CHAPTER 6
THE TRANSPORT LAYER
(传输层)

- **The Transport Service (传输服务)**
- **Elements of Transport Protocols (传输协议的若干问题)**
- **A Simple Transport Protocol (简单传输协议)**
- **The Internet Transport Protocols (Internet传输协议): UDP**
- **The Internet Transport Protocols (Internet传输协议): TCP**
- **Performance Issues (性能问题)**

The Transport Service

- **Services Provided to the Upper Layers**
- **Transport Service Primitives (简单传输服务原语)**
- **Berkeley Sockets (套接字)**
- **An Example of Socket Programming:**
  - **An Internet File Server**

The Transport Service: Services

- **Transport layer services (传输服务) :**
  - To provide efficient, reliable, and cost-effective service to its users, normally processes in the application layer.
  - To make use of the services provided by the network layer.
- **The transport entity (传输实体):** the hardware and/or software within the transport layer that does the work. Its positions:
  - In the OS kernel, in a separate user process, in a library package bound to network applications, or
  - On the network interface card.

Services Provided to the Upper Layers

- Transport Entity
- Provide efficient, reliable and cost-effective services
- Is run by user (hosts), not run by carrier (subnet)

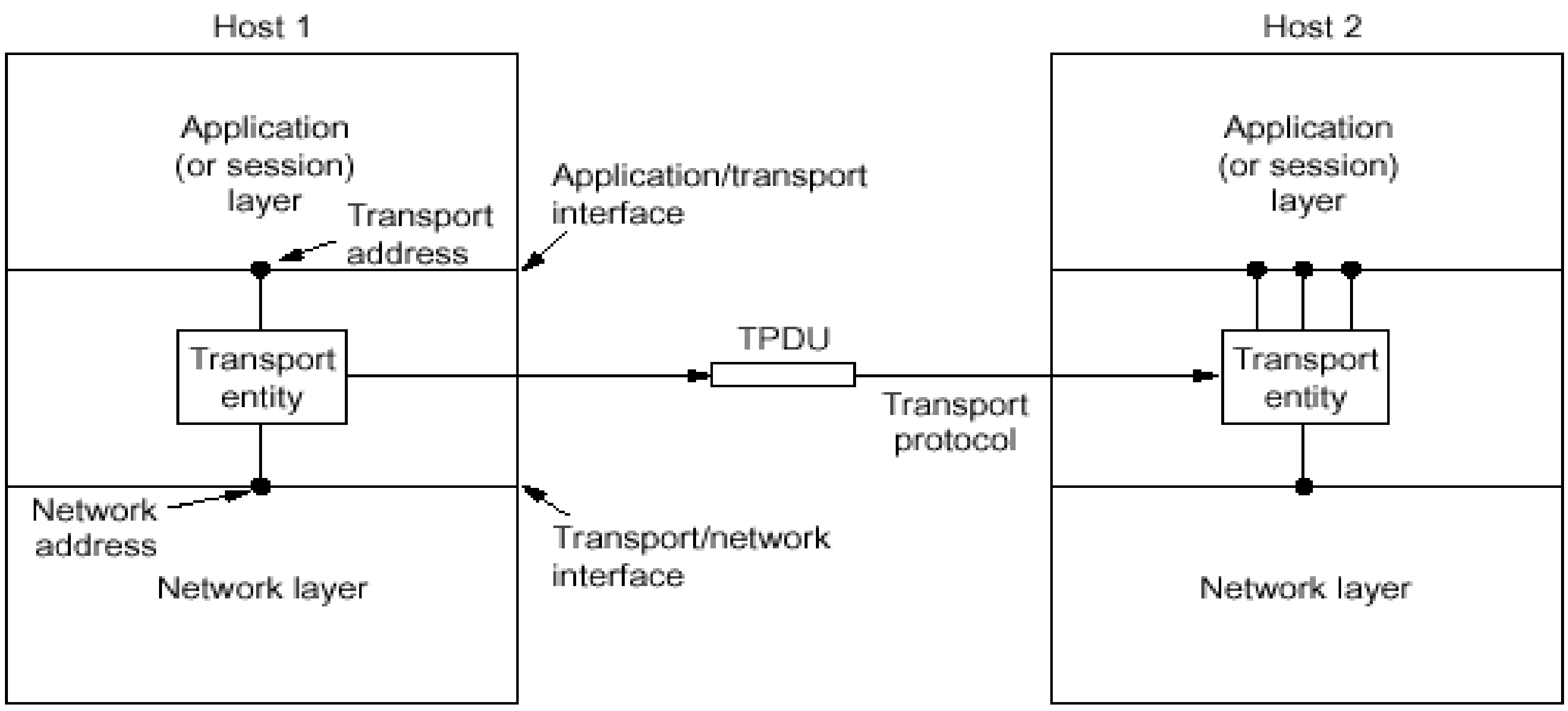
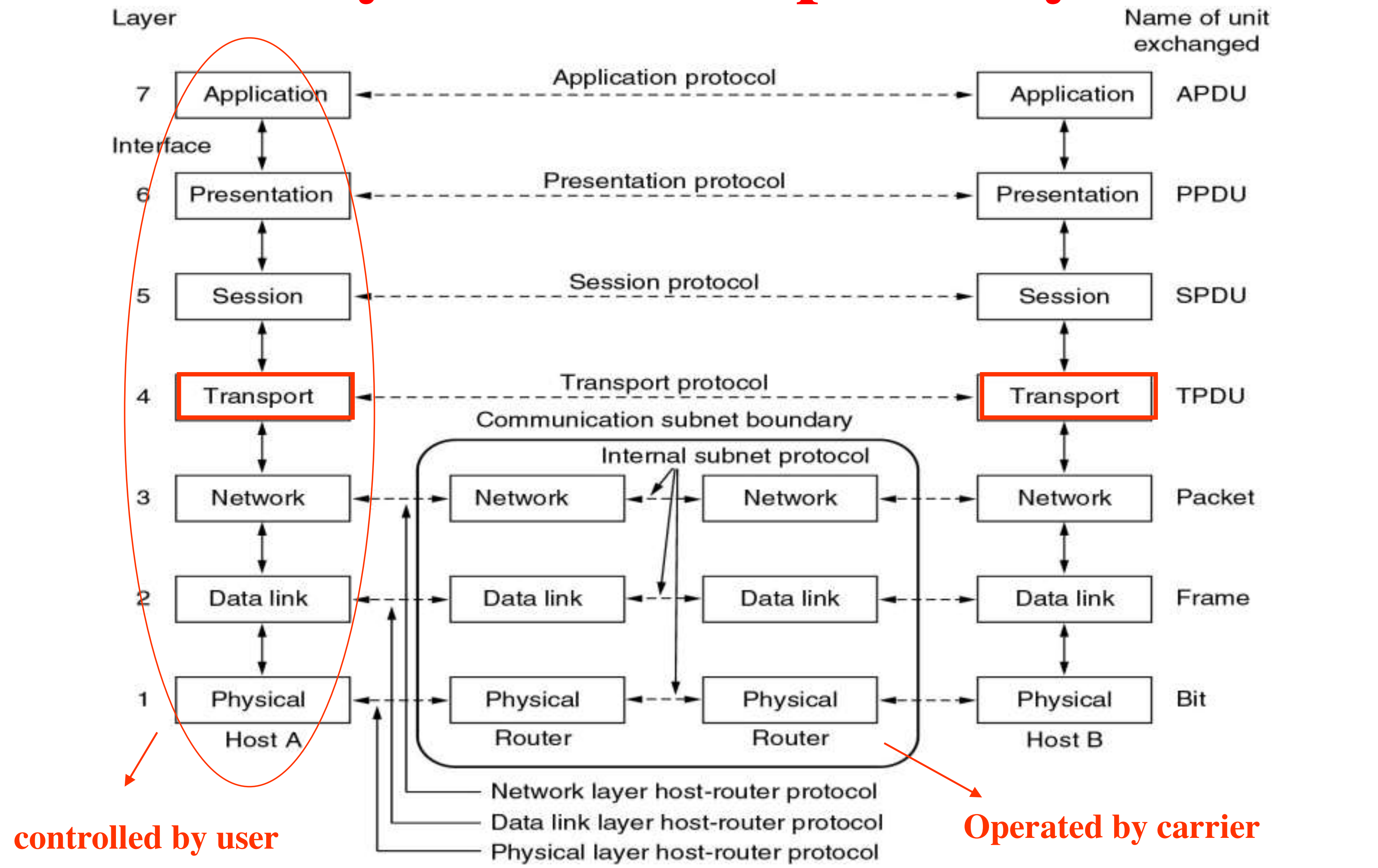


Fig. 6-1. The network, transport, and application layers.

Why Need Transport Layer



Transport Layer Function

- Connection-oriented or connectionless service
- Addressing
  - For application layer
- Flow control
- Congest control
- Error control
  - More reliable than the network service
- Multiplex
  - Use several network connection for one data transit
- Standard transport service interface
  - Independent to different network service primitives

Transport Service Primitives

- Provide a reliable service on top of an unreliable network
    - Error-free bit stream
- Ease to use

Primitive	TPDU 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 TPDU arrives
DISCONNECT	DISCONNECTION REQ.	This side wants to release the connection

Fig. 6-3. The primitives for a simple transport service.

Transport PDU

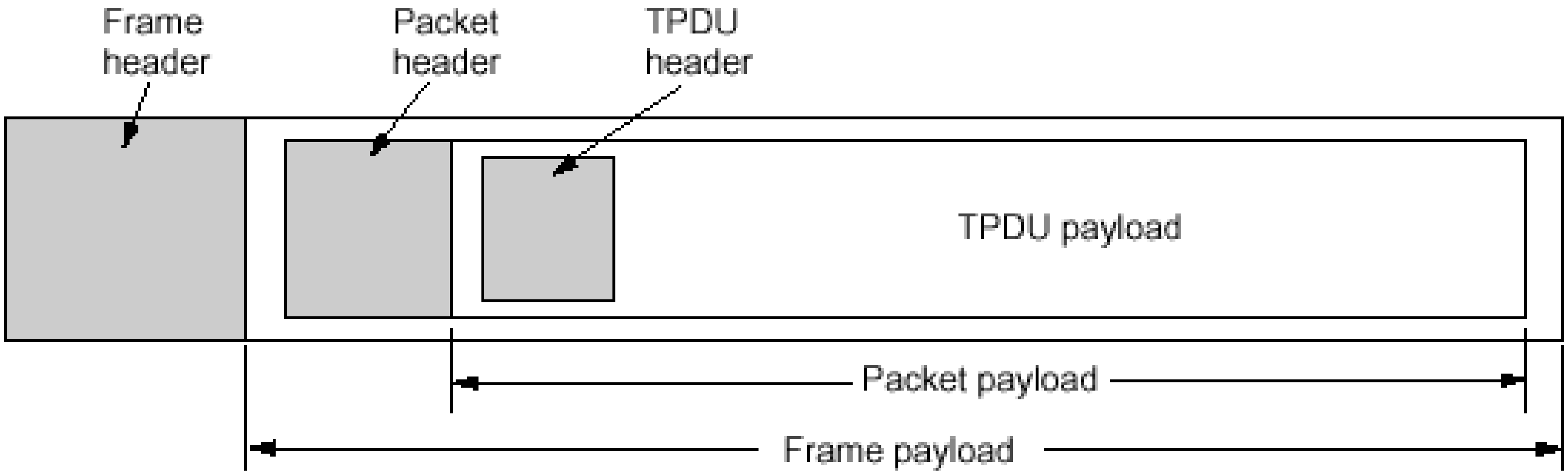
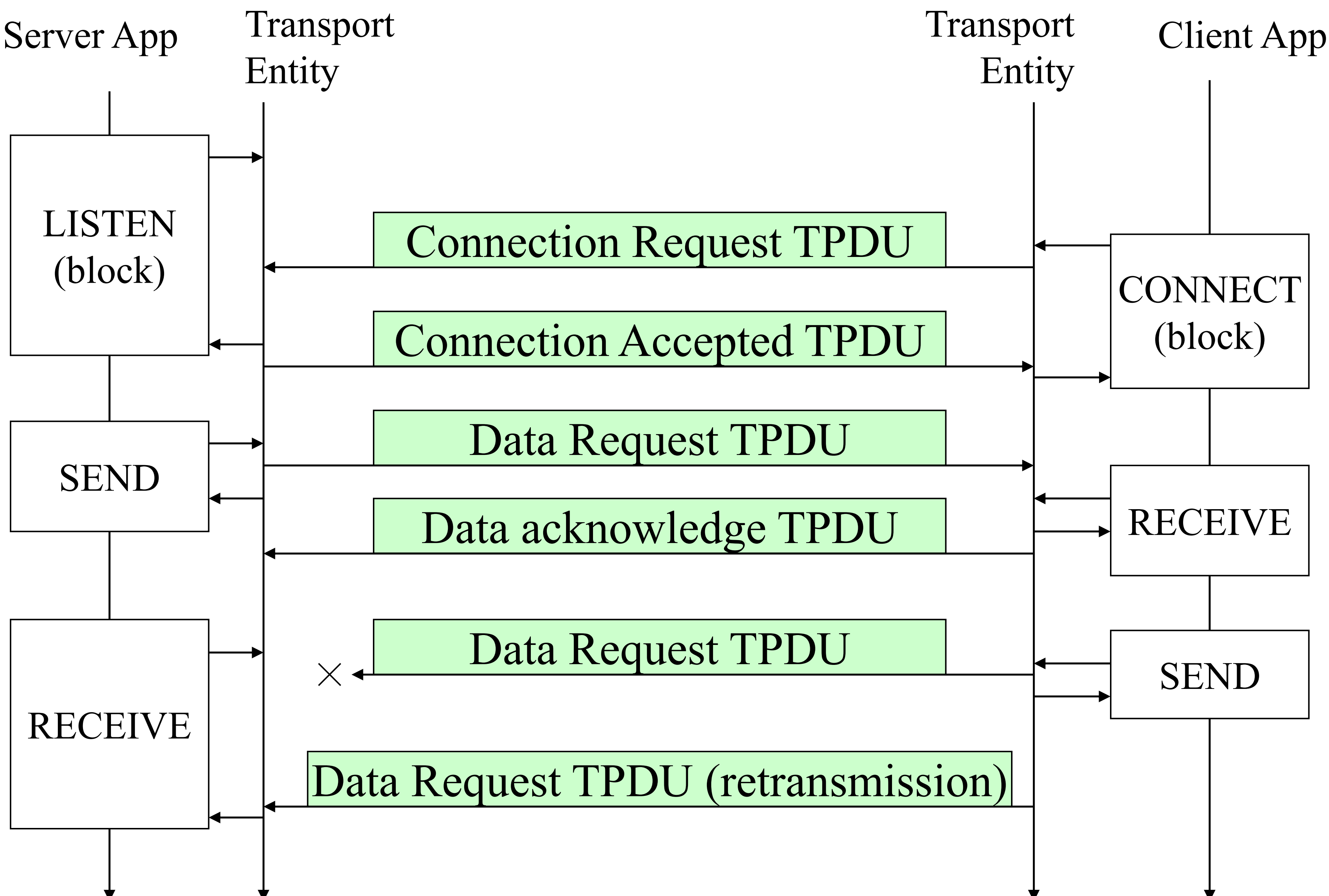
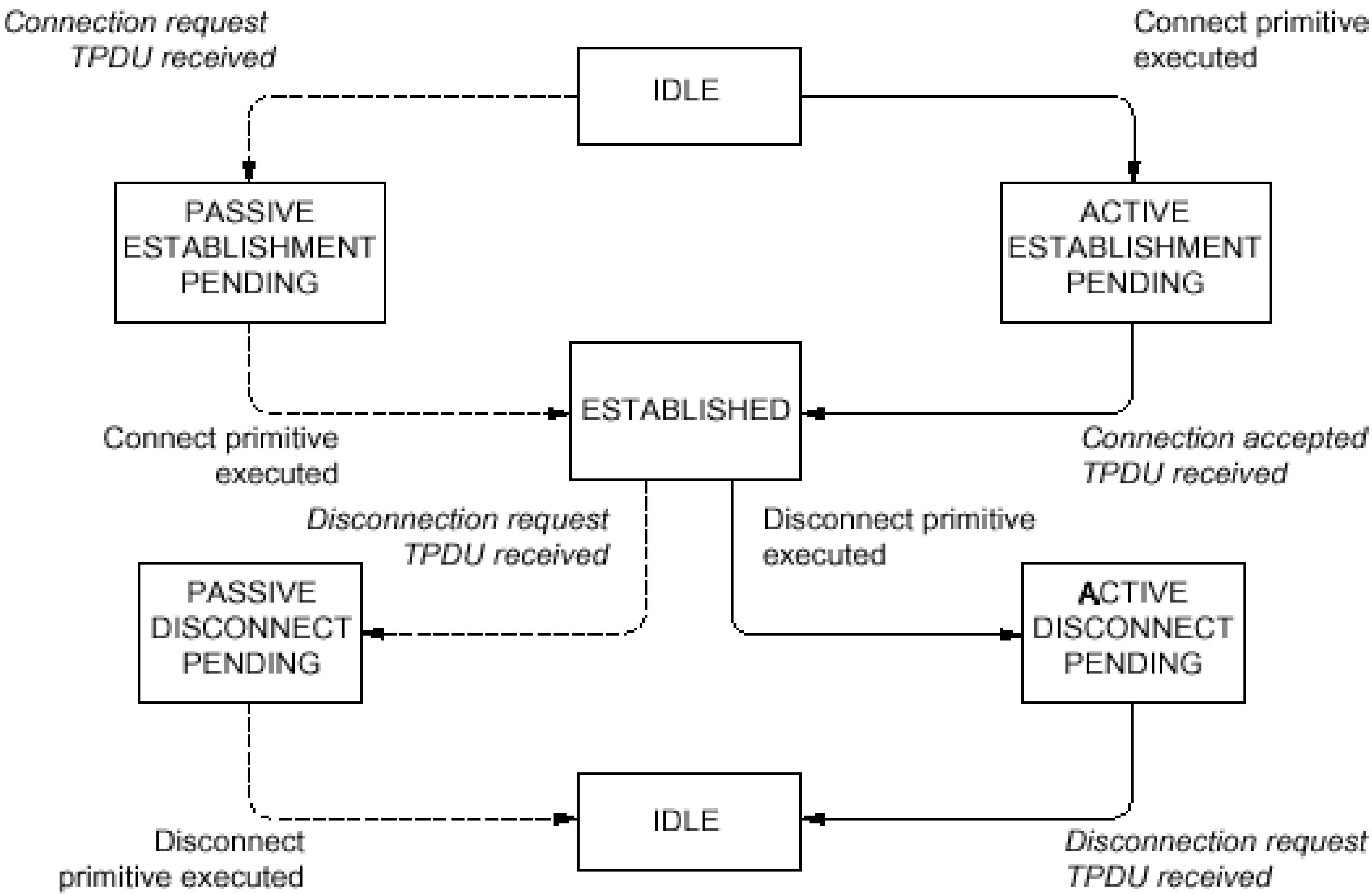


Fig. 6-4. Nesting of TPDUs, packets, and frames.

How to use transport primitives



Transport Entity State Diagram

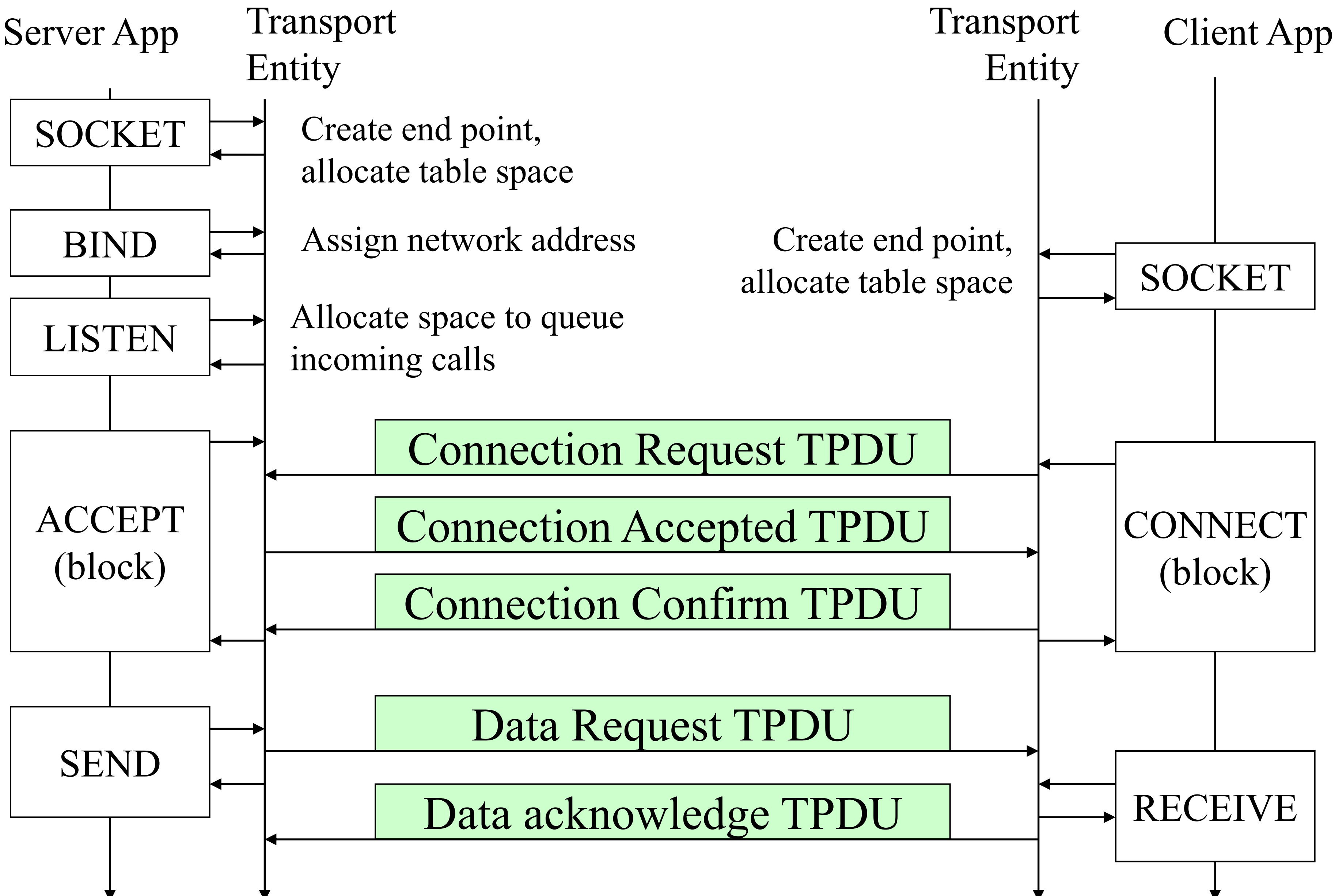


Berkeley Sockets for TCP

Primitive	Meaning
SOCKET	Create a new communication end point
BIND	Attach a local address to a socket
LISTEN	Announce willingness to accept connections; give queue size
ACCEPT	Block the caller until a connection attempt arrives
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

Fig. 6-6. The socket primitives for TCP.

