Transport

Review: slow start, additive inc

Say `MSS` = 1 KByte

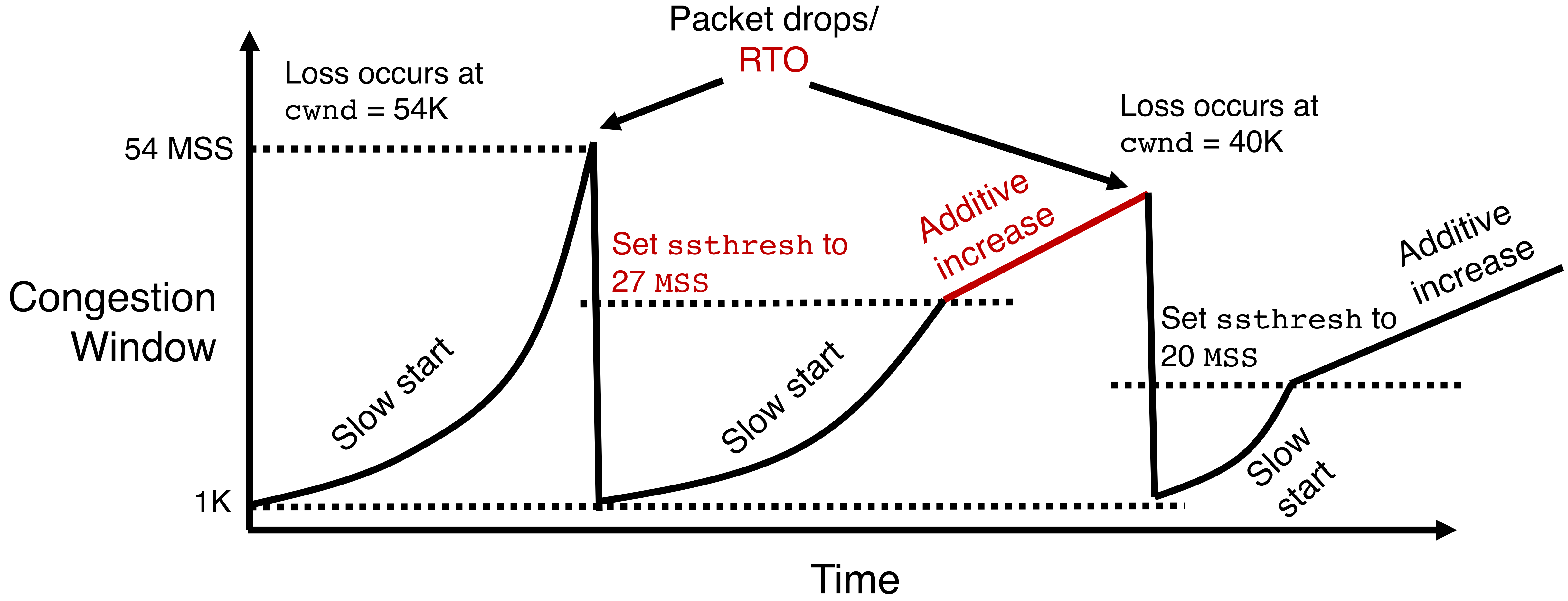
Default `ssthresh` = 64KB = 64 `MSS`

AI is slow.

