Computer Communication Networks

Chapter 6: Transport Layer

Prof. Xudong Wang

Wireless Networking and Artificial Intelligence Lab
UM-SJTU Joint Institute
Shanghai Jiao Tong University
Shanghai, China

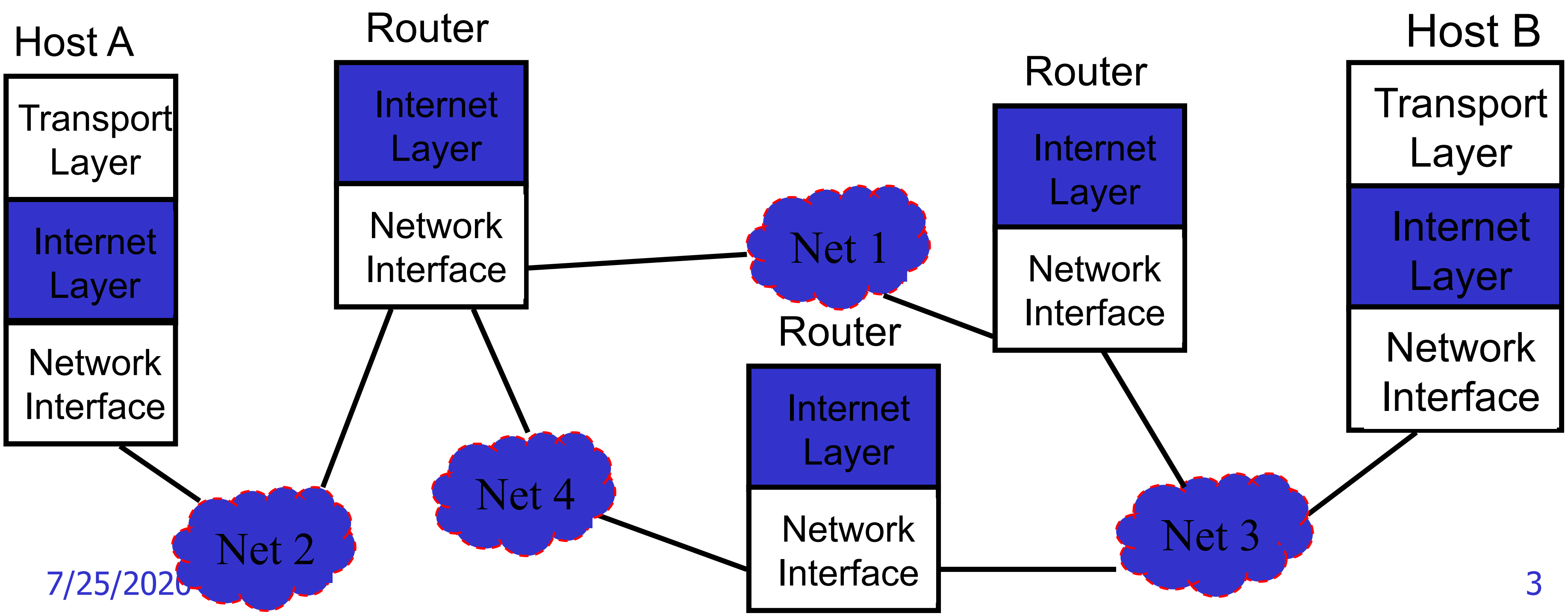
http://wanglab.sjtu.edu.cn

Outline

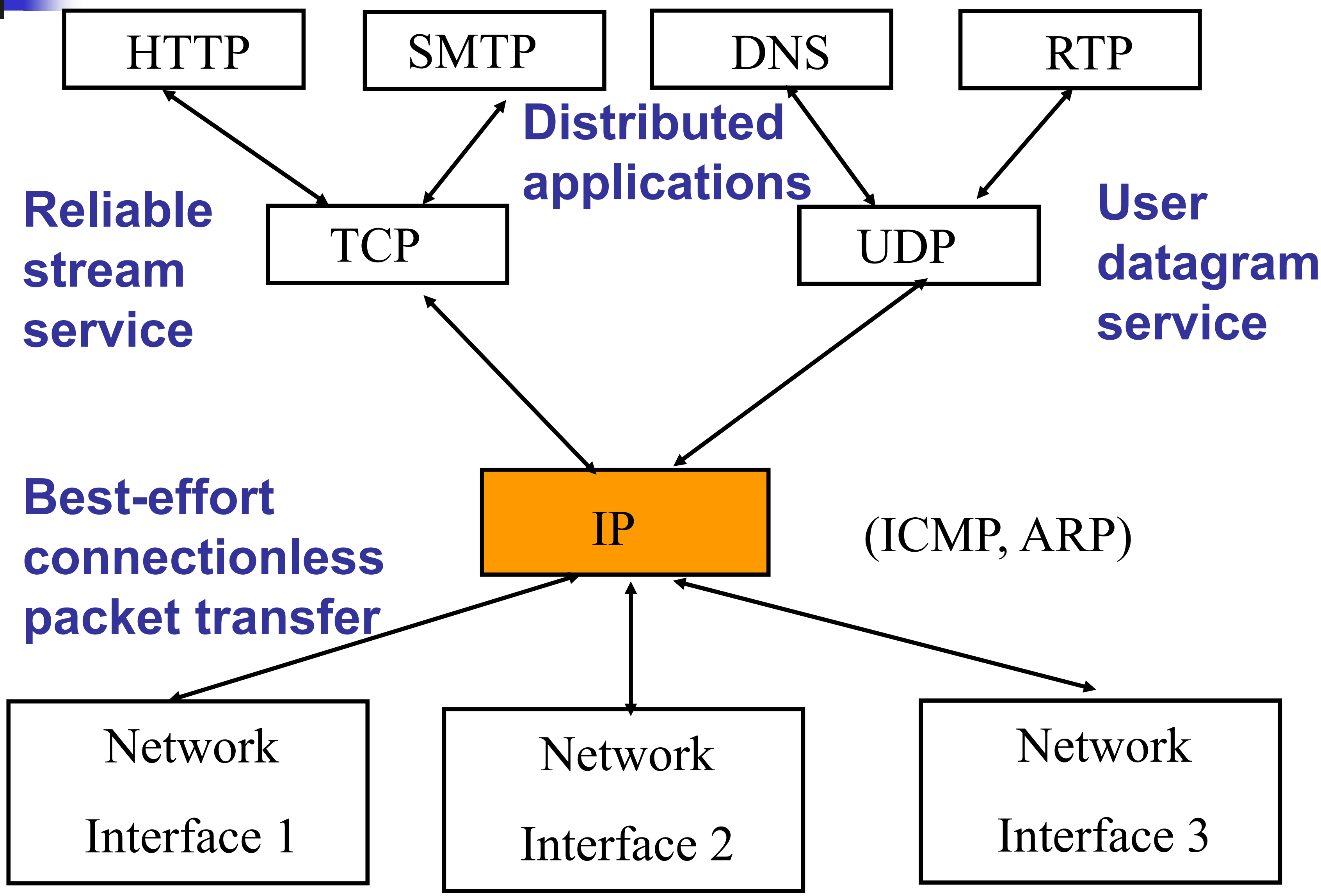
- UDP Protocol
- TCP Reliable Stream Service
- TCP Protocol
- TCP Connection Management
- TCP Flow Control
- TCP Congestion Control

Internet Protocol Approach

- IP packets transfer information across Internet
  Host A IP → router→ router…→ router→ Host B IP
- IP layer in each router determines next hop (router)
- Network interfaces transfer IP packets across networks



TCP/IP Protocol Suite



Diverse network technologies

UDP

- Best effort datagram service
- Multiplexing enables sharing of IP datagram service
- Simple transmitter & receiver
  - Connectionless: no handshaking & no connection state
  - Low header overhead
  - No flow control, no error control, no congestion control
  - UDP datagrams can be lost or out-of-order
- Applications
  - multimedia (e.g. RTP)
  - network services (e.g. DNS, RIP, SNMP)

UDP Datagram

0	16	31
Source Port	Destination Port	
UDP Length	UDP Checksum	
Data		

0-255
- Well-known ports

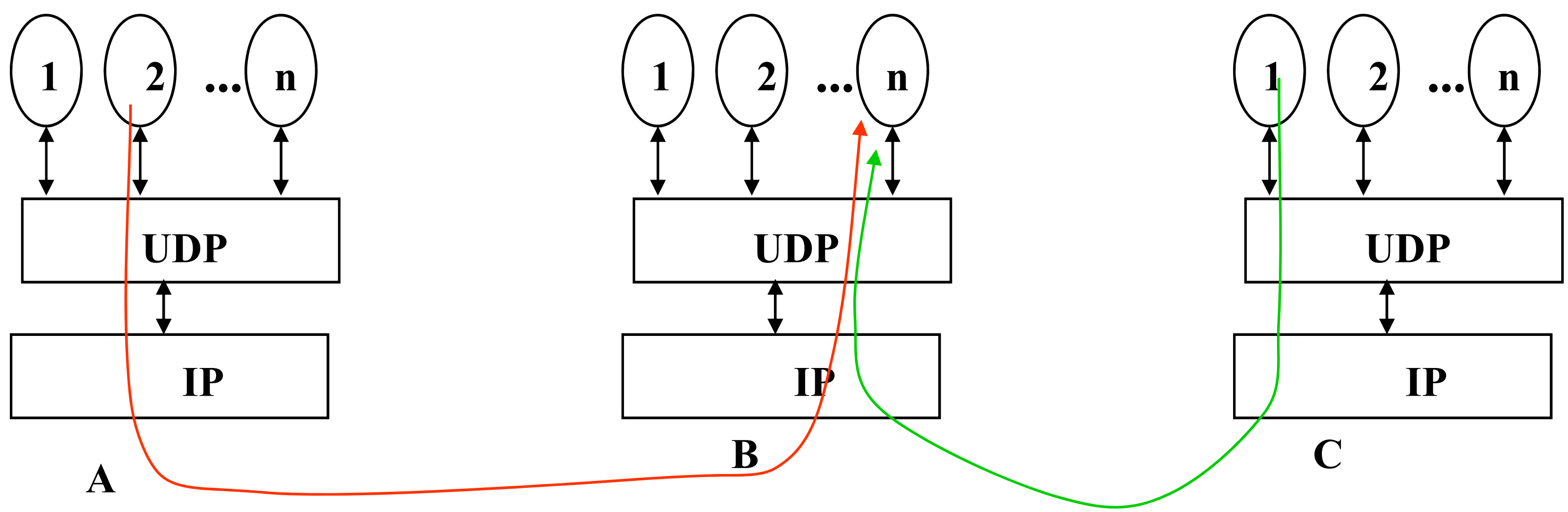
256-1023
- Less well-known ports

1024-65536
- Ephemeral client ports

- Source and destination port numbers
  - Client ports are ephemeral
  - Server ports are well-known
  - Max number is 65,535
- UDP length
  - Total number of bytes in datagram (including header)
  - 8 bytes $\leq$ length $\leq$ 65,535
- UDP Checksum
  - Optionally detects errors in UDP datagram

UDP Multiplexing

- All UDP datagrams arriving to IP address B and destination port number *n* are delivered to the same process
- Source port number is not used in multiplexing



UDP Checksum Calculation

0	8	16 … 31
Source IP Address		
Destination IP Address		
0 0 0 0 0 0 0 0	Protocol = 17	UDP Length