How to use Berkeley Socket



Example of Socket Programming

- A Internet File Server
- Server listen at 12345
- Client usage
  - Client [server address] [filepath/filename] > [writefile]
  - Get file from server, and write to local file
- Some function useful
  - htons(): convert short integer to standard format
  - htonl(): convert long integer to standard format
    - Big-endian (SPARC) and little-endian (Intel machine)
  - setsockopt(): set socket options

Client Side (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 Side (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 Side (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);
}
```

Server Side (1)

```
#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 Side (2)

```
 . . .
/* 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");
  . . .
```

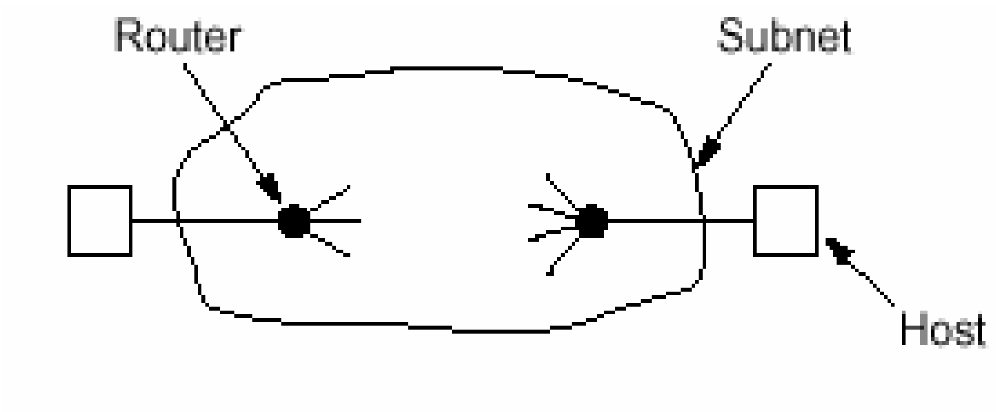
Server Side (3)

```
. . .
/* 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 */
}
}
```

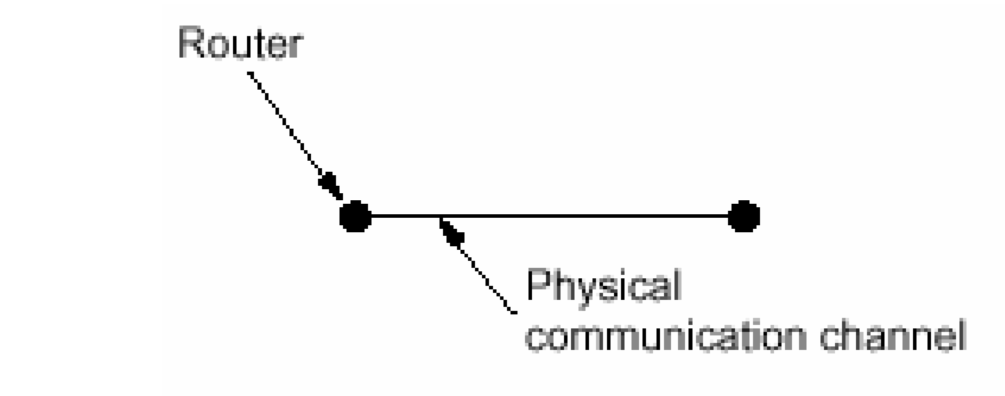
Elements of Transport Protocols

- Addressing
- Connection Establishment
- Connection Release
- Flow Control and Buffering
- Multiplexing
- Crash Recovery

- Environment of Transport Layers

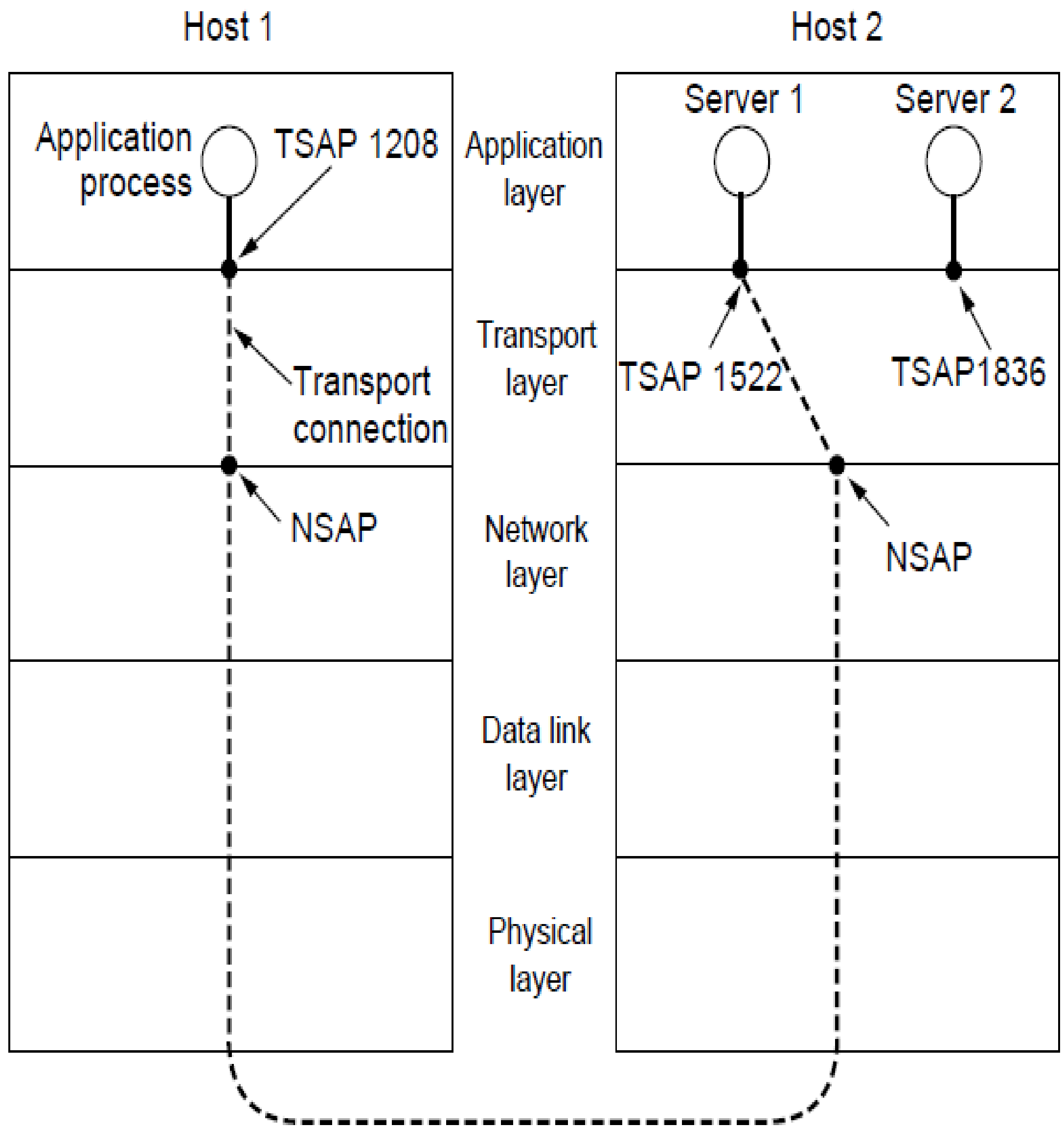


- Environment of Data Link Layers



Addressing

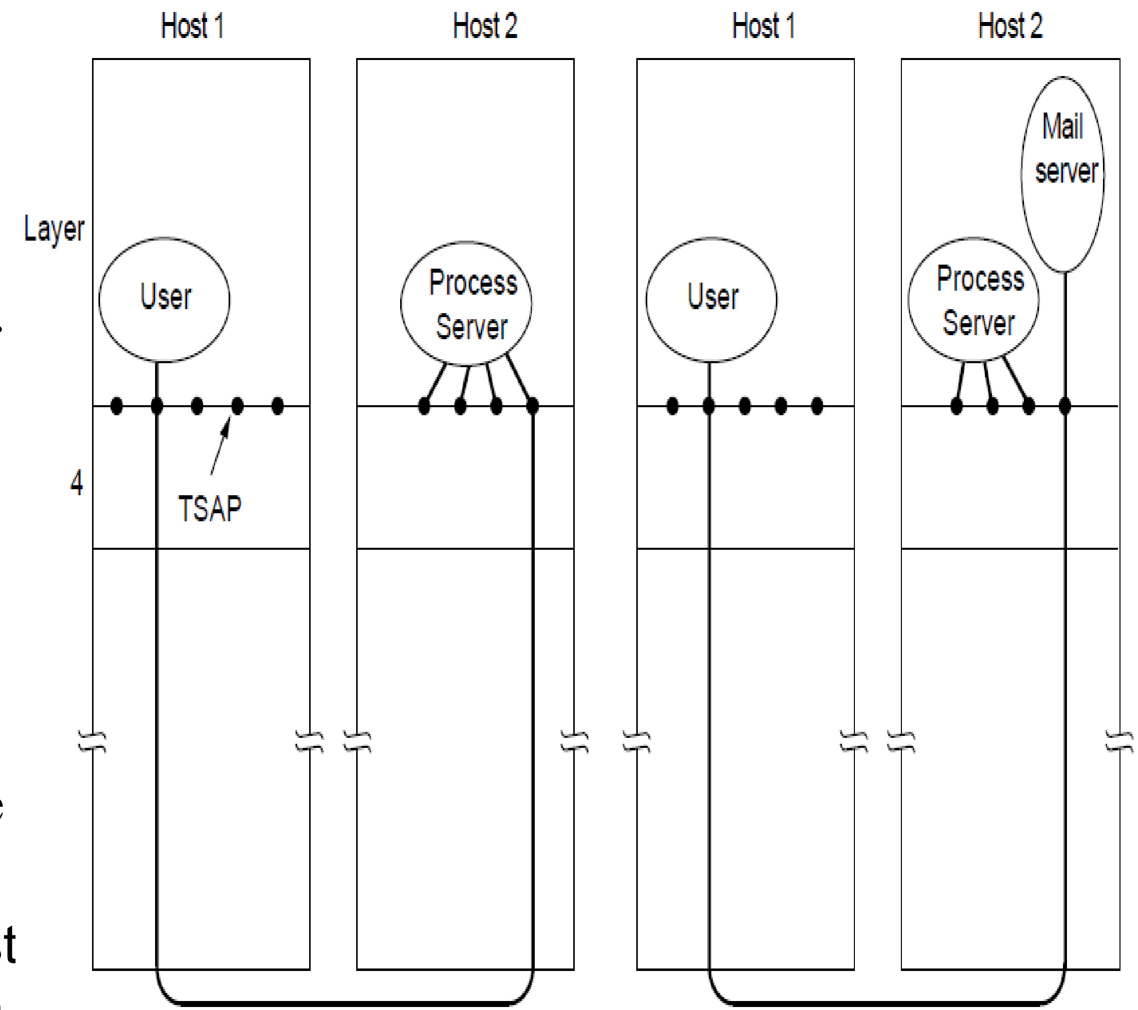
- TSAP, NSAP and connections
- Well known port: < 1024, see /etc/services



How client know server's TSAP address

- Permanent TSAP address
  - Well known port
- Initial connection protocol: Process server
  - Spawn servers needed
- Name Server or Directory Server
  - Listen in well known TSAP address
  - Translation service name to TSAP address

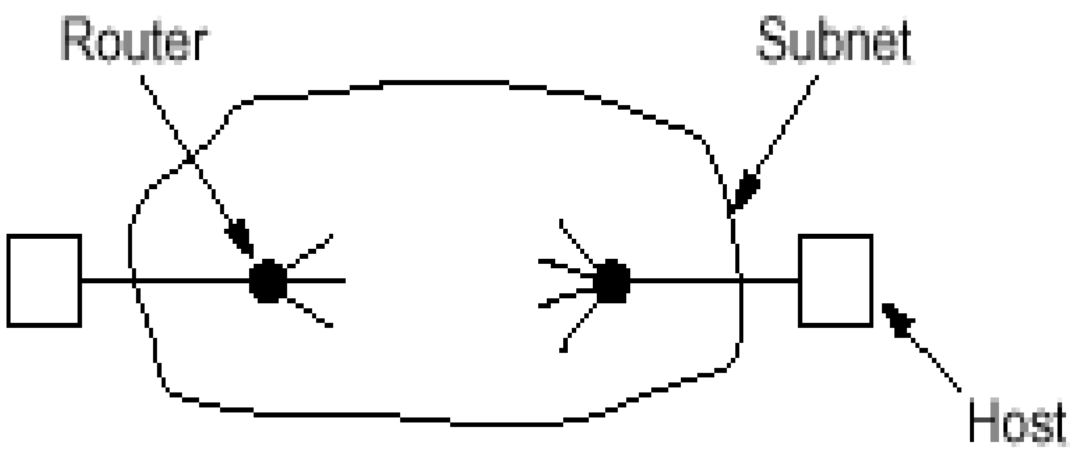
– How a user process in host 1 establishes a connection with a mail server in host 2 via a process server



(a)
(b)

Problem of Connection Establishment

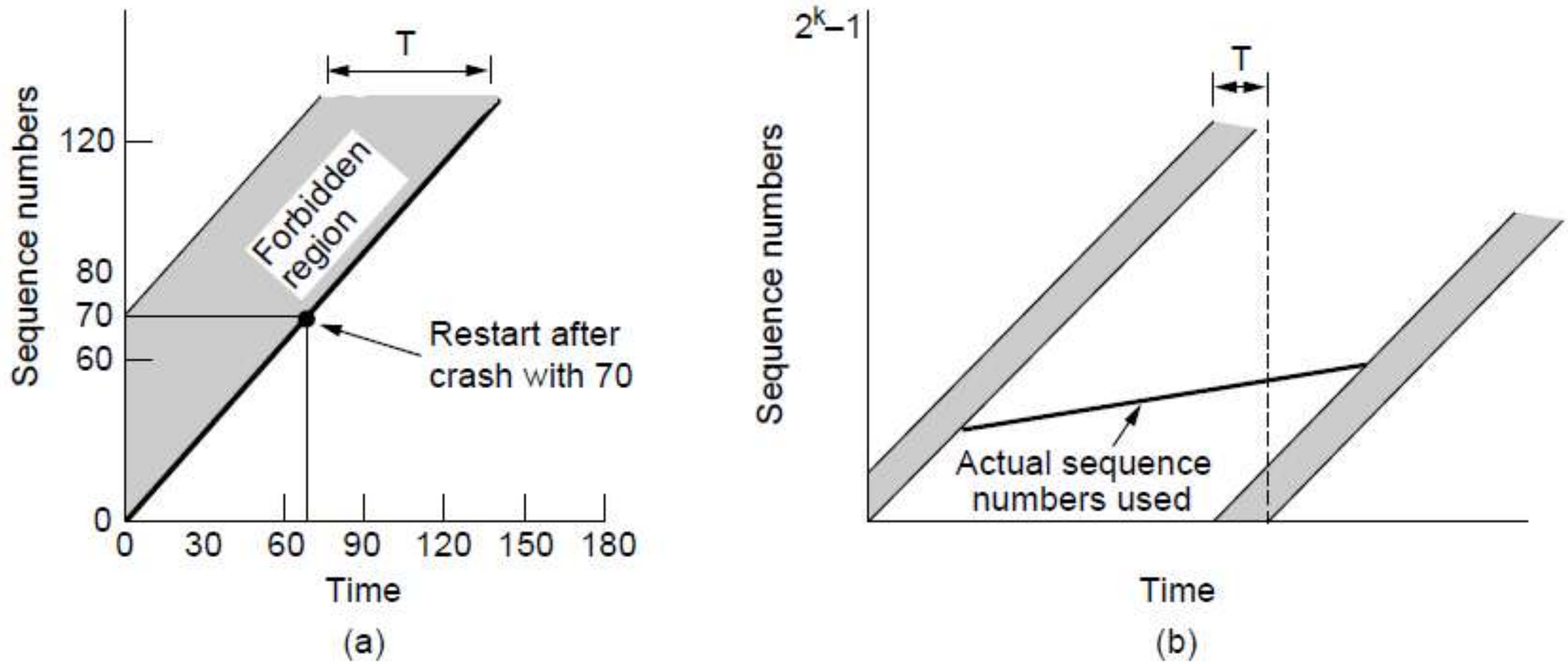
- Problem
  - Duplicate Connection Request and ACK
- Restrict packet life time, so duplicate packet will be discard during a max period
- Initial serial number, may use clock to generate



Techniques for restricting packet lifetime

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

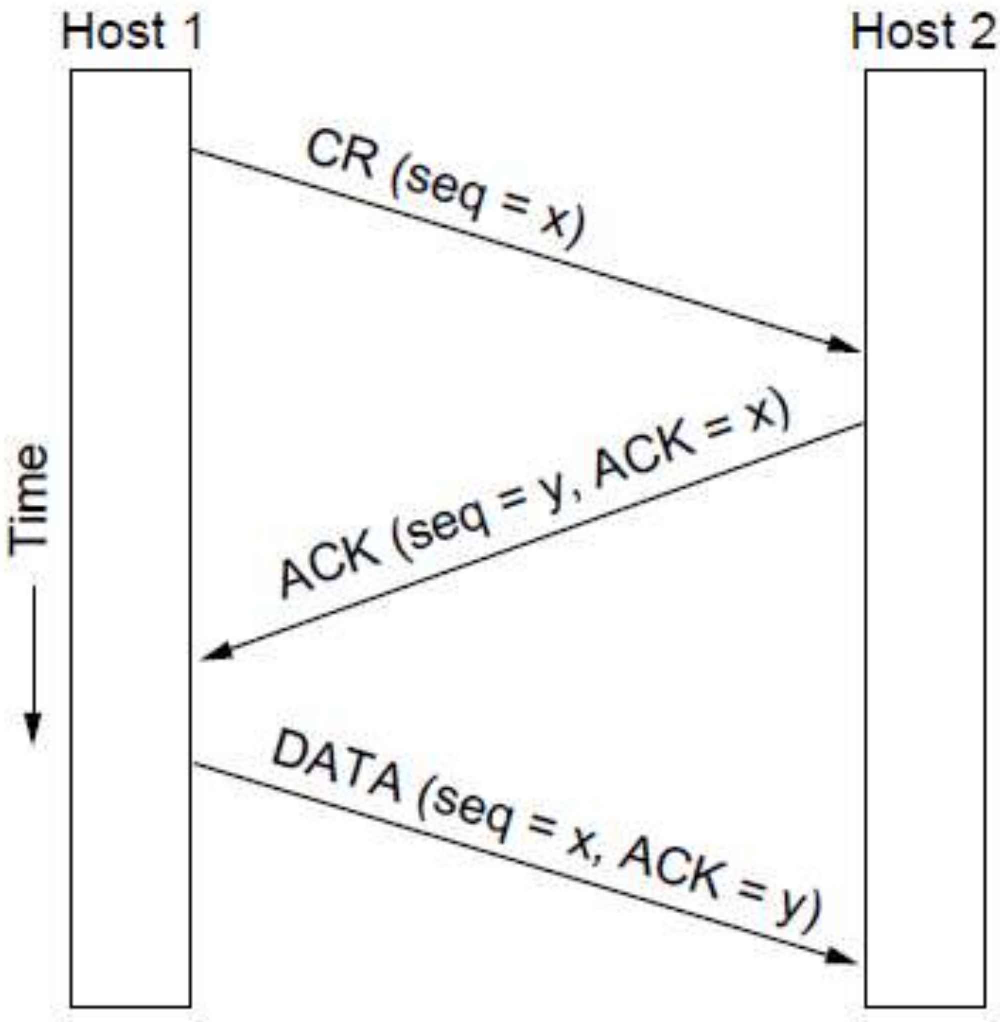
Sequence Number



(a)TPDUs may not enter the forbidden region.

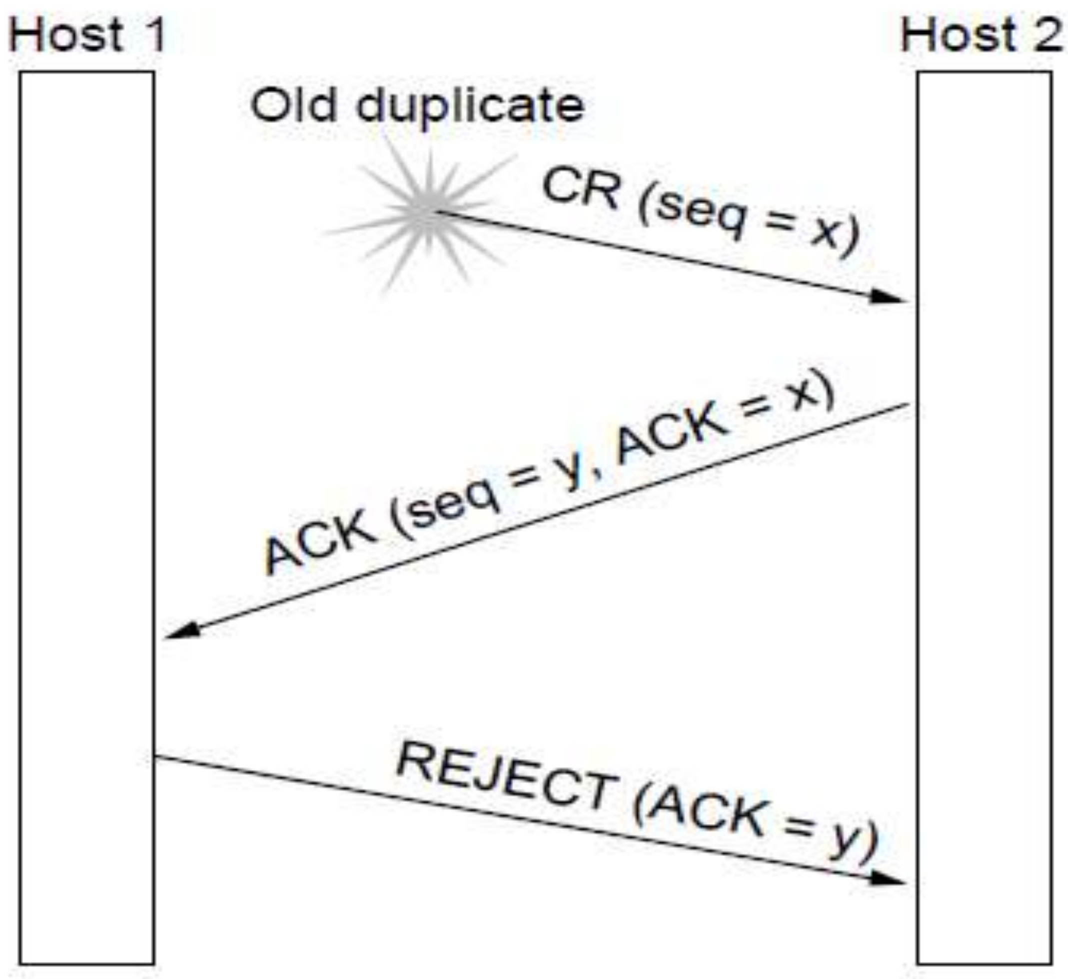
(b)The resynchronization problem.

