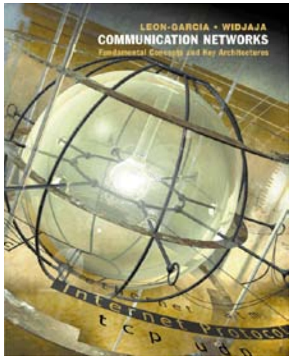
Chapter 8
Communication Networks and Services



Transport Layer Protocols: UDP and TCP

Outline

- UDP Protocol
- TCP – Quick Overview
- TCP Header
- TCP Connection Management
- TCP Congestion Control

UDP

- Best effort (unreliable) 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., VoIP, video, 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

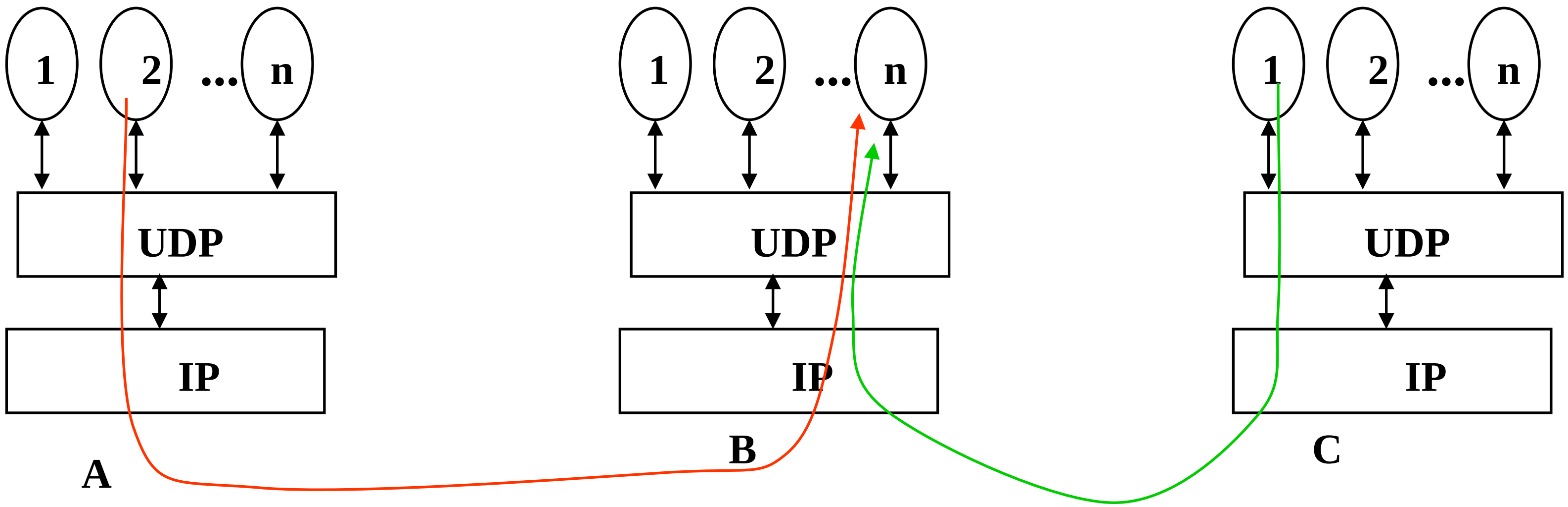
1024-65536

- Ephemeral client ports

- Source and destination ports:
  - Identify applications
  - 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 ≤ length ≤ 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



Outline

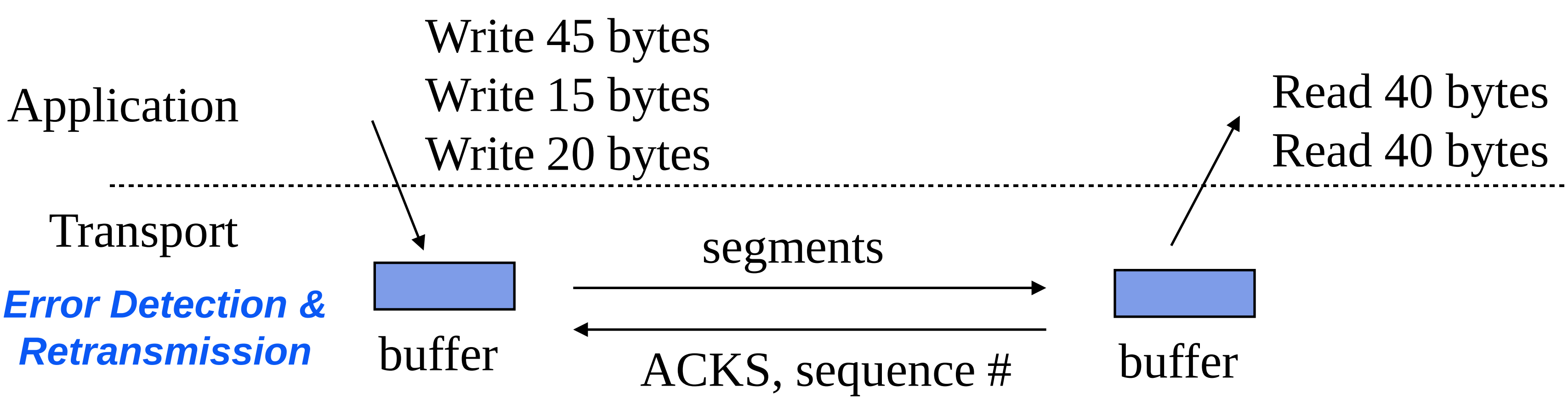
- TCP – Quick Overview
- TCP Header
- TCP Connection Management
- TCP Congestion Control

TCP-Quick Overview

- Reliable byte-stream service
- More complex transmitter & receiver
  - **Connection-oriented** (logical connection): full-duplex unicast connection between client & server processes
    - **Connection setup, connection state, connection release**
  - Higher delay than UDP
  - **Error control, flow control, and congestion control**
  - Higher header overhead
- 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
  - TCP groups bytes into **segments**
  - transmits segments as convenient
  - Application may send a 1000-byte message, TCP may transfer it into two chunks of 500-byte each or three chunks etc.
- Reliability
  - error control mechanism to deal with IP transfer impairments

