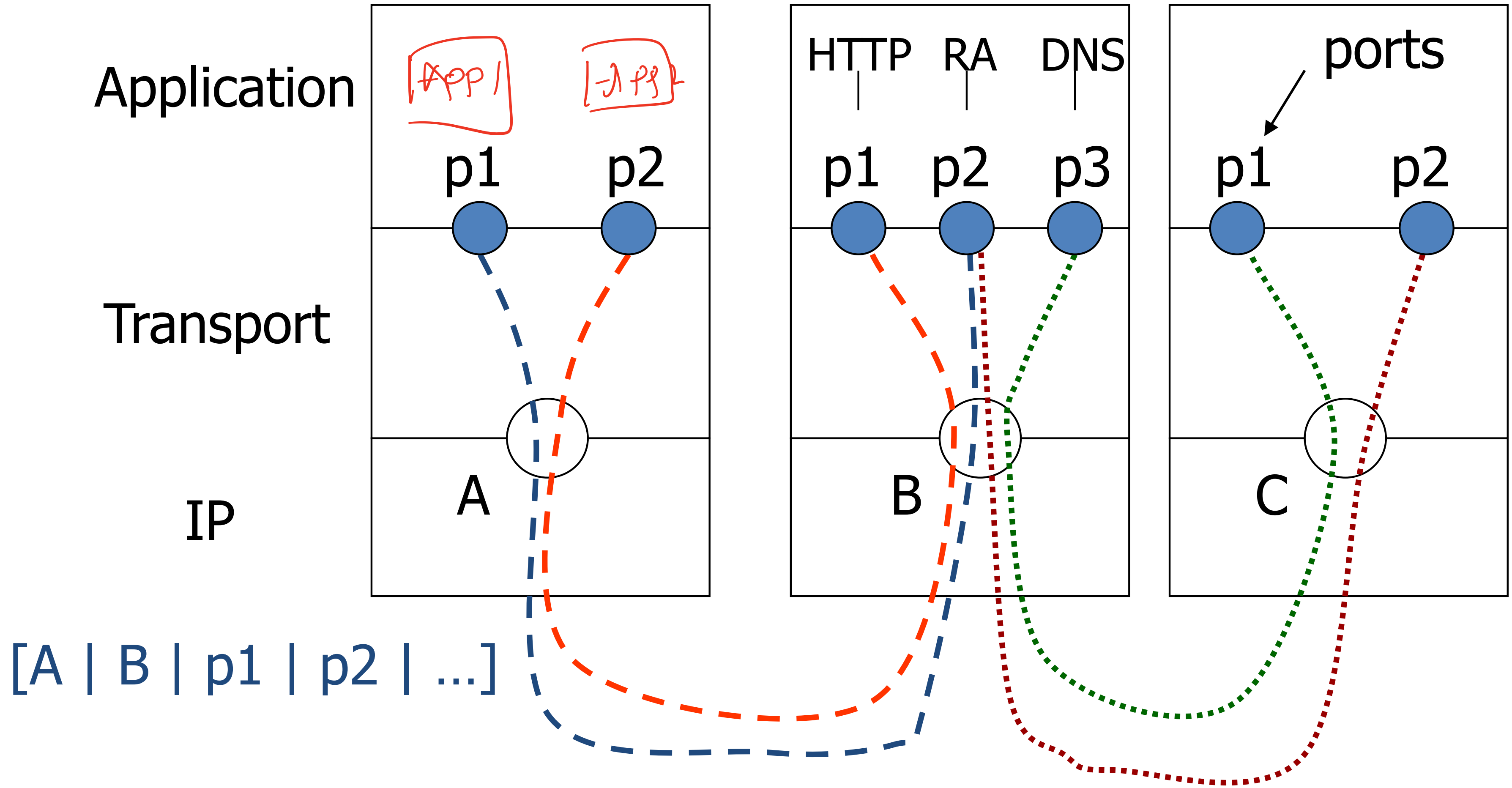
Transport

EECS 122, Spring 2024

Shyam Parekh

Transport Layer Multiplexing



Two prevalent choices:

UDP: Not reliable

TCP: Ordered, reliable, well-paced

Ports

- Need to decide which application gets which packets
- Solution: Create "*socket*" as combination of *port* & *IP address*
- Client must know server's port
- Separate 16-bit port address space for UDP and TCP
  - (src IP, src port, dst IP, dst port) uniquely identifies a TCP or UDP connection
- Well-known ports (0-1023): everyone agrees which services run on these ports
  - e.g., ssh:22, http:80, ftp (control):21
- Ephemeral ports (most 1024-65535): given to clients
  - e.g., chatclient gets one of these

UDP

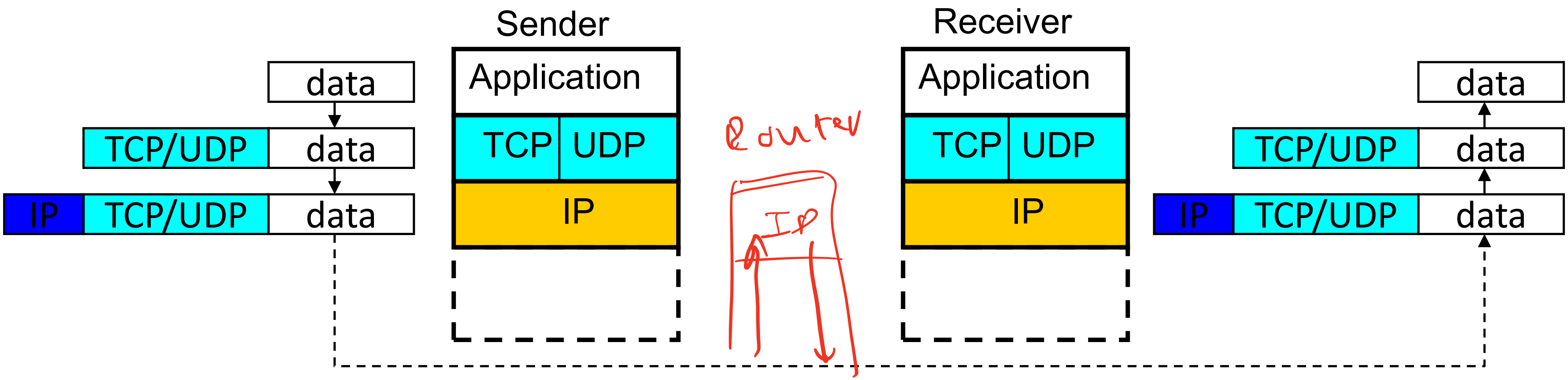
- User Datagram Protocol
- minimalistic transport protocol
- same best-effort service model as IP
- maximum datagram size is about 64KB, but is usually much smaller in practice
- provides multiplexing/demultiplexing to IP
- does not provide congestion control
- advantage over TCP: does not increase end-to-end delay over IP
- application example: video/audio streaming

TCP

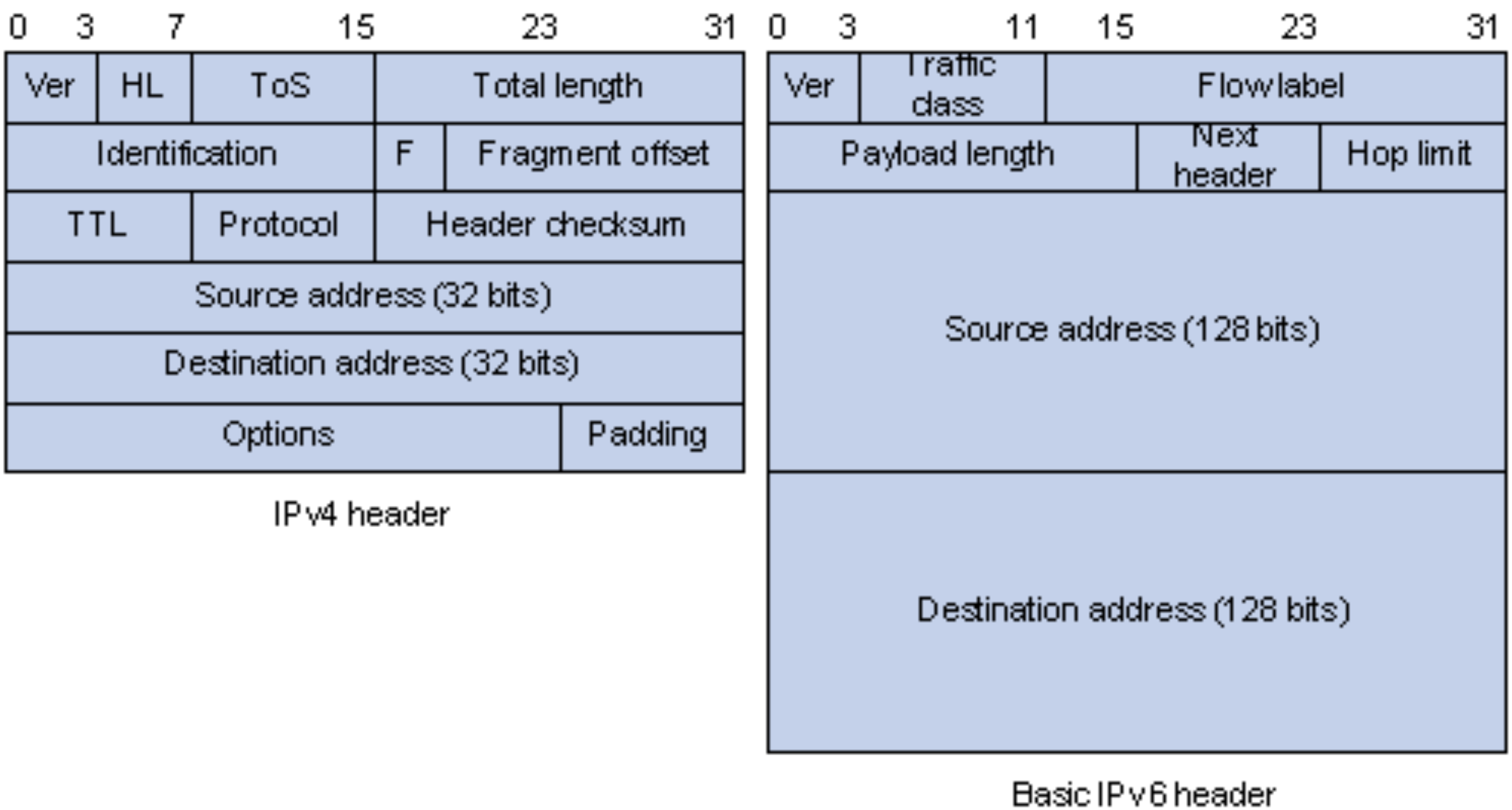
- Transmission Control Protocol
- reliable, in-order, and at most once delivery
- Maximum Segment Size (MSS) is determined at connection establishment (default is 536 bytes)
- provides multiplexing/demultiplexing to IP
- provides congestion control and avoidance to avoid network congestion
- provides flow control to avoid overwhelming the receiver
- increases end-to-end delay over IP
- e.g., file transfer, chat

