Transport Layer

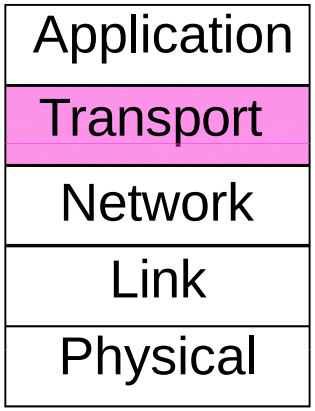
Chapter 6

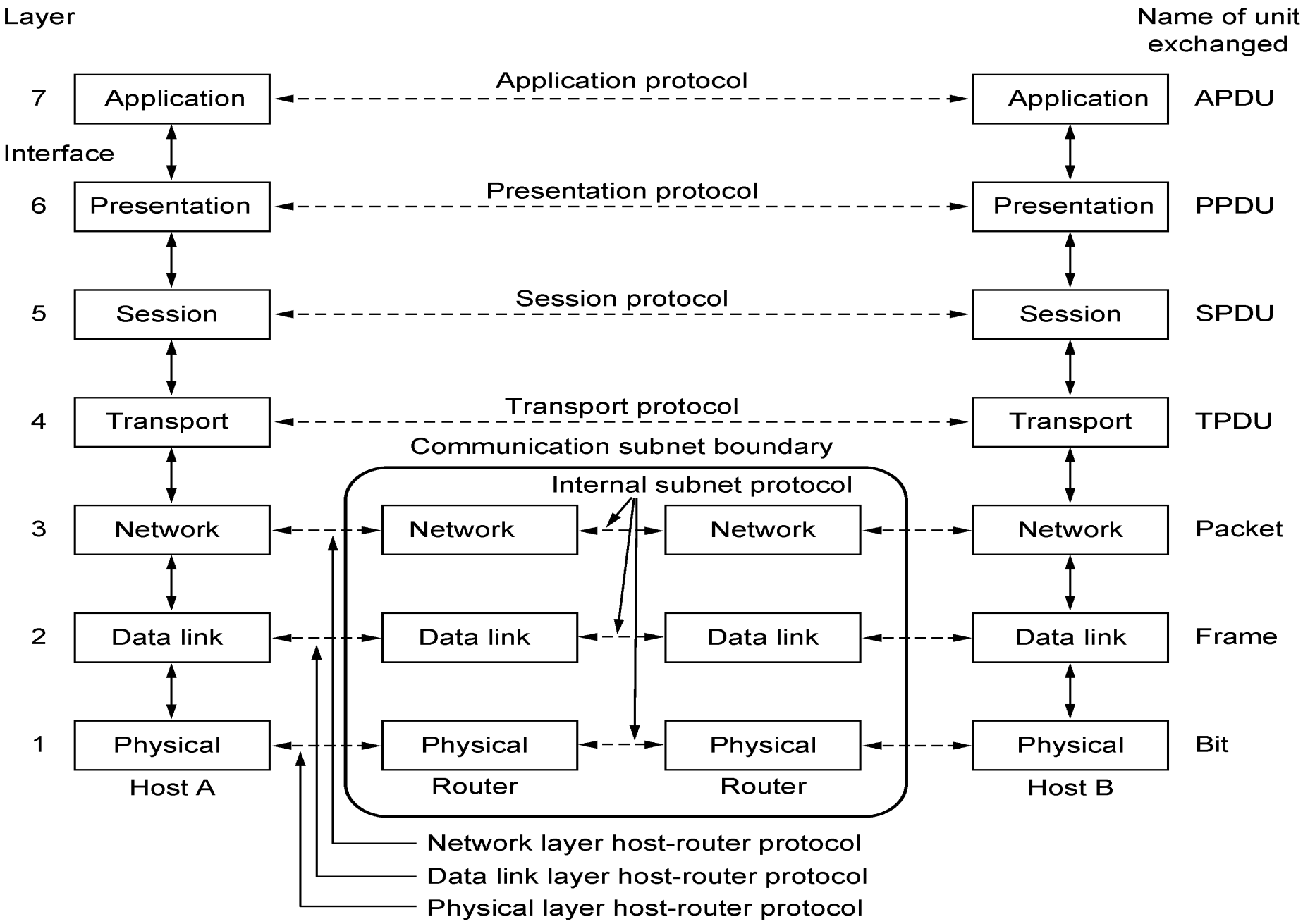
- Transport Service
- Elements of Transport Protocols
- Congestion Control
- Internet Protocols – UDP
- Internet Protocols – TCP
- Delay-Tolerant Networking

The Transport Layer

- Responsible for delivering data across networks with the desired reliability or quality

- Provide data support from a process on a source machine to a process on destination machine





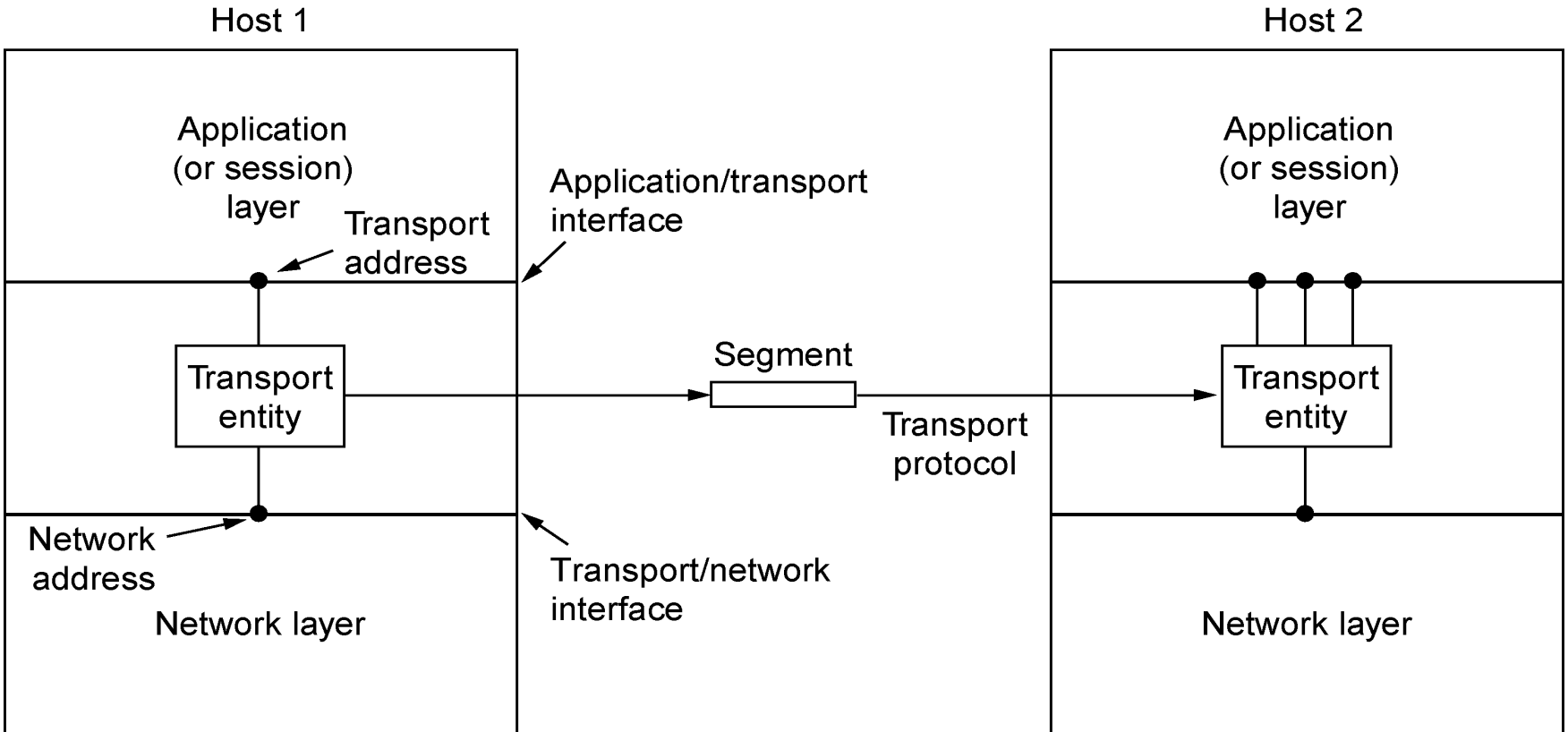
The OSI reference model.

Transport Service

- Services Provided to the Upper Layer »
- Transport Service Primitives »
- Berkeley Sockets »
- Socket Example: Internet File Server »

Services Provided to the Upper Layers (1)

- Goal
  - To provide efficient, reliable and cost-effective data transmission service to its users, i.e., to a process in the application layer
- Transport layer adds reliability to the network layer and offers
  - connectionless (e.g., UDP) and
  - connection- oriented (e.g, TCP) service to applications

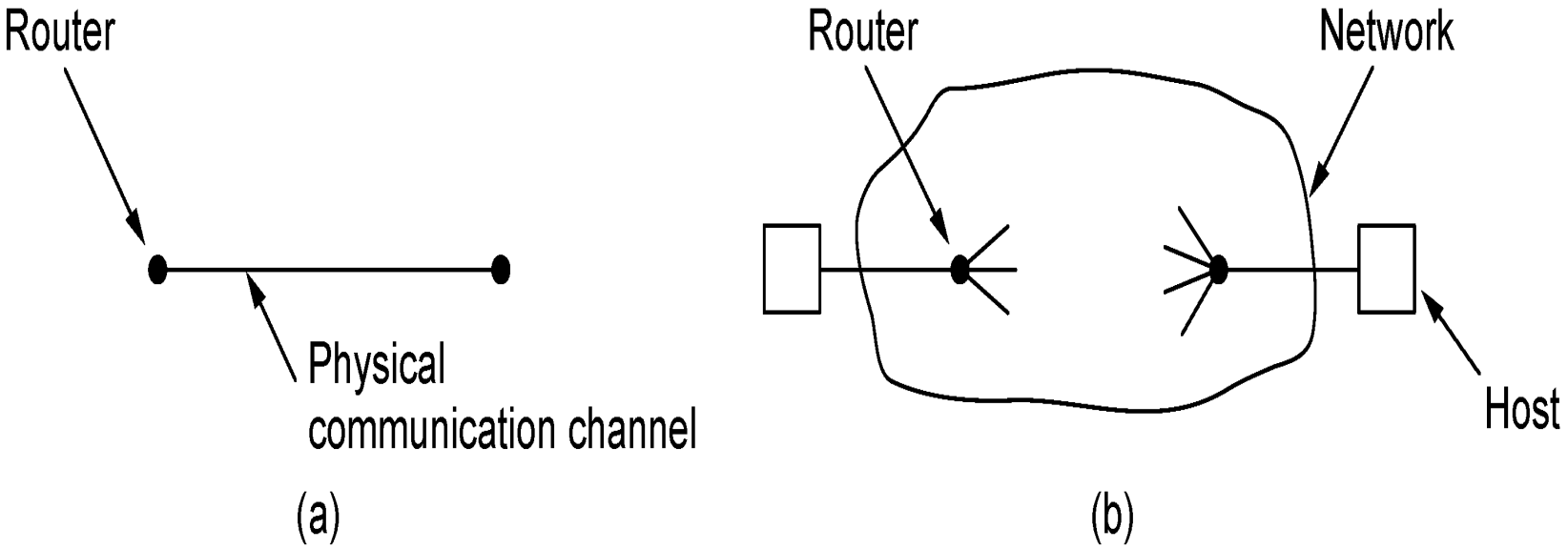


Why different layer?

- Transport service is similar to network layer. Then, why two distinct layers?
  - Transport layer runs entirely on users machine
  - Network layer runs on routers, which are operated by the carrier.
  - What happens if the network layer offers inadequate service?
  - What if it frequently loses packets?
  - What happens if the routers crash time to time?

About Transport Layer

- Essence of the transport service
  - More reliability to applications
- Users have no control over network layer.
- Transport layer can use retransmissions.
- Transport primitives are independent of network primitives.
  - Network service calls may be different for each network



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

Transport Service Primitives (1)

- Example:
- Two processes in a single machine connected through a pipe
  - Process A puts the data and Process B consumes

- Connection between them is 100 % perfect
  - **No acks, no lost packets, congestion or any thing.**

- Transport layer provides similar service in the network for connection-oriented service **with acks, lost packets, congestion and other mechanisms**

- Transport layer also provides unreliable datagram service

- Another difference
  - Network service is used by transport entities
  - Few users write their own transport entities
    - Few users writes programs using network services
  - Many programs use transport primitives.

Transport Service Primitives (1)

- Primitives that applications might call to transport data for a simple connection-oriented service:
  - Client calls CONNECT, SEND, RECEIVE, DISCONNECT
  - Server calls LISTEN, RECEIVE, SEND, DISCONNECT

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.	Request a release of the connection

The primitives for a simple transport service.

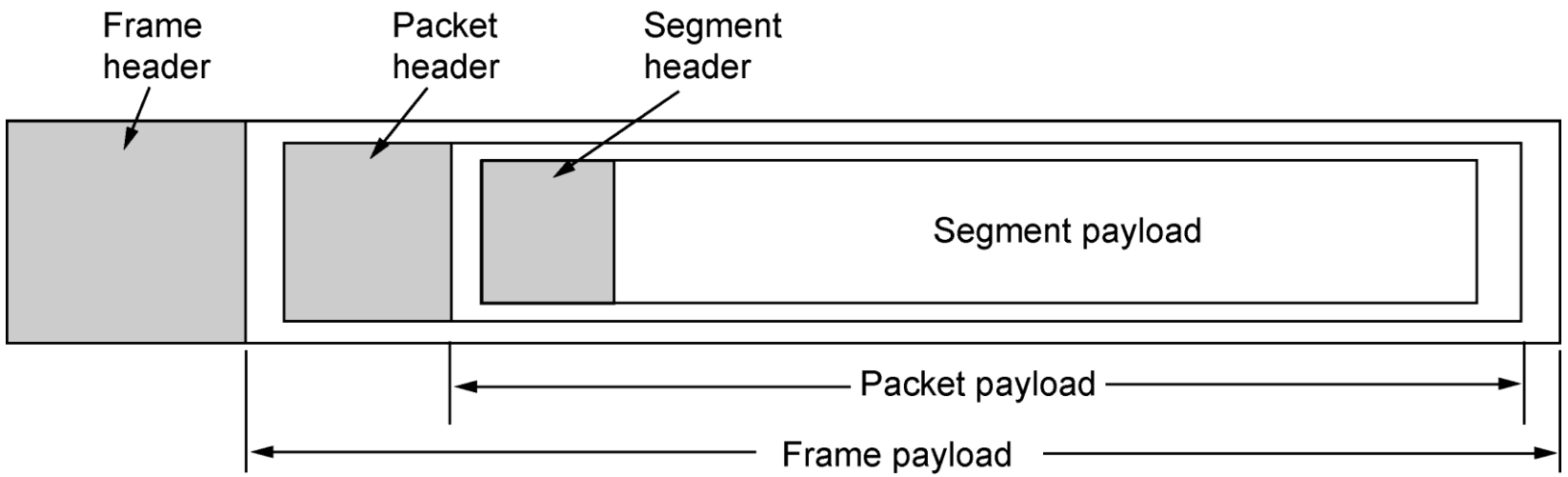
Transport Service Primitives (1)

- LISTEN: calling a library procedure that makes a system call that blocks the server until a client tuns up.
- CONNNECT: When client wants to talk to the server, it executes a CONNECT primitive. It results into CONNNECT REQ. segment sent to the server.
- The server unblocks the server and sends CONNECTION ACCCEPTED segment back to the client.
- Data can be exchanged using the SEND and RECEIVE primitives.
- Every data packet is acknowledged.

- When a connection is no longer needed, transport user issues a DISCONNECT primitive. Upon the arrival, the connection is released.
- Each direction is closed independently.

Services Provided to the Upper Layers (2)

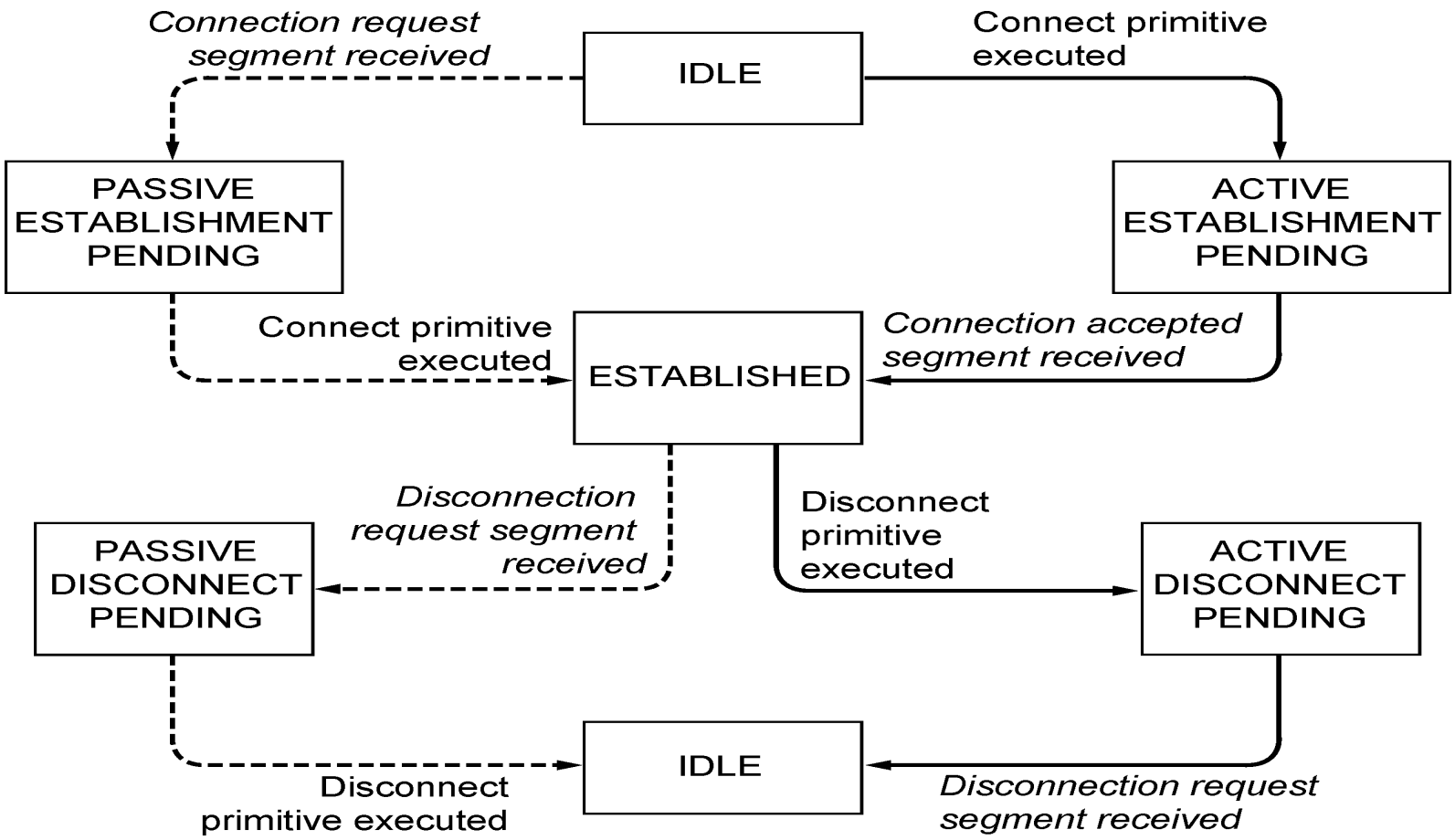
- Segments are contained in the packet sent by network layer, in turn packets are contained in frames exchanged by data link layer.



Nesting of segments, packets, and frames.

Transport Service Primitives (2)

State diagram for a simple connection-oriented service



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

Very widely used primitives started with TCP on UNIX

- Notion of “sockets” as transport endpoints
- Like simple set plus SOCKET, BIND, and ACCEPT

Primitive	Meaning
SOCKET	Create a new communication endpoint
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.

Socket Example – Internet File Server (1)

Client code

```
. . .

if (argc != 3) fatal("Usage: client server-name file-name");
h = gethostbyname(argv[1]);
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");

. . .
```

Get server's IP address

Make a socket

Try to connect

Socket Example – Internet File Server (2)

Client code (cont.)

