Data Center Transport

Lecture 10

Srinivas Narayana

http://www.cs.rutgers.edu/~sn624/553-S23

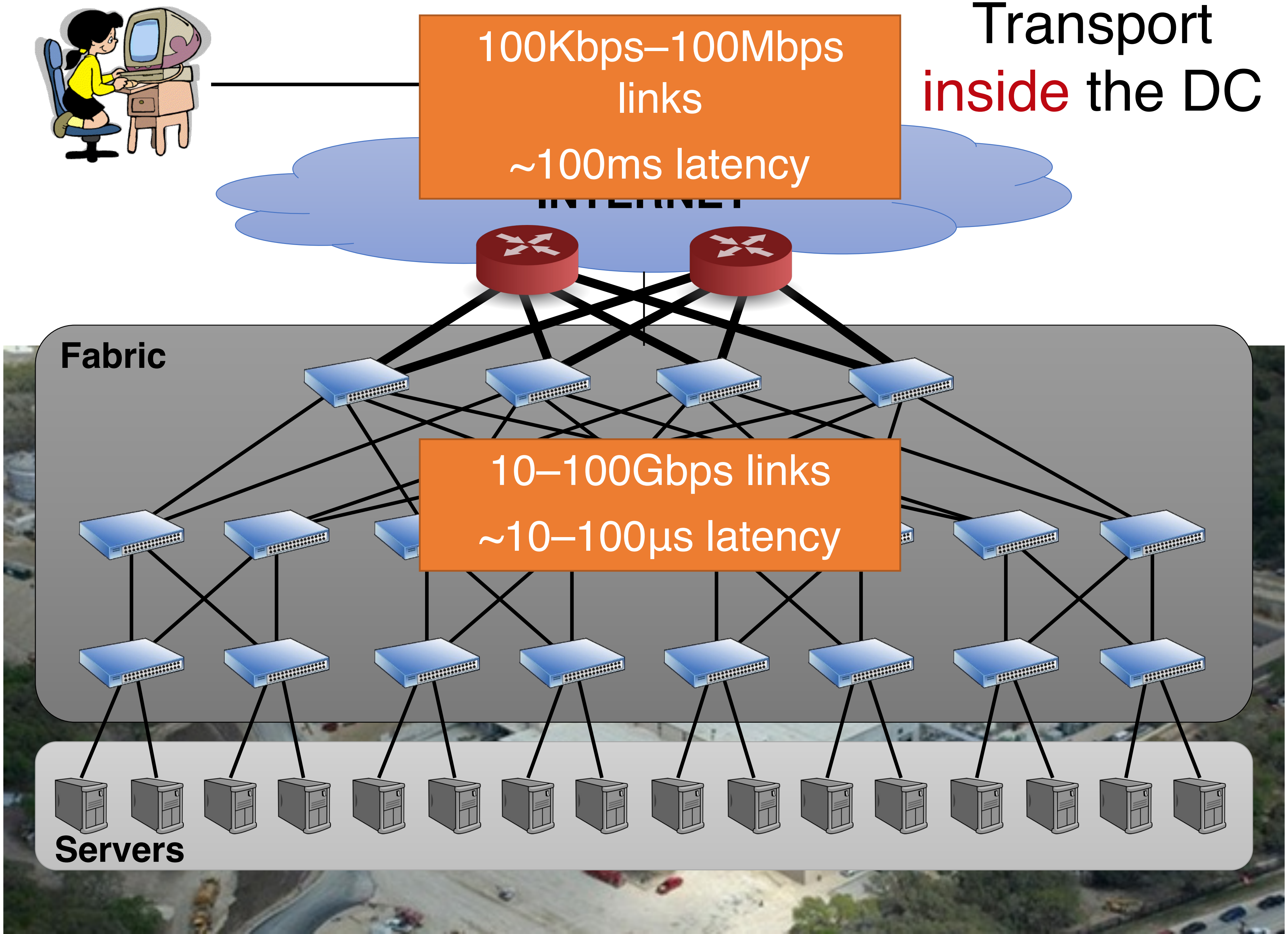
Parts of this lecture were heavily adapted from slides by Mohammad Alizadeh

RUTGERS
UNIVERSITY | NEW BRUNSWICK

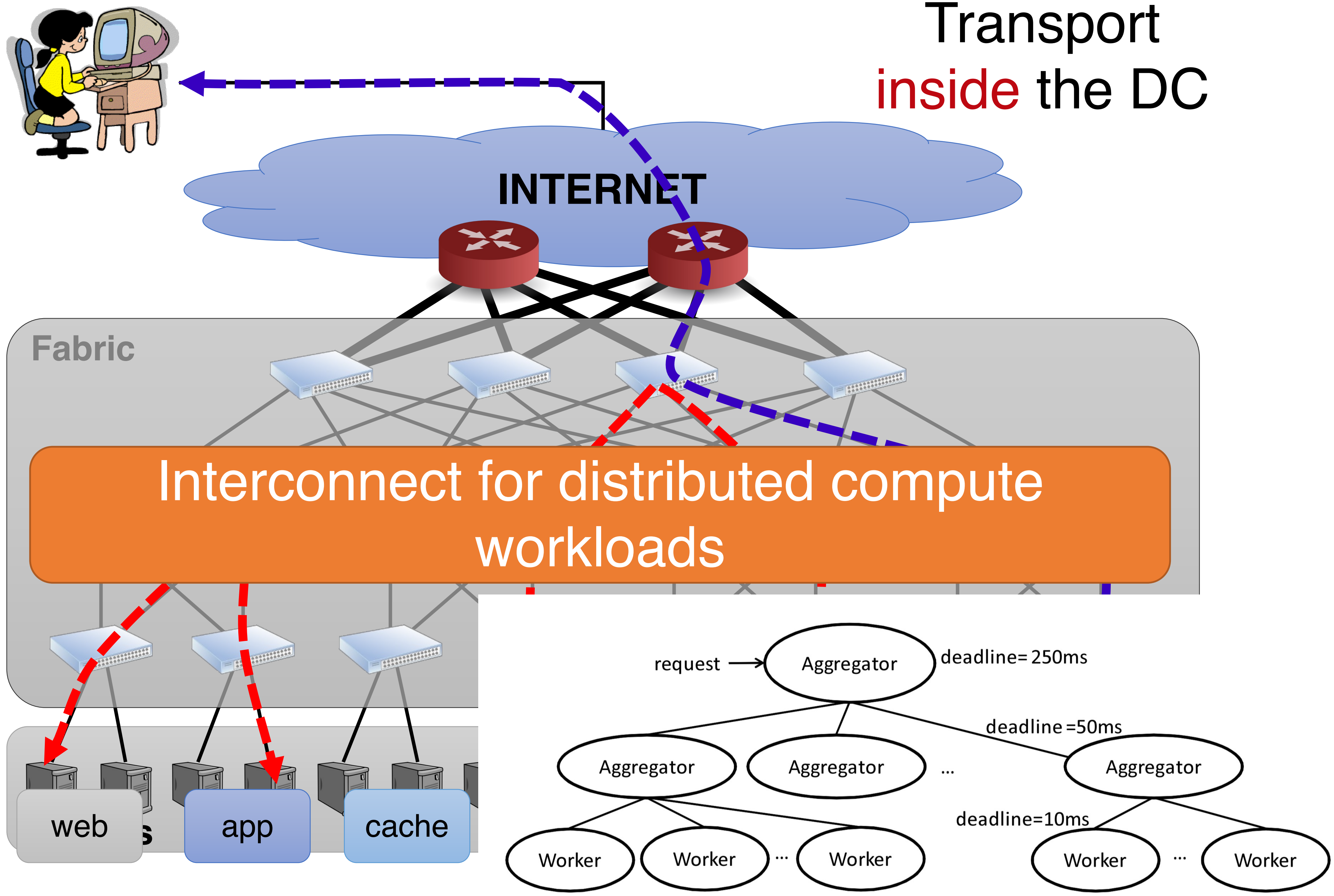
DC Transport Requirements

High throughput, low latency, burst tolerance

Transport **inside** the DC



Transport inside the DC



Data center workloads

Coexistence creates some performance impairments...