Review of Headers

- IP header → used for IP routing, fragmentation, header error detection (in IPv4), …
- UDP header → used for multiplexing/demultiplexing, error detection (optional), …
- TCP header → used for multiplexing/demultiplexing, error detection, flow and congestion control, …



IP Headers (Details FYI)

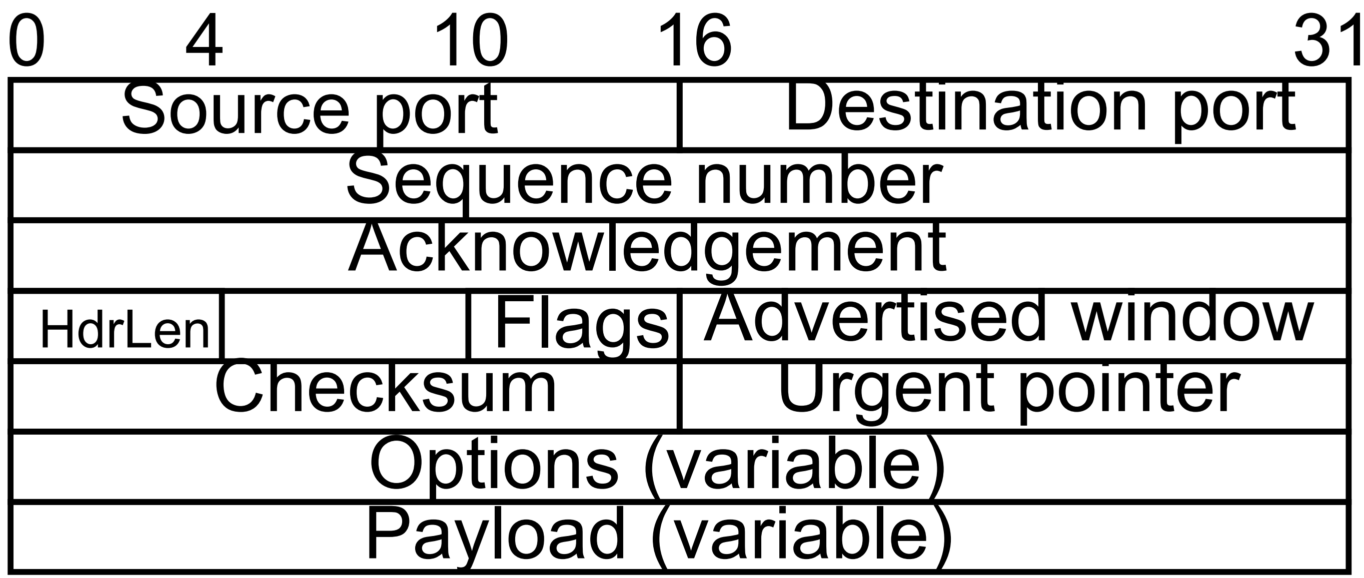


UDP Header (Details FYI)

0	16 31
Source port	Destination port
UDP length	UDP checksum
Payload (variable)	

- UDP length is UDP packet length
  (including UDP header and payload, but not IP header)
- Optional UDP checksum is over UDP packet

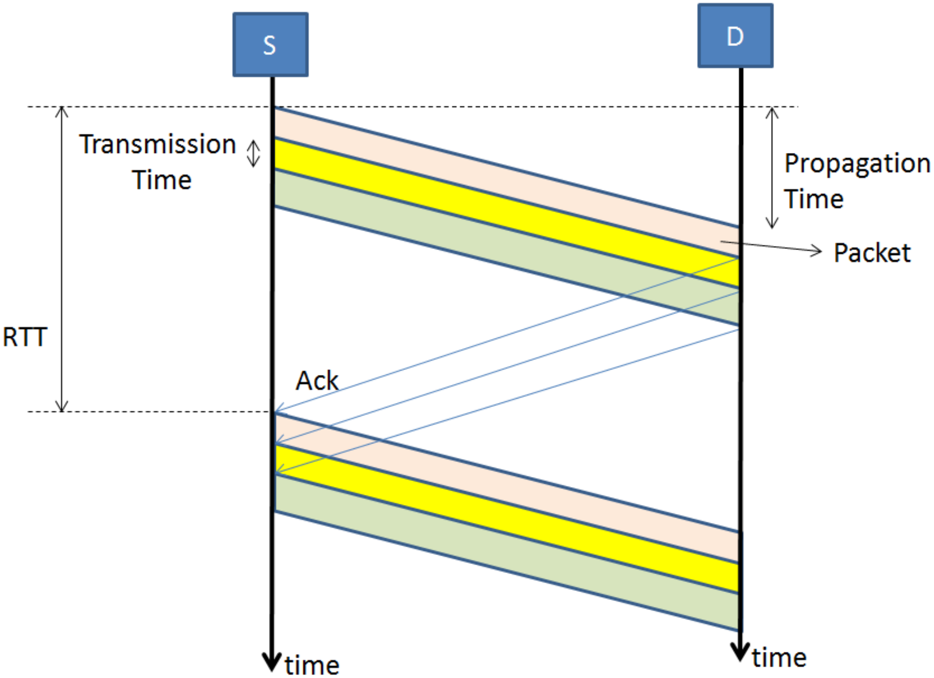
TCP Header (Details FYI)



- Sequence number, acknowledgement, and advertised window – used by sliding-window based flow control
- Flags (6 bits):
  - SYN, FIN – establishing/terminating a TCP connection
  - ACK – set when Acknowledgement field is valid
  - URG – urgent data; Urgent Pointer says where urgent data ends
  - PUSH – don't wait to fill segment
  - RESET – abort connection

Sliding Window

- Packets are sequentially numbered.
- Receiver sends cumulative acks by indicating which packet is expected next (and thus acknowledging all prior packets).
- Sender is allowed to have W number of packets that have not been acked yet (referred to as outstanding packets).
  - If W = 1, it's referred to as the stop-and-wait protocol.
  - Observe that sender "slides" the transmit window of size W forward upon receiving an ack.
- Let RTT denote the average Round-Trip Time (from the time sender starts sending a packet until an ack is received) in seconds.
  - Then, assuming no packet loss, average throughput is W/RTT packets/sec (or bytes/sec or bits/sec).
- Note: In TCP, Window Size is actually measured in bytes.