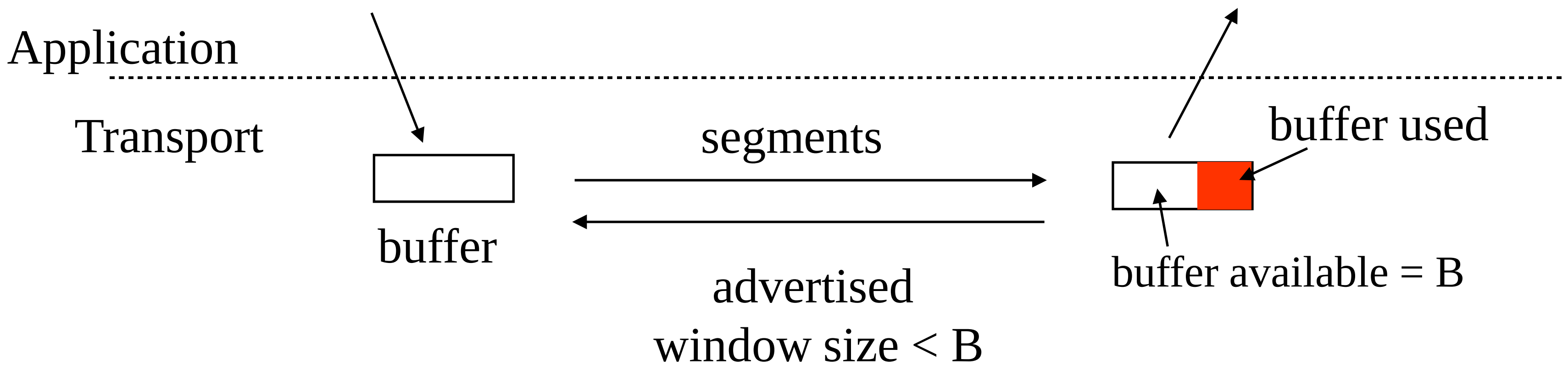


Flow Control between Hosts

- 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

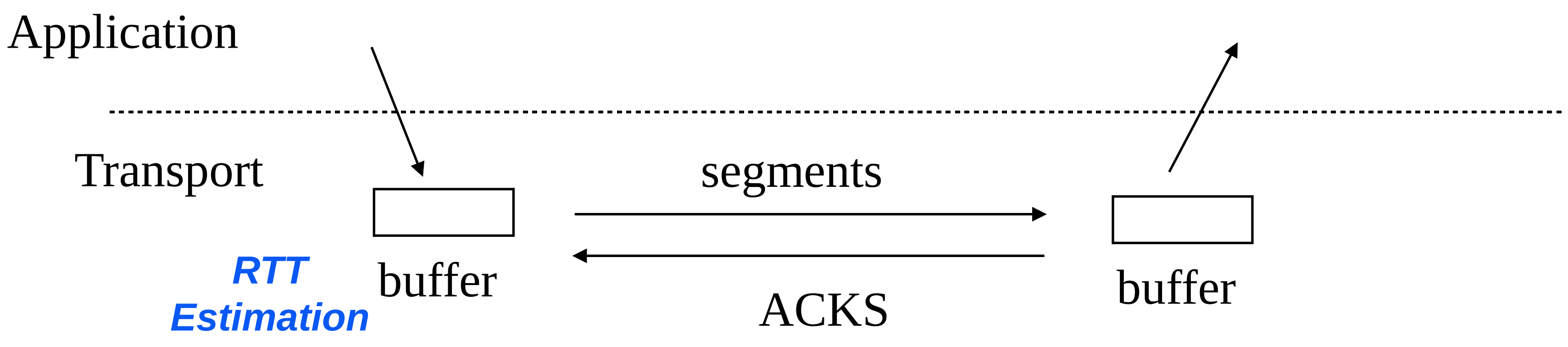


Congestion Control over the Network

- Available bandwidth to destination varies with activity of other users

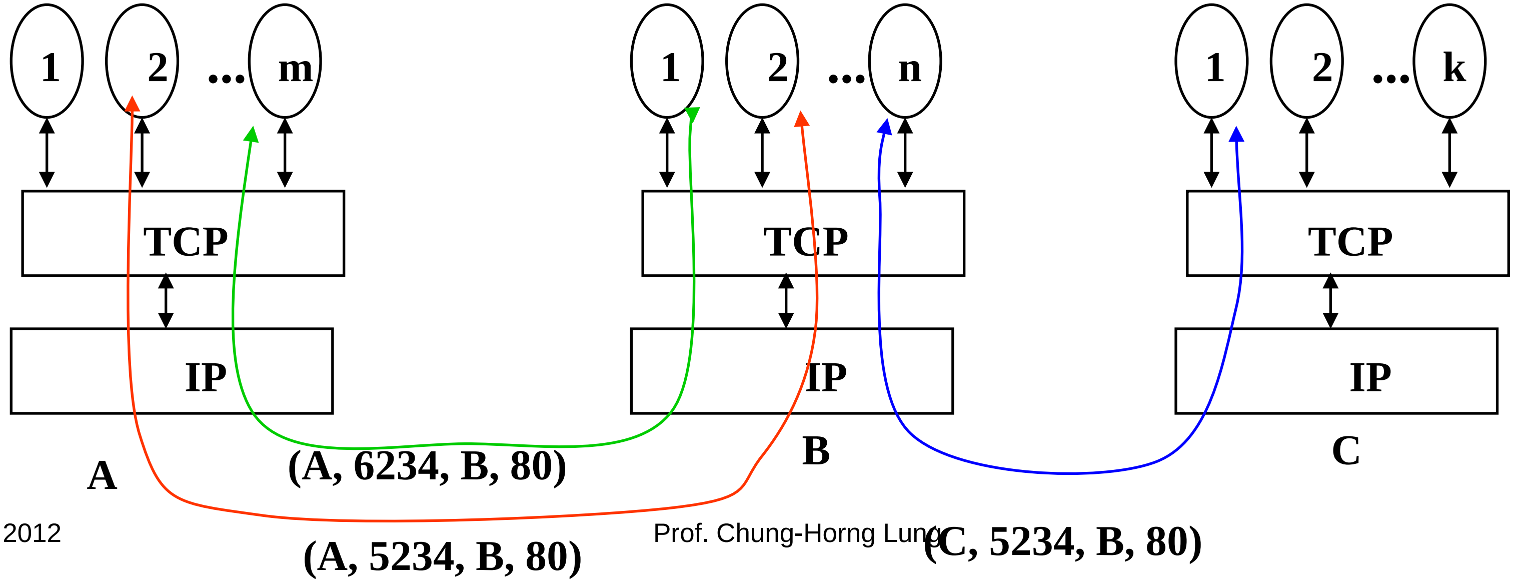
*

- Transmitter dynamically adjusts transmission rate according to **network congestion as indicated by RTT (round trip time) & ACKs**
- Elastic utilization of network bandwidth



TCP Multiplexing

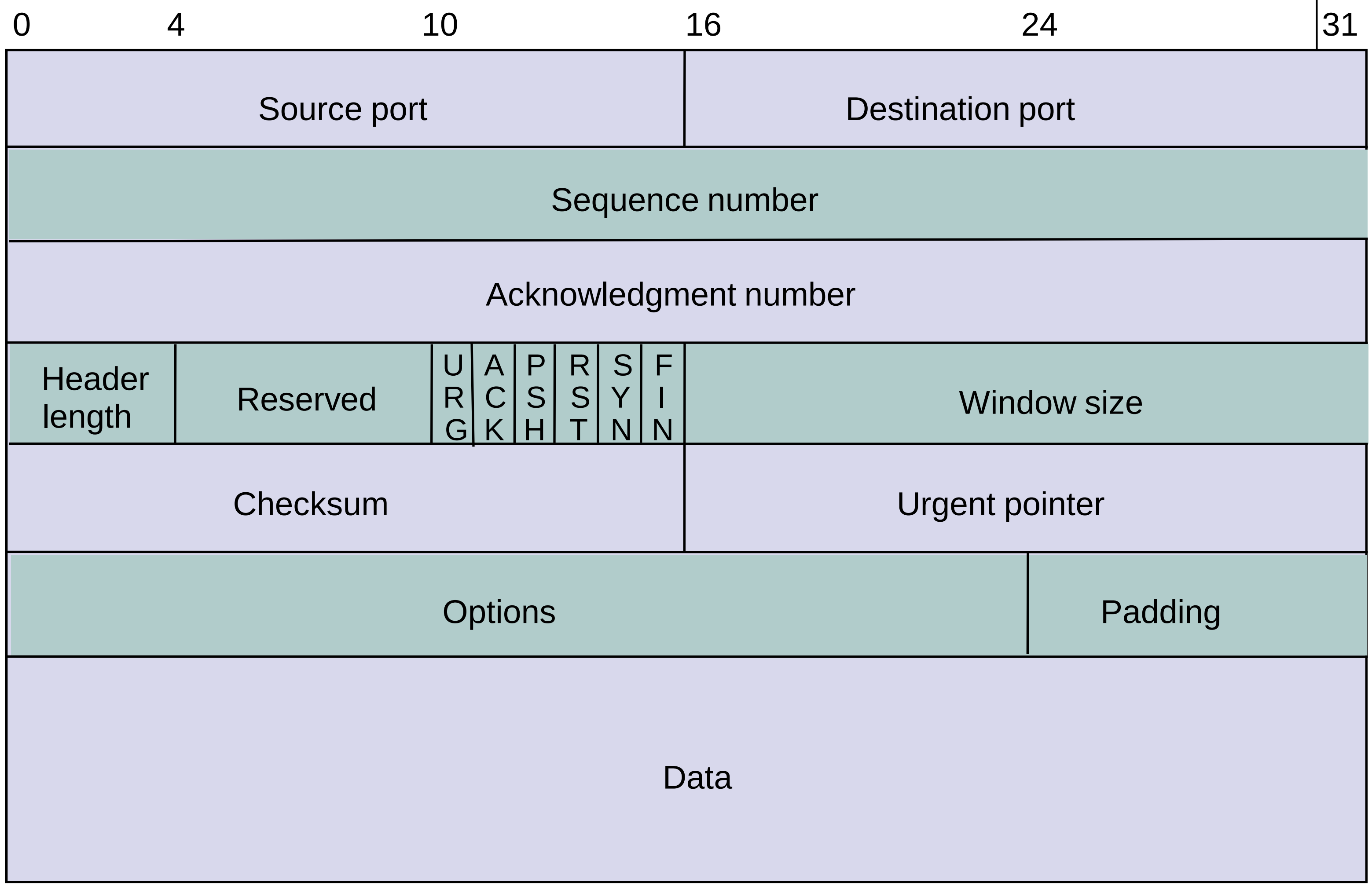
- A *TCP connection* is specified by a *4-tuple*
  - (source IP address, source port, destination IP address, destination port)
- TCP allows multiplexing of multiple connections between end systems to support multiple applications simultaneously
- Arriving segment directed according to connection 4-tuple



Outline

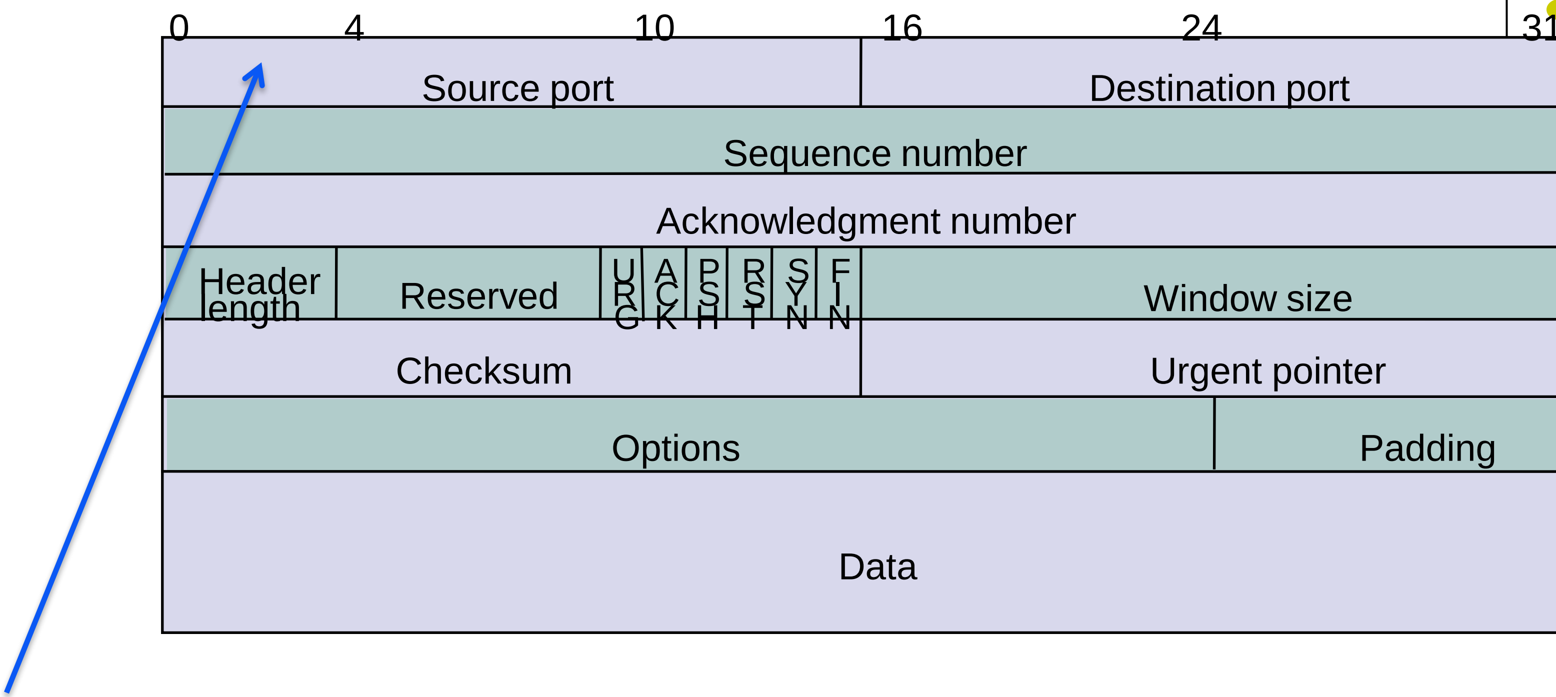
- TCP – Quick Overview
- TCP Header
- TCP Connection Management
- TCP Congestion Control