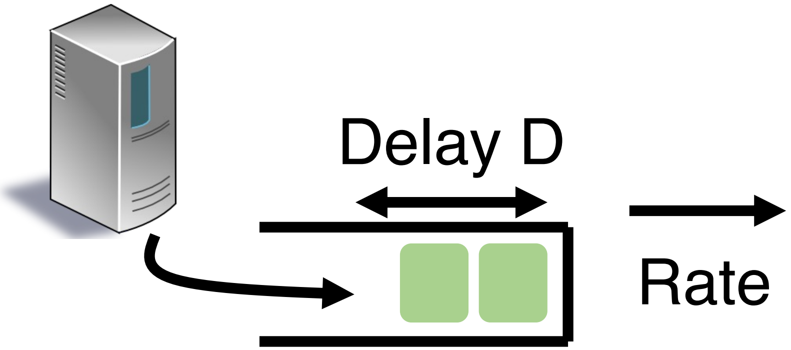
- Mice and Elephants

- Short messages
  (e.g., query, coordination)

→ Low Latency

- Large flows
  (e.g., data update, backup)

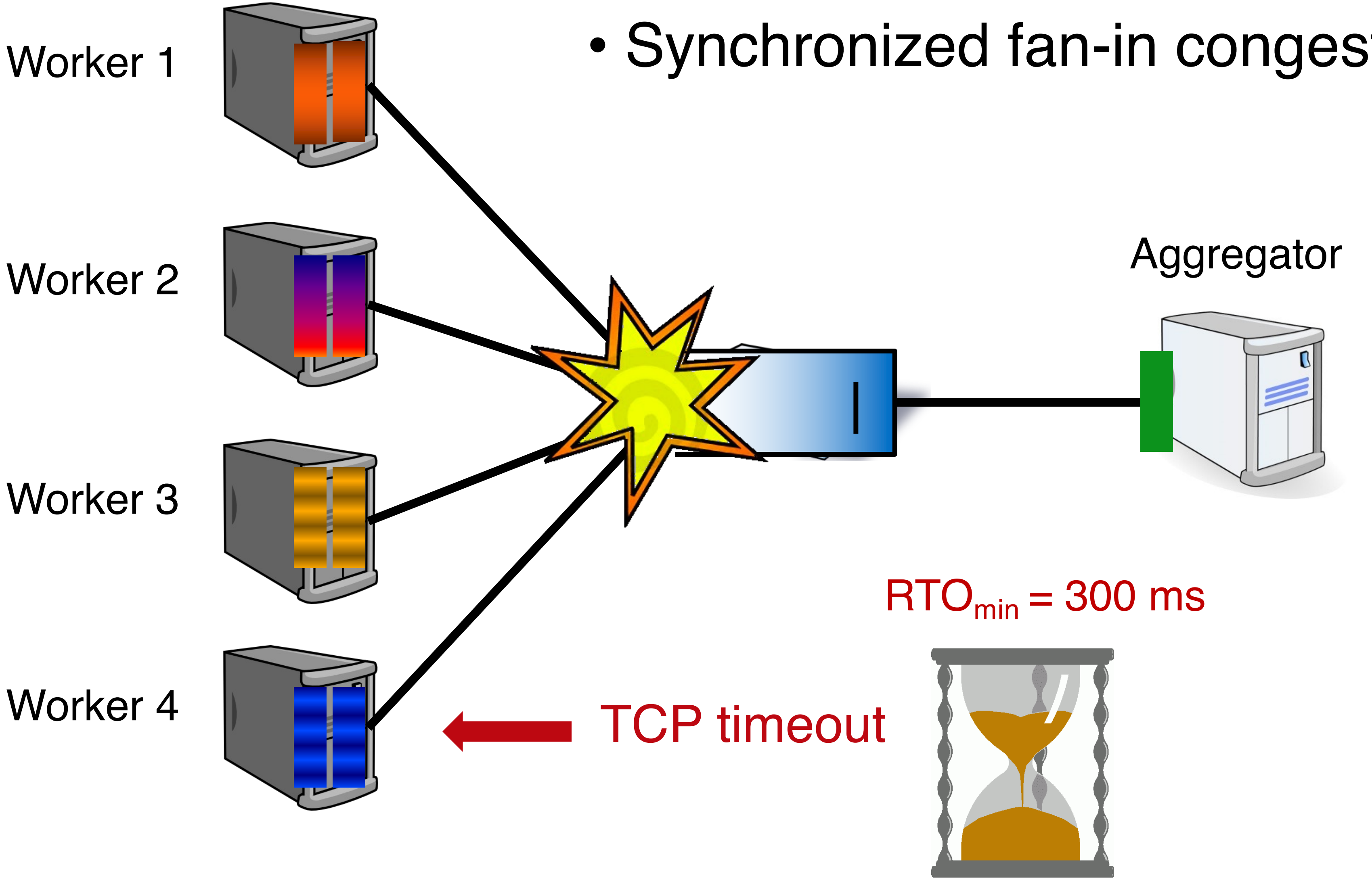
→ High Throughput



Completion time
D + B/R

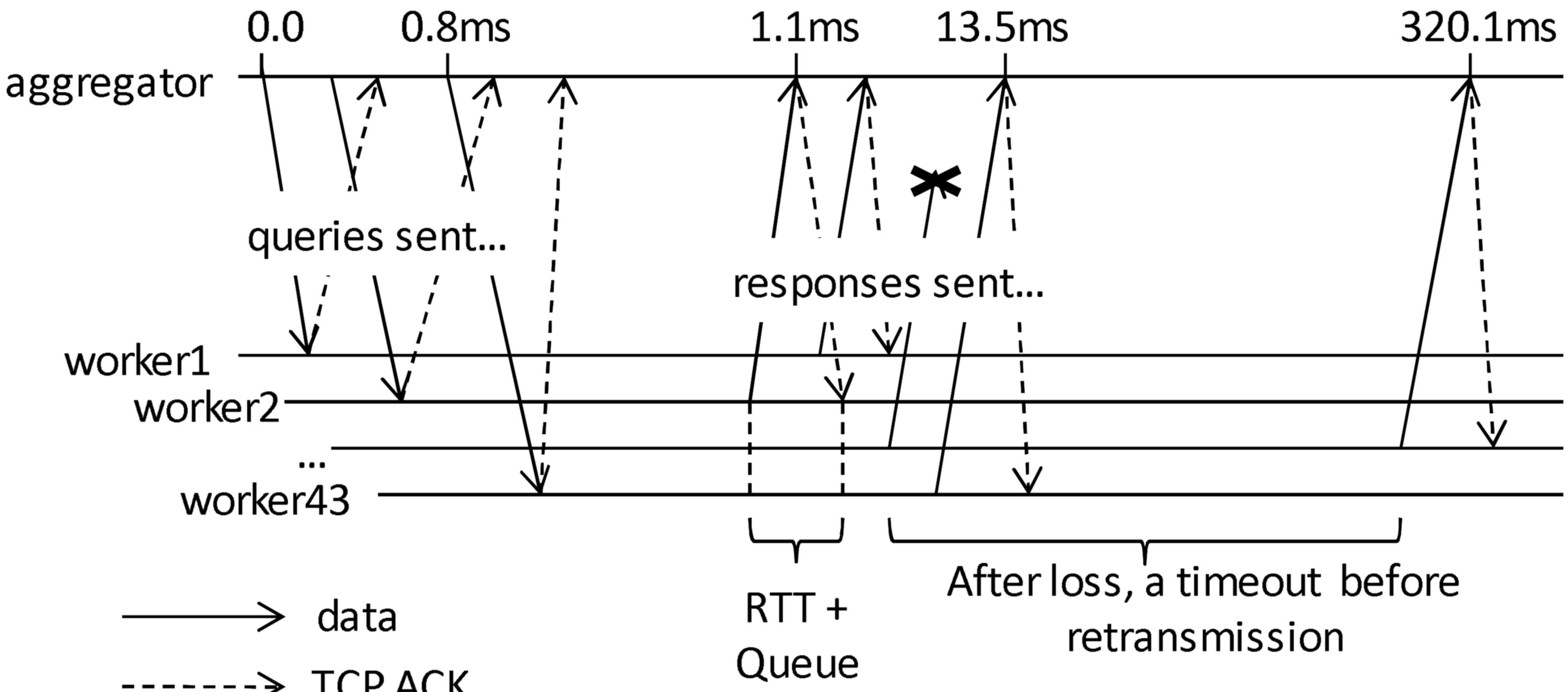
(1) Incast

- Synchronized fan-in congestion



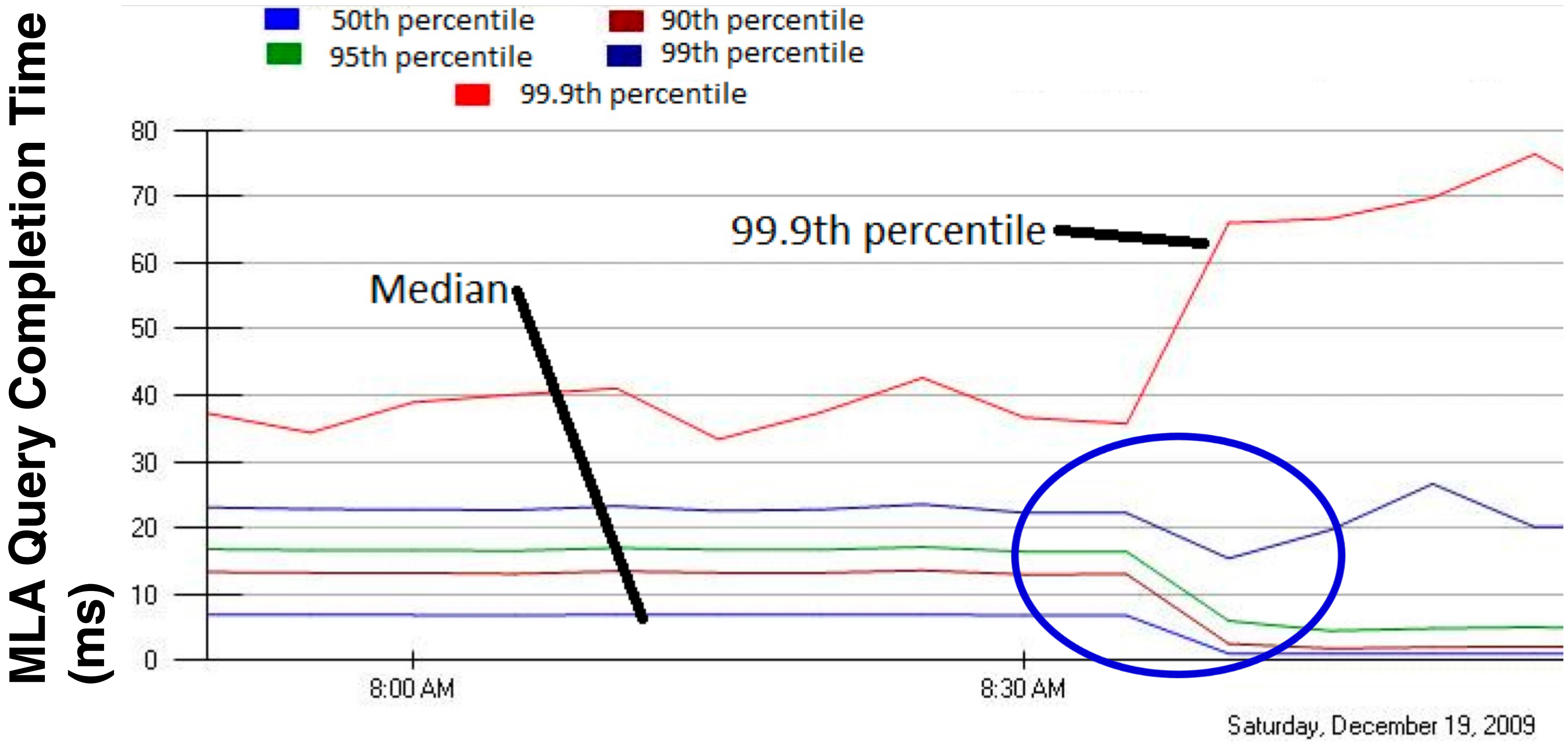
Vasudevan et al. (SIGCOMM’09)

Trace of a real incast event



One option: reduce RTO to mitigate this

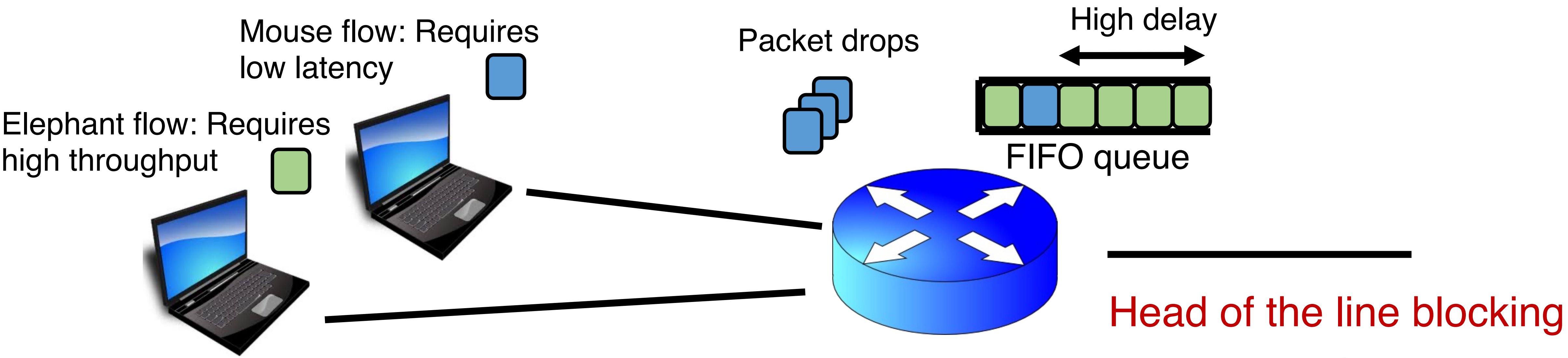
Another option: Jittering to avoid sync



- Jittering switched off around 8:30 am.