Three-way handshake (Normal)



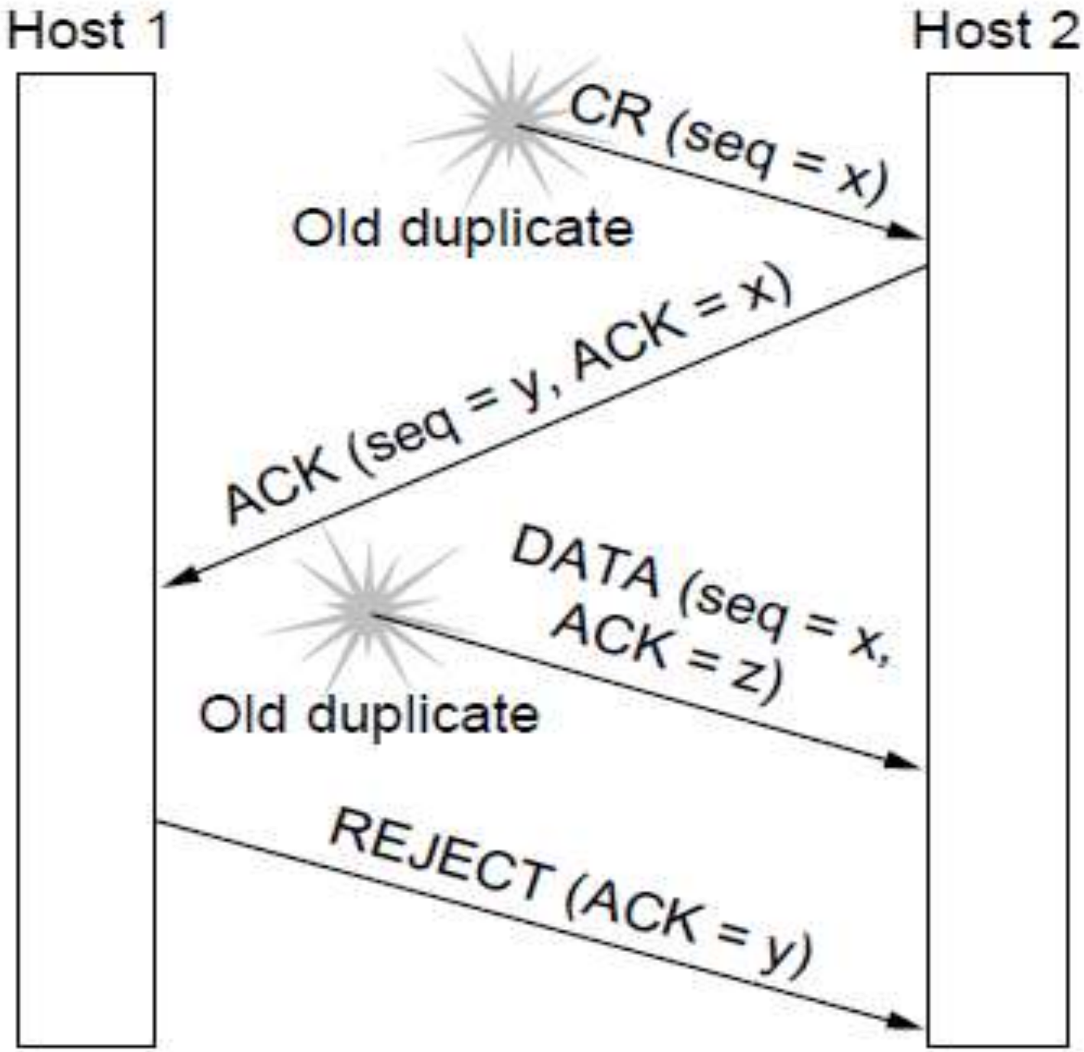
CR: Connection Request

Three-way handshake (Old duplicate)



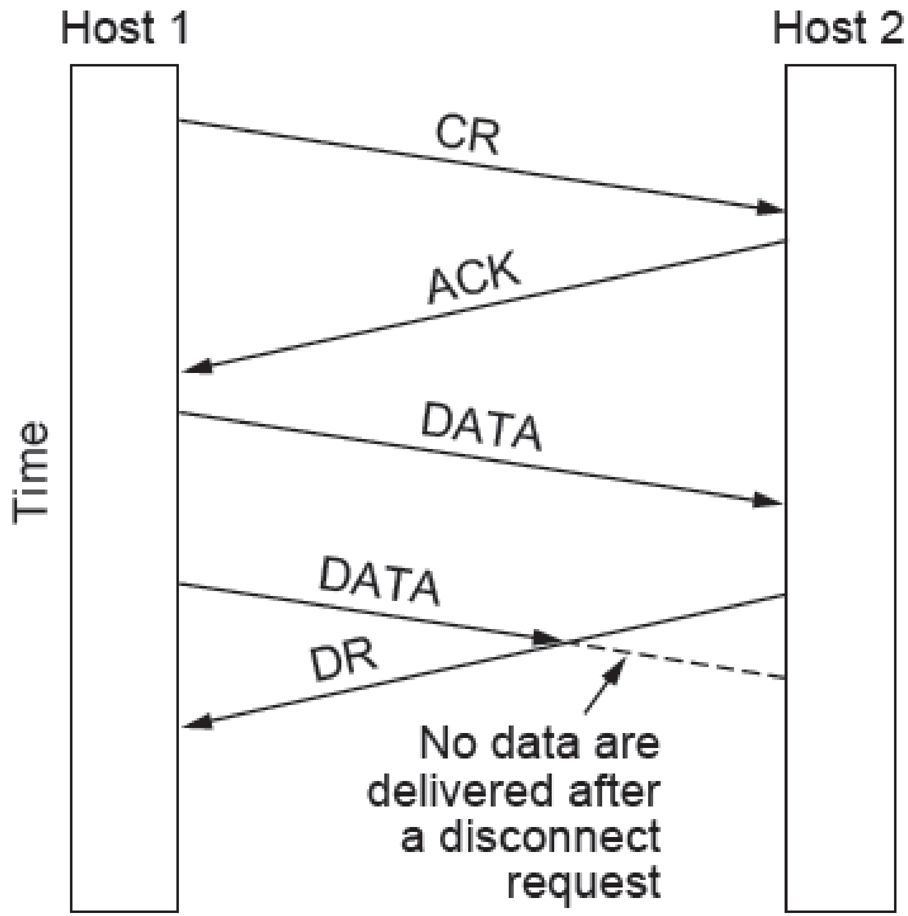
Old duplicate CONNECTION REQUEST appearing out of nowhere

Three-way handshake (duplicate CR)



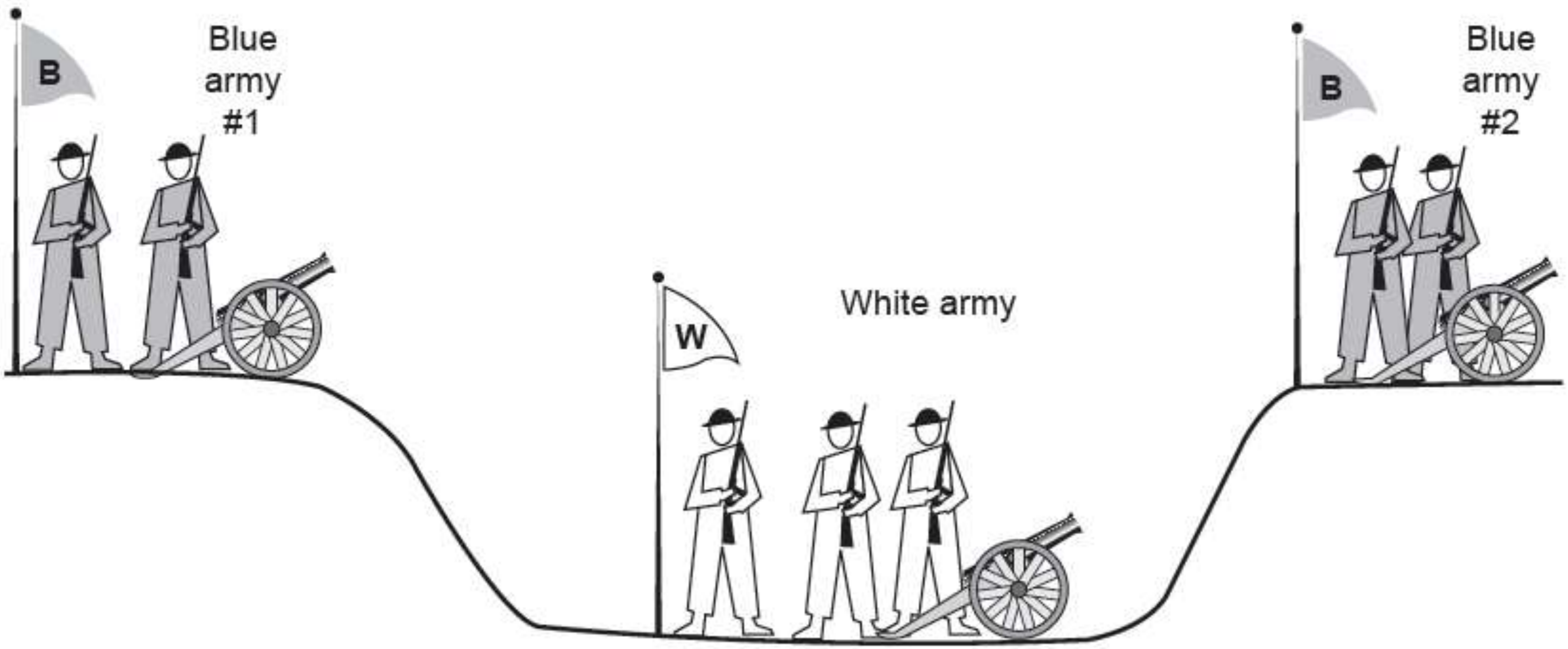
Duplicate CONNECTION REQUEST and duplicate ACK

Connection Release (1)

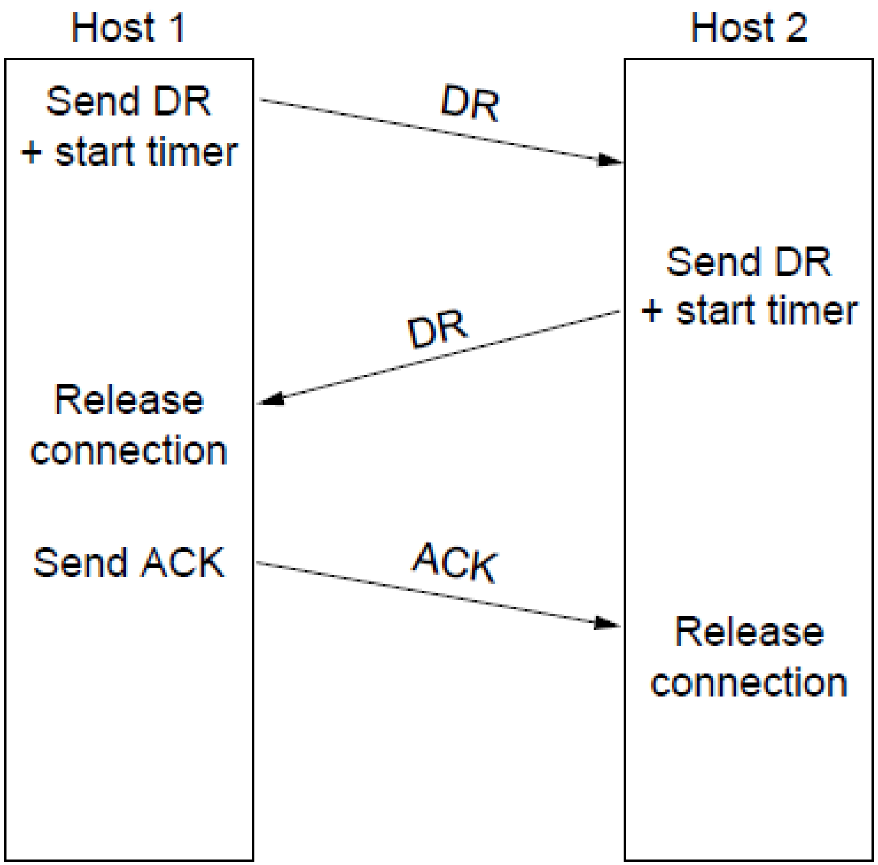


Abrupt disconnection with loss of data

Connection Release (2)

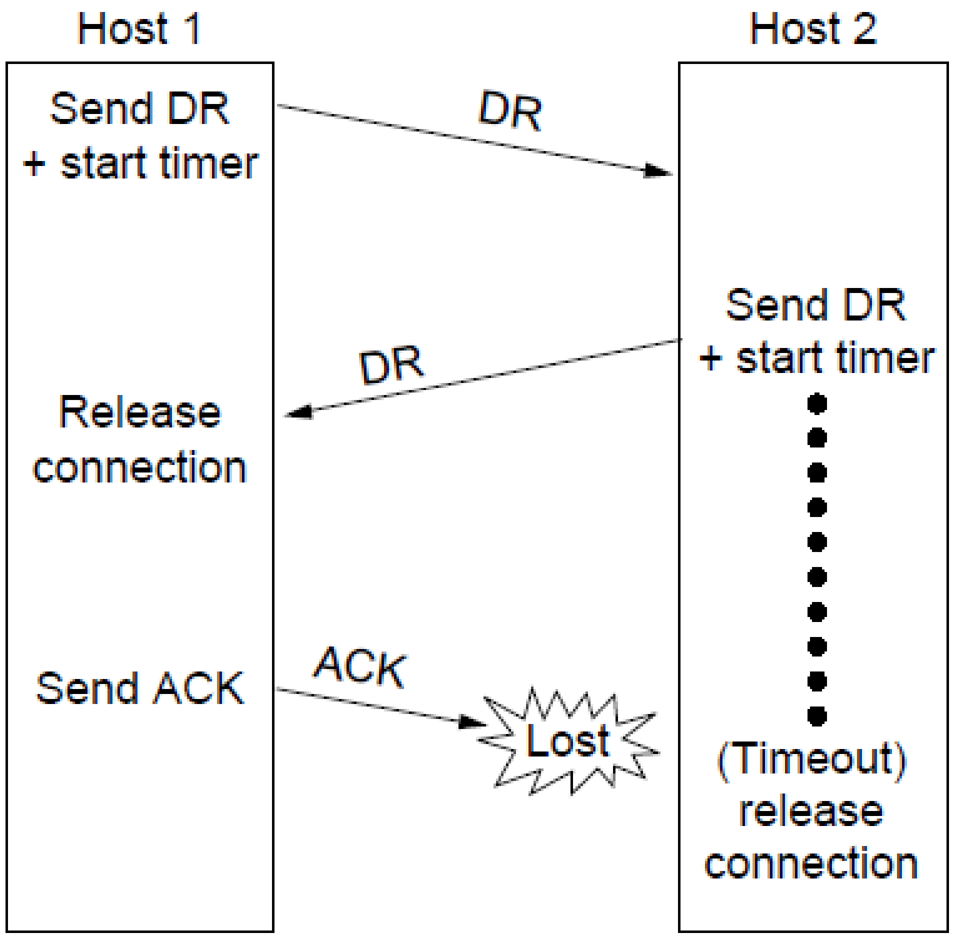


Three-way handshake release



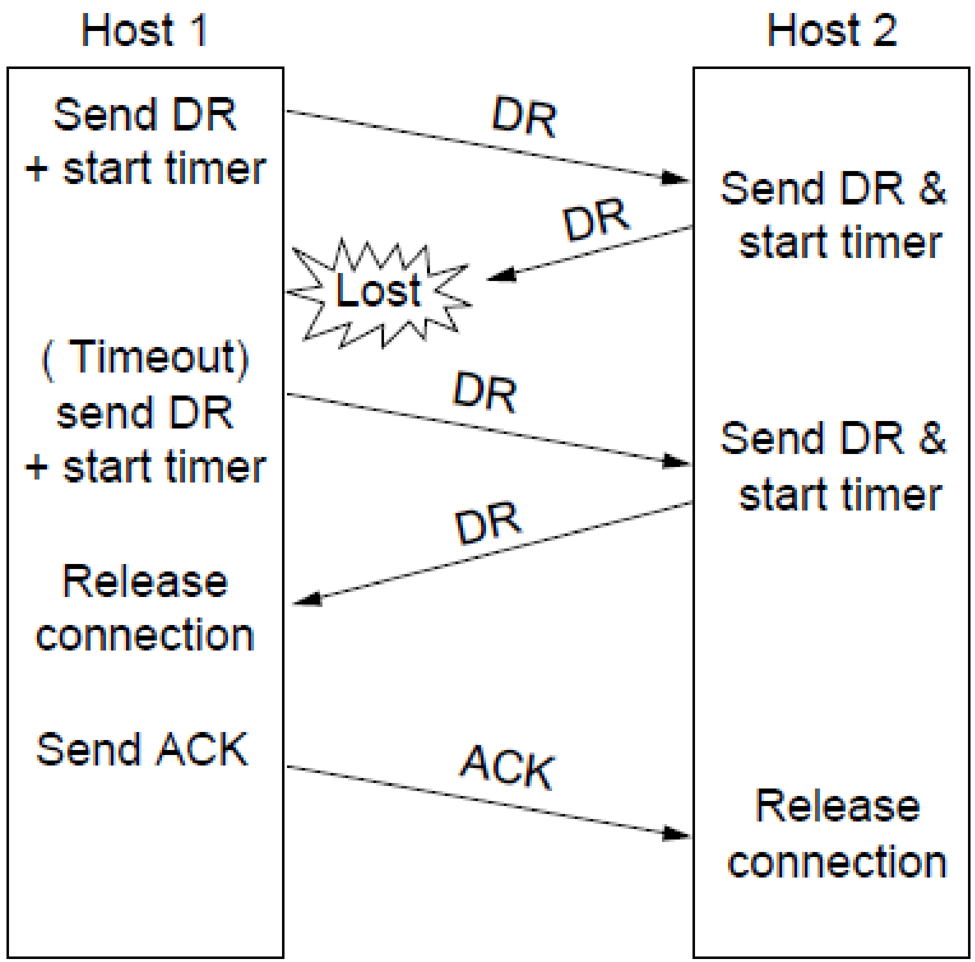
(a) Normal case of three-way handshake

Three-way handshake release



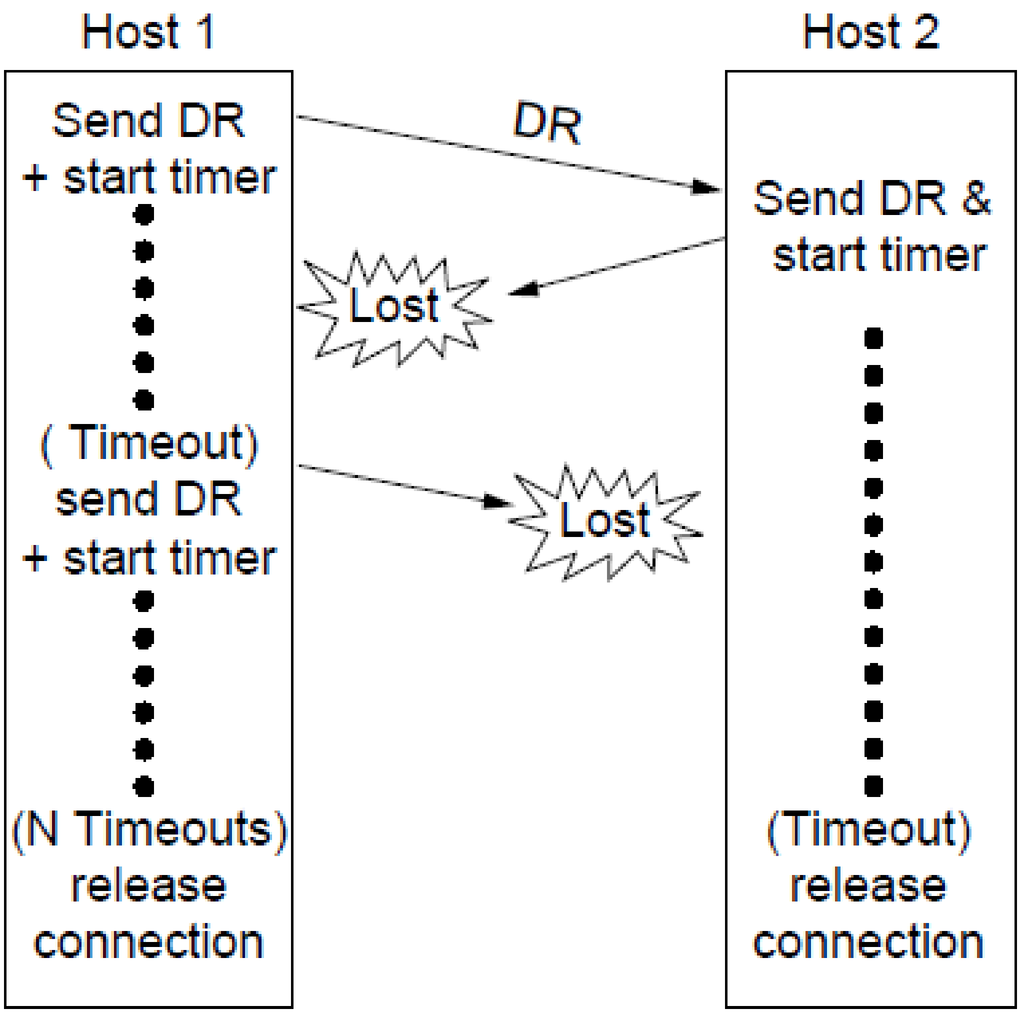
(b) Final ACK lost.

Three-way handshake release



(c) Response lost

Three-way handshake release

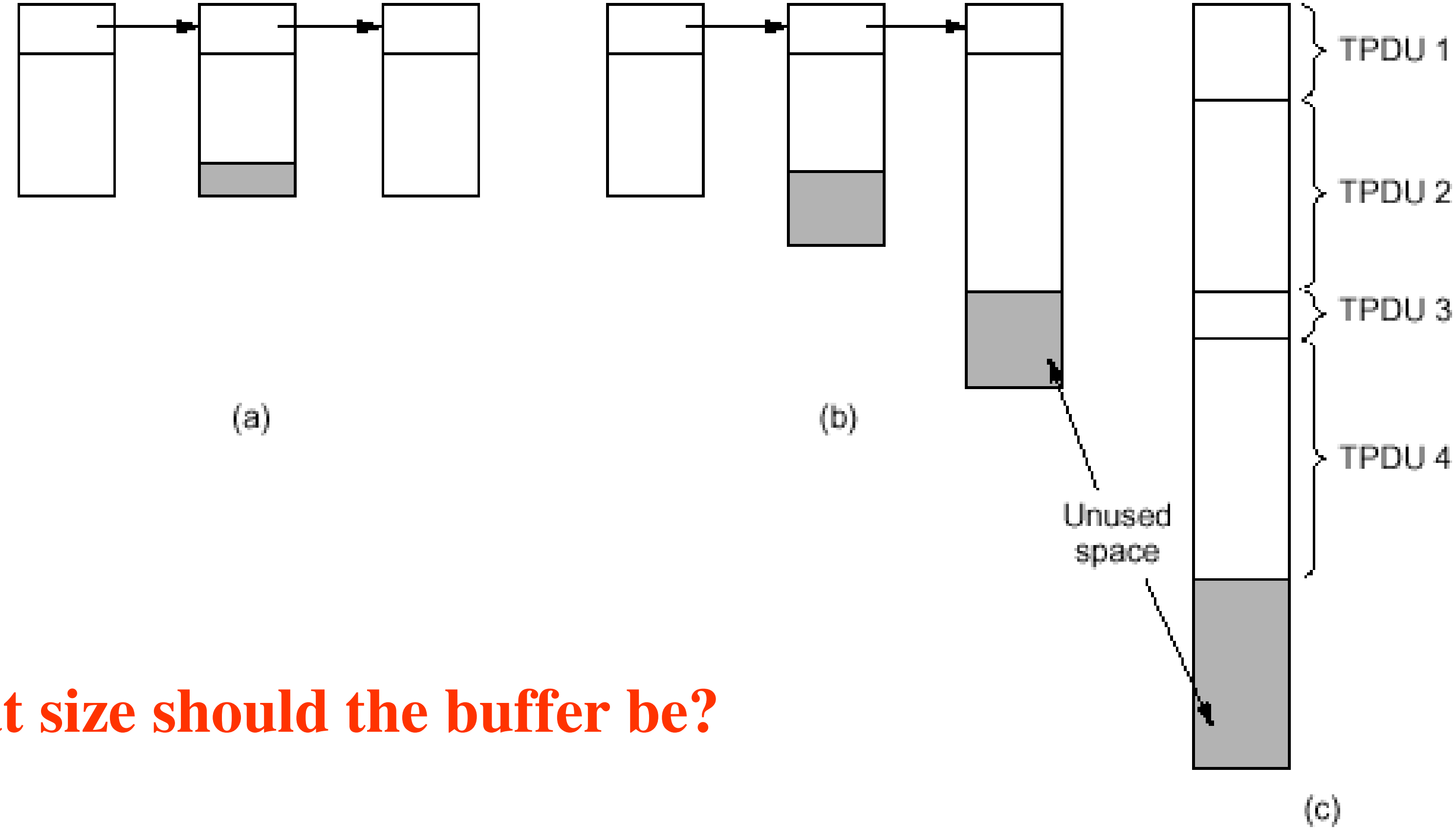


(d) Response lost and subsequent DRs lost.

Flow Control

- Sender should have buffer to store unacknowledged data
- Receiver should have buffer to store unread data
- Data maybe arrived correctly, but no receive buffer to store
- Sender should know how many idle buffers that receiver has

Buffering



What size should the buffer be?

Fig. 6-15. (a) Chained fixed-size buffers. (b) Chained variable-size buffers. (c) One large circular buffer per connection.

Flow Control and Buffering

	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

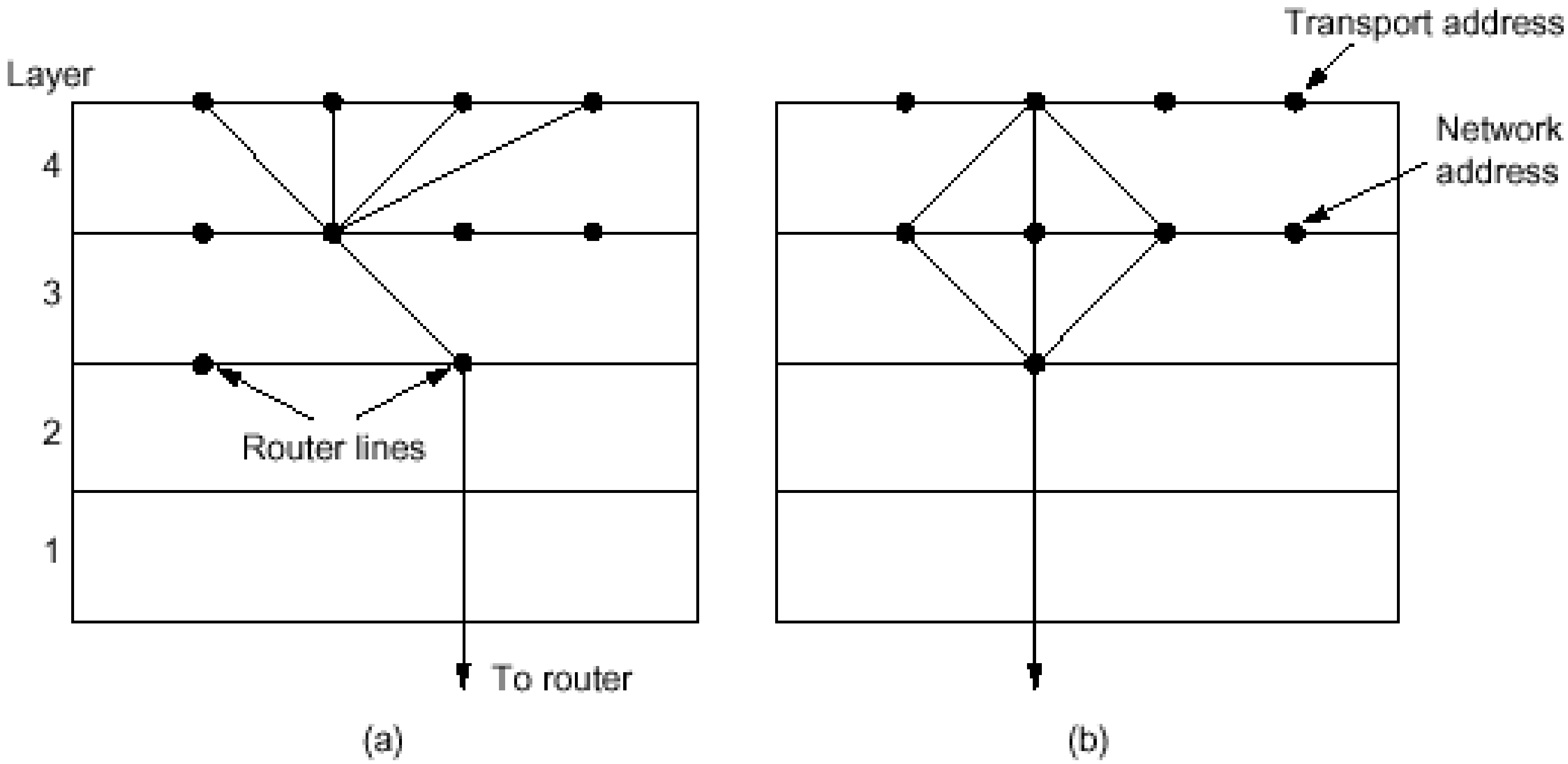


Fig. 6-17. (a) Upward multiplexing. (b) Downward multiplexing.