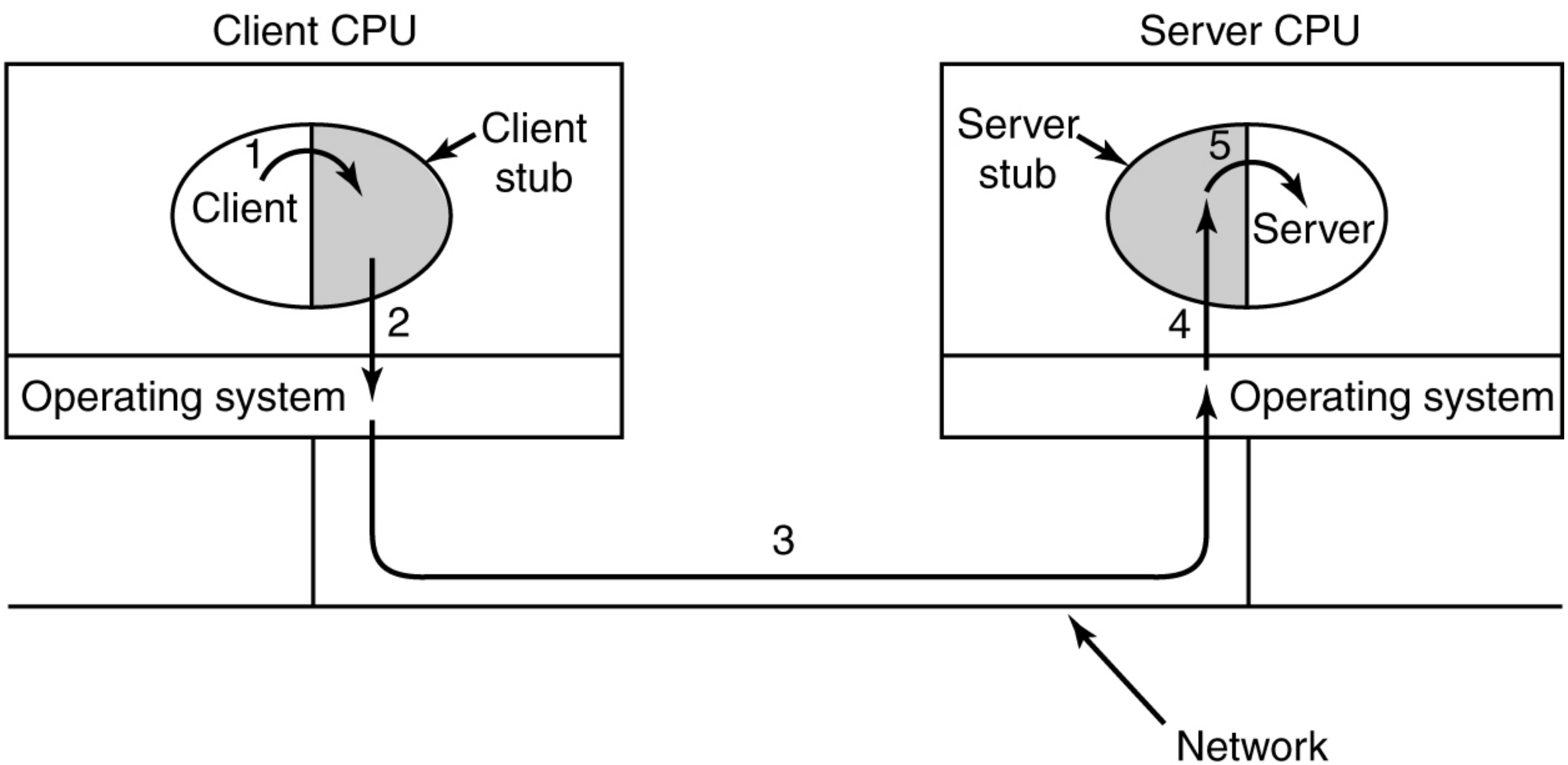
UDP pseudo-header

- *Only for checksum calculation; not transmitted*
- UDP checksum detects for end-to-end errors
- Covers pseudoheader followed by UDP datagram
- IP addresses included to detect against misdelivery
- UDP checksums set to zero during calculation
- Pad with 1 byte of zeros if UDP length is odd

UDP Receiver Checksum

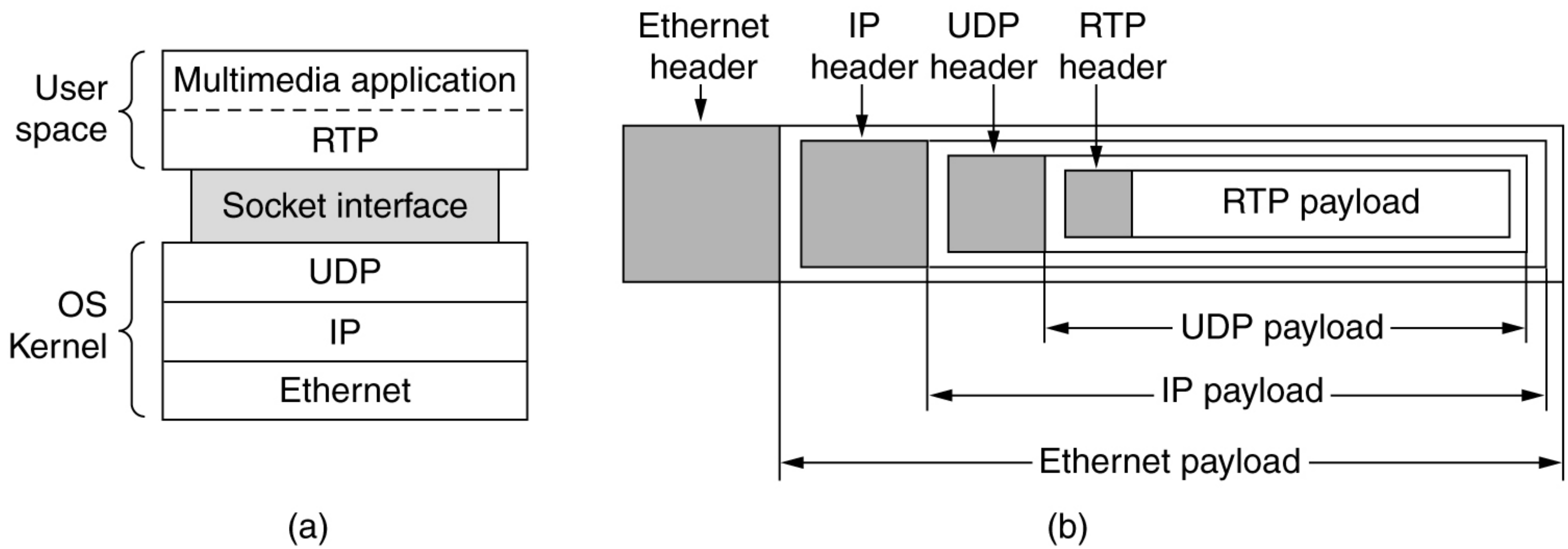
- UDP receiver recalculates the checksum and silently discards the datagram if errors detected
  - "silently" means no error message is generated
- The use of UDP checksums is optional
- But hosts are required to have checksums enabled

Remote Procedure Call



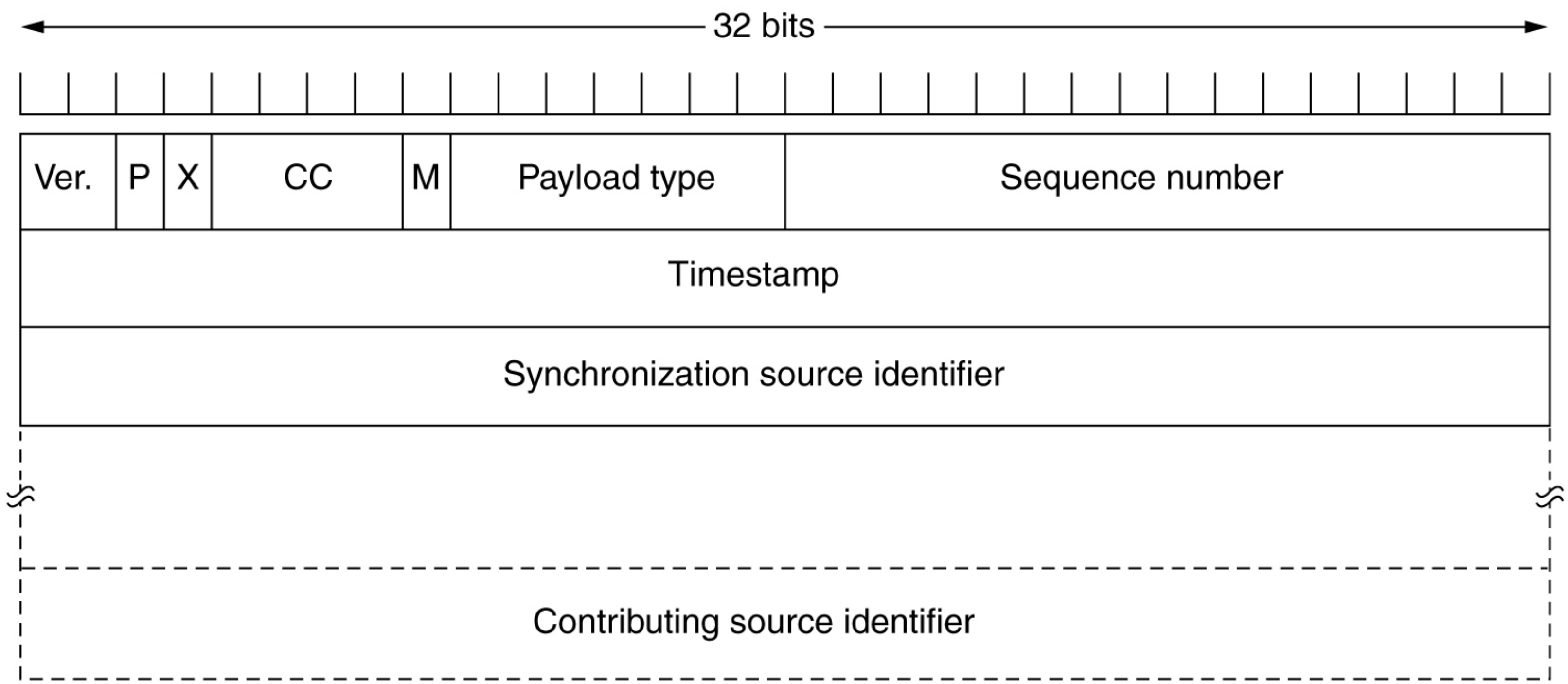
Steps in making a remote procedure call. The stubs are shaded.

The Real-Time Transport Protocol



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

The Real-Time Transport Protocol (2)



The RTP header.

Outline

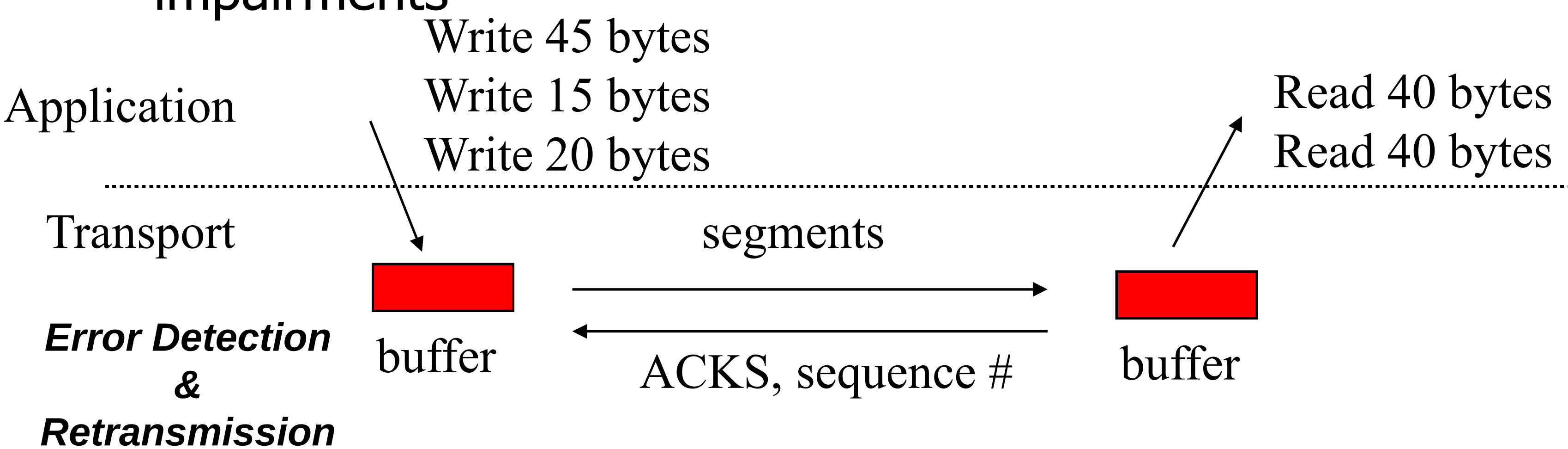
- UDP Protocol
- TCP Reliable Stream Service
- TCP Protocol
- TCP Connection Management
- TCP Congestion Control

TCP

- Reliable byte-stream service
- More complex transmitter & receiver
  - Connection-oriented: full-duplex unicast connection between client & server processes
  - Connection setup, monitor connection state, connection release
  - Higher header overhead
  - Error control, flow control, and congestion control
  - Higher delay than UDP
- Most applications use TCP
  - HTTP, SMTP, FTP, TELNET, POP3, …

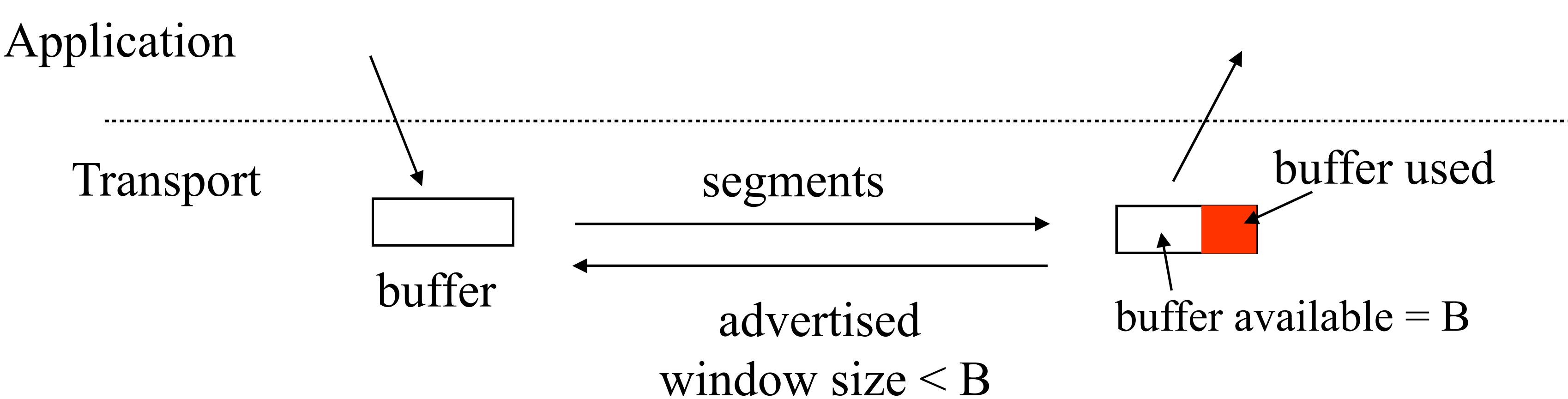
Reliable Byte-Stream Service

- Stream Data Transfer
  - transfers a contiguous stream of bytes across the network, with no indication of boundaries
  - groups bytes into segments
  - transmits segments as convenient (Push function defined)
- Reliability
  - error control mechanism to deal with IP transfer impairments