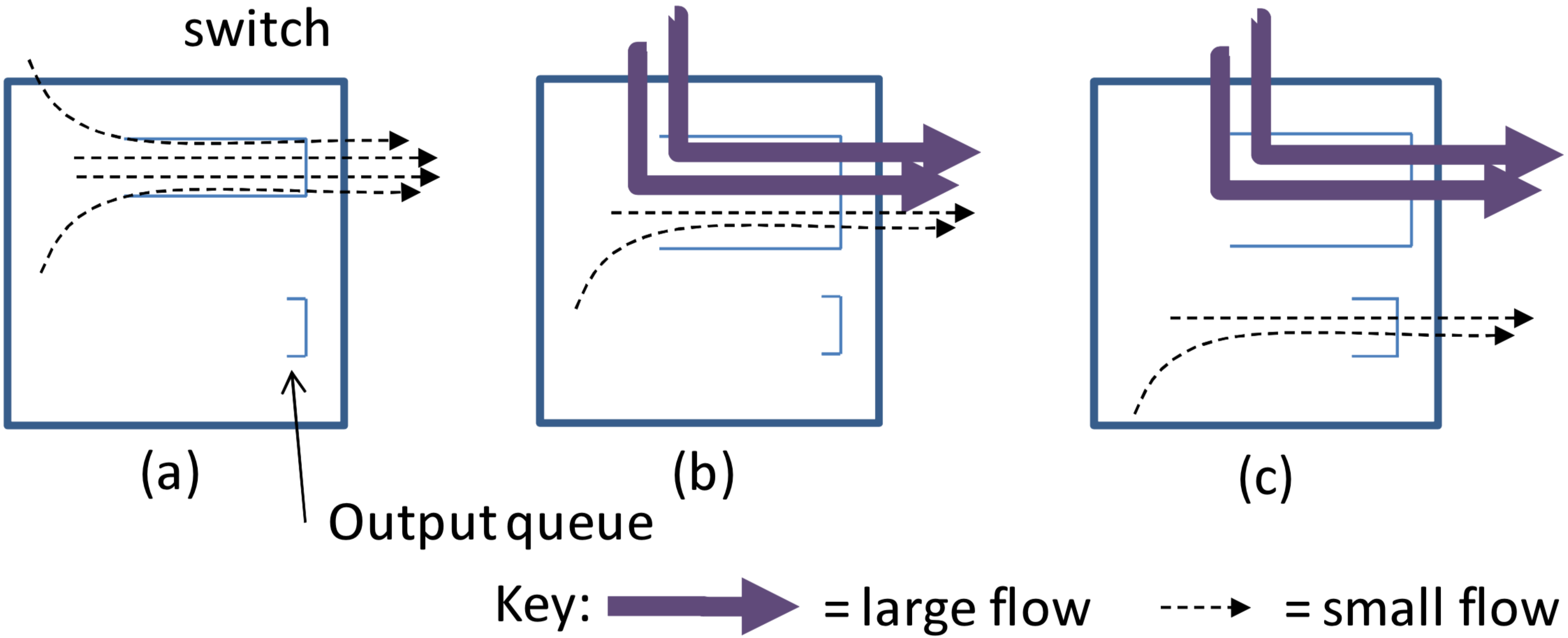
Jittering trades off median for high percentiles

(2) Head of line blocking: Queue buildup



- Resource contention occurs in the core of the network
- Congestion control will react, but may be too little & too late:
  - Congestion control can't prevent packet drops "now"
  - Congestion control won't prevent high-sending-rate flows from inflicting large delays or recurring drops

(2) HOL Blocking on a switch port

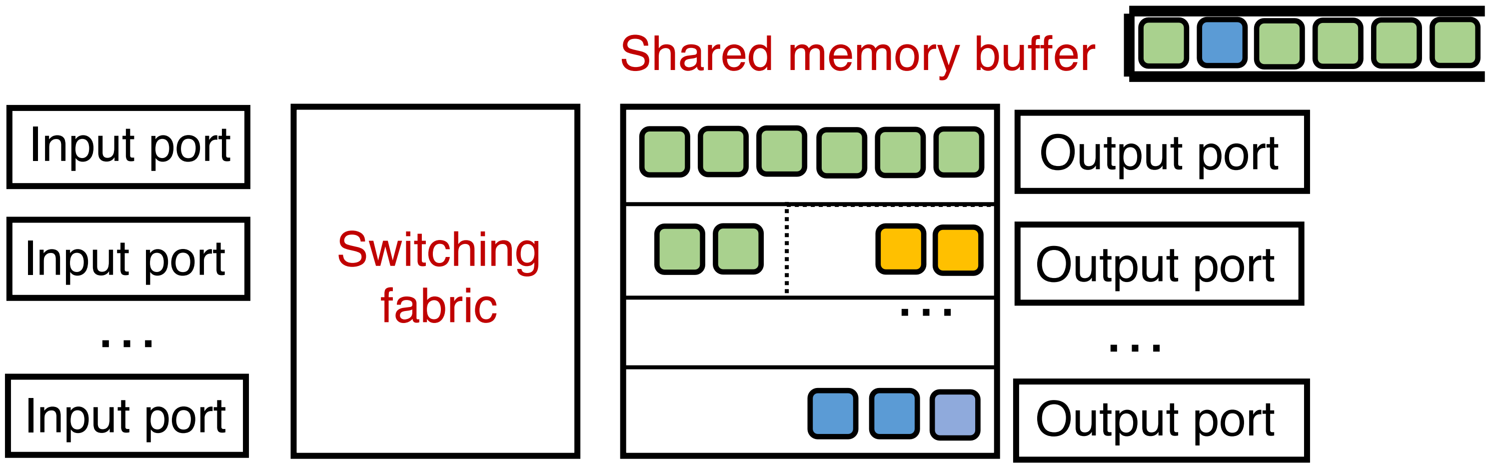


Queue buildup on a port increases delays for all flows using that port (head of the line blocking)