. . .

```
    write(s, argv[2], strlen(argv[2])+1);

    while (1) {
        bytes = read(s, buf, BUF_SIZE);
        if (bytes <= 0) exit(0);
        write(1, buf, bytes);
    }
}
```

Write data (equivalent to send)

Loop reading (equivalent to receive) until no more data; exit implicitly calls close

Socket Example – Internet File Server (3)

Server code

```
. . .

memset(&channel, 0, sizeof(channel));
channel.sin_family = AF_INET;
channel.sin_addr.s_addr = htonl(INADDR_ANY);
channel.sin_port = htons(SERVER_PORT);

s = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
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);
if (l < 0) fatal("listen failed");

. . .
```

Make a socket

Assign address

Prepare for incoming connections

Socket Example – Internet File Server (4)

Server code

```
. . .
   while (1) {
        sa = accept(s, 0, 0);
        if (sa < 0) fatal("accept failed");

        read(sa, buf, BUF_SIZE);

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

        while (1) {
              bytes = read(fd, buf, BUF_SIZE);
              if (bytes <= 0) break;
              write(sa, buf, bytes);
        }
        close(fd);
        close(sa);
   }
}
```

Block waiting for the next connection

Read (receive) request and treat as file name

Write (send) all file data

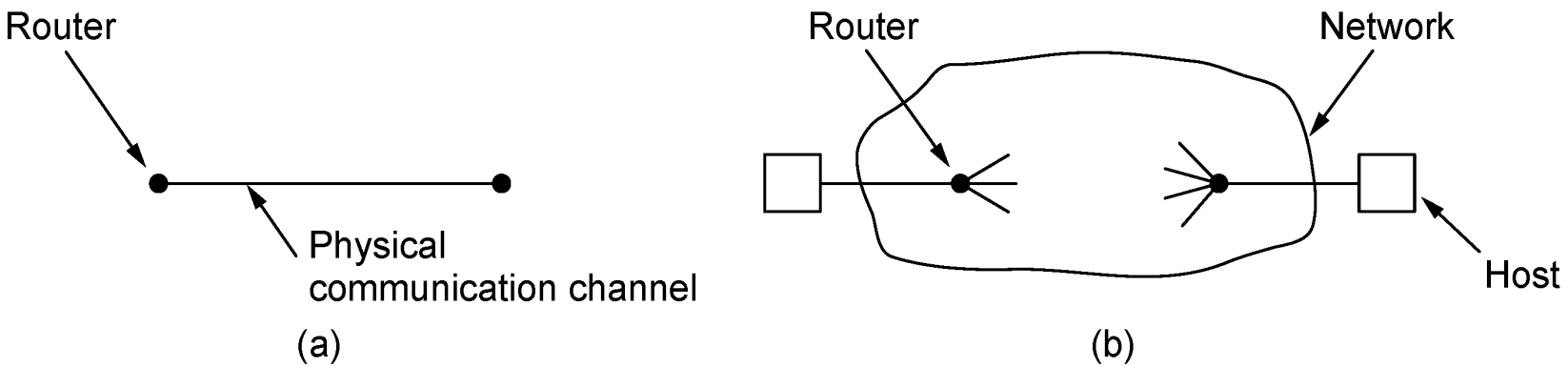
Done, so close this connection

Elements of Transport Protocols

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

About Transport Protocols

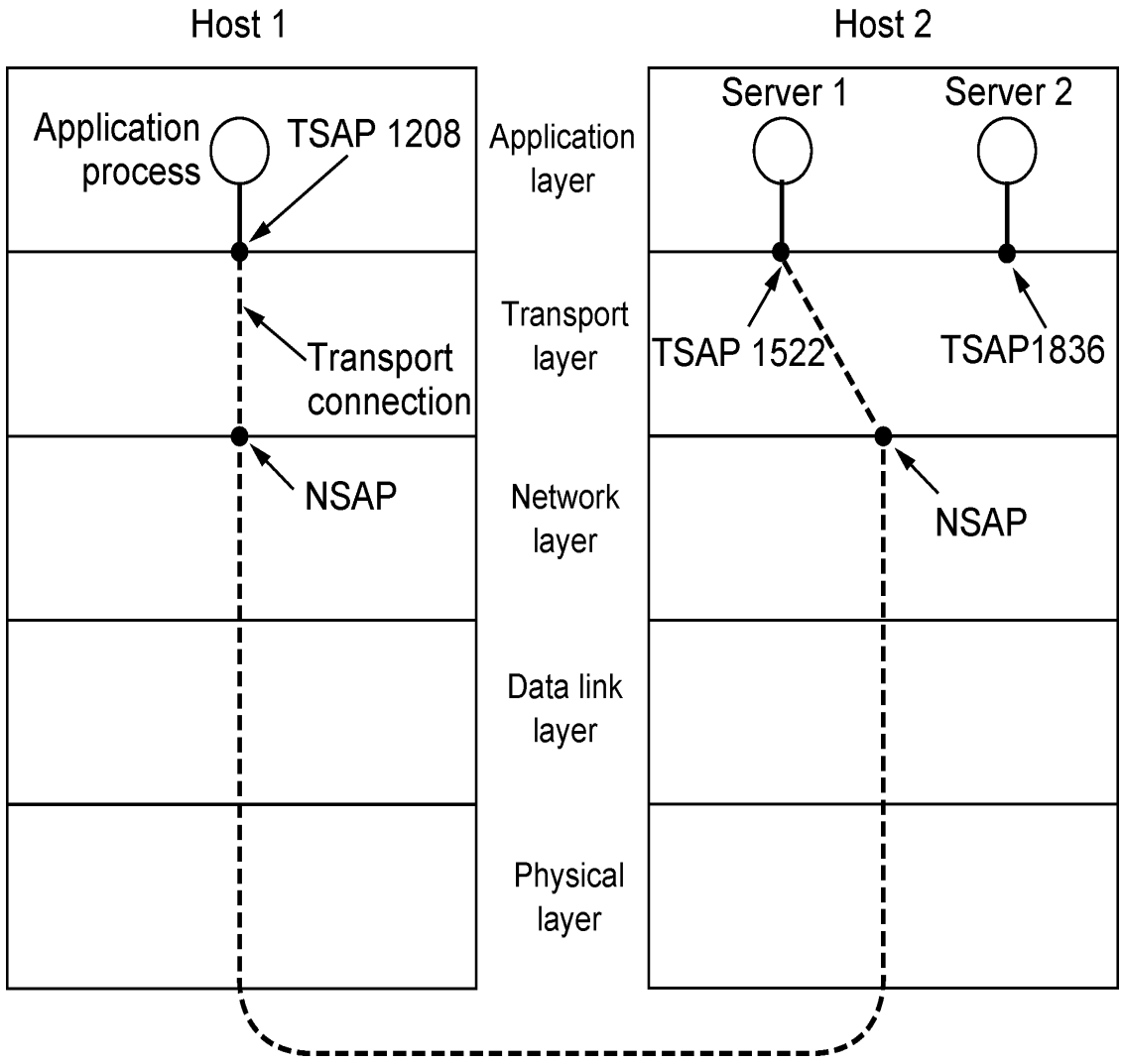
- Transport protocols resemble data link layer protocols
  - Both have to deal with error control, sequencing, and flow control, among other issues
- Significant differences exist due to environmental differences
- Differences
  - One: Data link layer is on direct connection whereas transport layer is on entire network.
  - Two: Delay is more in transport layer, more failures
  - Three: Number of connections are more in transport layer



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

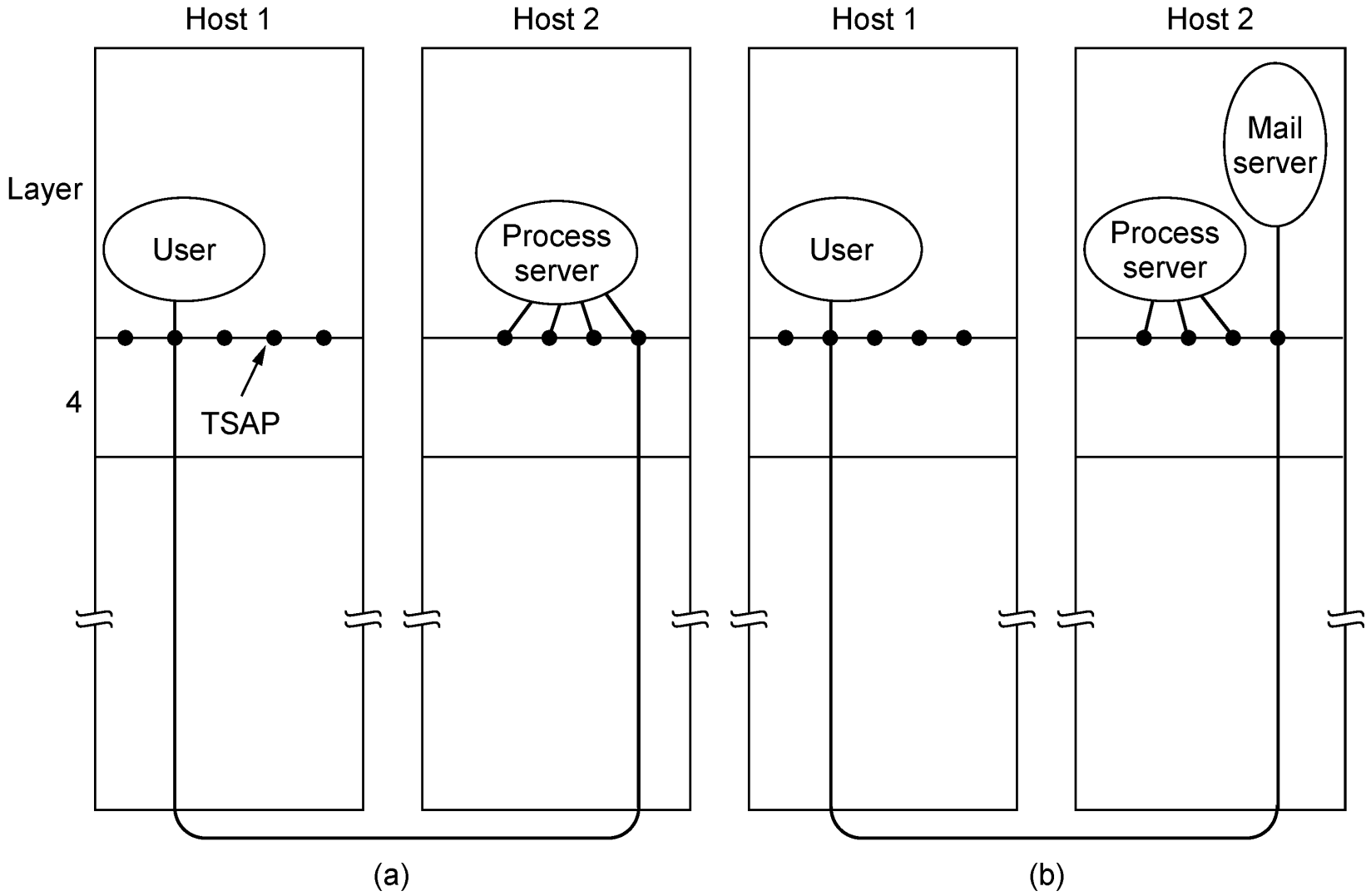
Addressing

- Transport layer adds TSAPs (Transport Service Access Points)
- Multiple clients and servers can run on a host with a single network (IP) address
- TSAPs are ports for TCP/UDP



TSAPs, NSAPs, and transport connections.

- Process server: inetd in UNIX
- Each machine runs a special process server.
- The process server spawns a thread and connects to the related server.



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

Connection Establishment (1)

- Key problem is to ensure reliability even though packets may be lost, corrupted, delayed, and duplicated
  - Don't treat an old or duplicate packet as new
  - (Use ARQ (Automatic Repeat Request) and checksums for loss/corruption)
- Key issue
  - Delayed duplicates
    - User sends packets, they may follow longest route.
    - User time out expires. Send the packets again.
    - Receiver receives both old and new packets
- Approaches
  - Restricted network login
  - Putting a hop counter in each packet
  - Timestamping each packet
- We should also ensure that all ack. are dead.

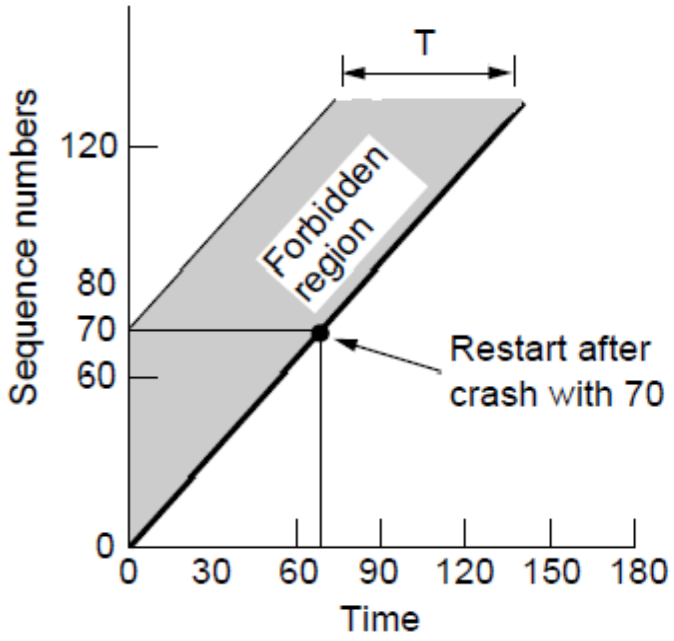
Connection Establishment (1)

Approach:

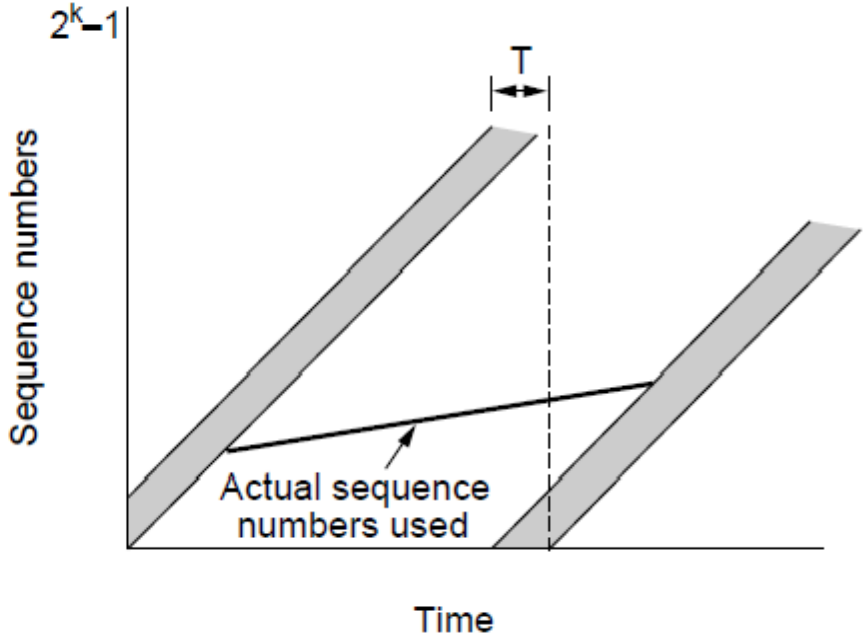
- Don’t reuse sequence numbers within twice the MSL (Maximum Segment Lifetime) of 2T=240 secs
- We can use the clock
- Once the transport entities agree to initial sequence number, any sliding window protocol can be used.
  - It discards duplicate packets
- Issues
  - How to keep packet sequence numbers out of forbidden region?
  - If the host sends too much data too fast, it results sequence numbers to enter into forbidden curve.
  - If the data rate is less than the clock rate, we also get into the trouble.
  - We have to remember the sequence numbers.
- Three-way handshake is proposed to solve the problem

Connection Establishment (2)

- Use a sequence number space large enough that it will not wrap, even when sending at full rate
  - Clock (high bits) advances & keeps state over crash



Need seq. number not to wrap within T seconds



Need seq. number not to climb too slowly for too long

Connection Establishment

Three-way handshake used for initial packet

- Since no state from previous connection
- Both hosts contribute fresh seq. numbers
- CR = Connect Request
- Protocol
  - Host 1 choses a sequence number and sends CR
  - Host 2 send CR with ack. and sending its own sequence number.
  - Host 1 ack. the choice of sequence numbers by host 2

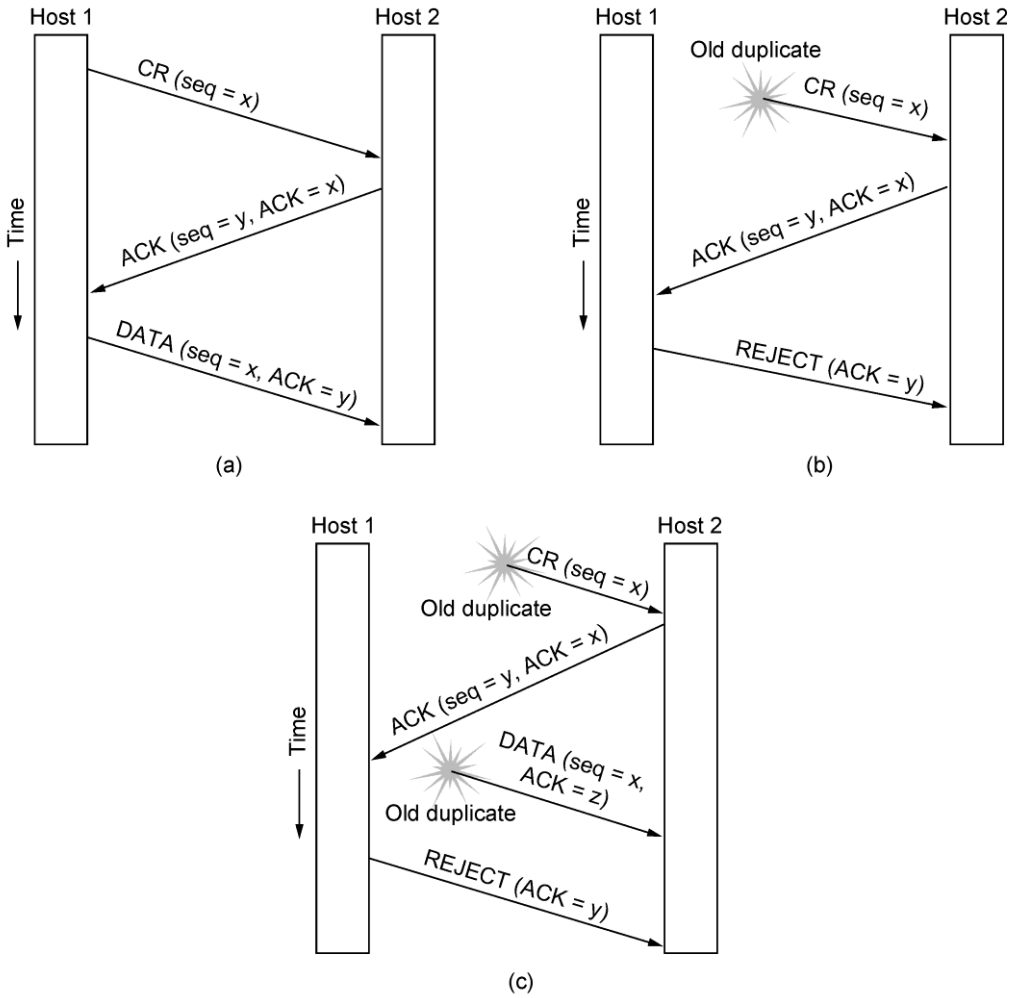
Connection Establishment (4)

Three-way handshake protects against odd cases:

a) Duplicate CR, Spurious ACK does not connect

b) Duplicate CR and DATA. Same plus DATA will be rejected (wrong ACK).