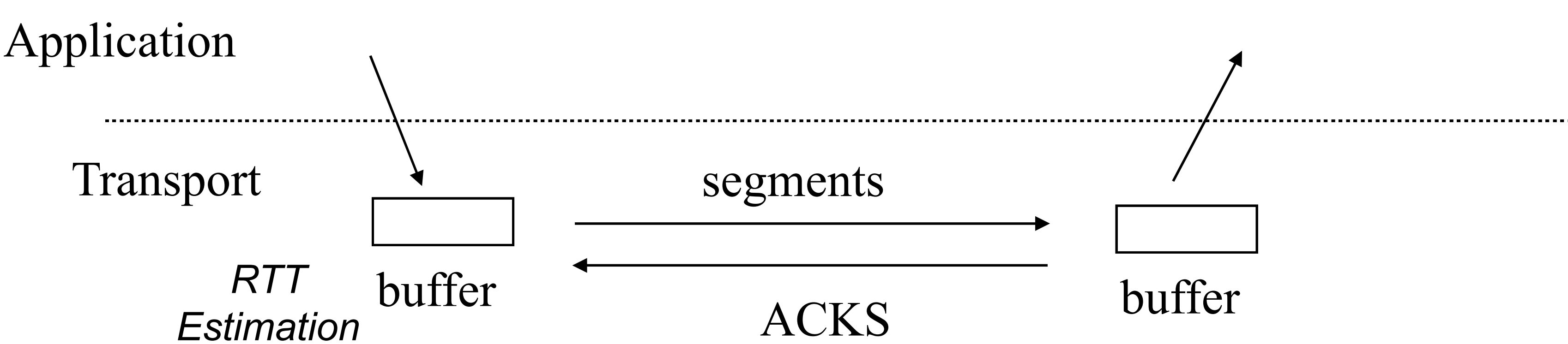
Flow Control

- Buffer limitations & speed mismatch can result in loss of data that arrives at destination
- Receiver controls rate at which sender transmits to prevent buffer overflow



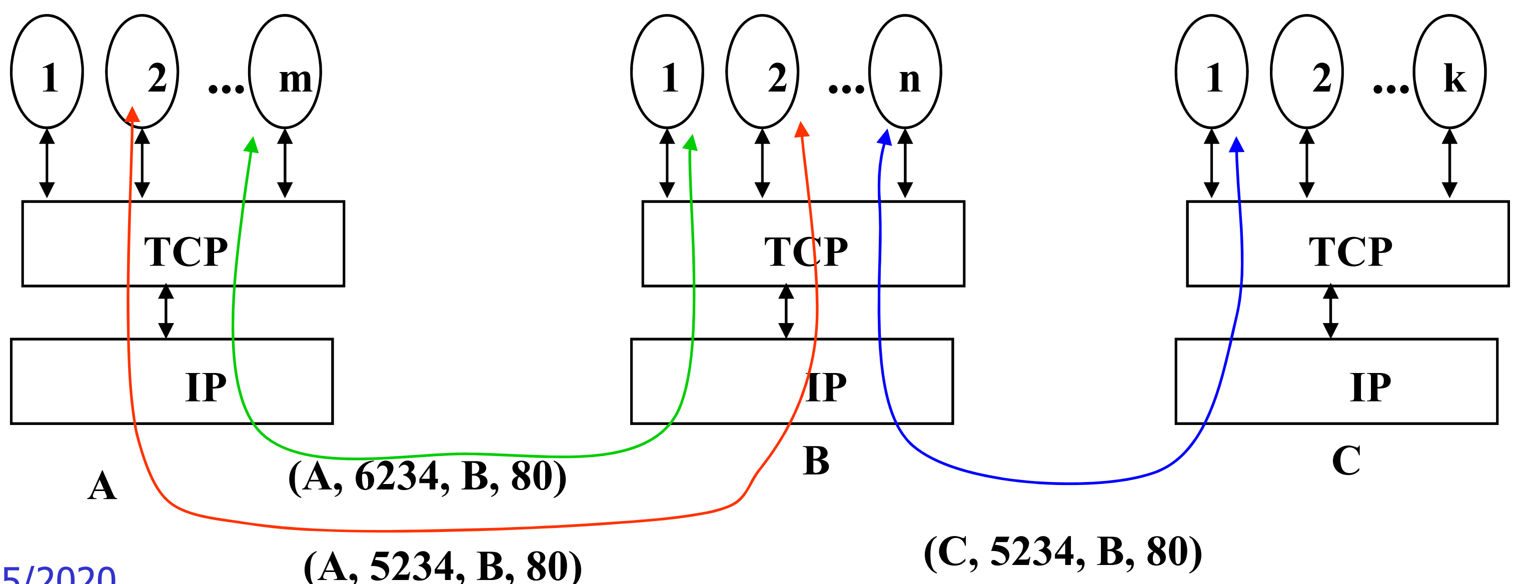
Congestion Control

- Available bandwidth to destination varies with activity of other users
- Transmitter dynamically adjusts transmission rate according to network congestion as indicated by RTT (round trip time) & ACKs
- Elastic utilization of network bandwidth



TCP Multiplexing

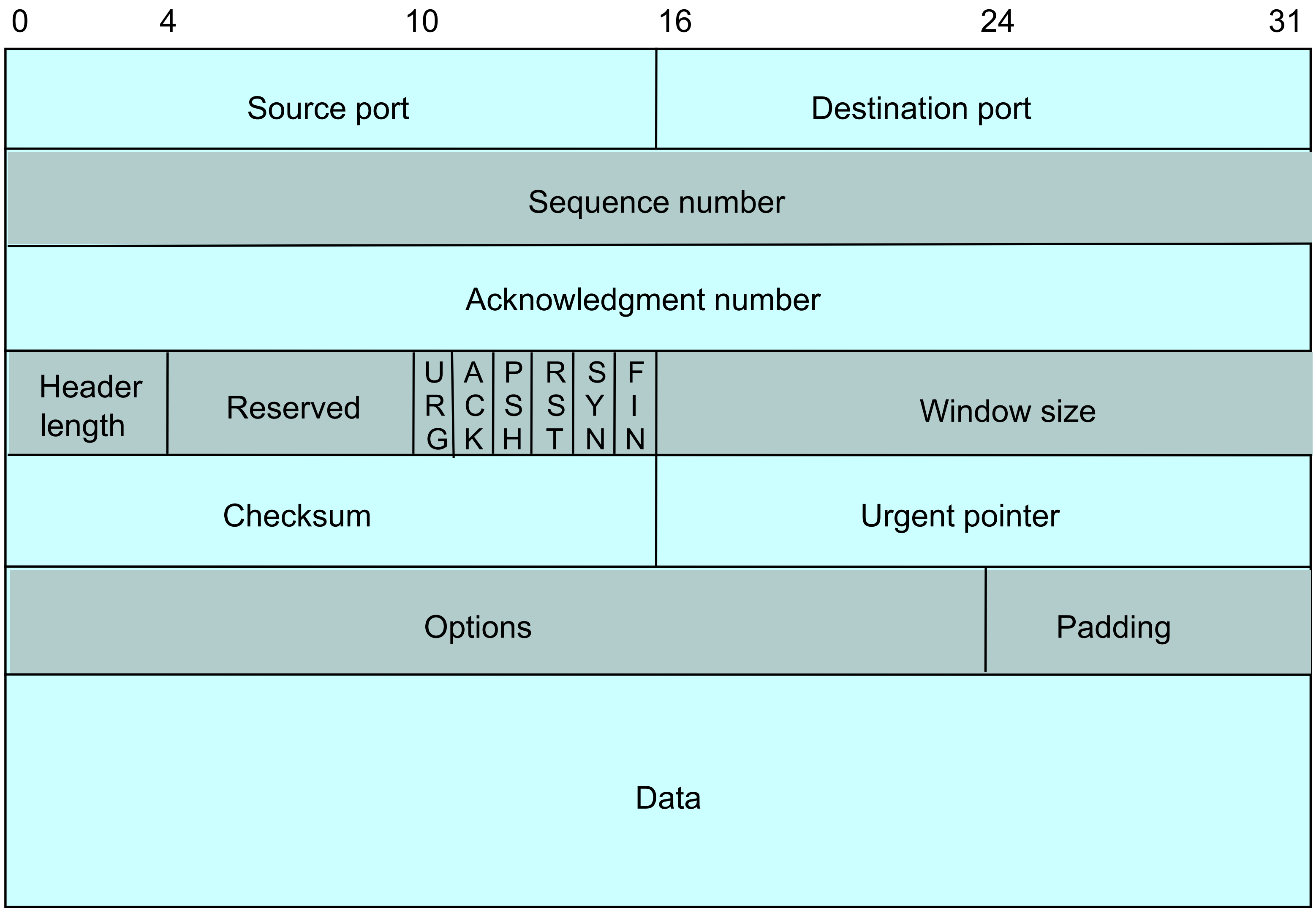
- A *TCP connection* is specified by a *4-tuple*
  - (source IP address, source port, destination IP address, destination port)
- TCP allows multiplexing of multiple connections between end systems to support multiple applications simultaneously
- Arriving segment directed according to connection 4-tuple



Outline

- UDP Protocol
- TCP Reliable Stream Service
- TCP Protocol
- TCP Connection Management
- TCP Congestion Control

TCP Segment Format



- Each TCP segment has header of 20 or more bytes + 0 or more bytes of data

TCP Header

Port Numbers

- A socket identifies a connection endpoint
  - IP address + port
- A connection specified by a *socket pair*
- Well-known ports
  - FTP 20
  - Telnet 23
  - DNS 53
  - HTTP 80

Sequence Number

- Byte count
- First byte in segment
- 32 bits long
- $0 \leq SN \leq 2^{32}-1$
- Initial sequence number selected during connection setup

TCP Header

Acknowledgement Number

- SN of next byte expected by receiver
- Acknowledges that all prior bytes in stream have been received correctly
- Valid if ACK flag is set

Header length

- 4 bits
- Length of header in multiples of 32-bit words
- Minimum header length is 20 bytes
- Maximum header length is 60 bytes

TCP Header

Reserved

- 6 bits

Control

- 6 bits
- URG: urgent pointer flag
  - Urgent message end = SN + **urgent pointer**
- ACK: ACK packet flag
- PSH: override TCP buffering
- RST: reset connection
  - Upon receipt of RST, connection is terminated and application layer notified
- SYN: establish connection
- FIN: close connection

TCP Header

Window Size

- 16 bits to advertise window size
- Used for flow control
- Sender will accept bytes with SN from ACK to ACK + window
- Maximum window size is 65535 bytes

TCP Checksum

- Internet checksum method
- TCP pseudoheader + TCP segment

TCP Checksum Calculation

0	8	16	31
Source IP address			
Destination IP address			
0 0 0 0 0 0 0 0	Protocol = 6	TCP segment length	

TCP pseudo-header

- TCP error detection uses same procedure as UDP

TCP Header

Options

- Variable length
- NOP (No Operation) option is used to pad TCP header to multiple of 32 bits
- Time stamp option is used for round trip measurements

