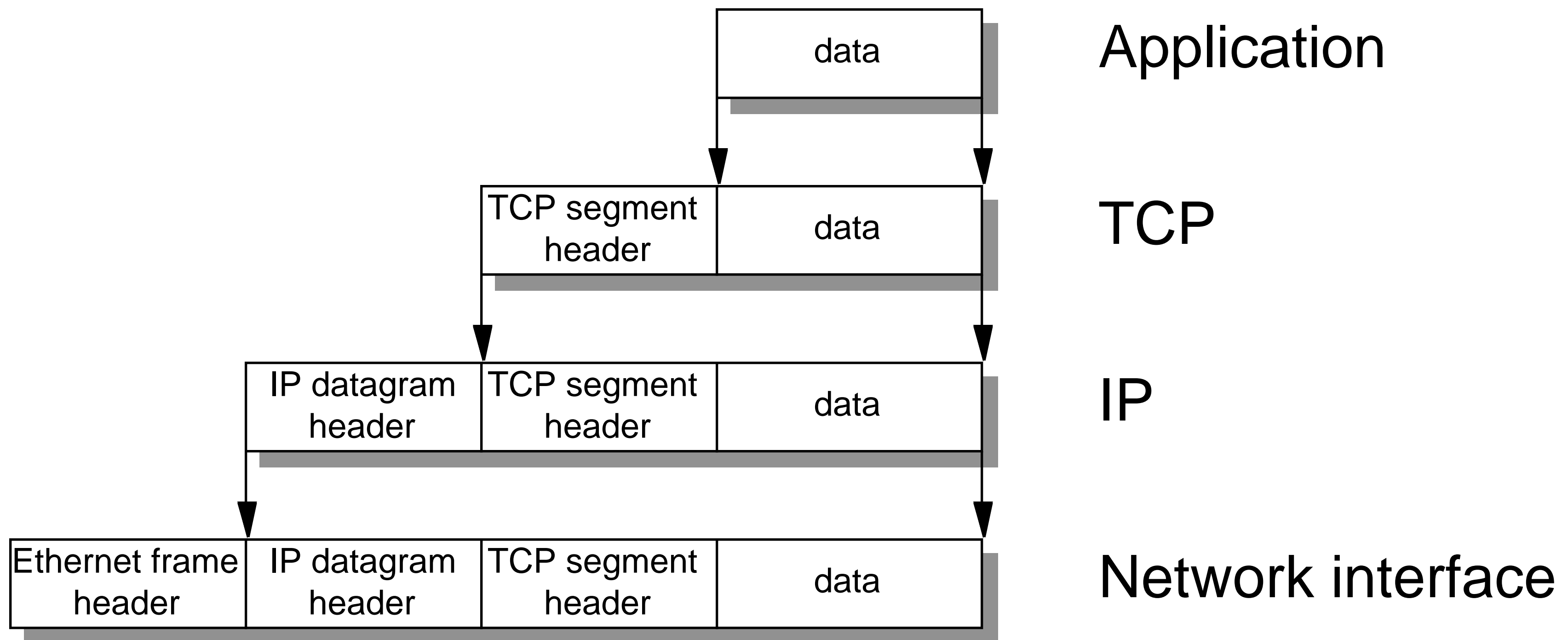
At this point we have an internet consisting of 6 computers built from 2 original networks. Each computer on our internet can communicate with any other computer. IP provides the illusion that there is just one network.



Internet protocol stack



Encapsulation



Internet addresses

Each host x has a physical address P(x) and a unique IP address I(x).

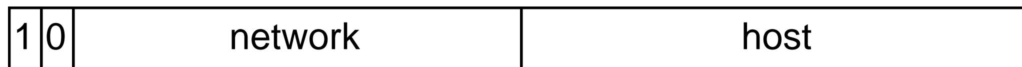
IP addresses are hierarchical.