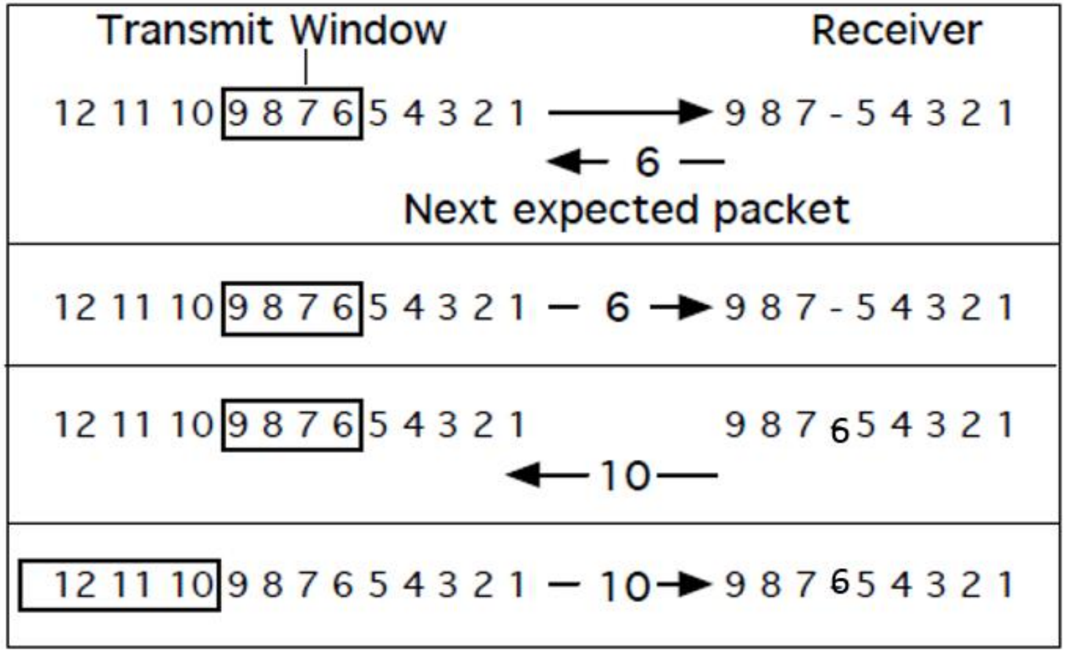
W = 3

$\boxed{\text{Thpt} \cdot \text{RTT} = W}$

app. Little's Law

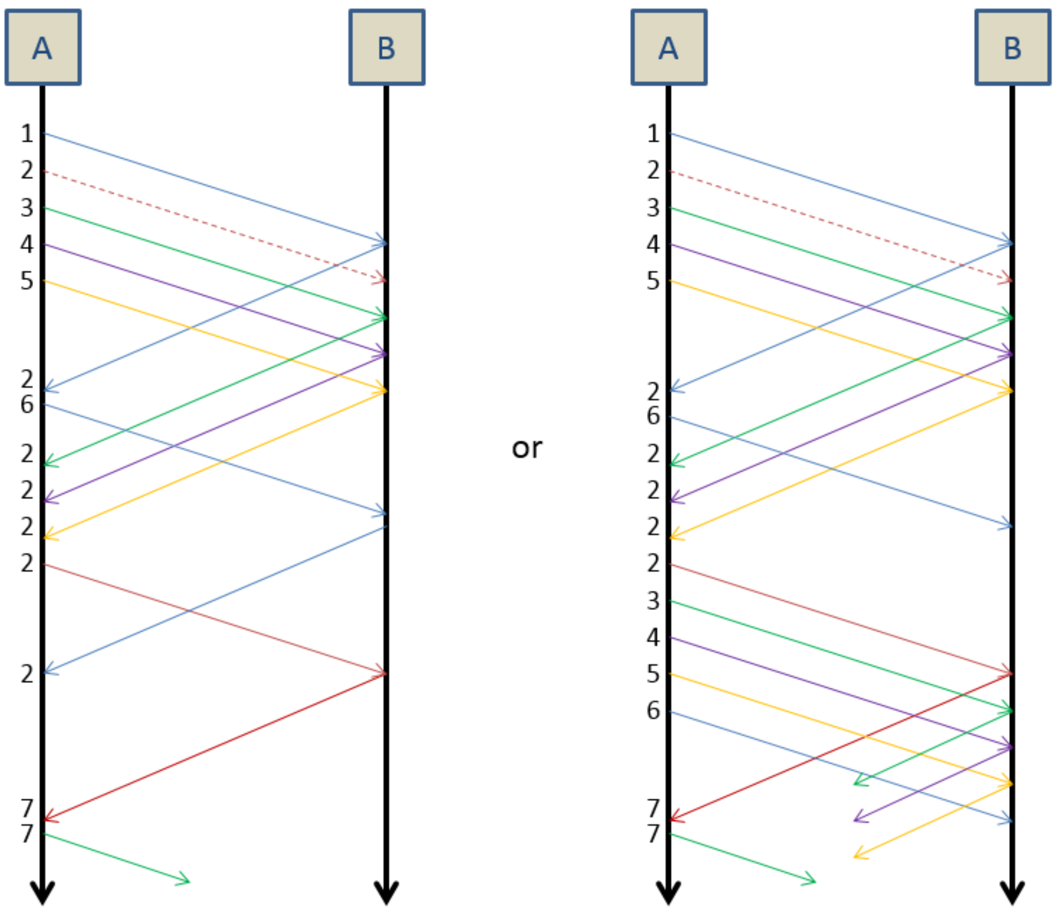
Sliding Window (2)

- Sliding Window scheme also specifies how many out of order frames can be accepted at the receiver (W_R - 1).
- If W = N and W_R = 1, the Sliding Window scheme is referred to as Go-Back-N ARQ (Automatic Repeat request).
- Example: W = 4 and W_R = 4 with packet loss



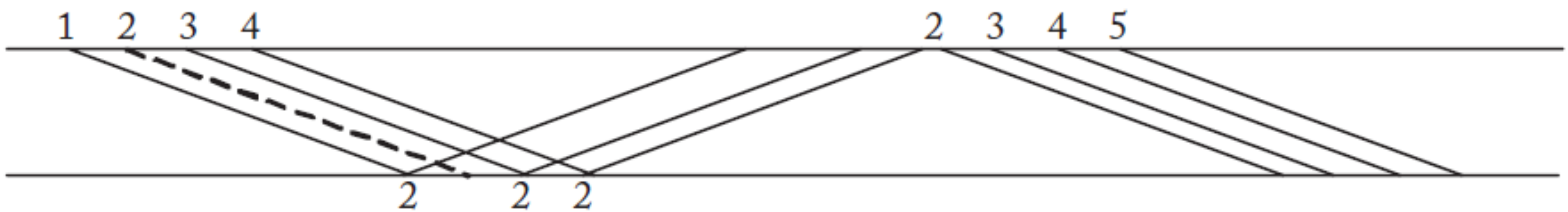
Sliding Window (3)

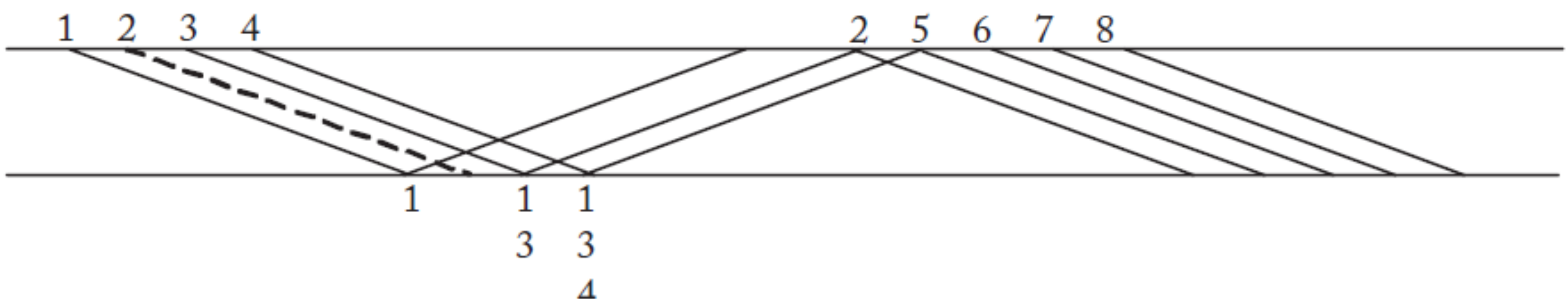
- Example: W = 5 and W_R = 5.
  - Two valid implementations differing in if the sender attempts to retransmit “optimistically”.



- HW 4 due date 3/20 11:59 PM
  HW 5 release also postpones.
-

Selective Acknowledgements





TCP

- TCP implements congestion and flow control using a variation of the generic Sliding Window scheme discussed earlier
  - TCP makes use of byte numbering (as opposed to packet numbering)
  - ACK # = next expected byte #
  - Effective Transmit Window = min{Congestion Window, Receiver Advertised Window (RAW)}, where Congestion Window is a dynamically changing version of W, and RAW is based on W_R
  - IF RAW is not limiting, Instantaneous Throughput = Congestion Window/RTT
- By adjusting the Congestion Window, TCP addresses congestion
- Rollovers of sequence numbers may cause confusion if rollover time is < max lifetime of packet:
  - Transmit: 1, 2, …, N, 1, 2 …N, …
  - Receiver (Rollover Time > Max Packet Lifetime):

    1, 2, …, ~~1~~, …, N, 1, 2, …, N, … (Receiver correctly concludes that second 1 should be discarded)
  - Receiver (Rollover Time < Max Packet Lifetime):

    1, 2, …, N, 1, ~~1~~, 2, …, N, … (Receiver can mistakenly accept second 1, and discard 1)
  - In practice, Rollover Time is quite large and Max Packet Lifetime is relatively small

TCP Timeout Timer

- Use exponential averaging: $T(n)$ is the observed RTT; $A(n)$ and $D(n)$ are the estimated average and "std dev" (really, an approximate estimate of std dev)

$$A(n) = gA(n-1) + (1-g)T(n)$$
$$D(n) = hD(n-1) + (1-h)|T(n) - A(n)|$$
$$\text{Timeout}(n) = A(n) + 4D(n)$$

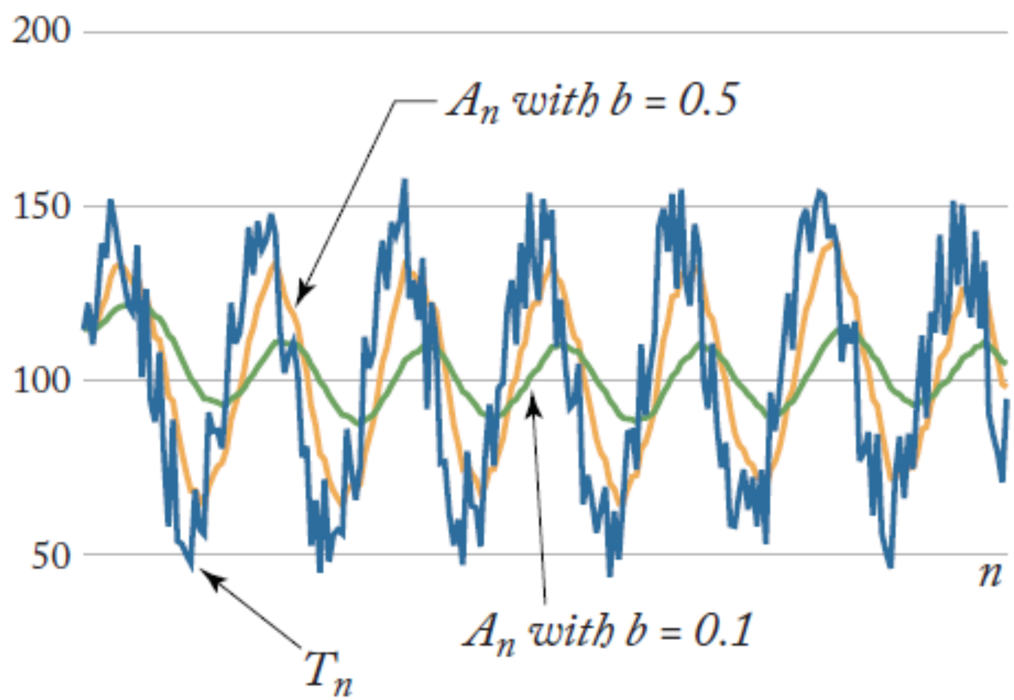
Notes:

1. Measure $T(n)$ only for original transmissions
2. Double Timeout after timeout …
   Justification: timeout indicates likely congestion;
   Further retransmissions would make things worse
3. Recommended values for g & h are 0.875 & 0.75, respectively

$$A(0) = 0$$

$$A(n) = \underbrace{(1-g)} \sum_{i=1}^{n} g^{n-i} \, T(i)$$

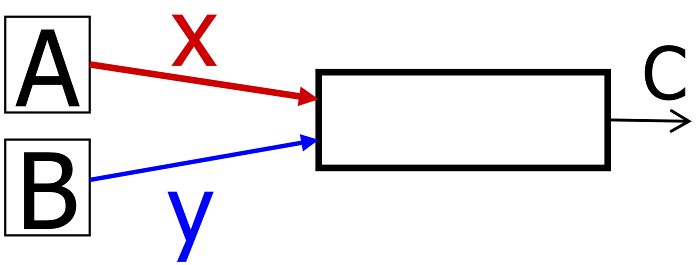
Exponential Averaging for RTT

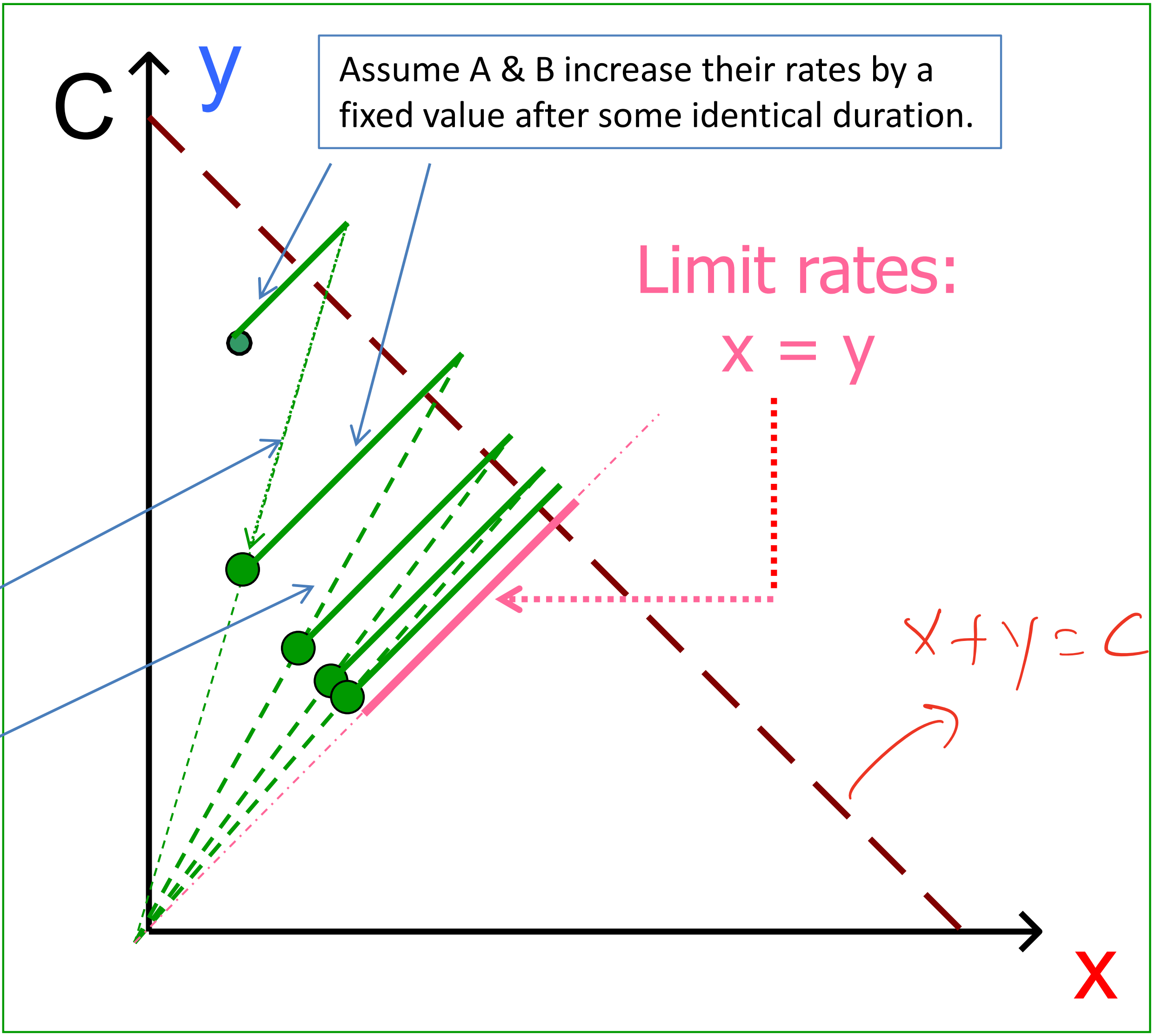


Note: b = (1-g) on the previous slide.

Additive Increase & Multiplicative Decrease (AIMD)

- TCP manages its Congestion Window using AIMD.
- In the illustration below, X and Y are rates for the connections from the sources A and B, respectively. Here, we assume equal RTT for the two connections.





Both A & B cut their rates by half.

TCP Phases for Evolution of Congestion Window

- Slow Start CWND
  - While (W < Slow Start Threshold) SSTHRESH
    - ➢ W = W + 1 for each new ack
    - ➢ Recovery Mechanism: Timeout

  CWND grows exponentially

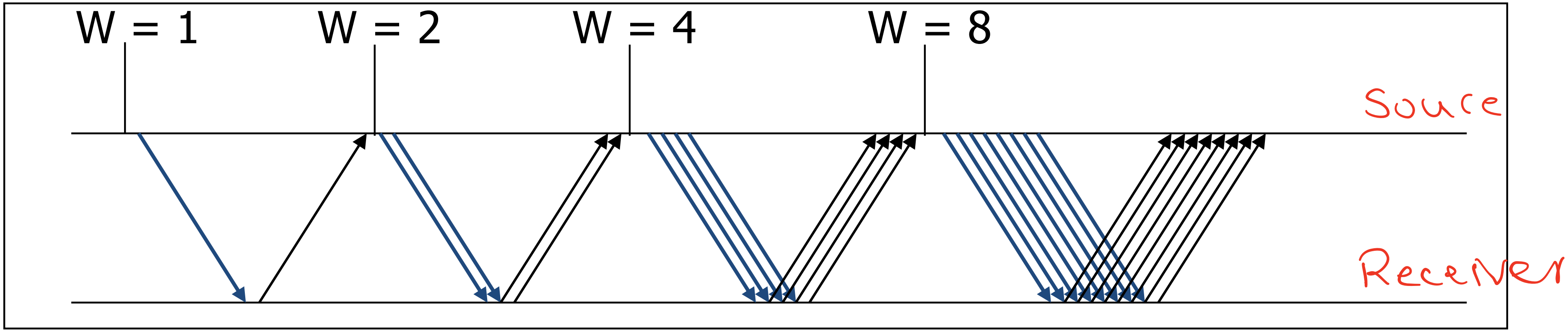
- Congestion Avoidance
  - While (W ≥ Slow Start Threshold)
    - ➢ W = W + 1/W for each new ack
    - ➢ Recovery Mechanism: Fast Transmit/Recovery and Timeout

