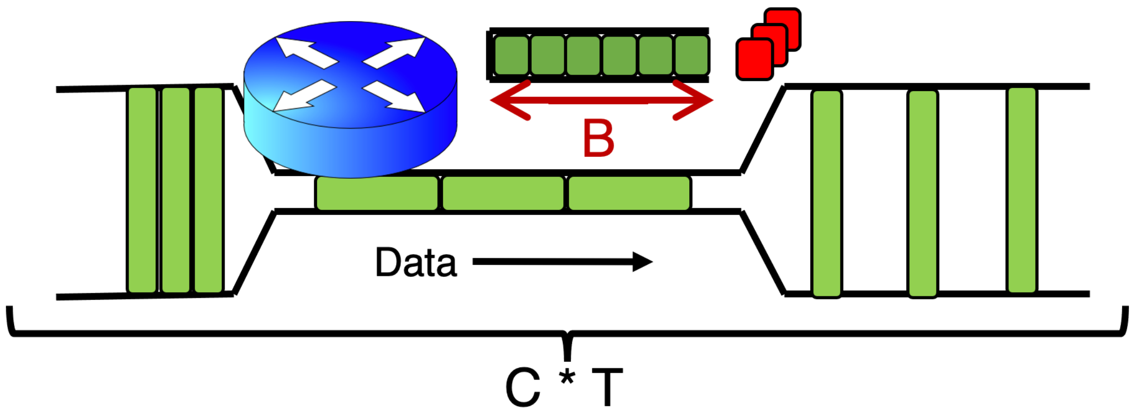
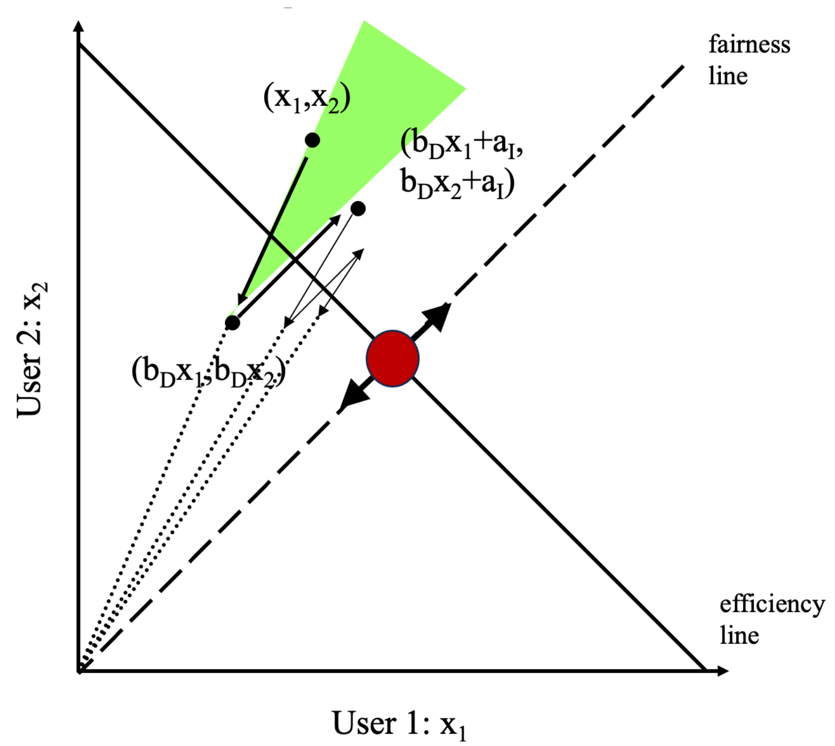
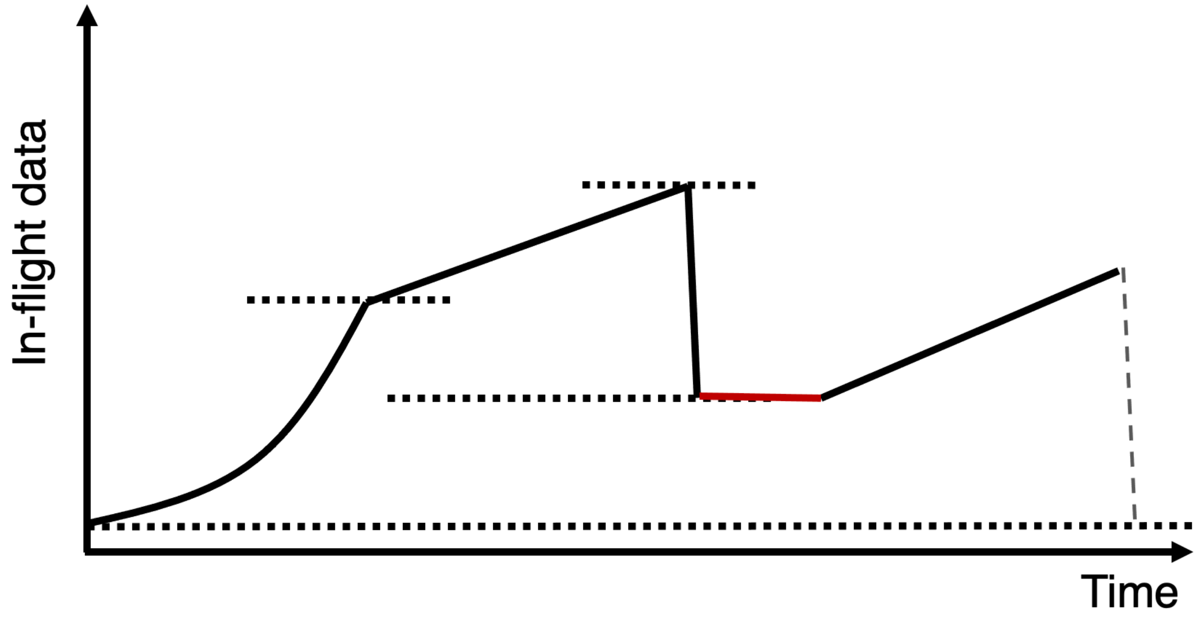
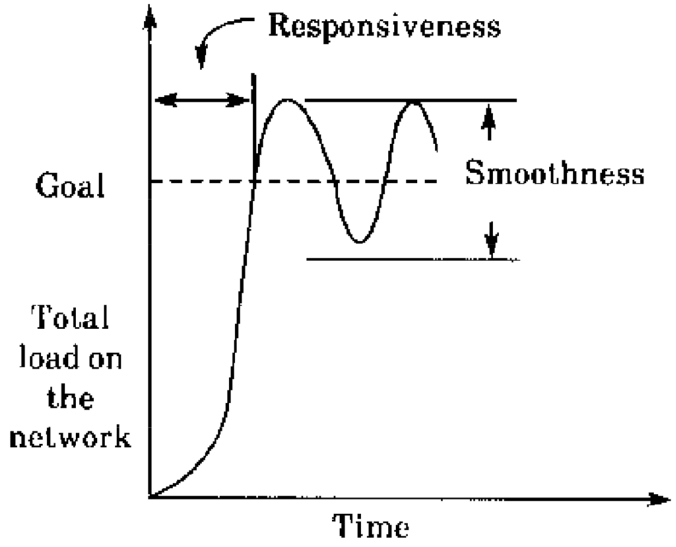
Transport

