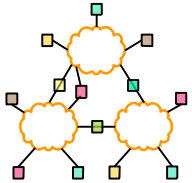


Computer Networks

(Graduate level)

Lecture 9: TCP Behavior and New Developments

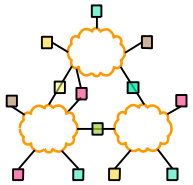
University of Tehran

Dept. of EE and Computer Engineering

By:

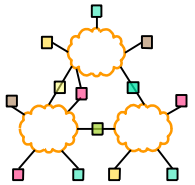
Dr. Nasser Yazdani

TCP and …



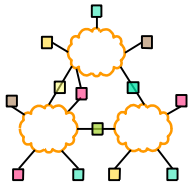
- TCP Latency Model and performance
- High Speed TCP
- Data Center TCP
- New developments
- Assigned reading
  - The Macroscopic Behavior of the TCP congestion Avoidance Algorithm
  - Modifying TCP's Congestion Control for High Speeds
  - DCTCP: Data Center TCP
  - Cubic TCP,
  - ….

Objectives



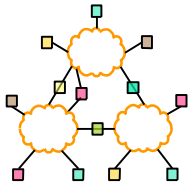
- Understanding TCP performance
  - packet loss probability and TCP performance
- Understanding TCP limitation and problems:
  - TCP behavior in high speed network
  - TCP fundamental limitation
- TCP fairness
  - Inter TCP versions and Intra TCP versions.
- Some new versions of TCP

TCP issues



- TCP Throughput
  - Slow start
  - Additive Increase Multiplicative Decrease (AIMD)
- TCP modeling: to have a realistic model and understanding of TCP behavior in the network
  - Packet lost pattern and probability
  - TCP end to end performance
  - Convergence time
- TCP operational enviroment
- TCP limitation: End point is slow to pump traffic

TCP latency modeling



Q: How long does it take to receive an object from a Web server after sending a request?

- TCP connection establishment
- data transfer delay

Notation, assumptions:

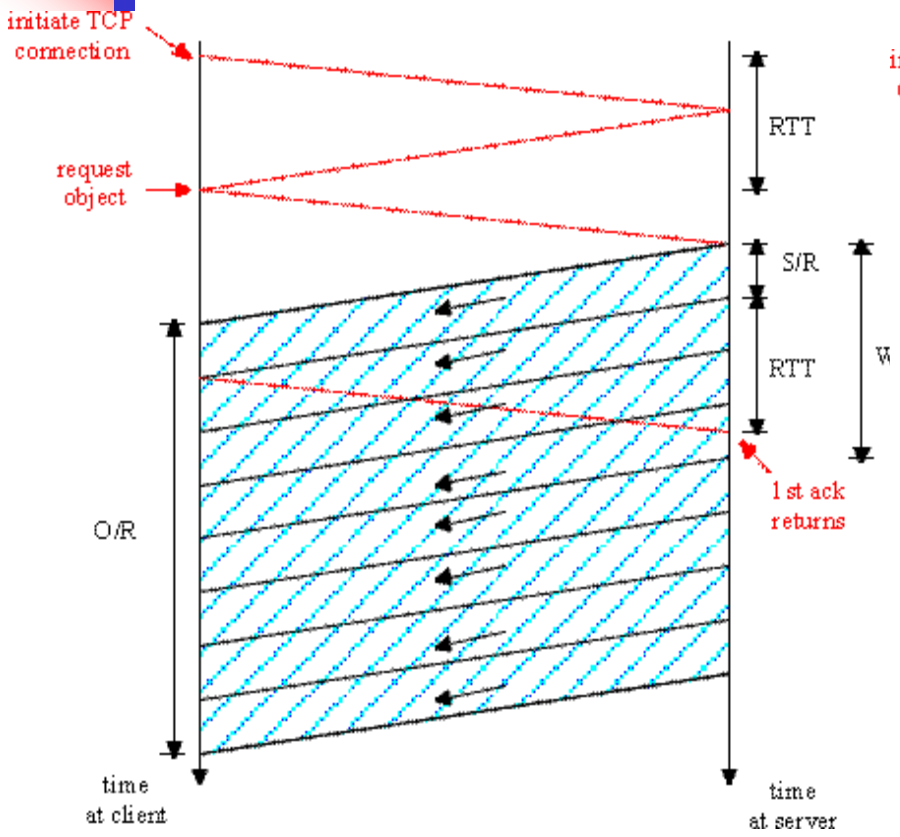
- Assume one link between client and server of rate R
- Assume: fixed congestion window, W segments
- S: MSS (bits)
- O: object size (bits)
- no retransmissions (no loss, no corruption)

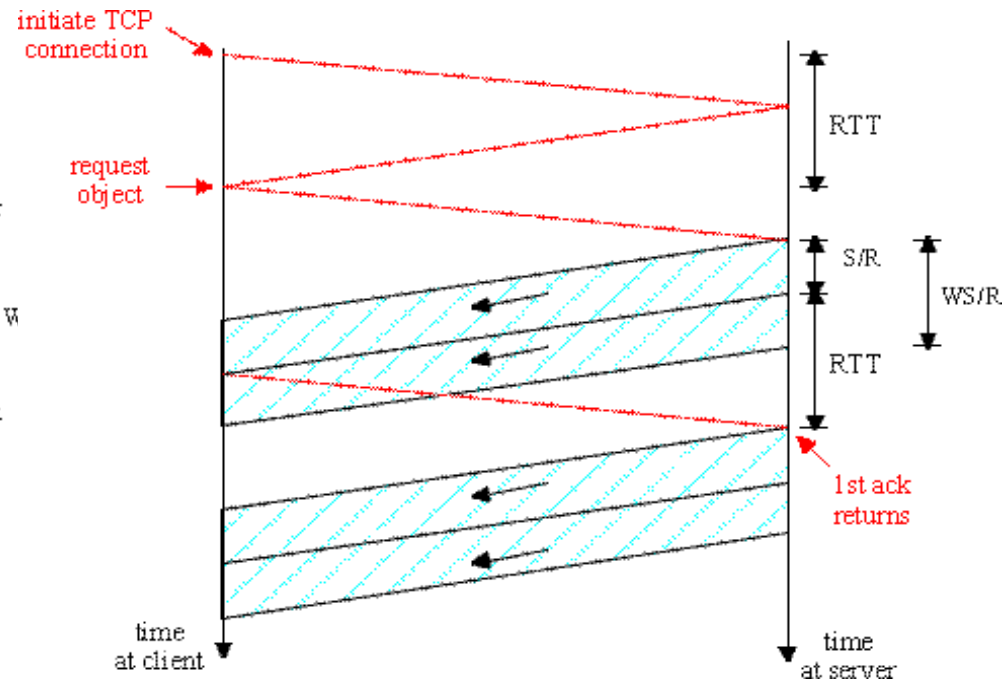
Two cases to consider:

- WS/R > RTT + S/R: ACK for first segment in window returns before window's worth of data sent
- WS/R < RTT + S/R: wait for ACK after sending window's worth of data sent

TCP latency Modeling

K:= O/WS

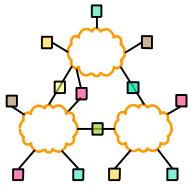




Case 1: latency = 2RTT + O/R

Case 2: latency = 2RTT + O/R + (K-1)[S/R + RTT - WS/R]

TCP Latency Modeling: Slow Start



- Now suppose window grows according to slow start.
- Will show that the latency of one object of size O is:

$$Latency = 2RTT + \frac{O}{R} + P\left[RTT + \frac{S}{R}\right] - (2^P - 1)\frac{S}{R}$$

where *P* is the number of times TCP stalls at server:

$$P = \min\{Q, K-1\}$$

- where Q is the number of times the server would stall if the object were of infinite size.

- and K is the number of windows that cover the object.

TCP Latency Modeling: Slow Start (cont.)

Example:

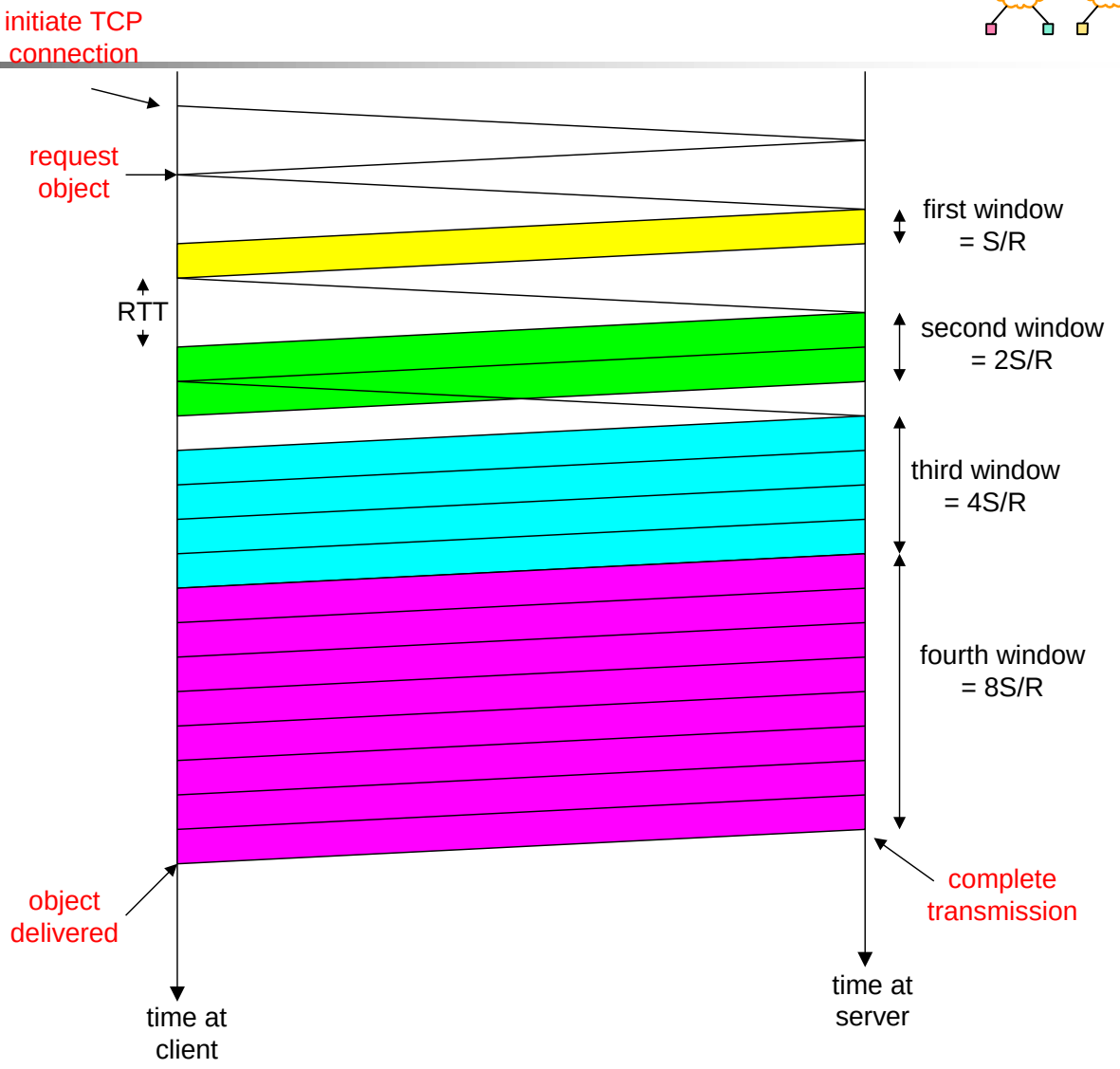
O/S = 15 segments

K = 4 windows

Q = 2

P = min{K-1,Q} = 2

Server stalls P=2 times.

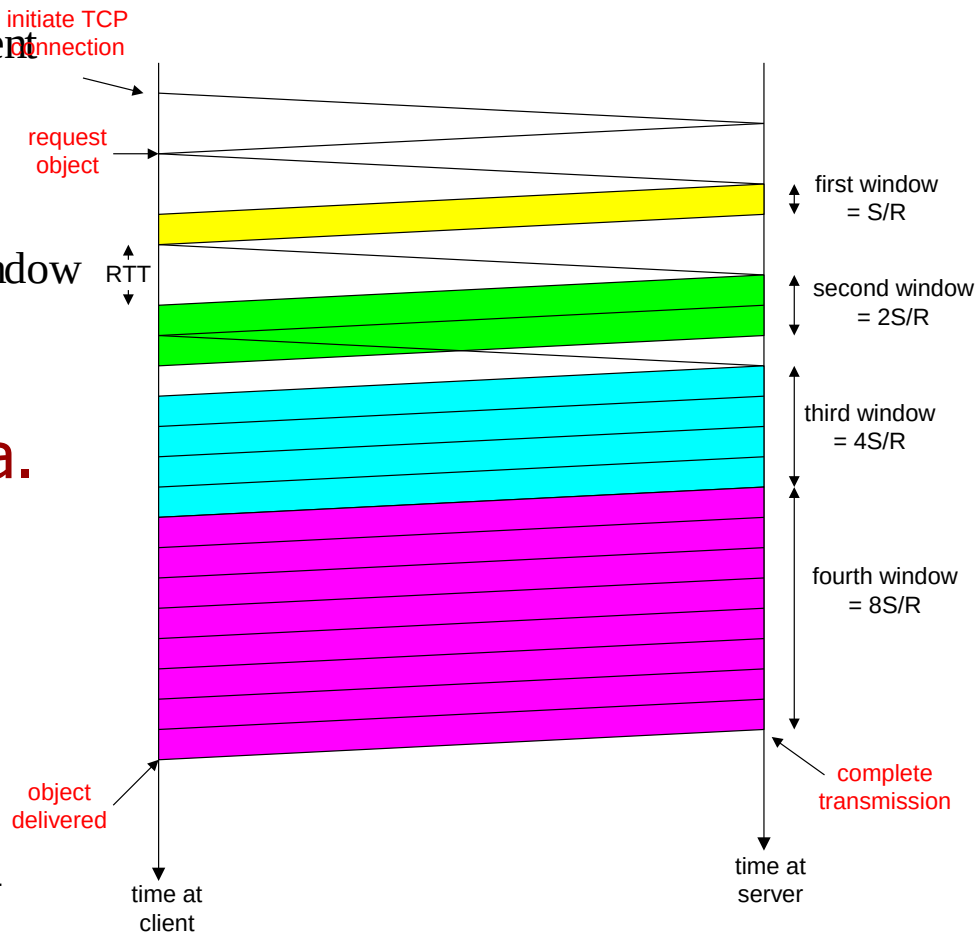


TCP Latency Modeling: Slow Start (cont.)

$$\frac{S}{R} + RTT = \text{time from when server starts to send segment until server receives acknowledgement}$$

$$2^{k-1}\frac{S}{R} = \text{time to transmit the kth window}$$

$$\left[\frac{S}{R} + RTT - 2^{k-1}\frac{S}{R}\right]^{+} = \text{stall time after the } k\text{th window}$$



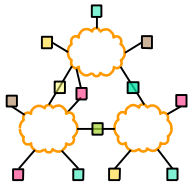
Stalling time: times BW was available but could not send data.

$$\text{latency} = \frac{O}{R} + 2RTT + \sum_{p=1}^{P} stallTime_p$$

$$= \frac{O}{R} + 2RTT + \sum_{k=1}^{P}\left[\frac{S}{R} + RTT - 2^{k-1}\frac{S}{R}\right]$$

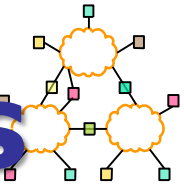
$$= \frac{O}{R} + 2RTT + P\left[RTT + \frac{S}{R}\right] - (2^{P} - 1)\frac{S}{R}$$

TCP Modeling



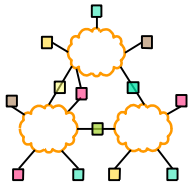
- TCP operating scale is very large
  - Models are required to gain deeper understanding of TCP dynamics
- Uncertainties can be modeled as stochastic processes
- Drive the design of TCP-friendly algorithms for multimedia applications
- Optimize TCP performance

TCP Modeling Essentials



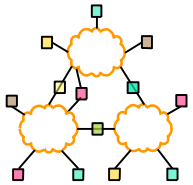
- Mainly Reno flavour are modelled
- Two main features are modelled
  - Window dynamics
  - Packet loss process
  - Try to evaluate TCP throughput and Latency
- Linear increase and multiplicative decrease is modelled
- The standard assumption: Throughput
  - $X(t) = W(t)/RTT$

Packet Loss Process



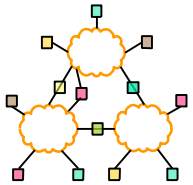
- Packet loss triggers window decrease
- Packet loss is uncertain
- This uncertainty is typically modelled as a stochastic process
  - E.g. probability *p* of losing a packet

Gallery of TCP Models



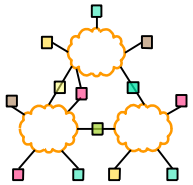
- Periodic model
- Detailed packet loss model
- Stochastic model
- Control system model
- Network system model

TCP Modeling



- Given the congestion behavior of TCP can we predict what type of performance we should get?
- What are the important factors
  - Loss rate
    - Affects how often window is reduced
  - RTT: Round Tripe Time
    - Affects increase rate and relates BW to window
  - RTO: Time out
    - Affects performance during loss recovery
  - MSS: Maximum Segment Size
    - Affects throughput

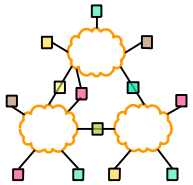
Simple TCP Model



- Some additional assumptions
  - Fixed RTT
  - No delayed ACKs
- In steady state, TCP losses packet each time window reaches to the network capacity, say W packets
  - Window drops to W/2 packets
  - Each RTT window increases by 1 packet
  - Utilized BW = MSS * avg window/RTT =
    - MSS * (W + W/2)/2 / RTT
    - .75 * MSS * W / RTT

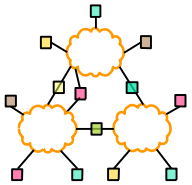
Ave. window size between two lost

TCP Performance



- Can TCP saturate a link?
- Congestion control
  - Increase utilization until… link becomes congested
  - React by decreasing window by 50%
  - Window is proportional to rate * RTT
- Doesn't this mean that the network oscillates between 50 and 100% utilization?
  - Average utilization = 75%??
  - No…this is *not* right!