Reducing RTO doesn't help latency when there are no drops

(3) Shared memory buffering

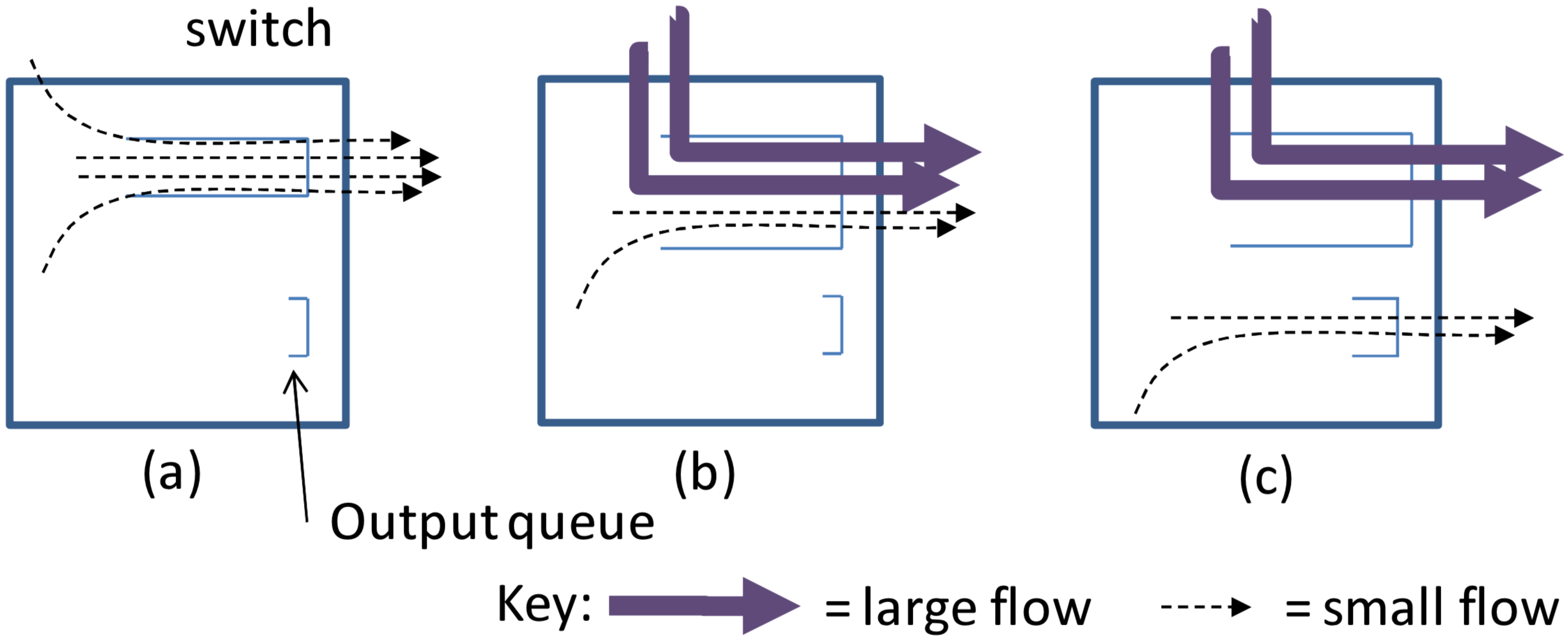
- Where should the packets not currently serviced wait?
- Input-queued vs. output-queued (preferable design)
- Buffer management: how to put packets into the buffer
- Scheduling: how to schedule packets leaving the buffer



Static buffer allocation

Dynamic buffer allocation (statistical multiplexing)

(3) Buffer Pressure



Shared memory buffering means that queues building up on a different port can also impact flows.

Need to keep queues small.
Use delay-based CC?

- Keep just a few packets in queues by observing delays

$$\texttt{queue\_use} = \texttt{cwnd} - \mathrm{BWE} \times \mathrm{RTT}_{\mathrm{noLoad}} = \texttt{cwnd} \times (1 - \mathrm{RTT}_{\mathrm{noLoad}} / \mathrm{RTT}_{\mathrm{actual}})$$

- Adjust window such that only a few packets are in queue

$$\alpha \leq \mathrm{queue\_use} \leq \beta$$

- RTT estimates need to be very accurate and precise
  - Difficult in low-RTT data centers.
  - Challenges: Software queueing & scheduling delays. Timer tick res

Data Center TCP (DCTCP)

Design of the congestion control algorithm

Review: TCP congestion control

- Keep some in-flight (un-ACK'ed) packets: congestion window

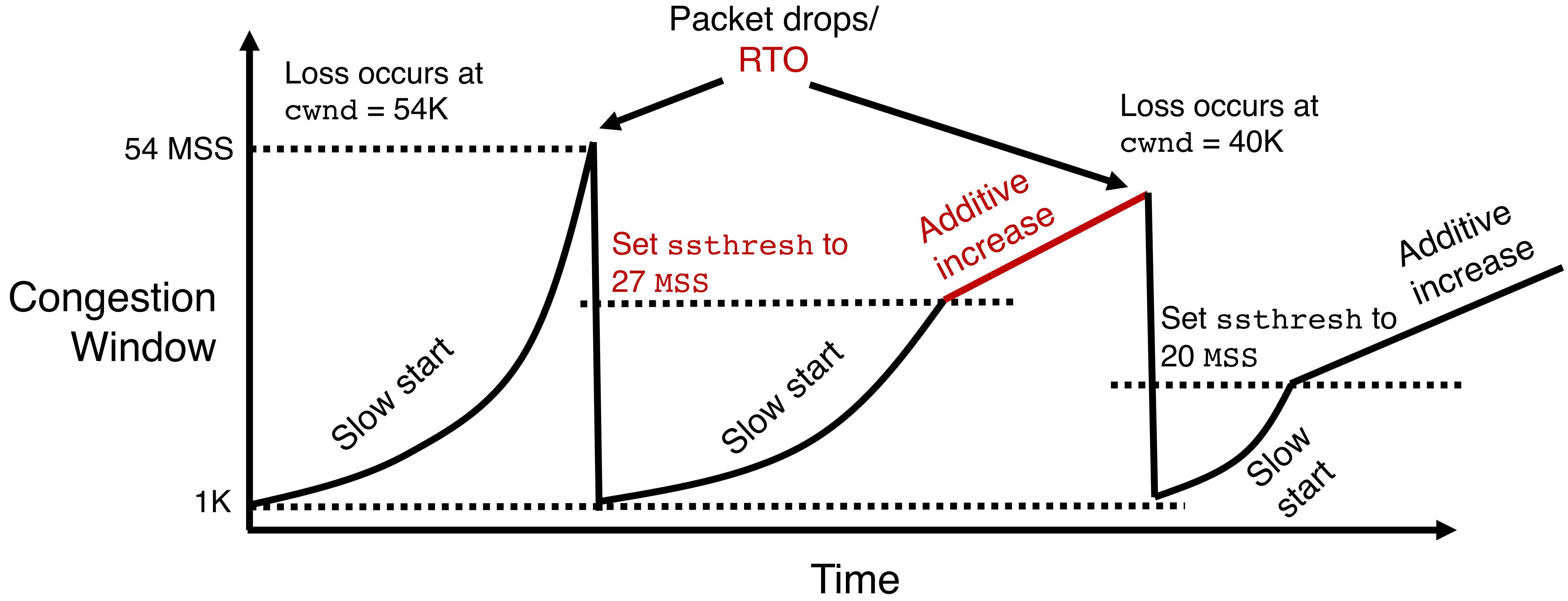
- Adjust window based on several algorithms: TCP New Reno:
  - Startup: slow start
  - Steady state: AIMD
  - Loss: fast retransmission, fast recovery

- Main question for this lecture:
  - (How) should this design change for data centers?