How to recovery protocol state if the host crash during data exchange ?

Crash Recovery

Strategy used by sending host	Strategy used by receiving host: First ACK, then write			First write, then ACK		
	AC(W)	AWC	C(AW)	C(WA)	W AC	WC(A)
Always retransmit	OK	DUP	OK	OK	DUP	DUP
Never retransmit	LOST	OK	LOST	LOST	OK	OK
Retransmit in S0	OK	DUP	LOST	LOST	DUP	OK
Retransmit in S1	LOST	OK	OK	OK	OK	DUP

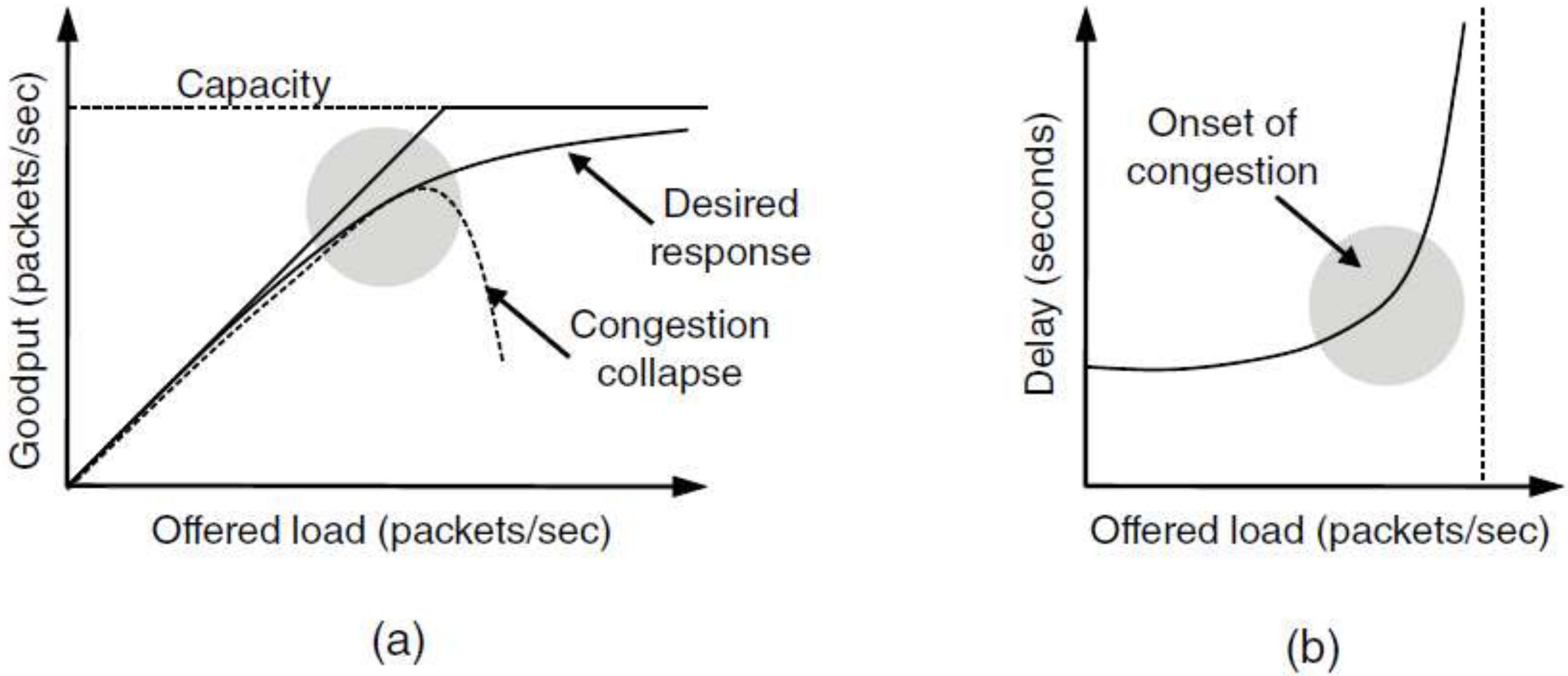
OK = Protocol functions correctly
DUP = Protocol generates a duplicate message
LOST = Protocol loses a message

- A: ACK sent W: Write to process C: Crash
- S1: one TPDU unacknowledged S0: no TPDU unacknowledged
- For each strategy there is some sequence of event that cause the protocol to fail

Congestion Control

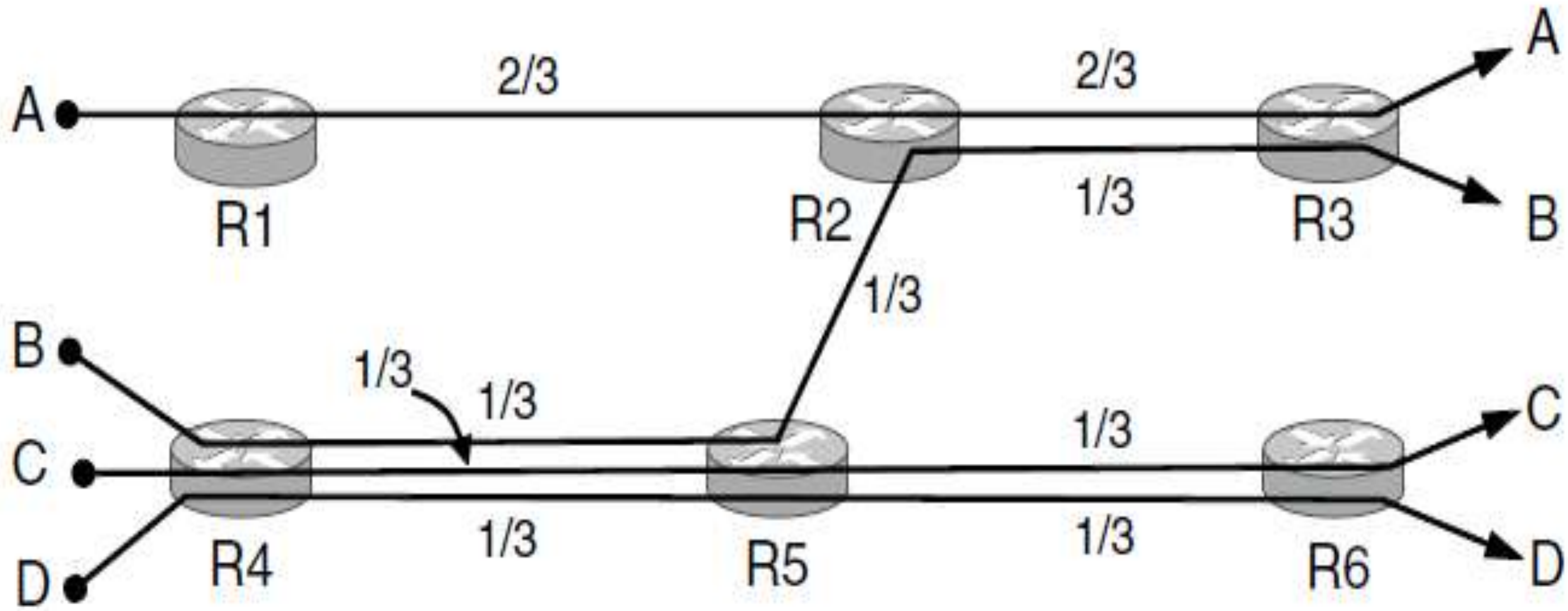
- Desirable bandwidth allocation
- Regulating the sending rate

Desirable Bandwidth Allocation (1)



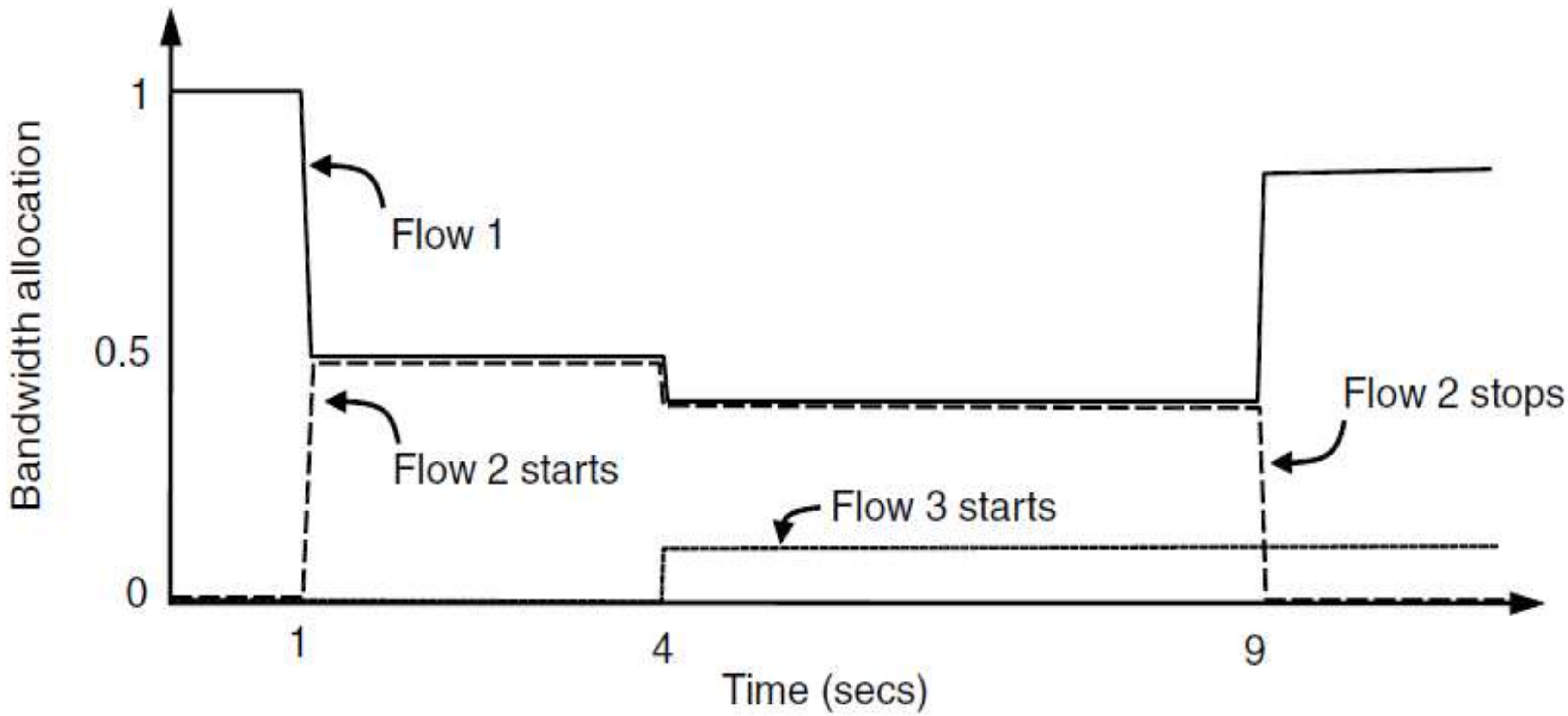
(a) Goodput and (b) delay as a function of offered load

Desirable Bandwidth Allocation (2)



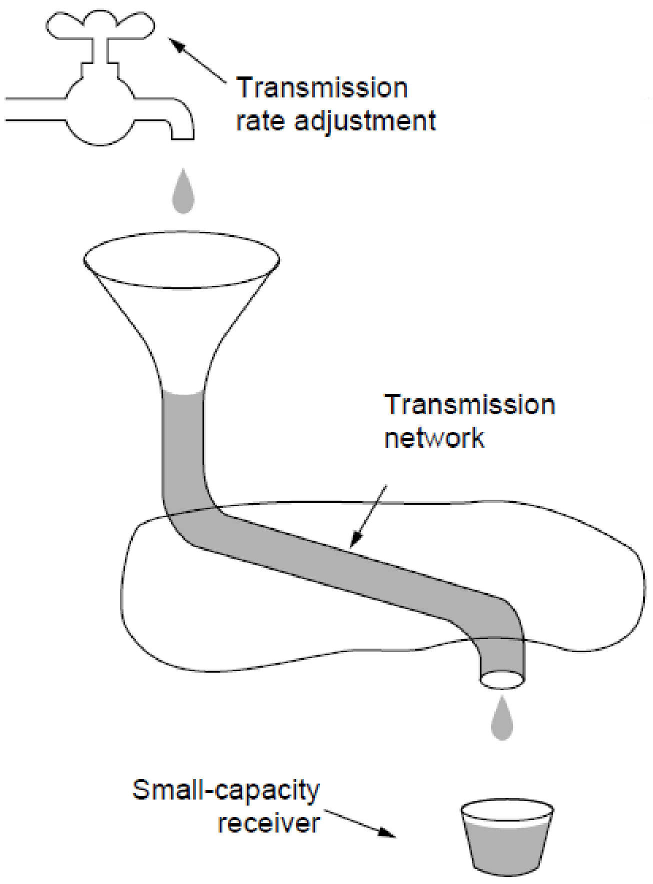
Max-min bandwidth allocation for four flows

Desirable Bandwidth Allocation (3)



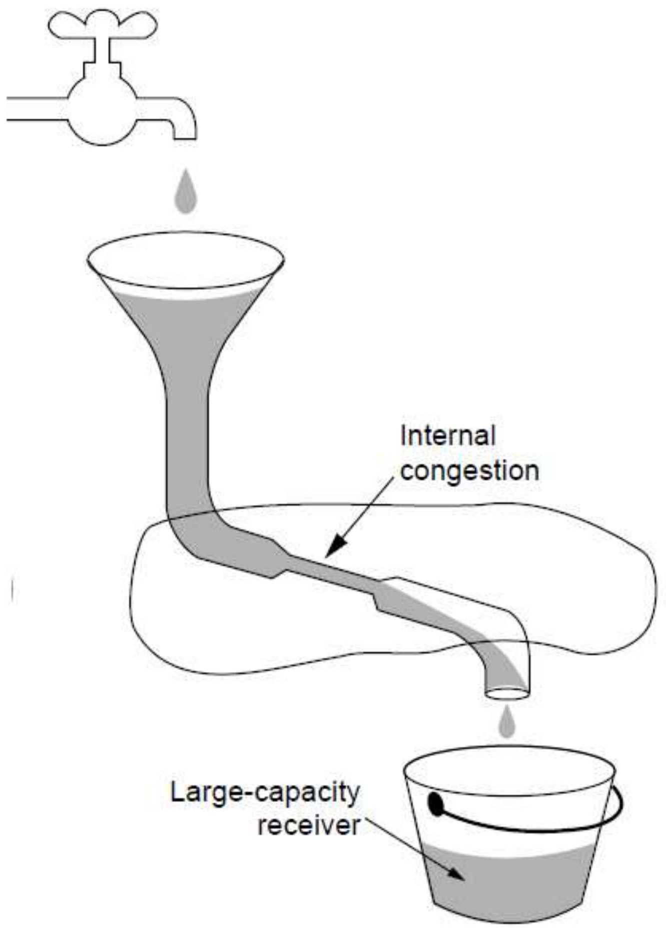
Changing bandwidth allocation over time

Regulating the Sending Rate (1)



A fast network feeding a low-capacity receiver

Regulating the Sending Rate (2)



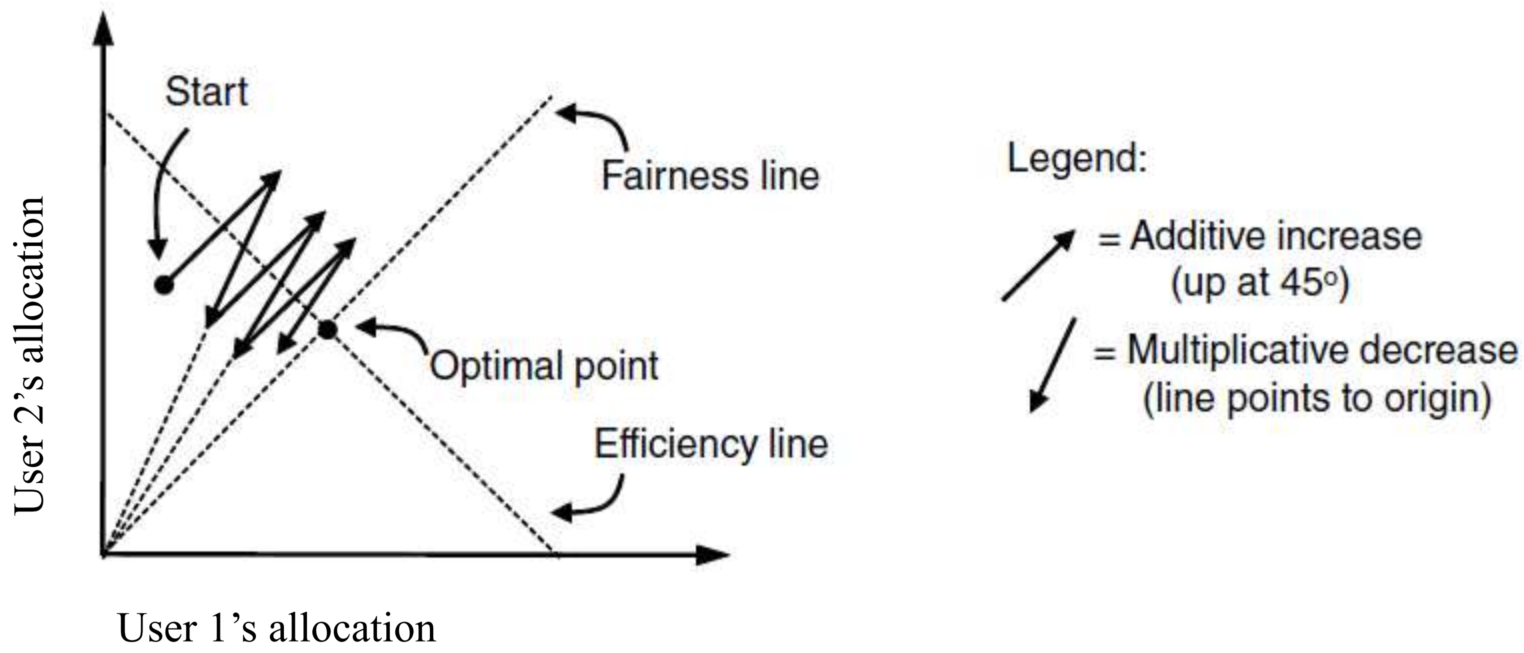
A slow network feeding a high-capacity receiver

Some congestion control protocols

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

Regulating the Sending Rate (3)

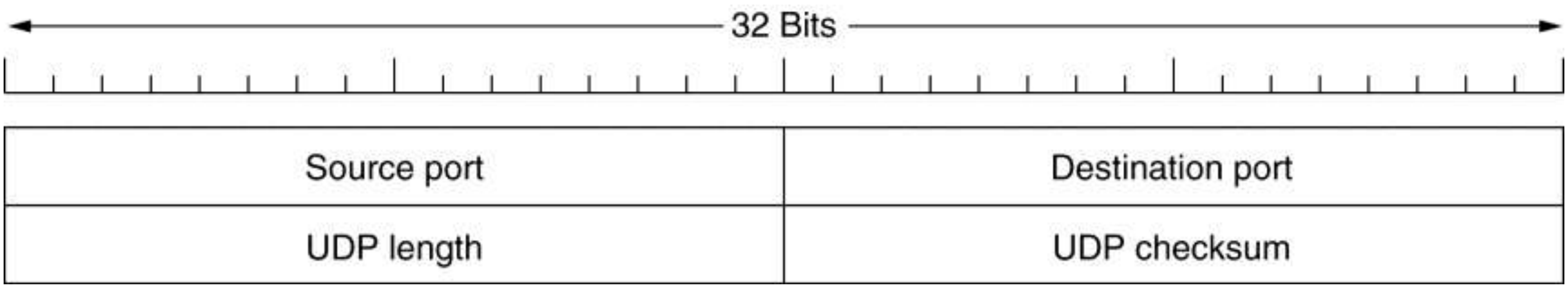
Additive Increase Multiplicative Decrease (AIMD) control law.



The Internet Transport Protocols: UDP

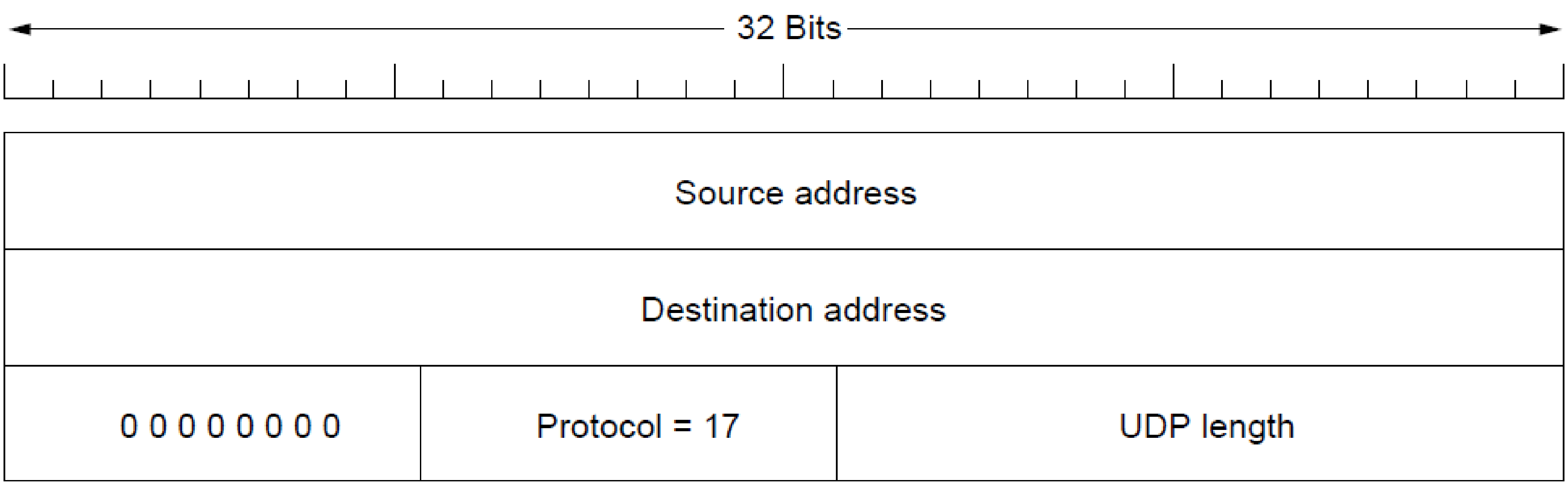
a) Introduction to UDP
b) Remote Procedure Call
c) The Real-Time Transport Protocol

Introduction to UDP



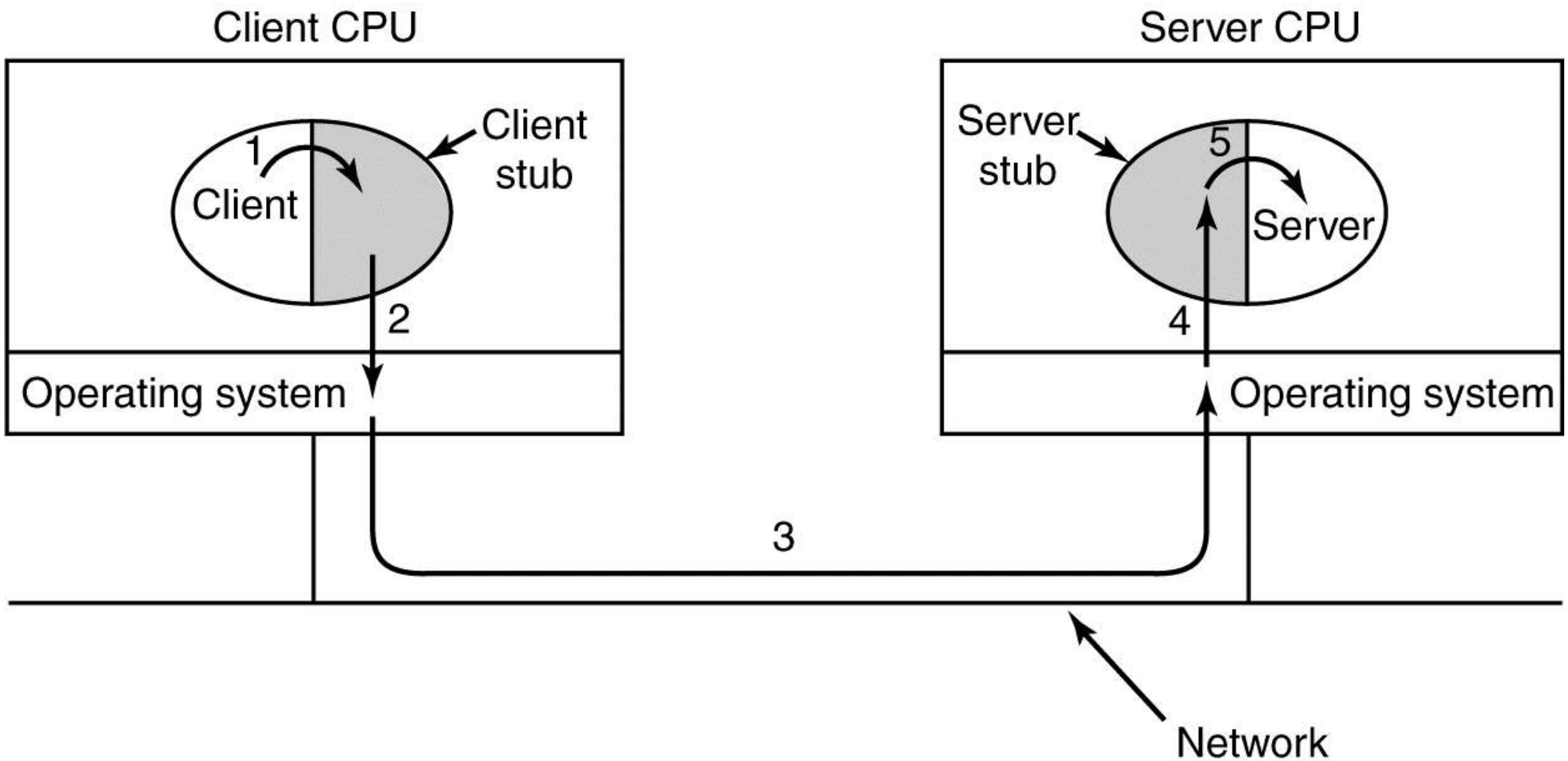
The UDP header.