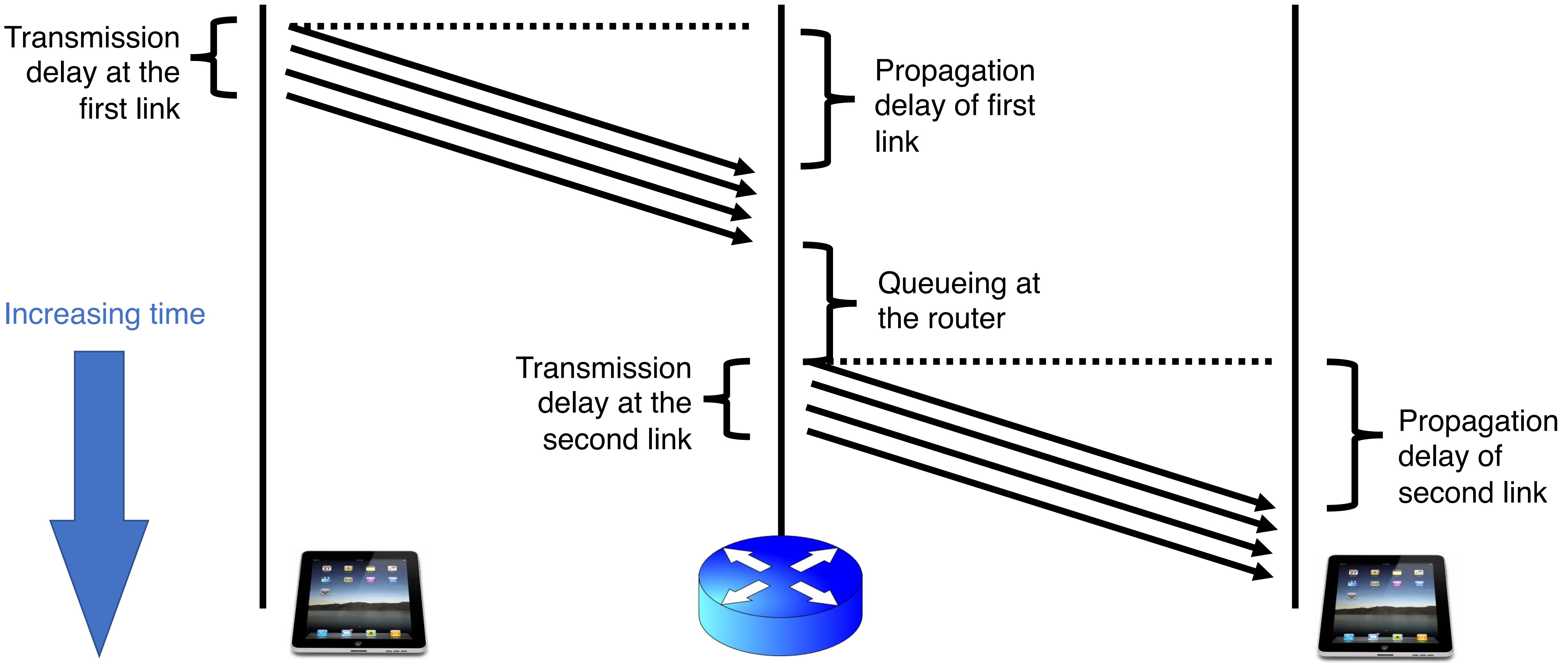
Persistent connections

Large window sizes

Different laws to evolve congestion window



The components of delay



Bandwidth-Delay Product

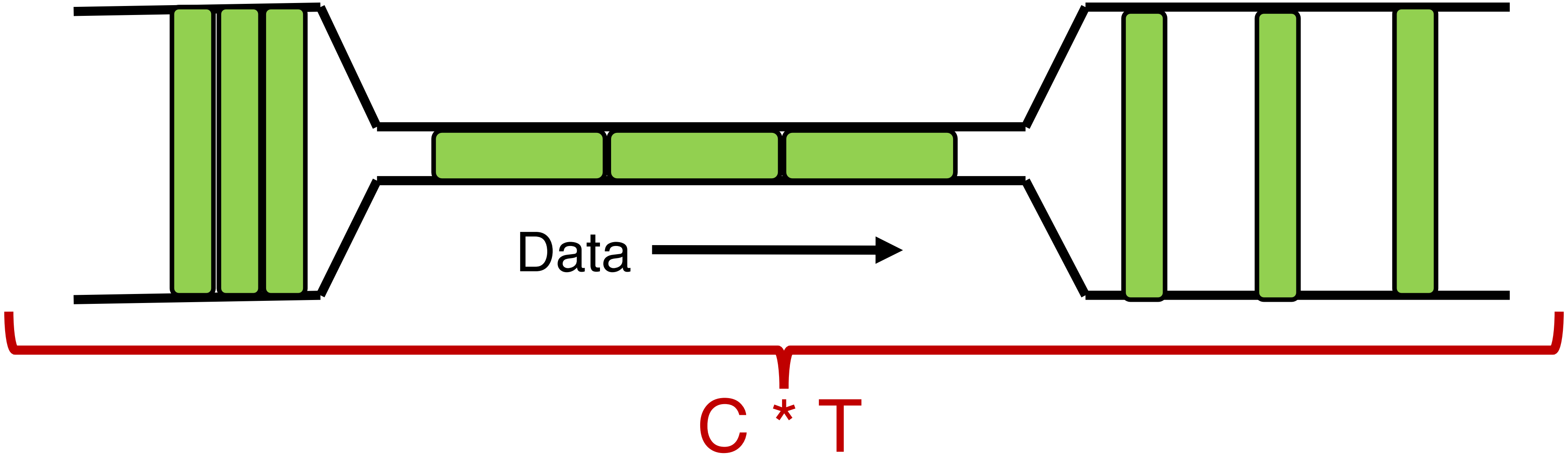
Steady state `cwnd` for a single flow

- Suppose the bottleneck link has rate C
- Suppose the propagation round-trip delay (propRTT) between sender and receiver is T
- Ignore transmission delays for this example;
- Assume steady state: highest sending rate with no bottleneck congestion
- Q: how much data is in flight over a single RTT?
- C * T data i.e., amount of data unACKed at any point in time
- ACKs take time T to arrive (without any queueing). In the meantime, sender is transmitting at rate C

The Bandwidth-Delay Product

- C * T = bandwidth-delay product:
  - The amount of data in flight for a sender transmitting at the ideal rate during the ideal round-trip delay of a packet

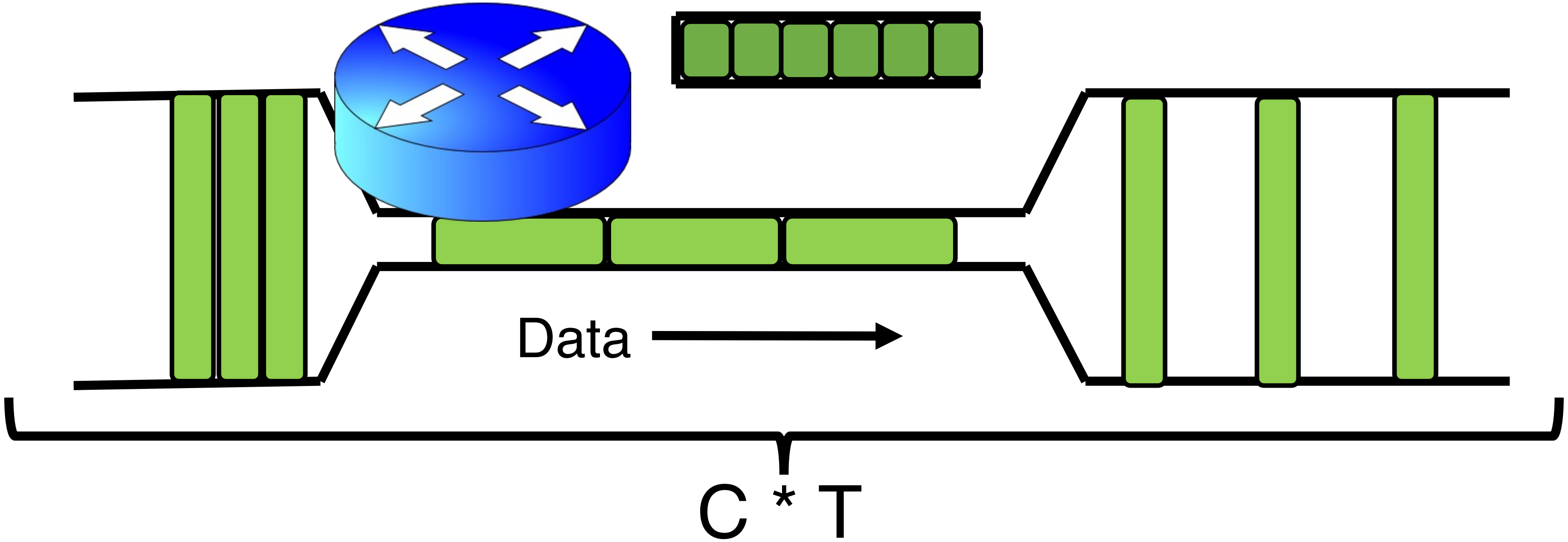
- Note: this is just the amount of data "on the pipe"



The Bandwidth-Delay Product

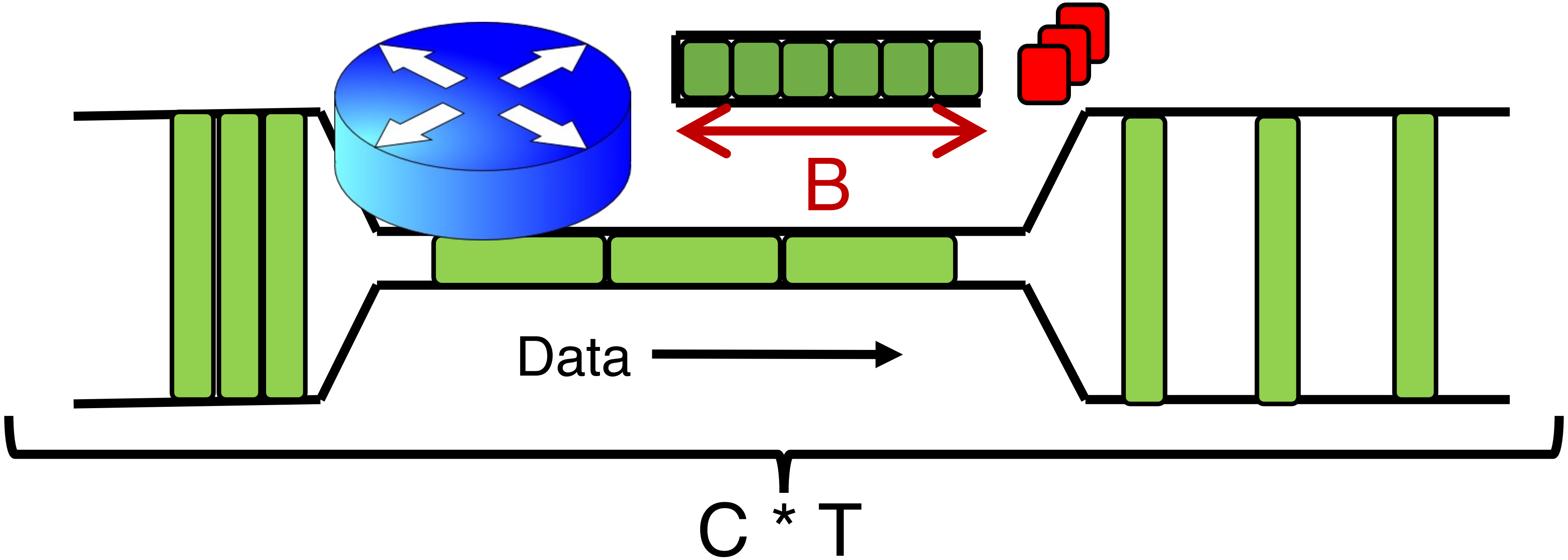
- Q: What happens if cwnd > C * T?
  - i.e., where are the rest of the in-flight packets?