RUTGERS
UNIVERSITY | NEW BRUNSWICK


B
Data
C * T


fairness line
(x_1,x_2)
$(b_Dx_1+a_I, b_Dx_2+a_I)$
(b_Dx_1,b_Dx_2)
User 2: x_2
efficiency line
User 1: x_1


In-flight data
Time


Responsiveness
Goal
Smoothness
Total load on the network
Time

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

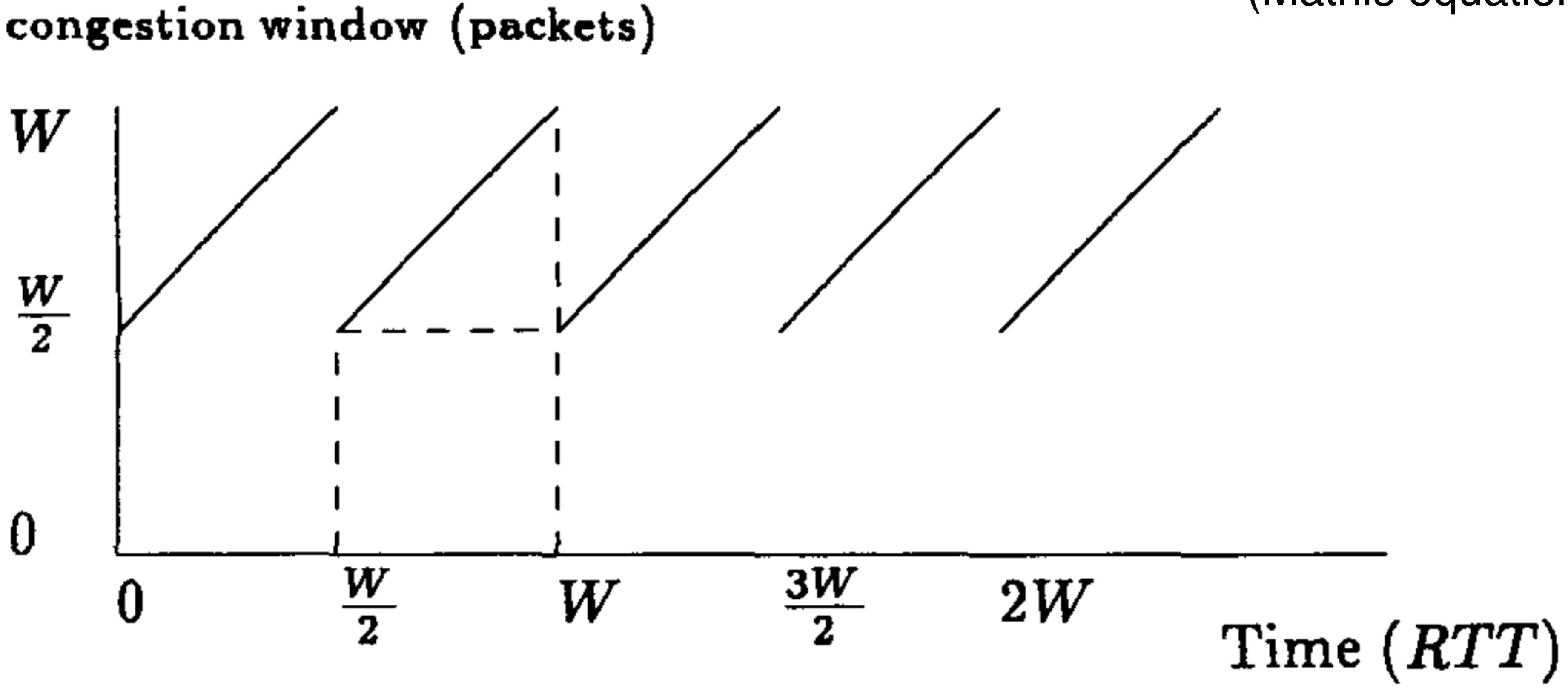
Given network characteristics, how quickly can TCP (New Reno) send data?

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

Steady AIMD

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

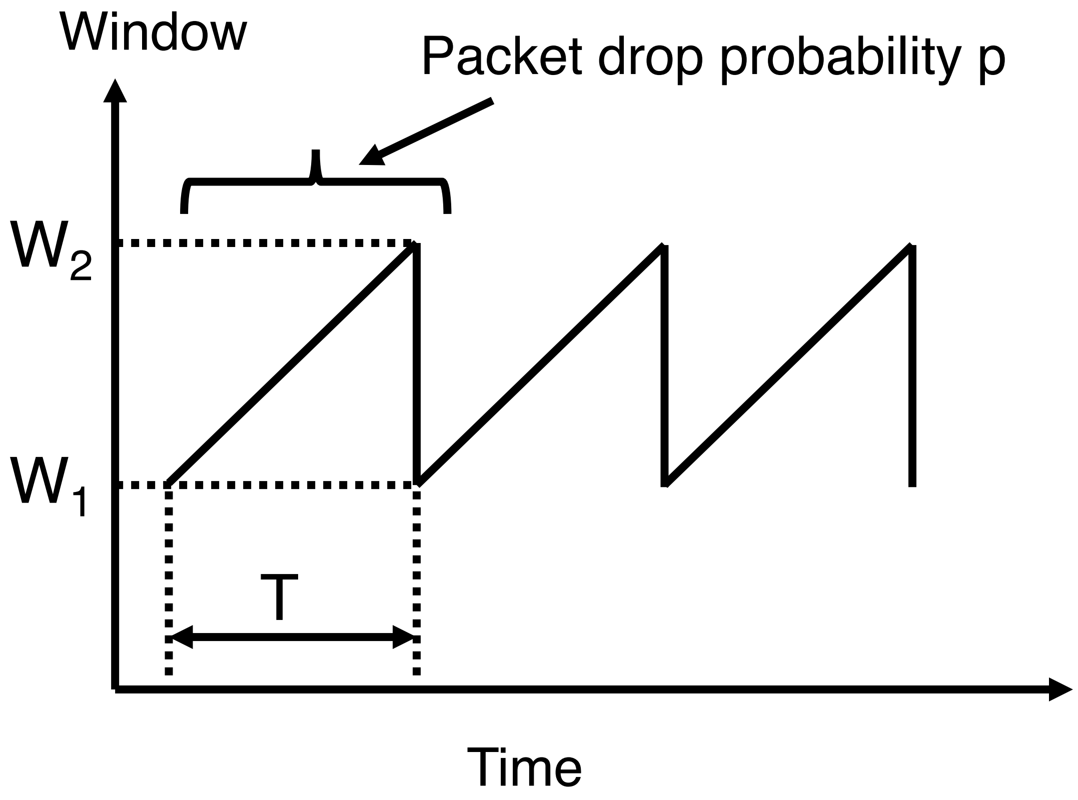
(Mathis equation)



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

Estimating TCP AIMD throughput

- Assumptions
- Single flow, repeating AIMD
- Loss occurs exactly in the last RTT before window reduction
  - Exactly one packet lost
- Assume RTT constant (ignore queueing delay change)
- Relationship between W_1, W_2?
- How many pkts sent over T?
- Relationship p and # pkts?