TCP Segment Format



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

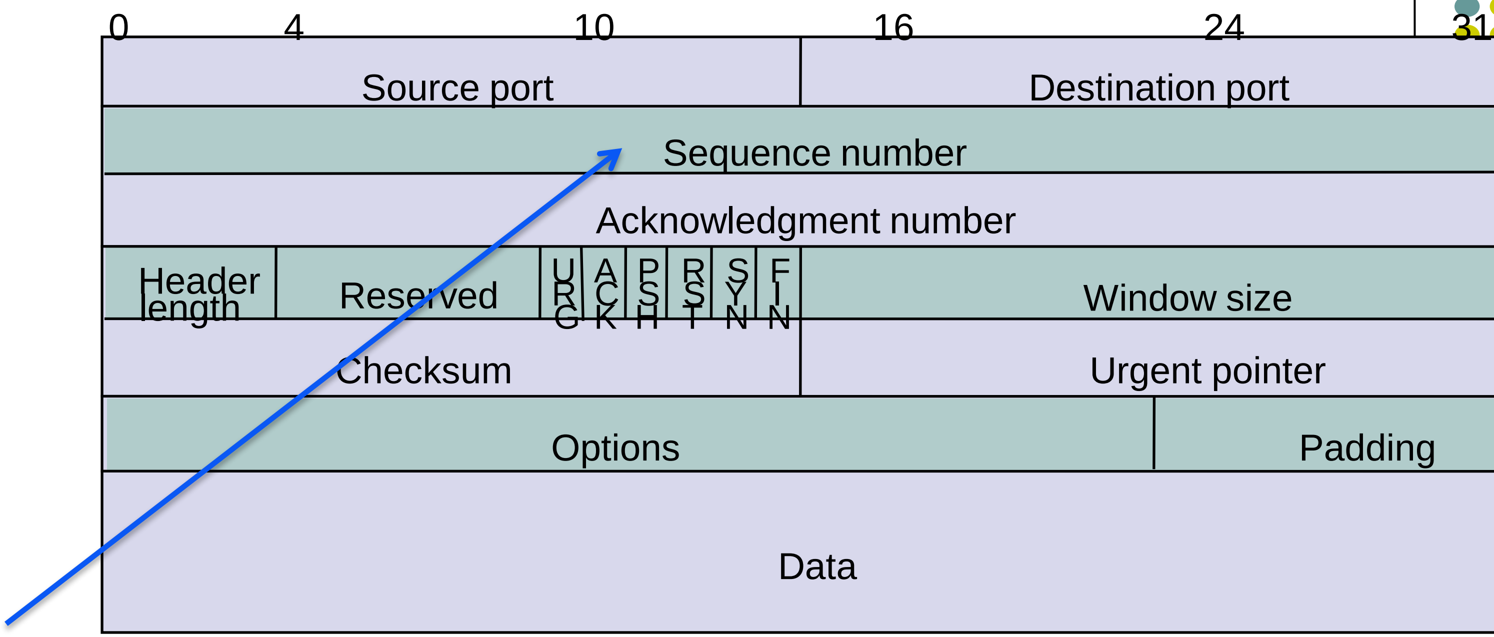
TCP Header



Port Numbers

- A socket identifies a connection endpoints or applications (processes)
  - IP address + port
- A connection specified by a *socket pair*
- Well-known ports: FTP 20, DNS 53, HTTP 80,

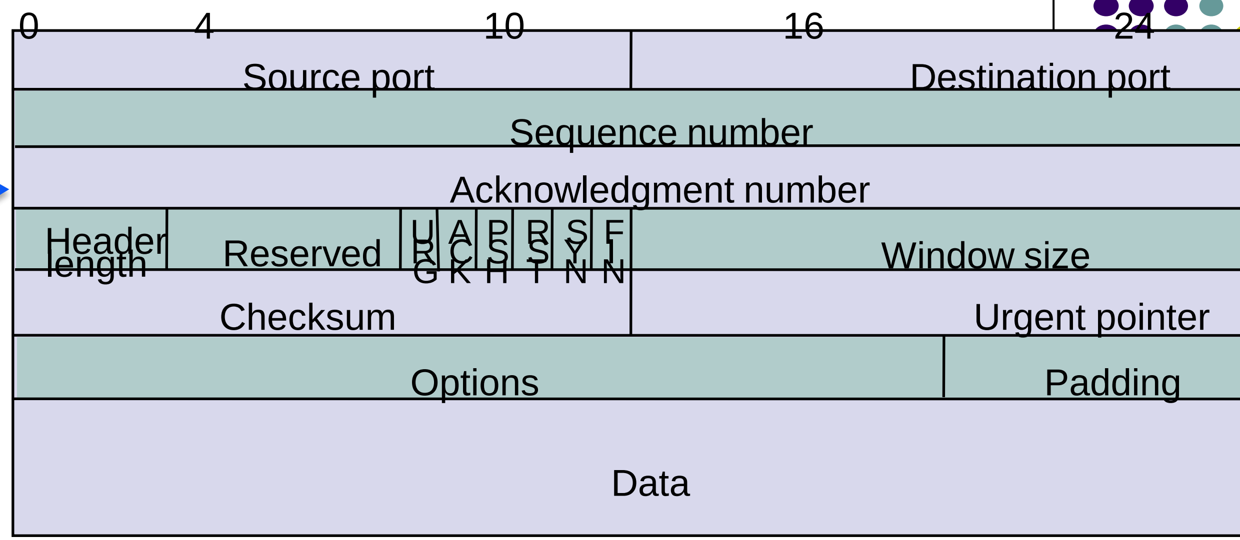
TCP Header



Sequence Number (SN): byte count, 32 bits ($0 \leq SN \leq 2^{32}-1$)

- Position of first data byte in segment (offset for the byte stream).
  - If SN=100 and there are 5 data bytes in the segment, then the next segment will have a SN=105.
- Initial sequence number selected during connection setup
  - If SYN=1(during connection establishment) the SN indicates the initial SN (ISN) of the senders byte stream. The sequence number for the first data byte in this stream will be ISN + 1.

TCP Header



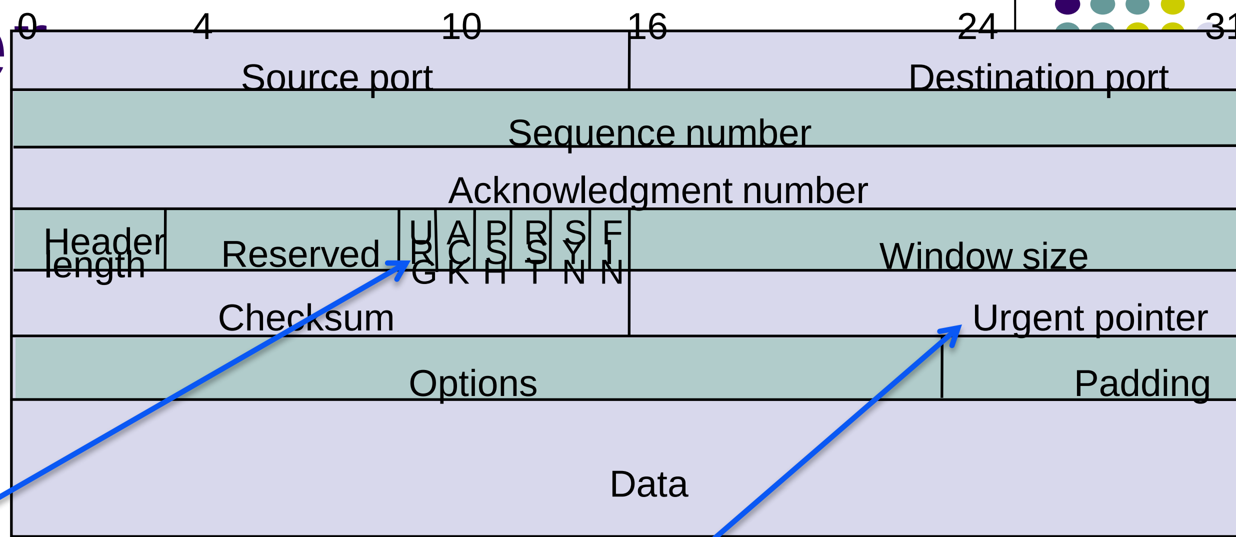
Acknowledgement Number (similar to ARQ)