Behavior of Additive Increase

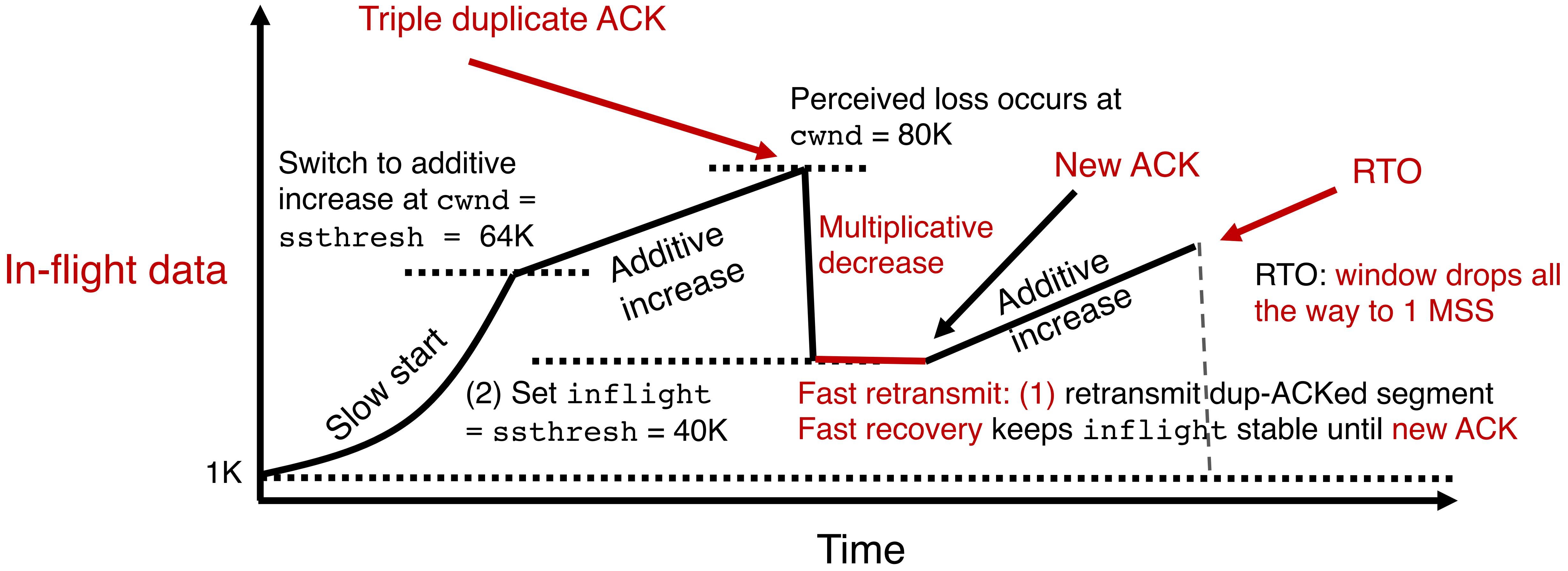
Say `MSS` = 1 KByte

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



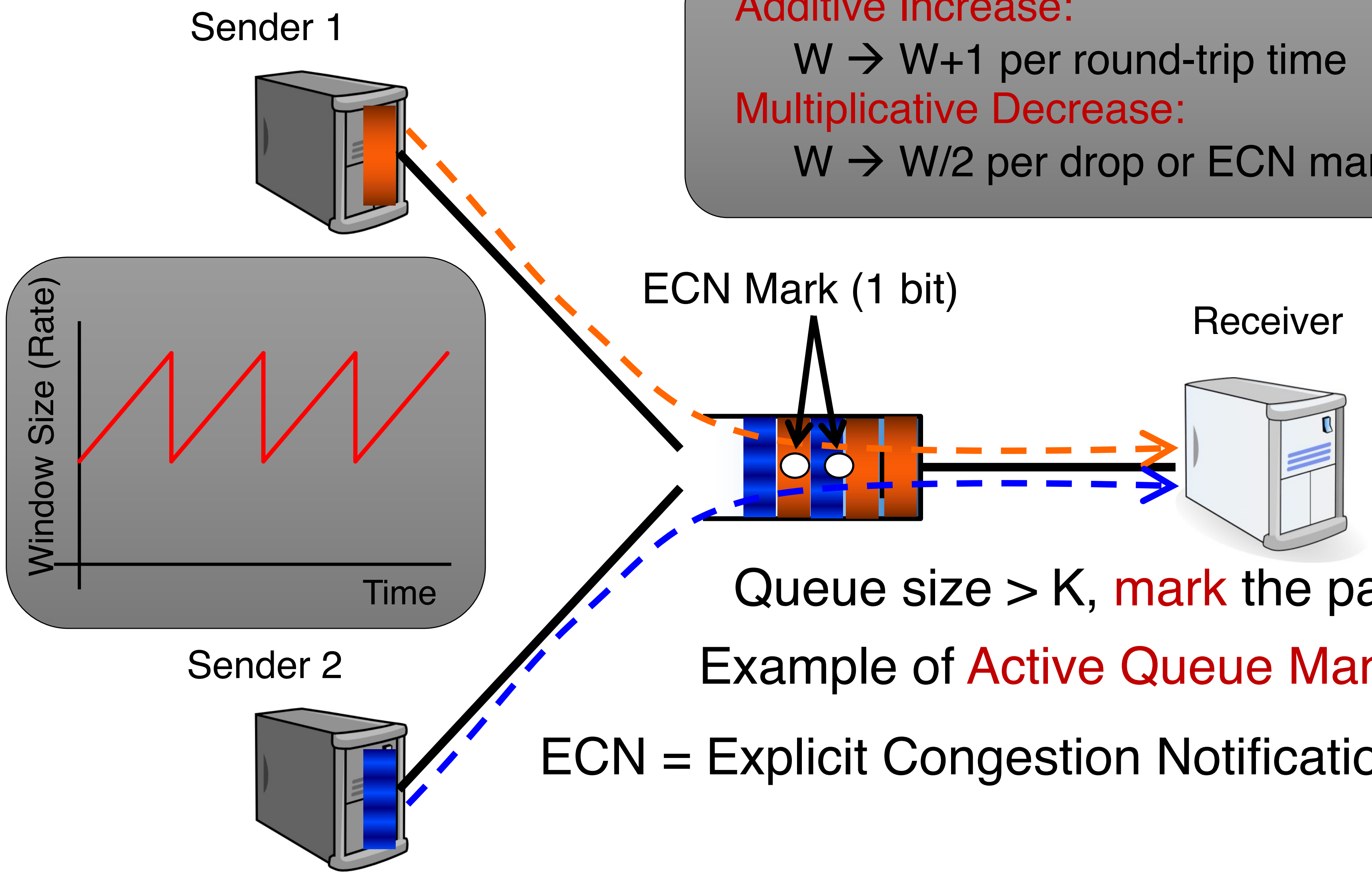
Additive Increase/Multiplicative Decrease

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



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

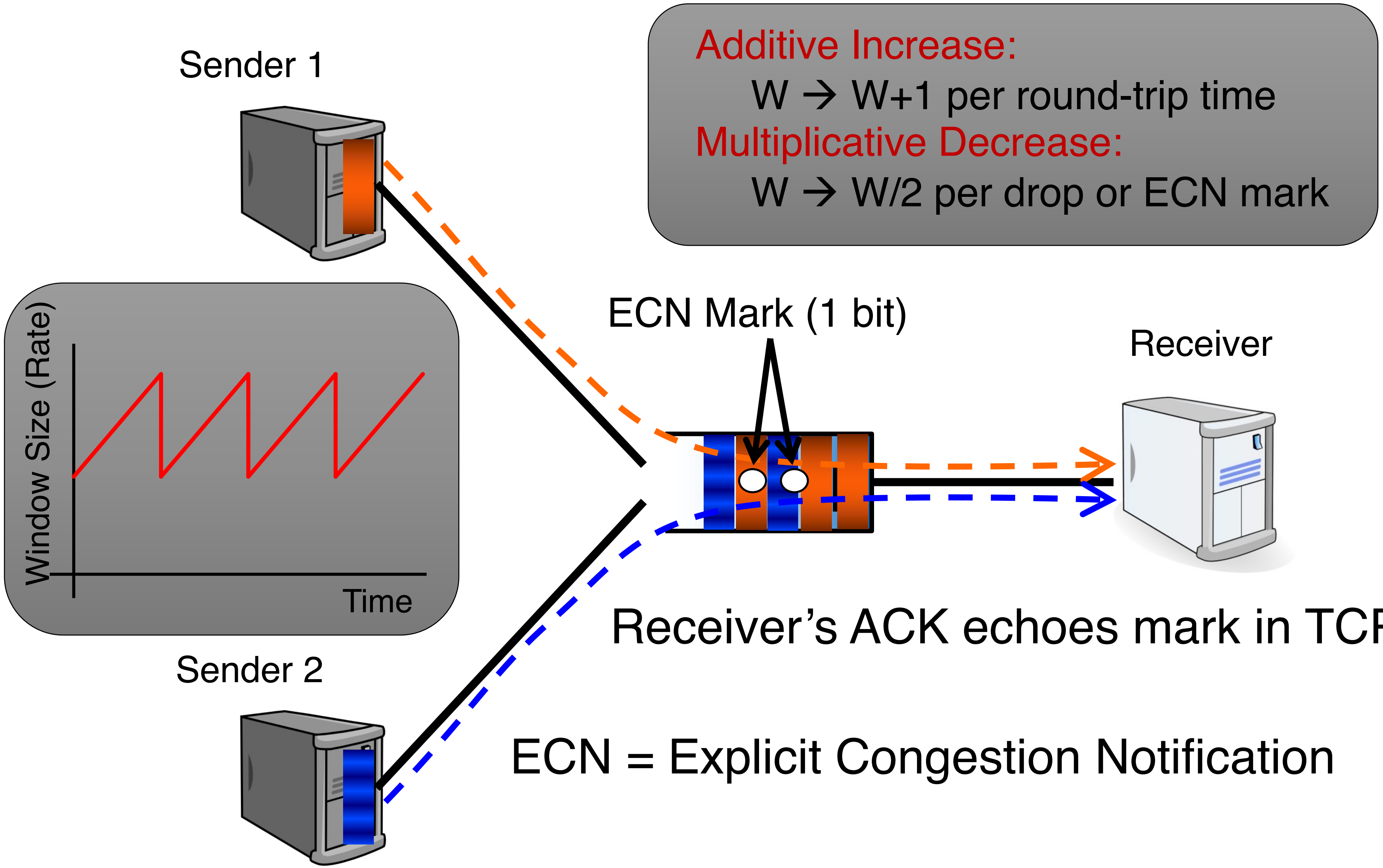
- `00` – Not ECN-Capable Transport, Not-ECT
- `01` – ECN Capable Transport(1), ECT(1)
- `10` – ECN Capable Transport(0), ECT(0)
- `11` – Congestion Experienced, CE.

