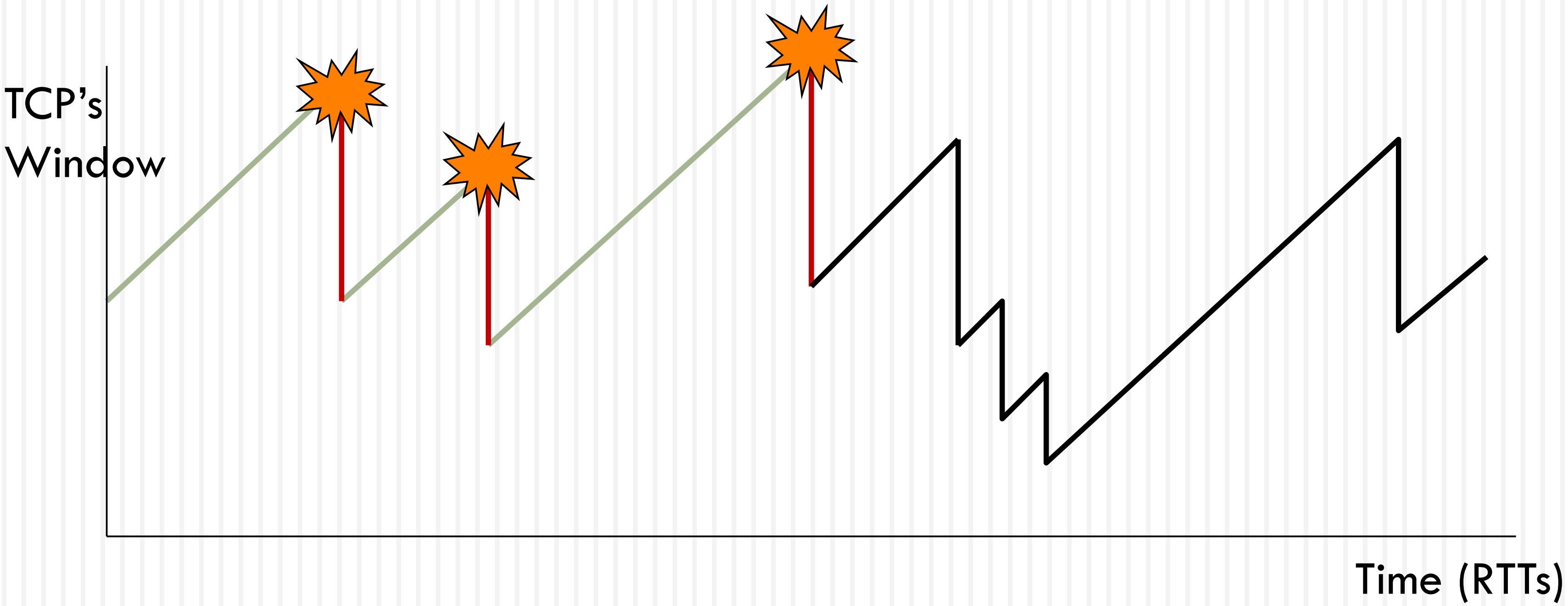
ADVANCED TOPICS FOR CONGESTION CONTROL

Congestion Control

- The Internet only functions because TCP's congestion control does an effective job of matching traffic demand to available capacity.



Limitations of AIMD Congestion Control

(Additive Increase, Multiplicative Decrease)

- Failure to distinguish congestion loss from corruption loss
  - Wireless
- Limited dynamic range

$$\text{transmit rate} \sim= \frac{\text{packet size}}{\text{RTT}\sqrt{\text{loss rate}}}$$

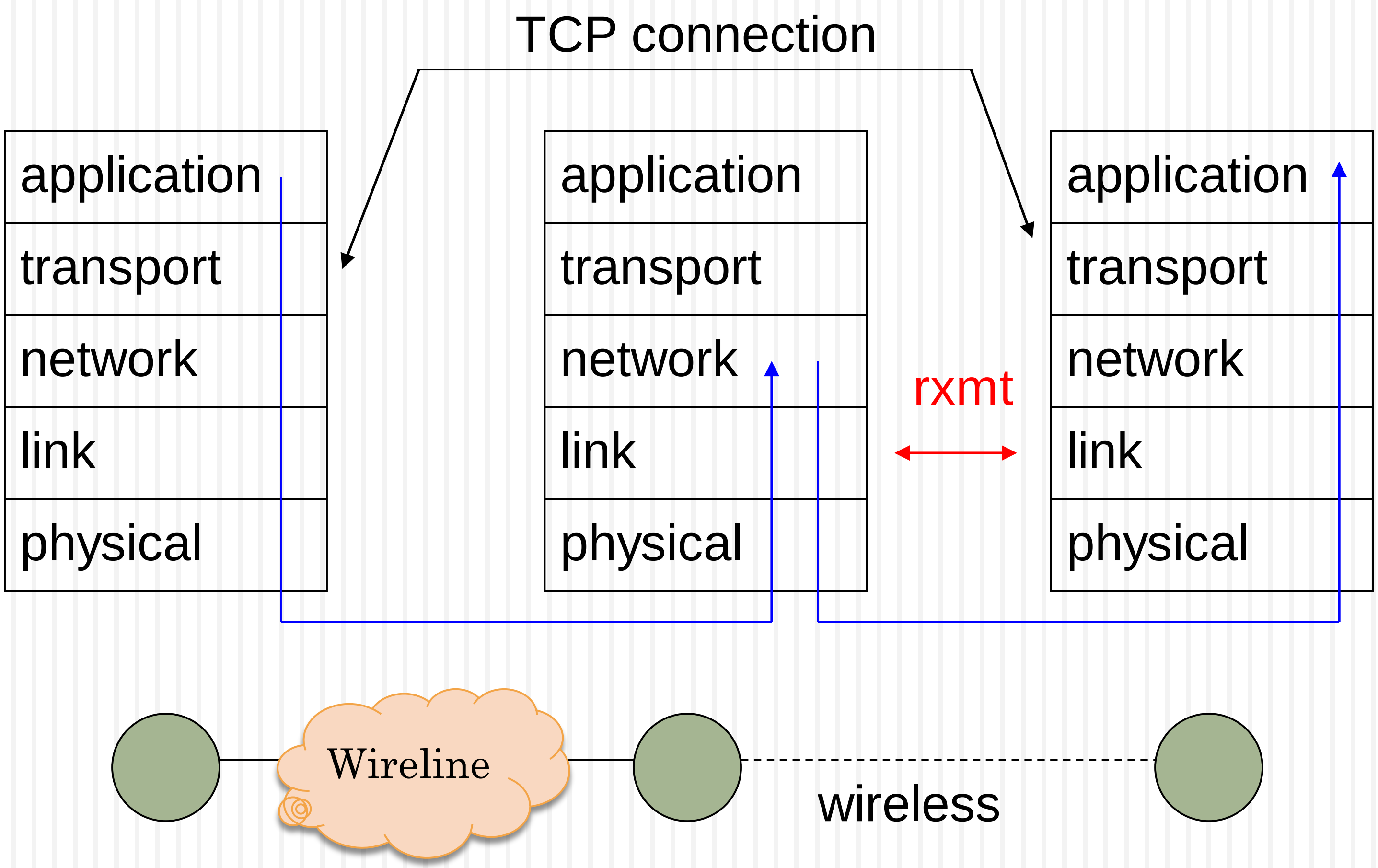
Renewed challenge

- Key assumption in TCP
  - A packet loss is indicative of network congestion
  - Source needs to regulate flow by reducing CW

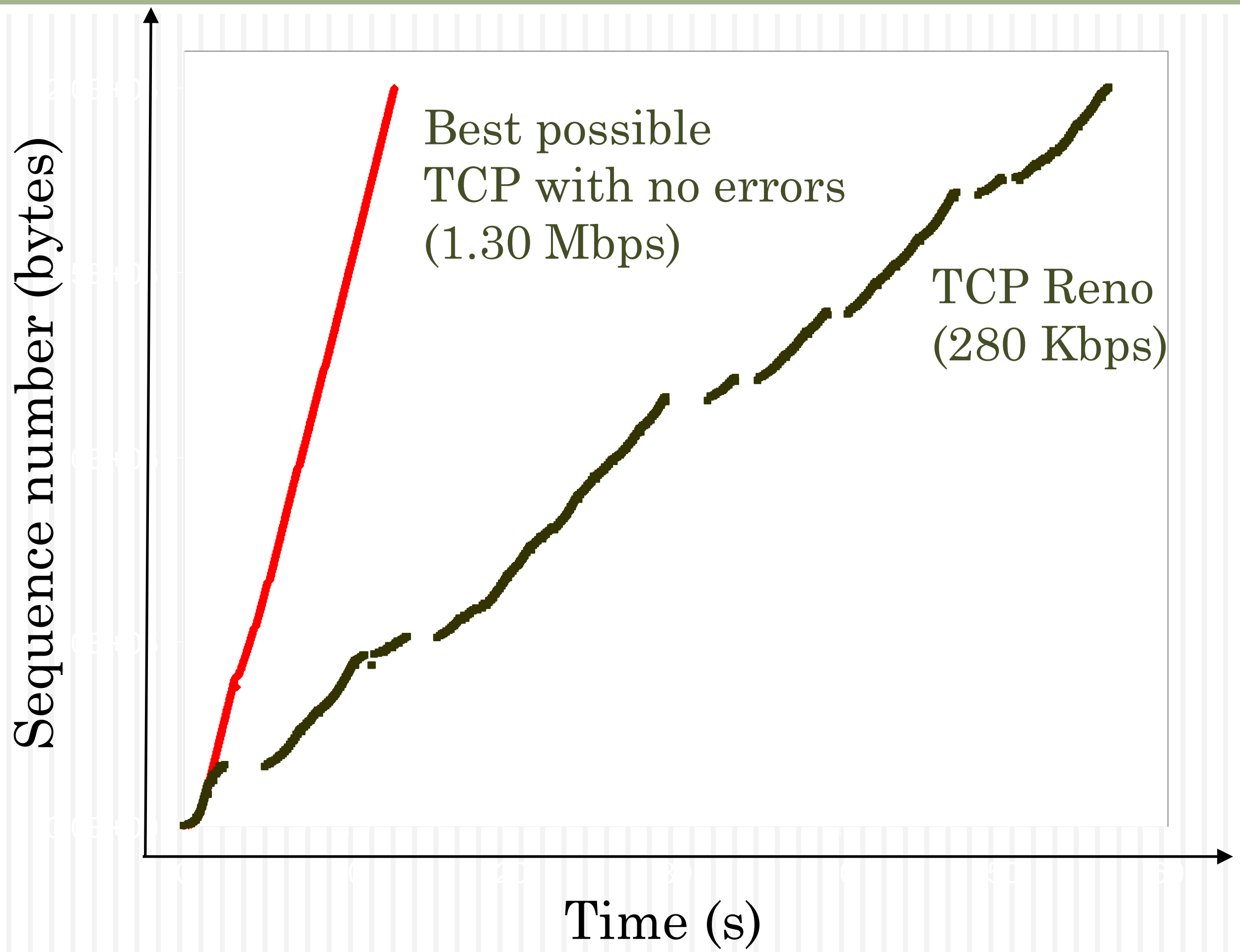
- Assumption closely true for wired networks
  - BER ~ 10^{-6}

- With wireless, errors due to fading, fluctuations
  - Need not reduce CW in response …
  - But, TCP is e2e → CANNOT see the network
  - Thus, TCP cannot classify the cause of loss → CHALLENGE

The problem model



Impact of misclassification



2 MB wide-area TCP transfer over 2 Mbps WaveLAN

The solution space

- Much research on TCP over wireless
- Techniques to Improve TCP Performance in Presence of Errors
  - Classification 1: based on nature of actions taken to improve performance
    - Hide error losses from the sender
      - if sender is unaware of the packet losses due to errors, it will not reduce congestion window
    - Let sender know, or determine, cause of packet loss
      - if sender knows that a packet loss is due to errors, it will not reduce congestion window

The solution space

- Much research on TCP over wireless
- Techniques to Improve TCP Performance in Presence of Errors
  - Classification 2: based on where modifications are needed
    - At the sender node only
    - At the receiver node only
    - At intermediate node(s) only
    - Combinations of the above

The solution space

- Difficult to cover complete ground
- We peek into some of the key ideas
  - Link layer mechanisms
  - Split connection approach
  - TCP-Aware link layer
  - TCP-Unaware approximation of TCP-aware link layer
  - Explicit notification
  - Receiver-based discrimination
  - Sender-based discrimination

Link layer mechanisms

- Forward error corrections (FEC)
  - Add redundancy in the packets to correct bit-errors
  - TCP retransmissions can be alleviated

- Link layer retransmissions
  - MAC layer ACKnowledgments
  - Overhead only when errors occur (unlike FEC)

Such mechanisms require no change in TCP

Issues with link layer mechanisms

- Link layer cannot guarantee reliability
  - Have to drop packets after some finite limit
  - What is the retransmission limit (??)

- Retransmission can take quite long
  - Can be significant fraction of RTT
  - TCP can timeout and retransmit the same packet again
    - Increasing RTO can avoid this
    - But that impacts TCP's recovery from congestion

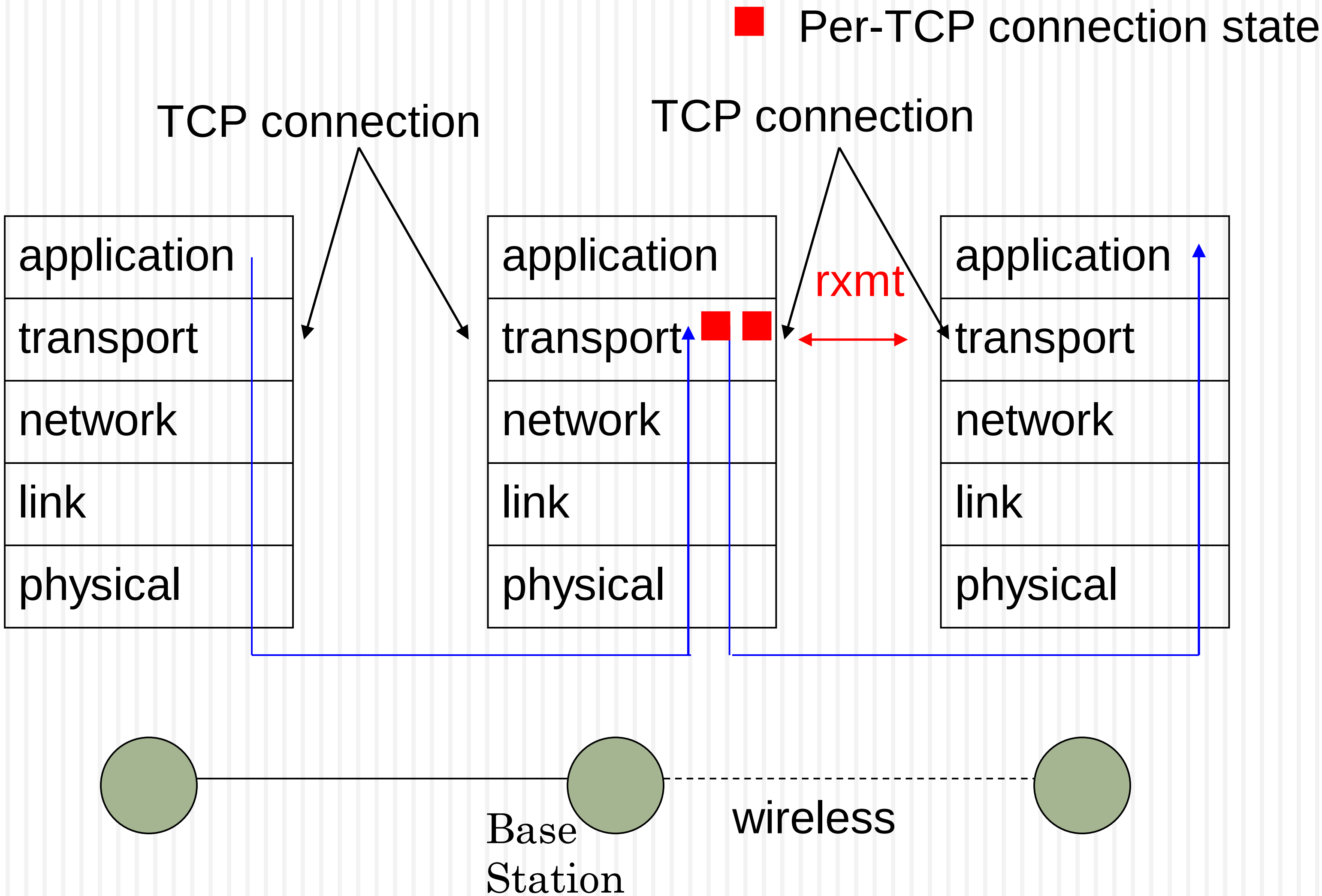
- Head of the line blocking
  - Link layer has to keep retransmitting even if bad channel
  - Blocks other streams

Findings

- Link layer retransmission good
  - When channel errors infrequent
  - When retransmit time << RTO
  - When modifying TCP is not an acceptable solution

Split Connection Approach

1 TCP = ½ TCP + ½ (TCP or XXX)



Splitting approaches

- Indirect TCP [Baker97]
  - Fixed host (FH) to base station (BS) uses TCP
  - BS to mobile host (MH) uses another TCP connection
- Selective Repeat [Yavatkar94]
  - Over FH to BS: Use TCP
  - Over BS to MH: Use selective repeat on top of UDP
- No congestion control over wireless [Haas97]
  - Also use less headers over wireless
    - Header compression

Issues with splitting

- E2E totally broken
  - 2 separate connections
- BS maintains hard state for each connection
  - What if MH disconnected from BS ?
  - Huge buffer requirements at BS
  - What if BS fails ?
  - Handoff between BS requires state transfer
- What if Data and ACK travel on different routes ?
  - BS will not see the ACK at all – splitting not feasible

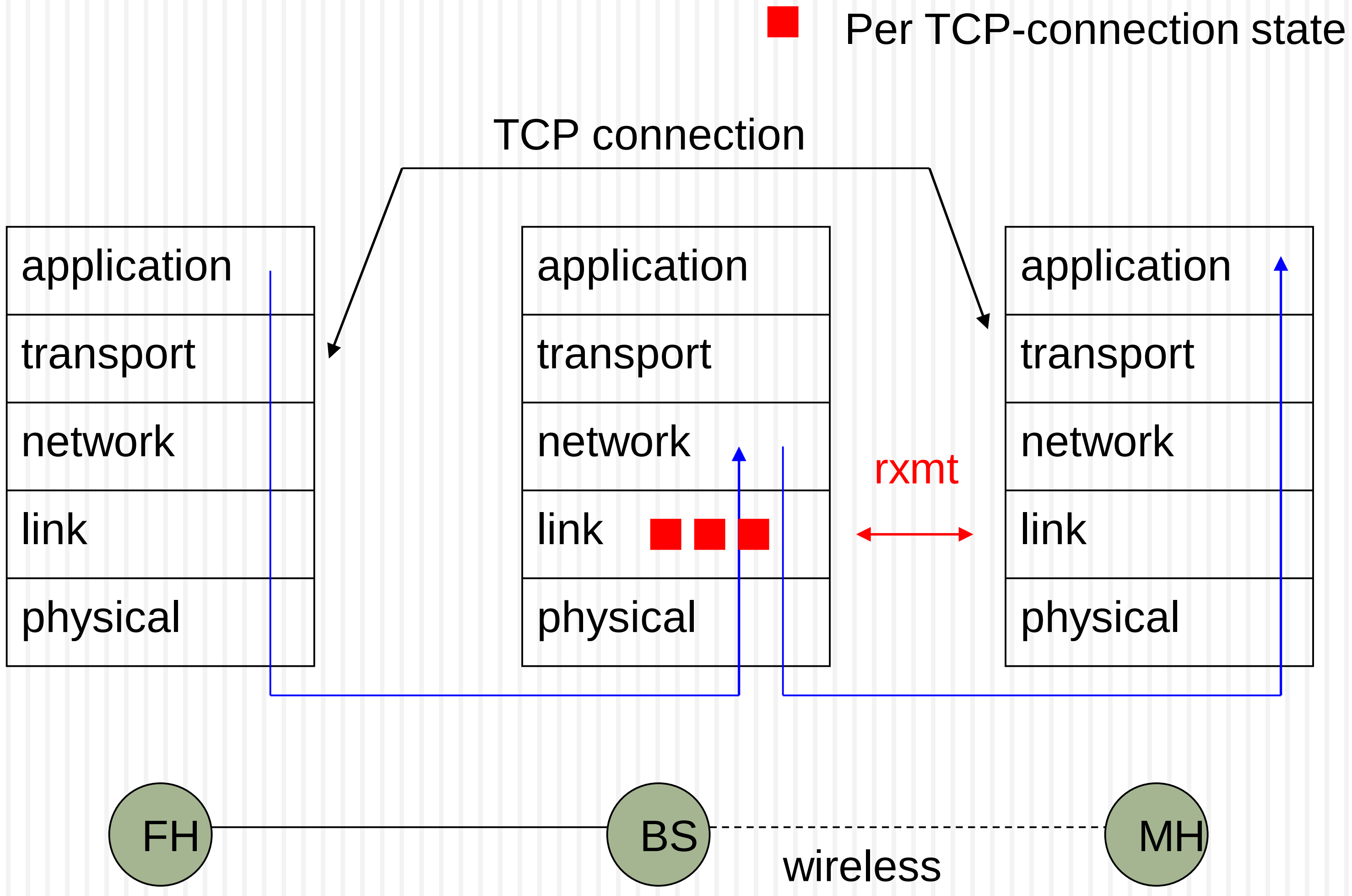
TCP-Aware Link Layer

Snoop Protocol

- Retains local recovery of Split Connection approach and link level retransmission schemes