- 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 (4 bytes)
- Minimum 20 bytes, maximum 60 bytes

TCP Heade



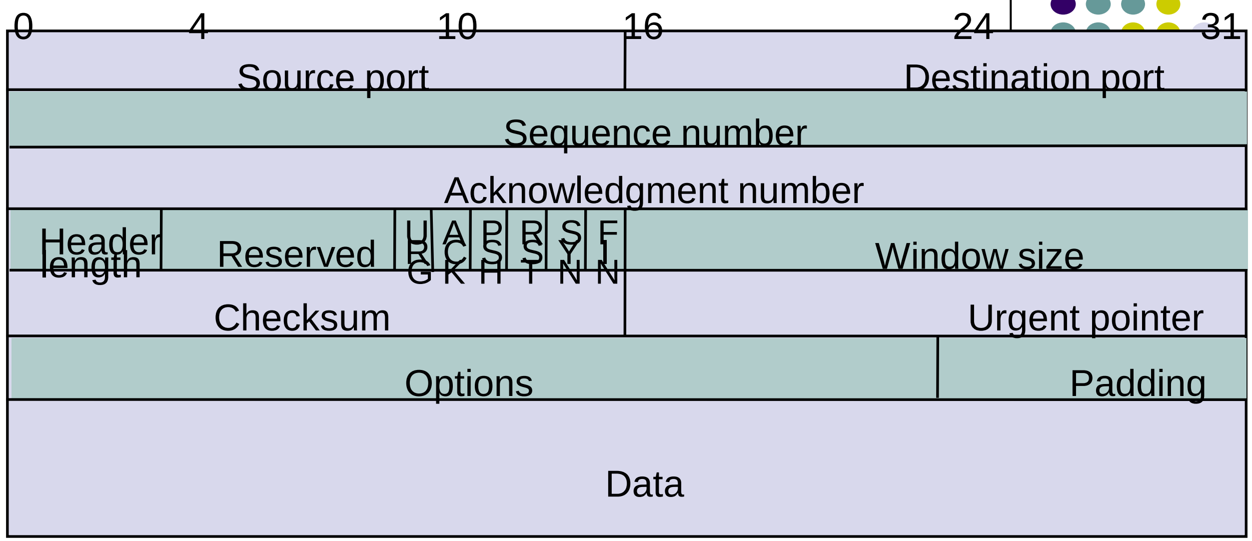
Reserved

- 6 bits
- Future use

Control (6 bits)

- URG: urgent pointer flag (data needs immediatey delivery)
  - Urgent message end = SN + **urgent pointer**
- **ACK**: ACK number is valid
- PSH: override TCP buffering, pass to the application immediately
- **RST**: **reset** connection
  - Connection is aborted (e.g., abnormal op) and application layer notified
- **SYN:** request a connection
- **FIN:** sender finishes sending, but still needs to get a FIN from receiver

TCP Header



Window Size (16 bits to advertise window size)

- Used for **flow and congestion 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
    - Pseudoheader: simplified header created by src and dest., not transmitted.

TCP Header

Options

- Variable length
- NOP (No Operation) option is used to pad TCP header to multiple of 32 bits
- Time stamp is used for:
  - Round trip measurements
  - Distinguish wrap around SNs for high speed routers

Options

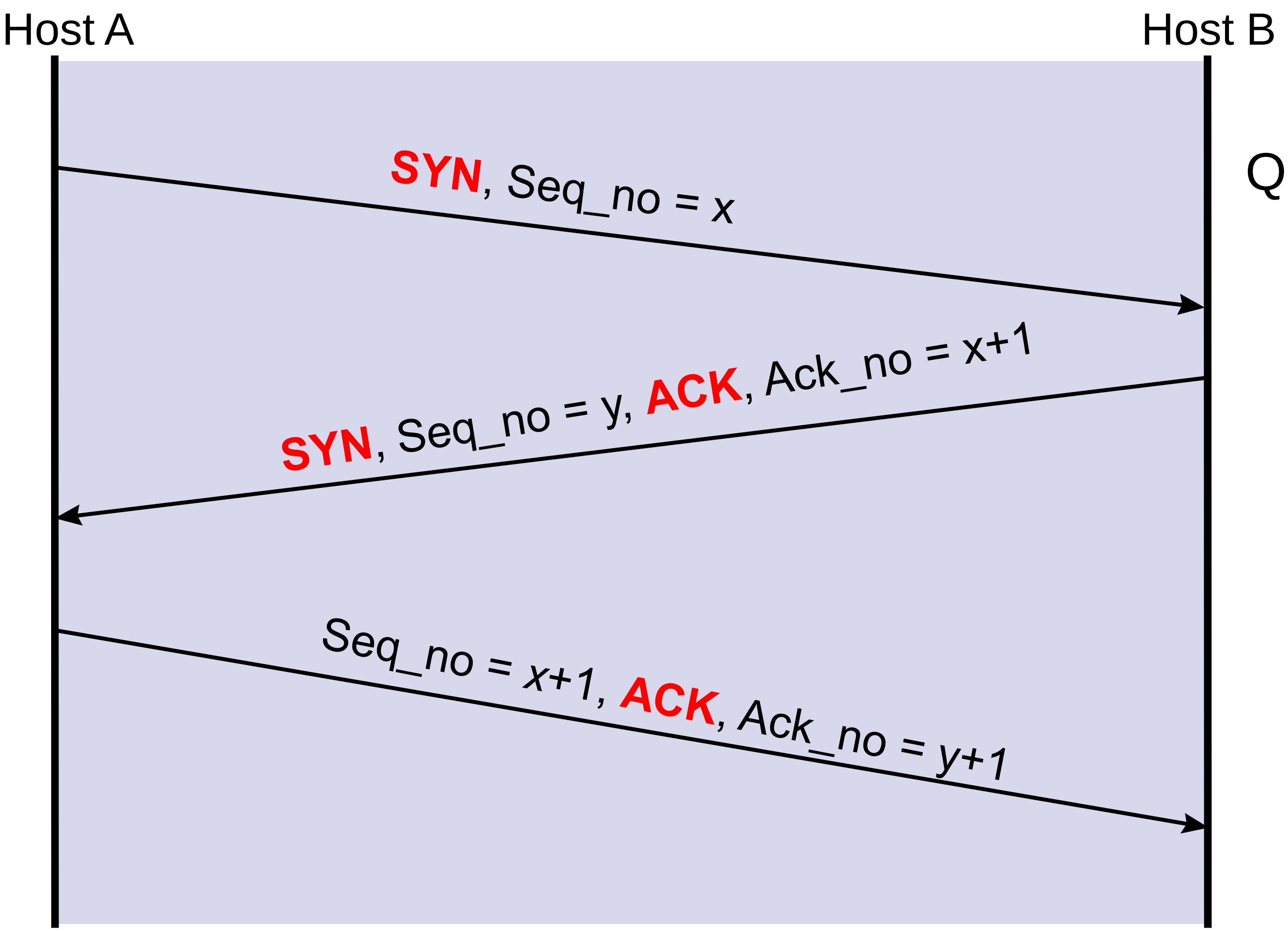
- Maximum Segment Size (MSS) option specifies largest segment a receiver wants to receive
  - Specified during connection setup.
- Window Scale option increases TCP window from 16 to 32 bits

Outline

- TCP – Quick Overview
- TCP Header
- TCP Connection Management
- TCP Congestion Control

TCP Connection Establishment

- "**Three-way Handshake**"
- ISN's protect against segments from prior connections