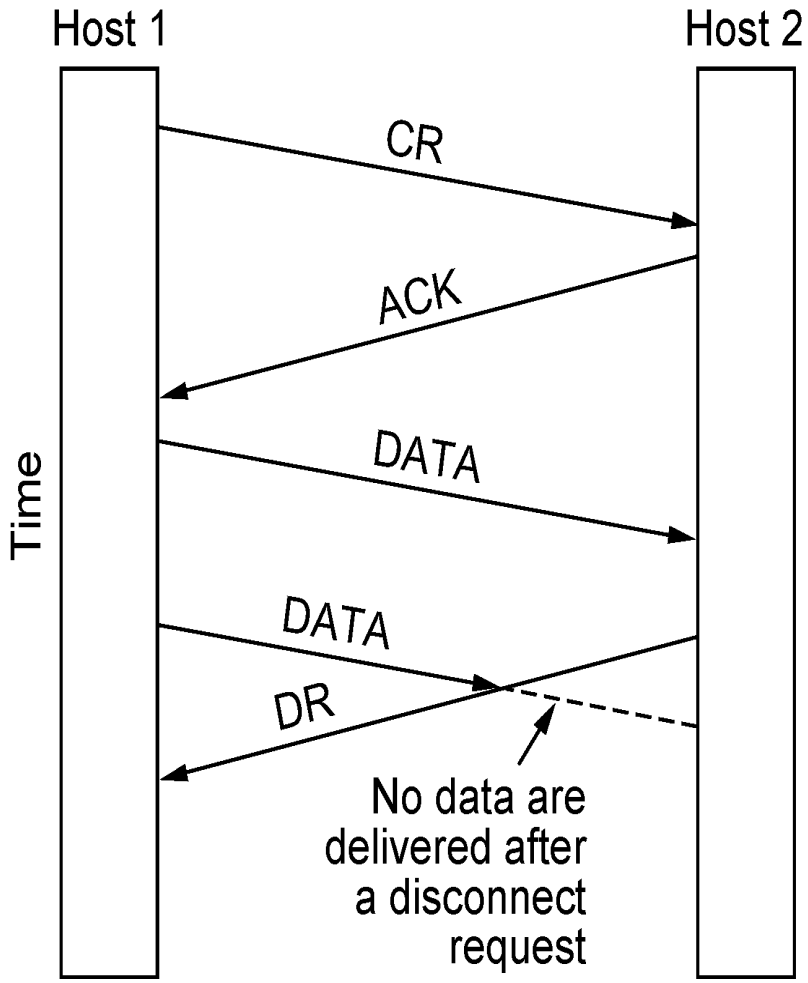
- TCP uses three-way handshake to establish connections



Three protocol scenarios for establishing a connection using a three-way handshake. CR denotes CONNECTION REQUEST. (a)Normal operation. (b)Old duplicate CONNECTION REQUEST appearing out of nowhere. (c)Duplicate CONNECTION REQUEST and duplicate ACK

Connection Release (1)

- Key problem is to ensure reliability while releasing
- Two kinds of releases
  - Asymmetric release
    - Like a telephone system. When one party hand up, connection is released
    - Asymmetric release (when one side breaks connection) is abrupt and may lose data
  - Symmetric release
    - Two separate releases
- Issue
  - How to avoid data loss?
    - Use symmetric release

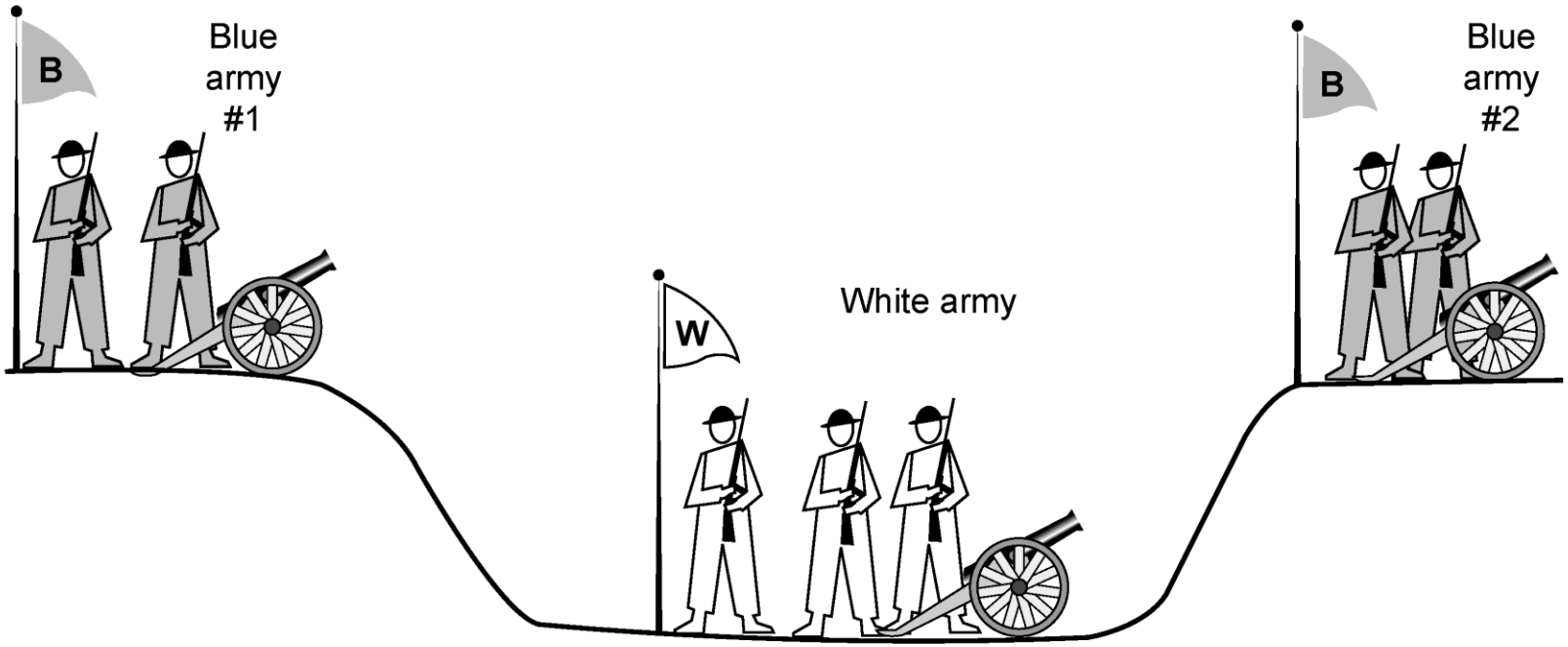


Abrupt disconnection with loss of data.

Connection Release (2)

Symmetric release (both sides agree to release) can't be handled solely by the transport layer

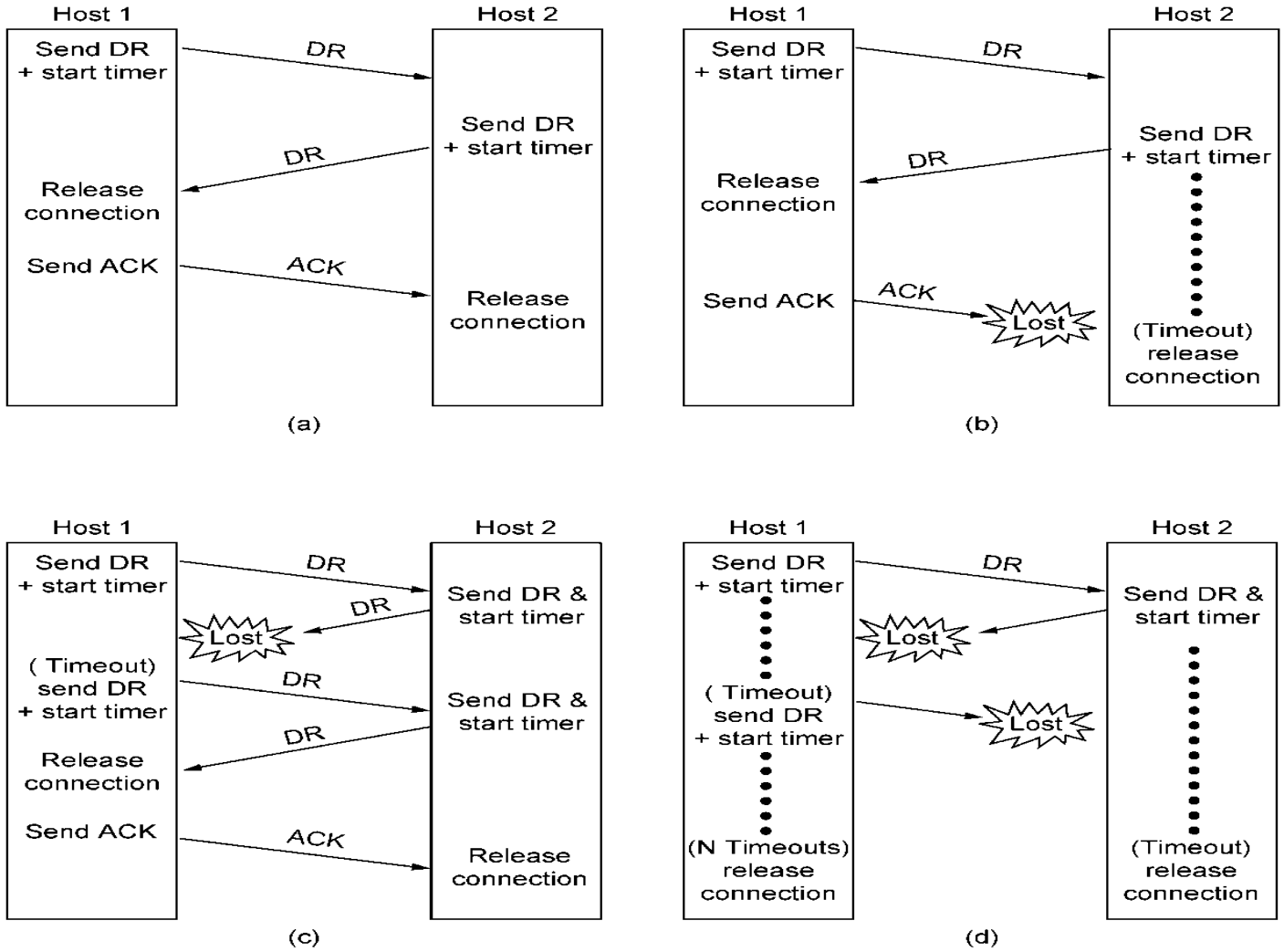
- Two-army problem shows pitfall of agreement
- Blue army units wants to synchronize the attack.
- No one is sure that his reply is got through.
- Three-way/four-way/… will not help.



The two-army problem.

Connection Release (4)

Error cases are handled with timer and retransmission



Four protocol scenarios for releasing a connection. (a) Normal case of three-way handshake. (b) Final lost. (c) Response lost. (d) Response lost and subsequent lost.

Error Control and Flow Control (1)

- Foundation for error control is a sliding window (from Link layer) with checksums and retransmissions
- Flow control manages buffering at sender/receiver
  - Issue is that data goes to/from the network and applications at different times
  - Window tells sender available buffering at receiver
  - Makes a variable-size sliding window

Error Control and Flow Control (1)

- Recap about data link layer mechanisms
  - A frame carries an error-correcting code (CRC or checksum)
  - A frame carries sequence number to identify itself and retransmitted by sender until receives an ack of successful receipt. This is called ARQ (Automatic Repeat Request)
  - There is a maximum number of frames the sender will allow. If the maximum is one, it is called stop-wait.
  - Sliding window protocol enable bidirectional data transfer.

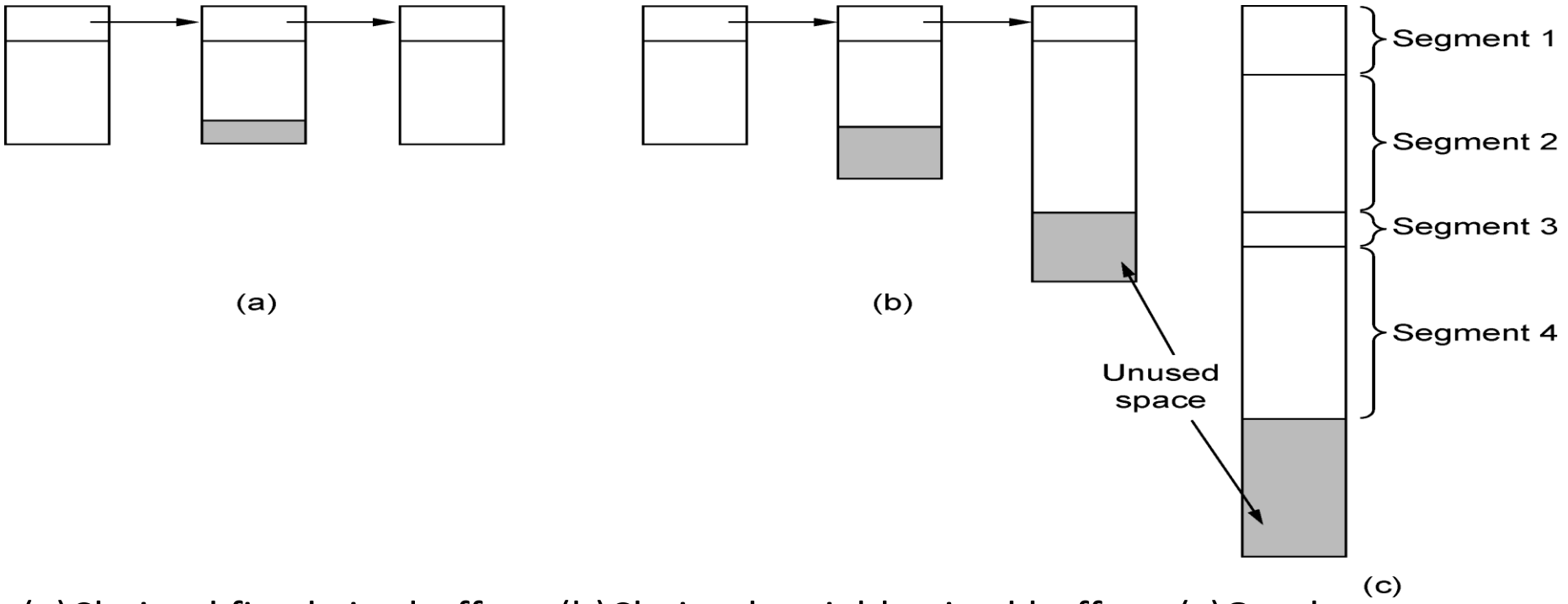
Error Control and Flow Control (1)

- Difference
  - Link layer checksum protects frames
  - Transport layer checksum protects segments

- A larger sliding window has to be used for Transport layer
  - Host may need a substantial amount of buffer space
  - To hold the data of unacknowledged segments
  - Receiver may not buffer it
- How to organize the buffer pool is the question.
  - Fixed
  - Variable-size
  - Chained large buffer
- Dynamic buffer allocation scheme is used.

Error Control and Flow Control (2)

Different buffer strategies trade efficiency / complexity

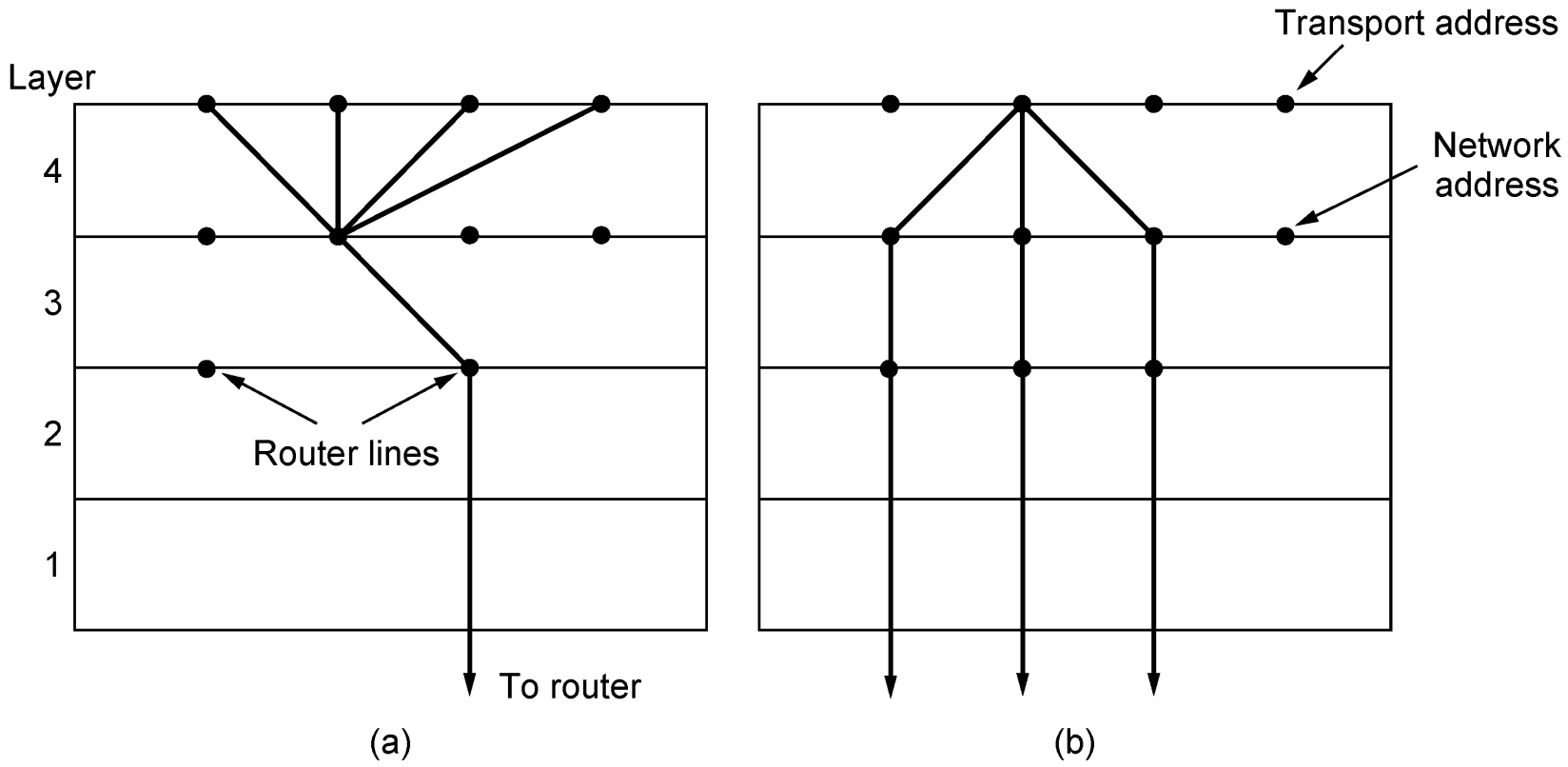


(a)Chained fixed-size buffers. (b)Chained variable-sized buffers. (c)One large circular buffer per connection.

Multiplexing

Kinds of transport / network sharing that can occur:

- Multiplexing: connections share a network address
- Inverse multiplexing: addresses share a connection



(a)Multiplexing. (b)Inverse multiplexing.

