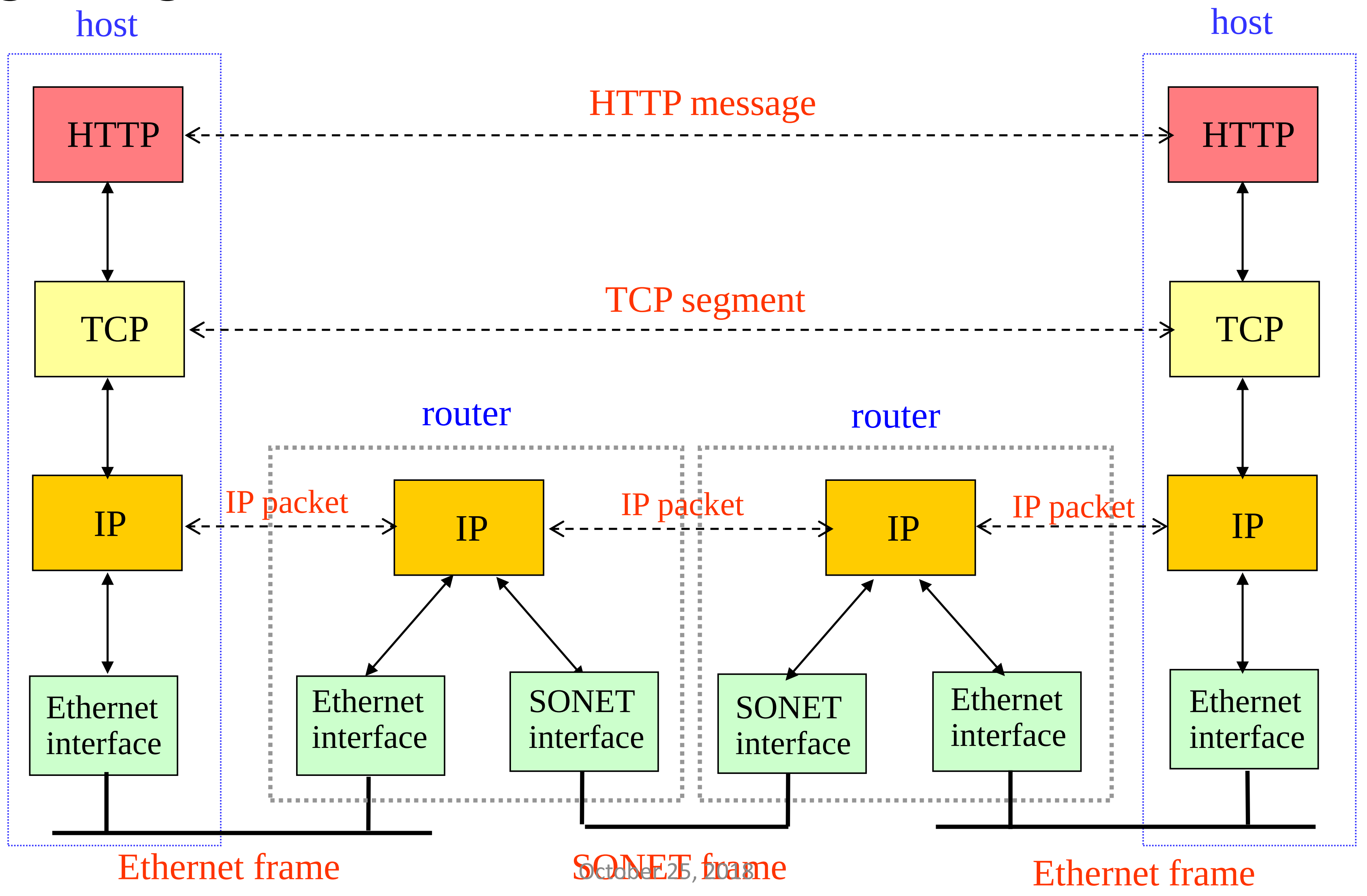
CMSC 417

Computer Networks

Prof. Ashok K Agrawala

Message, Segment, Packet, and Frame



Internet 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 »

TCP

- Reliable end-to-end byte stream over an unreliable internetwork
- Dynamically adapt to the properties of the network
- Robust

- RFC 793 – 1122 and 1323

TCP Transport Entity

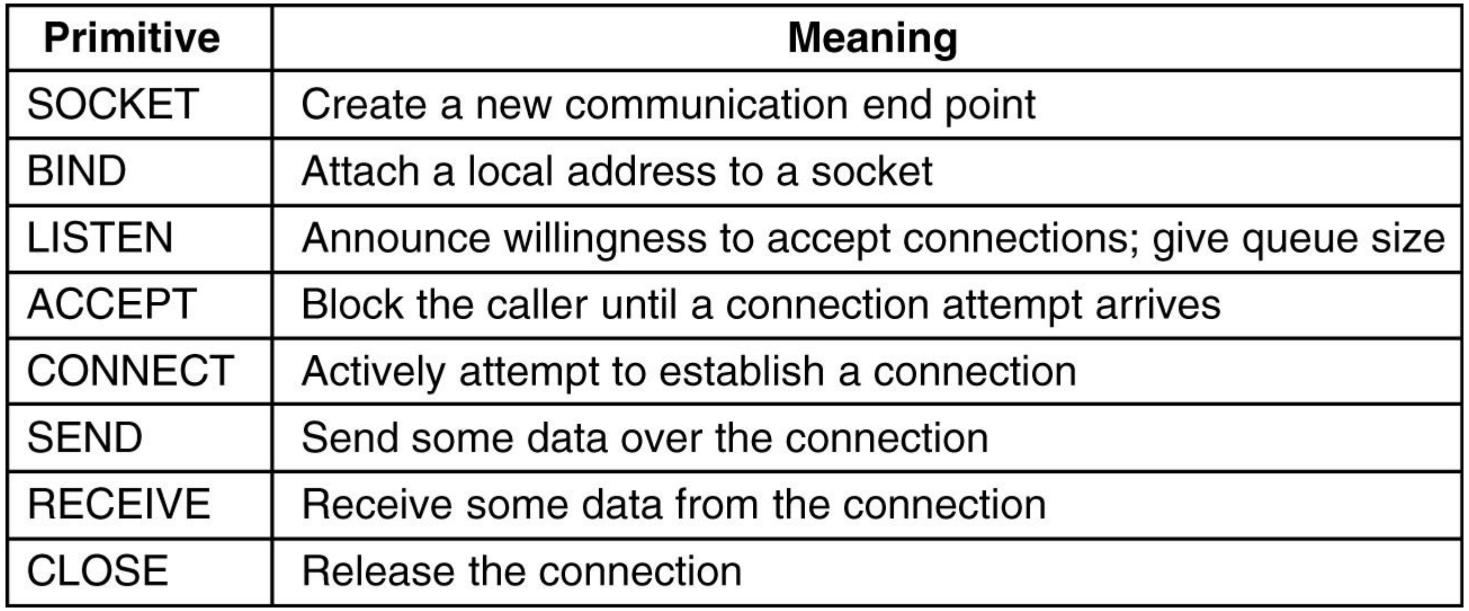
- Implemented as
  - Library procedures
  - User process, or
  - Part of the kernel
- Accepts data streams from local processes
- Breaks them into segments of >64KB (usually 1460 bytes)
- Sends each piece in a separate IP packet
- On receive side – reconstruct the original byte stream and give to a process.
- Must recover from errors – time outs, retransmissions, etc.

TCP Service Model

- End points called sockets
  - Socket number
    - IP address of the host
    - 16-bit number (called the *port*)
- Connection Oriented – Full Duplex, Point-to-Point
  - Establish a connection between sockets
  - An socket may be used for multiple connections at the same time
  - Connection (*socket1, socket2*)

Berkeley Sockets

The socket primitives for TCP.

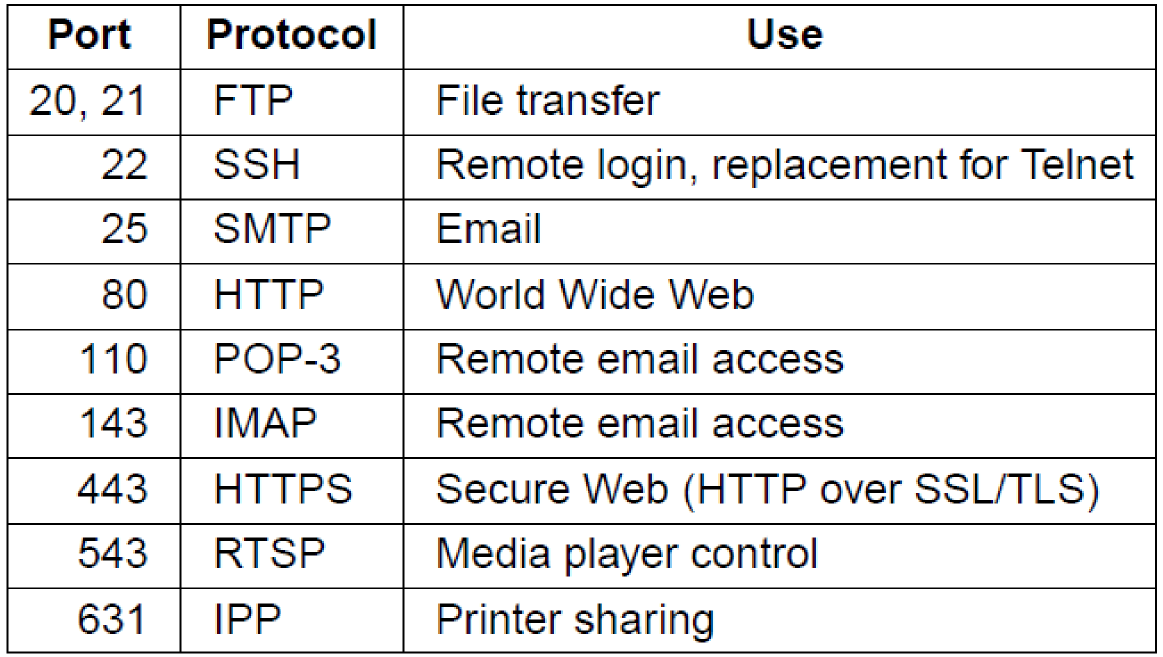


Primitive	Meaning
SOCKET	Create a new communication end point
BIND	Attach a local address to a socket
LISTEN	Announce willingness to accept connections; give queue size
ACCEPT	Block the caller until a connection attempt arrives
CONNECT	Actively attempt to establish a connection
SEND	Send some data over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

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

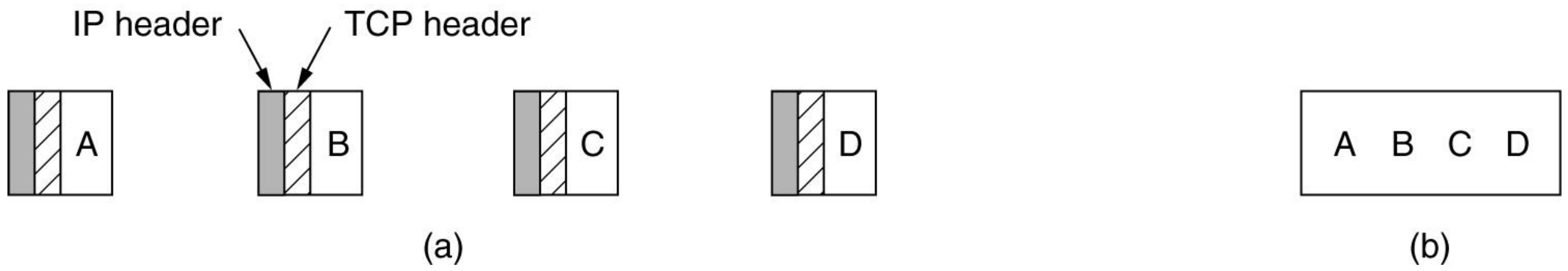


Port	Protocol	Use
20, 21	FTP	File transfer
22	SSH	Remote login, replacement for Telnet
25	SMTP	Email
80	HTTP	World Wide Web
110	POP-3	Remote email access
143	IMAP	Remote email access
443	HTTPS	Secure Web (HTTP over SSL/TLS)
543	RTSP	Media player control
631	IPP	Printer sharing

Internet Daemon

- Attaches to multiple well-known ports and waits
- When a connection comes in it forks off a new process and executes the appropriate daemon
- That daemon handles the request