- A: Waiting at the bottleneck router queues



Router buffers and the max `cwnd`

- Router buffer memory is finite: queues can only be so long
  - If the router buffer size is B, there is at most B data waiting in the queue
- If cwnd increases beyond C * T + B, data is dropped!



BDP is a crucial value for a flow

- Bandwidth-Delay Product (BDP) governs the window size of a single flow at steady state

- The bottleneck router buffer size governs how much the `cwnd` can exceed the BDP before packet drops occur

- BDP is the ideal desired window size to use the full bottleneck link, without any queueing.
  - Accommodating flow control, also the min socket buffer size to use the bottleneck link fully

Demo of the impact of BDP & B

- Utilization
- Congestion window

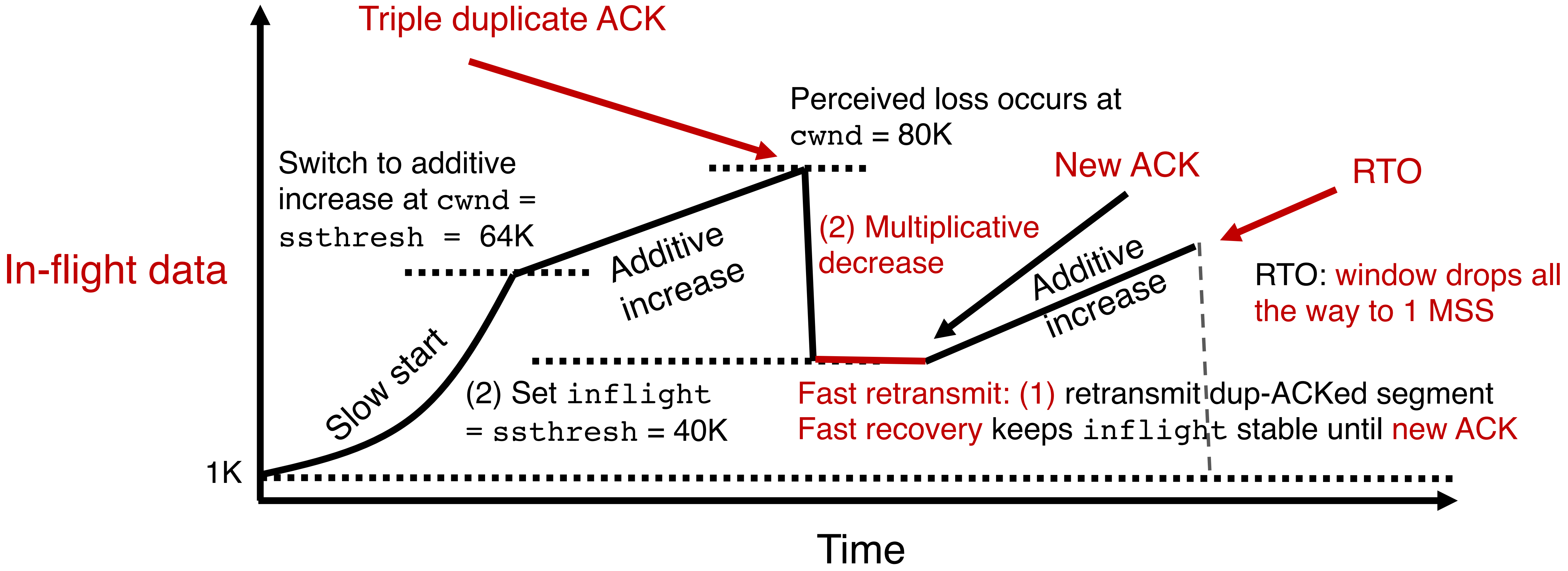
Detecting and Reacting Better to Packet Loss

Can we detect loss earlier than RTO?

- Key idea: use the information in the ACKs. How?
- Suppose successive (cumulative) ACKs contain the same ACK#
  - Also called duplicate ACKs
  - Occur when network is reordering packets, or one (but not most) packets in the window were lost
  - Fast retransmit: (1) Immediately retransmit packet
- Reduce `cwnd` when you see many duplicate ACKs
  - Consider many dup ACKs a strong indication that packet was lost
  - Default threshold: 3 dup ACKs, i.e., triple duplicate ACK
  - Make cwnd reduction gentler than setting cwnd = 1; recover faster
  - Fast retransmit: (2) reduce window to half of its current value

Additive Increase/Multiplicative Decrease

Say `MSS` = 1 KByte
Default `ssthresh` = 64KB = 64 `MSS`



TCP New Reno performs additive increase and multiplicative decrease of congestion window.

In short, we often refer to this as AIMD.

Multiplicative decrease is a part of all TCP algorithms. It is necessary for fairness across TCP flows.

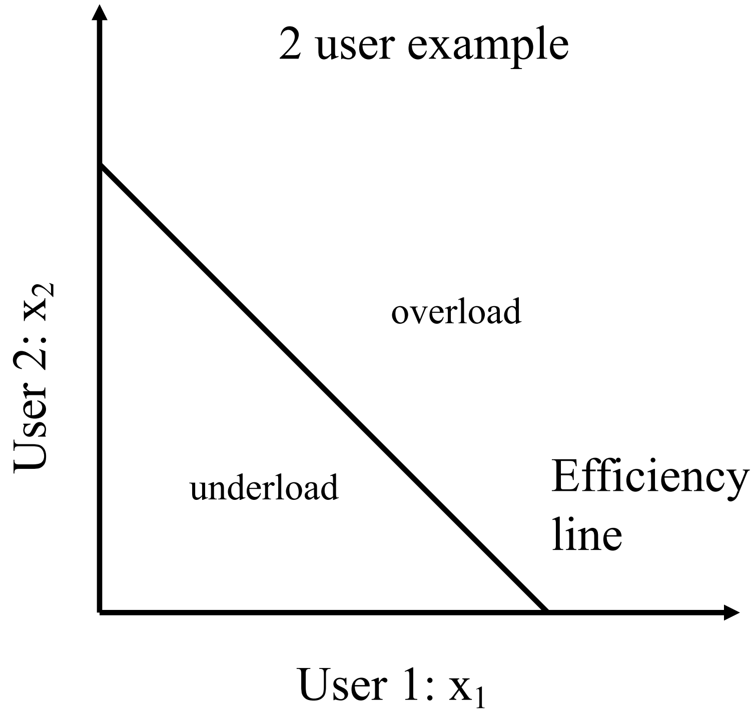
Why does multiplicative decrease help?