- **address contains hint about location**

3 classes of subnets:

0 1 2 3 4 8 16 24 31

Class A (lots of hosts/network)



Class B

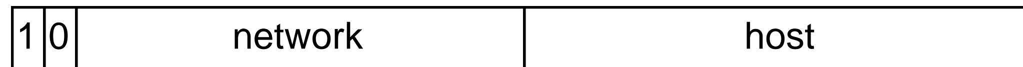
Class C (few hosts/network)

Example Internet addresses

Host	IP Number	Class	Network
`cs.cmu.edu`	`128.2.222.173`	`B`	`0x0002`
`cmu.edu`	`128.2.35.186`	`B`	`0x0002`
`cs.stanford.edu`	`171.64.64.64`	`B`	`0x2640`
`att.com`	`192.128.133.151`	`C`	`0x008085`

0 1 2 3 4 8 16 24 31

Class A (lots of hosts/network)



Class B

Class C (few hosts/network)

ARP: Address resolution protocol

Initially:

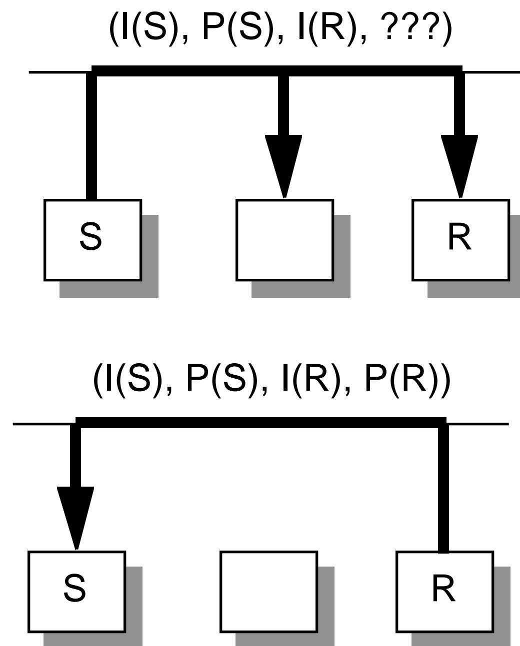
- Hosts S and R on the same subnet with IP addresses I(S) and I(R) and physical addresses P(S) and P(R).

Problem:

- Given I(R), host S wants to discover P(R).

Solution:

- Host S broadcasts triple (I(S), P(S), I(R),???) on subnet.
- Host R (and only host R) responds with tuple (I(S), P(S), I(R), P(R))
- Both sender and receiver maintain a software cache of IP to physical mappings.
- Time out old entries



IP: Internet protocol

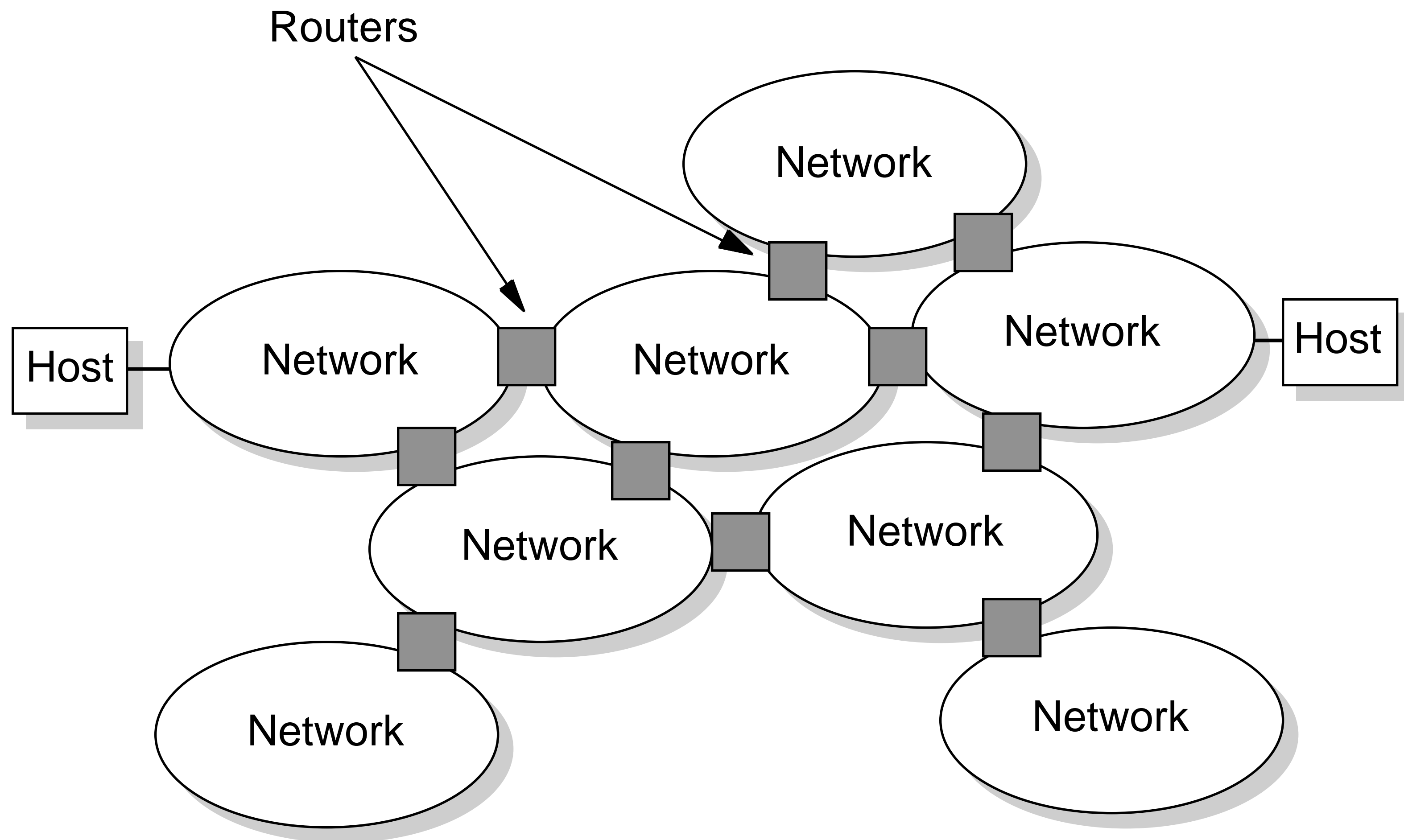
Unreliable, best-effort datagram transfer