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><th>0</th><th>0</th><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><th>4</th><th>32</th><td colspan="16">Identification</td><td colspan="3">Flags</td><td colspan="13">Fragment Offset</td></tr>
<tr><th>8</th><th>64</th><td colspan="8">Time To Live</td><td colspan="8">Protocol</td><td colspan="16">Header Checksum</td></tr>
<tr><th>12</th><th>96</th><td colspan="32">Source IP Address</td></tr>
<tr><th>16</th><th>128</th><td colspan="32">Destination IP Address</td></tr>
<tr><th>20</th><th>160</th><td colspan="32" rowspan="3">Options (if IHL > 5)</td></tr>
<tr><th>⋮</th><th>⋮</th></tr>
<tr><th>56</th><th>448</th></tr>
</table>

Explicit Congestion Notification



Receiver's ACK echoes mark in TCP header

ECN = Explicit Congestion Notification

ECN on the TCP header

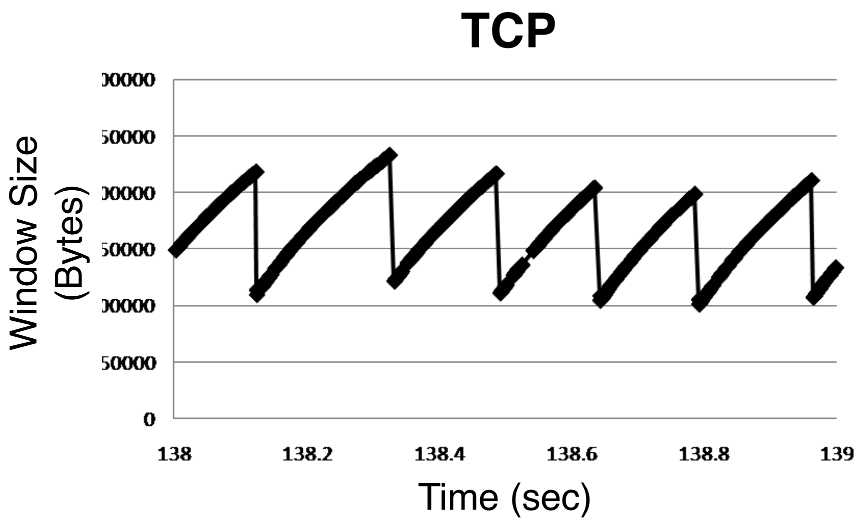
<table>
<caption>TCP segment header</caption>
<tr><th>Offsets</th><th>Octet</th><th colspan="8">0</th><th colspan="8">1</th><th colspan="8">2</th><th colspan="8">3</th></tr>
<tr><th>Octet</th><th>Bit</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th><th>7</th><th>6</th>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

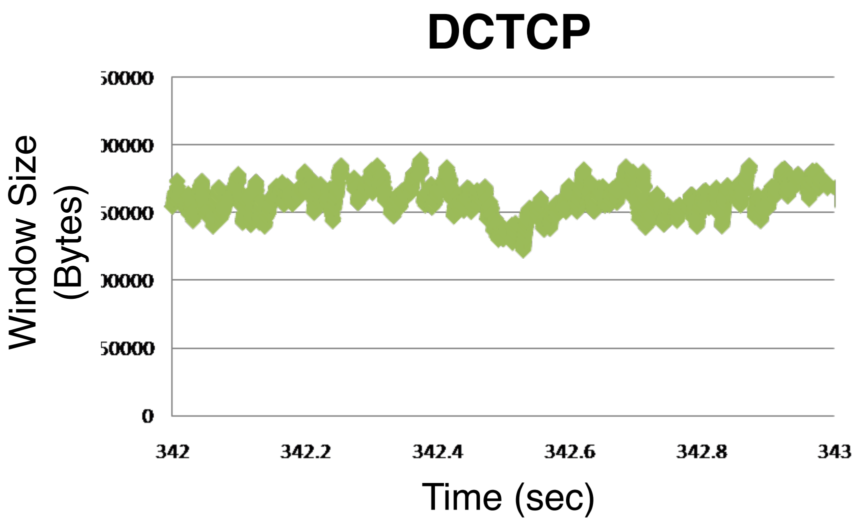
DCTCP: Main idea

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

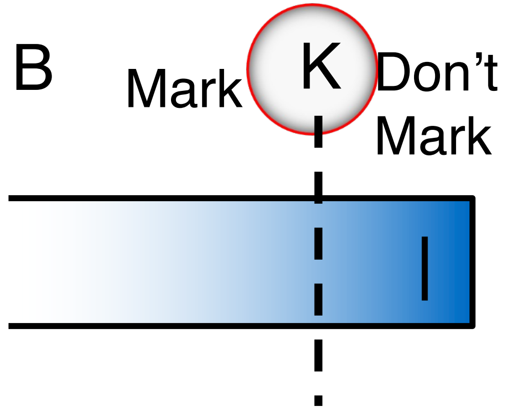
DCTCP: Main idea

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

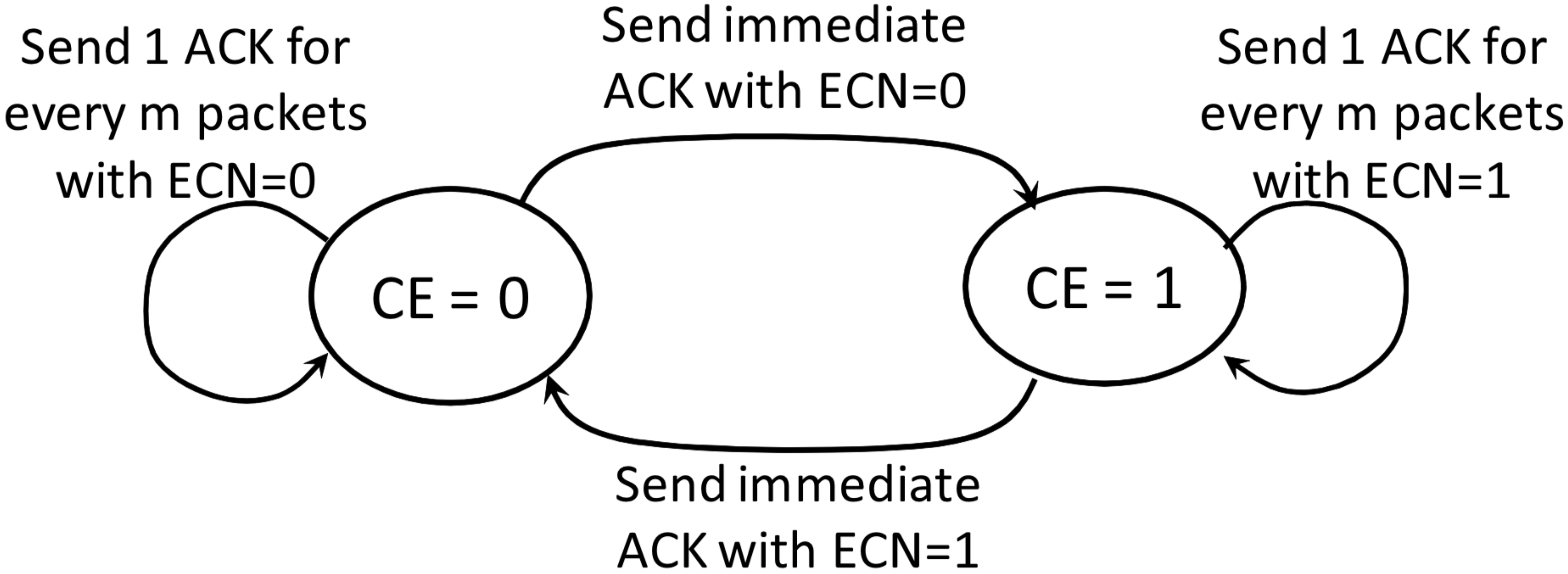
Delayed ACKs

- Not every packet is ACKnowledged by receiver

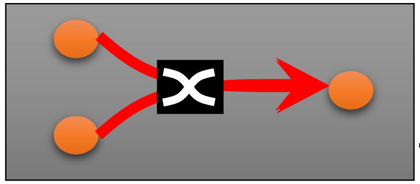
- Too many ACKs: increase packet processing load
  - Typical policy: ACK every m packets, or after sender has paused transmitting for a delayed ACK timeout

- How to allow the sender to see the full stream of ECN marks?