  CWND grows Linearly

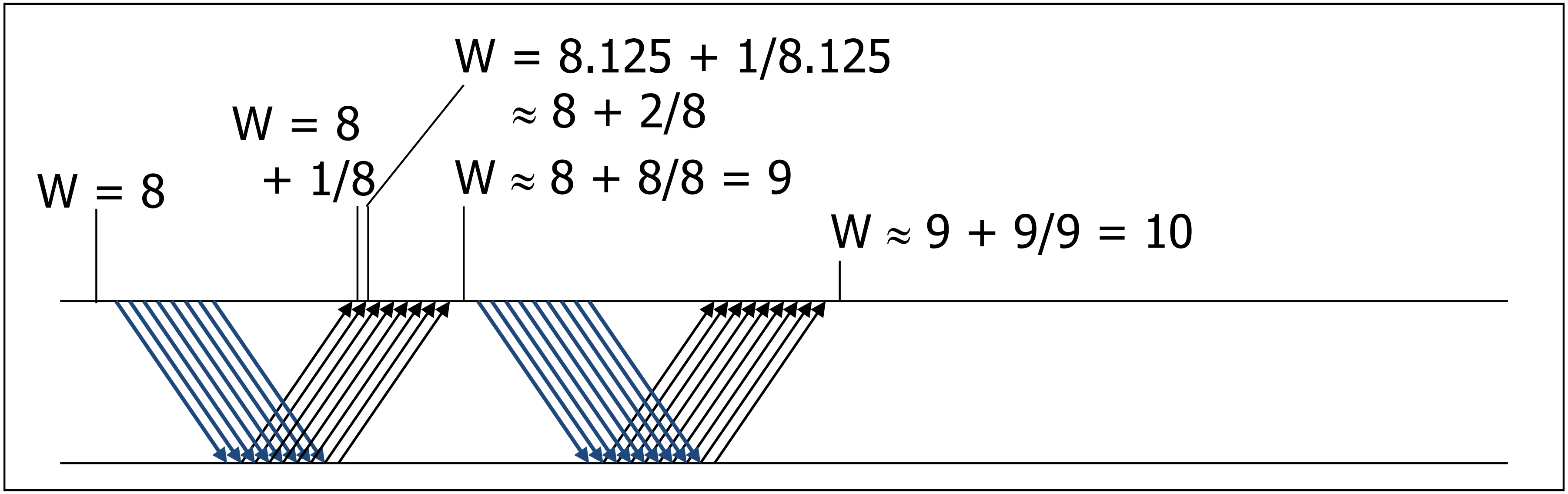
We next examine the details of the above phases

Illustration of Window Updates

- Exponential: W = W + 1 at each ACK, W roughly doubles every round-trip

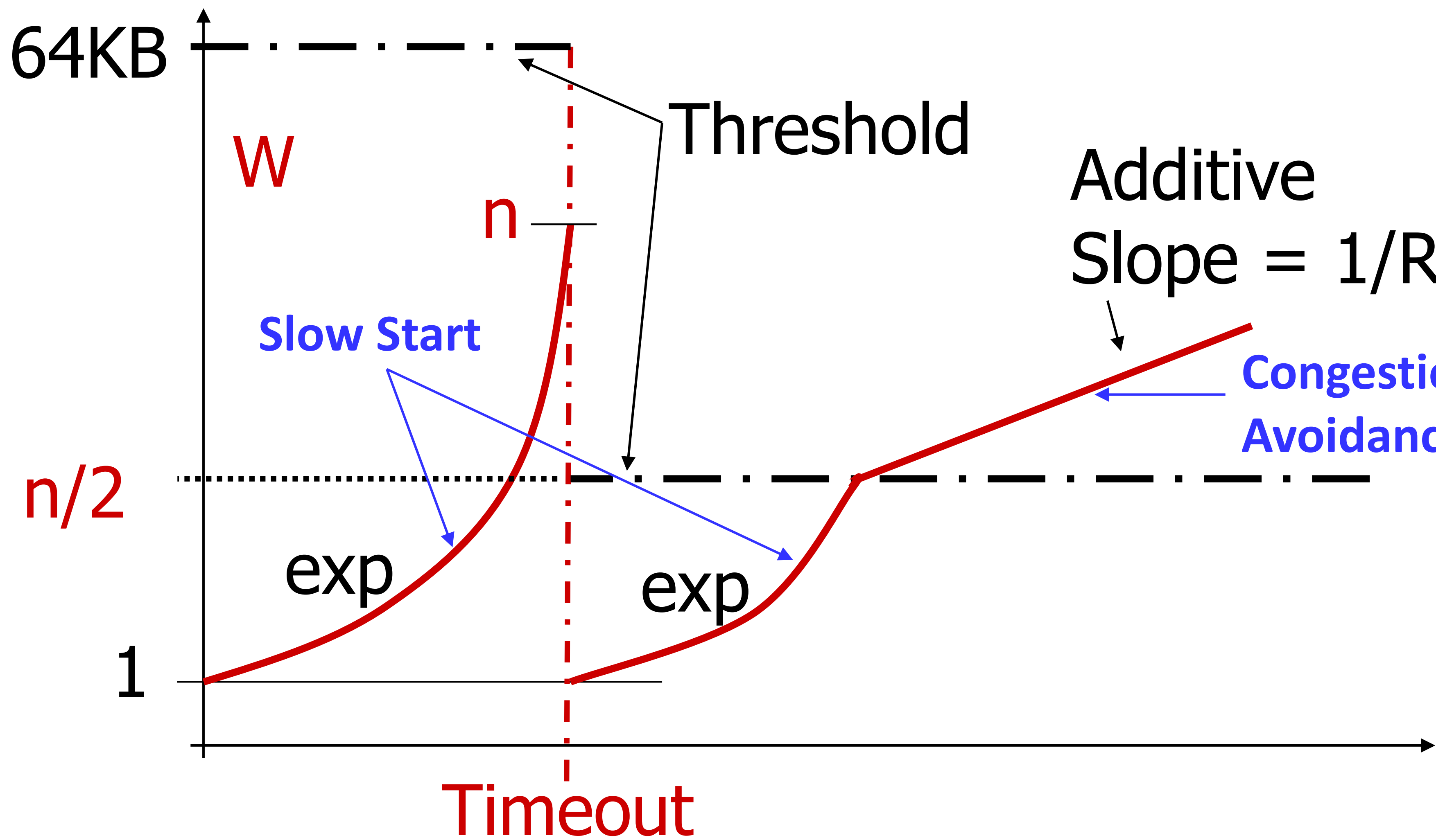


- Additive: W = W + 1/W at each ACK, W roughly increases by 1 every round-trip



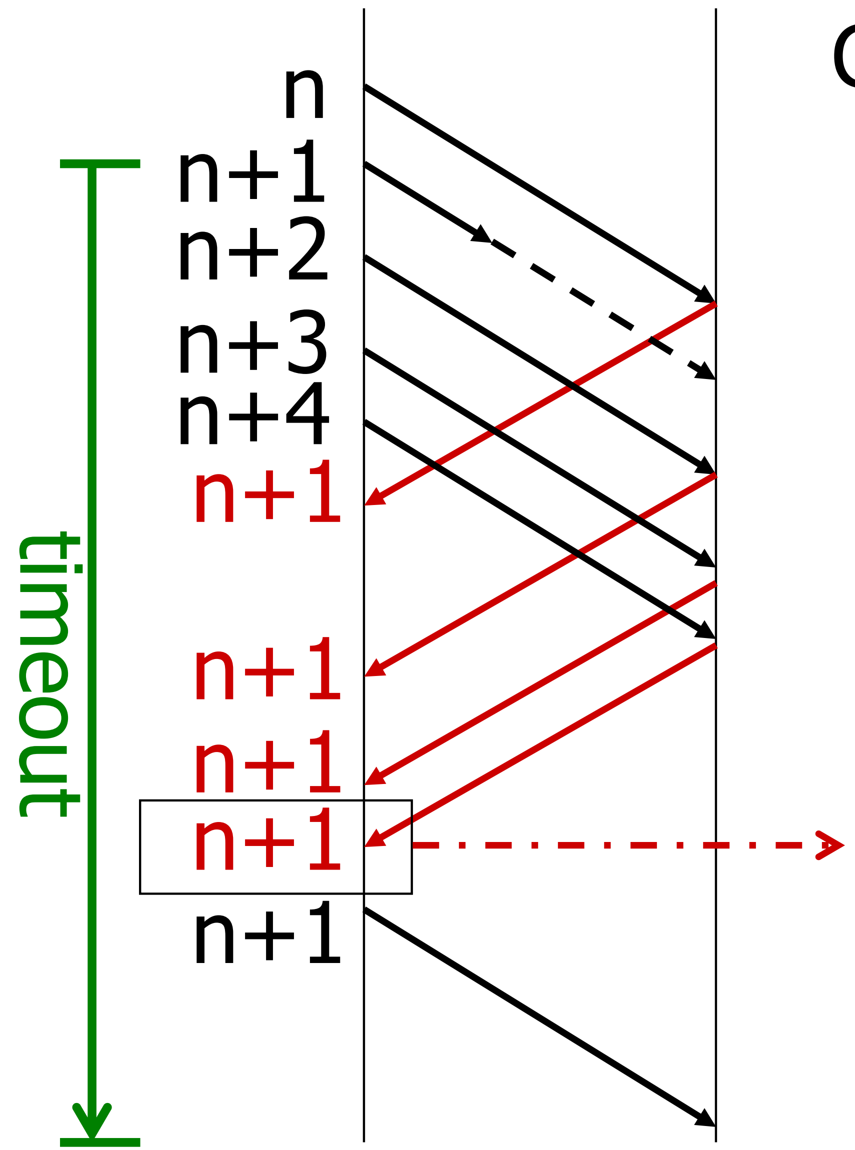
Refinements: Slow Start & Congestion Avoidance

- Objective: Discover available BW quickly
- Solution: Exponential increase of window (Slow Start)
- Probing for more BW: Additive increase (Congestion Avoidance)



Refinements: Fast Retransmit

TCP Reno



Cumulative ACKs:

ACK # = next expected #

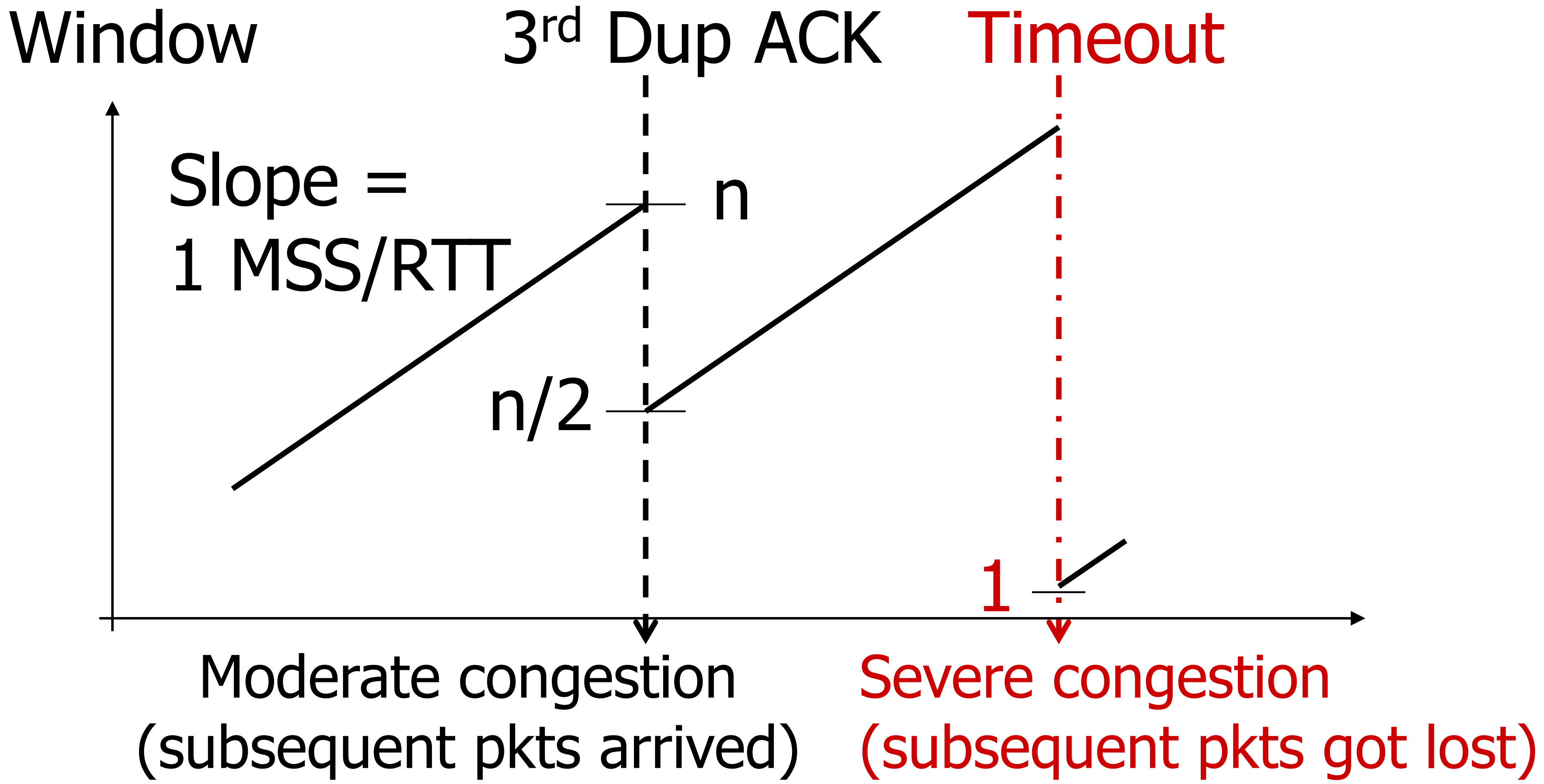
3rd duplicated ACK:

→ likely packet loss

→ retransmit

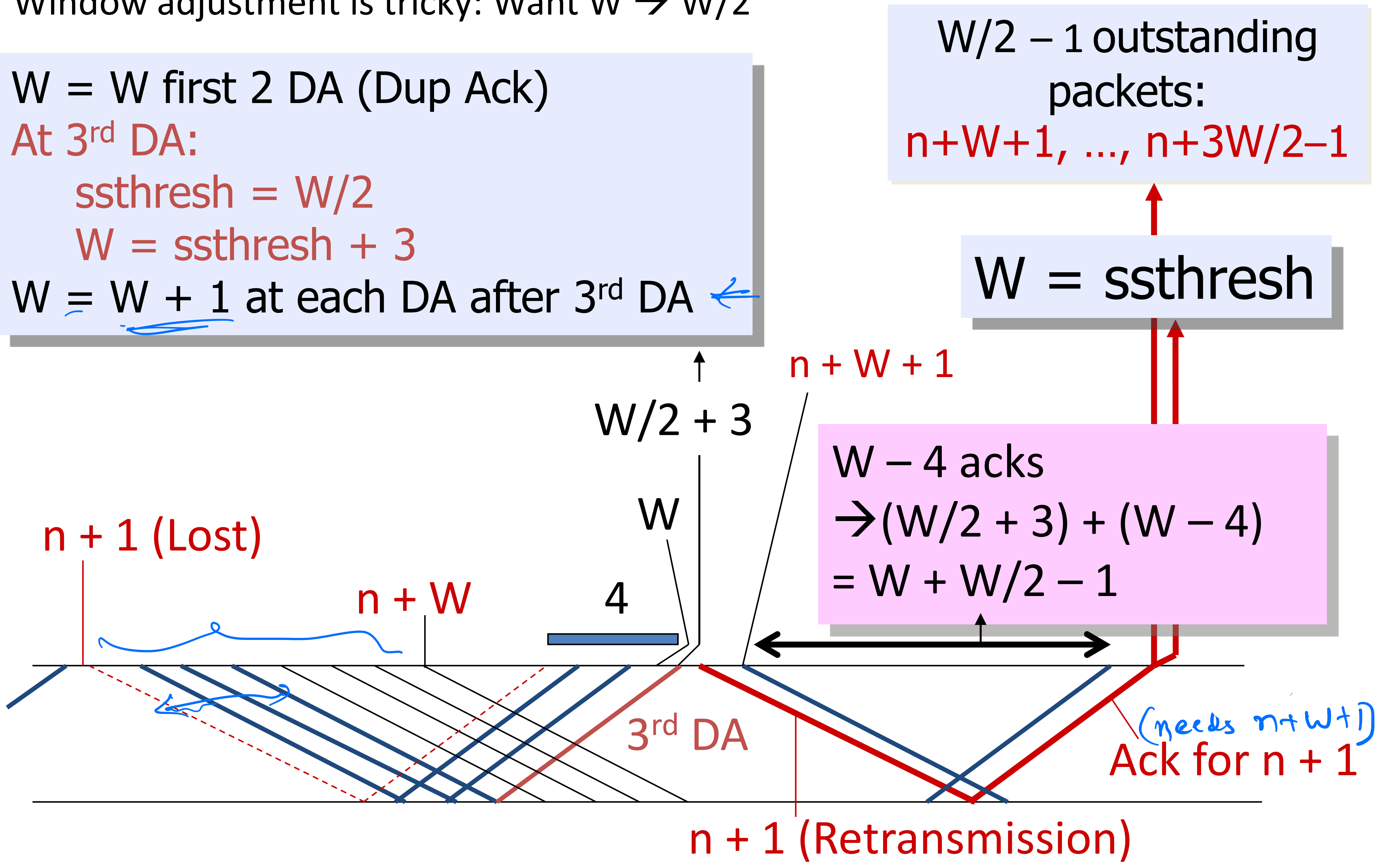
Refinements: Fast Recovery (1)

- Timeout → Reset Window = 1 unit (MSS)
- 3rd Dup ACK → Window/2



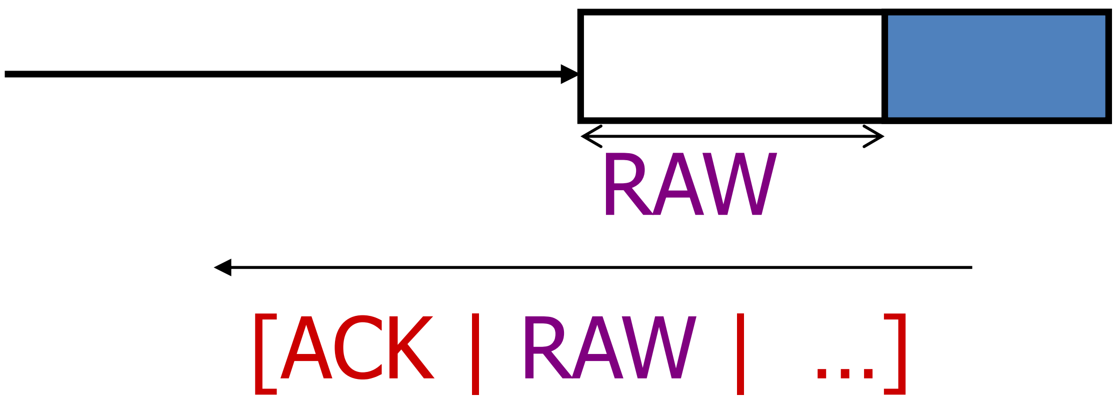
Refinements: Fast Recovery (2)

Window adjustment is tricky: Want W → W/2



Refinements: Flow Control

- Objective: Avoid saturating destination
- Algorithm: Receiver advertises window RAW



"Read" Buffer at Receiver
(Stores data to be acked)