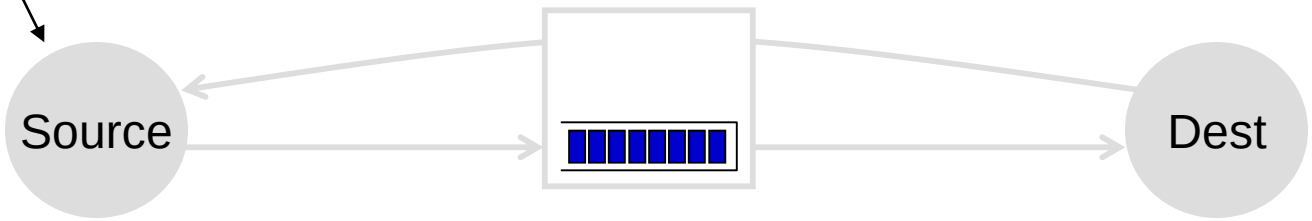
TCP Congestion Control

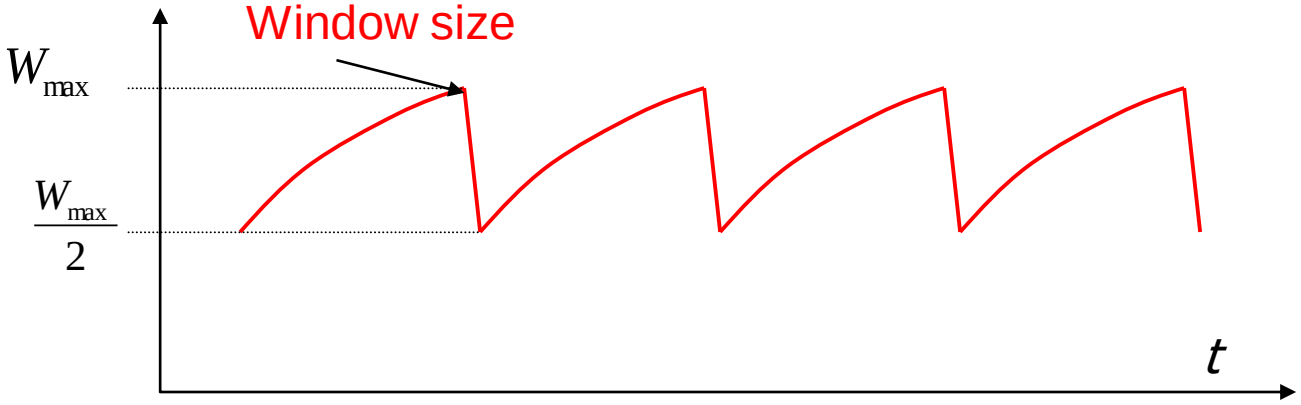


Rule for adjusting W

- If an ACK is received: W ← W+1/W
- If a packet is lost: W ← W/2

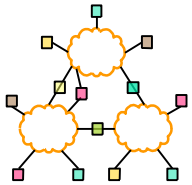
Only W packets may be outstanding

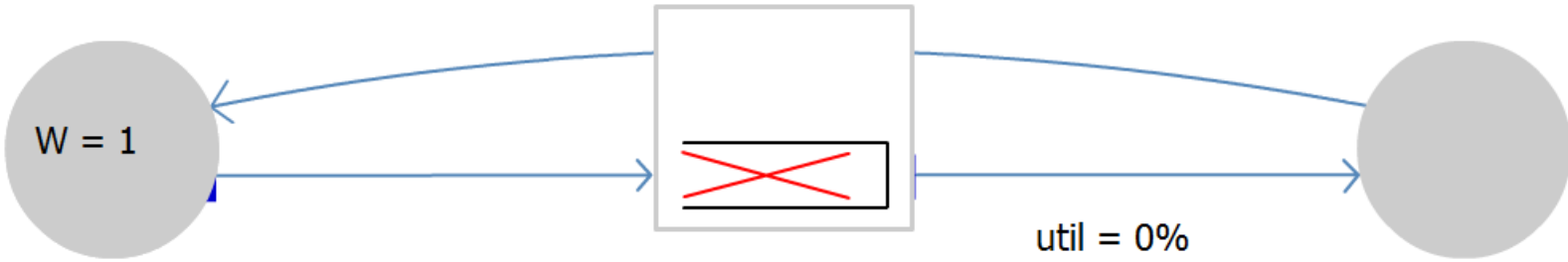




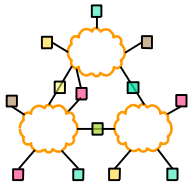
Single TCP Flow

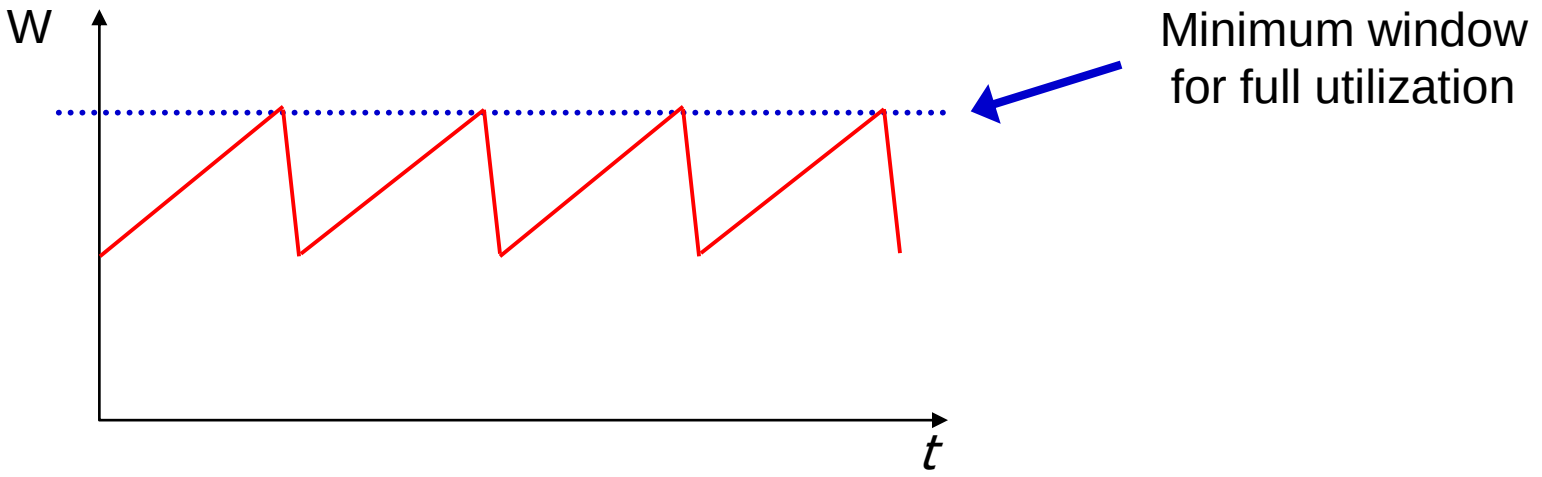
Router without buffers





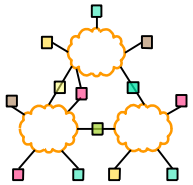
Summary Unbuffered Link





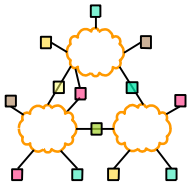
- The router can't fully utilize the link
  - If the window is too small, link is not full
  - If the link is full, next window increase causes drop
  - With no buffer it still achieves 75% utilization

TCP Performance



- In the real world, router queues play important role
  - Window is proportional to rate * RTT
    - But, RTT changes as well the window
  - Window to fill links = propagation RTT * bottleneck bandwidth
    - If window is larger, packets sit in queue on bottleneck link

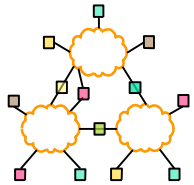
TCP Performance

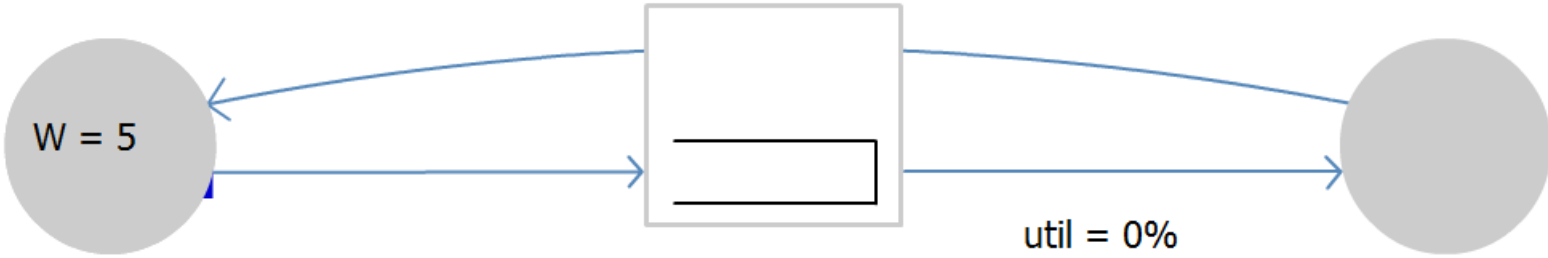


- If we have a large router queue → can get 100% utilization
  - But, router queues can cause large delays
- How big does the queue need to be?
  - Windows vary from W → W/2
    - Must make sure that link is always full
    - W/2 > RTT * BW
    - W = RTT * BW + Qsize
    - Therefore, Qsize > RTT * BW
  - Ensures 100% utilization
  - Delay?
    - Varies between RTT and 2 * RTT

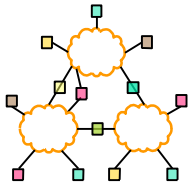
Single TCP Flow

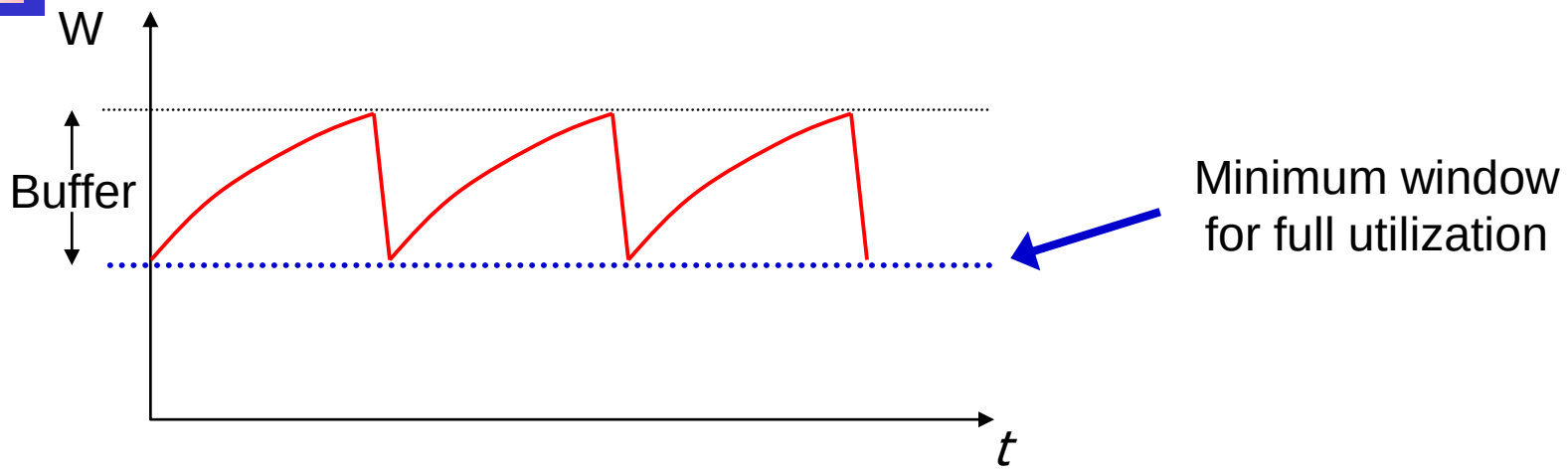
Router with large enough buffers for full link utilization





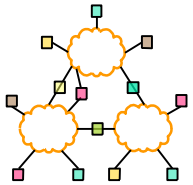
Summary Buffered Link





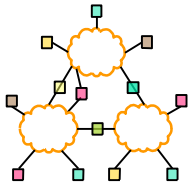
- With sufficient buffering we achieve full link utilization
  - The window is always above the critical threshold
  - Buffer absorbs changes in window size
    - Buffer Size = Height of TCP Sawtooth
    - Minimum buffer size needed is RTT*C, C is egress rate
  - This is the origin of the rule-of-thumb

Rule-of-thumb

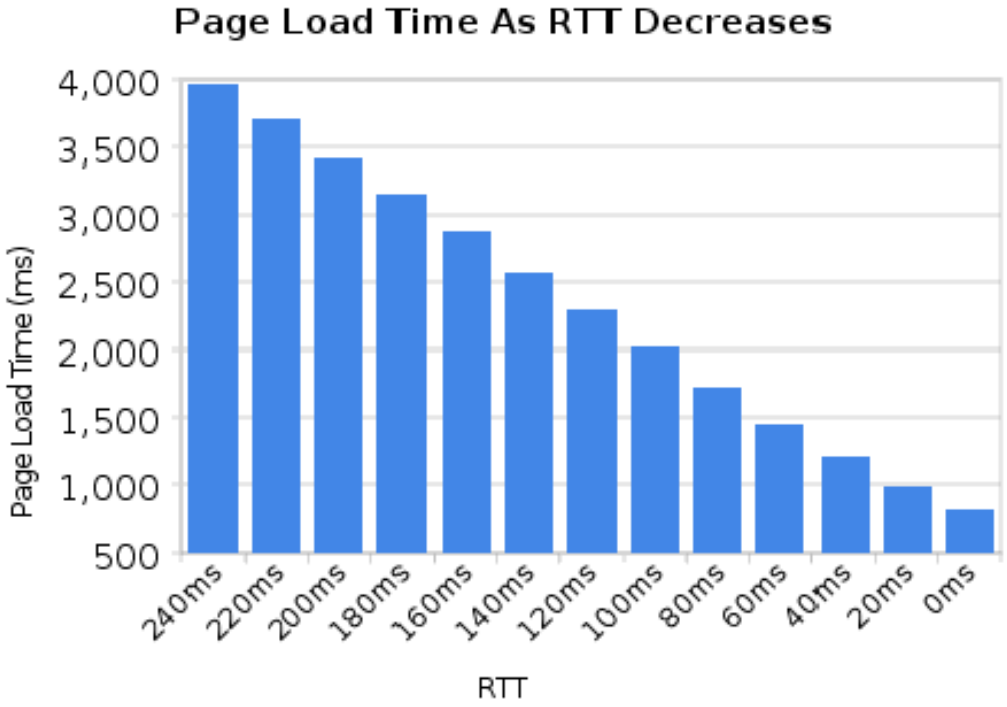


- In Rule of thumb keep windows large enough but
  - With high latency
  - Can we decrease latency? We must drain queue as much as we can, then make queue size RTT*BW/3? Where RTT*BW is the link capacity.
- Rule-of-thumb makes sense for one flow
  - Typical backbone link has > 20,000 flows
- Does the rule-of-thumb still hold?

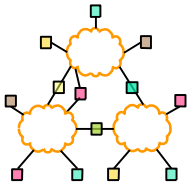
Some Concern?



- Increasing RTT slow down throughput
  - What is the relation between queue size and window dynamic?



Simple Loss Model

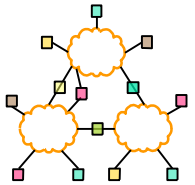


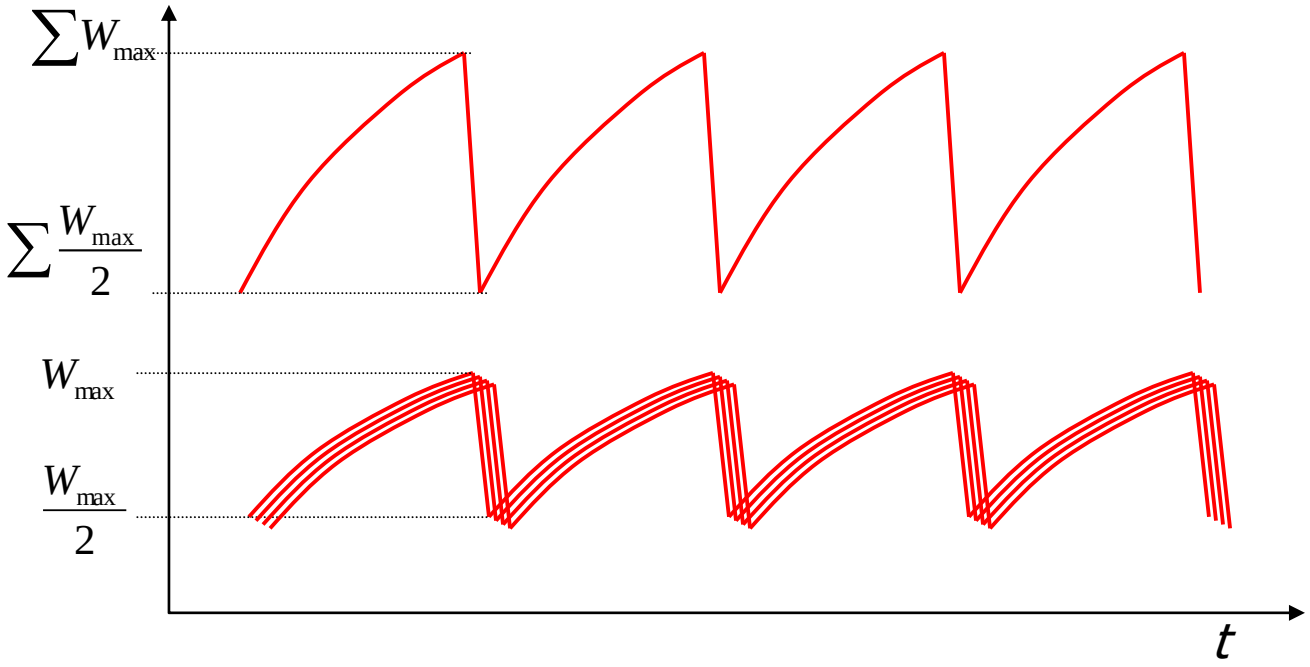
- What was the loss rate? No buffer
  - Packets transferred between losses =
    - Avg BW * time =
    - (.75 W/RTT) * (W/2 * RTT) = $3W^2/8$
  - 1 packet lost → loss rate => $p = 8/3W^2$

$$W = \sqrt{\frac{8}{3p}}$$

  - BW = .75 * MSS * W / RTT
  - BW = ^{3}MSS * $\sqrt{2}$ /RTT

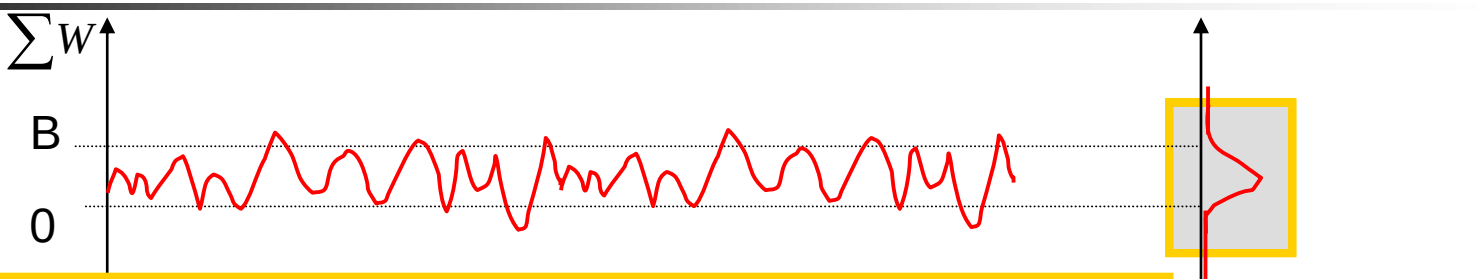
If flows are synchronized

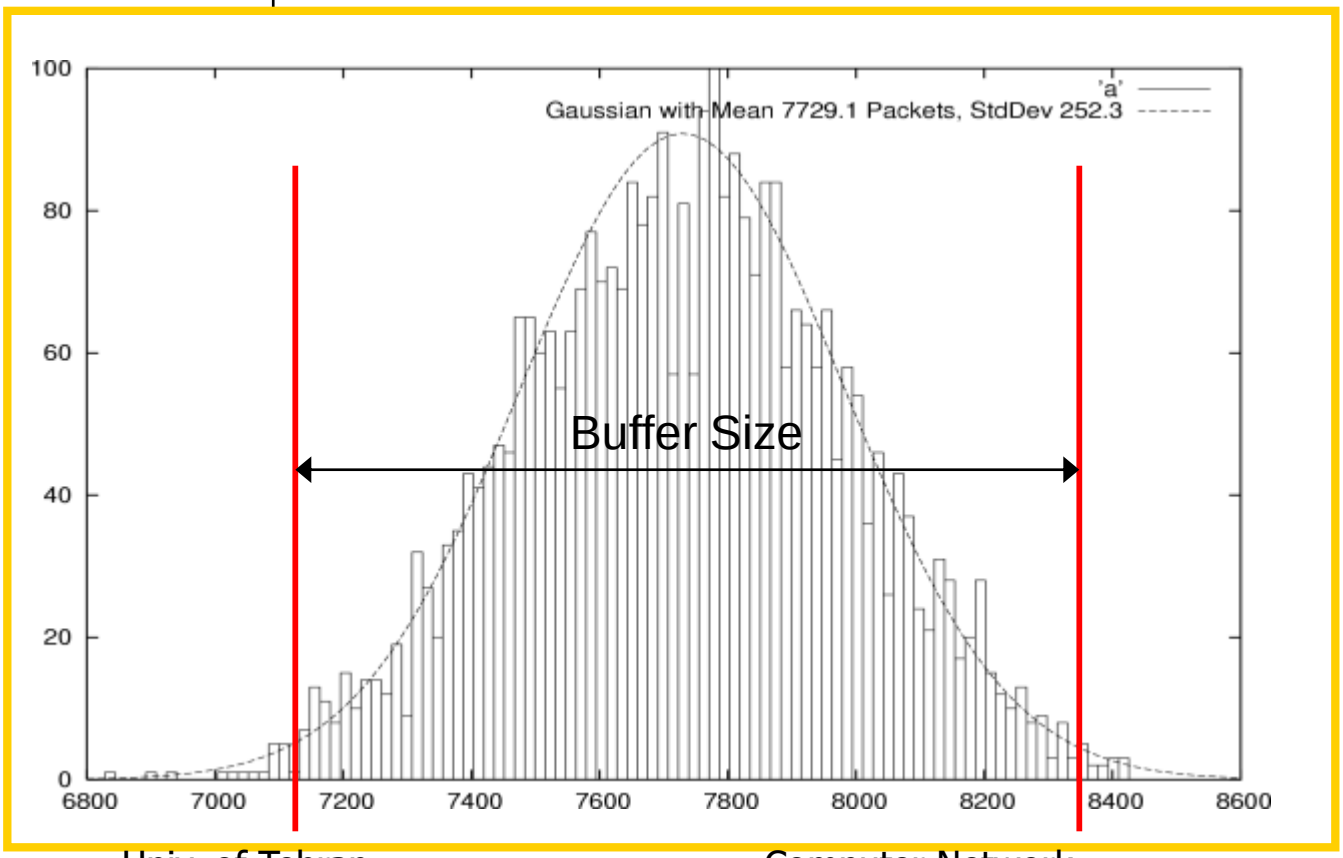


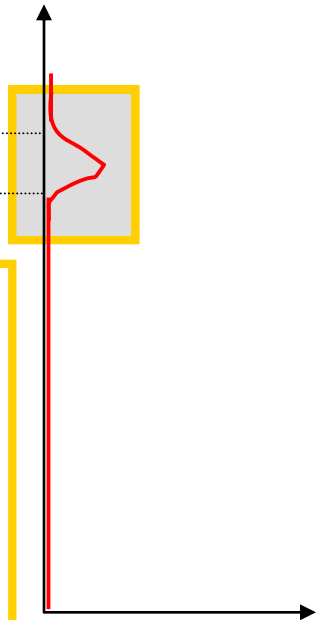


- Aggregate window has the same dynamics
- Therefore, buffer occupancy has same dynamics
- Rule-of-thumb still holds.

