Datacenters Networking

Why Datacenters?

Your <public-life, private-life, banks, government> live in my datacenter.

Security, Privacy, Control, Cost, Energy, (breaking) received wisdom; all this and more come together into sharp focus in datacenters.

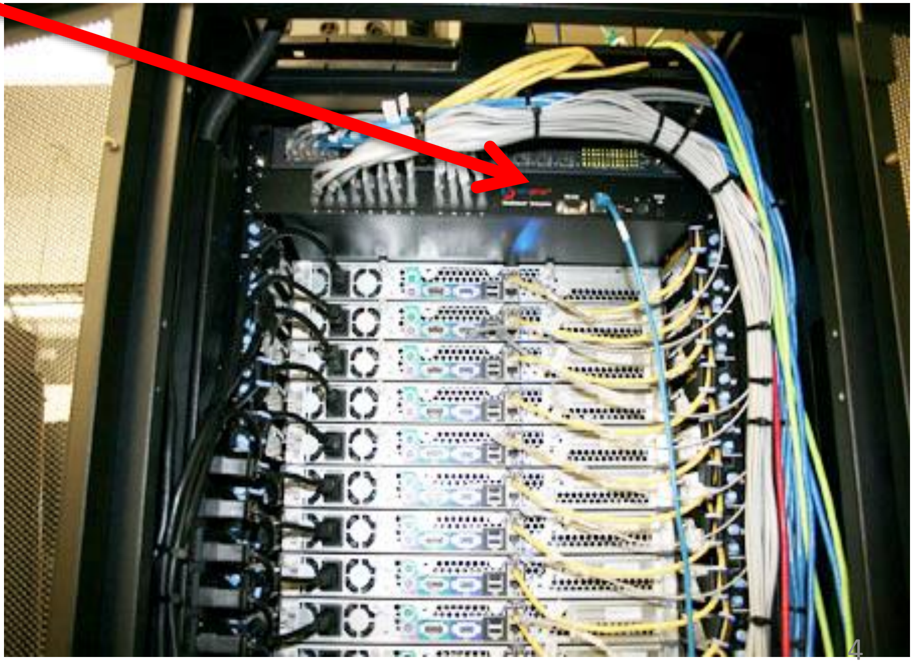
Do I need to labor the point?

What goes into a datacenter (network)?

- Servers organized in racks

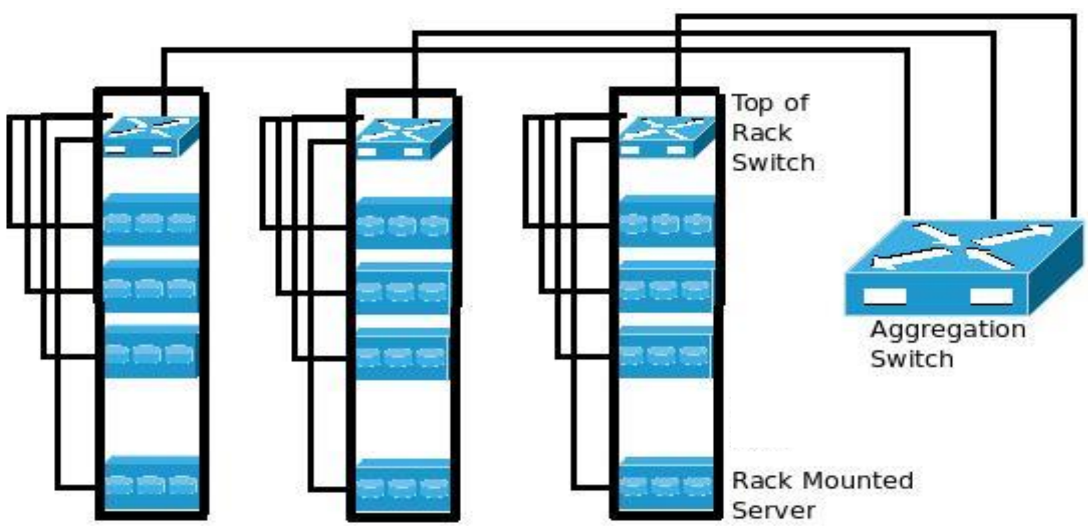
What goes into a datacenter (network)?

- Servers organized in racks
- Each rack has a `Top of Rack’ (ToR) switch



What goes into a datacenter (network)?

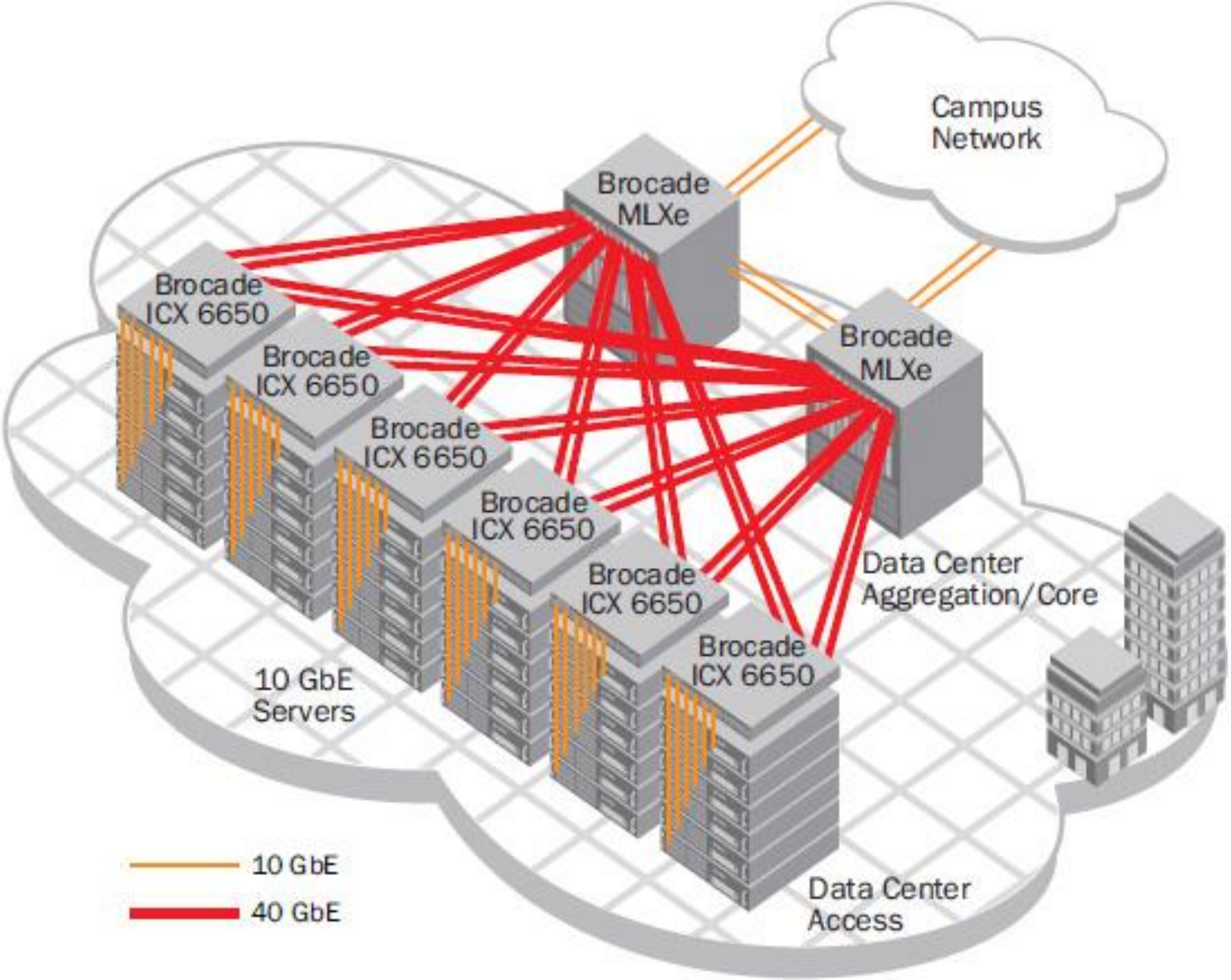
- Servers organized in racks
- Each rack has a `Top of Rack' (ToR) switch
- An `aggregation fabric' interconnects ToR switches



What goes into a datacenter (network)?

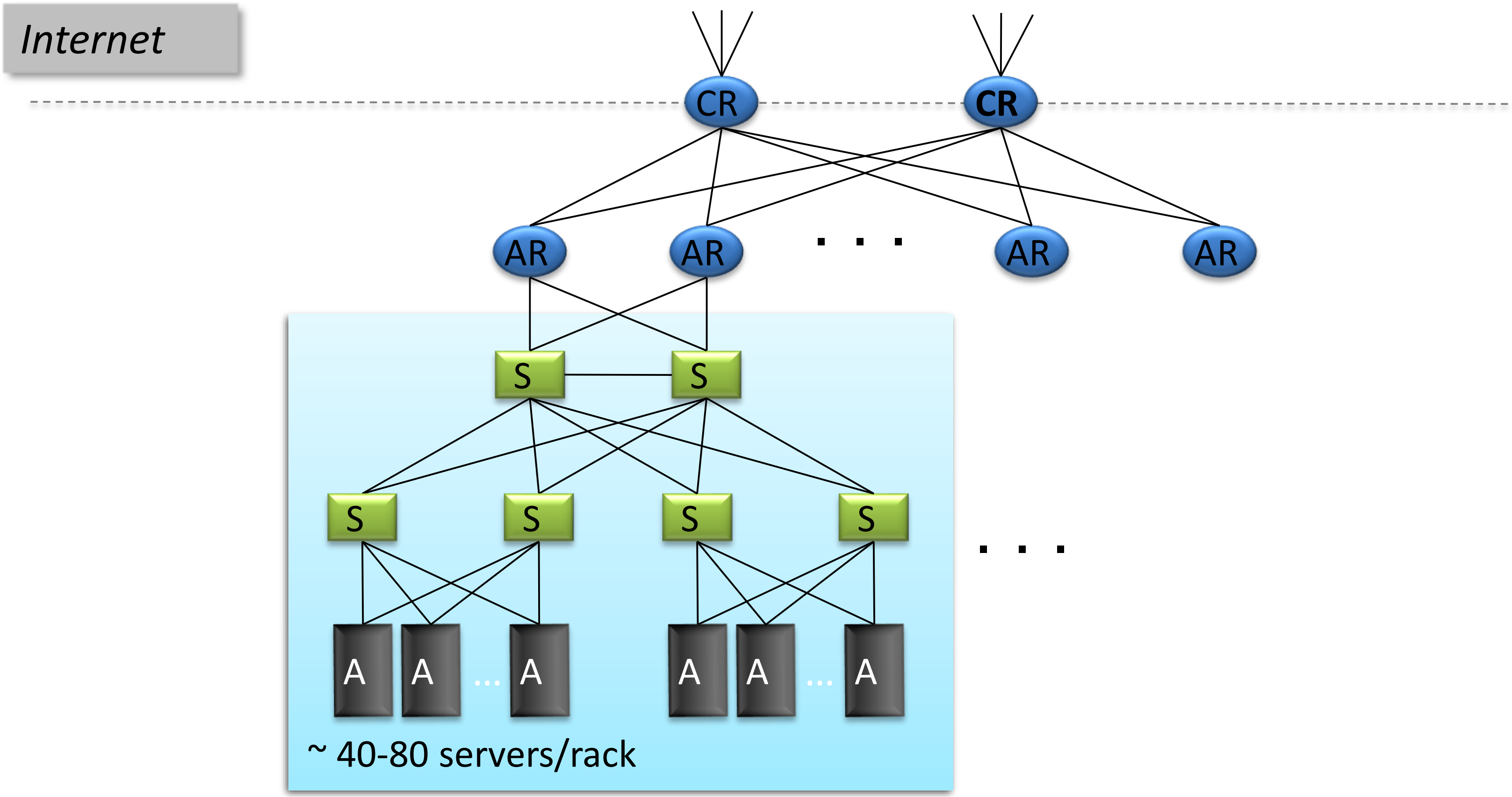
- Servers organized in racks
- Each rack has a `Top of Rack' (ToR) switch
- An `aggregation fabric' interconnects ToR switches
- Connected to the outside via `core' switches
  - note: blurry line between aggregation and core
- With network redundancy of ~2x for robustness

Example 1



Brocade reference design

Example 2



Cisco reference design

Observations on DC architecture

- Regular, well-defined arrangement
- Hierarchical structure with rack/aggr/core layers
- Mostly homogenous within a layer
- Supports communication between servers and between servers and the external world

Contrast: ad-hoc structure, heterogeneity of WANs

Datacenters have been around for a while

1949, EDSAC

What’s new?

SCALE!

How big exactly?

- 1M servers [Microsoft]
  - less than google, more than amazon

- > $1B to build one site [Facebook]

- >$20M/month/site operational costs [Microsoft ’09]

But only O(10-100) sites

What’s new?