The TCP Service Model



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

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

TCP Service

- No message boundaries are preserved
- Send data as received or buffer it
- PUSH Flag
  - Send data now
  - Useful in sending command from terminal
- Urgent Data
  - DEL or CTRL-C to break off a remote computation
  - Use URGENT flag – Transmit everything right now
    - Receiving application is interrupted

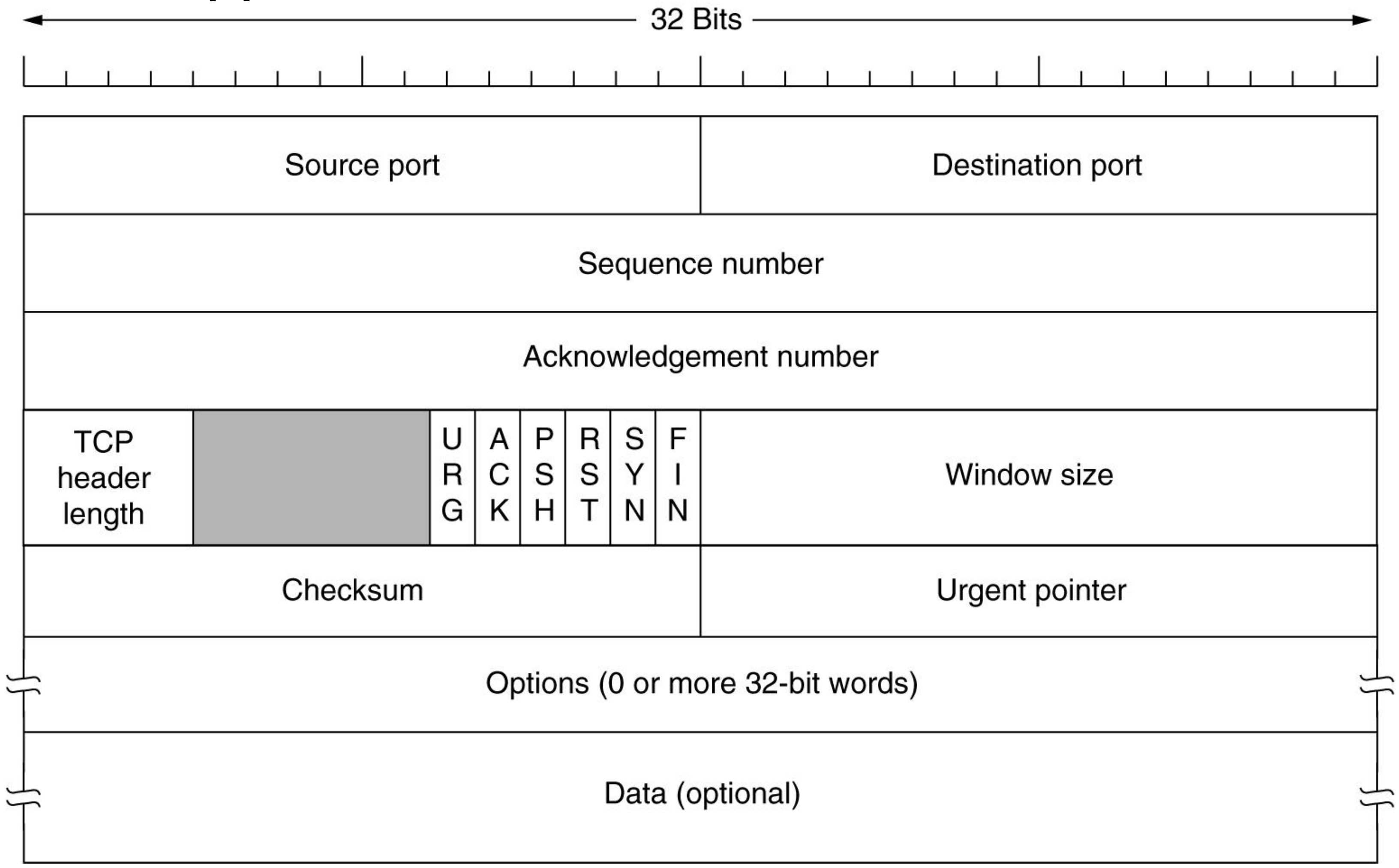
TCP Protocol

- Exchange segments
  - 20 byte header (plus optional parts)
  - 0 or more data bytes
- Accumulate data from several writes into one segment
- May split data from one write into multiple segments
- Each segment, including the header, must be <65515 byte IP payload
- Each network has MTU- Maximum Transfer Unit
  - Each segment must be less than or equal to MTU

TCP Protocol

- TCP uses sliding window protocol
- Sequence numbers are for bytes not segments
- Sending – start a timer
- Receiving – send an ack with sequence number = next sequence number expected

The TCP Segment Header



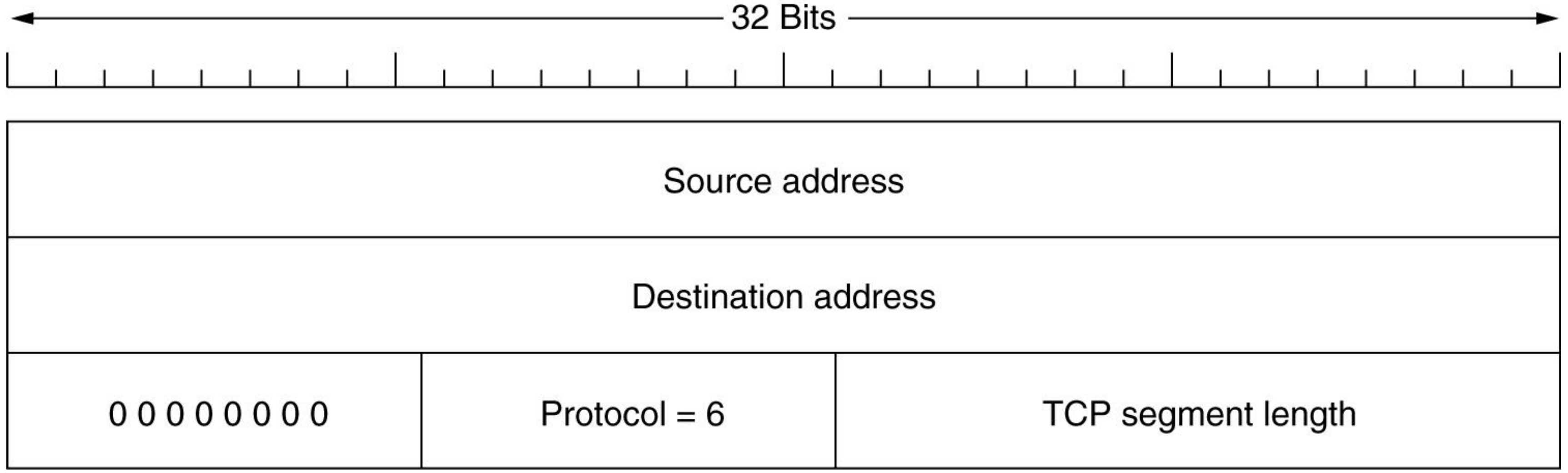
TCP Header.

TCP Segment Header

- Source and Destination Port
- Sequence number
- Acknowledgement Number
    - Next byte expected
- TCP Header Length – Number of 32 bit words in TCP header
- Flags
    - URG – set to 1 if Urgent Pointer is in use
        - Used to indicate a byte offset from the current seq no at which urgent data care there
    - ACK – set to 1 when ack no is valid
    - PSH – Push bit
    - RST – Reset
    - SYN – Used for connection establishment
    - FIN – Used to close a connection

The TCP Segment Header (2)

- The pseudoheader included in the TCP checksum
- Checksum the header, the data and the pseudoheader
- Add all 16-bit words in 1's complement and then take 1's complement of the sum
- To check calculate on the entire segment and result should be 0.



TCP Window

- Window size tells how many bytes may be sent starting at the byte acknowledged
  - If 0 means do not send now.
  - May send a segment with same ack no and non-zero window size.

Maximum segment Size

- All hosts are required to accept TCP segments of 536+20 =556 bytes
- May negotiate max segment size using options.
- Another negotiable parameter – Window Scale
  - May shift to the left by up to 14 bits
  - Giving a max window size of 2^{30} bytes

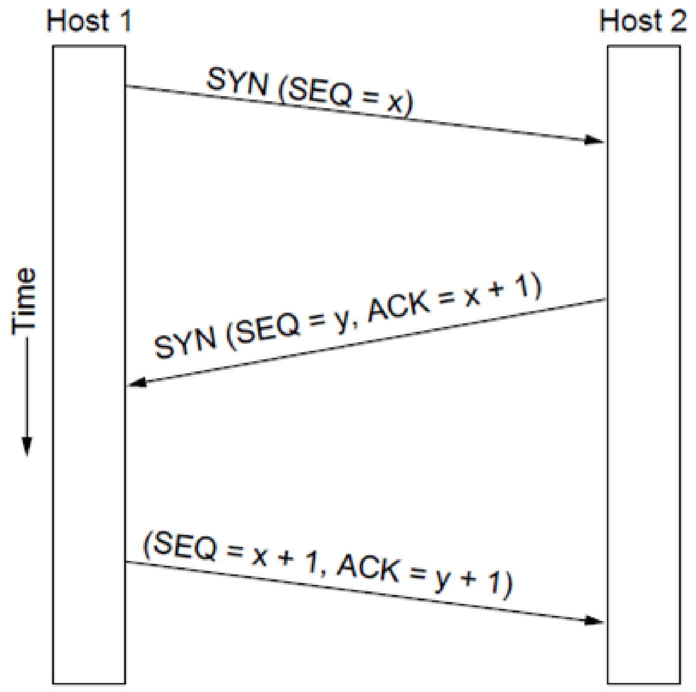
Connection Establishment

- Uses three-way handshake
- Server passively listens
  - LISTEN and ACCEPT primitives
- Client executes CONNECT
  - Send a TCP Segment with SYN bit on and ACK bit off.
- Check to see if there is a process listening
  - If not send a RST
  - If yes, then give the segment to the process
  - If accepted – send an ACK message