Crash Recovery

Application needs to help recovering from a crash

- Transport can fail since A(ck) / W(rite) not atomic
- Difficult problem

Congestion Control

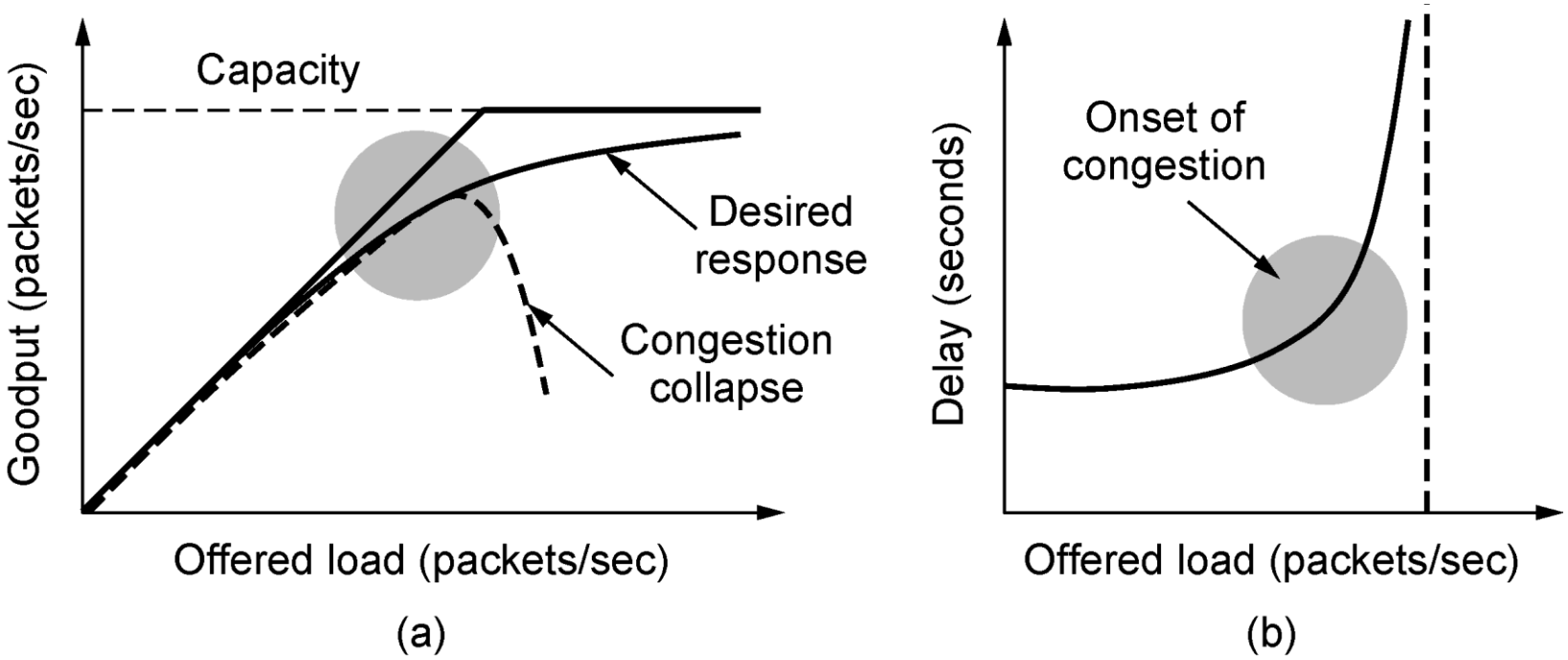
Two layers are responsible for congestion control:

- Transport layer, controls the offered load [here]
- Network layer, experiences congestion [previous]

- Desirable bandwidth allocation »
- Regulating the sending rate »
- Wireless issues »

Desirable Bandwidth Allocation (1)

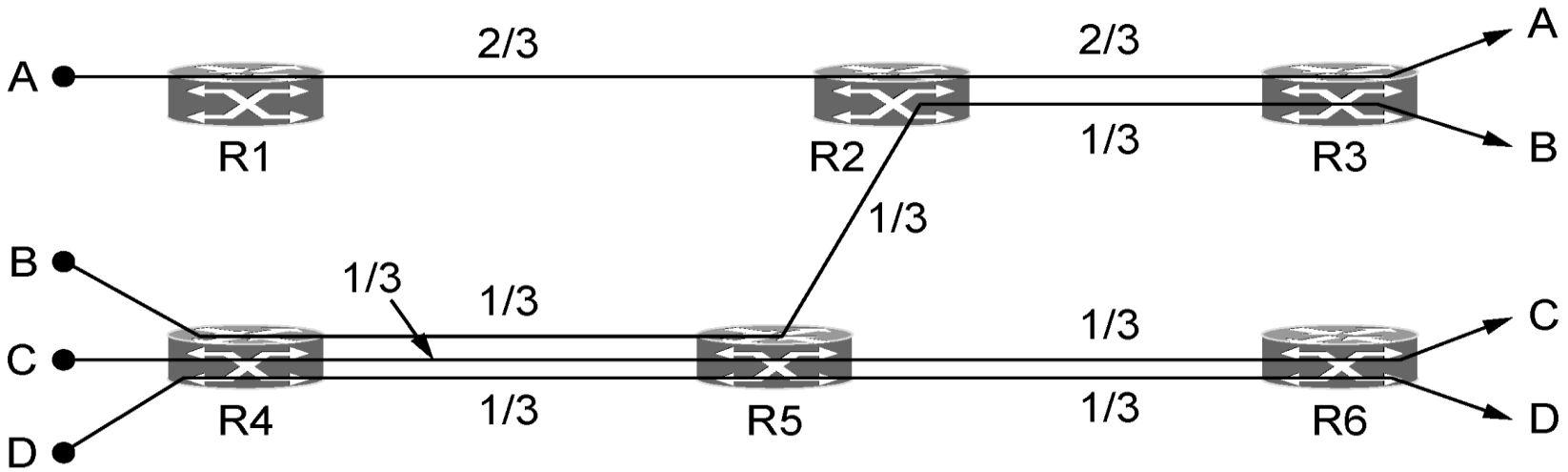
Efficient use of bandwidth gives high goodput, low delay



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

Desirable Bandwidth Allocation (2)

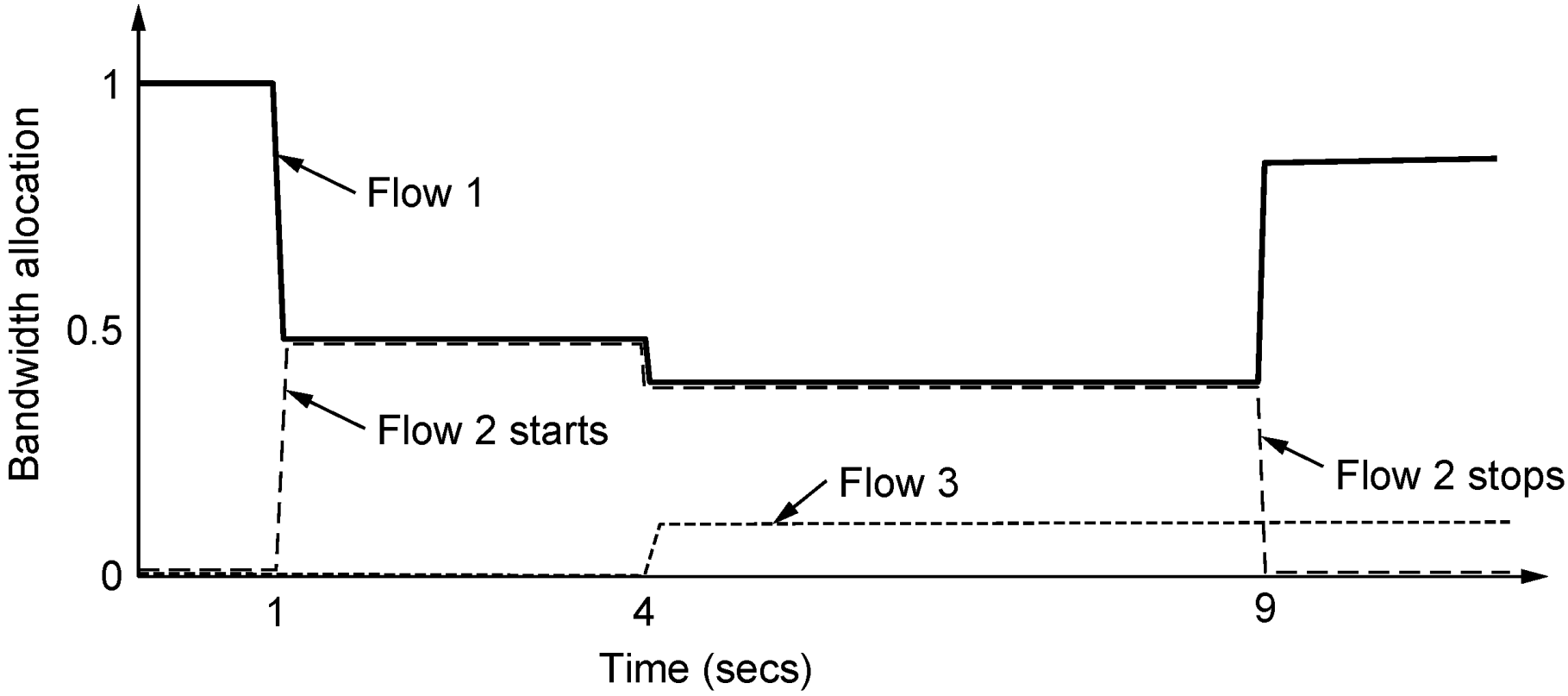
- Fair use gives bandwidth to all flows (no starvation)
  - Max-min fairness gives equal shares of bottleneck
  - An allocation is min-max fair, if the bandwidth given to one flow can not be increased without decreasing the bandwidth given to another flow with the allocation that is no longer.
  - Increase the bandwidth of the flow makes situation worse.



Max-min bandwidth allocation for four flows.

Desirable Bandwidth Allocation (3)

We want bandwidth levels to converge quickly when traffic patterns change

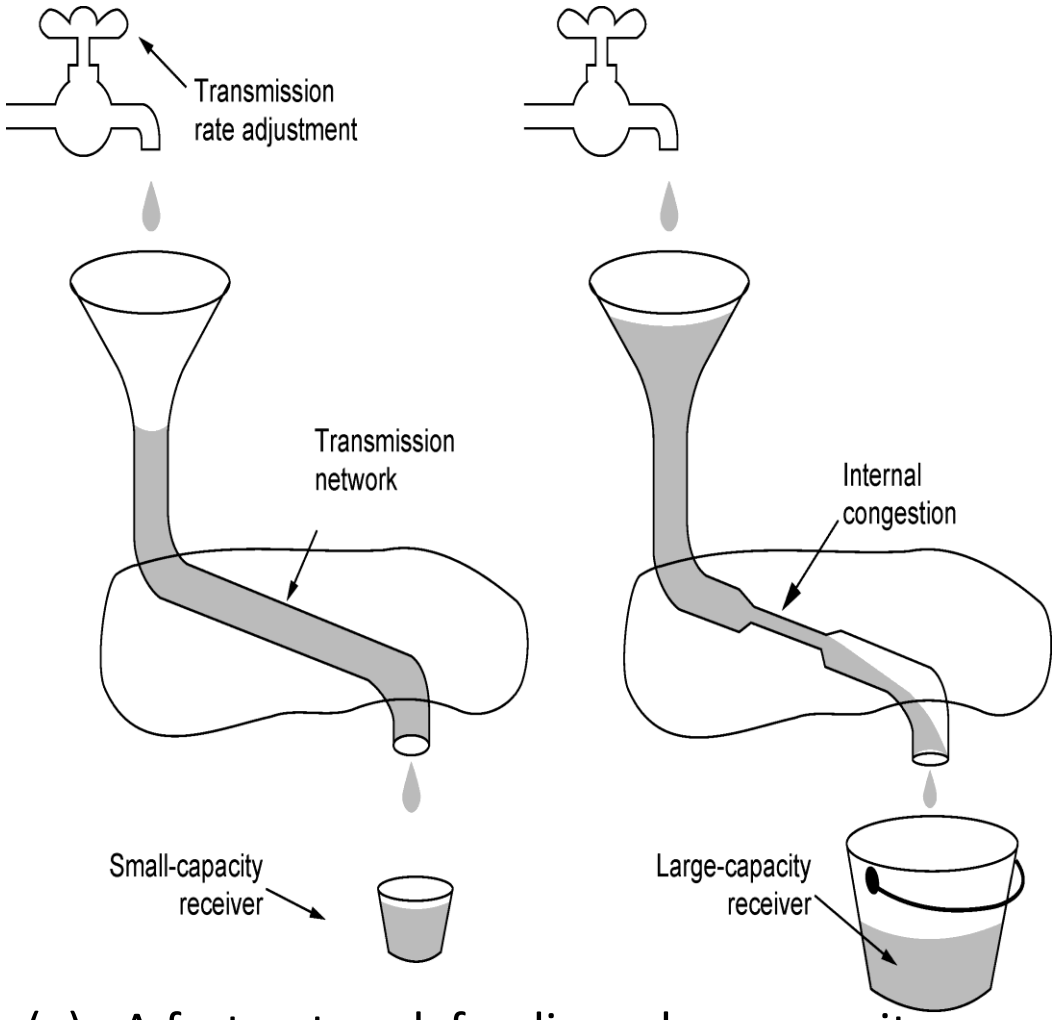


Changing bandwidth allocation over time.

Regulating the Sending Rate (1)

Sender may need to slow down for different reasons:

- Flow control, when the receiver is not fast enough [right]
- Congestion, when the network is not fast enough [over]



(a) A fast network feeding a low-capacity receiver.
(b) A slow network feeding a high-capacity receiver.

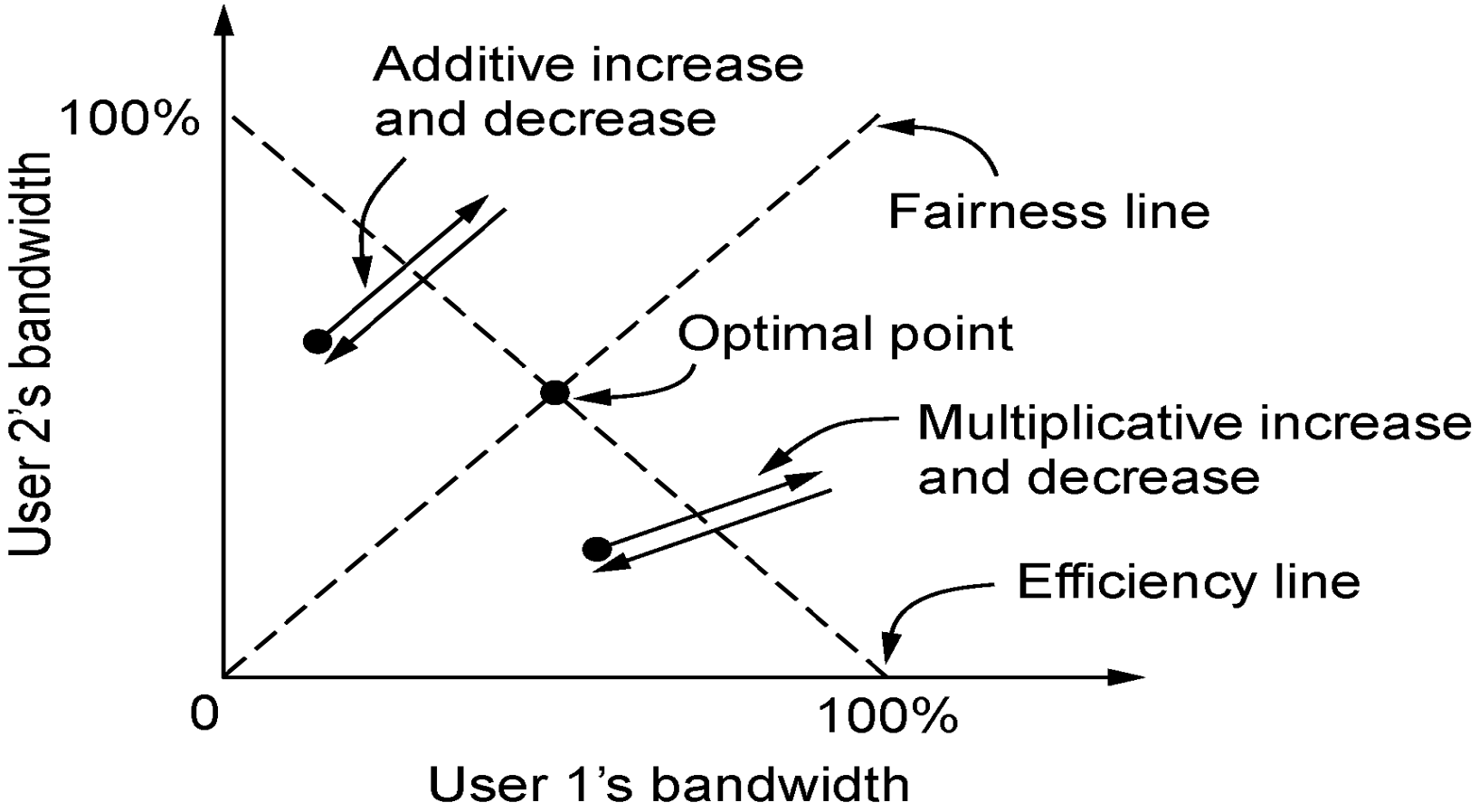
Regulating the Sending Rate (3)

Different congestion signals the network may use to tell the transport endpoint to slow down (or speed up)

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)

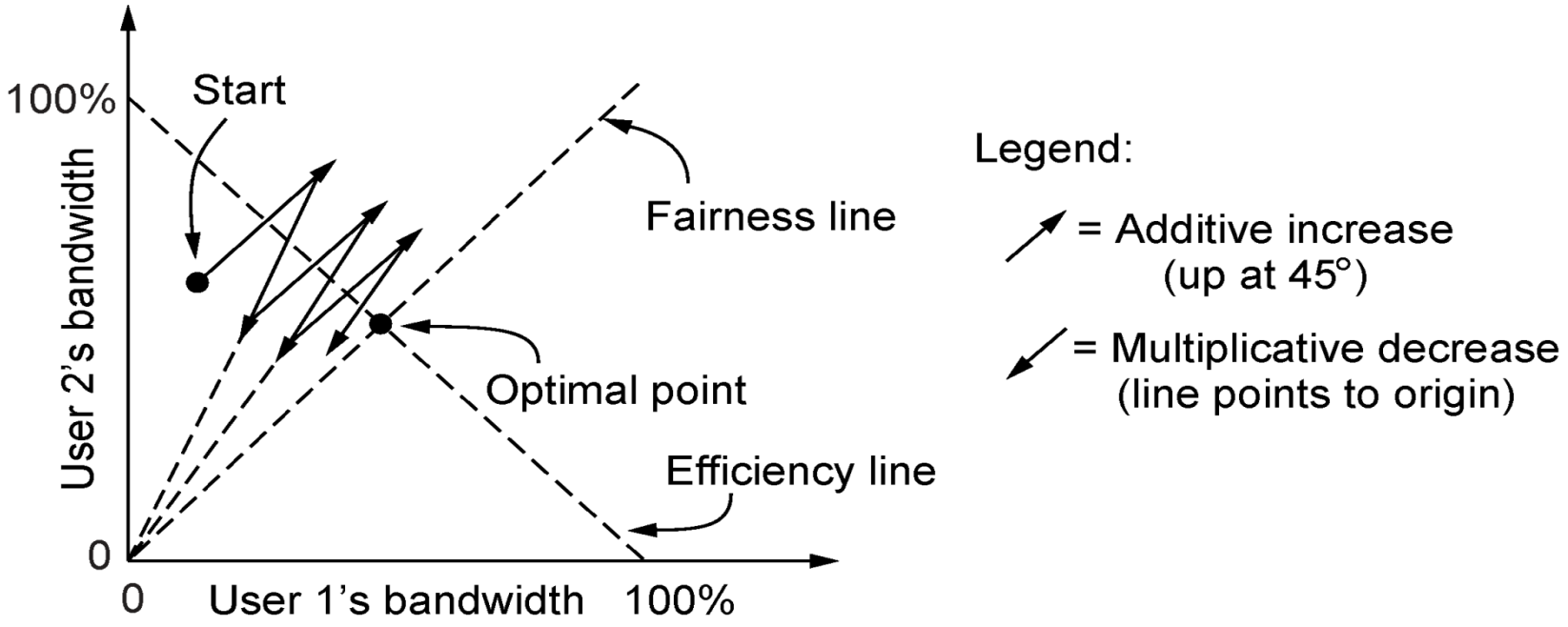
If two flows increase/decrease their bandwidth in the same way when the network signals free/busy they will not converge to a fair allocation.



Additive and multiplicative bandwidth adjustments.

Regulating the Sending Rate (4)

- The AIMD (Additive Increase Multiplicative Decrease) control law does converge to a fair and efficient point!
  - In the absence of congestion sender should increase the rate
  - In the presence of congestion signal senders to decrease the rate
- TCP uses AIMD for this reason



Additive Increase Multiplicative Decrease (AIMD) control law.