Implications

- Throughput has a `1/sqrt(p)` dependence on packet loss rate
- Getting full bottleneck throughput requires loss rate $1/(BDP)^2$

- 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

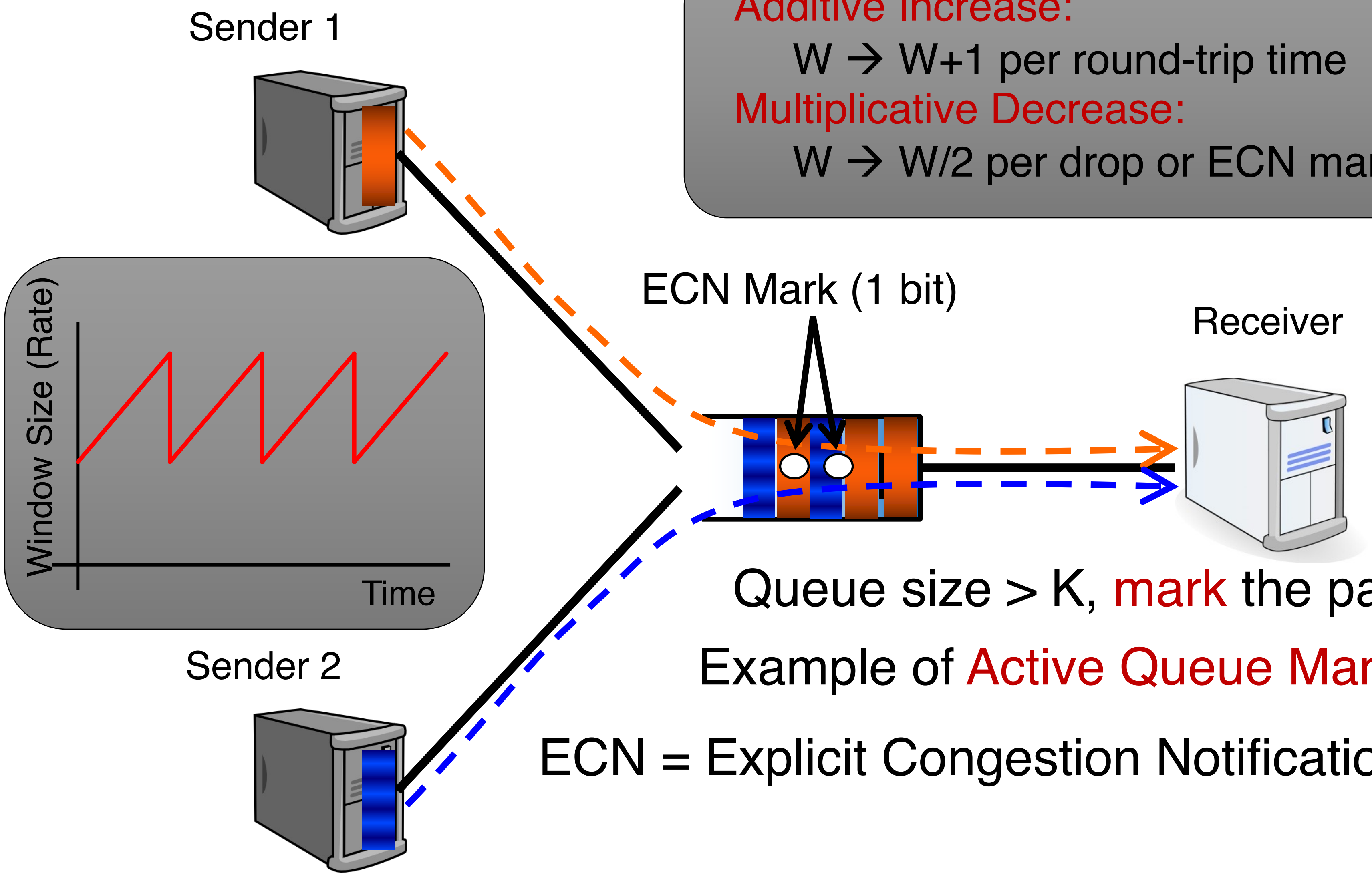
- Regular TCP: window evolution with all signals measured end to end

- 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 by router 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>

