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)
- Congestion Control (拥塞控制)*
- 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)*
- Delay-Tolerant Networking (容延网络)*

THE TRANSPORT SERVICE (传输服务)

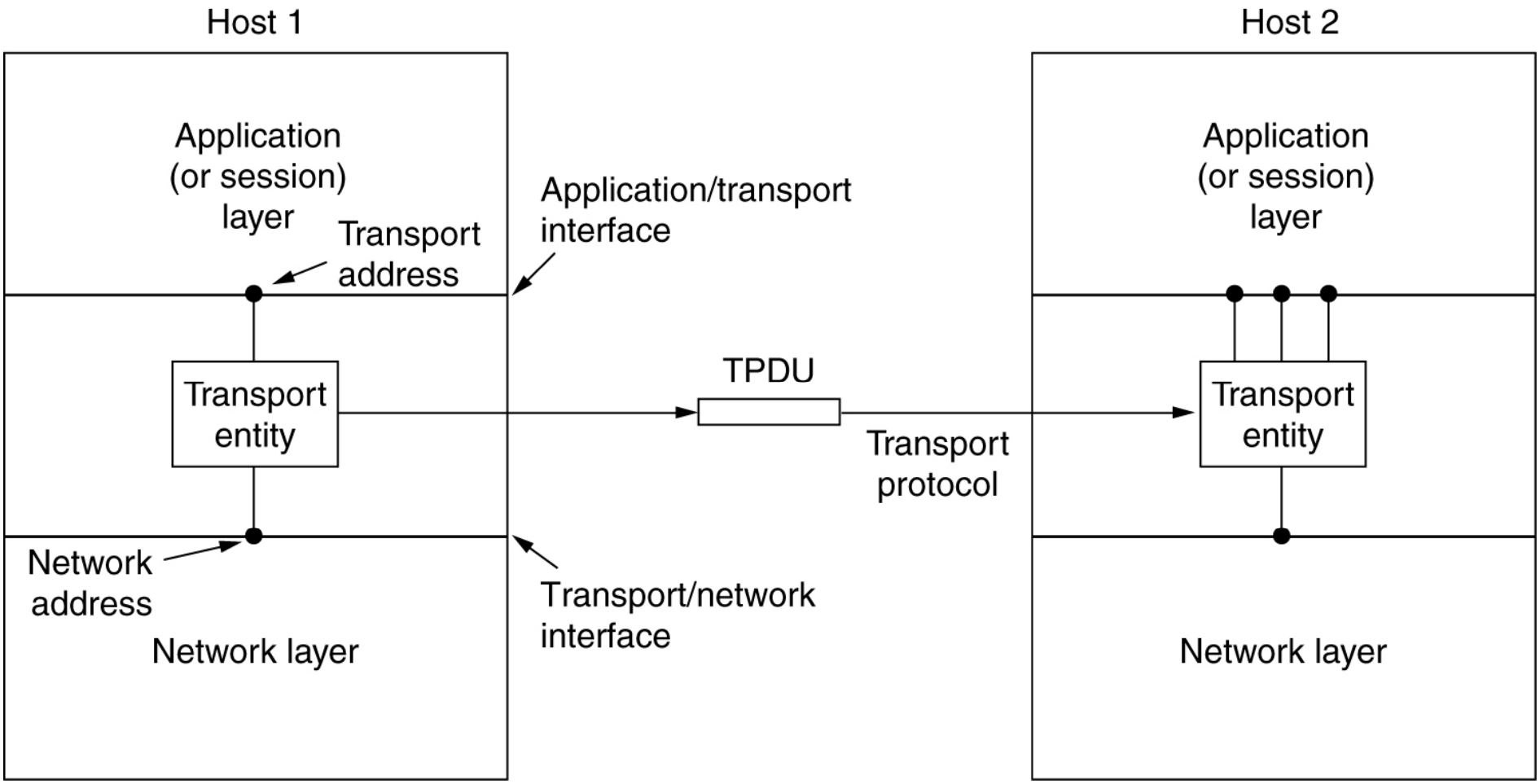
- Services Provided to the Upper Layers
- Transport Service Primitives (简单传输服务原语)
- Berkeley Sockets (套接字)
- An Example of Socket Programming:
  - An Internet File Server

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.

The Transport Service: Services

The network, transport and application layers



The Transport Service: Services

- There are two types transport services
  - Connection-oriented transport service
  - Connectionless transport service
- The similarities between transport services and network services
  - The connection–oriented service is similar to the connection-oriented network service in many ways:
    - Establishment, →data transfer, →release;
    - Addressing;
    - Flow control
  - The connectionless transport service is also similar to the connectionless network service.

The Transport Service: Services

- The differences between transport services and network services. Why are there two distinct layers?
  - The transport code runs entirely on the user's machines, but the network layer mostly runs on the routers which are usually operated by the carrier.
  - Network layer has problems (losing packets, router crashing, …)
  - The transport layer improves the QOS of the network layer.
  - → The transport service is more reliable than the network service.
  - →Applications programmers can write code according to a standard set of transport service primitives and have these programs work on a wide variety of networks.

The Transport Service: Hypothetical Primitives

- The transport layer provides some operations to applications programs, i.e., a transport service interface.
- Each transport service has its own interface.
- The transport service is similar to the network service, but there are also some important differences:
  - Network service models real network → unreliable.
    Transport services improves real network → reliable.
  - Network services are for network developers.
    Transport services are for application developers.

The Transport Service: Hypothetical 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

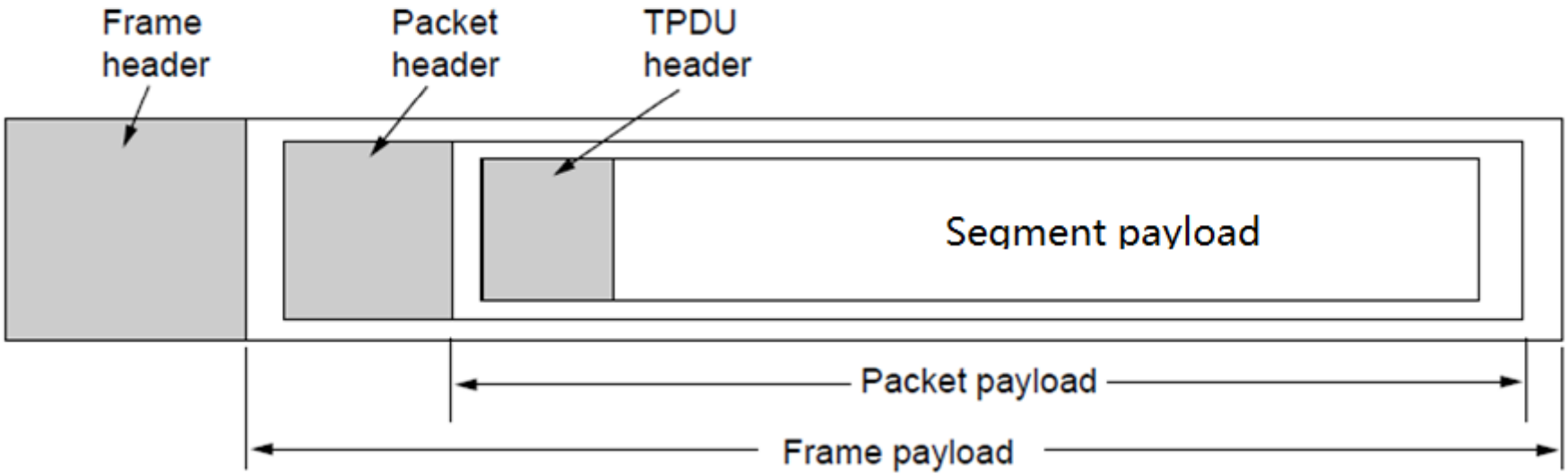
The Transport Service: Hypothetical Primitives

How to use these primitives for an application?

- Server App: LISTEN
- Client App: CONNECT
- Client ←SEND/RECEIVE→ Server
- Client Transport ← TPDU → Server Transport
  - Client network layer ← Packets→ Server network layer
  - Client data link layer ← Frames → Server data link layer
- Client/Server: DISCONNECT

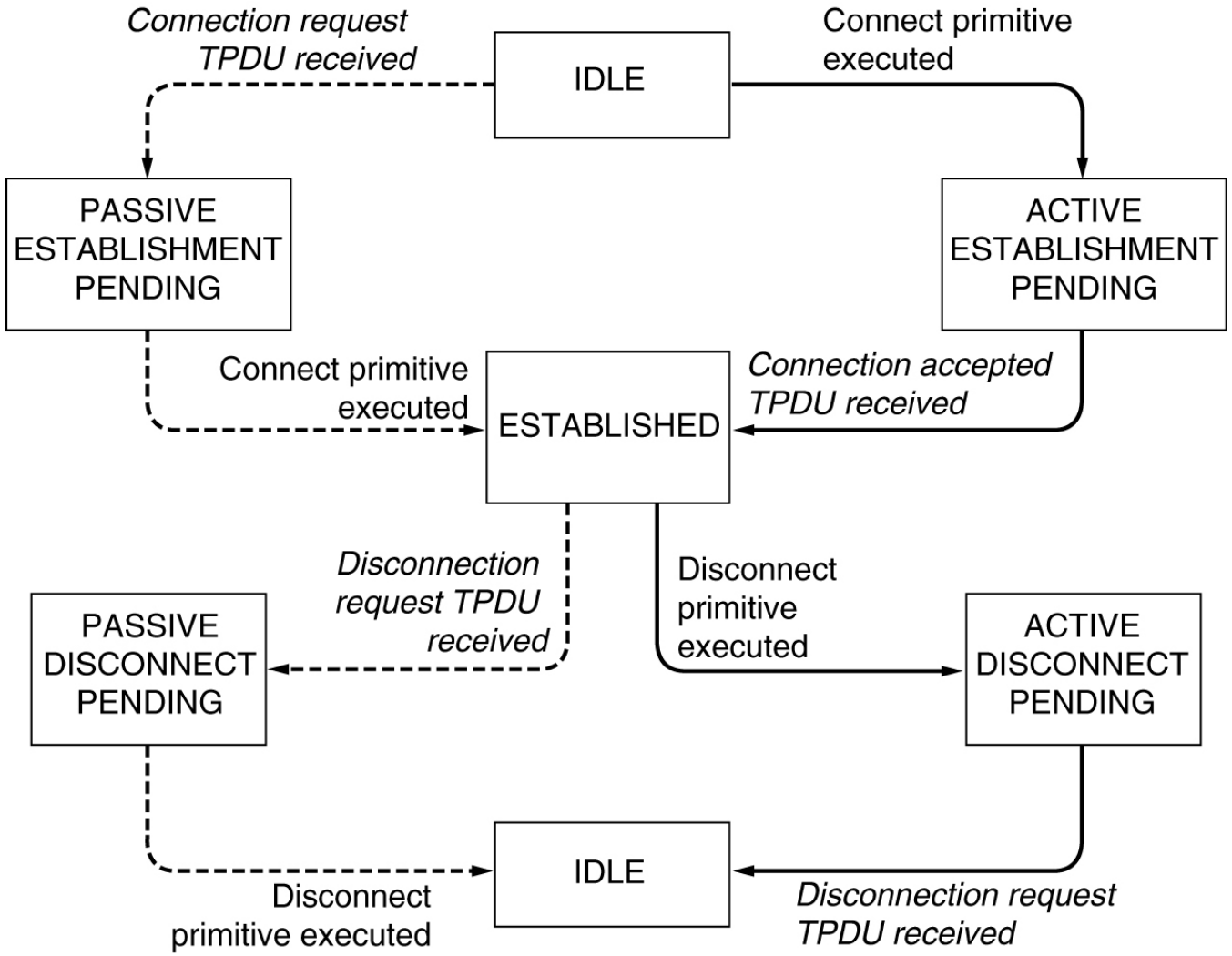
The Transport Service: Hypothetical Primitives

Nesting of Segments, packets, and frames



The Transport Service: Hypothetical Primitives

Connection establishment and connection release



The Transport Service: Berkeley Sockets

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

The Transport Service: Berkeley Sockets

- The C/S Application
  - Server: SOCKET/**BIND**/LISTEN/ACCEPT
  - Client: SOCKET/CONNECT
  - C/S: SEND/RECEIVE
  - C/S: CLOSE (symmetric)

The Transport Service: An example

- IFServer.c
- IFClient.c

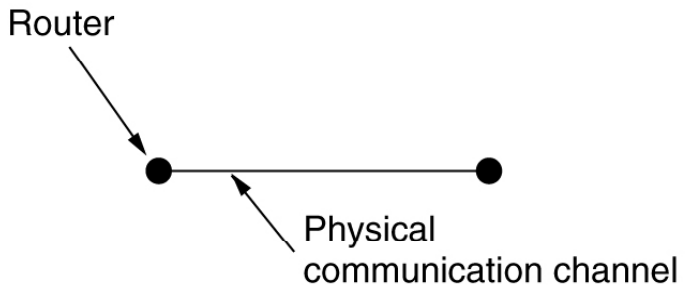
ELEMENTS OF TRANSPORT PROTOCOLS

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

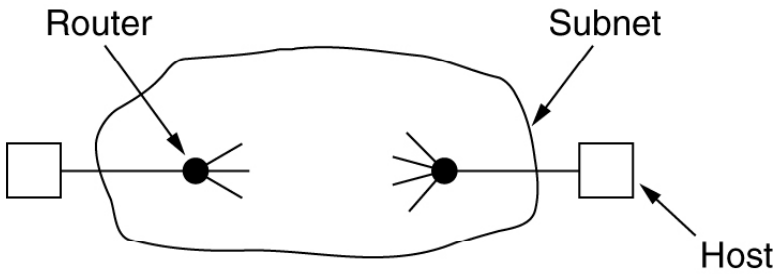
Elements of Transport Protocols

Transport protocol and data link protocol

- Similarities: Error control, sequencing, and flow control
- Differences
  - Environment: physical communication channel/subnet
  - Addressing: implicit/explicit
  - Connection establishment: simple/complicated
  - Storage capacity: no/yes (unpredictable/predictable)
  - Buffering and flow control: amount



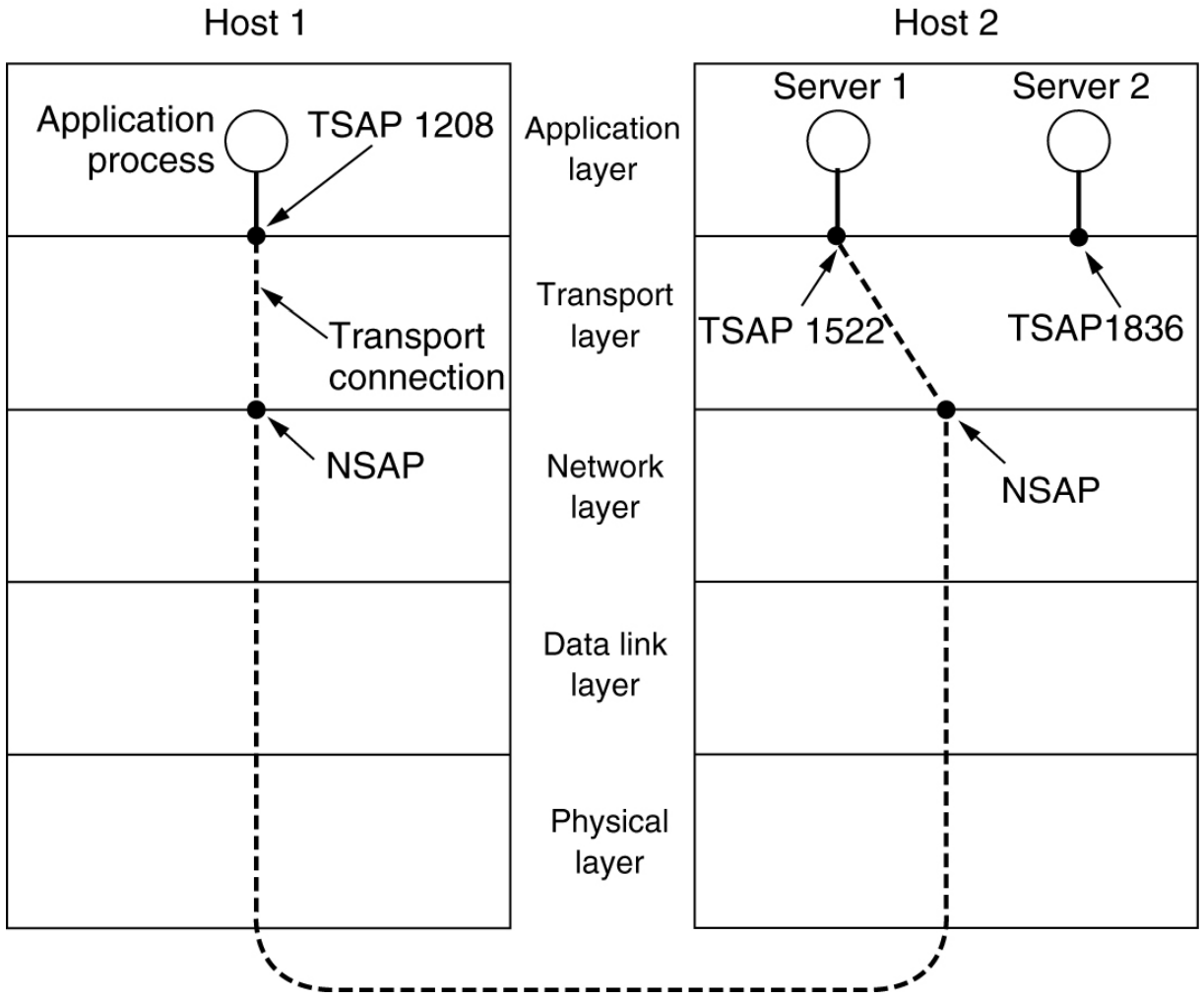
(a)



(b)

Elements of Transport Protocols: Addressing

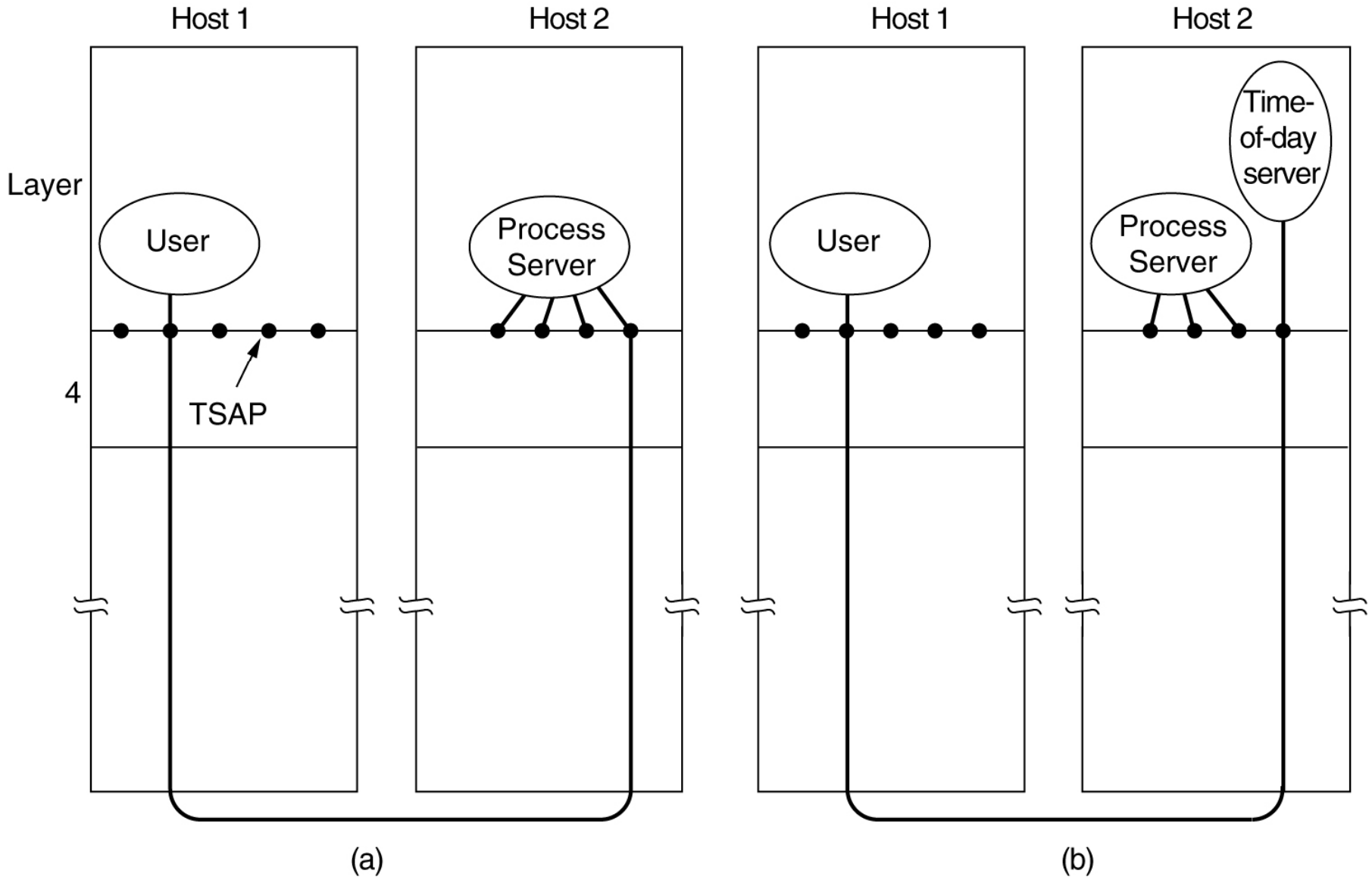
How to address?



Elements of Transport Protocols: Addressing

- NSAP: Network service access points
  - IP (32 bits)
- TSAP: Transport service access point
  - Port (16 bits)
- How to find out the server's TSAP
  - Some TSAP addresses are so famous that they are fixed.
  - Process server
    - To listen to a set of TSAPs at the same time.
    - To dispatch the request to the right server.
  - Process server + name server (directory server, 114)

Elements of Transport Protocols: Addressing



Elements of Transport Protocols: Connection Establishment

Connection establishment sounds easy, but …

- Naïve approach:
  - One sends a CONNECTION REQUEST TPDU to the other and wait for a CONNECTION ACCEPTED reply.
  - If ok, done; otherwise, retry.
  - Possible nightmare:
    - A user establishes a connection with a bank, sends messages telling the bank to transfer a large amount of money to the account of a not-entirely-trustworthy person, and then releases the connection.
    - Moreover, assume each packet in scenario is ***duplicated*** and stored in the subnet.
  - → ***delayed duplicates***.

Elements of Transport Protocols: Connection Establishment

Possible solutions for solving ***delayed duplicates***

- To give each connection a connection identifier (i.e., a sequence number incremented for each connection established) chosen by the initiating party and put in each TPDU, including the one requesting the connection.
- To use throw away transport addresses. Each time a transport address is needed, a new one is generated. When a connection is released, the address is discarded and never used again.
- To use sequence number and age
  - To limit packet lifetime.
  - To use the sequence number

Elements of Transport Protocols: Connection Establishment

- Packet lifetime can be restricted to a known maximum using one of the following techniques
  - Restricted subnet design.
  - Putting a hop counter in each packet.
  - Timestamping each packet. (router synchronization problem)
- In practice, we will need to guarantee not only that a packet is dead, but also that all acknowledgements to it are also dead, so we will now introduce ***T***, which is some small multiple of the true maximum packet lifetime.