Efficiency and Fairness

Chiu and Jain, “Increase and decrease algorithms for congestion avoidance”

Efficient allocation

- Don't want sources to transmit either too slow or too fast
  - Slow: Underutilize the network
  - Fast: High delays, lose packets
- Every endpoint is reacting
  - May *all* under/overshoot
  - Large oscillations possible!
- Optimal efficiency:
  - $\Sigma x_i = X_{goal}$ e.g., link capacity
- Efficiency = 1 - distance from efficiency line

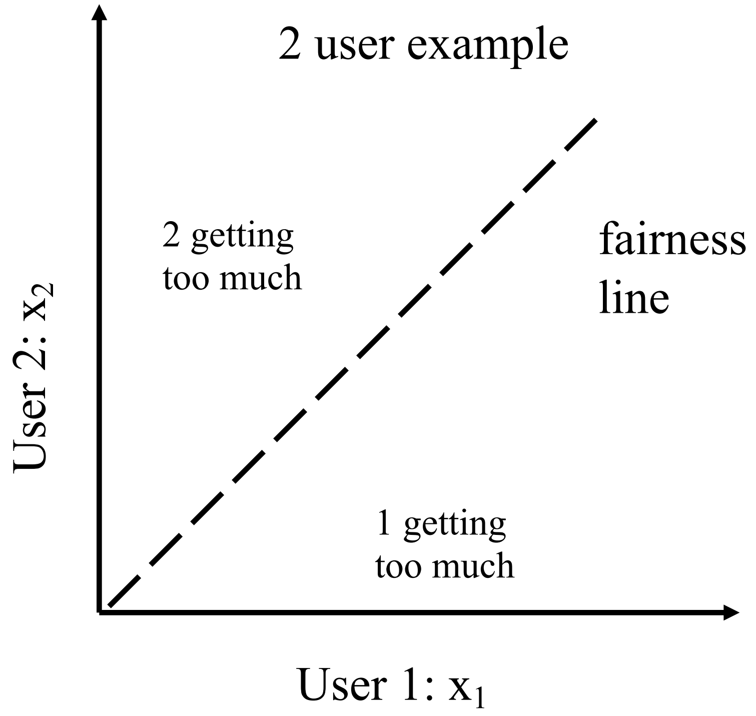


Fair allocation

- Max-min fairness
- Flows which share the same bottleneck get the same amount of bandwidth

$$F(x) = \frac{\left(\sum x_i\right)^2}{n\left(\sum x_i^2\right)}$$

- Fairness = 1 - distance from fairness line



How should transports react?

- Given efficiency and fairness goals above, how should transports behave?

- Consider $x(t)$, window or rate of a source, evolving over time t

- Assume discrete time steps.
- $x(t + 1) =$ function of $x(t)$, feedback from the network

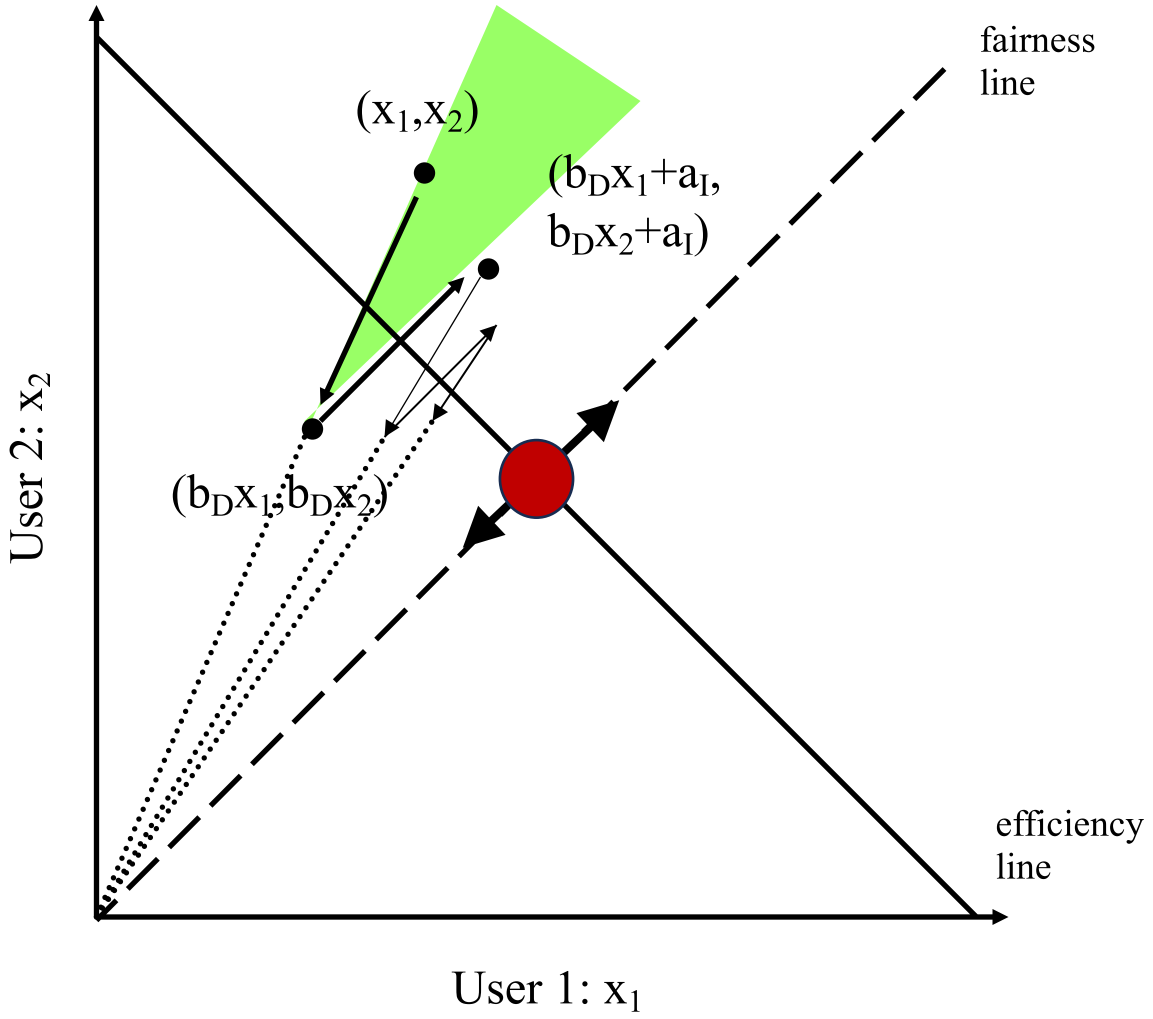
Linear control rules

$$x_i(t+1) = \begin{Bmatrix} a_I + b_I x_i(t) & increase \\ a_D + b_D x_i(t) & decrease \end{Bmatrix}$$