Options

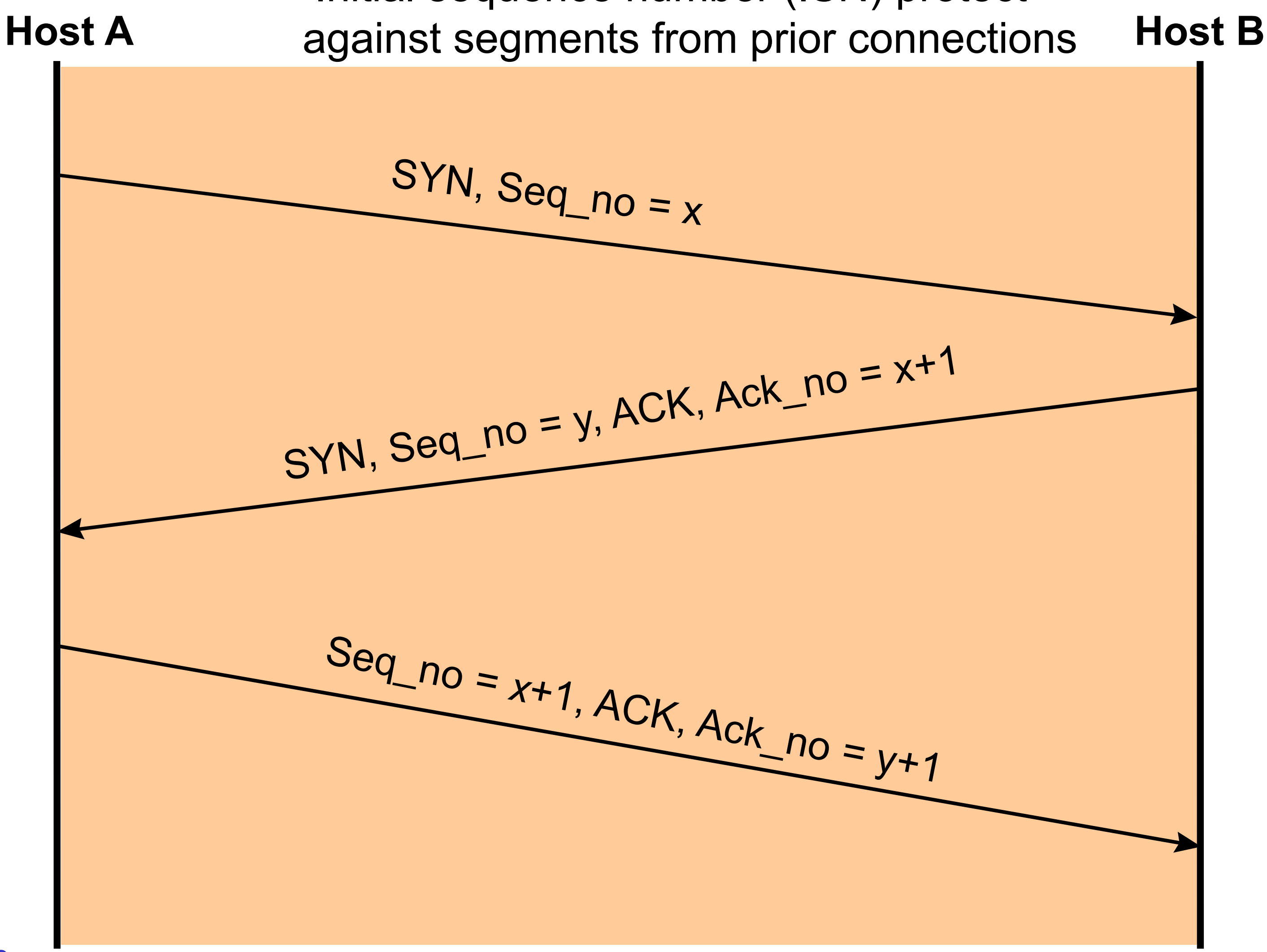
- Maximum Segment Size (MSS) option specifies largest segment a receiver wants to receive
- Window Scale option increases TCP window from 16 to 32 bits

Outline

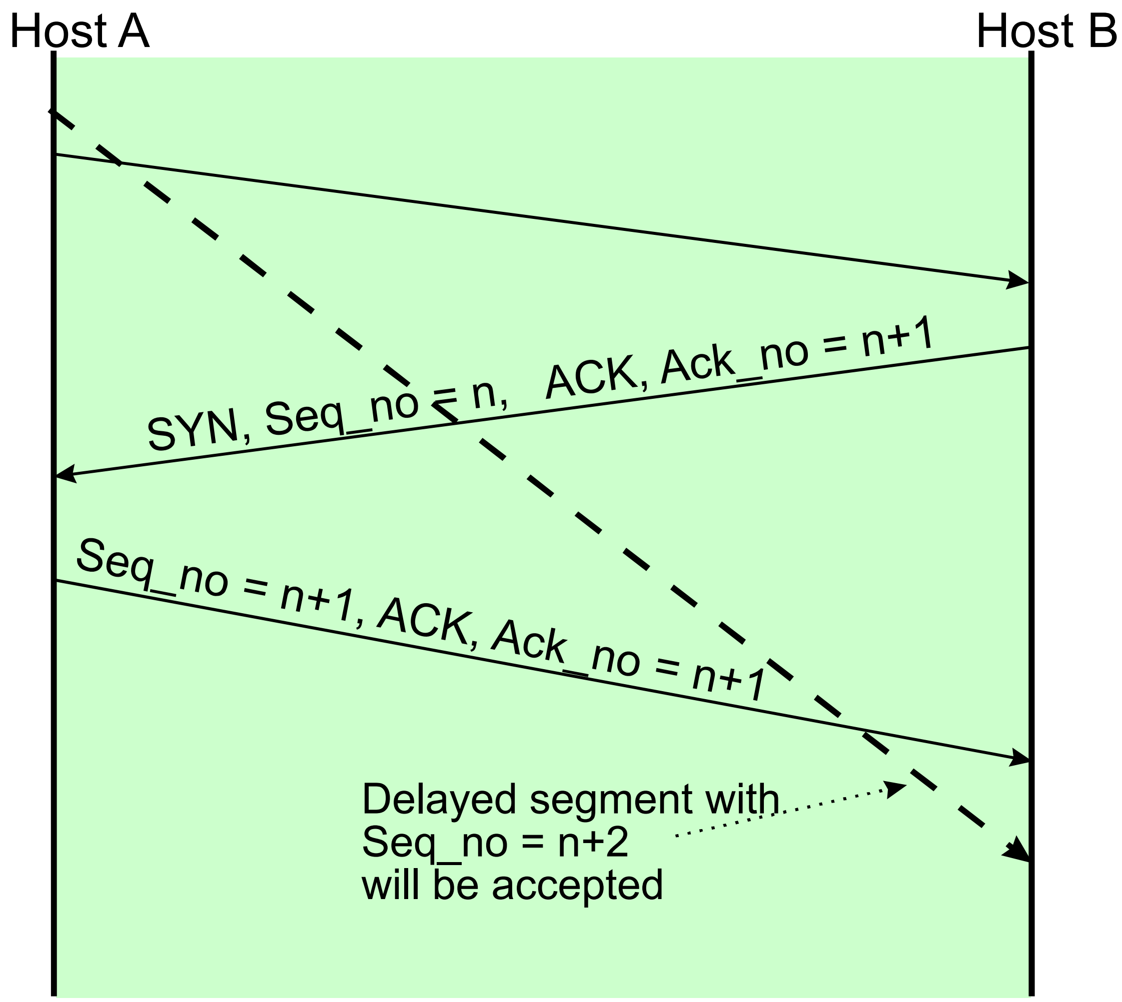
- UDP Protocol
- TCP Reliable Stream Service
- TCP Protocol
- TCP Connection Management
- TCP Congestion Control

TCP Connection Establishment

- "Three-way Handshake"
- Initial sequence number (ISN) protect against segments from prior connections



If host always uses the same ISN



Initial Sequence Number

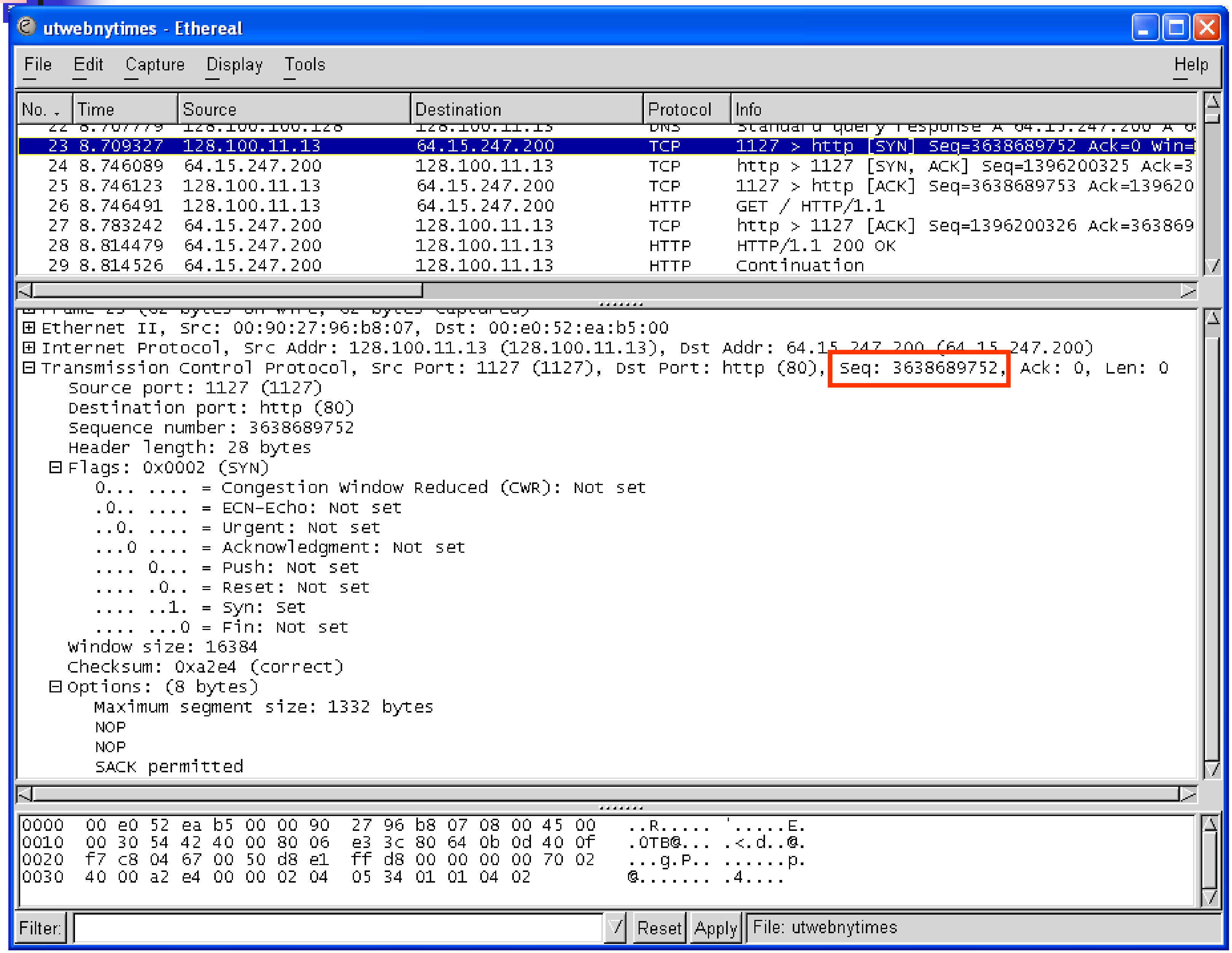
- Select initial sequence numbers (ISN) to protect against segments from prior connections (that may circulate in the network and arrive at a much later time)
- Select ISN to avoid *overlap* with sequence numbers of prior connections
- Use local clock to select ISN sequence number
- Time for clock to go through a full cycle should be greater than the maximum lifetime of a segment (MSL); Typically MSL=120 seconds
- High bandwidth connections pose a problem

- ***This problem also applies to SN wrap around!***

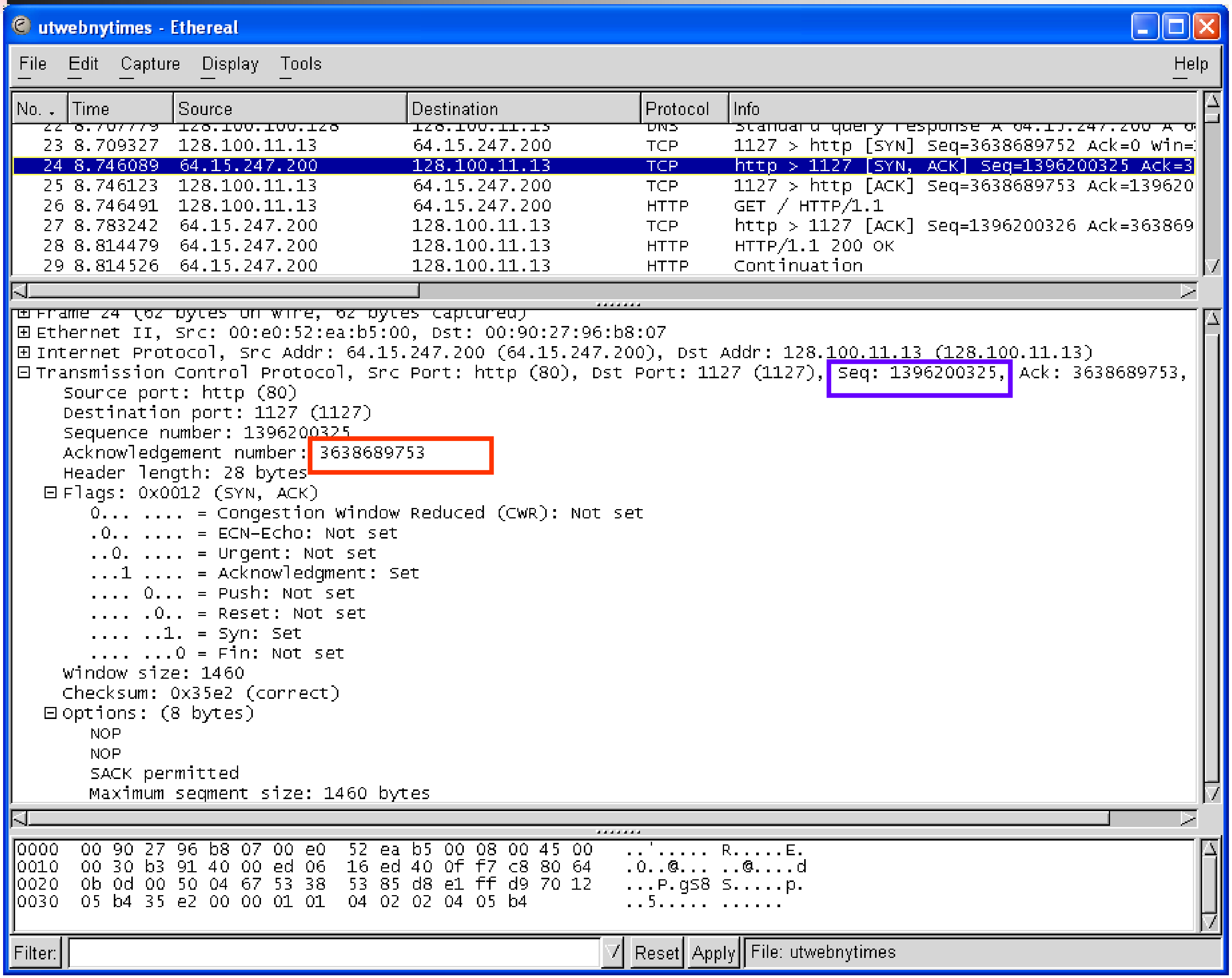
Maximum Segment Size

- Maximum Segment Size
  - largest block of data that TCP sends to other end
- Each end can announce its MSS during connection establishment
- Default is 576 bytes including 20 bytes for IP header and 20 bytes for TCP header
- Ethernet implies MSS of 1460 bytes