Elements of Transport Protocols: Connection Establishment

To ensure that two identically numbered TPDUs are never outstanding at the same time:

1. To equip each host with a time-of-day clock
   - Each clock is assumed to take the form of a binary counter that increments itself at uniform intervals.
   - The number of bits in the counter must equal or exceed the number of bits in the sequence numbers.
   - The clock is assumed to continue running even if the host goes down.
   - The clocks at different hosts need not be synchronized.

Elements of Transport Protocols: Connection Establishment

To ensure that two identically numbered TPDUs are never outstanding at the same time:

2. When a connection is setup, the low order k bits of the clock are used as the initial sequence number (also k bits). Each connection starts numbering its TPDUs with a different initial sequence number. The sequence space should be so large that by the time sequence numbers wrap around, old TPDUs with the same sequence number are long gone.
3. Once both transport entities have agreed on the initial sequence number, any sliding window protocol can be used for data flow control.

Elements of Transport Protocols: Connection Establishment

A problem occurs when a host crashes. When it comes up again, its transport entity does not know where it was in the sequence space.

- To require the transport entities to be idle for T seconds after a recovery to let all old TPDUs die off. (In a complex internetwork, *T* may be large, so this strategy is unattractive.)
- To avoid requiring *T* sec of dead time after a crash, it is necessary to introduce a new restriction on the use of sequence numbers.

Elements of Transport Protocols: Connection Establishment: The forbidden region

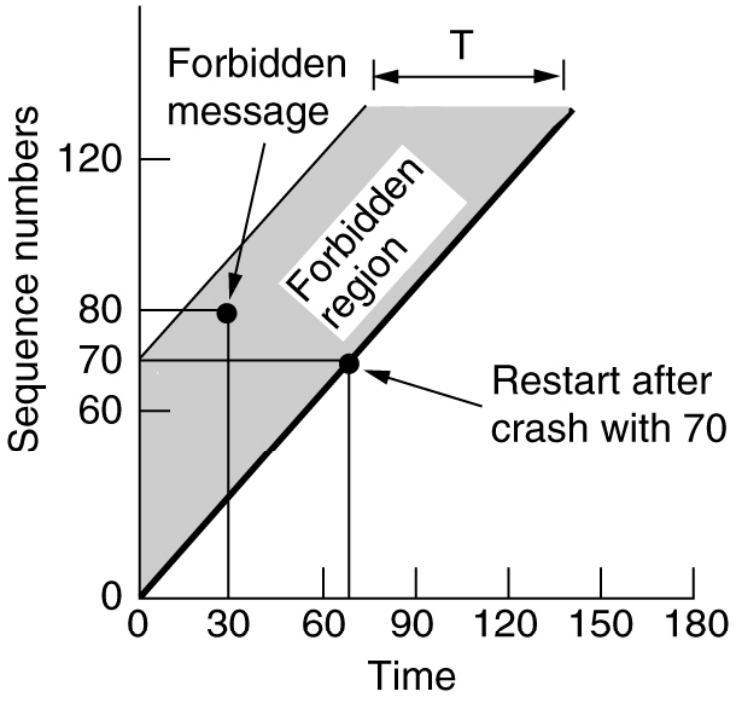
Restriction on the sequence numbers ?

- Let *T* (the maximum packet lifetime) be 60 sec and let the clock tick once per second. The initial sequence number for a connection opened at time *x* will be *x*.
- At *t*=30sec, an ordinary data TPDU being sent on (a previously opened) connection 5 (called as TPDU X) is given sequence number 80.
- After sending TPDU X, the host crashes and then quickly restarts.
- At *t* = 60sec, it begins reopening connections 0 through 4.

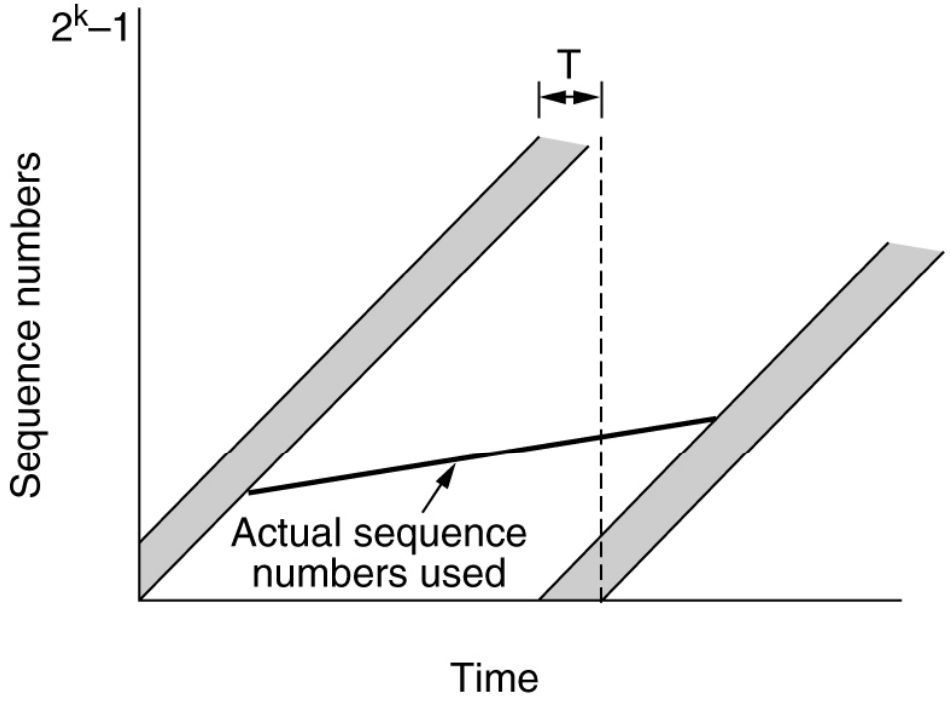
Elements of Transport Protocols: Connection Establishment: The forbidden region

- At t = 70sec, it reopens connection using initial sequence number 70 as required.
- During the next 15 sec it sends data TPDUs 70 through 80. Thus at t=85sec, a new TPDU with sequence number 80 and connection 5 has been injected into the subnet.
- ➔TPDU X and TPDU 80.
- **To prevent sequence numbers from being used for a time *T* before their potential use as initial sequence numbers. The illegal combinations of time and sequence number are called as the forbidden region. Before sending any TPDU on any connection, the transport entity must read the clock and check to see that it is no in the forbidden region.**

Elements of Transport Protocols: Connection Establishment



(a)



(b)

Elements of Transport Protocols: Connection Establishment: The forbidden region

- Too fast
- Too slow
- → **(Solution for the delayed TPDU)**

  Before sending every TPDU,

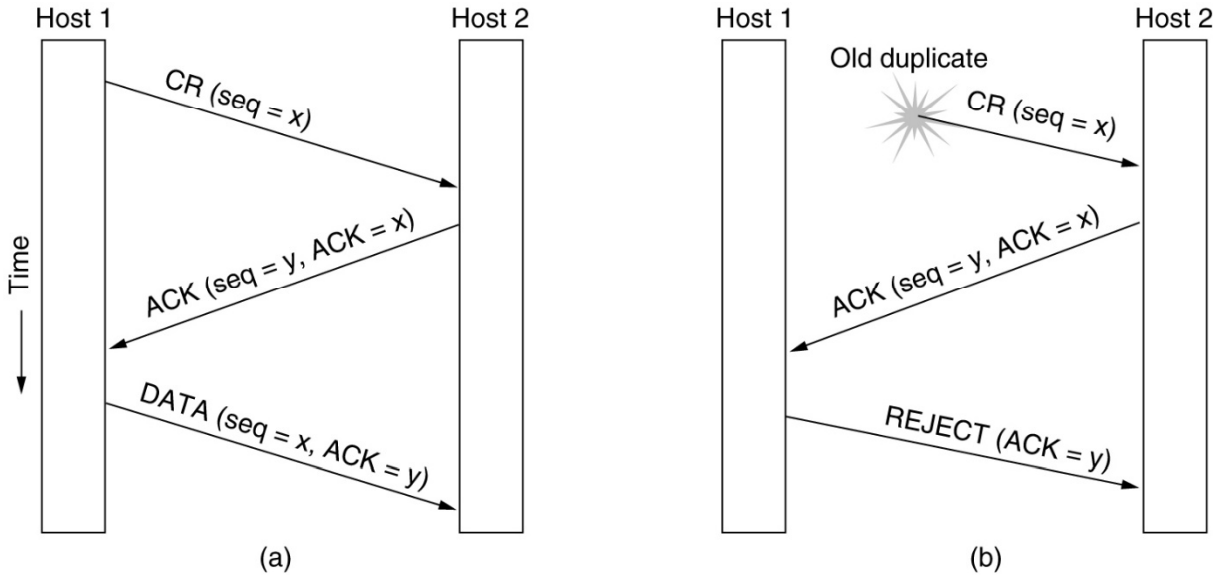
  the transport entity must check to see if it is about to enter the forbidden region,

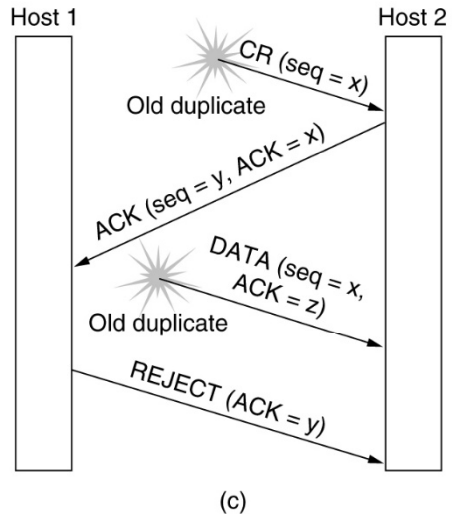
  and if so, either delay the TPDU for T sec or resynchronize the sequence numbers.

Elements of Transport Protocols: Connection Establishment

- Three-way handshake for connection establishment
  - Normal operation.
  - Old duplicate CONNECTION REQUEST appearing out of nowhere.
  - Duplicate CONNECTION REQUEST and duplicate ACK.
- Conclusion: there is no combination of old CONNECTION REQUEST, CONNECTION ACCEPTED, or other TPDUs that can cause the protocol to fail and have a connection setup by accident when no one wants it.

Elements of Transport Protocols: Connection Establishment



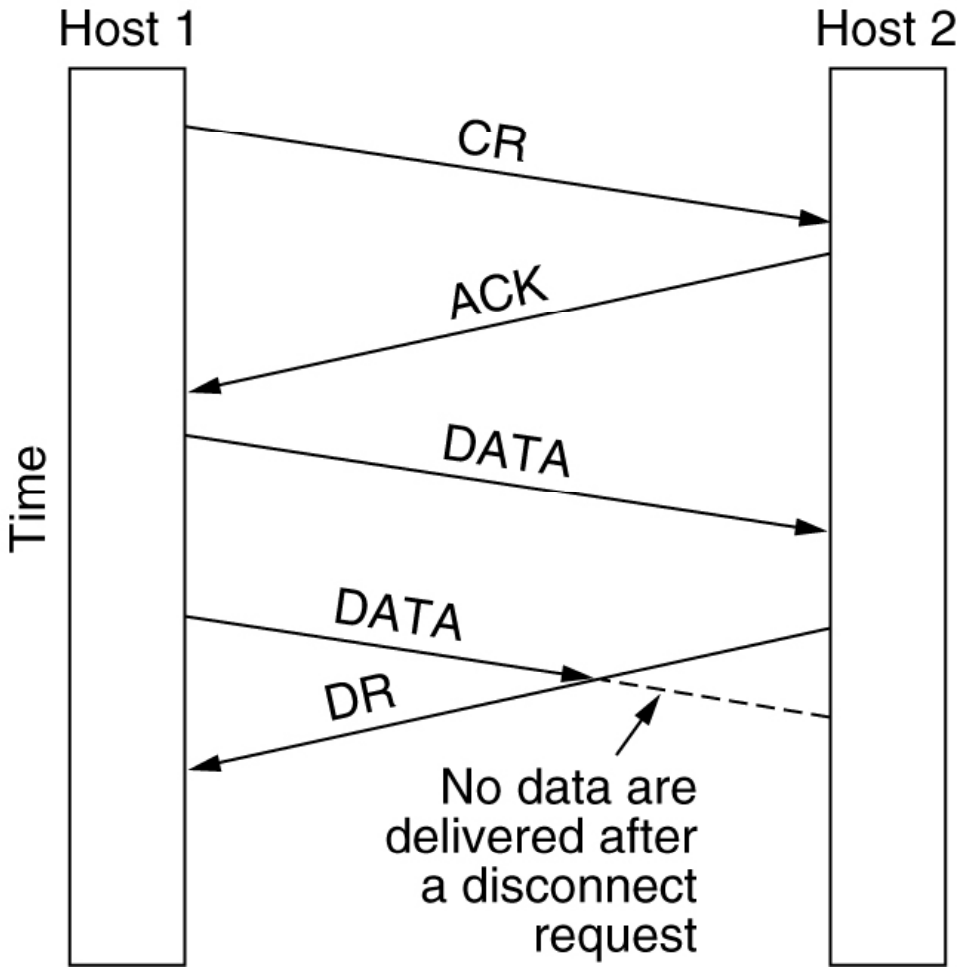


Elements of Transport Protocols: Connection Release

- Asymmetric release and symmetric release
  - Asymmetric release: When one part hangs up, the connection is broken.
  - Symmetric release: to treat the connection as two separate unidirectional connections and require each one to be released separately.
- Asymmetric release is abrupt and may result in data loss (See the next slide)
- One way to avoid data loss is to use symmetric release.

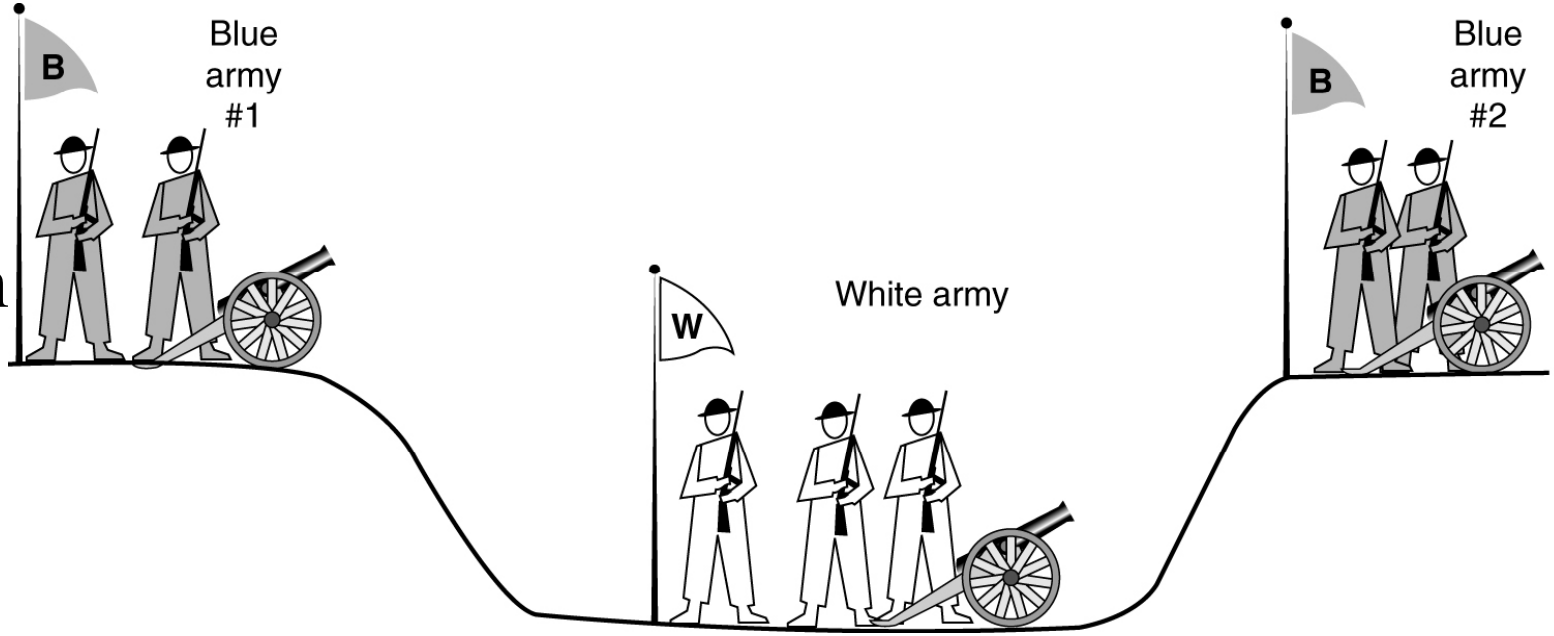
Elements of Transport Protocols: Connection Release

- Abrupt disconnection with loss of data



Elements of Transport Protocols: Connection Release

- Symmetric release does the job when each process has a fixed amount of data to send and clearly knows when it has sent it.
- Symmetric release has its problems if determining that all the work has been done and the connection should be terminated is not so obvious.
- The two army problem



Elements of Transport Protocols: Connection Release

- Two army-problem: A white army is encamped in a valley. On both of the surrounding hillsides are blue armies.

  one blue army < white army < two blue armies

- Does a a protocol exist that allows the blue armies to win?
  - Two-way handshake

    The commander of blue army #1: "I propose we attack at down on March 29. How about it?"

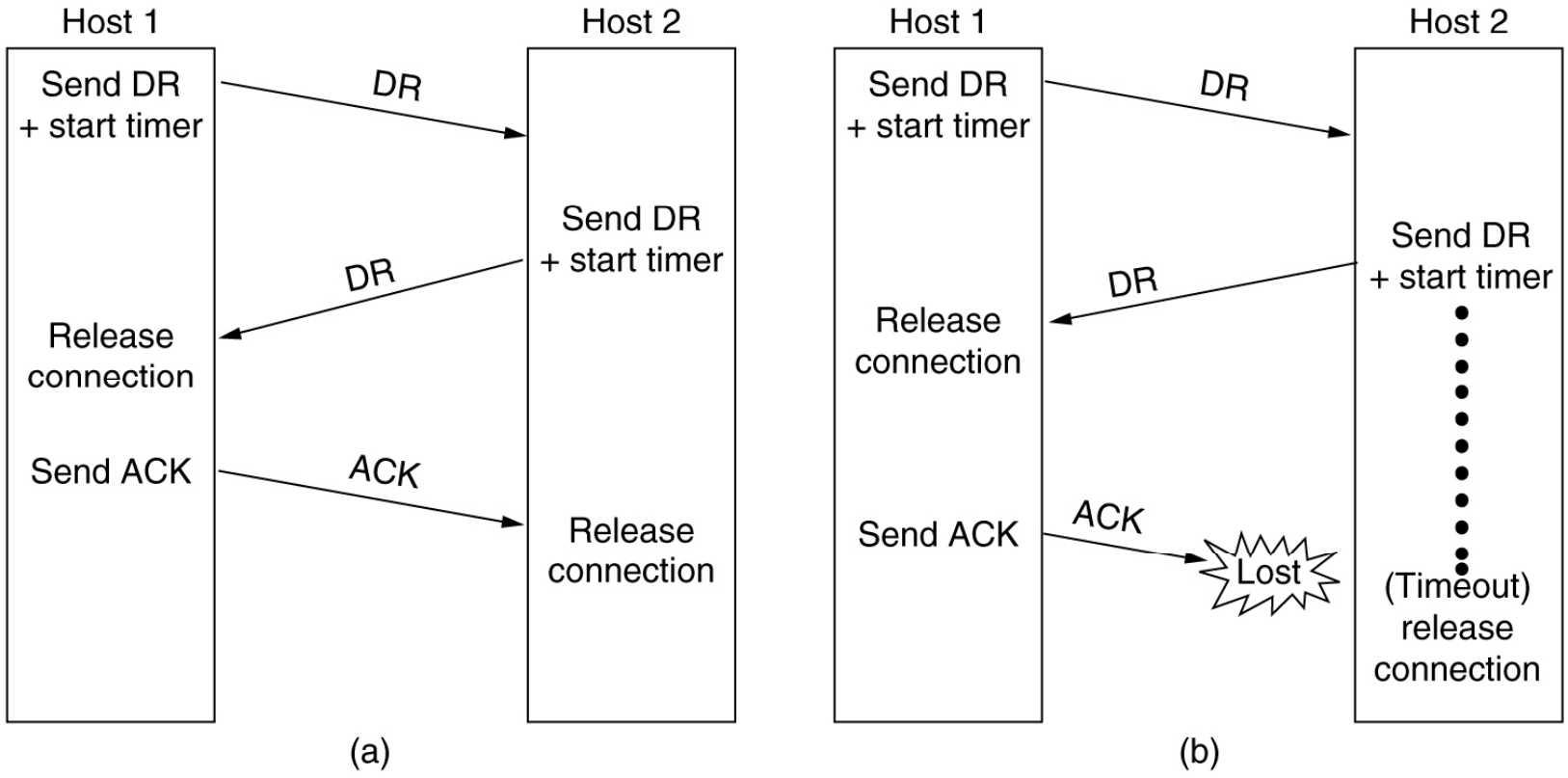
    The commander of blue army #2: "OK."

    → Will the attack happen?
  - Three-way handshake，
  - …
  - N-way handshake
  - ➔No protocol exists that works.
- Substitute "disconnect" for "attack". If neither side is prepared to disconnect until it is convinced that the other side is prepared to disconnect too, the disconnection will never happen.

