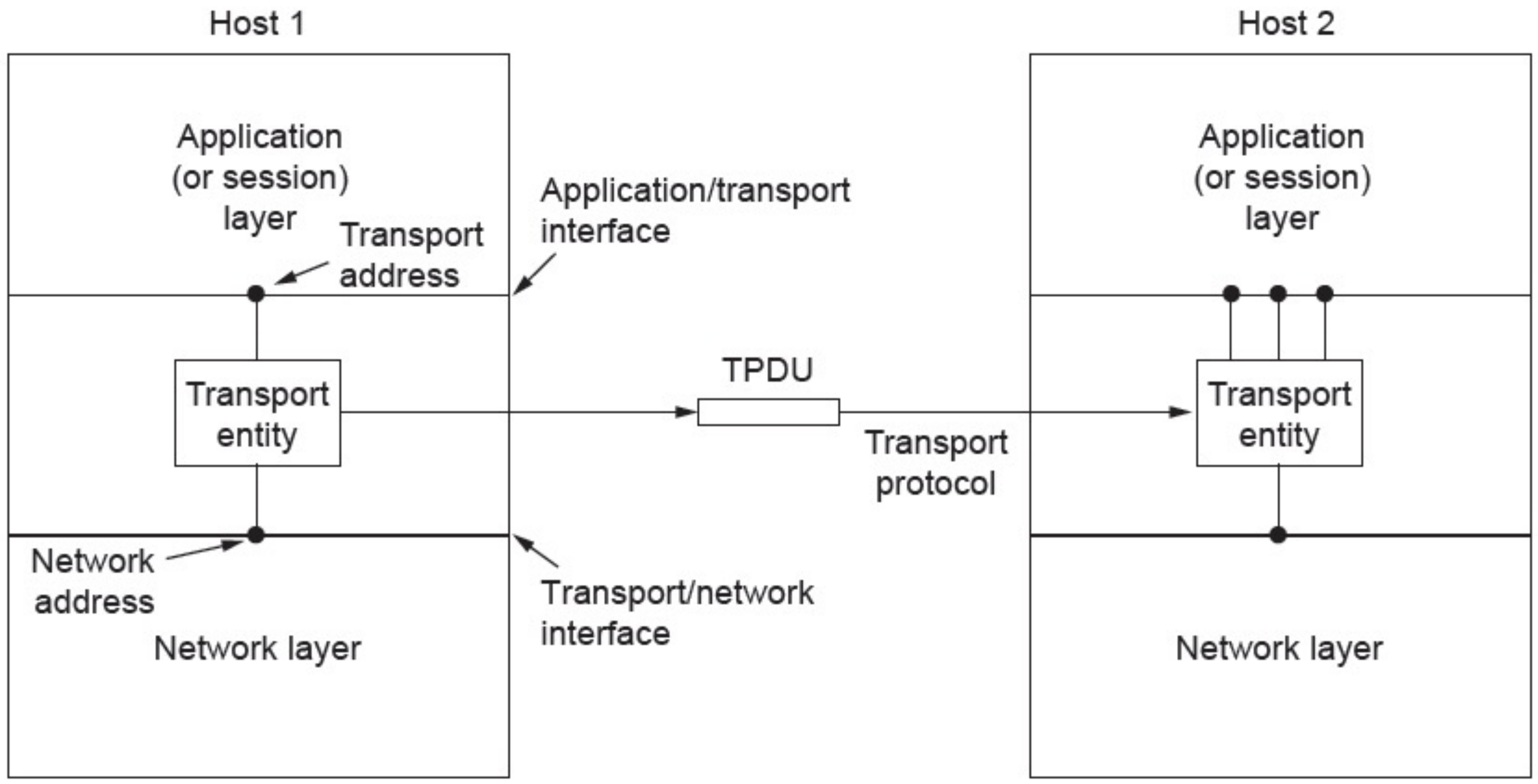
The Transport Layer

Chapter 6

Transport Service

- Upper Layer Services
- Transport Service Primitives
- Berkeley Sockets
- Example of Socket Programming: Internet File Server

Services Provided to the Upper Layers



The network, transport, and application layers

Services Provided to the Upper Layers

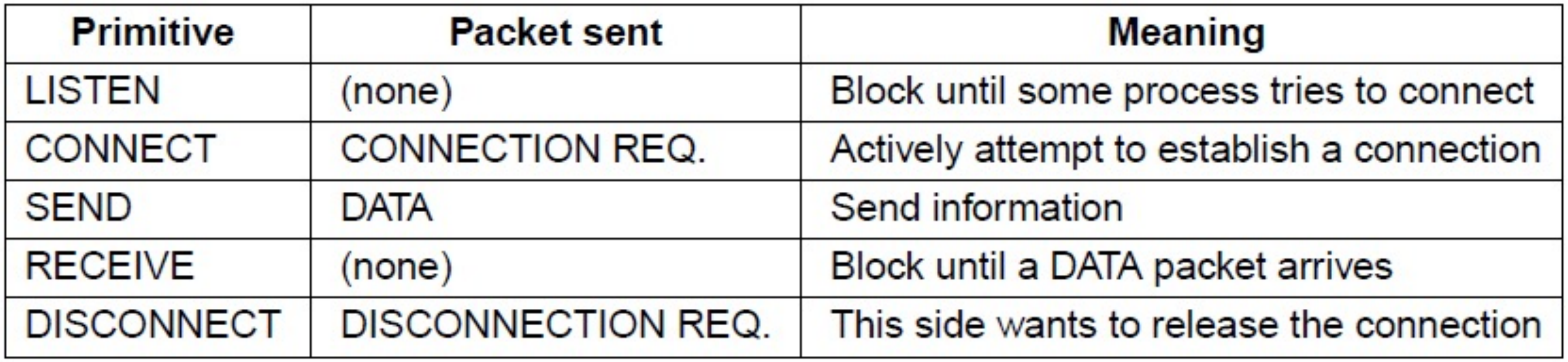
- Qualitative distinction between layers 1 through 4 and layer(s) above 4
- The bottom four layers can be seen as the **transport service provider**, whereas the upper layer(s) are the **transport service user**
- It forms the major boundary between the provider and user of the reliable data transmission service

Transport Service Primitives (1)

- To allow users to access the transport service, the transport layer must provide some operations to application programs, that is, a transport service interface
- Each transport service has its own interface
- The connection-oriented transport service, in contrast, is reliable. Of course, real networks are not error-free, but that is precisely the purpose of the transport layer—to provide a reliable service on top of an unreliable network
- As an example, consider two processes on a single machine connected by a pipe in UNIX (or any other interprocess communication facility). They assume the connection between them is 100% perfect. They do not want to know about acknowledgements, lost packets, congestion, or anything at all like that.
- The connection-oriented transport service is all about—hiding the imperfections of the network service so that user processes can just assume the existence of an error-free bit stream even when they are on different machines.
- The transport layer can also provide unreliable (datagram) service

Transport Service Primitives (2)

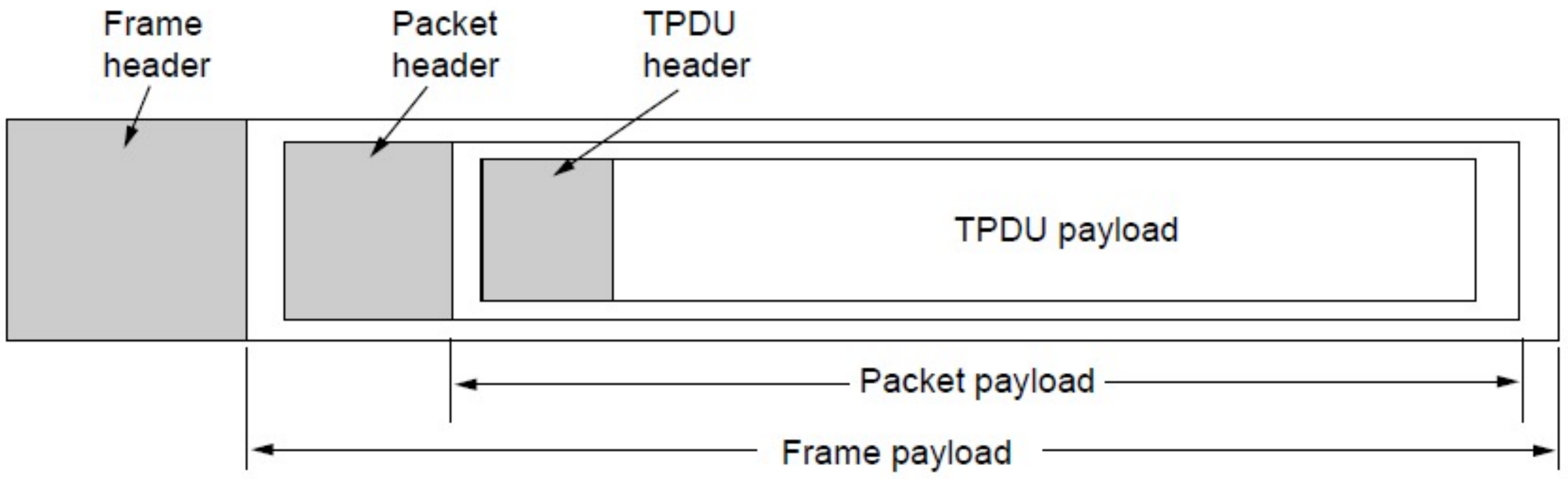
- A second difference between the network service and transport service is whom the services are intended for
- Very few users write their own transport entities, and thus few users or programs ever see the bare network service. In contrast, many programs (and thus programmers wiritng socket applications) see the transport primitives



Primitive	Packet sent	Meaning
LISTEN	(none)	Block until some process tries to connect
CONNECT	CONNECTION REQ.	Actively attempt to establish a connection
SEND	DATA	Send information
RECEIVE	(none)	Block until a DATA packet arrives
DISCONNECT	DISCONNECTION REQ.	This side wants to release the connection

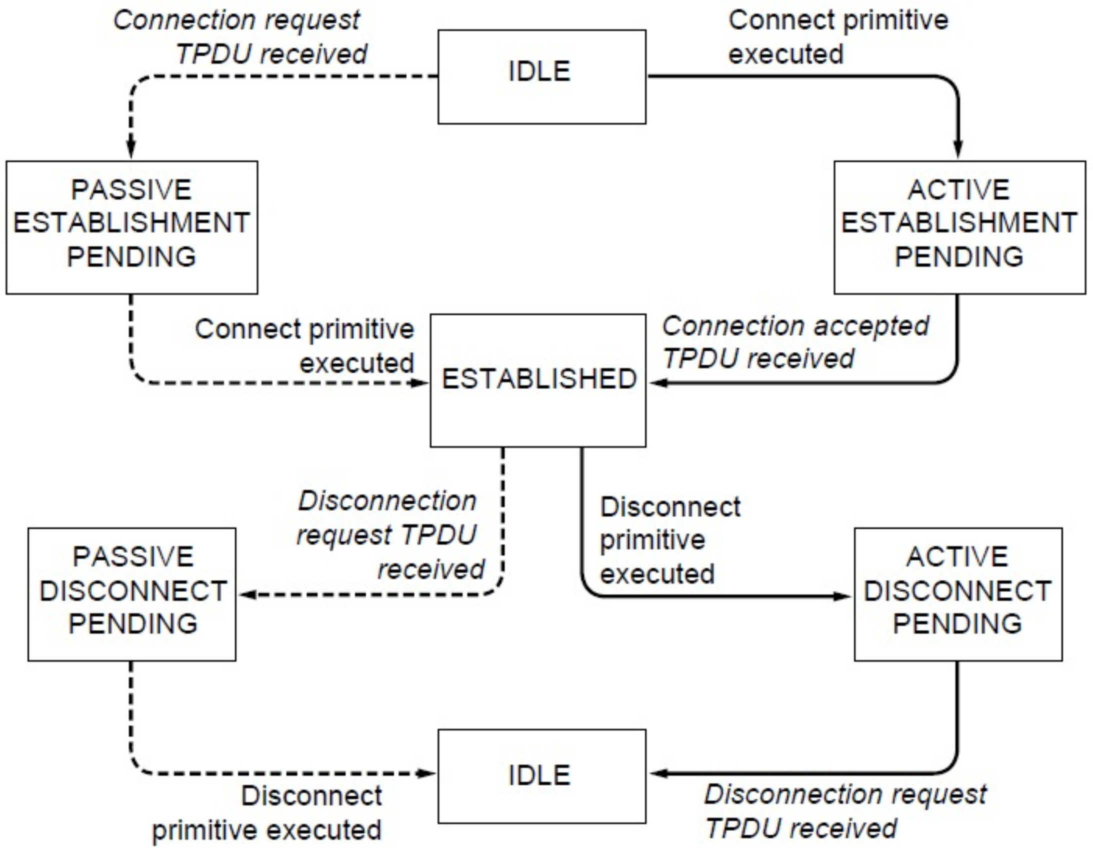
The primitives for a simple transport service

Transport Service Primitives (3)



Nesting of TPDUs, packets, and frames.

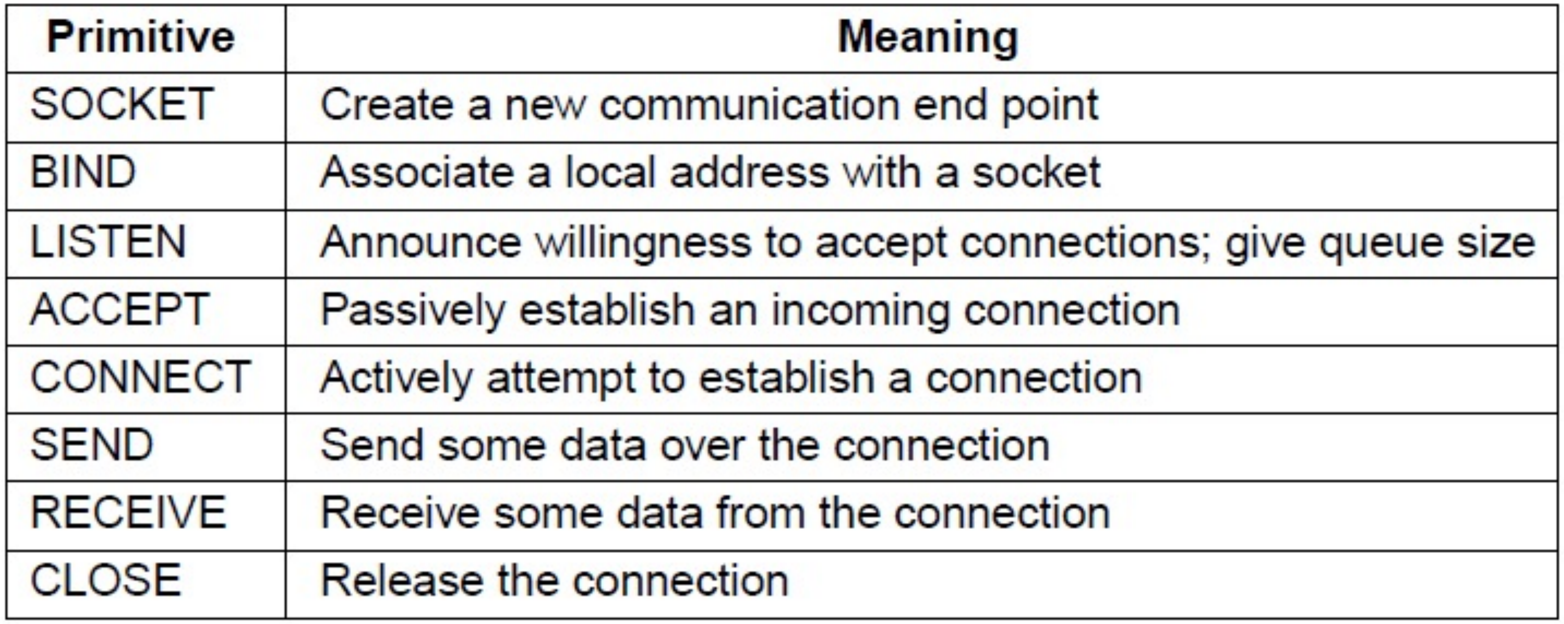
Connection Establishment



Connection request TPDU received
IDLE
Connect primitive executed
PASSIVE ESTABLISHMENT PENDING
ACTIVE ESTABLISHMENT PENDING
Connect primitive executed
ESTABLISHED
Connection accepted TPDU received
Disconnection request TPDU received
Disconnect primitive executed
PASSIVE DISCONNECT PENDING
ACTIVE DISCONNECT PENDING
IDLE
Disconnect primitive executed
Disconnection request TPDU received

A state diagram for a simple connection management scheme. Transitions labeled in italics are caused by packet arrivals. The solid lines show the client’s state sequence. The dashed lines show the server’s state sequence.

Berkeley Sockets (2)



Primitive	Meaning
SOCKET	Create a new communication end point
BIND	Associate a local address with a socket
LISTEN	Announce willingness to accept connections; give queue size
ACCEPT	Passively establish an incoming connection
CONNECT	Actively attempt to establish a connection
SEND	Send some data over the connection
RECEIVE	Receive some data from the connection
CLOSE	Release the connection

The socket primitives for TCP

Transport Service Primitives (1)

- The first four primitives in the list are executed in that order by servers.
- The SOCKET primitive creates a new endpoint and allocates table space for it within the transport entity. The parameters of the call specify the addressing format to be used, the type of service desired (e.g., reliable byte stream), and the protocol
- Newly created sockets are assigned network addresses with BIND primitive

Example of Socket Programming: An Internet File Server (1)

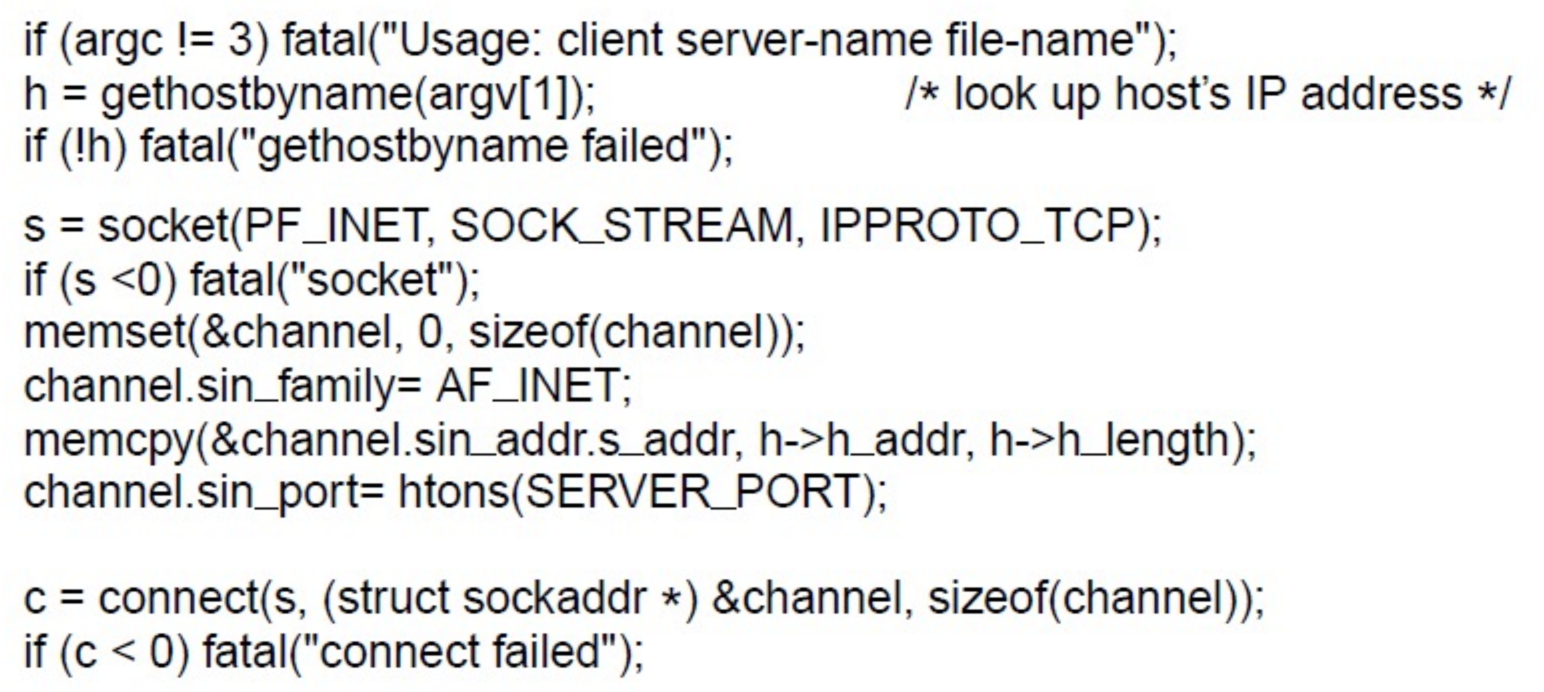
```
/* This page contains a client program that can request a file from the server program
 * on the next page. The server responds by sending the whole file.
 */

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

#define SERVER_PORT 12345                /* arbitrary, but client & server must agree */
#define BUF_SIZE 4096                    /* block transfer size */

int main(int argc, char **argv)
{
  int c, s, bytes;
  char buf[BUF_SIZE];                    /* buffer for incoming file */
  struct hostent *h;                     /* info about server */
  struct sockaddr_in channel;            /* holds IP address */
```

. . .

Client code using sockets

Example of Socket Programming: An Internet File Server (2)

. . .



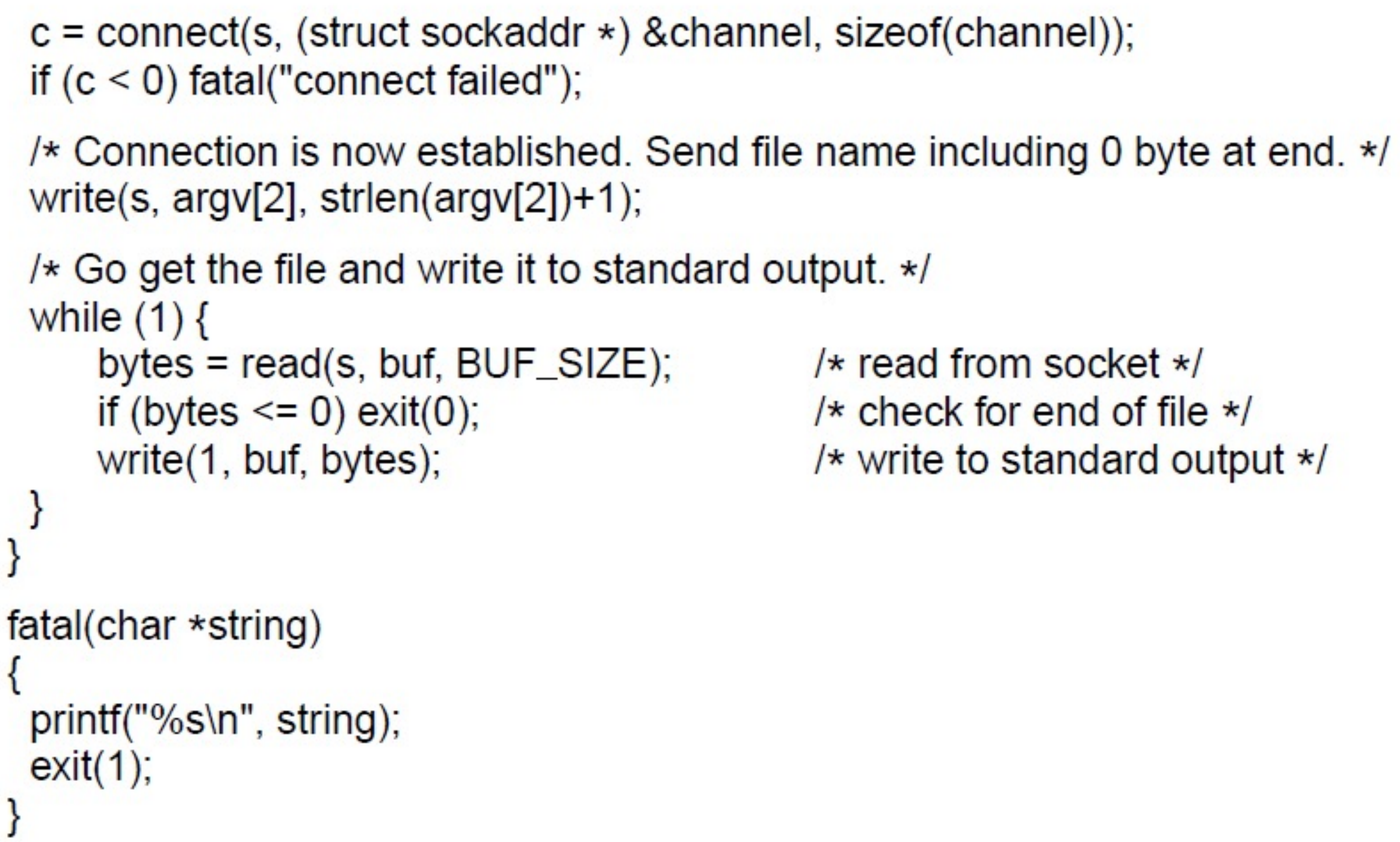
```
if (argc != 3) fatal("Usage: client server-name file-name");
h = gethostbyname(argv[1]);                    /* look up host's IP address */
if (!h) fatal("gethostbyname failed");

s = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
if (s <0) fatal("socket");
memset(&channel, 0, sizeof(channel));
channel.sin_family= AF_INET;
memcpy(&channel.sin_addr.s_addr, h->h_addr, h->h_length);
channel.sin_port= htons(SERVER_PORT);

c = connect(s, (struct sockaddr *) &channel, sizeof(channel));
if (c < 0) fatal("connect failed");
```

. . .