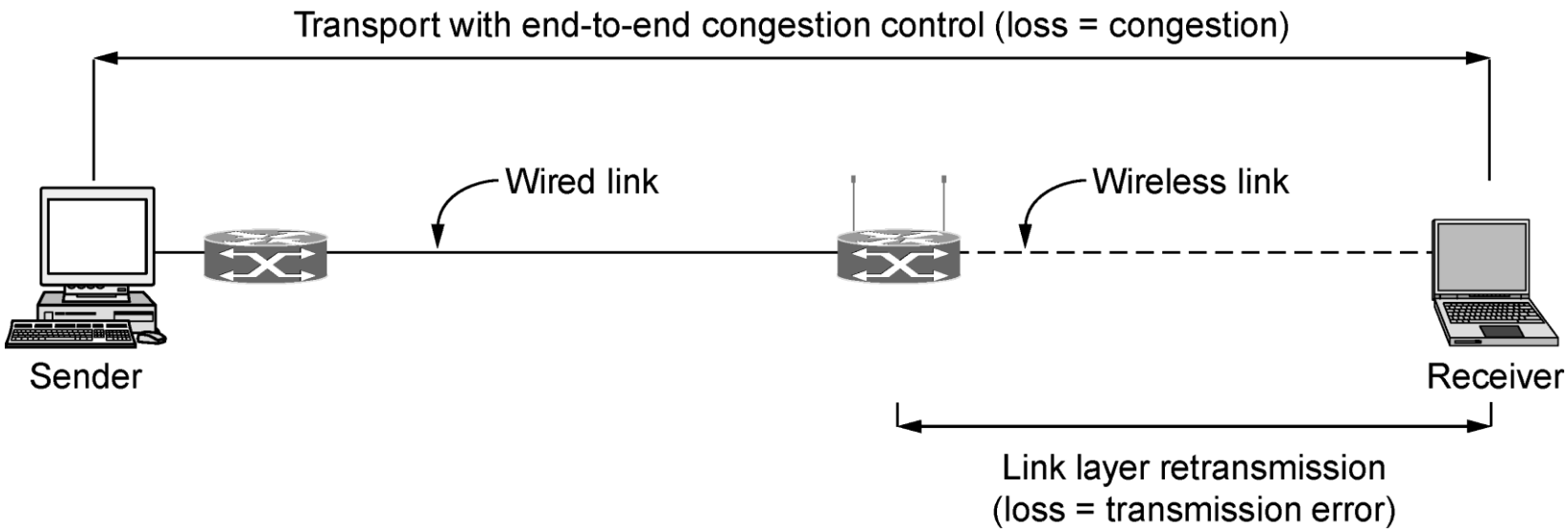
Wireless Issues

Wireless links lose packets due to transmission errors

- Do not want to confuse this loss with congestion
- Or connection will run slowly over wireless links!

Strategy:

- Wireless links use ARQ, which masks errors



Congestion control over a path with a wireless link.

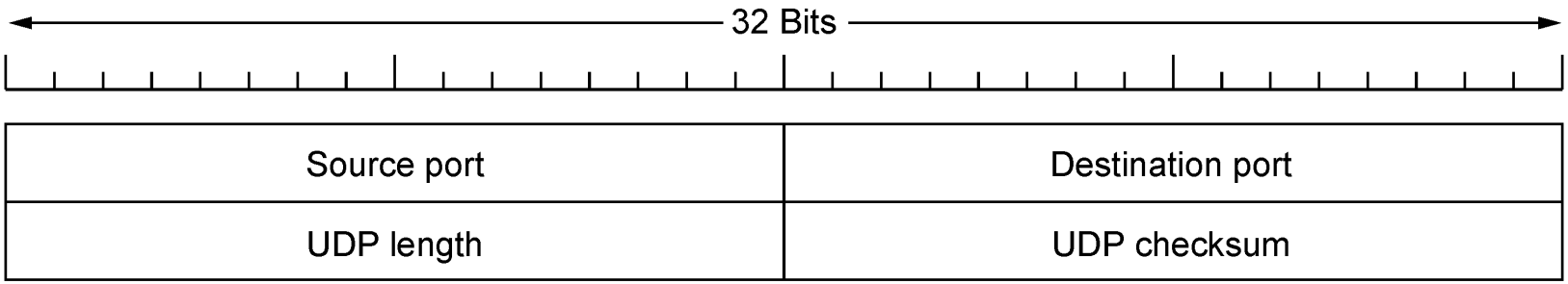
Internet Protocols – UDP

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

Introduction to UDP (1)

UDP (User Datagram Protocol) is a shim over IP

- Header has ports (TSAPs), length and checksum.

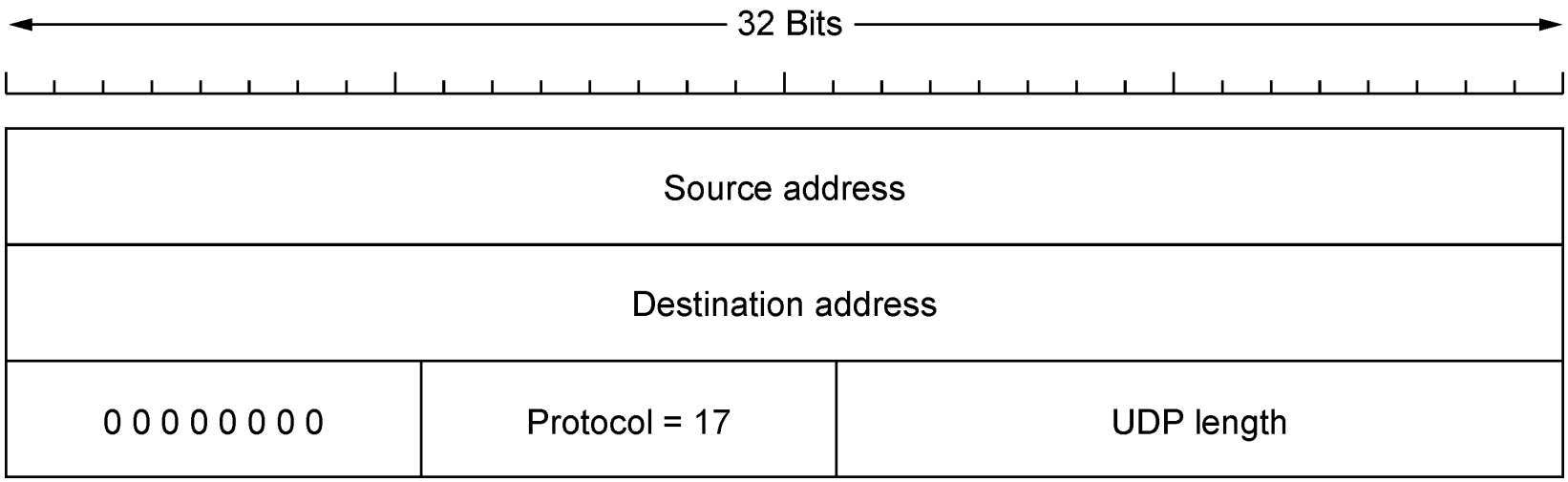


The UDP header.

Introduction to UDP (2)

Checksum covers UDP segment and IP pseudoheader

- Fields that change in the network are zeroed out
- Provides an end-to-end delivery check

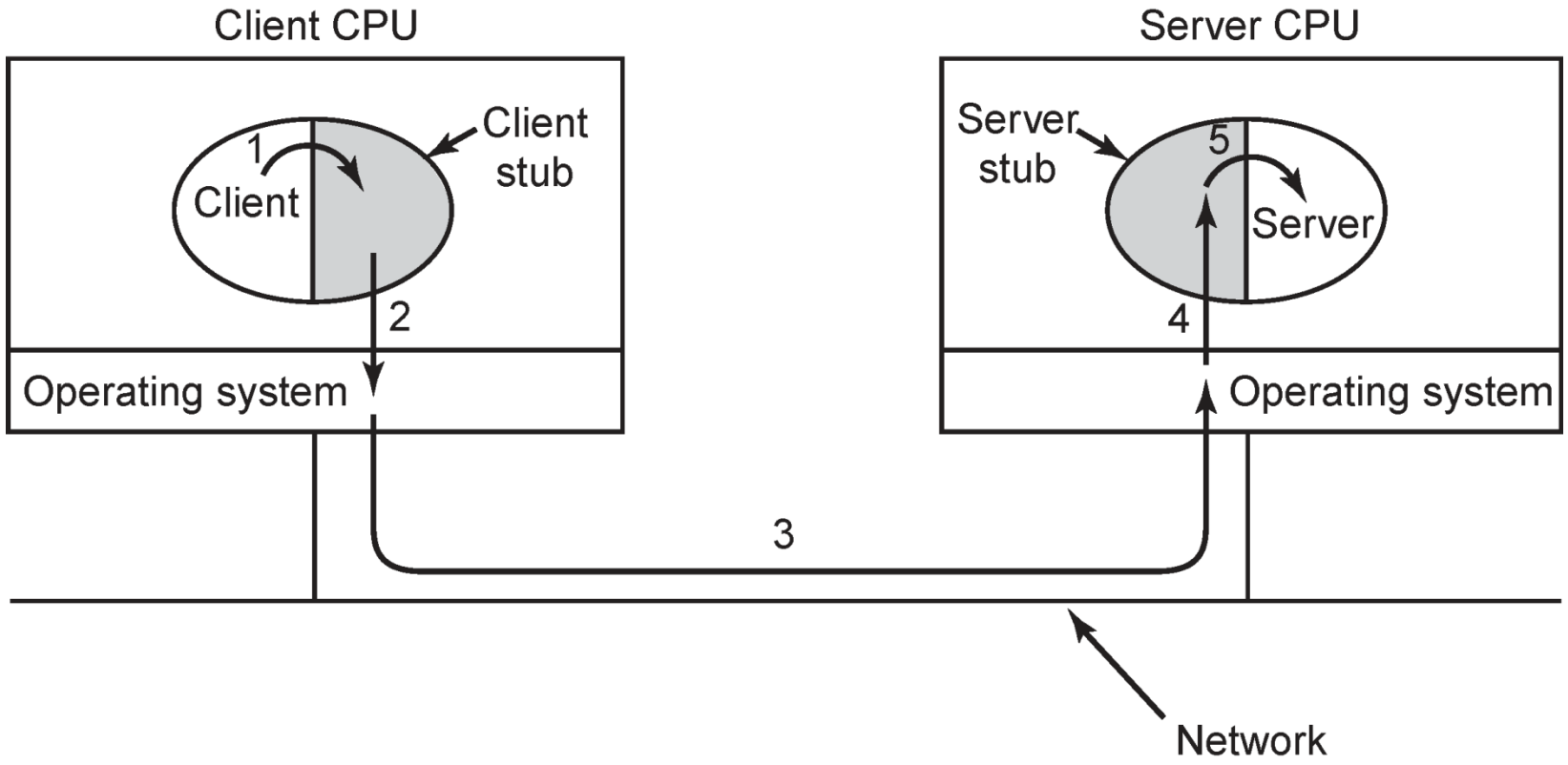


The IPv4 pseudoheader included in the UDP checksum.

RPC (Remote Procedure Call)

RPC connects applications over the network with the familiar abstraction of procedure calls

- Stubs package parameters/results into a message
- UDP with retransmissions is a low-latency transport

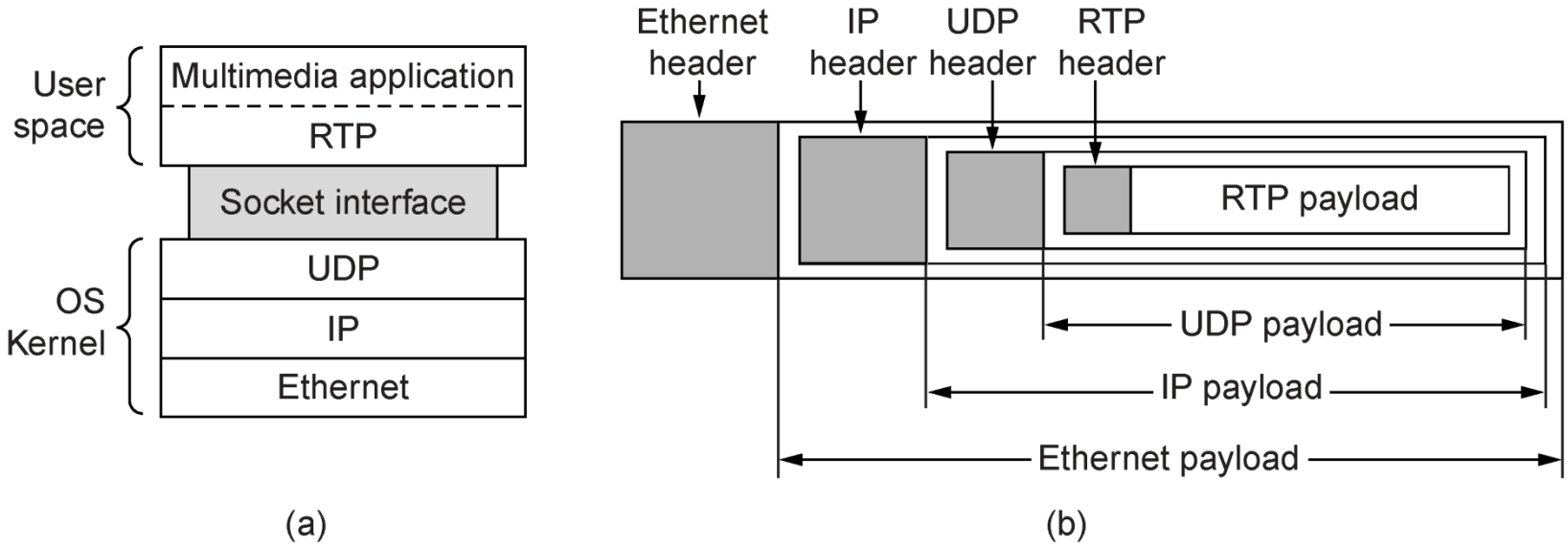


Steps in making a remote procedure call. The stubs are shaded.

Real-Time Transport (1)

RTP (Real-time Transport Protocol) provides support for sending real-time media over UDP

- Often implemented as part of the application

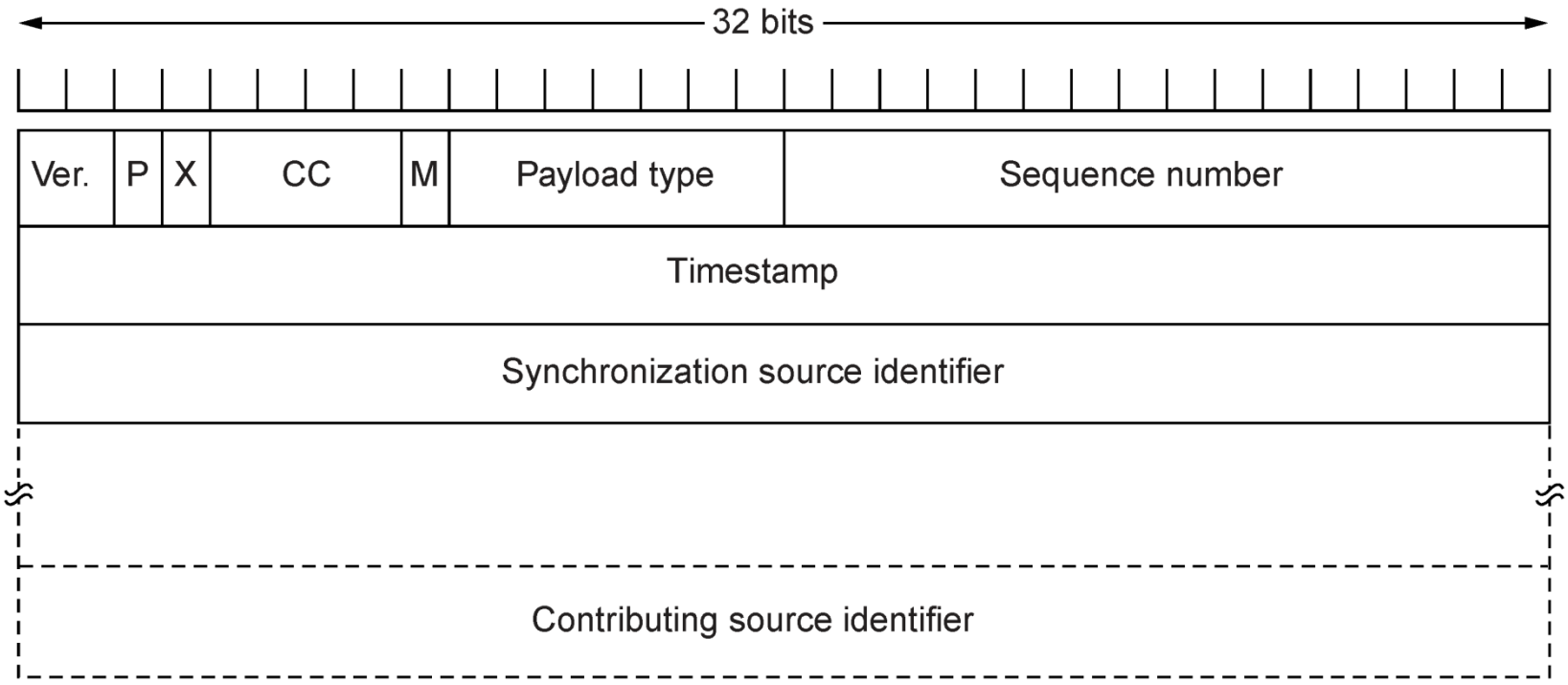


(a) The position of RTP in the protocol stack. (b) Packet nesting.

Real-Time Transport (2)

RTP header contains fields to describe the type of media and synchronize it across multiple streams

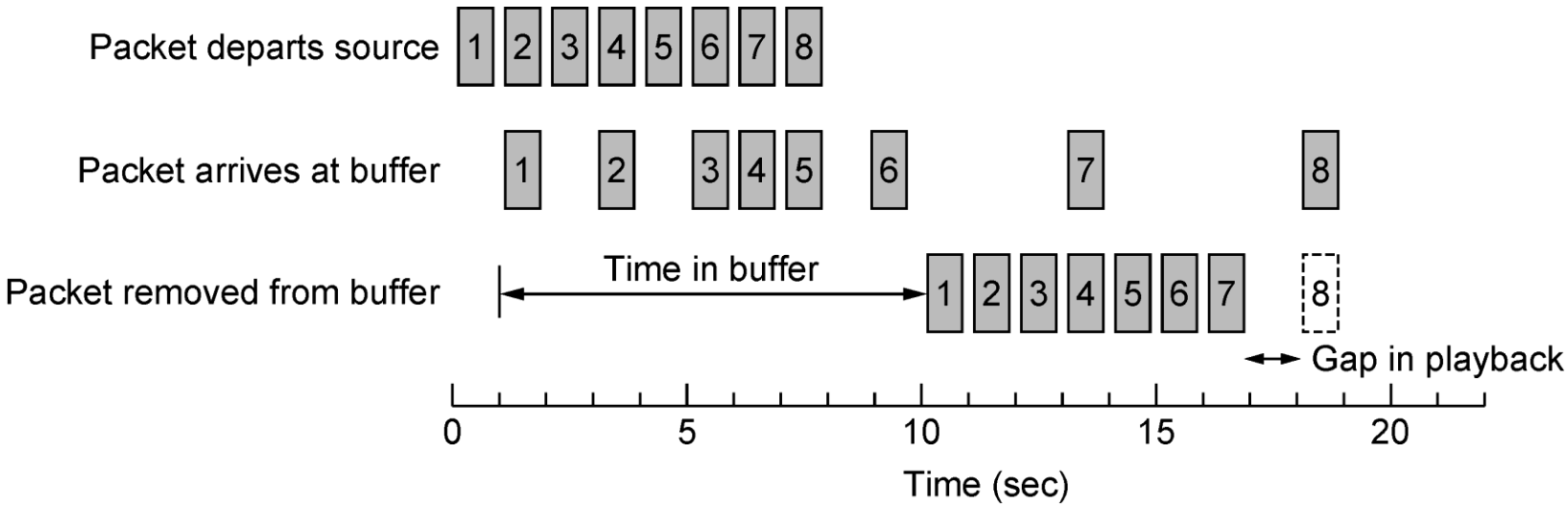
- RTCP sister protocol helps with management tasks



The RTP header.