Elements of Transport Protocols: Connection Release

(a) Normal case of three-way handshake

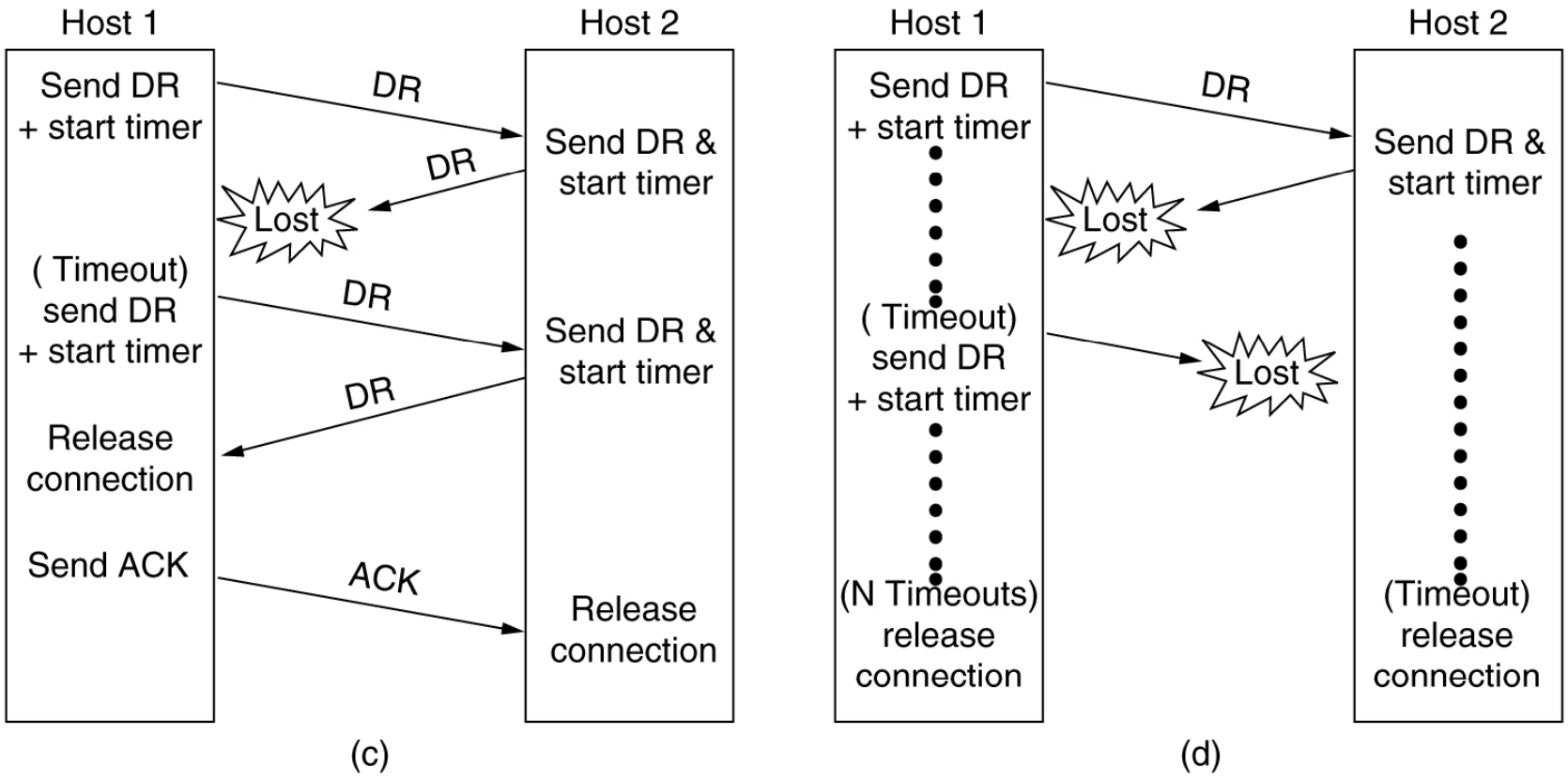
(b) Final ACK lost



Elements of Transport Protocols: Connection Release

(c) Response lost

(d) Response lost and subsequent DRs Lost



Elements of Transport Protocols: Connection Release

- Automatic disconnect rule
  - If no TPDUs have arrived for a certain number of seconds, the connection is then automatically disconnect.
  - Thus, if one side ever disconnects, the other side will detect the lack of activity and also disconnect.
- Conclusion: releasing a connection without data loss is not nearly as simple as it first appears.

Elements of Transport Protocols: Error Control and Flow Control

Error control is ensuring that the data is delivered with the desired level of reliability, usually that all of the data is delivered without any errors.

- Similarity: In both layers, error control has to be performed.
- Difference: The link layer checksum protects a frame while it crosses a single link. The transport layer checksum protects a segment while it crosses an entire network path. It is an end-to-end check, which is not the same as having a check on every link.

Elements of Transport Protocols: Error Control and Flow Control

Flow control in data link layer and transport layer

- Similarity: In both layers a sliding window or other scheme is needed on each connection to keep a fast transmitter from overrunning a slow receiver.
- Difference: A router usually has relative few lines, whereas a host may have numerous connections.

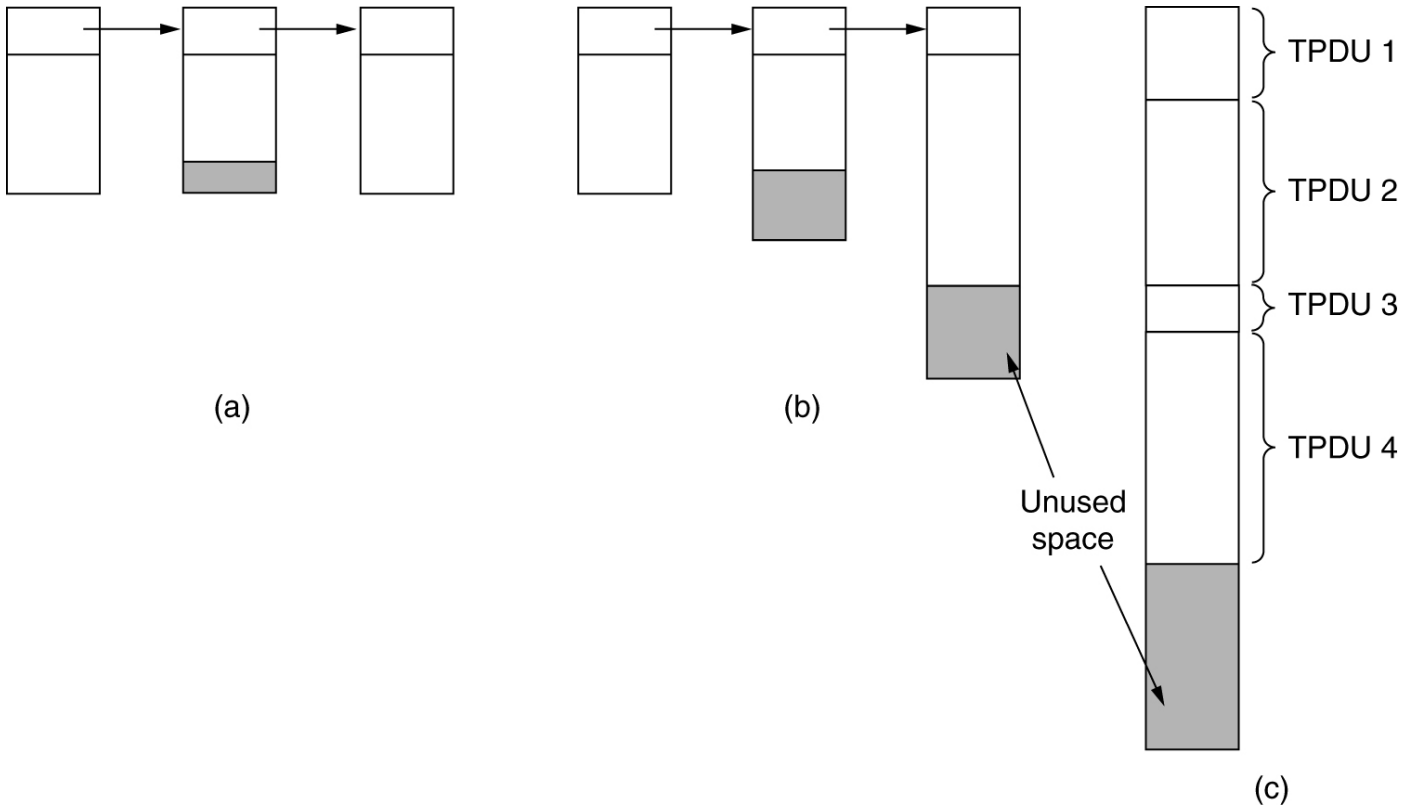
Buffering

- The sender: The sender must buffer all TPDUs sent if the network service is unreliable. The sender must buffer all TPDUs sent if the receiver cannot guarantee that every incoming TPDU will be accepted.
- The receiver: If the receiver has agreed to do the buffering, there still remains the question of the buffer size.

Elements of Transport Protocols: Flow control and buffering: (The receiver buffering)

Buffer sizes

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



Elements of Transport Protocols: Flow control and buffering

- **The optimum trade-off between sender buffering and receiver buffering** depends on the type of traffic carried by the connection.
  - For low-bandwidth bursty traffic, it is better to buffer at the sender,
  - For high-bandwidth smooth traffic, it is better to buffer at the receiver.
  - As connections are opened and closed, and as the traffic pattern changes, the sender and receiver need to dynamically adjust their buffer allocation.
  - Dynamic buffer allocation.

Elements of Transport Protocols: Flow control and buffering

Dynamic buffer allocation (receiver's buffering capacity).

	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

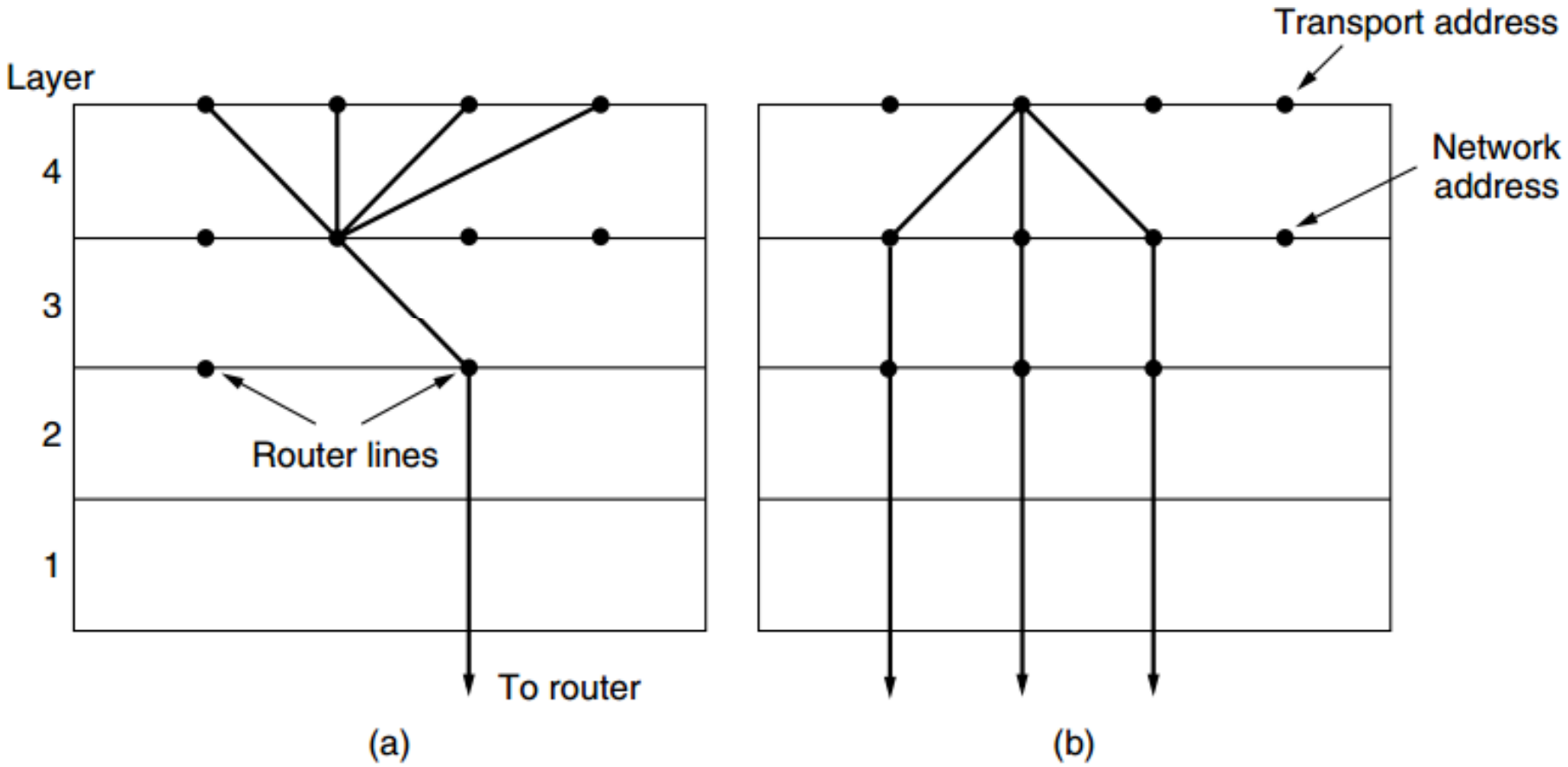
Elements of Transport Protocols: Flow control and buffering

- When buffer space no longer limits the maximum flow, another bottleneck will appear: the carrying capacity of the subnet.
  - The sender dynamically adjusts the window size to match the network's carrying capacity.
  - In order to adjust the window size periodically, the sender could monitor both parameters and then compute the desired window size.

Elements of Transport Protocols: Multiplexing

- Multiplexing and demultiplexing
  - Multiplexing: Application layer ➔Transport layer ➔Network layer ➔Data link layer ➔Physical layer
  - Demultiplexing: Physical layer ➔Data link layer ➔Network layer ➔Transport layer ➔Application layer
- Two multiplexing:
  - Upward multiplexing
  - Downward multiplexing (See the next slide)

Elements of Transport Protocols: Multiplexing



Elements of Transport Protocols: Crash Recovery

- Recovery from network and router crash is straightforward.
- Recovery from host crash is difficult.
- Assume that a client is sending a long file to a file server using a simple stop-and -wait protocol.
  - Part way through the transmission, the server crashes.
  - The server might send a broadcast TPDU to all other hosts, announcing that it had just crashed and requesting that its clients inform it of the status of all open connections.
  - Each client can be in one of two states: one TPDU outstanding, *S1*, or no TPDUs outstanding, *S0*.

Elements of Transport Protocols: Crash Recovery

- Some situations
  - The client should retransmit only if has an unacknowledged TPDU outstanding (S1) when it learns the crash.
    - If a crash occurs after the acknowledgement has been sent but before the write has been done, the client will receive the acknowledgement and thus be in state S0. → LOST. Problem!
    - If a crash occurs after the write has been done but before the acknowledgement has been sent, the client will not receive the acknowledgement and thus be in state S1. → Dup. Problem!
    - For more, see the next slide

Elements of Transport Protocols: Crash Recovery

Strategy used by sending host	Strategy used by receiving host: First ACK, then write — AC(W)	AWC	C(AW)	First write, then ACK — 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

Elements of Transport Protocols: Crash Recovery

- No matter how the sender and receive are programmed, there are always situations where the protocol fails to recover properly.
- **In more general terms, recovery from a layer N crash can only be done by layer N+1 and only if the higher layer retains enough status information.**

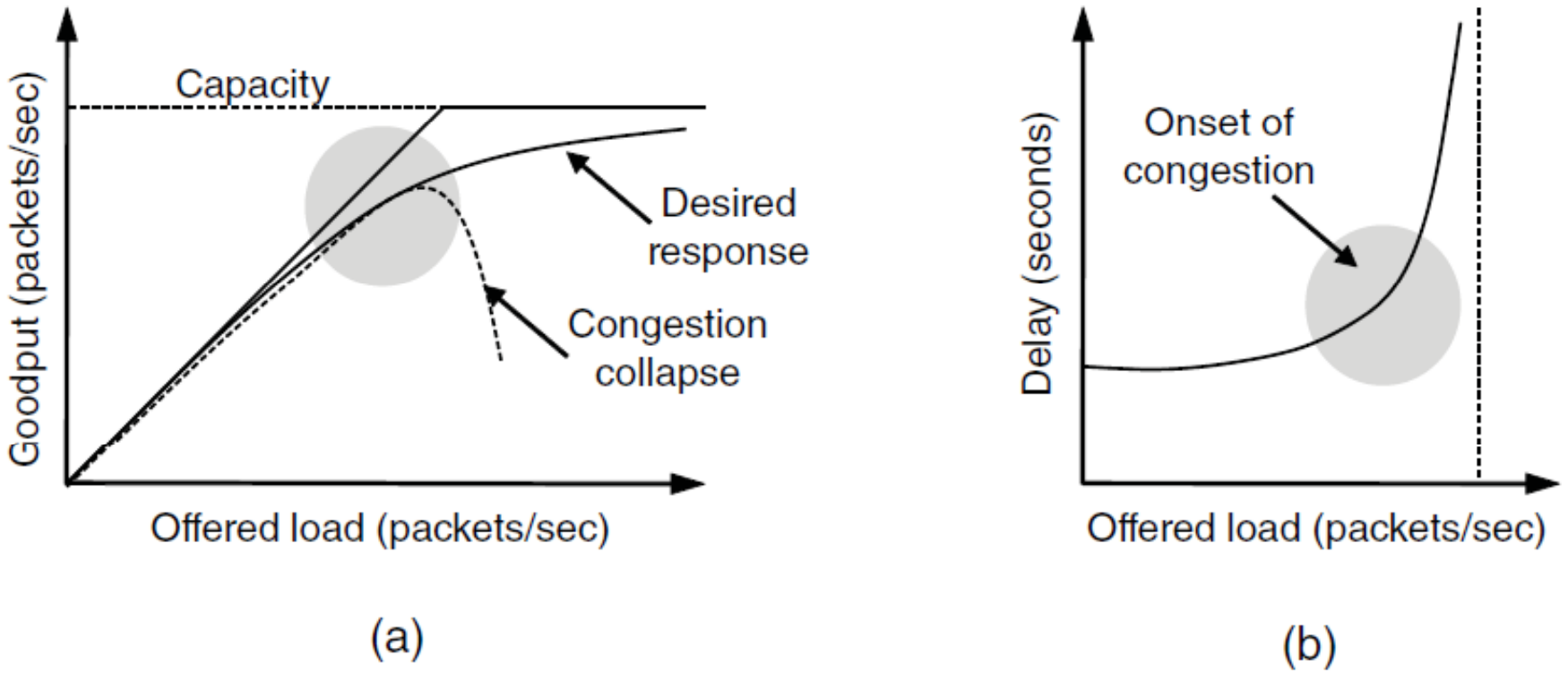
Congestion Control*

- Desirable Bandwidth Allocation
- Regulating the sending rate
- Wireless Issues

Congestion Control:
Desirable Bandwidth Allocation

- Efficiency and Power

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



Congestion Control:
Desirable Bandwidth Allocation

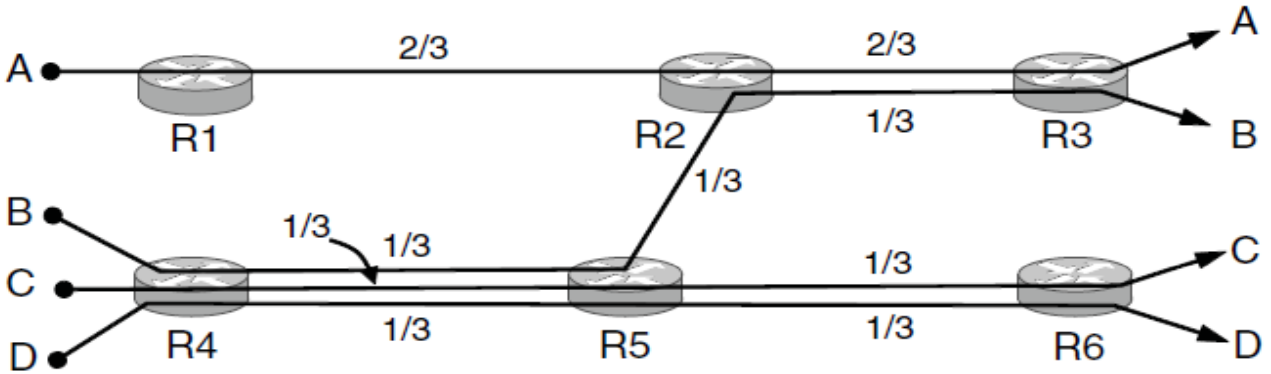
- To analyze the desirable bandwidth allocation, Kleinrock (1979) proposed the metric of power,

$$power = \frac{load}{delay}$$

- Power
  - will initially rise with offered load, as delay remains small and roughly constant,
  - but will reach a maximum and
  - fall as delay grows rapidly.
- The load with the highest power represents an efficient load for the transport entity to place on the network.

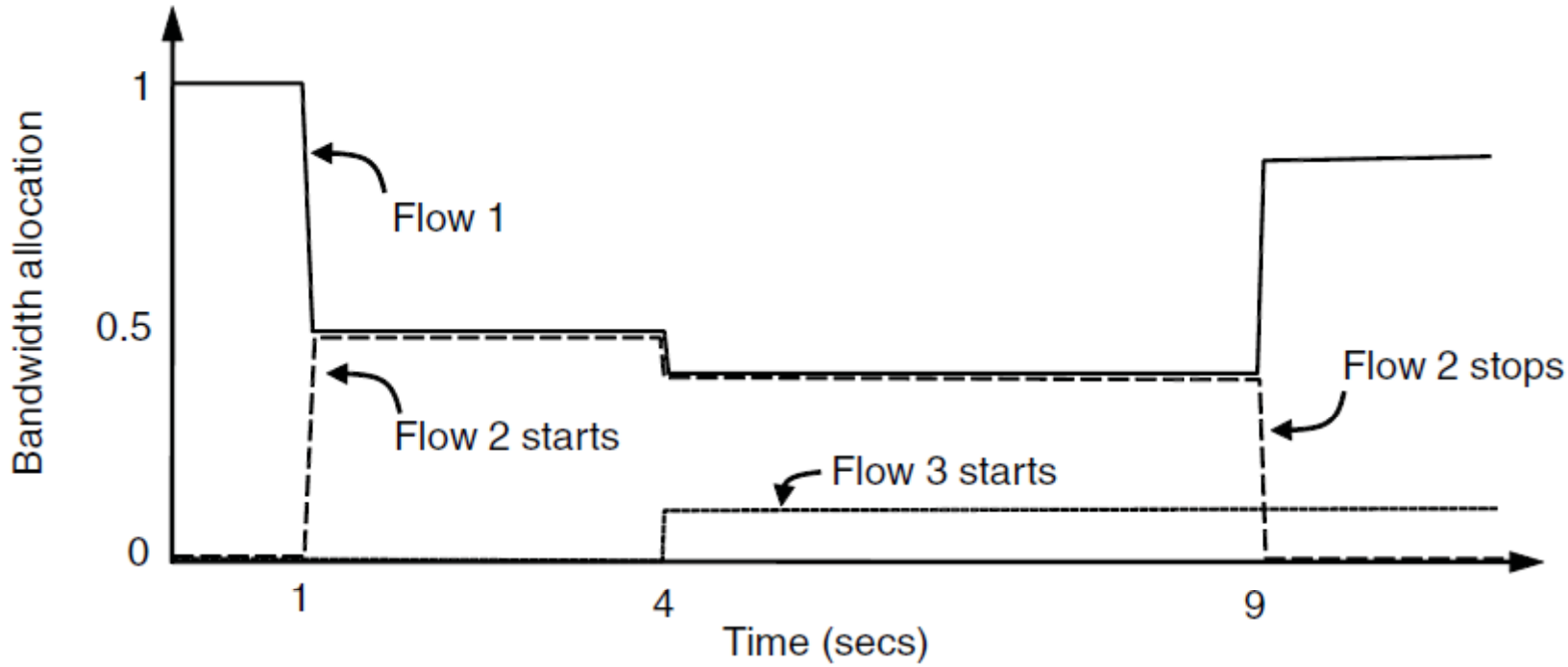
Congestion Control:
Max-Min Fairness

- How to divide bandwidth between different transport senders:
  - The first consideration is to ask what this problem has to do with congestion control.
  - A second consideration is what a fair portion means for flows in a network. The form of fairness that is often desired for network usage is max-min fairness.
  - A third consideration is the level over which to consider fairness.



