15-213

Internetworking II

November 24, 1998

Topics

- IP datagram forwarding, ARP
- IPv6
- End-to-end protocols: UDP, TCP
- End-to-end data: presentation formatting
- network programming: sockets interface

IP Datagram Forwarding

***Forwarding*: the process of copying an input packet from an input port to an output port.**

***Routing:* the process of building the tables on each router that allow the correct output port to be determined (beyond our scope)**

Key points

- **Every IP datagram contains the IP address of the destination.**
- **Network part of IP address uniquely identifies a single physical network.**
- **All hosts and routers with same network part are on the same physical network.**
- **Every physical network on the Internet has a router connected to at least one other physical network.**

IP Forwarding Algorithm

Algorithm for host *I(src)*, with 1 interface, sending to host *I(dst):*

if network part of *I(src)* = network part of *I(dst)*
deliver packet directly to *P(dst)* /* I->P mapping via ARP */
else
deliver packet to *P(default router)*

Algorithm for router receiving *packet for I(dst):*

for each interface *k*
if network part of *I(k)* = network part of *I(dst)*
deliver packet directly to *P(dst)* using interface *k*
else
if network part of *I(dst)* is in forwarding table
deliver packet to *P(NextHop router)*
else
deliver packet to *P(default router)*

Forwarding table consists of *(NetworkNum*, *NextHop)* pairs

IP Forwarding Algorithm

Algorithm for host S sending to host D:

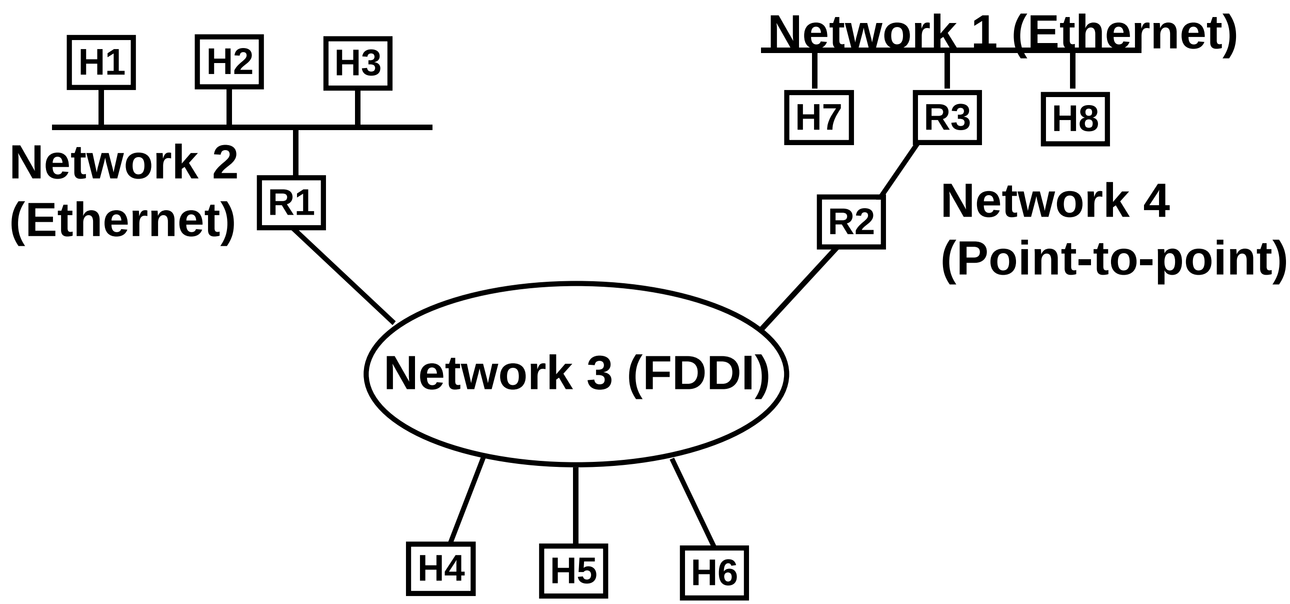
```
if (NetworkNum(S) == NetworkNum(D)) {
    deliver packet directly to D    /* IP->physical  mapping via ARP */
else
    deliver packet to default router
```