- Scale
- Service model
  - user-facing, revenue generating services
  - multi-tenancy
  - jargon: SaaS, PaaS, DaaS, IaaS, …

Implications

- Scale
  - need scalable solutions (duh)
  - improving efficiency, lowering cost is critical

  → *`scale out' solutions w/ commodity technologies*

- Service model
  - performance means $$
  - *virtualization* for isolation and portability

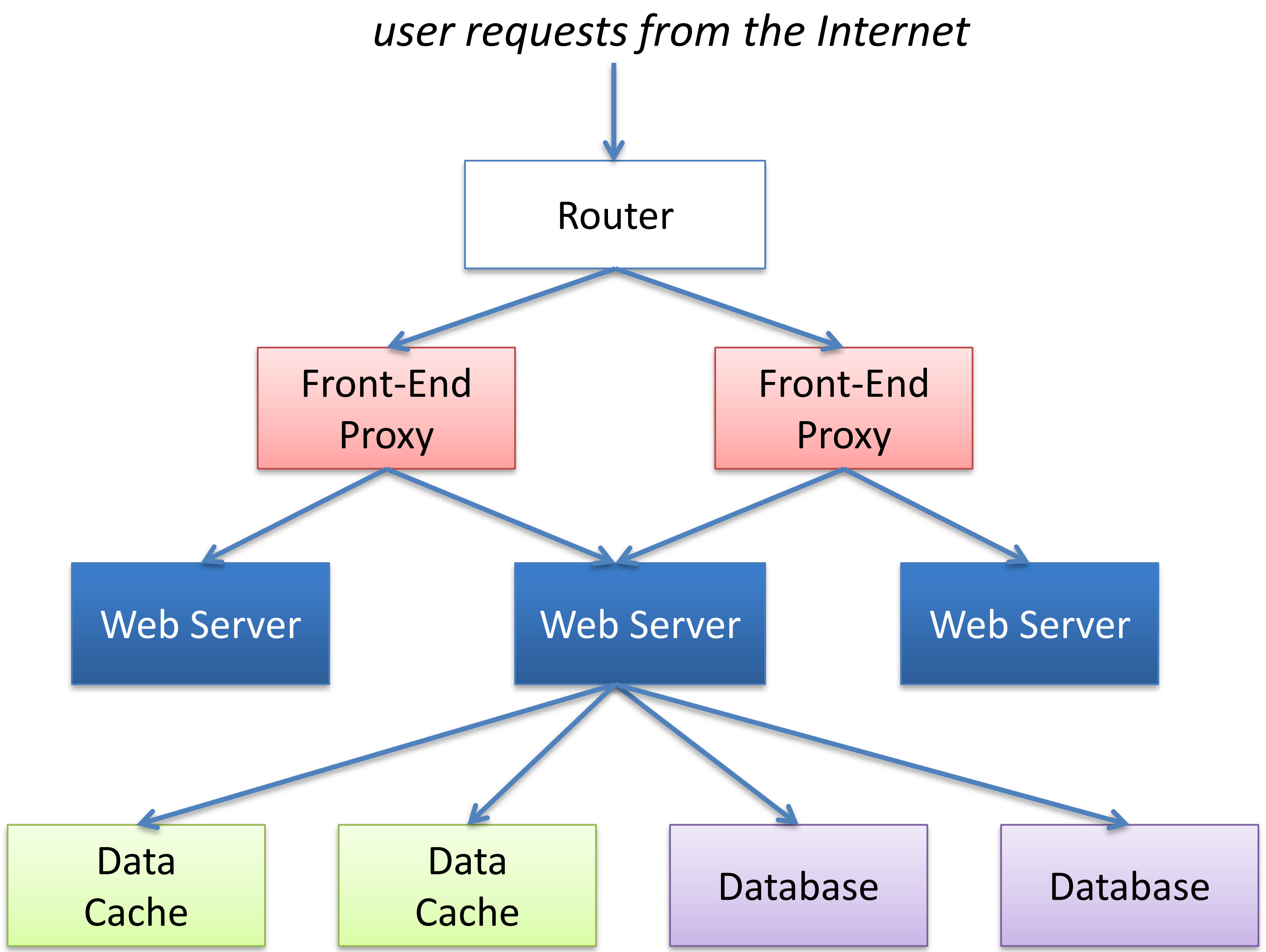
Multi-Tier Applications

- Applications decomposed into tasks
  - Many separate components
  - Running in parallel on different machines

Componentization leads to different types of network traffic

- "North-South traffic"
  - Traffic between external clients and the datacenter
  - Handled by front-end (web) servers, mid-tier application servers, and back-end databases
  - Traffic patterns fairly stable, though diurnal variations

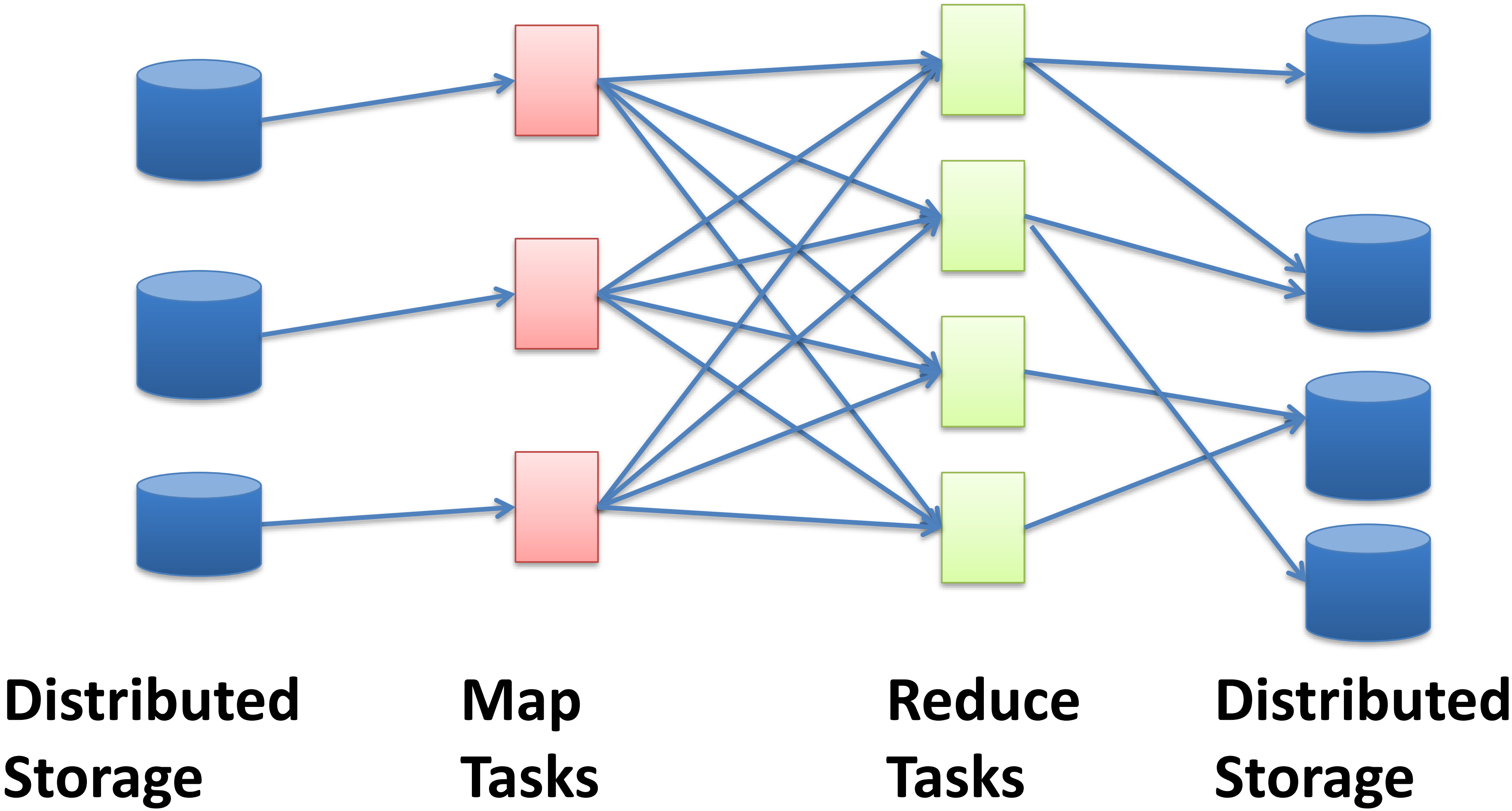
North-South Traffic



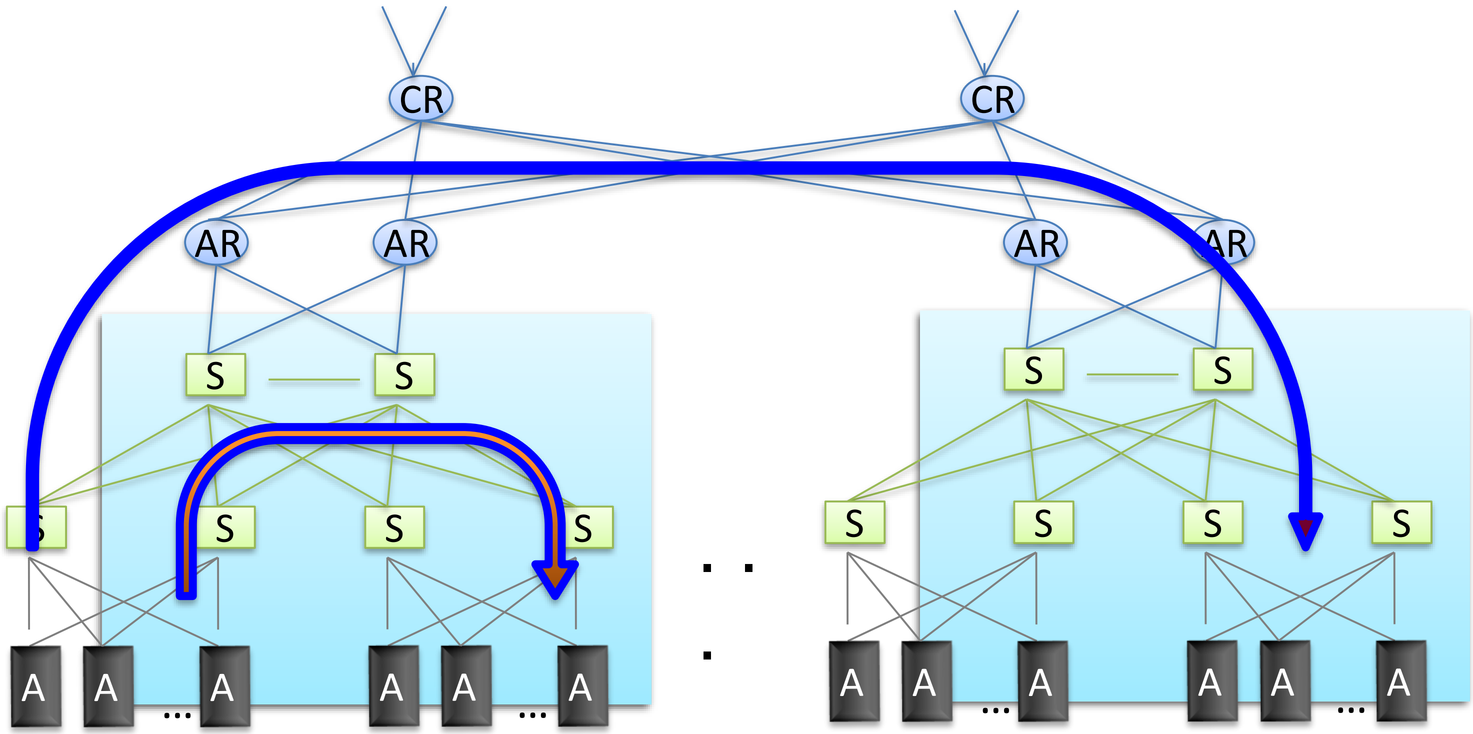
Componentization leads to different types of network traffic

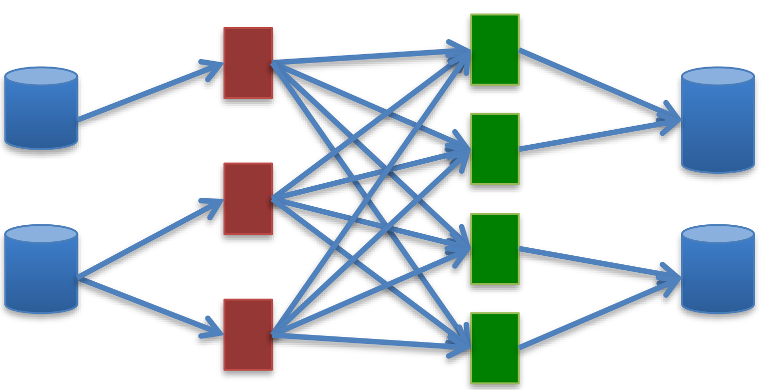
- "North-South traffic"
  - Traffic between external clients and the datacenter
  - Handled by front-end (web) servers, mid-tier application servers, and back-end databases
  - Traffic patterns fairly stable, though diurnal variations
- "East-West traffic"
  - Traffic between machines in the datacenter
  - Comm *within* "big data" computations (e.g. Map Reduce)
  - Traffic may shift on small timescales (e.g., minutes)

East-West Traffic

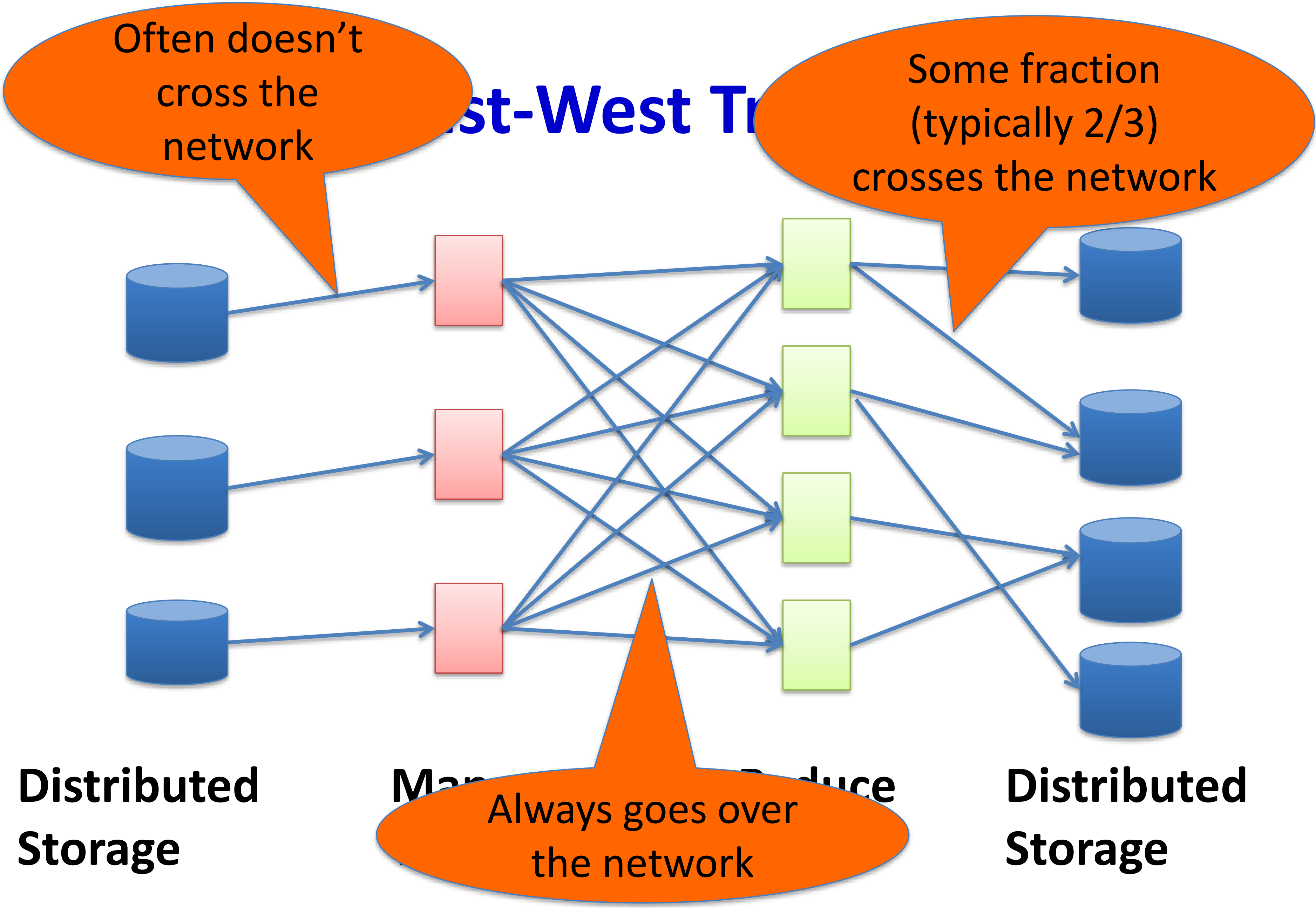


East-West Traffic





st-West Tr


Often doesn't cross the network
Some fraction (typically 2/3) crosses the network
Always goes over the network
Distributed Storage
Ma
duce
Distributed Storage

What's different about DC networks?