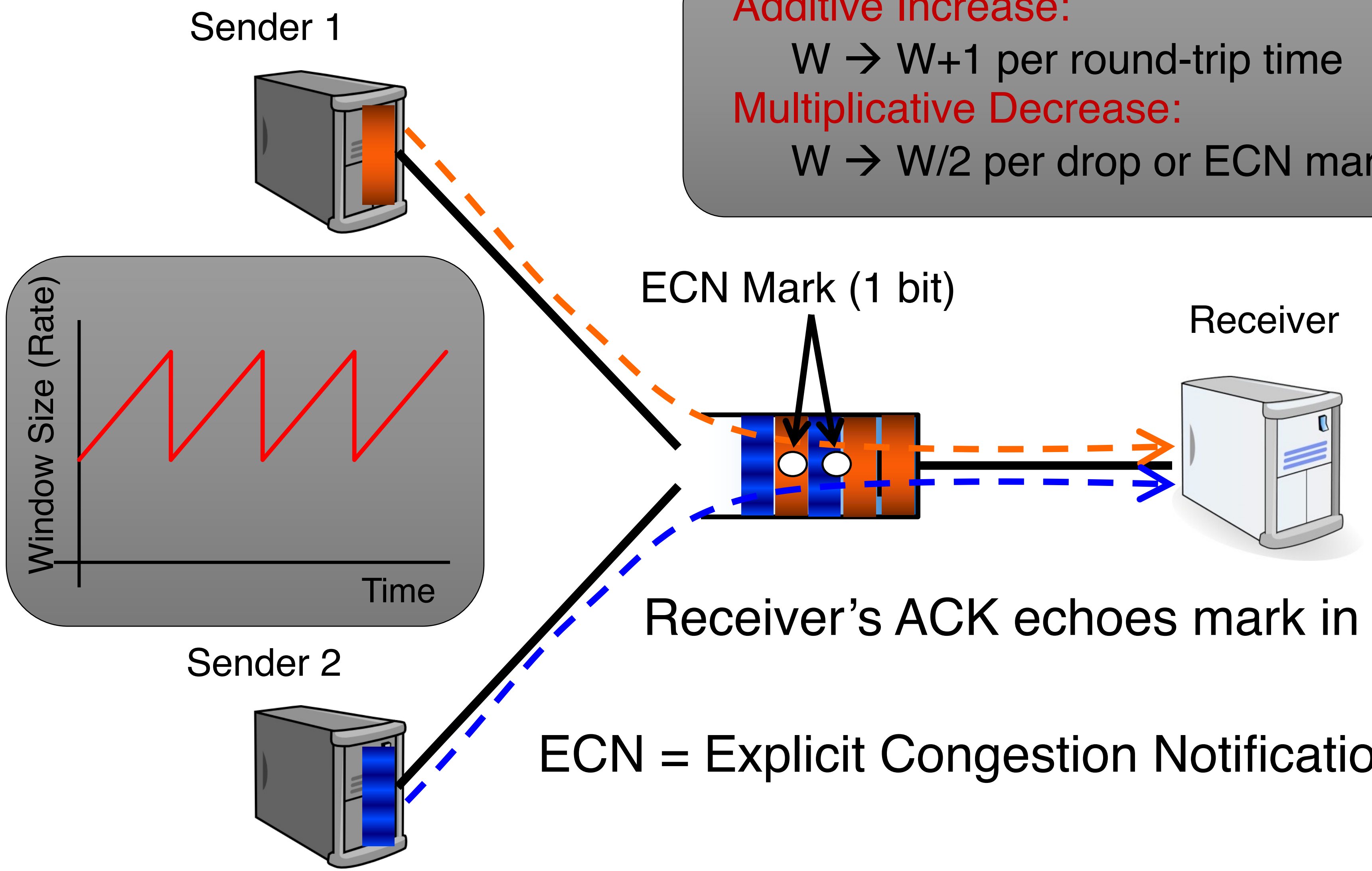
Explicit Congestion Notification

Additive Increase:

W → W+1 per round-trip time

Multiplicative Decrease:

W → W/2 per drop or ECN mark



Receiver's ACK echoes mark in TCP header

ECN = Explicit Congestion Notification

ECN on the TCP header

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="4">Reserved
0 0 0 0</td><td>CWR</td><td>ECE</td><td>URG</td><td>ACK</td><td>PSH</td><td>RST</td><td>SYN</td><td>FIN</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>

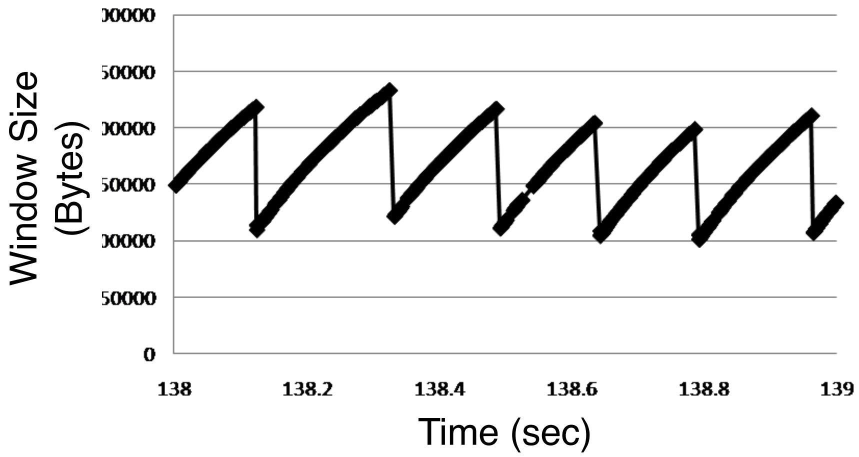
DCTCP: Main idea

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

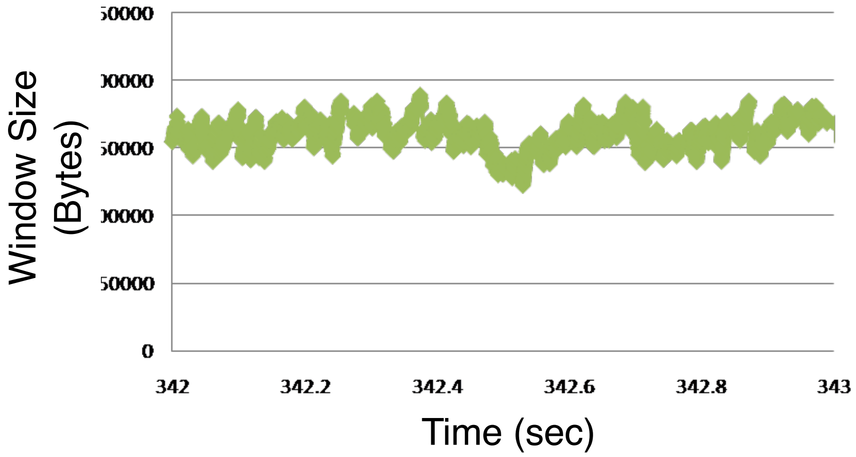
DCTCP: Main idea

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%

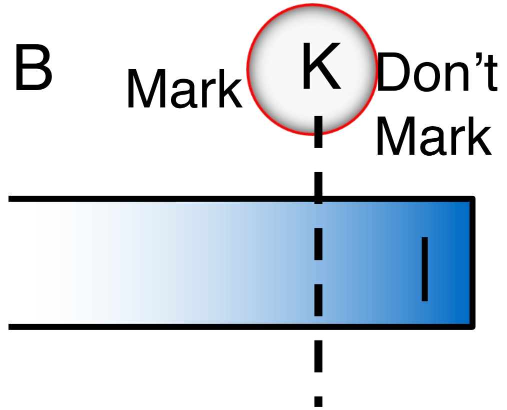
TCP



DCTCP



DCTCP algorithm



Router side:

- Mark packets when Queue Length > K.

Sender side:

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

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

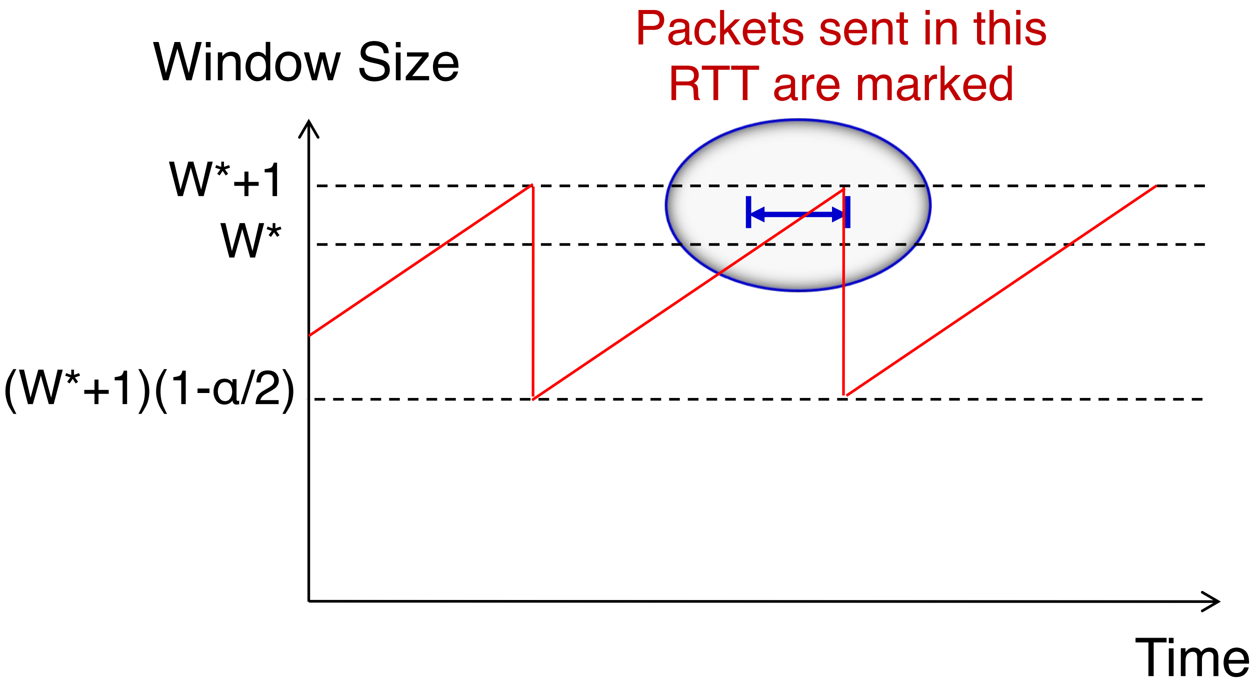
- Adaptive window decreases: $W \leftarrow (1-\frac{\alpha}{2})W$
  - Note: decrease factor between 1 and 2.

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

Setting protocol parameters

- When should router start marking? K:
  - Mark too late: higher queueing delay (and maybe loss)
  - Mark too early: queues too small, lose throughput
  - Want min queue size > 0 even when TCP windows drop
- What is the ideal buffer size for DCTCP?
  - Regular TCP: Bandwidth-delay product
  - Want buffer > max queue size