Algorithm for router receiving packet for host D

```
NextHop = lookup(NetworkNum(D));
if (NextHop is an interface)
        deliver packet directly to D using interface NextHop
else
    if (NextHop != <undefined>)
        deliver packet to NextHop (a router)
    else
        deliver packet to default router
```

Forwarding table consists of (NetworkNum, NextHop) pairs

IP Forwarding example



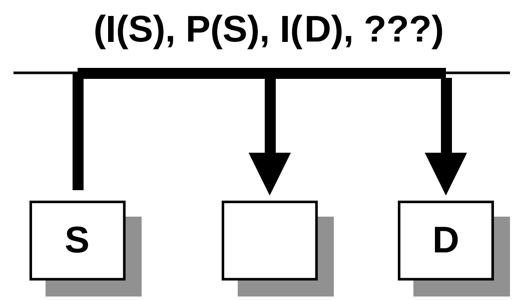
Router R2 forwarding table

NetworkNum	NextHop
1	R3
2	R1
3	Interface 1
4	Interface 0

ARP: Address resolution protocol

Initially:

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

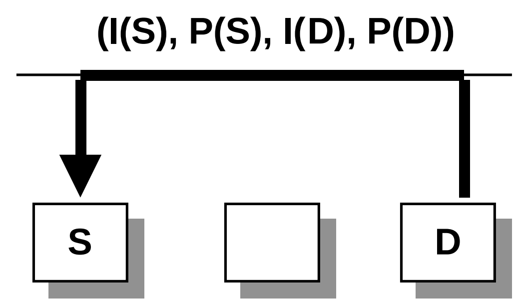


Problem:

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

Solution:

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



Subnetting

Problem: **IP addressing scheme makes inefficient use of addresses**

Partial solution: **subnetting**

- **physical network part of address identifies a "virtual" physical network to the external world.**
- **use some of the high order "host" bits to identify local physical networks within the "virtual" physical network.**

network number	host number	Class B address
		&
11111111 11111111 11111111	00000000	Subnet mask (255.255.255.0)
		=
xxxxxxxx xxxxxxxx xxxxxxxx	00000000	Subnet number

- All hosts on same physical network have same subnet number.
- There is exactly one subnet mask per subnet.
- All hosts on subnet configured with this mask (ifconfig)