<u>Characteristics</u>

- Huge scale:
  - ~20,000 switches/routers
  - *contrast: AT&T ~500 routers*

What’s different about DC networks?

Characteristics

- Huge scale:
- Limited geographic scope:
  - High bandwidth: 10/40/100G
  - *Contrast: Cable/aDSL/WiFi*
  - Very low RTT: 10s of microseconds
  - *Contrast: 100s of milliseconds in the WAN*

What's different about DC networks?

Characteristics

- Huge scale
- Limited geographic scope
- Single administrative domain
  - Can deviate from standards, invent your own, *etc.*
  - "Green field" deployment is still feasible

What's different about DC networks?

Characteristics

- Huge scale
- Limited geographic scope
- Single administrative domain
- Control over one/both endpoints
  – can change (say) addressing, congestion control, *etc.*
  – can add mechanisms for security/policy/etc. at the endpoints (typically in the hypervisor)

What's different about DC networks?

Characteristics

- Huge scale
- Limited geographic scope
- Single administrative domain
- Control over one/both endpoints
- Control over the *placement* of traffic source/sink
  – e.g., map-reduce scheduler chooses where tasks run
  – alters traffic pattern (what traffic crosses which links)

What's different about DC networks?

<u>Characteristics</u>

- Huge scale
- Limited geographic scope
- Single administrative domain
- Control over one/both endpoints
- Control over the *placement* of traffic source/sink
- Regular/planned topologies (e.g., trees/fat-trees)
  - Contrast: ad-hoc WAN topologies (dictated by real-world geography and facilities)

What's different about DC networks?

Characteristics

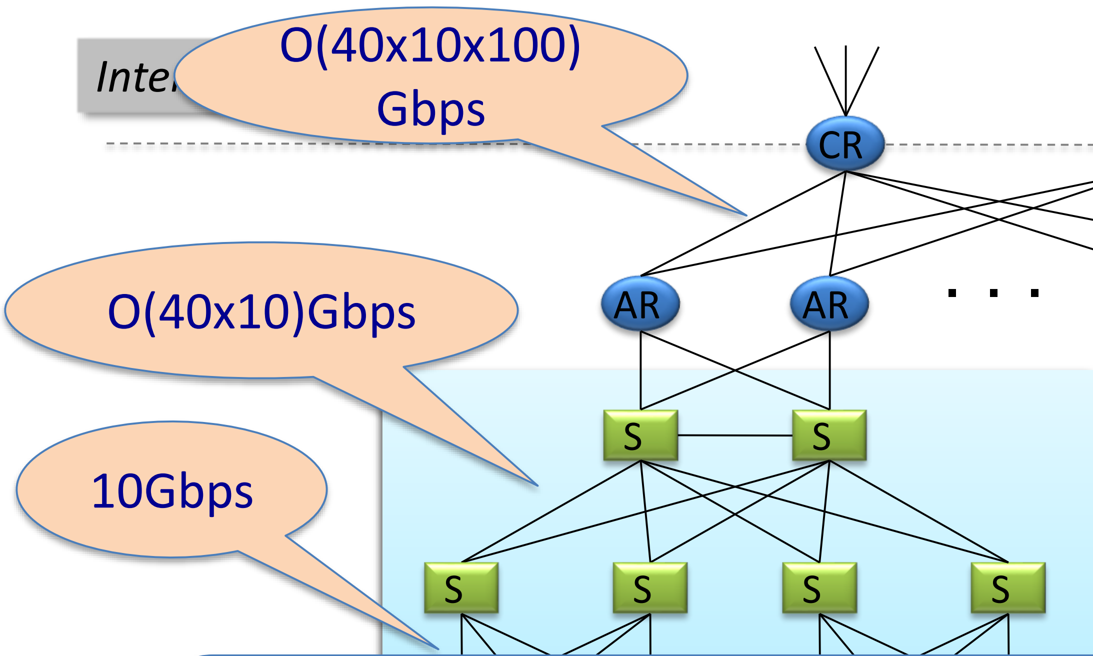
- Huge scale
- Limited geographic scope
- Single administrative domain
- Control over one/both endpoints
- Control over the *placement* of traffic source/sink
- Regular/planned topologies (e.g., trees/fat-trees)
- Limited heterogeneity
  - link speeds, technologies, latencies, …

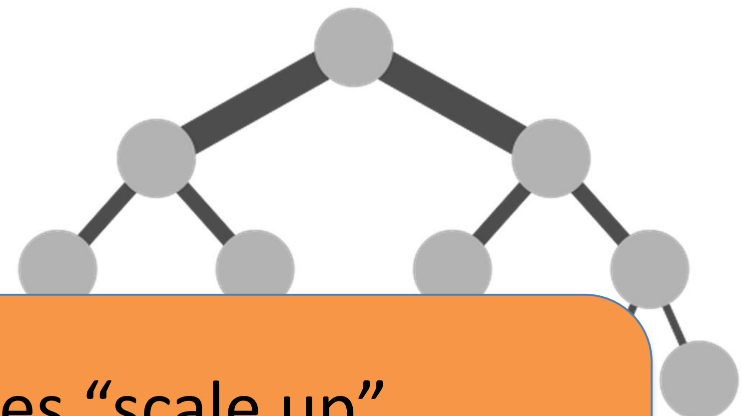
What’s different about DC networks?

Goals

- Extreme bisection bandwidth requirements
  - recall: all that east-west traffic
  - target: any server can communicate at its full link speed
  - problem: server’s access link is 10Gbps!

Full Bisection Bandwidth





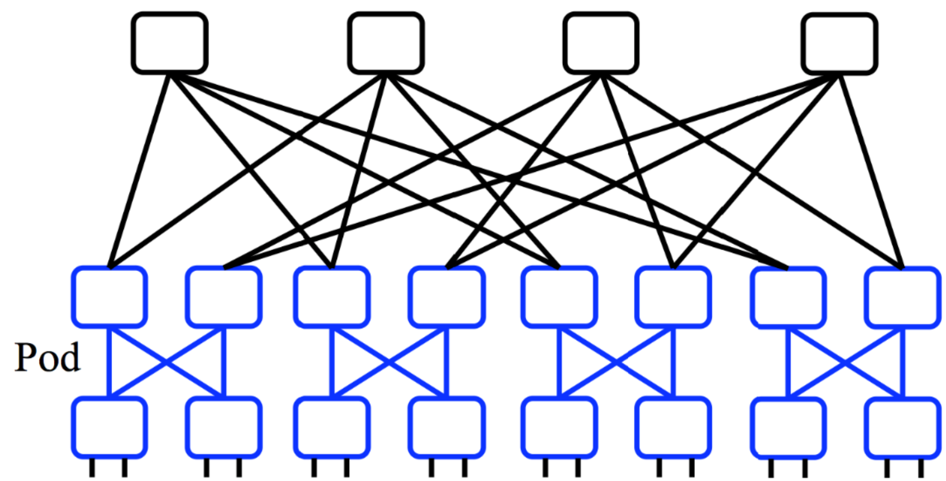
Traditional tree topologies "scale up"

- full bisection bandwidth is expensive
- typically, tree topologies "oversubscribed"

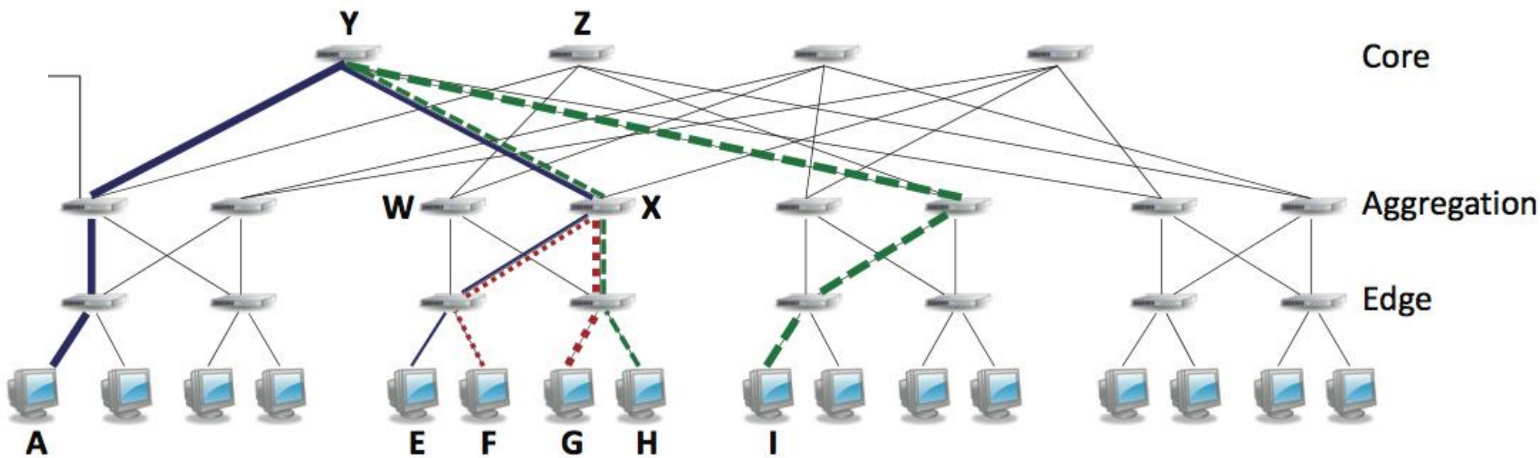
A “Scale Out” Design

- Build multi-stage `Fat Trees’ out of k-port switches
  - k/2 ports up, k/2 down
  - Supports $k^3/4$ hosts:
    - 48 ports, 27,648 hosts

All links are the same speed (e.g. 10Gps)



Full Bisection Bandwidth Not Sufficient



- To realize full bisectional throughput, routing must spread traffic across paths
- Enter load-balanced routing
  - How? (1) Let the network split traffic/flows at random (e.g., ECMP protocol -- RFC 2991/2992)
  - How? (2) Centralized flow scheduling?
  - Many more research proposals

What's different about DC networks?

<u>Goals</u>

- Extreme bisection bandwidth requirements
- Extreme latency requirements
  - real money on the line
  - current target: 1μs RTTs
  - how? cut-through switches making a comeback
    - reduces switching time

What’s different about DC networks?