Congestion Control:
Desirable Bandwidth Allocation

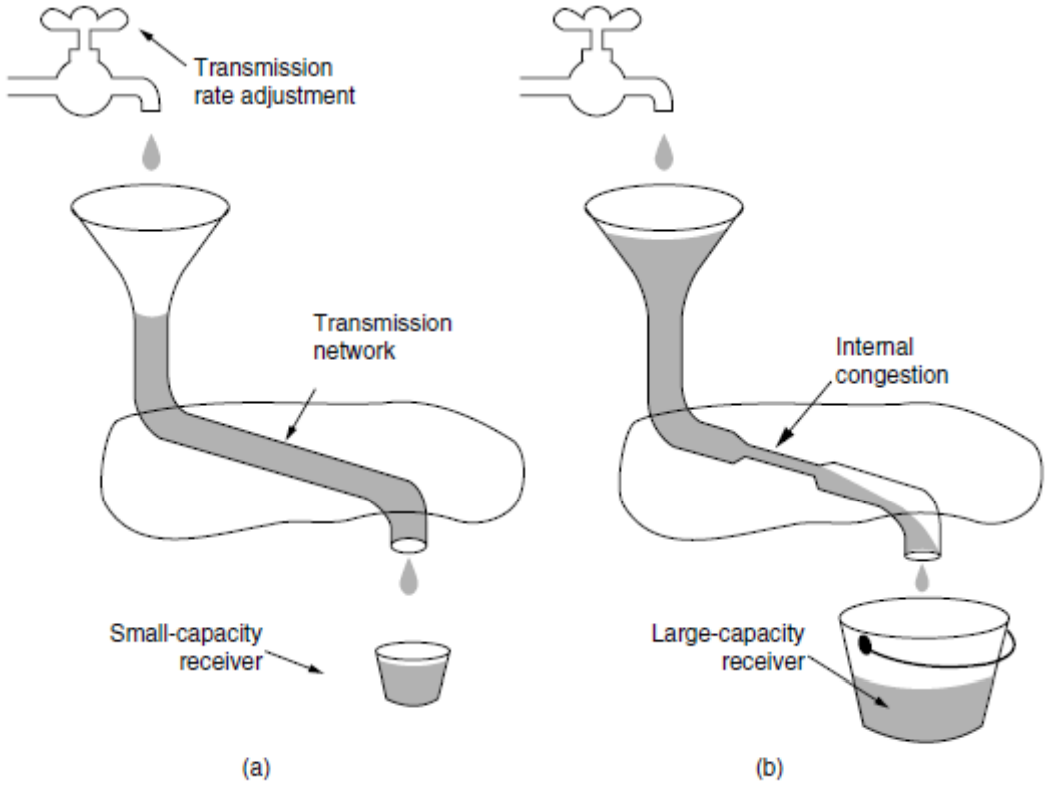
- Convergence: A final criterion is that the congestion control algorithm converge quickly to a fair and efficient allocation of bandwidth.



Congestion Control:
Regulating the sending rate

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

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

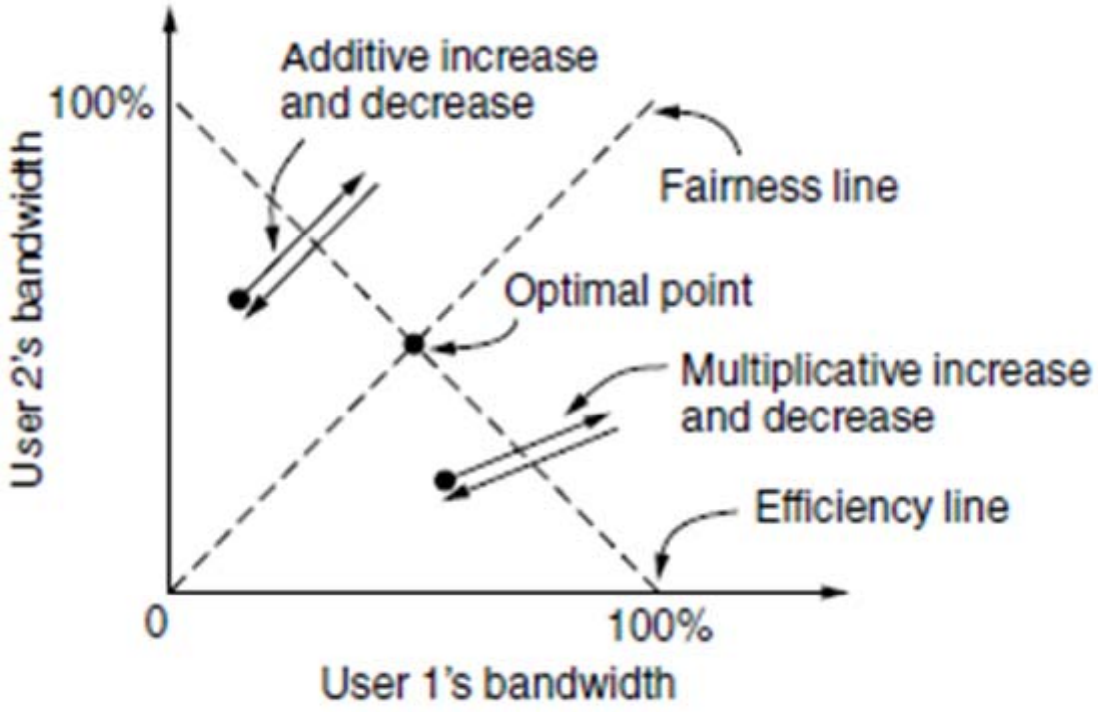


Congestion Control:
Regulating the sending rate

- Signals of some congestion control protocol

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
Compound TCP	Packet loss & end-to-end delay	No	Yes
CUBIC TCP	Packet loss	No	No
TCP	Packet loss	No	No

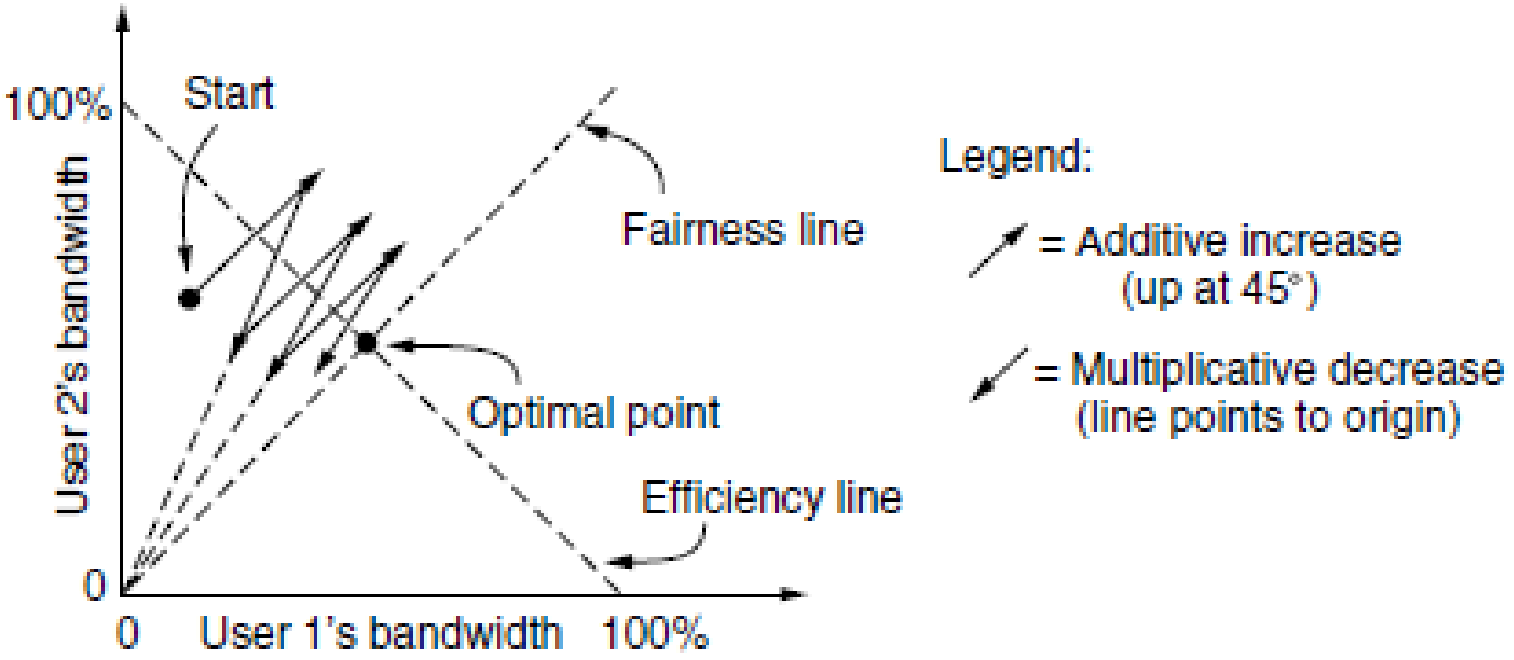
Congestion Control:
Regulating the sending rate



Additive and multiplicative bandwidth adjustments.

Congestion Control:
Regulating the sending rate

- Used in TCP



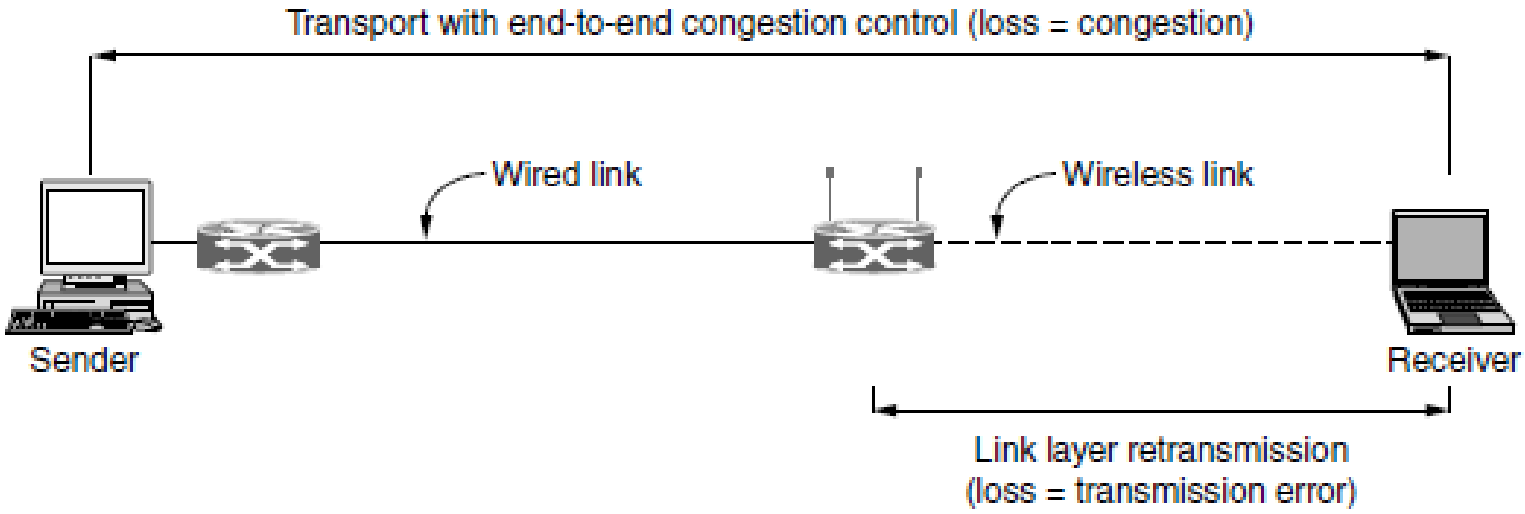
Additive Increase Multiplicative Decrease (AIMD) control law.

Congestion Control:
Wireless Issues

- Analyses by Padhye et al. (1998) show that the throughput goes up as the inverse square-root of the packet loss rate.
- What this means in practice is that the loss rate for fast TCP connections is very small; 1% is a moderate loss rate, and by the time the loss rate reaches 10% the connection has effectively stopped working.
- However, for wireless networks such as 802.11 LANs, frame loss rates of at least 10% are common.
- This difference means that, absent protective measures, congestion control schemes that use packet loss as a signal will unnecessarily throttle connections that run over wireless links to very low rates.

Congestion Control: Wireless Issues

- There are two aspects to note. First, the sender does not necessarily know that the path includes a wireless link, since all it sees is the wired link to which it is attached.
- The second aspect is a puzzle. The figure shows two mechanisms that are driven by loss: link layer frame retransmissions, and transport layer congestion control. TIMESCALE



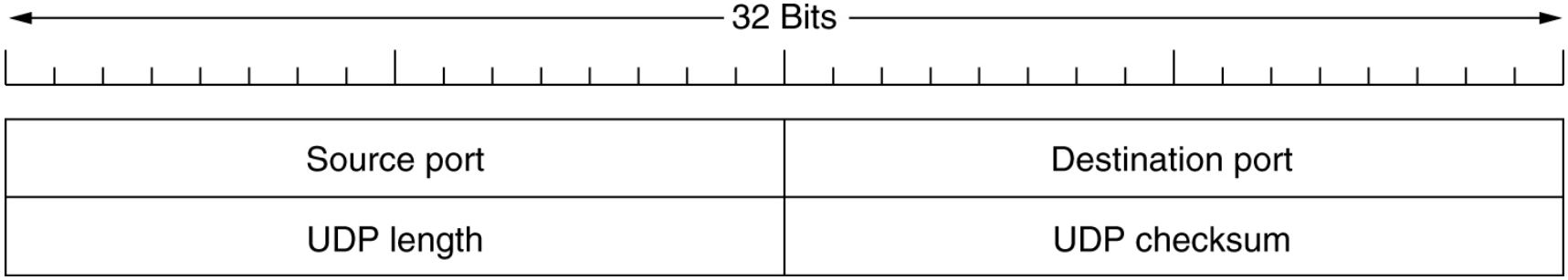
Congestion control over a path with a wireless link.

THE INTERNET TRANSPORT PROTOCOLS: UDP, RPC, RTP

- Two main transport protocols in the Internet
  - Connectionless protocol (UDP)
  - Connection-oriented protocol (TCP)
- UDP (User Datagram protocol)
- RPC (Remote Procedure Call)
- RTP (Real-time Transport Protocol)

The Internet Transport Protocols: UDP

- UDP (RFC768) provides a way for applications to
  - send encapsulated IP datagrams and
  - send them without having to establish a connection.
- UDP transmits segments consisting of an 8-byte header followed by the payload.



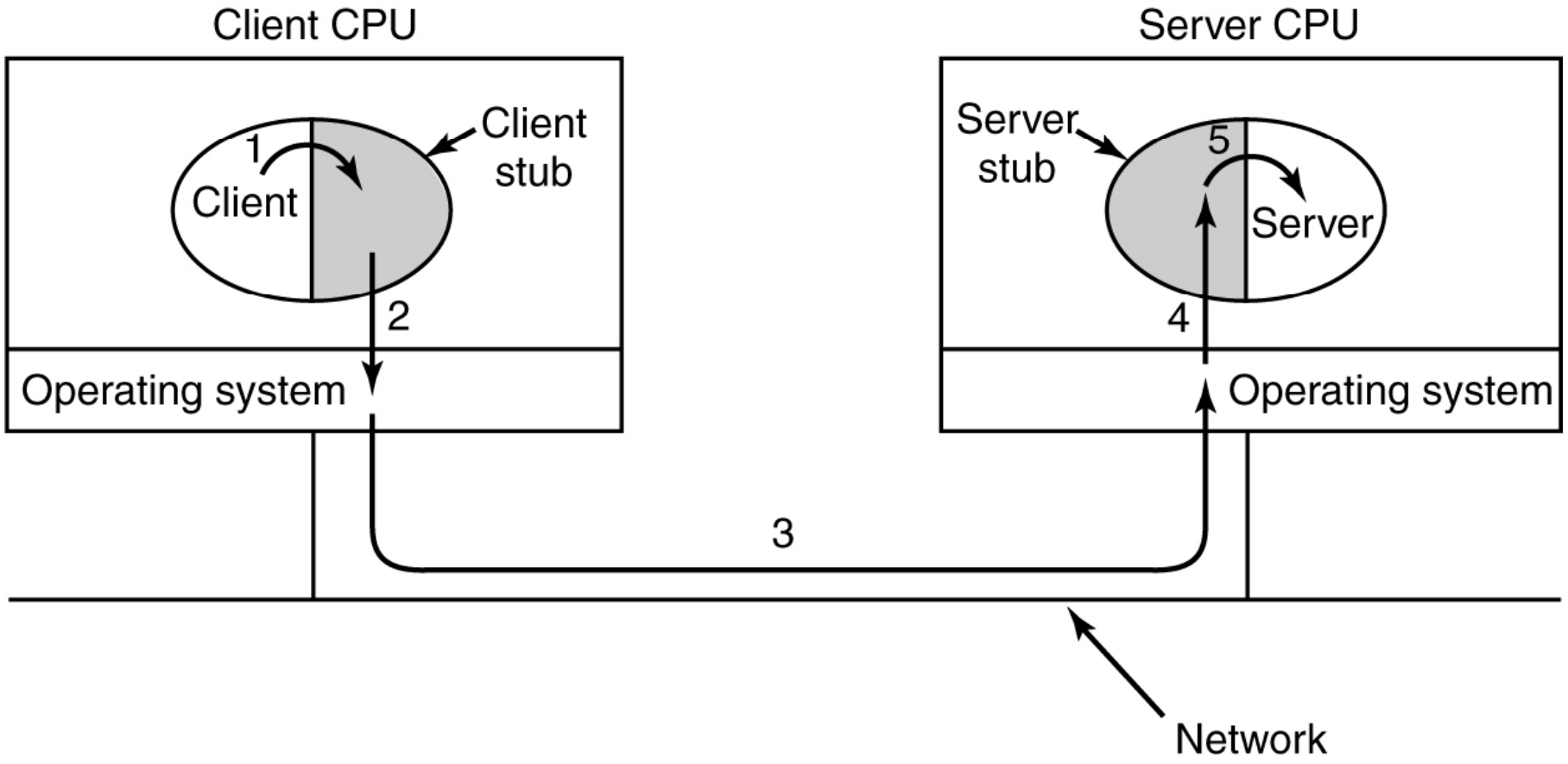
- UDP
  - Multiplexing and demultiplexing using ports.
  - No flow control, error control or retransmission.
  - Applications: RPC, RTP, DNS (Domain Name System)

The Internet Transport Protocols: RPC/UDP

- RPC (Remote Procedure Call) allows programs to call procedures located on remote hosts.
  - When a process on machine 1 calls a procedure on machine 2, the calling process on 1 is suspended and execution of the called procedure takes place on 2
  - Information can be transported from the caller to the callee in the parameters and can come back in the procedure result.
  - No message passing is visible to the programmer.
- **The idea behind RPC is to make a remote procedure call look as much as possible like a local one.**

C Proc ←→ C Stub ←→ S Stub ←→ S Proc

The Internet Transport Protocols: RPC/UDP



The Internet Transport Protocols: RPC/UDP

The steps in making an RPC

- Step 1 is the client calling the client stub.
- Step 2 is the client stub packing the parameters into a message (marshaling) and making a system call to send the message.
- Step 3 is the kernel sending the message from the client machine to the server machine.
- Step 4 is the kernel passing the incoming packet to the server stub and unpacking the packet to extract the parameters (unmarshaling).
- Step 5 is the server stub calling the server procedure with the unmarshaled parameters.

- The reply traces the same path in the other direction.

The Internet Transport Protocols: RPC/UDP

- A few snakes hiding under the grass (RPC)
  - The use of pointer parameters.
  - Some problems for weakly-typed languages (The length of an array).
  - It is not always possible to deduce the types of the parameters, not even from a formal specification or the code itself. (printf)
  - The use of global variables.
- **→Some restrictions are needed to make RPC work well in practice.**
- **RPC/TCP vs RPC/UDP**

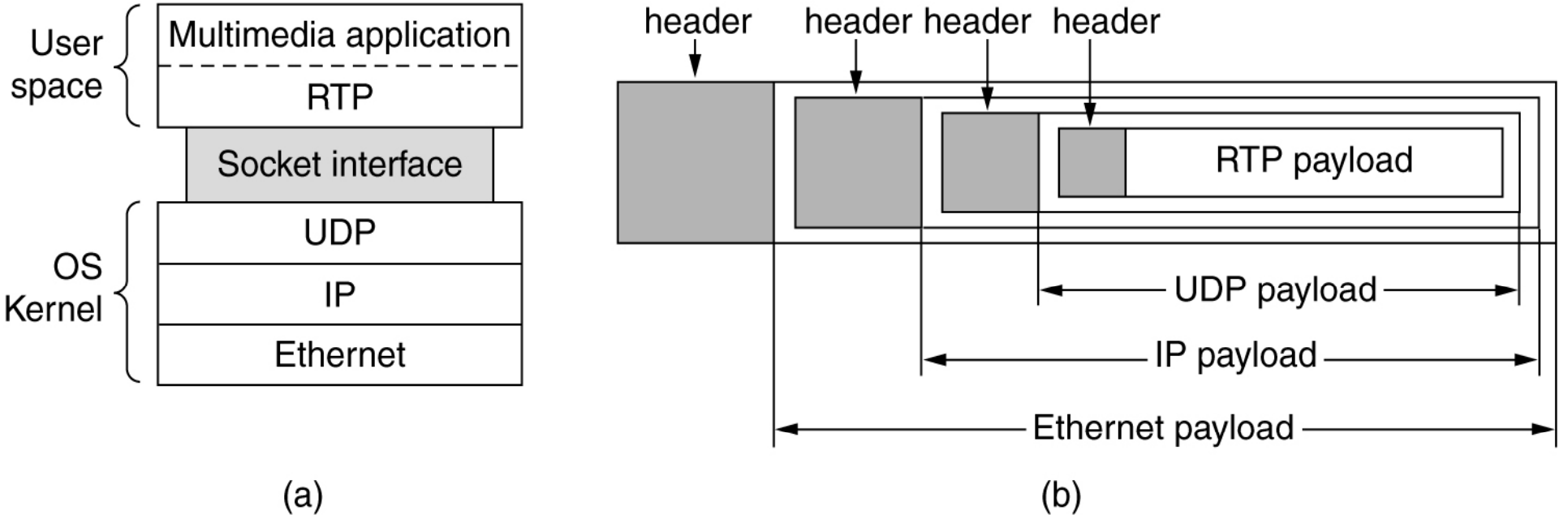
The Internet Transport Protocols: RTP/UDP

- Multimedia applications such as Internet radio, Internet telephony, music-on-demand, videoconferencing, video-on-demand, require real-time transport protocols. → RTP (Real-time Transport Protocol) (RFC1889)
- The RTP is in user space and runs over UDP. The RTP Ops:
  - The multimedia applications consists of multiple audio, video, text, and possibly other streams. These are fed into the RTP library.
  - This library then multiplexes the streams and encodes them in RTP packets, which it then puts into a socket.
  - UDP packets are generated and embedded in IP packets.
  - The IP packet are then put in frames for transmission.
  - …

The Internet Transport Protocols: RTP/UDP

(a)The position of RTP in the protocol stack

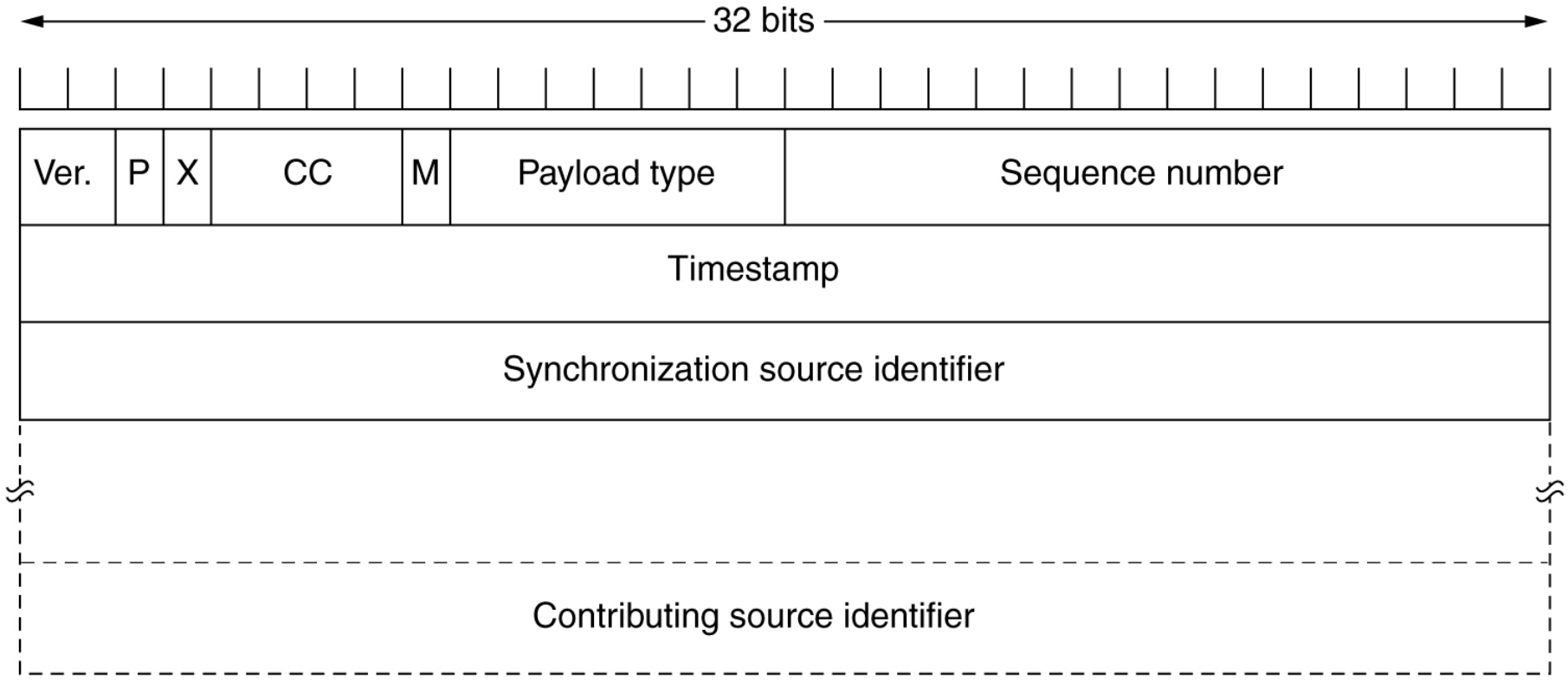
(b) packet nesting



The Internet Transport Protocols: RTP/UDP

- RTP is a transport protocol realized in the application layer.
- RTP is to multiplex several real-time data streams onto a single stream of UDP packets and unicast or multicast the UDP packets.
- Each RTP packet is given a number one higher than its predecessor. RTP has no flow control, no error control, no acknowledgements, and no retransmission support.
- Each RTP payload may contain multiple samples and they can be coded any way that the application wants. For example a single audio stream may be encoded as 8-bit PCM samples at 8kHz, delta encoding, predictive encoding, GSM encoding, MP3, and so on.
- RTP allows timestamping.