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.
Marshaling: packing the parameters

Some problem of RPC

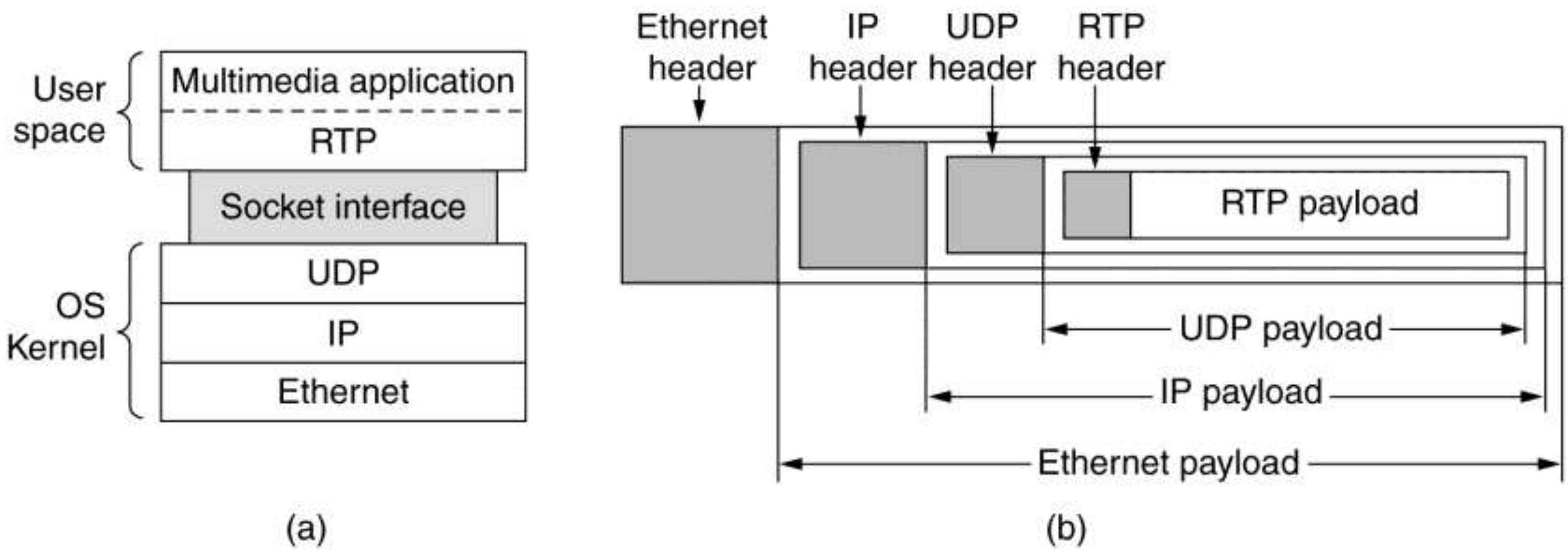
a) How to passing a pointer parameter
   – call-reference → copy-restore

b) How to know the size of a parameter
   – Char p[]

c) How to know the type of a parameter
   – Sprintf("", …)

d) How to pass global variables

The Real-Time Transport Protocol



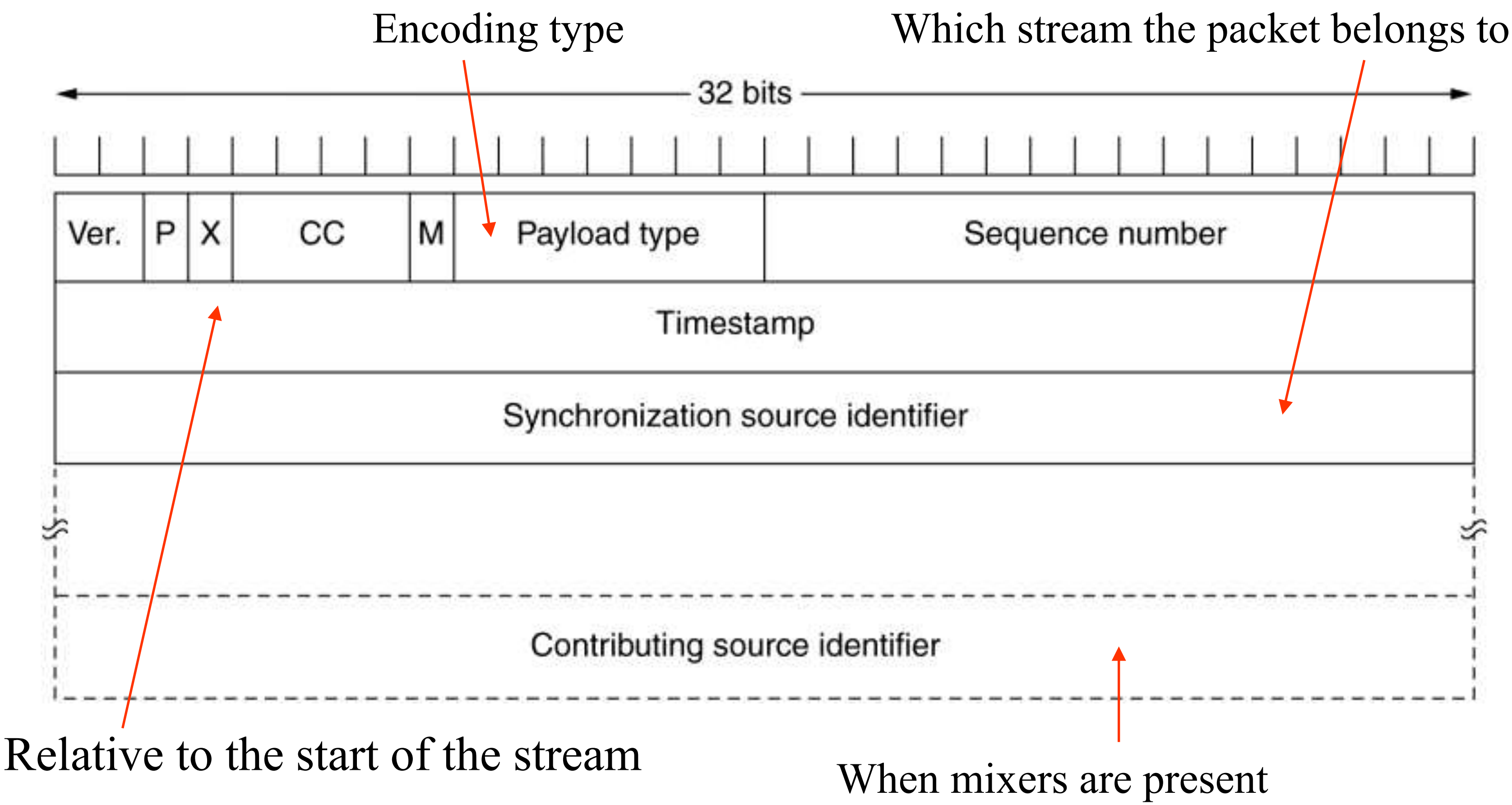
(a) The position of RTP in the protocol stack. (b) Packet nesting.

No flow control, no error control, no ack, no retransmission

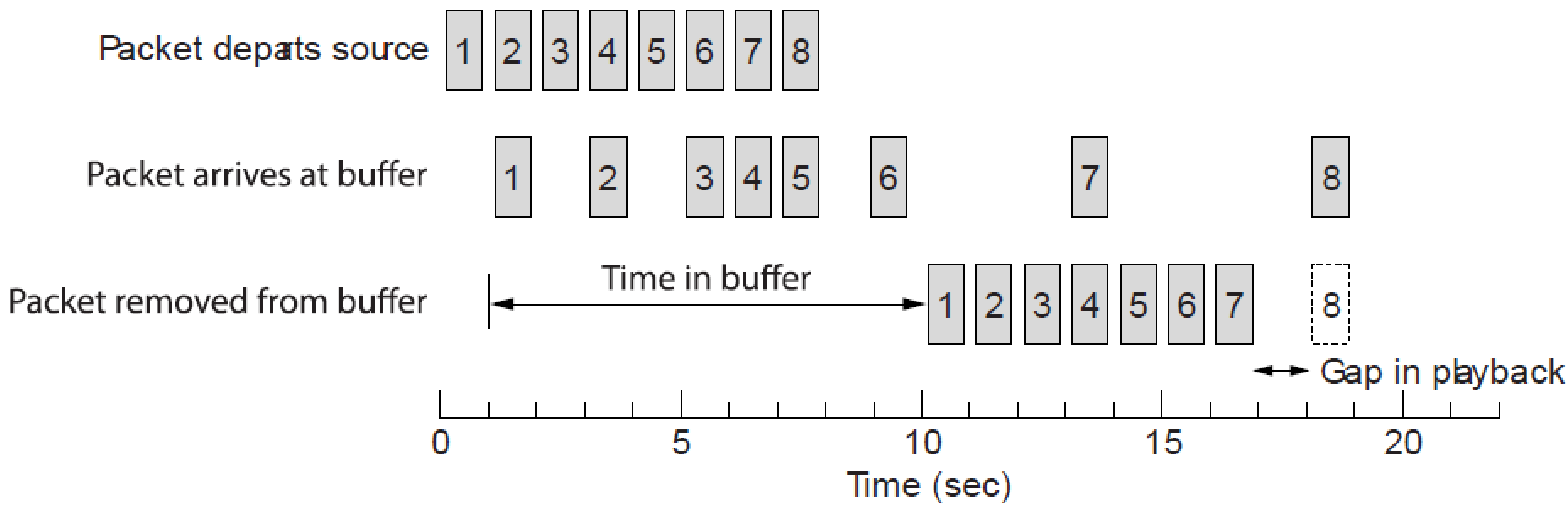
Payload may contain multiple samples, and they may be coded any way

The RTP header

P: Padded to multiple of 4 bytes
X: Extension head present
CC: How many contributing sources are present
M: marker bit for application specific

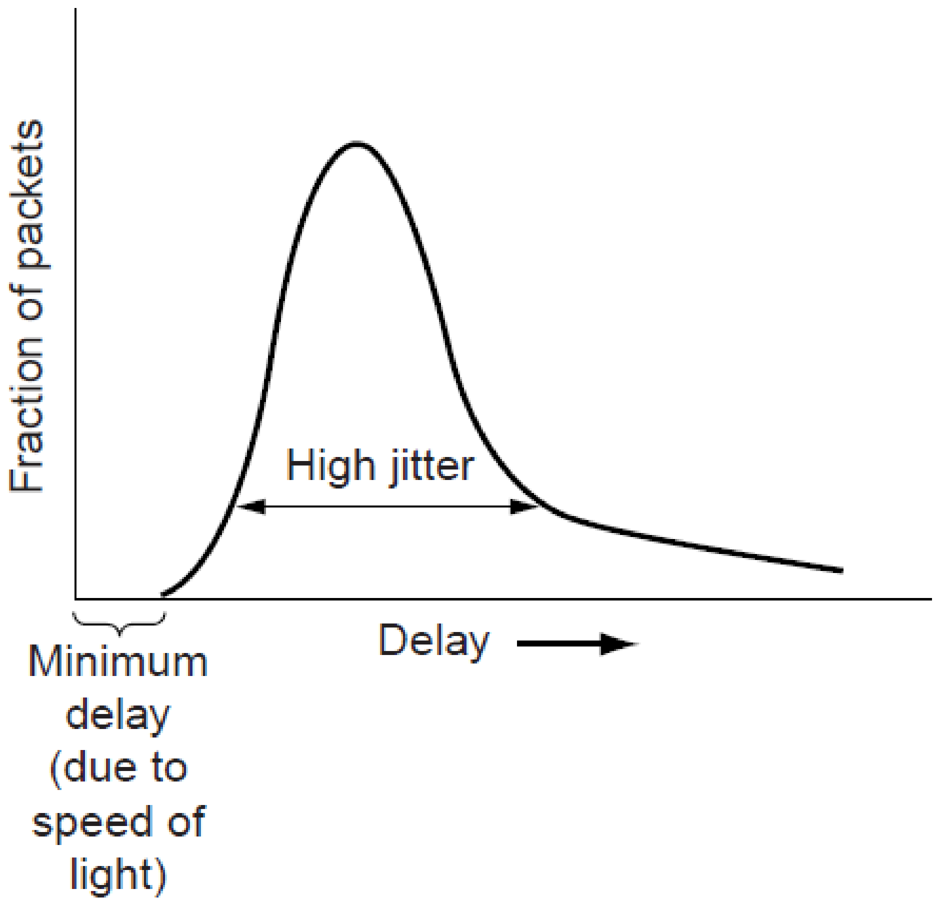


RTP: Smoothing the output stream



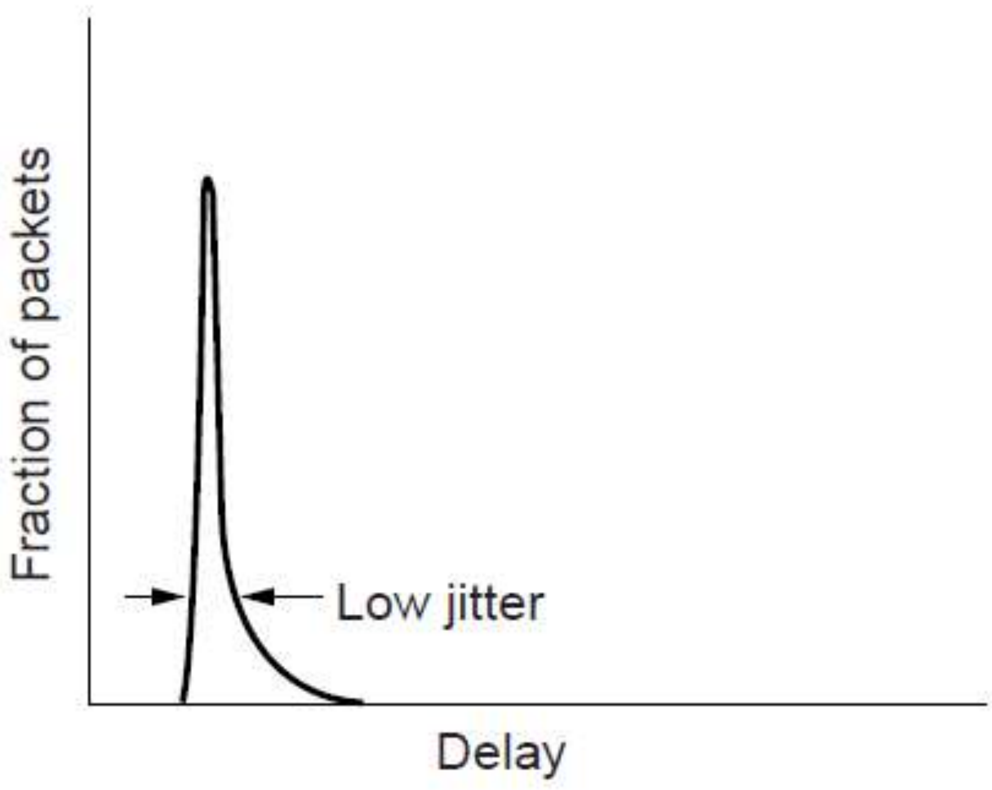
Smoothing the output stream by buffering packets

RTP: High Jitter



High jitter

RTP: Low Jitter



Low jitter

The Internet Transport Protocols: TCP

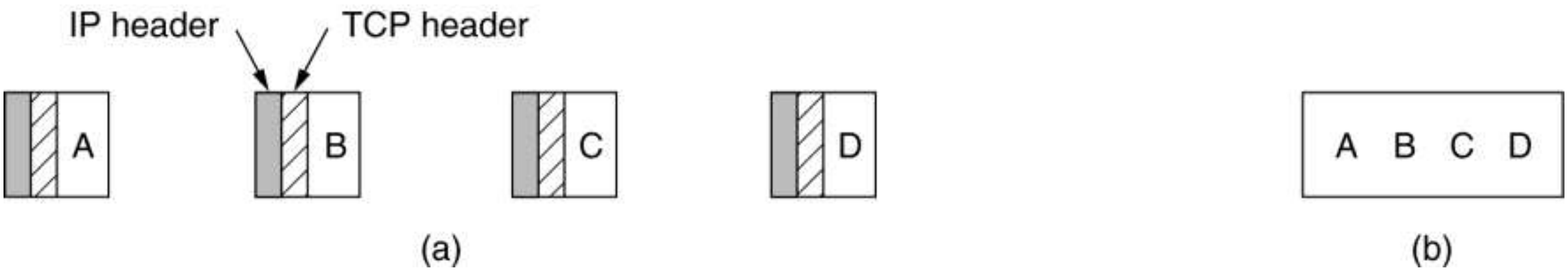
a) Introduction to TCP
b) The TCP Service Model
c) The TCP Protocol
d) The TCP Segment Header
e) TCP Connection Establishment
f) TCP Connection Release
g) TCP Connection Management Modeling
h) TCP Transmission Policy
i) TCP Congestion Control
j) TCP Timer Management
k) Wireless TCP and UDP
l) Transactional TCP

Some assigned ports

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 well known ports

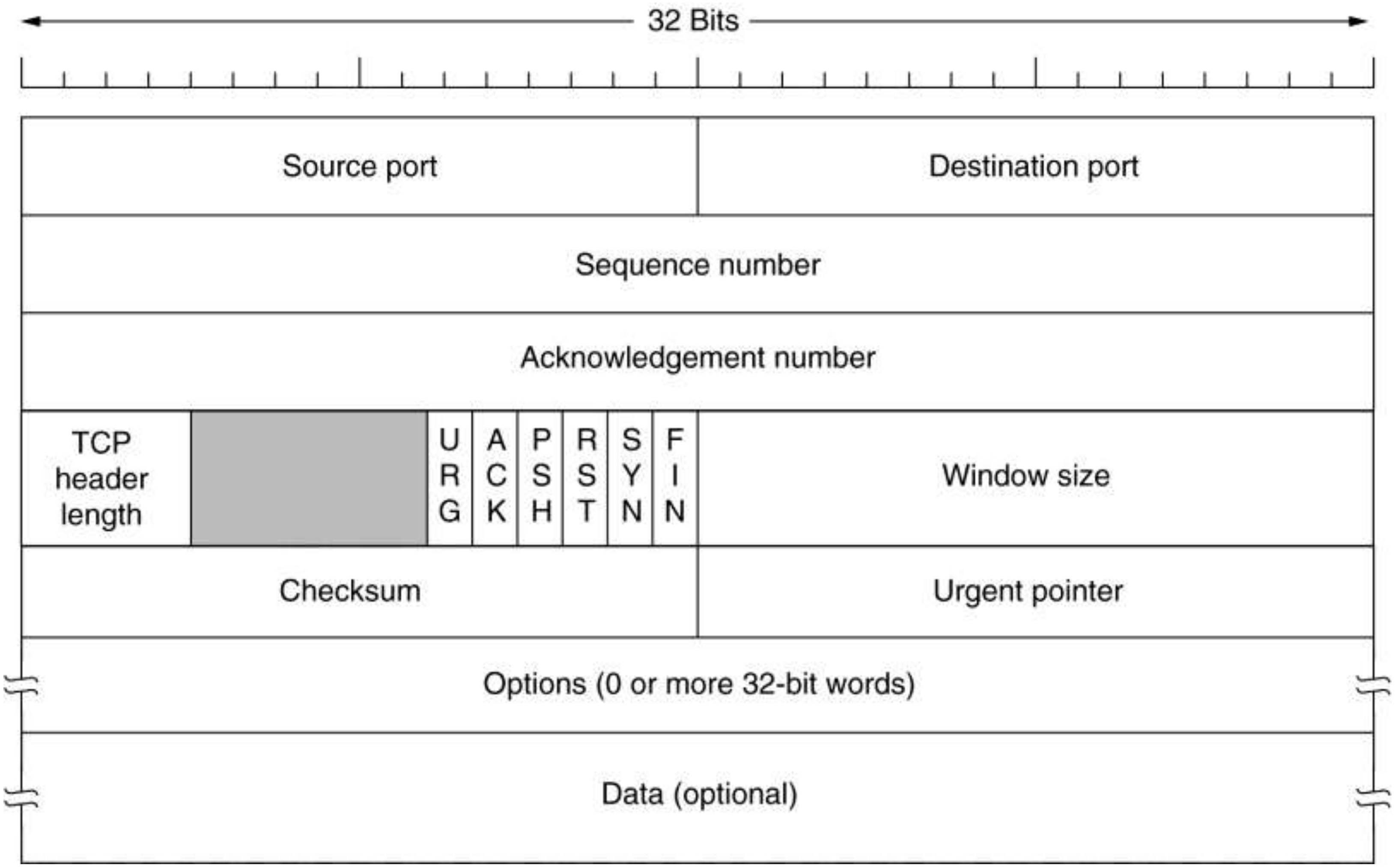
The TCP Service Model



(a) Four 512-byte segments sent as separate IP datagrams.