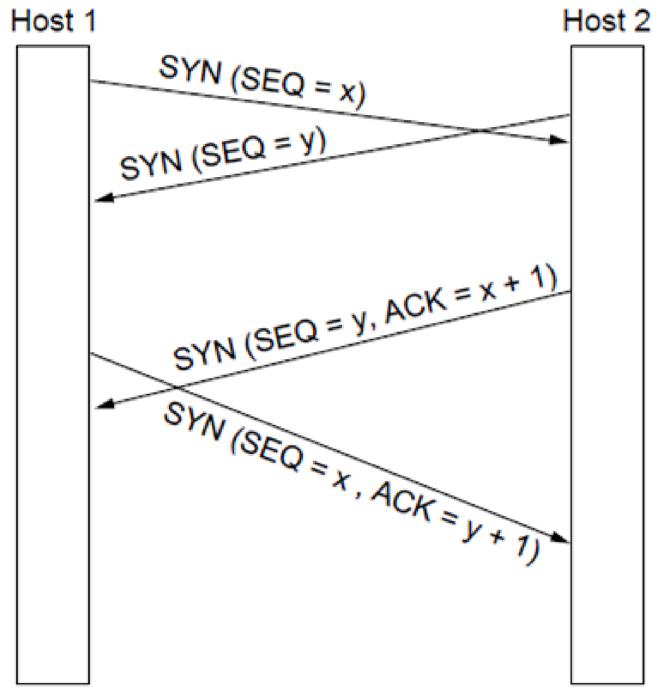
TCP Connection Establishment

TCP sets up connections with the three-way handshake

- Release is symmetric, also as described before



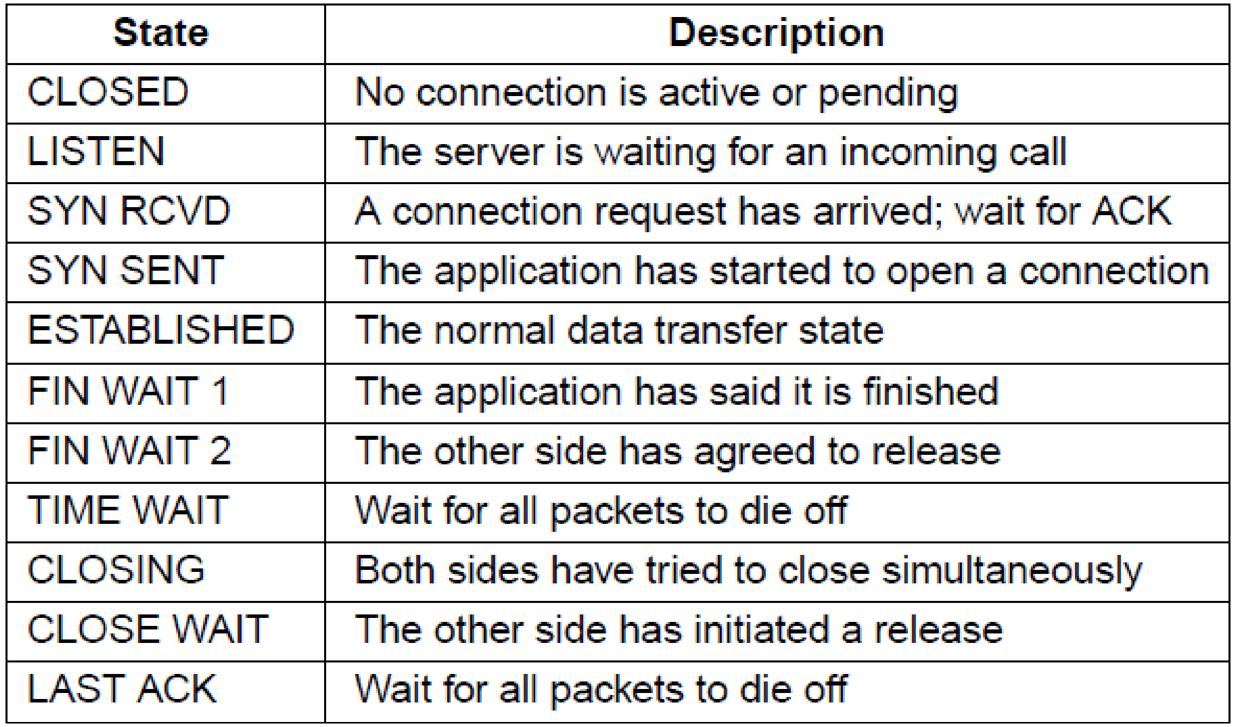
Normal case



Simultaneous connect

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

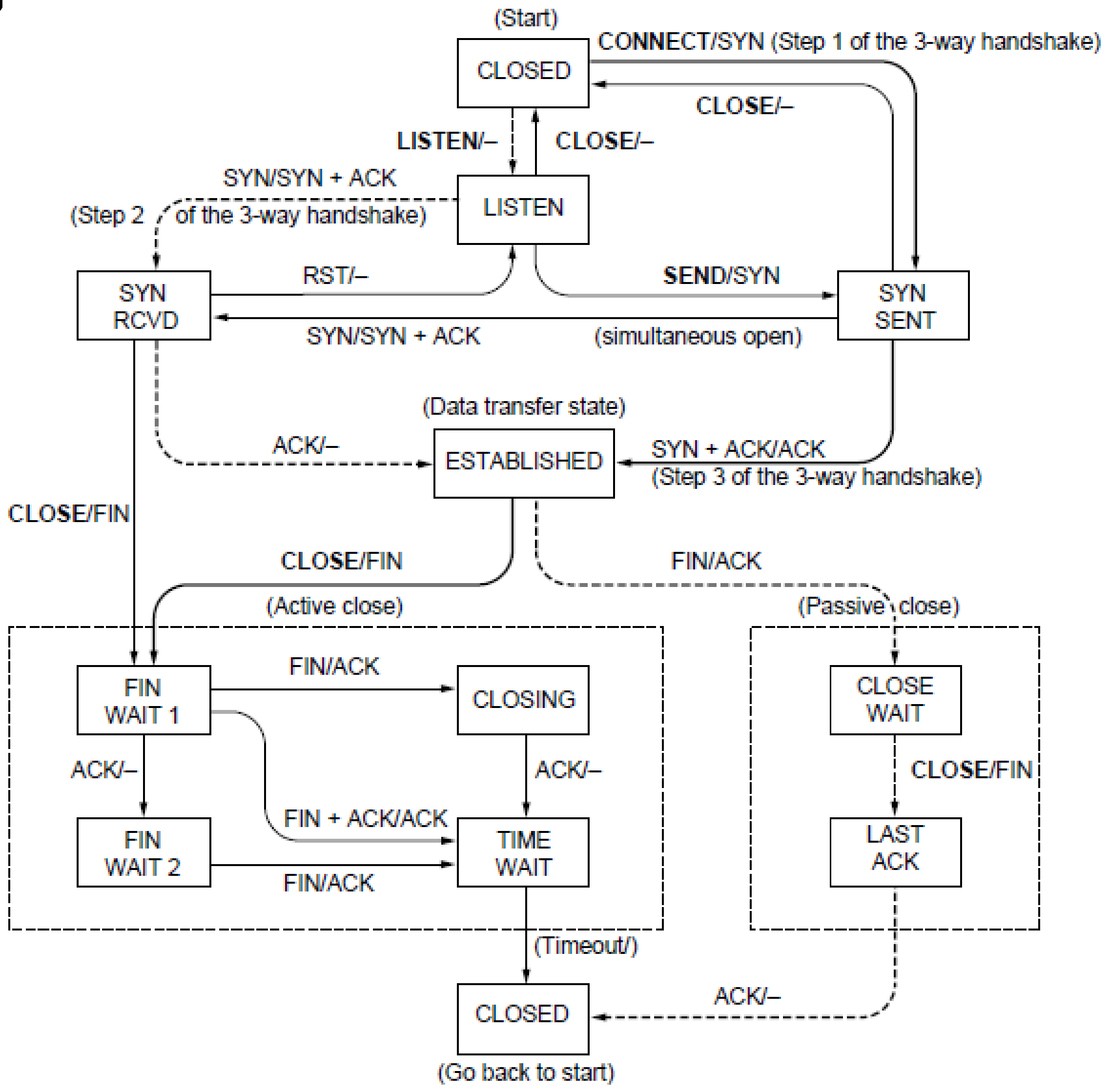
TCP Connection State Modeling (2)

Solid line is the normal path for a client.

Dashed line is the normal path for a server.

Light lines are unusual events.

Transitions are labeled by the cause and action, separated by a slash.



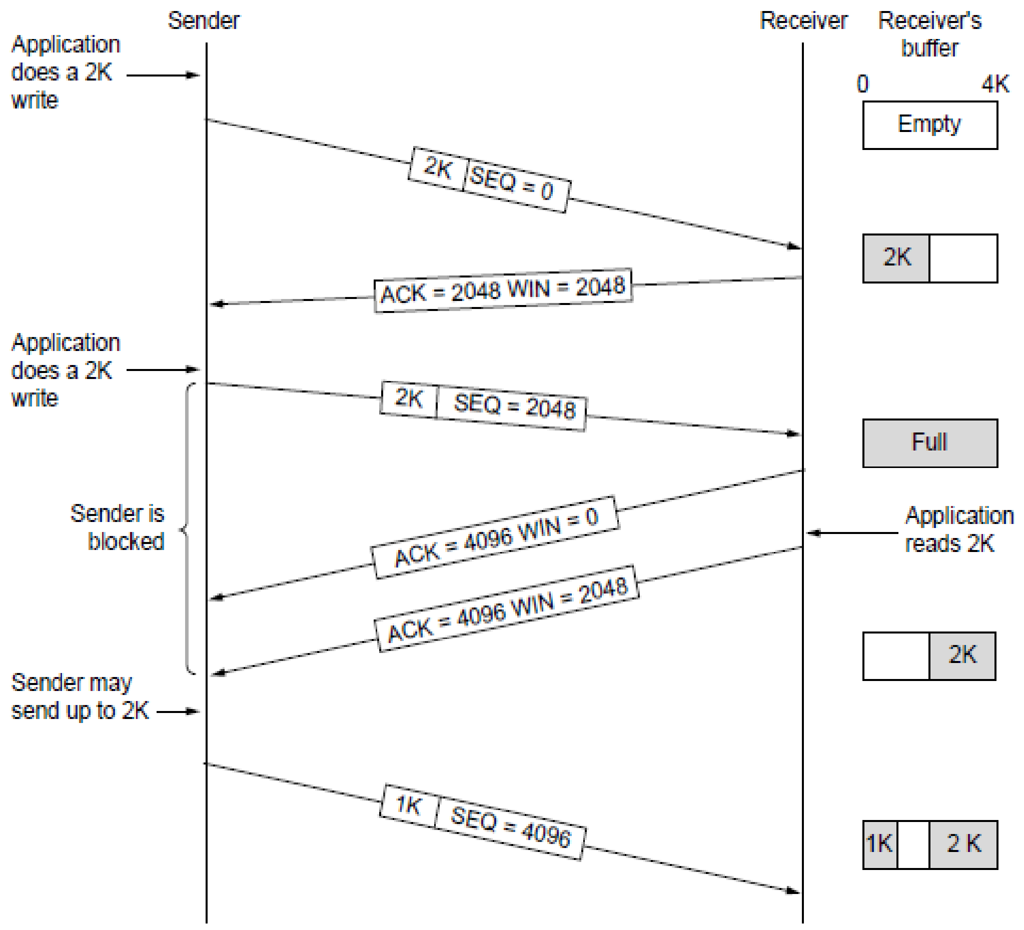
Connection Release

- Think of the connection as a pair of simplex connections
    - Each is released independently
- Either party sends a segment with FIN bit set
- When FIN acked that direction is shut down

TCP Sliding Window (1)

TCP adds flow control to the sliding window as before

- ACK + WIN is the sender's limit



Sliding Window

When Window Size = 0

- Sender stops sending, except
  - Urgent Data
  - Window Probe –
    - One Byte Segment – forcing receiver to re-announce the next byte expected as Window size.

Example Situation Telnet

- 1 Byte
  - 21 Byte Segment
  - 41 Byte Packet

- 40 Byte Ack
- 41 Byte Echo

- 40 Byte Ack

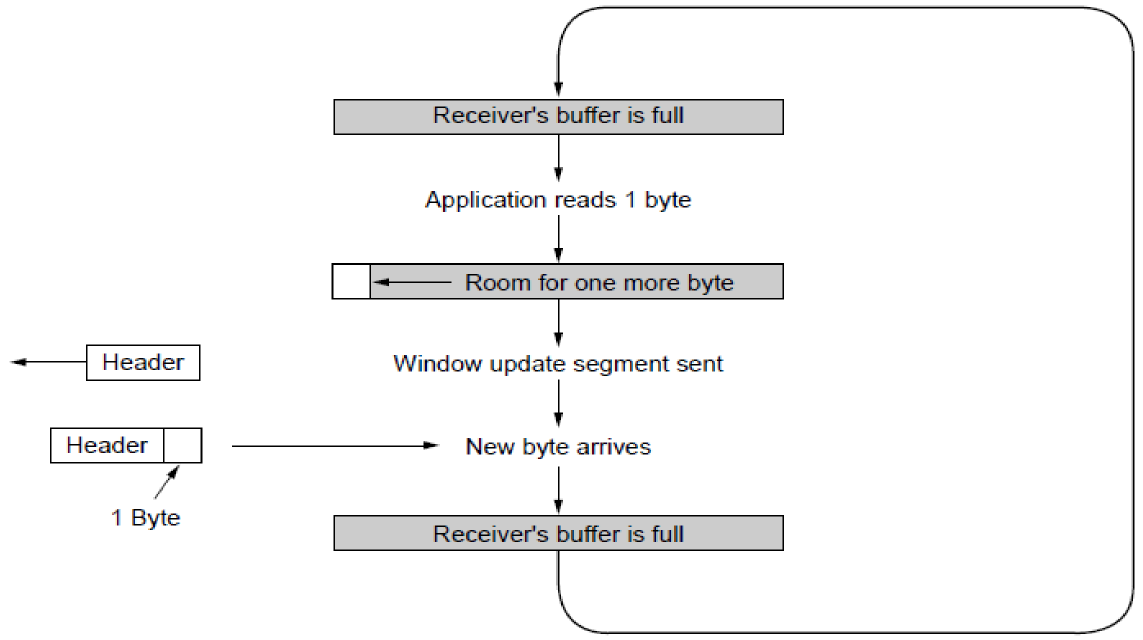
162 Bytes on the network for one byte data