Client code using sockets

Example of Socket Programming: An Internet File Server (3)

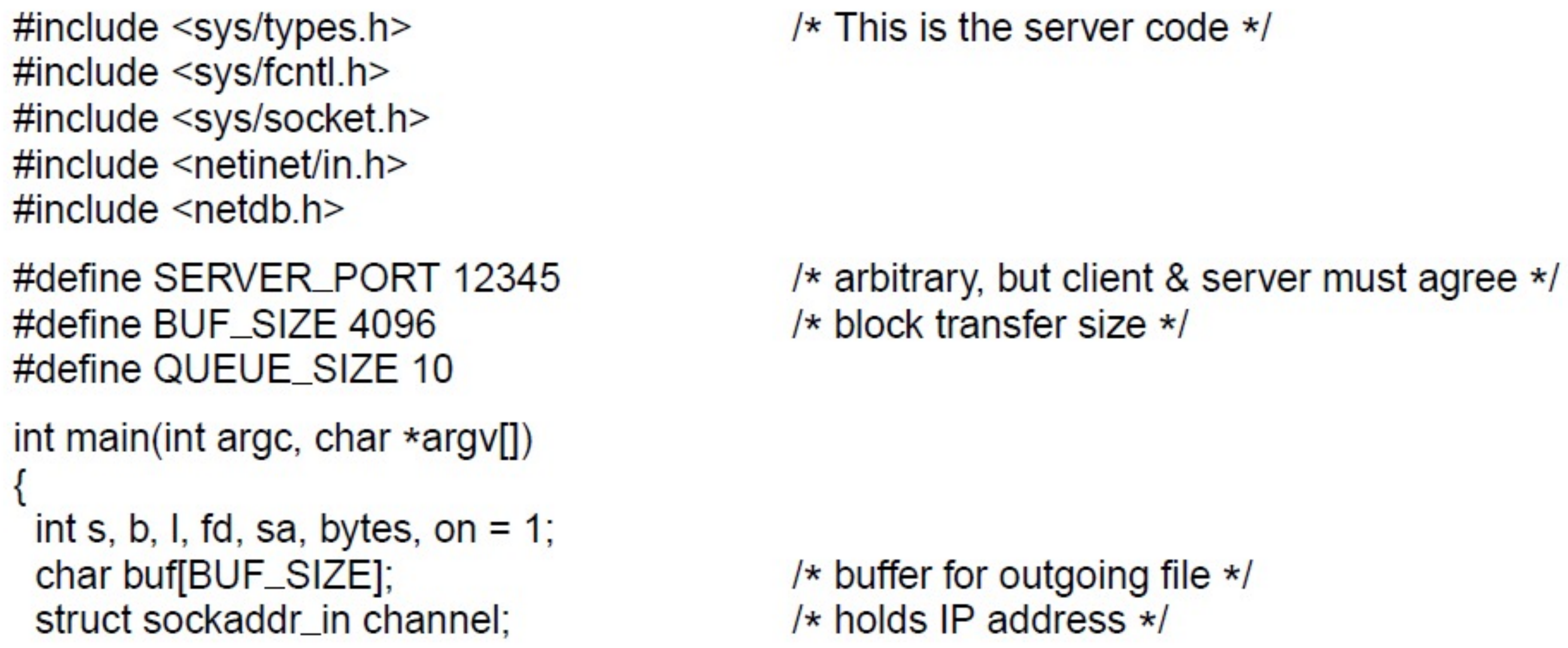
. . .



```
  c = connect(s, (struct sockaddr *) &channel, sizeof(channel));
  if (c < 0) fatal("connect failed");

  /* Connection is now established. Send file name including 0 byte at end. */
  write(s, argv[2], strlen(argv[2])+1);

  /* Go get the file and write it to standard output. */
  while (1) {
       bytes = read(s, buf, BUF_SIZE);          /* read from socket */
       if (bytes <= 0) exit(0);                 /* check for end of file */
       write(1, buf, bytes);                    /* write to standard output */
  }
}

fatal(char *string)
{
  printf("%s\n", string);
  exit(1);
}
```

Client code using sockets

Example of Socket Programming: An Internet File Server (4)



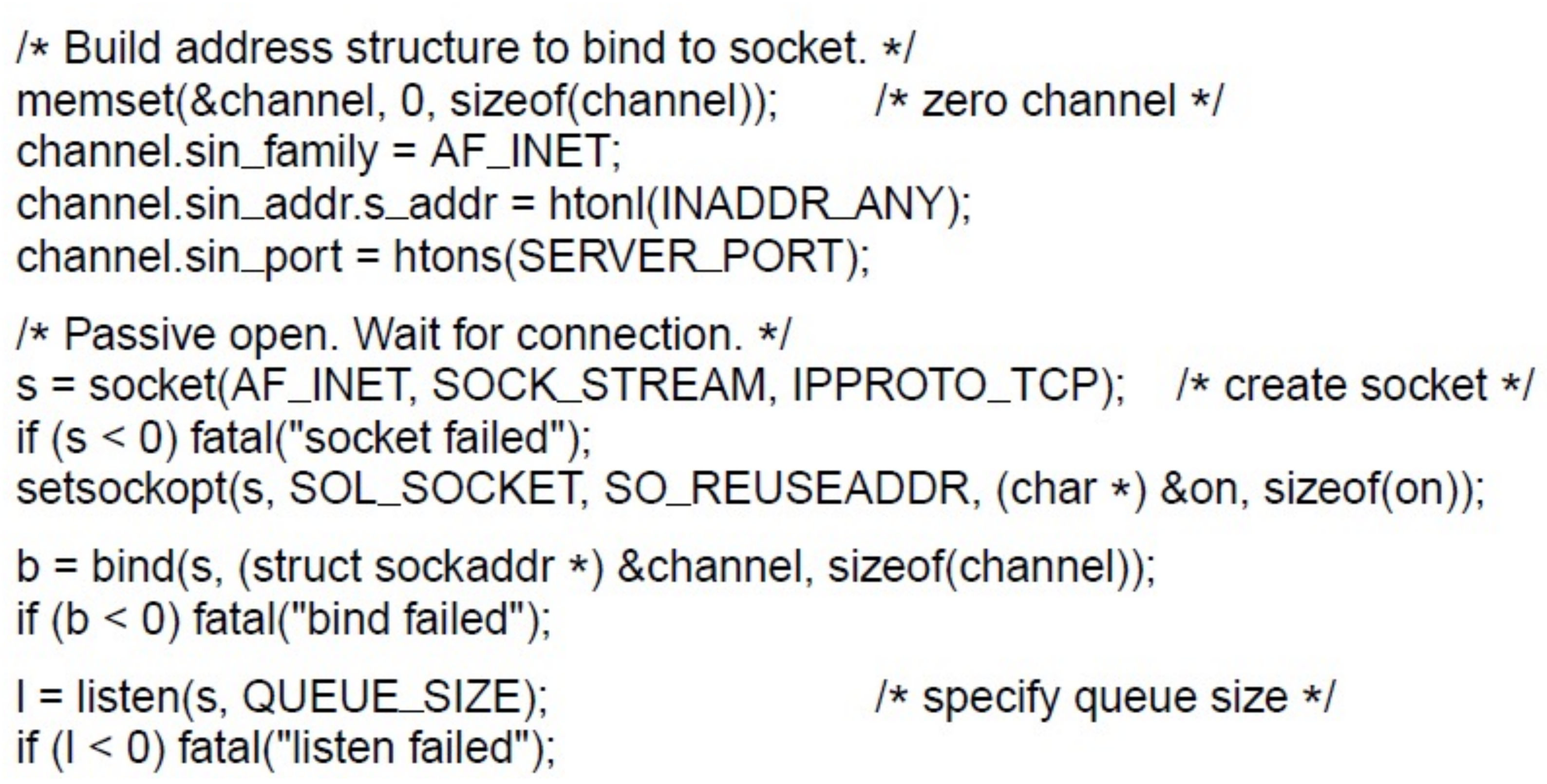
```
#include <sys/types.h>                              /* This is the server code */
#include <sys/fcntl.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

#define SERVER_PORT 12345                           /* arbitrary, but client & server must agree */
#define BUF_SIZE 4096                               /* block transfer size */
#define QUEUE_SIZE 10

int main(int argc, char *argv[])
{
  int s, b, l, fd, sa, bytes, on = 1;
  char buf[BUF_SIZE];                               /* buffer for outgoing file */
  struct sockaddr_in channel;                       /* holds IP address */
```

. . .

Server code

Example of Socket Programming: An Internet File Server (5)

. . .



```
/* Build address structure to bind to socket. */
memset(&channel, 0, sizeof(channel));        /* zero channel */
channel.sin_family = AF_INET;
channel.sin_addr.s_addr = htonl(INADDR_ANY);
channel.sin_port = htons(SERVER_PORT);

/* Passive open. Wait for connection. */
s = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);    /* create socket */
if (s < 0) fatal("socket failed");
setsockopt(s, SOL_SOCKET, SO_REUSEADDR, (char *) &on, sizeof(on));

b = bind(s, (struct sockaddr *) &channel, sizeof(channel));
if (b < 0) fatal("bind failed");

l = listen(s, QUEUE_SIZE);                   /* specify queue size */
if (l < 0) fatal("listen failed");
```

. . .

Server code

Example of Socket Programming: An Internet File Server (6)

. . .

```
/* Socket is now set up and bound. Wait for connection and process it. */
while (1) {
      sa = accept(s, 0, 0);                    /* block for connection request */
      if (sa < 0) fatal("accept failed");

      read(sa, buf, BUF_SIZE);                 /* read file name from socket */

      /* Get and return the file. */
      fd = open(buf, O_RDONLY);                /* open the file to be sent back */
      if (fd < 0) fatal("open failed");

      while (1) {
            bytes = read(fd, buf, BUF_SIZE); /* read from file */
            if (bytes <= 0) break;             /* check for end of file */
            write(sa, buf, bytes);             /* write bytes to socket */
      }
      close(fd);                               /* close file */
      close(sa);                               /* close connection */
}
}
```

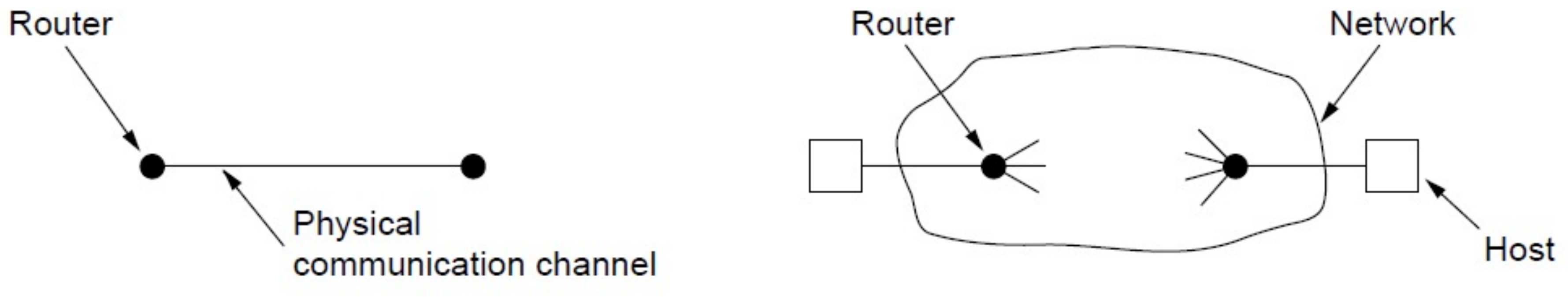
Server code

Elements of Transport Protocols (1)

- Addressing
- Connection establishment
- Connection release
- Error control and flow control
- Multiplexing
- Crash recovery

Elements of Transport Protocols (2)

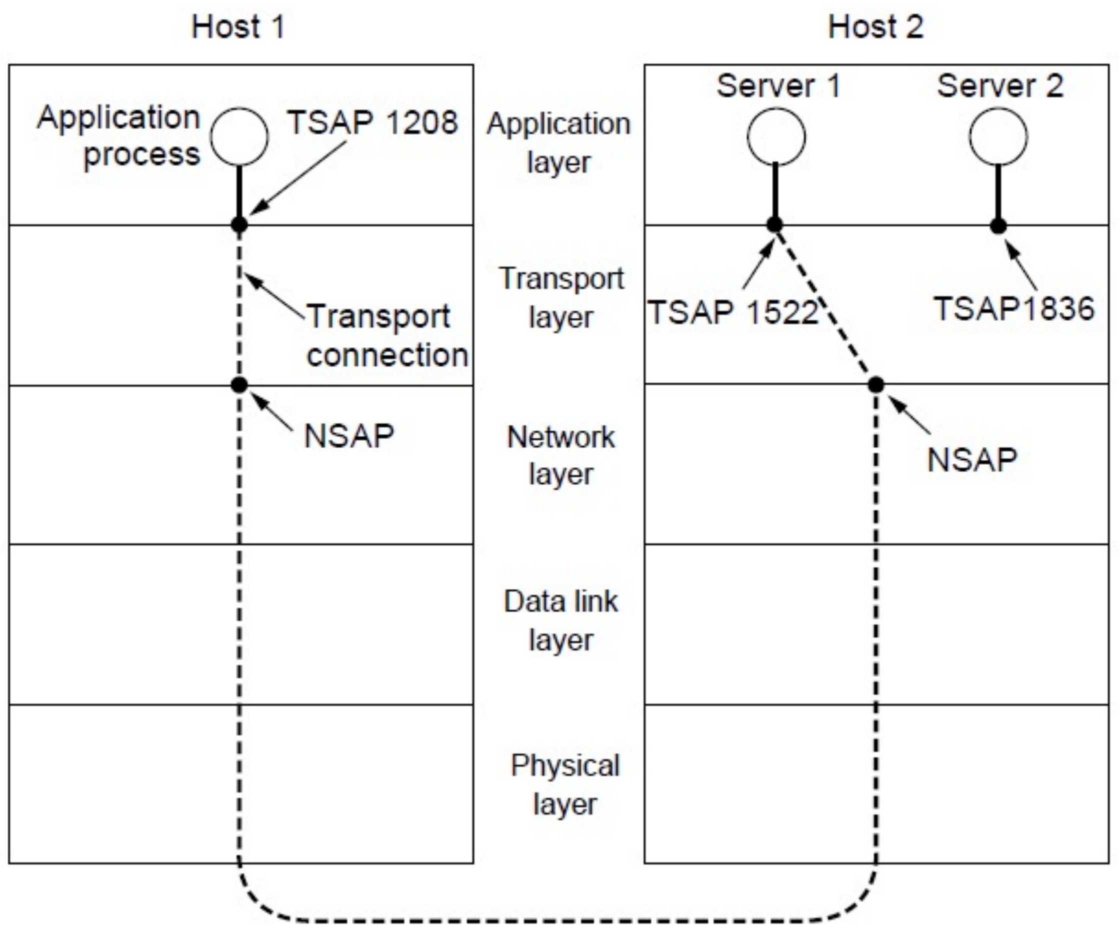
- The process of establishing a connection over the wire is simple
- The potential existence of storage capacity in the network
- Buffering and flow control are needed in both layers, but the presence of a large and varying number of connections with fluctuating bandwidth as the connections compete with each other may require a different approach than we used in the data link layer.



a) Environment of the data link layer. (b) Environment of the transport layer.

(a) (b)

Addressing (1)



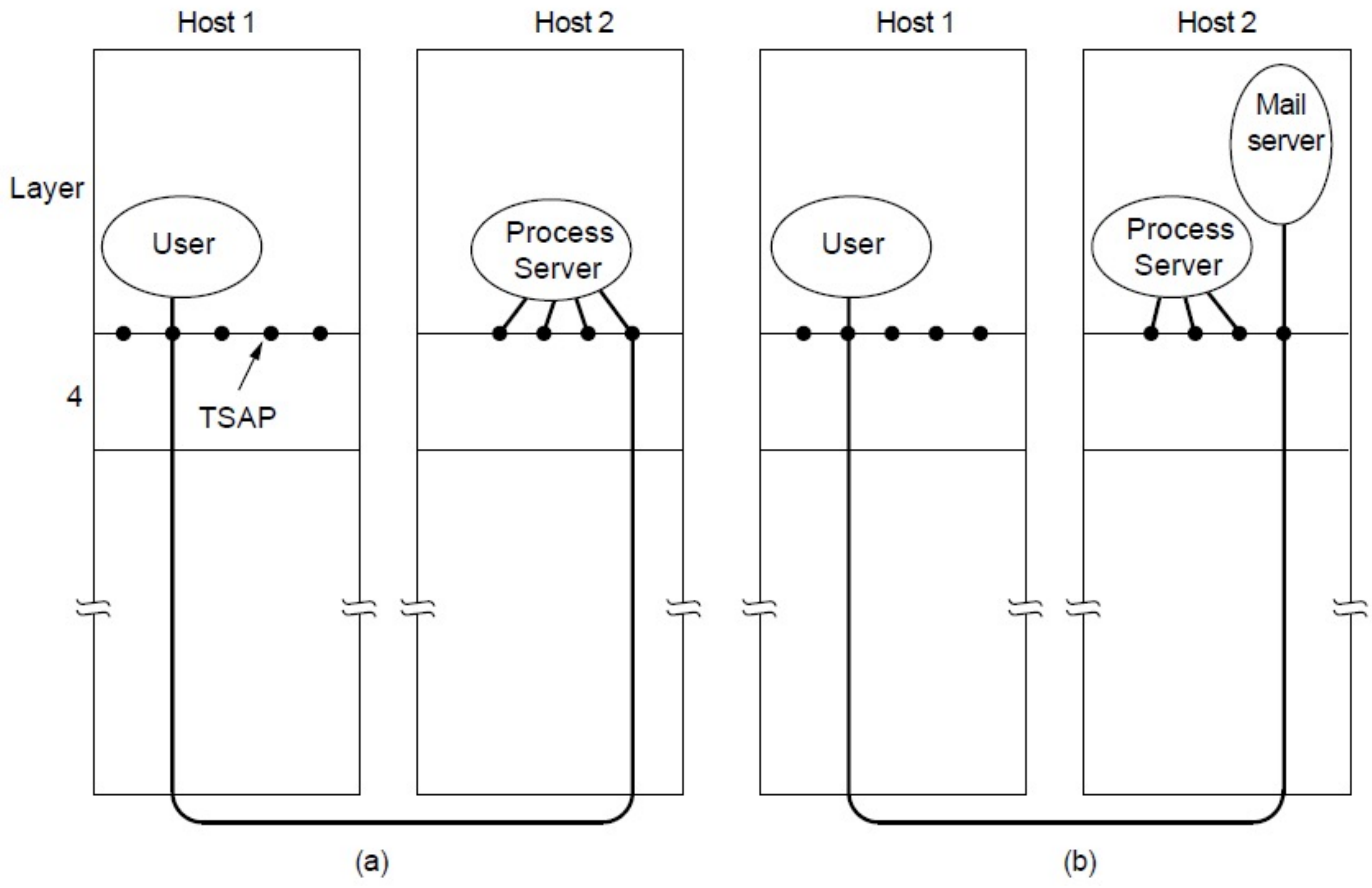
TSAPs, NSAPs, and transport connections

How does the host 1 know that the mail server is attached to TSAP 1522?

Addressing (2)

- How does the user process on host 1 know that the mail server is attached to TSAP 1522?
- One possibility is to fix that service on that port
- Use **Portmapper -** The user sends a message specifying the service name, and the portmapper sends back the TSAP address
- Use **ProcessServer -** acts as a proxy for less heavily used servers. It listens to a set of ports at the same time, then spawn the requested server

Addressing (3)



How a user process in host 1 establishes a connection with a mail server in host 2 via a process server.

Connection Establishment (1)

- If network is fully congested, neither data or ack are delivered, leading to multiple retransmissions.
- If user crashes and start using the same set of sequence used earlier, it might lead to duplicate packet delivery problem
- Fund transfer example
- Solution can be –

1. discard the port numbers used before crash
2. Assign each connection unique number

Connection Establishment (1)

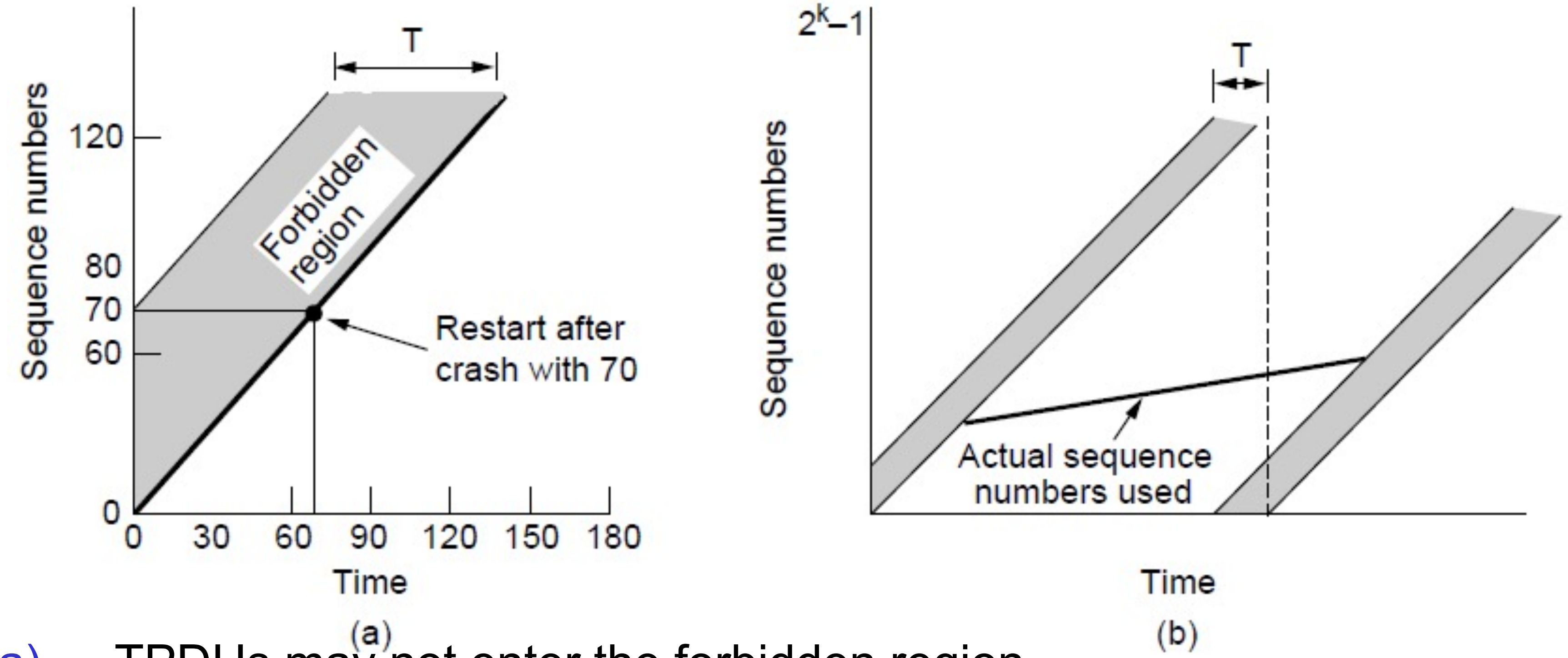
Techniques for restricting packet lifetime

- Restricted network design.
- Putting a hop counter in each packet.
- Timestamping each packet.

Connection Establishment (2)

* Forbidden region shows the times for which segment seq numbers are illegal

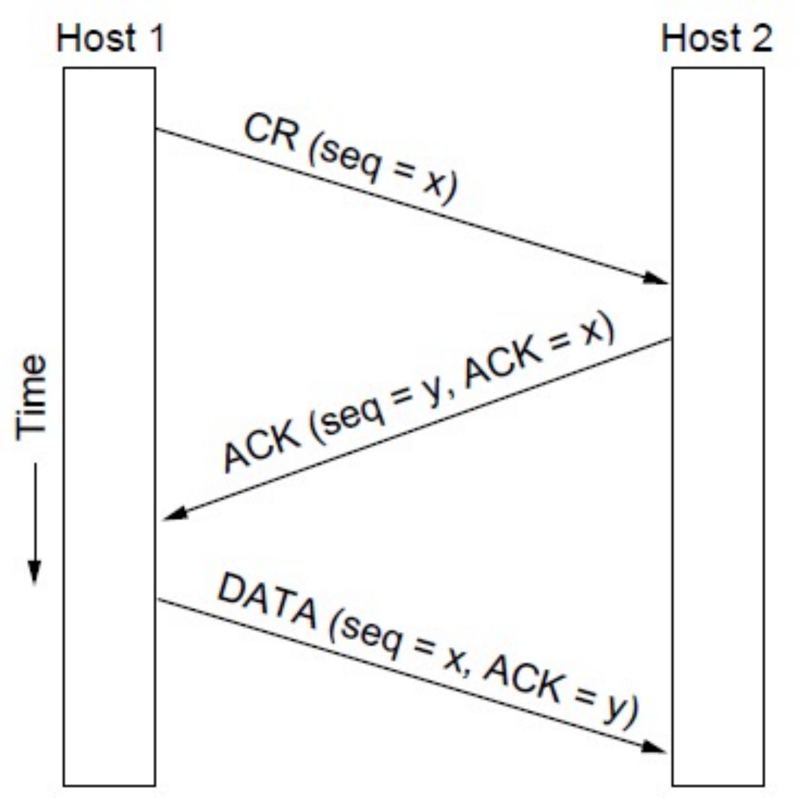
* If any segment is sent with a sequence number in this region, it could be delayed and impersonate a different packet with the same sequence number that will be issued slightly later



a) TPDUs may not enter the forbidden region.