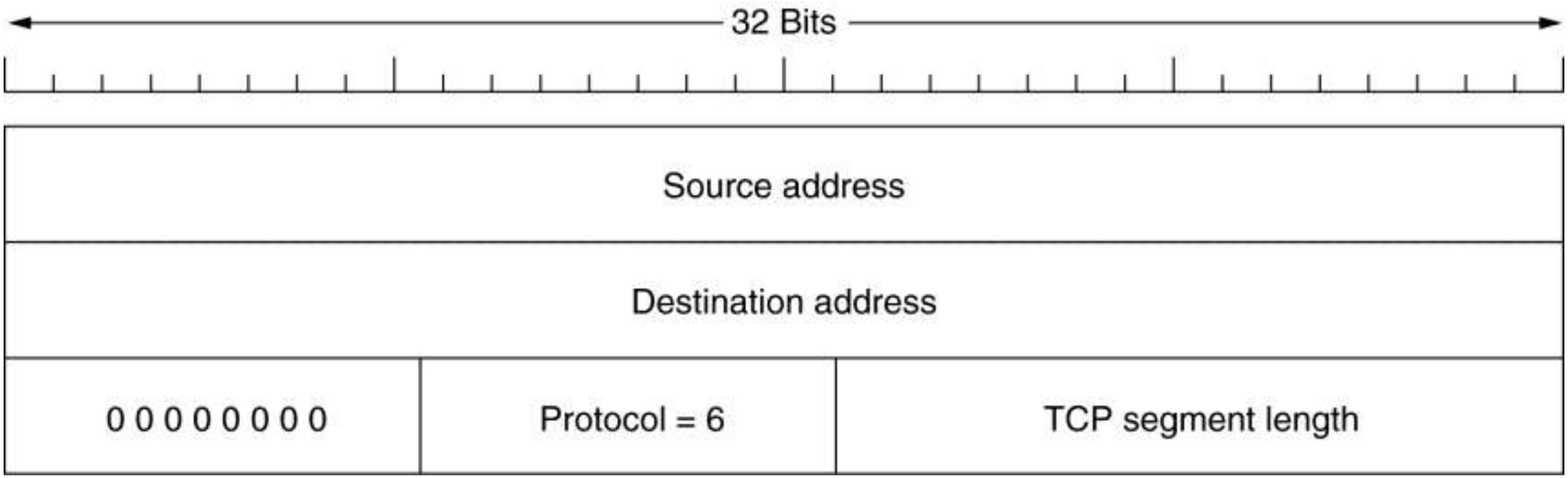
(b) The 2048 bytes of data delivered to the application in a single READ CALL.

The TCP Segment Header



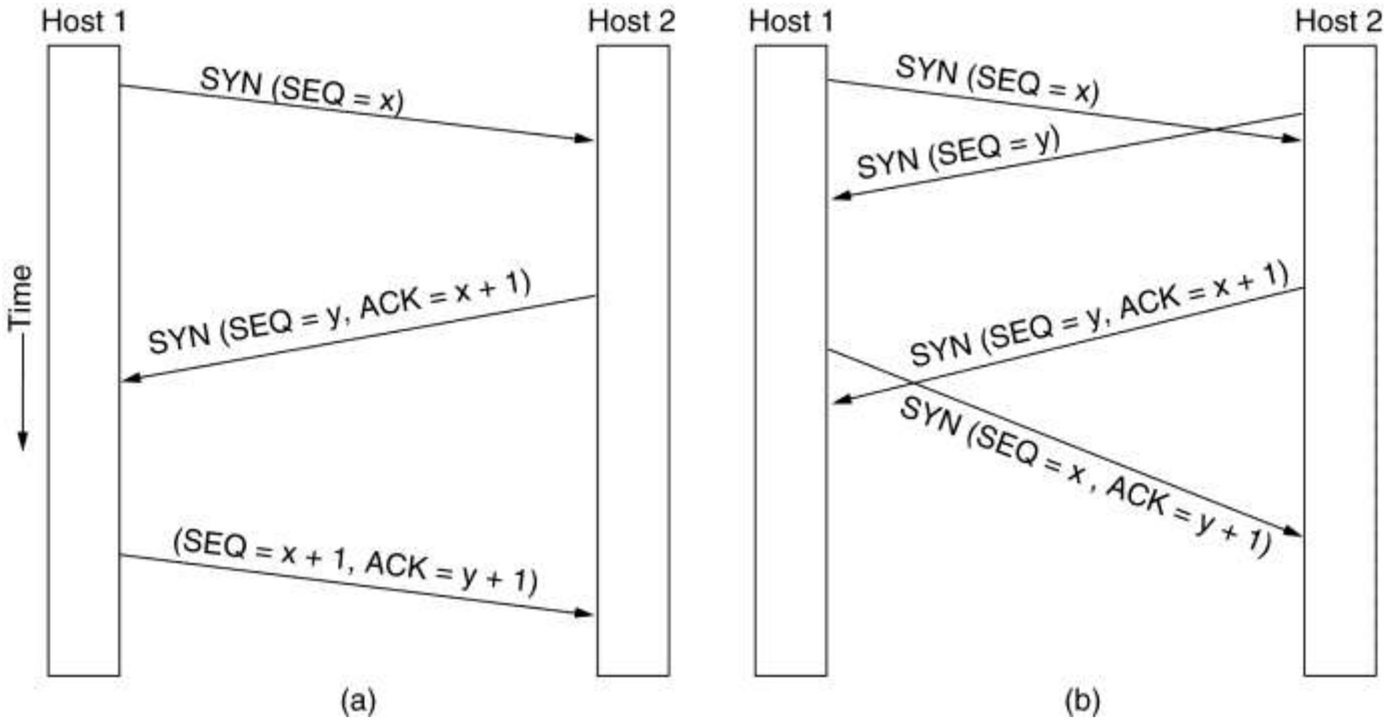
TCP Header.

The TCP Segment Pseudoheader



The pseudoheader included in the TCP checksum.

TCP Connection Establishment



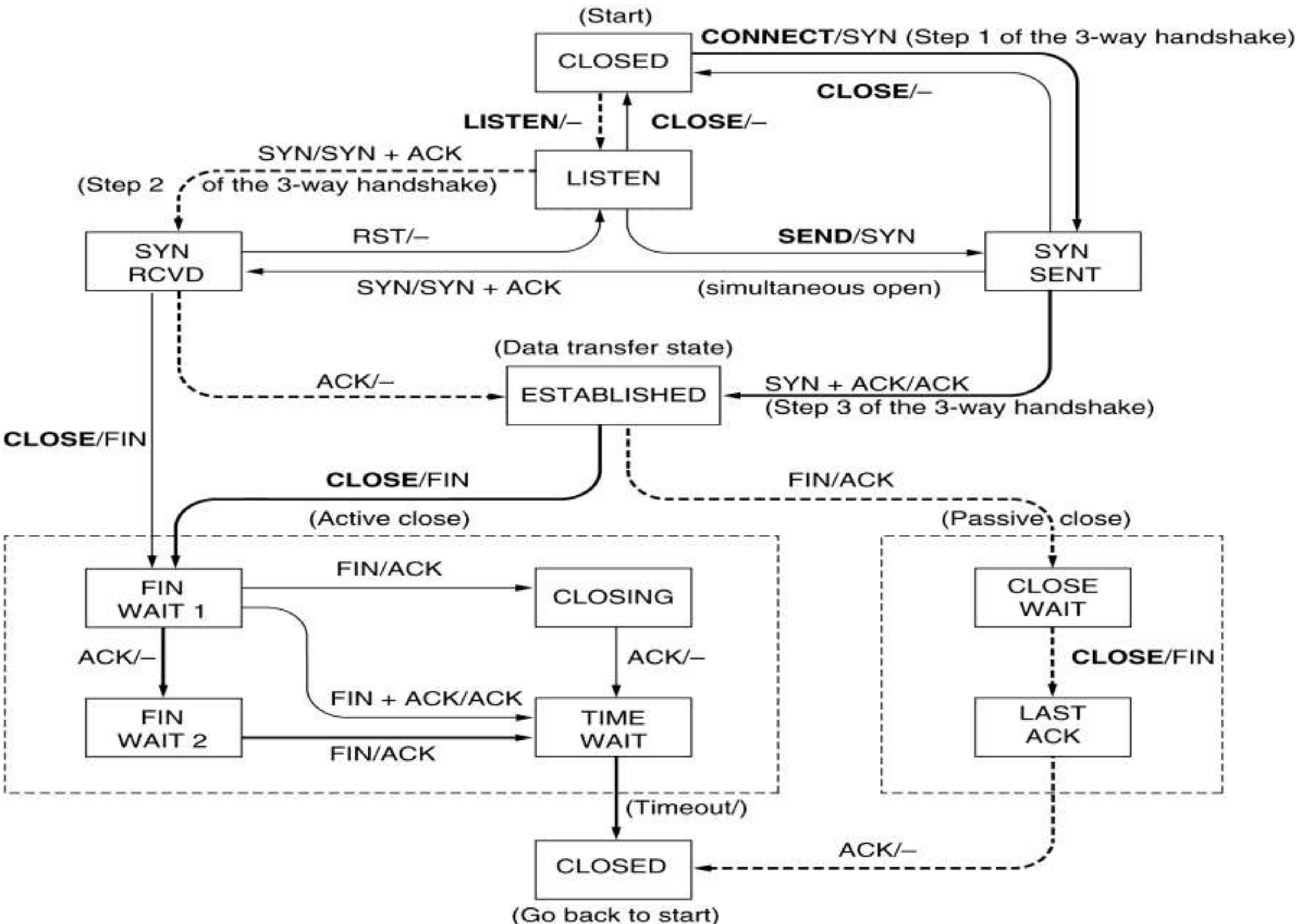
(a) TCP connection establishment in the normal case.
(b) Call collision.

TCP Connection Management Modeling

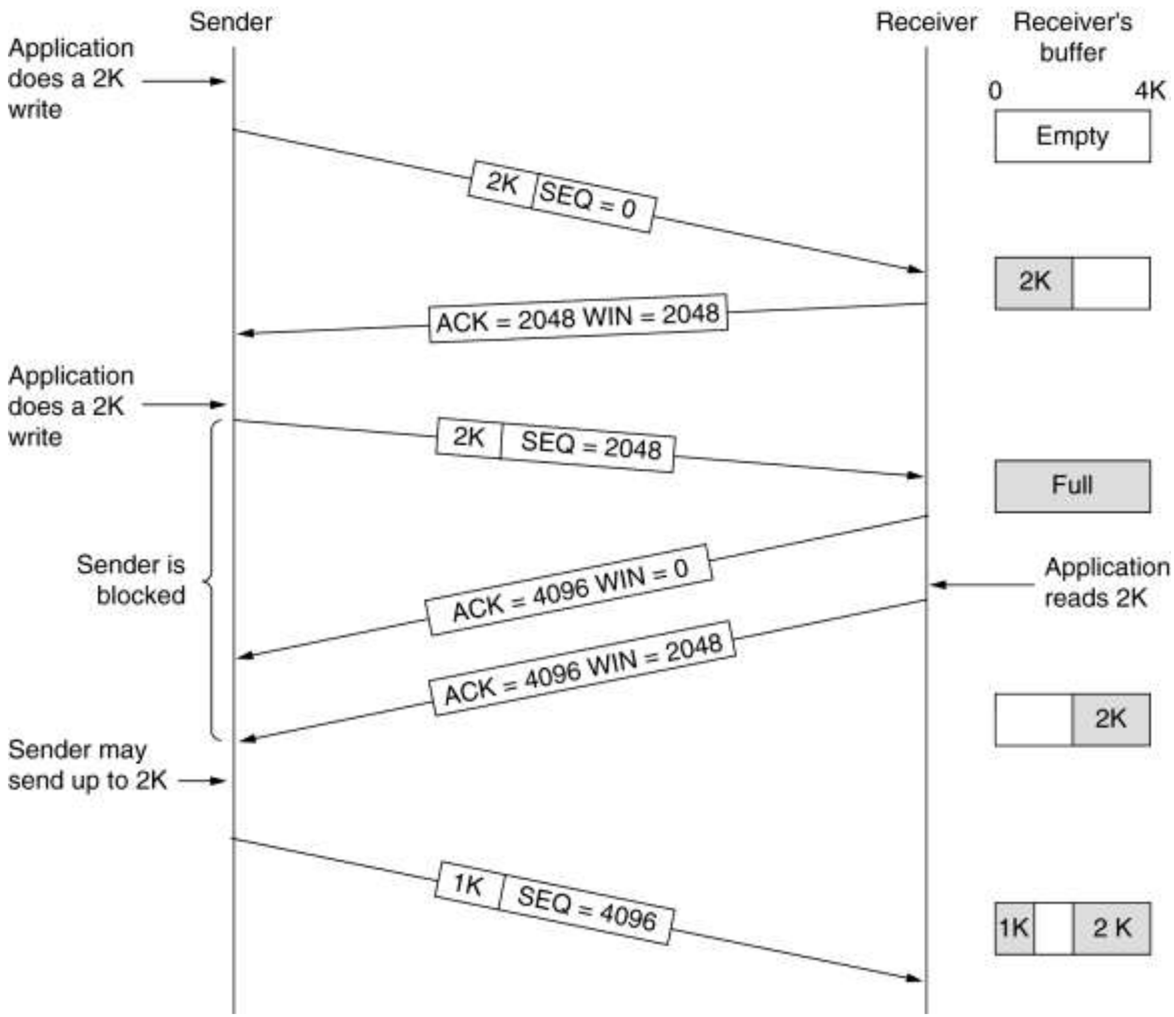
State	Description
CLOSED	No connection is active or pending
LISTEN	The server is waiting for an incoming call
SYN RCVD	A connection request has arrived; wait for ACK
SYN SENT	The application has started to open a connection
ESTABLISHED	The normal data transfer state
FIN WAIT 1	The application has said it is finished
FIN WAIT 2	The other side has agreed to release
TIMED 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 finite state machine

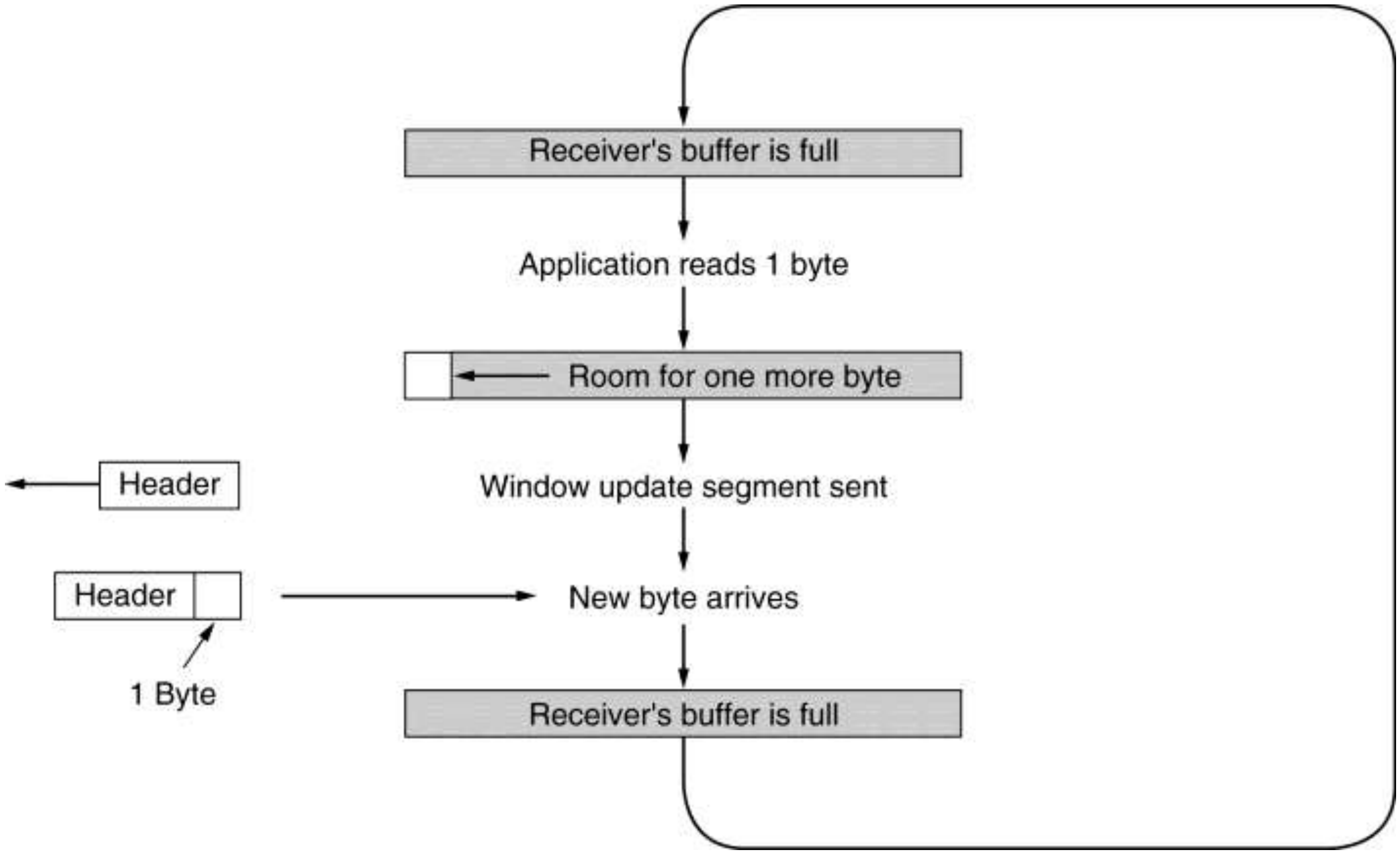


TCP Transmission Policy (Sliding Window)



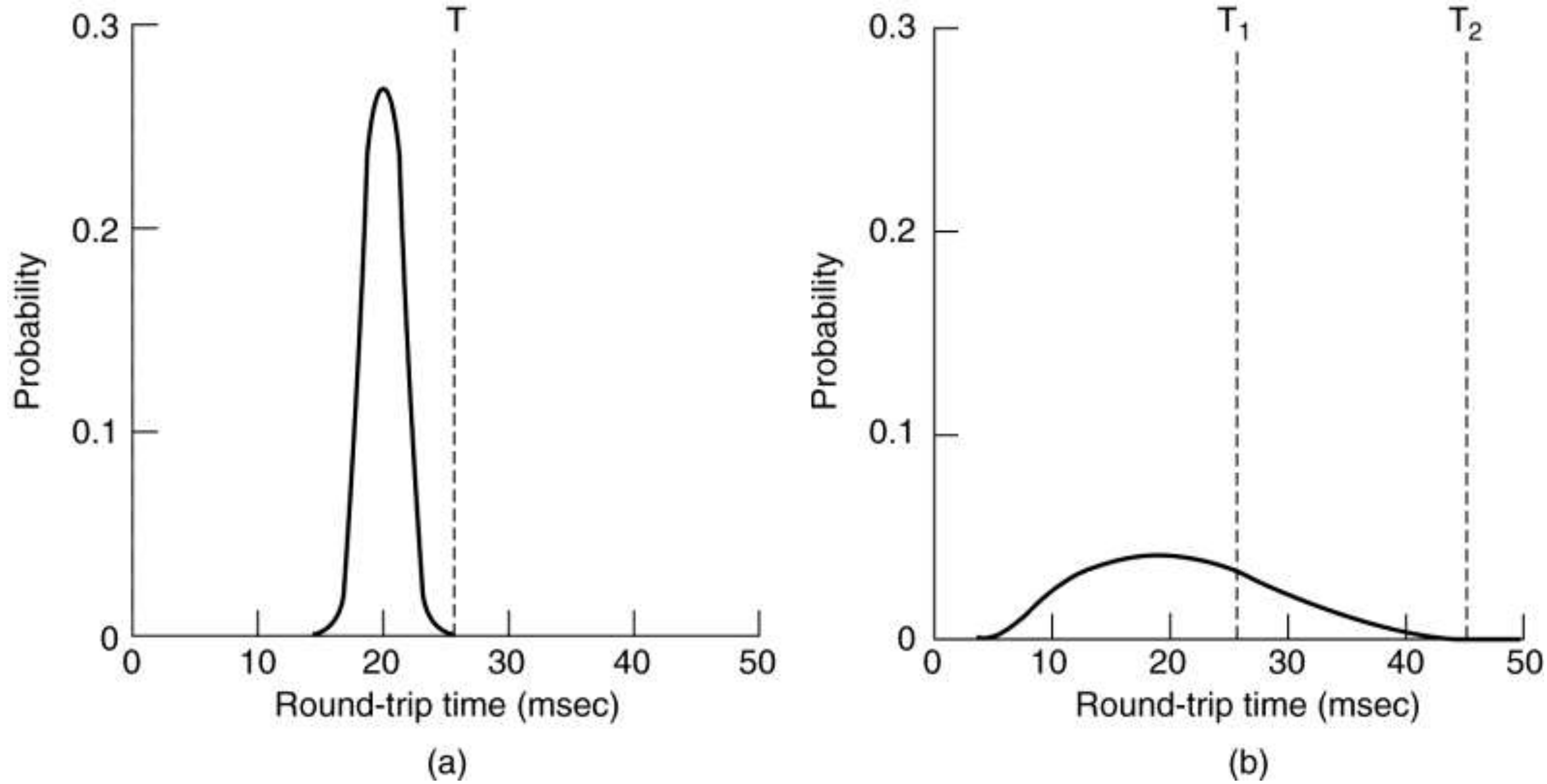
Window management in TCP.

TCP Transmission Policy (2)



Silly window syndrome.

TCP Timer Management



(a) Probability density of ACK arrival times in the data link layer.

(b) Probability density of ACK arrival times for TCP.

TCP: Timer Management

- **Retransmission timer**
  - RTT (Round-Trip Time)

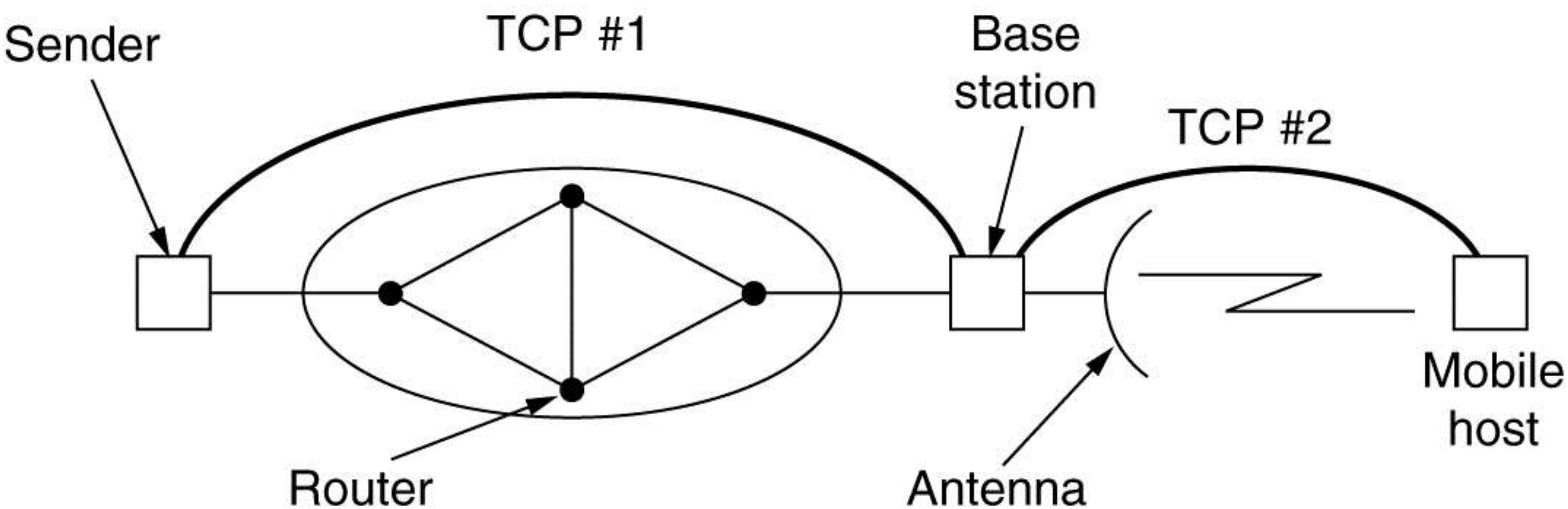
    RTT = alpha RTT + (1-alpha)M
  - Mean deviation

    D = alpha D + (1-alpha) | RTT –M |
  - **Timeout**

    = RTT + 4 x D
- **Persistence timer**: to prevent the deadlock.
- **Keepalive timer**: to check whether the other side is still there.
- Other timers such as the one used in the TIMED WAIT state.