Goals

- Extreme bisection bandwidth requirements
- Extreme latency requirements
  - real money on the line
  - current target: 1μs RTTs
  - how? cut-through switches making a comeback
  - how? avoid congestion
    - reduces queuing delay

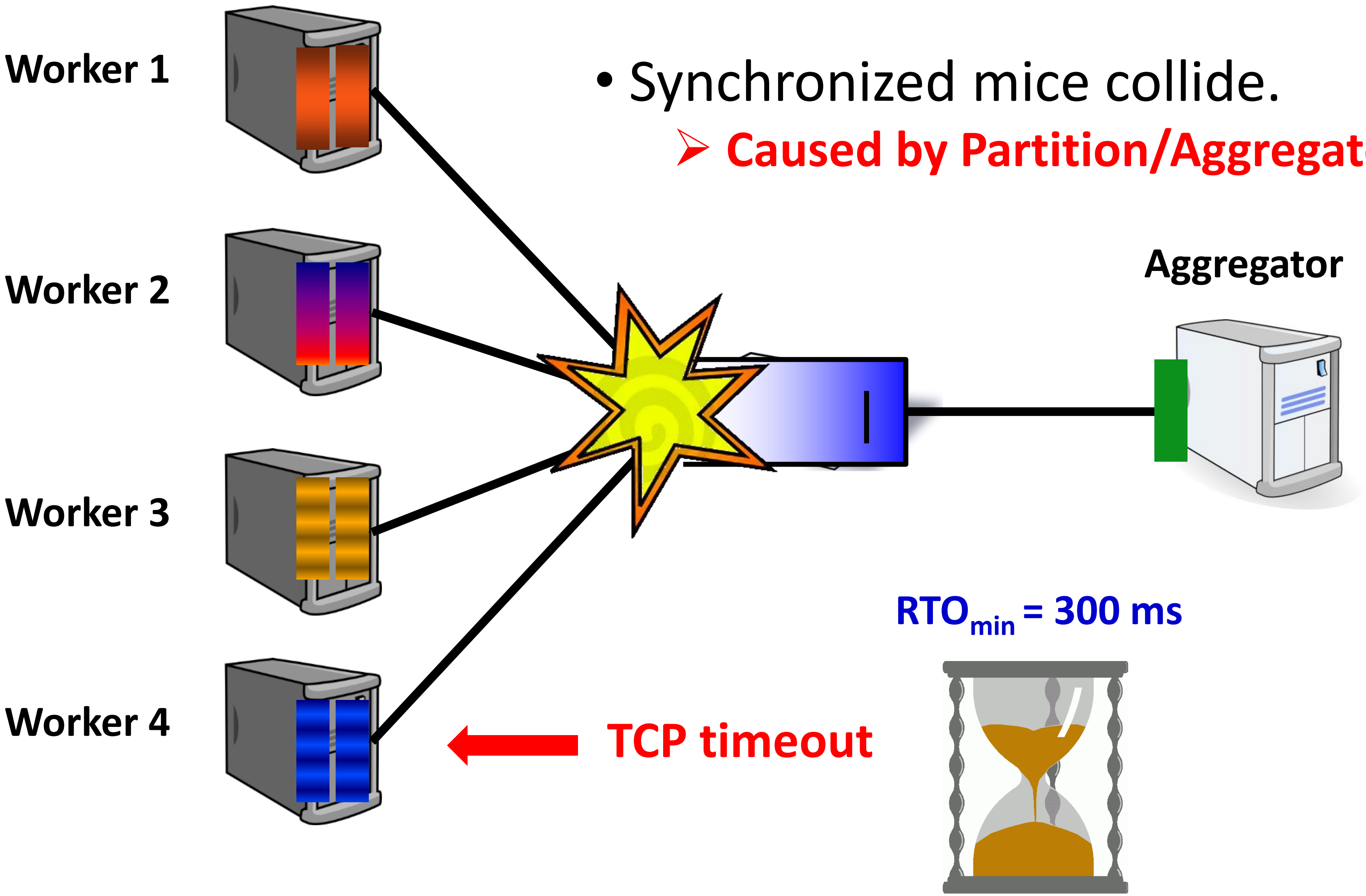
What's different about DC networks?

<u>Goals</u>

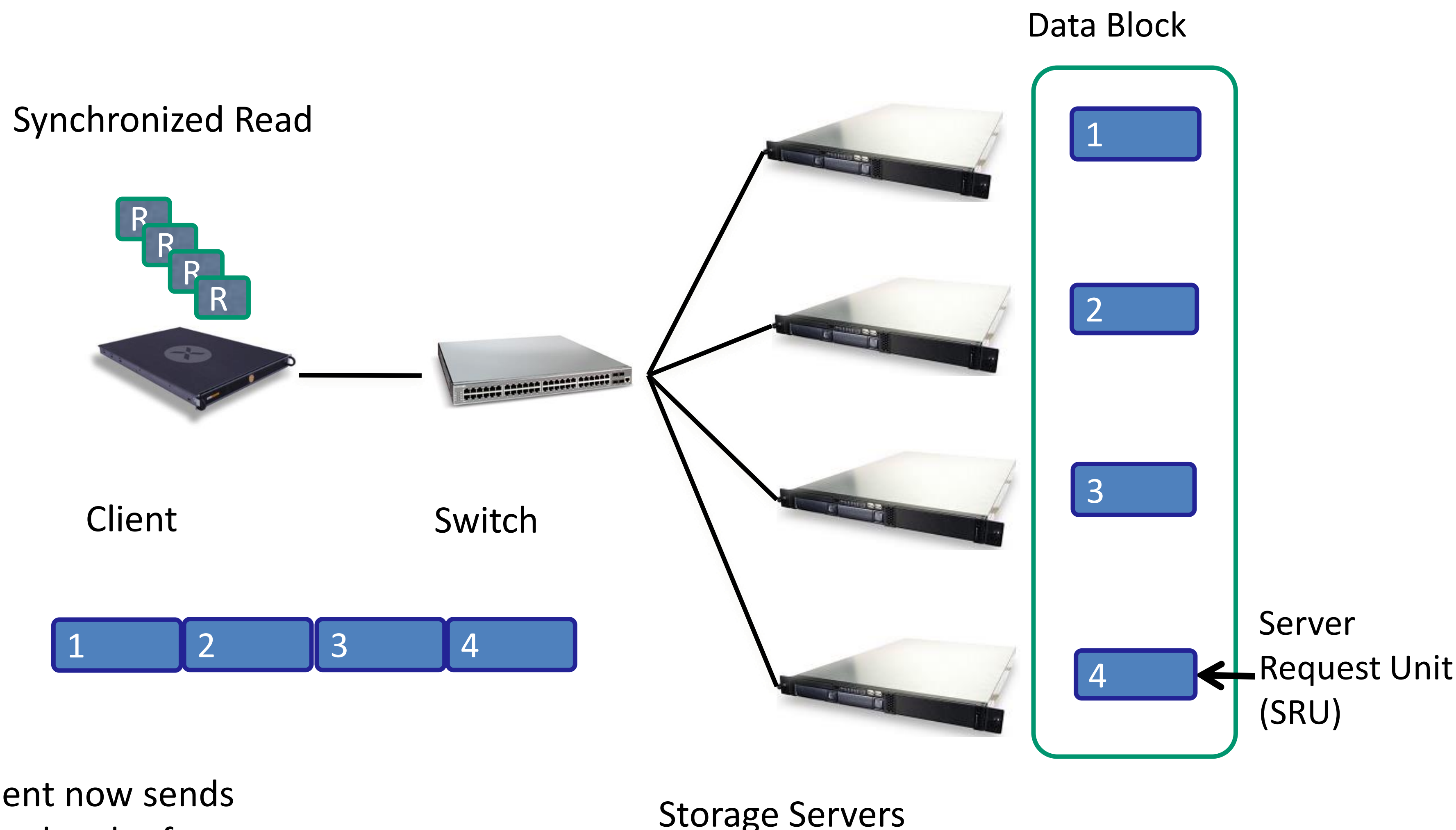
- Extreme bisection bandwidth requirements
- Extreme latency requirements
  - real money on the line
  - current target: 1μs RTTs
  - how? cut-through switches making a comeback (lec. 2!)
  - how? avoid congestion
  - how? fix TCP timers (e.g., default timeout is 500ms!)
  - how? fix/replace TCP to more rapidly fill the pipe