IP forwarding with subnetting

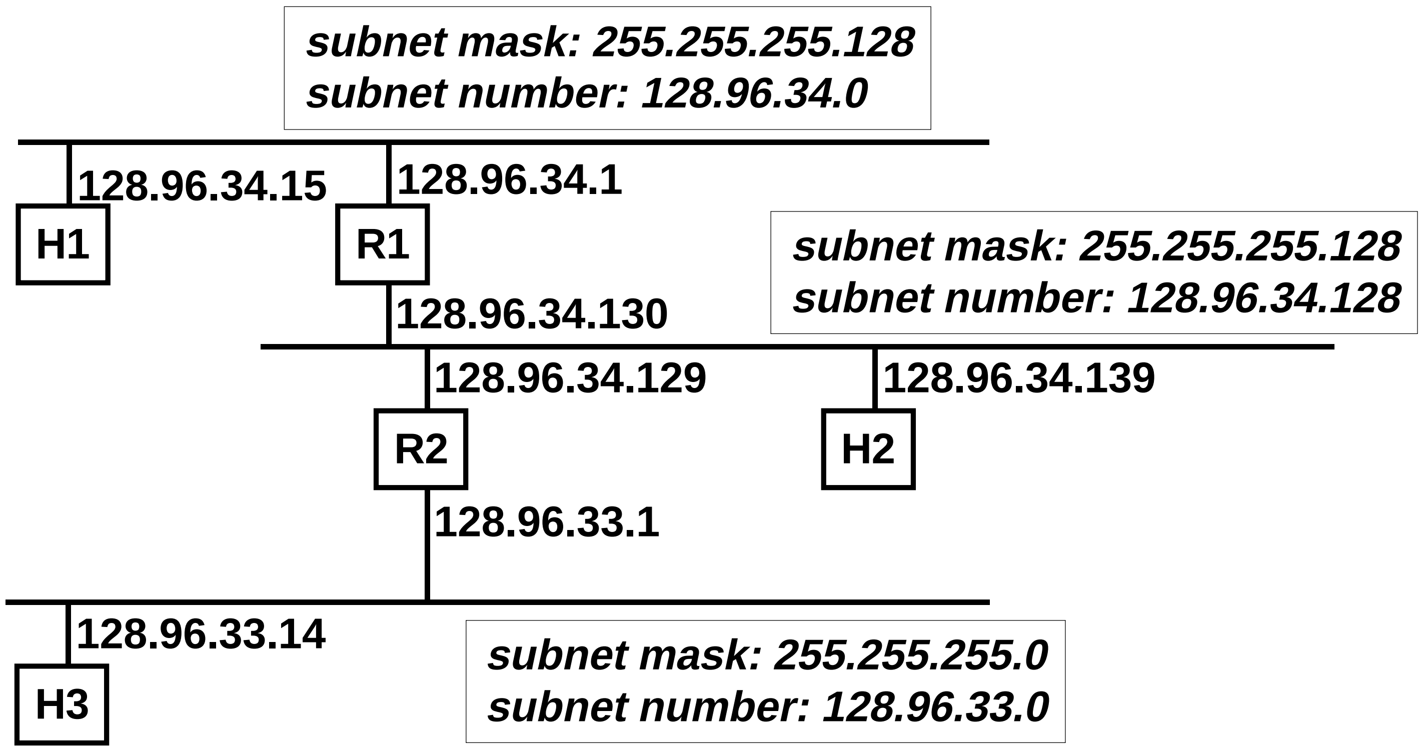
Algorithm on a host:

```
D1 = SubnetMask & destination IP address
if (D1 == MySubnetNum)
  deliver datagram directly to destination
else
  deliver datagram to default router
```

Algorithm on a router:

```
for each forwarding table entry <SubnetNum,SubnetMask,NextHop>
  D1 = SubnetMask & destination IP address
  if (D1 == SubnetNum)
    if (NextHop is an interface)
      deliver datagram directly to destination
    else
      deliver datagram to NextHop (a router)
```

Subnetting example



forwarding table for R1

SubnetNum	SubnetMask	NextHop
128.96.34.0	255.255.255.128	interface 0
128.96.34.128	255.255.255.128	interface 1
129.96.33.0	255.255.255.0	R2