The Internet Transport Protocols: RTP/UDP



The Internet Transport Protocols: RTP/UDP

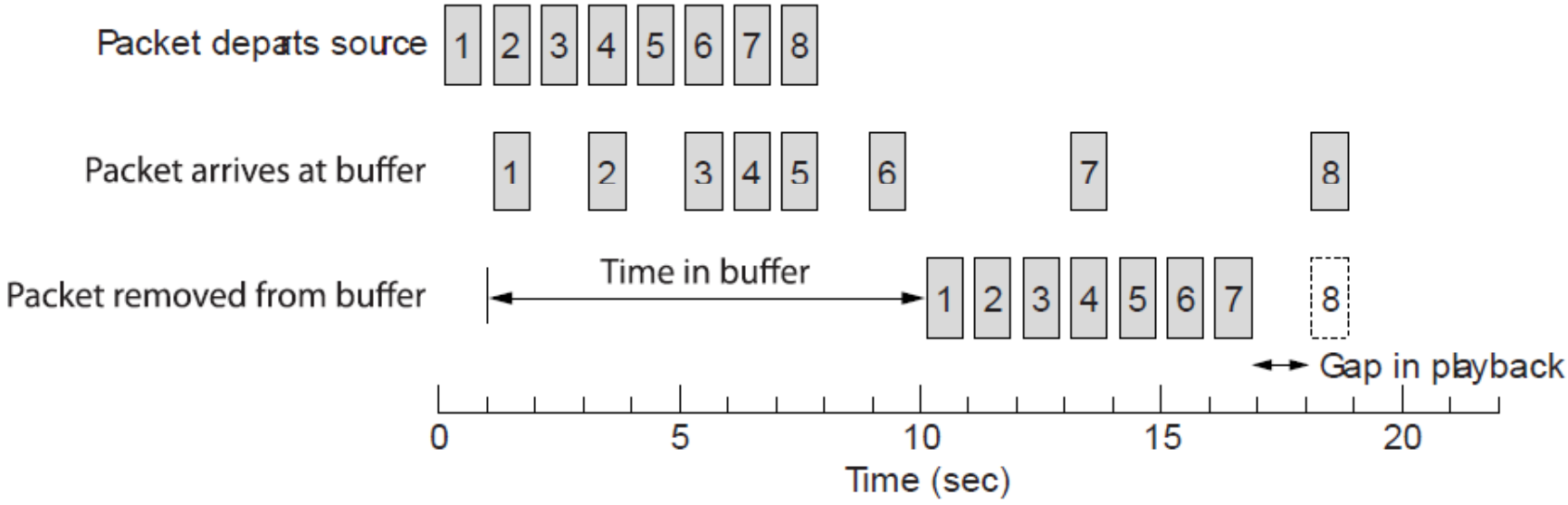
- Version: 2 bits, already at 2.
- P bit: padded to a multiple of 4 bytes or not.
- X bit: an extension header or not.
- CC: how many contributing sources are present (0-15).
- M bit: marker bit.
- Payload type: which encoding algorithm has been used.
- Sequence number: incremented on each RTP packet sent.
- Timestamp: reducing jitter.
- Synchronization source identifier: which stream the packet belongs to.
- Contributing source identifiers.

The Internet Transport Protocols: RTP/UDP

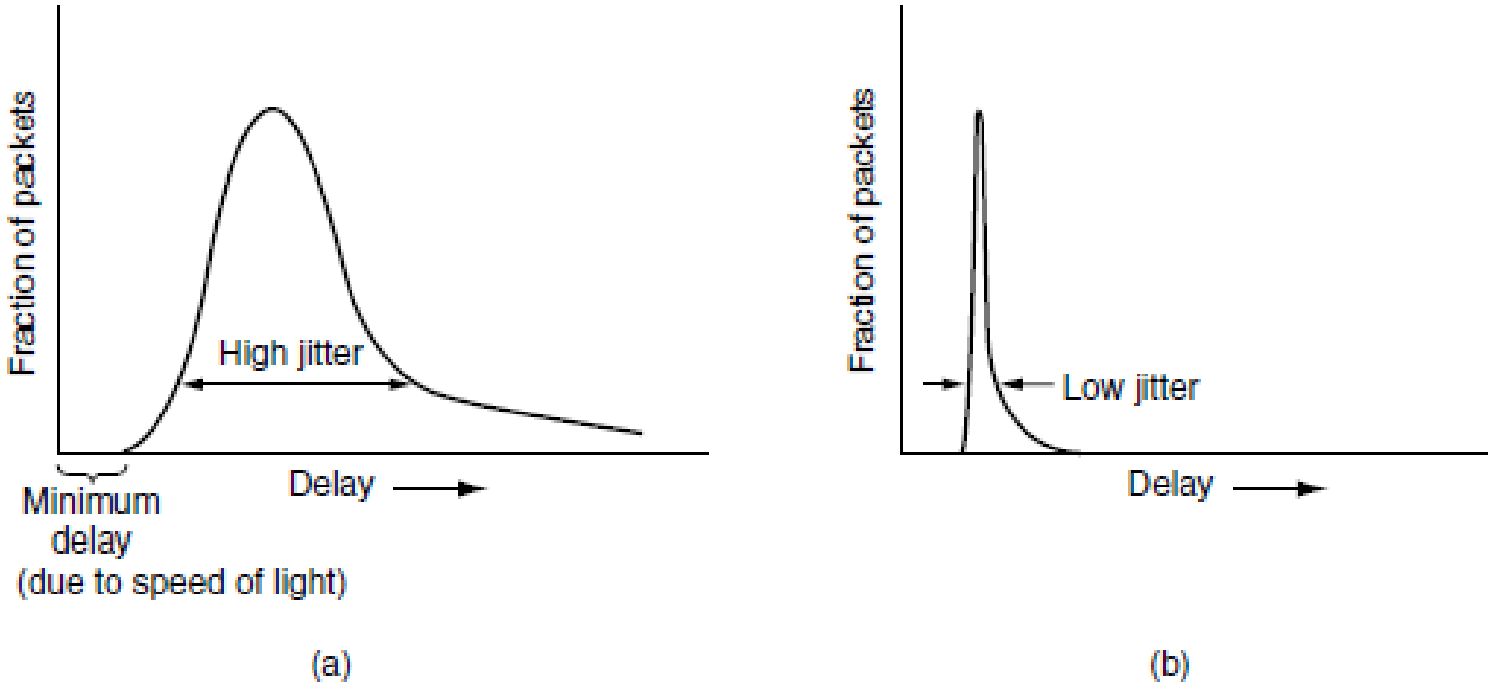
- RTCP (Real-time Transport Control Protocol) is a little sister protocol (little sibling protocol?) for RTP.
  - Does not transport any data
  - To handle feedback, synchronization, and the user interface
  - To handle interstream synchronization.
  - To name the various sources.
- For more information about RTP, *RTP: Audio and Video for the Internet* (Perkins, C.E. 2002, Addison-Wesley)

The Internet Transport Protocols: RTP/UDP

Smoothing the output stream by buffering packets



The Internet Transport Protocols: RTP/UDP



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

THE INTERNET TRANSPORT PROTOCOLS: TCP

- Introduction to TCP
- The TCP Service Model
- The TCP Protocol
- The TCP Segment Header
- TCP Connection Establishment
- TCP Connection Release
- TCP Connection Management Modeling
- TCP Sliding Window
- TCP Timer Management
- TCP Congestion Control
- The Future of TCP

TCP: Introduction

- TCP (Transmission Control Protocol) provides a reliable end-to-end byte stream over an unreliable internetwork. For TCP, see RFC 793, 1122, 1323, 2108,2581, 2873, 2988, 3168, 4614.
- The communication between TCP entities
  - A TCP entity accepts user data streams from local processes, breaks them up into pieces not exceeding 64KB (in practice, often 1500 − 20 − 20 data bytes), sends each piece as a separate IP datagram.
  - When datagrams containing TCP data arrive at a machine, they are given to the TCP entity, which constructs the original byte streams.
- TCP must furnish the reliability that most users want and that IP does not provide.

TCP: The Service Model

- For any TCP service to be obtained, a connection must be explicitly established between a socket on the sending machine and a socket on the receiving machine.
  - Connections are identified by the socket identifiers at both ends, that is, (socket1, socket2)
  - A socket number (address) consisting of the IP address of the host and a 16-bit number local to that host, called a port.
  - Port numbers below 1024 are called well-known ports and are reserved for standard services. (see the next slide.)

TCP: The Service Model

- Some well-known ports
  - 23 for TELNET
  - 25 for SMTP
  - 69 for TFTP
  - 79 for Finger
  - 119 for NNTP

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

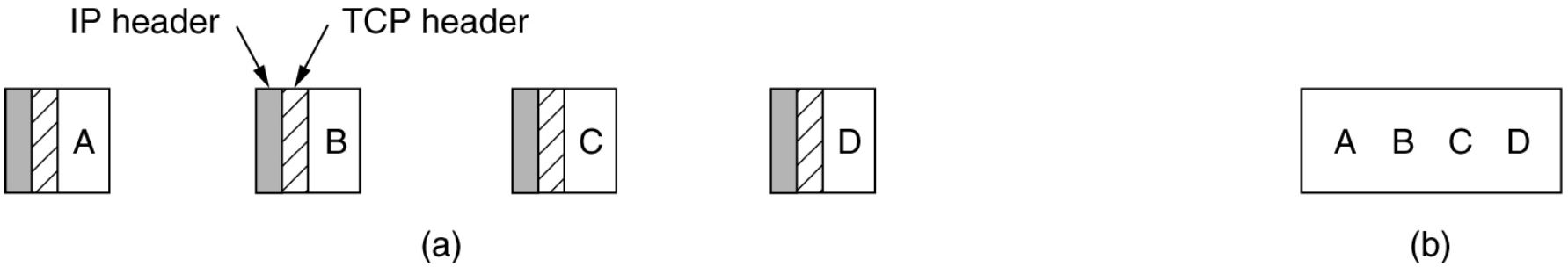
TCP: The Service Model

- To have many daemons standby
- To have one master daemon inetd or xinetd standby
  - The master daemon attaches itself to multiple ports and wait for the first incoming connection
  - When one incoming connection request arrives, inetd or xinetd forks off a new process and executes the appropriate daemon on it, letting that daemon handle the request.

- All TCP connections are full duplex and point-point.
- A TCP connection is a byte stream, not a message queue. Message boundaries are not preserved end to end.

TCP: The Service Model

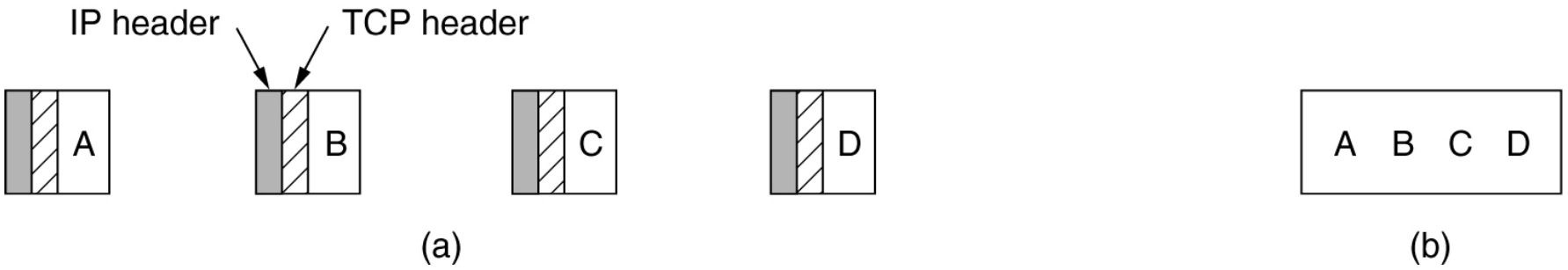
- All TCP connections are full duplex and point-point.
- A TCP connection is a byte stream, not a message queue. Message boundaries are not preserved end to end.



(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.

TCP: The 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.

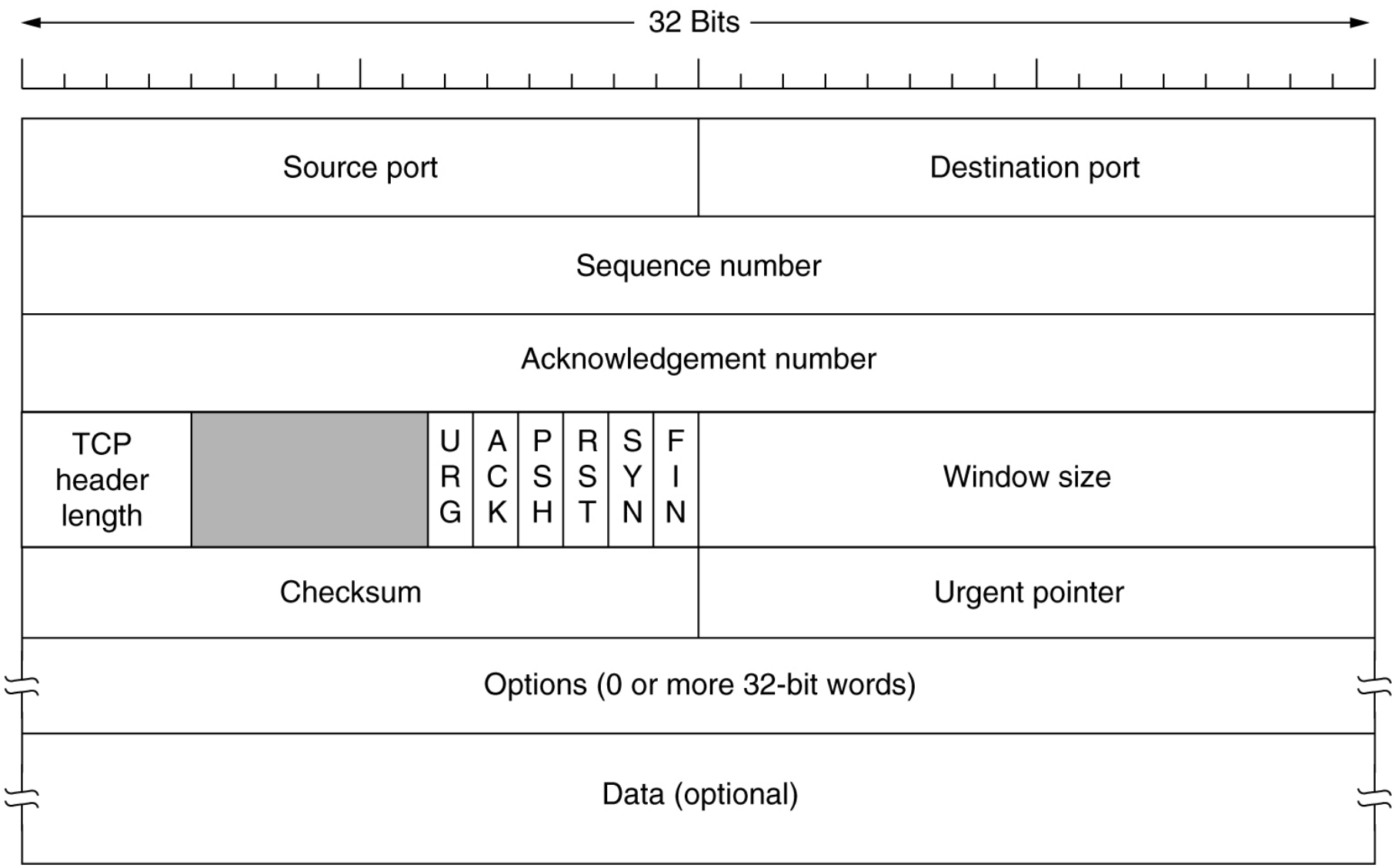
TCP: The Service Model

- When an application passes data to TCP, TCP may send it immediately or buffer it at its own discretion.
  - To force data out, applications can use the PUSH flag, which tells TCP not to delay the transmission.
- TCP supports urgent data (now rarely used).
  - When the urgent data are received at the destination, the receiving application is interrupted (e.g, given a signal in UNIX terms) so it can stop whatever it was doing and read the data stream to find the urgent data.
    - The end of the urgent data is marked so the application know when it is over.
    - The start of the urgent data is not marked. It is up to the application to figure that out.

TCP: The Overview

- Every byte on a TCP connection has its own 32-bit sequence number.
- The sending and receiving TCP entities exchange data in the form of segments.
  - Each segment, including the TCP header, must fit in the 65515 (=65535-20) byte IP payload.
  - Each network has a maximum transfer unit or MTU (1500 for the Ethernet).
- The TCP entities use the sliding window protocol
  - Segments can arrive out of order.
  - Segments can be delayed.
  - The retransmissions may include different byte ranges than the original transmission.

TCP: The Header



TCP: The Header

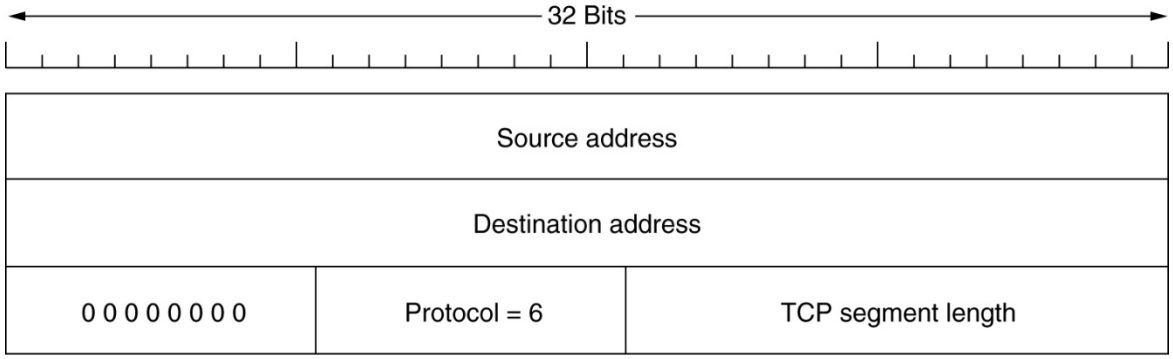
- Source port and destination port: to identify the local end points of the connection. A port plus its host's IP address forms a 48-bit unique end point. The source and destination end points together identify the connection.
- Sequence number and acknowledgement number: 32 bits long (every byte of data is numbered in a TCP stream).
- TCP header length: how many 32-bit words are contained in the TCP header.
- 4-bit field not used.
- CWR: Congestion Window Reduced from a sender
- ECE: ECN (Explicit Congestion Notification)-Echo to a sender

TCP: The Header

- URG bit: the Urgent pointer is in use or not.
- ACK bit: the Acknowledgement number is valid or not.
- PSH bit: PUSHed data or not.
- RST bit: to reset a connection that has become confused due to a host crash or some other reason.
- SYN bit: to used to establish connections.
  - SYN for CONNECTION REQUEST,
  - SYN+ACK for CONNECTION ACCEPTED.
- FIN bit: used to release a connection.
- Window size: to tell how many bytes may be sent starting at the byte acknowledged.