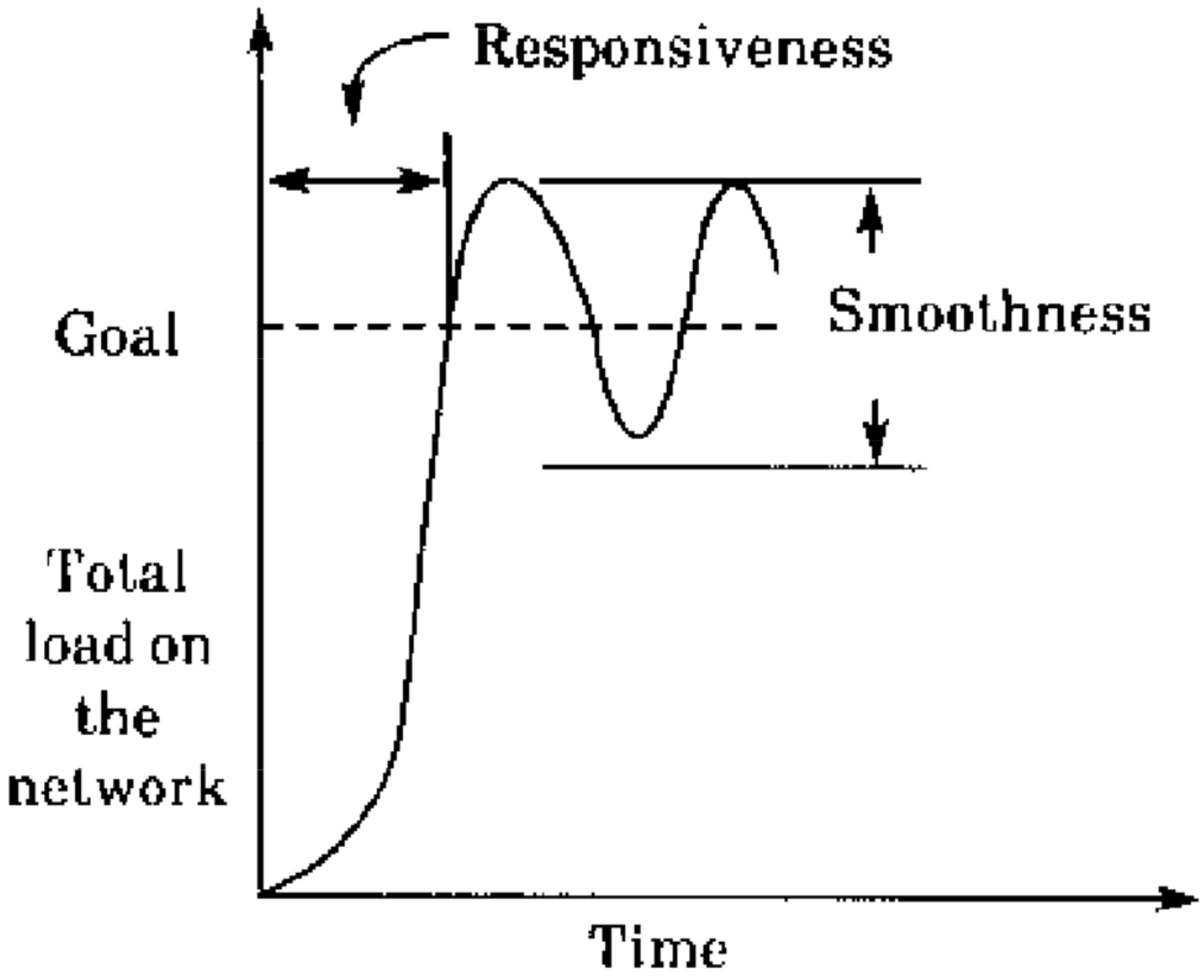
- $x_i(t)$: window or rate of the i^{th} user at time t
- a_I, a_D, b_I, b_D: constant increase/decrease coefficients
- Assumption: All users receive same network feedback
  - *Binary* feedback: sense congestion or available capacity
- Assumption: All users increase or decrease simultaneously

Additive increase, multiplicative decrease

- $b_I = 1$, $a_D = 0$
- Multiplicative decrease enables converging to fairness
- Oscillates around the most efficient point



Convergent doesn't mean static



Simple models are useful

- Chiu and Jain’s model isn’t indicative of all TCP/AIMD behavior
  - But it’s “realistic” enough
  - Stands the test of time: many more sources, much higher bandwidth, …
- Models should be simple
  - For us to work with
  - For others to understand
  - But they don’t have to mimic the “real” thing in every way
- A real, complex model is likely useless, but a realistic, simple model might teach us something