If flows are not synchronized

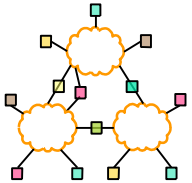






Probability Distribution

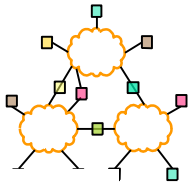
Central Limit Theorem

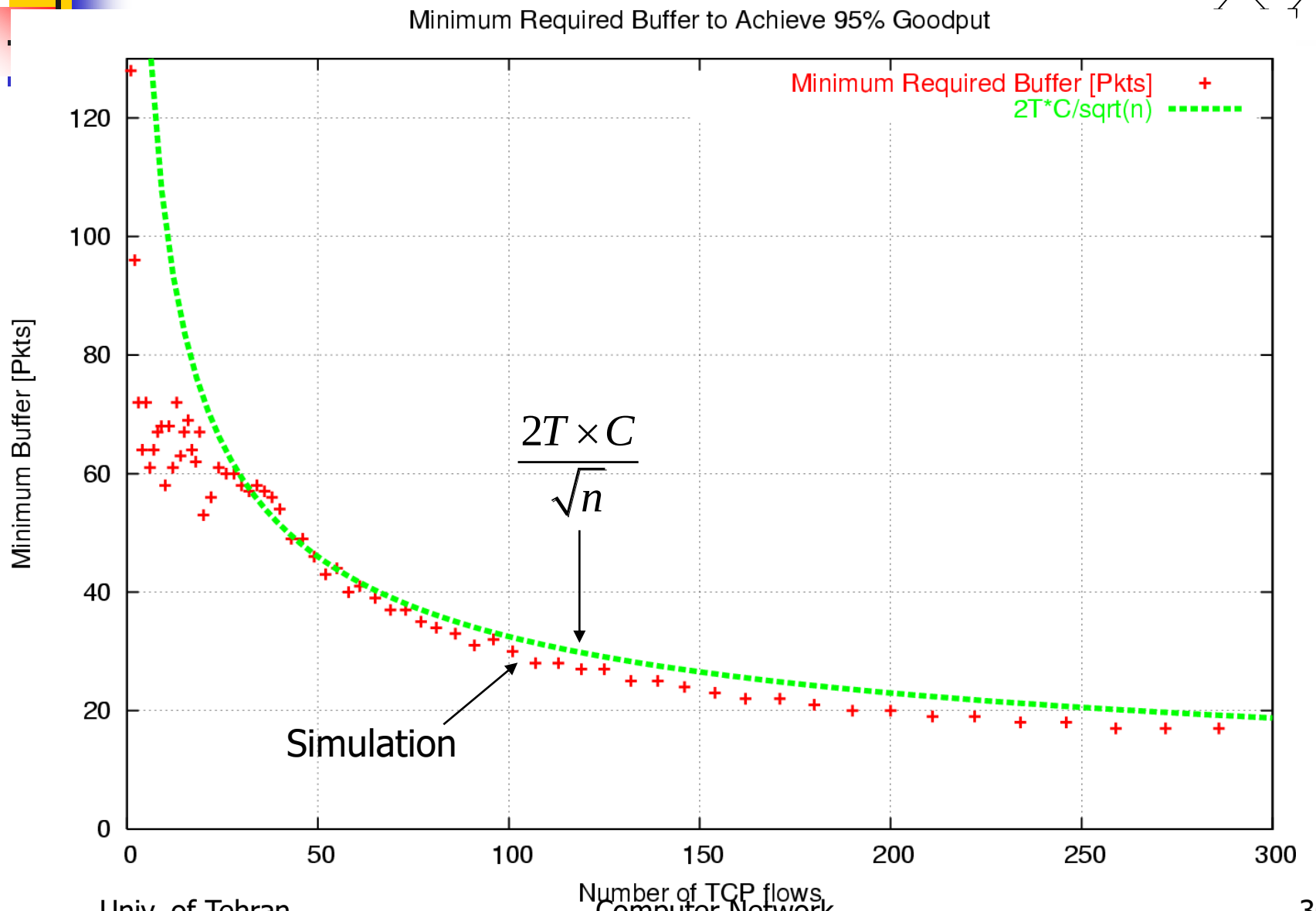


- CLT tells us that the more variables (Congestion Windows of Flows) we have, the narrower the Gaussian (Fluctuation of sum of windows)
  - Width of Gaussian decreases with $\frac{1}{\sqrt{n}}$
  - Buffer size should also decreases with $\frac{1}{\sqrt{n}}$

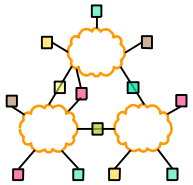
$$B \rightarrow \frac{B_{n=1}}{\sqrt{n}} = \frac{2T \times C}{\sqrt{n}}$$

Required buffer size

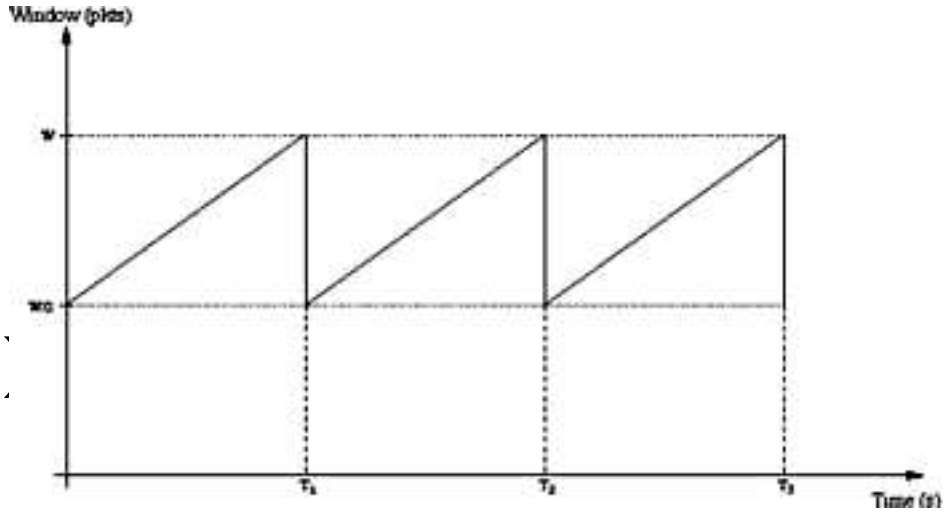




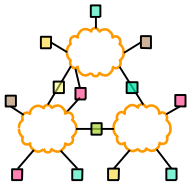
Periodic Model



- Simplest model for TCP
  - No specific flavor assumed
- Assumes a periodic pattern of congestion window
  - See Fig 5.1
- X(p) = 1/RTT*Root(3/2p)
  - P is the packet loss probability



Example 1 - Question



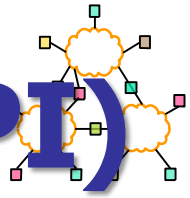
If a TCP connection has an average RTT of 200ms, and packet loss probability 0.05, what is the average throughput?

RTT=0.2, p=0.05.

Using Eq., throughput is:

X=(1/0.2)*(root(3/2x0.05)) = 27.4 pkts/sec

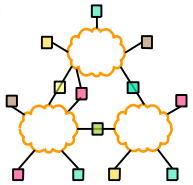
Proportional Integral (PI)



- Try to balance between queue size and increase in latency
- Three Ideas:

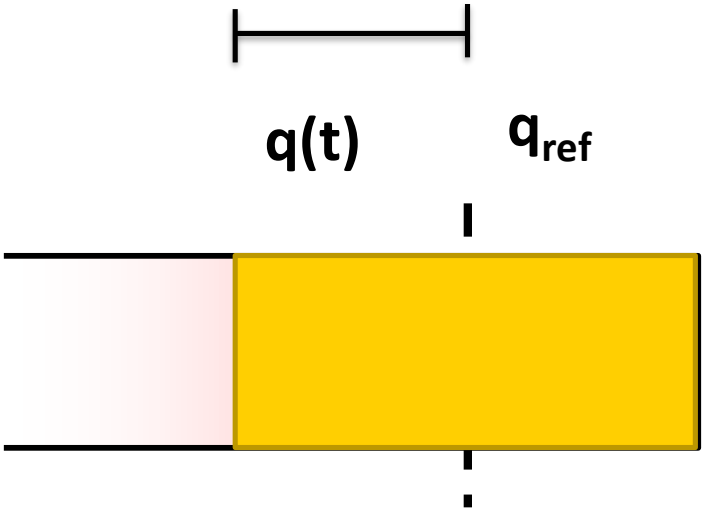
1. Remove EWMA → Faster response than RED
2. Integral control → Decouples queue length & number of flows
3. Use *derivative* of queue → Improves stability (less oscillation)

Proportional Integral (PI) Algorithm



Goal: Drive error to zero

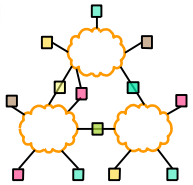
$e(t) = q(t) - q_{ref}$ ("error")

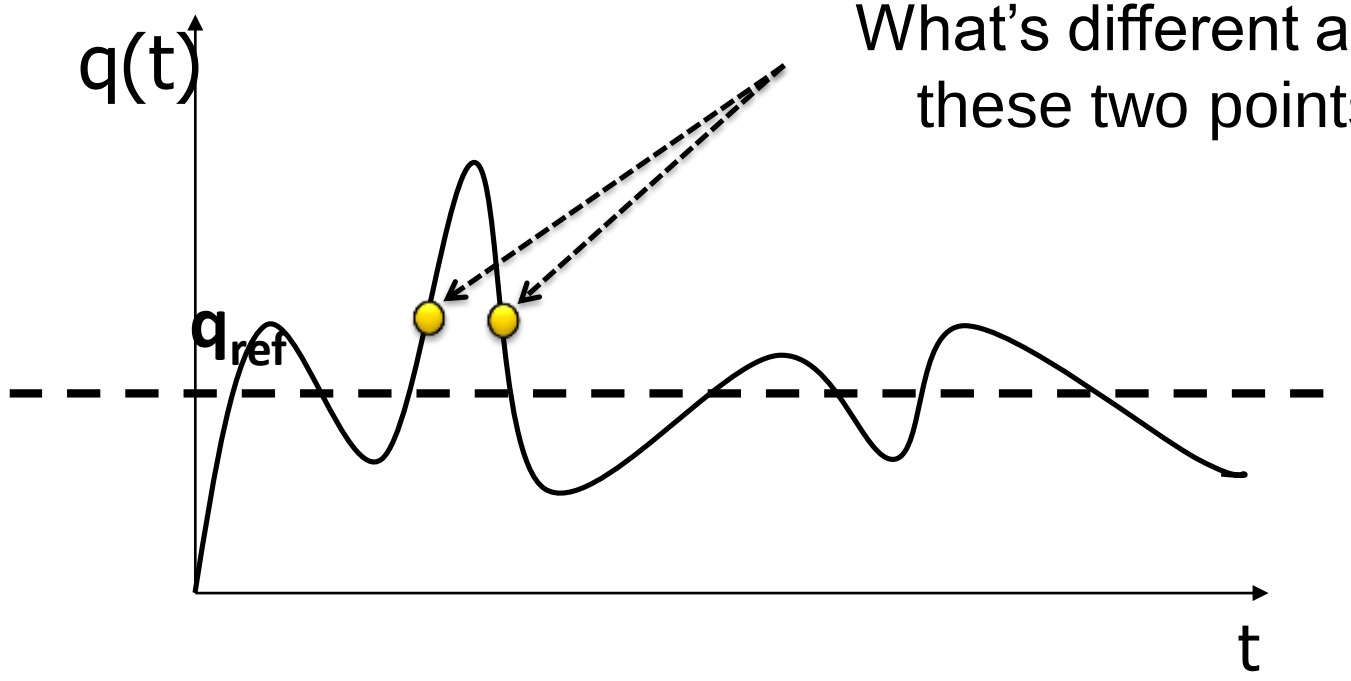


Every T:

$p(t) \leftarrow p(t - T) + \alpha^*(q(t) - q_{ref})$ ("Integral control")

Proportional Integral (PI) Algorithm

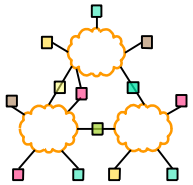




Every T:

$$p(t) \leftarrow p(t - T) + \alpha^*(q(t) - q_{ref}) + \beta^*(q(t) - q(t-T))$$

PIE – PI "Enhanced"



- Control delay instead of queue length
- Auto-tune parameters α and β based on value of p
- Other heuristics (e.g. burst allowance)

Internet Draft
Active Queue Management
Working Group
Intended Status: Experimental Track

R. Pan, P. Natarajan, F. Baker
G. White, B. VerSteeg, M.S. Prabhu
C. Piglione, V. Subramanian

Expires: October 6, 2016

April 4, 2016

PIE: A Lightweight Control Scheme To Address the Bufferbloat Problem

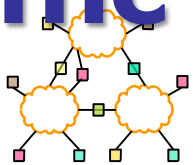
draft-ietf-aqm-pie-06

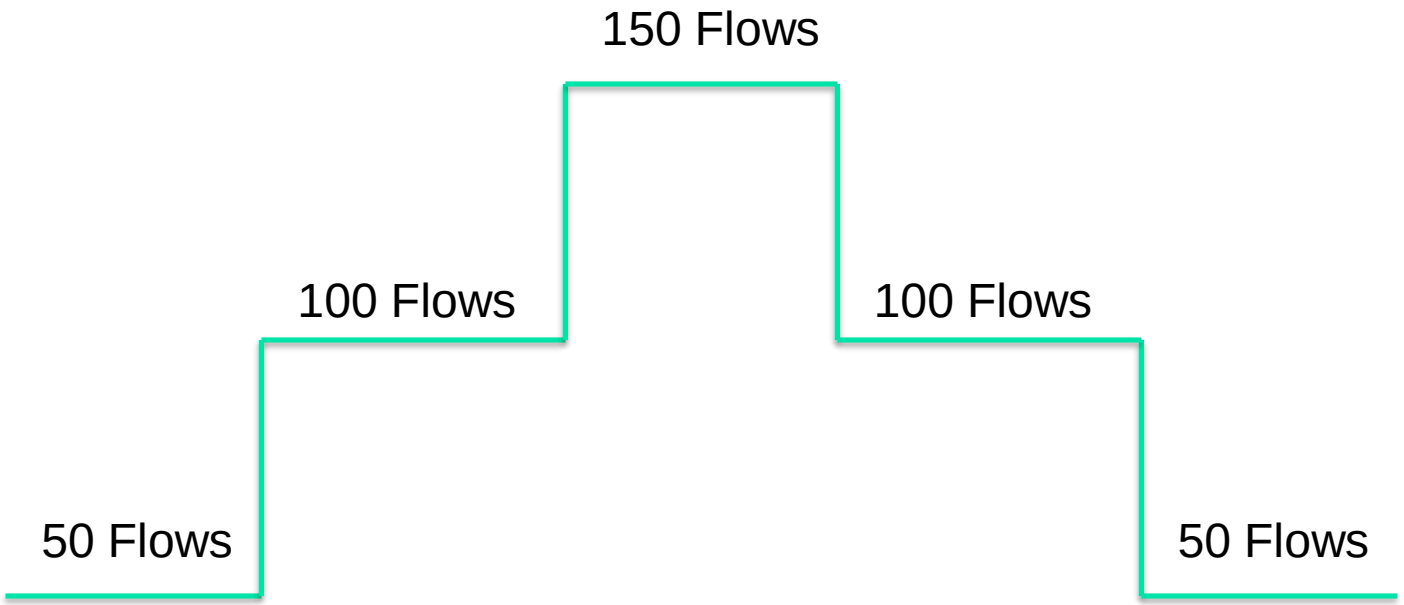
Abstract

Bufferbloat is a phenomenon where excess buffers in the network cause high latency and jitter. As more and more interactive applications (e.g. voice over IP, real time video streaming and financial transactions) run in the Internet, high latency and jitter degrade application performance. There is a pressing need to design intelligent queue management schemes that can control latency and jitter; and hence provide desirable quality of service to users.

This document presents a lightweight active queue management design, called PIE (Proportional Integral controller Enhanced), that can effectively control the average queueing latency to a target value. Simulation results, theoretical analysis and Linux testbed results have shown that PIE can ensure low latency and achieve high link utilization under various congestion situations. The design does not require per-packet timestamp, so it incurs very small overhead and is simple enough to implement in both hardware and software.

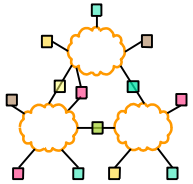
Example: Varying TCP Traffic Intensity on 10Mbps Link

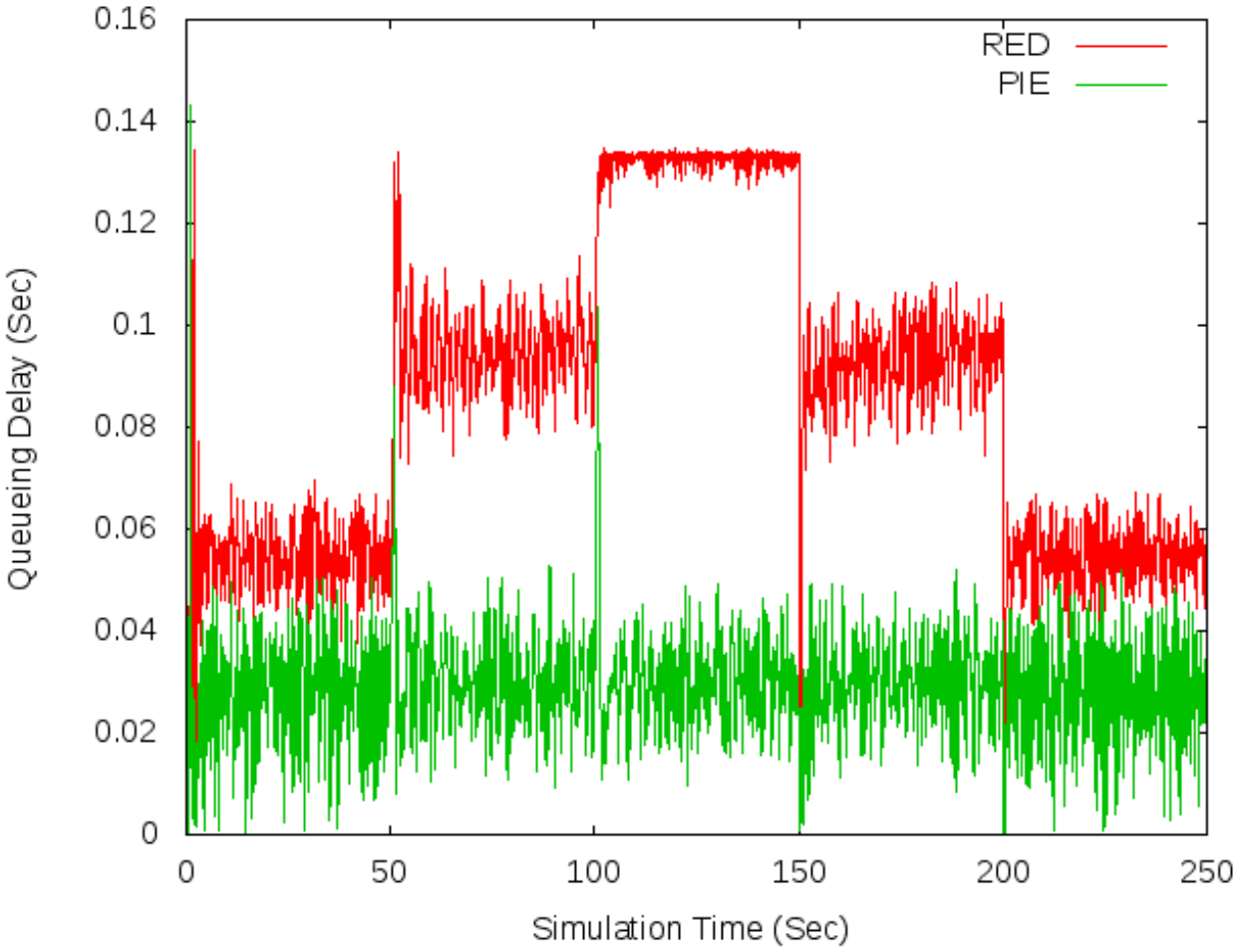




- Credit: Rong Pan (Cisco)