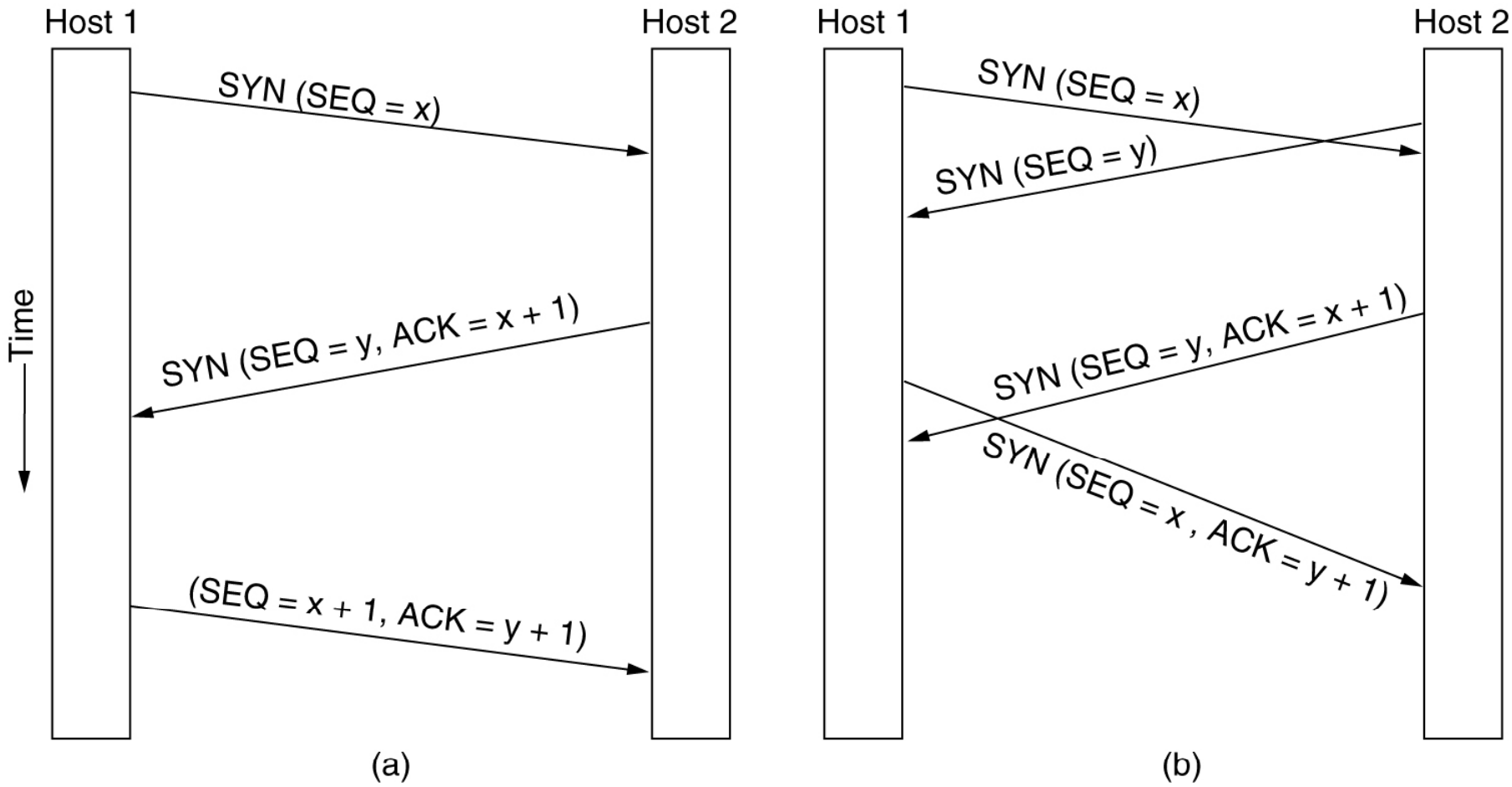
TCP: The Header

- Checksum: provided for extra reliability.



- Urgent pointer: used to indicate a byte offset from the current sequence number at which urgent data are to be found.
- Options:
  - To allow each host to specify the maximum TCP payload it is willing to accept
  - Window scale option
  - To use selective repeat instead of go back n protocol

TCP: Connection Establishment



TCP: Connection Release

- TCP connections are full duplex and can be treated as a pair of simplex connections.
- Each simplex connection is released independently of its sibling.
  - To release a connection, a party can send a TCP segment with the FIN bit set, which means that it has no more data to transmit.
  - When the FIN is acknowledged, that direction is shut down for the new data.
  - When both directions have been shut down, the connection is released.
- To avoid the two-army problem, timers are used.

TCP: Connection Management Policy

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

TCP: Connection Management Policy

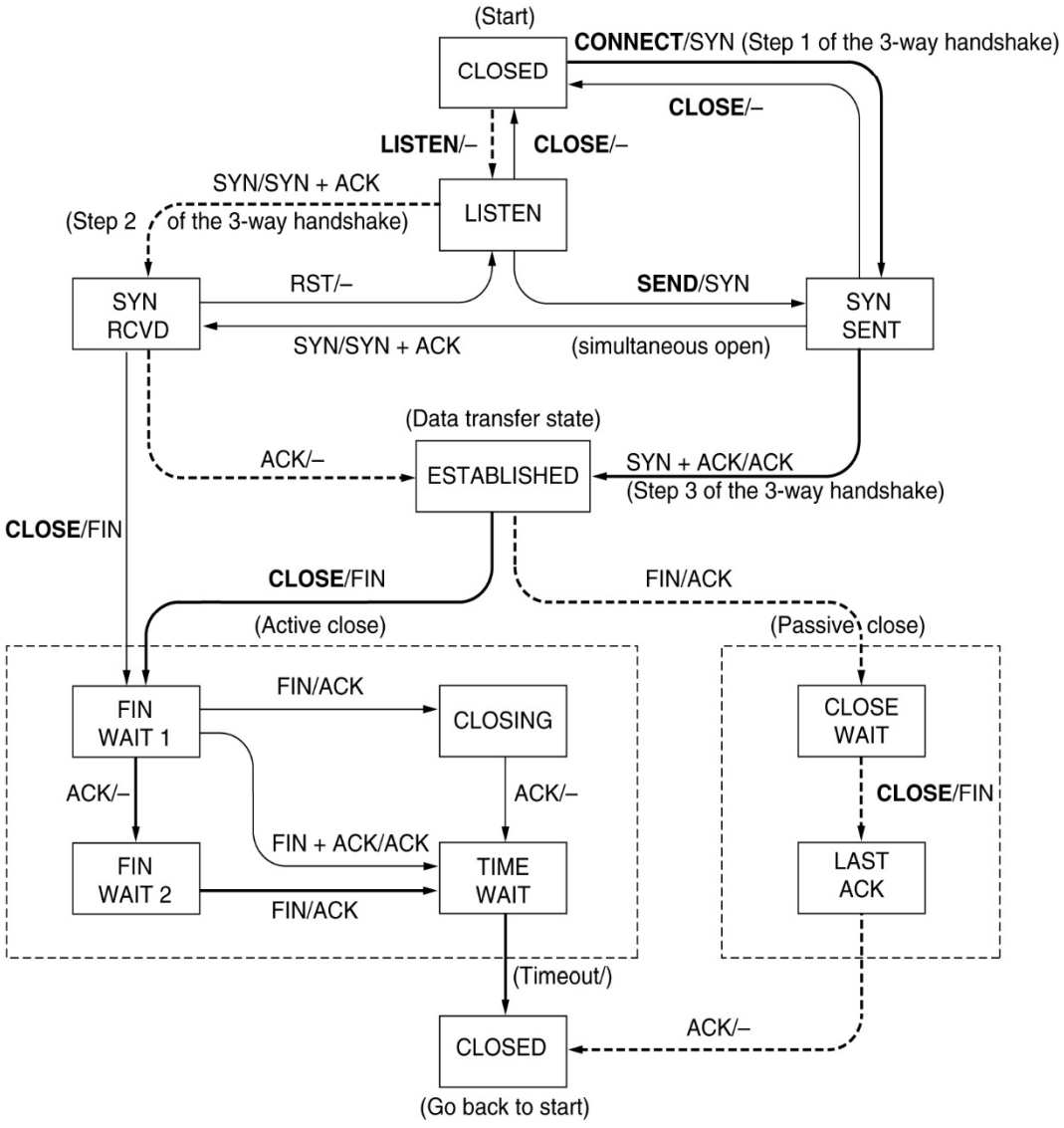
TCP connection management finite state machine (FSM).

The heavy solid line is the normal path for a client.

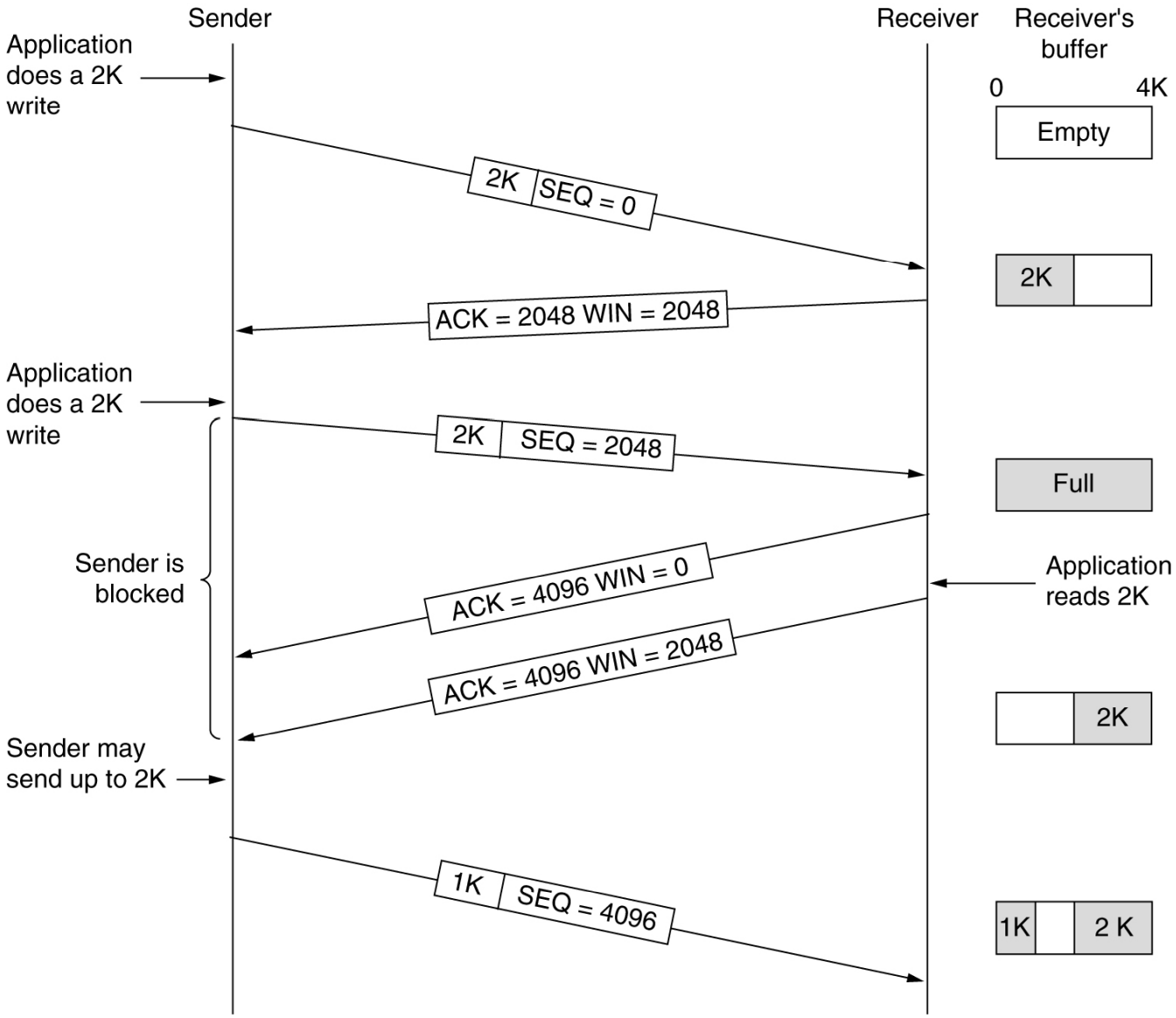
The heavy dashed line is the normal path for a server.

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

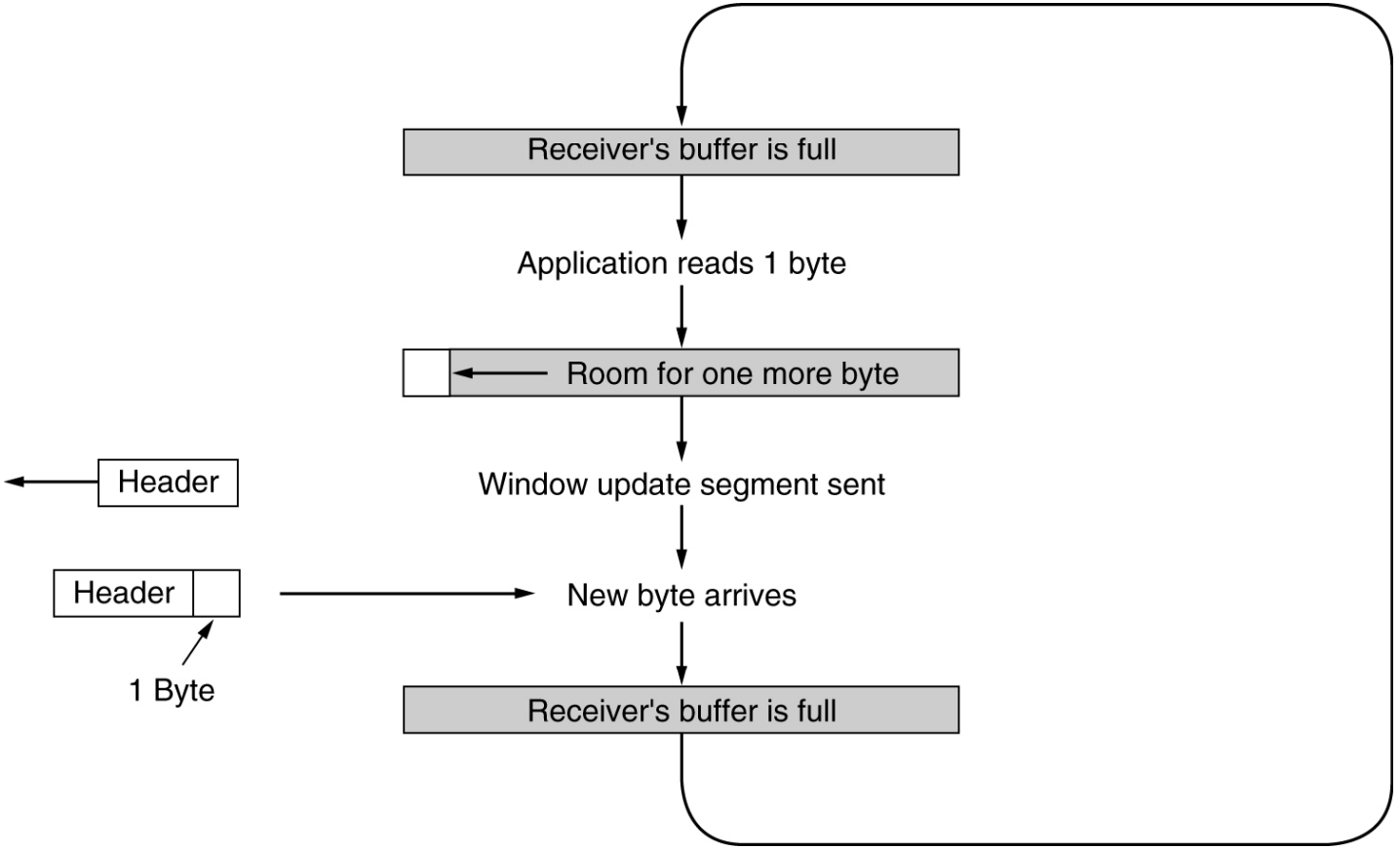


TCP: Sliding Window

- Nagle's algorithm
  - Consider a telnet connection to an interactive editor that reacts on every keystroke.
  - Solution 1: to delay acknowledgements and window updates for 500 msec in the hope of acquiring some data on which to hitch a free ride.
  - Solution 2: Nagle's algithm
    - When data come into the sender one byte at a time, just send the first byte and buffer all the rest until the outstanding byte is acknowledged. Then send all the buffered characters in one TCP segment and start buffering again until they are acknowledged.
    - To disable it by TCP_NODELAY option

TCP: Transmission Policy

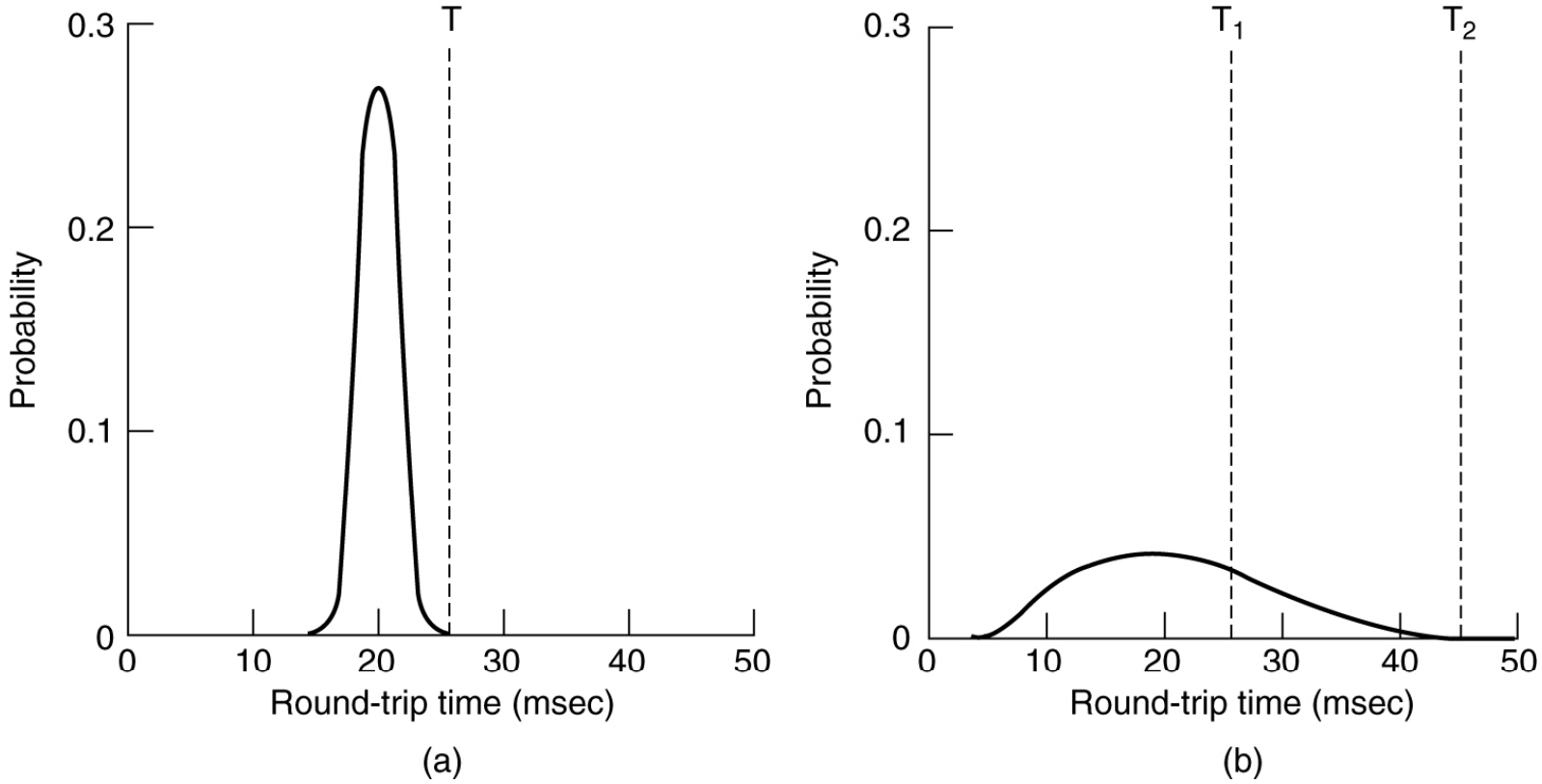
- Clark's solution for silly window syndrome
  - To prevent the receiver from sending a window update for 1 byte.



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**
  - SRTT (Smoothed Round-Trip Time) (α=7/8)

$$\mathbf{SRTT = \alpha\ SRTT + (1- \alpha)\ R}$$

  - RTTVAR(Round-Trip Time VARiation) (β=3/4)

$$\mathbf{RTTVAR = \beta\ RTTVAR + (1 - \beta)\ |\ SRTT - R\ |}$$

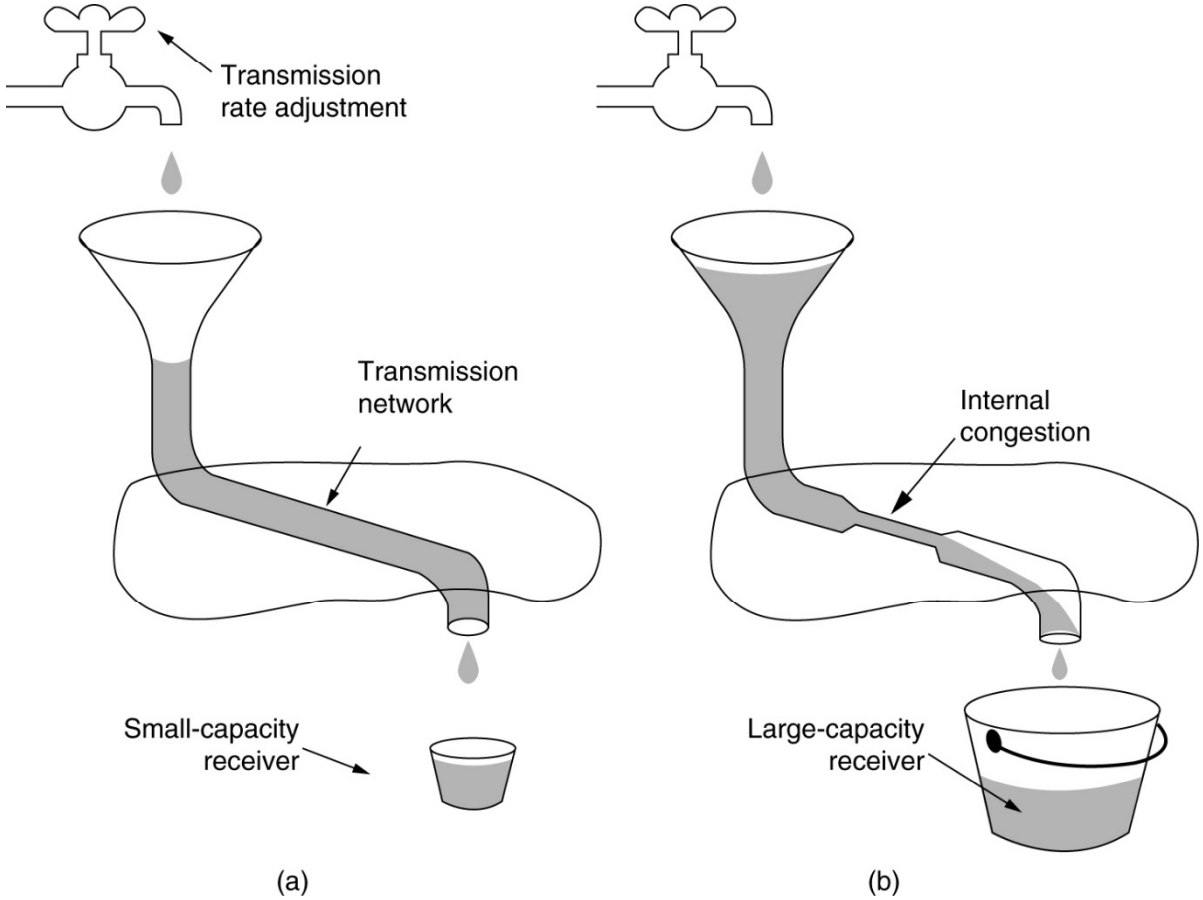
  - RTO (Retransmission TimeOut)

$$\mathbf{RTO = SRTT = 4 * RTTVAR}$$

- **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: Congestion Control

- Congestion window
- Receiver window

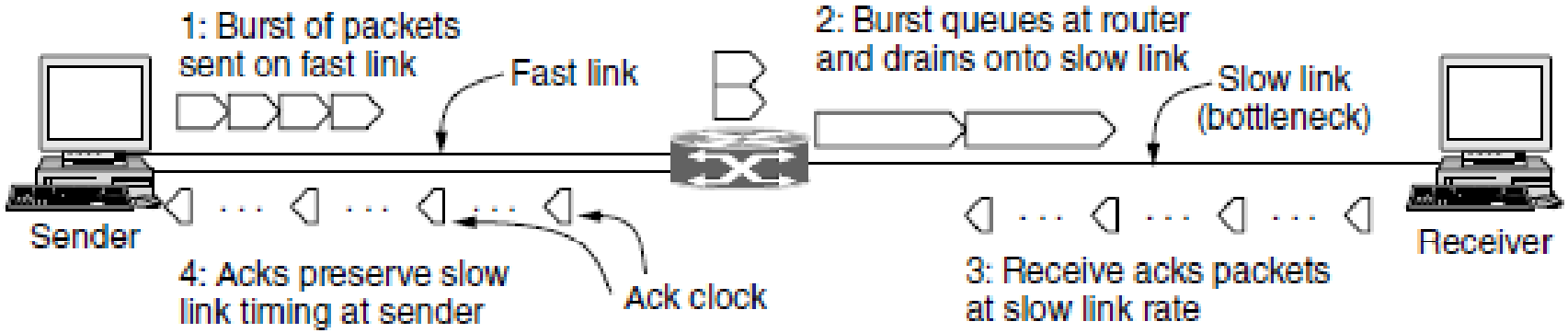


TCP: Congestion Control

- Congestion Control Algorithm by Van Jacobson (1988)
  - To approximate an AIMD congestion window
  - To represent congestion signal by packet loss
  - To measure packet loss by a retransmission timer
  - To split data into segments (Ack Clock)
  - To use the optimal congestion window