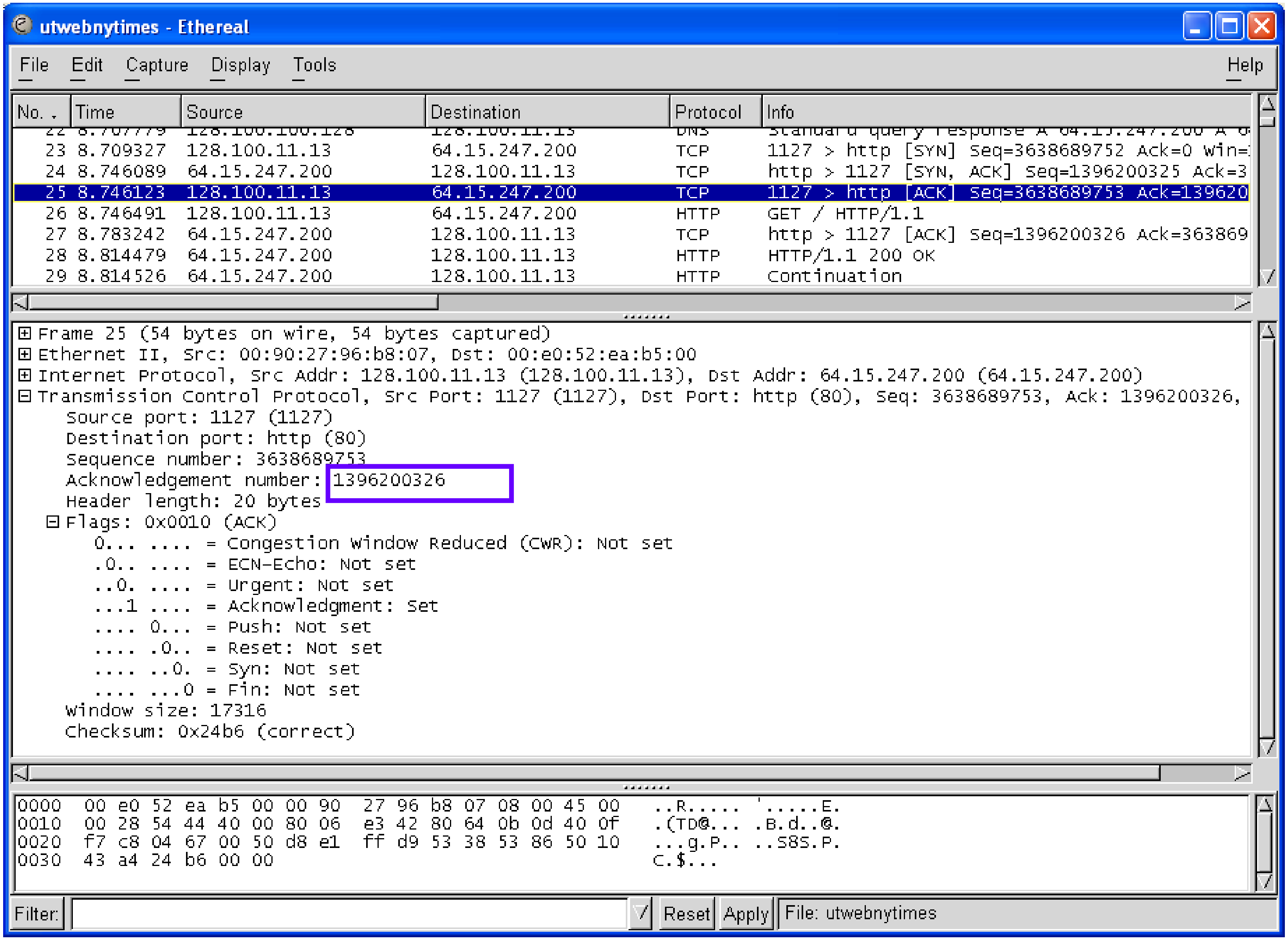
Near End: Connection Request



Far End: Ack and Request

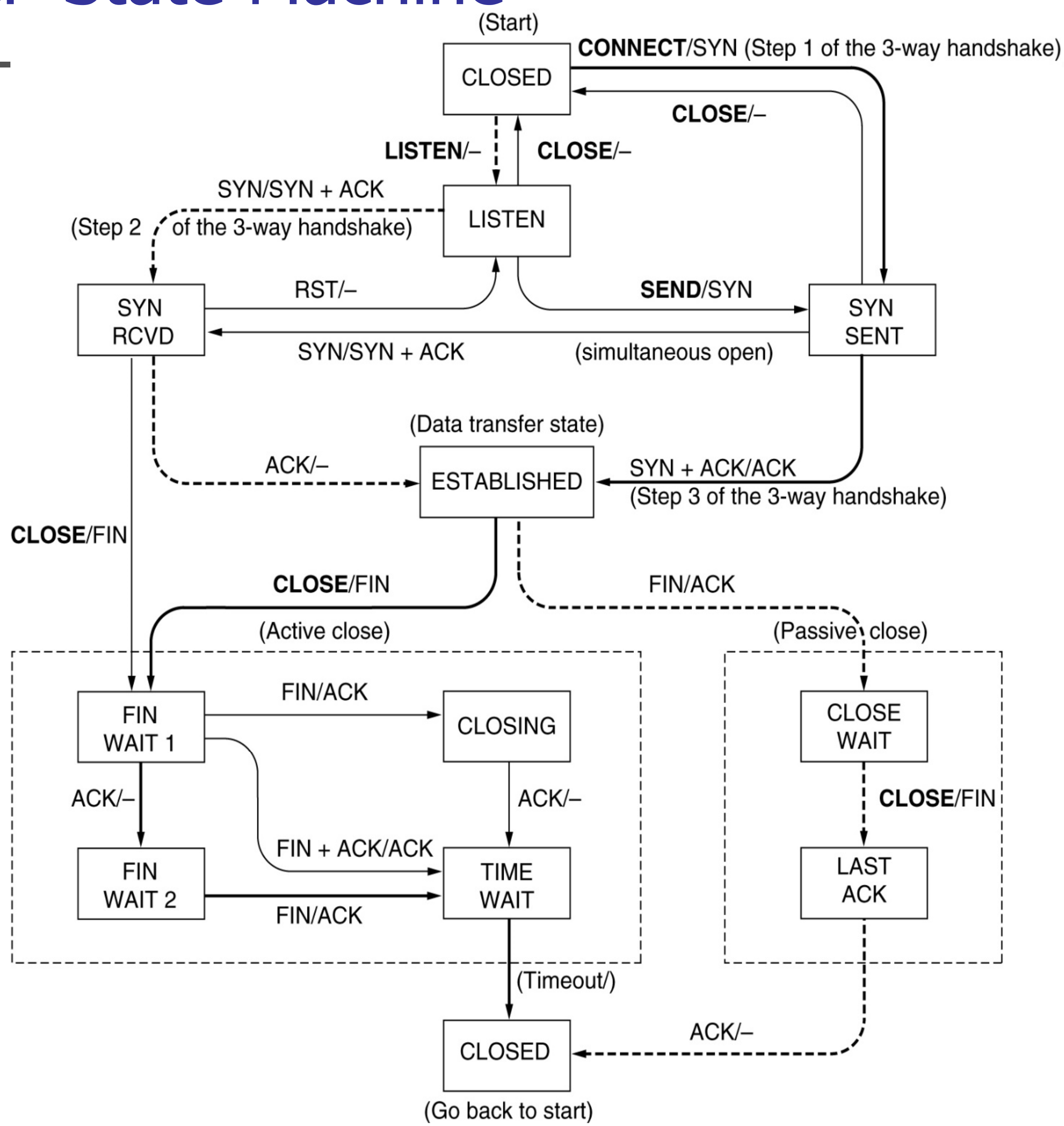


Near End: Ack

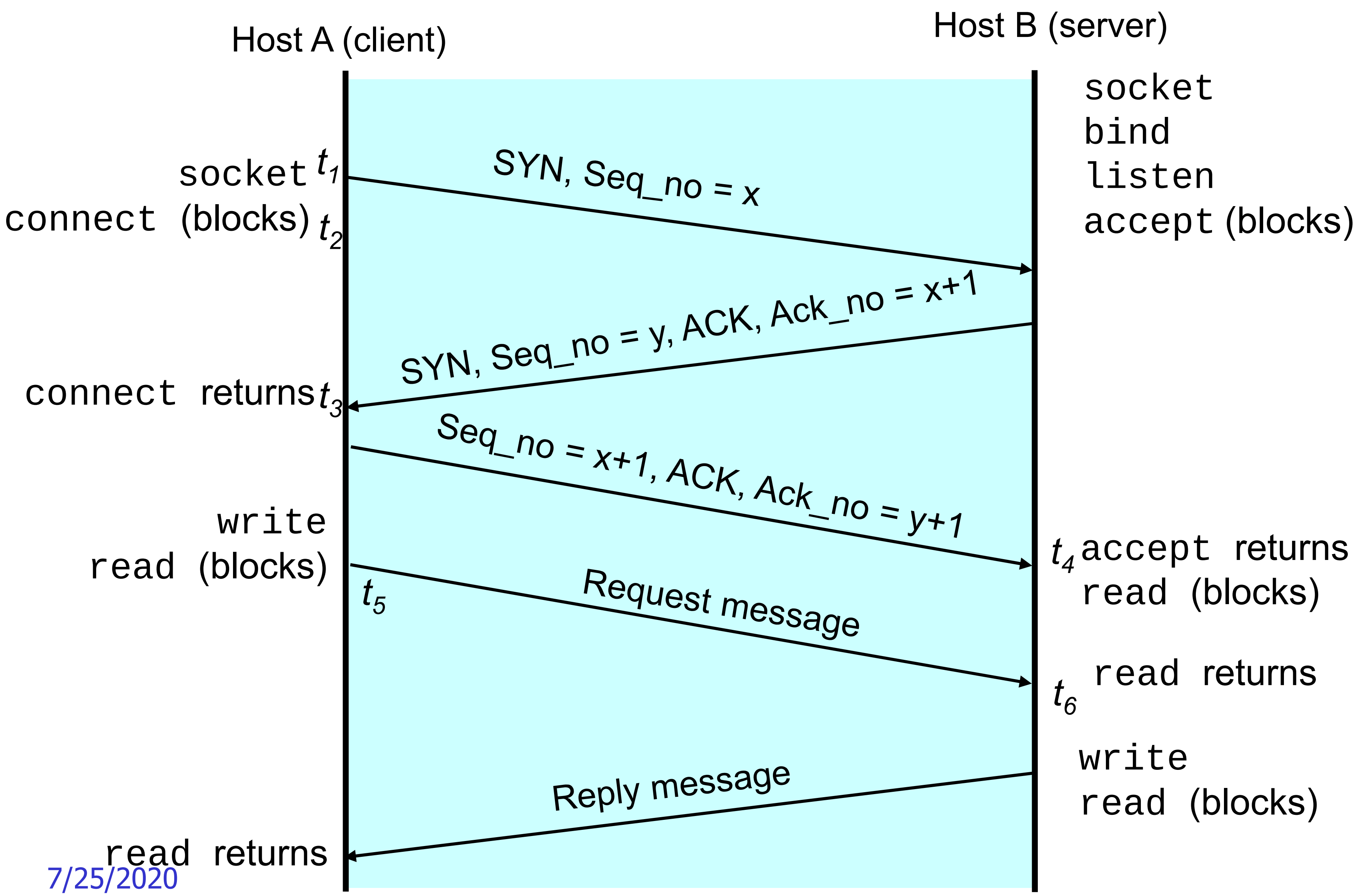


utwebnytimes - Ethereal
File Edit Capture Display Tools Help
No. Time Source Destination Protocol Info
23 8.709327 128.100.11.13 64.15.247.200 TCP 1127 > http [SYN] Seq=3638689752 Ack=0 Win=
24 8.746089 64.15.247.200 128.100.11.13 TCP http > 1127 [SYN, ACK] Seq=1396200325 Ack=3
25 8.746123 128.100.11.13 64.15.247.200 TCP 1127 > http [ACK] Seq=3638689753 Ack=139620
26 8.746491 128.100.11.13 64.15.247.200 HTTP GET / HTTP/1.1
27 8.783242 64.15.247.200 128.100.11.13 TCP http > 1127 [ACK] Seq=1396200326 Ack=363869
28 8.814479 64.15.247.200 128.100.11.13 HTTP HTTP/1.1 200 OK
29 8.814526 64.15.247.200 128.100.11.13 HTTP Continuation
Frame 25 (54 bytes on wire, 54 bytes captured)
Ethernet II, Src: 00:90:27:96:b8:07, Dst: 00:e0:52:ea:b5:00
Internet Protocol, Src Addr: 128.100.11.13 (128.100.11.13), Dst Addr: 64.15.247.200 (64.15.247.200)
Transmission Control Protocol, Src Port: 1127 (1127), Dst Port: http (80), Seq: 3638689753, Ack: 1396200326,
Source port: 1127 (1127)
Destination port: http (80)
Sequence number: 3638689753
Acknowledgement number: 1396200326
Header length: 20 bytes
Flags: 0x0010 (ACK)
0... .... = Congestion Window Reduced (CWR): Not set
.0.. .... = ECN-Echo: Not set
..0. .... = Urgent: Not set
...1 .... = Acknowledgment: Set
.... 0... = Push: Not set
.... .0.. = Reset: Not set
.... ..0. = Syn: Not set
....0 = Fin: Not set
Window size: 17316
Checksum: 0x24b6 (correct)
0000 00 e0 52 ea b5 00 00 90 27 96 b8 07 08 00 45 00
0010 00 28 54 44 40 00 80 06 e3 42 80 64 0b 0d 40 0f
0020 f7 c8 04 67 00 50 d8 e1 ff d9 53 38 53 86 50 10
0030 43 a4 24 b6 00 00
Filter: Reset Apply File: utwebnytimes

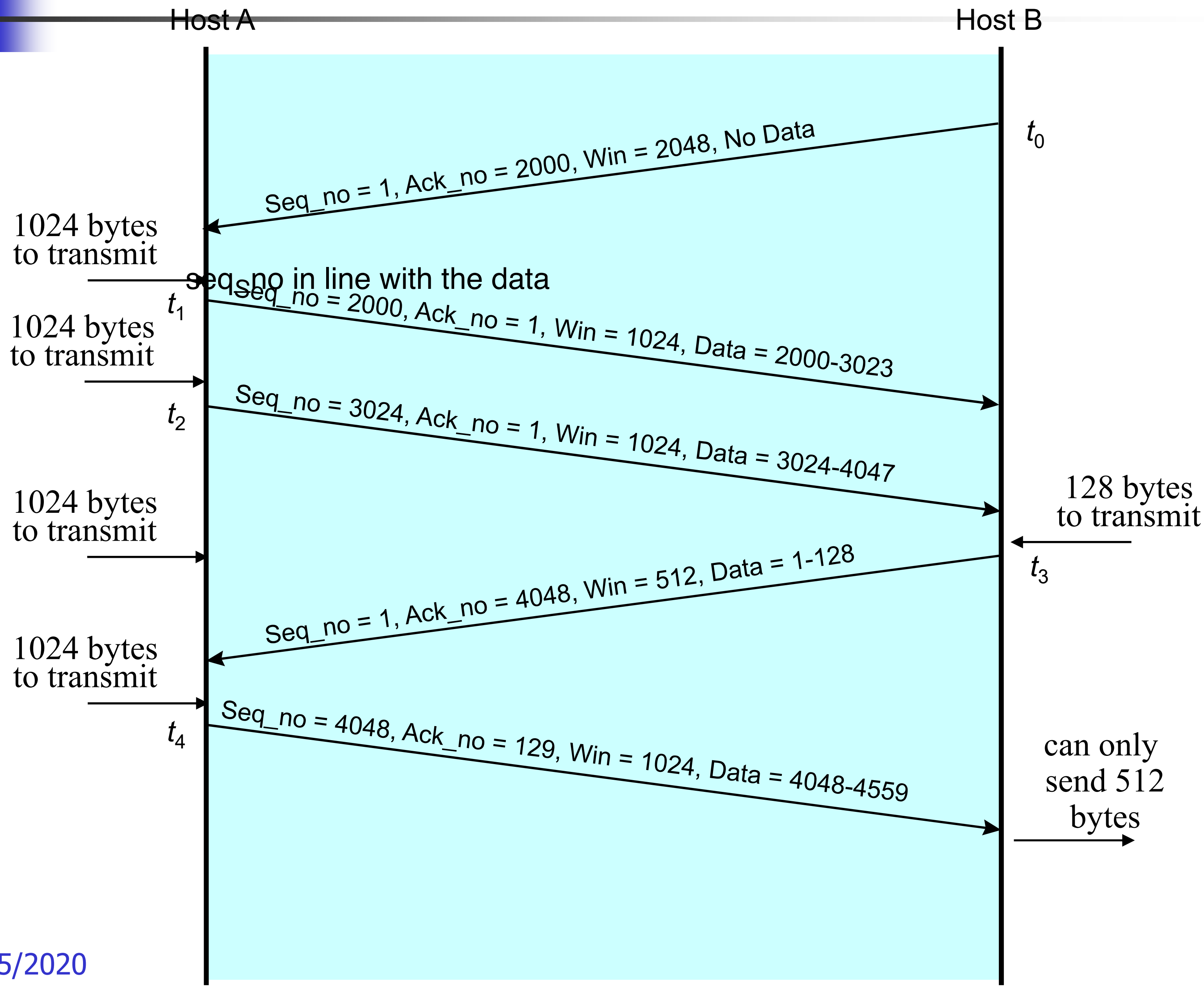
TCP State Machine



Client-Server Application



TCP Window Flow Control



Nagle Algorithm

- Situation: user types 1 character at a time