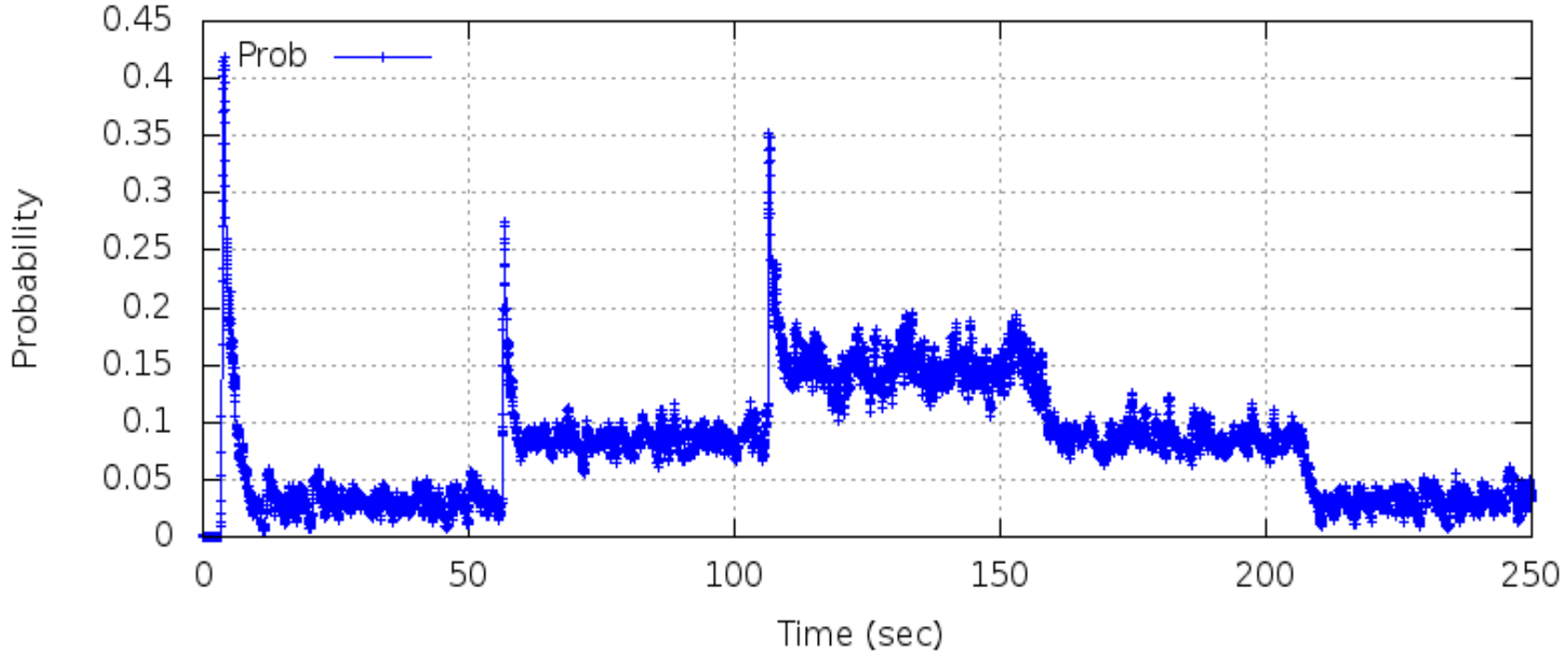
PIE vs. RED Queuing Latency





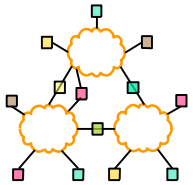
✧ Credit: Rong Pan (Cisco)

PIE Drop Probability



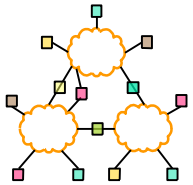
✧ Credit: Rong Pan (Cisco)

Different TCP versions



- Loss based: TCP Reno, Scalable TCP, High speed TCP, …
- Delay based: TCP Vegas, Fast TCP.
- Different Increase/decrease policy
  - AIMD
  - MIMD
  - AIAD
- Fairness: interaction of different TCPs, Usually MIMD, Scalable TCP act aggressively.
  - Late flows get less Bandwidth.
  - AIMD version usually starve when operating with MIMD
  - AIAD get along well with MIMD

Reaching link capacity



- 10 Gbps link and 1500-byte packets and 100ms RTT

Full link utilization =window of 83,333 packets

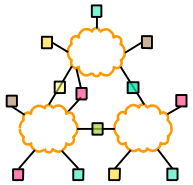
17 RTT or close to 2 sec. in slow start.

- In CA, with cwnd=1 takes

83,332 RTT or 8,333 seconds, more than two hours

- With cwnd/2 almost one hour to reach the link capacity.

HighSpeed-TCP



- More aggressive window increase to get the available BW
  - *cwnd = cwnd + A, in CA*
  - *Ssthresh = cwnd*(1-B) if no triple ack,*
  - *Cwnd = cwnd*(1-B)*

A and B are function of the window size

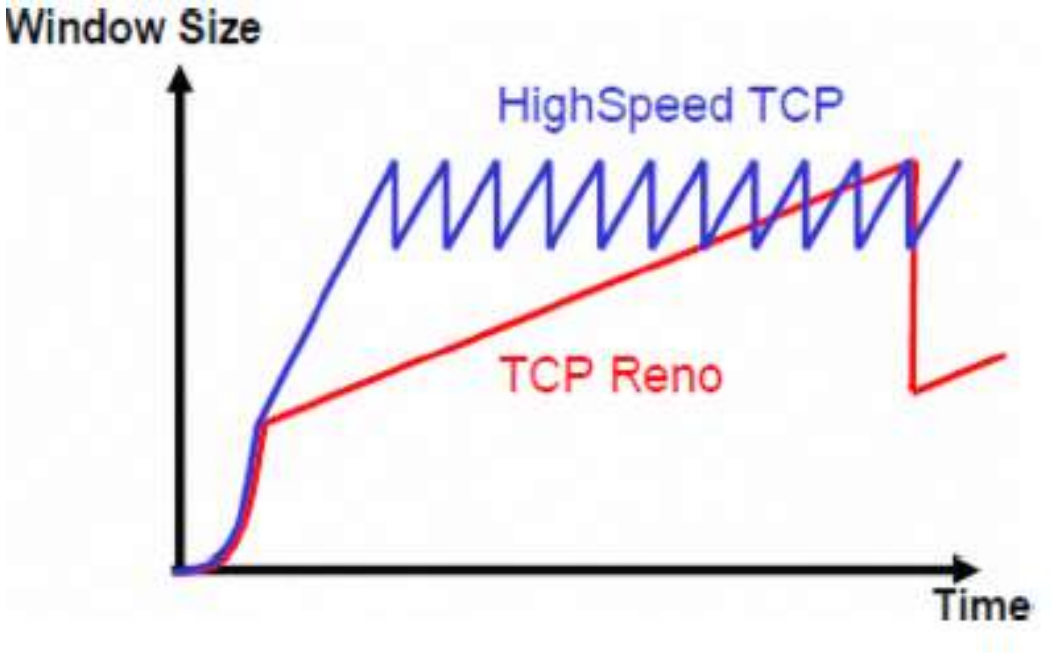
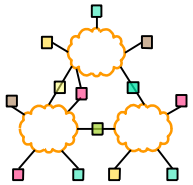


Fig. 2. Congestion Window behavior in HighSpeed-TCP

Scalable-TCP



- more aggressive increase, and less aggressive decrease, A and B the same

if *(cwnd > Wlow) A = 0.01and B 0.125*

otherwise, the sar

as Reno

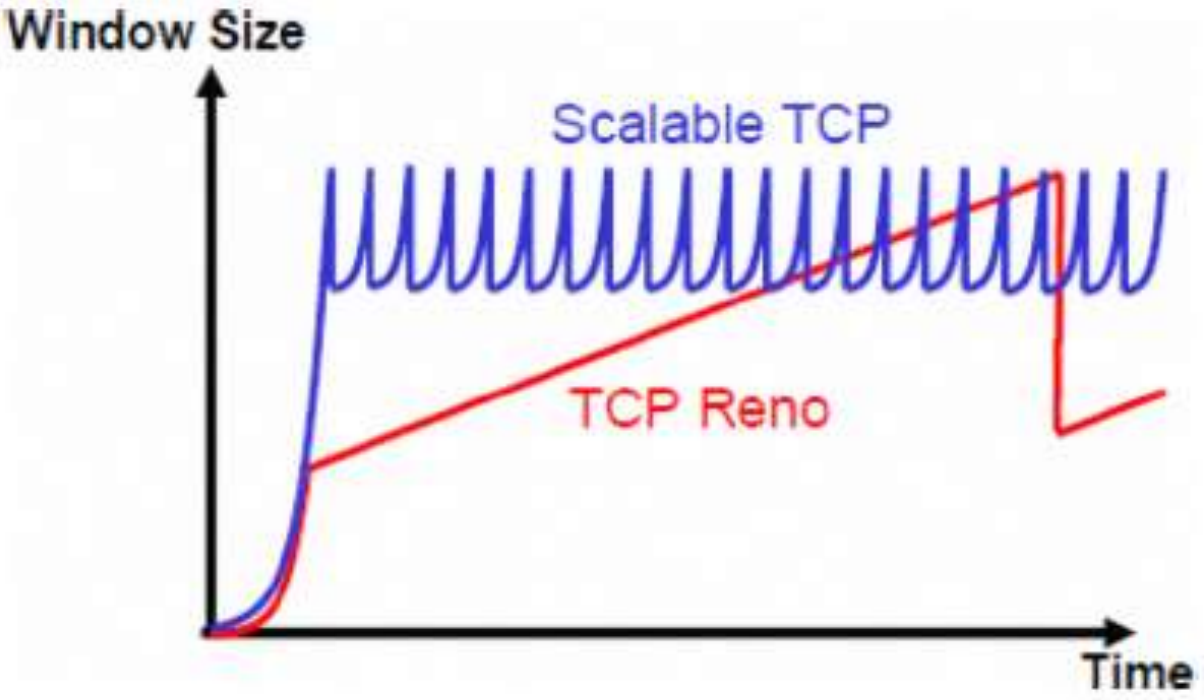
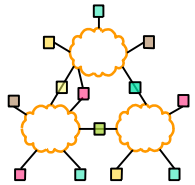


Fig. 3. Congestion Window behavior in Scalable-TCP

TCP-xHS (High-speed)



- Friendly to TCP-Reno especially when sharing high bandwidth links
- Use combination of Reno and Vegas by keeping an extra window

B are like HighSpeed TCP and A Differently.

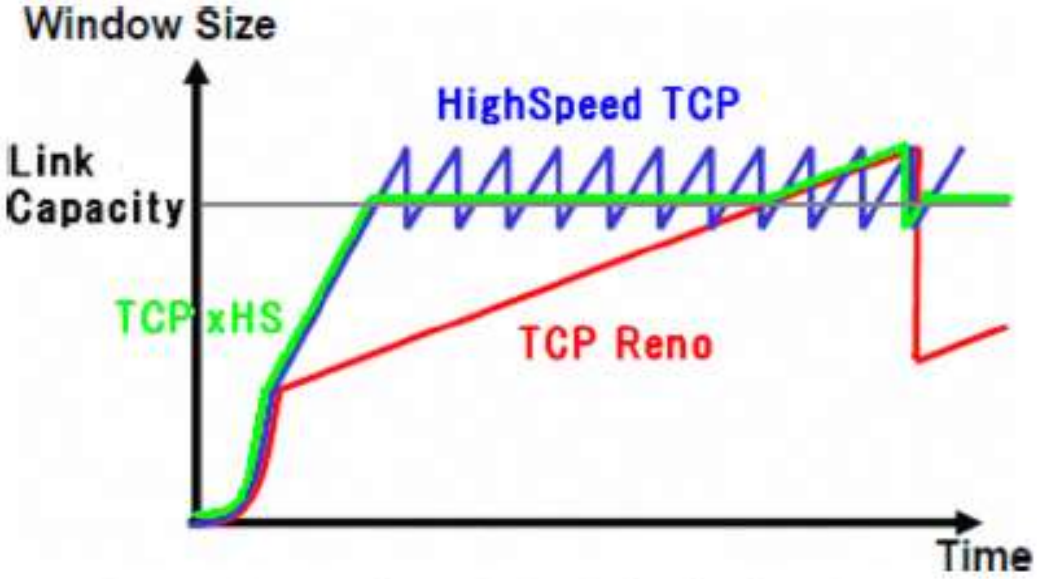
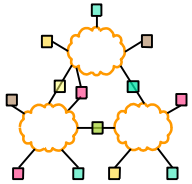
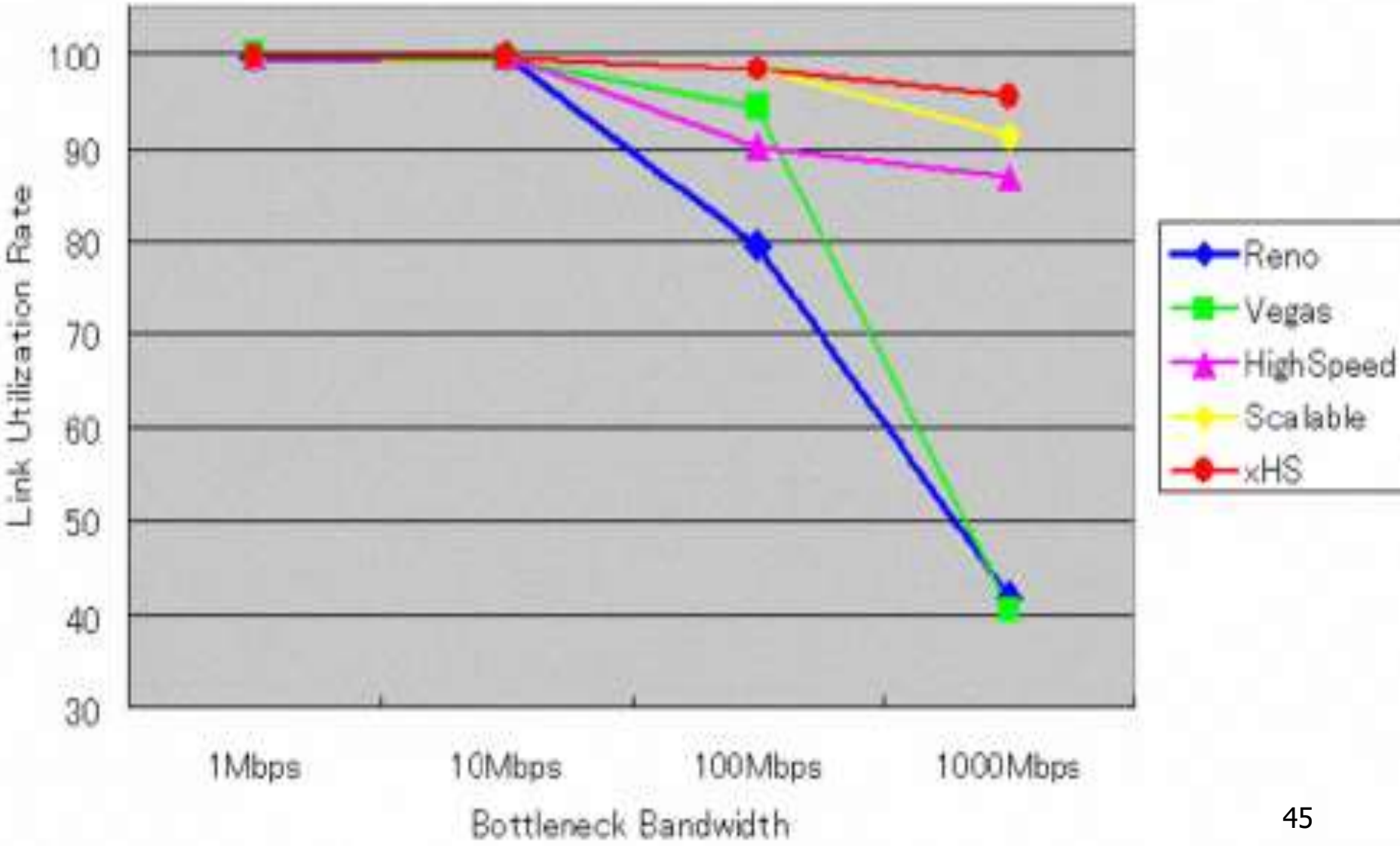


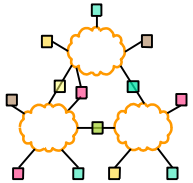
Fig. 4. Congestion window behavior in TCP-xHS

Link utilization





Congestion Windows



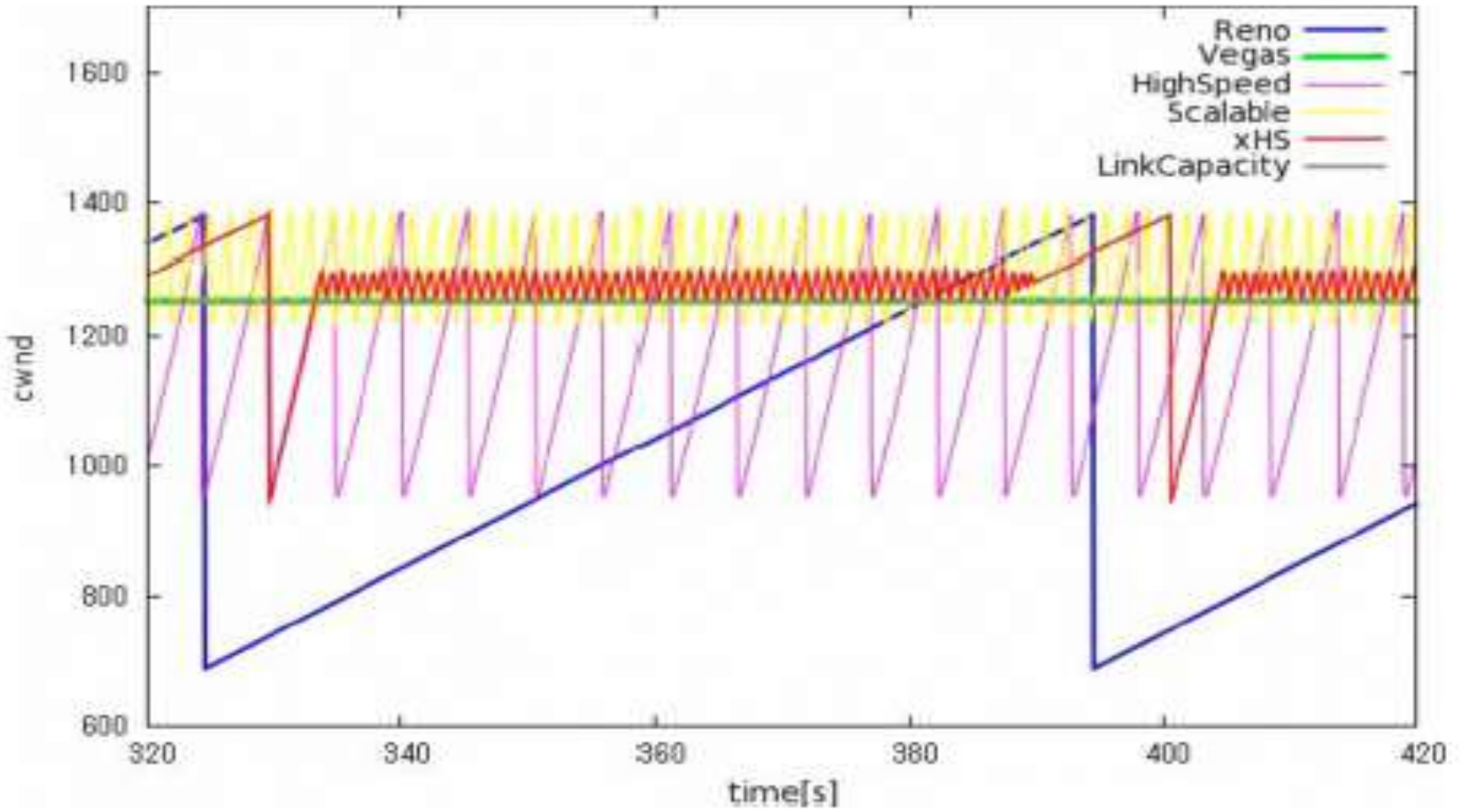
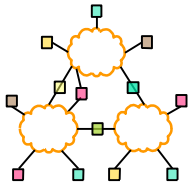


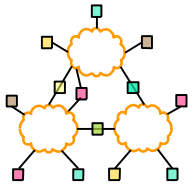
Fig. 7. Congestion window evolution of TCP versions (at 100Mbps)

TCP Friendliness



- Flows with different RTTs sharing the same bottleneck link will not get equal portions of the available bandwidth.
  - Inversely proportional to the ratio of their RTTs.
- Queue management or Active Queue Management, AQM, plays an important role.
- Different Studies, usually simulation under different topologies and conditions.
- Unfairness is especially severe in high-speed variants of TCP