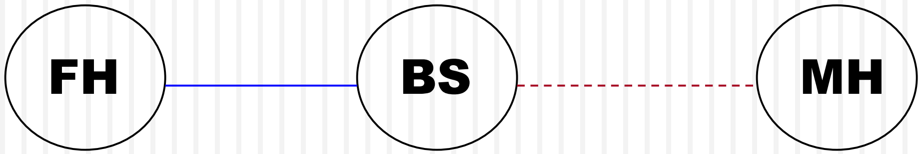
- Improves on split connection
  - end-to-end semantics retained
  - soft state at base station, instead of hard state

Snoop Protocol

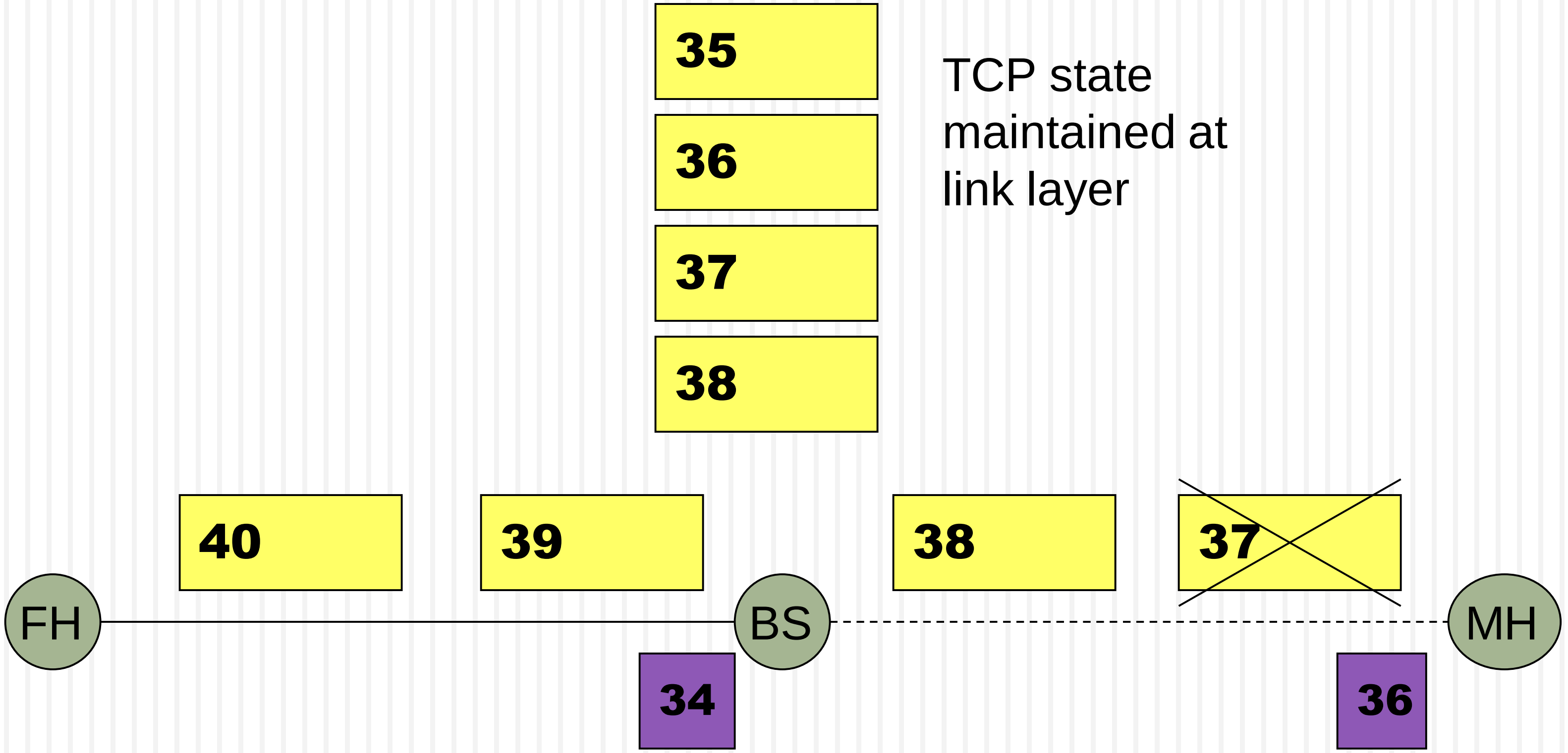


Snoop Protocol

- **Buffers data** packets at the base station BS
  - to allow link layer retransmission
- When dupacks received by BS from MH, **retransmit** on wireless link, if packet present in buffer
- **Prevents fast retransmit** at TCP sender FH by dropping the dupacks at BS