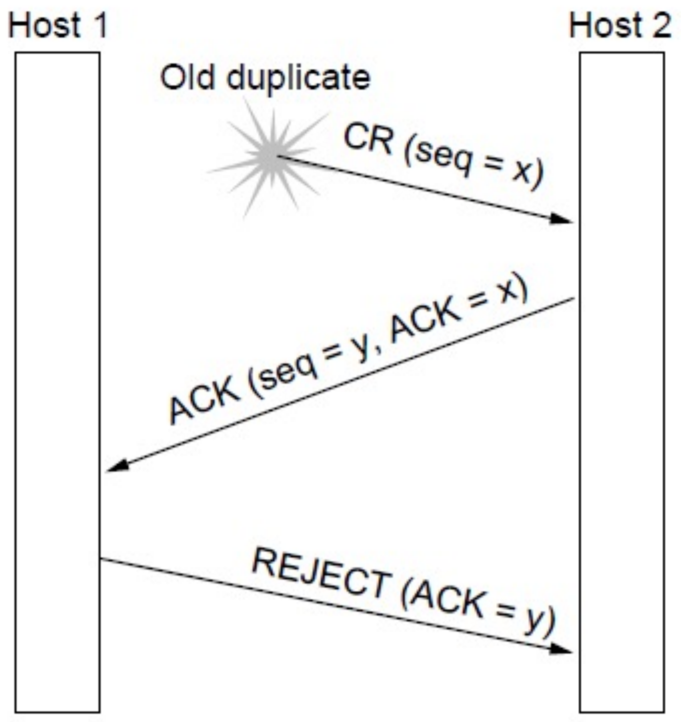
b) The resynchronization problem.

Connection Establishment (3)



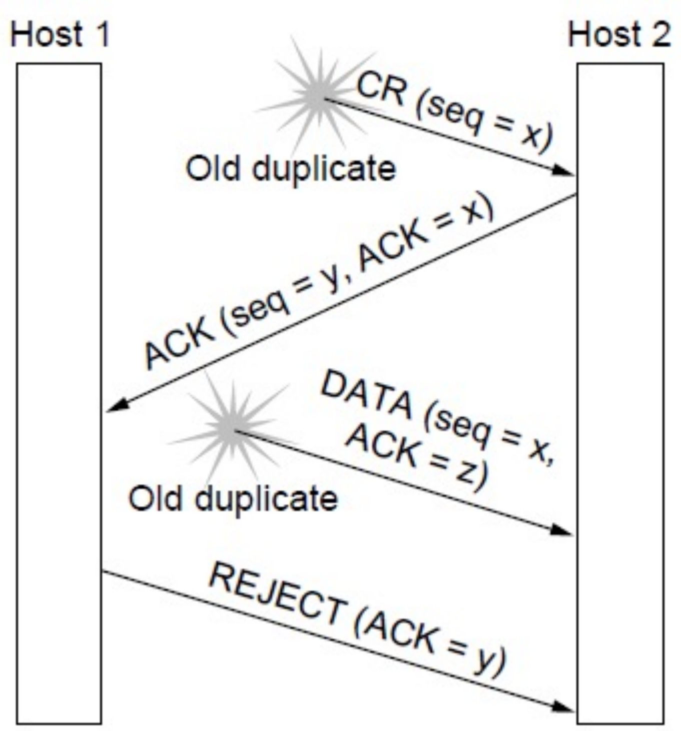
Three protocol scenarios for establishing a connection using a three-way handshake. CR denotes CONNECTION REQUEST. Normal operation.

Connection Establishment (4)



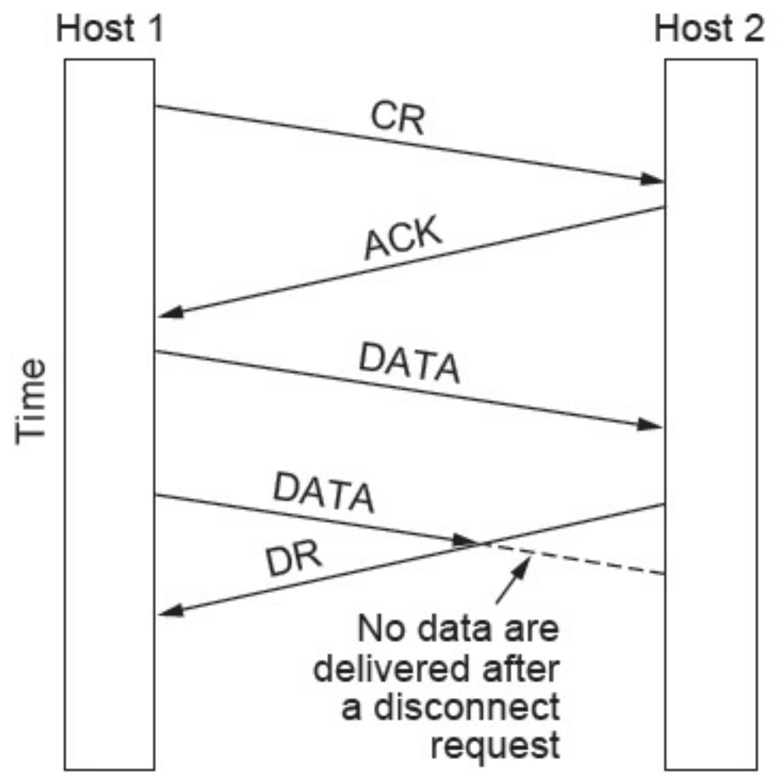
Three protocol scenarios for establishing a connection using a three-way handshake. CR denotes CONNECTION REQUEST. Old duplicate CONNECTION REQUEST appearing out of nowhere.

Connection Establishment (5)



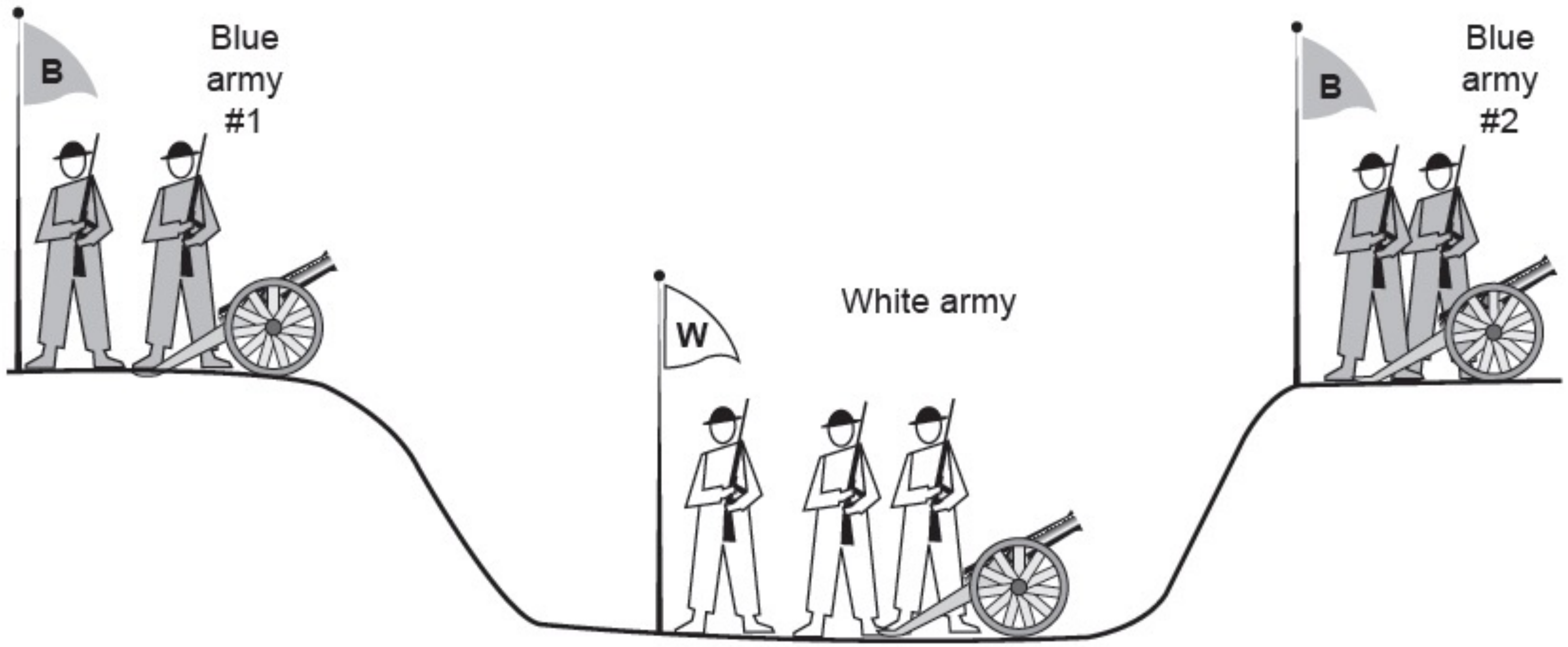
Three protocol scenarios for establishing a connection using a three-way handshake. CR denotes CONNECTION REQUEST. Duplicate CONNECTION REQUEST and duplicate ACK

Connection Release (1)



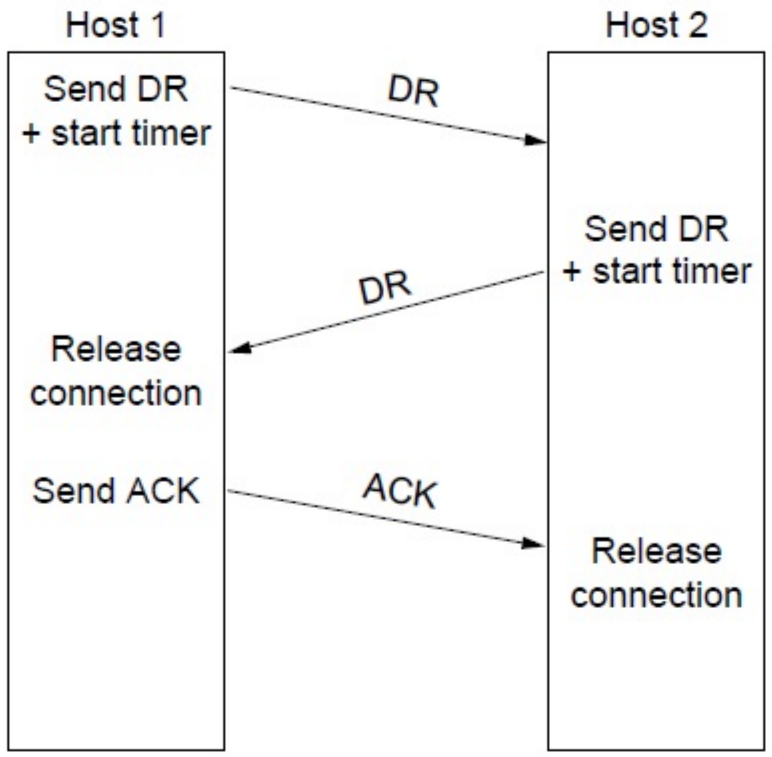
Abrupt disconnection with loss of data

Connection Release (2)



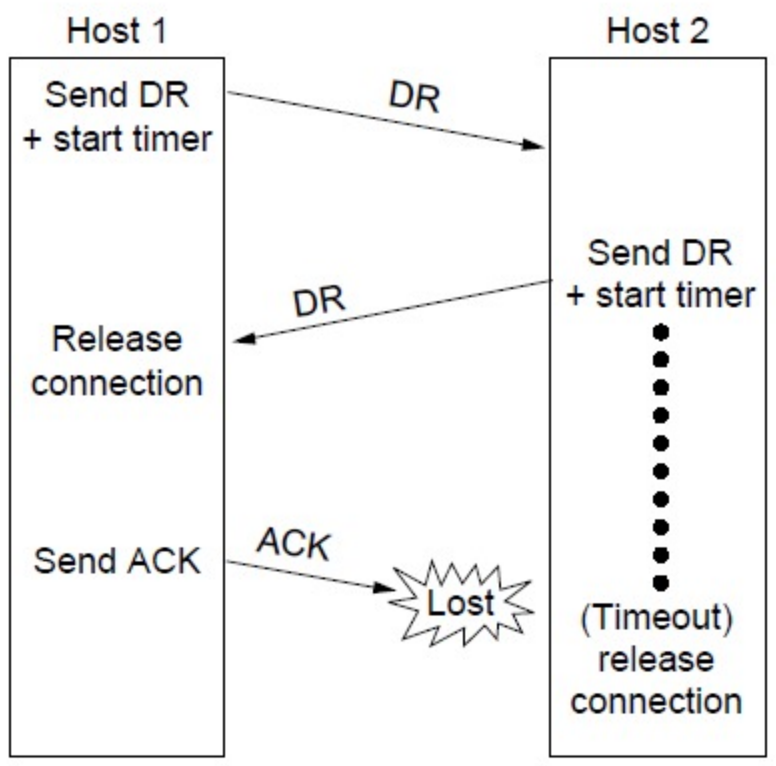
The two-army problem

Connection Release (3)



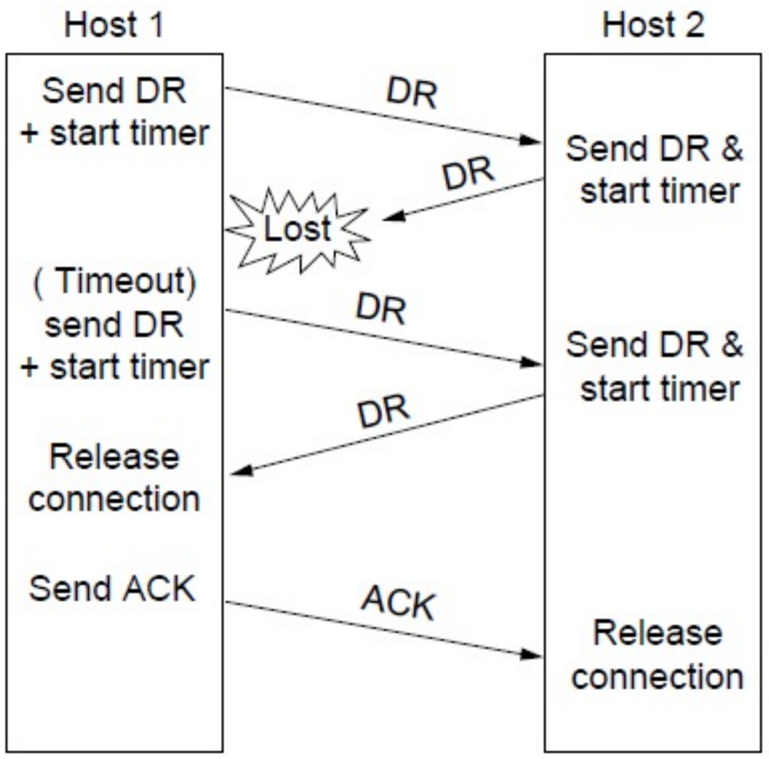
Four protocol scenarios for releasing a connection.
(a) Normal case of three-way handshake

Connection Release (4)



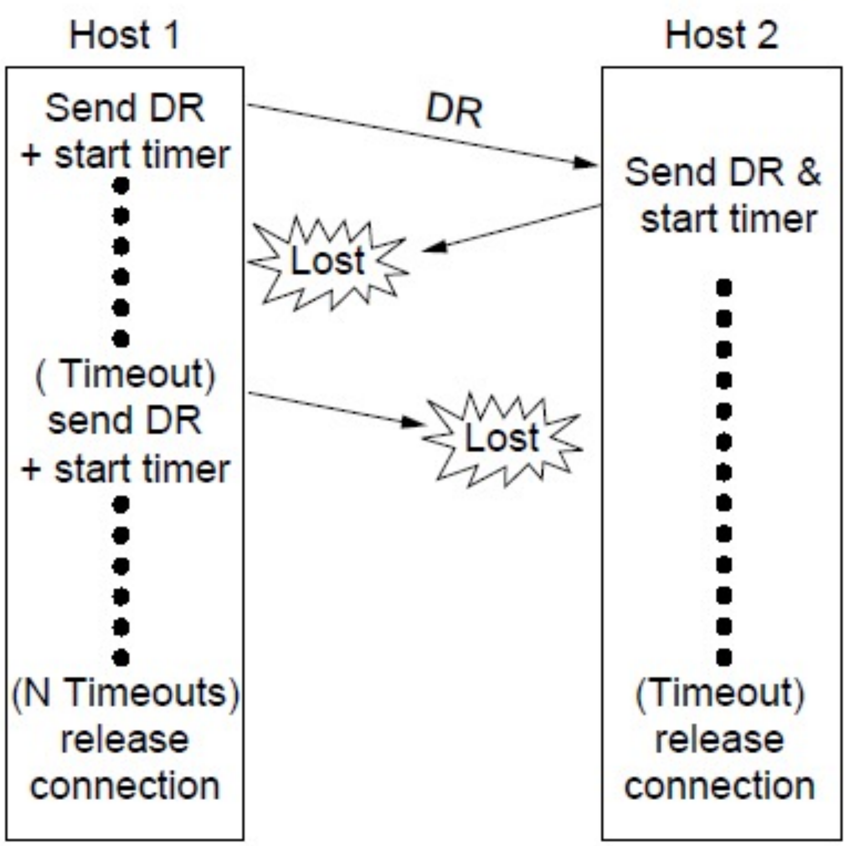
Four protocol scenarios for releasing a connection.
(b) Final ACK lost.

Connection Release (5)



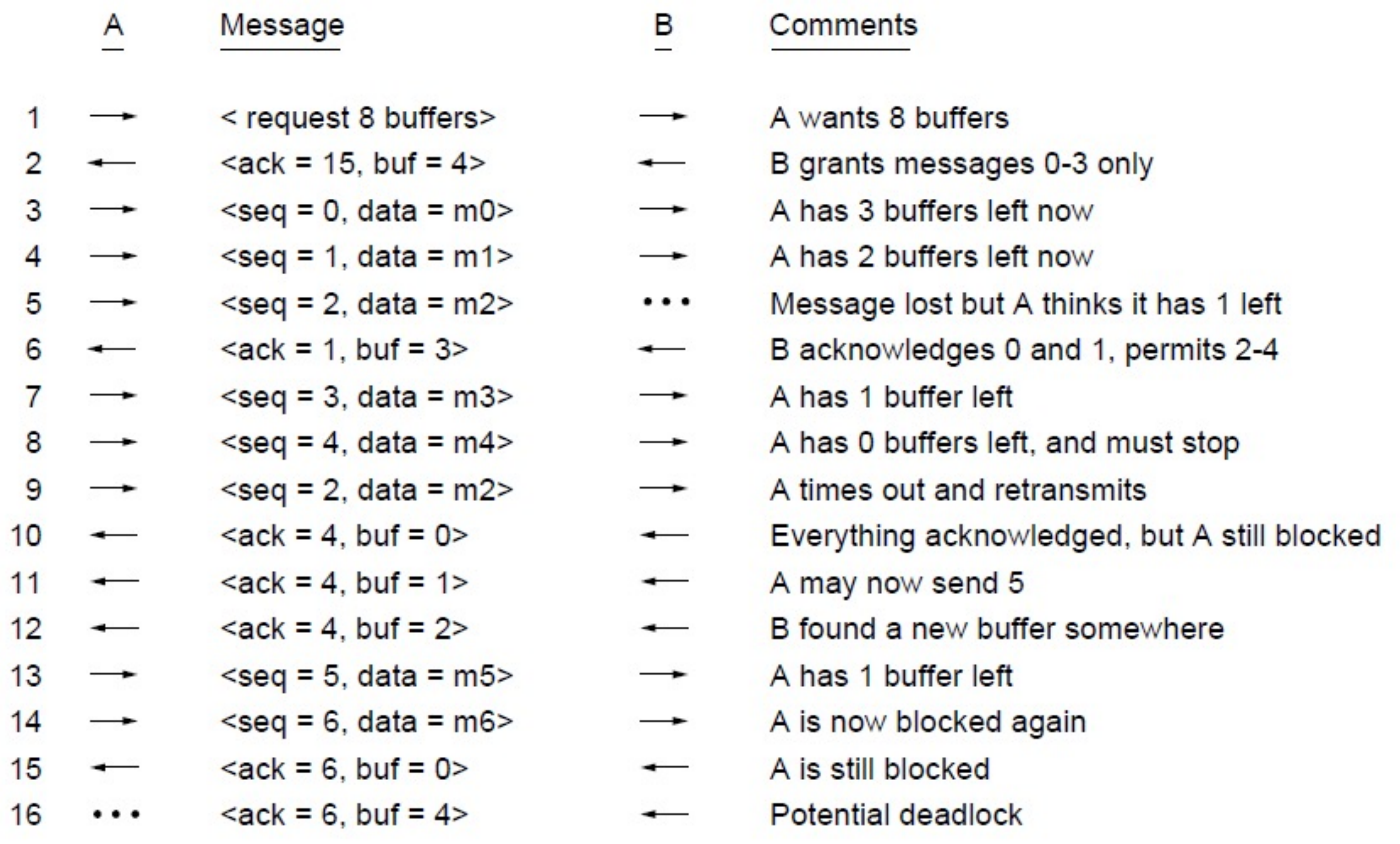
Four protocol scenarios for releasing a connection.
(c) Response lost

Connection Release (6)



Four protocol scenarios for releasing a connection.
(d) Response lost and subsequent DRs lost.

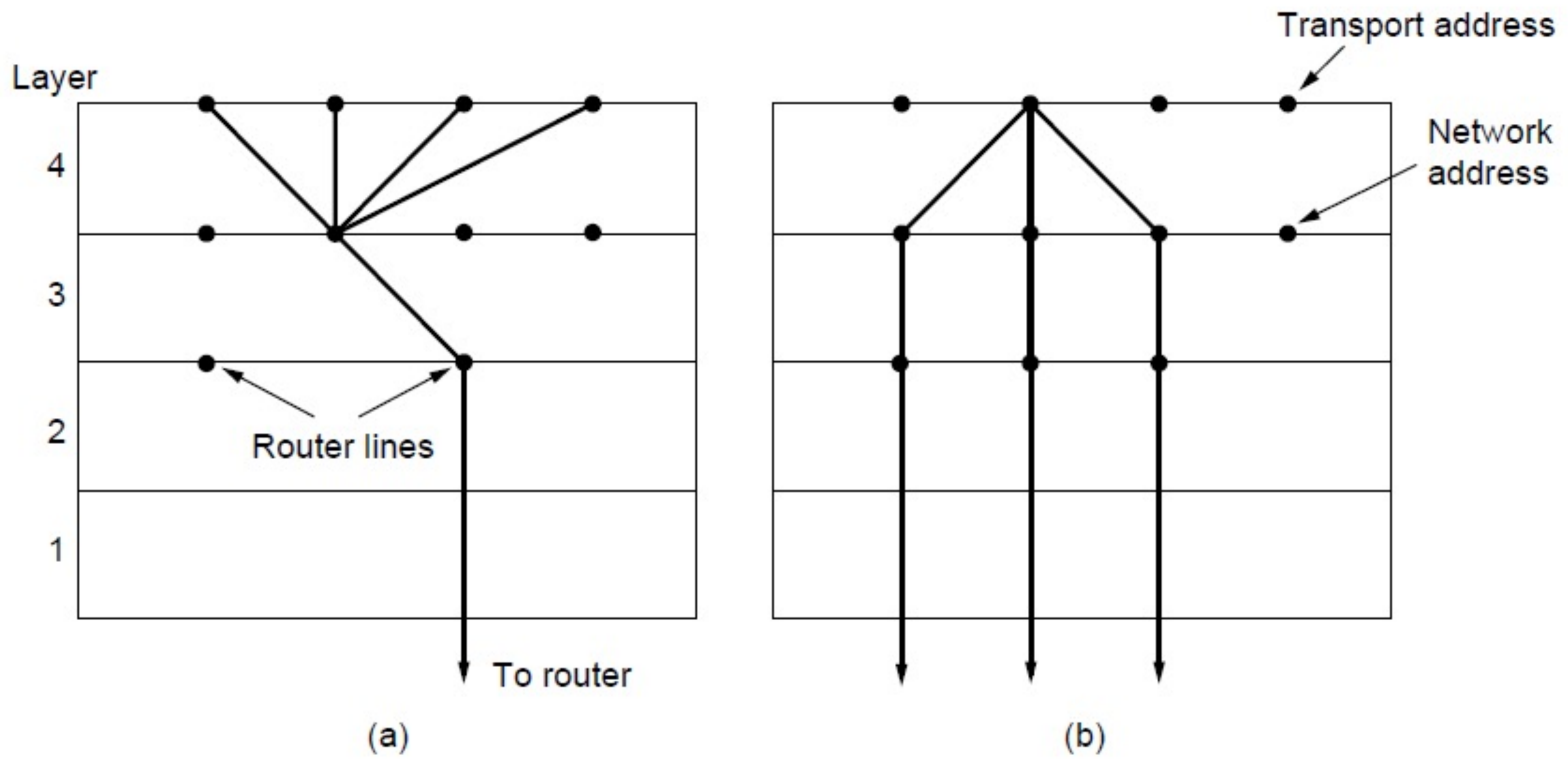
Error Control and Flow Control (2)