Delayed Ack - 500 ms

TCP Sliding Window (2)

Need to add special cases to avoid unwanted behavior

- E g silly window syndrome [below]



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

Handling Silly Window Problem

- Delay ack and window updates for 500 ms.
- Nagle’s Algorithm
  - When data comes in one byte at a time
    - Send the first byte and buffer the rest till the outstanding byte is acknowledged
    - Then send all the buffered characters in one TCP segment
  - Mouse movements have to be sent – Burst does not work well.
- Clark’s Solution
  - Wait until decent amount of space available then advertise
    - Max segment size or half buffer
  - Sender not send tiny segments

Timer Management

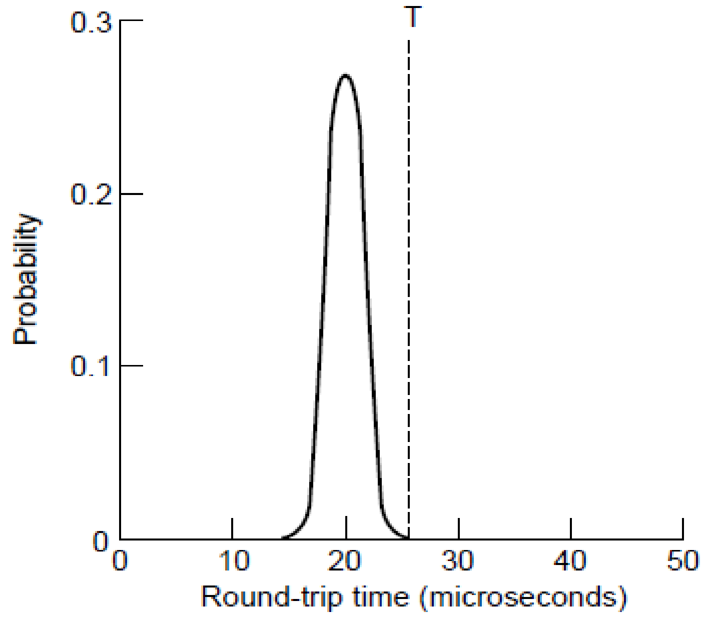
- TCP uses multiple timers
  - Most important is the *retransmission timer*
  - What value to set it at??
- Round Trip time
  - Highly variable
  - Varies over time
  - Have to track it
  - Estimate it
  - M is a new measurement
  - α = 7/8
  - Use βRTT for retransmission timer
  - Initial values of β were 2 – make is proportional to standard deviation of M

$$RTT = \alpha RTT + (1 - \alpha) M$$

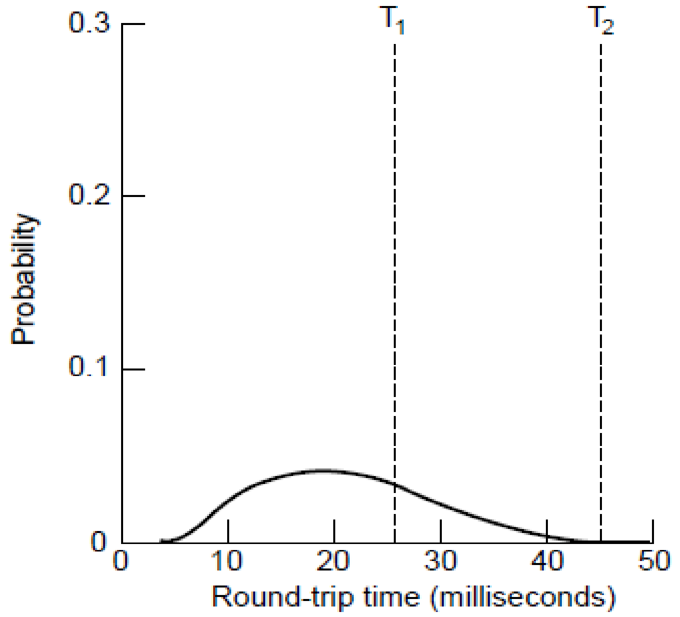
TCP Timer Management

TCP estimates retransmit timer from segment RTTs

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



LAN case – small, regular RTT



Internet case – large, varied RTT

Timer Management

- Jacobson Approach
- Mean deviation estimate
- $D = \gamma D + (1-\gamma)\,|RTT - M|$
- Timeout = RTT + 4 D

- What to do on retransmissions
  - Do not know if the ack is for the first or second
- Karn Algorithm
  - Do not update RTT on any segments that have been retransmitted

Persistence Timer

- Receiver sends a window of 0
- Later sends a window size but that packet is lost
  - Both wait
- Persistence Timer
  - When it goes off – sender sends a probe request to receiver to get a window size
  - If still zero – continue to wait and reset persistence timer
- Keepalive Timer
  - When a connection is idle for a long time – check if the other side is still there

TCP Congestion Control

- Congestion – a function of total number of packets in the network, and where they are
- First step – detection
  - Is packet loss an indication of congestion??
  - All TCP algorithms assume timeouts are caused by congestion
- Initial steps
  - When connection is established – use suitable window size
    - Loss will not occur due to buffers at receiver
- Two issues
  - Network Capacity
  - Receiver Capacity

TCP Congestion Control

- Network Capacity and Receiver Capacity
- Maintain two windows
  - Receiver window
  - Congestion window
  - Use the min (Receiver window and Congestion window)
- Initially
  - Sender sets congestion window to MSS (Max Seg Size)
  - If acked add one more MSS – 2 now
  - Repeat for each acked MSS
  - Congestion window grows exponentially
  - If timeout – go back to previous window size
  - SLOW START

Internet Congestion Control

- Use a Threshold – initially 64 KB
- When a timeout occurs set threshold to half the current congestion window and reset congestion window to 1 MSS
- Use slow start till the threshold is reached
- Then successful transmissions grow congestion window linearly

TCP Congestion Control (1)

TCP uses AIMD with loss signal to control congestion

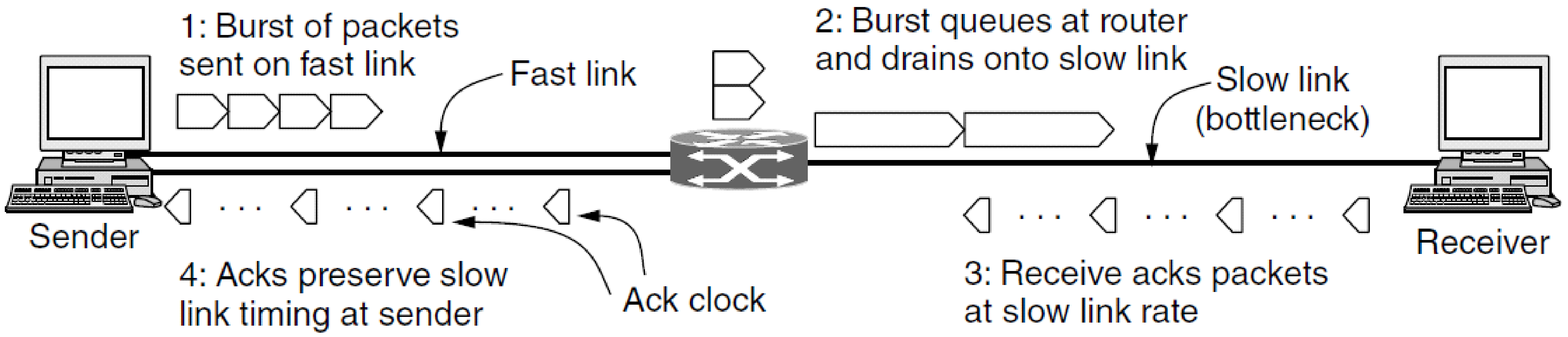
- Implemented as a congestion window (cwnd) for the number of segments that may be in the network
- Uses several mechanisms that work together

Name	Mechanism	Purpose
ACK clock	Congestion window (cwnd)	Smooth out packet bursts
Slow-start	Double cwnd each RTT	Rapidly increase send rate to reach roughly the right level
Additive Increase	Increase cwnd by 1 packet each RTT	Slowly increase send rate to probe at about the right level
Fast retransmit / recovery	Resend lost packet after 3 duplicate ACKs; send new packet for each new ACK	Recover from a lost packet without stopping ACK clock

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

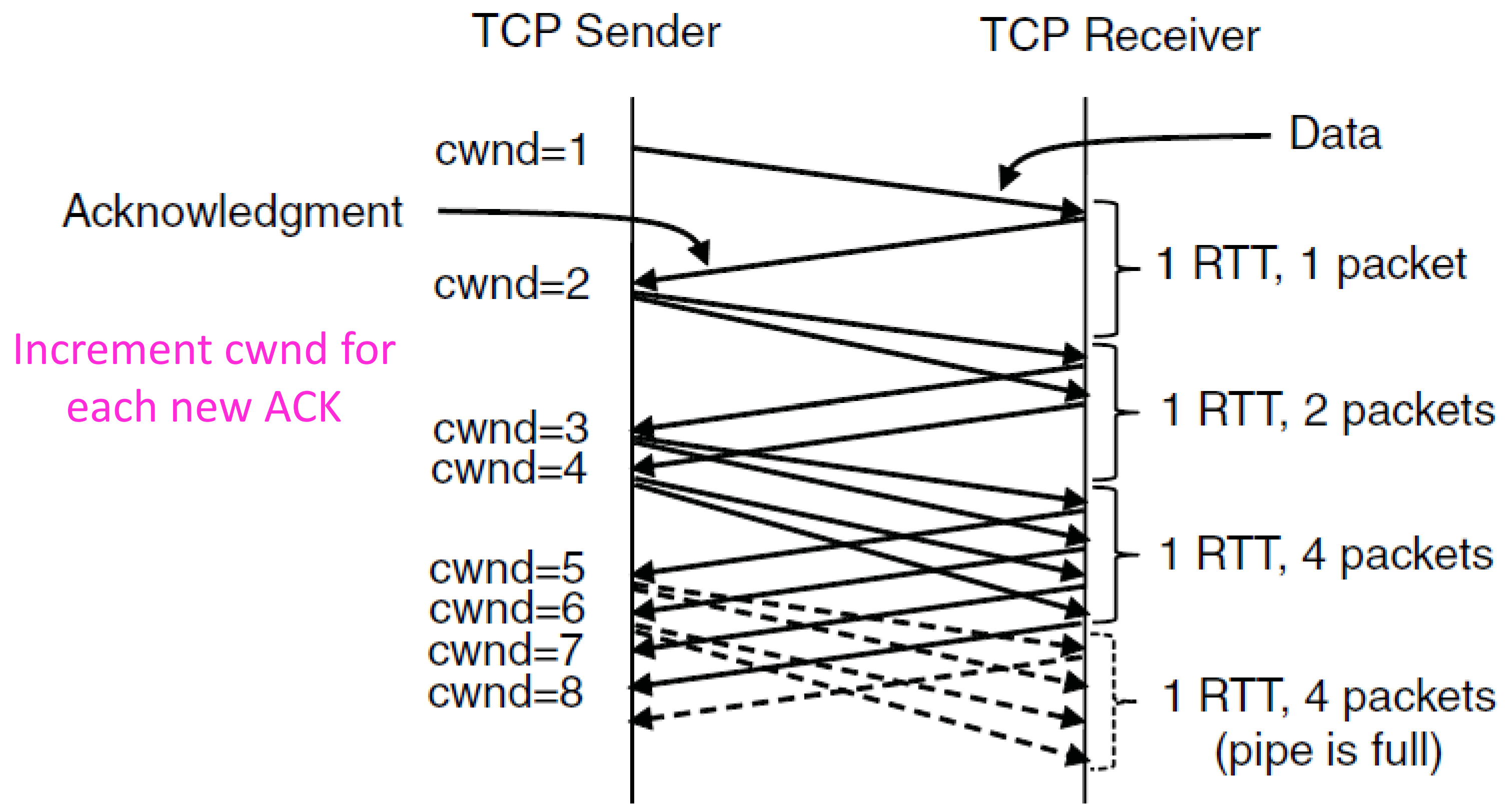

1: Burst of packets sent on fast link
Fast link
2: Burst queues at router and drains onto slow link
Slow link (bottleneck)
Sender
Receiver
4: Acks preserve slow link timing at sender
Ack clock
3: Receive acks packets at slow link rate

ACKs pace new segments into the network and smooth bursts

TCP Congestion Control (3)