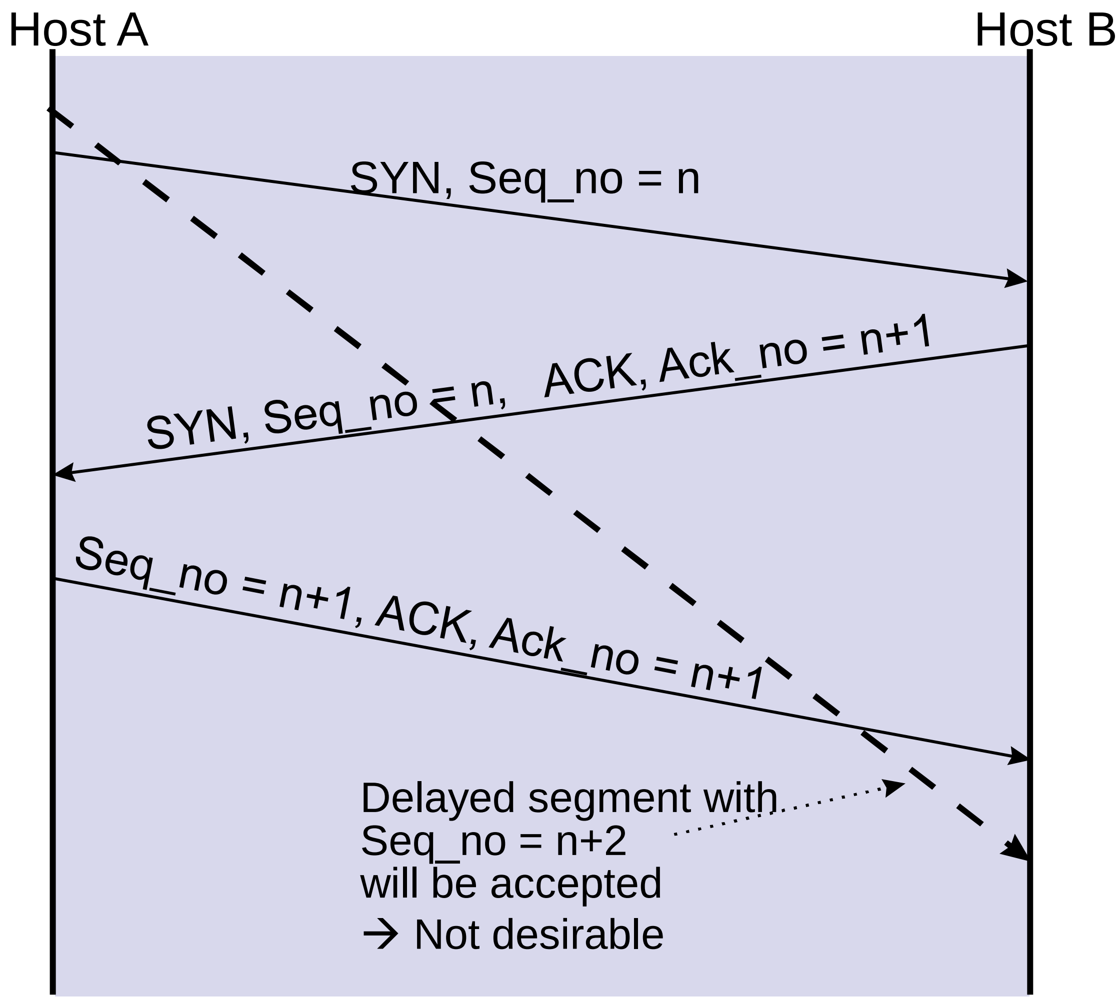


Q: What is the ACK no from receiver? From the sender? Seq. no?

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 (ISN is increased by 1 every four microseconds)
- High bandwidth connections pose a problem
  - Use timestamps to distinguish wrap around SNs

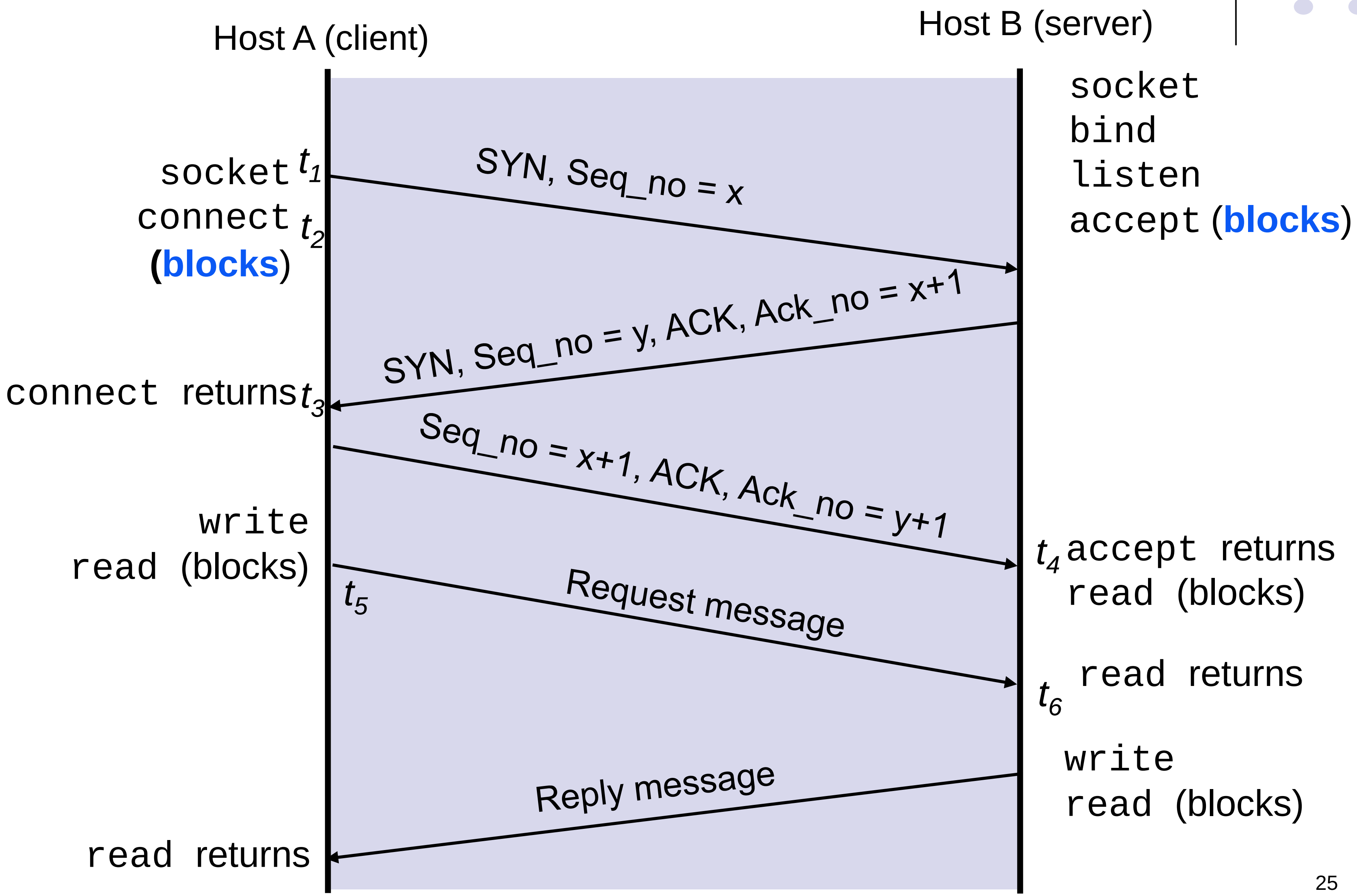
If host always uses the same ISN (p.609)



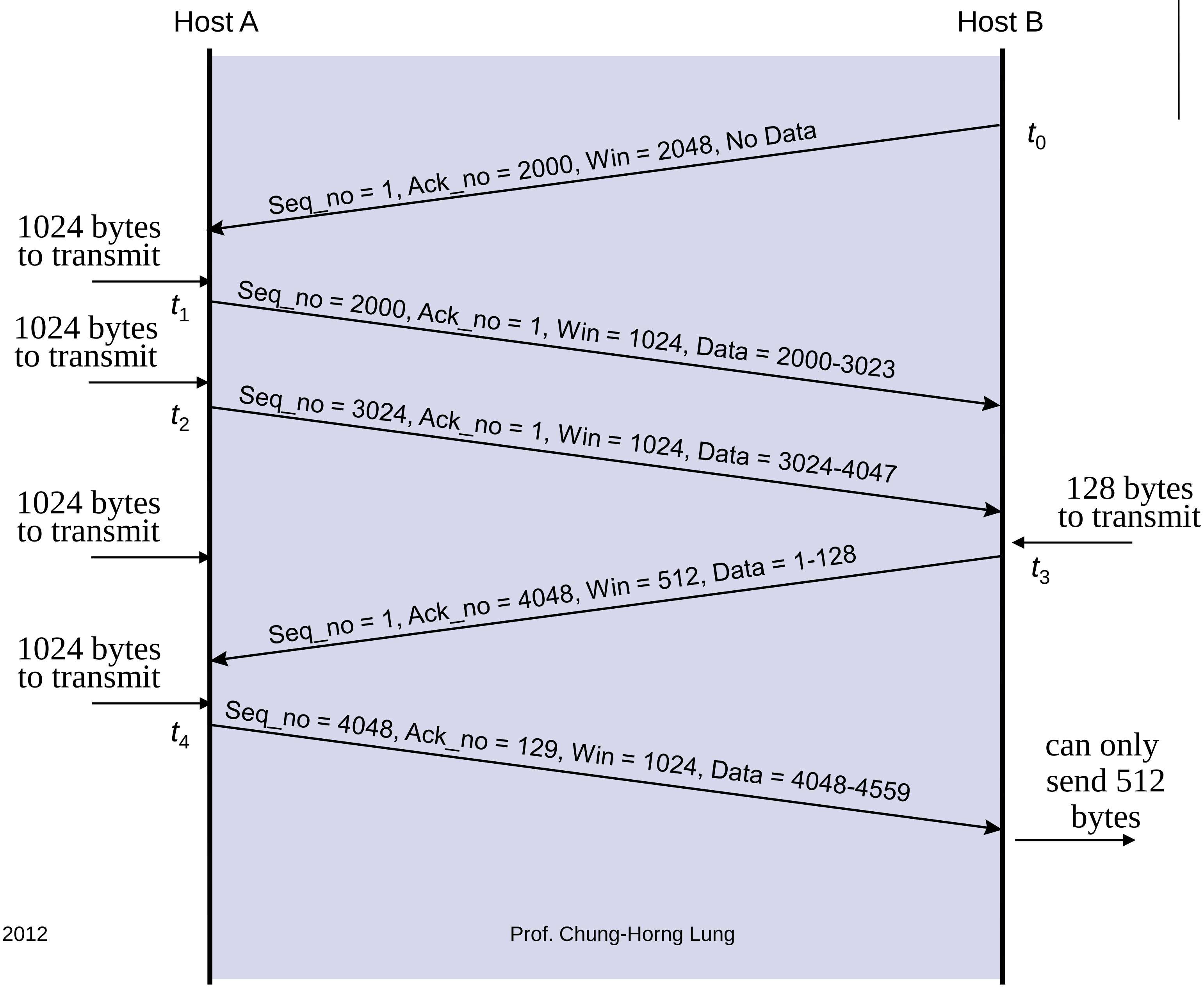
Maximum Segment Size

- Maximum Segment Size (MSS)
  - 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