IPv6

Also called Next Generation IP and IPng

Extends address space from 32 bits to 128 bits

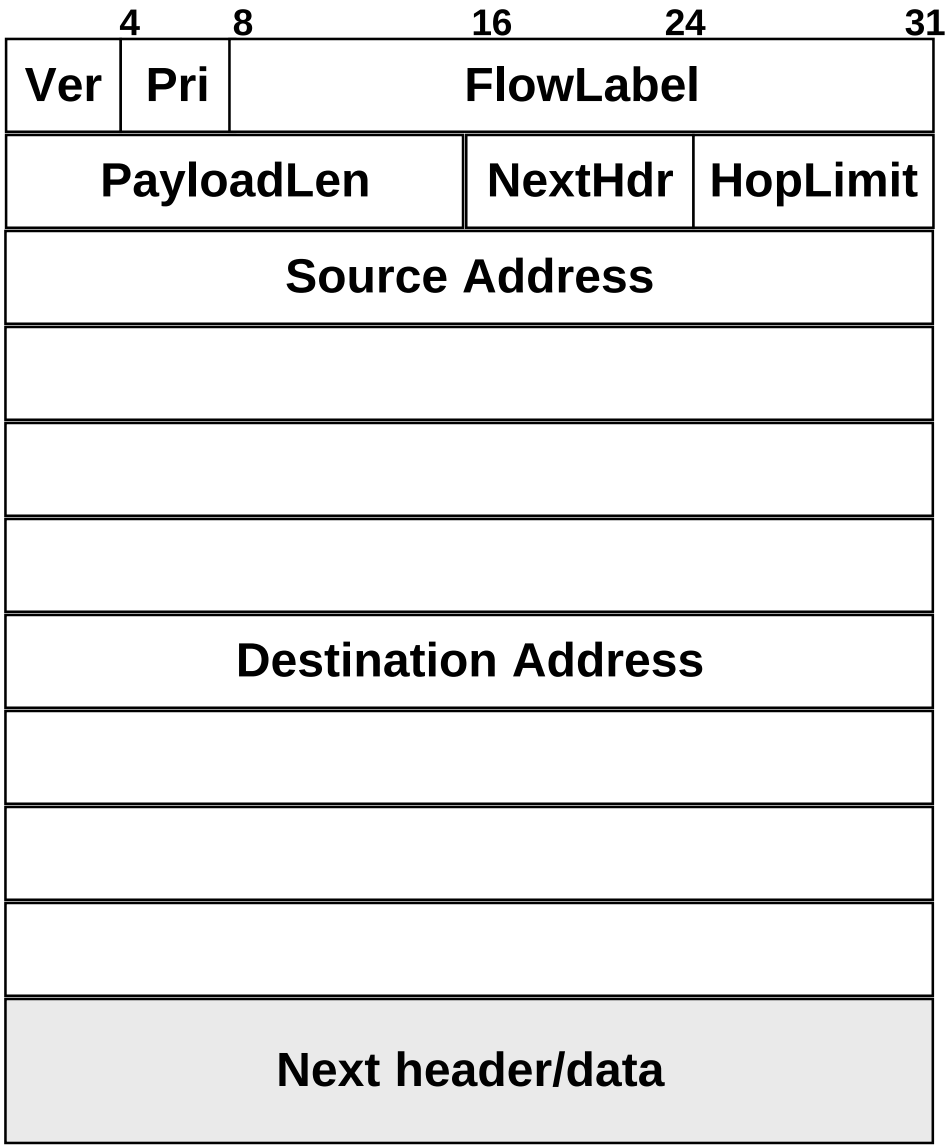
Hierarchical address space:

3					48
010	registryID	providerID	SubscriberID	SubnetID	InterfaceID

neat feature

- **embedded InterfaceID allows host to assign itself an IP address!**

IPv6 packet format



Ver	**IP version (6)**
Pri/Flowlabel	**Quality of Service)**
PayloadLen	**packet len (max 64KB)**
NextHdr	**optional/encapsulated header type**
HopLimit	**same as TTL in IPv4**
Source Address	**128-bit source addr**
Dest Address	**128-bit dest addr**

Optional header examples:

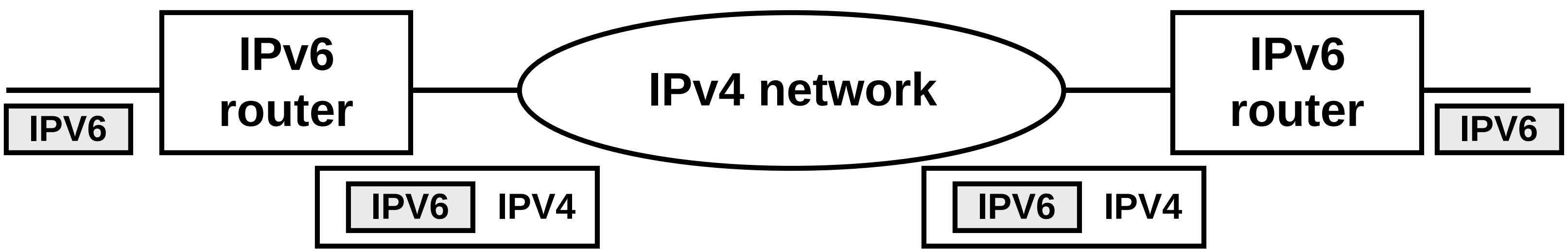
fragmentation (44)
authentication (51)
TCP (6)