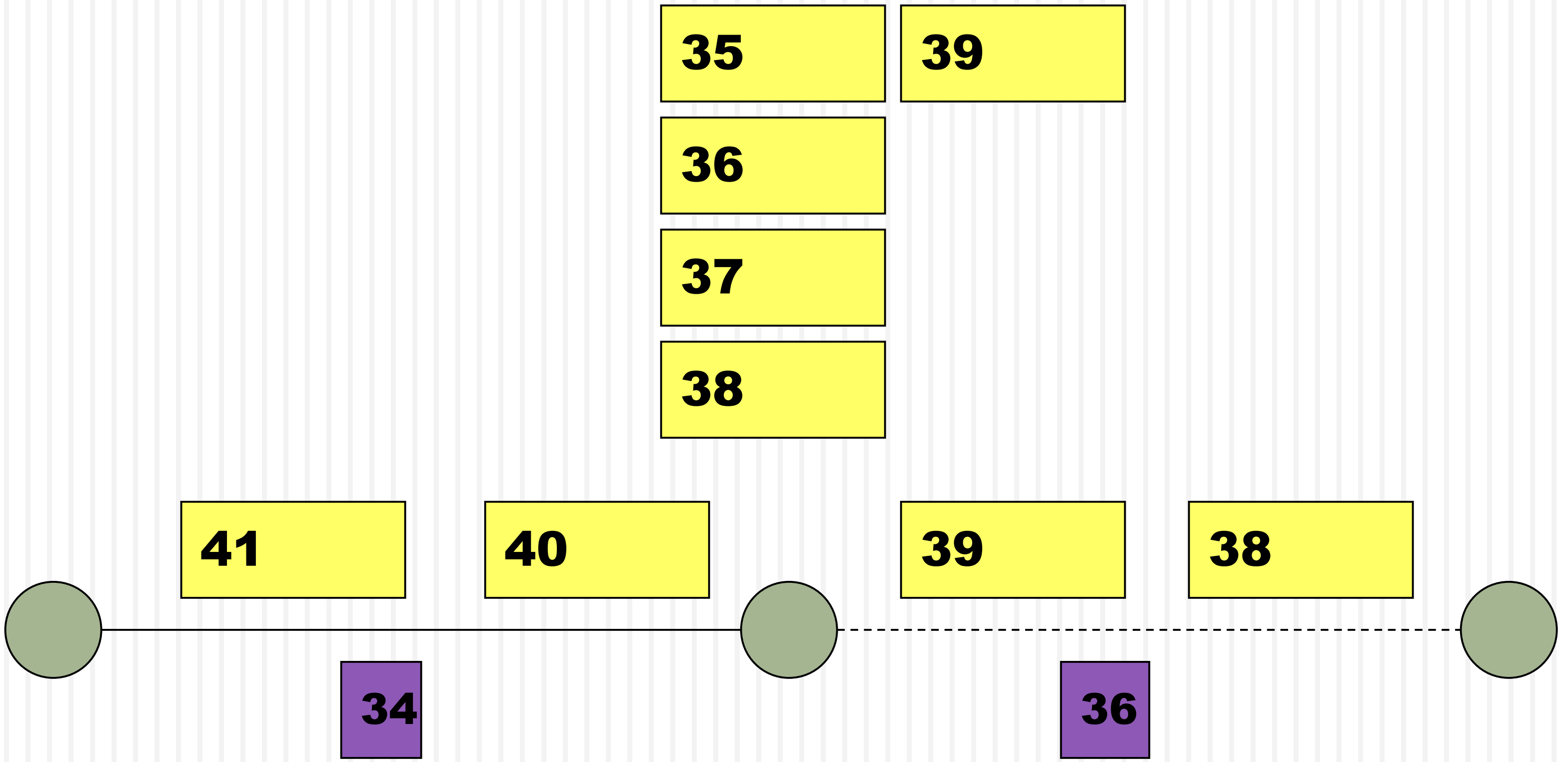


Snoop : Example

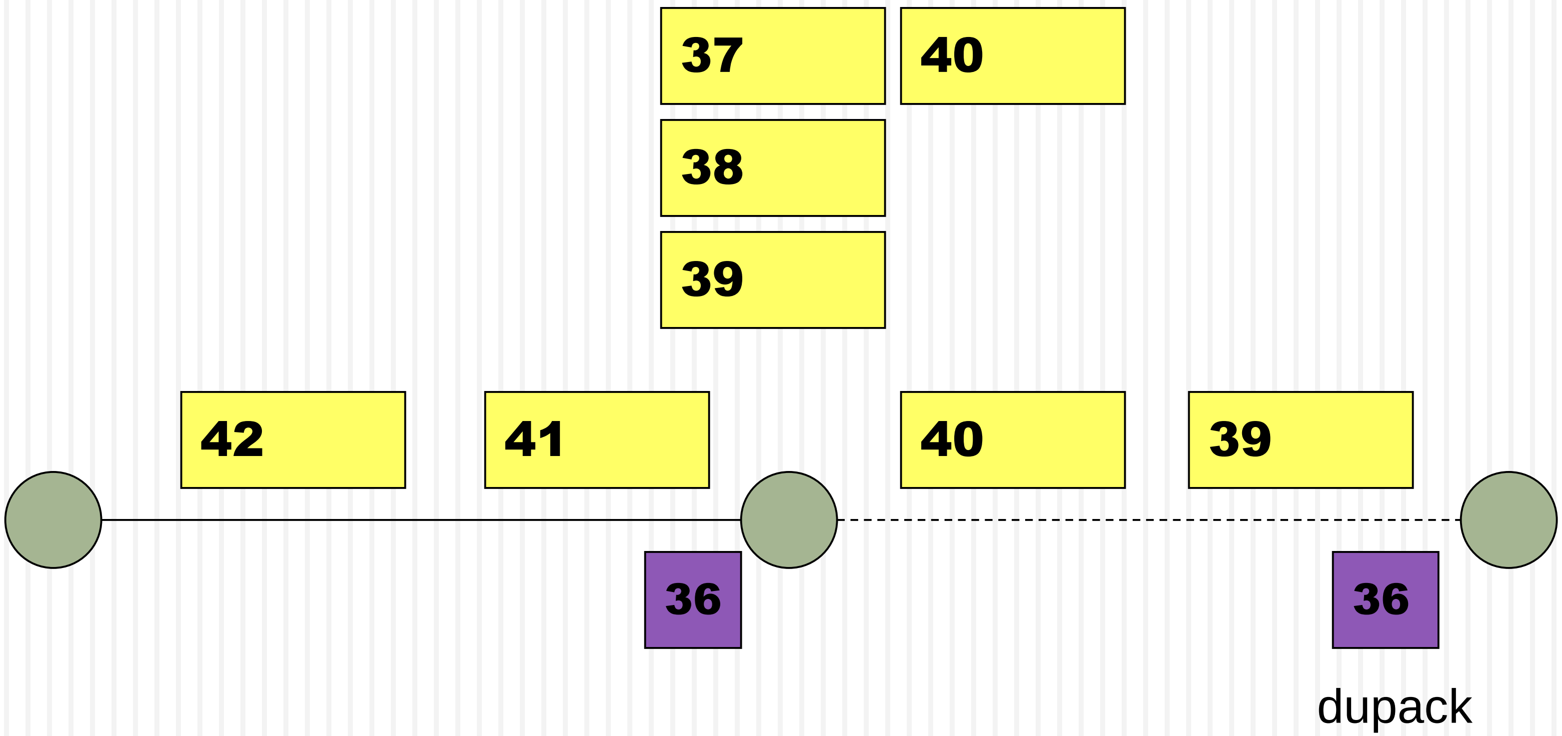


Example assumes delayed ack - every other packet ack'd

Snoop : Example

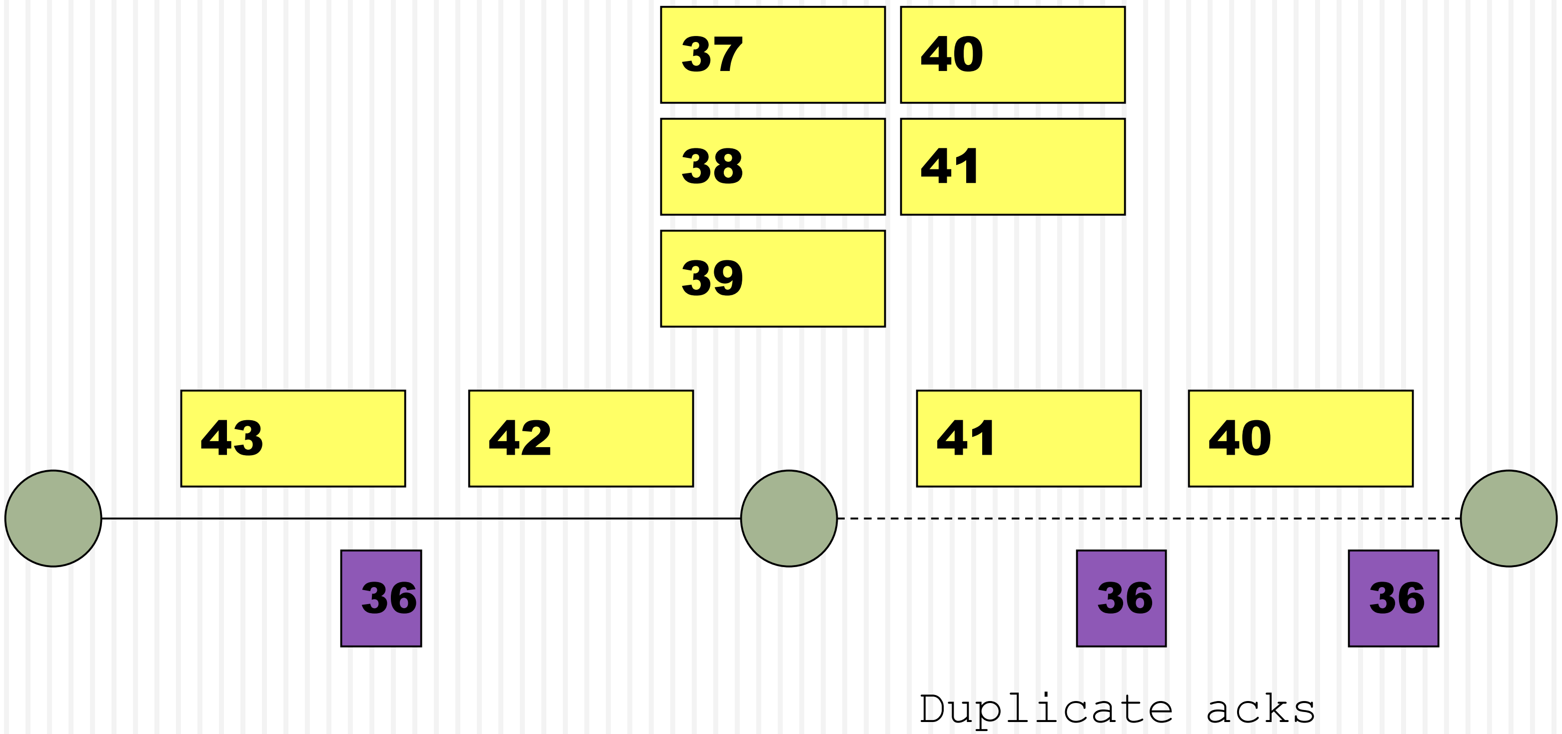


Snoop : Example

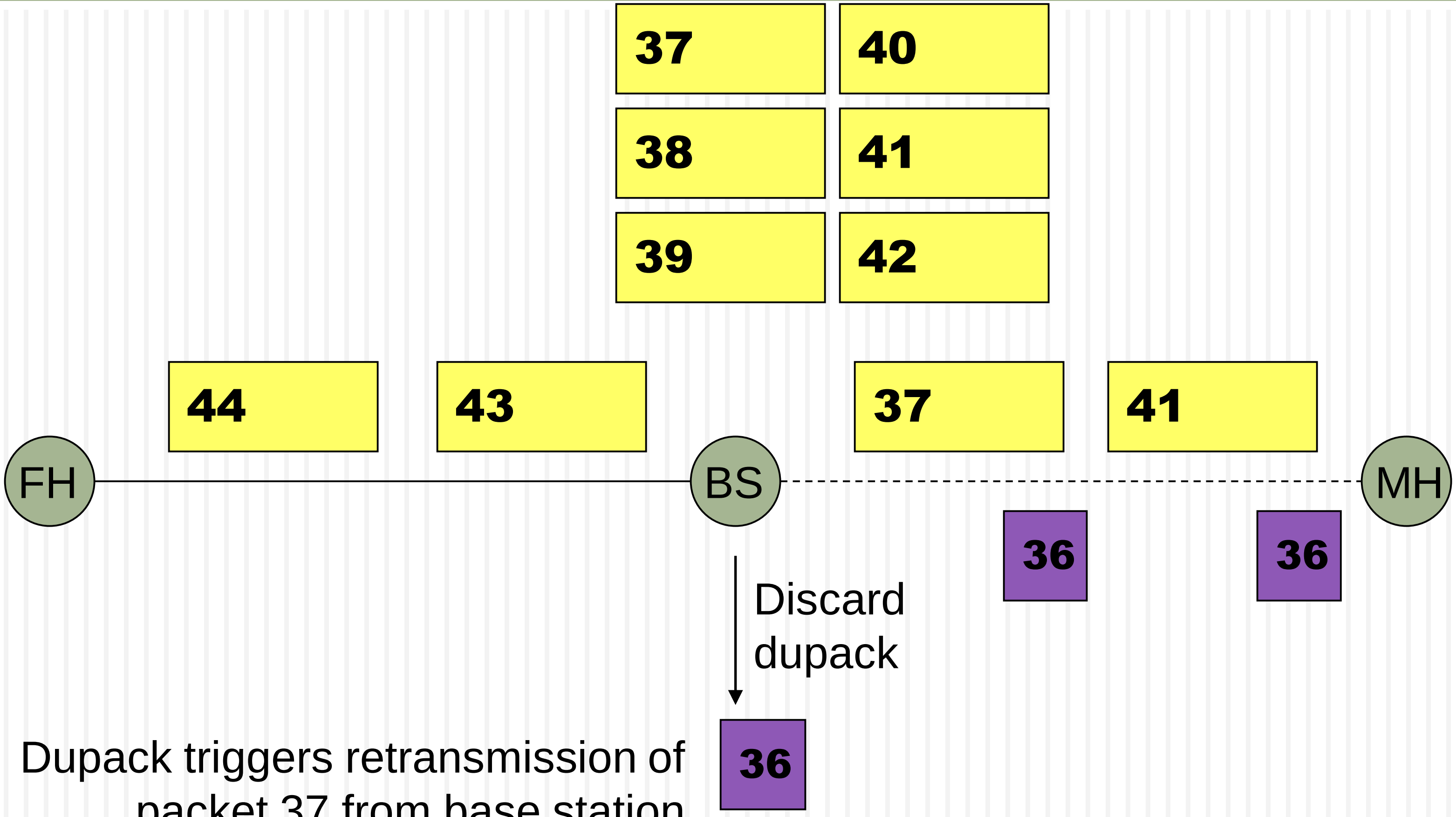


Duplicate acks are not delayed

Snoop : Example



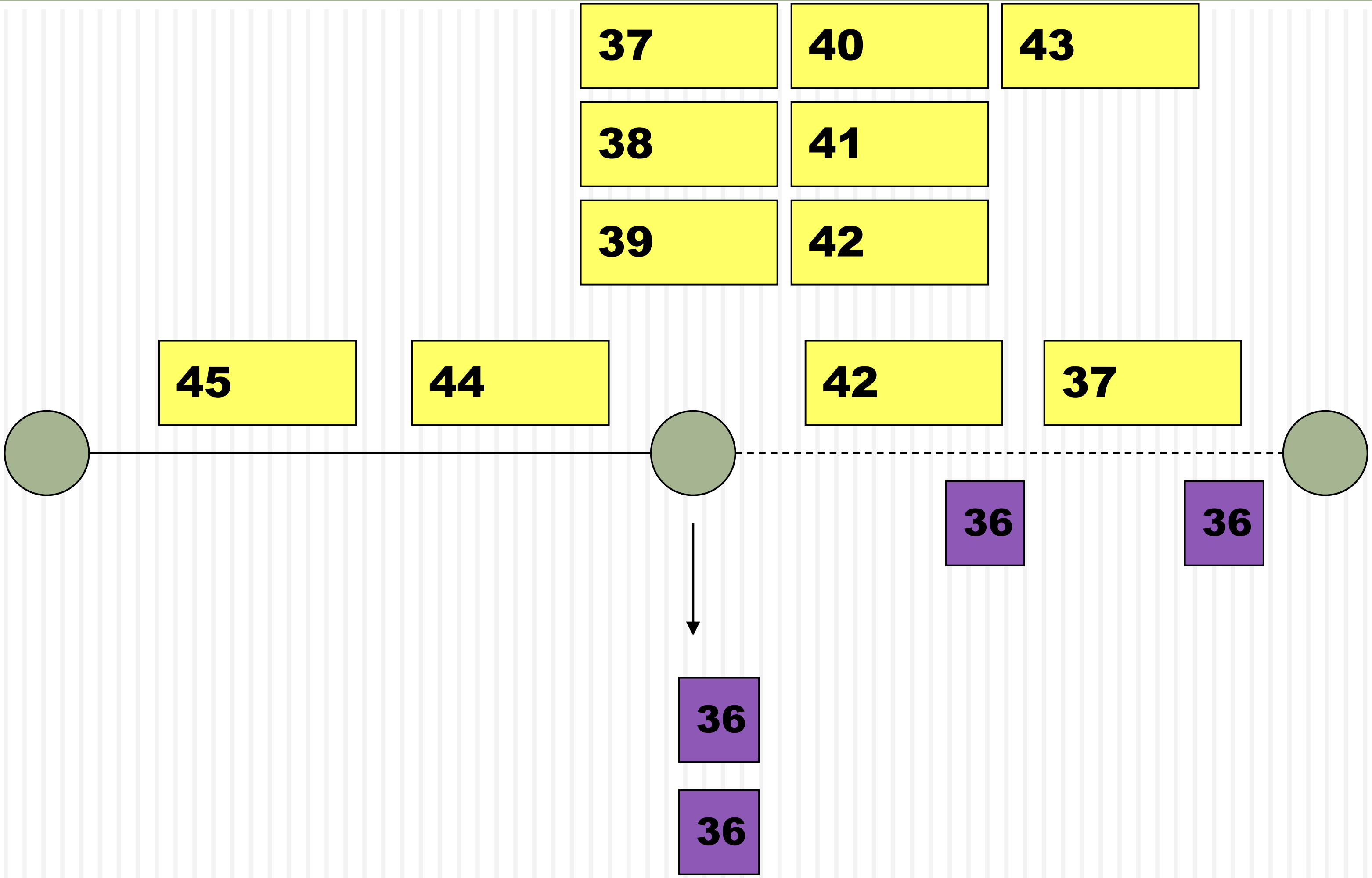
Snoop : Example



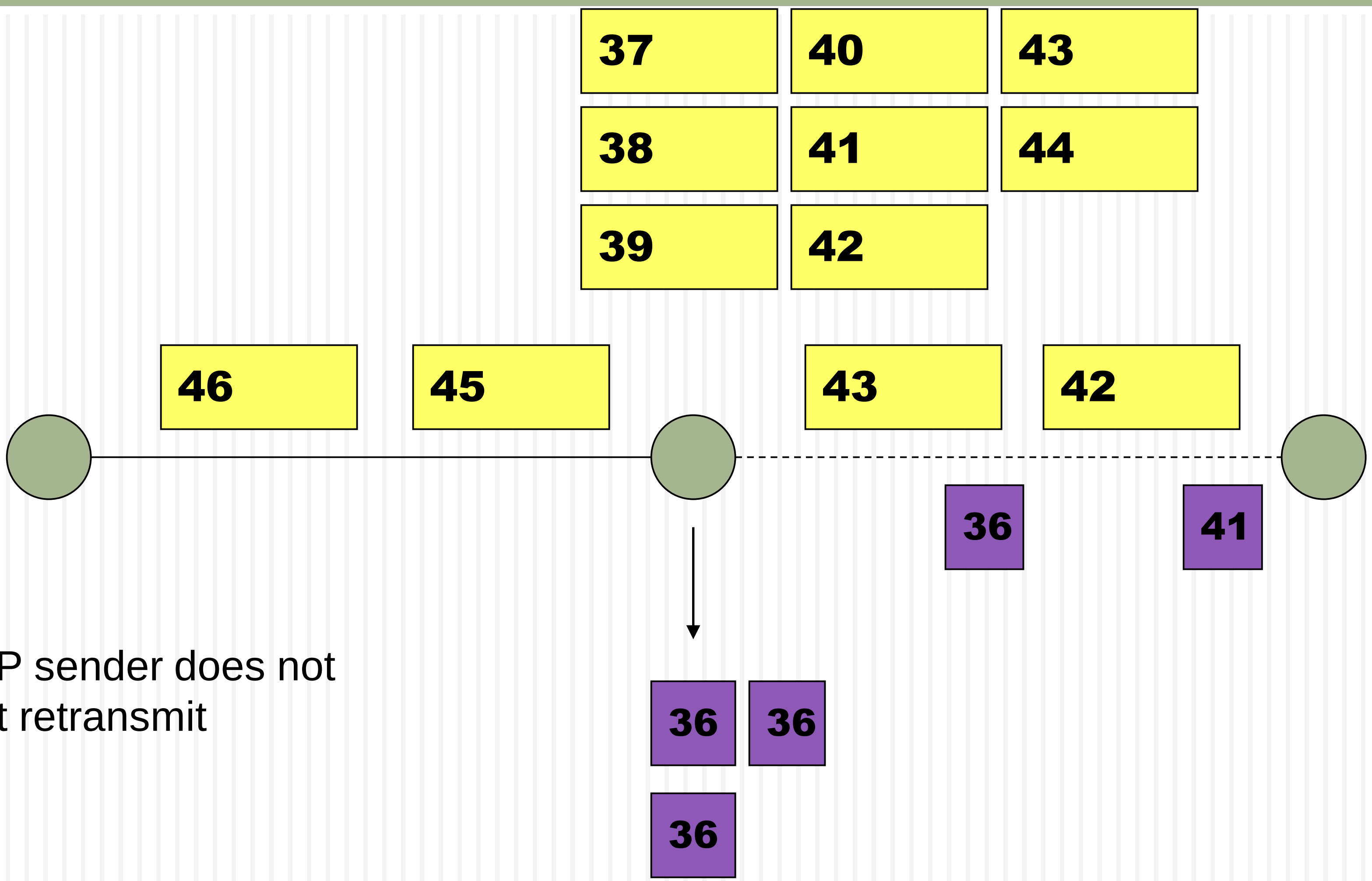
Dupack triggers retransmission of packet 37 from base station

BS needs to be TCP-aware to be able to interpret TCP headers

Snoop : Example

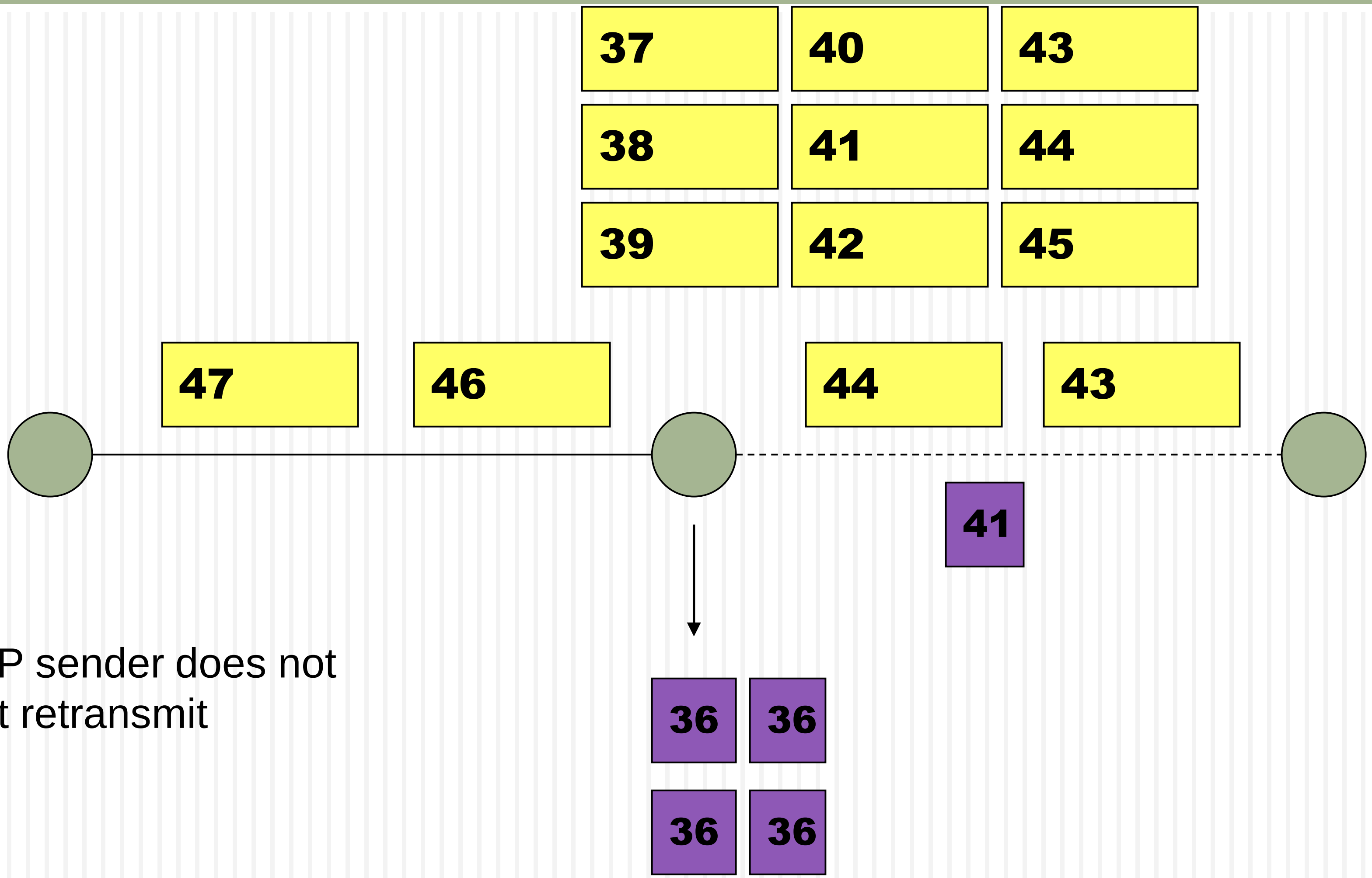


Snoop : Example



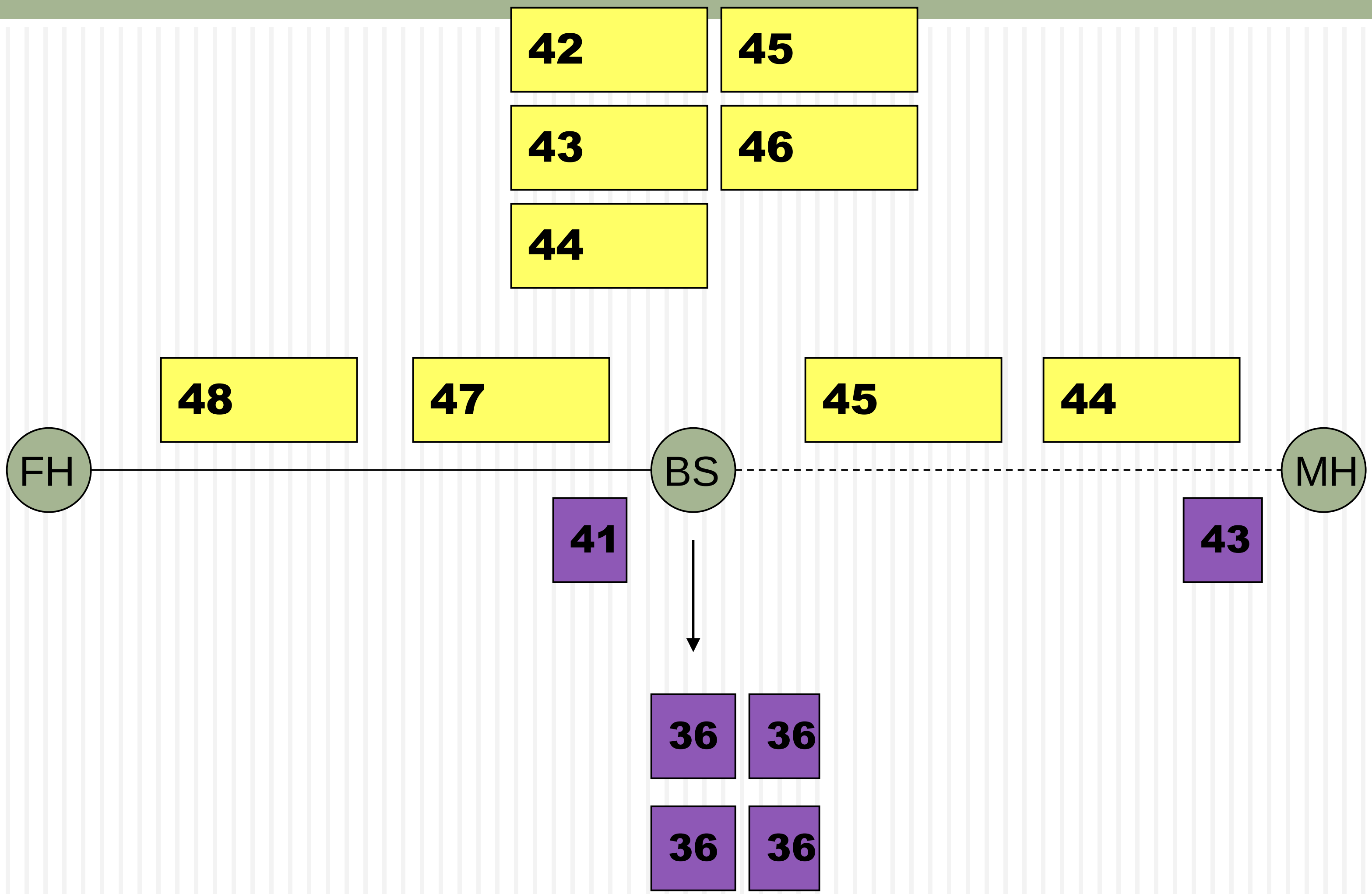
TCP sender does not fast retransmit

Snoop : Example

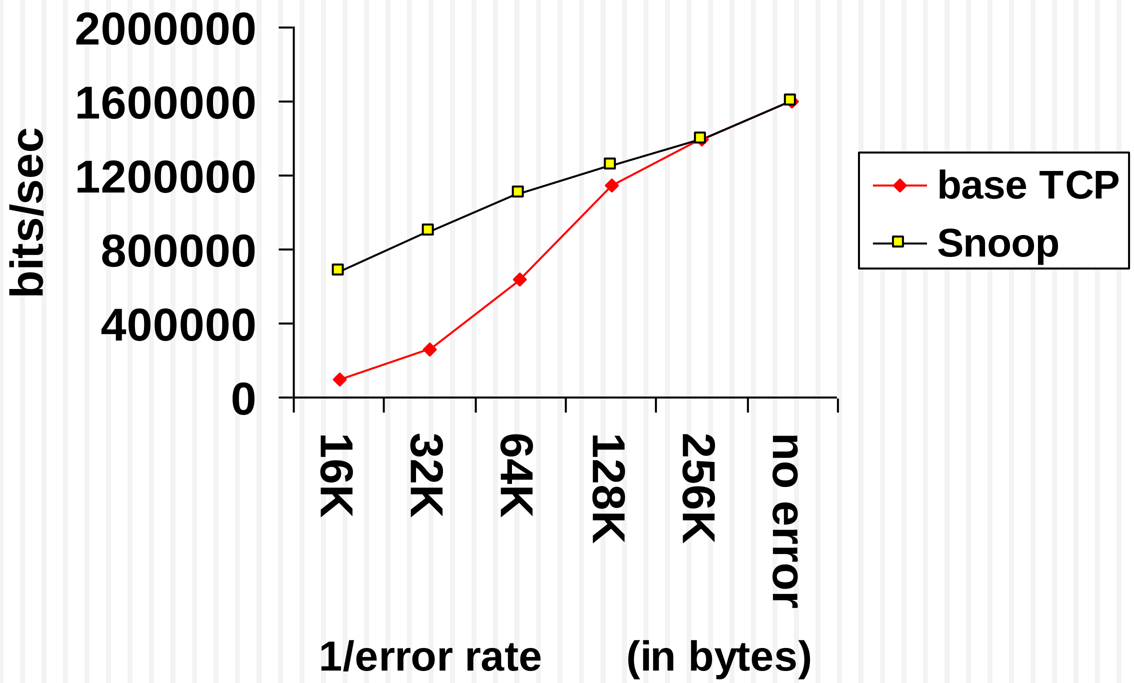


TCP sender does not
fast retransmit

Snoop : Example



Snoop



2 Mbps Wireless link

Snoop Protocol : Classification

- Hides wireless losses from the sender
- Requires modification to only BS (network-centric approach)

Snoop Protocol : Advantages

- High throughput can be achieved
  - performance further improved using selective acks
- Local recovery from wireless losses
- Fast retransmit not triggered at sender despite out-of-order link layer delivery
- End-to-end semantics retained
- Soft state at base station
  - loss of the soft state affects performance, but not correctness

Snoop Protocol : Disadvantages

- Link layer at base station needs to be TCP-aware
- Not useful if TCP headers are encrypted (IPsec)
- Cannot be used if TCP data and TCP acks traverse different paths (both do not go through the base station)

Limitations of AIMD Congestion Control

- Failure to distinguish congestion loss from corruption loss
  - Wireless
- Limited dynamic range

$$\text{transmit rate} \sim= \frac{\text{packet size}}{\text{RTT}\sqrt{\text{loss rate}}}$$

AIMD: Limited Dynamic Range

One loss every half hour, 200ms RTT, 1500bytes/pkt.