Client-Server Application



TCP Window Flow Control



*

Nagle's Algorithm

- Situation: user types 1 character at a time
  - Transmitter sends TCP segment per character (41Bytes)
  - Receiver sends ACK (40Bytes)
  - Receiver echoes received character (41Bytes)
  - Transmitter ACKs echo (40 Bytes)
  - 162 bytes transmitted to transfer 1 character!
- Solution:
  - TCP sends data & waits for ACK
  - New characters buffered
  - Send new characters when ACK arrives
  - Algorithm adjusts to RTT
    - Short RTT send frequently at low efficiency
    - Long RTT send 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 advertise window until window is at least ½ of receiver buffer or maximum segment size
  - Transmitter refrains from sending small segments

Sequence Number Wraparound

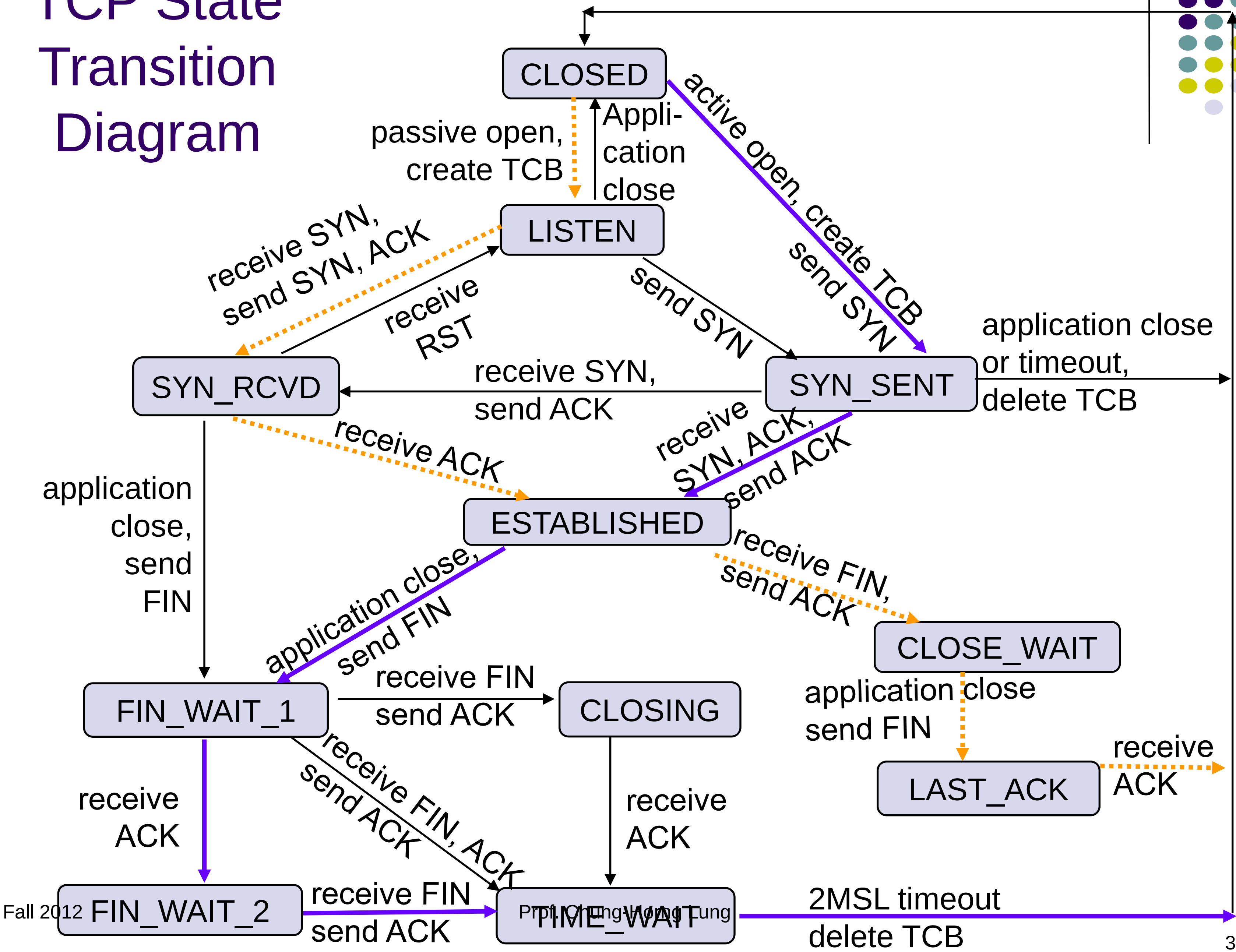
- $2^{32} = 4.29 \times 10^9$ bytes = 34.3×10^9 bits
  - At 1 Gbps, sequence number wraparound in 34.3 seconds.
- Timestamp option: Insert 32 bit timestamp in header of each segment
  - Timestamp + sequence no → 64-bit seq. no
  - Timestamp clock must:
    - tick forward at least once every 2^{31} bits
    - Not complete cycle in less than one MSL
    - Example: clock tick every 1 ms @ 8 Tbps wraps around in 25 days

Delay-BW Product & Advertised Window Size

- Suppose RTT=100 ms, R=2.4 Gbps
  - # bits in pipe → 30 Mbytes
- If single TCP process occupies pipe, then required advertised window size is
  - RTT x Bit rate = 30 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 State Transition Diagram

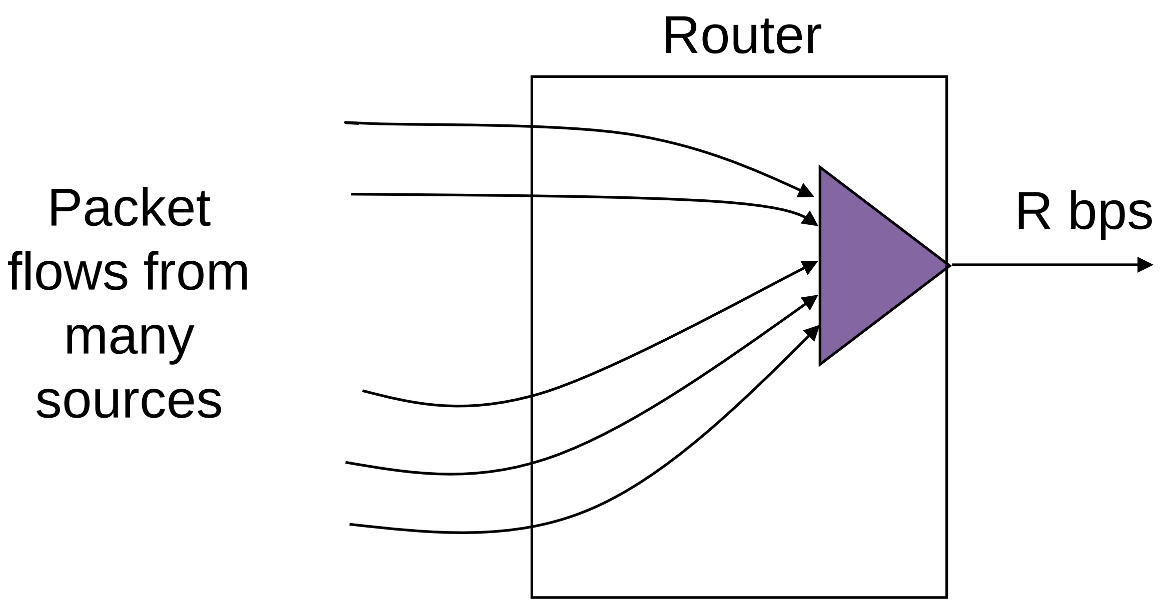


Outline

- TCP – Quick Overview
- TCP Header
- TCP Connection Management
- TCP Congestion Control