- Addressing/routing.
- Fragmentation.
- Time to live.

VER/HL	TOS	Length
Datagram ID		Flags/Frag Off.
TTL	Prot.	HDR. Checksum
Source IP address		
Destination IP address		
Options..		

VER	IP version
HL	Header length (in 32-bit words)
TOS	Type of service (unused)
Length	Datagram length (max 64K bytes)
ID	Unique datagram identifier
Flags/Frag	Control flags/fragment offset
TTL	Time to Live
Prot	Higher level protocol (e.g., TCP, UDP)

Routing problem



Internet routing

Routing decision: translate IP address of final destination into physical address of next hop.

(1) Extract final destination host IP address from datagram.

(2) Look up final IP address in routing table.

(3) Returns IP address of next hop.

(4) Use ARP to discover corresponding physical address of next hop.

(5) Forward datagram to next hop.

Fragmentation

Different networks have a different maximum transfer unit (MTU).

A problem can occur if packet is routed onto network with a smaller MTU.

- e.g. FDDI (4,500B) onto Ethernet (1,500B)

Solution: break packet into smaller fragments.

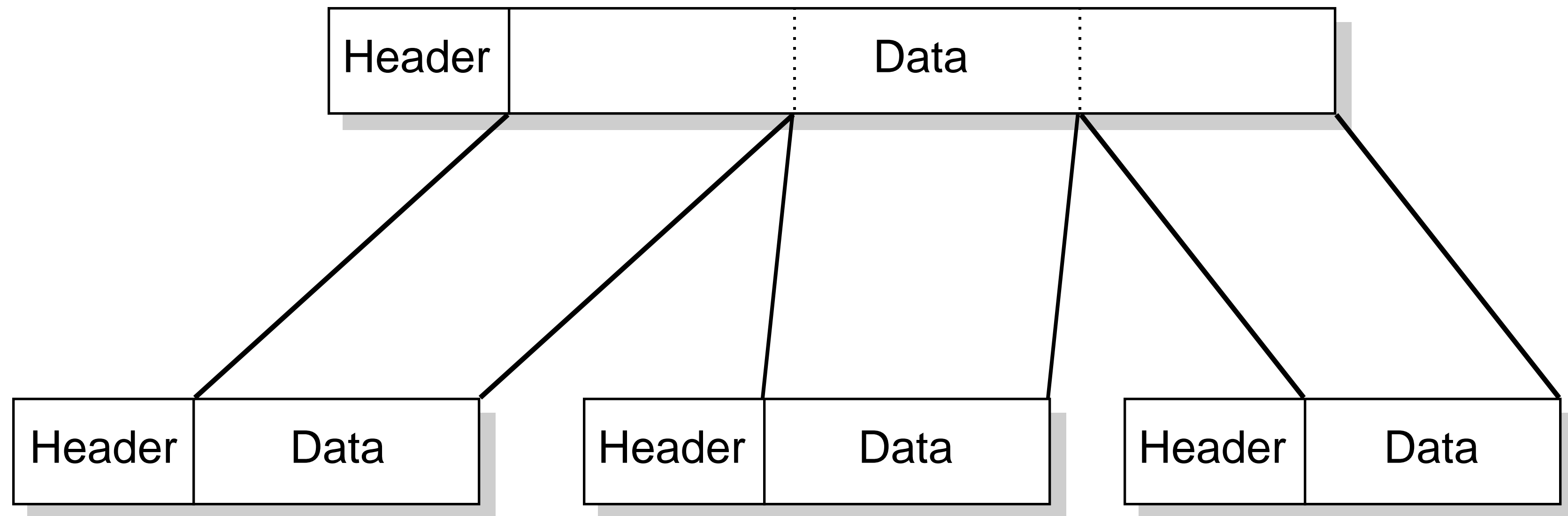
- each fragment has identifier and sequence number