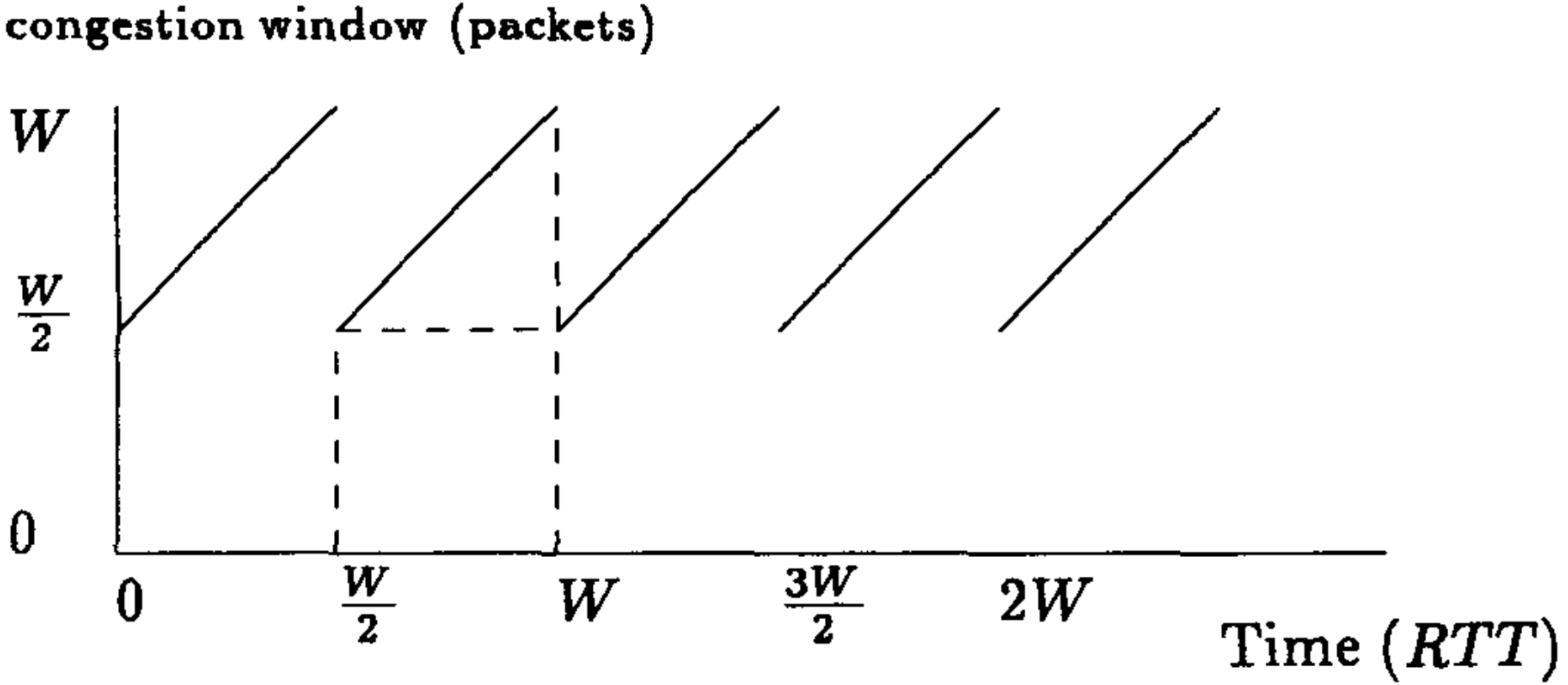
Modeling TCP throughput

Given network characteristics, how quickly can TCP (New Reno) send data?

Mathis et al., “Macroscopic behavior of TCP congestion avoidance”

Steady AIMD

$$BW < \left(\frac{MSS}{RTT}\right)\frac{1}{\sqrt{p}}$$



Implications

- Throughput has a `1/sqrt(p)` dependence on packet loss rate

- RTT unfairness
  - Flows with a smaller RTT get better throughput (ramp up faster)

- Engineering implications:
  - Split TCP (CDNs, data center frontends, …)
  - Special considerations for long-distance connections

Widely Deployed TCPs

Data Center TCP

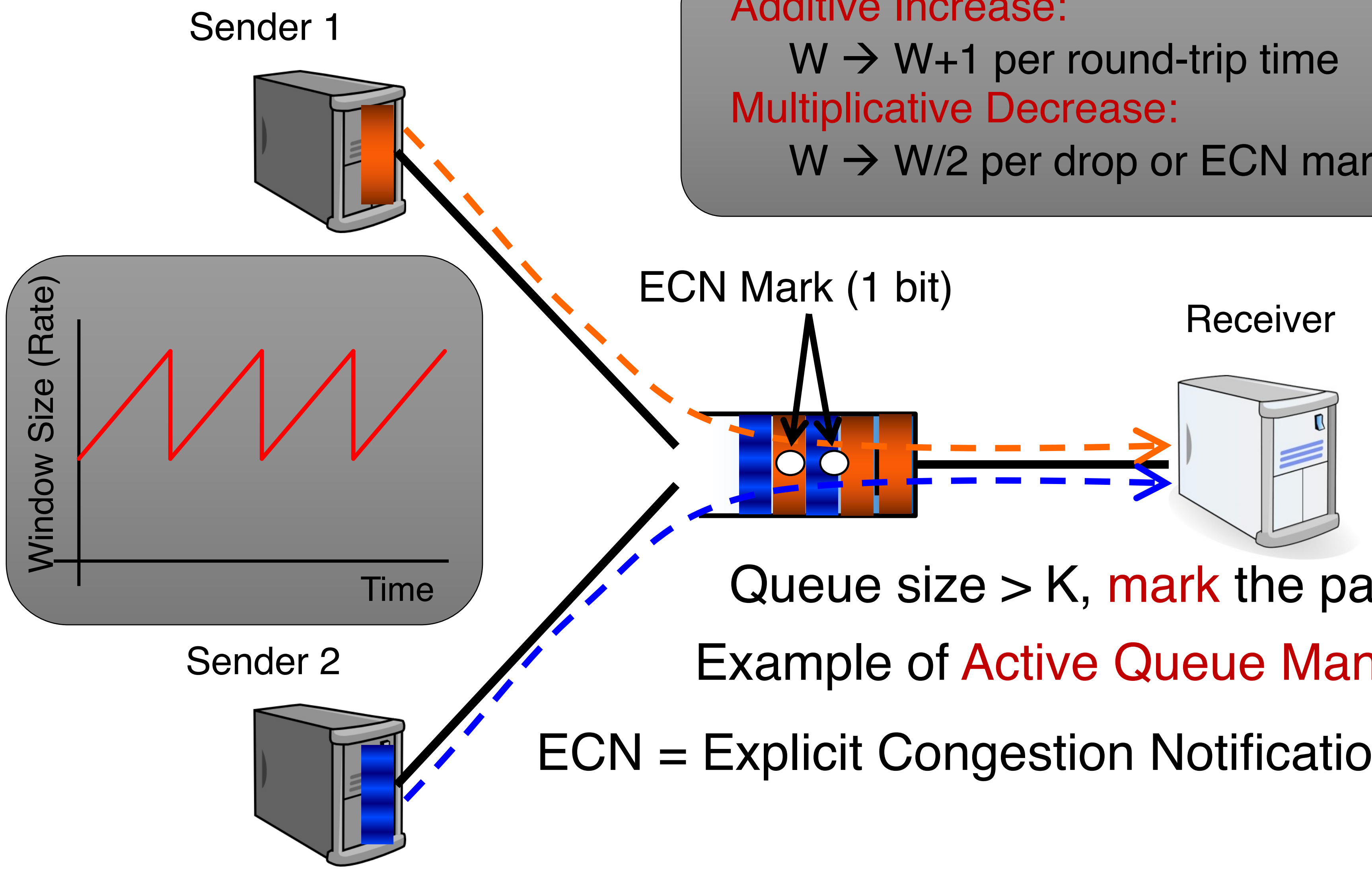
Alizadeh et al.

Context

- Regular TCP: window evolution with all signals measured end to end
- Data centers: hardware under single administrative control
  - Could network switches do better?
- What if switches provided better feedback than loss?

Explicit Congestion Notification

Additive Increase:
W → W+1 per round-trip time
Multiplicative Decrease:
W → W/2 per drop or ECN mark



Queue size > K, mark the packet.

Example of Active Queue Management

(RED, PI, etc.)