Real-Time Transport (3)

Buffer at receiver is used to delay packets and absorb jitter so that streaming media is played out smoothly

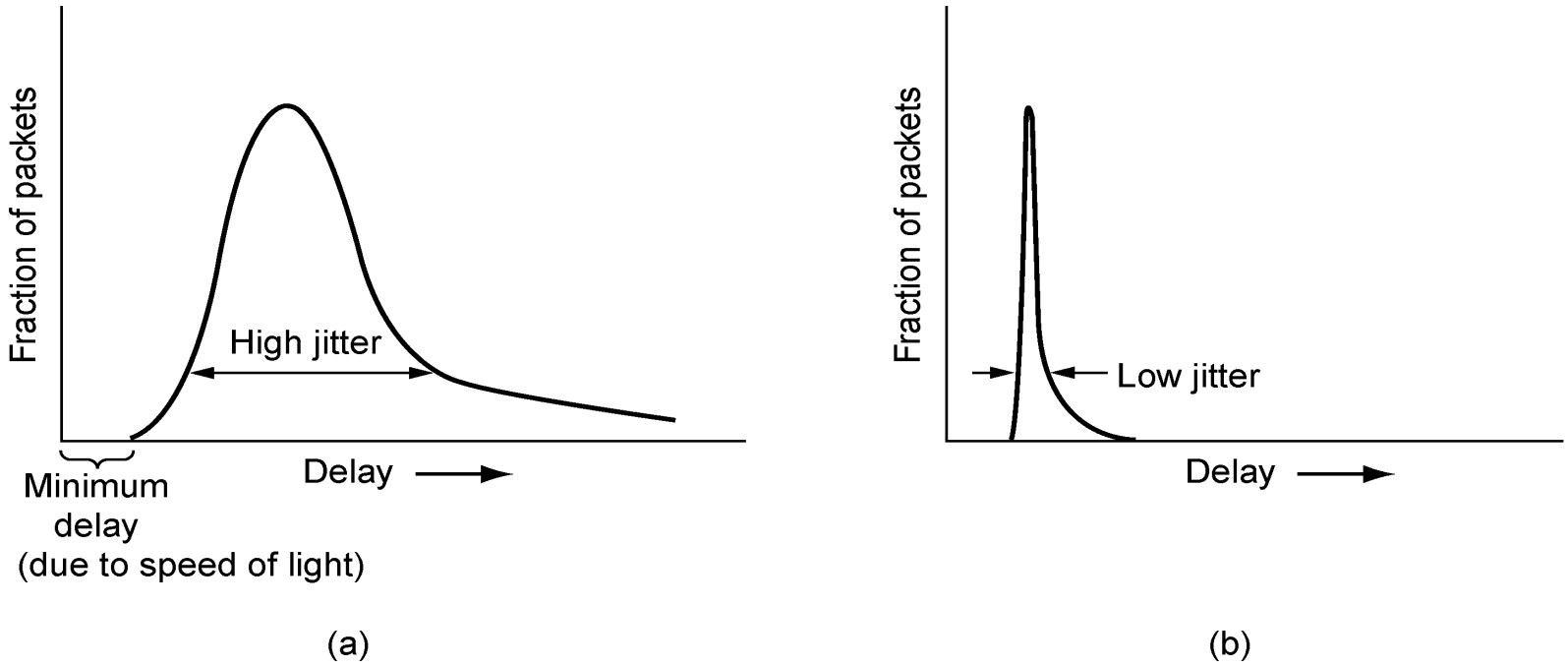


Smoothing the output stream by buffering packets.

Real-Time Transport (3)

High jitter, or more variation in delay, requires a larger playout buffer to avoid playout misses

- Propagation delay does not affect buffer size



(a) High jitter. (b) Low jitter.

The Internet Transport Control Protocols: TCP

- TPC (transport Control Protocol) is designed to provide reliable connection over unreliable internetwork.
- Internetwork
  - Different topologies, bandwidths, delays, packet sizes, and other parameters.
  - TCP is designed to be robust against many types of failures.
- The TCP stream may accept user data stream from local processes, breaks them up into pieces not exceeding 64KB (fits iin the Ethernet frame) and sends this piece as a separate TCP datagram.
- When datagrams arrive at a machine, they are given to the TCP entity, which reconstructs the original byte streams.
- The IP layer gives no guarantee that datagrams will be delivered properly and any indication of how fast these will be delivered.

The Internet Transport Control Protocols: TCP

- It is up to the TCP to send datagrams fast enough to make enough capacity but not cause congestion and to timeout and retransmit messages that are not delivered.
- It should also assemble in the order.

The Internet Transport Control Protocols: TCP

- The TCP service model »
- The TCP segment header »
- TCP connection establishment »
- TCP connection state modeling »
- TCP sliding window »
- TCP timer management »
- TCP congestion control »

The TCP Service Model (1)

- TCP service is obtained by both the sender and the receiver creating end points called sockets.
- Each socket has a socket number consisting of of IP address and port number of the machine.
- Port is the TCP name for a TSAP.
- A connection must be explicitly established between a socket on one machine and a socket on other machine.
- Port numbers below 1024 are reserved for well-known services.
  - They can only be started by privileged users
- Other port numbers 1024 to 49151 are reserved for unprivileged users.

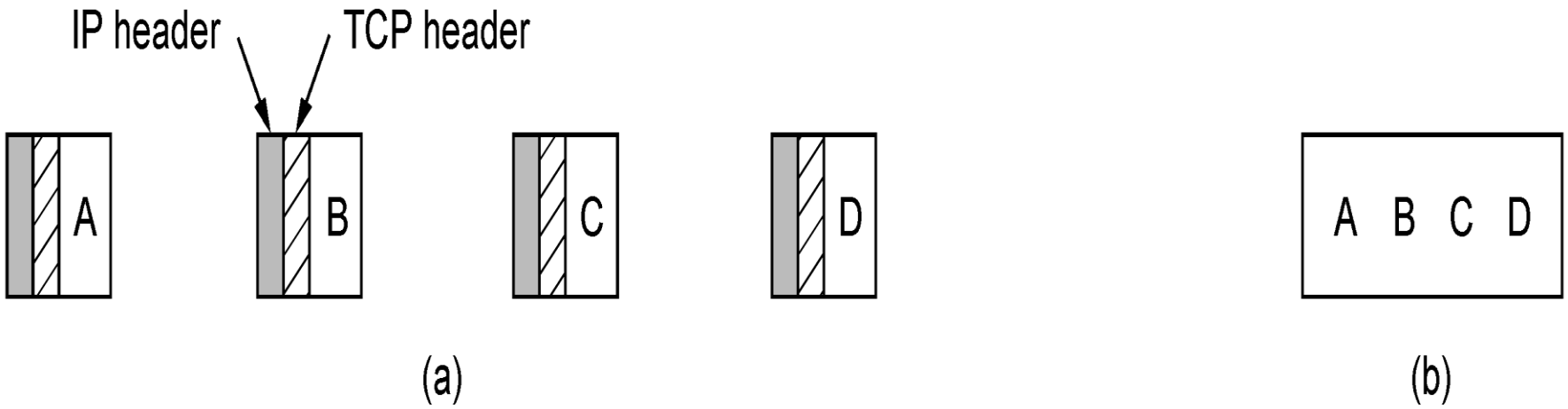
The TCP Service Model (1)

- TCP provides applications with a reliable byte stream between processes; it is the workhorse of the Internet
  - Popular servers run on well-known 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

The TCP Service Model (2)

- Applications using TCP see only the byte stream [right] and not the segments [left] sent as separate IP packets



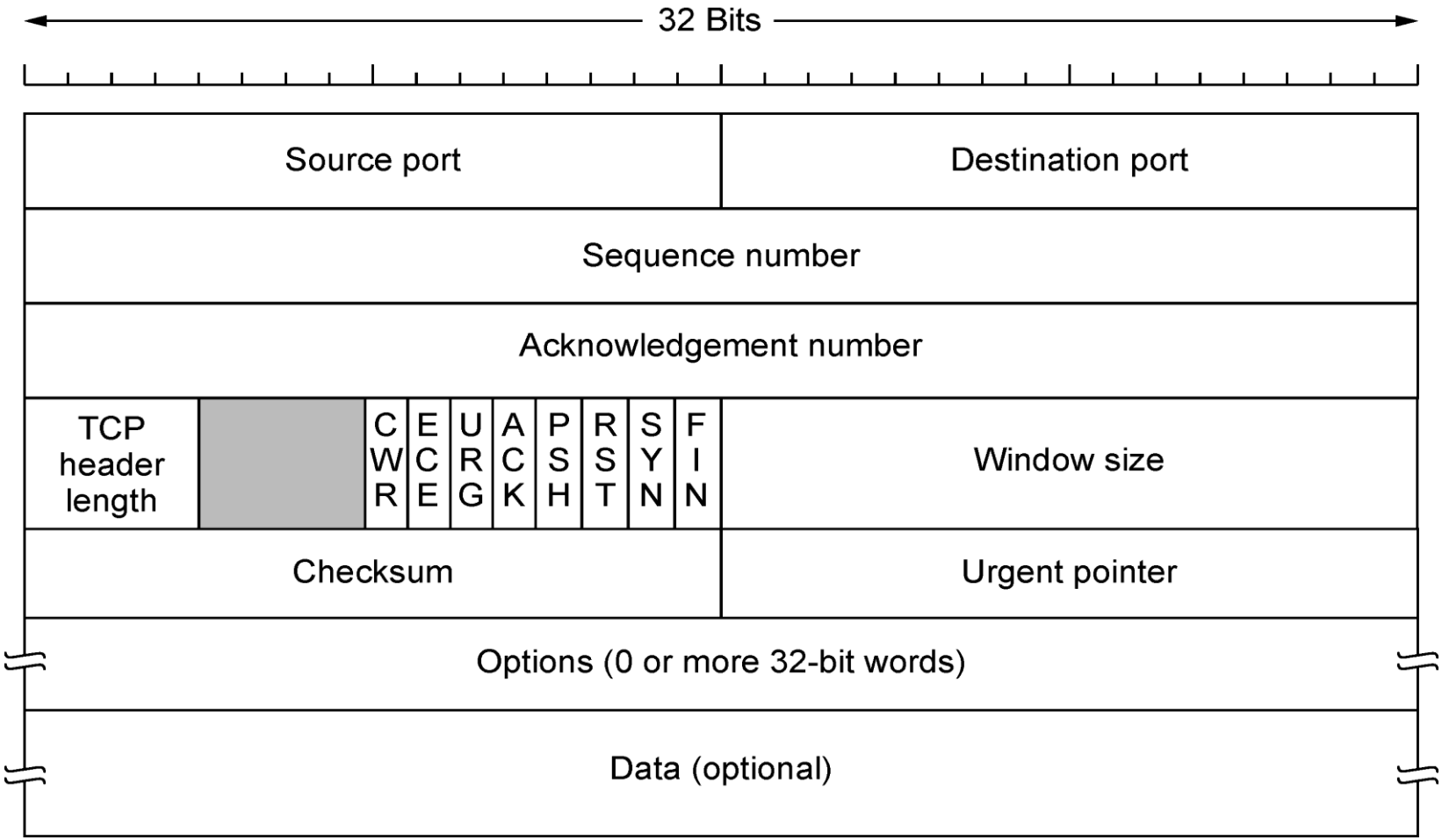
(a) Four 512-byte segments sent as separate IP datagrams. (b) The 2048 bytes of data delivered to the application in a single call.

Overview of The TCP Protocol

- Data is exchanged in the form is segments.
- Each segment has 20 byte header +optional part followed by data.
- Each segment must fit into 65515 byte payload (including header).
- Basic protocol
  - Sliding window protocol with a dynamic window size
  - When a sender transmits a segment, it also starts a timer.
  - The receiver send ack segment with the expected sequence number.
  - If the sender timer goes off, before the ack is received, the sender retransmits the message again.
- Issues
  - Out of order arrival of segments
  - Sender times out and retransmits

The TCP Segment Header

- TCP header includes addressing (ports), sliding window (seq. / ack. number), flow control (window), error control (checksum) and more.



The TCP header.

The TCP Segment Header

- Port addresses
- Sequence number
- Ack number
- TCP header length: number of 32-bit words in TCP header.
- 1 bit flags
  - CWR and ECE are used for signal congestion when ECN is used.
  - ECE is set to signal an ECN-Echo to a TCP sender.
  - CWR (Congestion Window Reduced) is set by sender to inform the receiver that it has reduced the load and can stop sending ECN-Echo
  - URG is set to 1 of the urgent pointer is used.
  - ACK is set to one to indicate ACK is valid.
  - PSH it indicated pushed data. Do not wait for complete data.
  - RST bit is set to abruptly reset connection.
  - SYN bit is used to establish connection.
  - FIN bit is used to release connection.

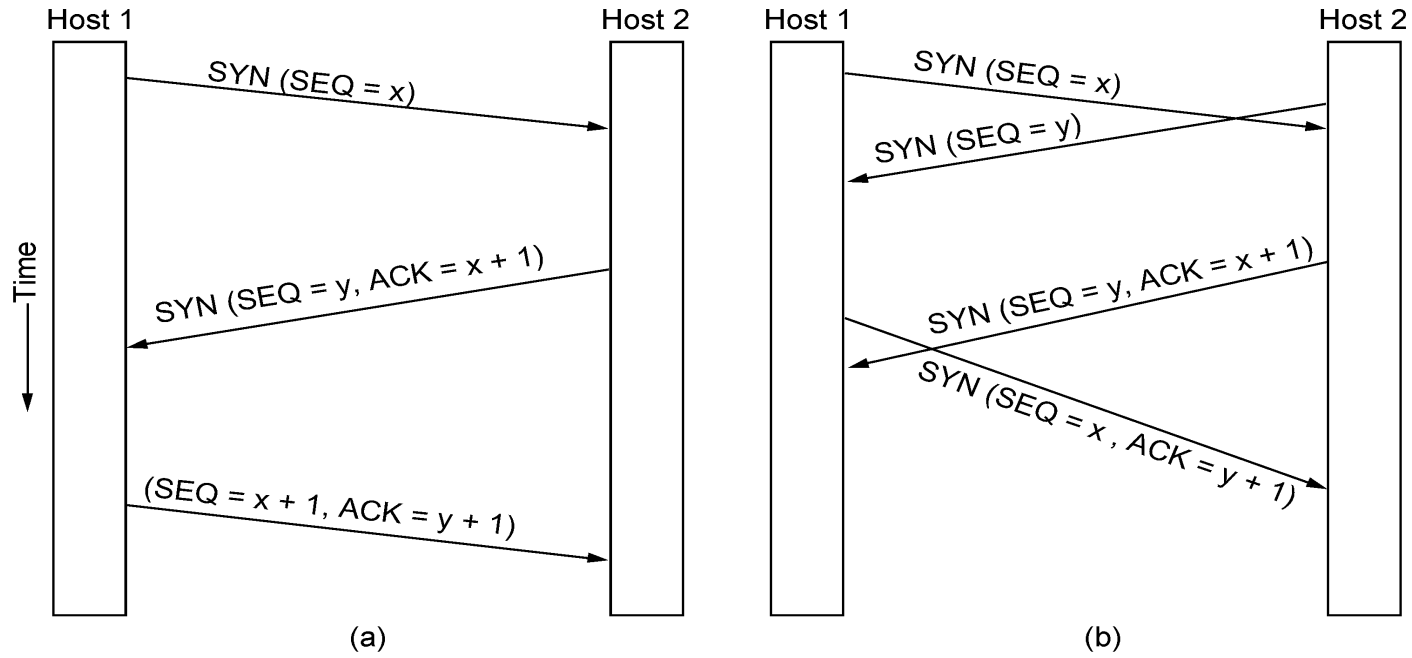
Overview of The TCP Protocol

- **Window-size (WIN)** tells how many bytes may be sent starting at the byte acknowledged.
  - If WIN=0, the sender must stop.
- Options
  - MSS (Maximum segment size)
    - Using larger segment is better as the header data can be amortized over more data
  - Timestamp
  - SACK (Selective acknowledgement)
    - Lets the receiver to send a ranges of sequence numbers that it has received.

TCP Connection Establishment

TCP sets up connections with the three-way handshake

- Release is symmetric, also as described before



(a) TCP connection establishment in the normal case.
(b) Simultaneous connection establishment on both sides.

TCP Connection Release

- To release a connection, either party can send a TCP segment with the FIN bit set.
- When FIN is acknowledged, the direction is shutdown.
  - Other direction may continue.
- To avoid two-army problem, timers are used
  - If a response to a FIN is not forthcoming within two packet lifetimes, the sender pf FIN releases the connection.
  - The other side will time out as well.

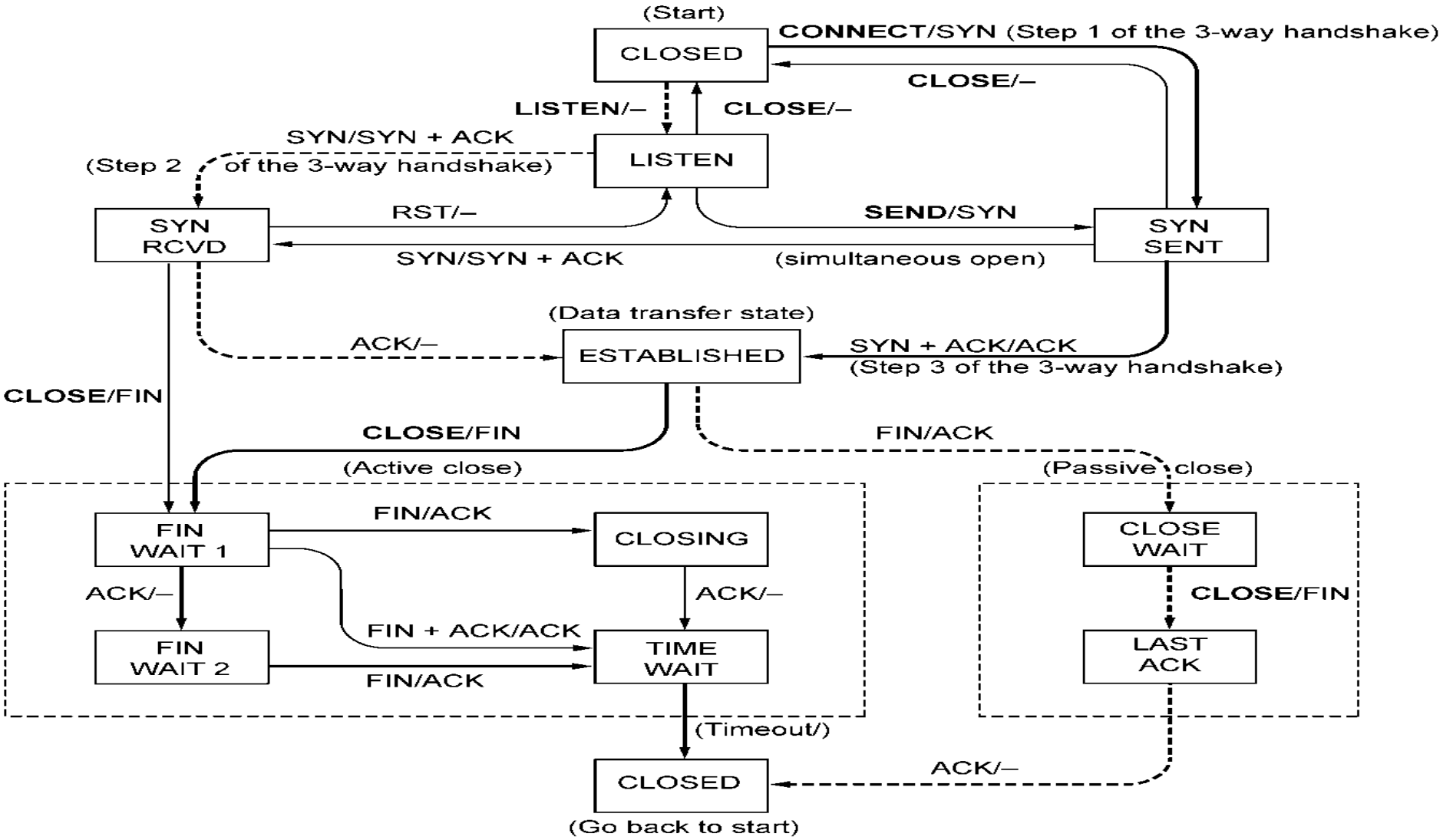
TCP Connection State Modeling (1)

The TCP connection finite state machine has more states than our simple example from earlier.

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.