⇒ 9000 RTTs increase between losses

⇒ peak window size = 18000 pkts

⇒ mean window size = 12000 pkts

⇒ 18MByte/RTT

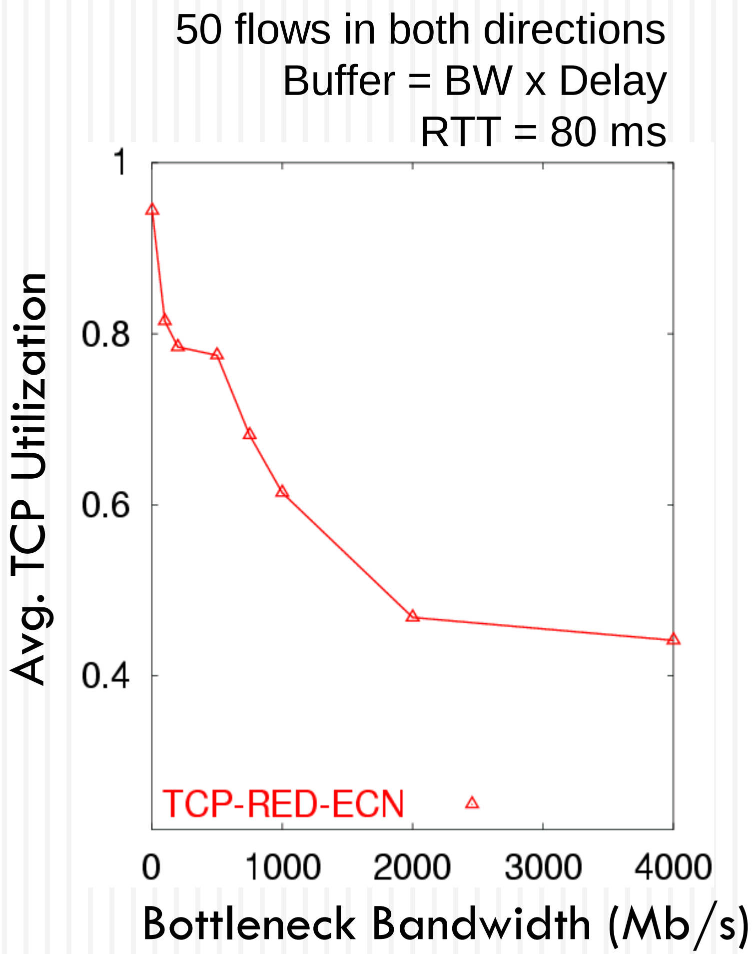
⇒ 720Mbit/s

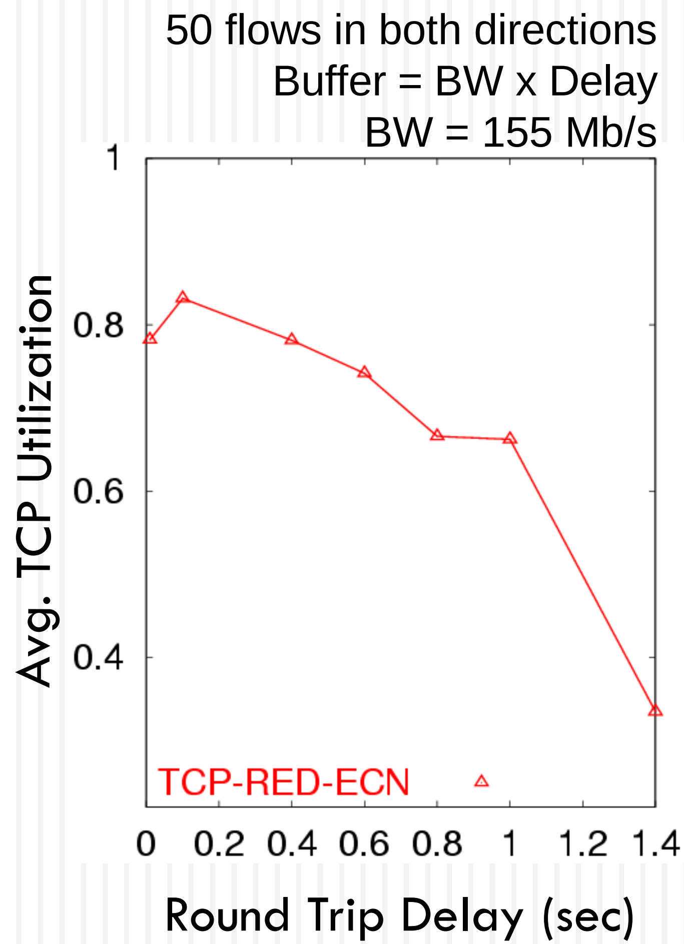
⇒ Needs a bit-error rate of better than 1 in 10^12

⇒ Takes a very long time to converge or recover from a burst of loss

TCP Congestion Control Performs Poorly as Bandwidth or Delay Increases

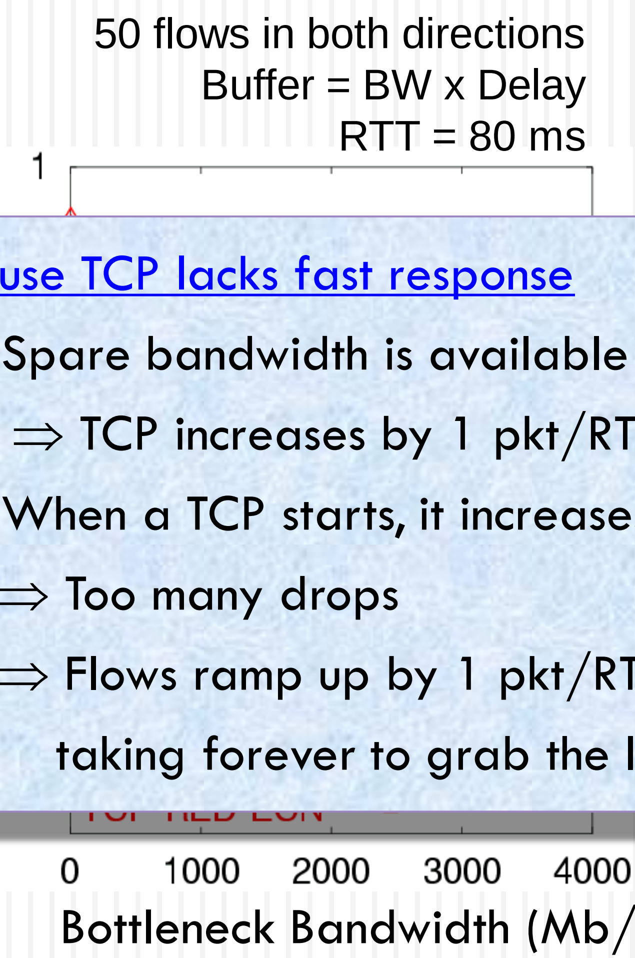
Shown analytically in [Low01] and via simulations

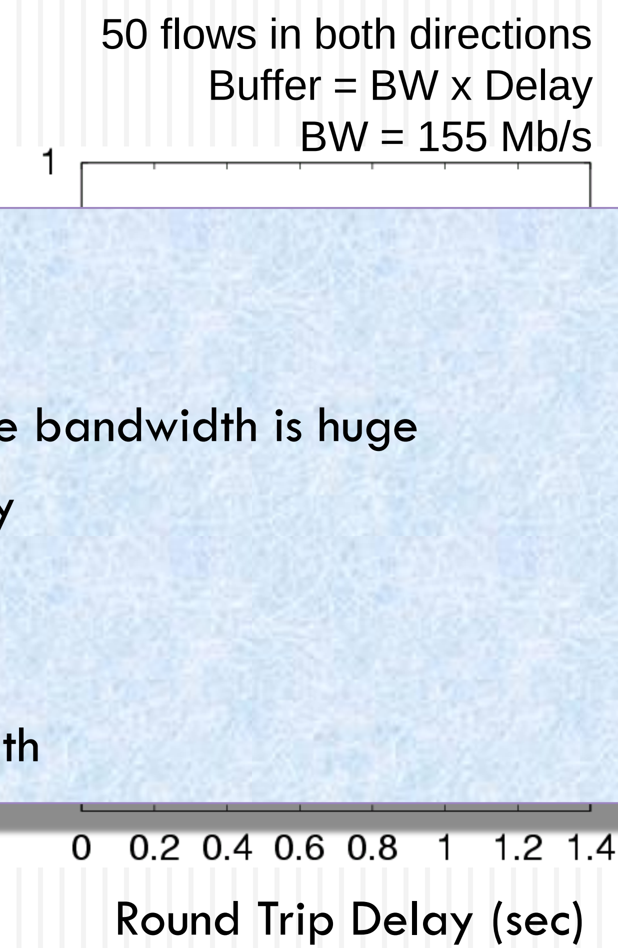




TCP Congestion Control Performs Poorly as Bandwidth or Delay Increases

Shown analytically in [Low01] and via simulations





Because TCP lacks fast response

- Spare bandwidth is available

  ⇒ TCP increases by 1 pkt/RTT even if spare bandwidth is huge

- When a TCP starts, it increases exponentially

  ⇒ Too many drops

  ⇒ Flows ramp up by 1 pkt/RTT,

  taking forever to grab the large bandwidth

Trends in Future Internet

- Links
  - High Bandwidth
    - Gigabit Links – optical fibers
  - High Latency
    - Satellite links
    - Wireless links

The Bandwidth delay products will increase

Efficiency vs. Fairness

- Efficiency
  - Determined by congestion control algorithm
  - Involves only aggregate traffic behavior
  - To maximize it you need
    - High utilization, few drops and small queues
  - Has to be aggressive
- Fairness
  - Relative throughput of all flows in the link
  - Deals with every flow

Proposed Solution:

Decouple Congestion Control from Fairness

High Utilization; Small Queues; Few Drops

Bandwidth Allocation Policy

Proposed Solution:

Decouple Congestion Control from Fairness

Coupled because a *single* mechanism controls both

Example: In TCP, Additive-Increase Multiplicative-Decrease (AIMD) controls both

How does decoupling solve the problem?

1. To control congestion: use MIMD which shows fast response
2. To control fairness: use AIMD which converges to fairness

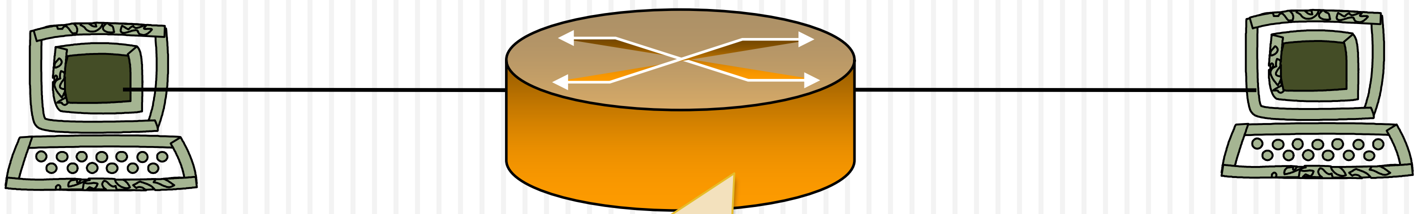
Let's do it all over again

- Build new congestion control architecture
  - Design goal: Stable + efficient + fair
- Ideas
  - Packet loss is a poor signal of congestion
    - Congestion is not a binary variable
    - We want precise feedback
  - Efficiency independent of number of flows
  - Decouple congestion and fairness control

Design Ideas (cont.)

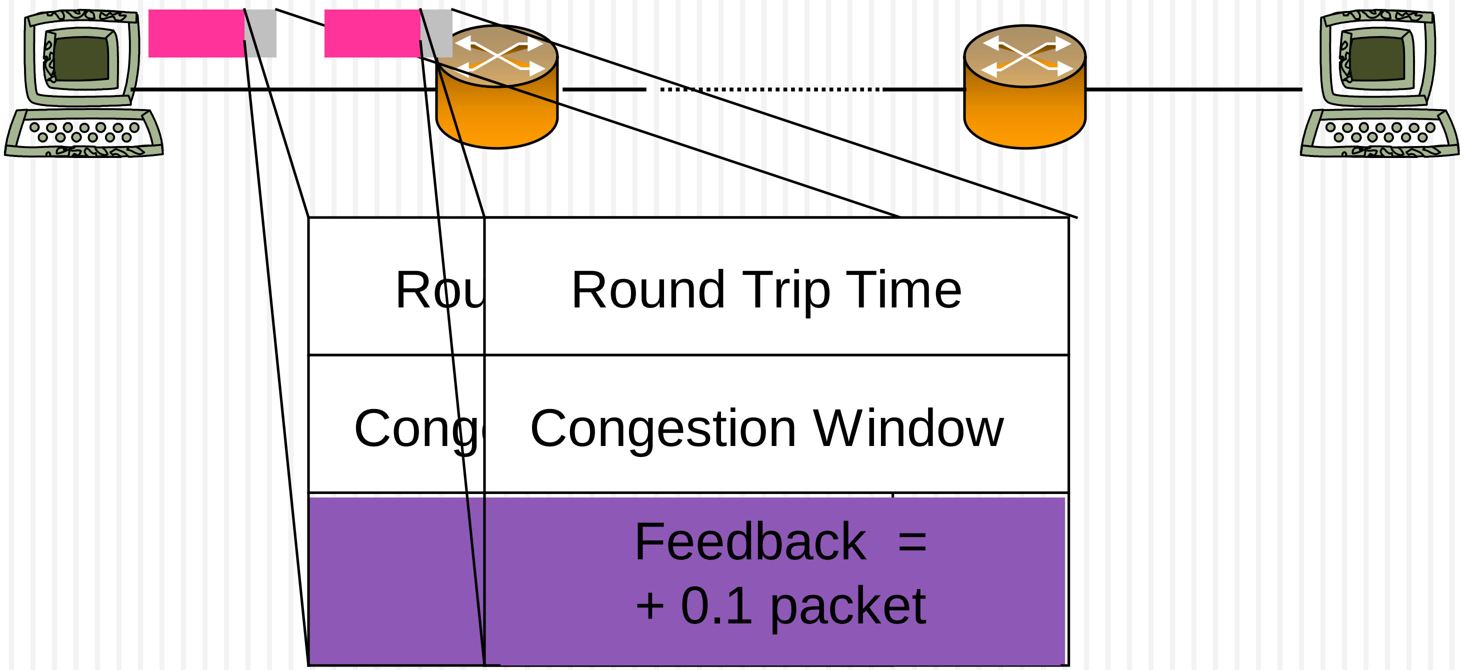
- Make the network intelligent
  - Routers give explicit congestion feedback
- Controlling aggressiveness of source
  - As delay increases rate change should be slower
- Explicit congestion notification (ECN)
  - IP extension providing advance congestion notification
- Core-stateless fair queuing (CSFQ)
  - Edge routers estimate incoming flow rates
  - Use these rates to label packets

XCP: An eXplicit Control Protocol



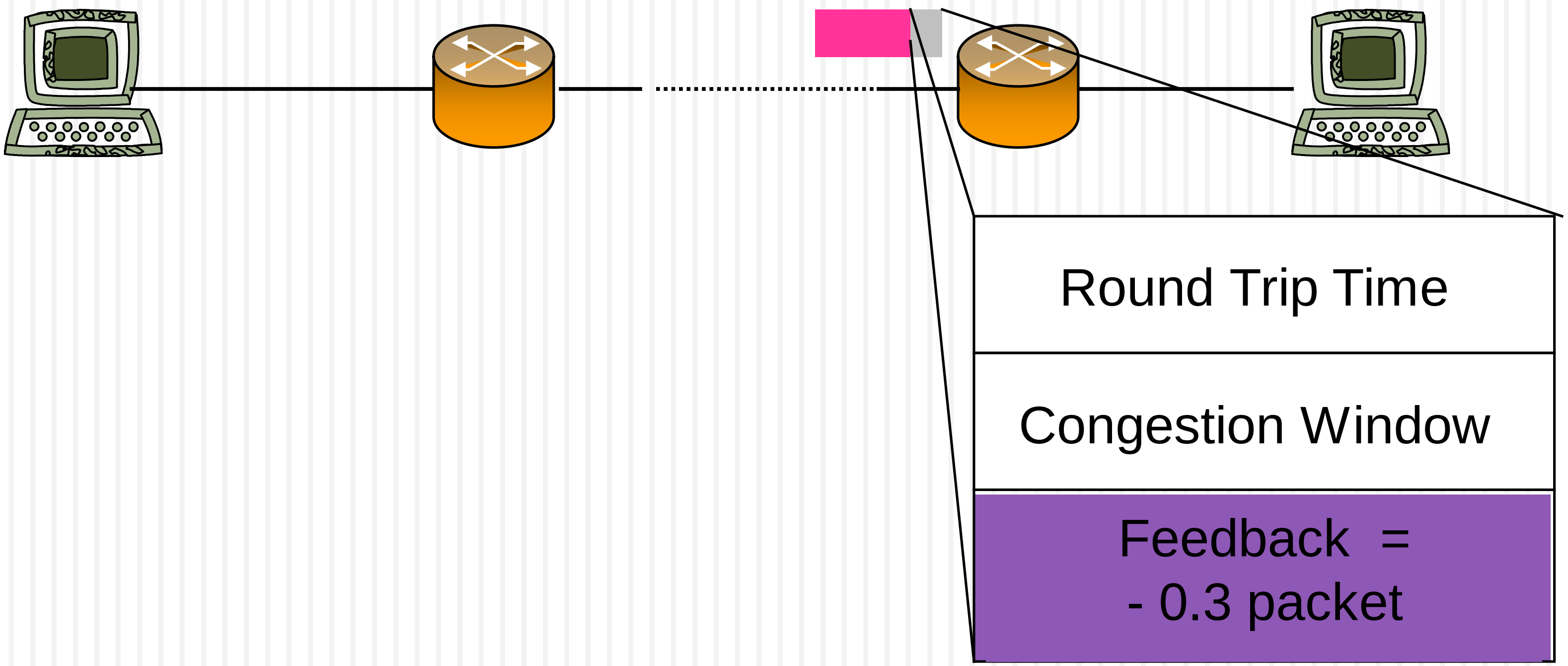
1. Congestion Controller
2. Fairness Controller

How does XCP Work?

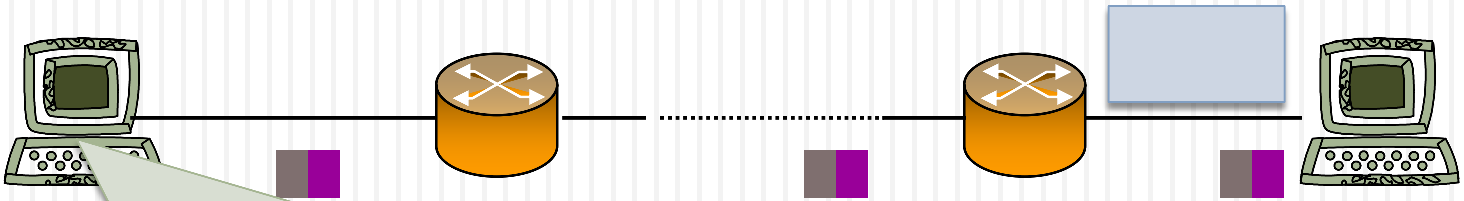


Congestion Header

How does XCP Work?



How does XCP Work?