Destination reassembles packet before handing it up in the stack.

- alternative would be to reassemble when entering network with larger MTU

Sender can disable fragmentation using flag.

Fragmentation example



Time to live

Goal: drop packets that are stuck in infinite loop in the network.

IP Solution: Decrement TTL field in each hop and drop packet if it reaches 0.

- **field initialized by sender**
- **hop that drops packet notifies sender**

UDP: User datagram protocol

Uses IP to provide unreliable best effort datagram delivery.

"Thin layer" over IP

- **data corruption protection.**

Source Port	Dest. Port
Length	D. Checksum

UDP data corruption protection

Optional end-end checksum.

- protects against any data corruption errors between source and destination (links, switches/routers, bus)
- does not protect against packet loss, duplication or reordering

Checksum calculation:

- one complement add
- includes pay load plus part of the IP and UDP header
- layering?

TCP: Transmission control protocol

Uses IP to provide reliable byte stream delivery.

- **stream orientation**
  - sender transfers stream of bytes; receiver gets identical stream
- **virtual circuit connection**
  - stream transfer analogous to placing phone call
  - sender initiates connection which must be accepted by receiver.
- **buffered data transfer**
  - protocol software free to use arbitrary size transfer units
- **unstructured streams**
  - stream is just a sequence of bytes, just like Unix files
- **full duplex**
  - concurrent transfers in both directions along a connection

TCP functions

Connections

Sequence numbers

Sliding window protocol

Reliability and congestion control.

Source Port	Dest. Port
Sequence Number	
Acknowledgment	
Hlen/Flags	Window
D. Checksum	Urgent Pointer
Options..	

Connections

Connection is fundamental TCP communication abstraction.

- data sent along a connection arrives in order
- implies allocation of resources (buffers) on hosts

The endpoint of a connection is a pair of integers:

- (IP address, port)

A connection is defined by a pair of endpoints:

- ((128.2.254.139, 1184), (128.10.2.3, 53))

Sequence space

Each stream split into a sequence of segments which are encapsulated in IP datagrams.

Each byte in the byte stream is numbered.

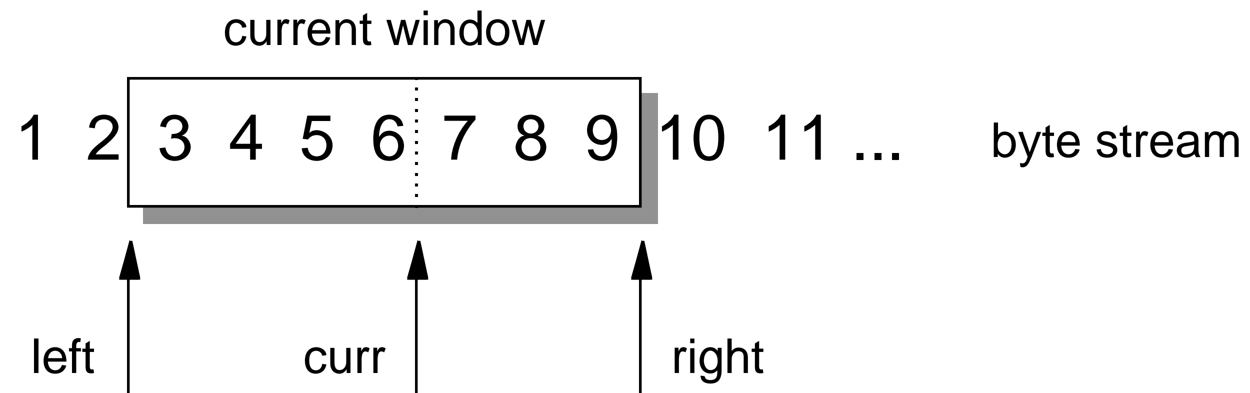
- 32 bit value
- wraps around
- initial values selected at runtime