An example problem at scale - INCAST

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



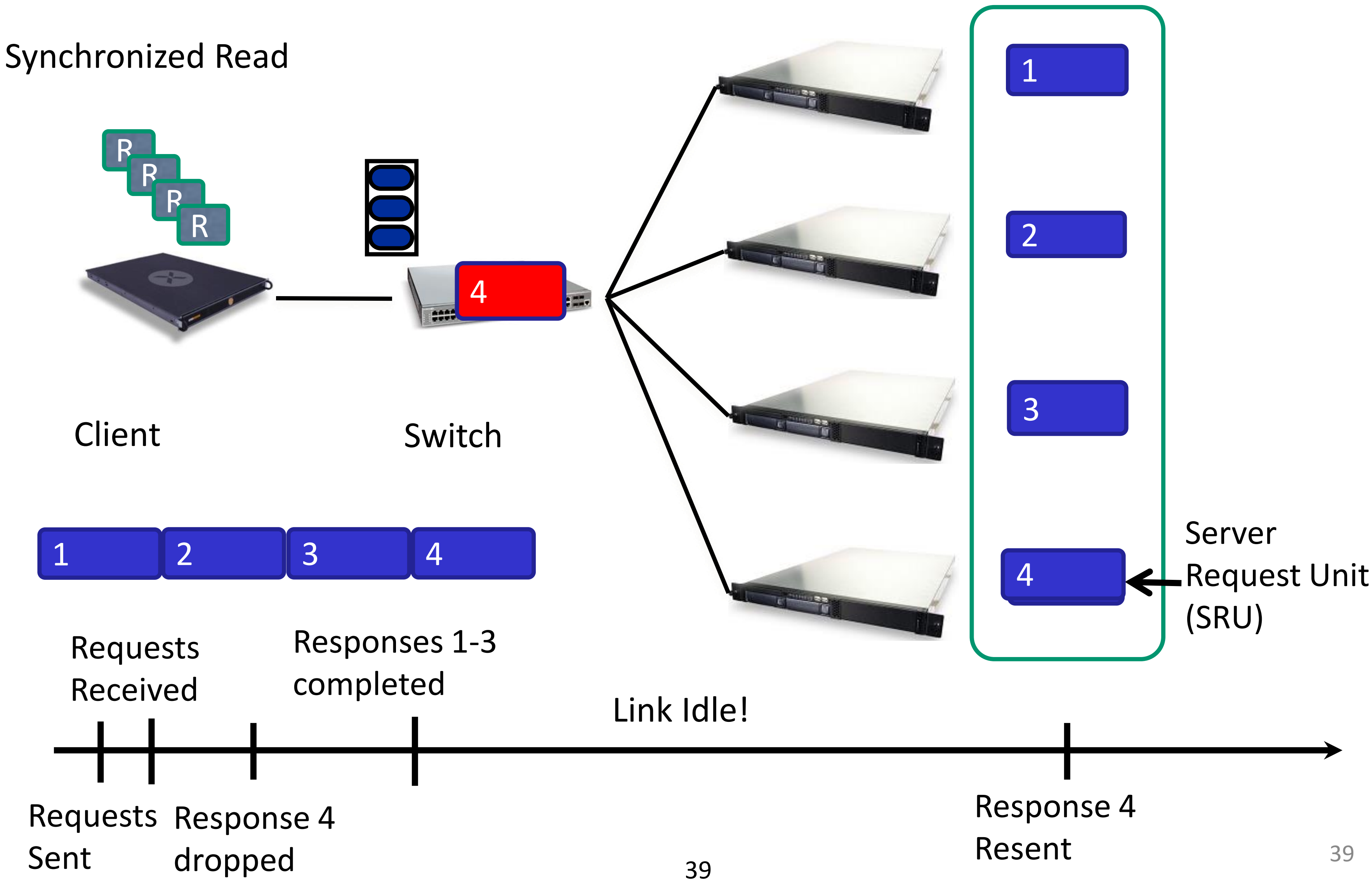
The Incast Workload



Client now sends
next batch of requests

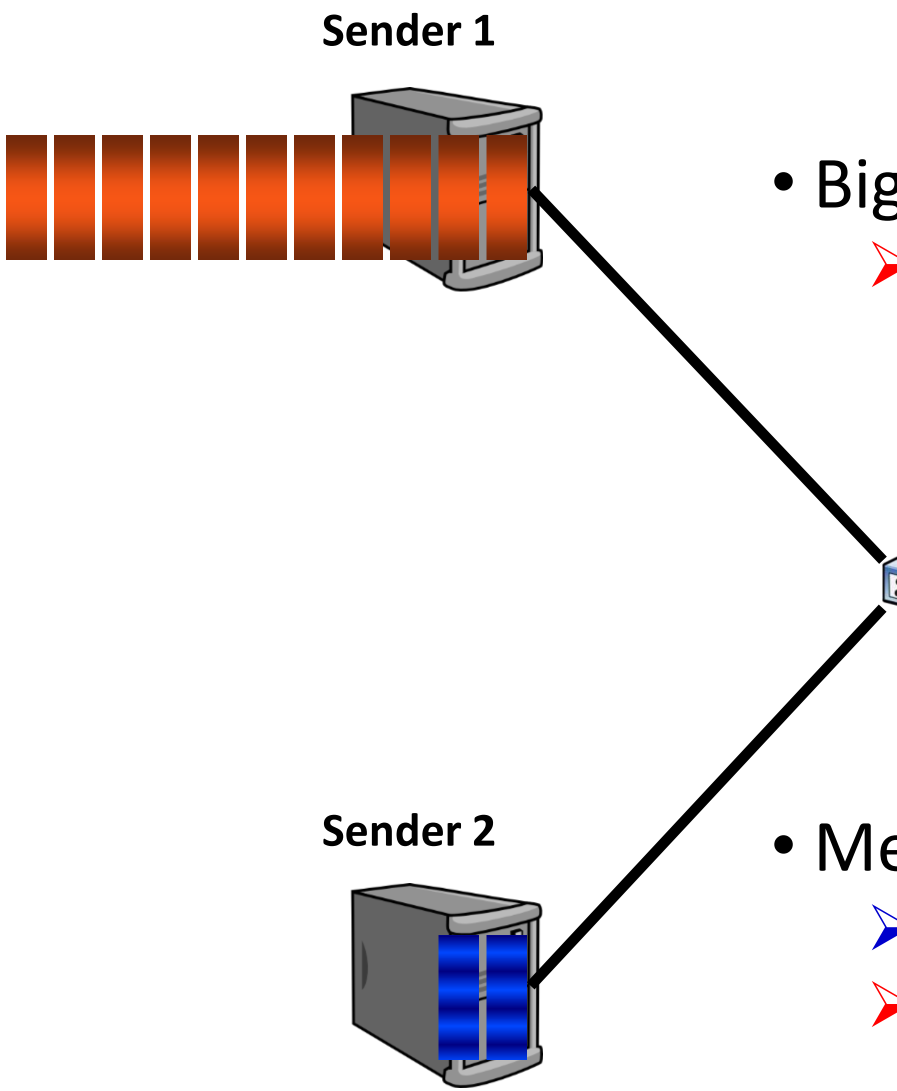
38

Incast Workload Overfills Buffers

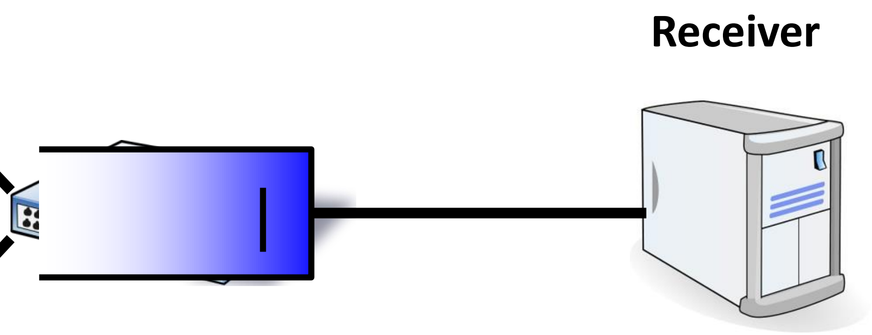


39

Queue Buildup



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

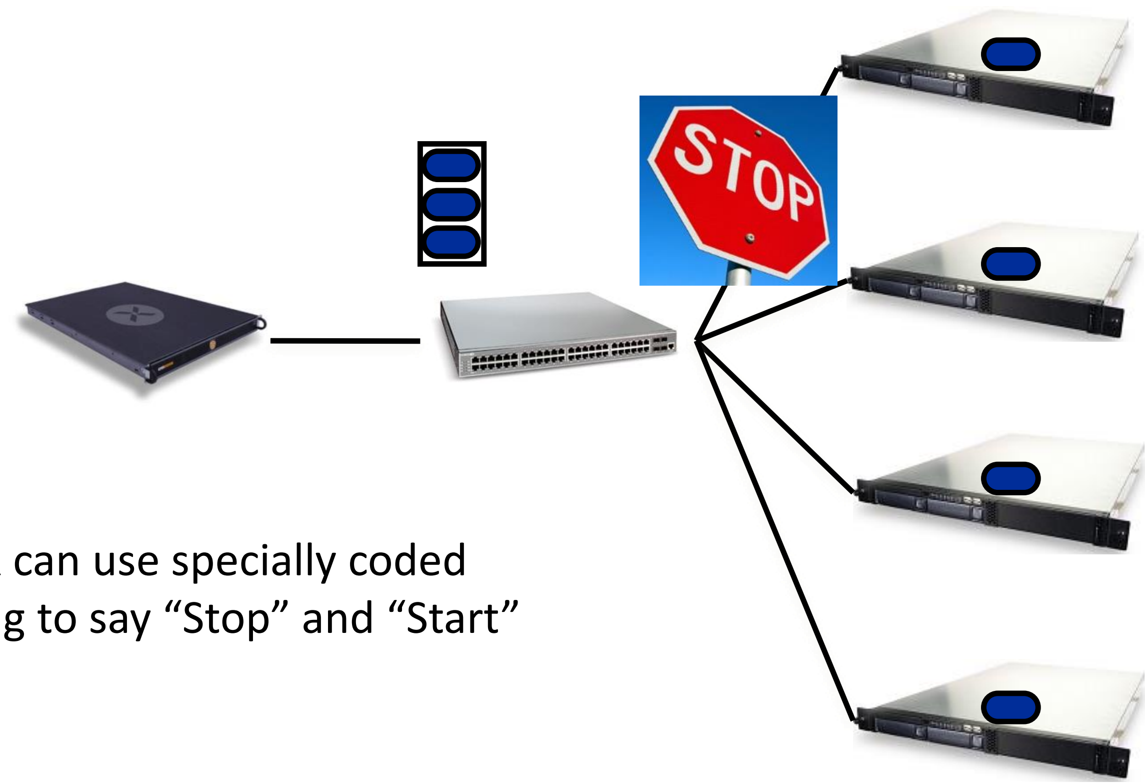


- Measurements in Bing cluster
  - ➢ **For 90% packets: RTT < 1ms**
  - ➢ **For 10% packets: 1ms < RTT < 15ms**

Link-Layer Flow Control

Common between switches but this is flow-control to the end host too…

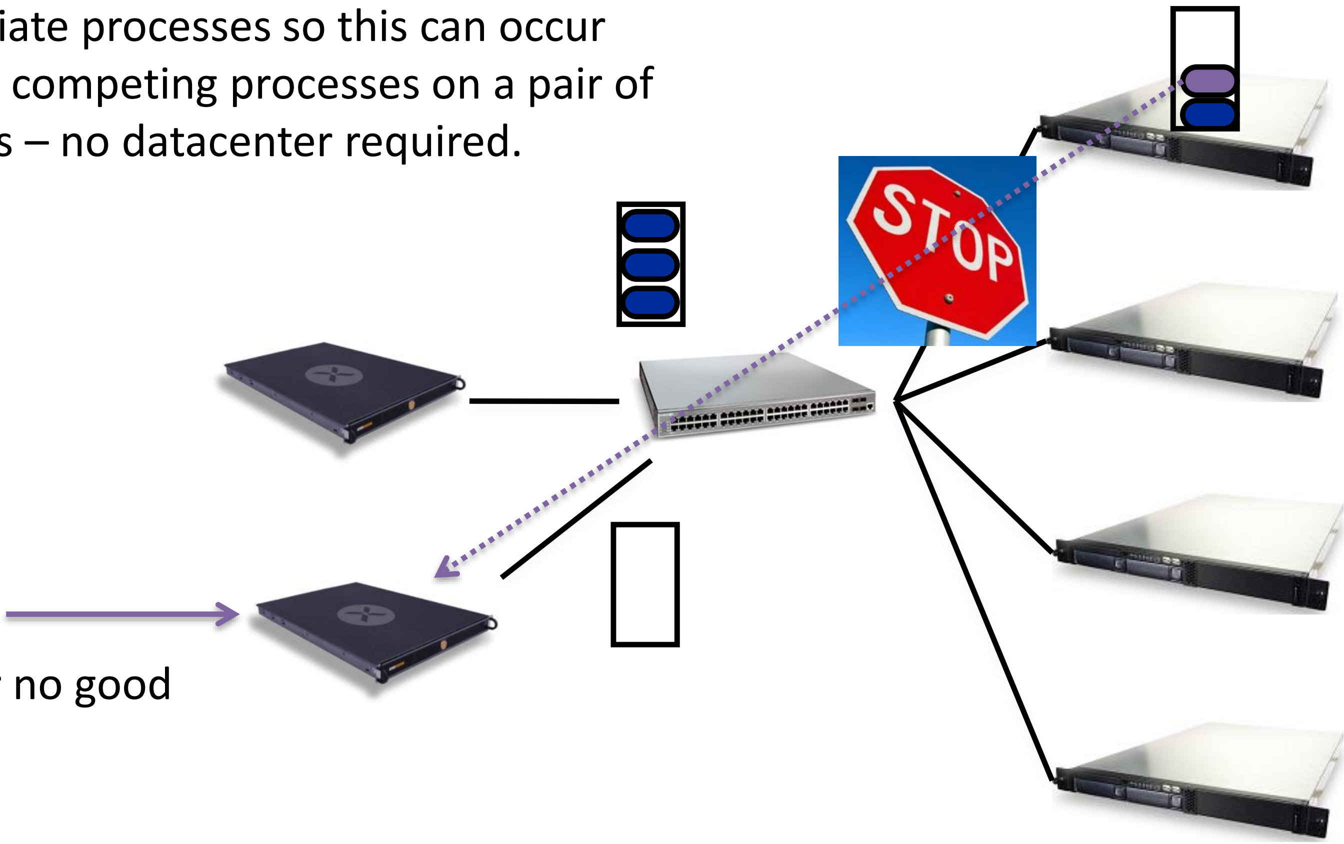
- Another idea to reduce incast is to employ Link-Layer Flow Control…..



Recall: the Data-Link can use specially coded symbols in the coding to say “Stop” and “Start”

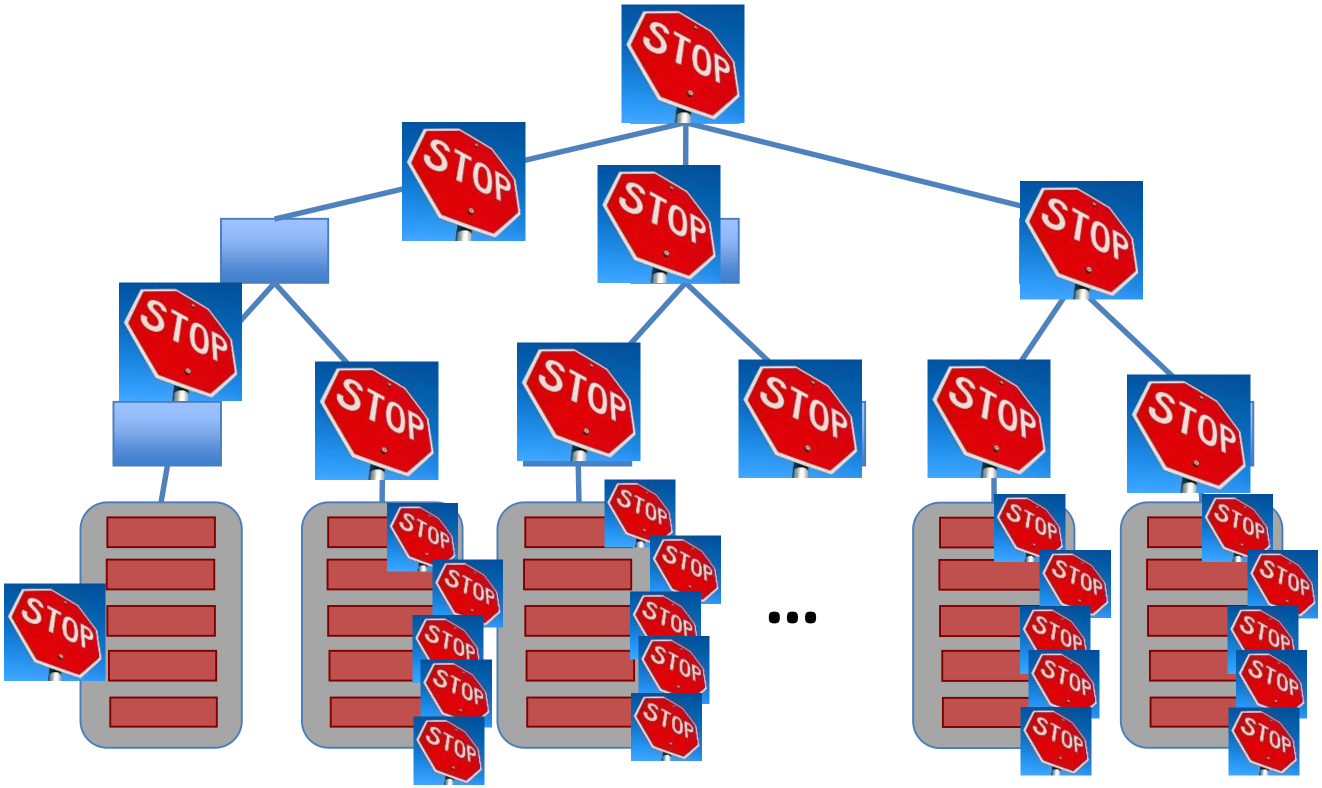
Link Layer Flow Control – The Dark side Head of Line Blocking….

Such HOL blocking does not even differentiate processes so this can occur between competing processes on a pair of machines – no datacenter required.



Waiting for no good reason….

Link Layer Flow Control
But its worse that you imagine….



Double down on trouble….

Did I mention this is Link-Layer!

That means no (IP) control traffic, no routing messages….

a whole system waiting for one machine

Incast is very unpleasant.

Reducing the impact of HOL in Link Layer Flow Control can be done through priority queues and *overtaking*….

What’s different about DC networks?

Goals

- Extreme bisection bandwidth requirements
- Extreme latency requirements
- *Predictable, deterministic* performance
  - “your packet will reach in Xms, or not at all”
  - “your VM will always see at least YGbps throughput”
  - Resurrecting `best effort’ vs. `Quality of Service’ debates
  - How is still an open question

What's different about DC networks?

Goals

- Extreme bisection bandwidth requirements
- Extreme latency requirements
- *Predictable, deterministic* performance
- Differentiating between tenants is key
  - e.g., "No traffic between VMs of tenant A and tenant B"
  - "Tenant X cannot consume more than XGbps"
  - "Tenant Y's traffic is low priority"

What’s different about DC networks?

Goals

- Extreme bisection bandwidth requirements
- Extreme latency requirements
- *Predictable, deterministic* performance
- Differentiating between tenants is key
- Scalability (of course)
  - Q: How’s Ethernet spanning tree looking?

What's different about DC networks?

Goals

- Extreme bisection bandwidth requirements
- Extreme latency requirements
- *Predictable, deterministic* performance
- Differentiating between tenants is key
- Scalability (of course)
- Cost/efficiency
  - focus on commodity solutions, ease of management
  - some debate over the importance in the network case

Summary

- new characteristics and goals
- some liberating, some constraining
- scalability is the baseline requirement
- more emphasis on performance
- less emphasis on heterogeneity
- less emphasis on interoperability

Congestion Control in Data Center Networks

Overview

- Why is the problem different from that in the Internet?
- What are possible solutions?

DC Traffic Patterns

- In-cast applications
  - Client send queries to servers
  - Responses are synchronized

- Few overlapping long flows
  - According to DCTCP's measurement

Data Center TCP (DCTCP)

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

Microsoft Research Stanford University

Data Center Packet Transport

- Large purpose-built DCs
  - Huge investment: R&D, business
- Transport **inside** the DC
  - TCP rules (99.9% of traffic)
- How's TCP doing?