Congestion Window = Congestion Window + Feedback

XCP extends ECN and CSFQ (Core-Stateless Fair Queueing)

Routers compute feedback without any per-flow state

XCP Header

H_cwnd (set to sender's current cwnd)
H_rtt (set to sender's rtt estimate)
H_feedback (initialized to demands)

- H_cwnd – sender's current cong. Window
- H_rtt – sender's current RTT estimate
- H_feedback – Initialized by sender but modified by routers along path to **directly control the congestion windows**

The Players- XCP Sender

Initialization steps:

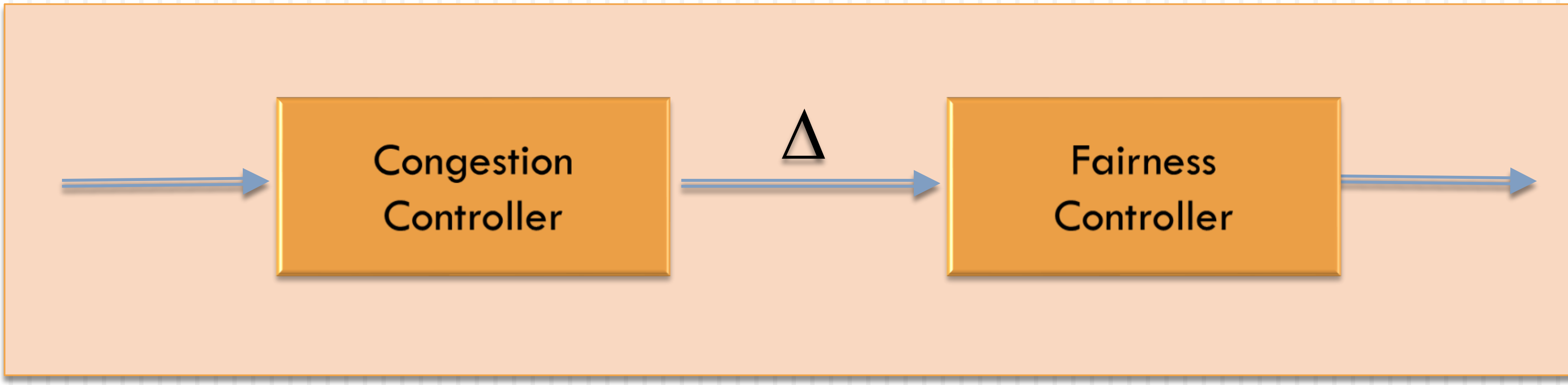
- In first packet of flow, H_rtt is set to zero
- H_feedback is set to the desired window increase
  - E.g. For desired rate r:
    - H_feedback = (r * rtt – cwnd) / # packets in window
- When Acks arrive:
  - $Cwnd = max(cwnd + H\_feedback, s)$

    s => packet size

The Players- XCP Receiver

- When sending the ack to sender it copies the congestion header onto the packet
- No other difference than TCP

The Players – XCP Router



- Computes the feedback for the host
- Makes decision every average RTT
- Operates on top of other dropping policy
- Efficiency controller and fairness controller

How Does an XCP Router Compute the Feedback?

Congestion Controller

<u>Goal:</u> Matches input traffic to link capacity & drains the queue

Looks at aggregate traffic & queue

MIMD

<u>Algorithm:</u>

Aggregate traffic changes by Δ

Δ ~ Spare Bandwidth (diff between the input traffic rate and link capacity)

Δ ~ - Queue Size

So, $\Delta = \alpha \, d_{avg} \, Spare - \beta \, Queue$

feedback in byte

Average RTT

Fairness Controller

<u>Goal:</u> Divides Δ between flows to converge to fairness

Looks at a flow's state in Congestion Header

AIMD

<u>Algorithm:</u>

If $\Delta > 0 \Rightarrow$ Divide Δ equally between flows

If $\Delta < 0 \Rightarrow$ Divide Δ between flows proportionally to their current rates

Implementation

Implementation uses few multiplications & additions per packet → Practical!

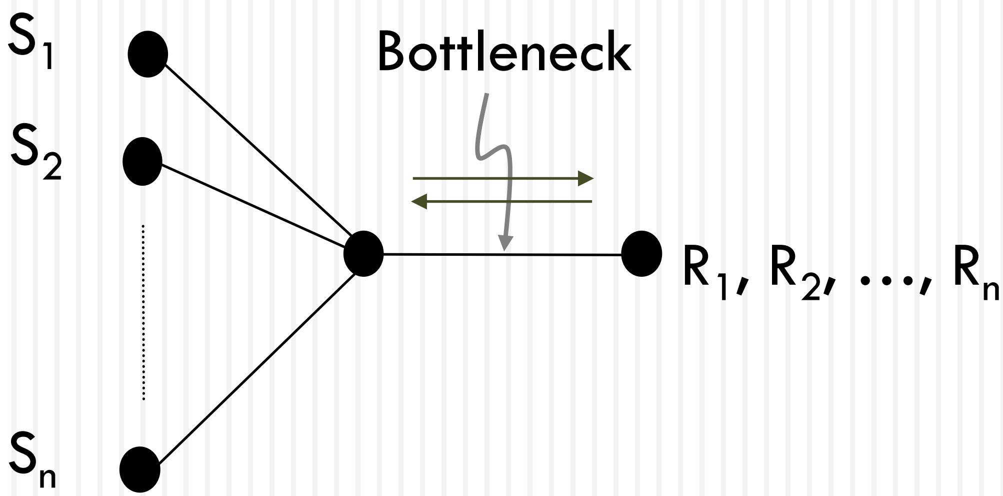
<u>Liars?</u>

- Policing agents at edges of the network or statistical monitoring
- Easier to detect than in TCP

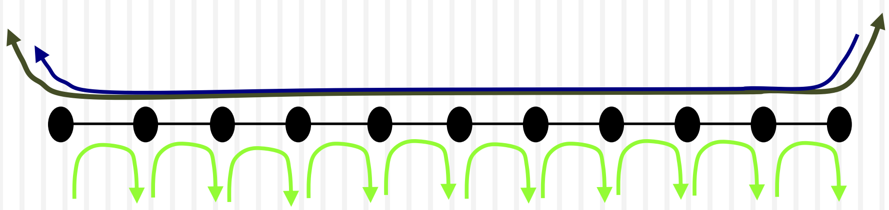
<u>Gradual Deployment</u>

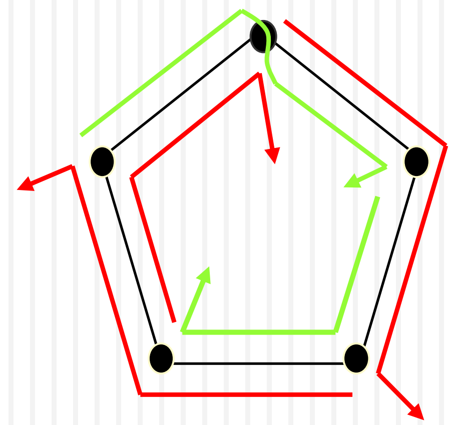
XCP can co-exist with TCP and can be deployed gradually

Performance: Subset of Results



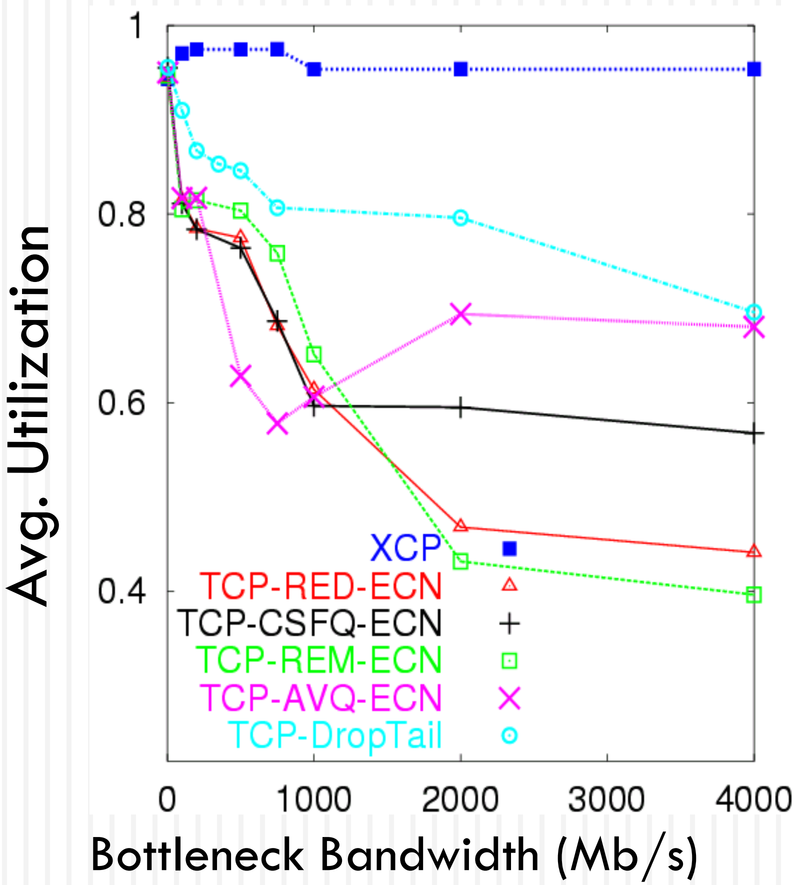
Similar behavior over:



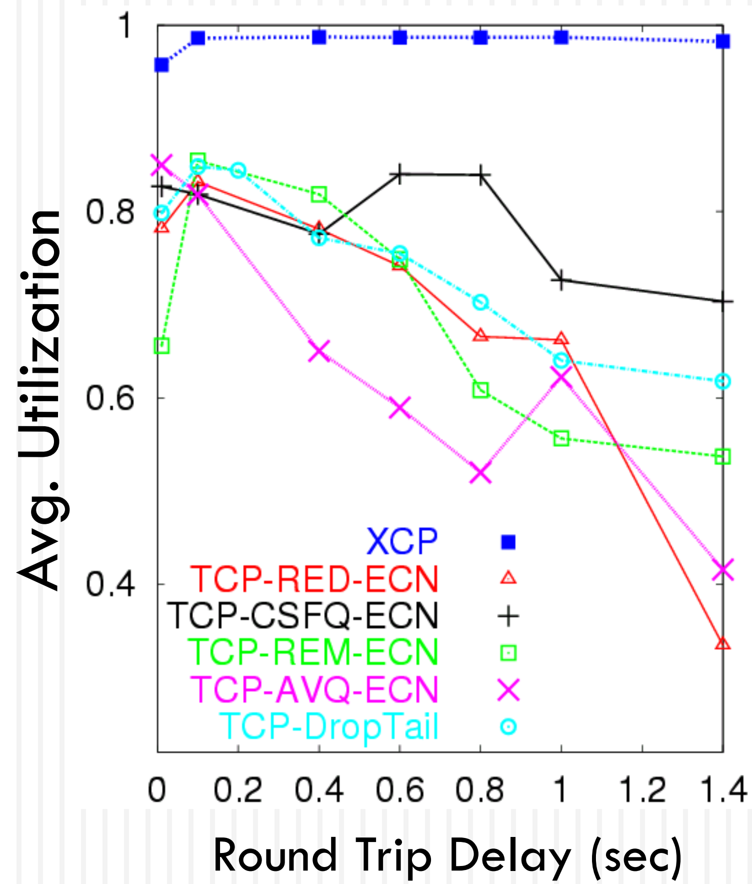


XCP Remains Efficient as Bandwidth or Delay Increases

Utilization as a function of Bandwidth

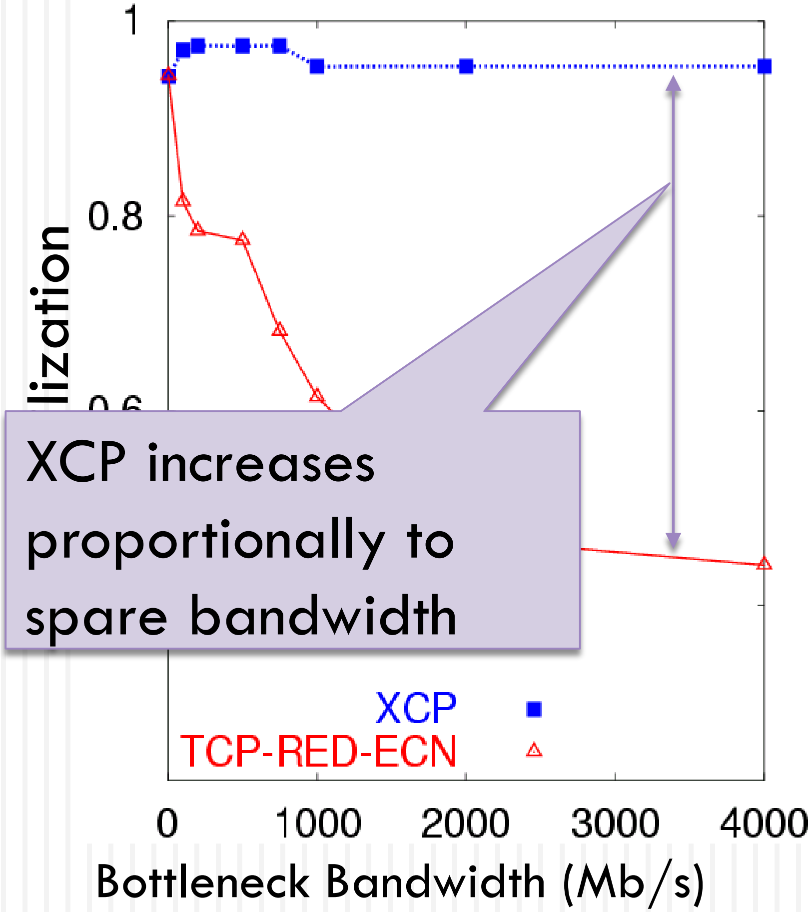


Utilization as a function of Delay

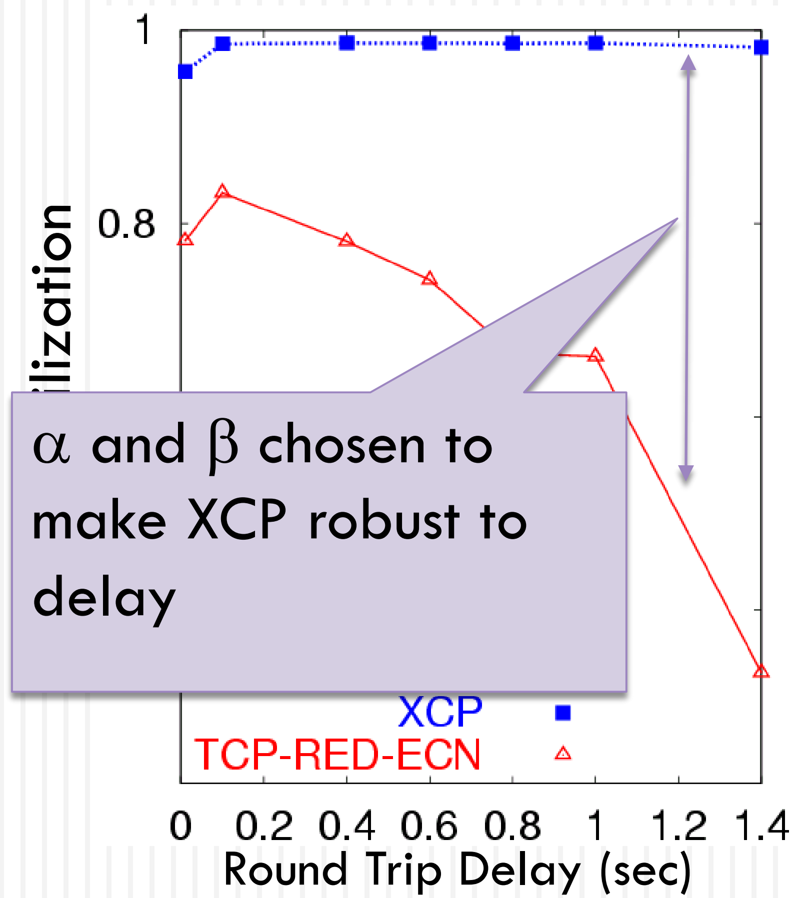


XCP Remains Efficient as Bandwidth or Delay Increases

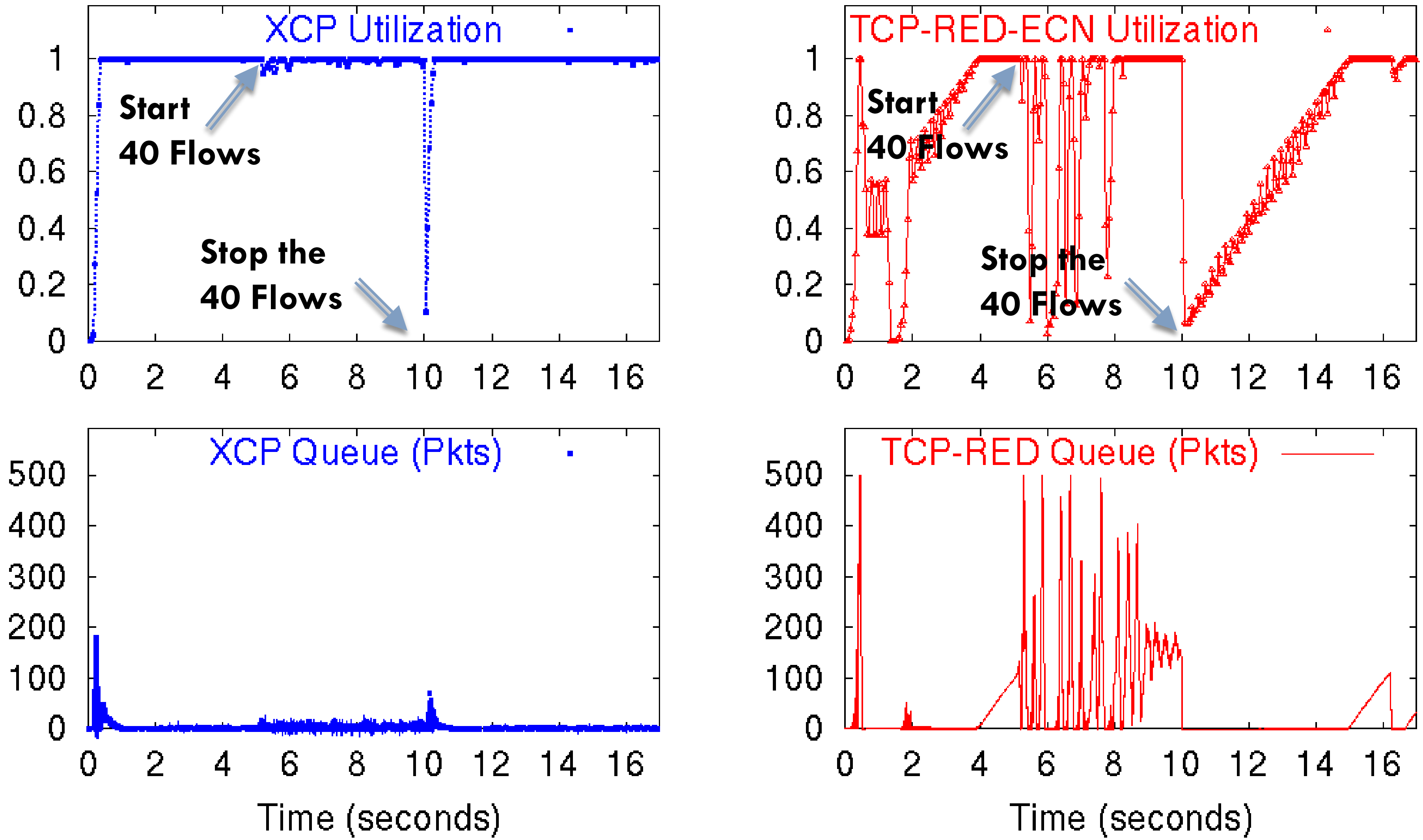
Utilization as a function of Bandwidth



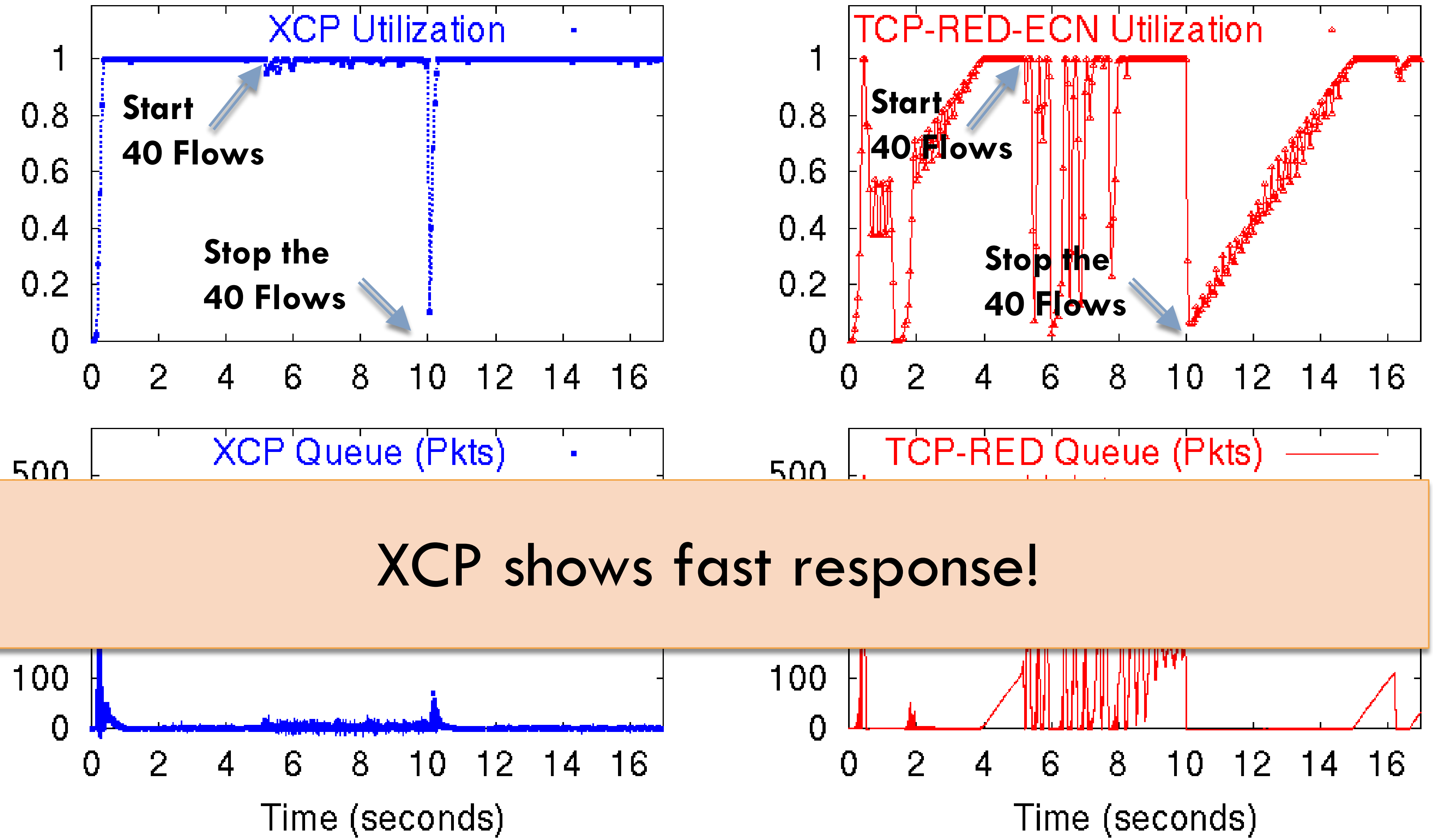
Utilization as a function of Delay



XCP Shows Faster Response than TCP



XCP Shows Faster Response than TCP



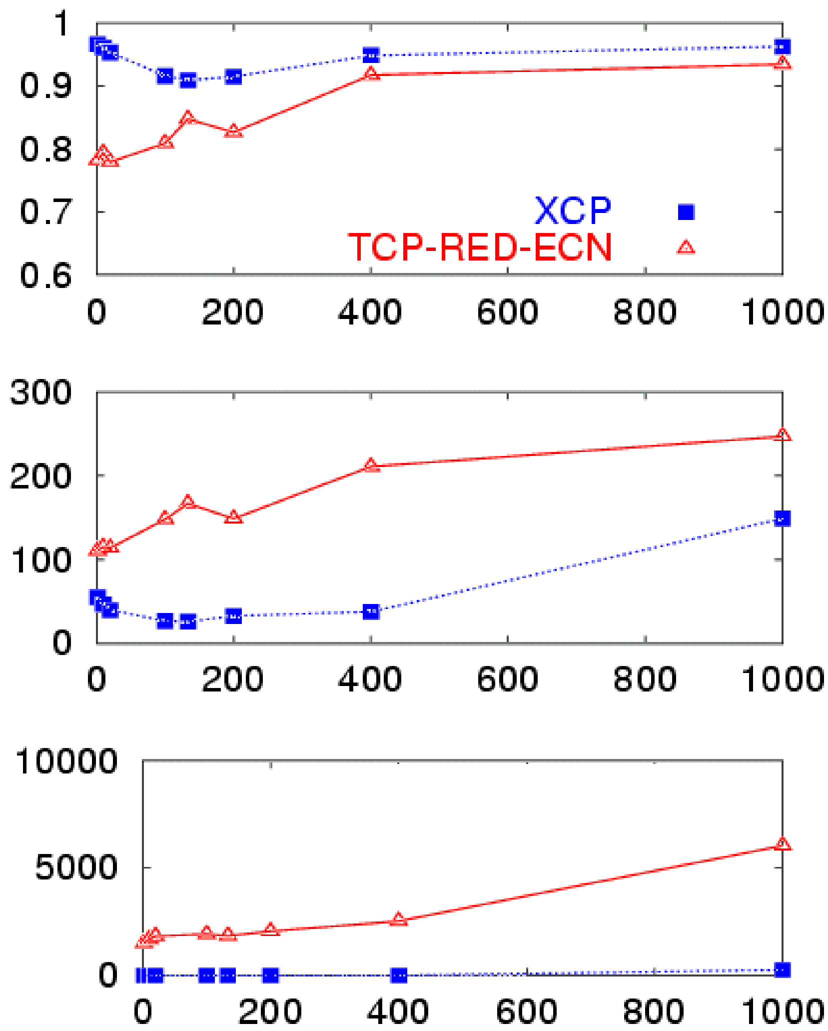
XCP shows fast response!

XCP Deals Well with Short Web-Like Flows

Average Utilization

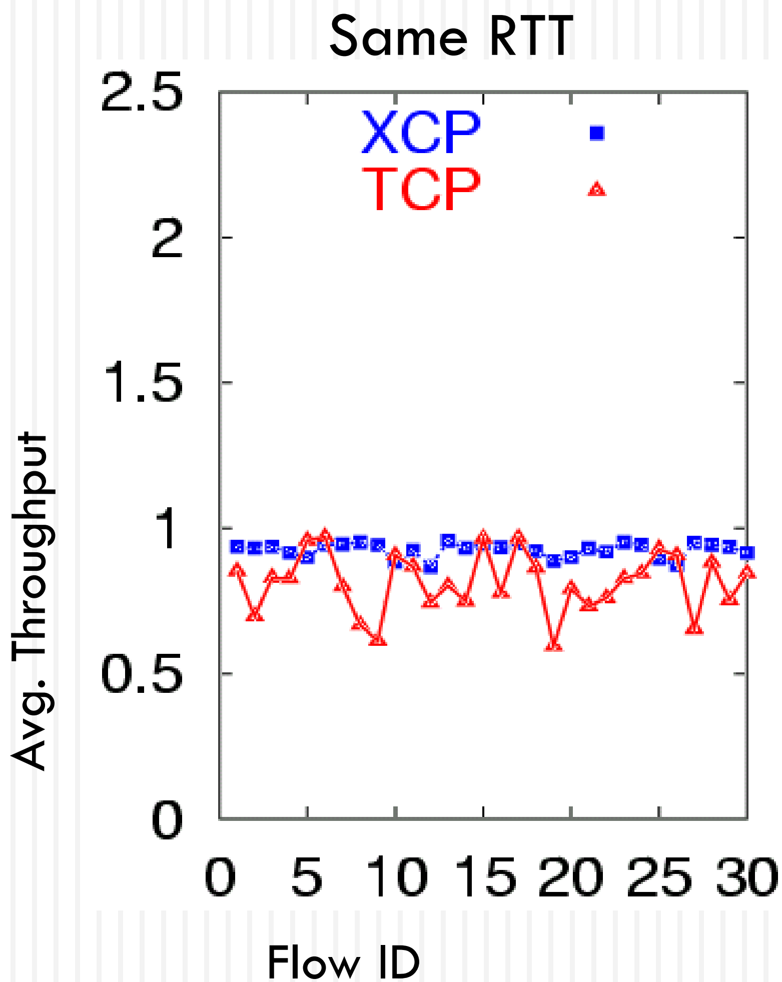
Average Queue

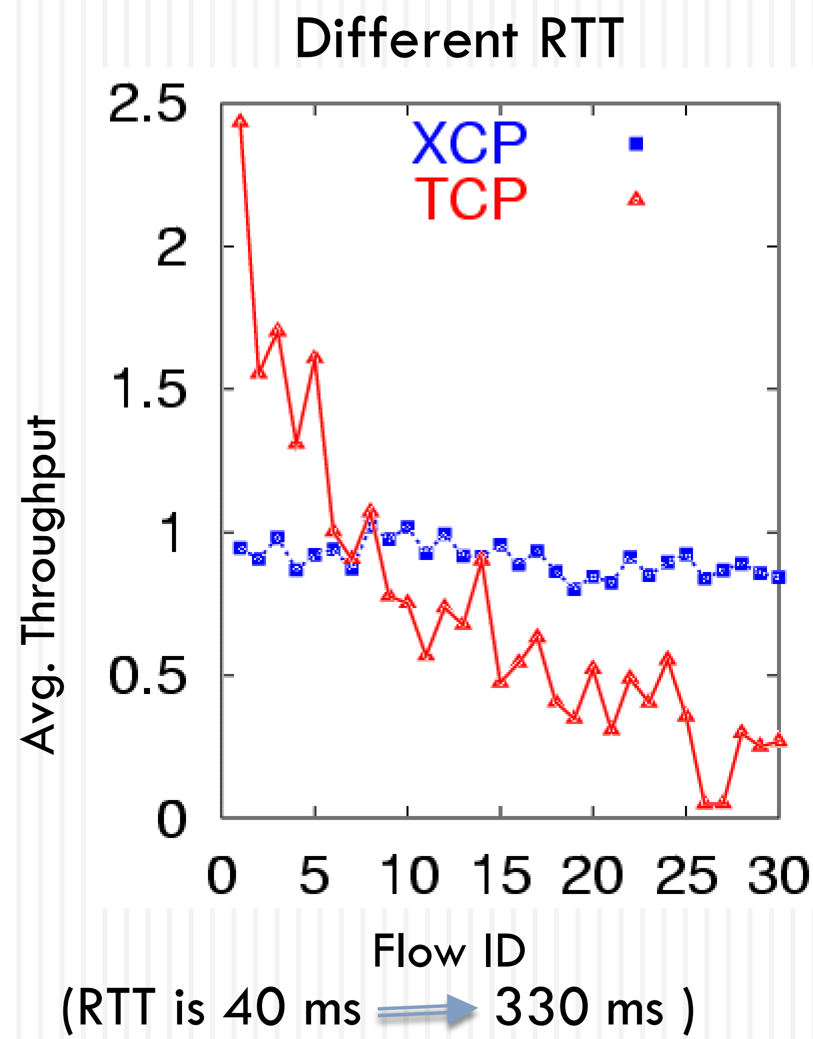
Drops



Arrivals of Short Flows/sec

XCP is Fairer than TCP





XCP Summary

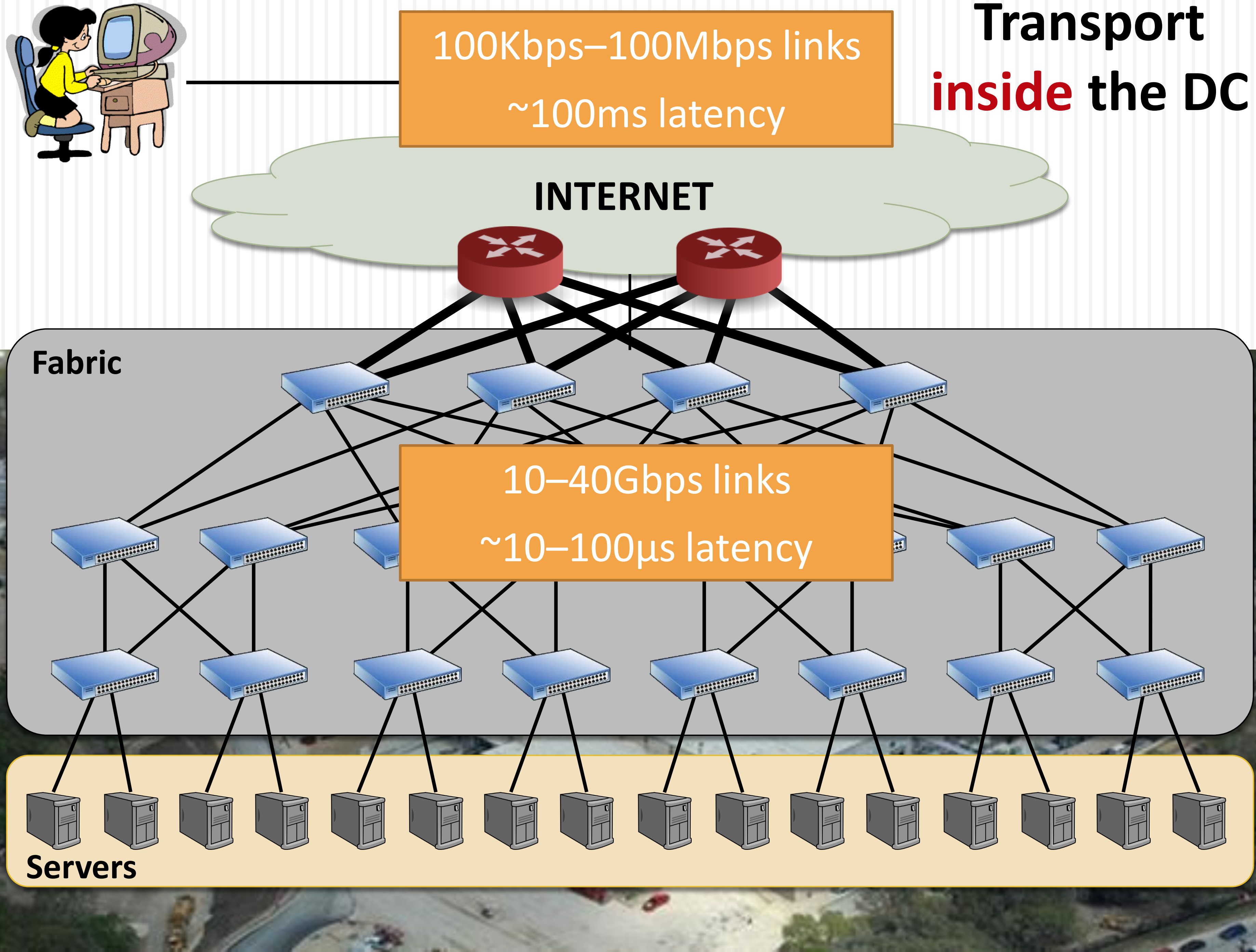
- XCP
  - Outperforms TCP
  - Efficient for any bandwidth
  - Efficient for any delay
  - Scalable
- Benefits of Decoupling
  - Use MIMD for congestion control which can grab/release large bandwidth quickly
  - Use AIMD for fairness which converges to fair bandwidth allocation

NS Code & More Information at:
http://www.isi.edu/isi-xcp/

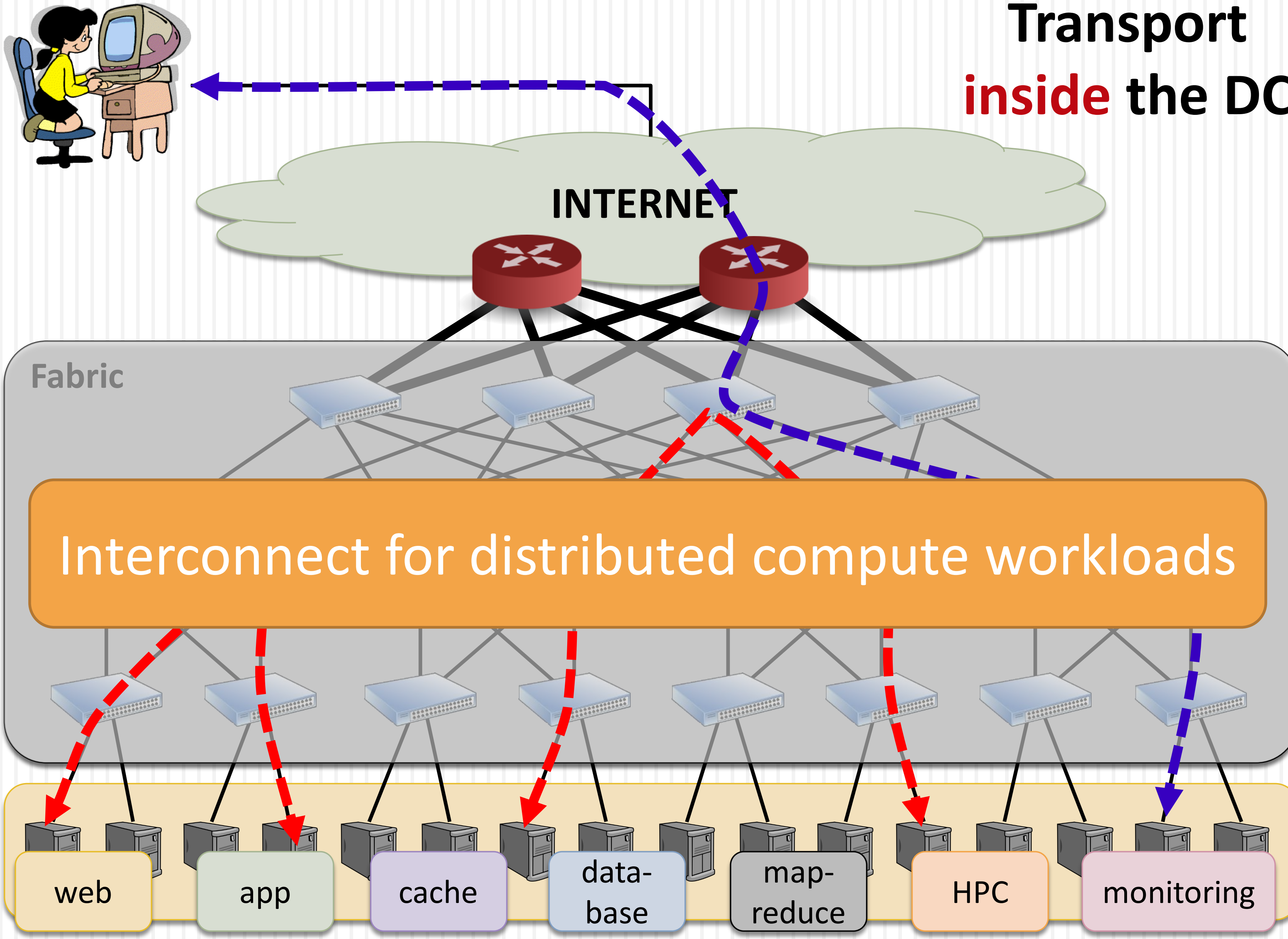
DATA CENTER CONGESTION CONTROL

Thanks for slides from Mohammad Alizadeh (MIT)

Transport **inside** the DC



Transport inside the DC



What's Different About DC Transport?