	A	Message	B	Comments
1	→	< request 8 buffers>	→	A wants 8 buffers
2	←	<ack = 15, buf = 4>	←	B grants messages 0-3 only
3	→	<seq = 0, data = m0>	→	A has 3 buffers left now
4	→	<seq = 1, data = m1>	→	A has 2 buffers left now
5	→	<seq = 2, data = m2>	• • •	Message lost but A thinks it has 1 left
6	←	<ack = 1, buf = 3>	←	B acknowledges 0 and 1, permits 2-4
7	→	<seq = 3, data = m3>	→	A has 1 buffer left
8	→	<seq = 4, data = m4>	→	A has 0 buffers left, and must stop
9	→	<seq = 2, data = m2>	→	A times out and retransmits
10	←	<ack = 4, buf = 0>	←	Everything acknowledged, but A still blocked
11	←	<ack = 4, buf = 1>	←	A may now send 5
12	←	<ack = 4, buf = 2>	←	B found a new buffer somewhere
13	→	<seq = 5, data = m5>	→	A has 1 buffer left
14	→	<seq = 6, data = m6>	→	A is now blocked again
15	←	<ack = 6, buf = 0>	←	A is still blocked
16	• • •	<ack = 6, buf = 4>	←	Potential deadlock

Dynamic buffer allocation. The arrows show the direction of transmission. An ellipsis (...) indicates a lost TPDU

Multiplexing

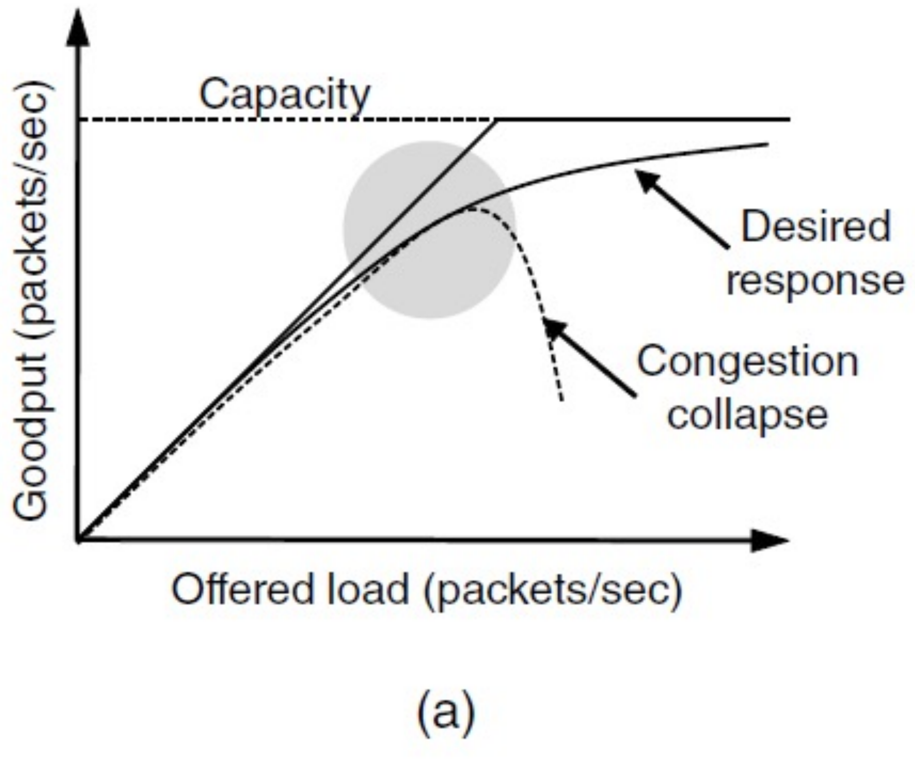


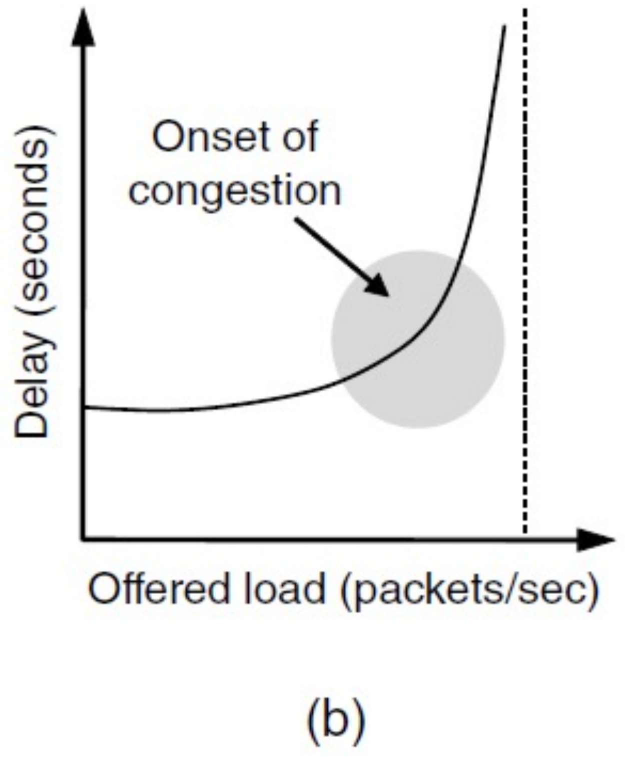
(a) Multiplexing. (b) Inverse multiplexing.

Congestion Control

- Desirable bandwidth allocation
- Regulating the sending rate

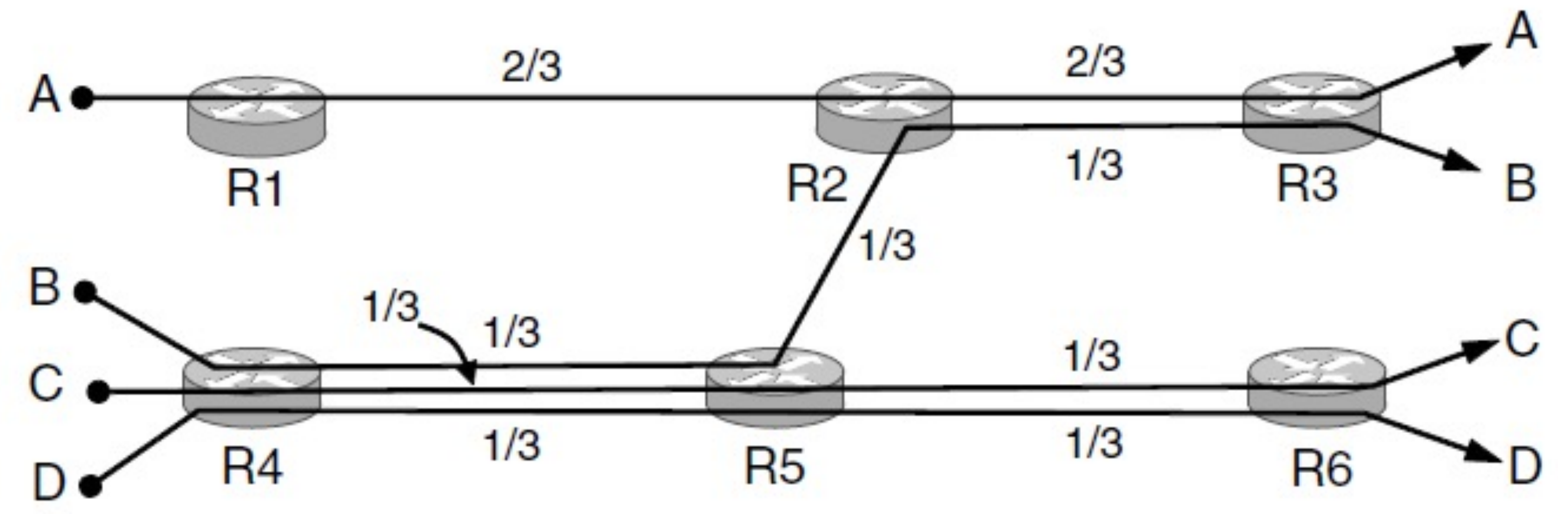
Desirable Bandwidth Allocation (1)





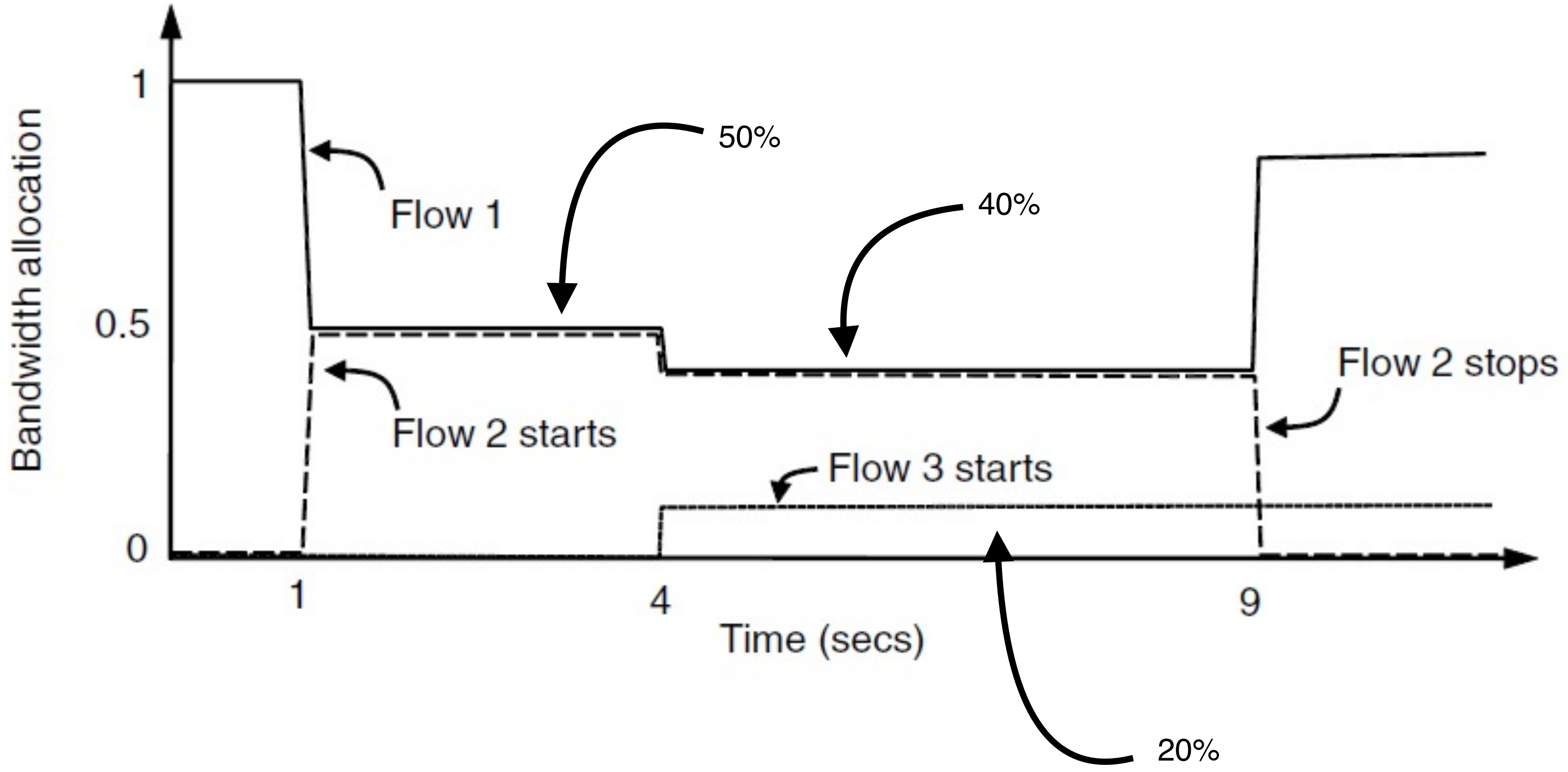
(a) Goodput and (b) delay as a function of offered load

Desirable Bandwidth Allocation (2)



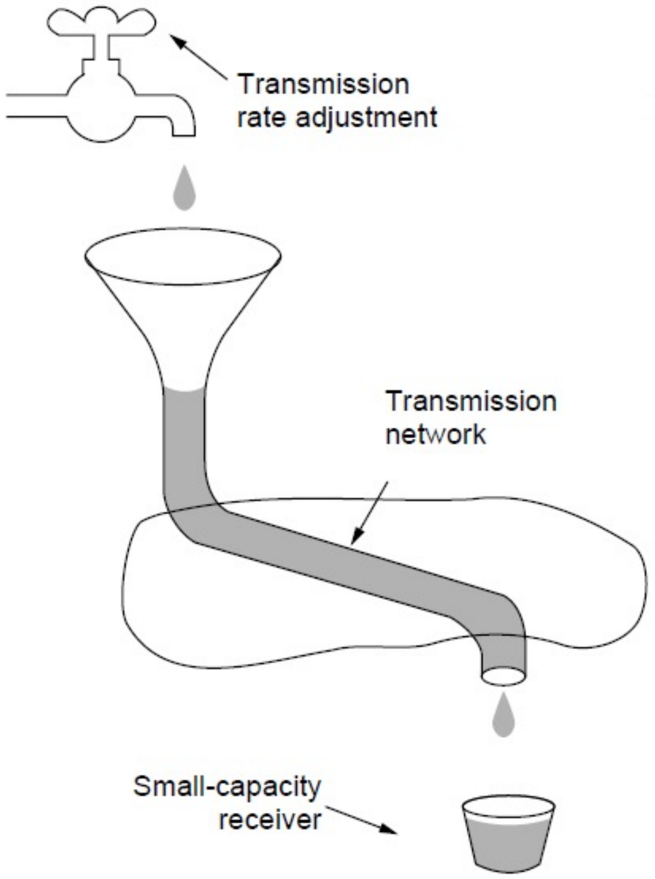
Max-min bandwidth allocation for four flows

Desirable Bandwidth Allocation (3)



Changing bandwidth allocation over time

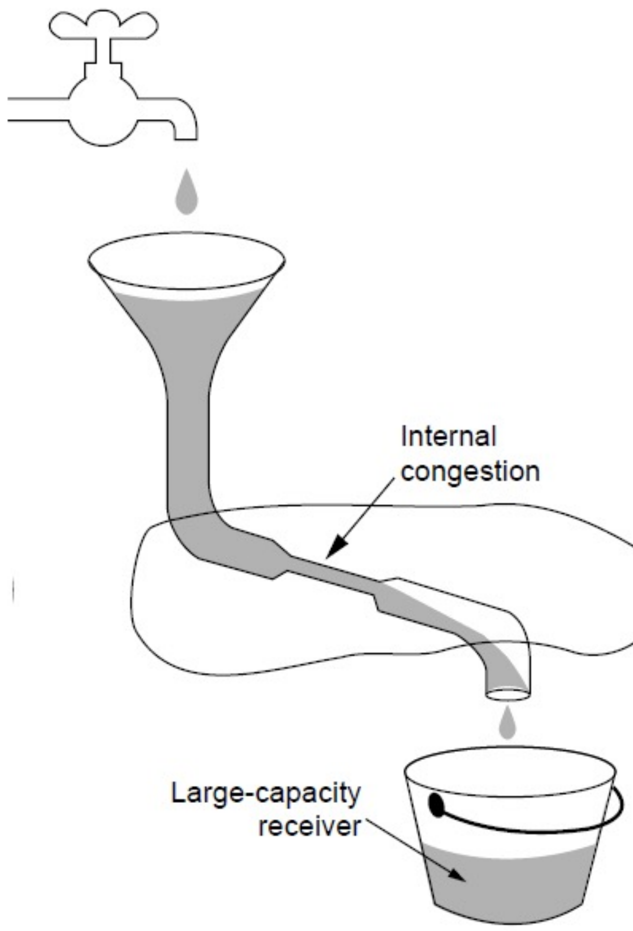
Regulating the Sending Rate (1)



not about congestion
its flow control

A fast network feeding a low-capacity receiver

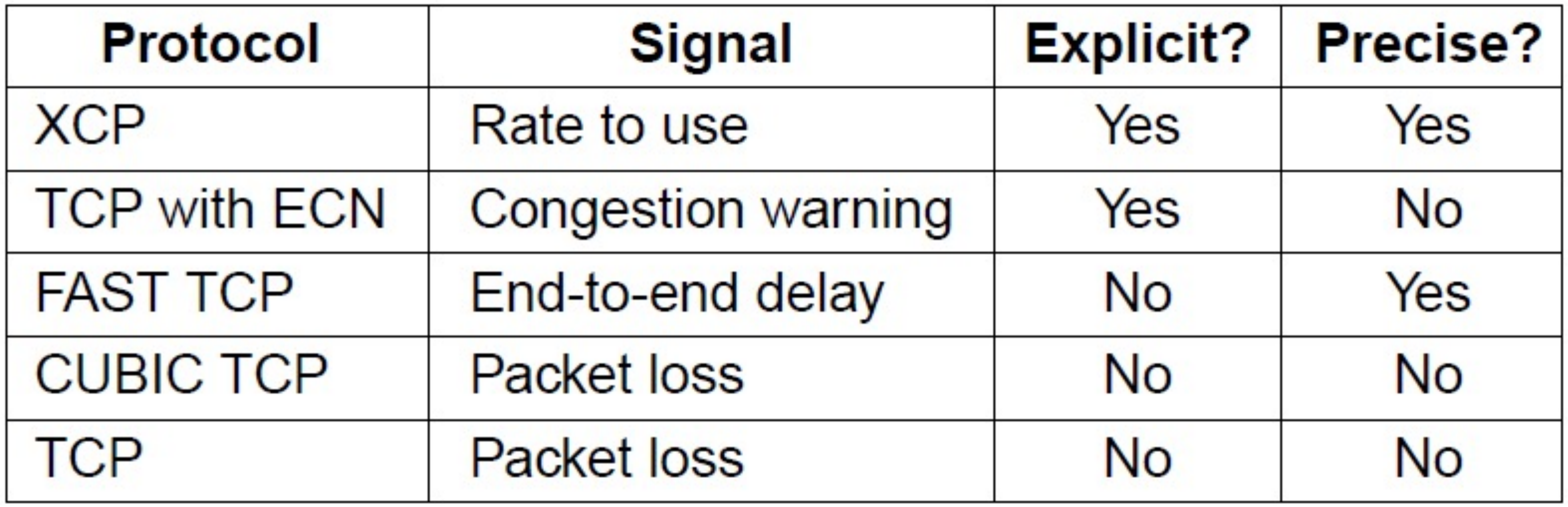
Regulating the Sending Rate (2)



its congestion bottleneck is there

A slow network feeding a high-capacity receiver

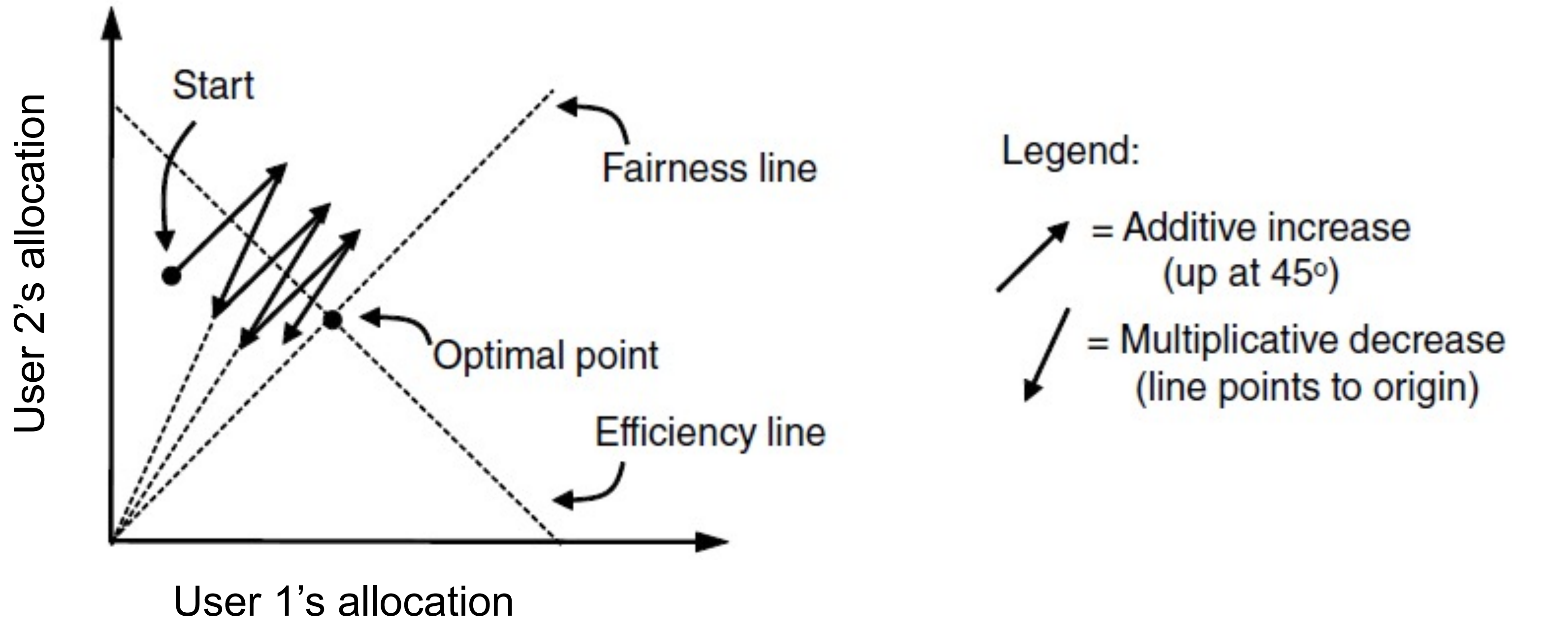
Regulating the Sending Rate (3)



Protocol	Signal	Explicit?	Precise?
XCP	Rate to use	Yes	Yes
TCP with ECN	Congestion warning	Yes	No
FAST TCP	End-to-end delay	No	Yes
CUBIC TCP	Packet loss	No	No
TCP	Packet loss	No	No

Some congestion control protocols

Regulating the Sending Rate (4)

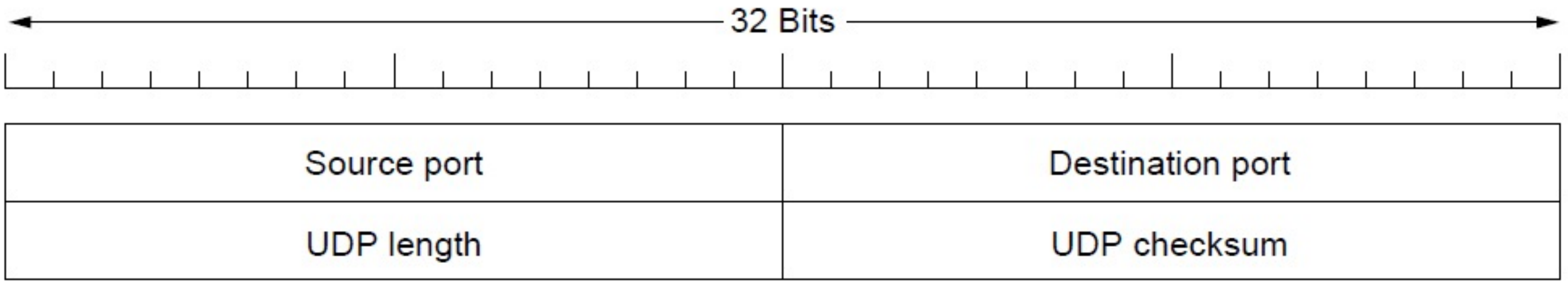


Additive Increase Multiplicative Decrease (AIMD) control law.