Evaluation Scenario



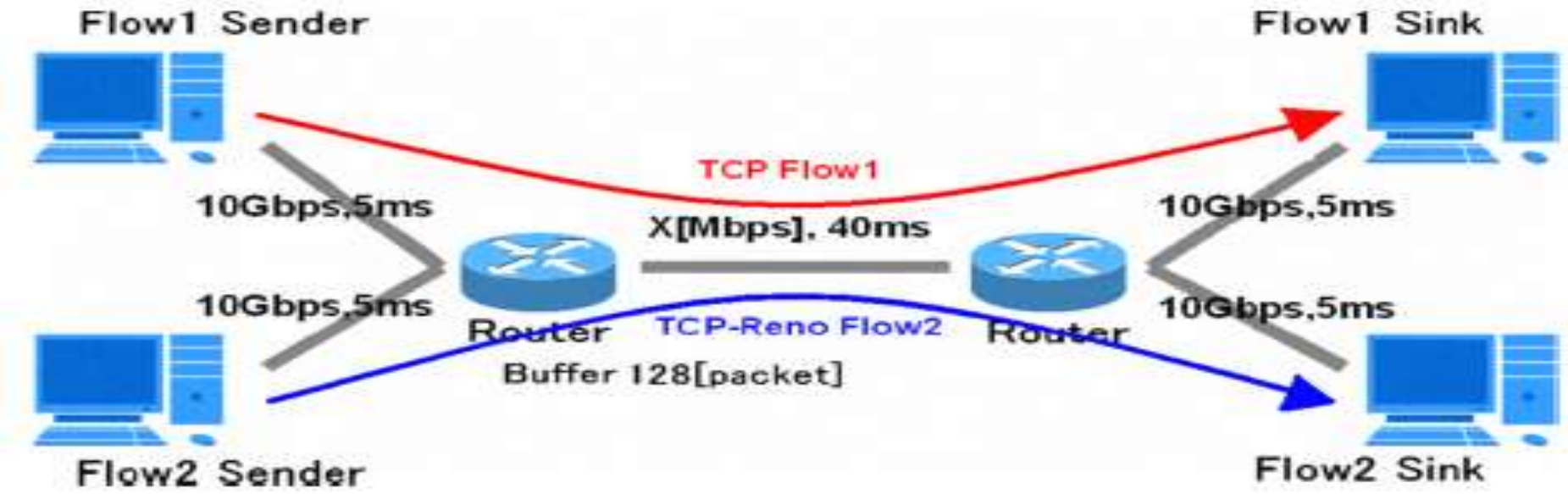
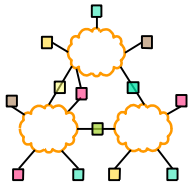
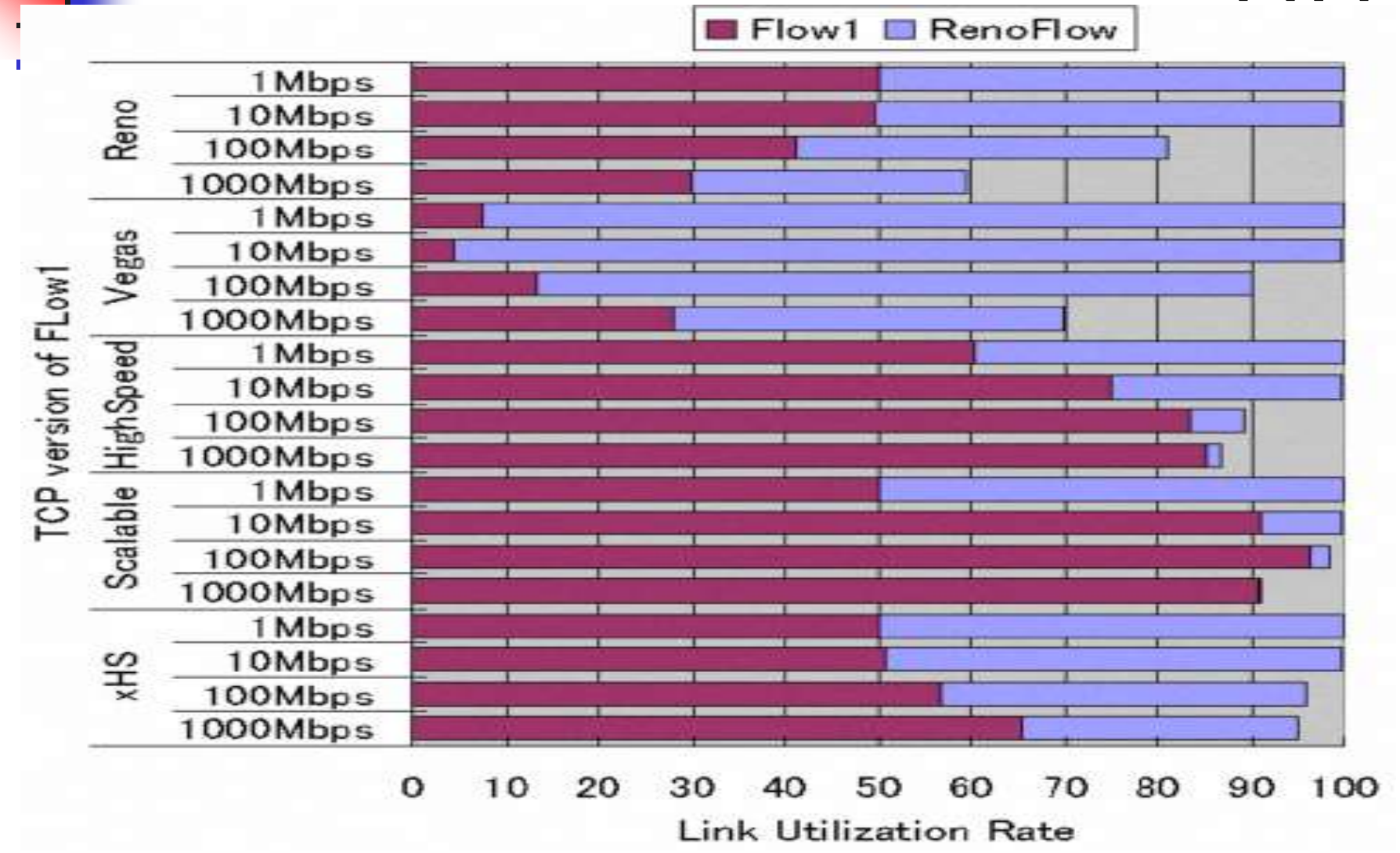


Fig. 8. Network model

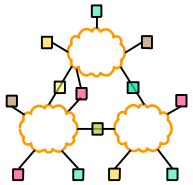
- Flow1 high speed TCP
- Flow2 TCP-Reno connection
- Bottleneck link varies from IMbps, IOMbps, 100Mbps and 1000Mbps

Evaluation Results



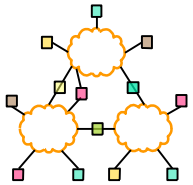


Changing Workloads



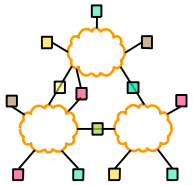
- New applications are changing the way TCP is used
- 1980's Internet
  - Telnet & FTP → long lived flows
  - Well behaved end hosts
  - Homogenous end host capabilities
  - Simple symmetric routing
- 2000's Internet
  - Web & more Web → large number of short xfers
  - Wild west – everyone is playing games to get bandwidth
  - Cell phones and toasters on the Internet
  - Policy routing
- How to accommodate new applications?

Dependence on TCP/IP (cont.)



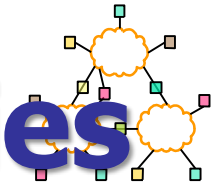
- We depend on TCP at office, home, and while on the move
- We depend on TCP not only for research, but also for critical business transactions and entertainment
- Internet (powered by TCP/IP) is now a key tool used by our kids to prepare their projects at schools

Critical Role of TCP



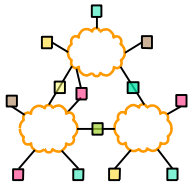
- Many believe that network performance can be boosted by simply upgrading hardware
  - Not correct in many occasions
- TCP sits between application and network
- TCP has total control of how application data should be released to the network
- TCP is a complex protocol which interacts with many network elements in the end to end path
- Unless TCP is optimized, hardware alone cannot boost network performance

Emergence of New Networking Technologies



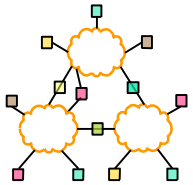
- We are witnessing proliferation of different networking technologies
  - Wireless, satellite, optical etc.
- TCP algorithms suitable for one environment, do not always work best in another
- Need for research into new algorithms
  - Evidence: large number of articles on TCP are published every year in top journals and conferences

IP Convergence



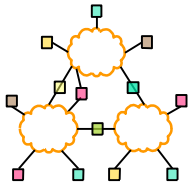
- Many traditional non-TCP/IP industries are converging to TCP/IP
  - E.g. cellular communication, video and other entertainments, etc.
- Understanding TCP/IP performance fundamentals thus becomes important to scientists and engineers working in all these industries

Coping with changes?



- Design New protocol?
- Select base on the application?
  - Very challenging task.
- Model and Optimize the network and protocol.
  - Many parameters and very complicated.
- Learn from the network and adapt it
  - How?
- Transport layer should adapt to whatever?
  - Network does
  - Application wants

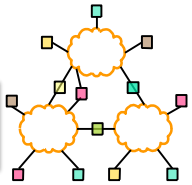
Current TCP Problem

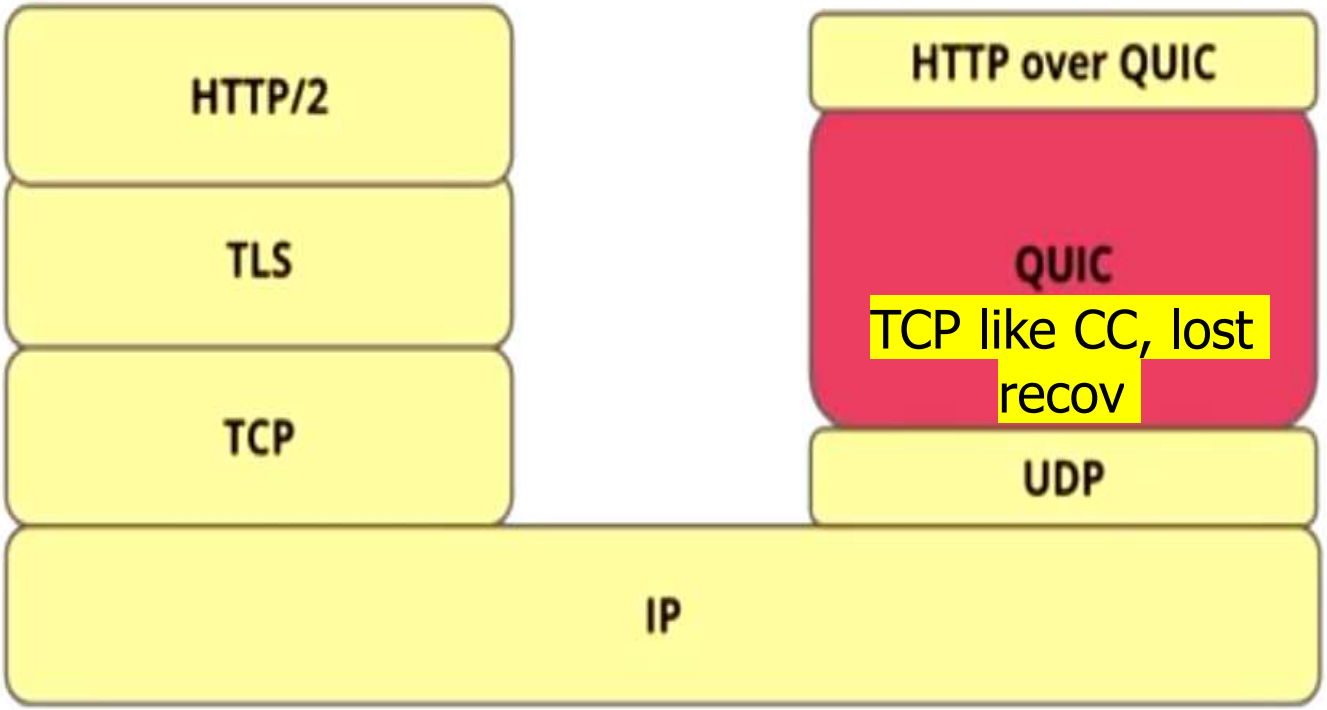


- TCP Head of Line Blocking
- Handshake cost by 3way.
- Slow start and small init win. size
- Increase socket buffer
- NAT timeout and IP roaming
- TCP Buffer Bloat

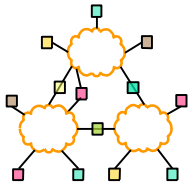
- Need upgrade OS on both end and middle box

Quic Transport protocol

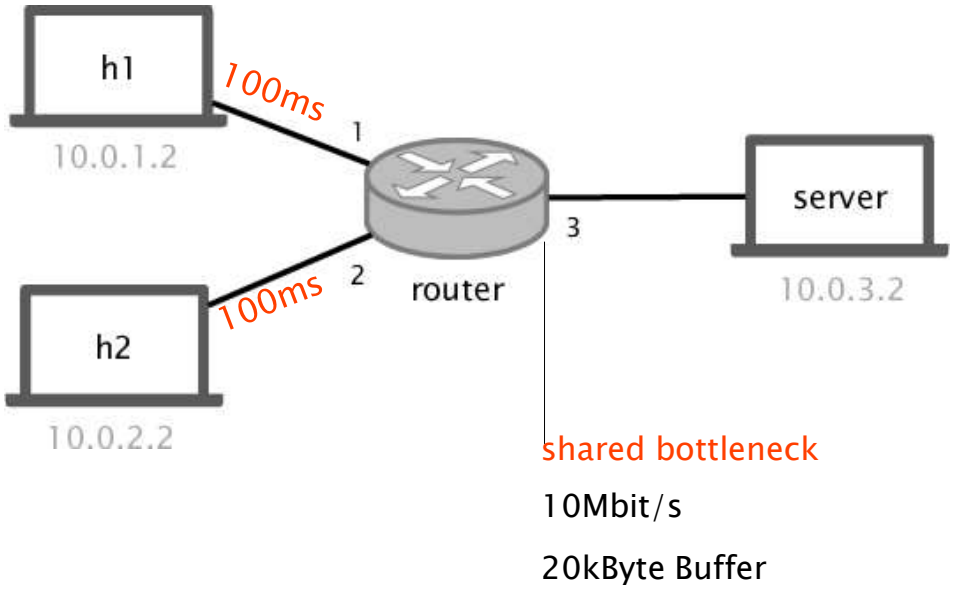




Competing TCP flows

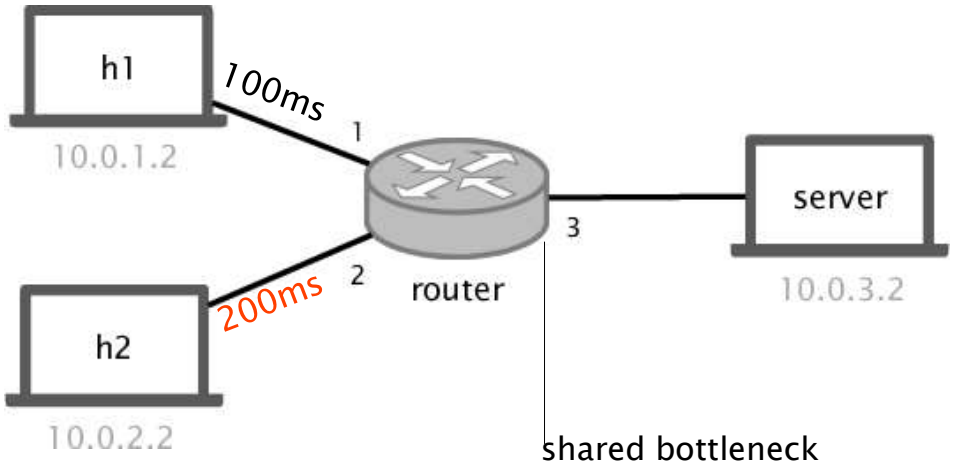


Competing TCP flows should each get a fair share of bandwidth.



Live demo (1/3): two RENO flows on links with equal RTT.

Competing TCP flows

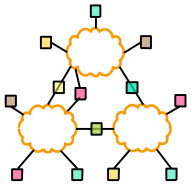


shared bottleneck

10Mbit/s

20kByte Buffer

Live demo recap:



RENO

both flows have 100ms RTT

blue flow RTT changes from 100ms to 200ms

CUBIC

both flows have 100ms RTT

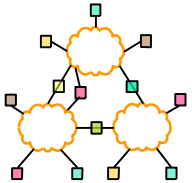
blue flow RTT changes from 100ms to 200ms

CUBIC is fairer if competing flows have different RTTs.

Live demo recap:

CUBIC is fairer if competing flows have different RTTs.

(but it isn't perfect either)



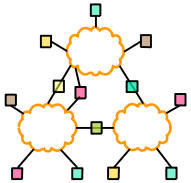
RENO

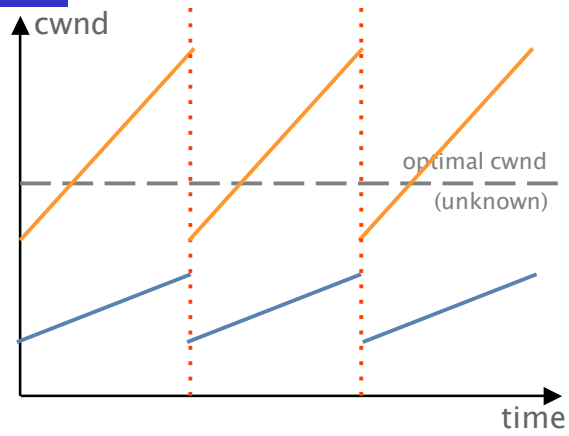
CUBIC

difference between flows less

difference means fairer

proportionally to the path RTT.

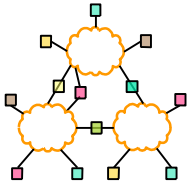


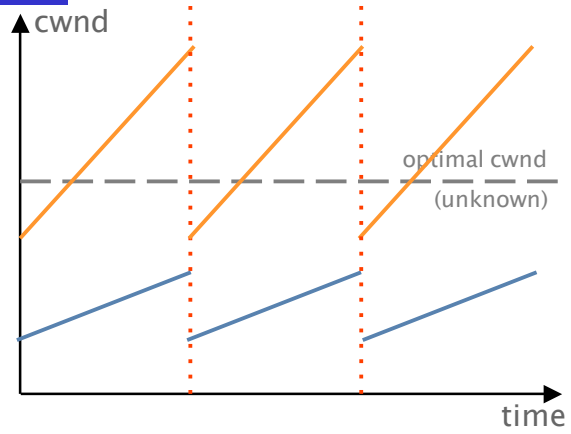


TCP RENO

- Between congestion events, short-RTT flows ramp up quicker.
- long-RTT flows ramp up more slowly.
- on a shared bottleneck, long-RTT flows starve.

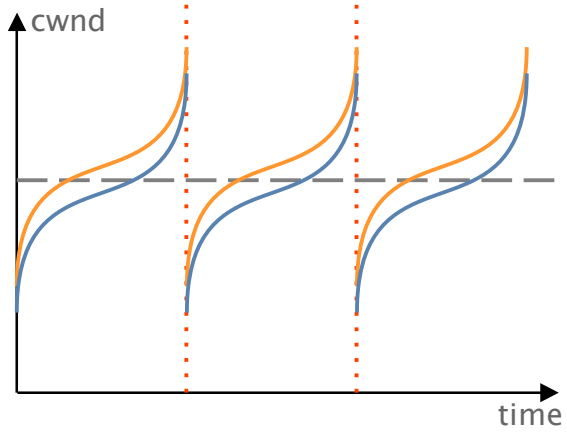
to time.





TCP RENO

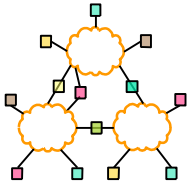
- Between congestion events, short-RTT flows ramp up quicker.
- long-RTT flows ramp up more slowly.
- on a shared bottleneck, long-RTT flows starve.

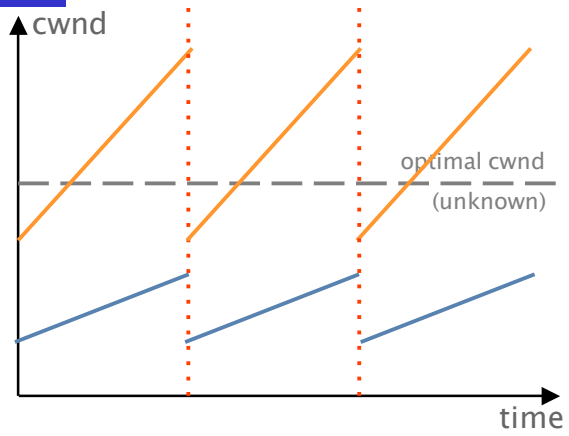


TCP CUBIC

- Between congestion events, cwnd grows with time.
- short- and long-RTT flows grow equally.

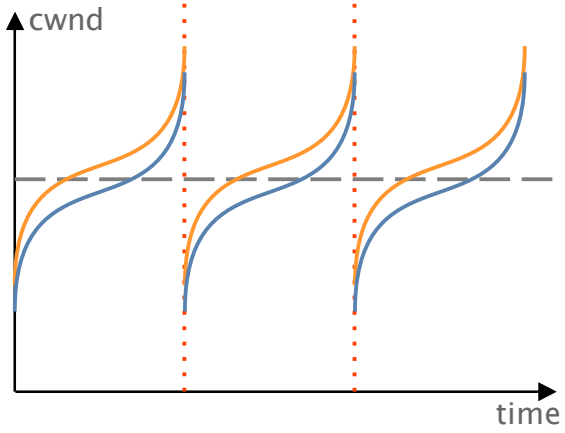
proportionally to time.





TCP RENO

- Between congestion events, short-RTT flows ramp up quicker.
- long-RTT flows ramp up more slowly.
- on a shared bottleneck, long-RTT flows starve.



TCP CUBIC

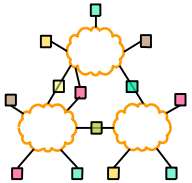
- Between congestion events, cwnd grows with time.
- short- and long-RTT flows grow equally.
- short-RTTs still have some advantage (shorther reaction time).
- CUBIC grows the cwnd always as least as fast as RENO as RENO can be very aggressive if RTTs are low.

Data Center TCP (DCTCP)

Mohammad Alizadeh, Albert Greenberg, David A. Maltz, Jitendra Padhye
Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, Murari Sridharan

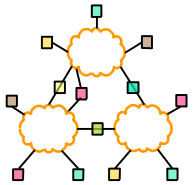
Modified by Feng Xie

Data Center Packet Transport



- Cloud computing service provider
  - Amazon,Microsoft,Google
- Transport **inside** the DC
  - TCP rules (99.9% of traffic)
- How's TCP doing?