Use model to set parameters



$$W^* = (C \times RTT + K)/N$$

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

$$\alpha \approx \sqrt{2/W^*}$$

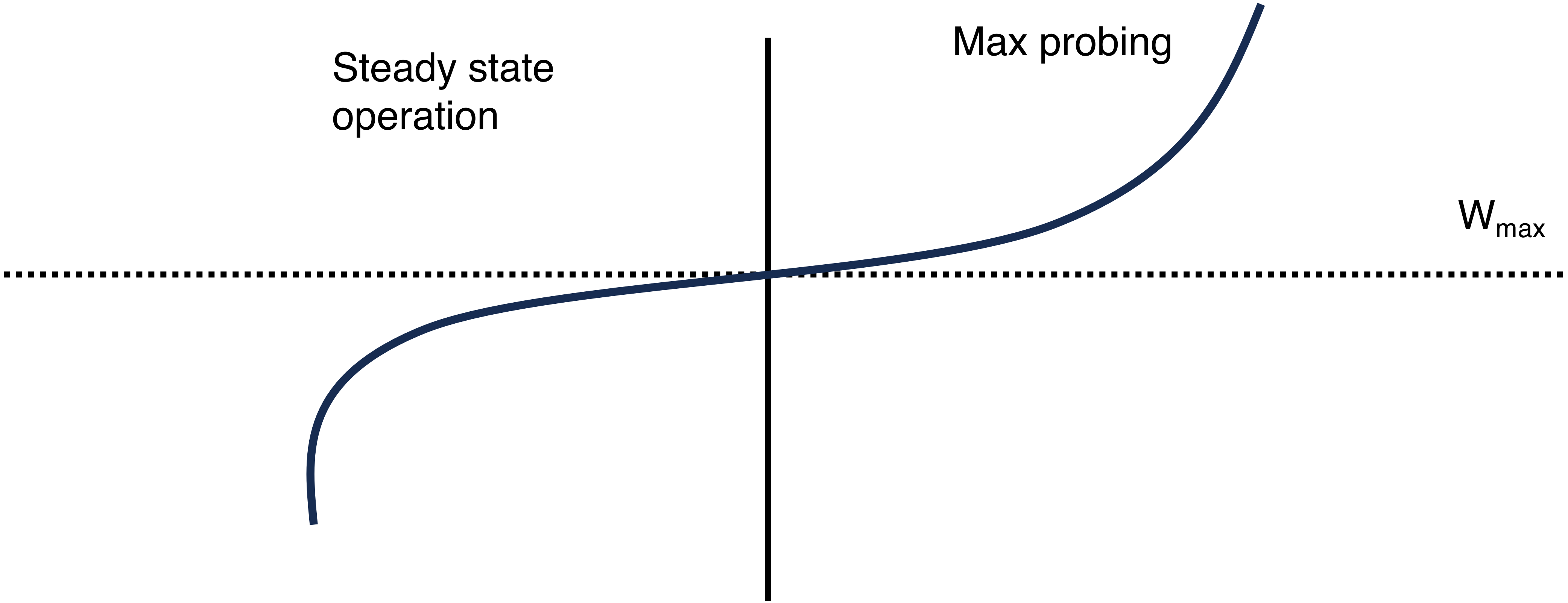
$$Q_{max} = K + N$$

$$Q_{min} = K + N - \frac{1}{2}\sqrt{2N(C \times RTT + K)}$$

$$K > (C \times RTT)/7$$

TCP Cubic

Support faster window growth



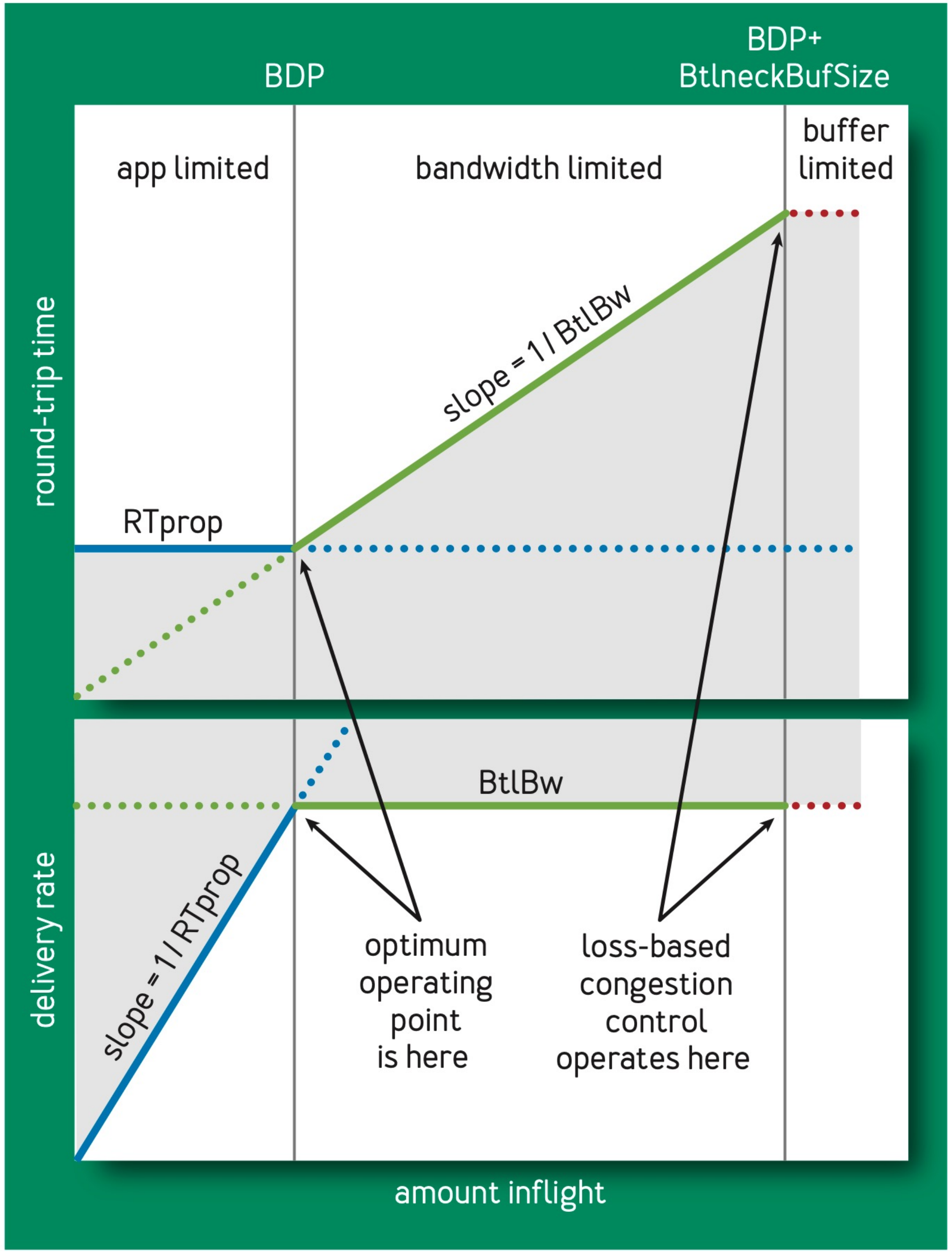
$$W(t) = C(t-K)^3 + W_{max}$$

TCP BBR

Cardwell et al., Google

Principle of Operation

- Find optimal operating point in terms of bottleneck bandwidth and delays
- Cannot simultaneously probe bottleneck bandwidth and propagation delay
- ➔ Occasionally drop window to estimate propagation RTT



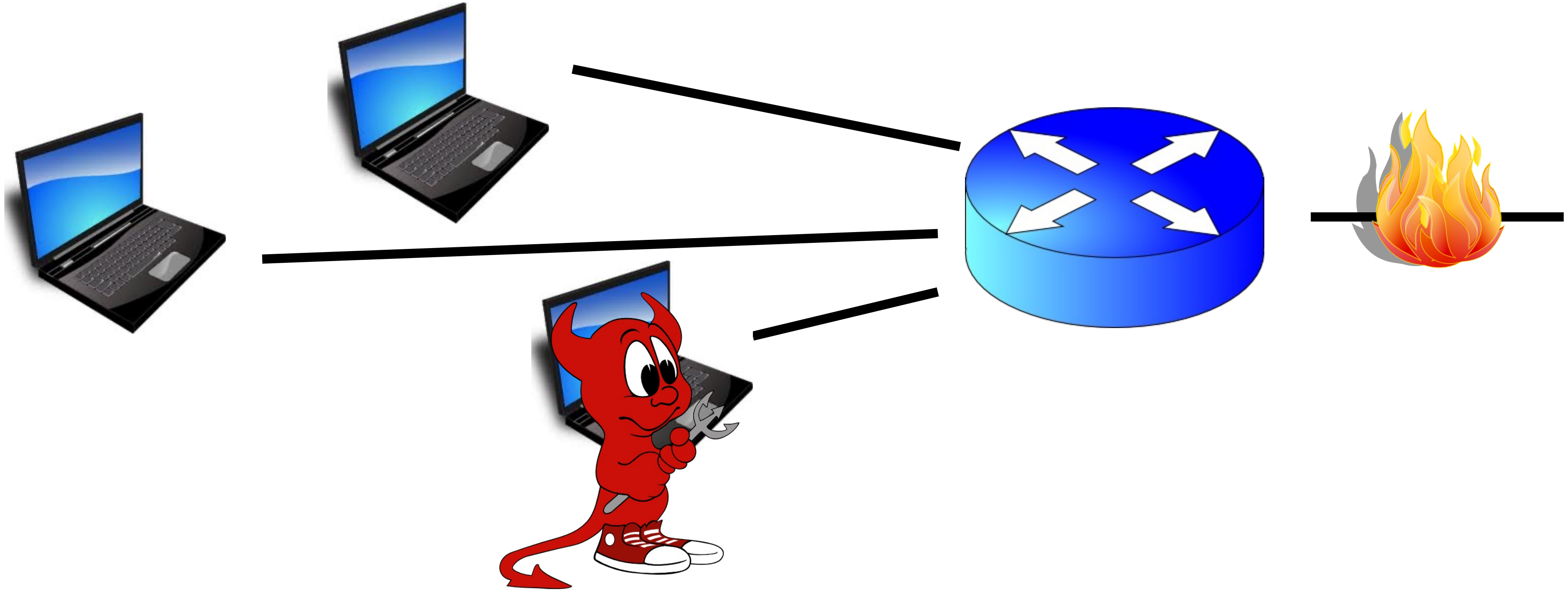
Endpoint algorithms alone are insufficient

The approach that the Internet takes to allocate resources in the network core is to use a distributed algorithm (congestion control) running at endpoints.

This allows the Internet to scale to a large # of endpoints.