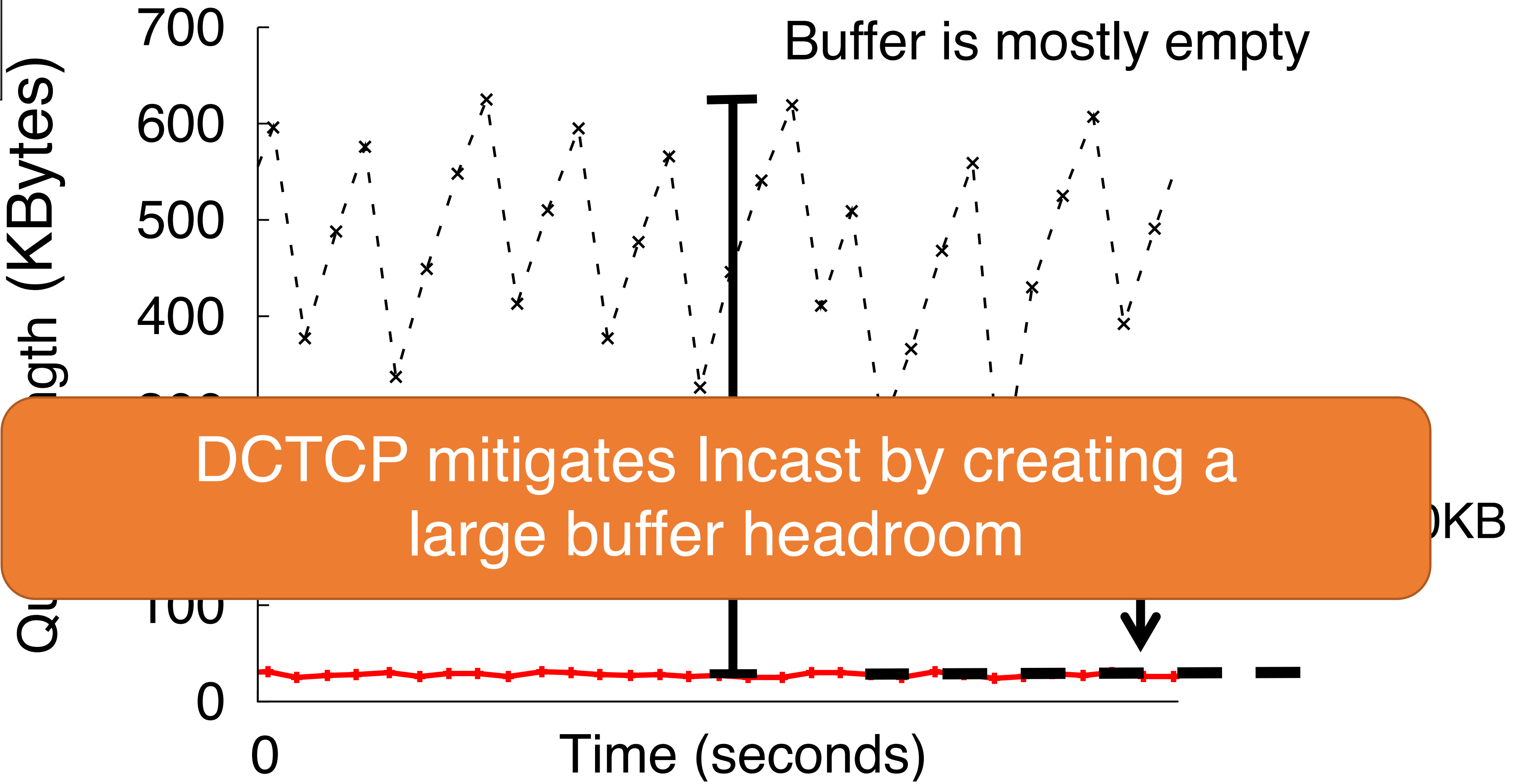
Efficient and “lossless” ACK generation



DCTCP vs TCP



Experiment: 2 flows (Win 7 stack), Broadcom 1Gbps Switch



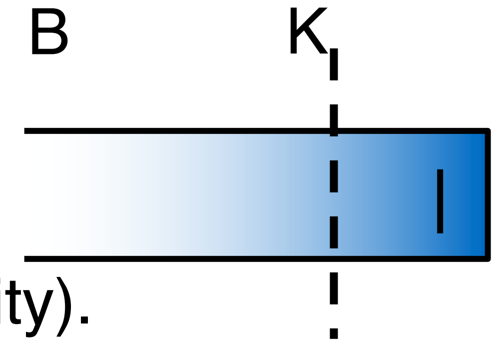
DCTCP mitigates Incast by creating a large buffer headroom

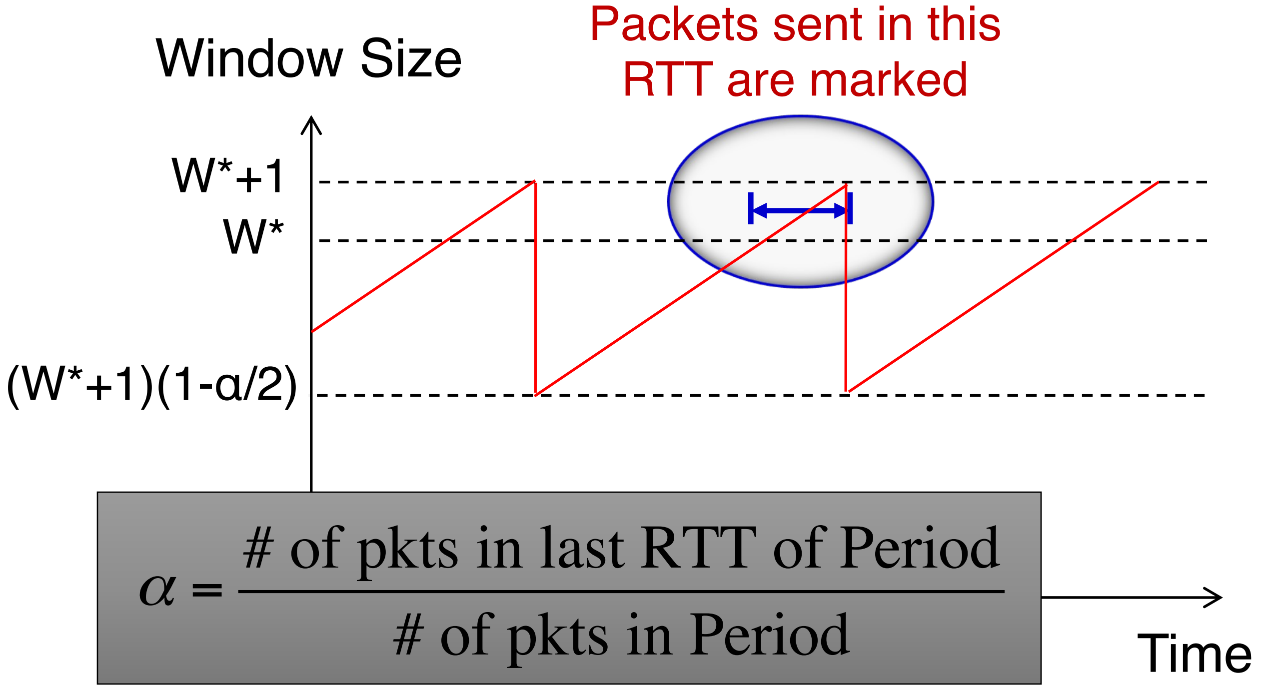
Why it works

1. Low Latency
   - ✓ Small buffer occupancies → low queuing delay
2. High Throughput
   - ✓ ECN averaging → smooth rate adjustments, low variance
3. High Burst Tolerance
   - ✓ Large buffer headroom → bursts fit
   - ✓ Aggressive marking → sources react before packets are dropped

Setting parameters: A bit of analysis

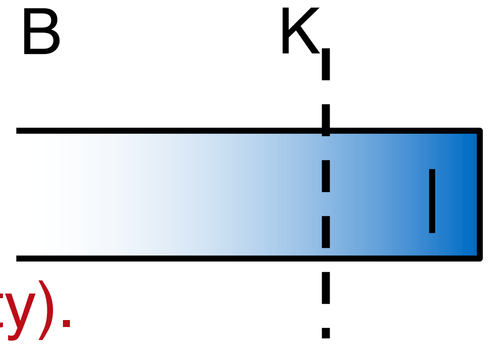
- How much buffering does DCTCP need for 100% throughput?
  - Need to quantify queue size oscillations (stability).