TCP in the Data Center

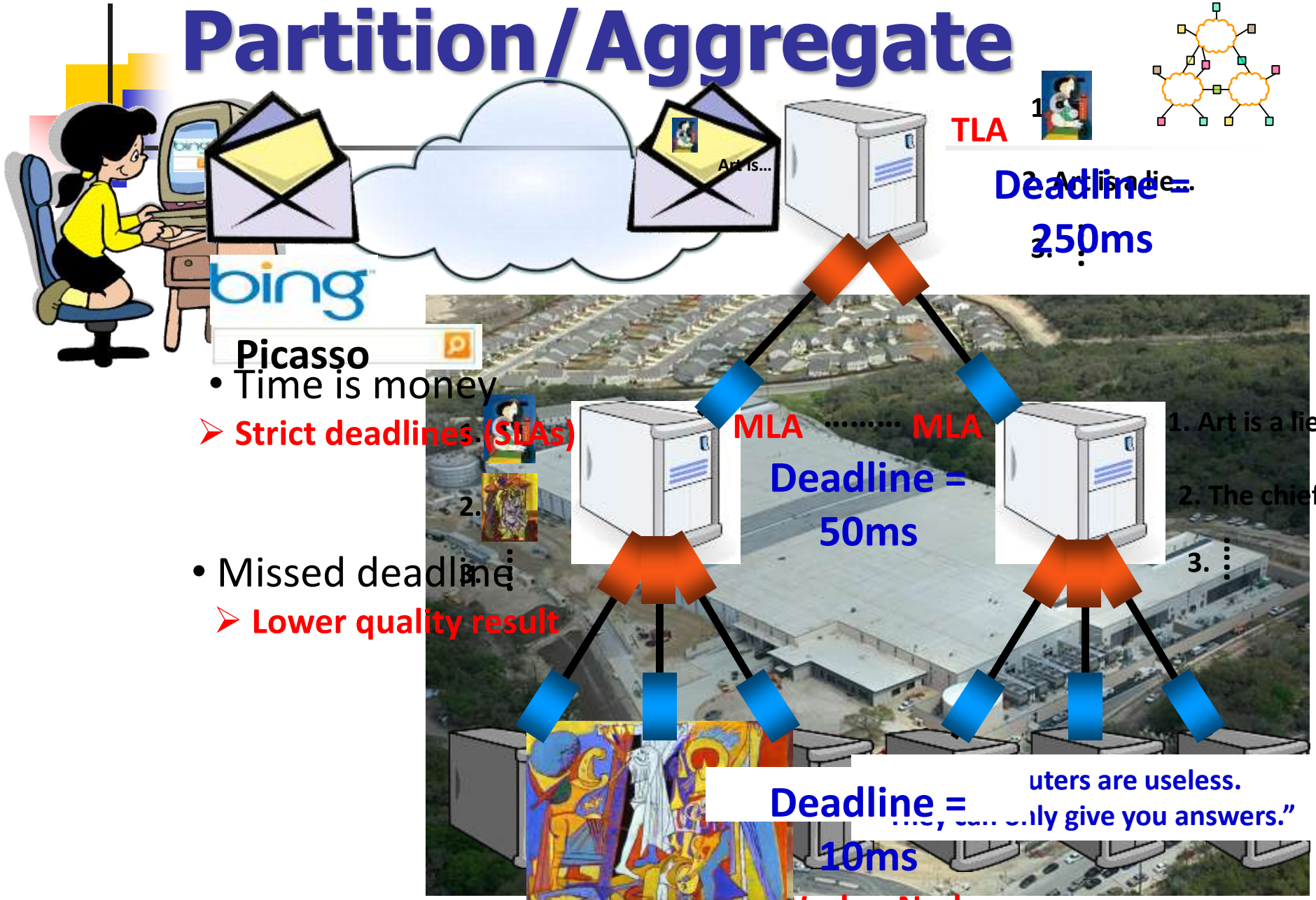


- TCP does not meet demands of apps.
  - Incast
    - Suffers from bursty packet drops
    - Not fast enough utilize spare bandwidth
  - Builds up large queues:
    - Adds significant latency.
    - Wastes precious buffers, esp. bad with shallow-buffered switches.
- Operators work around TCP problems.
  - Ad-hoc, inefficient, often expensive solutions
- Our solution: **Data Center TCP**

Case Study: Microsoft Bing

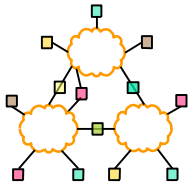
- Measurements from 6000 server production cluster
- Instrumentation passively collects logs
  - Application-level
  - Socket-level
  - Selected packet-level
- More than **150TB** of compressed data over a month

Partition/Aggregate



- Time is money
  - Strict deadlines (SLAs)
- Missed deadline
  - Lower quality result

Workloads



- Partition/Aggregate → Delay-sensitive

  (Query)

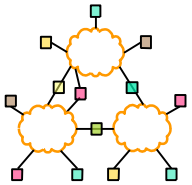
- Short messages [50KB-1MB] → Delay-sensitive

  (Coordination, Control state)

- Large flows [1MB-50MB] → Throughput-sensitive

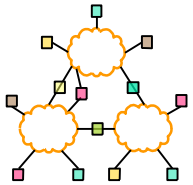
  (Data update)

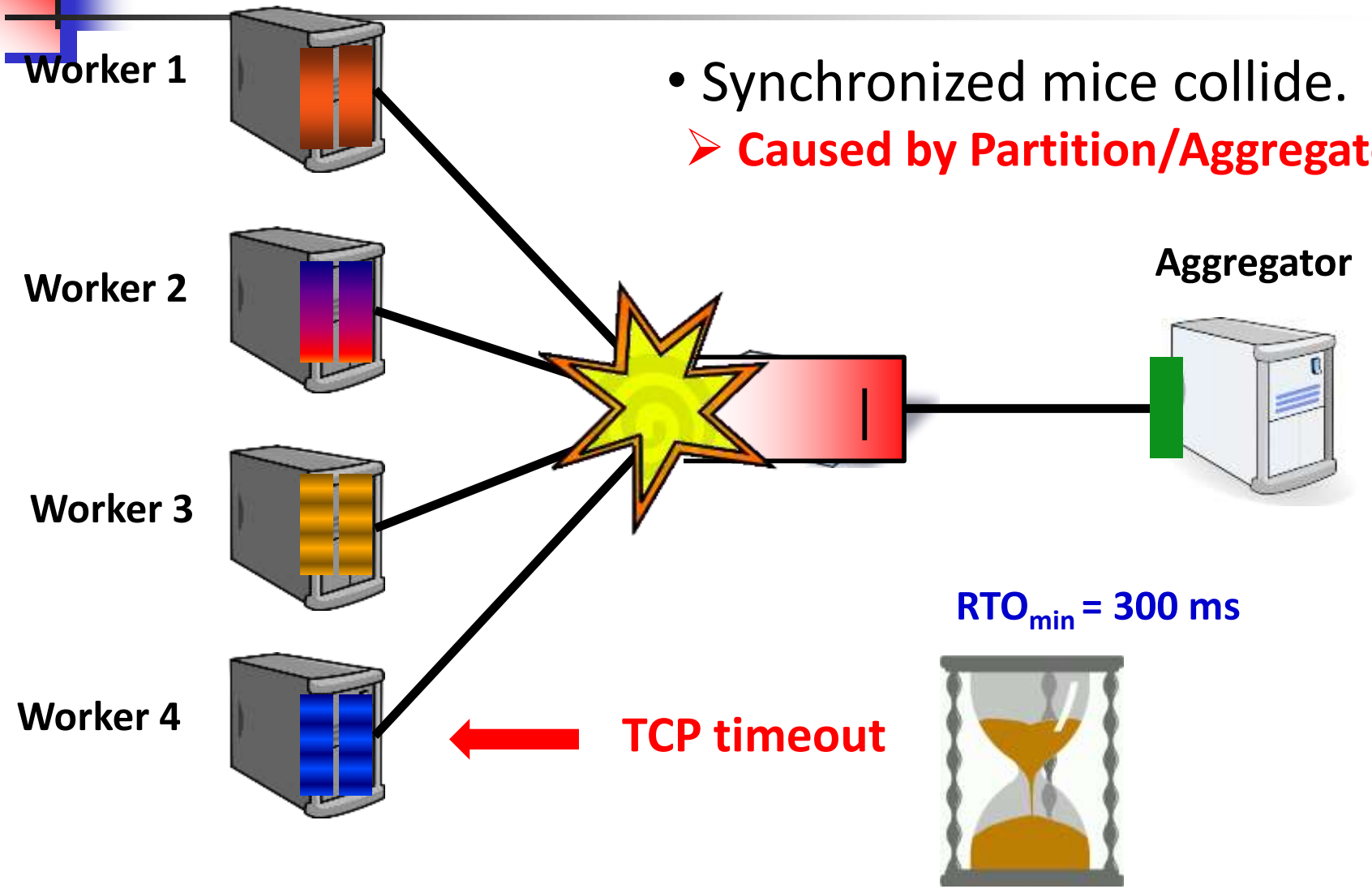
Impairments



- Incast
- Queue Buildup
- Buffer Pressure

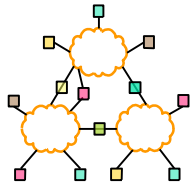
Incast

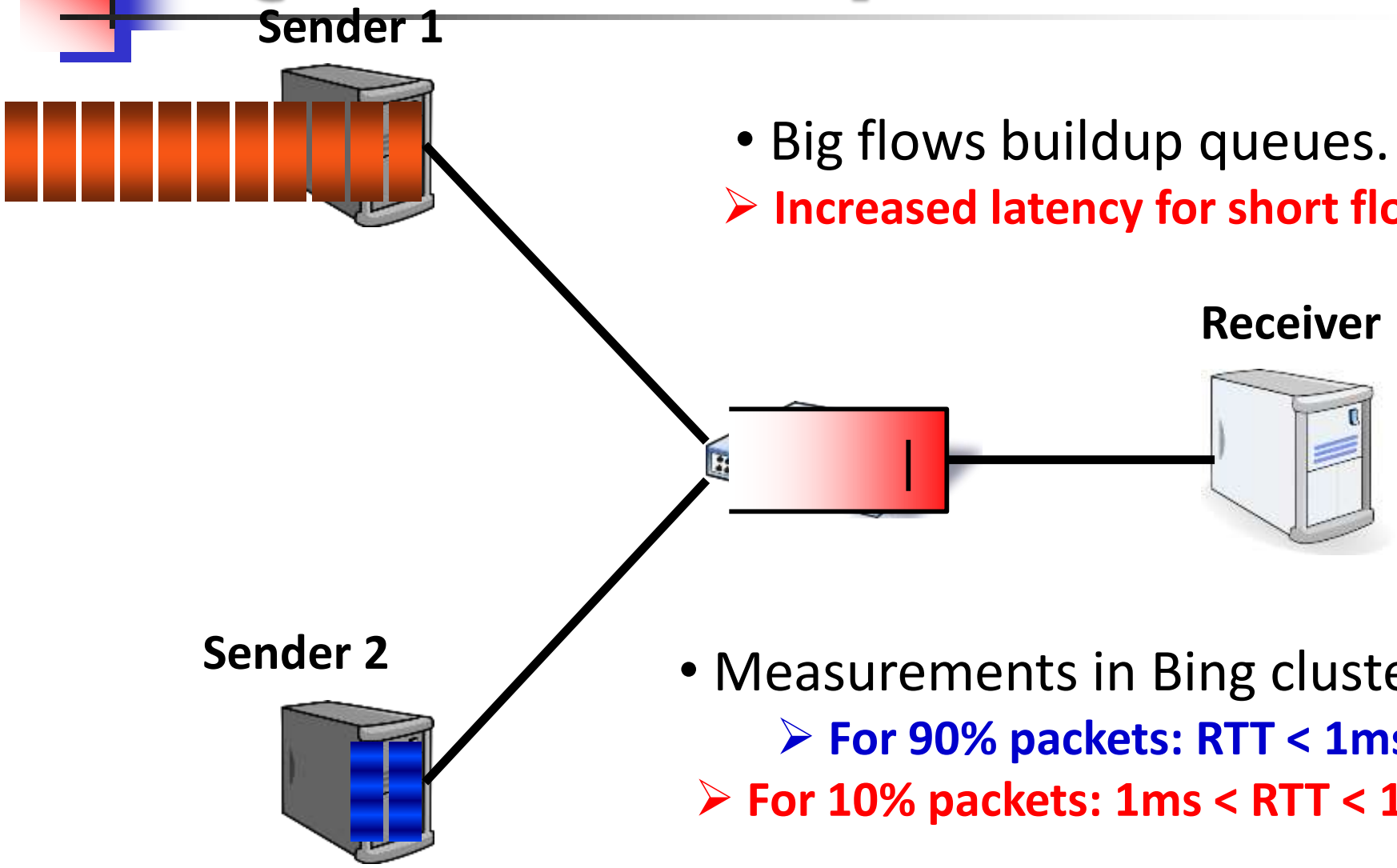




- Synchronized mice collide.
  - ➢ **Caused by Partition/Aggregate.**

Queue Buildup

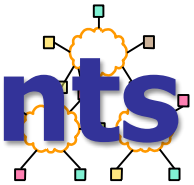




- Big flows buildup queues.
  - ➢ **Increased latency for short flows.**

- Measurements in Bing cluster
  - ➢ **For 90% packets: RTT < 1ms**
  - ➢ **For 10% packets: 1ms < RTT < 15ms**

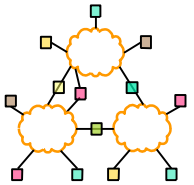
DC Transport Requirements



1. **High Burst Tolerance**
   - Incast due to Partition/Aggregate is common.
2. **Low Latency**
   - Short flows, queries
3. **High Throughput**
   - Large file transfers

The challenge is to achieve these three together.

Balance Between Requirements



High Throughput
High Burst Tolerance

Low Latency

Deep Buffers:

- Queuin
  Increas

Shallow Buffers:

Bursts &
oughput

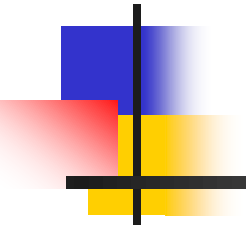
Objective:
Low Queue Occupancy & High Throughput

Reduced RTO_{min} (SIGCOMM '09)

- Doesn't Help Latency

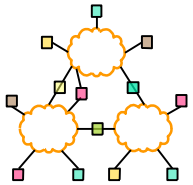
AQM – RED:

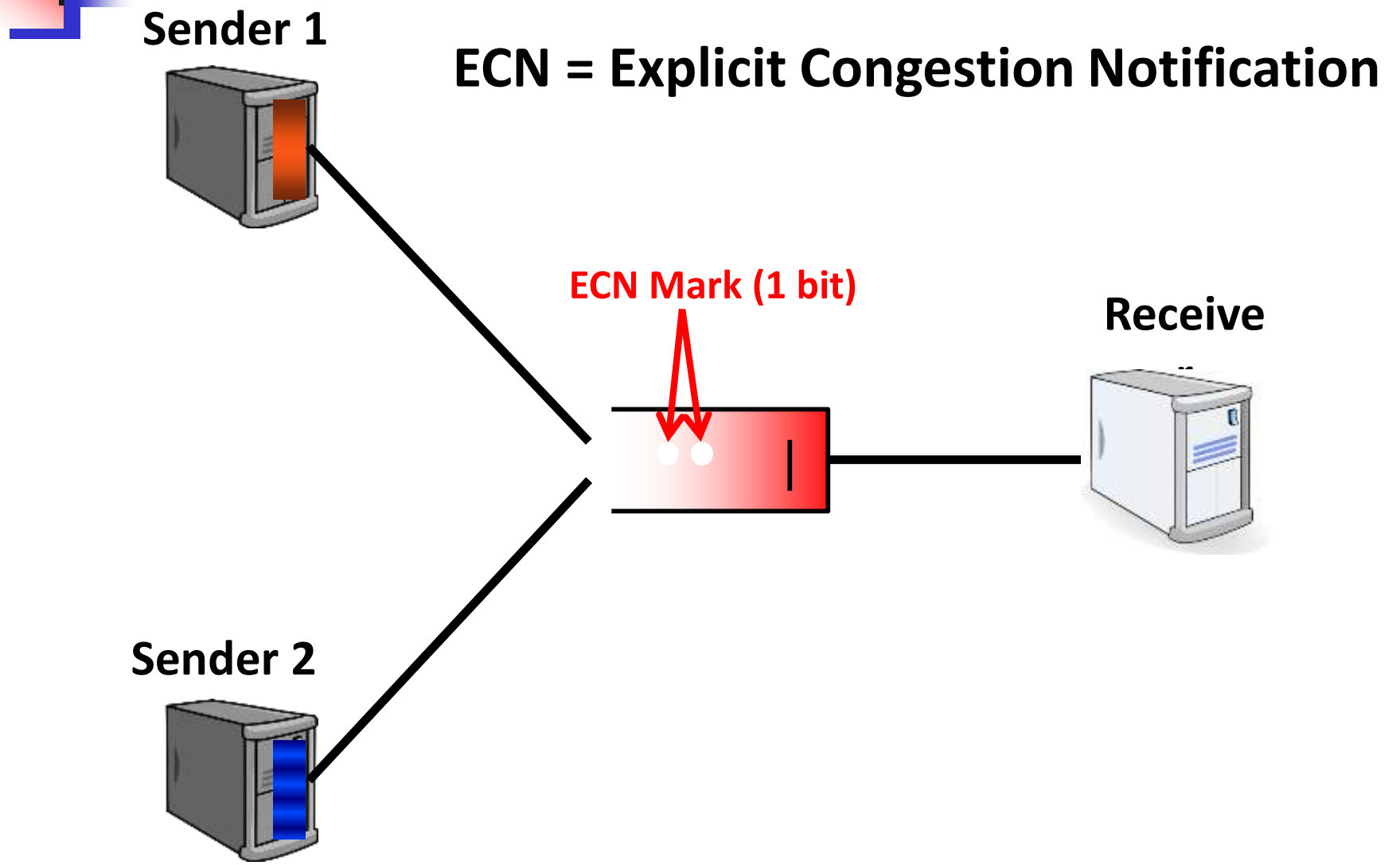
- Avg Queue Not Fast Enough for Incast



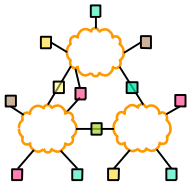
The DCTCP Algorithm

Review: The TCP/ECN Control Loop





Two Key Ideas



1. React in proportion to the **extent** of congestion, not its **presence**.
   - ✓ Reduces **variance** in sending rates, lowering queuing requirements.

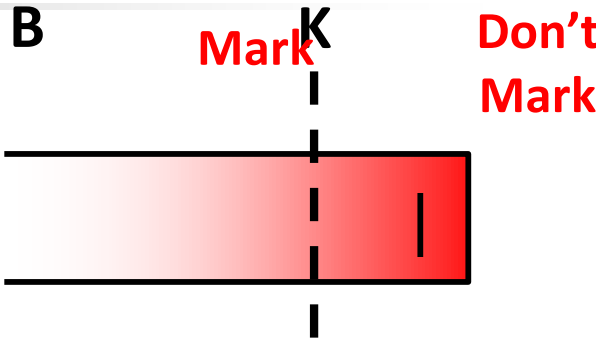
ECN Marks	TCP	DCTCP
1 0 1 1 1 1 0 1 1 1	Cut window by 50%	Cut window by 40%
0 0 0 0 0 0 0 0 0 1	Cut window by 50%	Cut window by 5%

2. Mark based on **instantaneous** queue length.
   - ✓ Fast feedback to better deal with bursts.

Data Center TCP Algorithm

Switch side:

- Mark packets when **Queue Length > K.**



Sender side:

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

In each RTT:

$$F = \frac{\#\ of\ marked\ ACKs}{Total\ \#\ of\ ACKs}$$

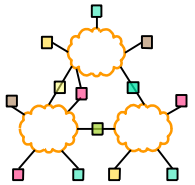
$$\alpha \leftarrow (1-g)\alpha + gF$$

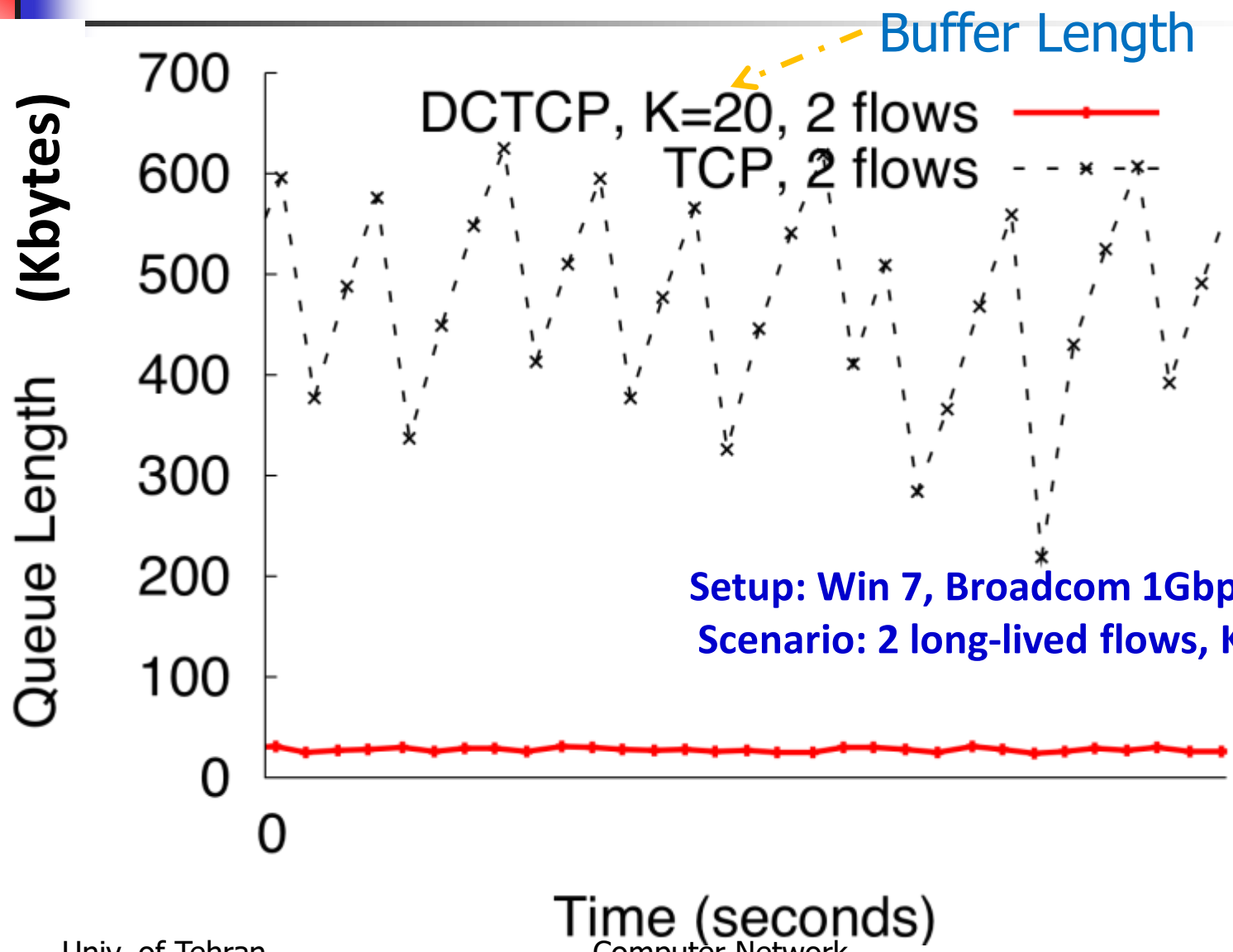
➢ **Adaptive window decreases:**

$$Cwnd \leftarrow (1-\frac{\alpha}{2})Cwnd$$

– Note: decrease factor between 1 and 2.