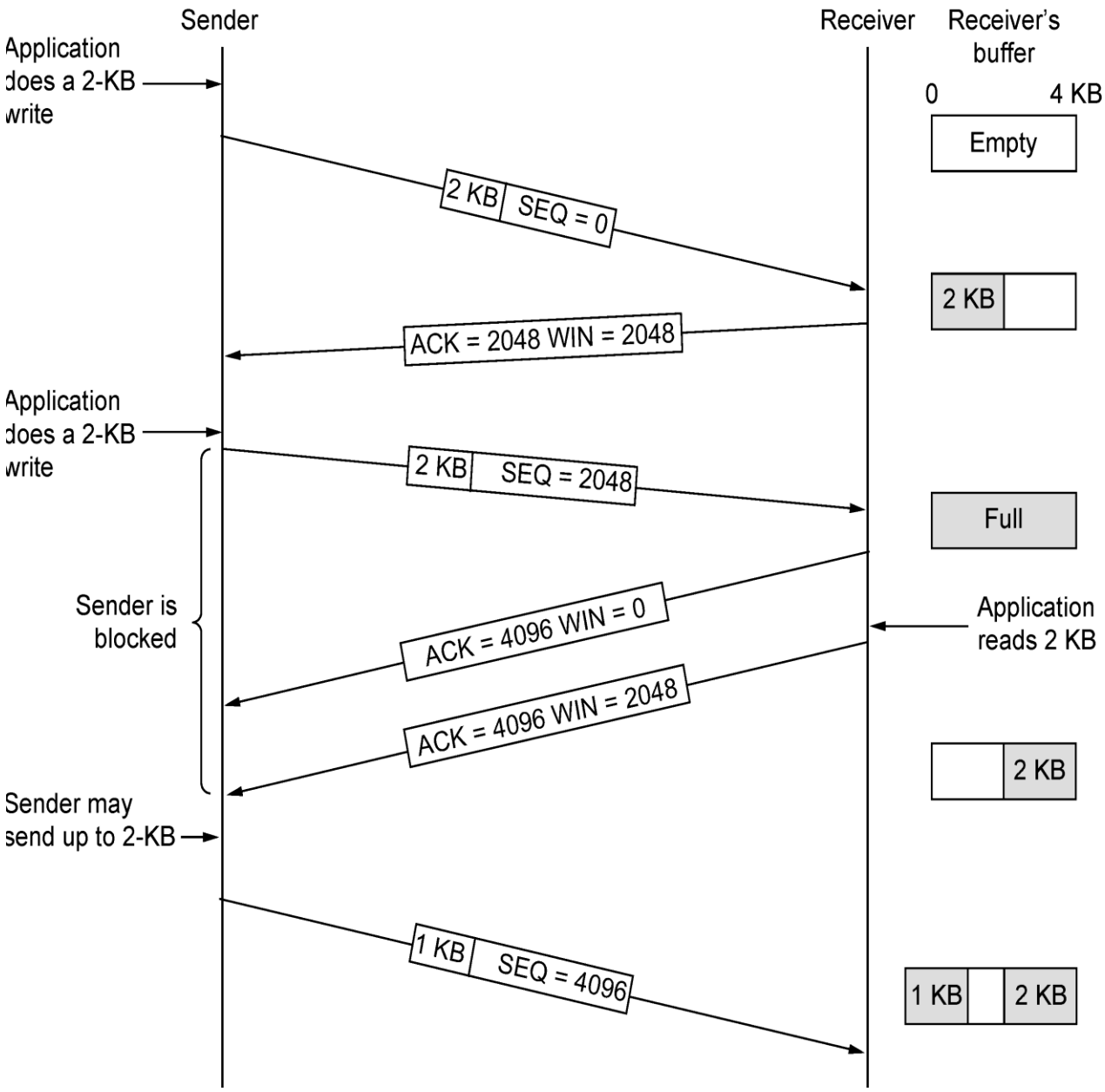
TCP Connection State Modeling (2)



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 with the event causing it and the action resulting from it, separated by a slash.

TCP Sliding Window (1)

- TCP adds flow control to the sliding window as before
- ACK + WIN is the sender's limit



Window management in TCP.

TCP Sliding Window

- If the sender transmits 2048 data, receiver stores in the buffer and sends ack.
- Sender sends ACK and WIN=2048
- Sender sends another 2KB.
  - As the buffer is full, receiver sends ack with WIN=0
- Sender should stop sending
- Sender resumes after receiver send ACK with WIN=2048.

- Window probe
  - Sender may send 1-byte segment to force the receiver to renounce the next byte expected.
  - This is to prevent the deadlock, if the window updates ever gets lost.

TCP Sliding Window..

- Senders need not transmit data as soon as it receives from the application.
- Receivers need not send ACKs as soon as possible.
- This freedom can be used to improve performance.

- Issue:
  - some times each character creates a 21 byte segment, which it gives to IP 41-byte IP datagram.
  - At the receiver side, TCP immediately sends a 40-byte ack.
  - 162 bytes of bandwidth are used and four segments are sent for each character typed.

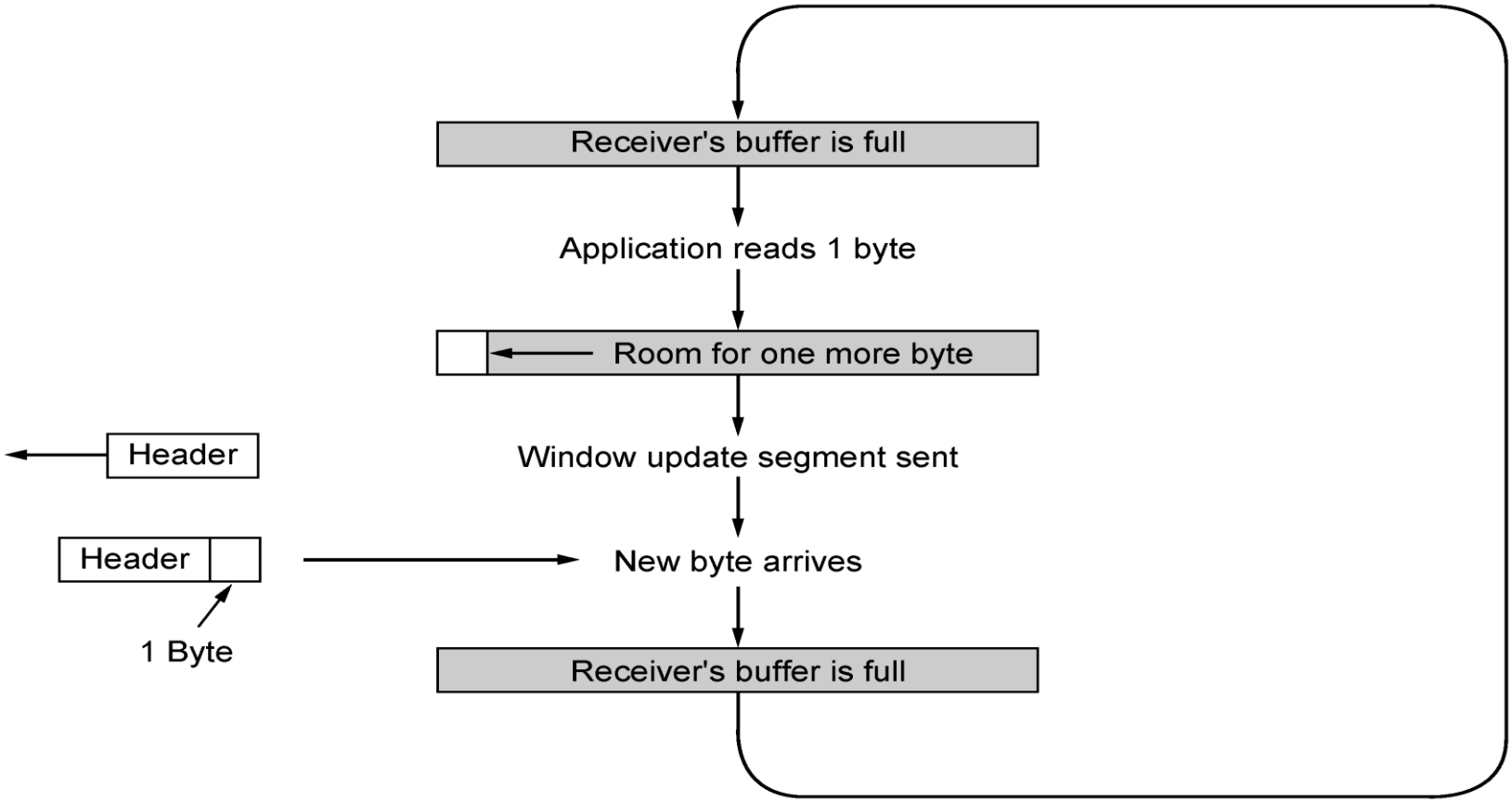
TCP Sliding Window..

- Solution: Delayed acks.
  - Delay ack up to 500 msec.
  - But if sender sends small packets, we will have the same problem.
- Nagle's algorithm
  - Send the first piece and buffer the rest until the first piece is ack.
  - Send all the buffered data in one TCP segment.
  - This is widely used algorithm, but for some applications, like interactive games, it is not applicable.
- Silly window syndrome
  - Receiving end only processes one byte at a time.
- Clark's algorithm
  - Wait until decent amount of space is available and advertise that instead.

TCP Sliding Window (2)

Need to add special cases to avoid unwanted behavior

- E.g., silly window syndrome [below]



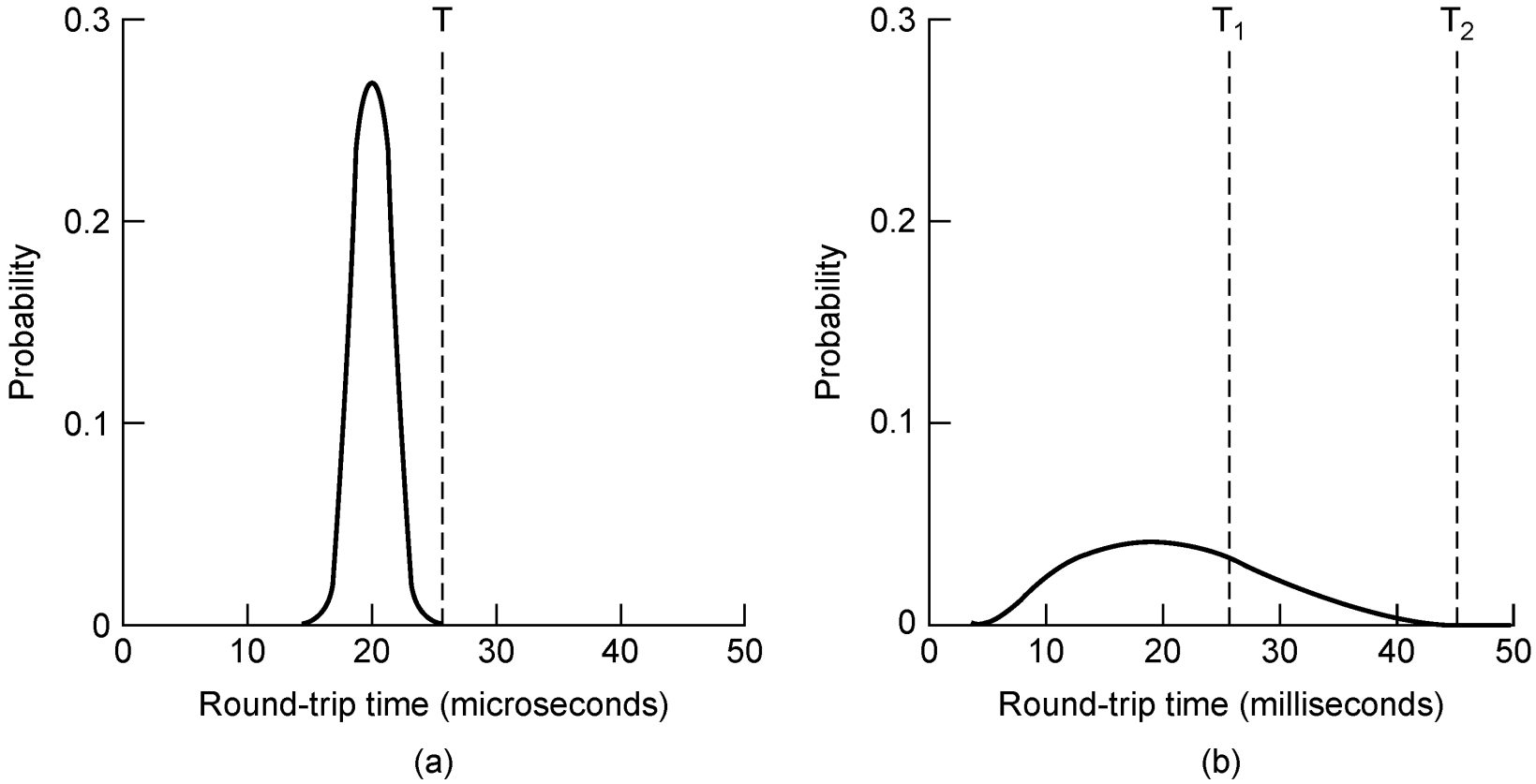
Silly window syndrome.

Receiver application reads single bytes, so sender always sends one byte segments

TCP Timer Management

TCP estimates retransmit timer from segment RTTs

- Tracks both average and variance (for Internet case)
- Timeout is set to average plus 4 x variance



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

TCP Timer Management

- Probability density function varies significantly.
- Variance is significant.
  - Things can change quickly.
- Change T based on the performance
- Jocobson algorithm
  - TCP maintains a variable SRTT (Smoothened round trip time)
  - TCP measures how long the ack took
  - It updates SRTT = α SRTT+(1- α) SRTT
  - The formula is called EWMA (exponentially Weighted Moving Average).
- Improved formula
  - Sensitive to variance in round trip times

TCP Timer Management

- Two more timers
- Persistence timer
  - Receiver sends ack asking sender to wait.
  - Later receiver updates the window, the packet with the update is lost
  - Situation: Both sender and receiver are waiting.
  - When persistence time goes off, the sender retransmits the probe. If it is still 0, the timer is initialized and cycle repeats
- Keepalive timer
  - If the connection is idle for a long time, one side will check whether the other side is live or not. If iit fails to respond, the connection is terminated.

TCP Congestion Control (1)

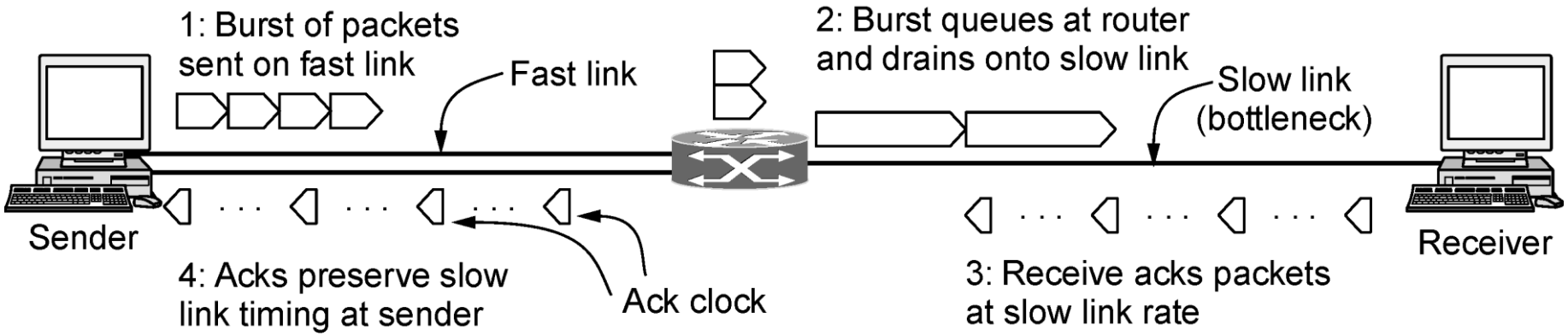
TCP uses AIMD with loss signal to control congestion

- Implemented as a congestion window (cwnd) for the number of segments that may be in the network
- Uses several mechanisms that work together

TCP Congestion Control (2)

Congestion window controls the sending rate

- Rate is cwnd / RTT; window can stop sender quickly
- ACK clock (regular receipt of ACKs) paces traffic and smoothes out sender bursts

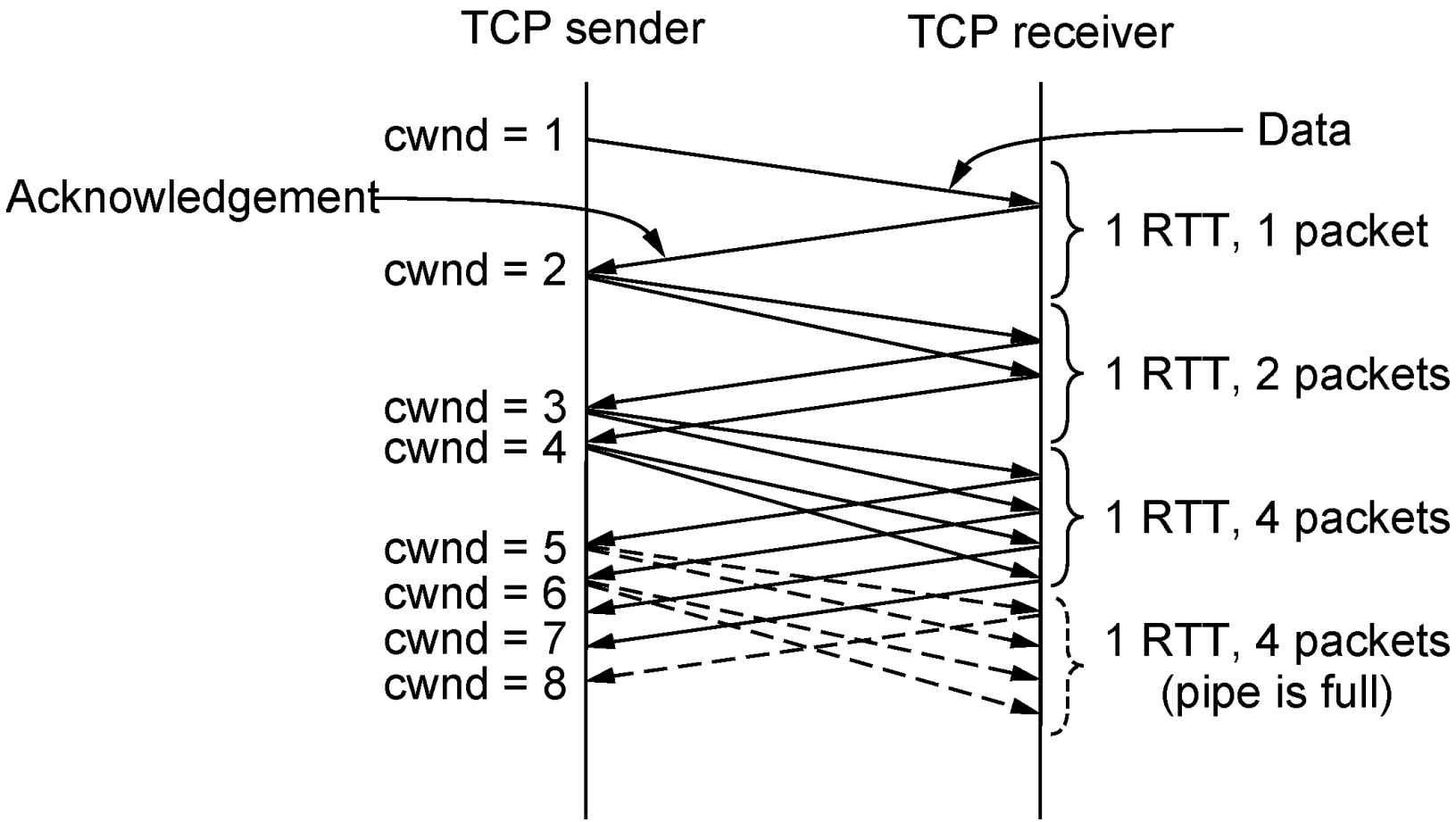


A burst of packets from a sender and the returning ack clock.