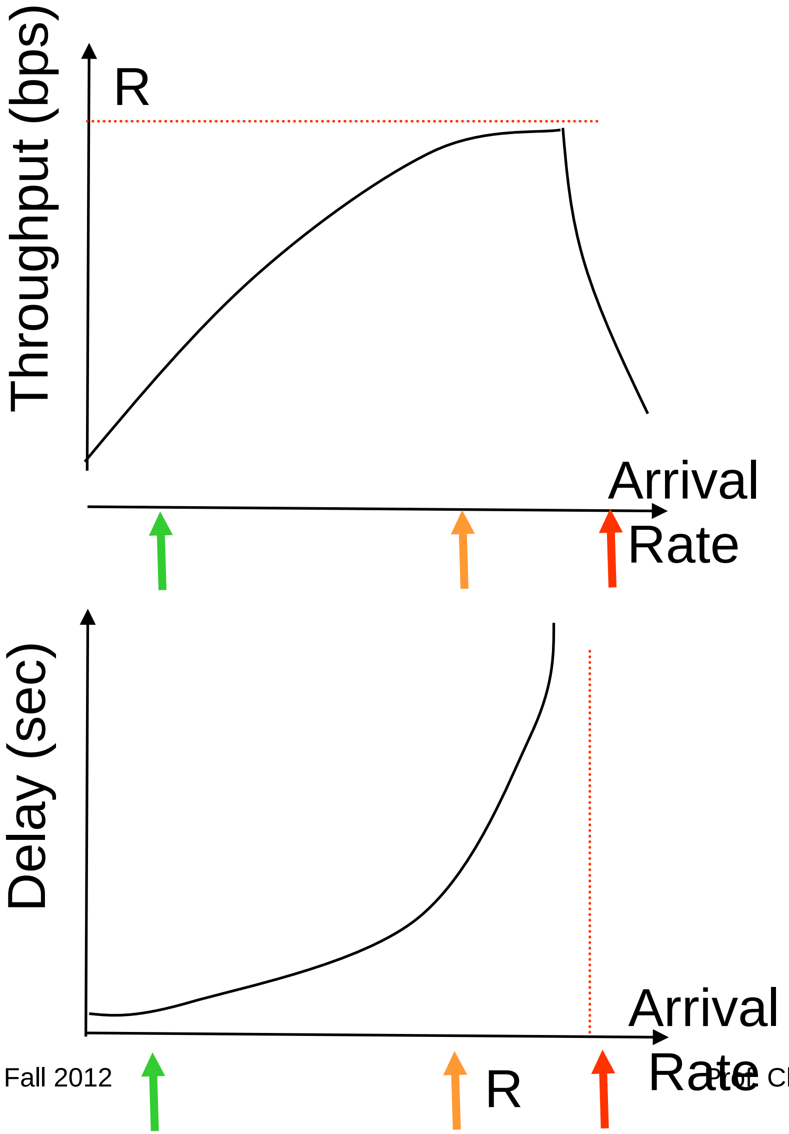
TCP Congestion Control

- *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 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

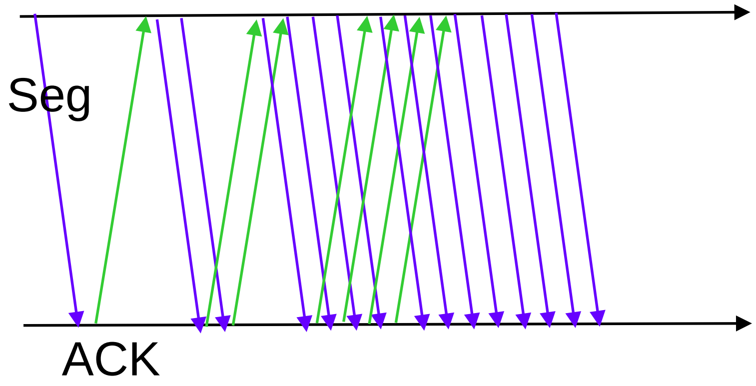
Congestion Window

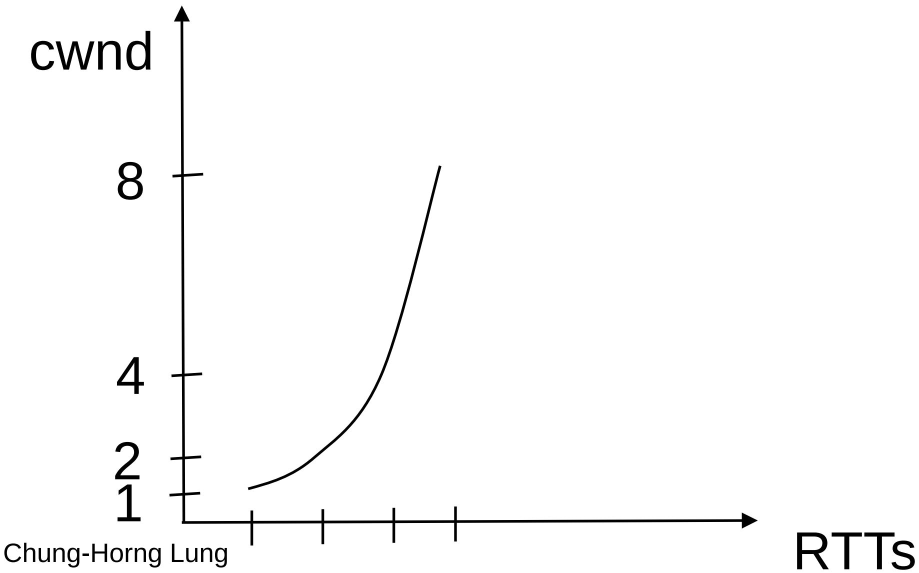
*

- 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 aggresively
- 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**

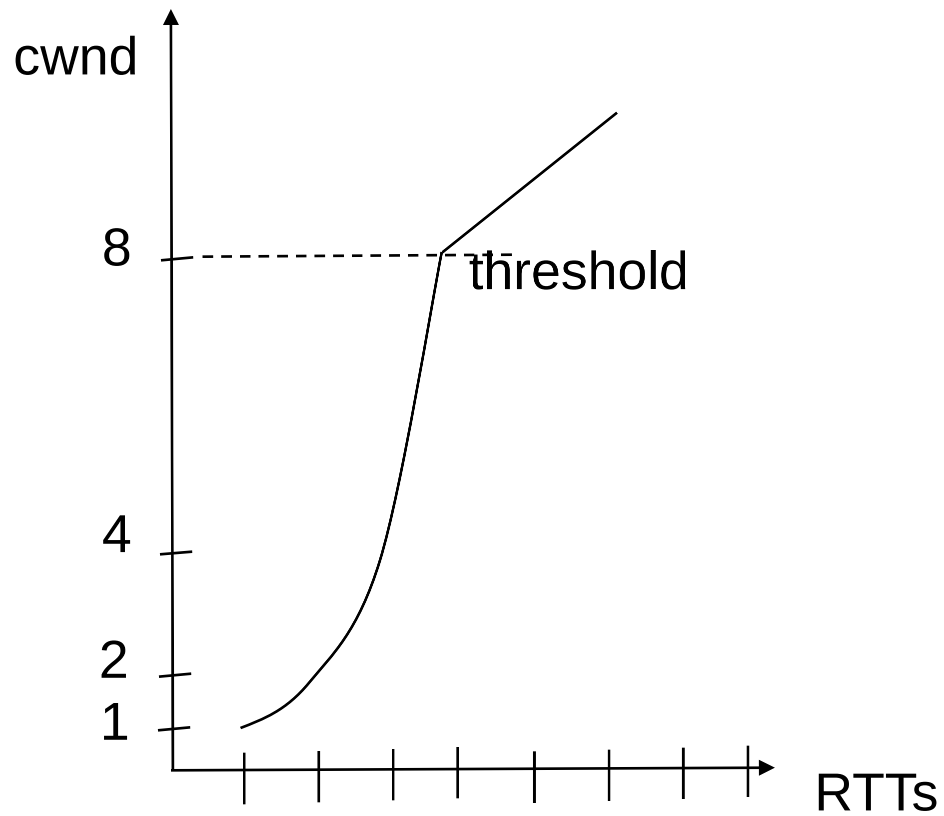




TCP Congestion Control: Congestion Avoidance

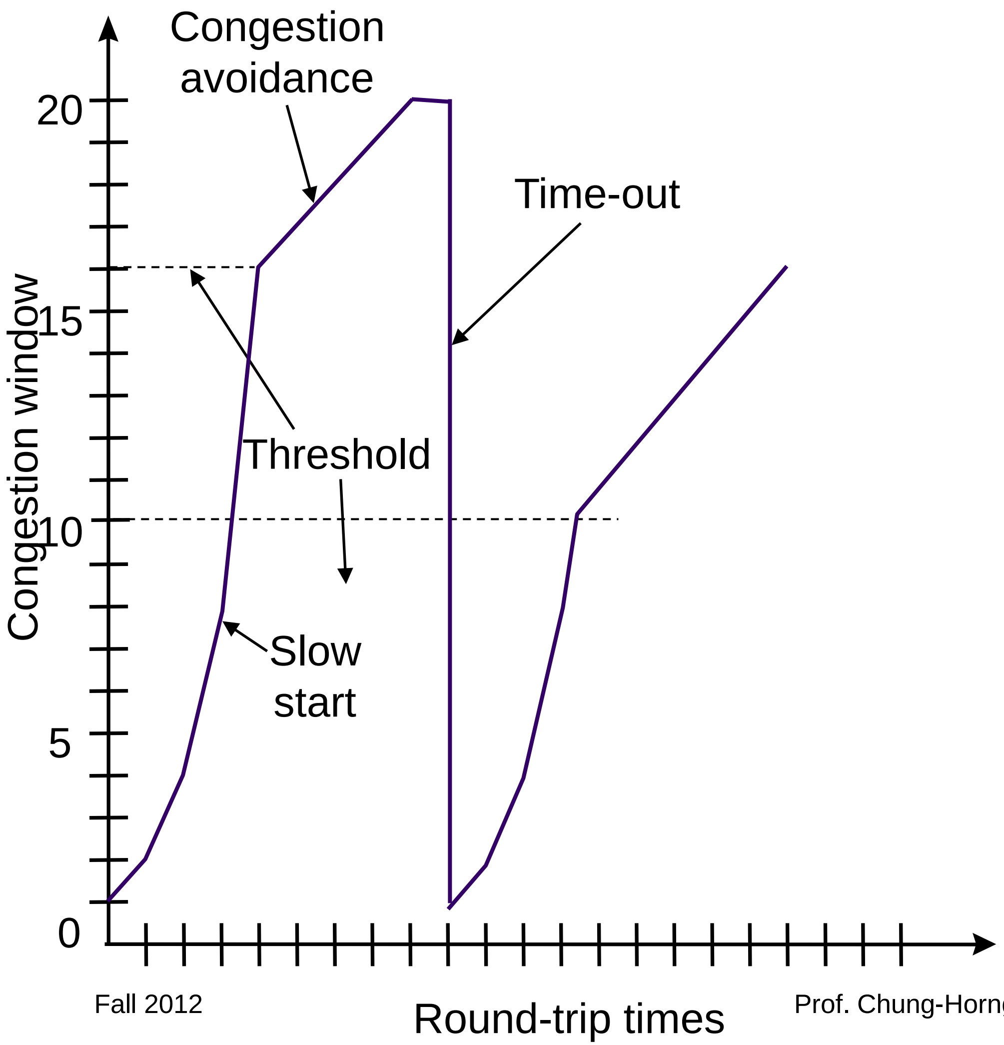
*

- 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)
  - cwnd grows **linearly** with time