The Internet Transport Protocols: UDP

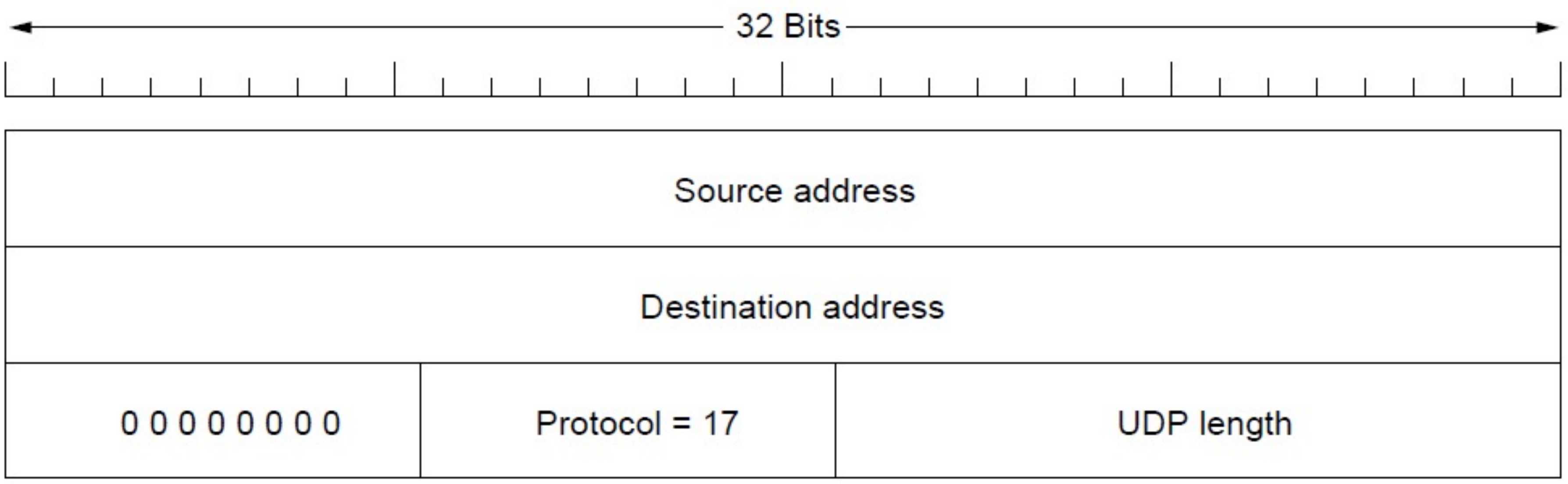
- Introduction to UDP
- Remote Procedure Call
- Real-Time Transport

Introduction to UDP (1)



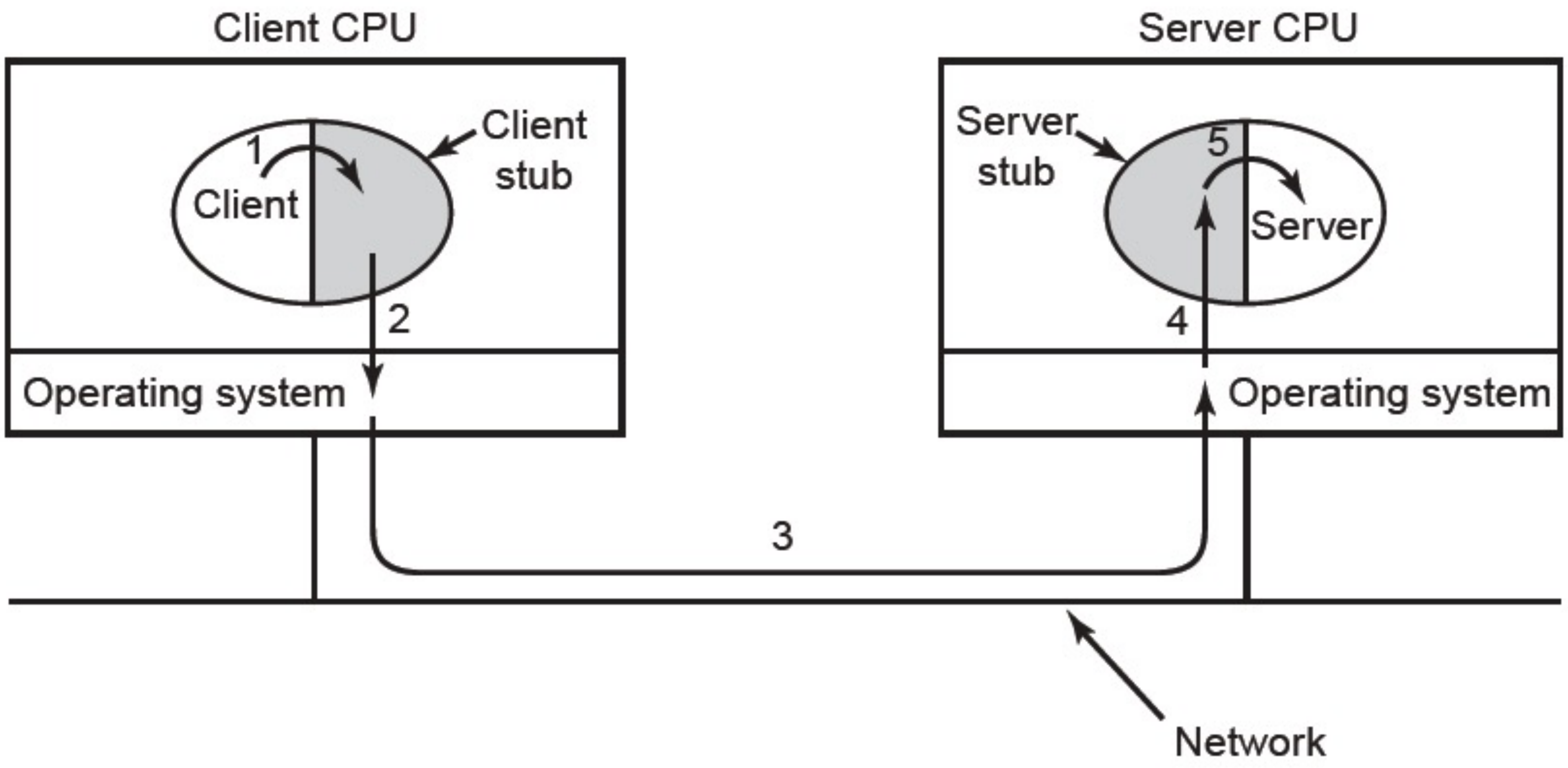
The UDP header.

Introduction to UDP (2)



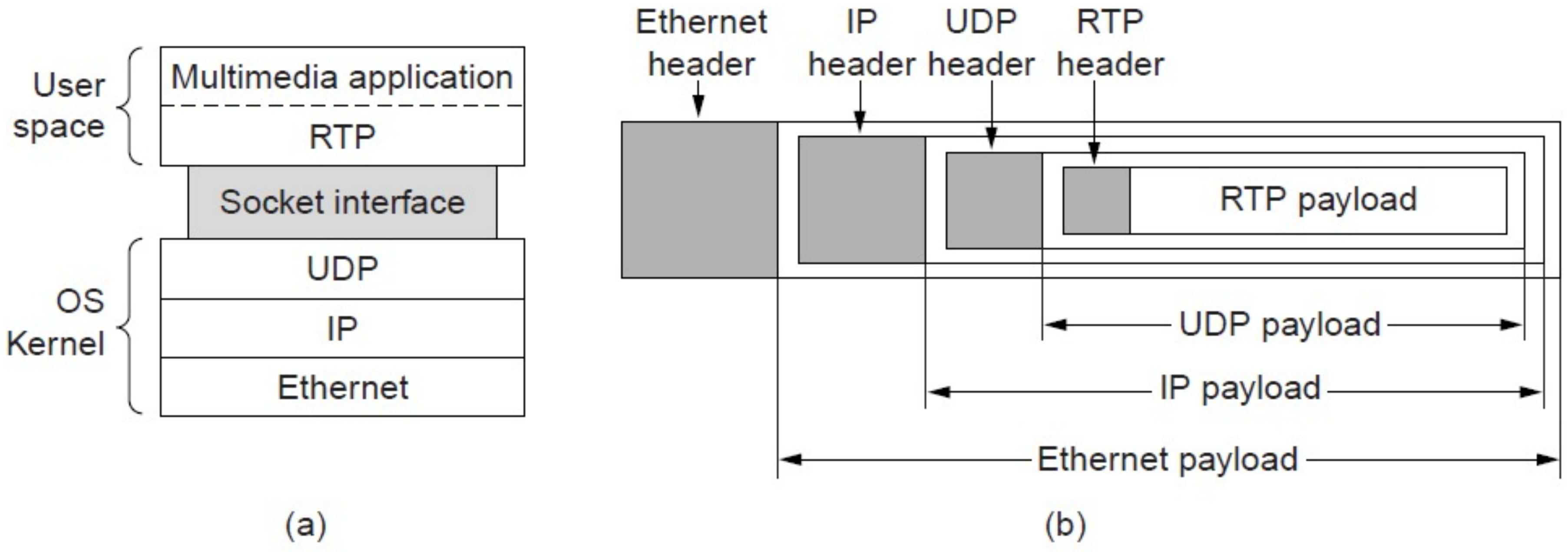
The IPv4 pseudoheader included in the UDP checksum.

Remote Procedure Call



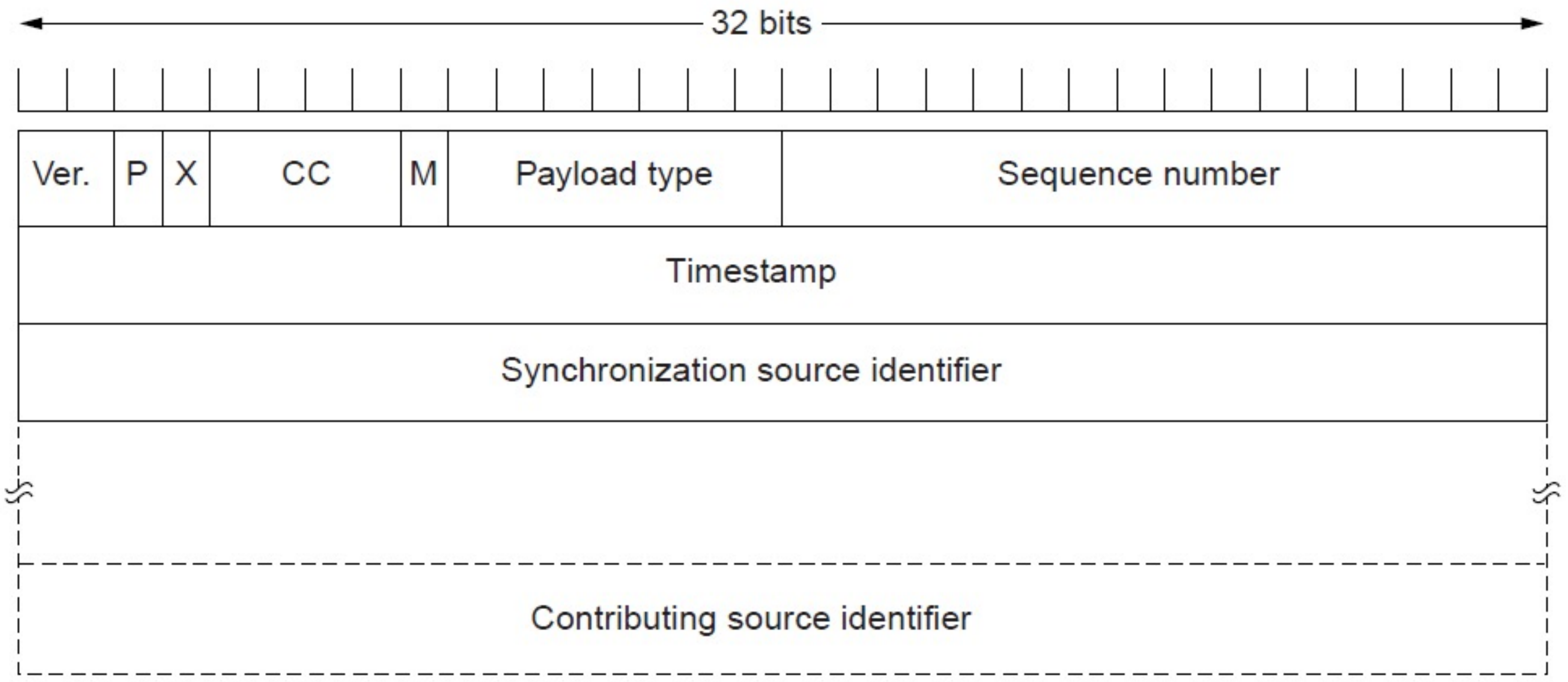
Steps in making a remote procedure call. The stubs are shaded.

Real-Time Transport (1)



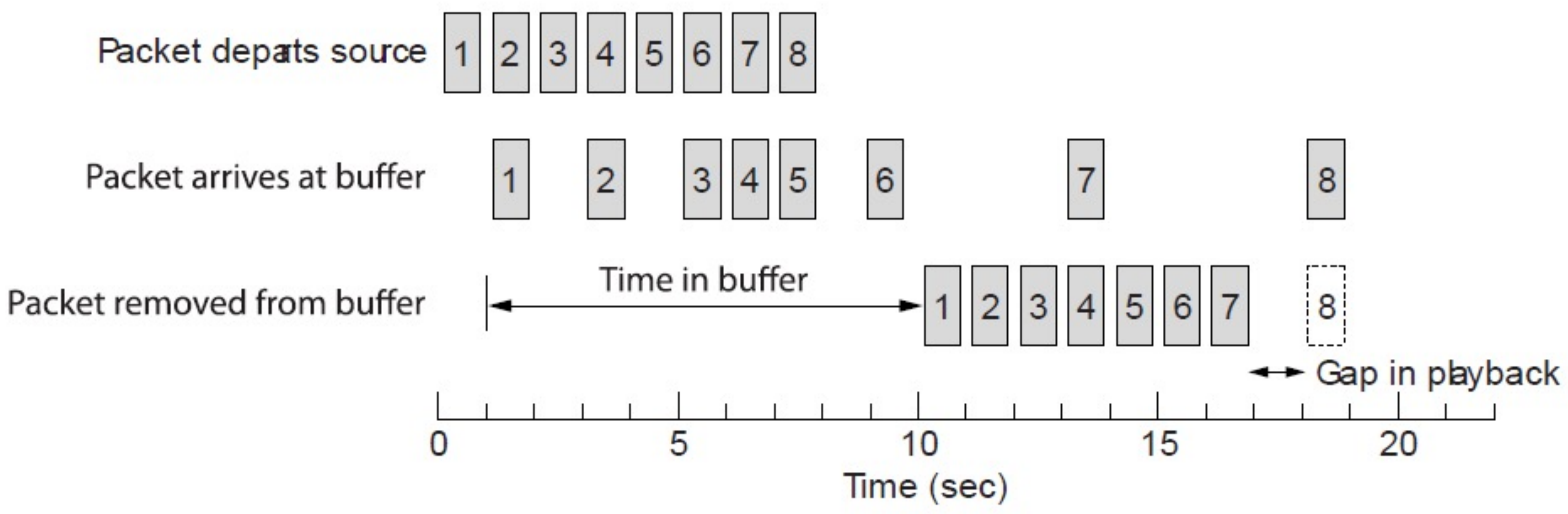
(a) The position of RTP in the protocol stack. (b) Packet nesting.

Real-Time Transport (2)


32 bits
Ver.
P
X
CC
M
Payload type
Sequence number
Timestamp
Synchronization source identifier
Contributing source identifier

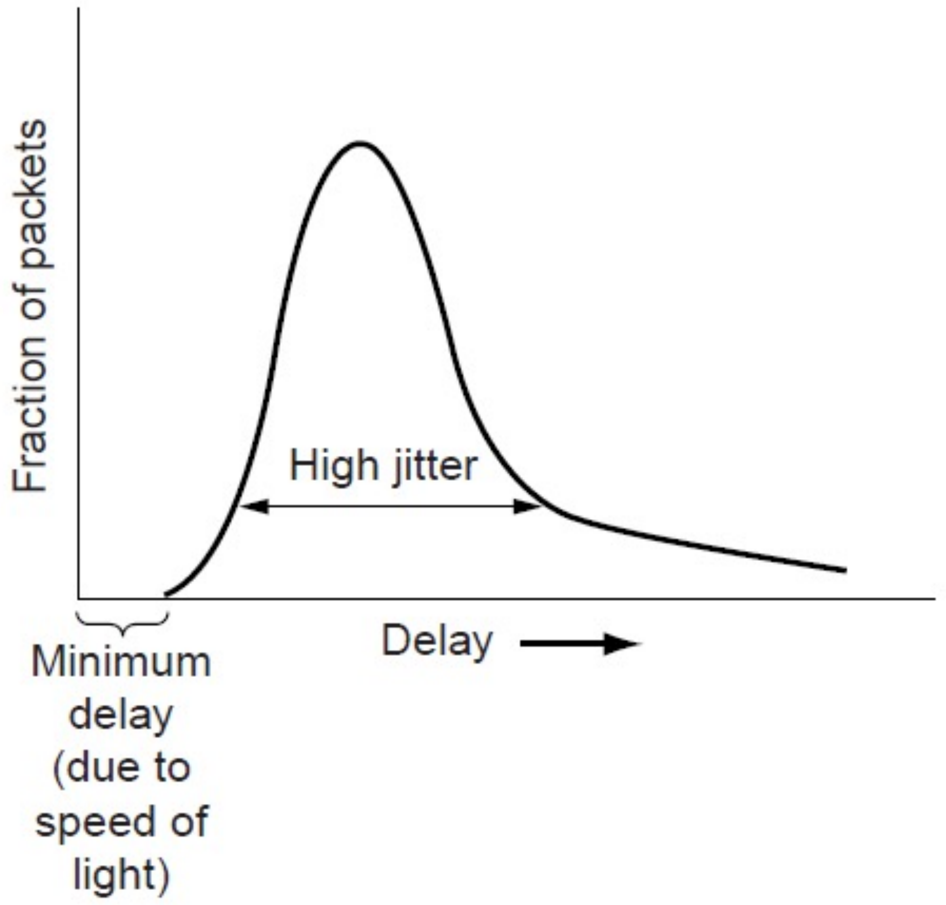
The RTP header

Real-Time Transport (3)



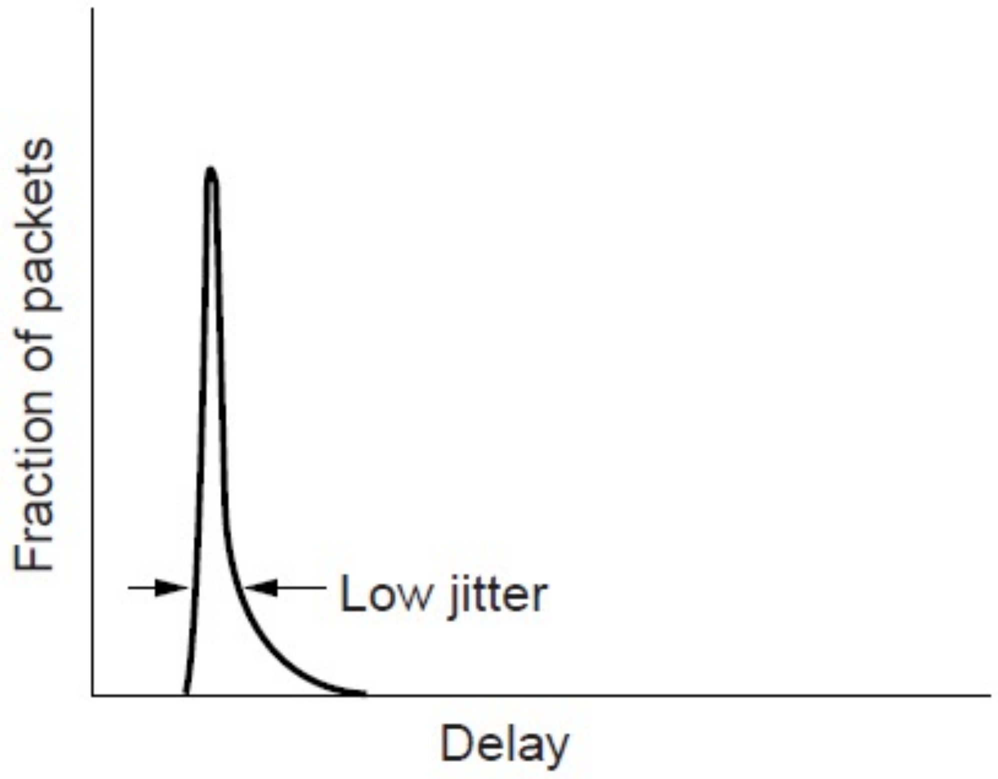
Smoothing the output stream by buffering packets

Real-Time Transport (3)



High jitter

Real-Time Transport (4)



Low jitter

The Internet Transport Protocols: TCP (1)

- Introduction to TCP
- The TCP service model
- The TCP protocol
- The TCP segment header
- TCP connection establishment
- TCP connection release

The Internet Transport Protocols: TCP (2)

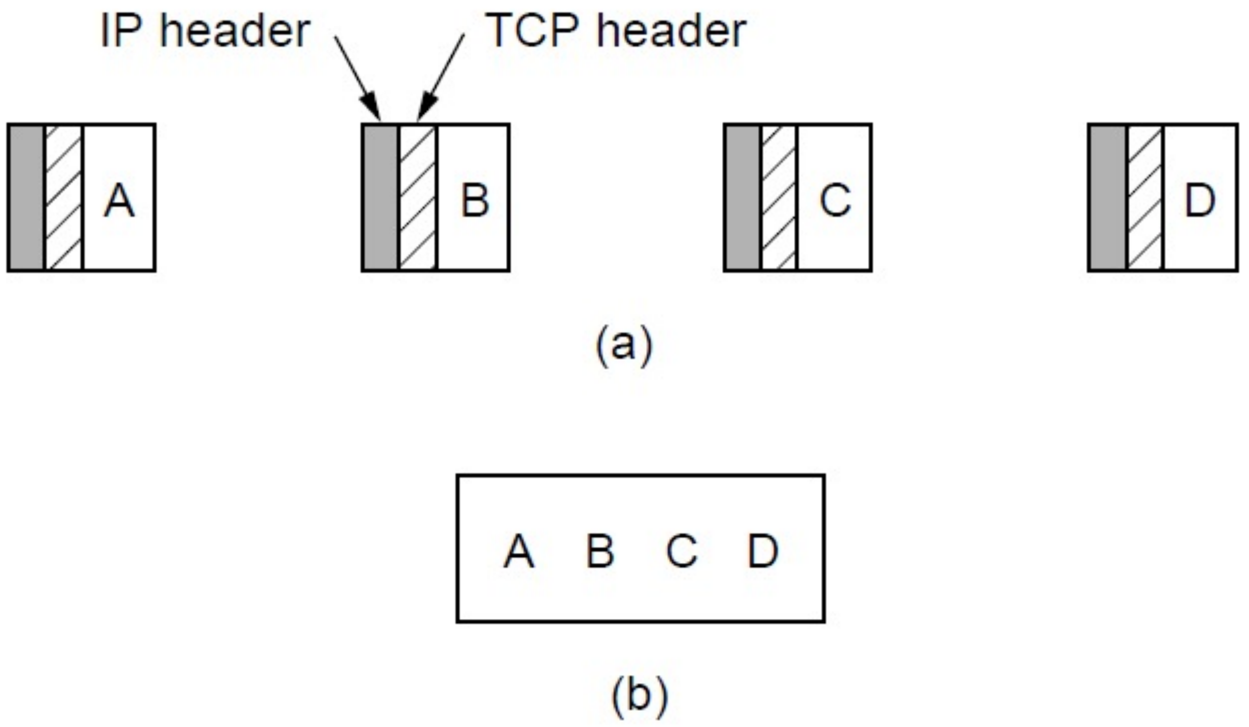
- TCP connection management modeling
- TCP sliding window
- TCP timer management
- TCP congestion control
- TCP futures

The TCP Service Model (1)

Port	Protocol	Use
20, 21	FTP	File transfer
22	SSH	Remote login, replacement for Telnet
25	SMTP	Email
80	HTTP	World Wide Web
110	POP-3	Remote email access
143	IMAP	Remote email access
443	HTTPS	Secure Web (HTTP over SSL/TLS)
543	RTSP	Media player control
631	IPP	Printer sharing