  - Transmitter sends TCP segment per character (41B)
  - Receiver sends ACK (40B)
  - Receiver echoes received character (41B)
  - Transmitter ACKs echo (40 B)
  - 162 bytes transmitted to transfer 1 character!
- Solution:
  - TCP sends data & waits for ACK
  - New characters buffered
  - Send new characters when ACK arrives
  - Equivalent to an algorithm adapting to RTT
    - Short RTT sends characters frequently at low efficiency (but this is fine, as this is like a light-loaded network)
    - Long RTT sends characters less frequently at greater efficiency

Silly Window Syndrome

- Situation:
  - Transmitter sends large amount of data
  - Receiver buffer depleted slowly, so buffer fills
  - Every time a few bytes read from buffer, a new advertisement to transmitter is generated
  - Sender immediately sends data & fills buffer
  - Many small, inefficient segments are transmitted
- Solution:
  - Receiver does not advertize window until window is at least ½ of receiver buffer or maximum segment size
  - Transmitter refrains from sending small segments

Sequence Number Wraparound

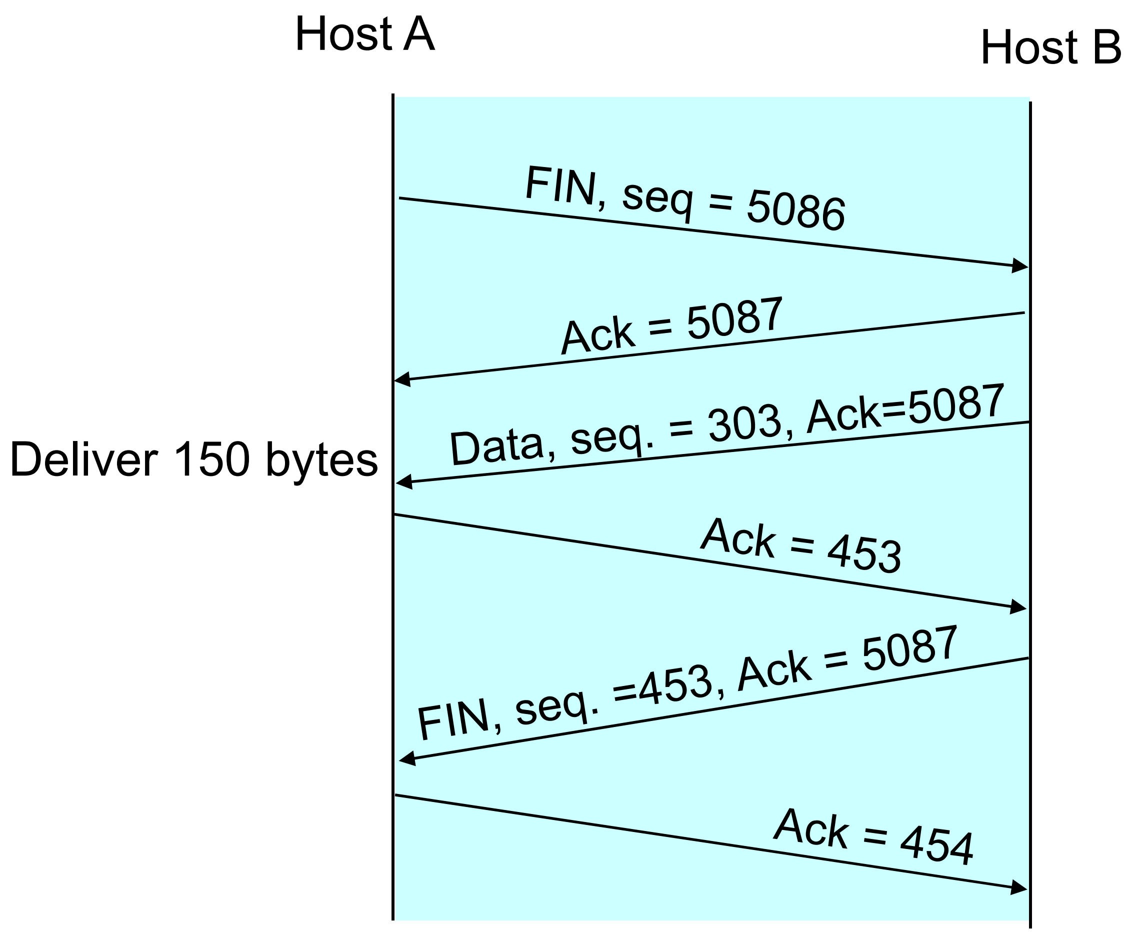
- $2^{32} = 4.29 \times 10^9$ bytes $= 34.3 \times 10^9$ bits
  - At 1 Gbps, sequence number wraparound in 34.3 seconds.
- Timestamp option: Insert 32 bit timestamp in header of each segment
  - Timestamp + sequence no → 64-bit seq. no
  - Timestamp clock must:
    - tick forward at least once every 2^{31} bytes
    - Not complete cycle in less than one MSL
    - Example: clock tick every 1 ms (can support a number of Tbps) wraps around in 25 days

BW-Delay Product & Advertised Window Size

- Suppose RTT=100 ms, R=2.4 Gbps
  - # bits in pipe = 3 Mbytes
- If single TCP process occupies pipe, then required advertised window size is
  - RTT x Bit rate = 3 Mbytes
  - Normal maximum window size is 65535 bytes
- Solution: Window Scale Option
  - Window size up to 65535 x 2^{14} = 1 Gbyte allowed
  - Requested in SYN segment