DCTCP in Action

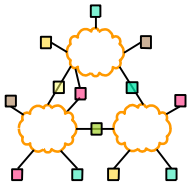




Setup: Win 7, Broadcom 1Gbps Switch

Scenario: 2 long-lived flows, K = 30KB

Why it Works

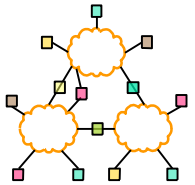


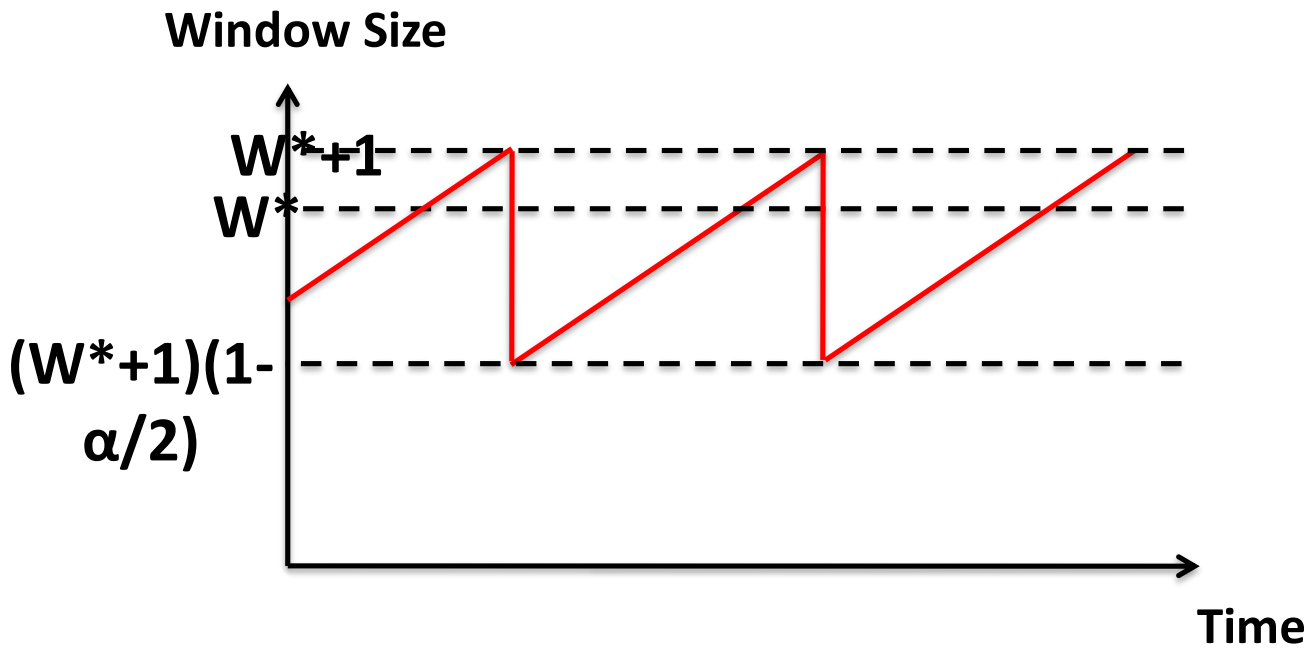
1. **High Burst Tolerance**
   - ✓ **Large buffer headroom** → bursts fit.
   - ✓ **Aggressive marking** → sources react before packets are dropped.

2. **Low Latency**
   - ✓ **Small buffer occupancies** → low queuing delay.

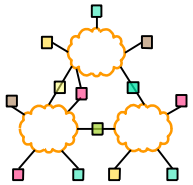
3. **High Throughput**
   - ✓ **ECN averaging** → smooth rate adjustments, cwind low variance.

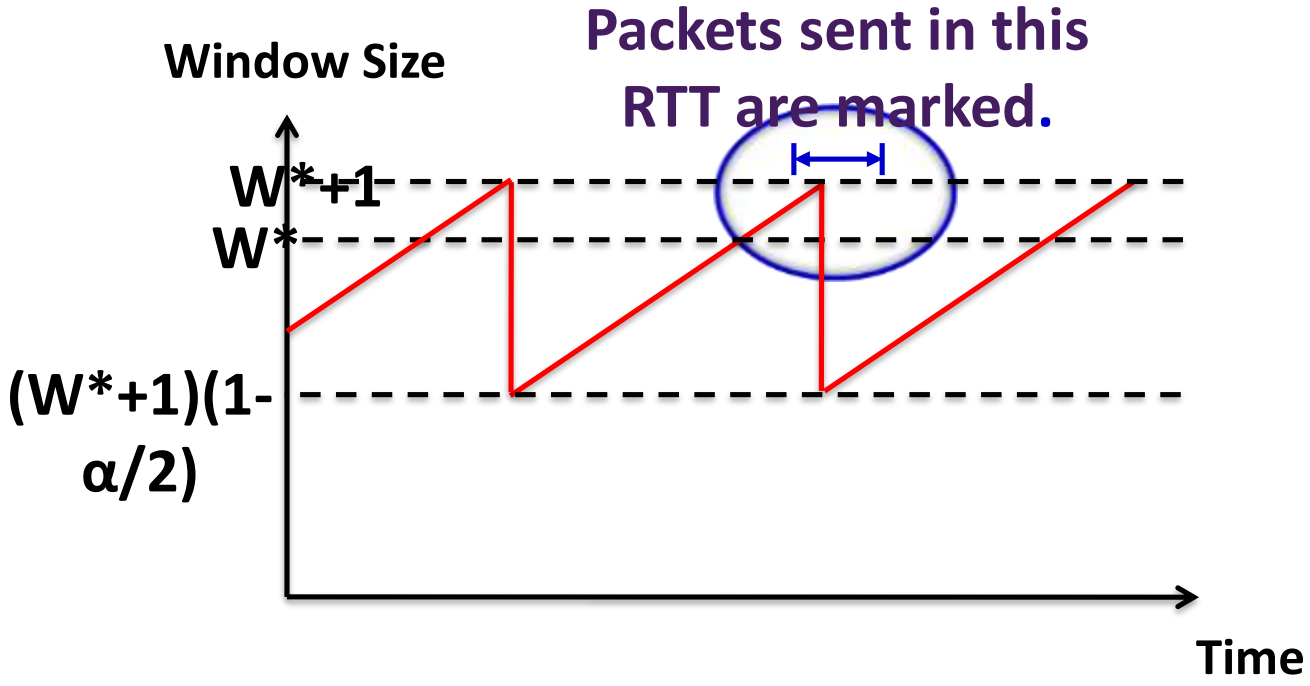
Analysis



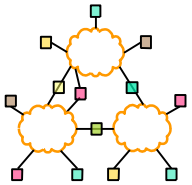


Analysis




Window Size
Packets sent in this RTT are marked.
W*+1
W*
(W*+1)(1-α/2)
Time

Analysis

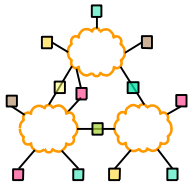


- How low can DCTCP maintain queues without loss of throughput?
- How do we set the DCTCP parameters?
  - ➢ **Need to quantify queue size oscillations (Stability).**

$$K > \frac{1}{7} C \times RTT$$

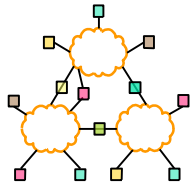
85% Less Buffer than TCP

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 benchmarks
  - **Throughput and Queue Length**
  - **Multi-hop**
  - **Queue Buildup**
  - **Buffer Pressure**
  - **Fairness and Convergence**
  - **Incast**
  - **Static vs Dynamic Buffer Mgmt**

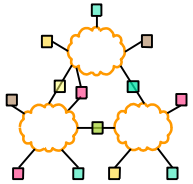
Experiment implement



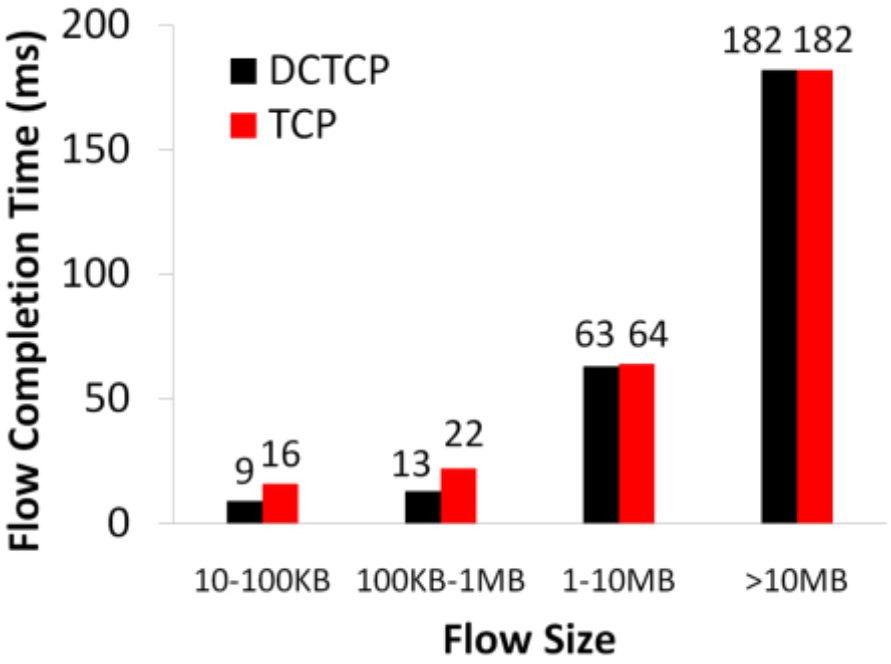
- 45 1G servers connected to a Triumph, a 10G server extern connection
  - 1Gbps links K=20
  - 10Gbps link K=65
- Generate query, and background traffic
  - 10 minutes, 200,000 background, 188,000 queries
- Metric:
  - Flow completion time for queries and background flows.

We use RTO_{min} = 10ms for both TCP & DCTCP.

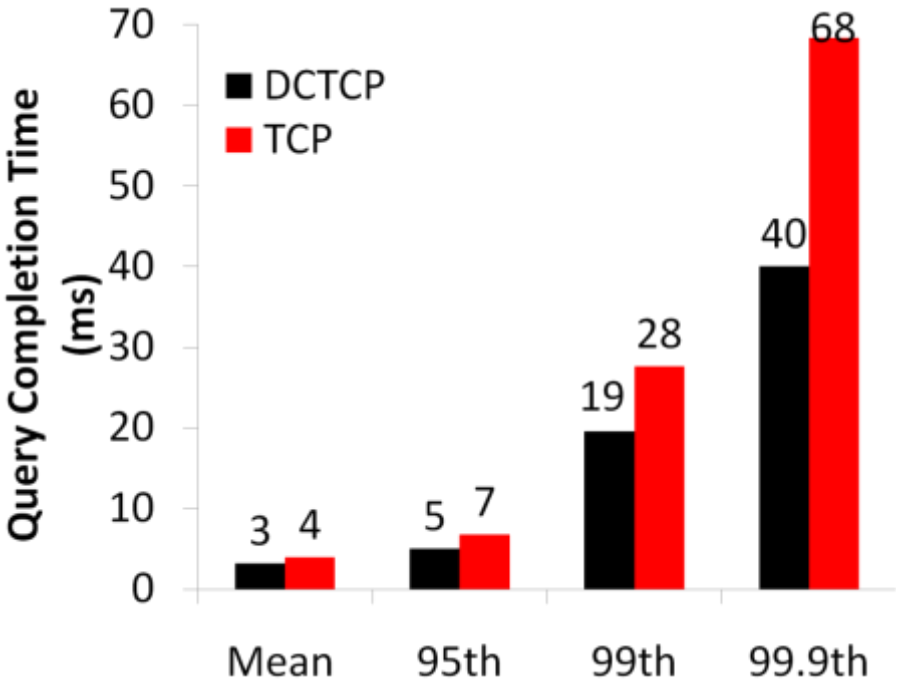
Baseline



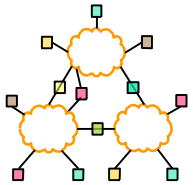
Background Flows



Query Flows

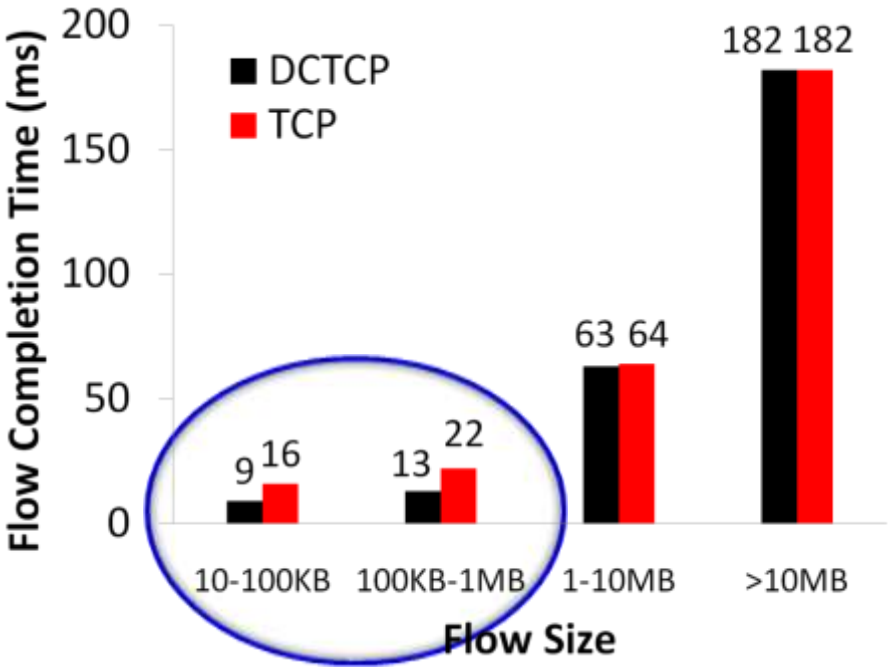


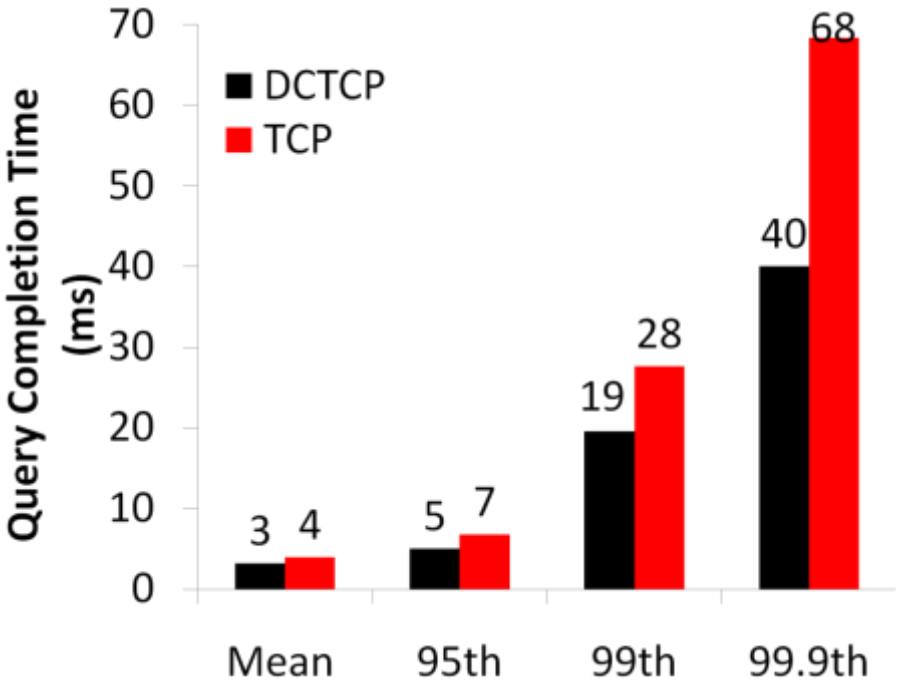
Baseline



Background Flows

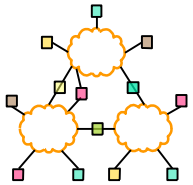
Query Flows



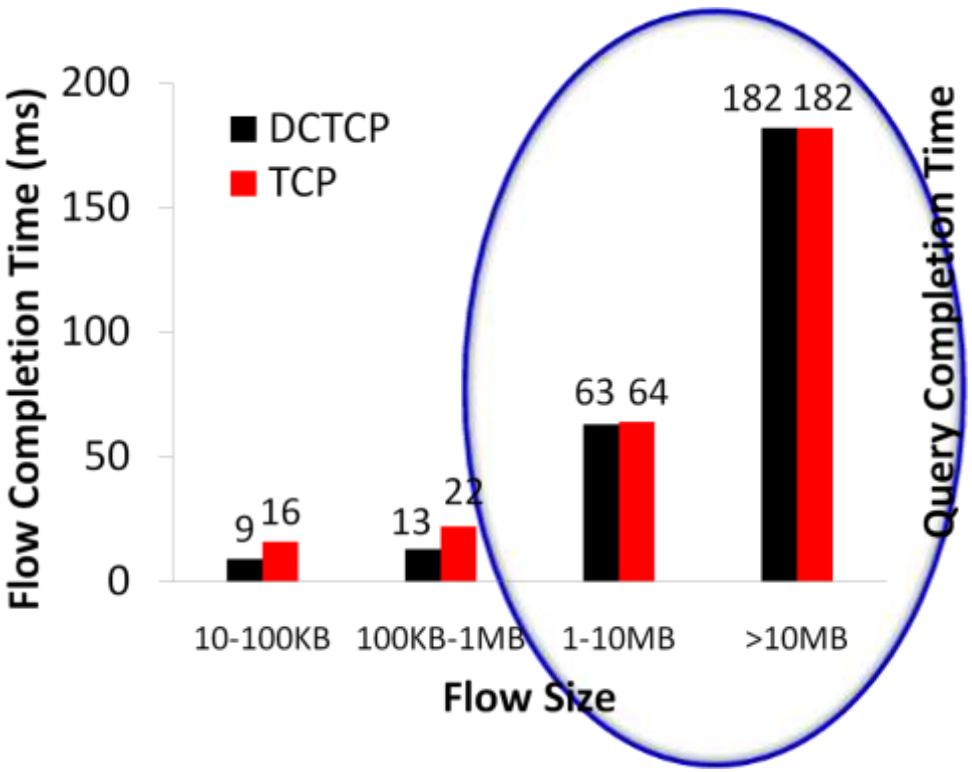


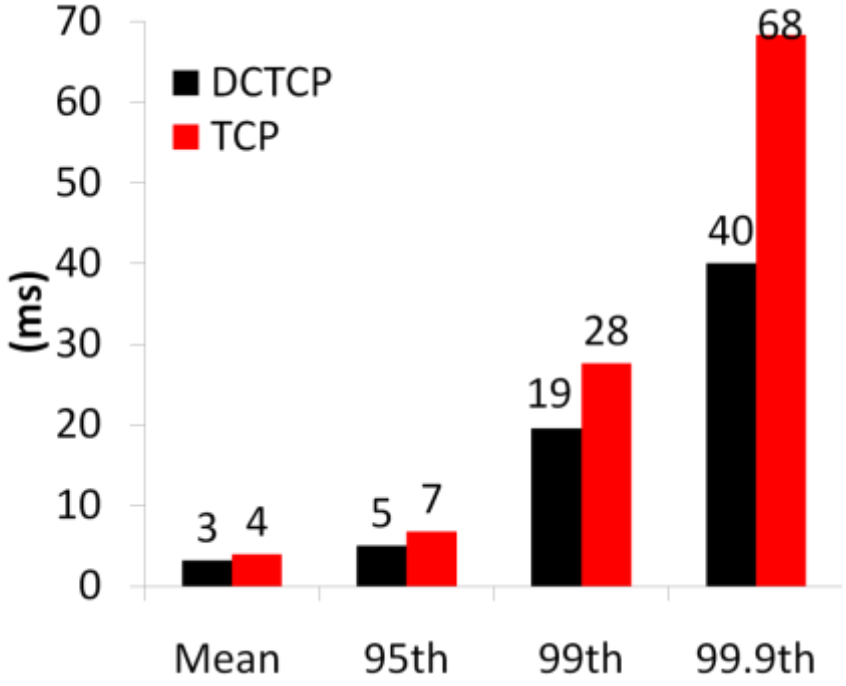
✓ Low latency for short flows.