Some assigned ports

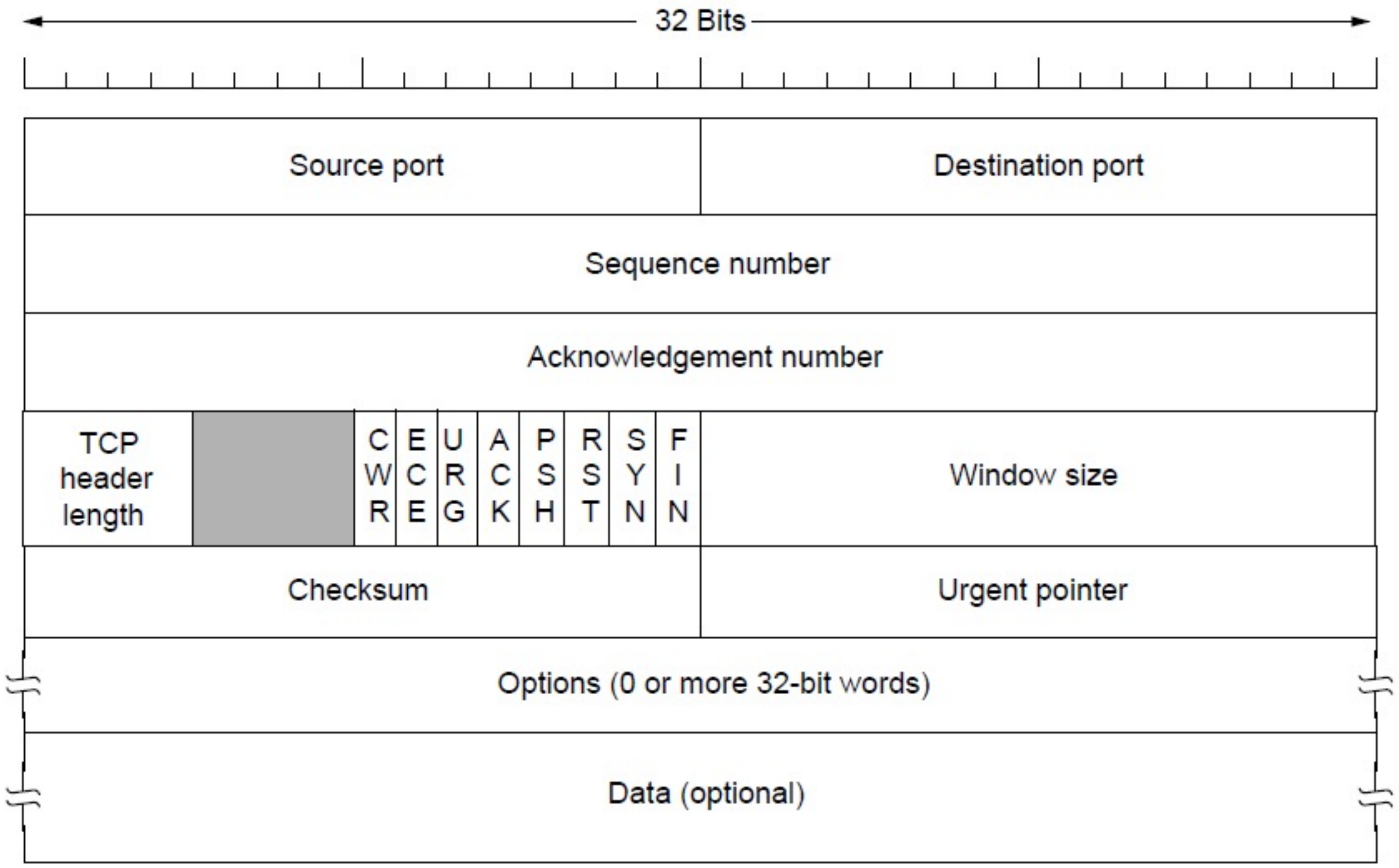
The TCP Service Model (2)



a) Four 512-byte segments sent as separate IP diagrams

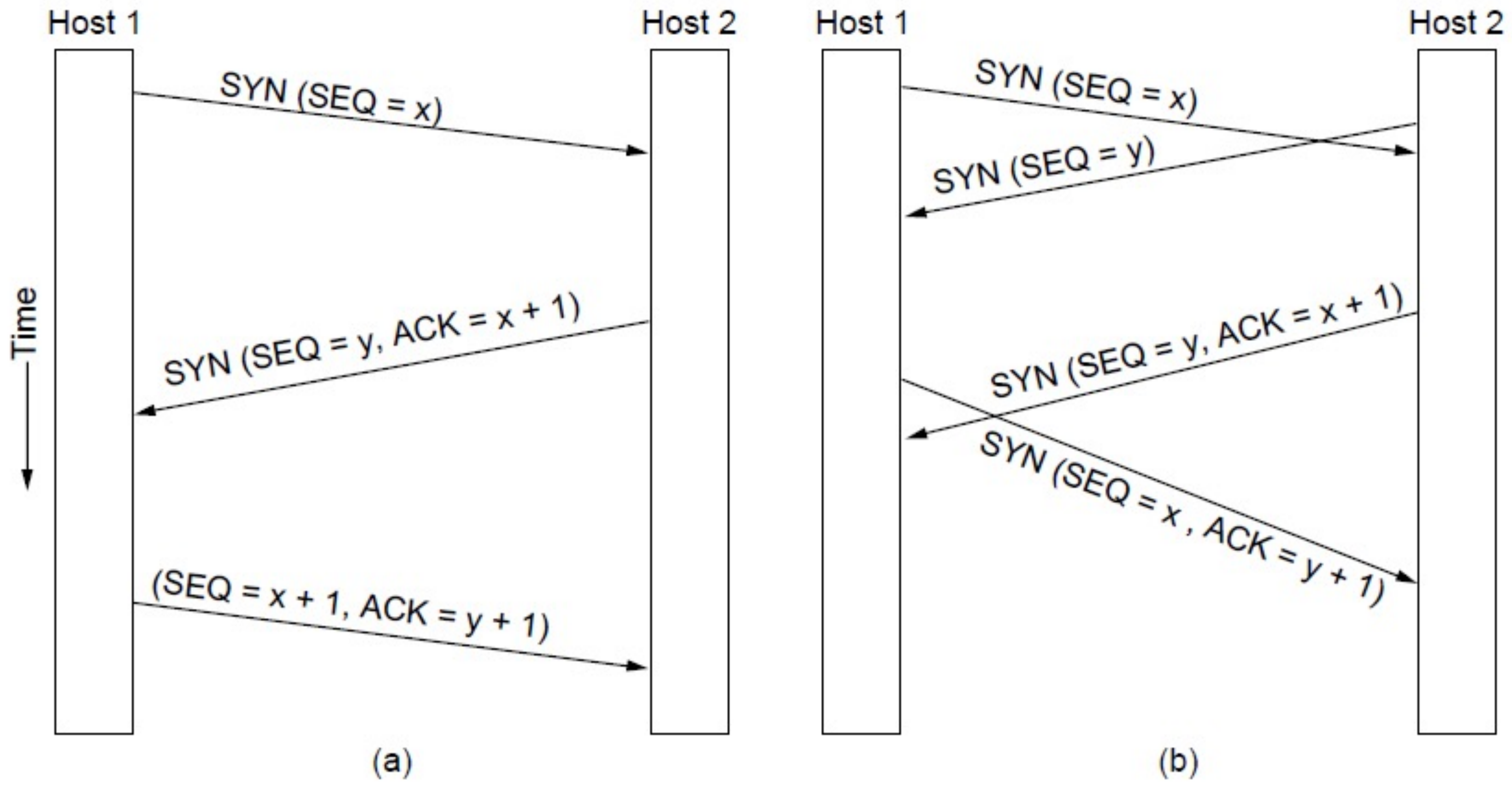
b) The 2048 bytes of data delivered to the application in a single READ call

The TCP Segment Header



The TCP header.

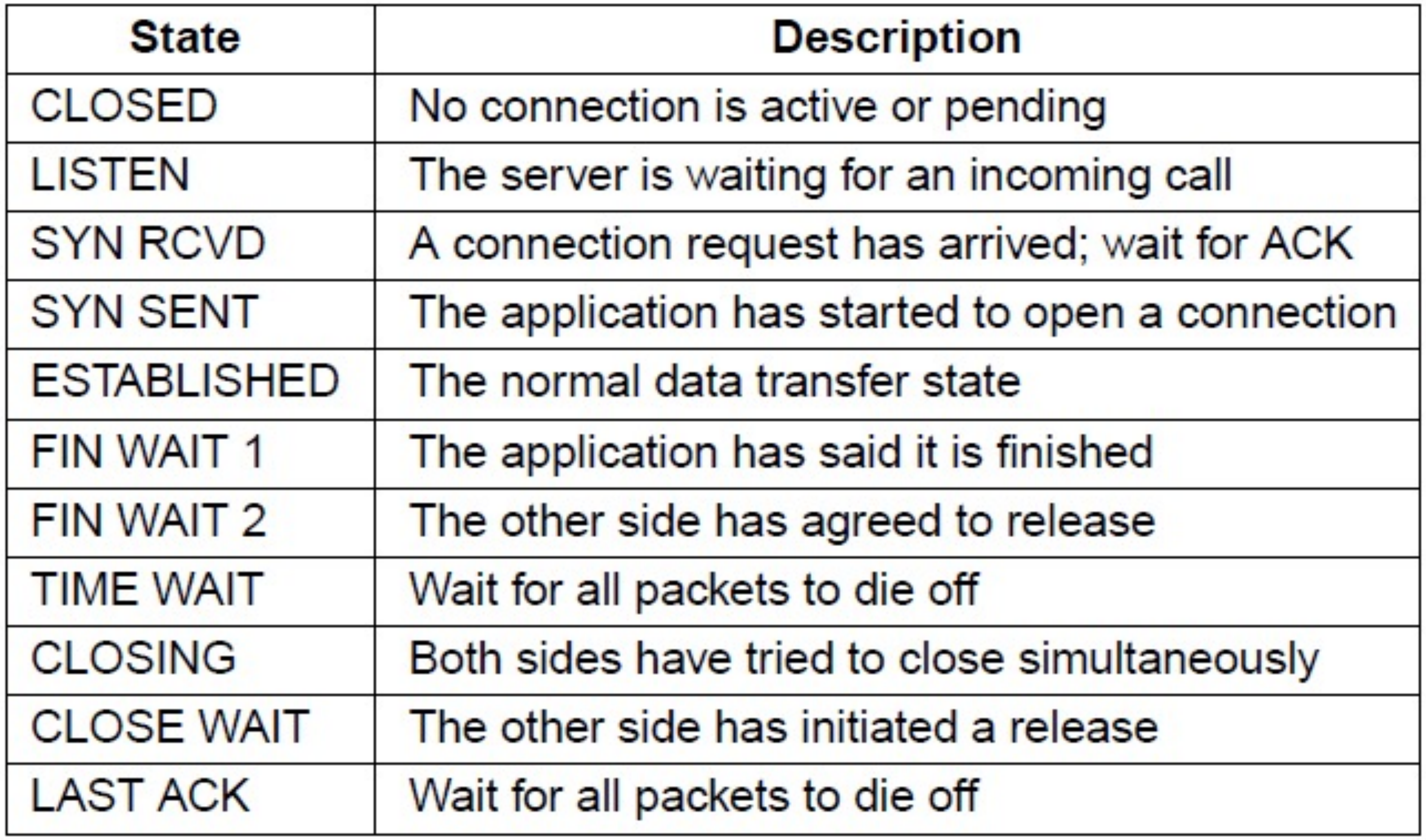
TCP Connection Establishment


Host 1
Host 2
Time
SYN (SEQ = x)
SYN (SEQ = y, ACK = x + 1)
(SEQ = x + 1, ACK = y + 1)
(a)
Host 1
Host 2
SYN (SEQ = x)
SYN (SEQ = y)
SYN (SEQ = y, ACK = x + 1)
SYN (SEQ = x, ACK = y + 1)
(b)

a) TCP connection establishment in the normal case.

b) Simultaneous connection establishment on both sides.

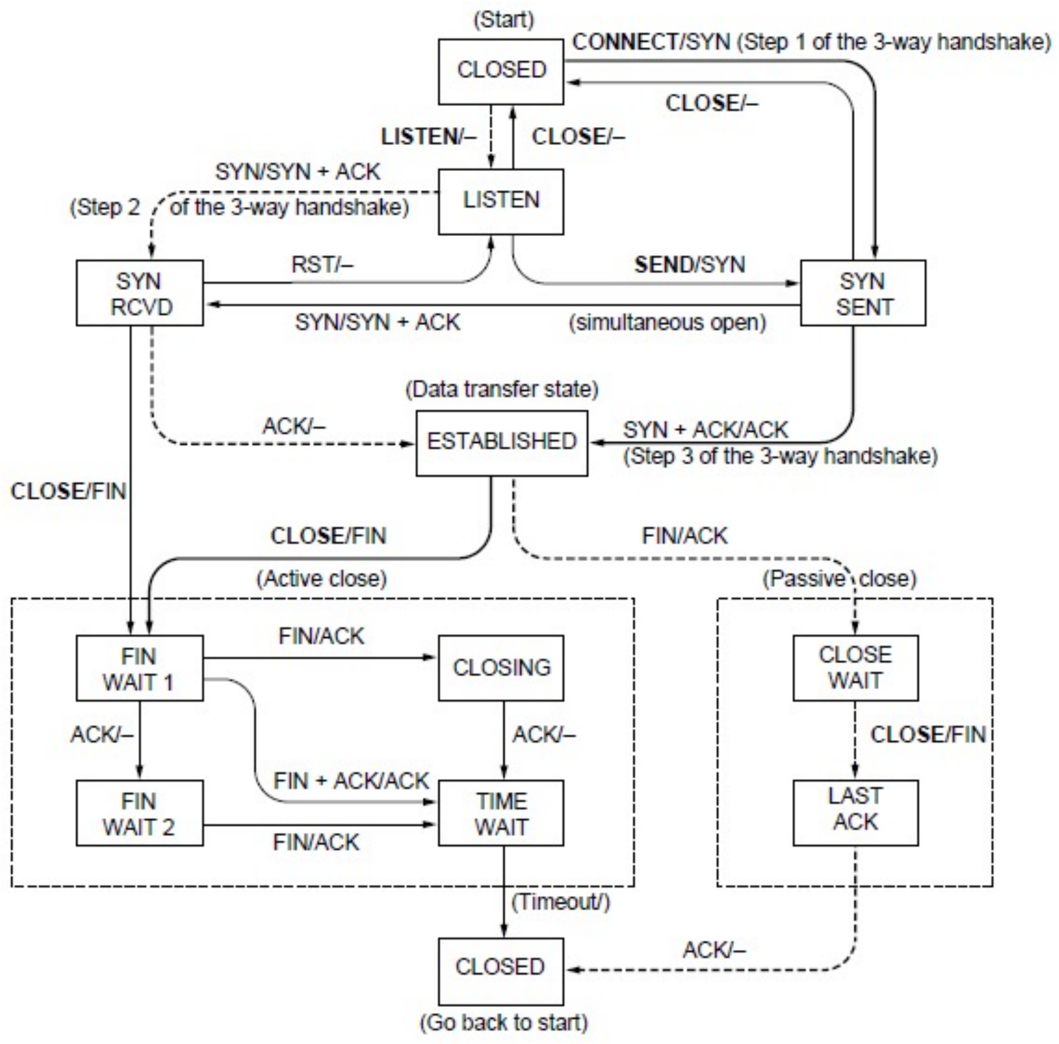
TCP Connection Management Modeling (1)



State	Description
CLOSED	No connection is active or pending
LISTEN	The server is waiting for an incoming call
SYN RCVD	A connection request has arrived; wait for ACK
SYN SENT	The application has started to open a connection
ESTABLISHED	The normal data transfer state
FIN WAIT 1	The application has said it is finished
FIN WAIT 2	The other side has agreed to release
TIME WAIT	Wait for all packets to die off
CLOSING	Both sides have tried to close simultaneously
CLOSE WAIT	The other side has initiated a release
LAST ACK	Wait for all packets to die off

The states used in the TCP connection management finite state machine.

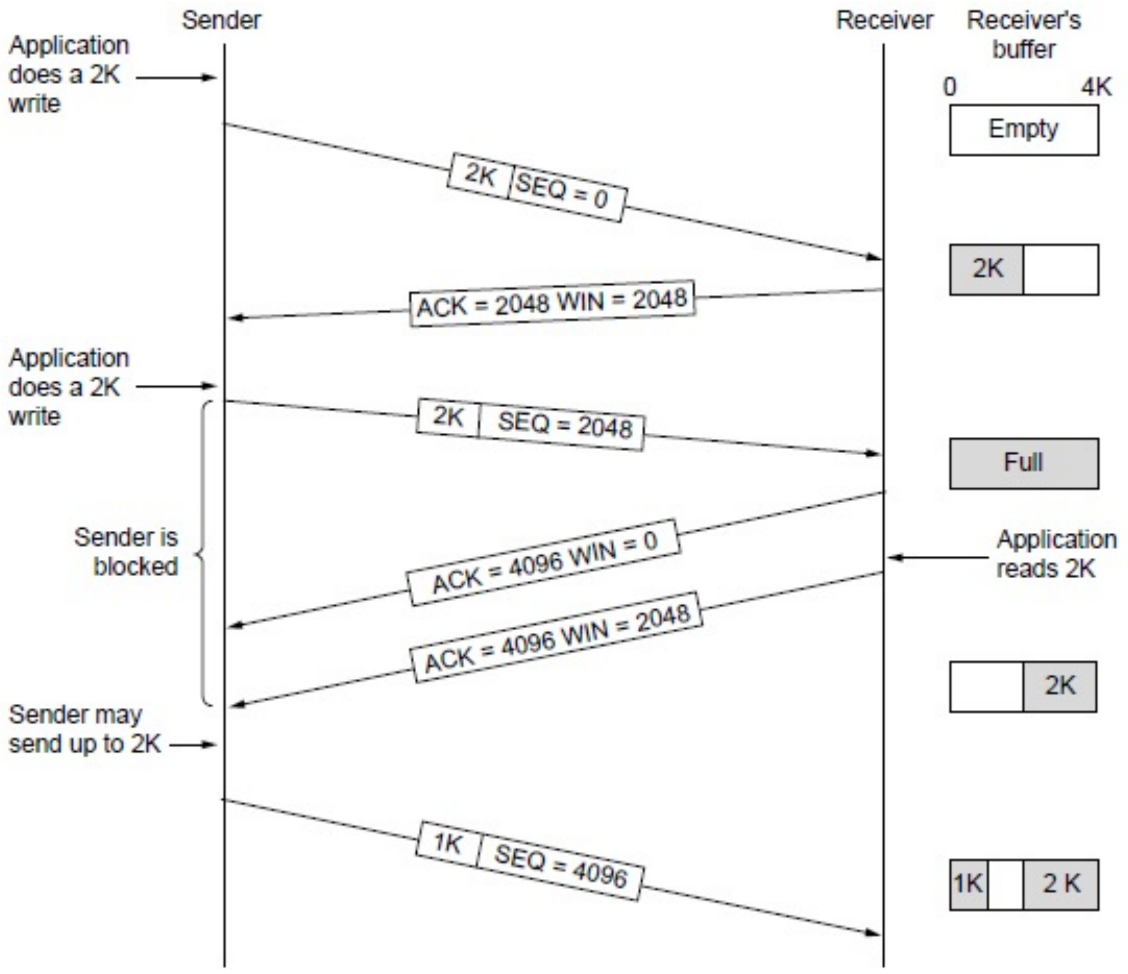
TCP Connection Management Modeling (2)



TCP connection management finite state machine.

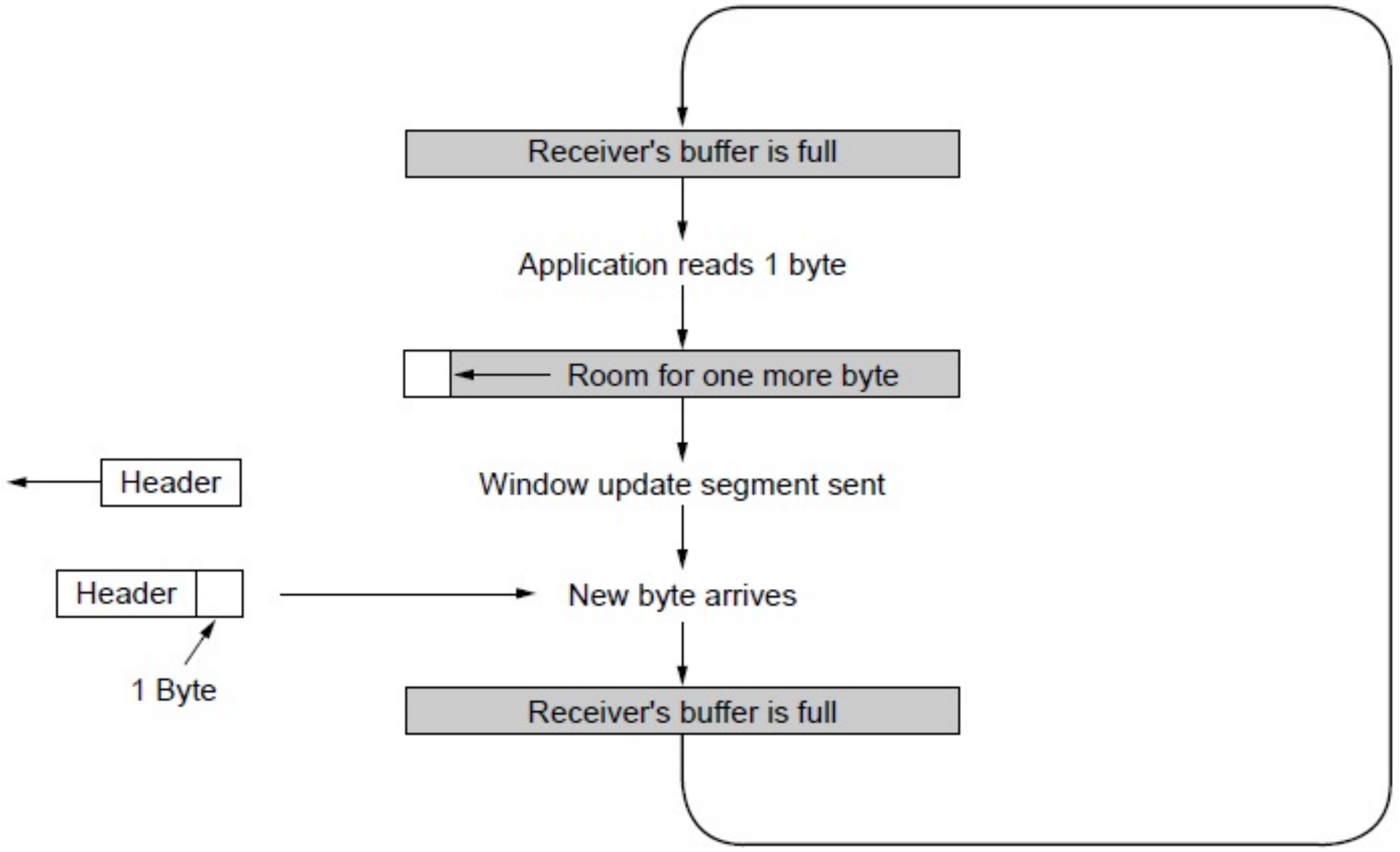
The heavy solid line is the normal path for a client. The heavy dashed line is the normal path for a server. The light lines are unusual events. Each transition is labeled by the event causing it and the action resulting from it, separated by a slash.

TCP Sliding Window (1)



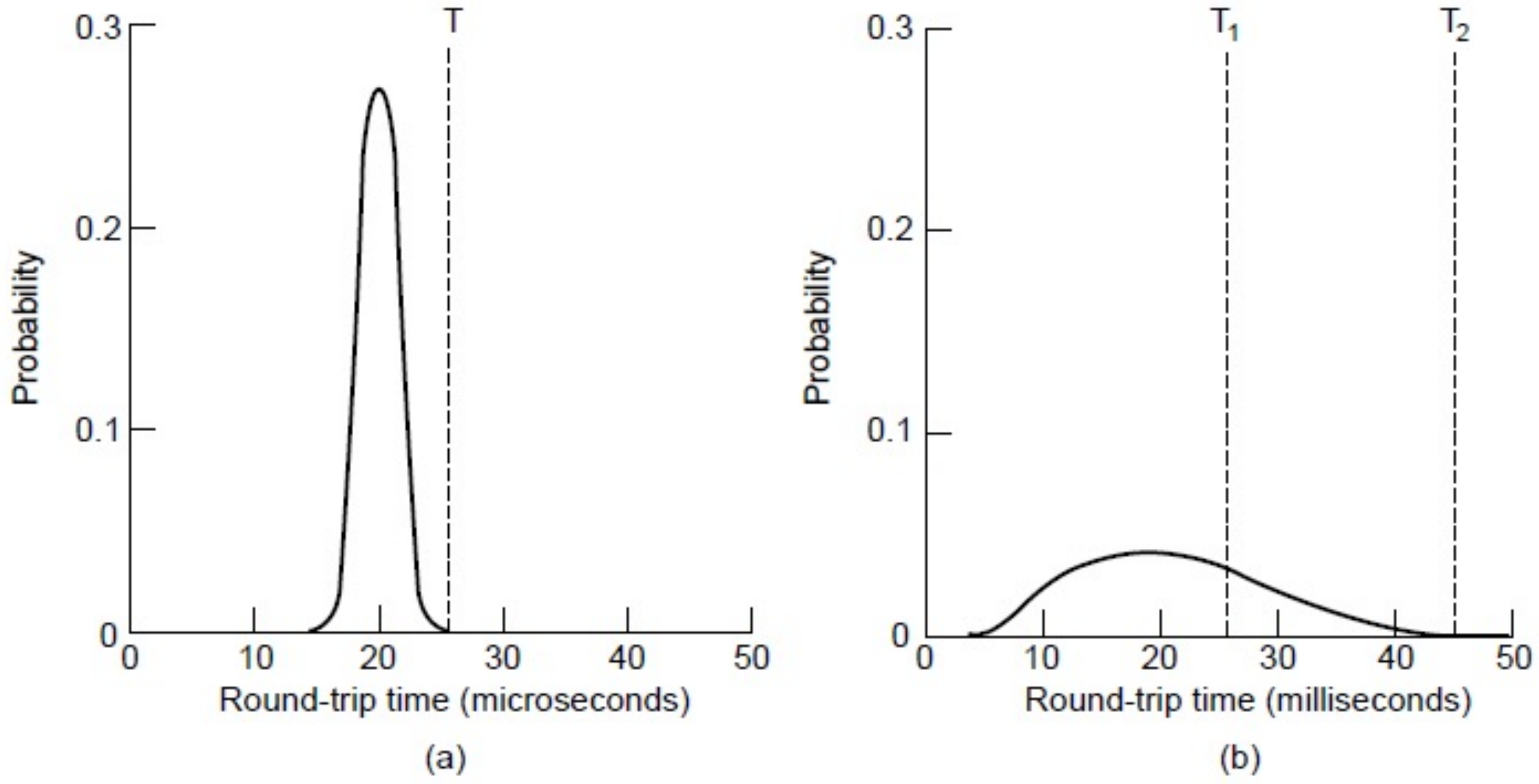
Window management in TCP

TCP Sliding Window (2)