TCP Connection Closing

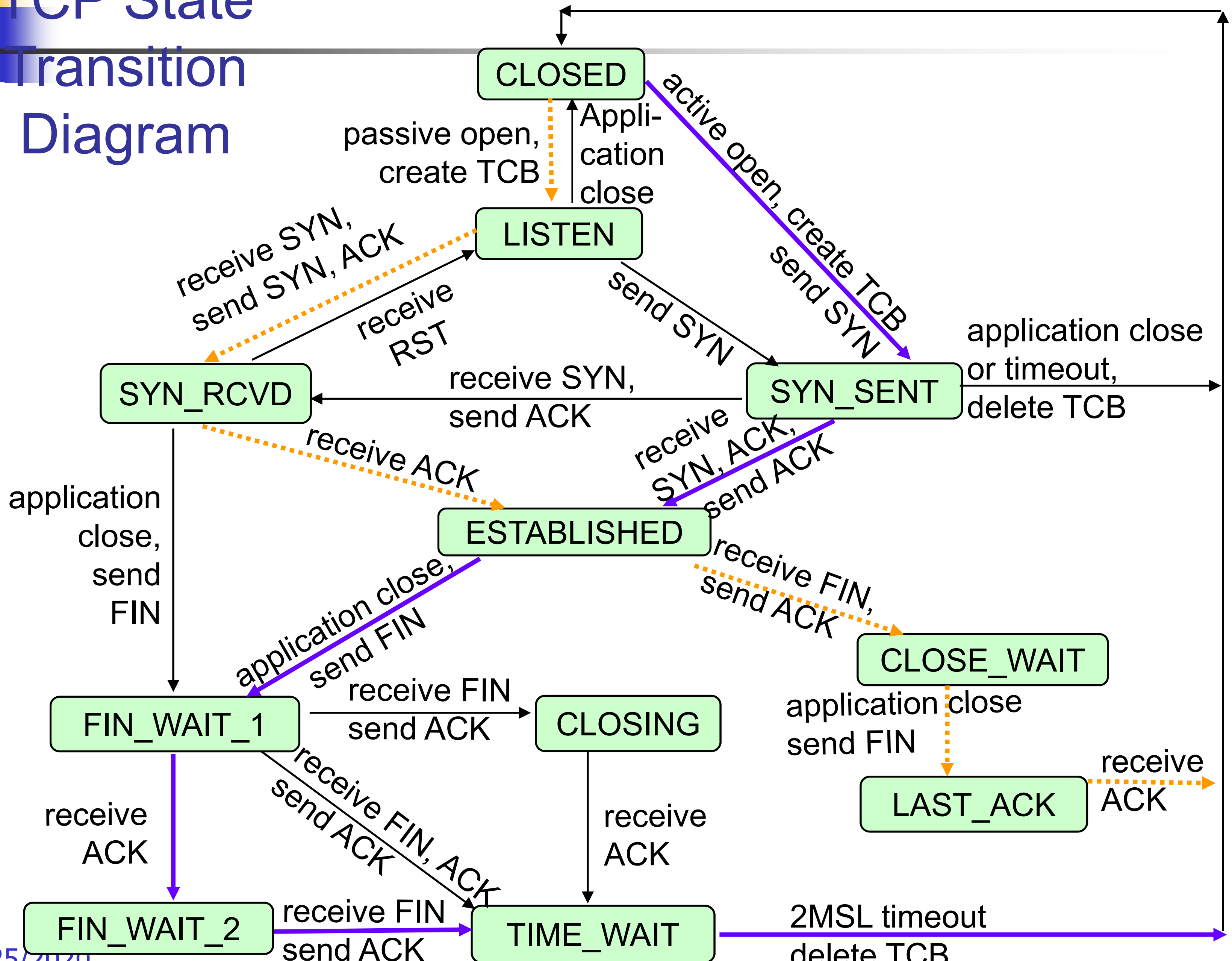
"Graceful Close"



TIME_WAIT state

- When TCP receives ACK to last FIN, TCP enters TIME_WAIT state
  - Protects future incarnations of connection from delayed segments
  - TIME_WAIT = 2 x MSL
  - Only valid segment that can arrive while in TIME_WAIT state is FIN retransmission
    - If such segment arrives, resent ACK & restart TIME_WAIT timer
  - When timer expires, close TCP connection & delete connection record

TCP State Transition Diagram

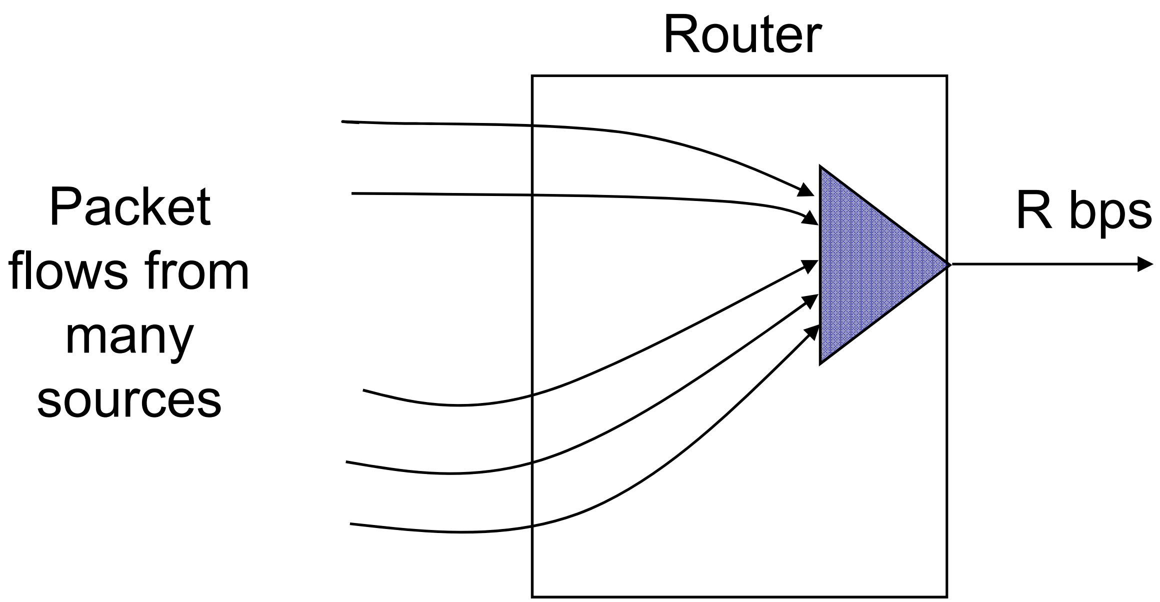


Outline

- UDP Protocol
- TCP Reliable Stream Service
- TCP Protocol
- TCP Connection Management
- TCP Congestion Control

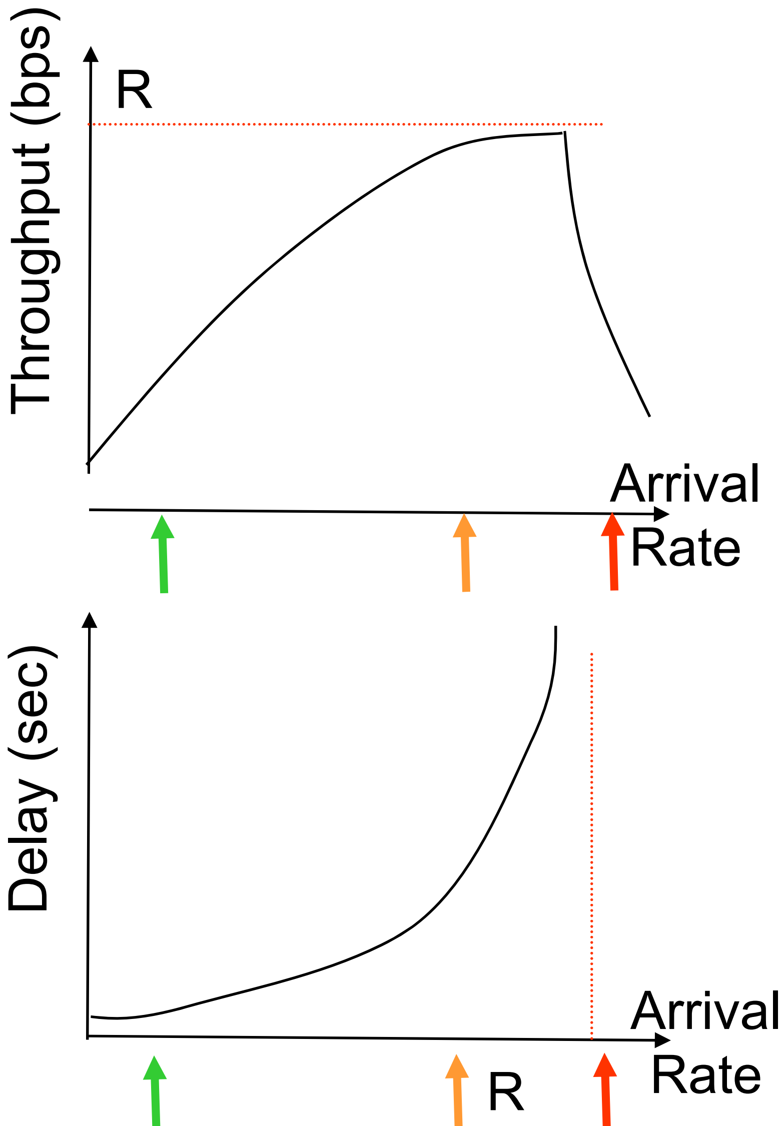
TCP Congestion Control

- *Advertised window* size is used to ensure that receiver's buffer will not overflow
- However, buffers at intermediate routers between source and destination may overflow



- Congestion occurs when total arrival rate from all packet flows exceeds R over a sustained period of time
- Buffers at multiplexer will fill and packets will be lost

Phases of Congestion Behavior



1. Light traffic
   - Arrival Rate << R
   - Low delay
   - Can accommodate more
2. Knee (congestion onset)
   - Arrival rate approaches R
   - Delay increases rapidly
   - Throughput begins to saturate
3. Congestion collapse
   - Arrival rate > R
   - Large delays, packet loss
   - Useful application throughput drops

Window Congestion Control

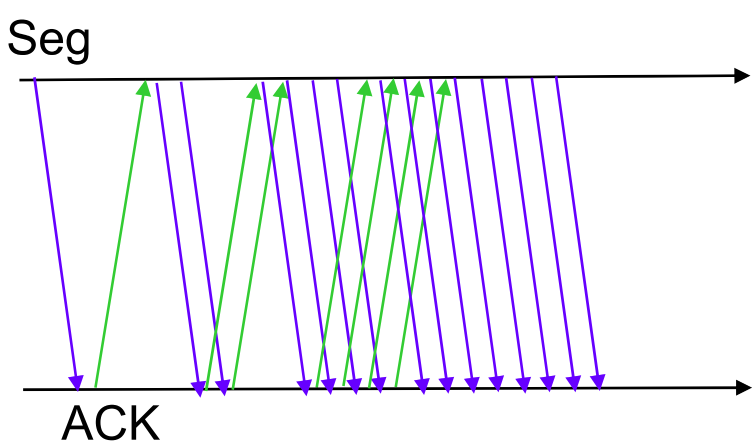
- Desired operating point: just before knee
  - Sources must control their sending rates so that aggregate arrival rate is just before knee
- TCP sender maintains a *congestion window* cwnd to control congestion at intermediate routers
- ***Effective window is the minimum of congestion window and advertised window***
- Problem: source does not know what its “fair” share of available bandwidth should be
- Solution: adapt dynamically to available BW
  - Sources probe the network by increasing cwnd
  - When congestion detected, sources reduce rate
  - Ideally, sources sending rate stabilizes near ideal point

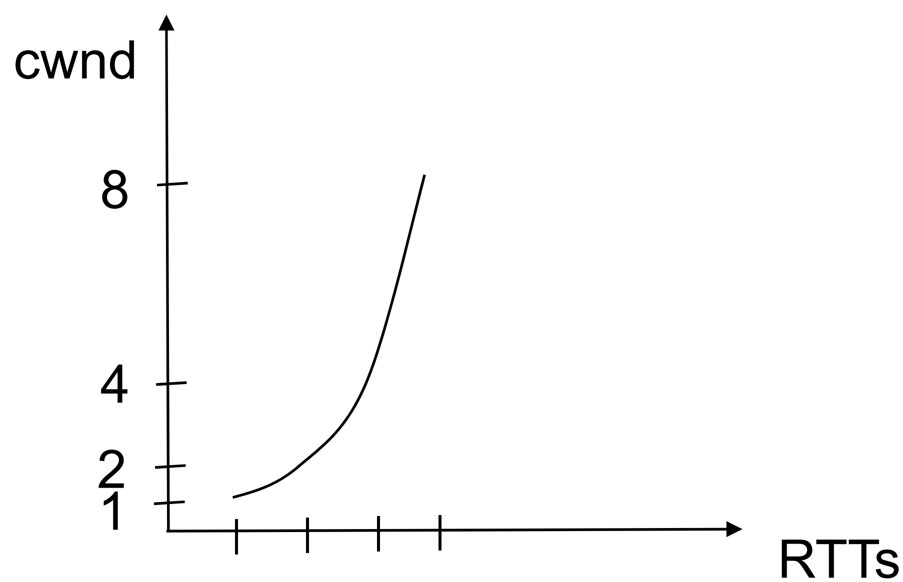
Congestion Window

- How does the TCP congestion algorithm change congestion window dynamically according to the most up-to-date state of the network?
- At light traffic: each segment is ACKed quickly
  - Increase cwnd aggressively
- At knee: segment ACKs arrive, but more slowly
  - Slow down increase in cwnd
- At congestion: segments encounter large delays (so retransmission timeouts occur); segments are dropped in router buffers (resulting in duplicate ACKs)
  - Reduce transmission rate, then probe again