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

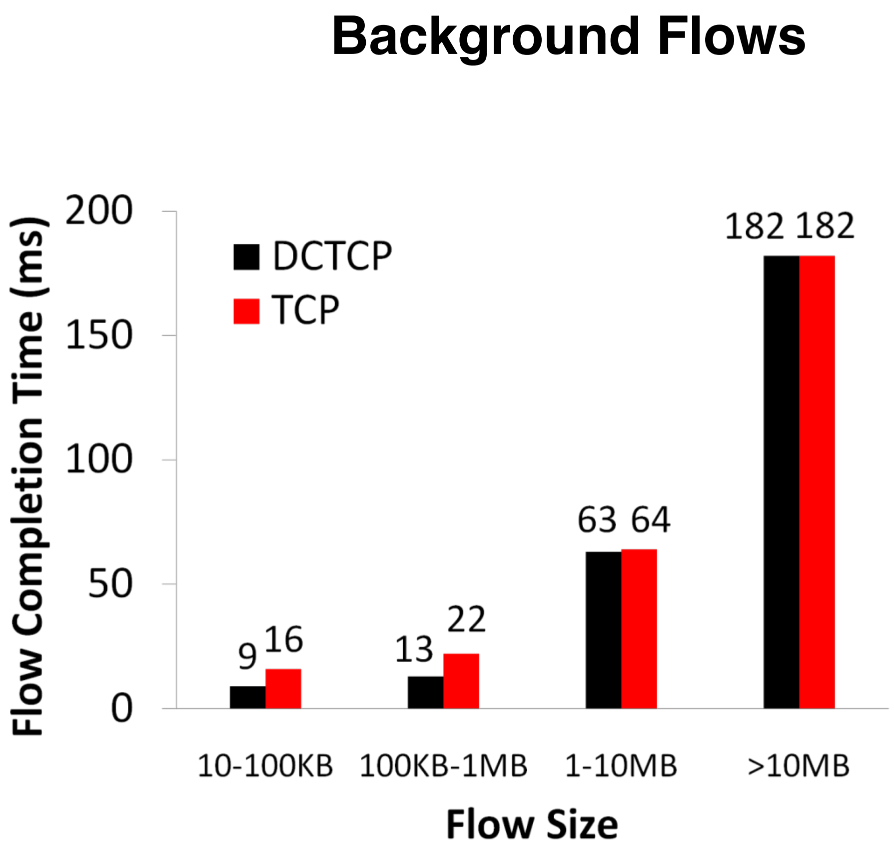
Setting parameters: A bit of analysis

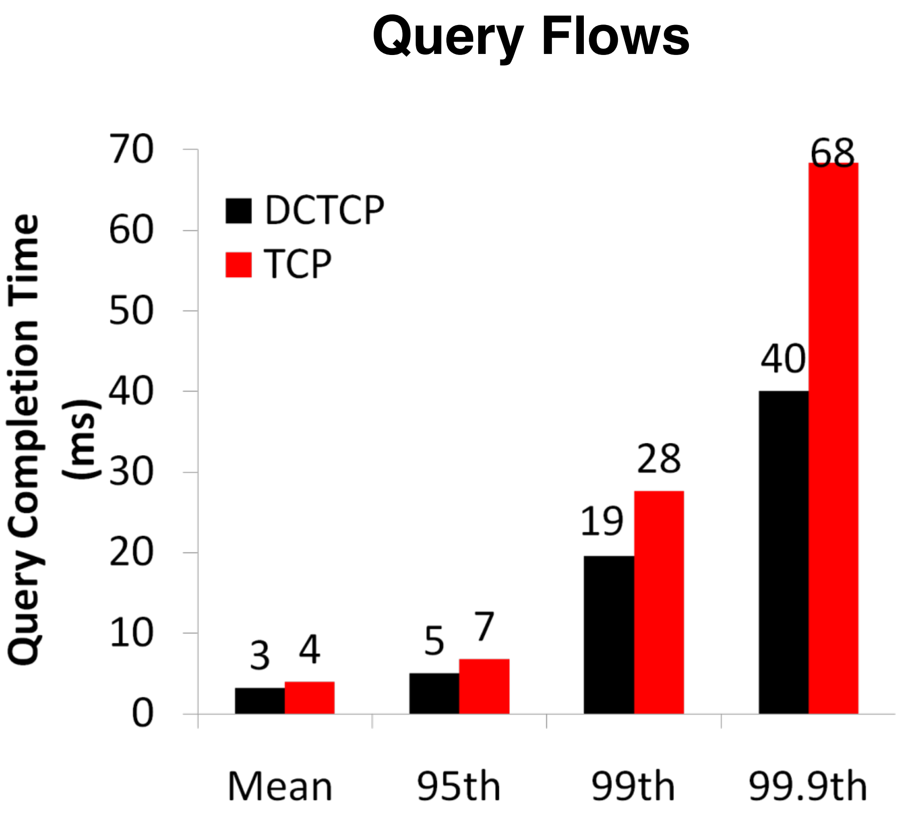


- How small can queues be without loss of throughput?
  - ➢ Need to quantify queue size oscillations (Stability).

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

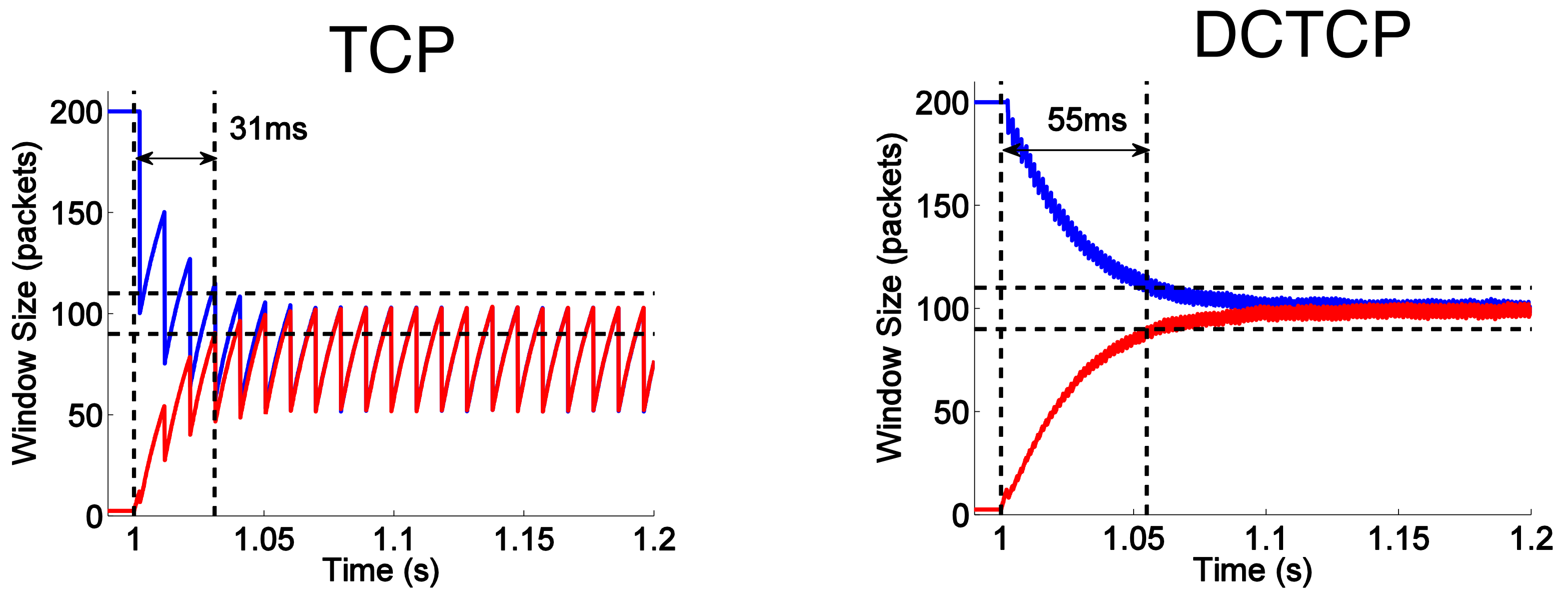
Bing benchmark (baseline)





Convergence time

- DCTCP takes at most ~40% more RTTs than TCP
  - “Analysis of DCTCP”, SIGMETRICS 2011
- Intuition: DCTCP makes smaller adjustments than TCP, but makes them much more frequently



CC evaluation: many aspects to consider

- Throughput, delays, flow completion times
- Fairness, convergence times
- Specific impairments:
    - incast (many to one, all to all)
    - Queue buildup
    - Buffer pressure
    - Collateral damage from incast
- Multi-hop versus single-hop bottlenecks
- Comparison against existing TCPs ad AQMs
- How deployable: app awareness, hardware compatibility, …

CC Deployment Concerns

Life isn't easy in the fast lane

Practical deployment concerns in DCs

- Coexistence with legacy protocols like TCP Cubic
  - Application code can't be upgraded in one shot
- Minimum window size matters during heavy incast events
  - e.g., 2 packets versus 1 packet: no reactive scheme can work if buffers are so small that drop in one RTT
- Enabling appropriate options at senders, receivers, and routers
  - Non "ECN-capable" flagged packets will be dropped when Q > K
  - … including the SYN packets of any connection
- Receive-side buffer tuning
  - Reacting to increasing buffer demands at the endpoints takes time
  - Static: Usually, receive buffer must be at least BDP; also influenced significantly by queueing