Slow start grows congestion window exponentially

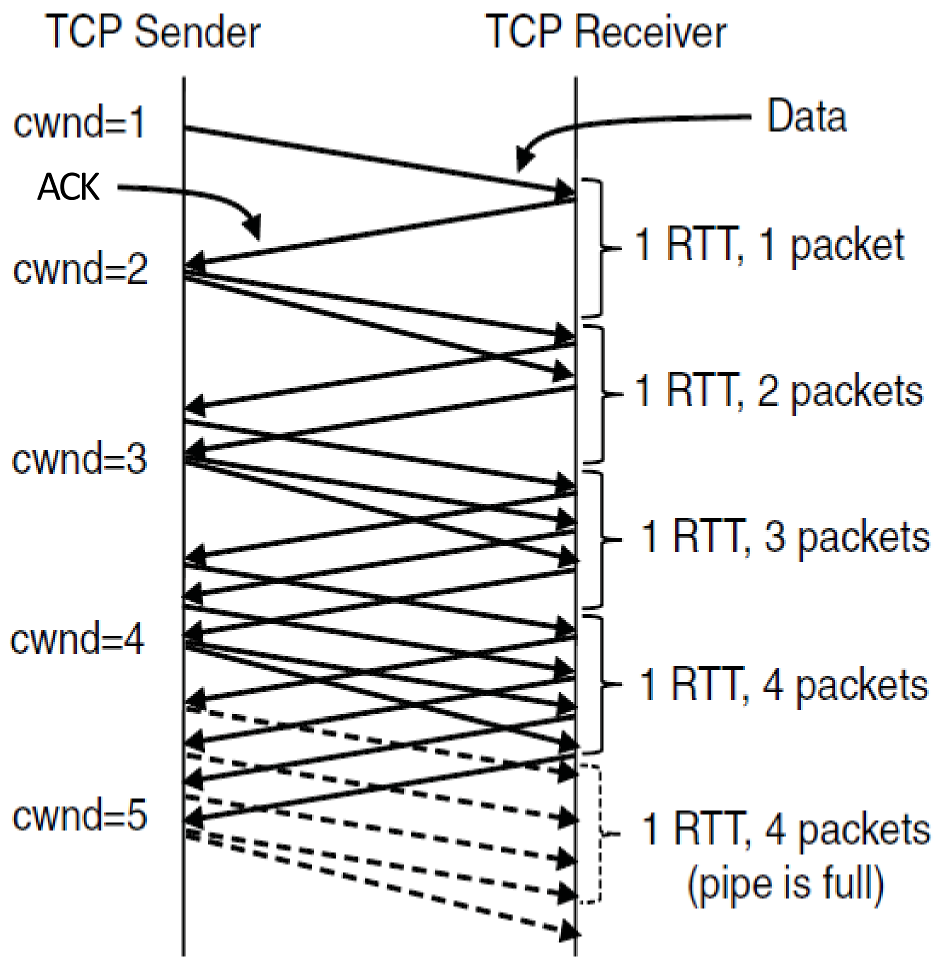
- Doubles every RTT while keeping ACK clock going



TCP Congestion Control (4)

Additive increase grows cwnd slowly

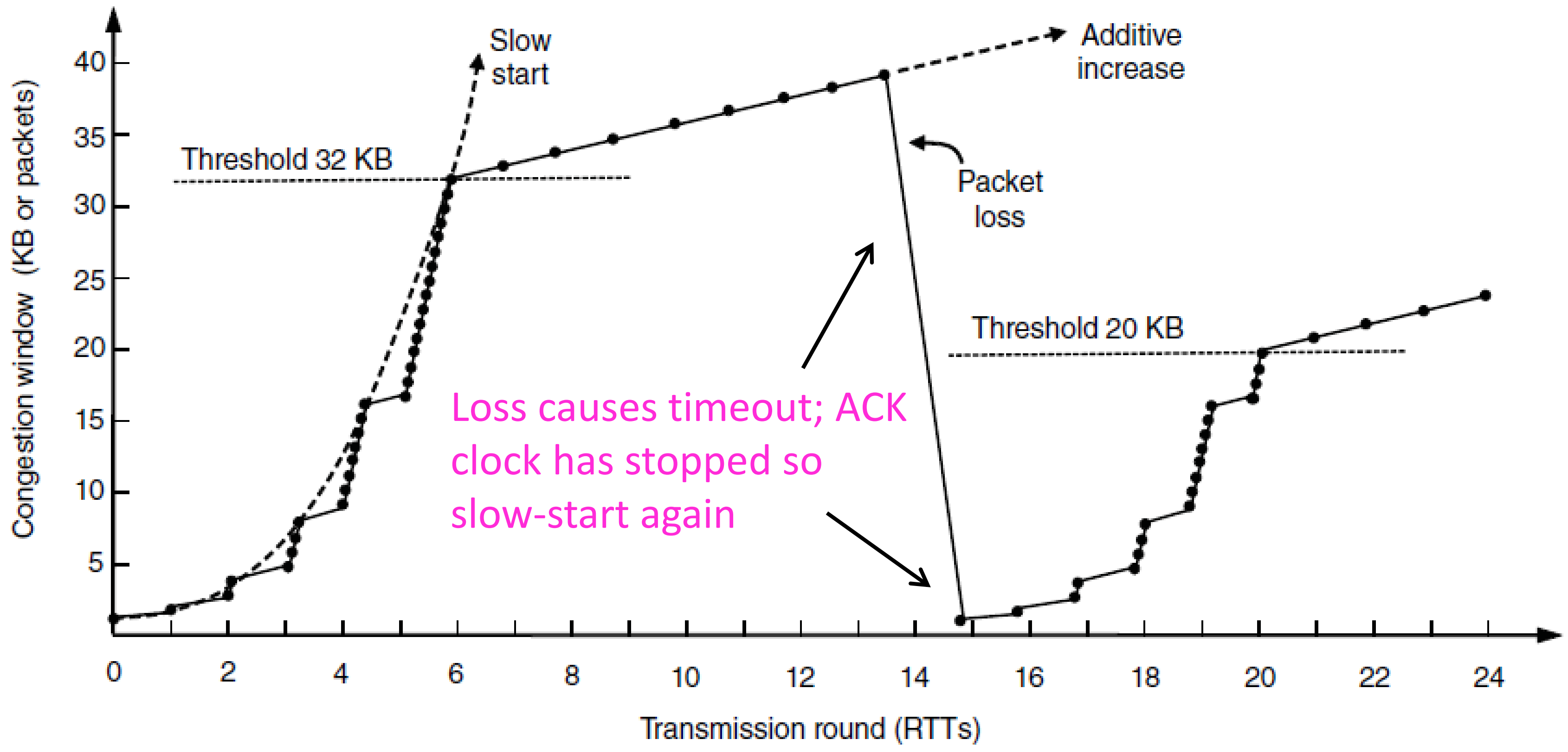
- Adds 1 every RTT
- Keeps ACK clock



TCP Congestion Control (5)

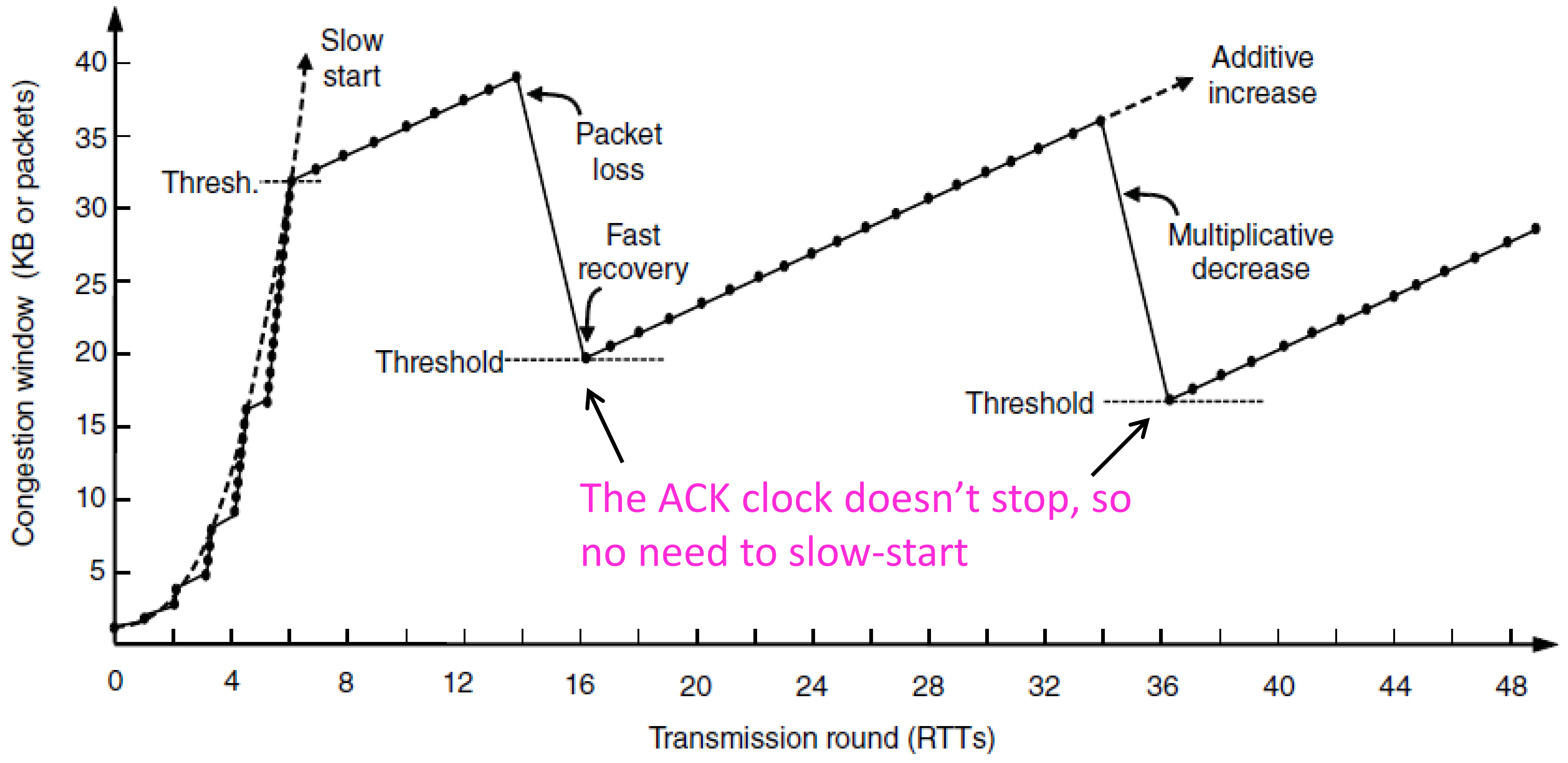
Slow start followed by additive increase (TCP Tahoe)

- Threshold is half of previous loss cwnd



TCP Congestion Control (6)

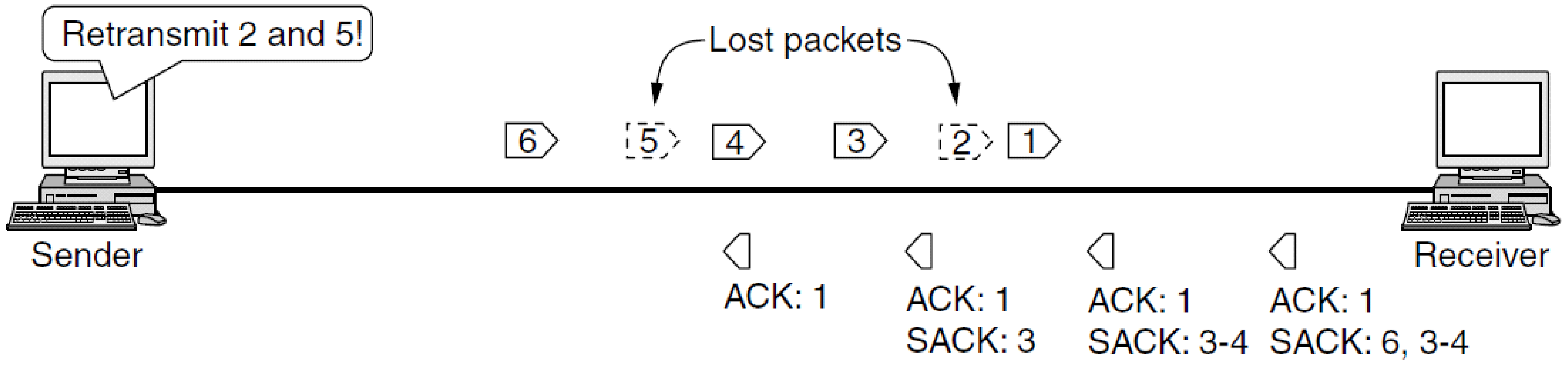
- With fast recovery, we get the classic sawtooth (TCP Reno)
  - Retransmit lost packet after 3 duplicate ACKs
  - New packet for each dup. ACK until loss is repaired