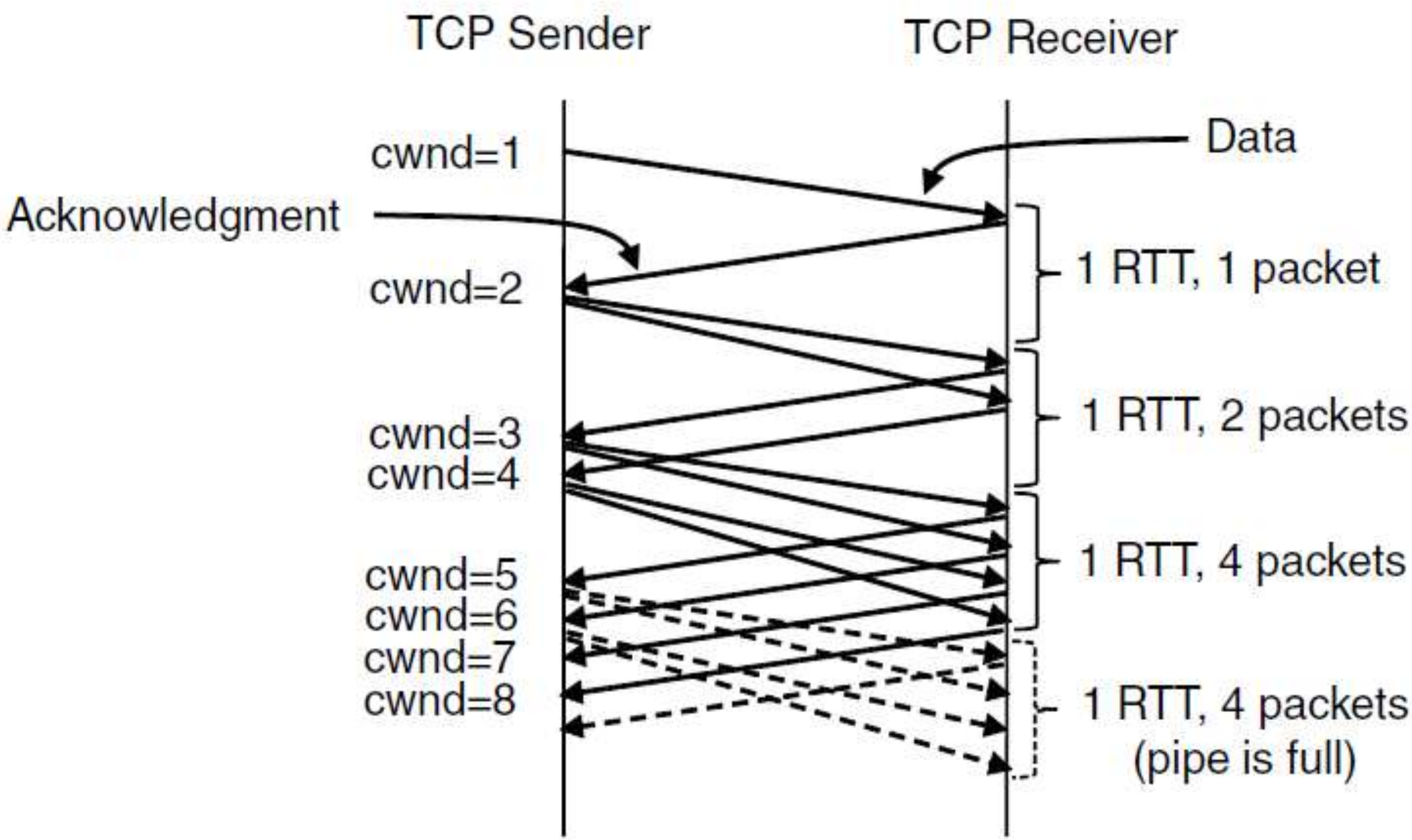
TCP: Wireless TCP and UDP

- Congestion control
  - Wired TCP: to slow down and send less.
  - Wireless TCP: to send ASAP.
  - Wired and Wireless TCP
    - Indirect TCP



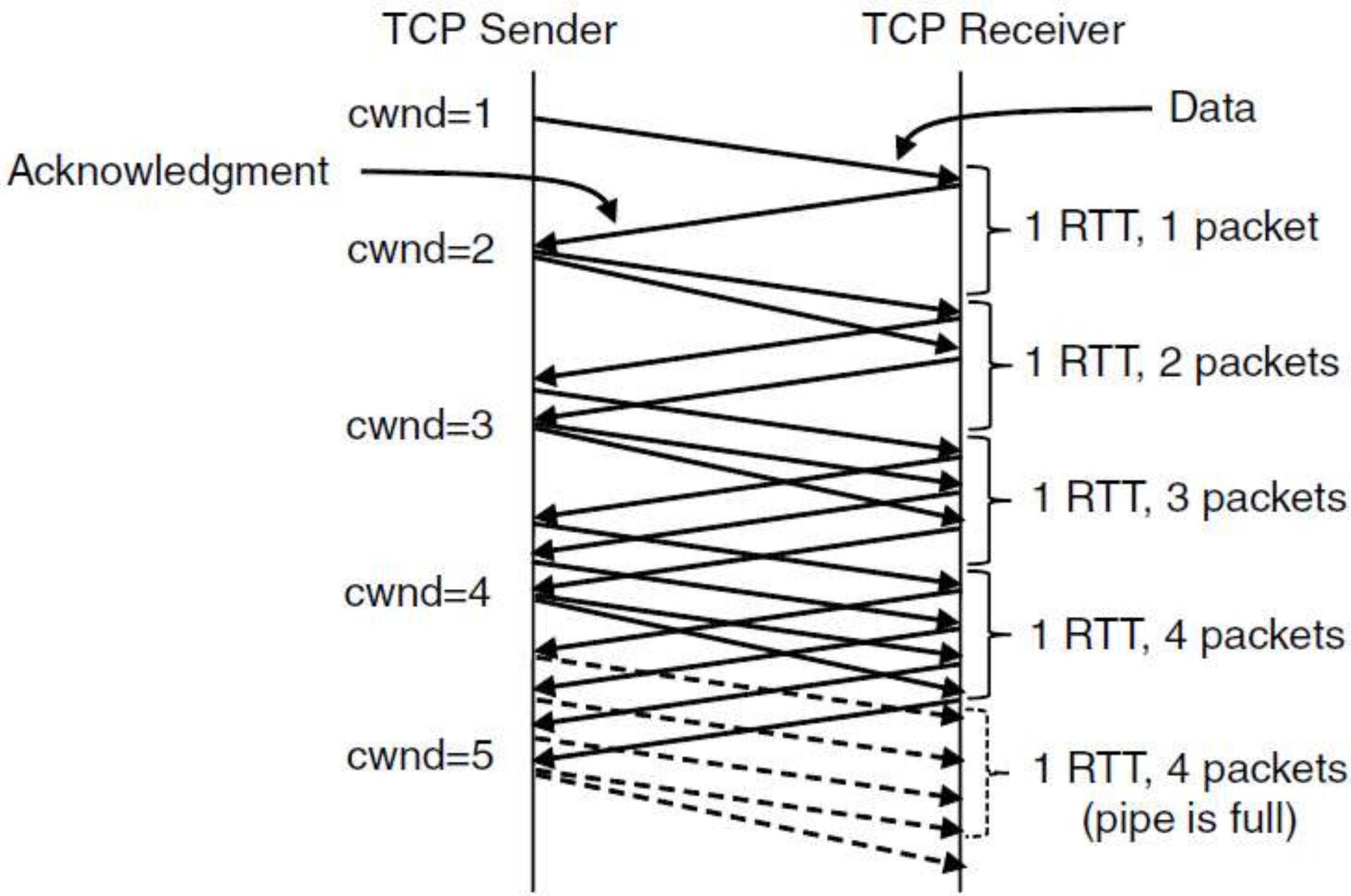
    - Snooping agent for sending to mobile hosts

TCP Congestion Control (1)



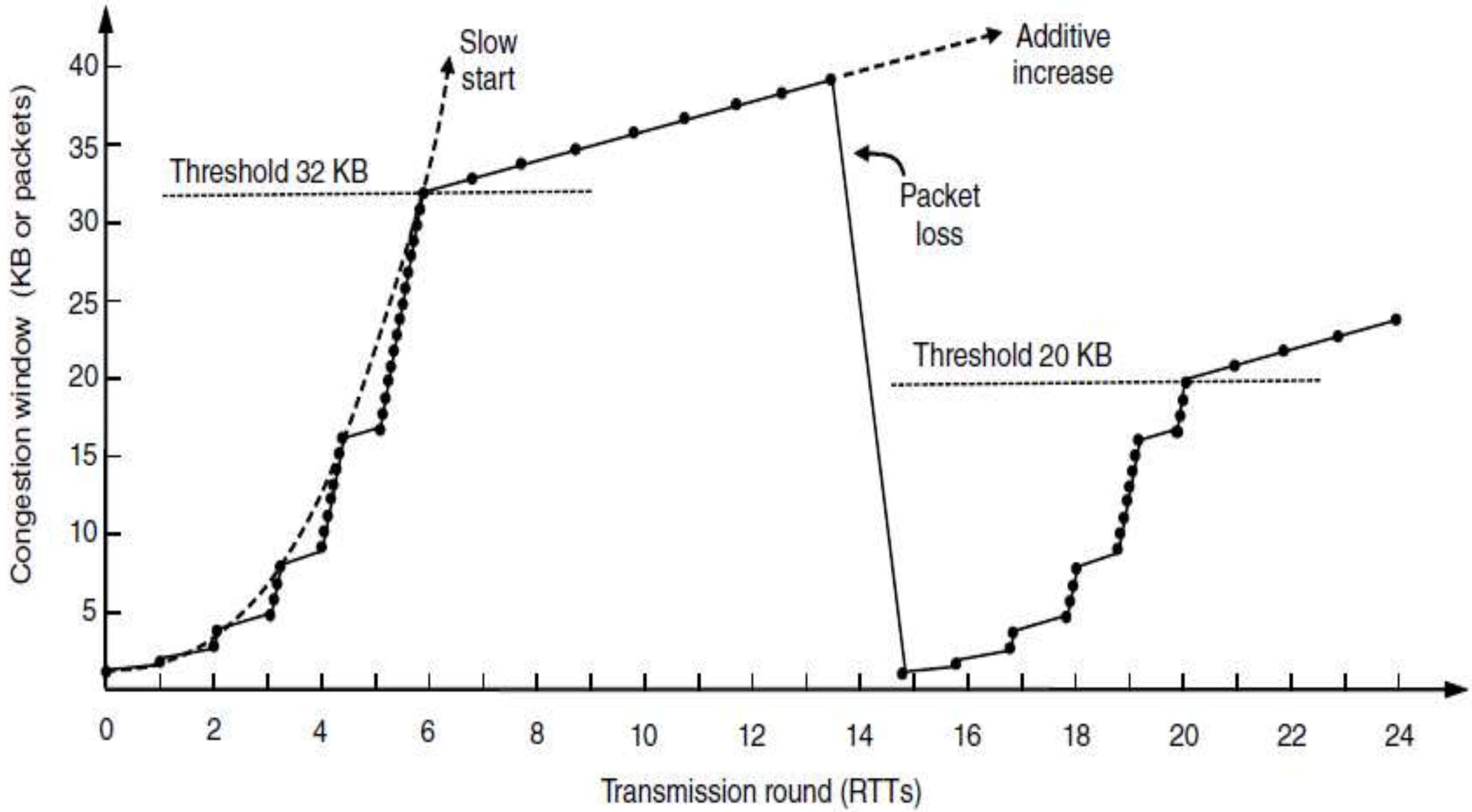
Slow start from an initial congestion window of 1 segment

TCP Congestion Control (2)



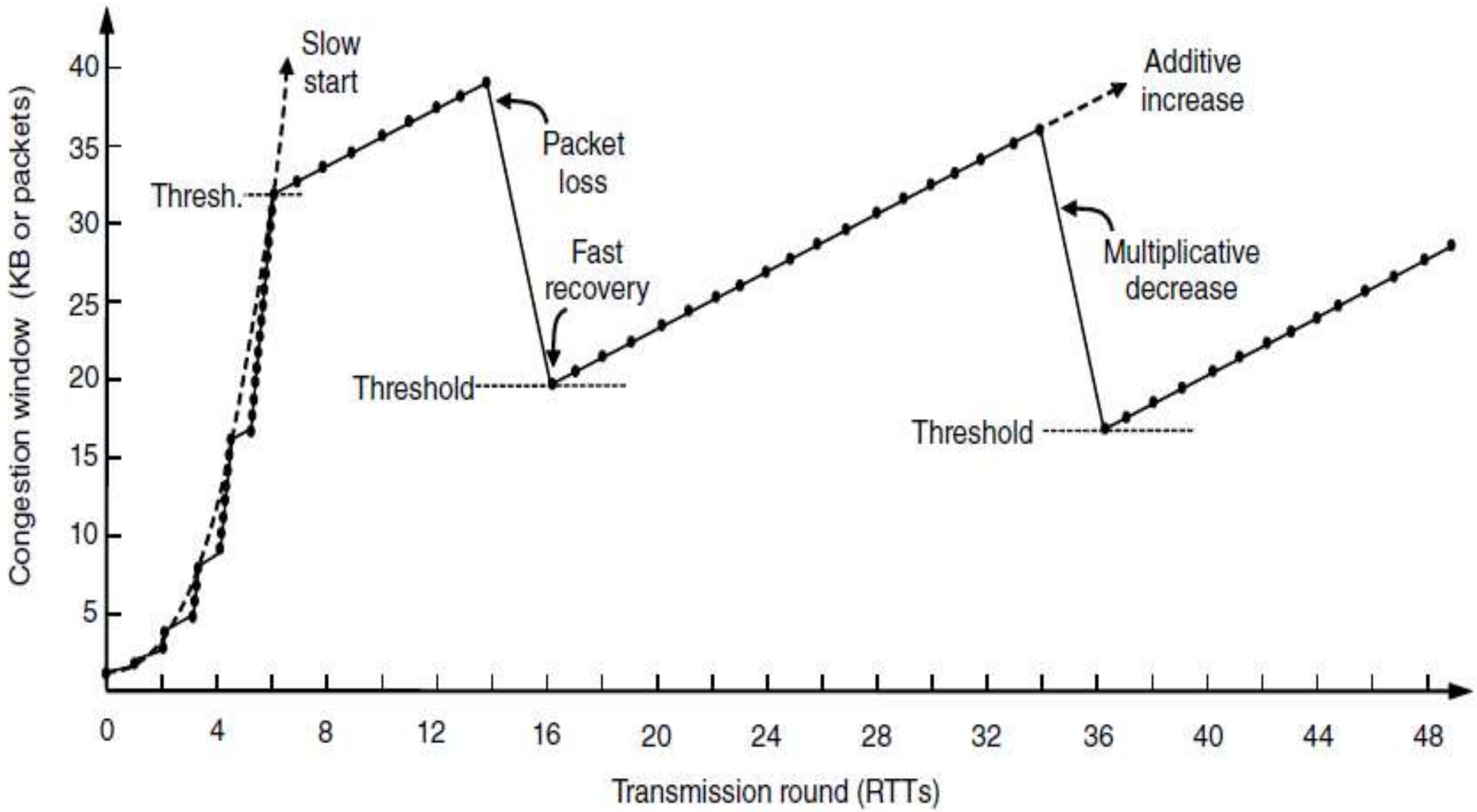
Additive increase from an initial congestion window of 1 segment.

TCP Congestion Control (3)



Slow start followed by additive increase in TCP Tahoe.

TCP Congestion Control (4)

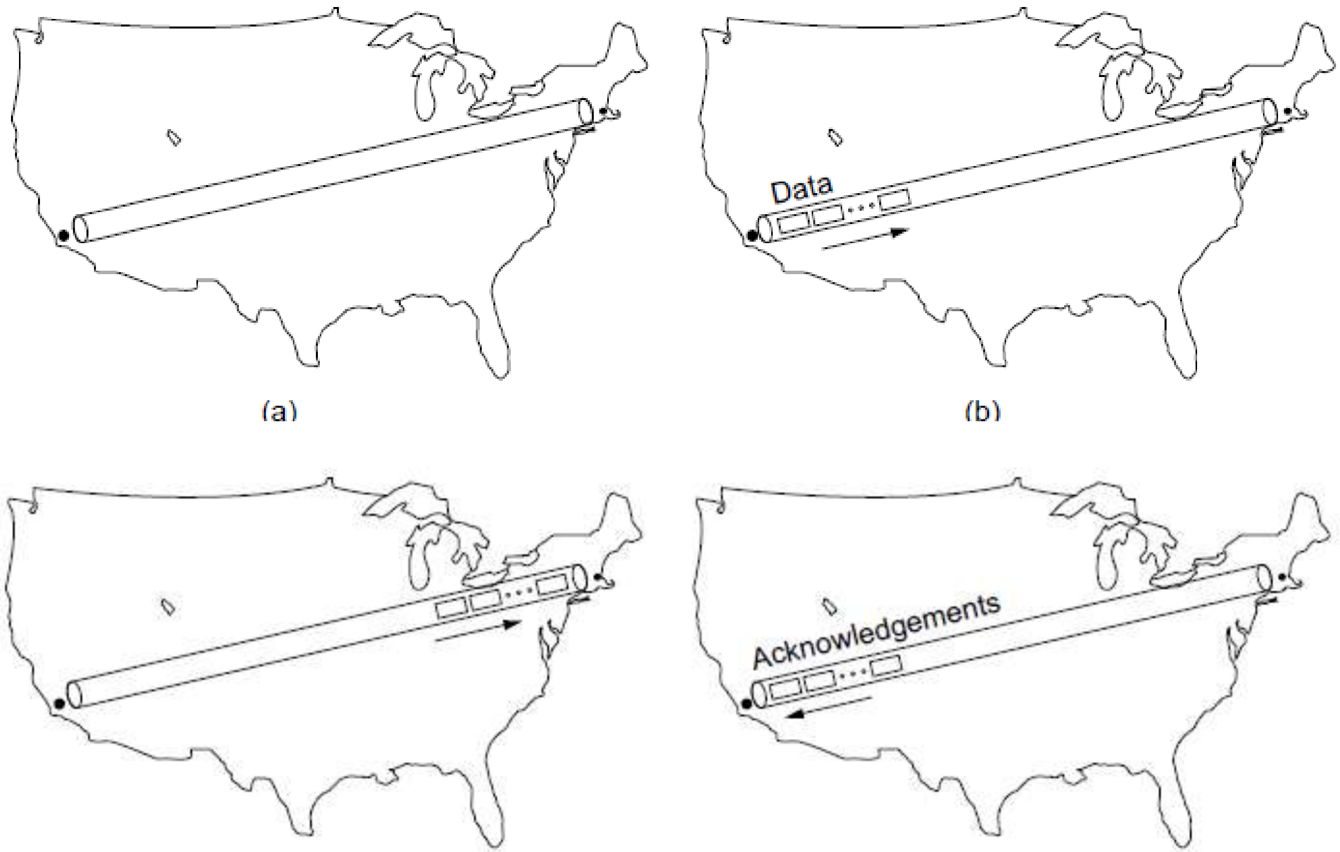


Fast recovery and the sawtooth pattern of TCP Reno.

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

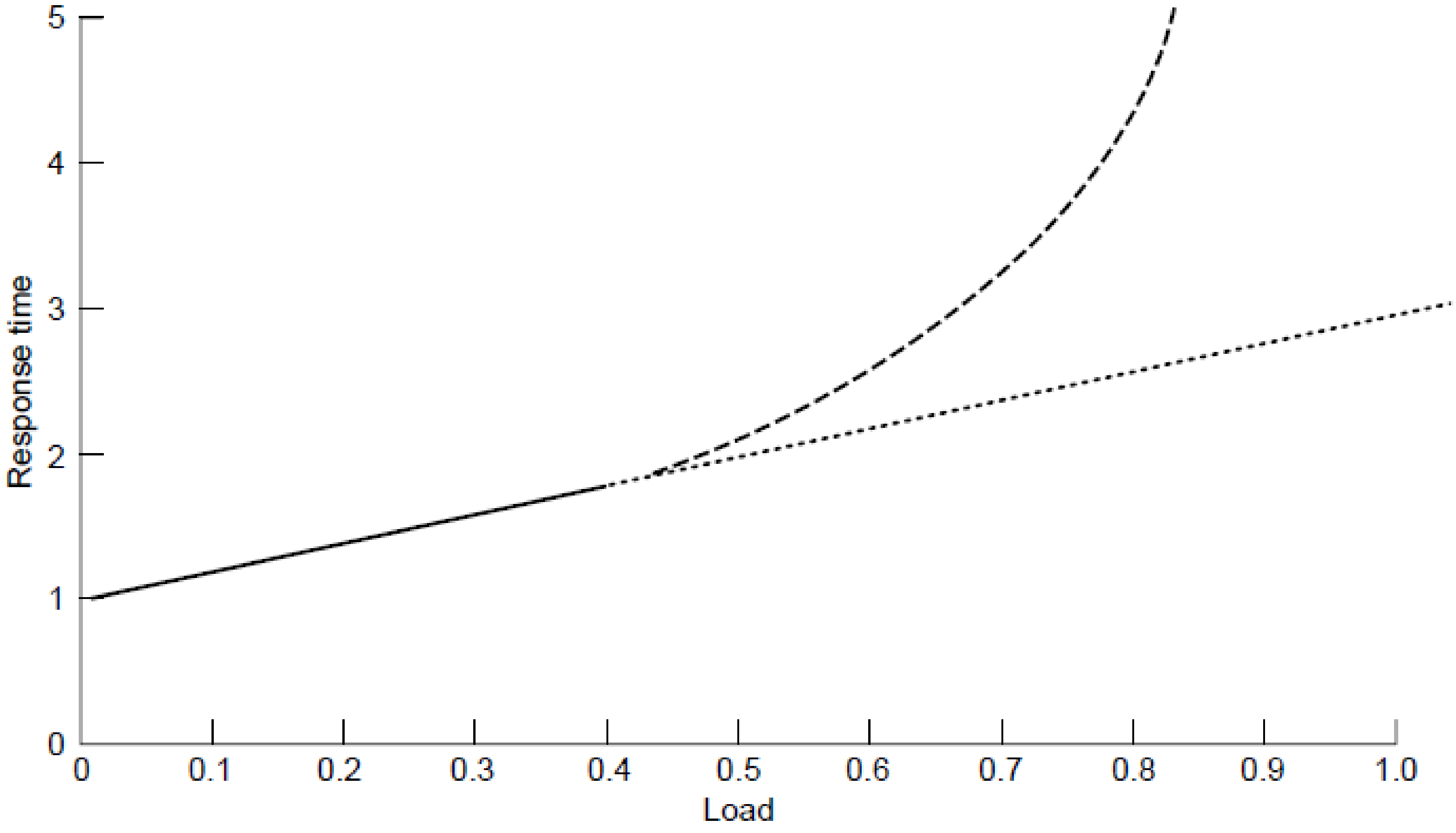
1. Measure relevant network parameters, performance.
2. Try to understand what is going on.
3. Change one parameter.

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)



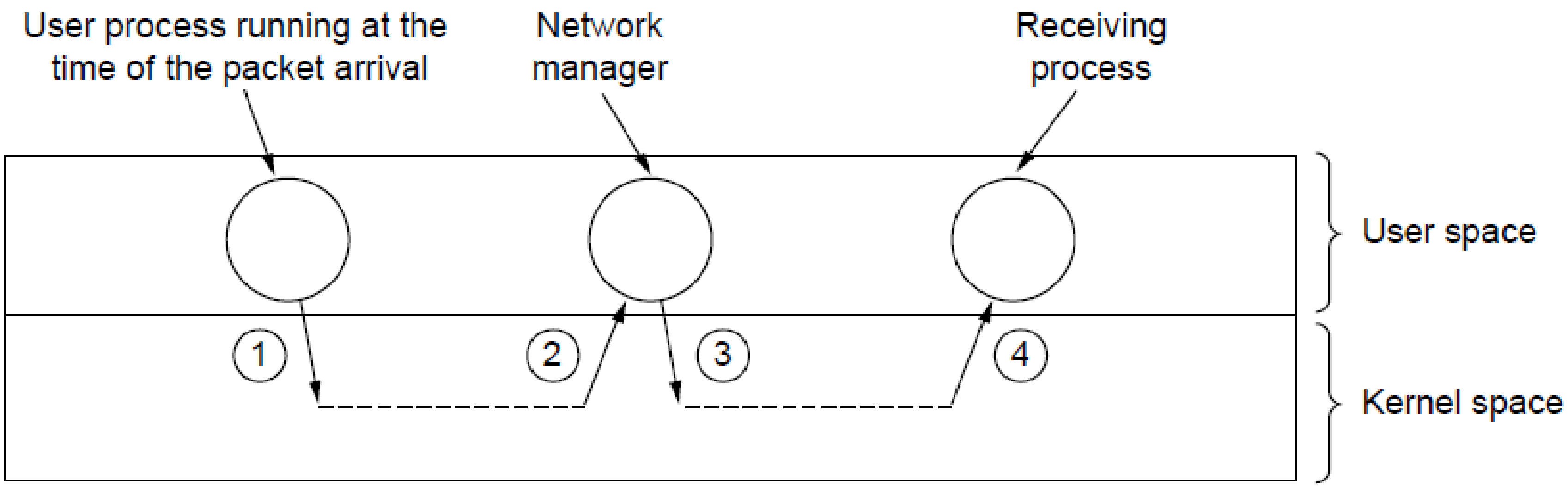
Response as a function of load.