Each segment has a sequence number.

- indicates the sequence number of its first byte
- Detects lost, duplicate or out of order segments

Sliding window protocol (sender)

Sender maintains a "window" of unacknowledged bytes that it is allowed to send, and a pointer to the last byte it sent:



Bytes through 2 have been sent and acknowledged (and thus can be discarded)
Bytes 3 -- 6 have been sent but not acknowledged (and thus must be buffered)
Bytes 7 -- 9 have been not been sent but will be sent without delay.
Bytes 10 and higher cannot be sent until the right edge of window moves.

Sliding window protocol (receiver)

Receiver acknowledges receipt of a segment with two pieces of information:

- **ACK: the sequence number of the next byte in the contiguous stream it has already received**
- **WIN: amount of available buffer space.**

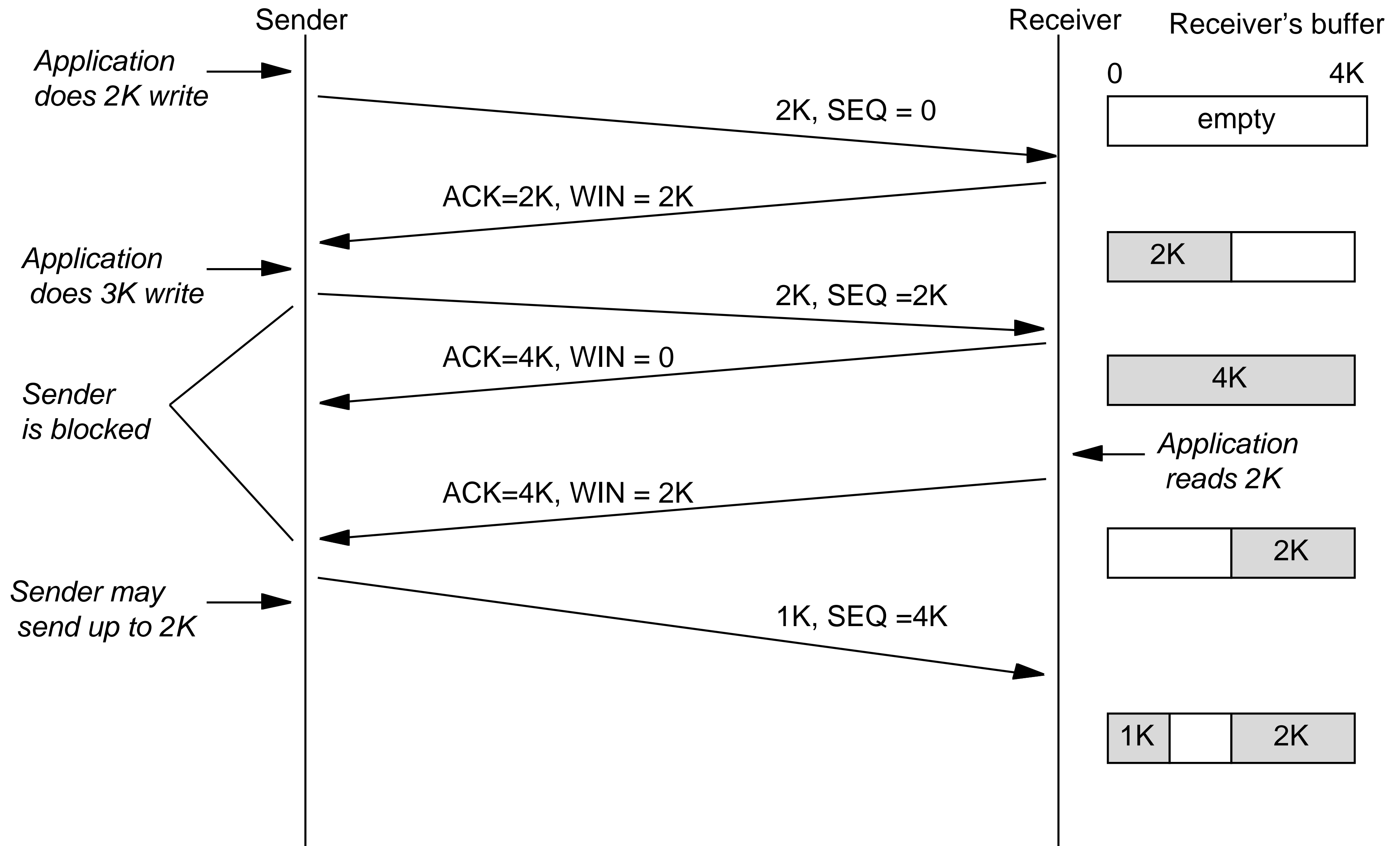
ACK indicates that data was received correctly.