TCP Congestion Control (7)

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

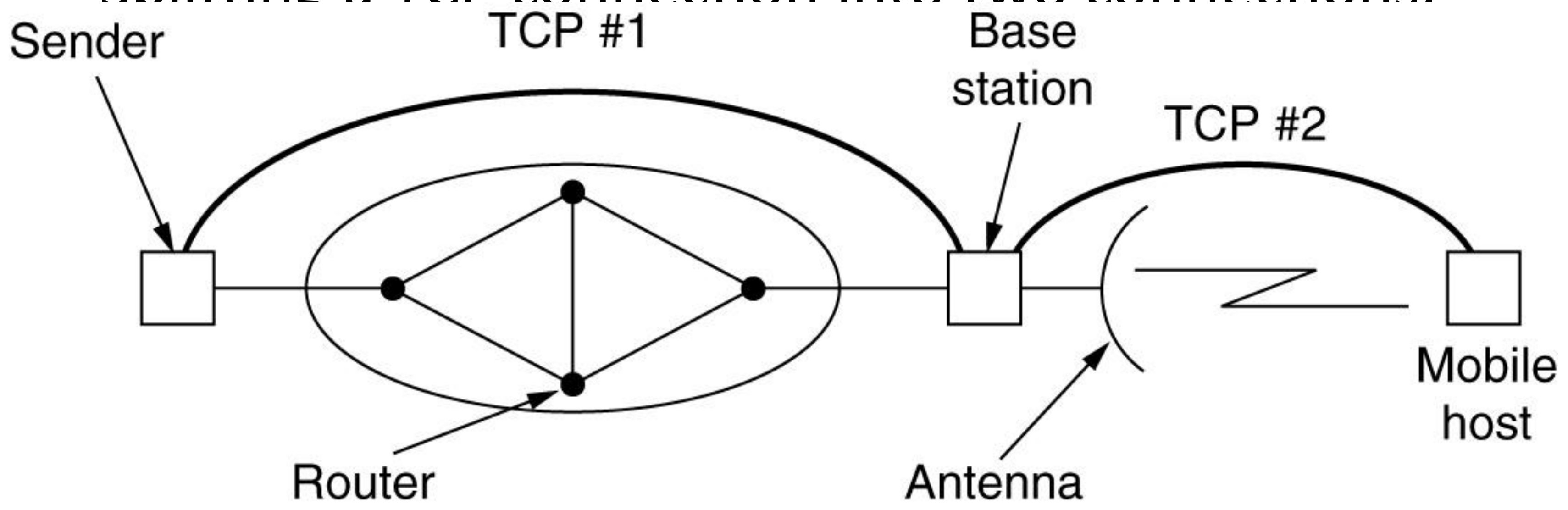
- Allows for more accurate retransmissions / recovery



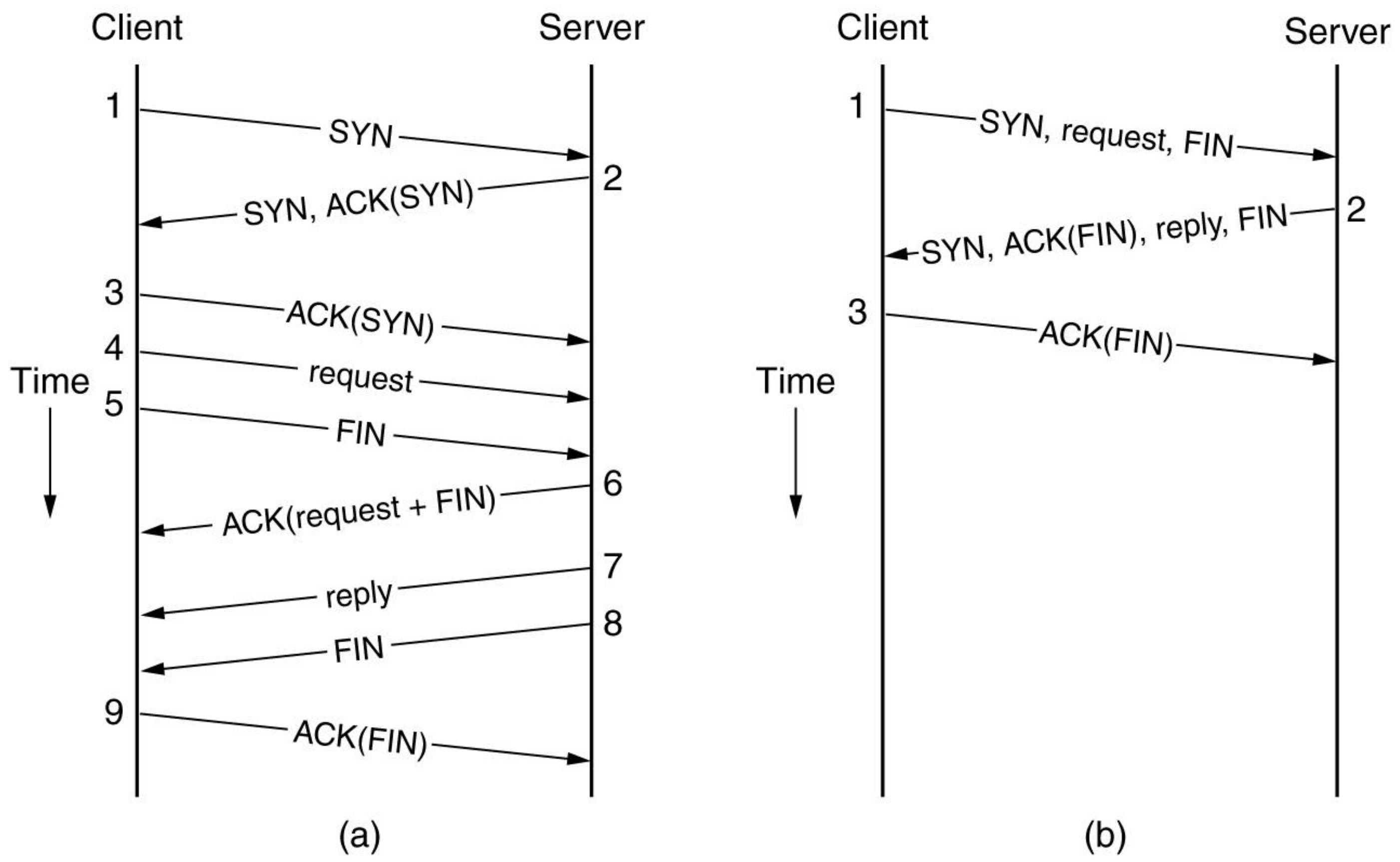
No way for us to know that 2 and 5 were lost with only ACKs

Wireless TCP and UDP

Splitting a TCP connection into two connections.



Transactional TCP



(a) RPC using normal TPC.

(b) RPC using T/TCP.

Performance Issues

Many strategies for getting good performance have been learned over time

- Performance problems »
- Measuring network performance »
- Host design for fast networks »
- Fast segment processing »
- Header compression »
- Protocols for “long fat” networks »

Performance Problems

Unexpected loads often interact with protocols to cause performance problems

- Need to find the situations and improve the protocols

Examples:

- Broadcast storm: one broadcast triggers another
- Synchronization: a building of computers all contact the DHCP server together after a power failure
- Tiny packets: some situations can cause TCP to send many small packets instead of few large ones

Host Design for Fast Networks

Poor host software can greatly slow down networks.

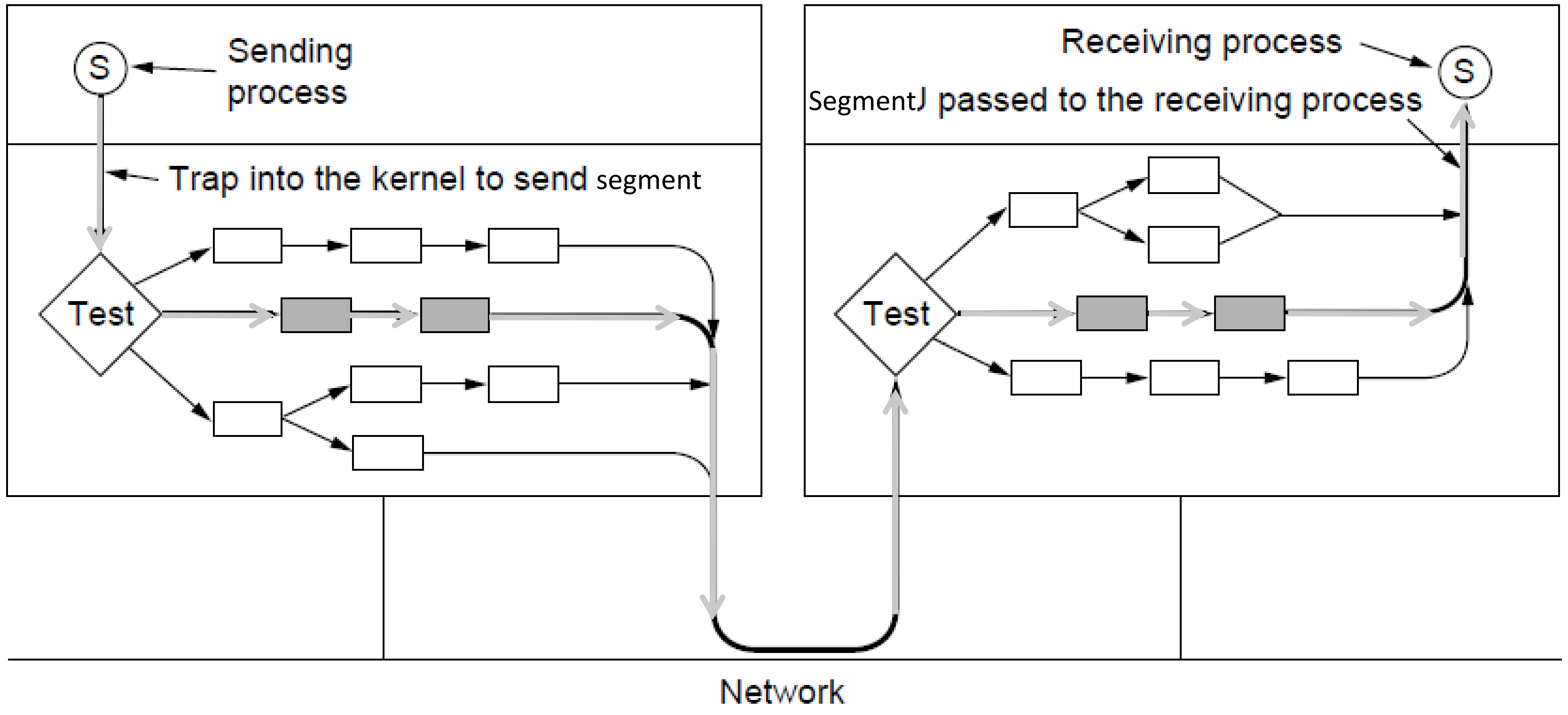
Rules of thumb for fast host software:

- Host speed more important than network speed
- Reduce packet count to reduce overhead
- Minimize data touching
- Minimize context switches
- Avoiding congestion is better than recovering from it
- Avoid timeouts

Fast Segment Processing (1)

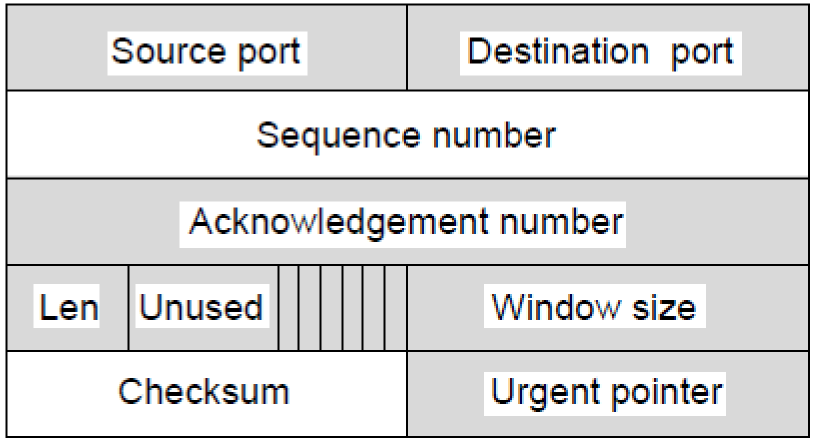
Speed up the common case with a fast path [pink]