System Design for Better Performance (1)

Rules of thumb

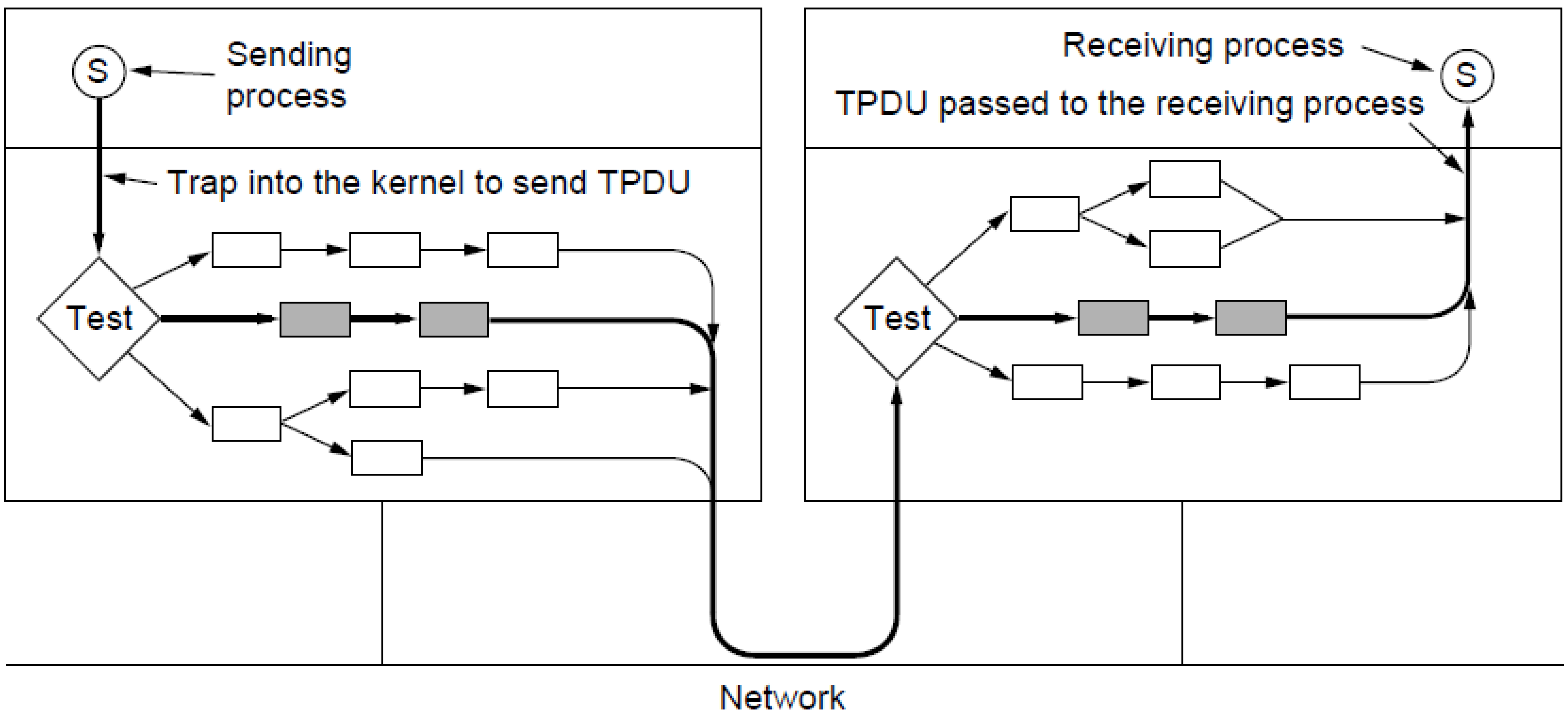
1. CPU speed more important than network speed
2. Reduce packet count to reduce software overhead
3. Minimize data touching
4. Minimize context switches
5. Minimize copying
6. You can buy more bandwidth but not lower delay
7. Avoiding congestion is better than recovering from it
8. Avoid timeouts

System Design for Better Performance (2)



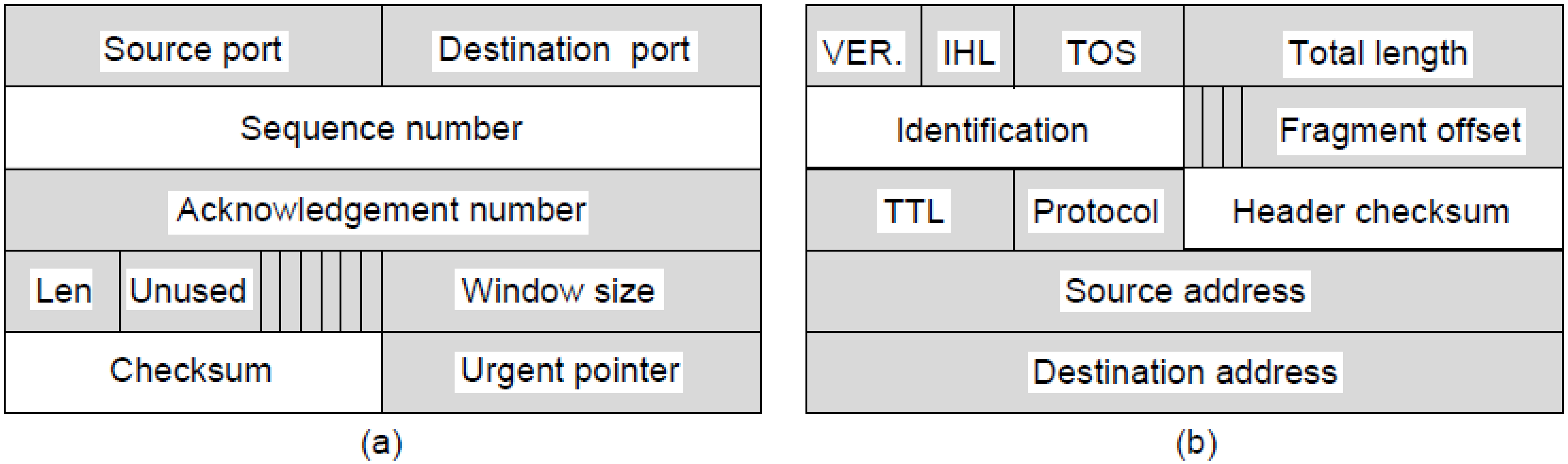
Four context switches to handle one packet with a user-space network manager.

Fast TPDU Processing (1)



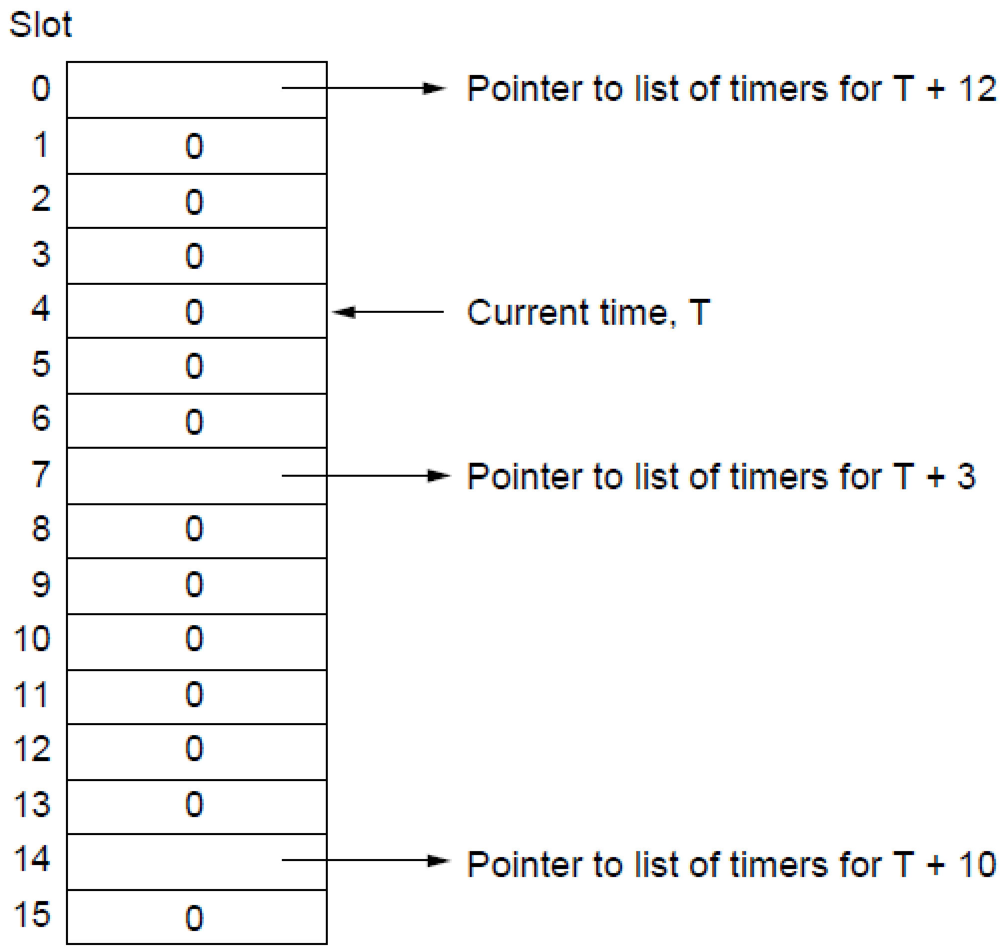
The fast path from sender to receiver is shown with a heavy line.
The processing steps on this path are shaded.

Fast TPDU Processing (2)



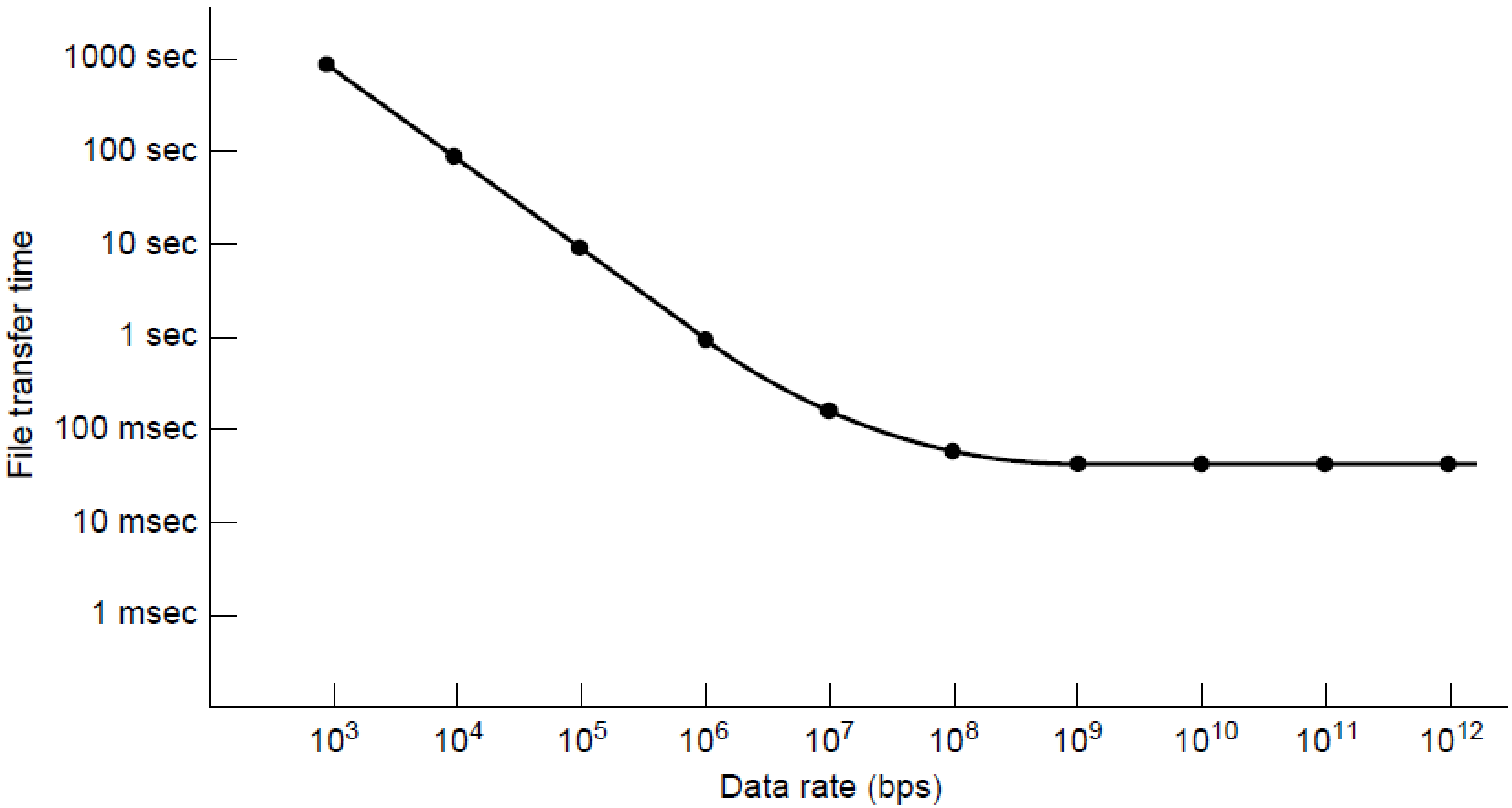
(a) TCP header. (b) IP header. In both cases, the shaded fields are taken from the prototype without change.

Protocols for High-Speed Networks (1)



A timing wheel

Protocols for High-Speed Networks (2)

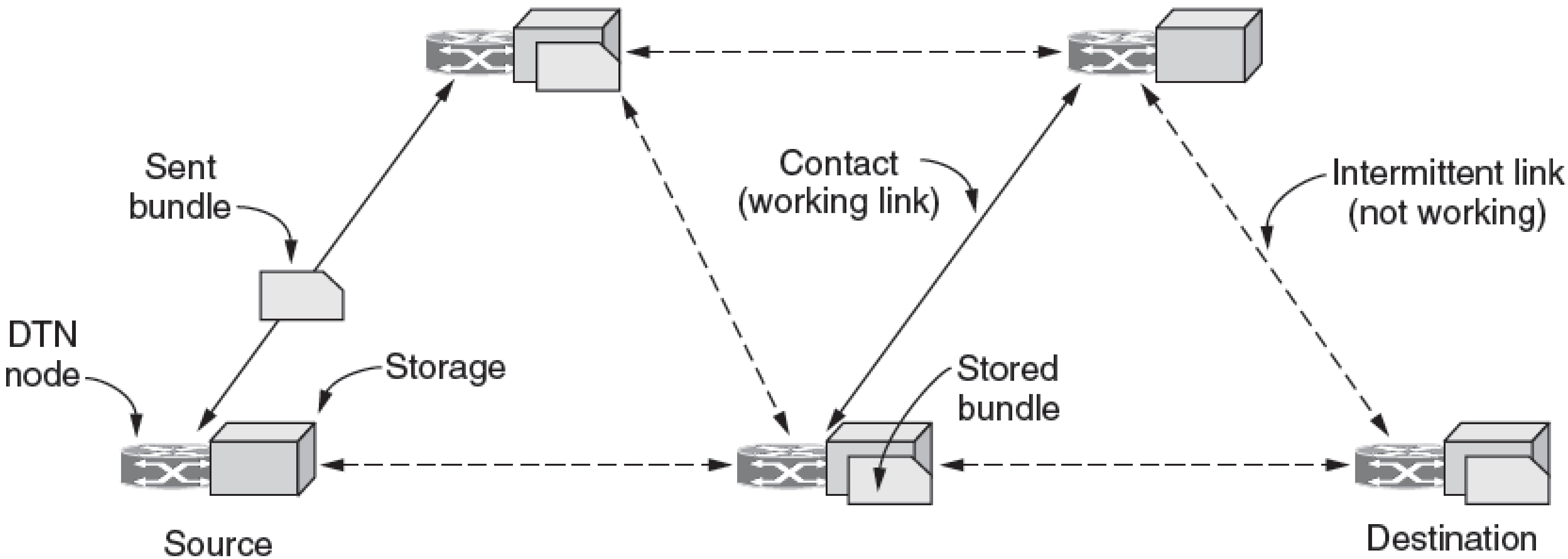


Time to transfer and acknowledge a
1-megabit file over a 4000-km line

Delay Tolerant Networking

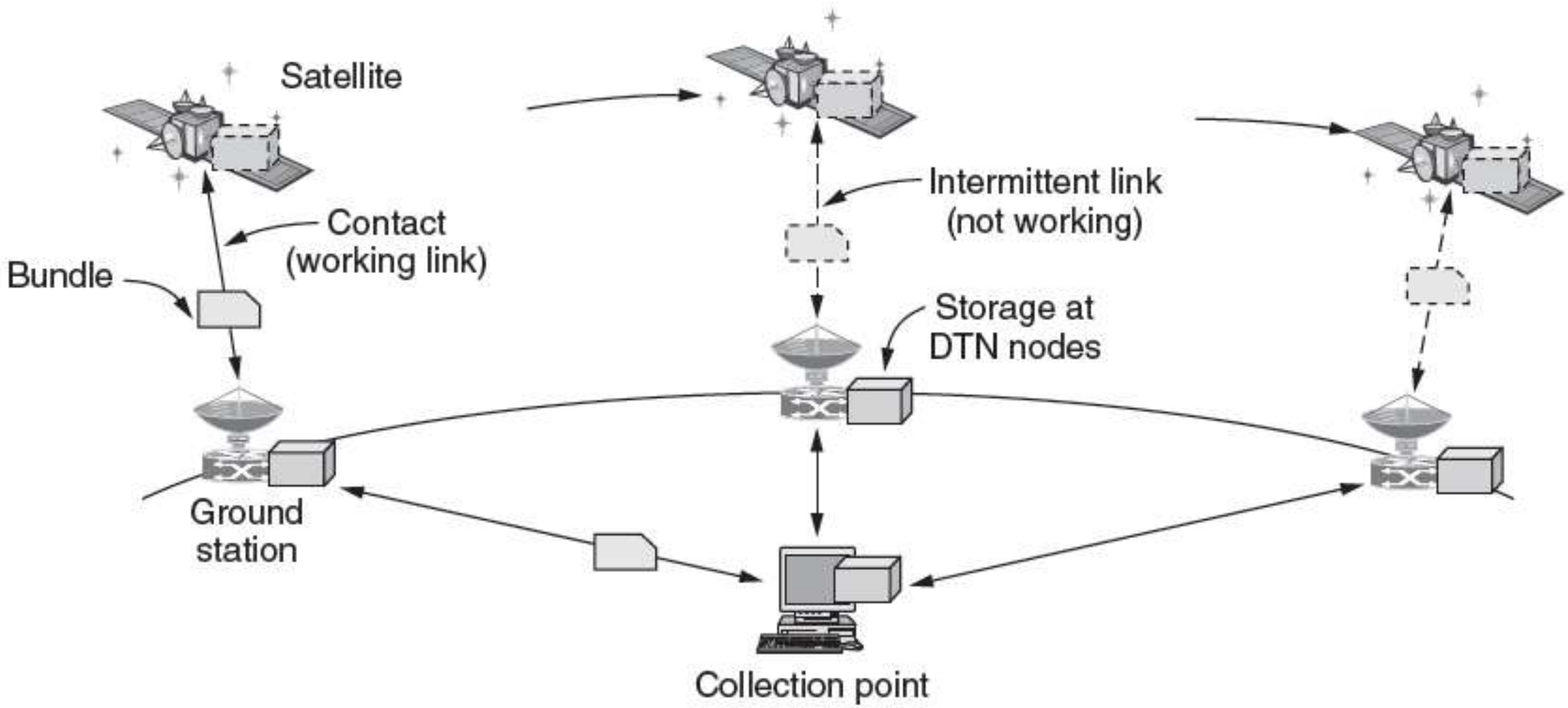
- DTN Architecture
- The Bundle Protocol

DTN Architecture (1)



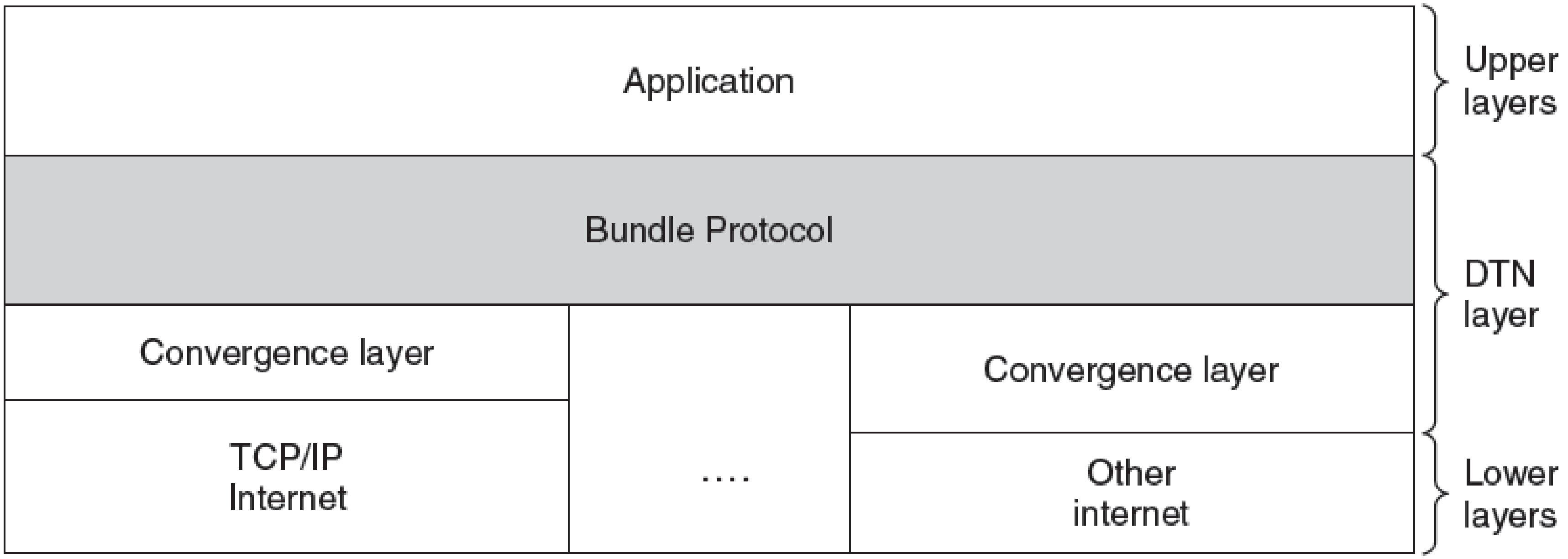
Delay-tolerant networking architecture

DTN Architecture (2)



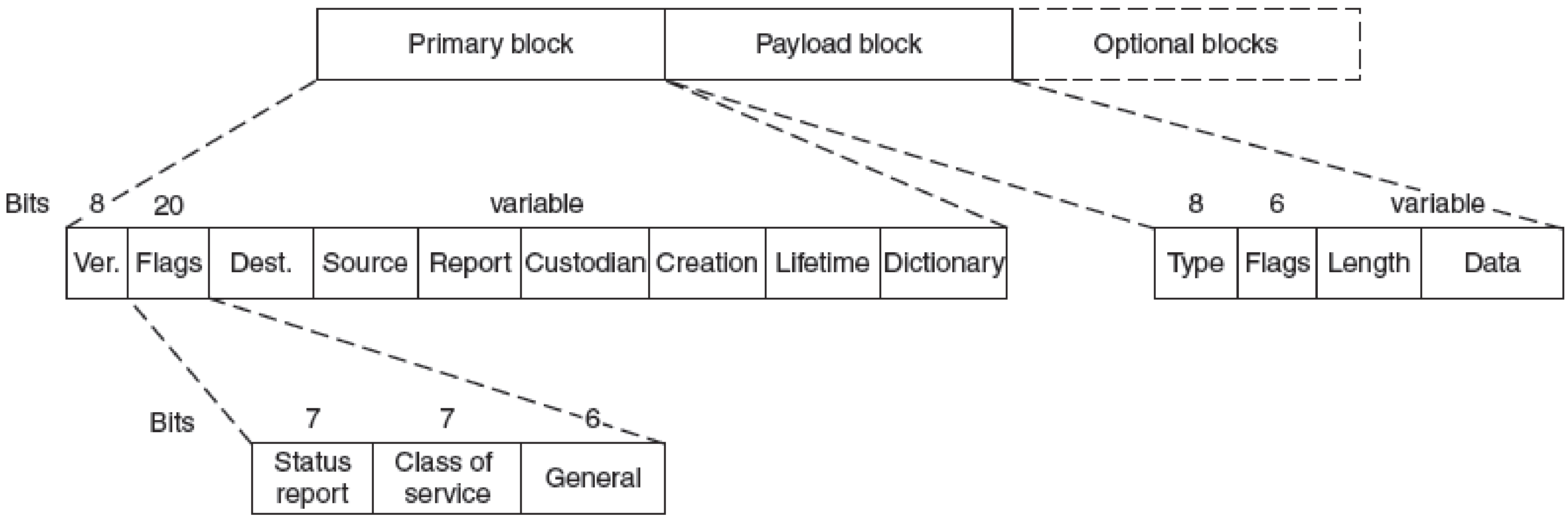
Use of a DTN in space.

The Bundle Protocol (1)



Delay-tolerant networking protocol stack.

The Bundle Protocol (2)



Bundle protocol message format.

Recommended Exercises

In 4th Edition:

- 16,19, 20, 23, 25, 28-31, 33,36,40

In 5th Edition:

- 13,18-19, 21, 25-28,30, 33,35