TCP Congestion Control: Slow Start

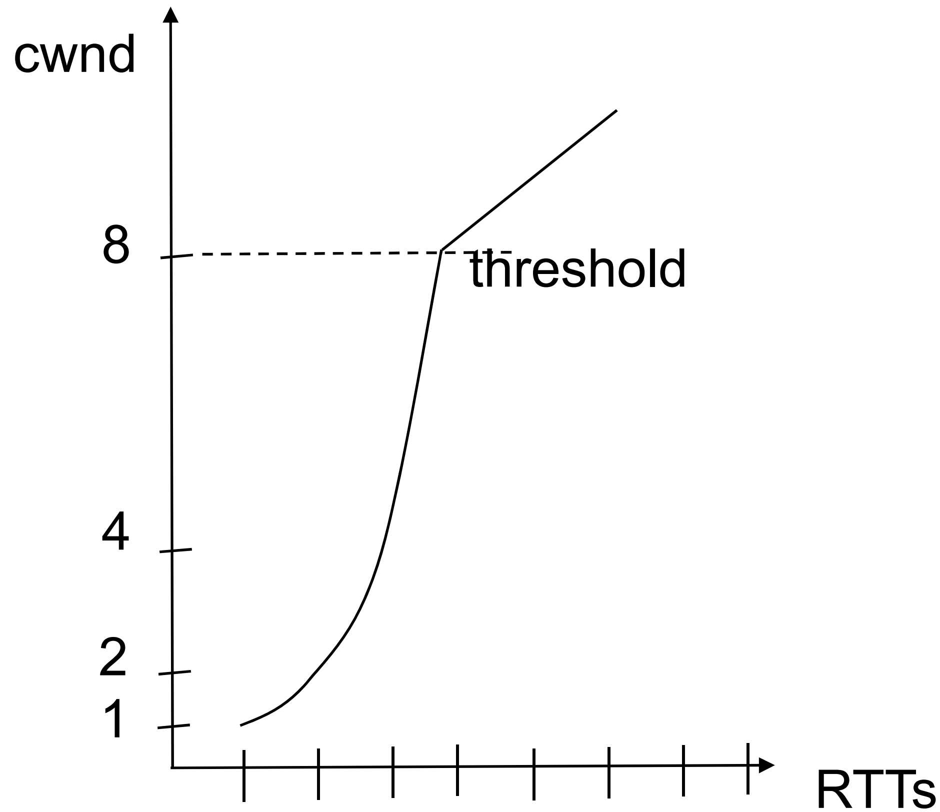
- **Slow start**: increase congestion window size by one segment upon receiving an ACK from receiver
  - initialized at ≤ 2 segments
  - used at (re)start of data transfer
  - congestion window increases exponentially





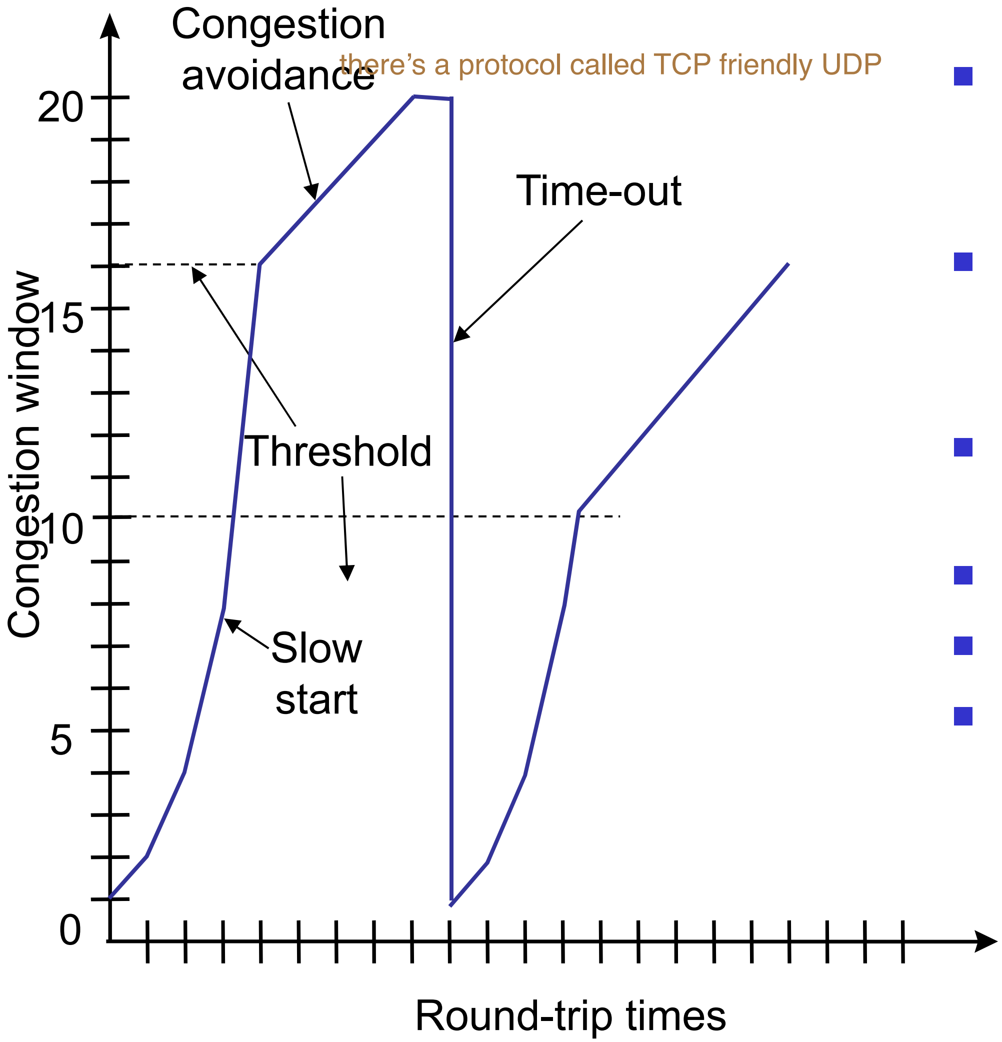
TCP Congestion Control: Congestion Avoidance

- Algorithm progressively sets a *congestion threshold*
  - When cwnd > threshold, slow down rate at which cwnd is increased
- Increase congestion window size by one segment per round-trip-time (RTT)
  - Each time an ACK arrives, cwnd is increased by 1/cwnd
  - In one RTT, cwnd segments are sent, so total increase in cwnd is cwnd x 1/cwnd = 1
  - cwnd grows linearly with time



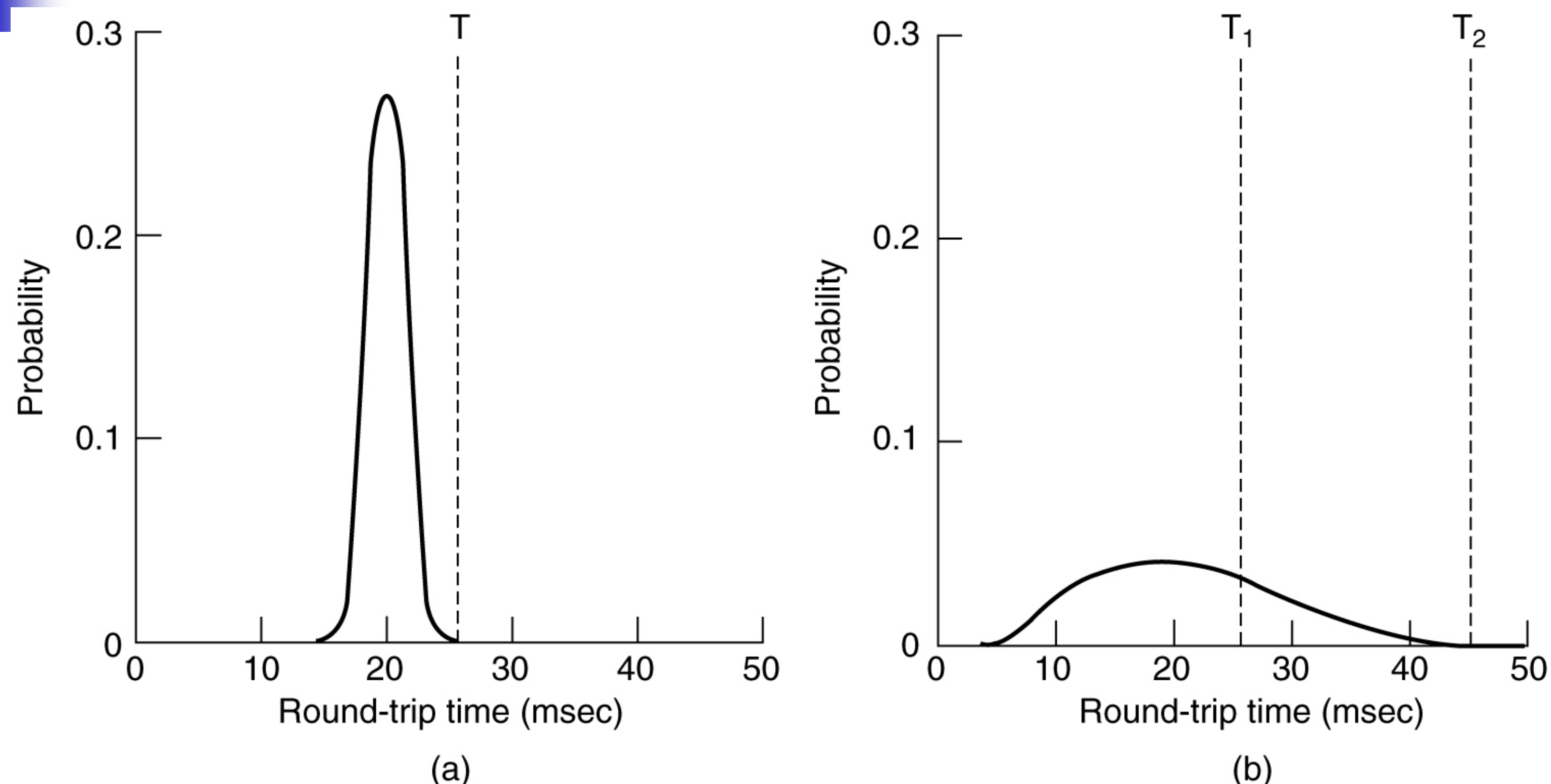
self-learning

TCP Congestion Control: Congestion



- Congestion is detected upon timeout or receipt of duplicate ACKs
- Assume current cwnd corresponds to available bandwidth
- Adjust congestion threshold = ½ x current cwnd
- Reset cwnd to 1
- Go back to slow-start
- Over several cycles expect to converge to congestion threshold equal to about ½ the available bandwidth

TCP Timer Management

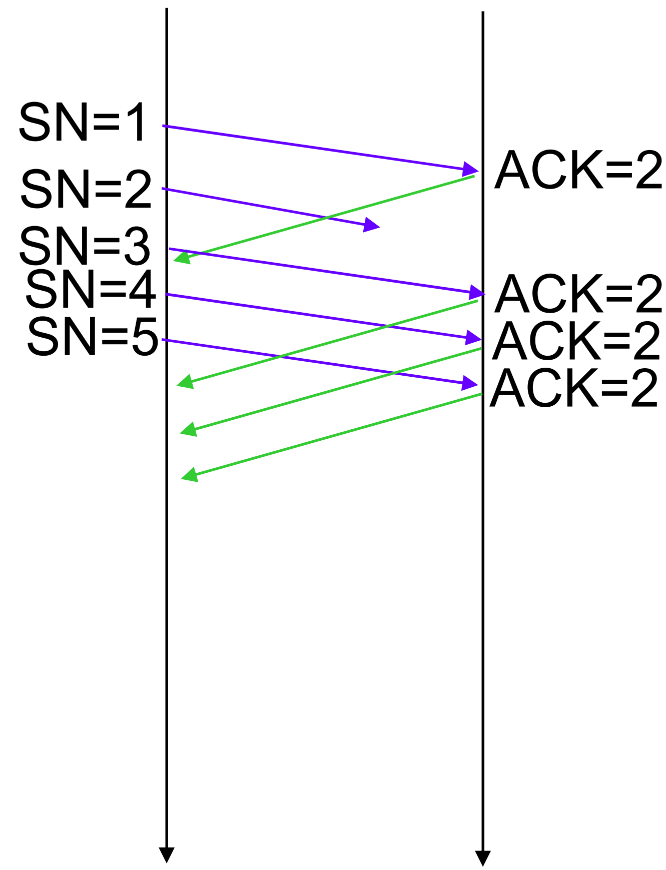


- (a) Probability density of ACK arrival times in the data link layer.
- (b) Probability density of ACK arrival times for TCP.

$$\textit{Timeout} = RTT + 4xD$$
$$D = \alpha D+(1- \alpha)|RTT-M|$$
$$RTT = \beta RTT+(1- \beta)M$$
M : time to get ACK back

Fast Retransmit & Fast Recovery

- Congestion causes many segments to be dropped
- If only a single segment is dropped, then subsequent segments trigger duplicate ACKs before timeout
- Can avoid large decrease in cwnd as follows:
  - When three duplicate ACKs arrive, retransmit lost segment immediately
  - Reset congestion threshold to ½ cwnd
  - Reset cwnd to congestion threshold + 3 to account for the three segments that triggered duplicate ACKs
  - Remain in congestion avoidance phase
  - However if timeout expires, reset cwnd to 1
  - In absence of timeouts, cwnd will oscillate around optimal value



TCP Congestion Control: Fast Retransmit & Fast Recovery