- **sender can increment left edge of window**
- **sender can delete data to the left of the window.**

WIN indicates that more buffer space was freed up.

- **sender can increment the right edge of its window**
- **sender can transmit more data.**

Sliding window protocol (example)



Reliability and congestion control

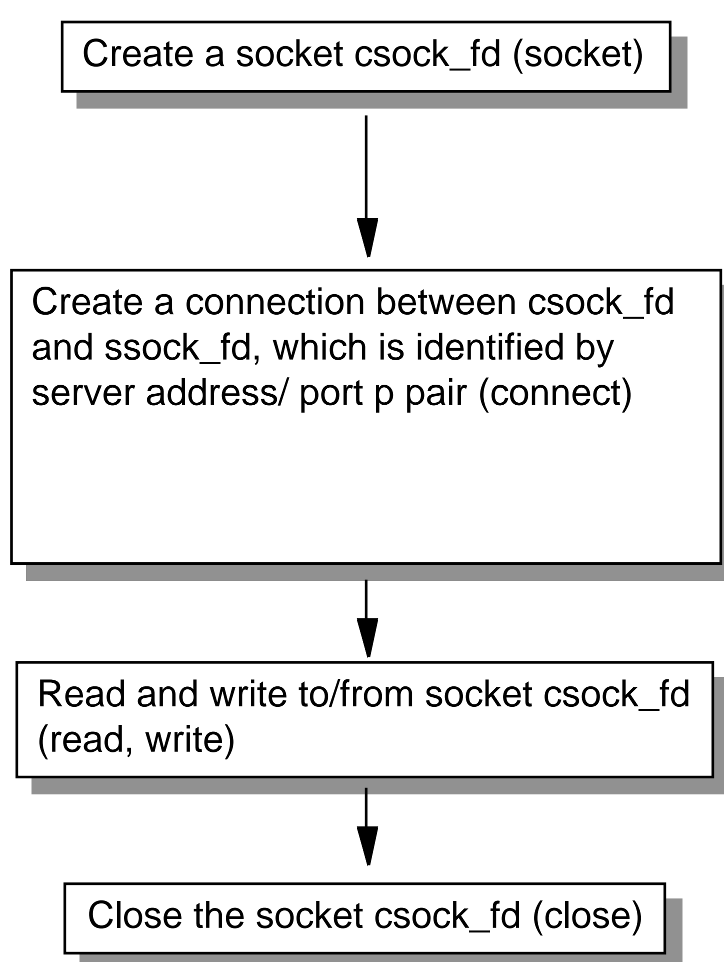
Reliability:

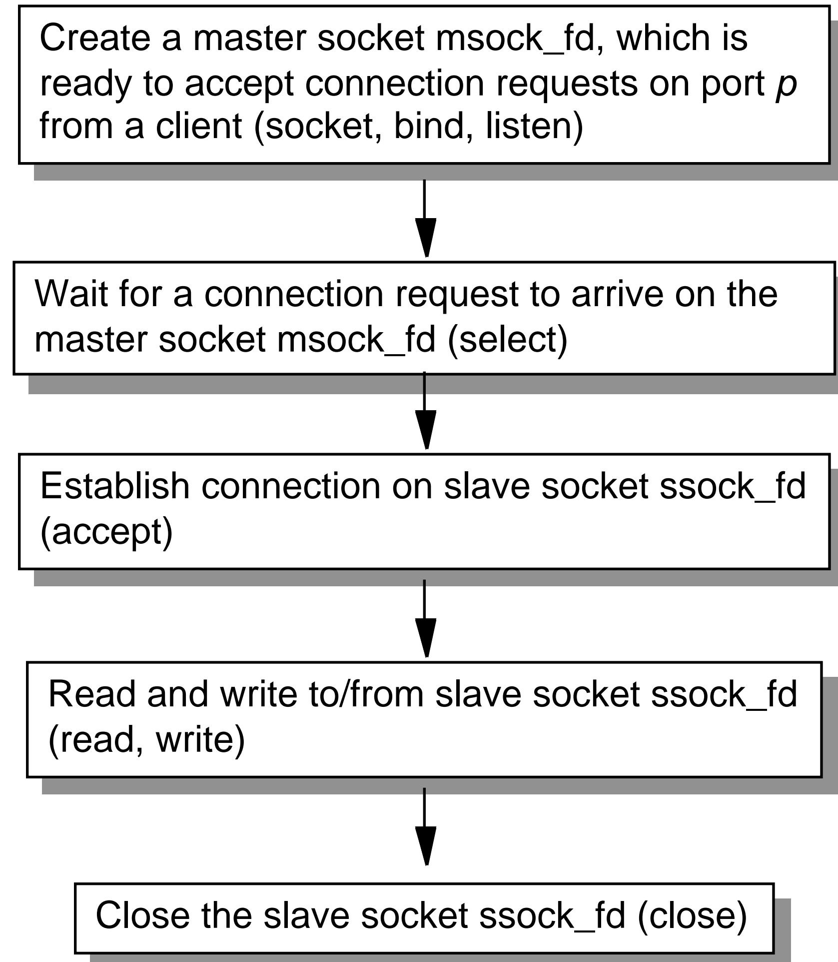
- **sender**
  - saves segments inside its window
  - uses timeouts and sequence numbers in ACKS to detect lost segments.
  - retransmit seqments it thinks are lost
- **receiver**
  - uses sequence numbers to assemble segments in order
  - also to detect duplicate segments (how might this happen?)

Congestion control

- **sender maintains separate separate congestion window**
- **uses smaller of the two windows**
- **users "slow start" algorithm to adaptively set congestion window size.**

The socket interface





Example client code

```
/* the client writes a sequence of messages to a server */
for (k=0; k<msgs; k++) {
    /* setup a tcp connection with the server */
    sockfd = connectsock(host, PORT, "tcp");

    /* write  the data buffer to the socket */
    cnt = sendsock(sockfd, msg.buf, msglen);
    if (cnt < msglen)
      errexit("sendsock failed\n");

    /* take down the connection */
    close(sockfd);
  }
```

Example server code

```
/* create master socket ready to accept connections
   from client */
  master_sockfd = passivesock(PORT, "tcp");

  /*
   * the server loops forever, waiting until a conn request
   * is pending, opening the connection, reading msg,
   * and closing connection
  */
  while (1) {

    /* loop until a connection request is pending
       on master socket */
    ready = 0;
    while (ready < 1) {
      ready = readysock(master_sockfd);
      if (ready == 0)  sleep(1);
    }
```

```
 /* establish the pending connection */
    arch_starttimer(&st);
    slave_sockfd = acceptsock(master_sockfd);
    if (slave_sockfd < 0)
      errexit("accept failed\n");


/* read the data into a buffer */
    cnt = recvsock(slave_sockfd, msg.buf,
                                 MAX_BUF);
    if (cnt < 0)
      errexit("recvsock failed\n");

    /* take down the connection */
    close(slave_sockfd);

 } /* end while(1) */
```

Key Themes in (Inter)Networking

Protocol Layering

- Way to structure complex system
- Handle different concerns at different layers

Must Cope with Imperfect Environment

- Packets get corrupted & lost

No One has Complete Routing Table

- Too many hosts
- Hosts continually being added and removed
- In the future, they will start moving around