TCP: Congestion Control

- The way packets are sent into the network must be matched to the network path eve over short periods of times. (Ack clock)



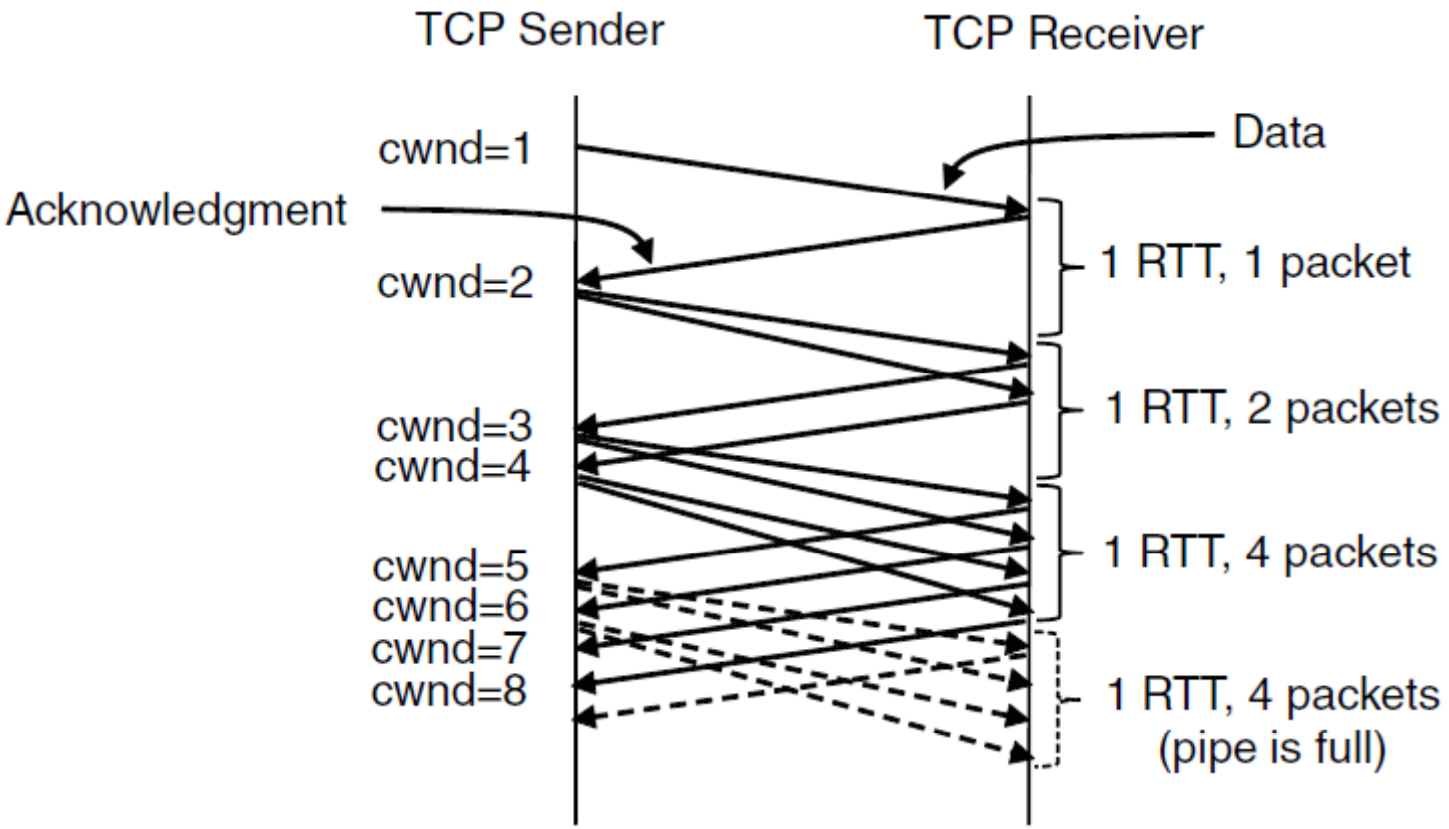
A burst of packets from a sender and the returning ack clock.

TCP: Congestion Control

- The AIMD rule will take a very long time to reach a good operating point on fast networks if the congestion window is started from a small size.
  - Slow start from an initial congestion window of 1 segment
  - Additive increase from an initial congestion window of 1 segment.
  - Slow start followed by additive increase in TCP Tahoe.
  - Fast recovery and the sawtooth pattern of TCP Reno.
  - To use selective ackowledgements

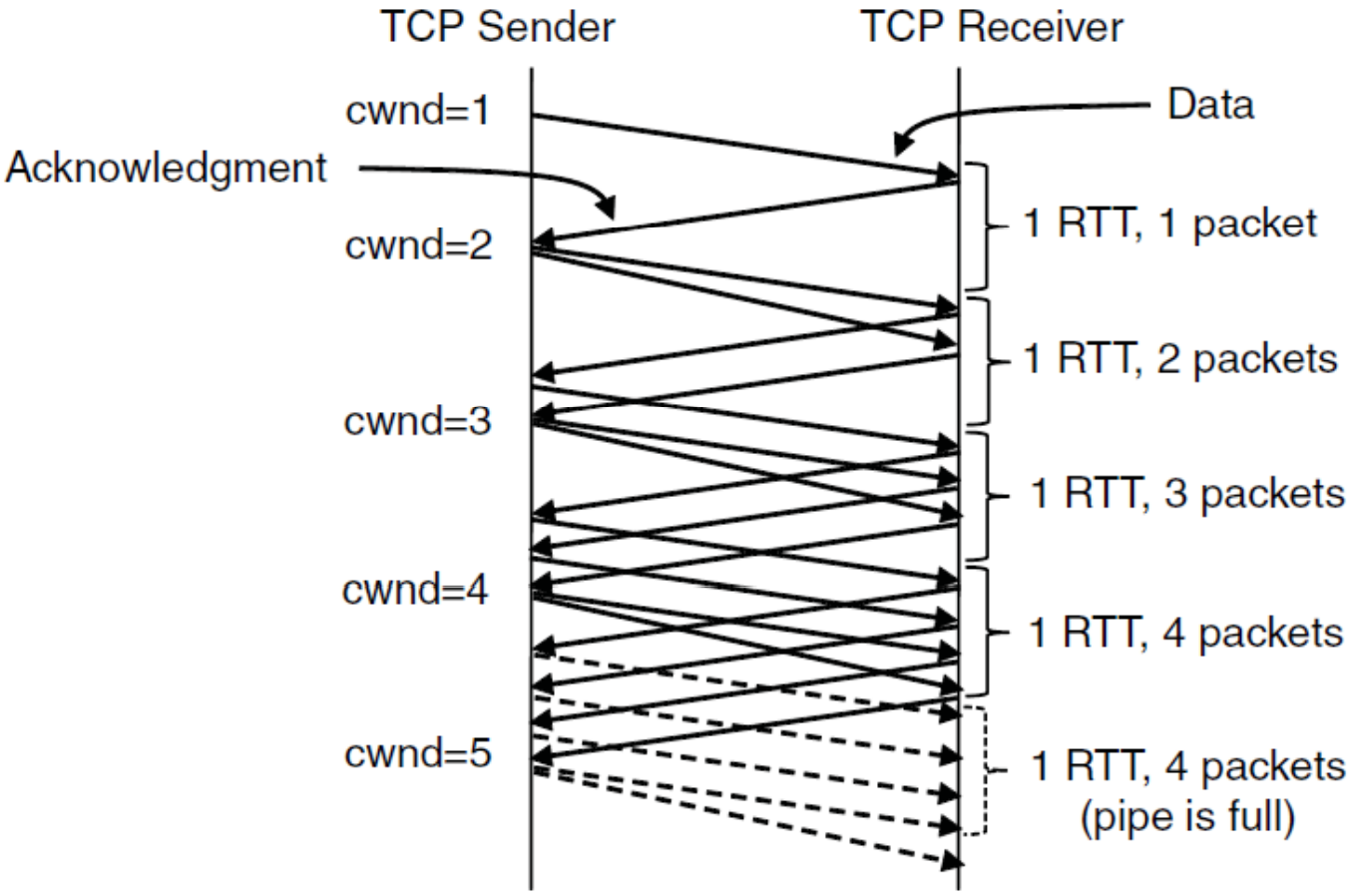
TCP: Congestion Control

Slow start from an initial congestion window of 1 segment



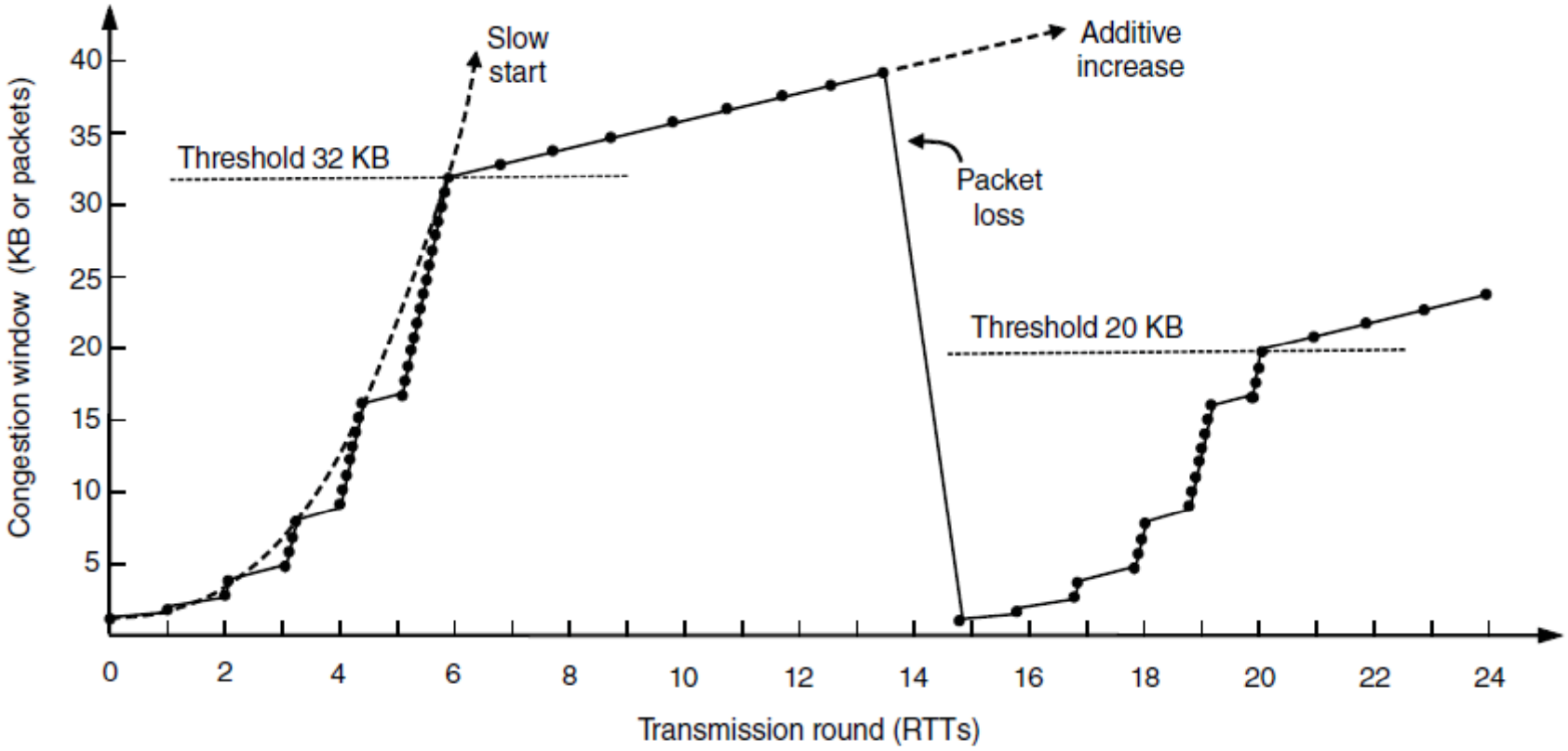
TCP: Congestion Control

Additive increase from an initial congestion window of 1 segment.



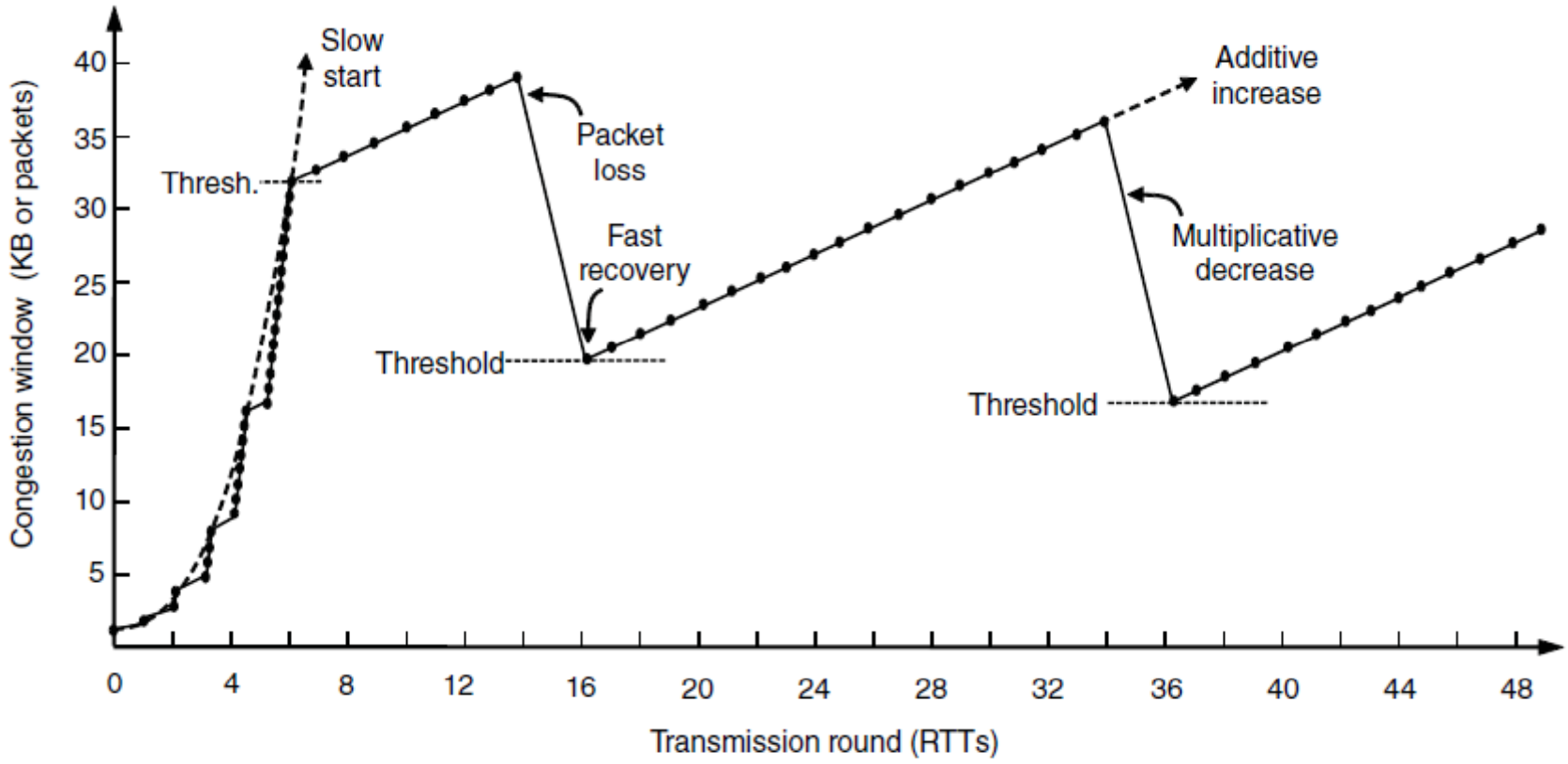
TCP: Congestion Control

Slow start followed by additive increase in TCP Tahoe.



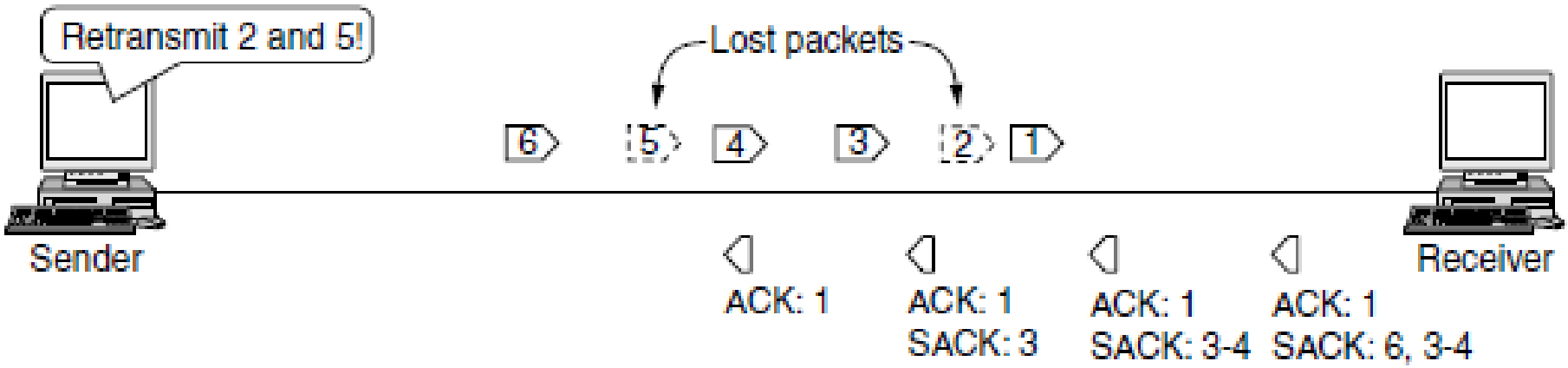
TCP: Congestion Control

Fast recovery and the sawtooth pattern of TCP Reno.



TCP: Congestion Control

Selective ackowledgement



TCP: The Future

- SCTP (Stream Control Transmission Protocol) (RFC 4960) is message-oriented like UDP and ensures reliable, in-sequence transport of messages.
- SST (Structured Stream Transport) is an experimental transport protocol designed to address the needs of modern applications that need to juggle many asynchronous communication activities in parallel, such as downloading different parts of a web page simultaneously and playing multiple audio and video streams at once.
- FAST TCP (Wei et al. 2006)
- Transactional TCP
- …

PERFORMANCE ISSUES

- Performance Problems in Computer Networks
- Network Performance Measurement
- System Design for Better Performance
- Fast TPDU Processing
- Protocols for Gigabit Networks

Performance Issues: Performance Problems

- Some performance problems, such as congestion, are caused by temporary resource overloads.
- Performance also degrades when there is a structural resource imbalance.
- Overloads can also be synchronously triggered.
  - Broadcast storm due to errors.
  - DHCP after an electrical power failure.
- Poor performance can occur due to lack of system tuning.
  - Low priority for network processing
  - Less buffers for network processing
  - Setting timeouts incorrectly

Performance Issues: Performance Problems

- Gigabit networks bring with them new performance problems.

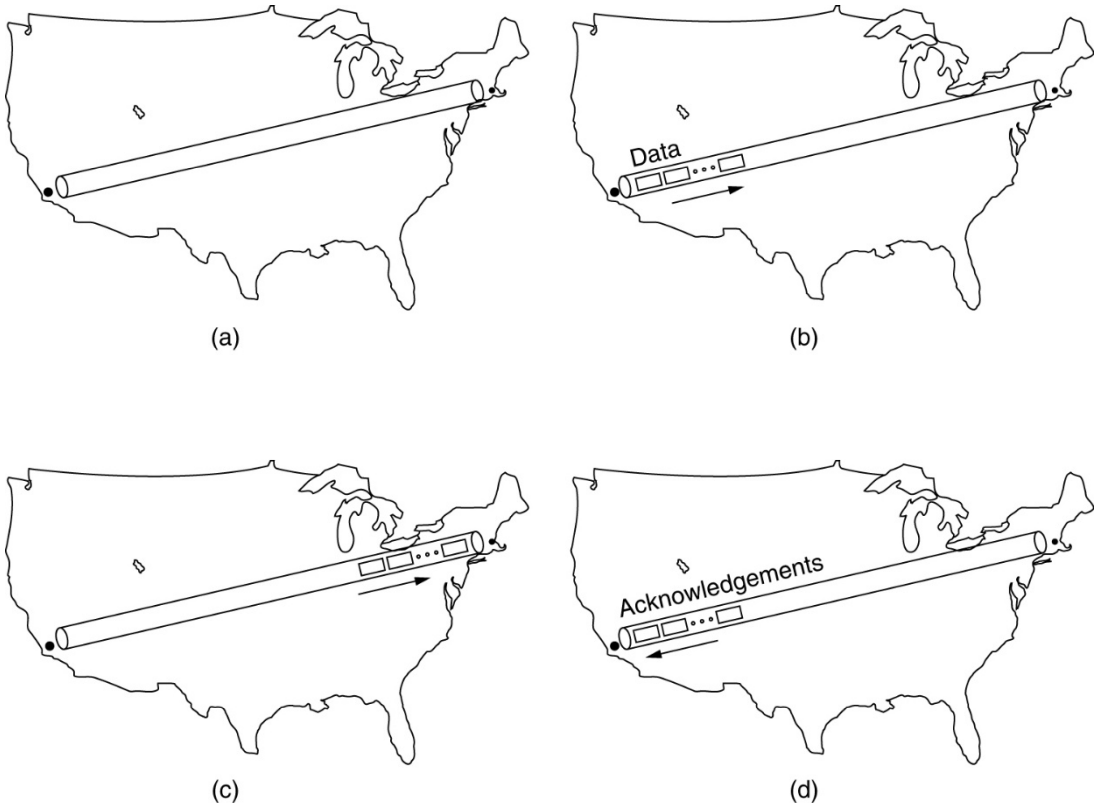
  The state of transmitting one megabit from San Diego to Boston

  (a) At t = 0,

  (b) After 500 μsec,

  (c) After 20 msec,

  (d) after 40 msec.