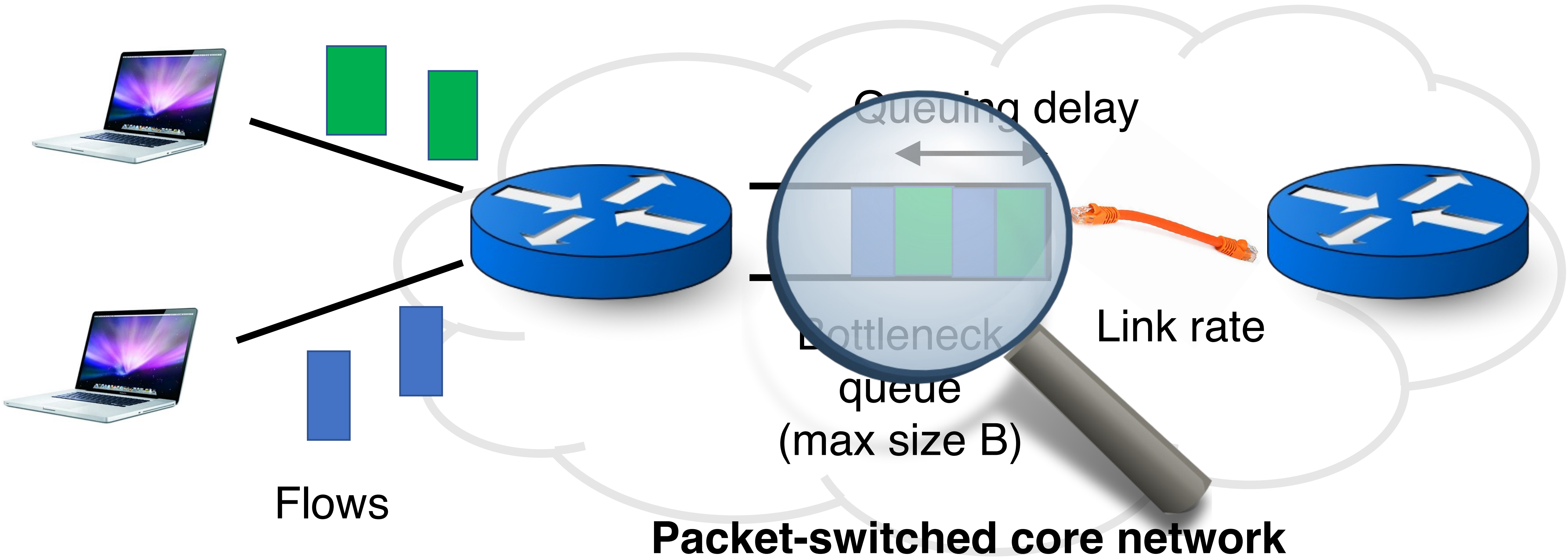
However, it also places trust in endpoints.

Uncooperative, buggy, malicious endpoints

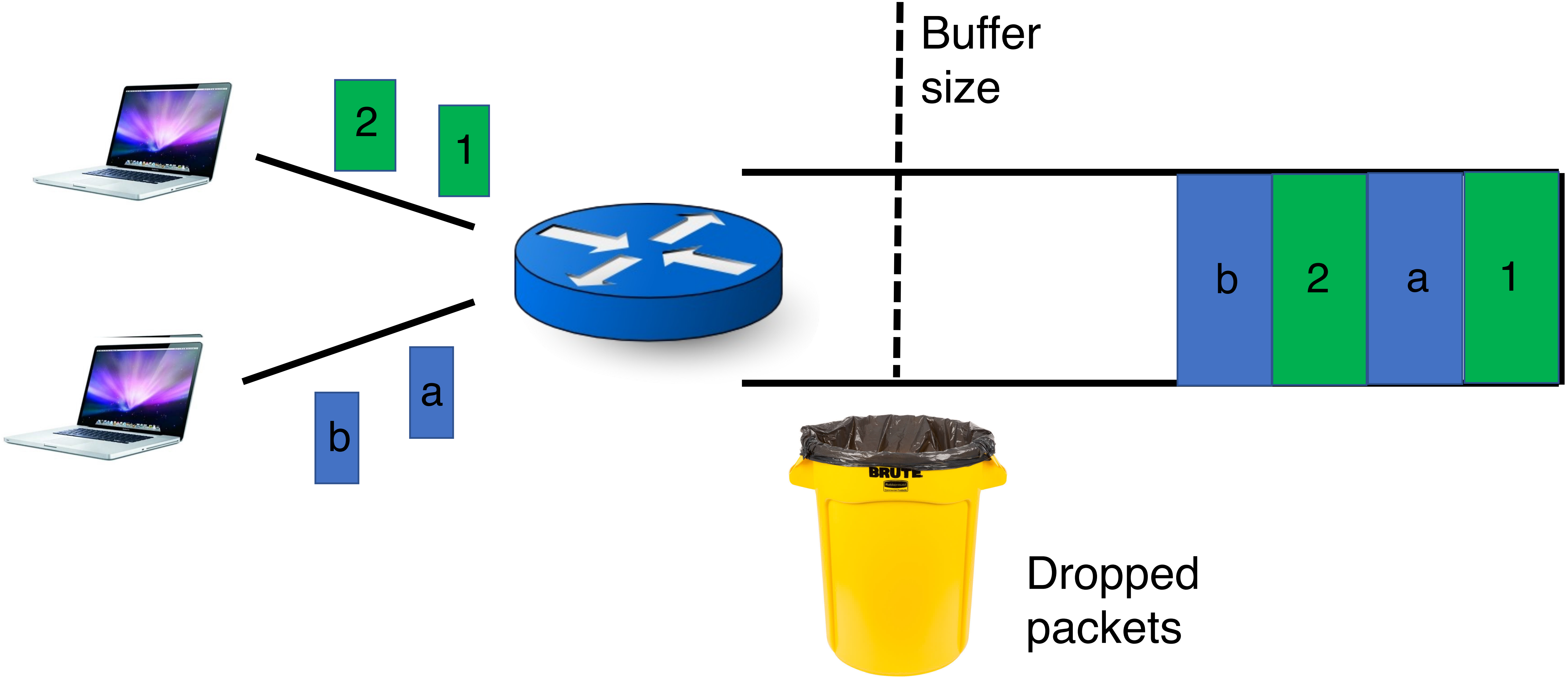


- What if an endpoint is buggy, or malicious?
- We'd like the network core to do something better than best-effort