- Handles packets with expected header; OK for others to run slowly

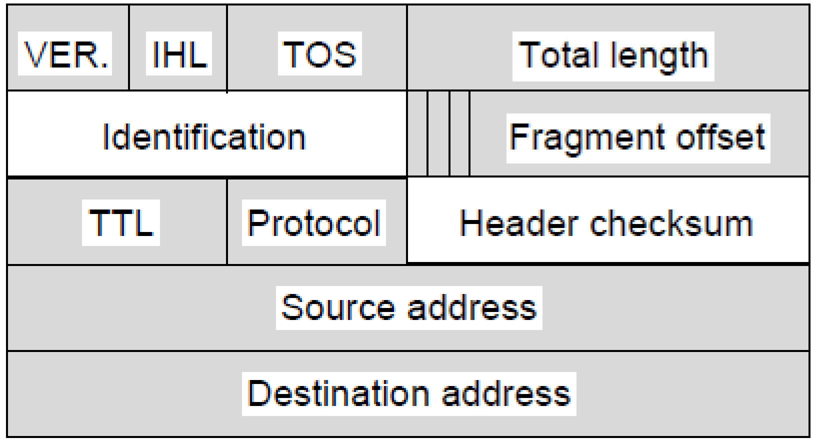

Sending process
Trap into the kernel to send segment
Test
Receiving process
SegmentJ passed to the receiving process
Test
S
S
Network

Fast Segment Processing (2)

Header fields are often the same from one packet to the next for a flow; copy/check them to speed up processing

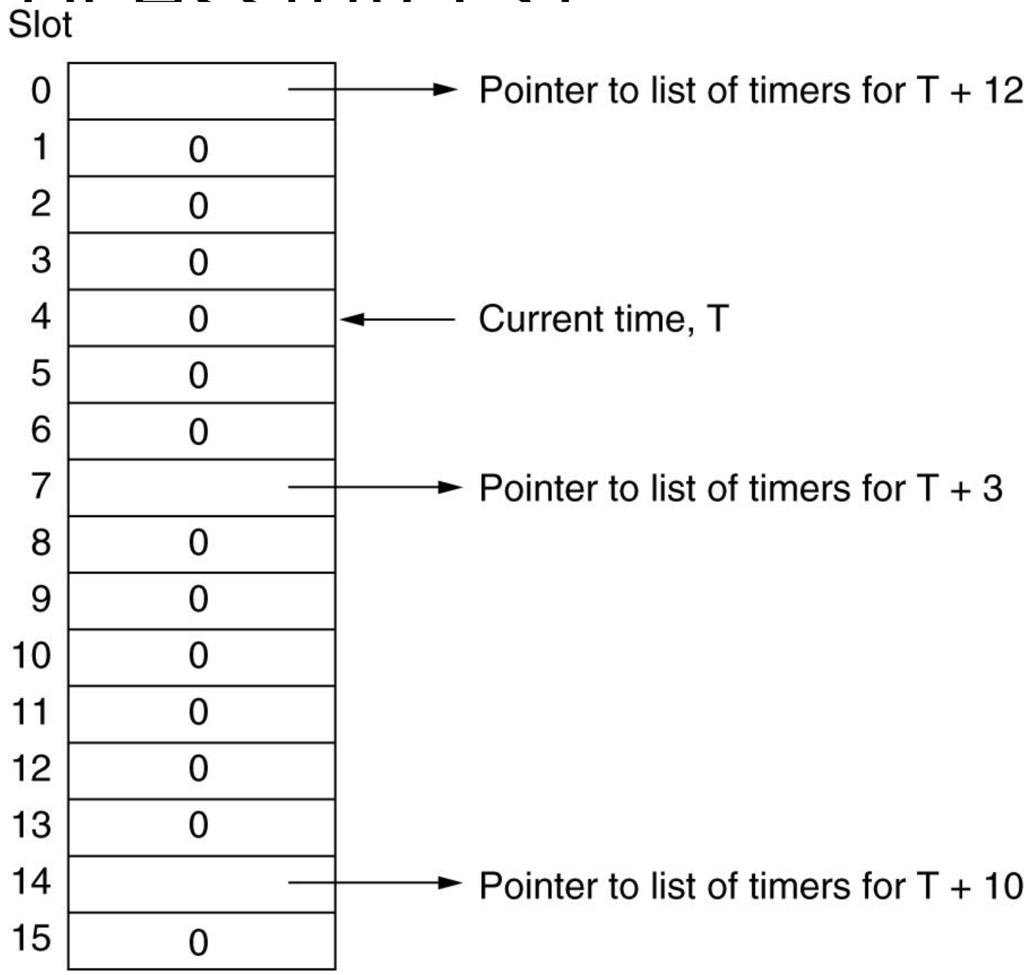


TCP header fields that stay the same for a one-way flow (shaded)



IP header fields that are often the same for a one-way flow (shaded)

Fast TPDU Processing (3)


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

A timing wheel.

Header Compression

Overhead can be very large for small packets

- 40 bytes of header for RTP/UDP/IP VoIP packet
- Problematic for slow links, especially wireless

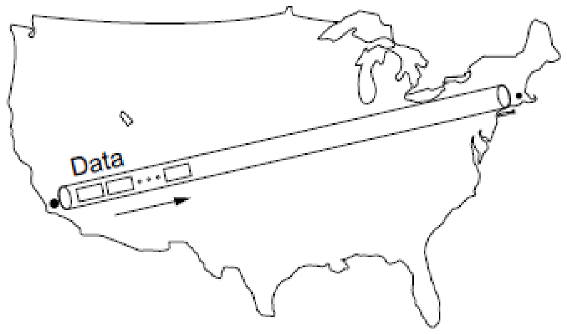
Header compression mitigates this problem

- Runs between Link and Network layer
- Omits fields that don't change or change predictably
  - 40 byte TCP/IP header → 3 bytes of information
- Gives simple high-layer headers and efficient links

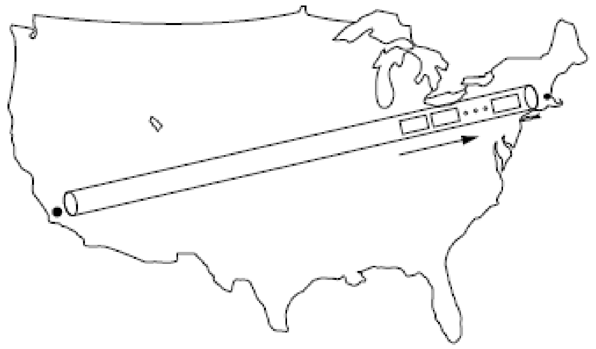
Protocols for "Long Fat" Networks (1)

Networks with high bandwidth ("Fat") and high delay ("Long") can store much information inside the network

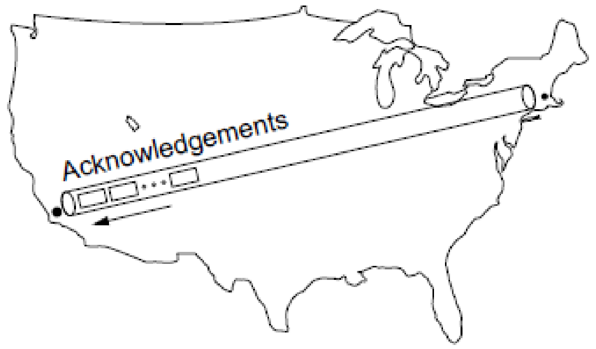
- Requires protocols with ample buffering and few RTTs, rather than reducing the bits on the wire



Starting to send 1 Mbit San Diego → Boston



20ms after start

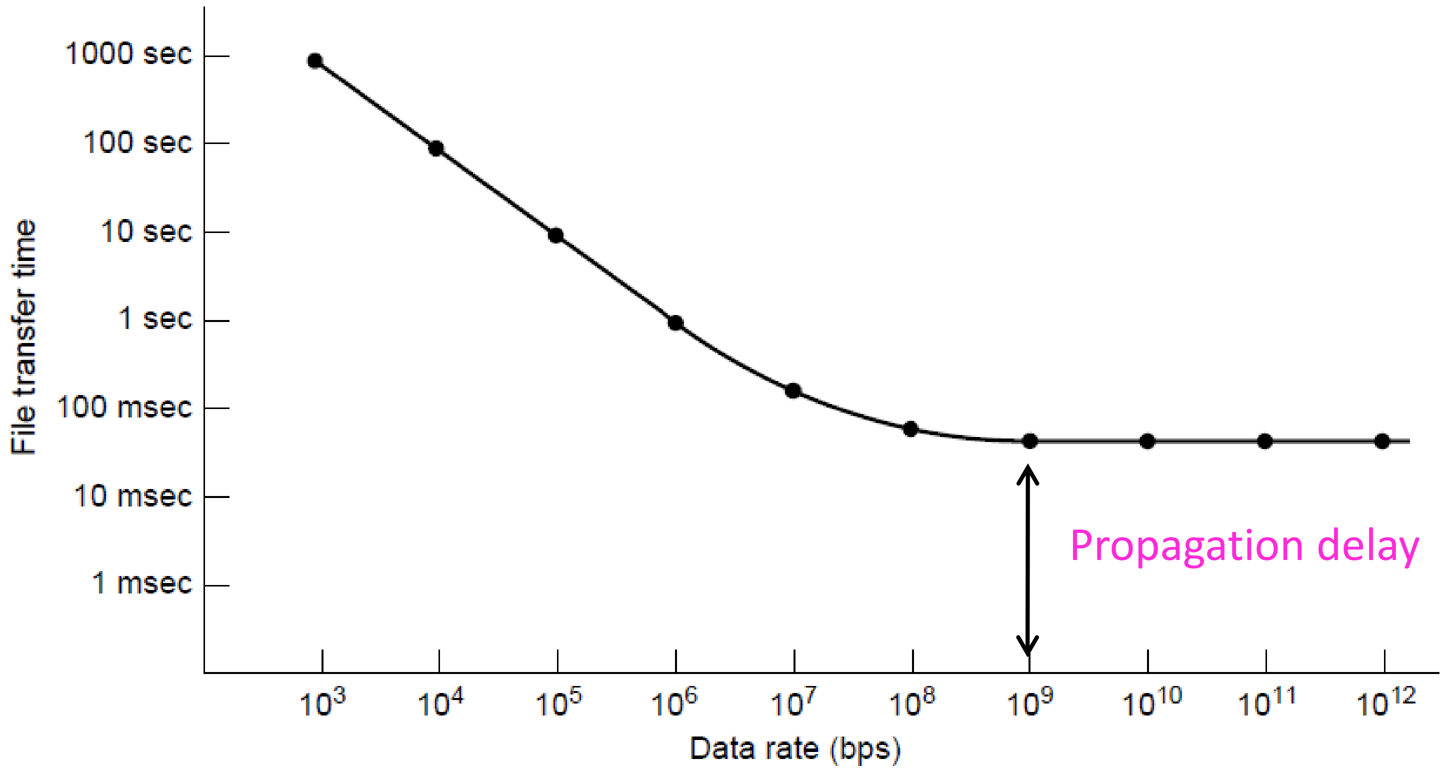


40ms after start

Protocols for “Long Fat” Networks (2)