TCP Congestion Control: Congestion

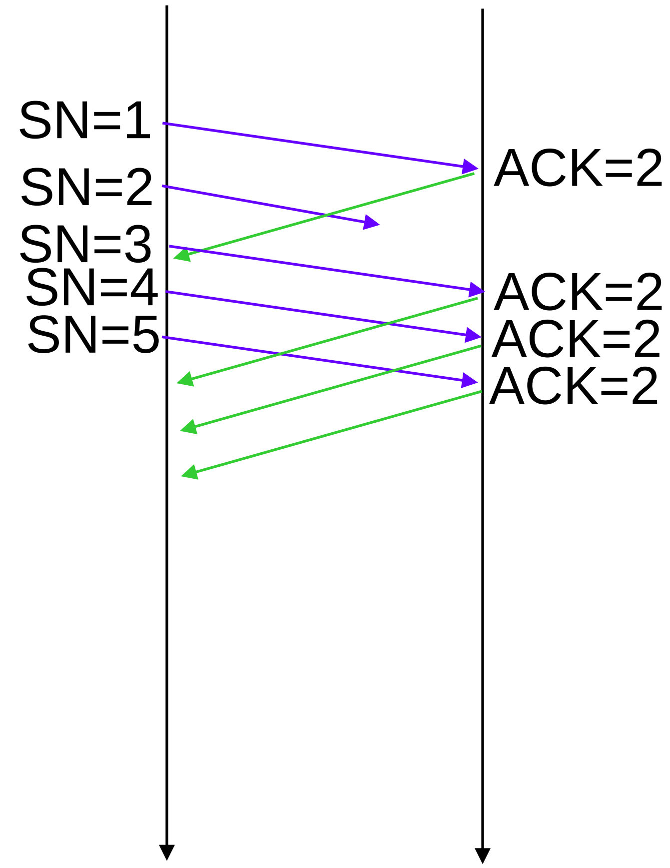
*



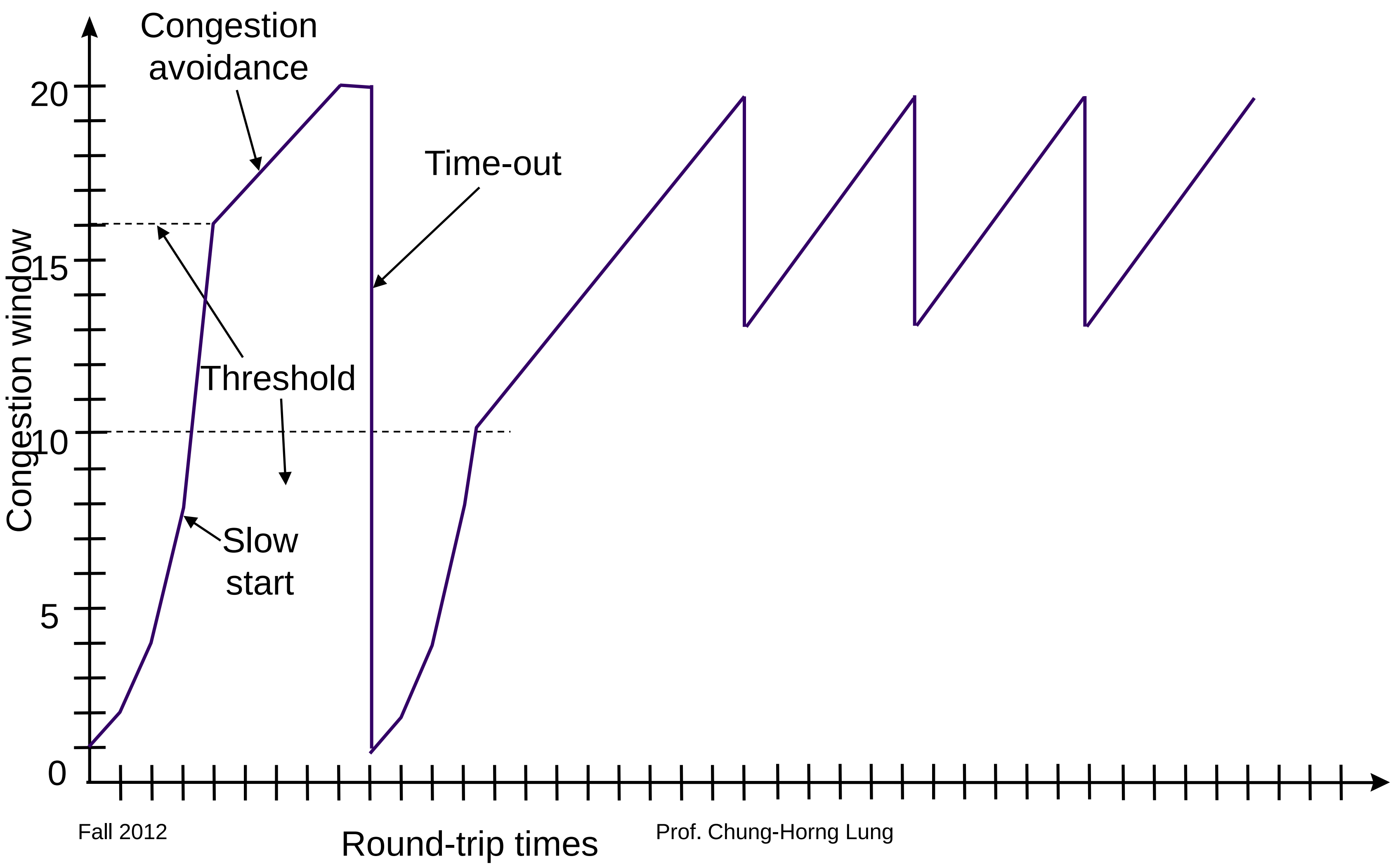
- Congestion is detected upon timeout or receipt of duplicate ACKs
- Assume current cwnd corresponds to available bandwidth
- Adjust congestion threshold = ½ x current cwnd
- Reset cwnd to 1
- Go back to slow-start
- Over several cycles expect to converge to congestion threshold equal to about ½ the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

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



TCP Retransmission Timeout

- TCP retransmits a segment after timeout period
  - Timeout too short: excessive number of retransmissions
  - Timeout too long: recovery too slow & slow reaction to loss
  - Timeout depends on RTT: time from when segment is sent to when ACK is received
- Round trip time (RTT) in Internet is highly variable
  - Routes vary and can change in mid-connection
  - Traffic fluctuates, multiple traffic flows
- TCP uses adaptive estimation of RTT
  - Measure RTT each time ACK received: M_n

$$t_{RTT}(\text{new}) = \alpha\, t_{RTT}(\text{old}) + (1 - \alpha)\, M_n$$

  - $\alpha = 7/8$ typical

RTT Variability

- Estimate variance σ^2 of RTT variation
- Estimate for timeout:

$$t_{out} = t_{RTT} + k\, d_{RTT}$$

- If RTT highly variable, timeout increase accordingly
- If RTT nearly constant, timeout close to RTT estimate

- Approximate estimation of deviation

$$d_{RTT}(new) = \beta\, d_{RTT}(old) + (1-\beta)\, |\, M_n - t_{RTT}\, |$$

$$\boxed{t_{out} = t_{RTT} + 4\, d_{RTT} \quad (I.e.\ k=4)}$$

RTT and Timeout: an Example

- For packet (n), use timeout (n-1).

- Example: At time 0 the TCP round trip time is actually 30 msec. For the following packets, acknowledgements came back after 26, 32, 24 msec, respectively. Apply the dynamic timeout Jacobson's algorithm to calculate the best timeout estimate at the end. Use α =0.9 and β =0.9. (The notations used in the following are simplified.)

- **Assume at the start d(0)= 0 msec and RTT(0) = 30.**
- **Measured values: M(0)=30, M(1)=26, M(2)=32, M(3)=24.**

- **RTT(n) = α * RTT(n-1) + (1- α) *M (n)** — **d(n) = β * d (n-1) + (1- β) * | RTT(n) -M(n) |**
- **RTT(1) = 0.9 x 30 + 0.1 x 26=29.6** — **d(1)= 0.9 x 0 + 0.1 x |29.6-26|=0.36**
- **RTT(2) = 0.9 x 29.6 + 0.1 x 32=29.84** — **d(2)= 0.9 x .36 + 0.1 x |29.84-32|=0.54**
- **RTT(3) = 0.9 x 29.84 + 0.1 x 24=29.256** — **d(3)= 0.9 x 0.54 + 0.1 x |29.256-24|=1.01**

- **Timeout(n) = RTT(n) + 4 * d(n)**

  Timeout(3) = RTT(3) + 4 * d(3) = 29.256 + 4 x 1.01 = 33.3 msec