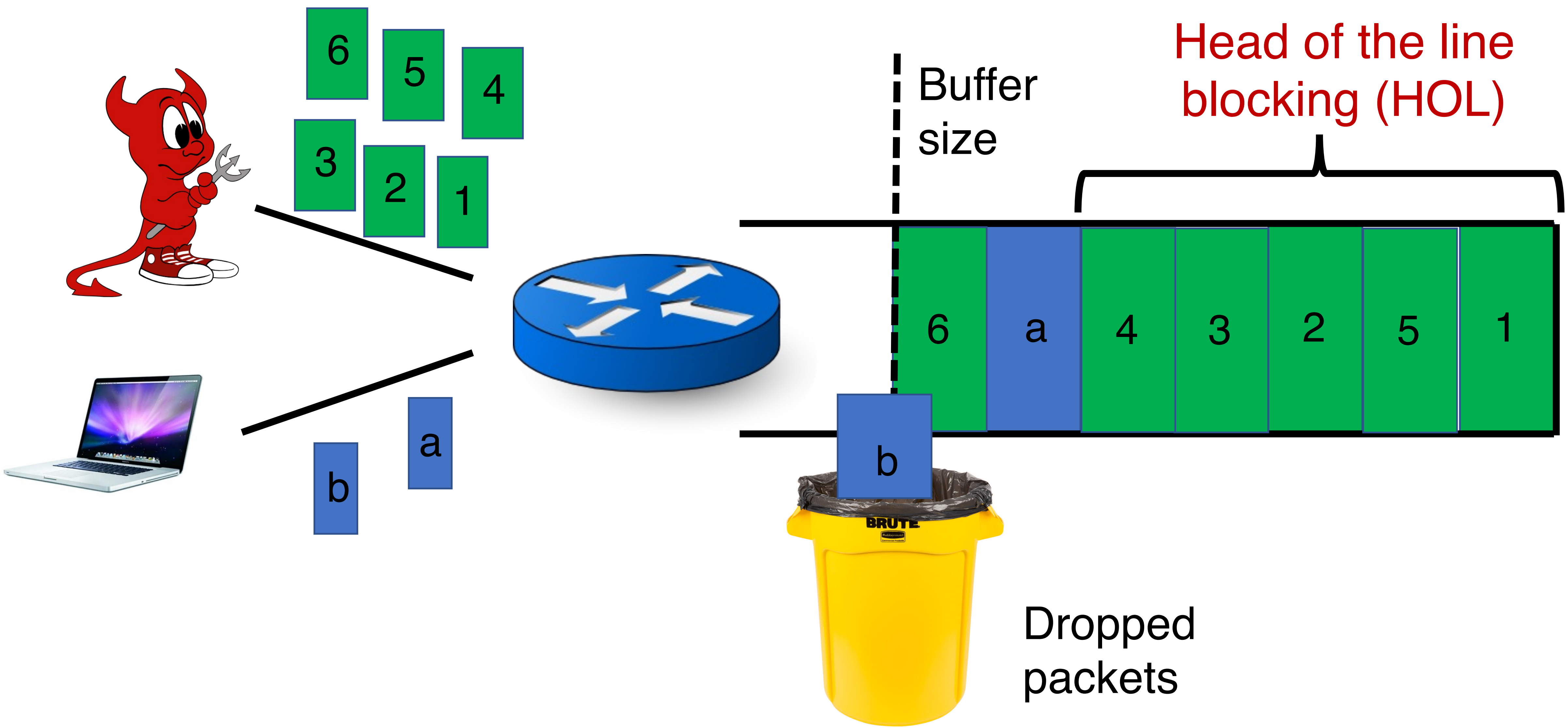
Simplified model of bottleneck link



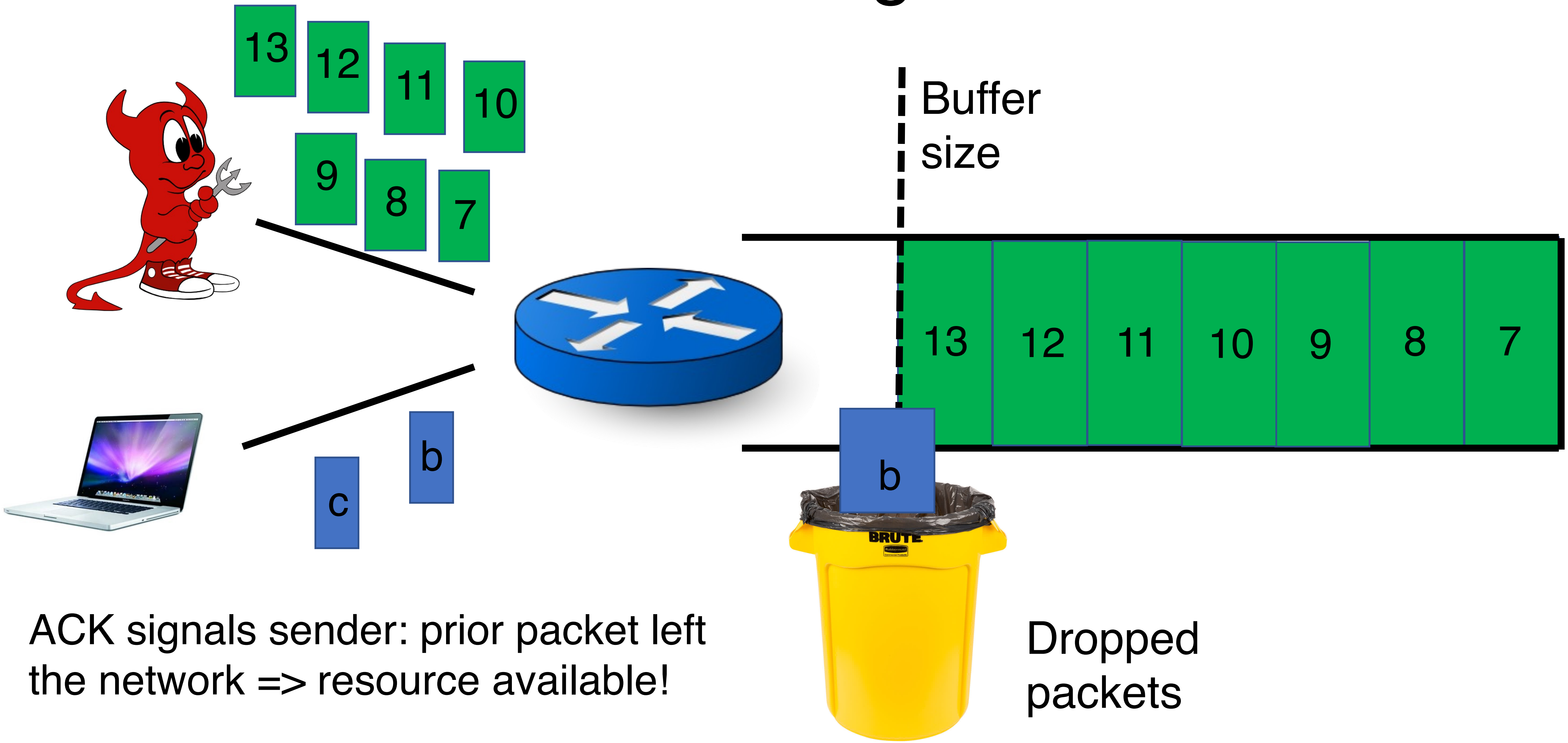
FIFO scheduling + Tail-drop buffer mgmt



FIFO scheduling + Tail-drop buffer mgmt



Next RTT: ACK-clocking makes it worse



ACK signals sender: prior packet left the network => resource available!

Network can be monopolized by bad endpoints

- ACK clocking synchronizes senders to when resource is available
  - Conversely, packet losses desynchronize the sender
- Contending packet arrivals may not be random enough
  - e.g., Blue flow can't capture buffer space for *a few* round-trips
- Can observe this effect when many TCP flows compete
  - Some TCP flows can never get off the ground
- A FIFO tail-drop queue incentivizes sources to misbehave

Packet scheduling disciplines @ routers

- Significantly influences how packets are treated regardless of the endpoint's transmissions
  - Implementations of Quality of Service (QoS) within large networks
  - Implications for net neutrality debates

- Intellectually interesting and influential question
  - Important connections to job scheduling in systems

- Just like in life, how you schedule work is highly impactful

Relationship: scheduling & transport

- Packet scheduling is dealing with things transport has already put into the network
- Transport requires a few round-trip times to react; scheduling does something “immediately”
- If you could schedule transmission out of the endpoint, you could get them to zoom through the network without waiting anywhere in queues
- In modern clusters, goal of transport is to often act as if a centralized scheduler directly chose the packet to transmit from the endpoint (leaving other packets waiting at the endpoint)