Baseline



Background Flows **Query Flows**

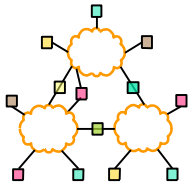




✓ **Low latency for short flows.**

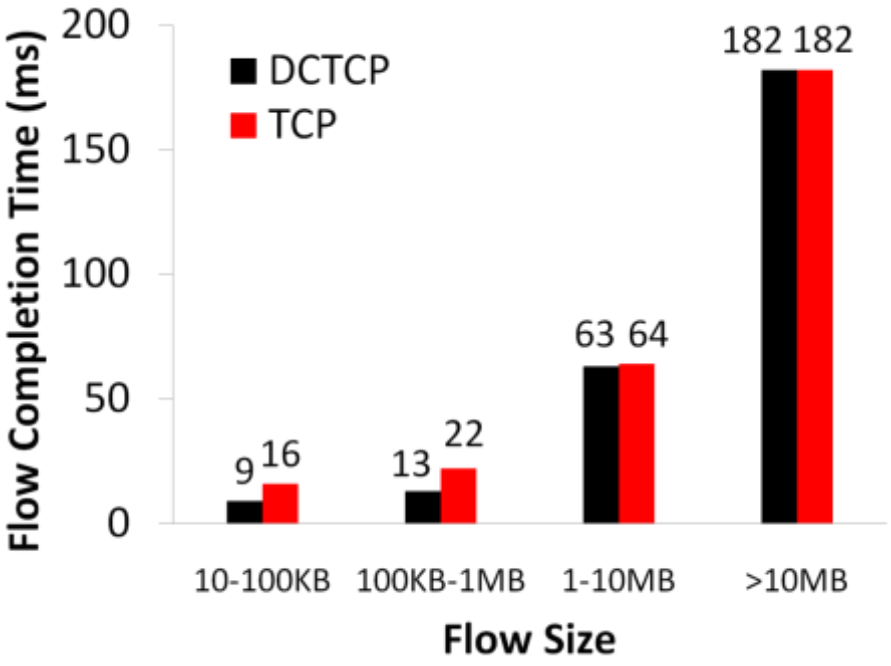
✓ **High throughput for long flows.**

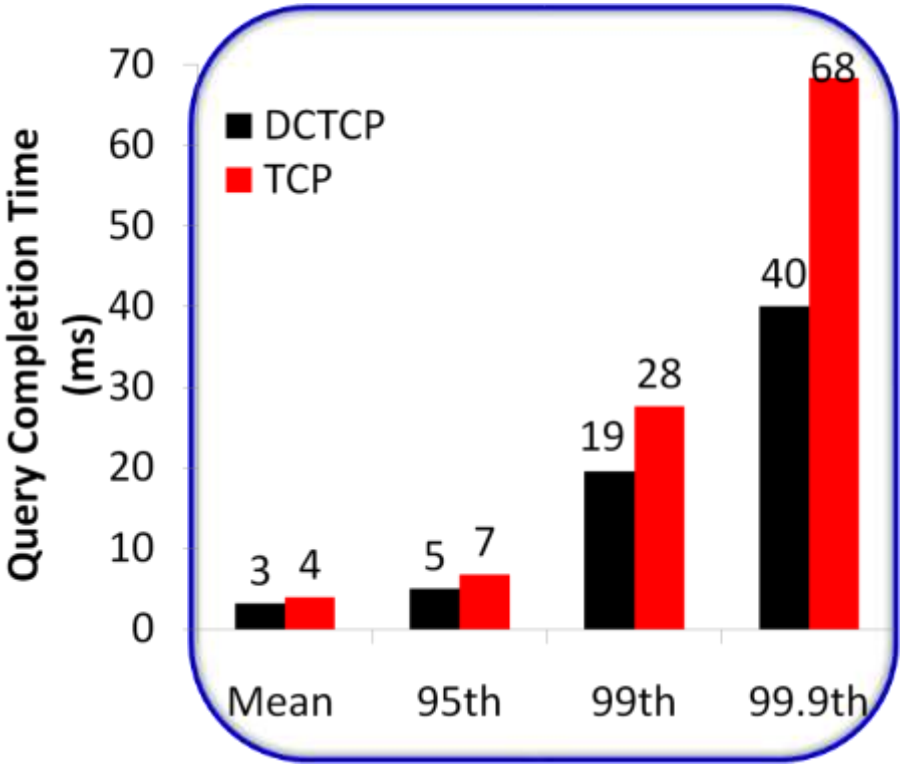
Baseline



Background Flows

Query Flows





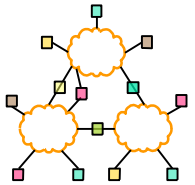
✓ Low latency for short flows.

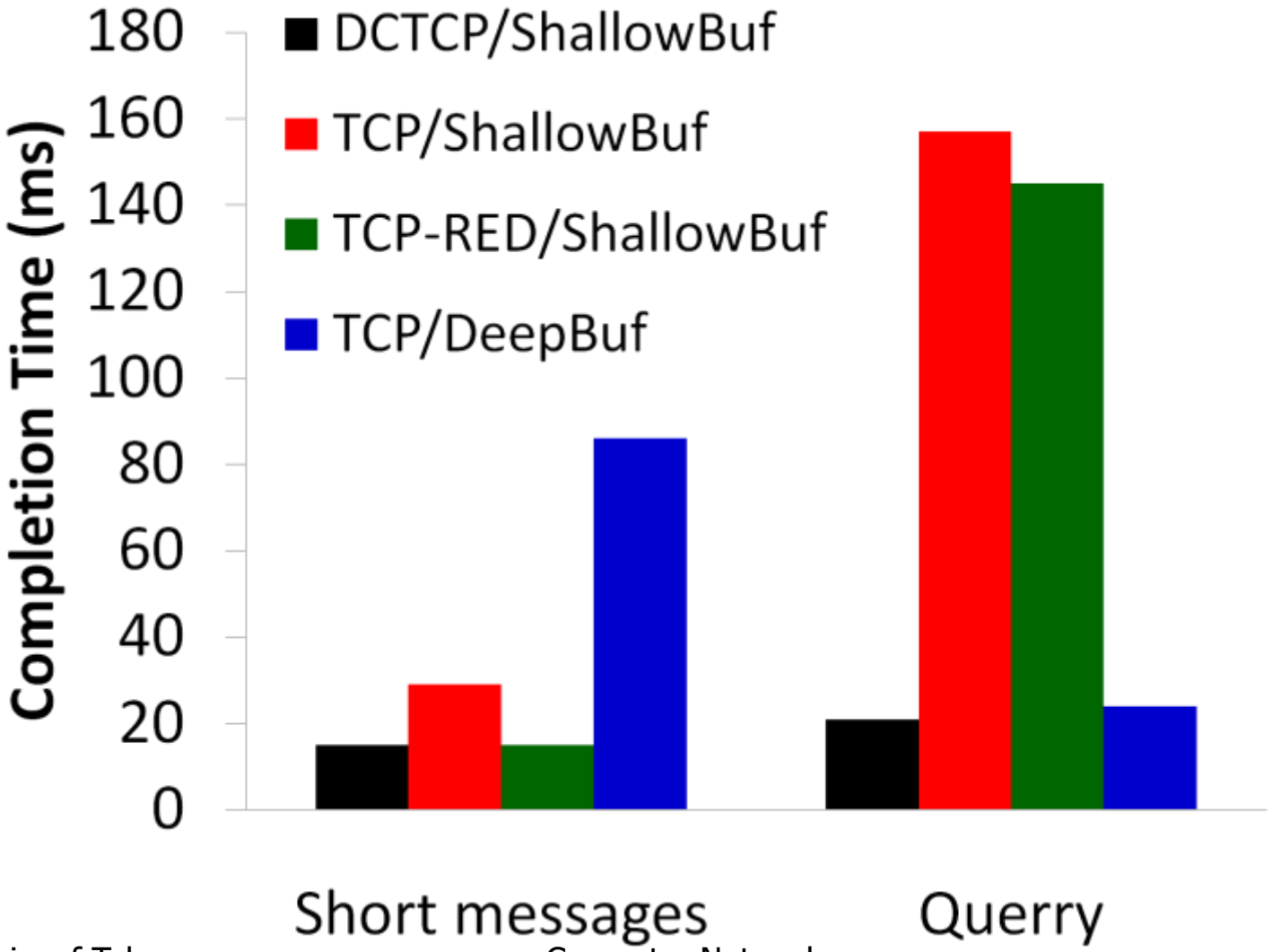
✓ High throughput for long flows.

✓ High burst tolerance for query flows.

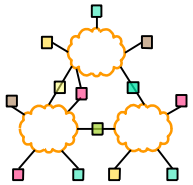
Scaled Background & Query

10x Background, 10x Query



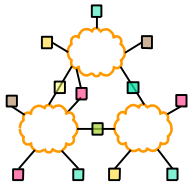


DCTCPConclusions



- DCTCP satisfies all our requirements for Data Center packet transport.
  - ✓ **Handles bursts well**
  - ✓ **Keeps queuing delays low**
  - ✓ **Achieves high throughput**
- Features:
  - ✓ **Very simple change to TCP and a single switch parameter K.**
  - ✓ **Based on ECN mechanisms already available in commodity switch.**

New Facts

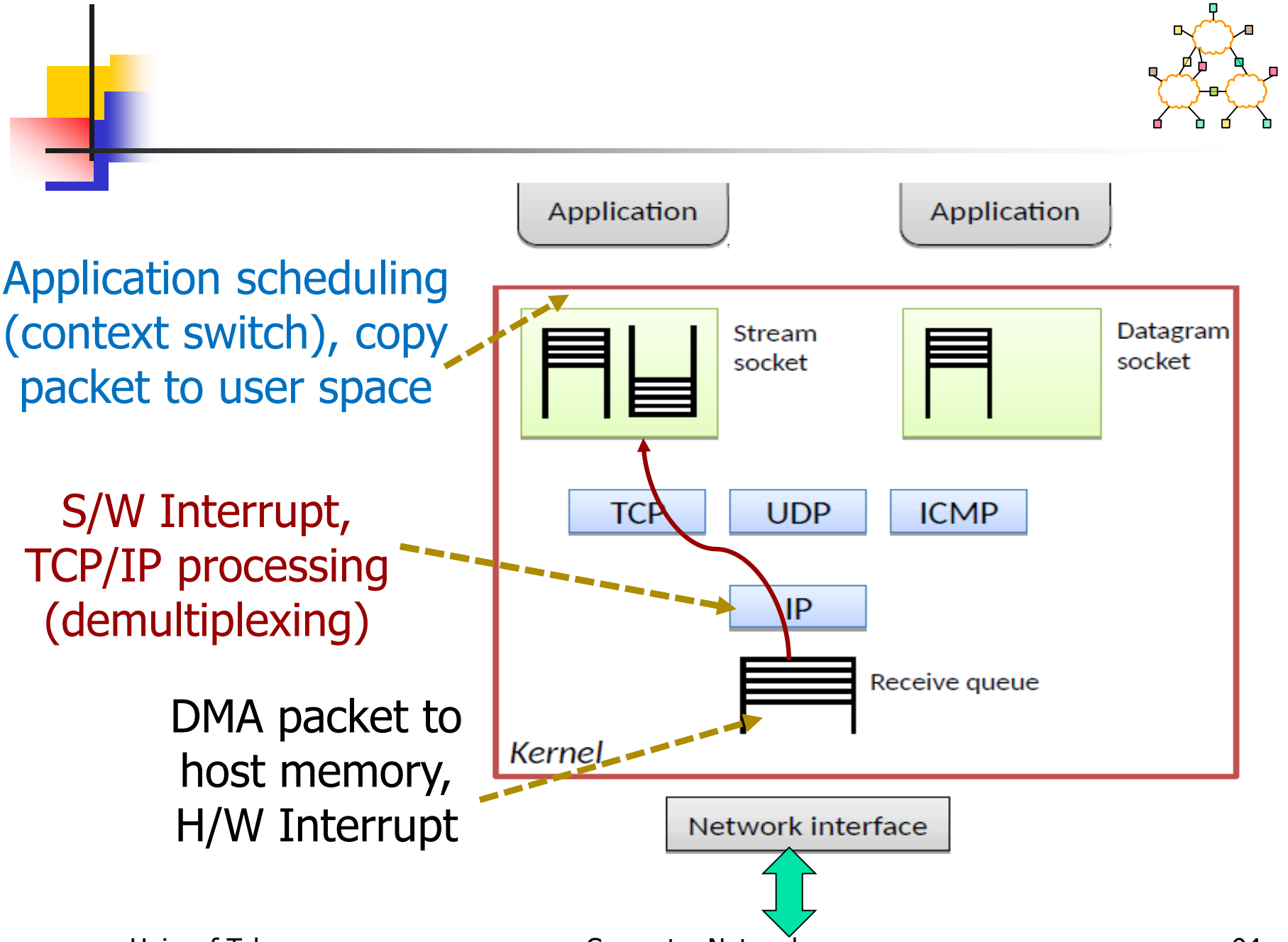


Component	Delay
Network switch	1 μs
Network adaptor	1 μs
OS network stack	5 μs
Speed of light (in fiber)	negligible

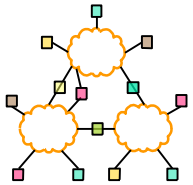
- OS overhead per packet exchanged between two hosts attached to the same switch:

  (2*5)/(1+2*1+2*5)=77% (!)

- It is a serious problem in very high links, 100G
- What can be done?

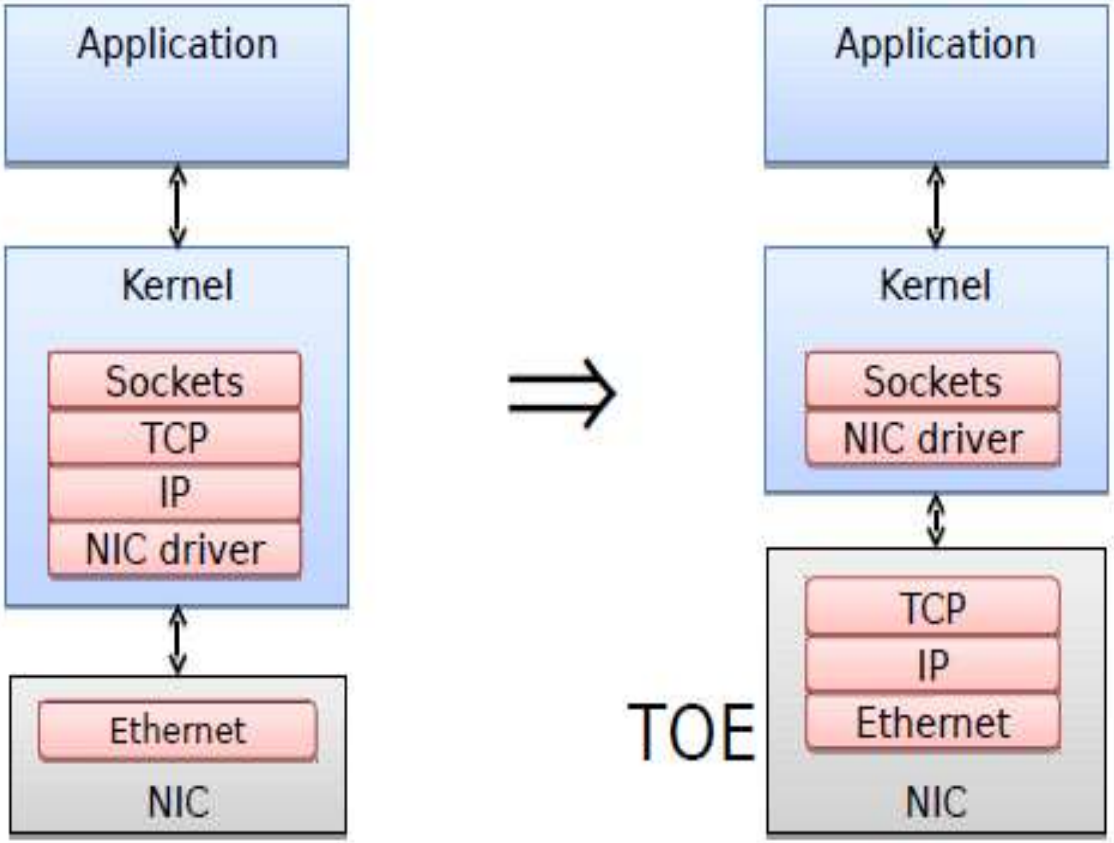

Application scheduling (context switch), copy packet to user space
S/W Interrupt, TCP/IP processing (demultiplexing)
DMA packet to host memory, H/W Interrupt
Application
Application
Stream socket
Datagram socket
TCP
UDP
ICMP
IP
Receive queue
Kernel
Network interface

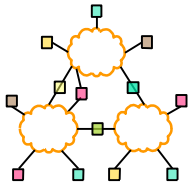
TCP offload



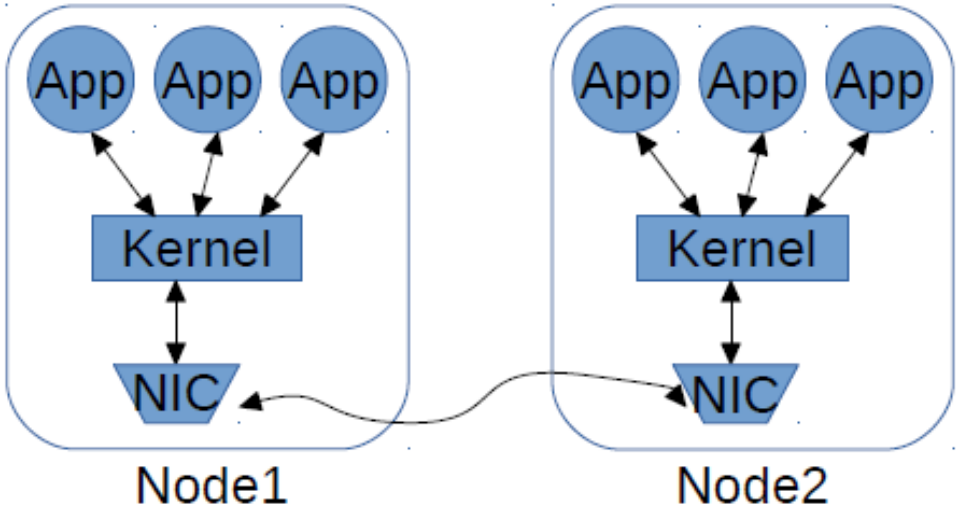
- Moving IP and TCP processing to the Network Interface (NIC)

✓Reduction of CPU cycles for header processing, checksumm
✓Fewer CPU interrupts
✓Fewer bytes copied over the memory bus
✓Potential to offload expensive features such as encryption

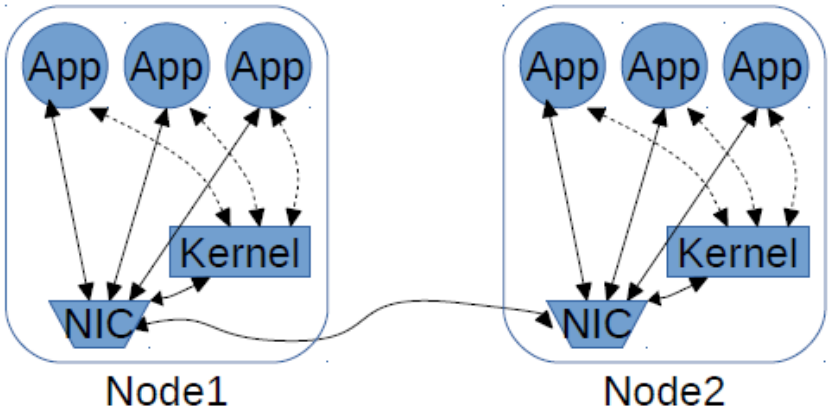




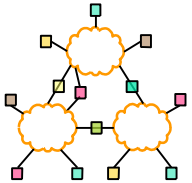
Traditional Networking



User-level Networking



Next Lecture: Network Virtualization



- How to virtualize Network resources
- Assigned reading
  - Network Virtualization – a View from the Bottom