TCP in the Data Center

- We'll see TCP does not meet demands of apps.
  - Suffers from bursty packet drops, Incast [SIGCOMM '09], ...
  - 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
  - No solid understanding of consequences, tradeoffs

Roadmap

- What's really going on?
  - Interviews with developers and operators
  - Analysis of applications
  - Switches: shallow-buffered vs deep-buffered
  - Measurements

- A systematic study of transport in Microsoft's DCs
  - Identify **impairments**
  - Identify **requirements**

- 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 Application Structure

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

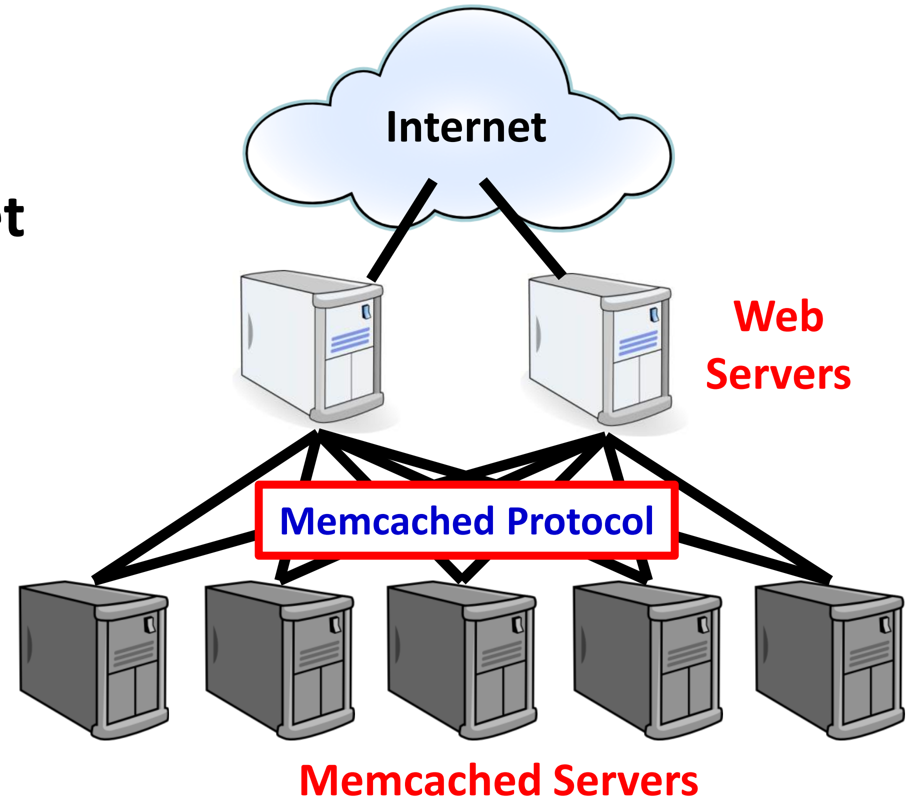
Generality of Partition/Aggregate

- The foundation for many large-scale web applications.
  - Web search, Social network composition, Ad selection, etc.

- Example: **Facebook**

Partition/Aggregate ~ Multiget

- Aggregators: **Web Servers**
- Workers: **Memcached Servers**



Workloads

- Partition/Aggregate
  (Query) → **Delay-sensitive**

- Short messages [50KB-1MB]
  (Coordination, Control state) → **Delay-sensitive**

- Large flows [1MB-50MB]
  (Data update) → **Throughput-sensitive**

Impairments

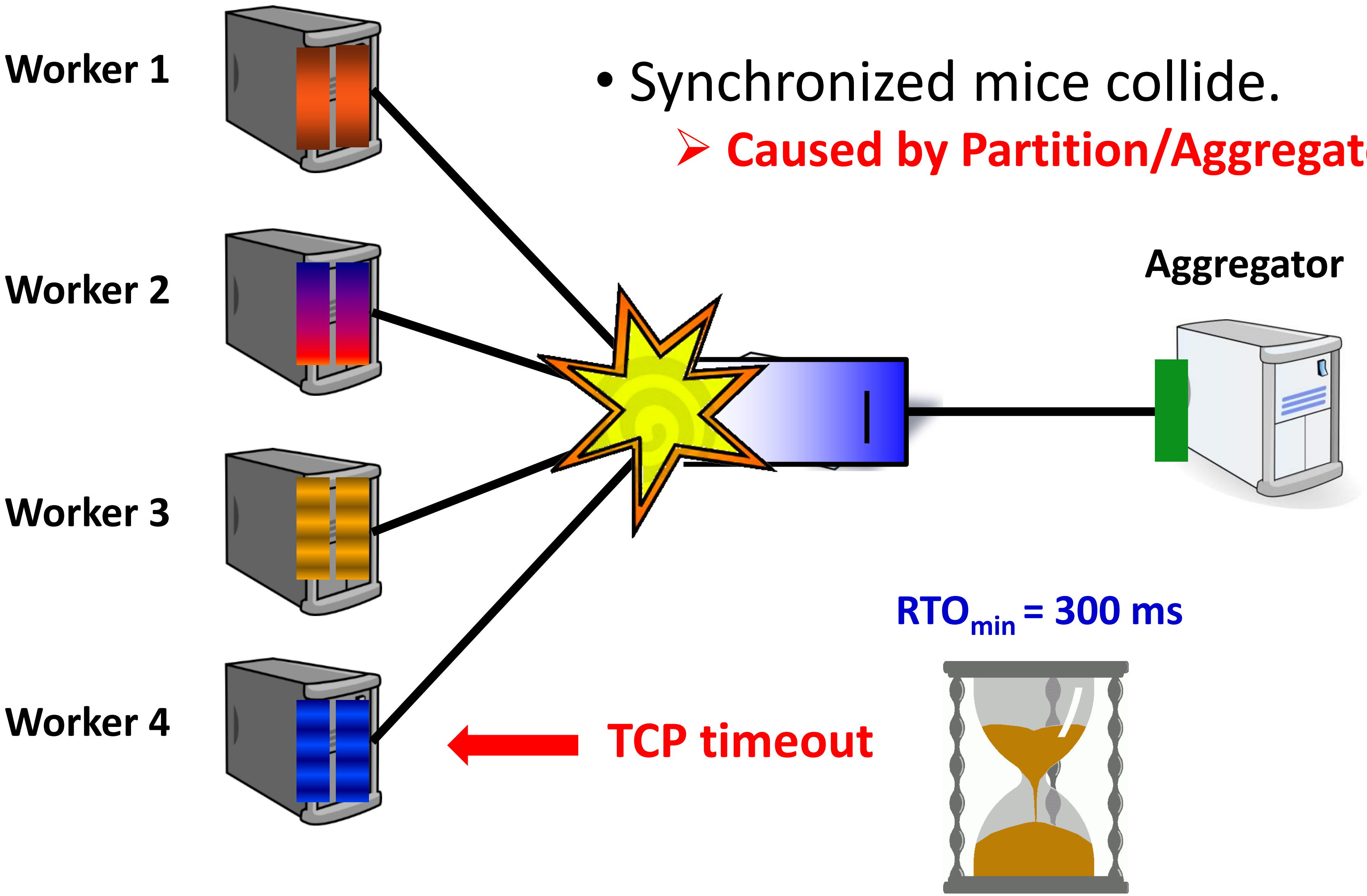
- Incast

- Queue Buildup

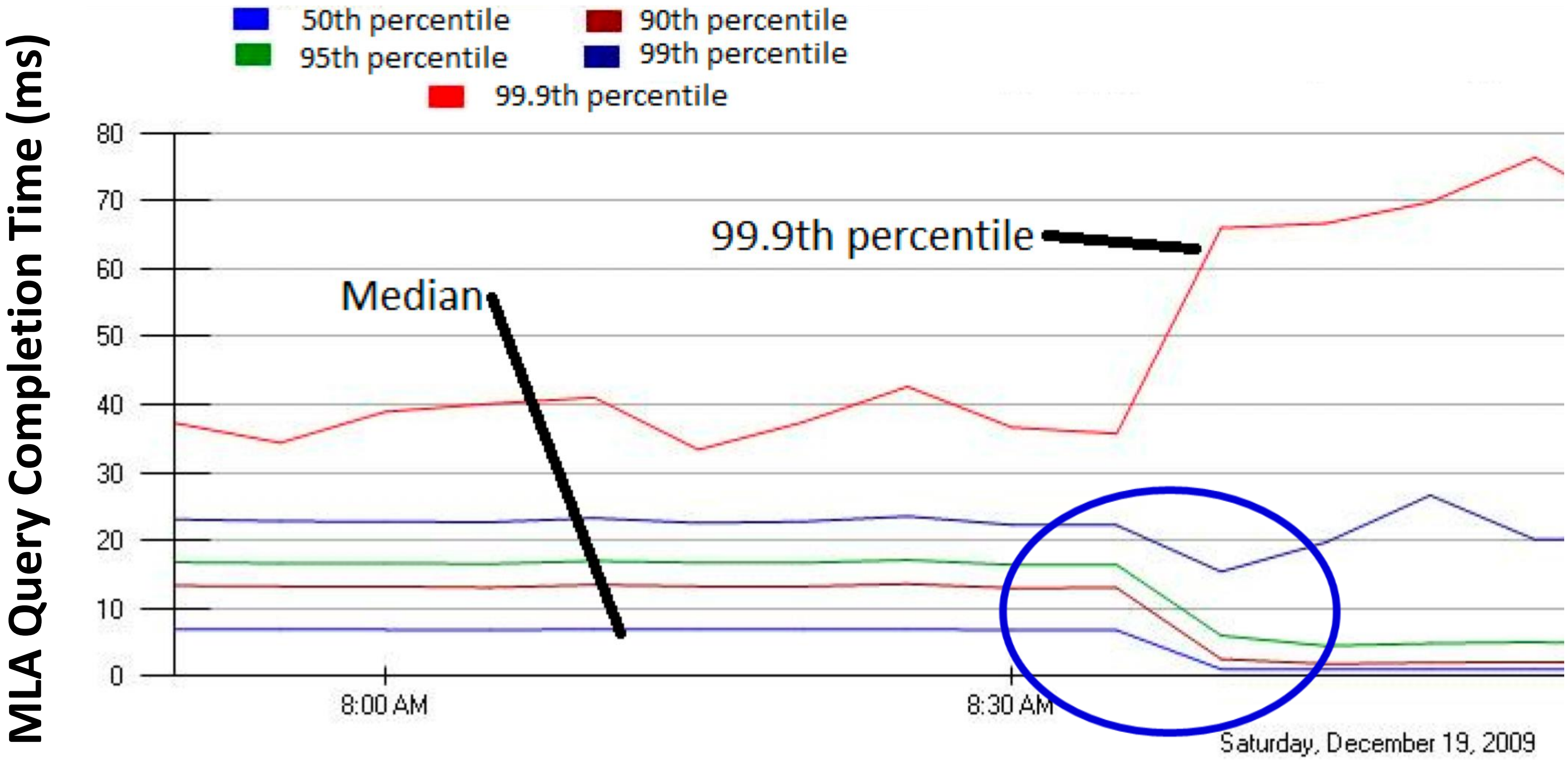
- Buffer Pressure

Incast

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



Incast Really Happens



Jittering

99.9^th^ percentile is being tracked.

ntiles.

InCast: Goodput collapses as senders increase

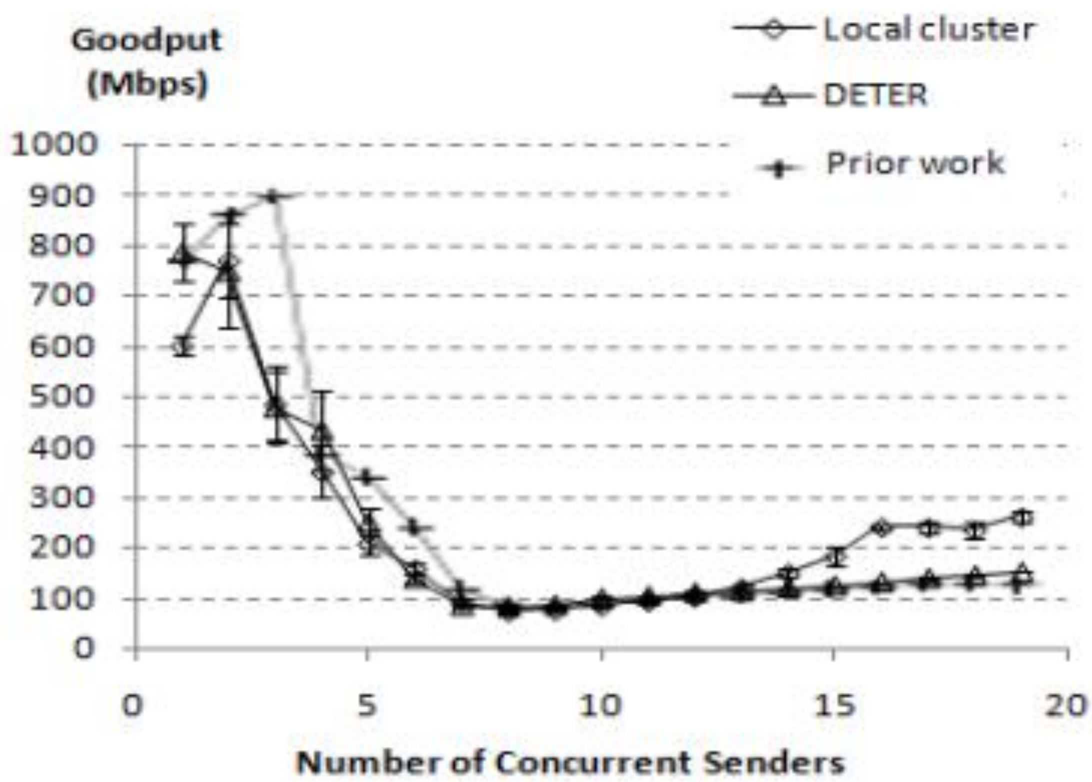


Figure 1: TCP Incast goodput collapse up to 20 senders for three different environments