ECN = Explicit Congestion Notification

ECN set on the IP header by routers

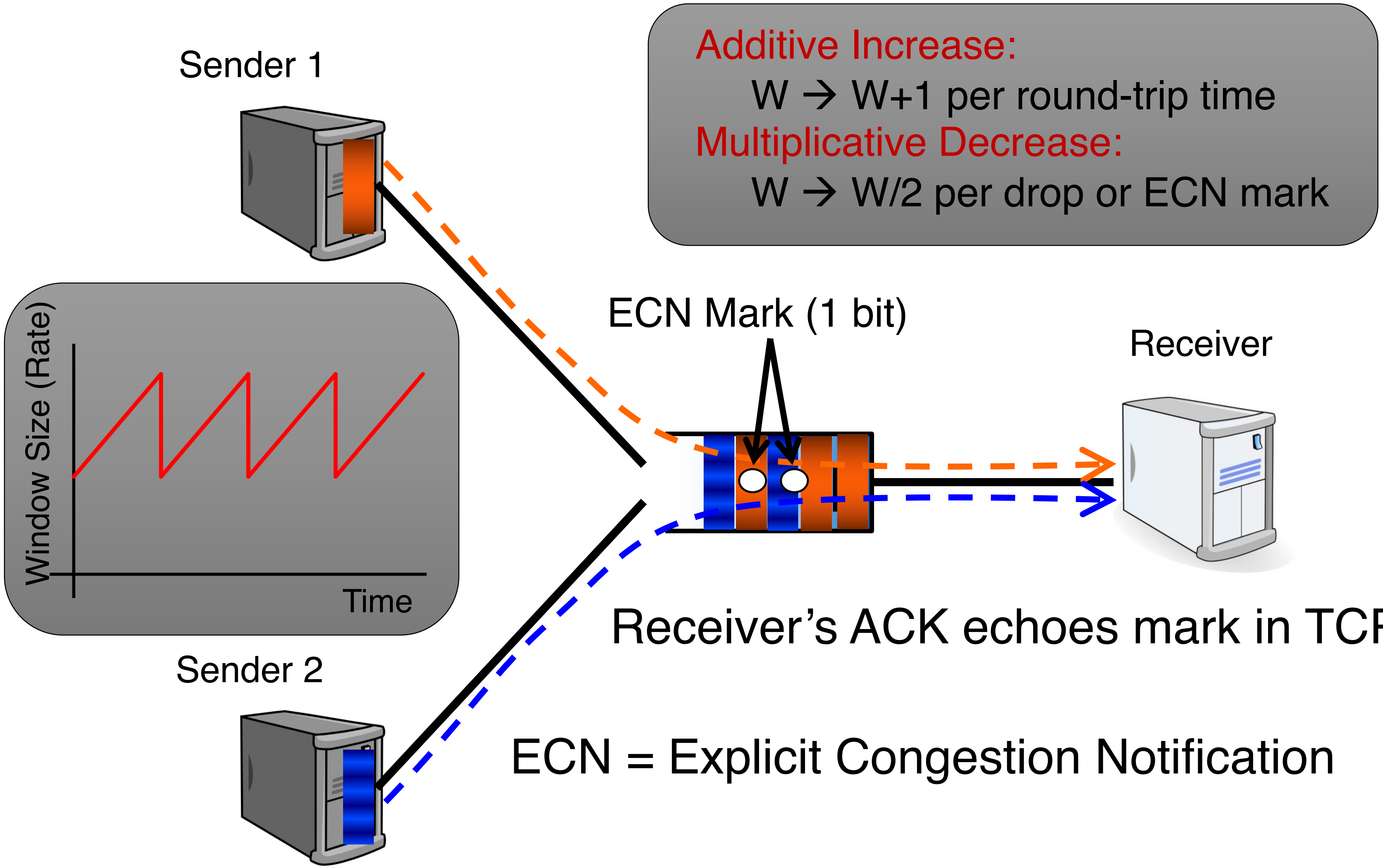
- `00` – Not ECN-Capable Transport, Not-ECT
- `01` – ECN Capable Transport(1), ECT(1)
- `10` – ECN Capable Transport(0), ECT(0)
- `11` – Congestion Experienced, CE.

Dropped if TCP sender is not ECN enabled

IPv4 header format

<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>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>11</th><th>12</th><th>13</th><th>14</th><th>15</th><th>16</th><th>17</th><th>18</th><th>19</th><th>20</th><th>21</th><th>22</th><th>23</th><th>24</th><th>25</th><th>26</th><th>27</th><th>28</th><th>29</th><th>30</th><th>31</th></tr>
<tr><td>0</td><td>0</td><td colspan="4">Version</td><td colspan="4">IHL</td><td colspan="6">DSCP</td><td colspan="2">ECN</td><td colspan="16">Total Length</td></tr>
<tr><td>4</td><td>32</td><td colspan="16">Identification</td><td colspan="3">Flags</td><td colspan="13">Fragment Offset</td></tr>
<tr><td>8</td><td>64</td><td colspan="8">Time To Live</td><td colspan="8">Protocol</td><td colspan="16">Header Checksum</td></tr>
<tr><td>12</td><td>96</td><td colspan="32">Source IP Address</td></tr>
<tr><td>16</td><td>128</td><td colspan="32">Destination IP Address</td></tr>
<tr><td>20</td><td>160</td><td colspan="32" rowspan="3">Options (if IHL > 5)</td></tr>
<tr><td>⋮</td><td>⋮</td></tr>
<tr><td>56</td><td>448</td></tr>
</table>

Explicit Congestion Notification



Receiver's ACK echoes mark in TCP header

ECN = Explicit Congestion Notification

ECN on the TCP header

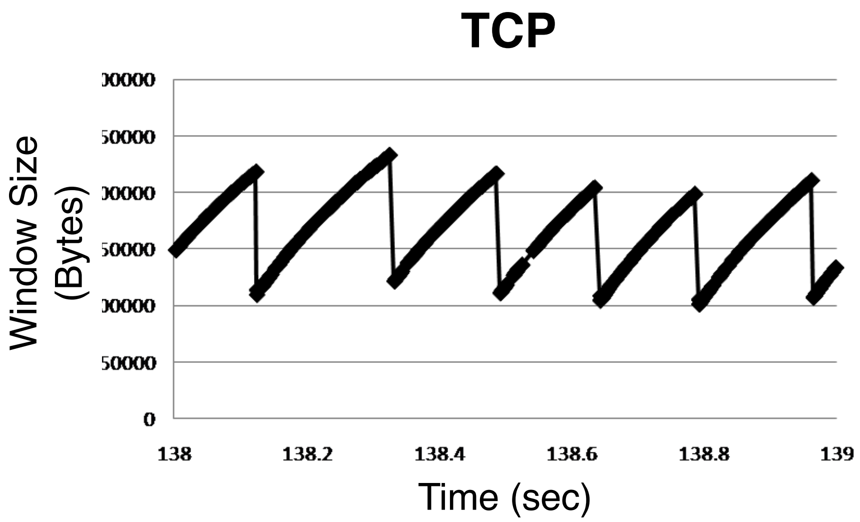
<table>
<caption>TCP segment header</caption>
<tr><th>Offsets</th><th>Octet</th><th colspan="8">0</th><th colspan="8">1</th><th colspan="8">2</th><th colspan="8">3</th></tr>
<tr><th>Octet</th><th>Bit</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th><th>7</th><th>6</th><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="4">Reserved
0 0 0 0</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>56</th><th>448</th></tr>
</table>