Converting from IPv4 to IPv6

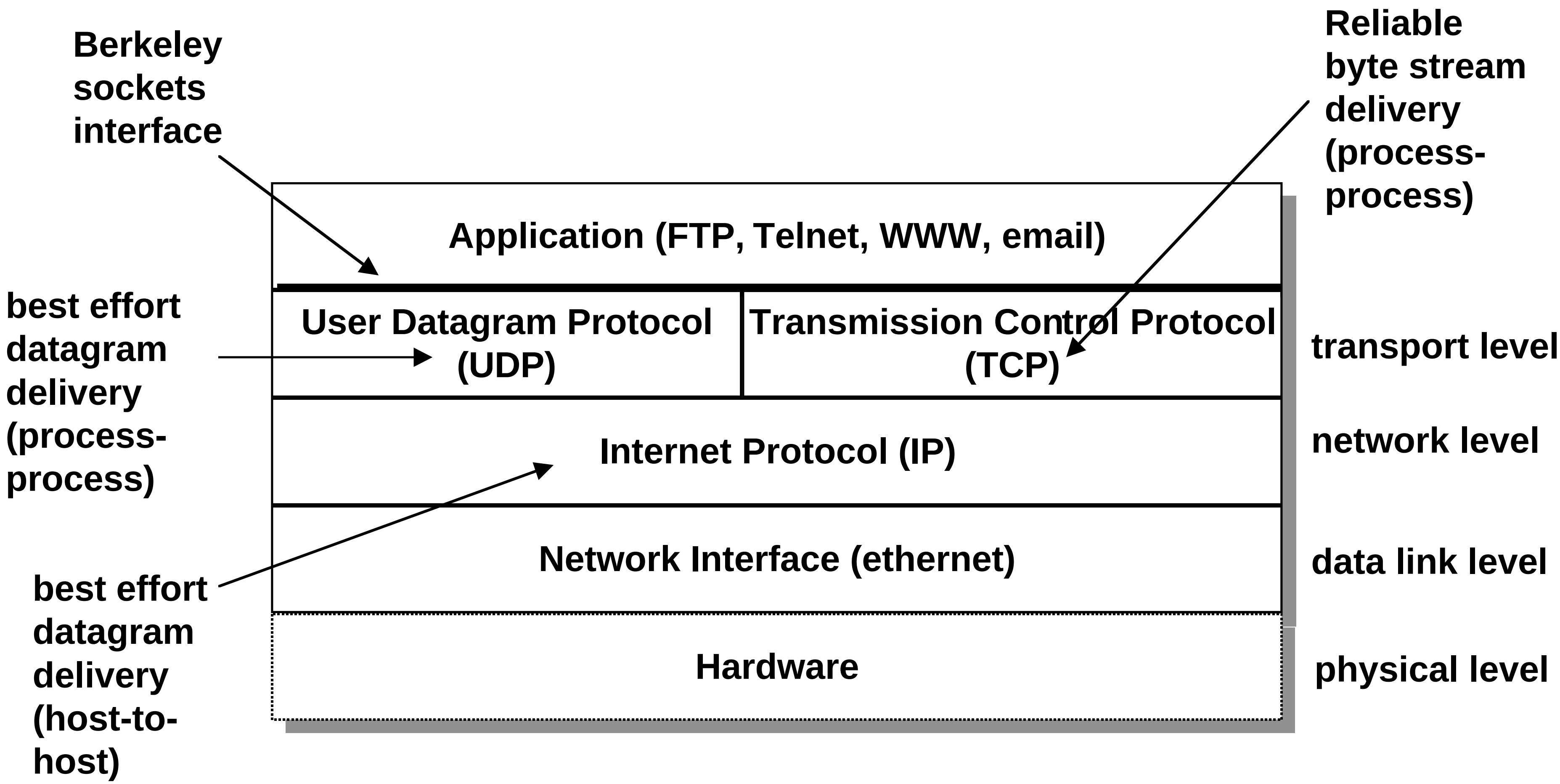
Not possible to have a “flag day”