InCast: Synchronized timeouts

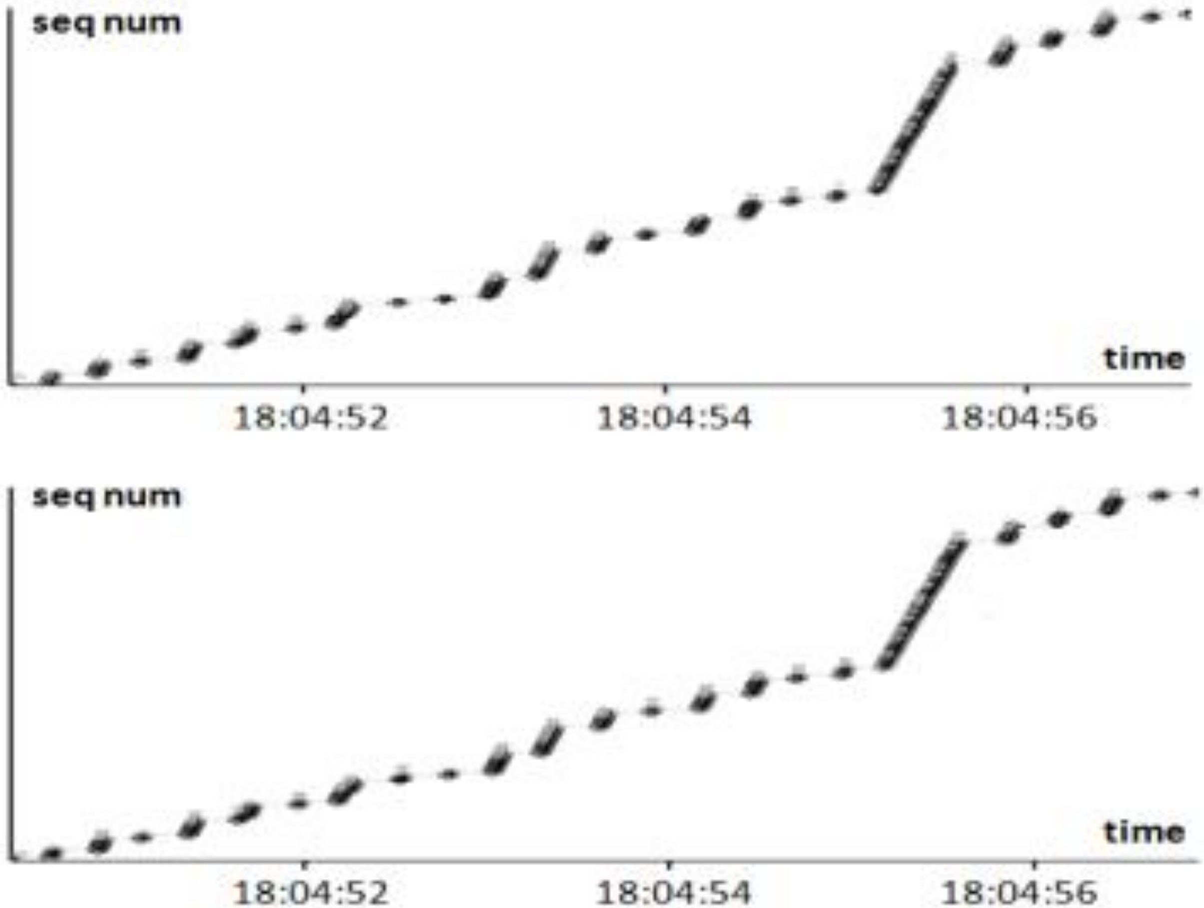
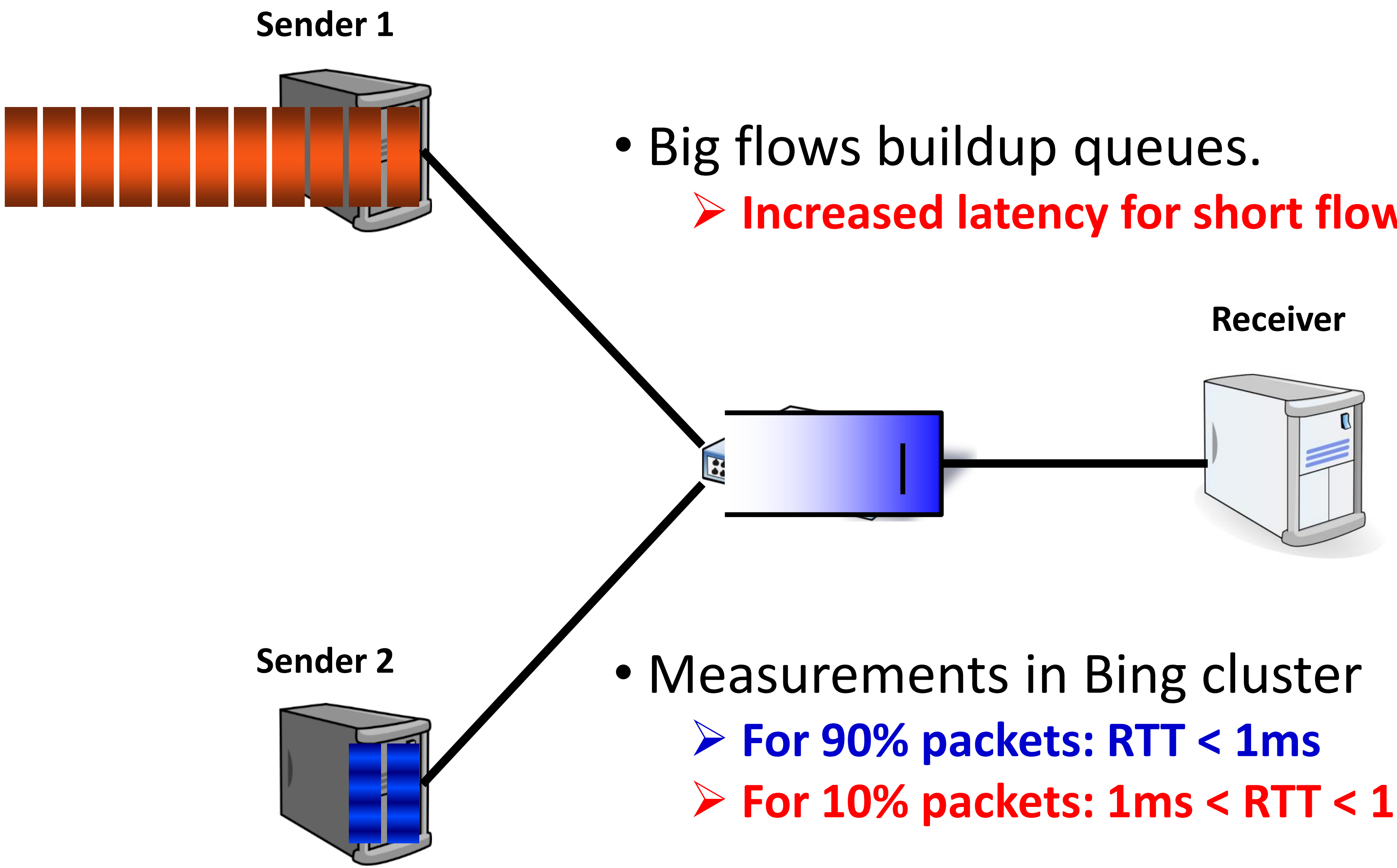


Figure 2: TCP sequence numbers vs. time for two senders in a 5-to-1 setup

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**

Data Center Transport Requirements

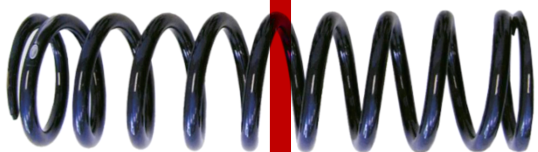
1. **High Burst Tolerance**
   - Incast due to Partition/Aggregate is common.
2. **Low Latency**
   - Short flows, queries
3. **High Throughput**
   - Continuous data updates, large file transfers

The challenge is to achieve these three together.

Tension Between Requirements

High Throughput

High Burst Tolerance



Low Latency

Deep Buffers:

- Qu
  Inc

Shallow Buffers:

&

Objective:

Low Queue Occupancy & High Throughput

Reduced RTO_{min} (SIGCOMM '09)

- Doesn't Help Latency

AQM – RED:

- Avg Queue Not Fast Enough for Incast

Quiz

Solutions to TCP "Incast" Problem?

- ☐ Smaller Packets
- ☐ Finer granularity timers
- ☐ Fewer acknowledgments
- ☐ More senders

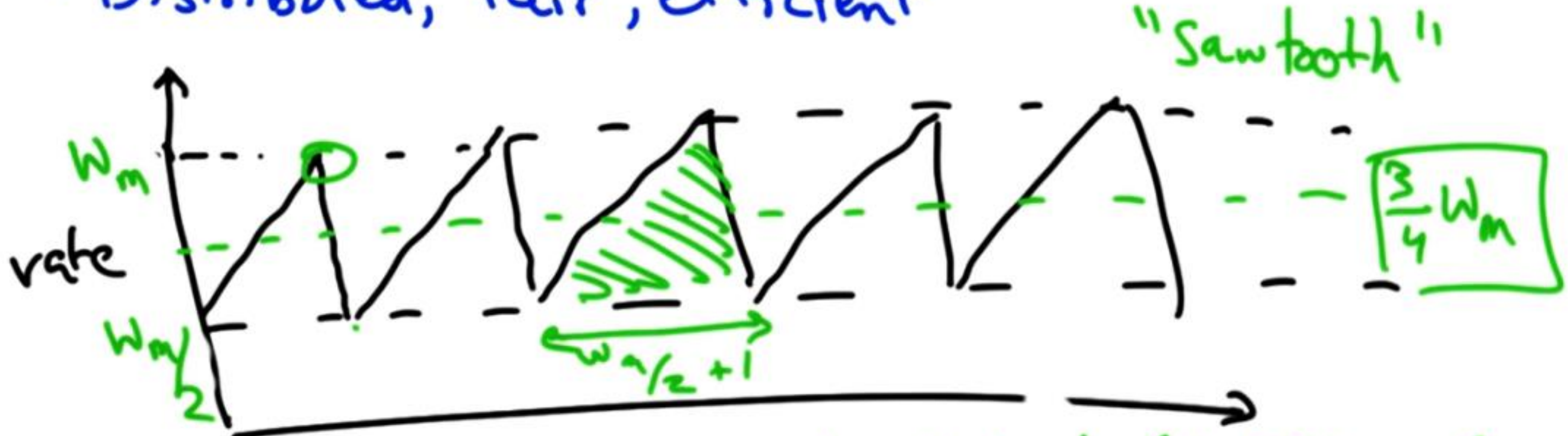
Quiz

Solutions to TCP "Incast" Problem?

- ☐ Smaller Packets
- ☑ Finer granularity timers
- ☑ Fewer acknowledgments
- ☐ More senders

AIMD (TCP Congestion Control)

- Distributed, fair, efficient



$\text{Tput} = \frac{3}{4} \cdot \frac{W_m}{RTT} = \lambda$

$\lambda \approx k / RTT \cdot \sqrt{p}$

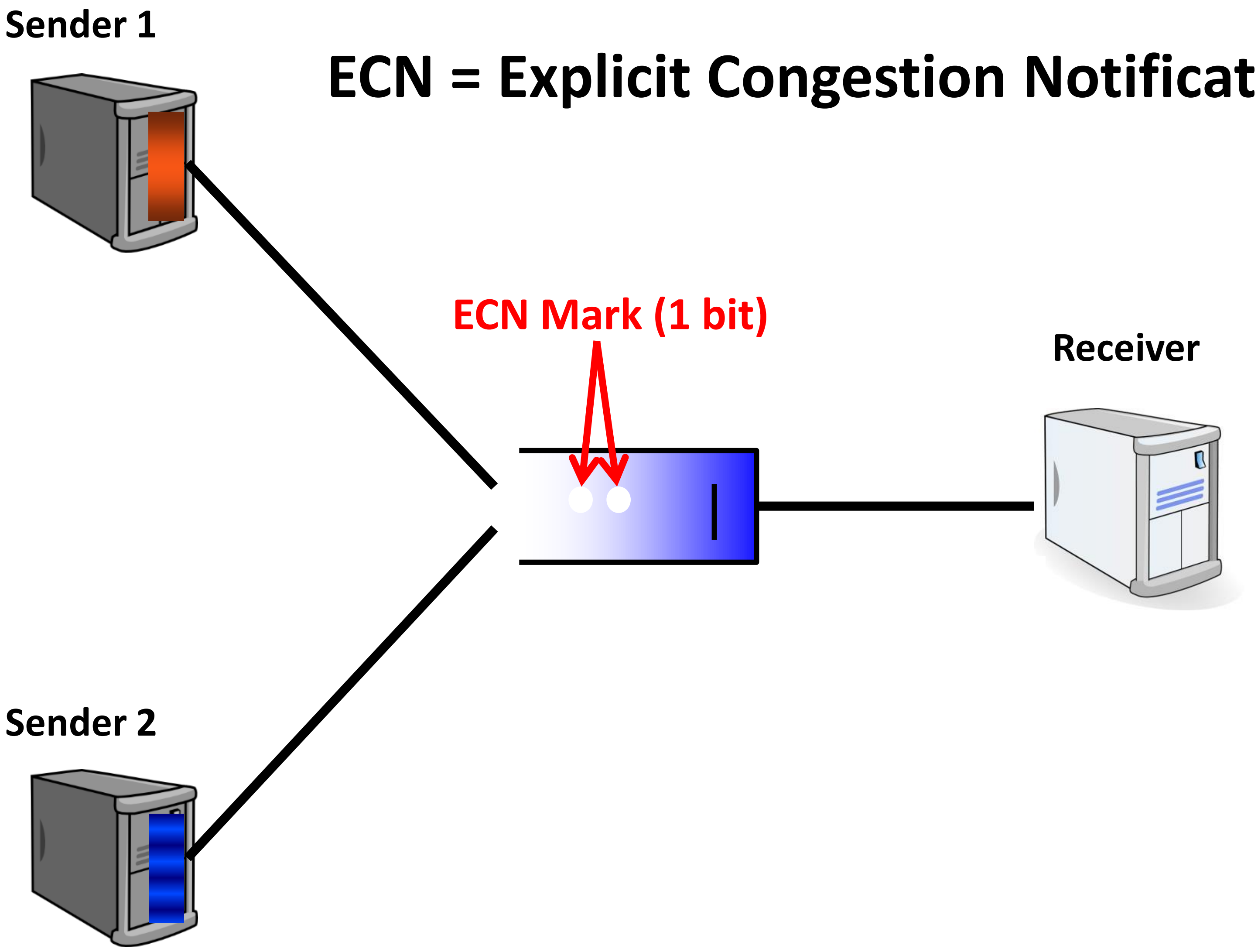
Loss Rate: $\frac{1}{2} \cdot \left(\frac{W_m}{2}\right)\left(\frac{W_m}{2} + 1\right)$

$1/p \approx \frac{W_m^2}{8} \rightarrow W_m \approx \left(k \cdot \sqrt{p}\right)^{-1}$