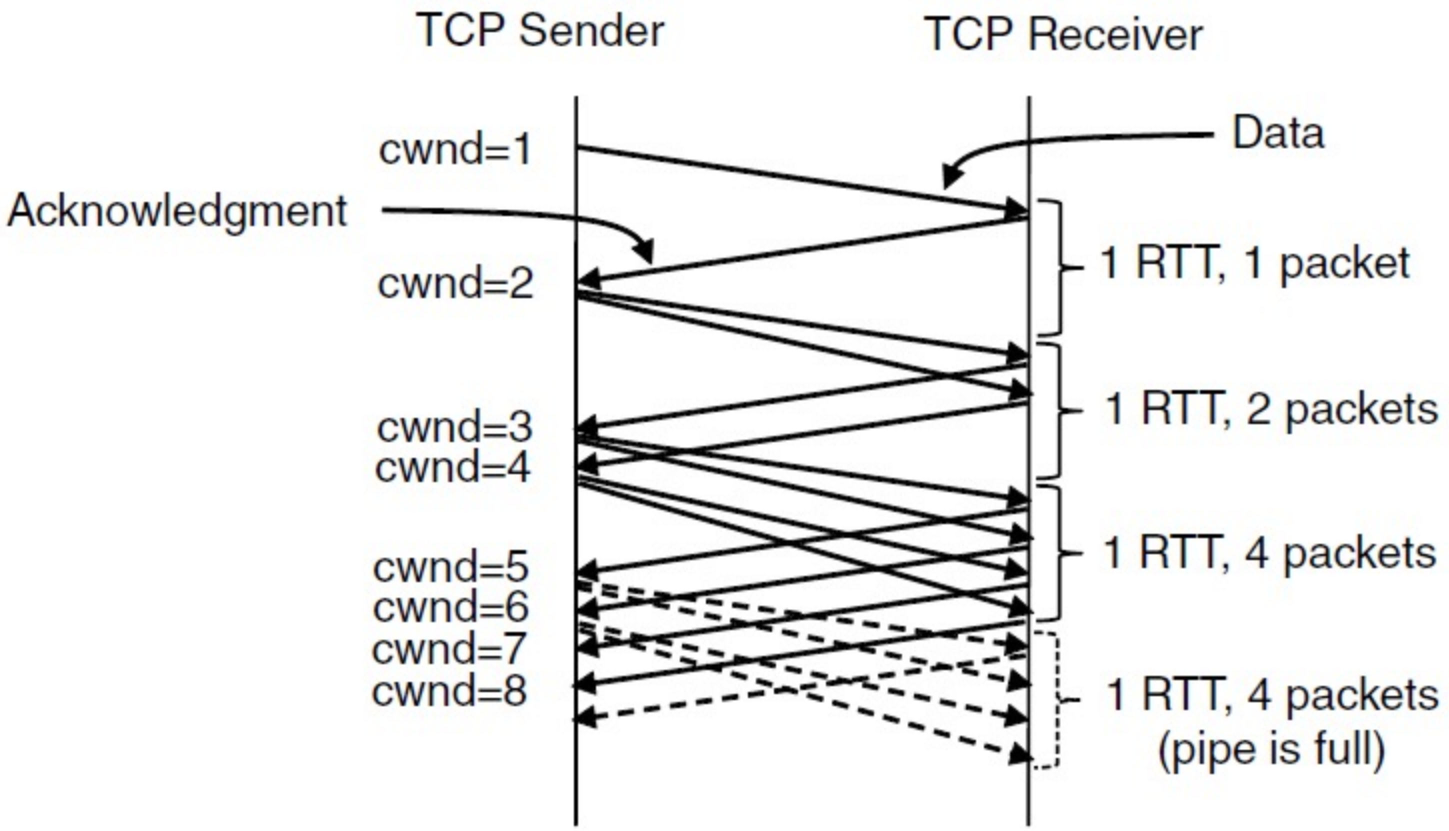
Silly window syndrome

TCP Timer Management



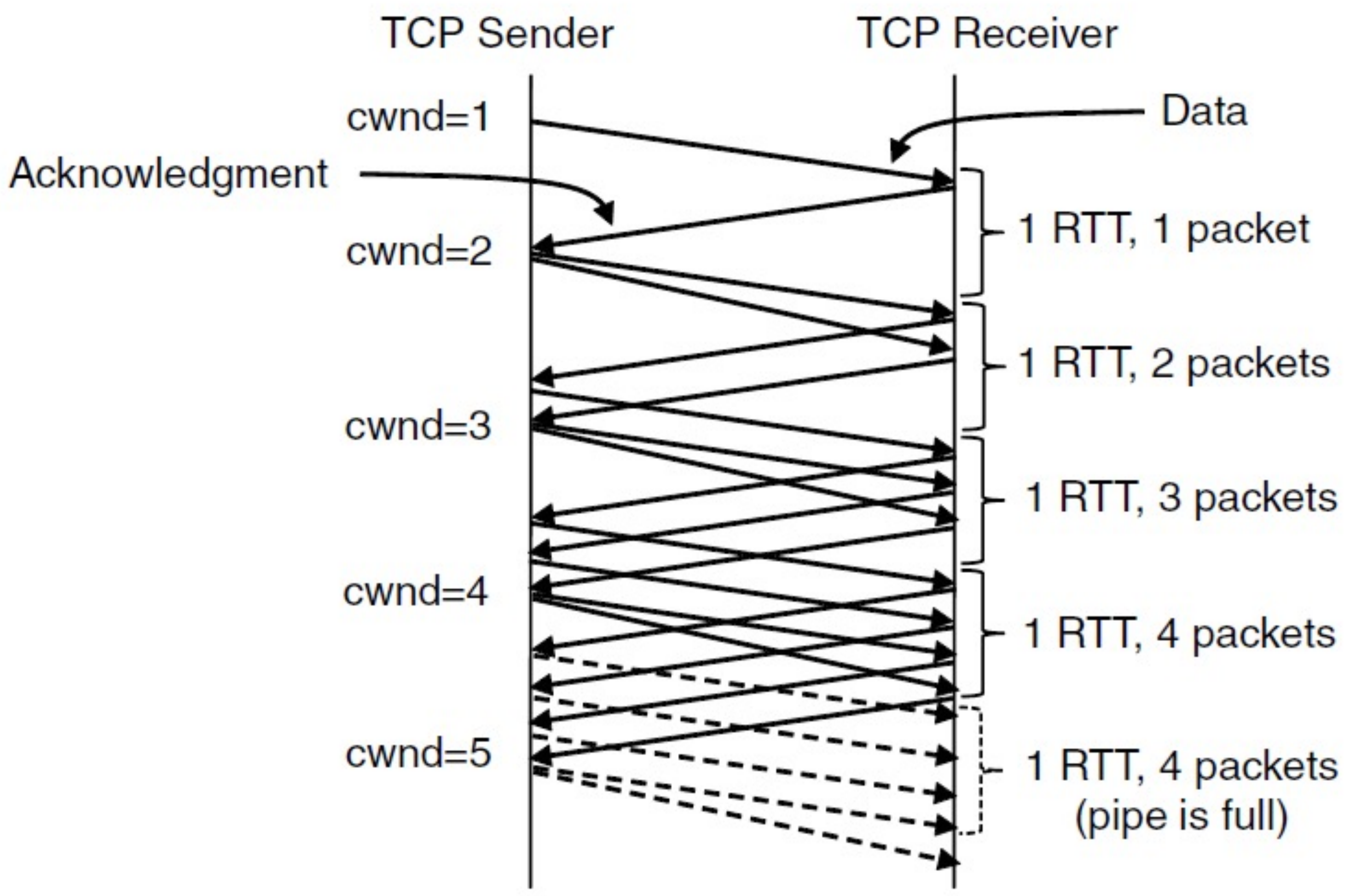
a) Probability density of acknowledgment arrival times in data link layer. (b) … for TCP

TCP Congestion Control (1)



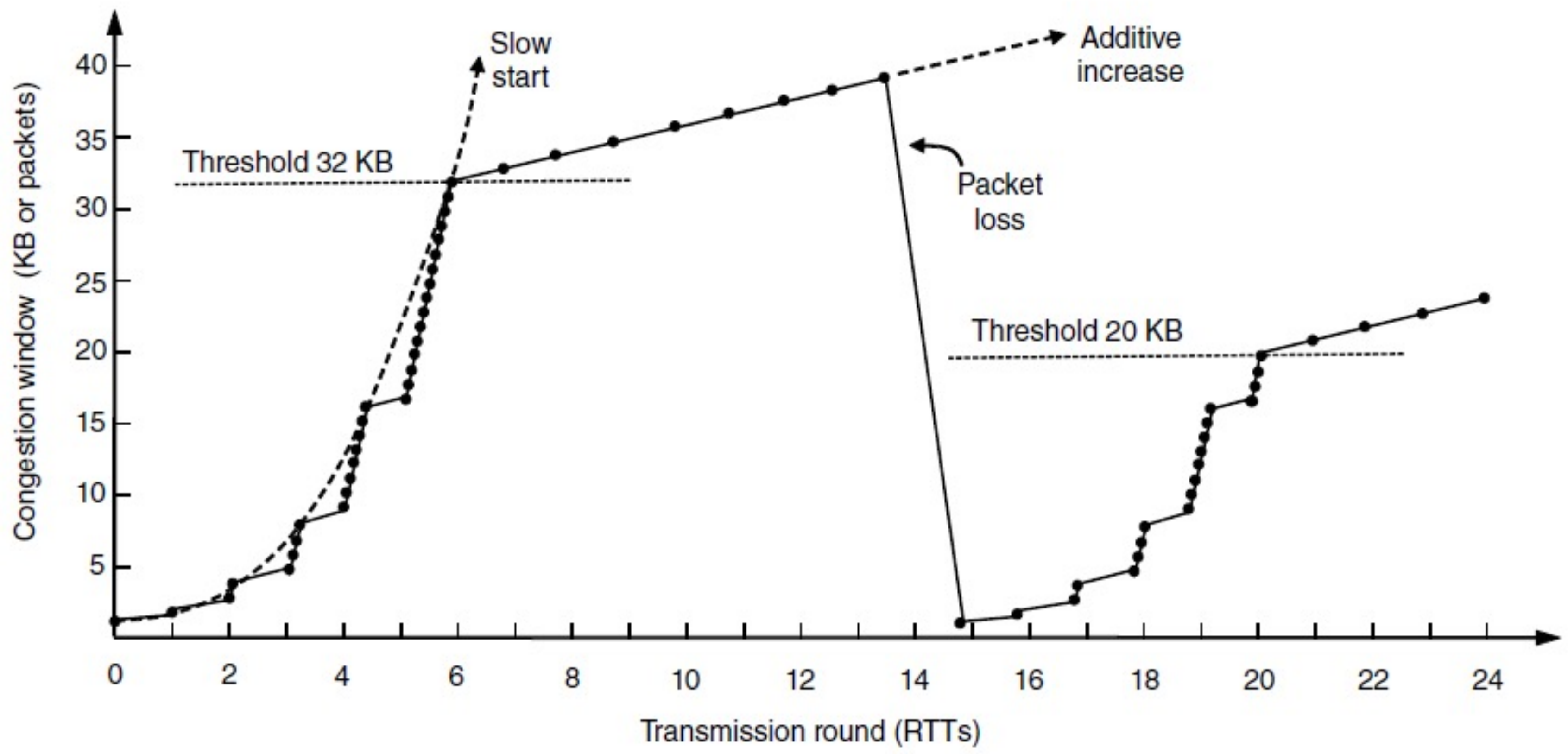
Slow start from an initial congestion window of 1 segment

TCP Congestion Control (2)



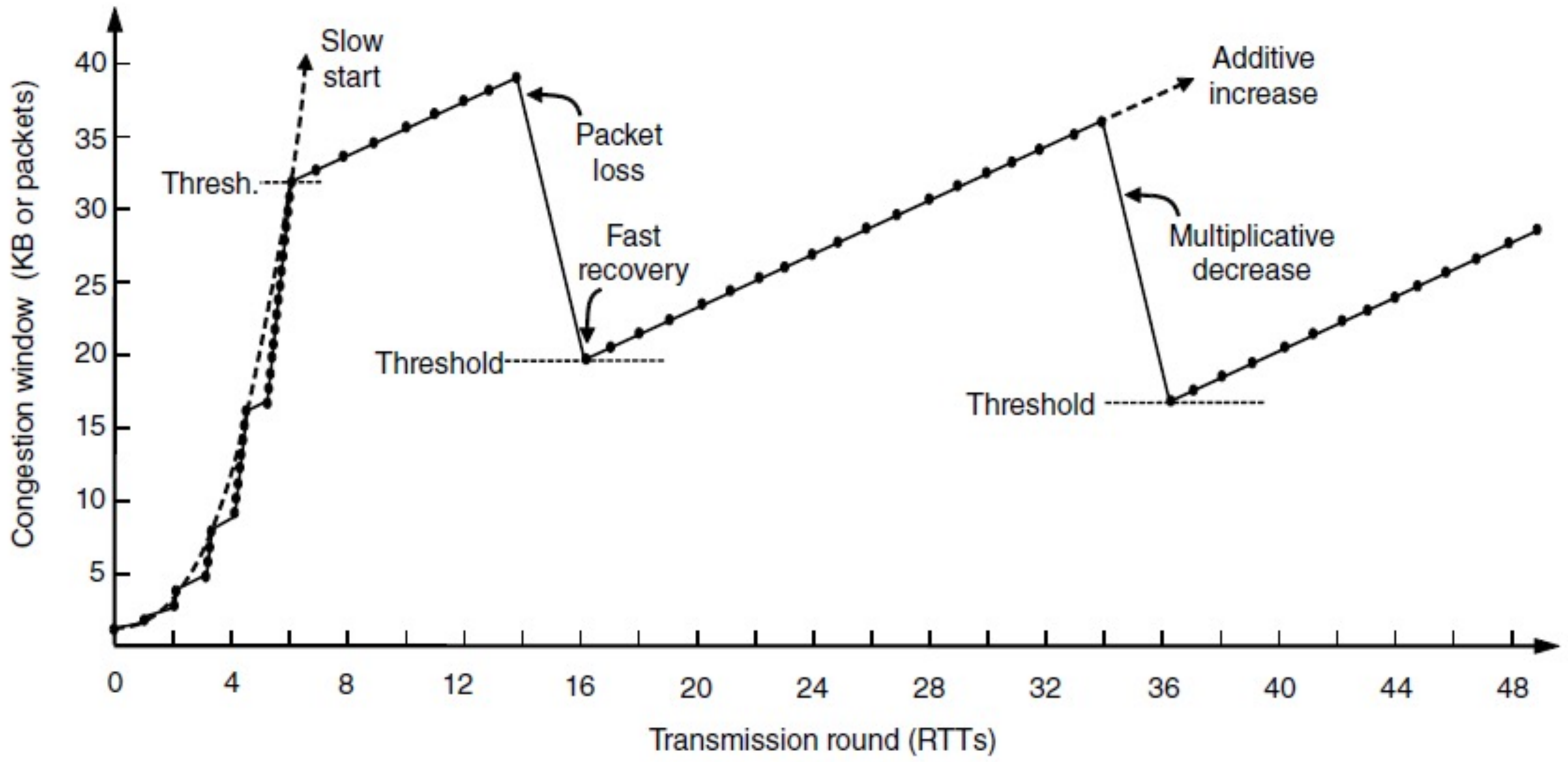
Additive increase from an initial congestion window of 1 segment.

TCP Congestion Control (3)



Slow start followed by additive increase in TCP Tahoe.

TCP Congestion Control (4)

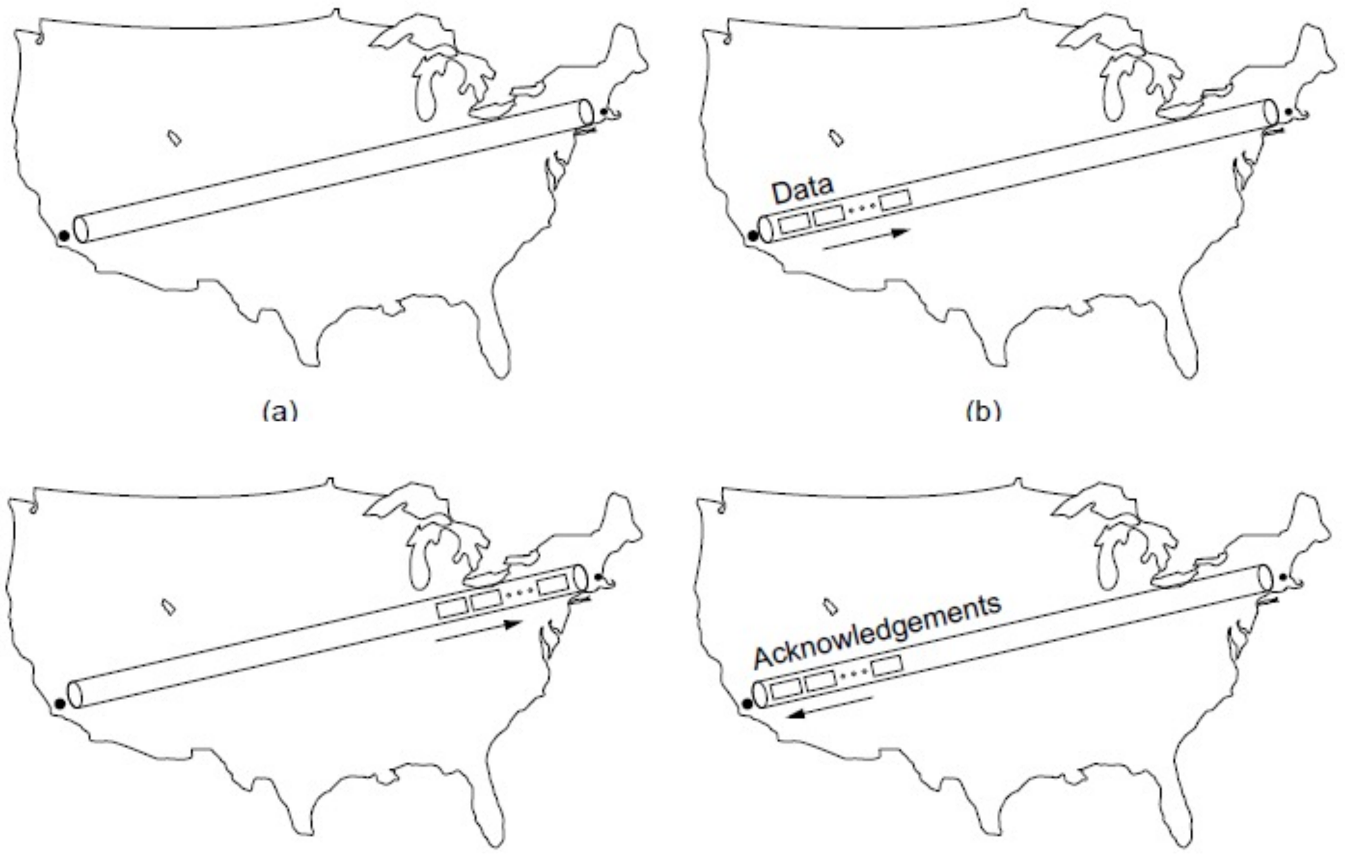


Fast recovery and the sawtooth pattern of TCP Reno.

Performance Issues

- Performance problems in computer networks
- Network performance measurement
- System design for better performance
- Fast TPDU processing
- Protocols for high-speed networks

Performance Problems in Computer Networks



The state of transmitting one megabit from San Diego to Boston.
(a) At *t = 0.* (b) *After 500 μ sec.*
(c) *After 20 msec.* (d) *After 40 msec.*

Network Performance Measurement (1)

Steps to performance improvement

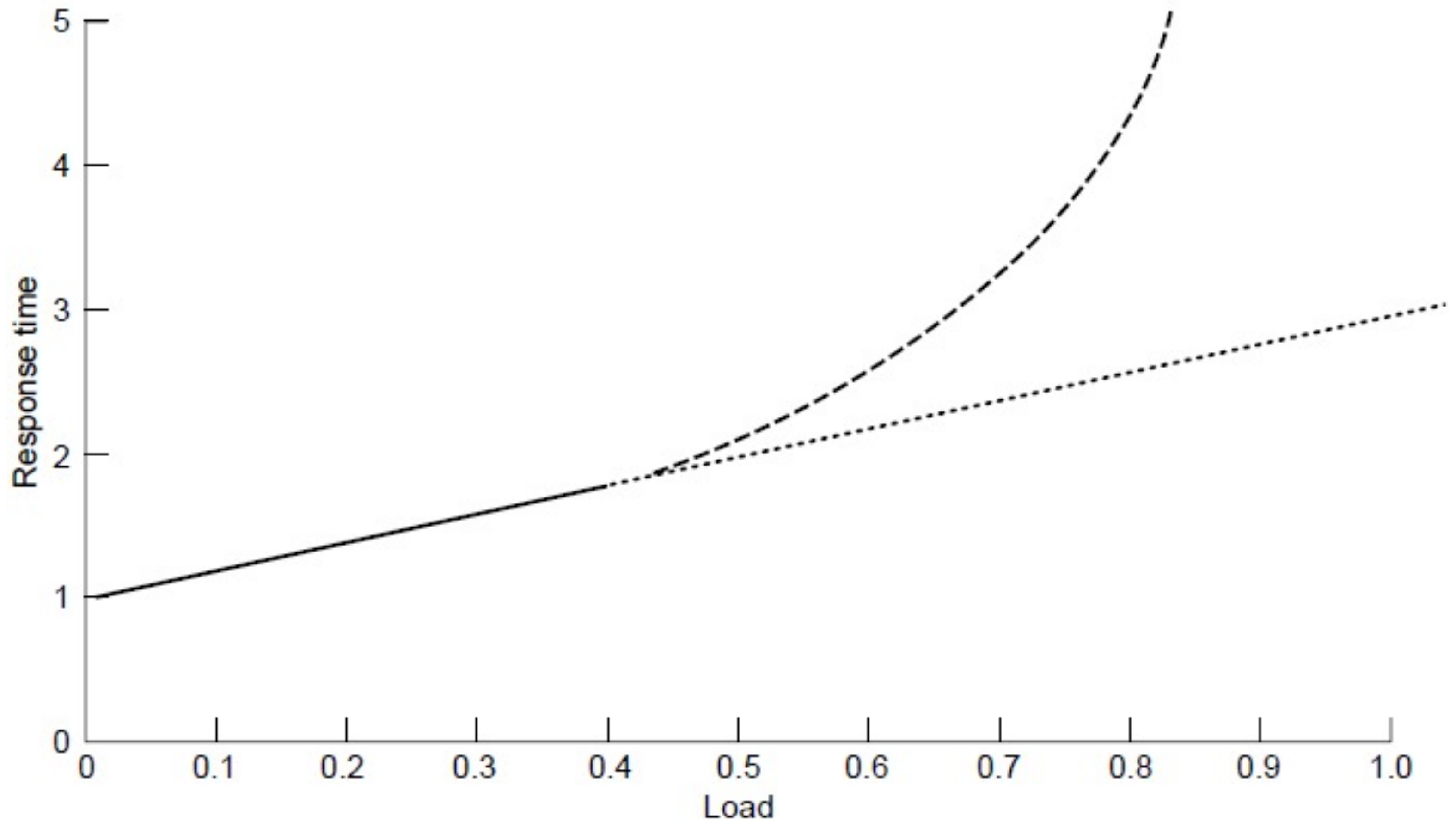
a) Measure relevant network parameters, performance.
b) Try to understand what is going on.
c) Change one parameter.

Network Performance Measurement (2)

Issues in measuring performance

- Sufficient sample size
- Representative samples
- Clock accuracy
- Measuring typical representative load
- Beware of caching
- Understand what you are measuring
- Extrapolate with care

Network Performance Measurement (3)



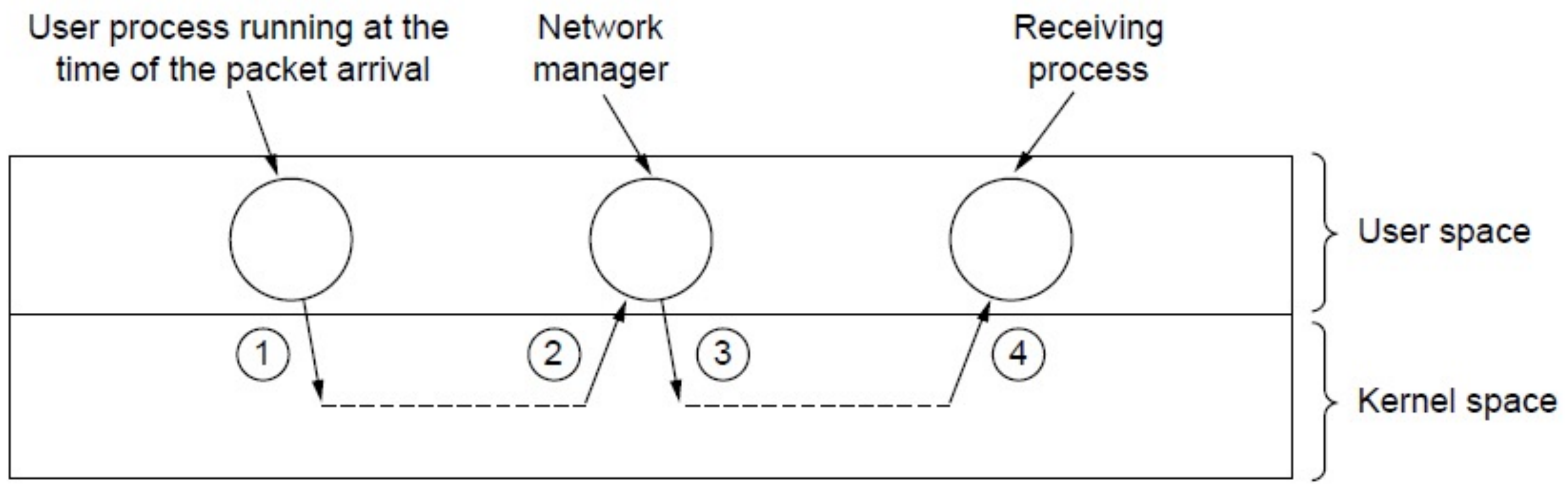
Response as a function of load.