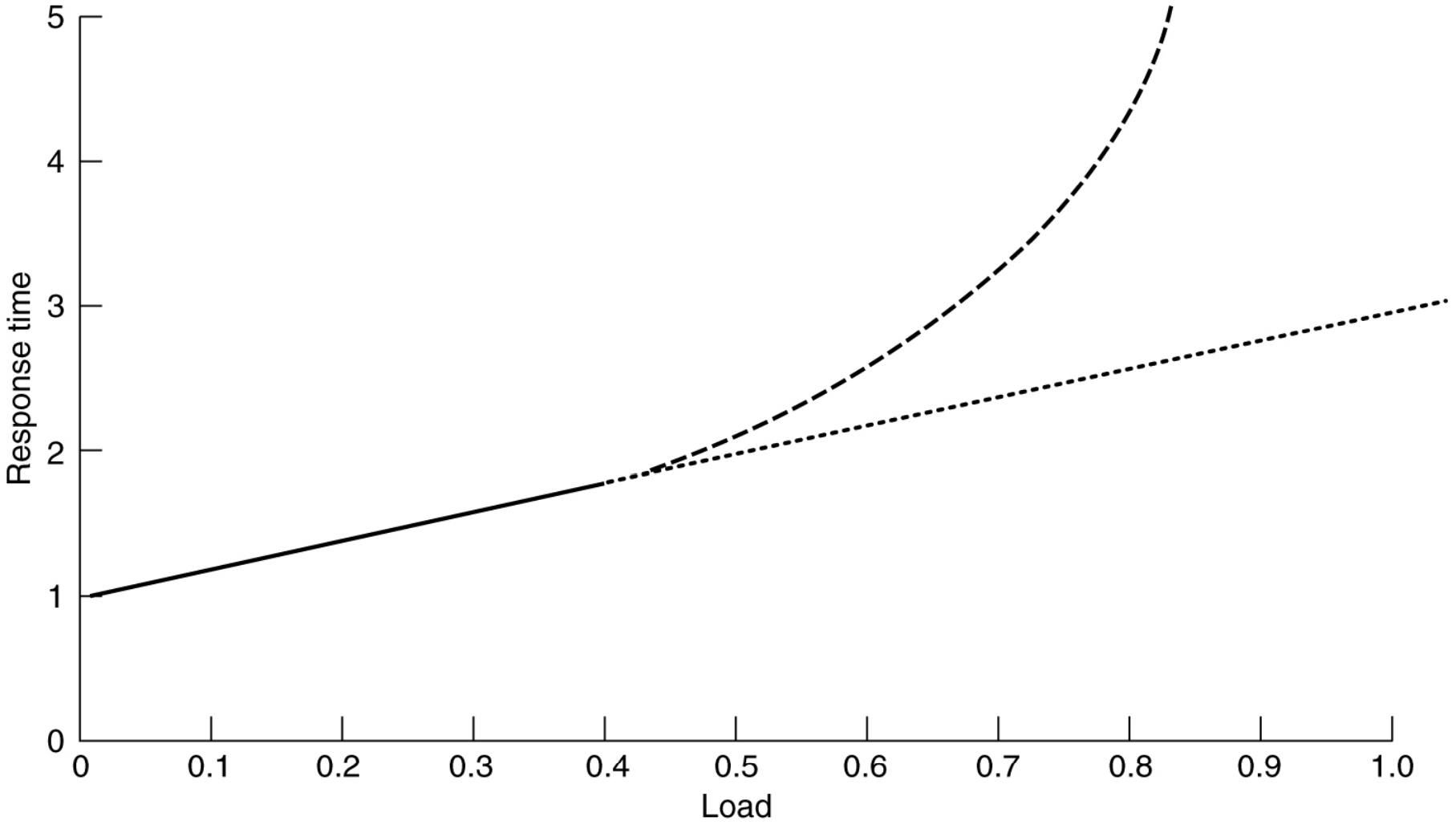
Performance Issues: Performance Measurement

- The basic loop for improving network performance.
  - Measure relevant network parameters and performance.
  - Try to understand what is going on.
  - Change one parameter to .
- These steps are repeated until the performance is good enough or it is clear that the last drop of improvement has been squeezed out.

Performance Issues: Performance Measurement

- Make sure that the sample size is large enough.
- Make sure that the samples are representative.
- Be careful when using a coarse-grained clock.
- Be sure that nothing unexpected is going on during your tests.
- Caching can wreak havoc with measurements.
- Understand what you are measuring.
- Be careful about extrapolating the results

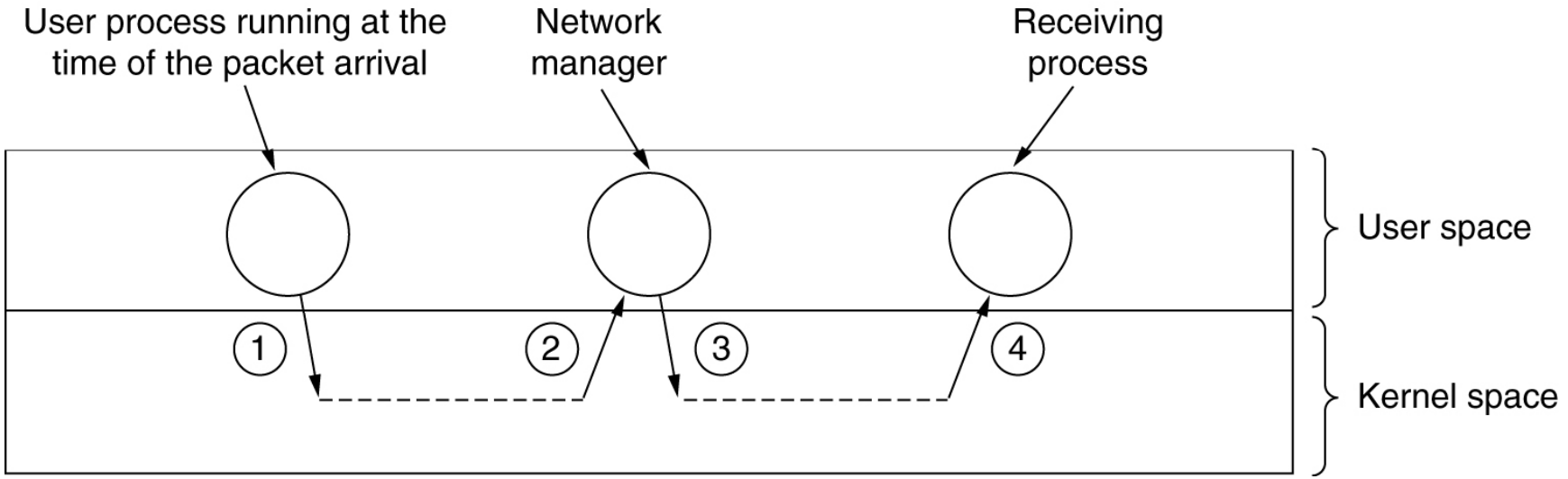
Performance Issues: Performance Measurement



Performance Issues: Design for Better Performance

- CPU speed is more important than network speed.
- Reduce packet count to reduce software overhead.
  - Nagle's algorithm and Clark's solution.
- Minimize context switches.

Four context switches to handle one packet



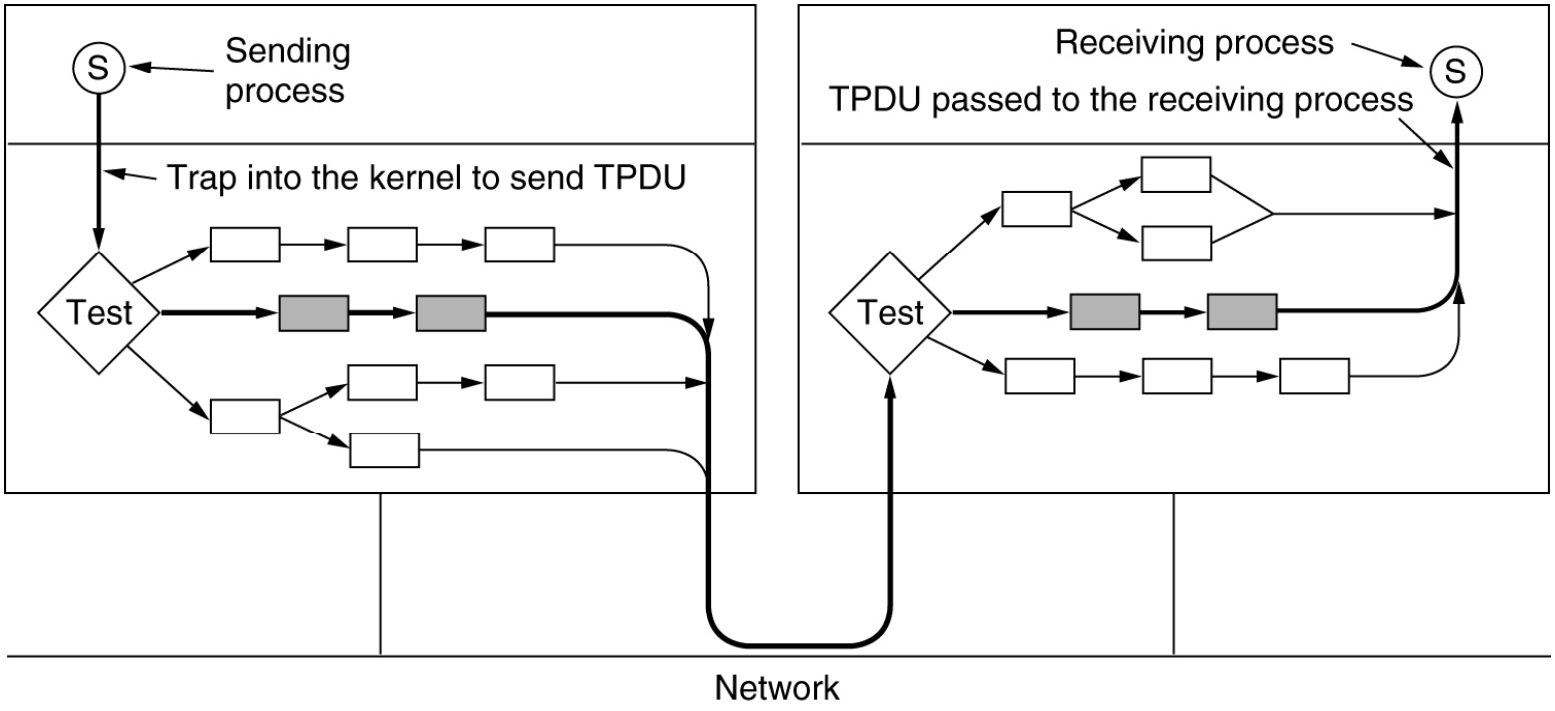
Performance Issues: Design for Better Performance

- Minimize copying:
  - On a 50-MIPS machine, making three copies of each packet at five instructions per 32-bit word copied requires 75 nsec per incoming byte. => 107Mbps
  - Overhead for header processing, interrupt handling, and context switches => 50Mbps
  - Memory operations (not register-register operations) => 16Mbps
- You can buy more bandwidth but not lower delay.
- Avoiding congestion is better than recover it (An ounce of prevention is worth a pound of cure).
- Avoid timeouts.

Performance Issues: Fast TPDU Processing

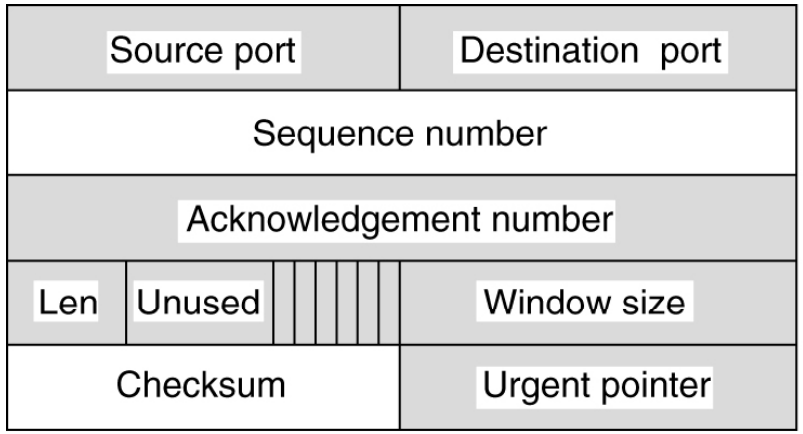
The fast path from sender to receiver is shown with a heavy line.

The processing steps on this path are shaded.

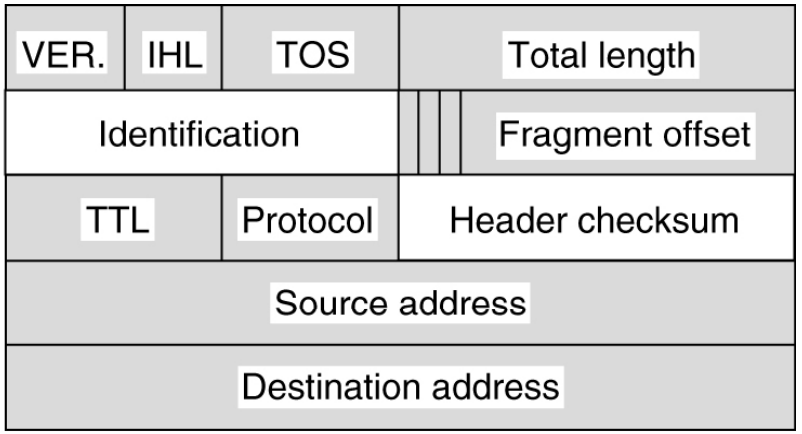


Performance Issues: Fast TPDU Processing

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



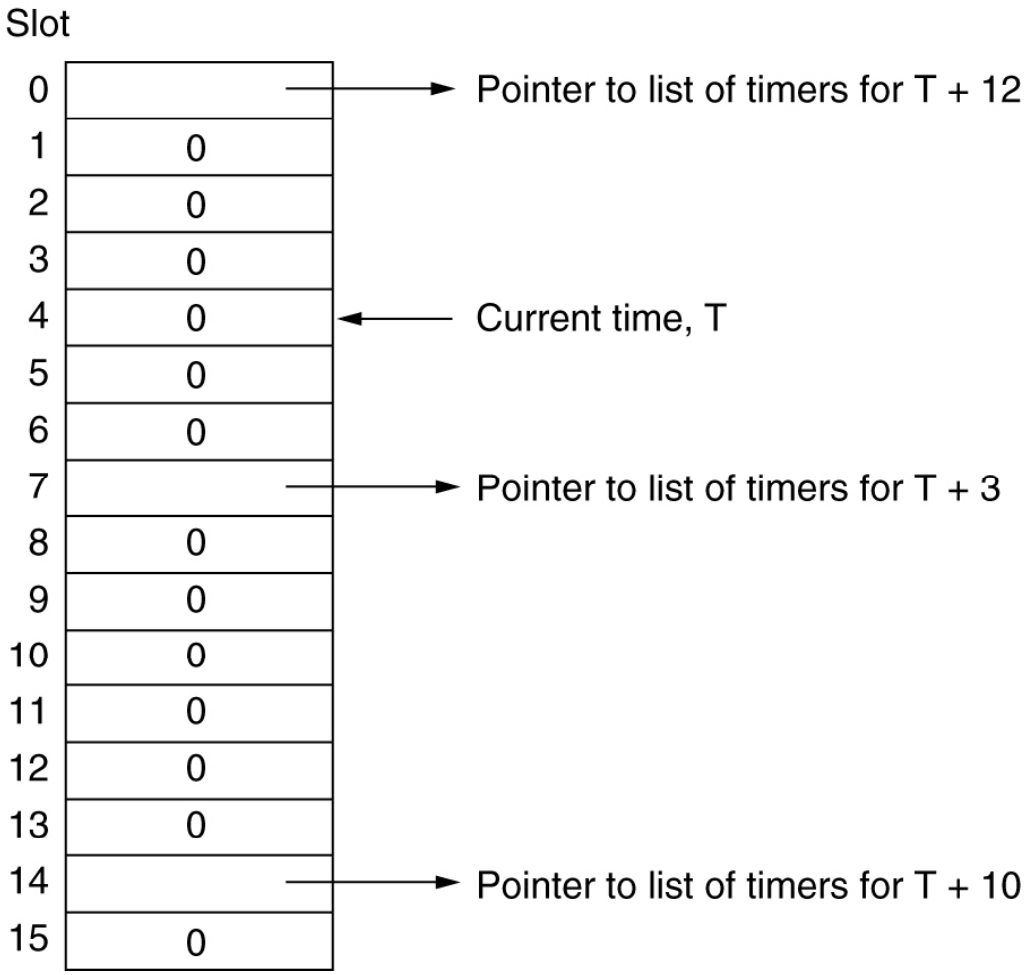
(a)



(b)

Performance Issues: Fast TPDU Processing

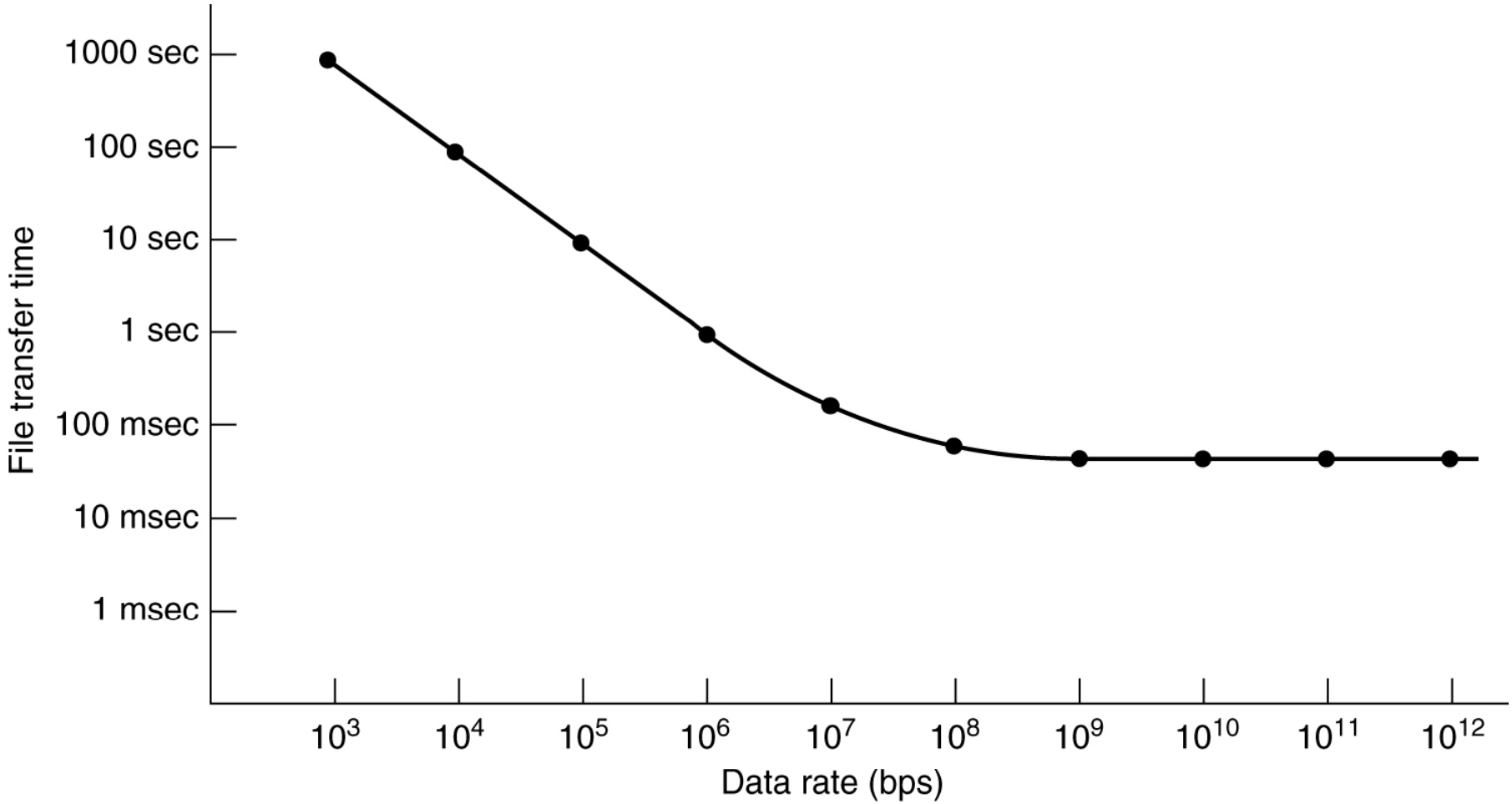
- A timing wheel.



Performance Issues: Gigabit Network Protocols

- The problems with gigabit network:
  - Many protocols use 32-bit sequence numbers
    - For 56 kbps, the wrap time is over 1 week.
    - For 10Mbps, the wrap time is 57 minute.
    - For 1 Gbps, the wrap time is 34 seconds.
  - Communication speeds have improved much faster then computing speeds.
  - The go back n protocol performs poly on lines with a large bandwidth-delay product.
  - Gigabit lines are fundamentally different from megabit lines.
  - The variance in the packet arrival times is as important as the mean delay itself.

Performance Issues: Gigabit Network Protocols



Performance Issues: Gigabit Network Protocols

- Solutions
  - **Design for speed, not for bandwidth optimization.**
  - To use a rate-based protocol rather than a sliding window protocol.
  - Packets should be well layout.
  - The header and data should be separately checksummed.
  - The maximum data size should be large.
  - To send a normal amount of data along with the connection request.
  - …

Homework

- **6.4**

  In both parts of Fig. 6-6, there is a comment that the value of SERVERPORT must be the same in both client and server. Why is this so important?

- **6.9**

  Imagine that a two-way handshake rather than a three-way handshake were used to set up connections. In other words, the third message was not required. Are deadlocks now possible? Give an example or show that none exist.

Homework

- **6.15**

  Why does UDP exist? Would it not have been enough to just let user processes send raw IP packets?

- **6.17**

  A client sends a 128-byte request to a server located 100 km away over a 1-gigabit optical fiber. What is the efficiency of the line during the remote procedure call?

- **6.23**

  Datagram fragmentation and reassembly are handled by IP and are invisible to TCP. Does this mean that TCP does not have to worry about data arriving in the wrong order?

- **6.28**

  The maximum payload of a TCP segment is 65,495 bytes. Why was such a strange number chosen?

Homework

- **6.32**

  If the TCP round-trip time, RTT, is currently 30 msec and the following acknowledgements come in after 26, 32, and 24 msec, respectively, what is the new RTT estimate using the Jacobson algorithm? Use α=0.9.

- **6.36**

  In a network whose max segment is 128 bytes, max segment lifetime is 30 sec, and has 8-bit sequence numbers, what is the maximum data rate per connection?

- **6.39**

  To get around the problem of sequence numbers wrapping around while old packets still exist, one could use 64-bit sequence numbers. However, theoretically, an optical fiber can run at 75 Tbps. What maximum packet lifetime is required to make sure that future 75-Tbps networks do not have wrap around problems even with 64-bit sequence numbers? Assume that each byte has its own sequence number, as TCP does.