- Network characteristics
  - Very high link speeds (Gb/s); very low latency (microseconds)
- Application characteristics
  - Large-scale distributed computation
- Challenging traffic patterns
  - Diverse mix of mice & elephants
  - Incast
- Cheap switches
  - Single-chip shared-memory devices; shallow buffers

Data Center Workloads

Mice & Elephants

- Short messages

  (e.g., query, coordination) → **Low Latency**

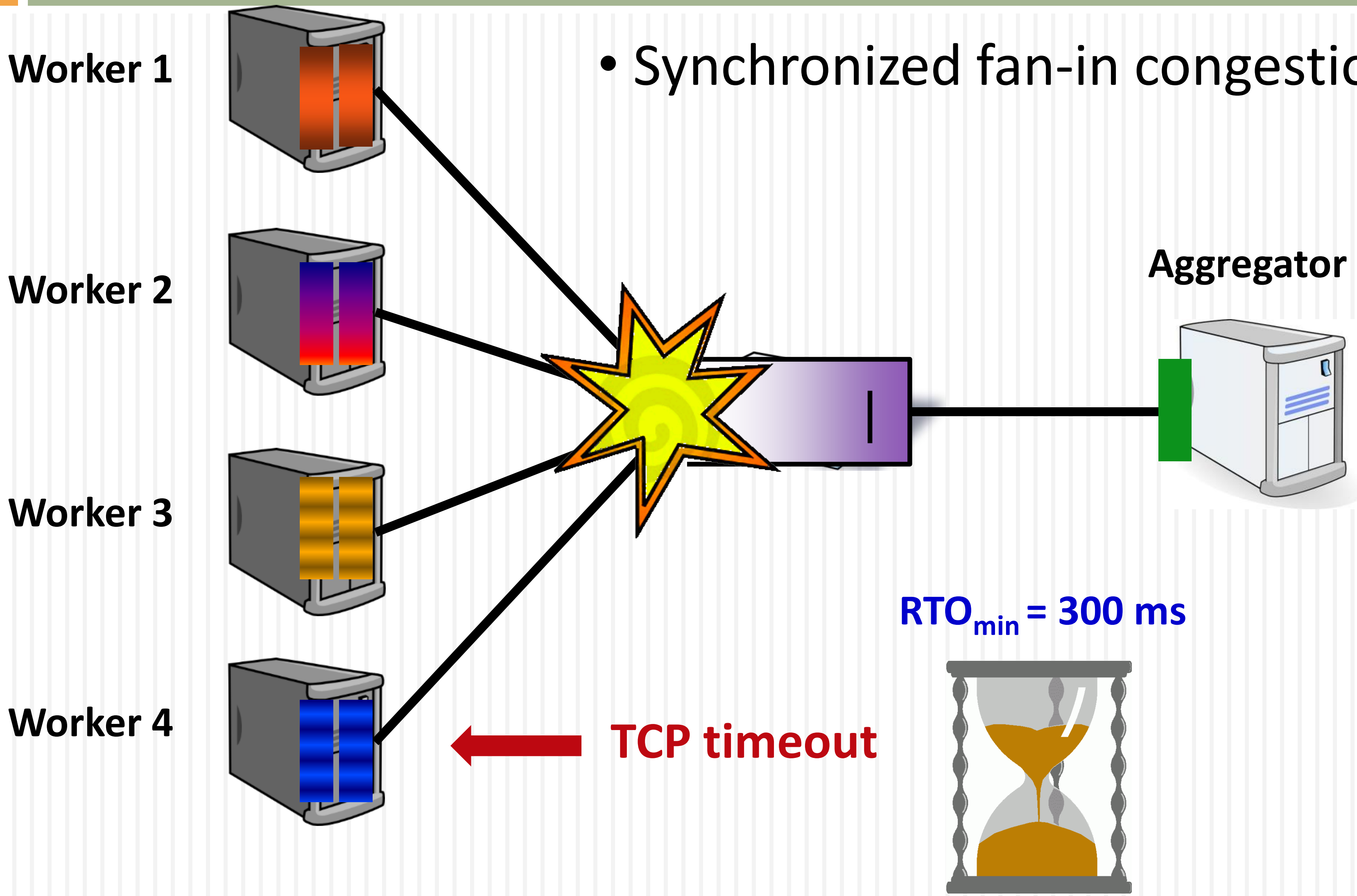
- Large flows

  (e.g., data update, backup) → **High Throughput**

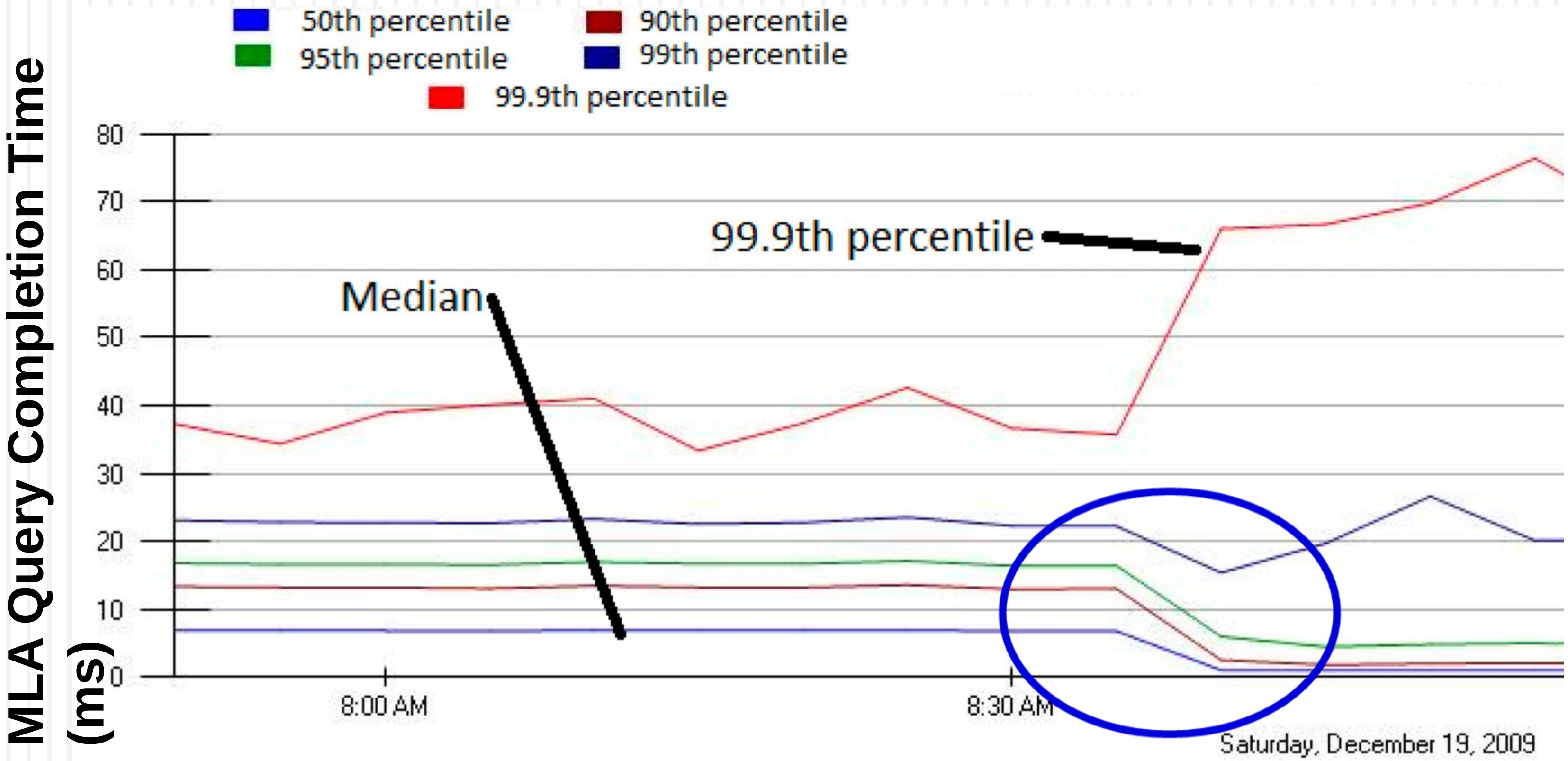
Incast

✧ Vasudevan et al. (SIGCOMM'09)

- Synchronized fan-in congestion



Incast in Bing



Jittering trades of median for high percentiles

DC Transport Requirements

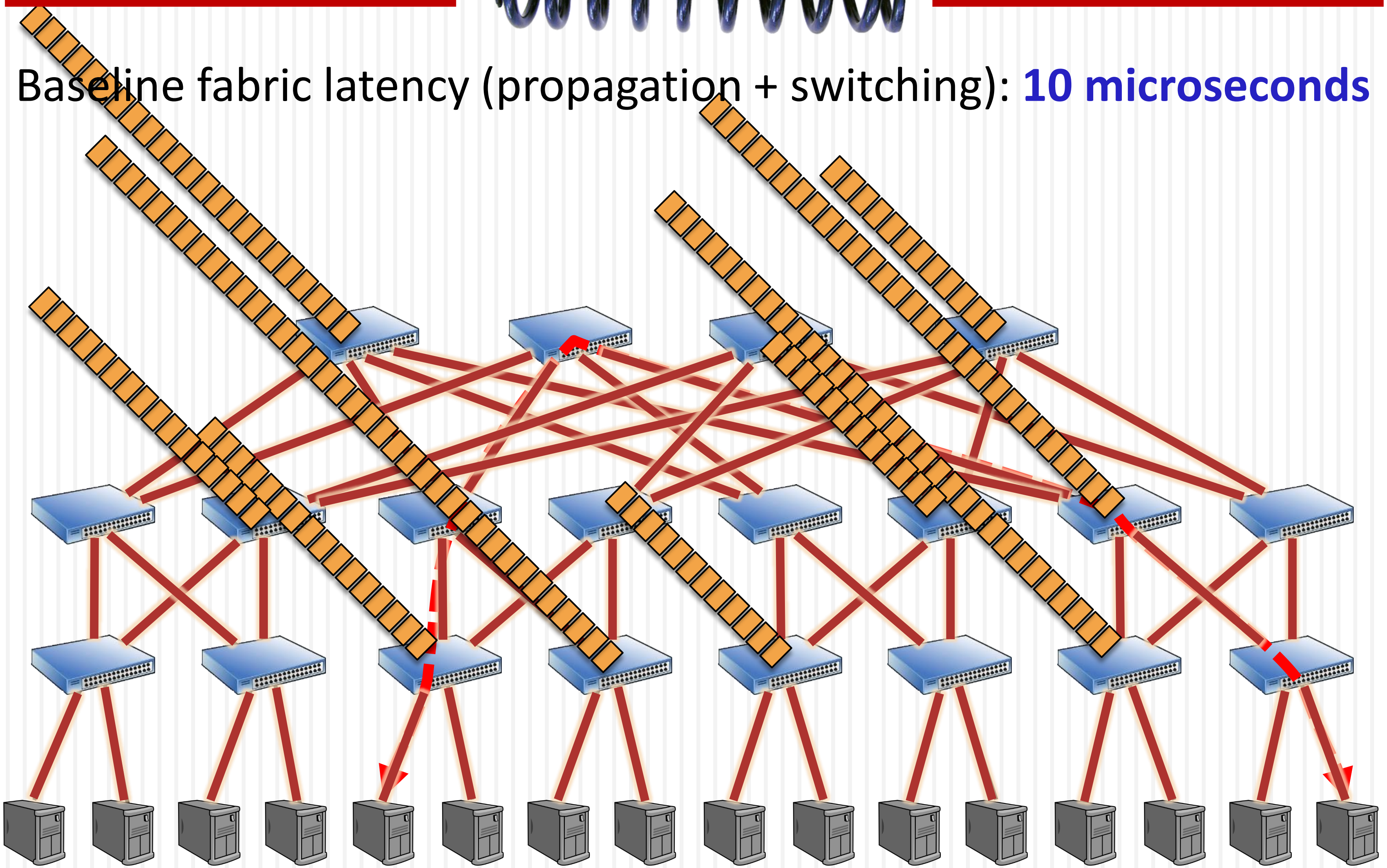
1. **Low Latency**
   – Short messages, queries
2. **High Throughput**
   – Continuous data updates, backups
3. **High Burst Tolerance**
   – Incast

The challenge is to achieve these *together*

High Throughput

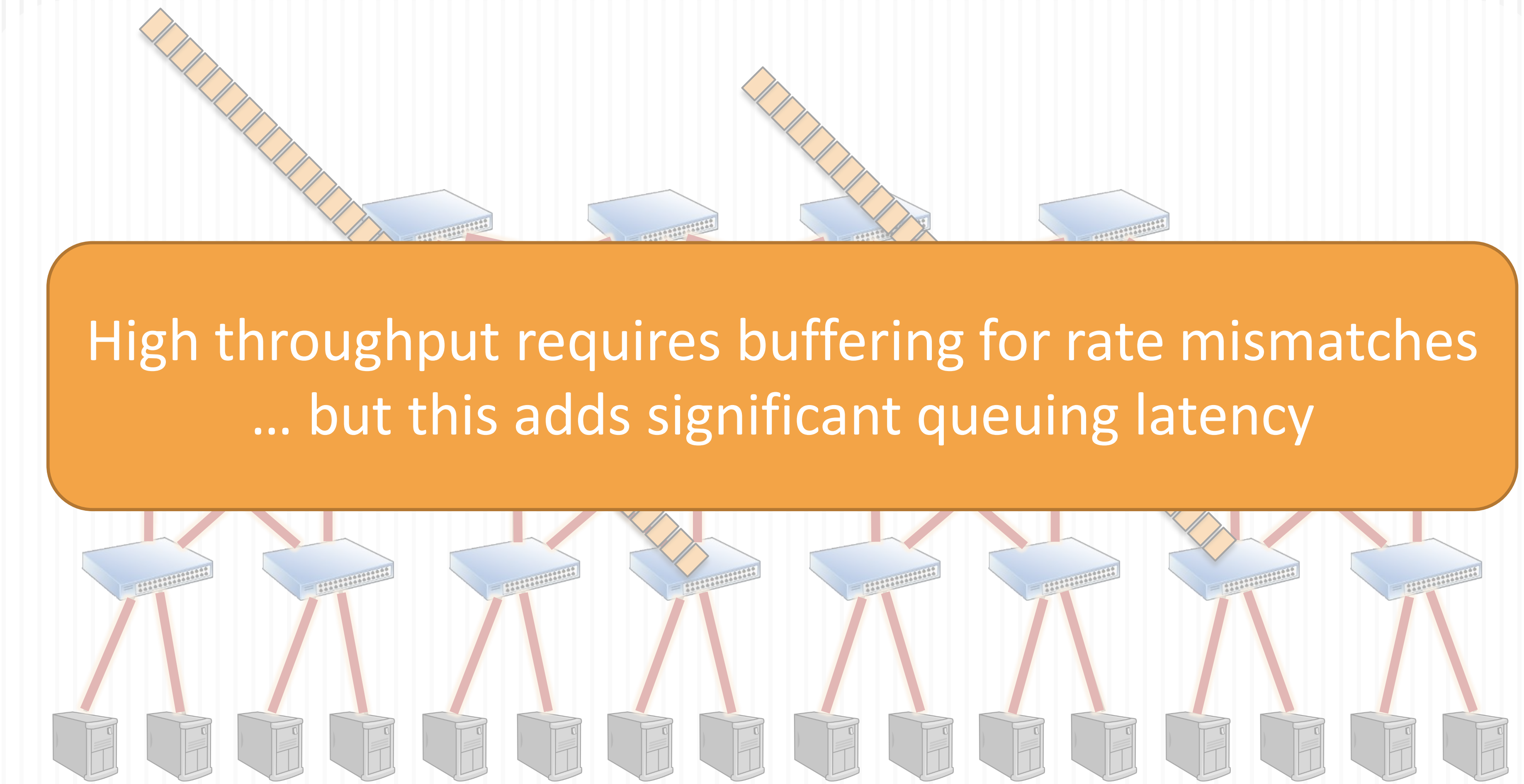
Low Latency

Baseline fabric latency (propagation + switching): **10 microseconds**



High Throughput  **Low Latency**

Baseline fabric latency (propagation + switching): **10 microseconds**



High throughput requires buffering for rate mismatches
… but this adds significant queuing latency

DATA CENTER TCP

TCP in the Data Center

- TCP [Jacobsen et al.'88] is widely used in the data center
  - More than **99%** of the traffic

- Operators work around TCP problems
  - Ad-hoc, inefficient, often expensive solutions
  - TCP is deeply ingrained in applications

Practical deployment is hard
→ keep it simple!

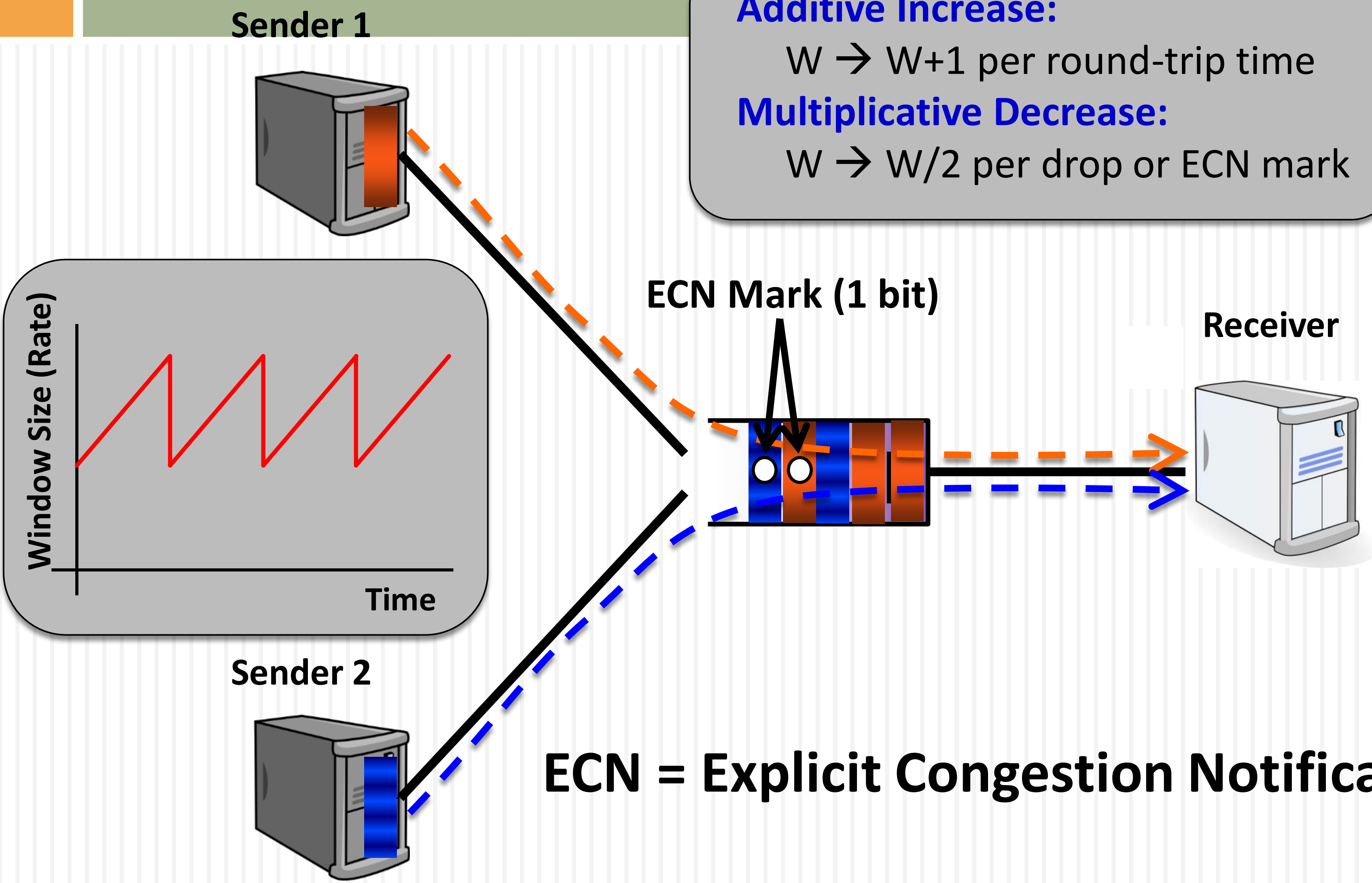
Review: The TCP Algorithm

Additive Increase:

W → W+1 per round-trip time

Multiplicative Decrease:

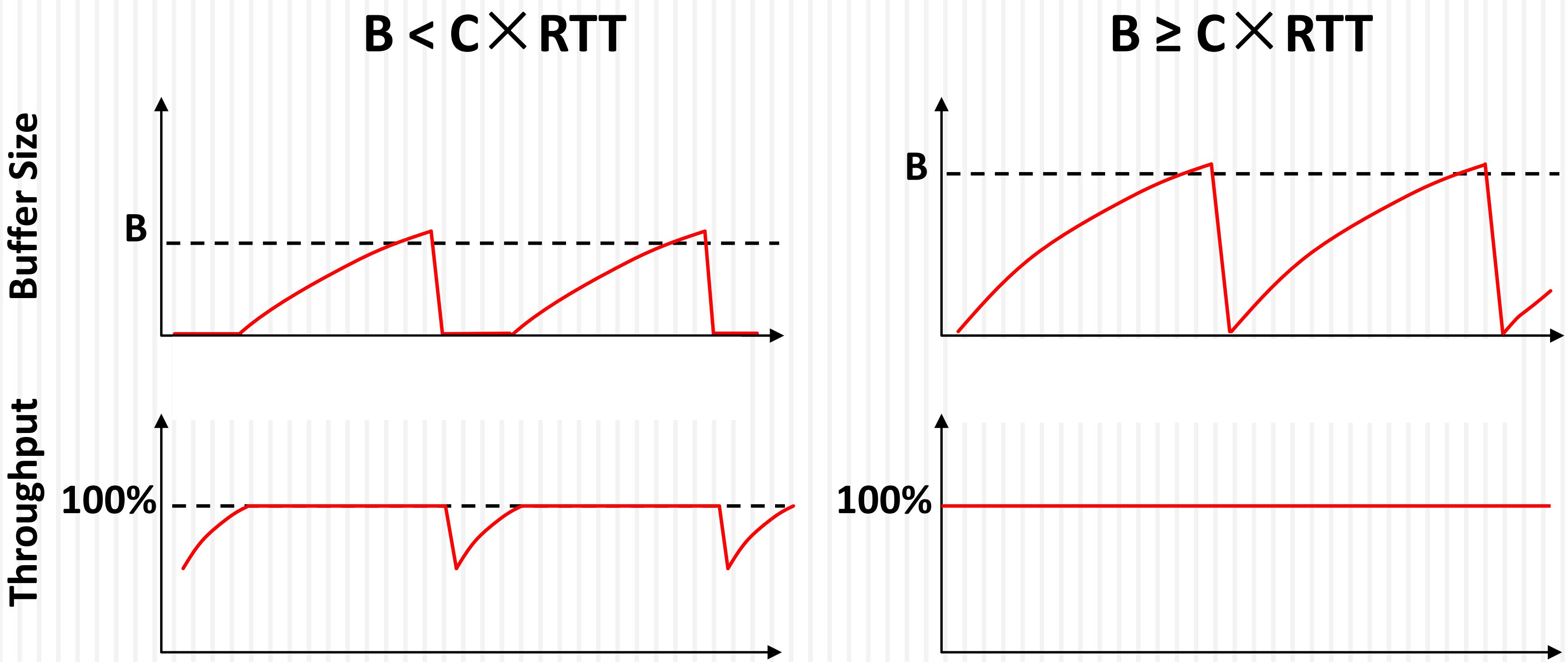
W → W/2 per drop or ECN mark



ECN = Explicit Congestion Notification

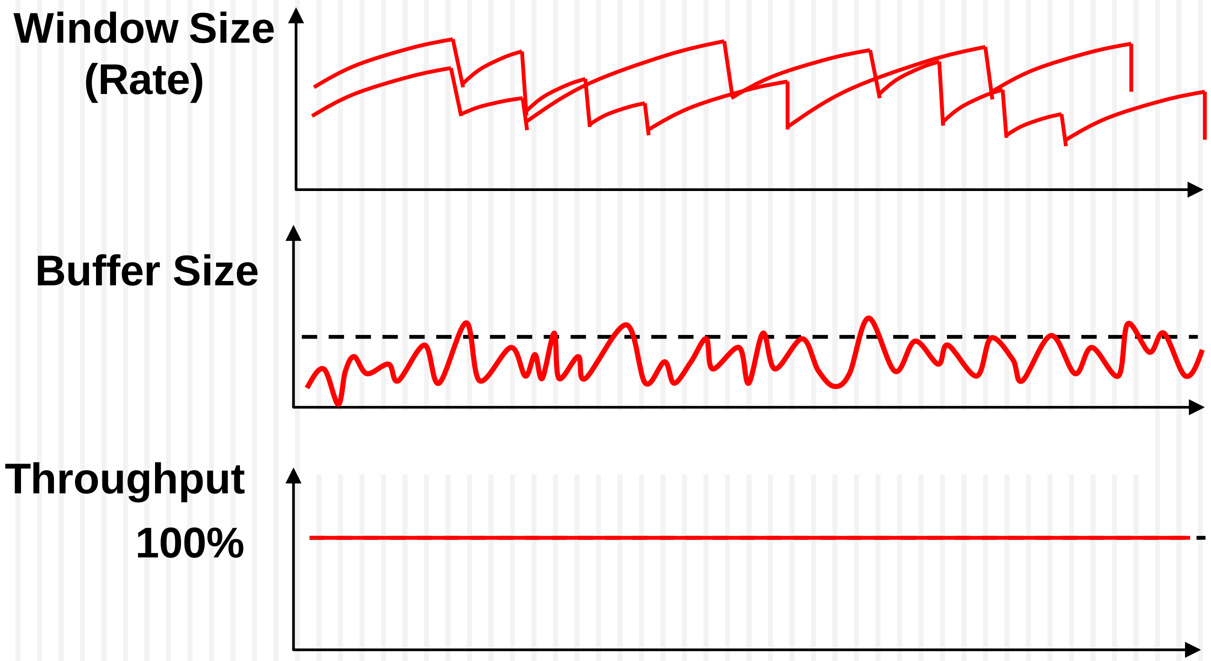
TCP Buffer Requirement

- Bandwidth-delay product rule of thumb:
  - A single flow needs **C×RTT** buffers for **100% Throughput.**



Reducing Buffer Requirements

- Appenzeller et al. (SIGCOMM '04):
  - Large # of flows: $C \times RTT/\sqrt{N}$ is enough.



Reducing Buffer Requirements

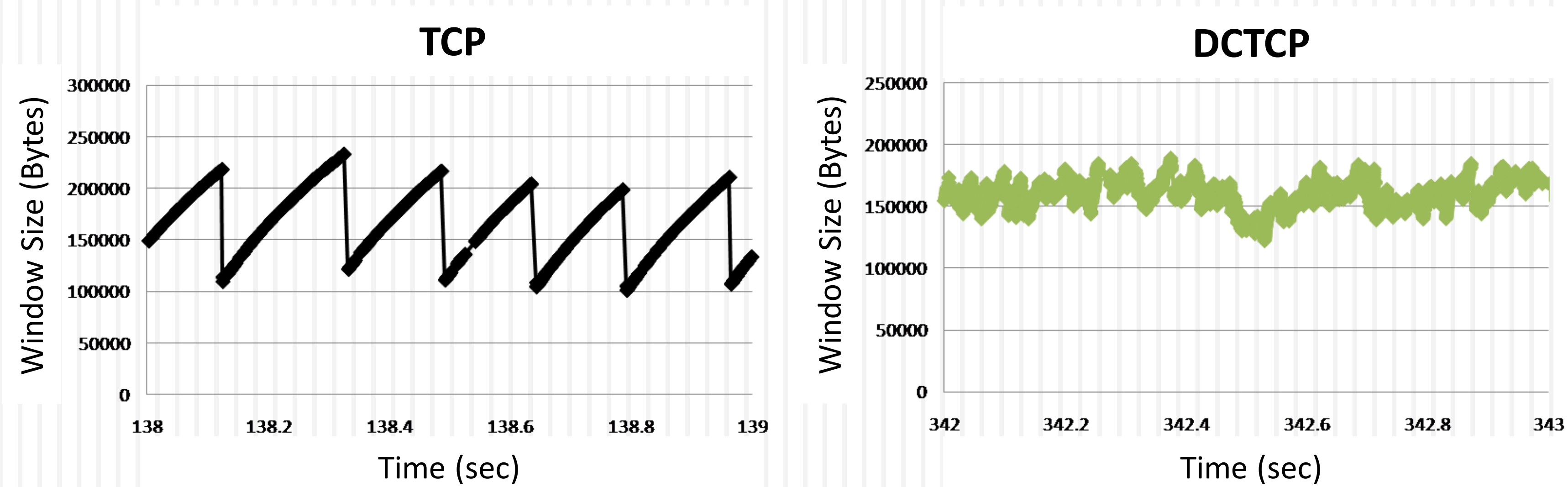
- Appenzeller et al. (SIGCOMM '04):
  - Large # of flows: $C \times RTT/\sqrt{N}$ is enough
- Can't rely on stat-mux benefit in the DC.
  - Measurements show typically **only 1-2 large flows** at each server

Key Observation:
Low variance in sending rate → Small buffers suffice

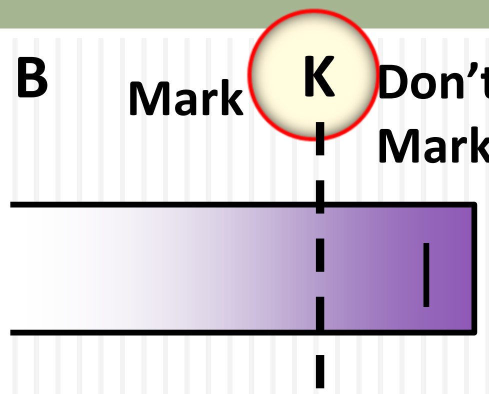
DCTCP: Main Idea

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

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:

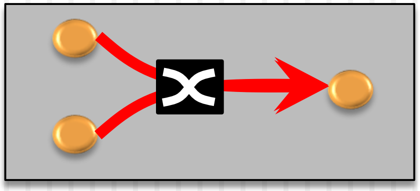
– Maintain running average of ***fraction*** of packets marked **(*α*)**.

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

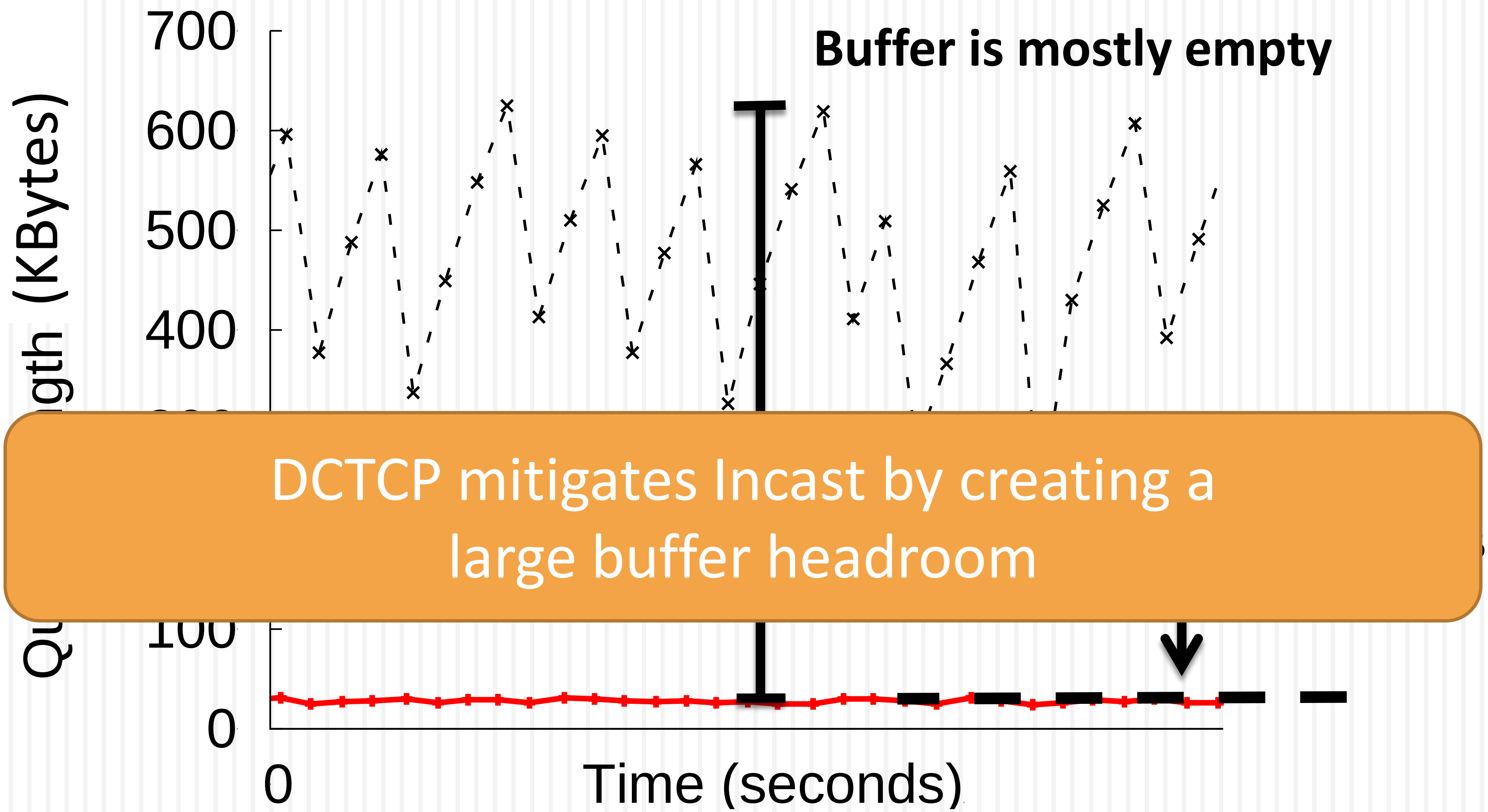
➢ **Adaptive window decreases:** $W \leftarrow (1 - \frac{\alpha}{2})W$

– Note: decrease factor between 1 and 2.

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

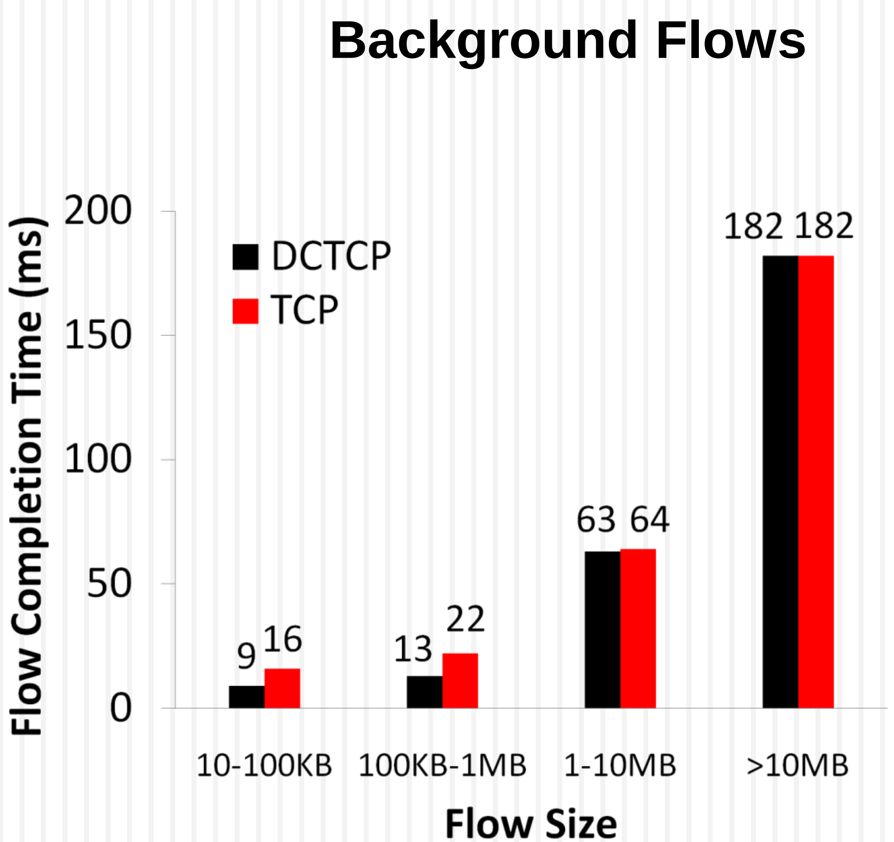
3. **High Burst Tolerance**
   - ✓ **Large buffer headroom** → bursts fit
   - ✓ **Aggressive marking** → sources react before packets are dropped

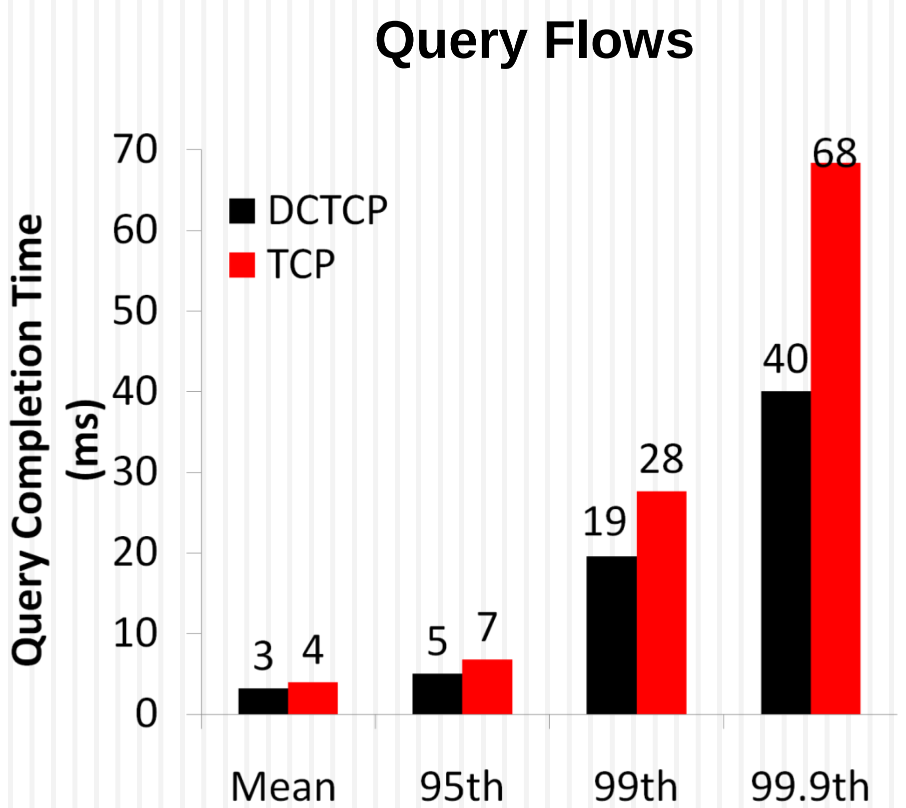
DISCUSSION

Evaluation

- Implemented in Windows stack.
- Real hardware, **1Gbps and 10Gbps** experiments
  - **90 server testbed**
  - **Broadcom Triumph** **48 1G ports – 4MB shared memory**
  - **Cisco Cat4948** **48 1G ports – 16MB shared memory**
  - **Broadcom Scorpion** **24 10G ports – 4MB shared memory**
- Numerous micro-benchmarks
  - **Throughput and Queue Length**
  - **Multi-hop**
  - **Queue Buildup**
  - **Buffer Pressure**
  - **Fairness and Convergence**
  - **Incast**
  - **Static vs Dynamic Buffer Mgmt**
- **Bing cluster benchmark**

Bing Benchmark (baseline)





Bing Benchmark (scaled 10x)