System Design for Better Performance (1)

Rules of thumb

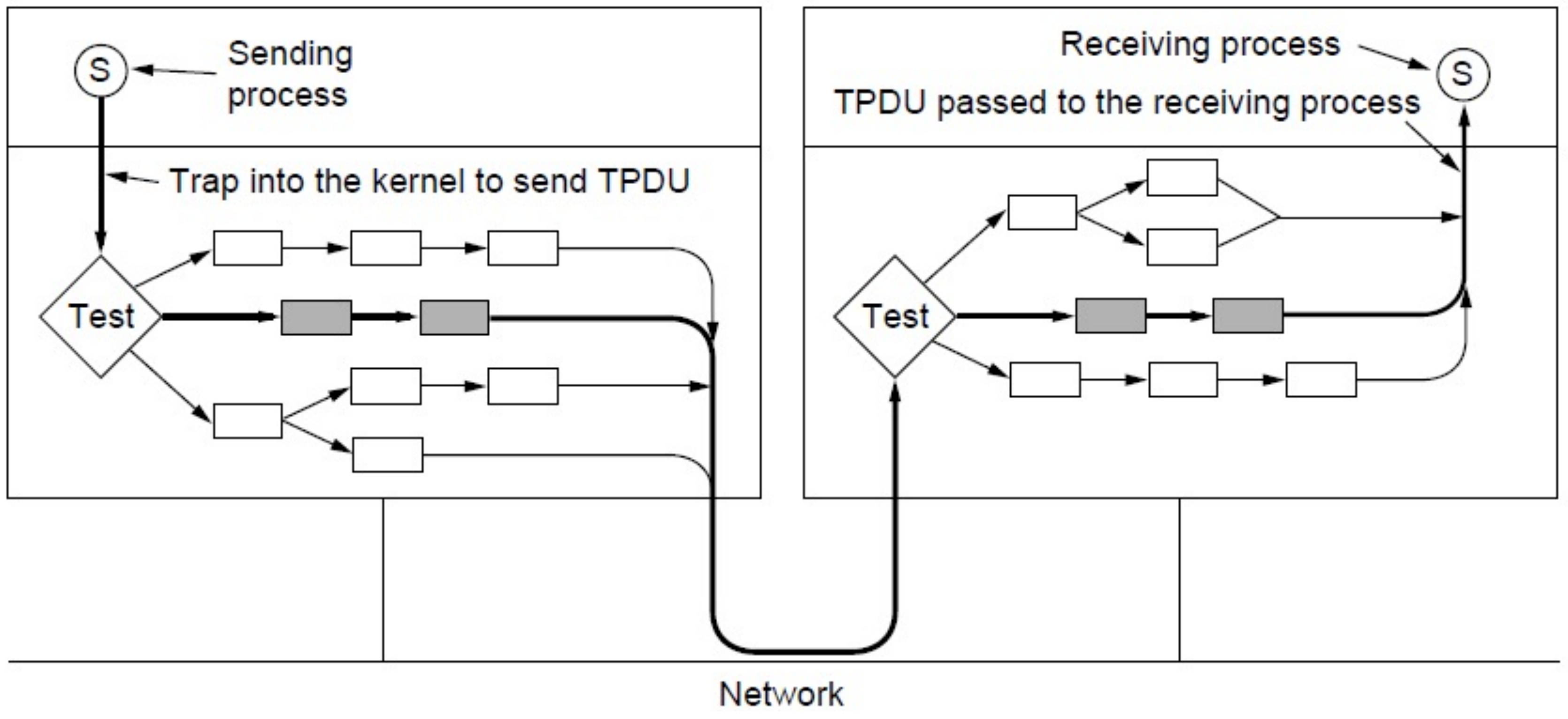
a) CPU speed more important than network speed
b) Reduce packet count to reduce software overhead
c) Minimize data touching
d) Minimize context switches
e) Minimize copying
f) You can buy more bandwidth but not lower delay
g) Avoiding congestion is better than recovering from it
h) Avoid timeouts

System Design for Better Performance (2)



Four context switches to handle one packet with a user-space network manager.

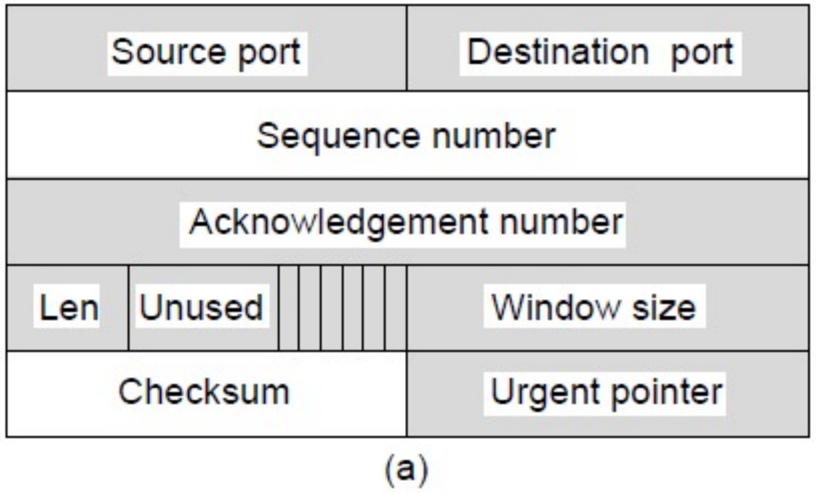
Fast TPDU Processing (1)

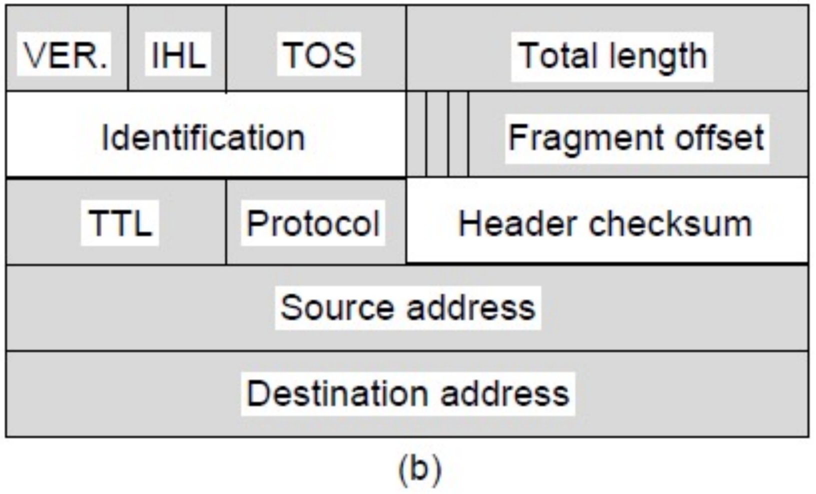


Sending process
S
Trap into the kernel to send TPDU
Test
Receiving process
S
TPDU passed to the receiving process
Test
Network

The fast path from sender to receiver is shown with a heavy line. The processing steps on this path are shaded.

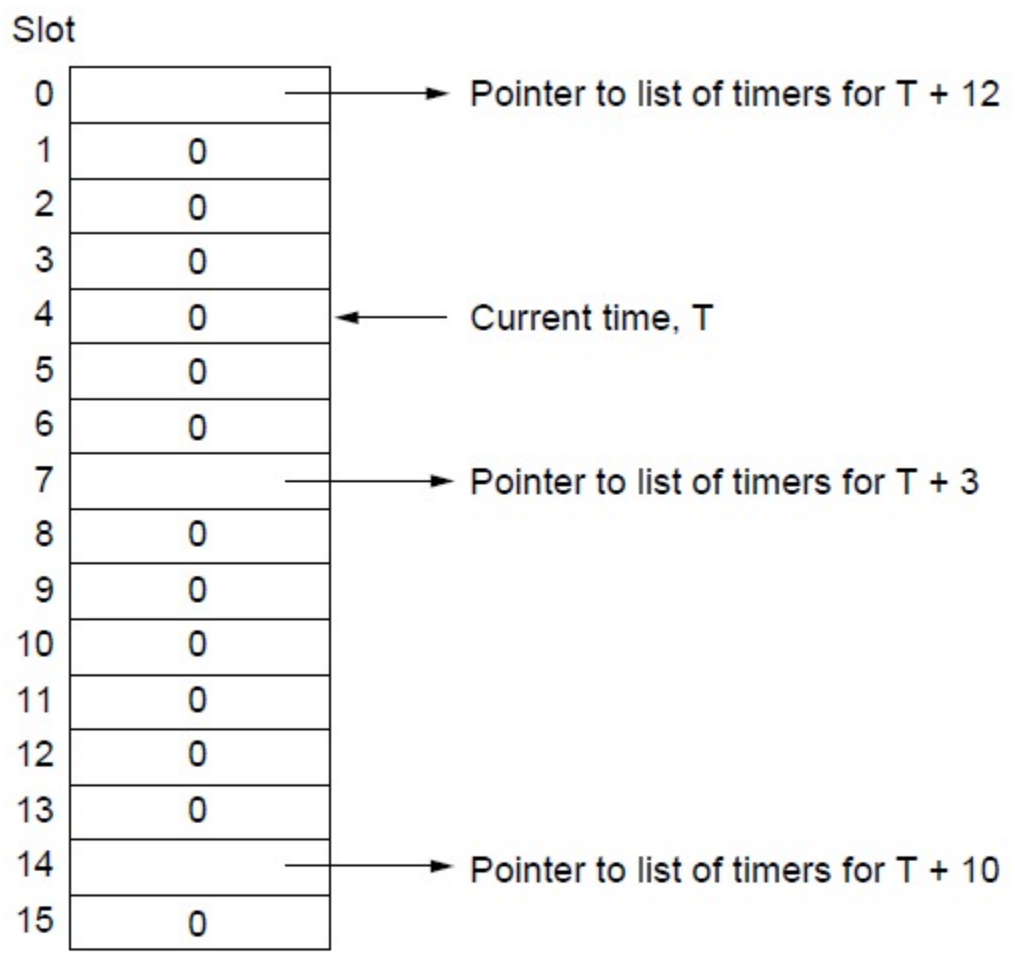
Fast TPDU Processing (2)





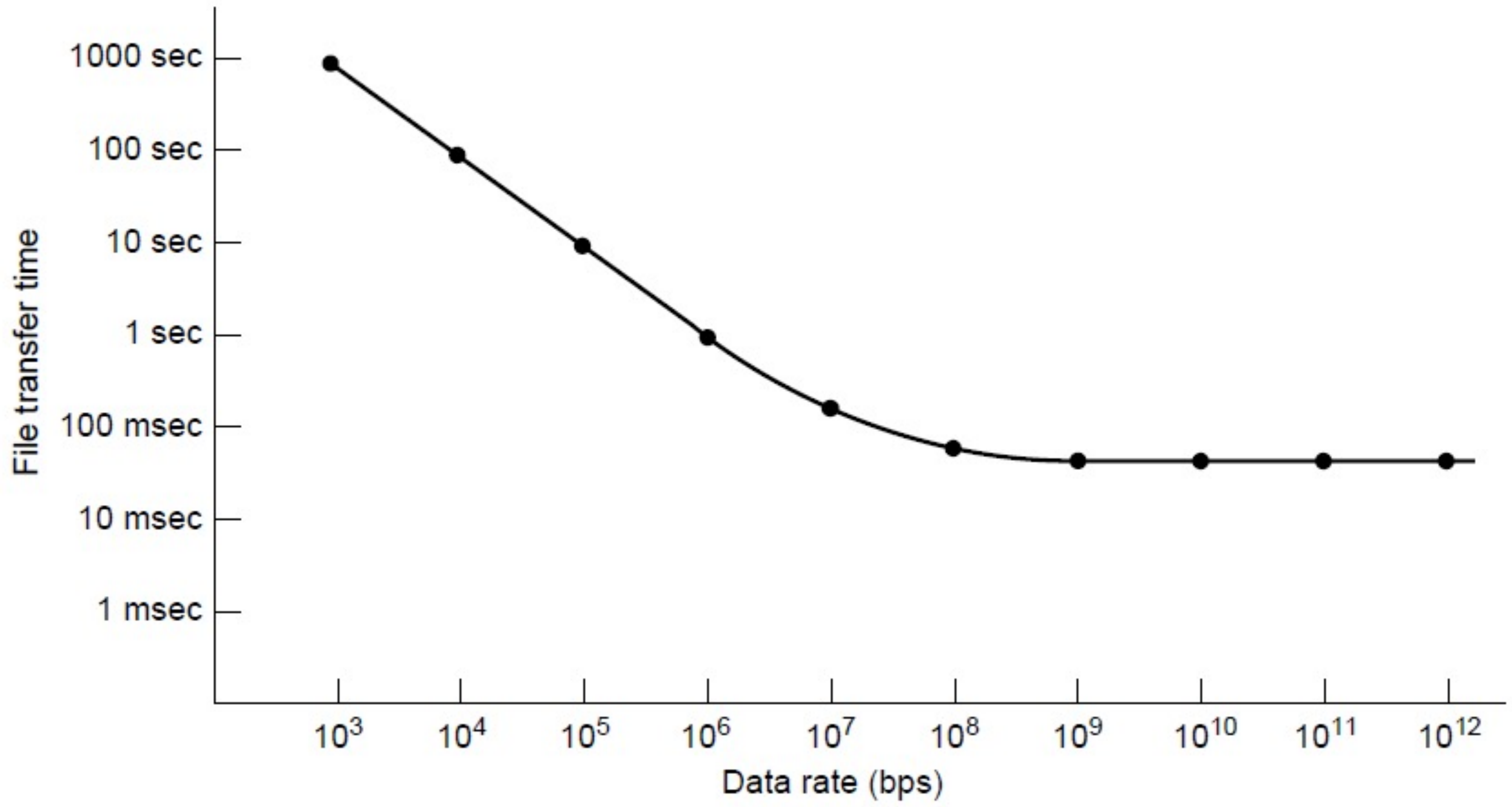
(a) TCP header. (b) IP header. In both cases, the shaded fields are taken from the prototype without change.

Protocols for High-Speed Networks (1)


Slot
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
0
0
0
0
0
0
0
0
0
0
0
0
0
Pointer to list of timers for T + 12
Current time, T
Pointer to list of timers for T + 3
Pointer to list of timers for T + 10

A timing wheel

Protocols for High-Speed Networks (2)

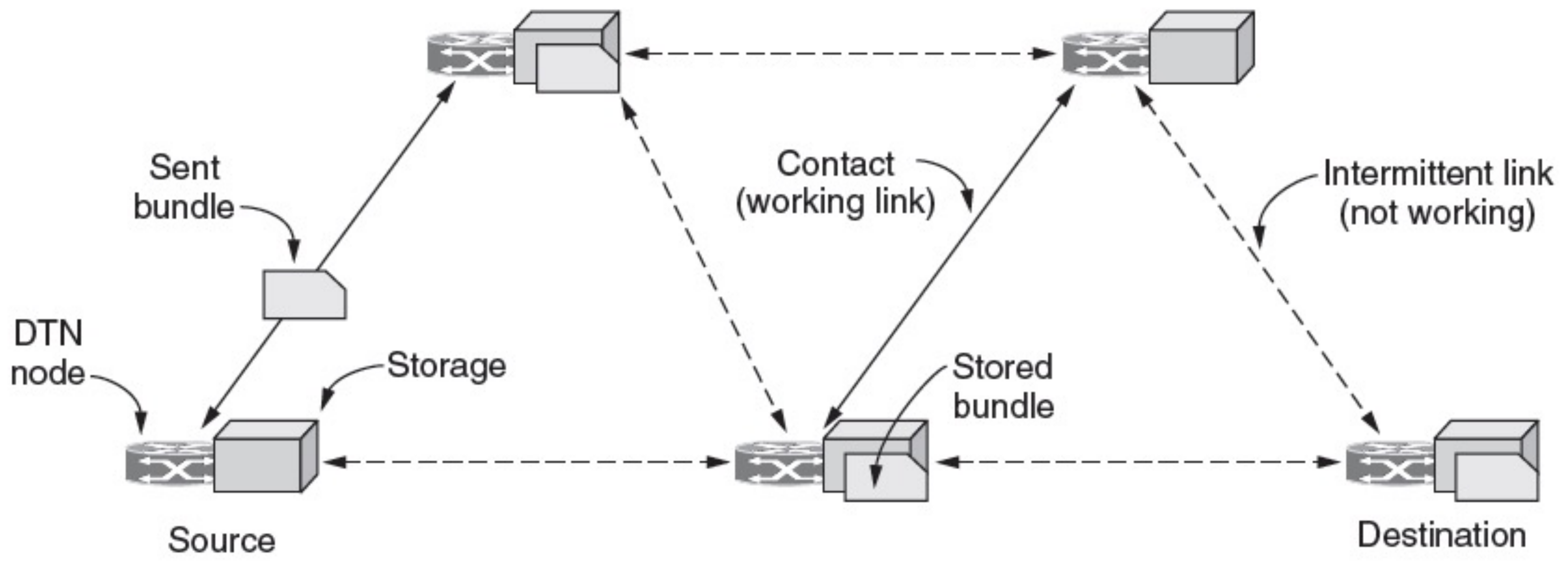


Time to transfer and acknowledge a 1-megabit file over a 4000-km line

Delay Tolerant Networking

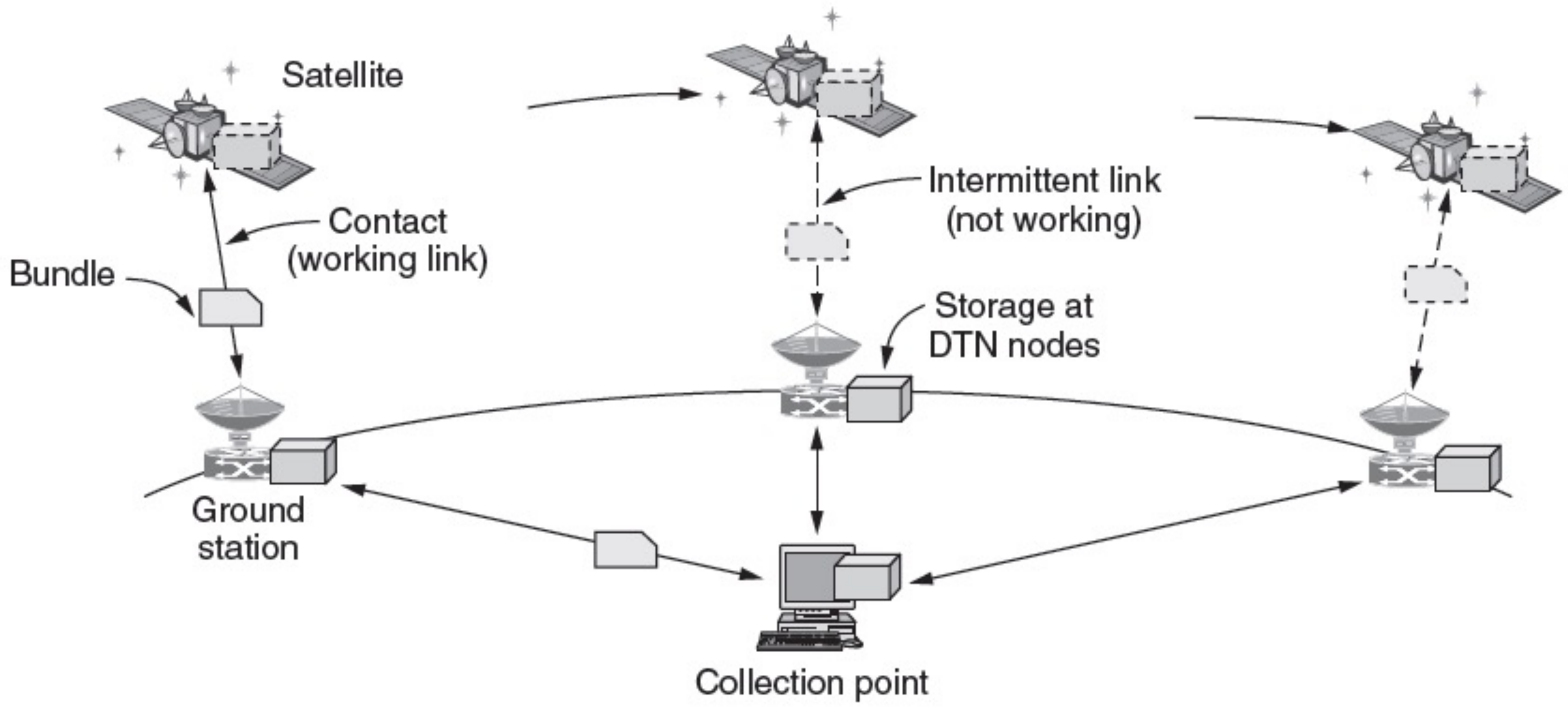
- DTN Architecture
- The Bundle Protocol

DTN Architecture (1)



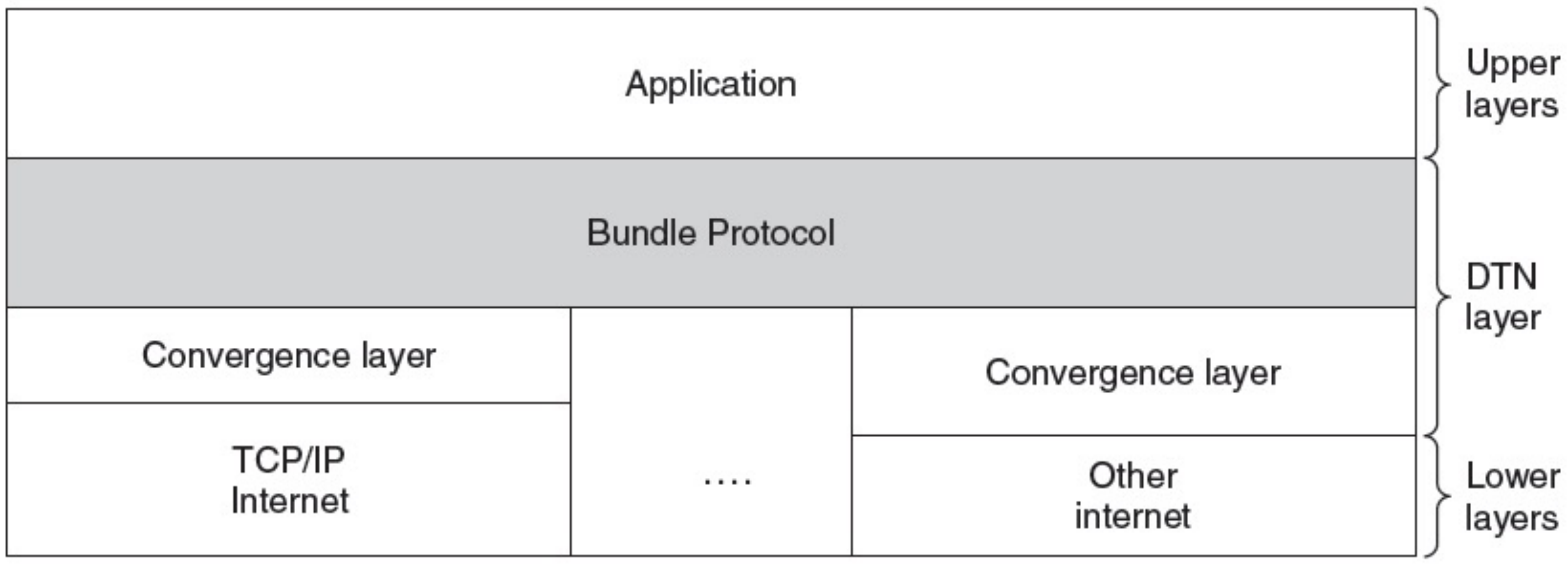
Delay-tolerant networking architecture

DTN Architecture (2)



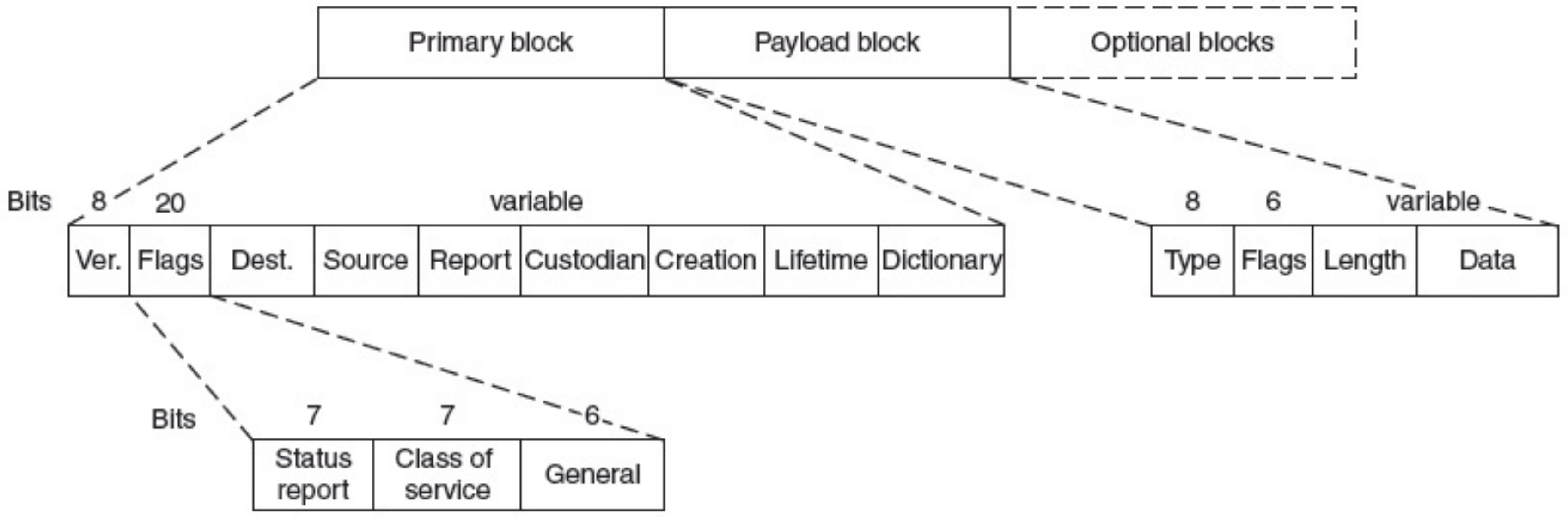
Use of a DTN in space.

The Bundle Protocol (1)



Delay-tolerant networking protocol stack.

The Bundle Protocol (2)



Bundle protocol message format.

End

Chapter 6