TCP Congestion Control (3)

Slow start grows congestion window exponentially

- Doubles every RTT while keeping ACK clock going

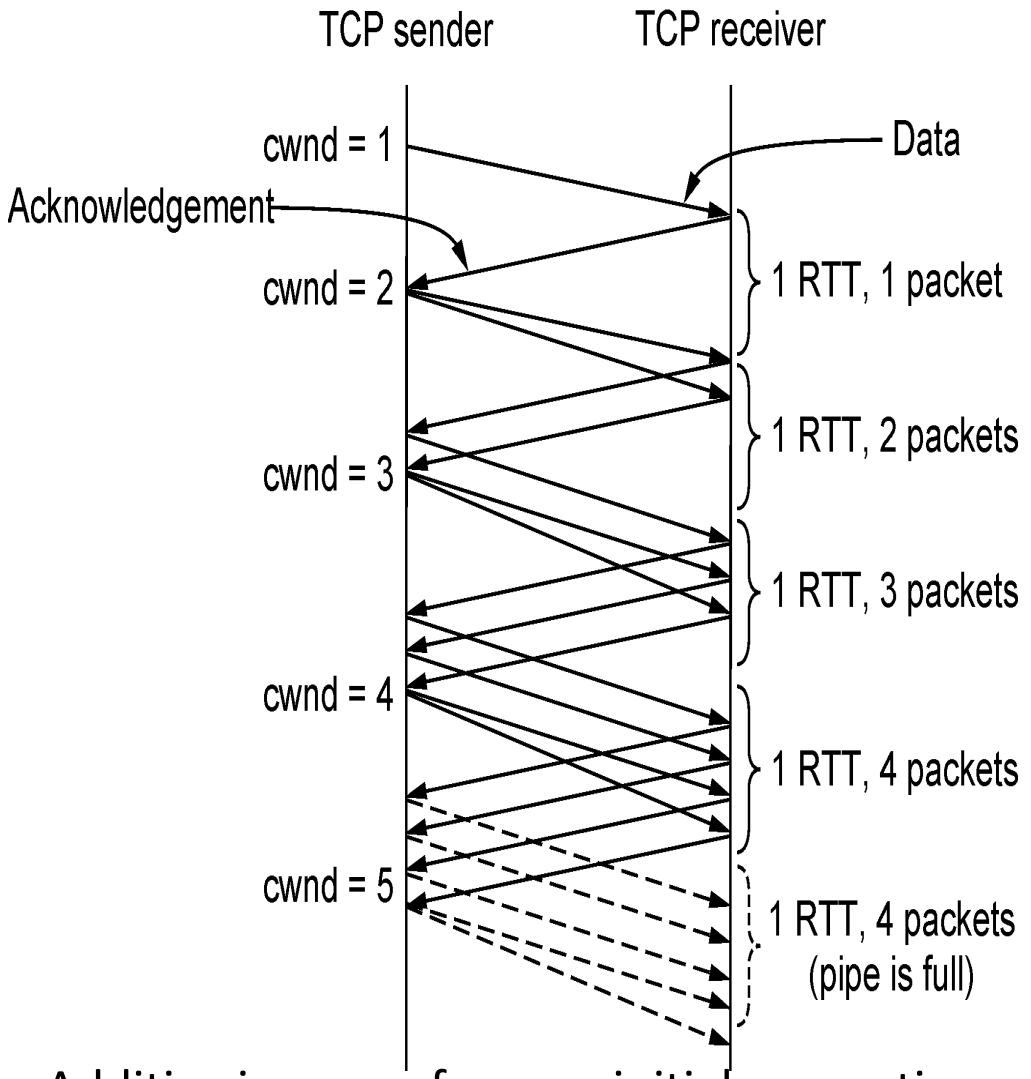


Slow start from an initial congestion window of one segment.

TCP Congestion Control (4)

Additive increase grows cwnd slowly

- Adds 1 every RTT
- Keeps ACK clock

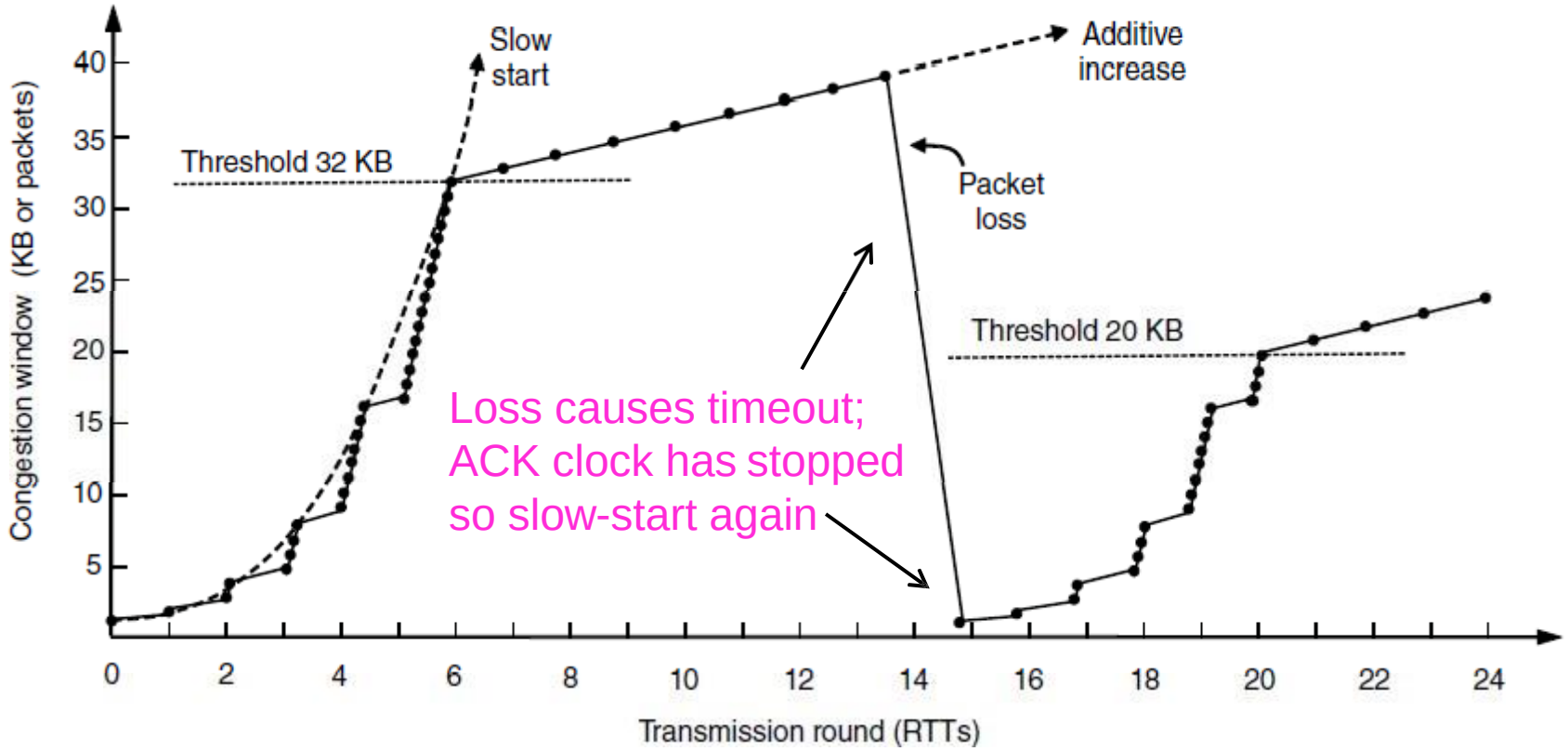


Additive increase from an initial congestion window of one segment.

TCP Congestion Control (5)

Slow start followed by additive increase (TCP Tahoe)

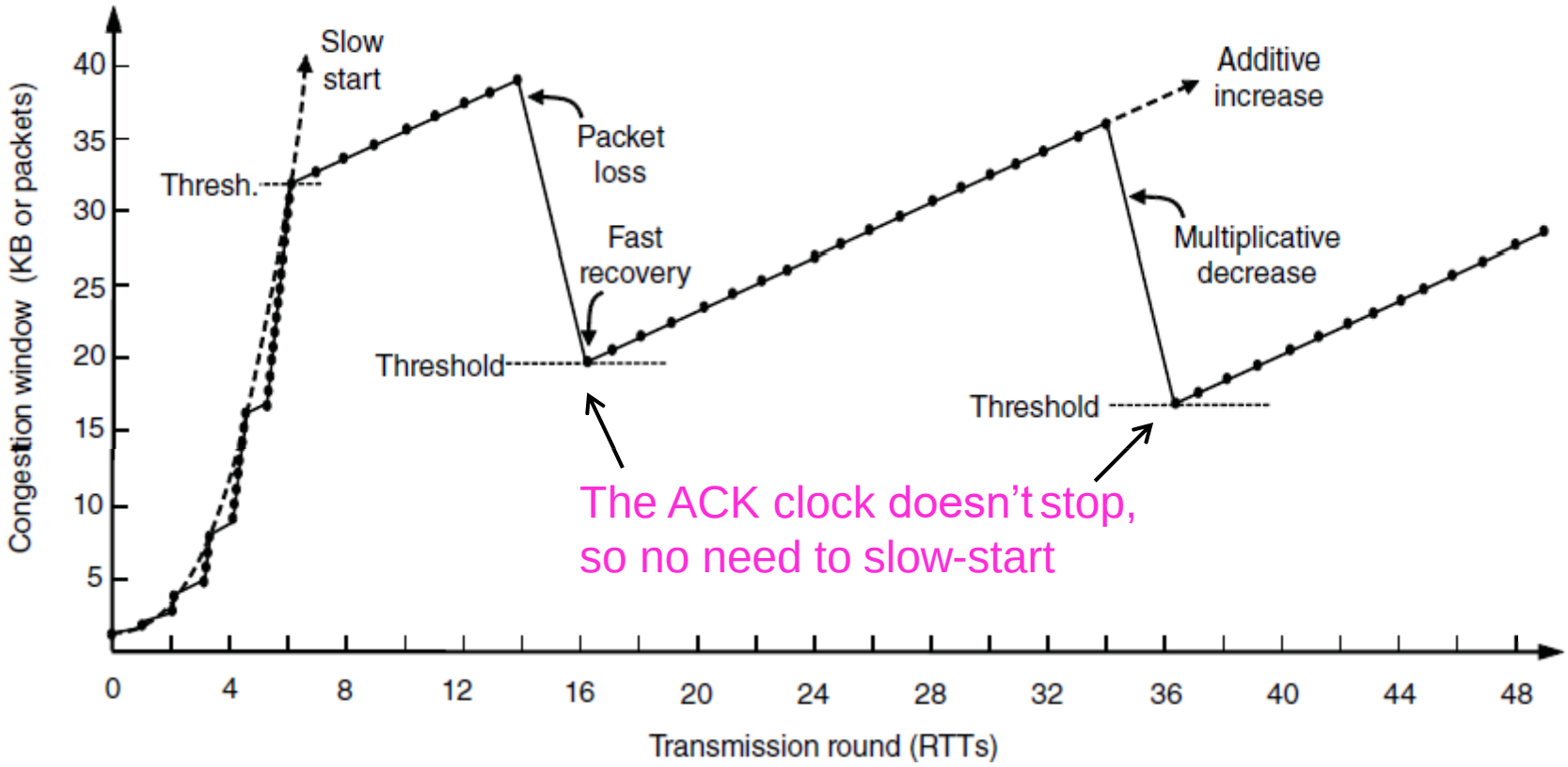
- Threshold is half of previous loss cwnd



TCP Congestion Control (6)

With fast recovery, we get the classic sawtooth (TCP Reno)

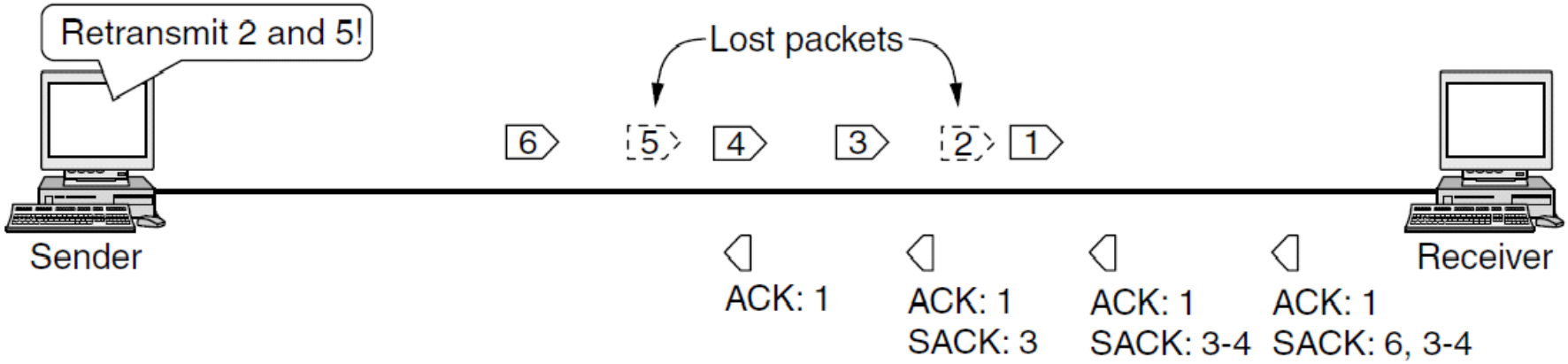
- Retransmit lost packet after 3 duplicate ACKs
- New packet for each dup. ACK until loss is repaired



TCP Congestion Control (7)

SACK (Selective ACKs) extend ACKs with a vector to describe received segments and hence losses

- Allows for more accurate retransmissions / recovery



No way for us to know that 2 and 5 were lost with only ACKs

Delay Tolerant Networking

- Assumption so far:
  - Sender and receiver are continuously connected by some working path.
- Other applications
  - Space network where satellites pass in and out of range of ground stations.
  - Networks involve submarine, buses, mobile phones have a intermittent connectivity

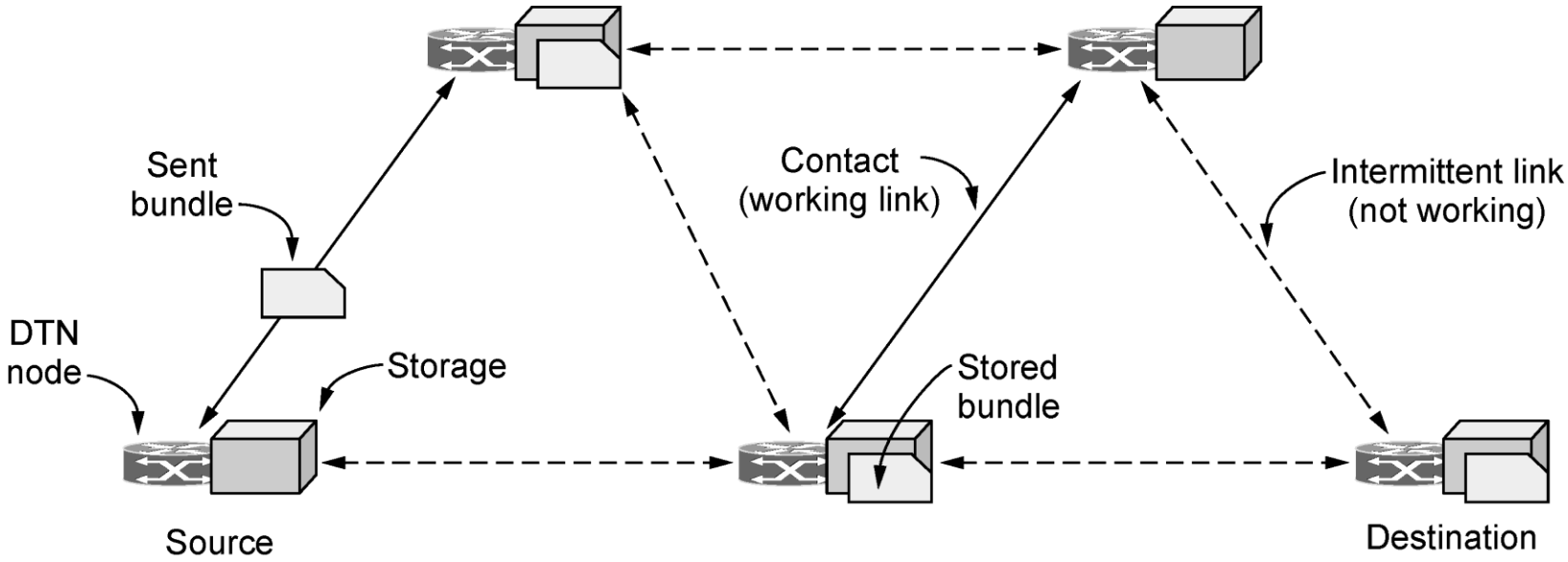
- DTNs (Delay Tolerant Networks) store messages inside the network until they can be delivered

- DTNs (Delay Tolerant Networks) store messages inside the network until they can be delivered

- This model accepts delays like postal system.
- Several applications
  - Copying of data to data centers
  - Sending traffic during off times.
  - Doing backups at night
  - It is possible to send huge datasets during off-time.

DTN Architecture (1)

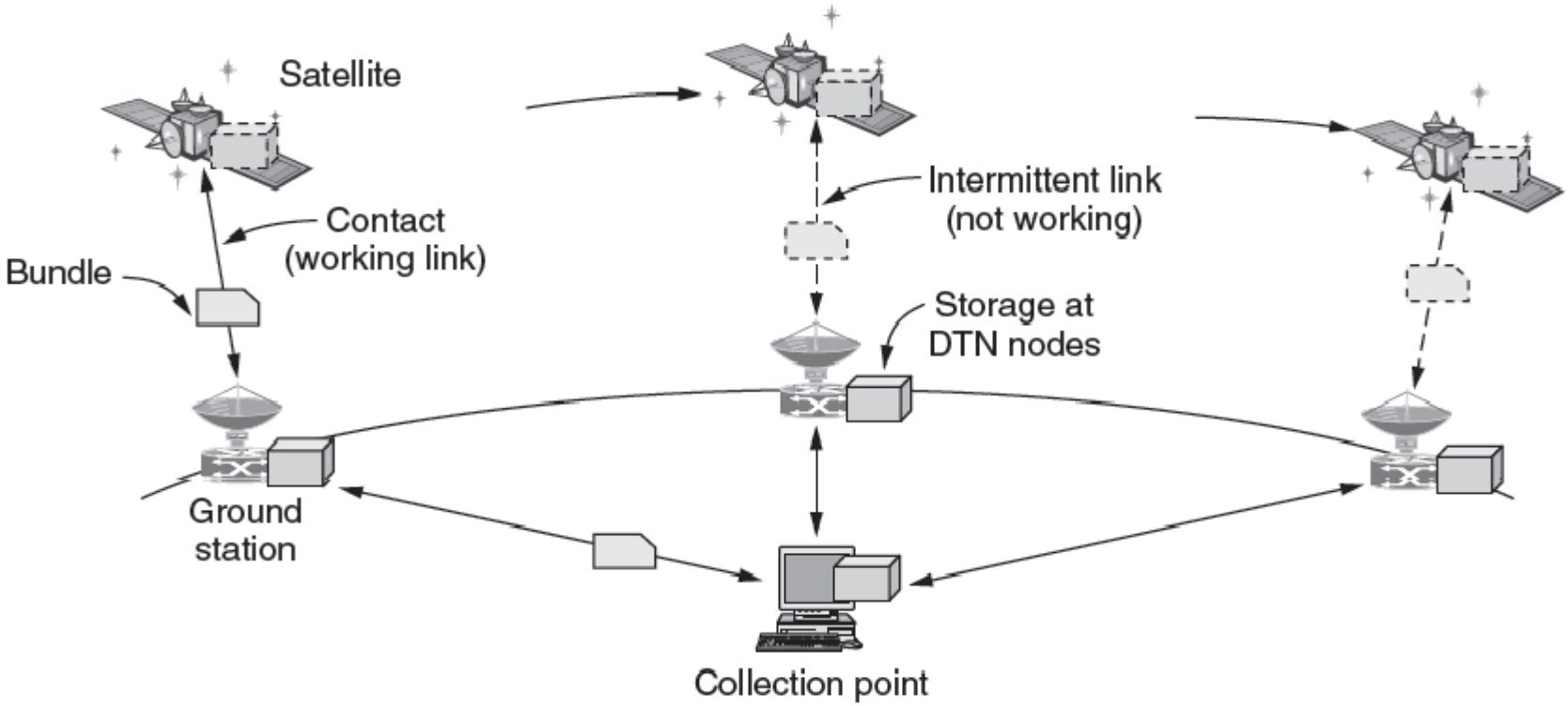
- Messages called bundles are stored at DTN nodes while waiting for an intermittent link to become a contact
  - DTN nodes are are equipped with storage
  - Bundles might wait hours, not milliseconds as in routers
  - May be no working end-to-end path at any time



Delay-tolerant networking architecture.

DTN Architecture (2)

Example DTN connecting a satellite to a collection point



End

Chapter 6