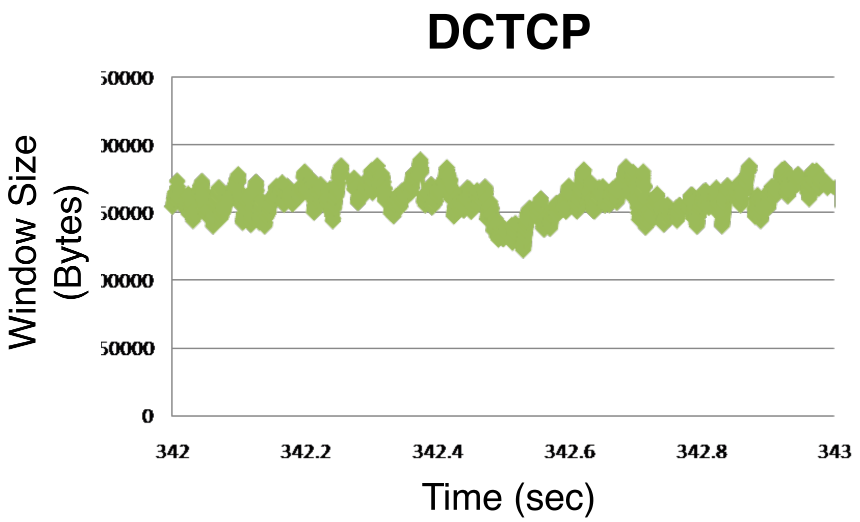
DCTCP: Main idea

- Extract multi-bit feedback from single-bit stream of ECN marks
  - Reduce window size based on **fraction** of marked packets

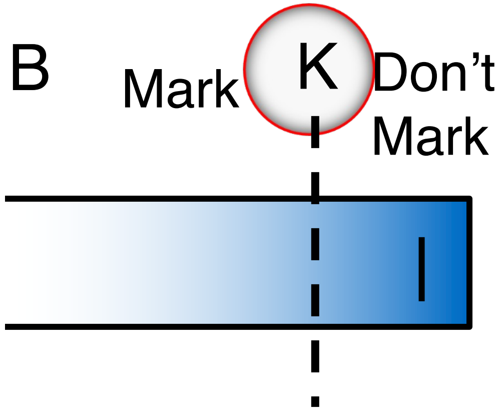
DCTCP: Main idea

ECN Marks	TCP	DCTCP
1 0 1 1 1 1 0 1 1 1	Cut window by 50%	Cut window by 40%
0 0 0 0 0 0 0 0 0 1	Cut window by 50%	Cut window by 5%





DCTCP algorithm



Switch side:

- Mark packets when Queue Length > K.

Sender side:

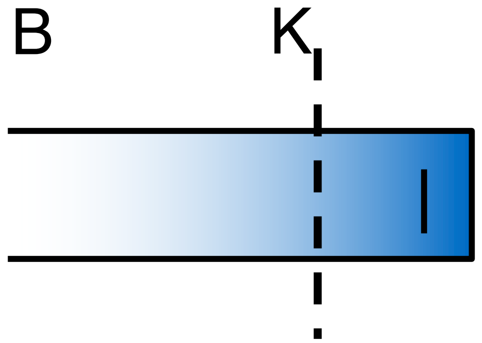
- Maintain running average of *fraction* of packets marked (α).

$$\text{each RTT}: \ F = \frac{\#\ \text{of marked ACKs}}{\text{Total}\ \#\ \text{of ACKs}} \ \Rightarrow \ \alpha \leftarrow (1-g)\alpha + gF$$

- Adaptive window decreases: $W \leftarrow (1 - \frac{\alpha}{2})W$
  - Note: decrease factor between 1 and 2.

Reacting to and controlling queue size distribution, specifically, the region above K.

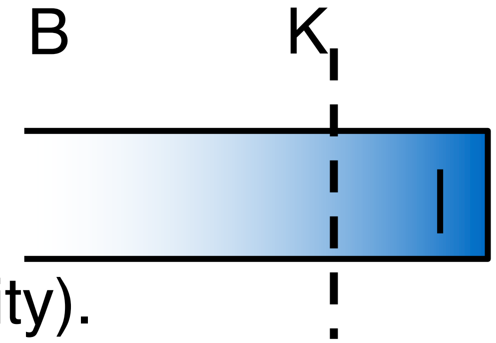
Setting parameters: Marking threshold K



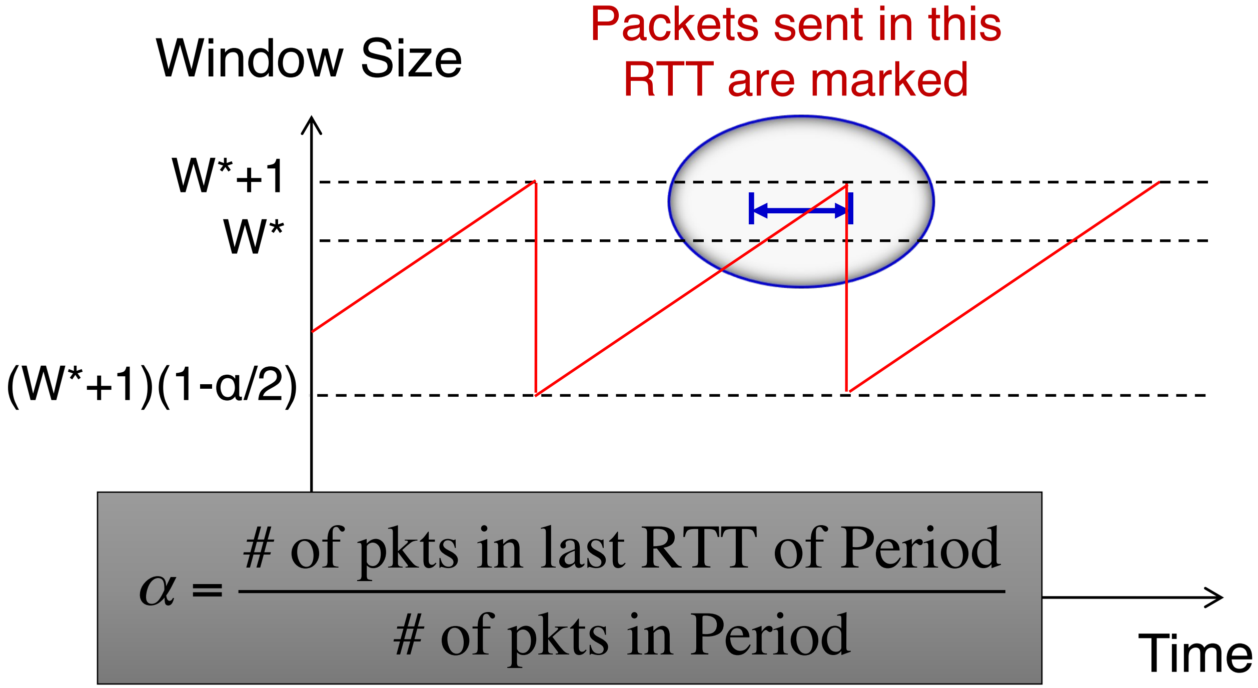
- Tradeoff!
- Mark too late: too much queueing & pkt loss
- Mark too early: queues too small, lose throughput
- How small can queues be without loss of throughput?

K > (1/7) C x RTT ⟷ for TCP: K > C x RTT

Buffering requirements



- How much buffering does DCTCP need for 100% throughput?
  - Need to quantify queue size oscillations (stability).



$$\alpha = \frac{\text{\# of pkts in last RTT of Period}}{\text{\# of pkts in Period}}$$