Must upgrade incrementally

- **dual stack operation**
  - IPv6 nodes run both IPv4 and IPv6 protocol stacks
- **IP tunneling**
  - IP packet sent as payload of another IP packet
  - networking community’s version of indirection!



Internet protocol stack



UDP: User datagram protocol

Extends IP to provide *process-to-process* (end-to-end) datagram delivery

Mechanism for demultiplexing IP packets

Based on port abstraction

Process identified by <host, port> pair.

SrcPort	**DstPort**
CheckSum	**Length**
Data	

TCP: Transmission control protocol

Uses IP to provide reliable process-to-process byte stream delivery.

- **stream orientation**
  - sender transfers ordered 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 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.

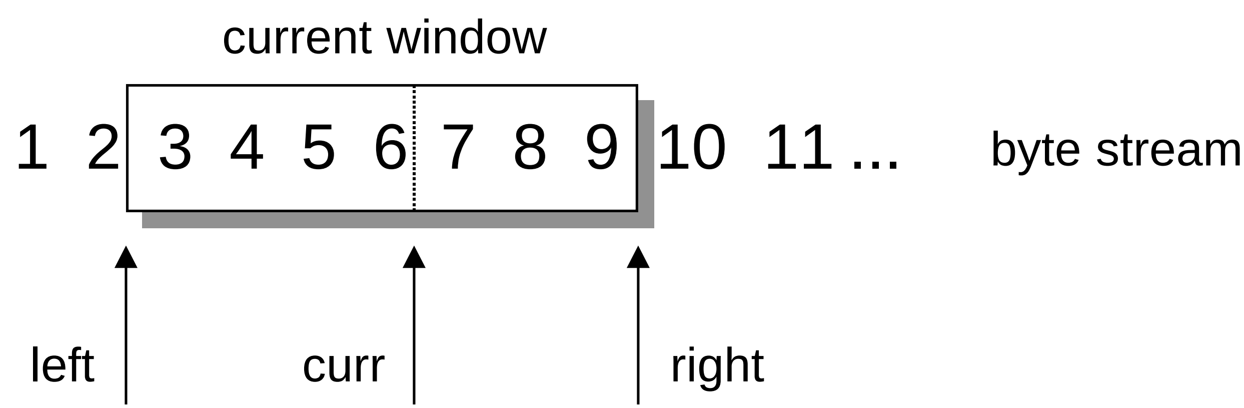
- 32 bit value
- wraps around
- initial values selected at runtime

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.