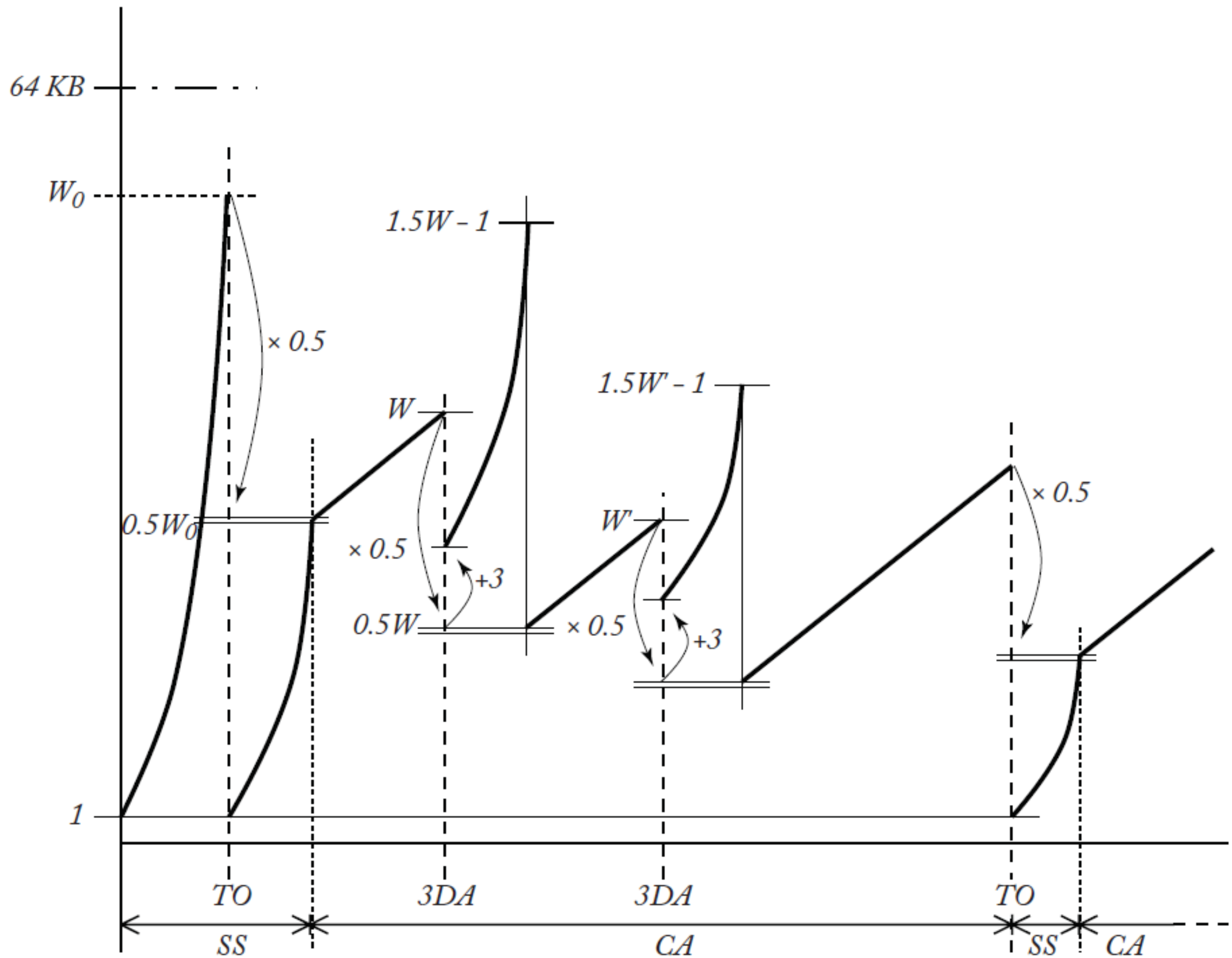
More precisely,
RAW = "Read" Buffer Size –
(Last Received – Last Acked)

CWND

Effective Transmit Window = min {RAW, W}
Effective Transmit Window Open =
Effective Transmit Window - OUT
where
OUT = Outstanding = Last Sent – Last Acked
W = Congestion Window from AIMD + Refinements

Refinements: Summary

- Effective Transmit Window= min {RAW, W}

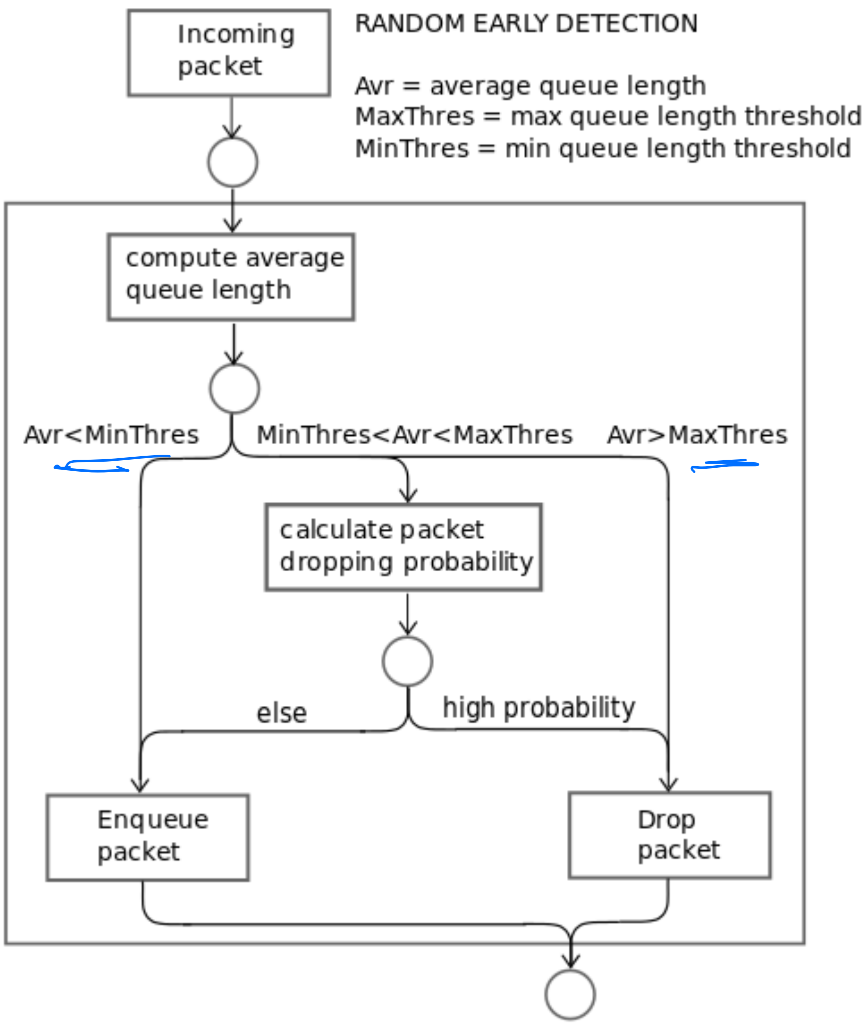


TCP Vegas

- Don't rely on packet losses to control windows.
- Idea: Maintain enough window to obtain maximal throughput.
- Vegas Congestion Avoidance Algorithm:
  - Compute Diff = ExpectedRate – ActualRate, where ExpectedRate = CongestionWindow/BaseRTT and BaseRTT = minimum of all measured RTTs.
  - Define thresholds α and β ($\alpha < \beta$).
  - Increase or reduce CongestionWindow linearly during the next RTT if Diff $< \alpha$ or if Diff $> \beta$, respectively, and leave CongestionWindow unchanged if $\alpha <$ Diff $< \beta$.
  - This is equivalent to attempt to control the backlog due to this connection within a range.
- Problem: Vegas users get clobbered in presence of Reno users.

Random Early Detection (RED)

or Discard

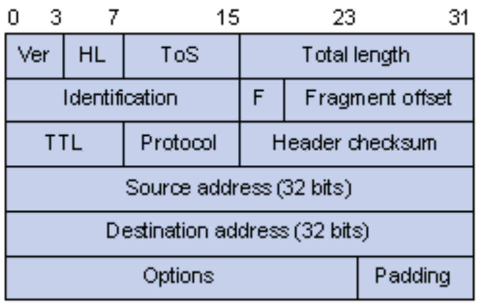


- Active Queue Management (AQM) suite includes RED.
- RED drops/marks packets proactively.
- Benefits of RED:
  - Works cooperatively with TCP to reduce network congestion proactively.
  - Helps avoid TCP global synchronization.
  - Helps mitigate unfairness against bursty traffic in the usual tail drop buffers.

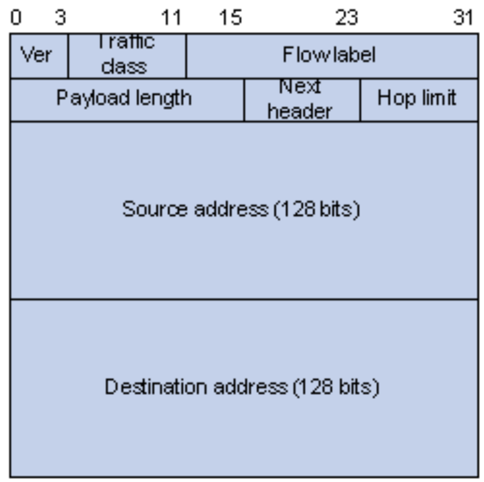
Explicit Congestion Notification (ECN)

- Purpose: Indicate network congestion to TCP sender without dropping a packet.
  - TCP sender can reduce congestion window proactively.
- RED mechanism can be used to mark a packet for indicating network congestion.
- Basic Idea:
  - Network elements indicate in IP header if they are ECN capable and if congestion is encountered.
  - Congestion indication is reflected back to the TCP sender using bits in the TCP header.

ECN Operation with TCP



IPv4 header



Basic IPv6 header

ToS/Traffic Class

7	6	5	4	3	2	1	0
DSCP						ECN	

- `00` – Non ECN-Capable Transport, Non-ECT
- `10` – ECN Capable Transport, ECT(0)
- `01` – ECN Capable Transport, ECT(1)
- `11` – Congestion Encountered, CE.