You can buy more bandwidth but not lower delay

- Need to shift ends (e.g., into cloud) to lower further



Minimum time to send and ACK a 1-Mbit file over a 4000-km line

Delay Tolerant Networking

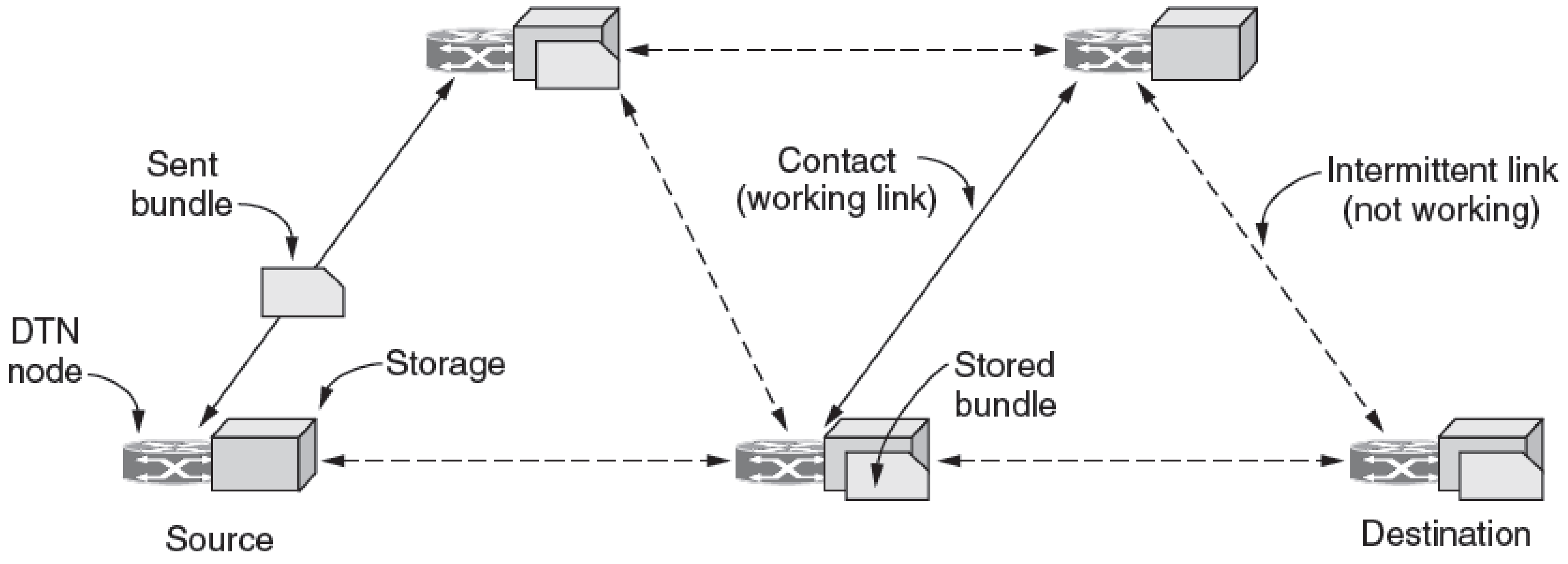
DTNs (Delay Tolerant Networks) store messages inside the network until they can be delivered

- DTN Architecture »
- Bundle Protocol »

DTN Architecture (1)

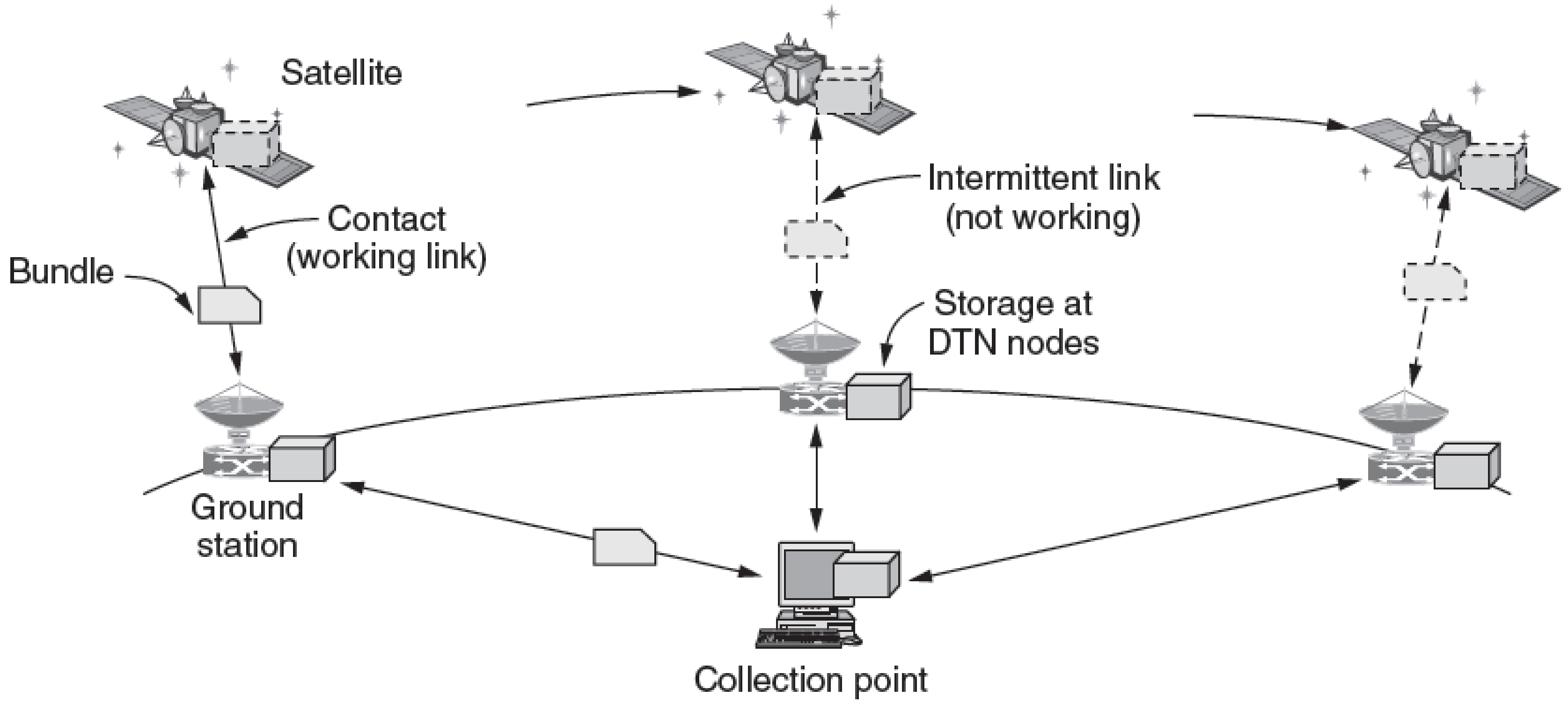
Messages called bundles are stored at DTN nodes while waiting for an intermittent link to become a contact

- Bundles might wait hours, not milliseconds in routers
- May be no working end-to-end path at any time



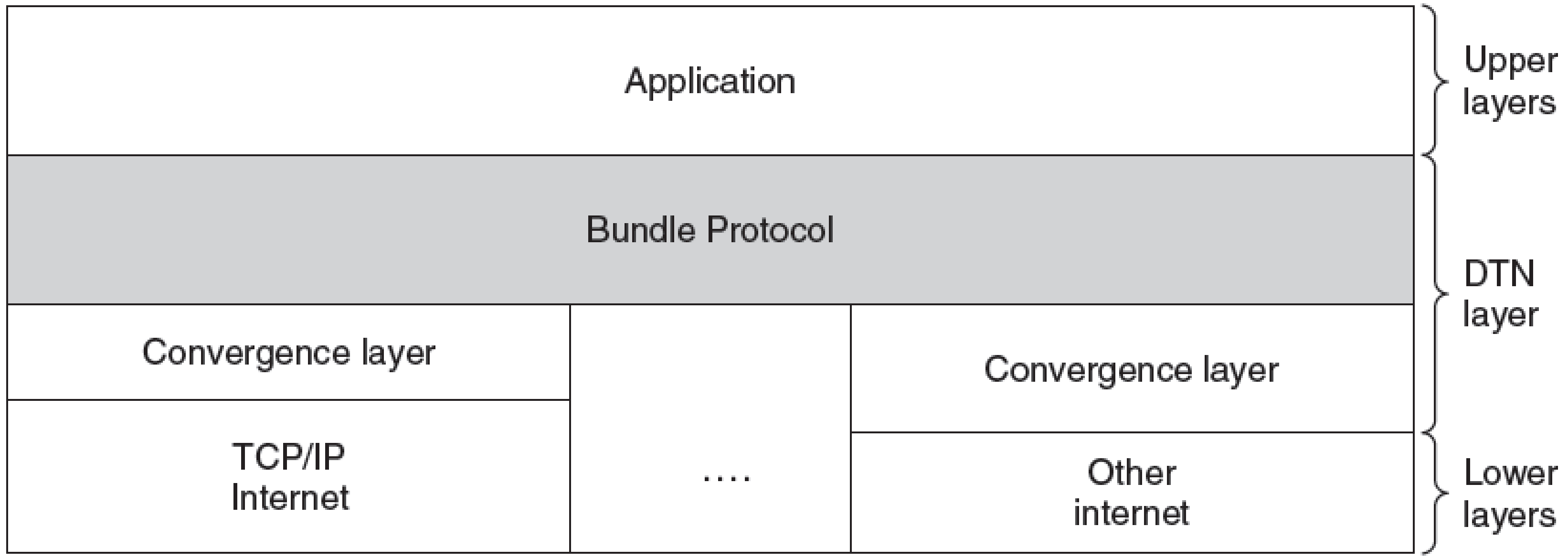
DTN Architecture (2)

Example DTN connecting a satellite to a collection point



Bundle Protocol (1)

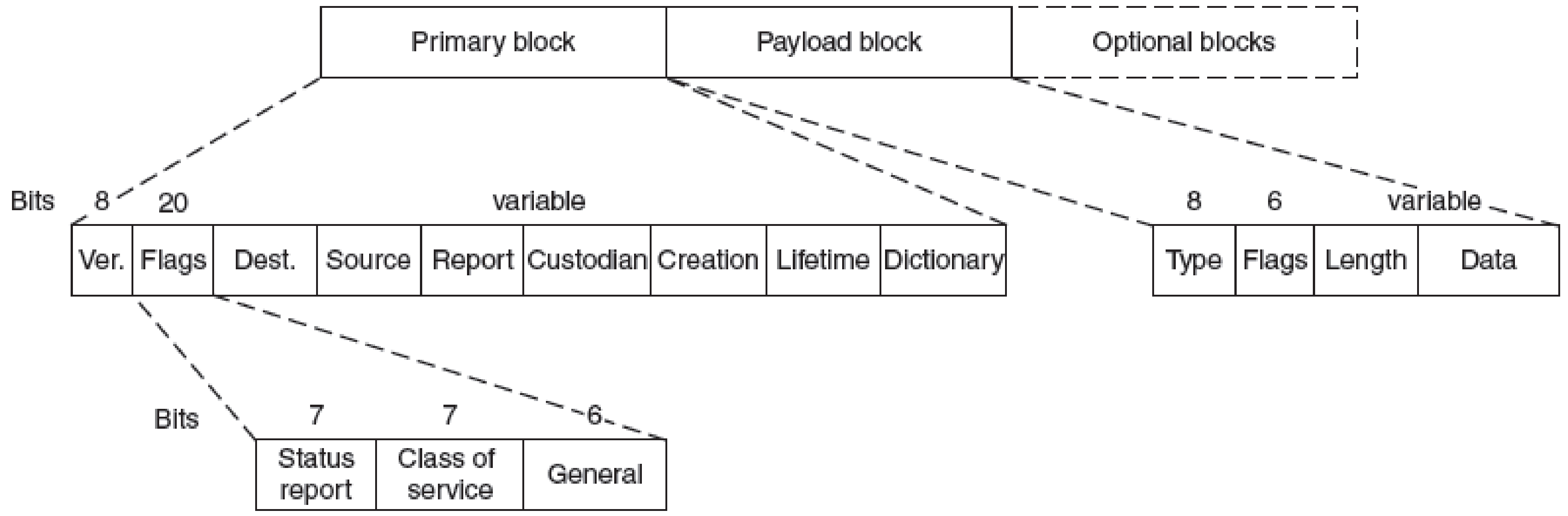
The Bundle protocol uses TCP or other transports and provides a DTN service to applications



Bundle Protocol (2)

Features of the bundle message format:

- Dest./source add high-level addresses (not port/IP)
- Custody transfer shifts delivery responsibility
- Dictionary provides compression for efficiency


Primary block
Payload block
Optional blocks
Bits
8
20
variable
Ver.
Flags
Dest.
Source
Report
Custodian
Creation
Lifetime
Dictionary
8
6
variable
Type
Flags
Length
Data
Bits
7
7
6
Status report
Class of service
General