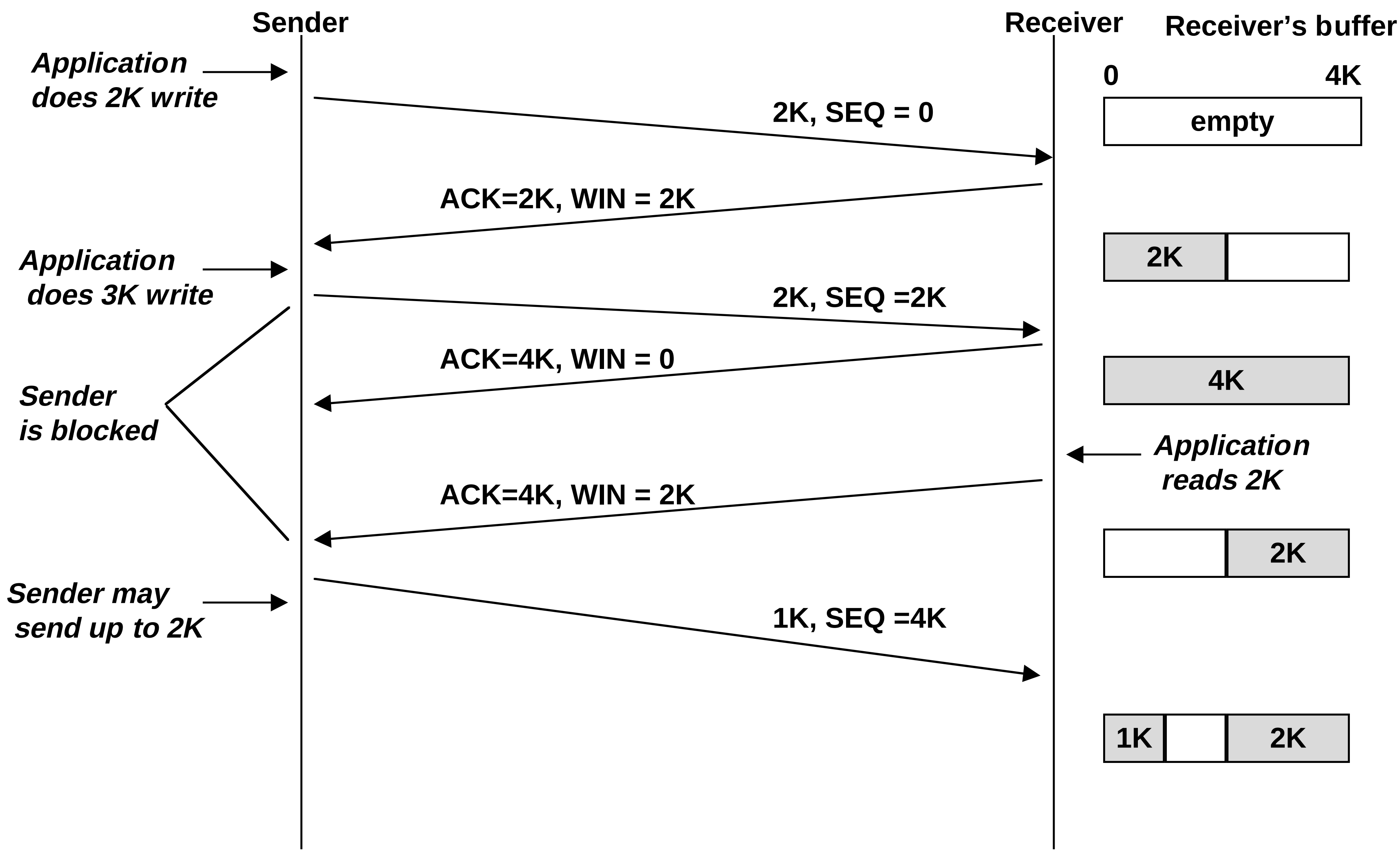
ACK indicates that data was received correctly.

- 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 segments 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.**

End-to-end data issues

Presentation formatting

- **must account for different data formats on different machines**
  - different byte orders
  - different word sizes

Compression

- **data can be compressed/decompressed on the endpoints to save network bandwidth (beyond our scope)**

Encryption

- **sensitive data can be encrypted/unencrypted on the endpoints.**

Authentication

- **Receivers may want to verify that messages really do come from the sender.**

Network byte order

ntohs

- convert unsigned short from network byte order (big endien) to host byte order.

htons

- convert unsigned short from host byte order to network byte order.