Differentiated Services Code Point

DSCP Value	Meaning	Drop Probability	Equivalent IP Precedence Value
101 110 (46)	High Priority Expedited Forwarding (EF)	N/A	101 – Critical
000 000 (0)	Best Effort	N/A	000 – Routine
001 010 (10)	AF11	Low	001 – Priority
001 100 (12)	AF12	Medium	001 – Priority
001 110 (14)	AF13	High	001 – Priority
010 010 (18)	AF21	Low	001 – Immediate
010 100 (20)	AF22	Medium	001 – Immediate
010 110 (22)	AF23	High	001 – Immediate

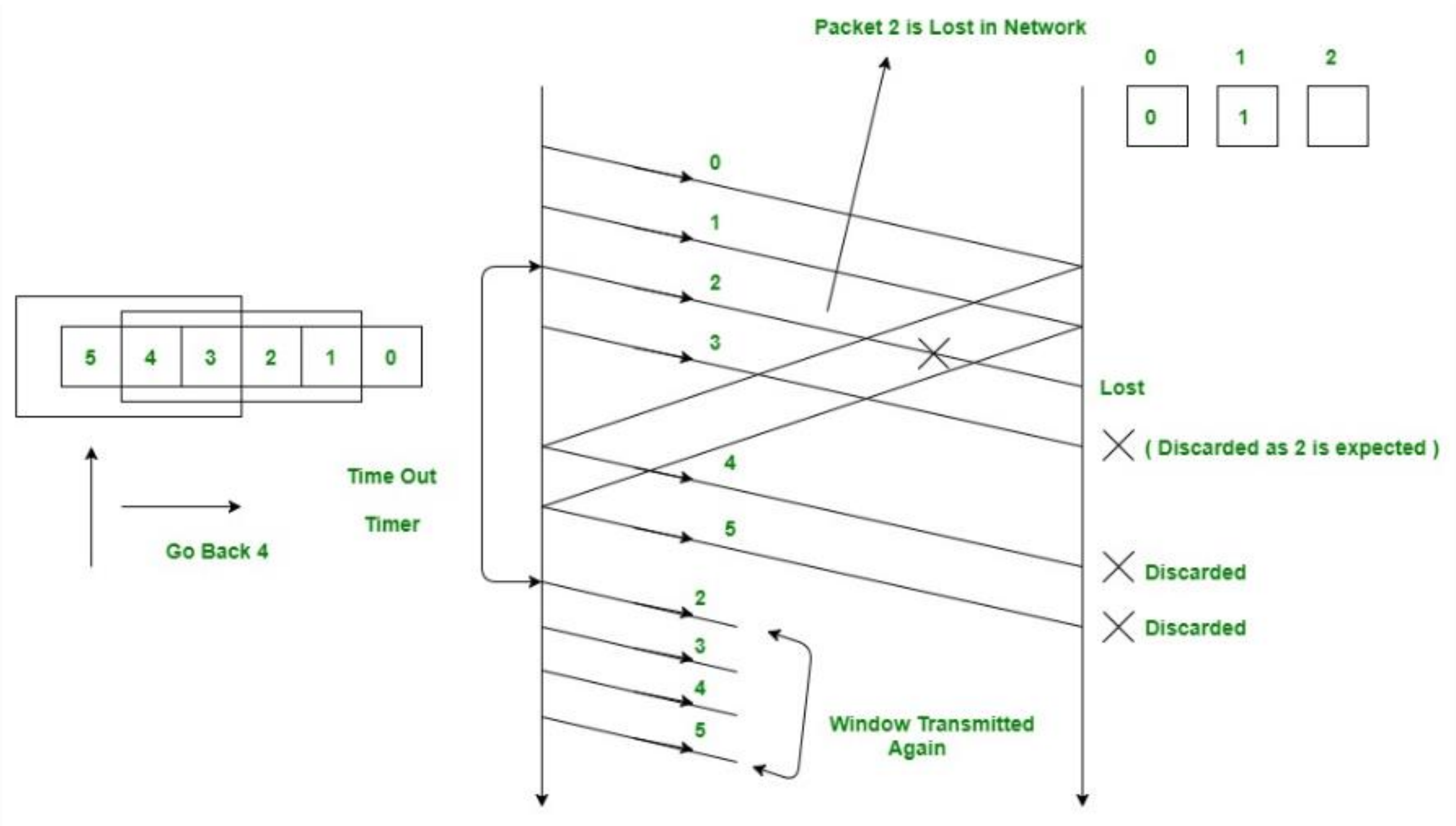
TCP segment header

<table>
<tr><th>Offsets</th><th>Octet</th><th colspan="8">0</th><th colspan="8">1</th><th colspan="8">2</th><th colspan="8">3</th></tr>
<tr><th>Octet</th><th>Bit</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><th>0</th><th>0</th><td colspan="16">Source port</td><td colspan="16">Destination port</td></tr>
<tr><th>4</th><th>32</th><td colspan="32">Sequence number</td></tr>
<tr><th>8</th><th>64</th><td colspan="32">Acknowledgment number (if ACK set)</td></tr>
<tr><th>12</th><th>96</th><td colspan="4">Data offset</td><td colspan="3">Reserved
0 0 0</td><td>N
S</td><td>C
W
R</td><td>E
C
E</td><td>U
R
G</td><td>A
C
K</td><td>P
S
H</td><td>R
S
T</td><td>S
Y
N</td><td>F
I
N</td><td colspan="16">Window Size</td></tr>
<tr><th>16</th><th>128</th><td colspan="16">Checksum</td><td colspan="16">Urgent pointer (if URG set)</td></tr>
<tr><th>20</th><th>160</th><td colspan="32" rowspan="3">Options (if data offset > 5. Padded at the end with "0" bits if necessary.)</td></tr>
<tr><th>⋮</th><th>⋮</th></tr>
<tr><th>60</th><th>480</th></tr>
</table>

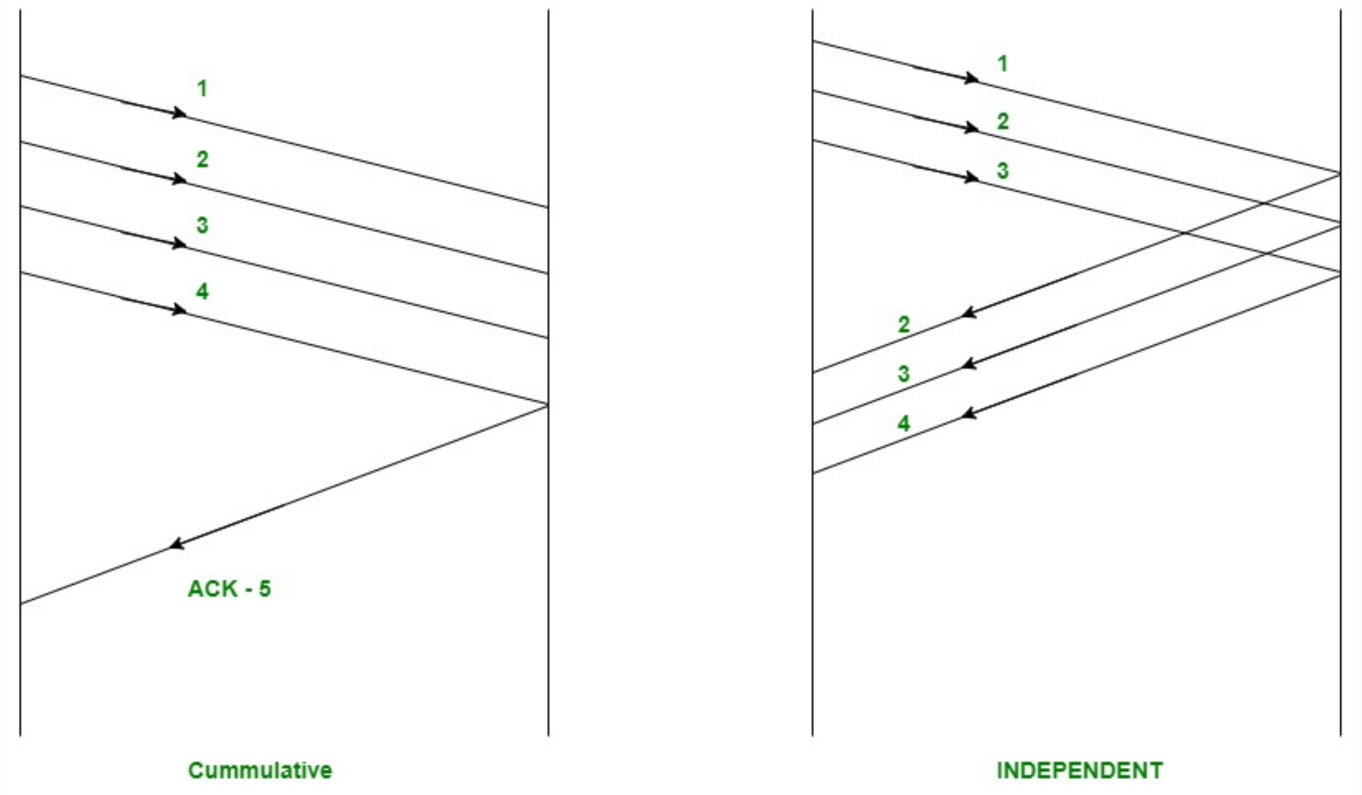
CWR = Congestion Window Reduced, ECE = ECN-Echo

Go Back N ARQ Example

Go Back N ARQ: W = N = 4, WR = 1



From https://www.geeksforgeeks.org/sliding-window-protocol-set-2-receiver-side/



Note: Go Back N ARQ uses cumulative acknowledgements.

From https://www.geeksforgeeks.org/sliding-window-protocol-set-2-receiver-side/