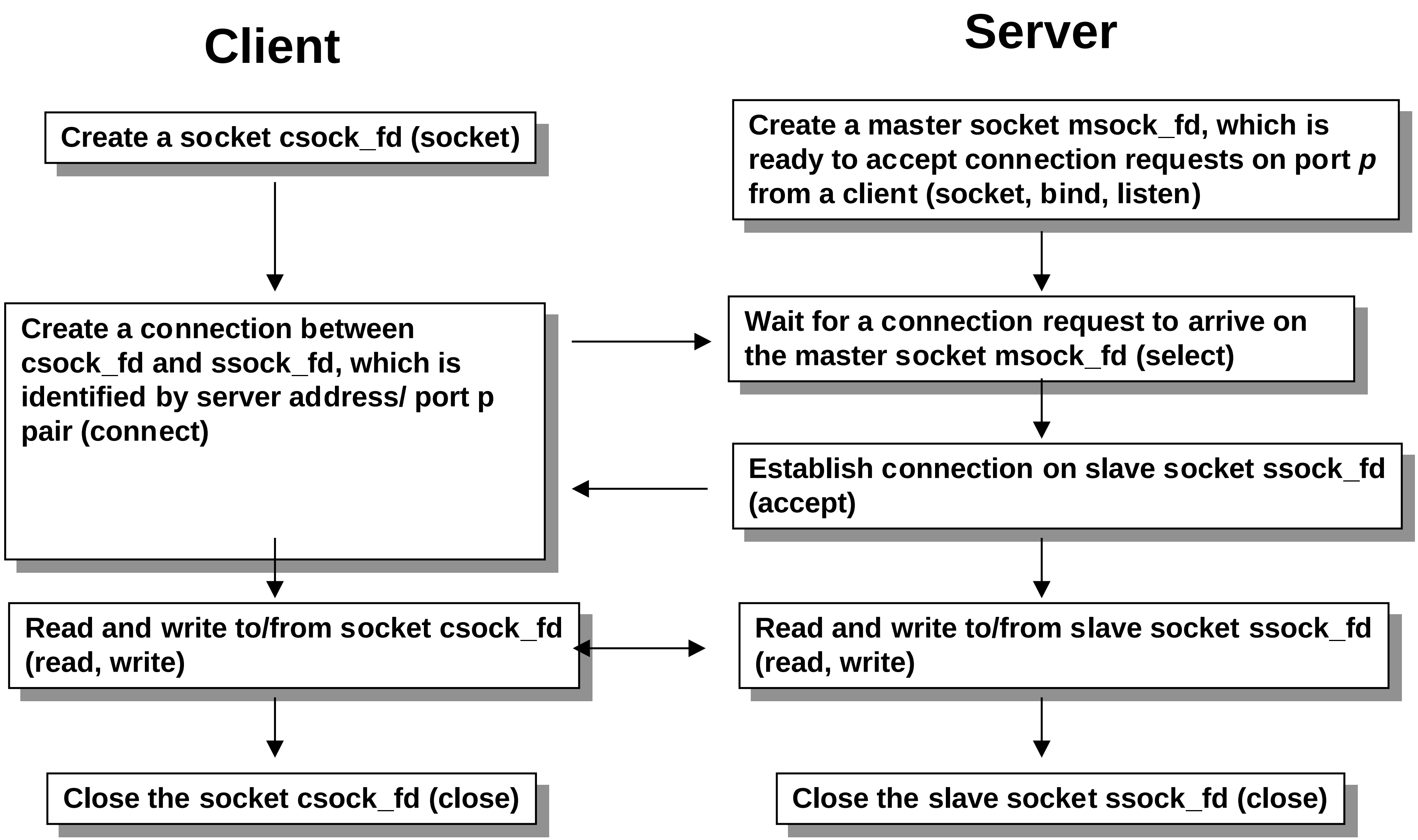
ntohl

- convert unsigned long from network byte order to host byte order.

htonl

- convert unsigned long from host byte order to network byte order.

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 conn request 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) {
       ready = readysock(master_sockfd);
       if (ready == 0)
          sleep(1);
    }
```

Example server code (cont)

```
    /* establish the pending connection */
    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) loop */
```

Key themes in Internetworking

Protocol layering

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

Must cope with heterogeneous networks

Must cope with huge scale

Must cope with imperfect environment

- Packets get corrupted and lost

No one has complete routing table

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