The DCTCP Algorithm

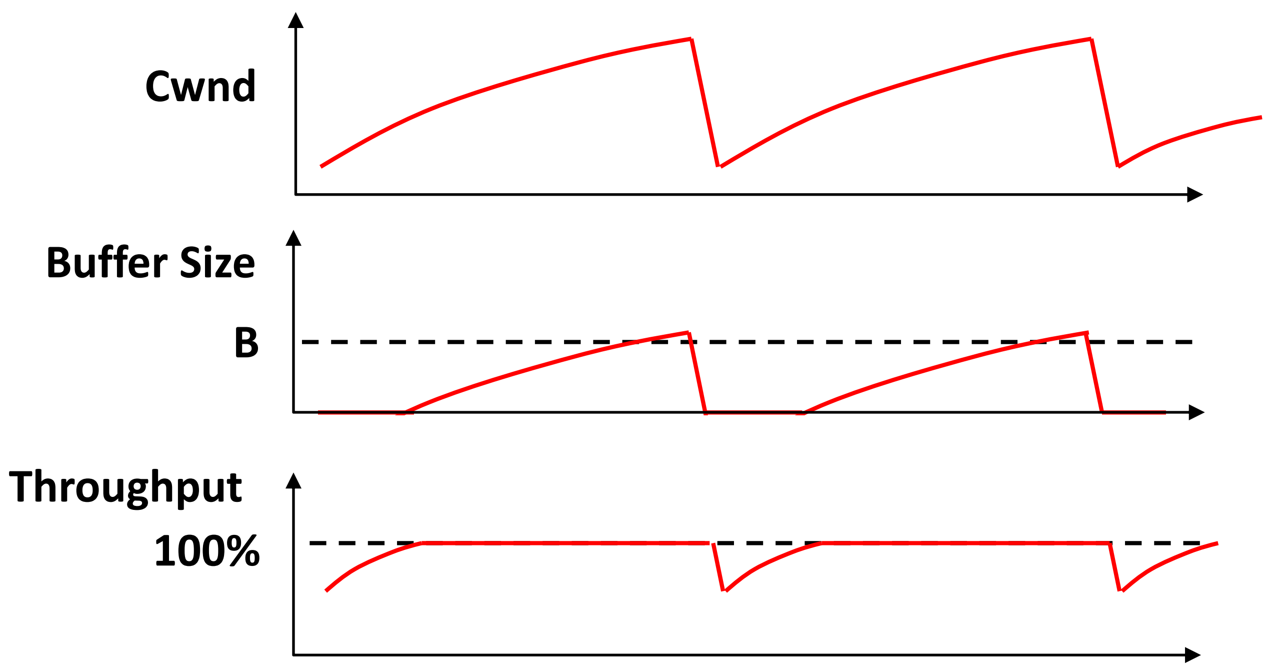
Review: The TCP/ECN Control Loop

ECN = Explicit Congestion Notification



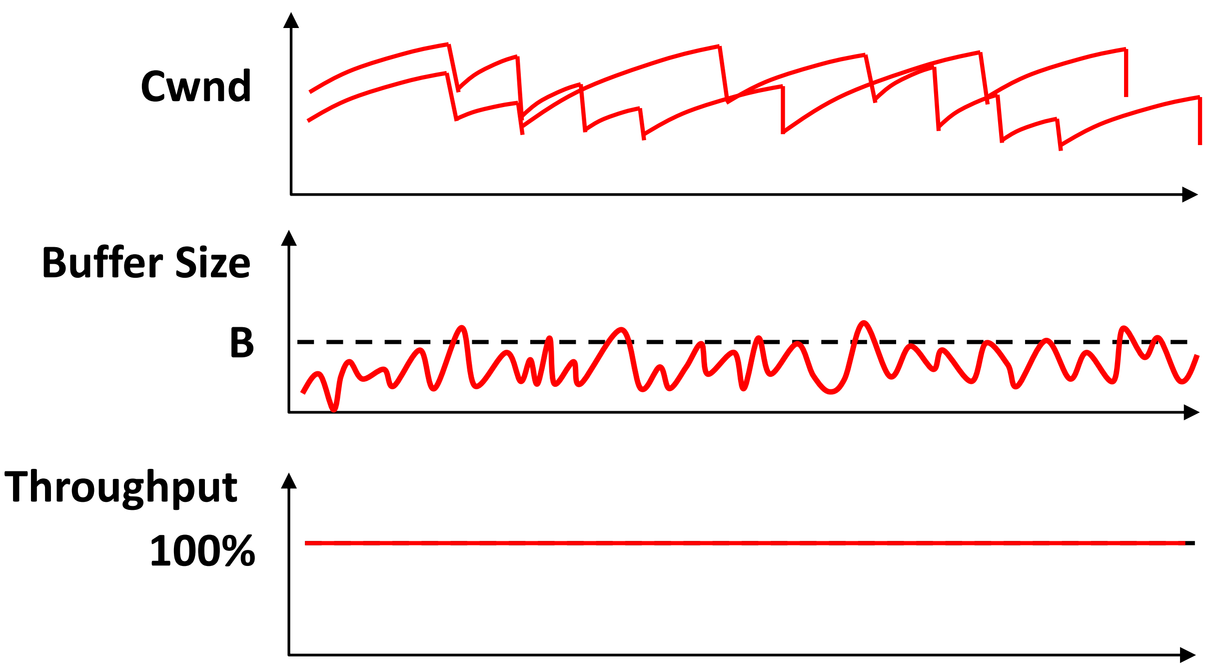
Small Queues & TCP Throughput:
The Buffer Sizing Story

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



Small Queues & TCP Throughput:
The Buffer Sizing Story

- Bandwidth-delay product rule of thumb:
  - A single flow needs $C \times RTT$ buffers for **100% Throughput.**
- Appenzeller rule of thumb (SIGCOMM ‘04):
  - Large # of flows: $C \times RTT/\sqrt{N}$ is enough.



Small Queues & TCP Throughput:
The Buffer Sizing Story

- Bandwidth-delay product rule of thumb:
  - A single flow needs $C \times RTT$ buffers for **100% Throughput.**
- Appenzeller rule of thumb (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 1-2 big flows** at each server**,** **at most 4.**

Small Queues & TCP Throughput:
The Buffer Sizing Story

- Bandwidth-delay product rule of thumb:
  - A single flow needs $C \times RTT$ buffers for **100% Throughput.**
- Appenzeller rule of thumb (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 1-2 big flows** at each server, **at most 4.**

Real Rule of Thumb:
Low Variance in Sending Rate → Small Buffers Suffice

Two Key Ideas

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

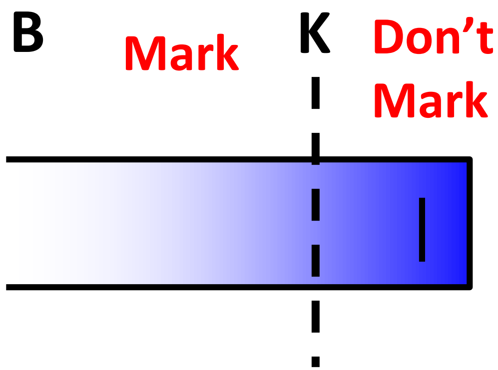
ECN Marks	TCP	DCTCP
1 0 1 1 1 1 0 1 1 1	Cut window by 50%	Cut window by 40%
0 0 0 0 0 0 0 0 0 1	Cut window by 50%	Cut window by 5%

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)**.

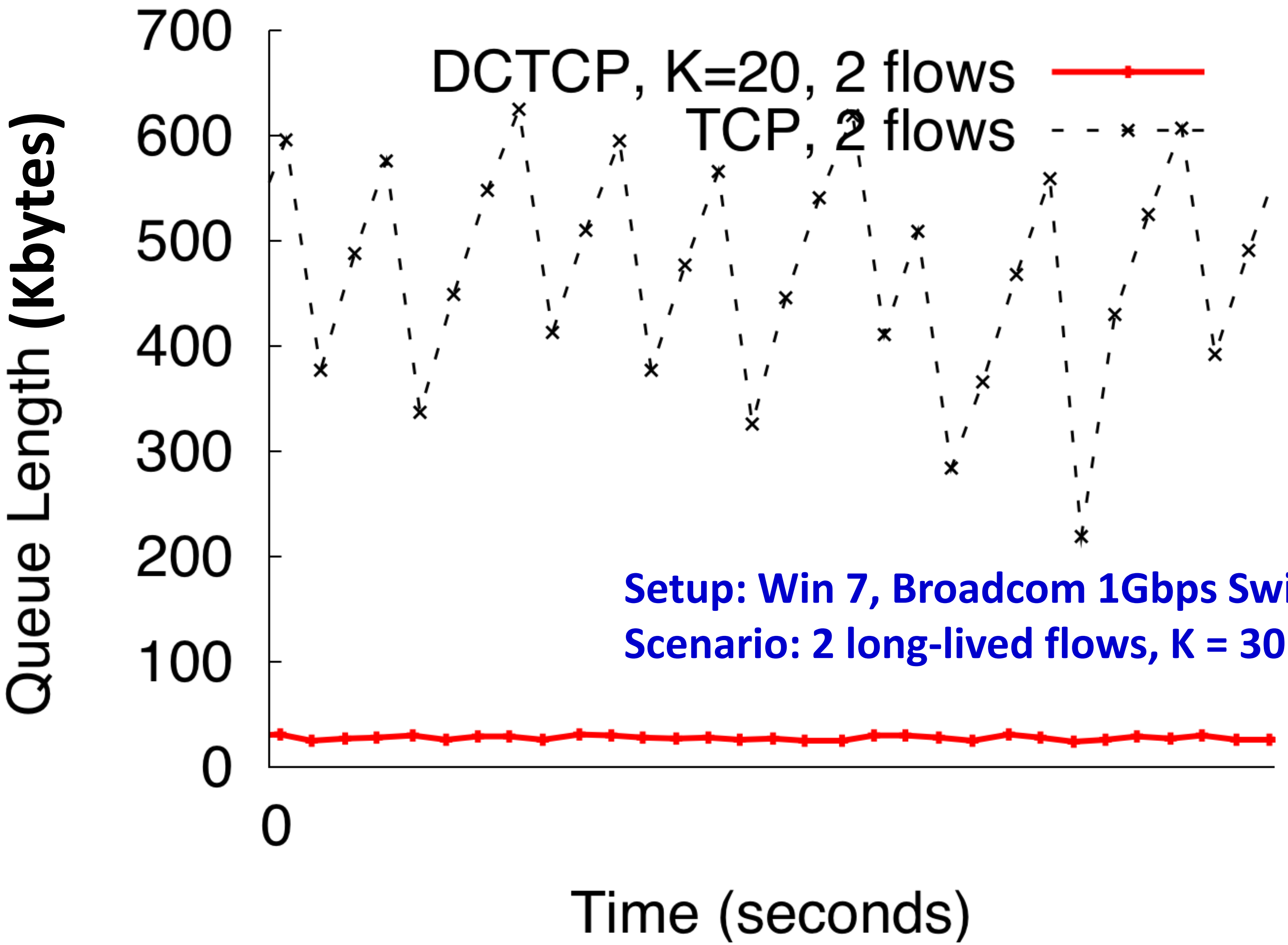
In each RTT:

$$F = \frac{\#\ of\ marked\ ACKs}{Total\ \#\ of\ ACKs} \qquad \alpha \leftarrow (1-g)\alpha + gF$$

➢ **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, low variance.

Evaluation

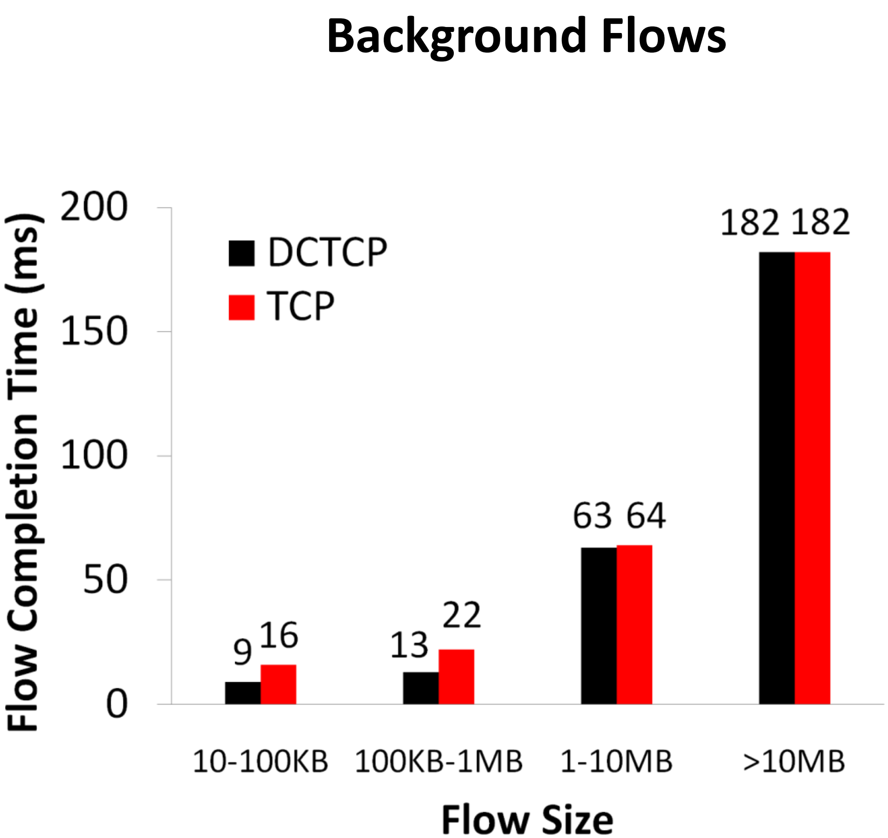
- Implemented in Windows stack.
- Real hardware, **1Gbps and 10Gbps** experiments
  - **90 server testbed**
  - **Broadcom Triumph** **48 1G ports – 4MB shared memory**
  - **Cisco Cat4948** **48 1G ports – 16MB shared memory**
  - **Broadcom Scorpion** **24 10G ports – 4MB shared memory**
- Numerous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

Cluster Traffic Benchmark

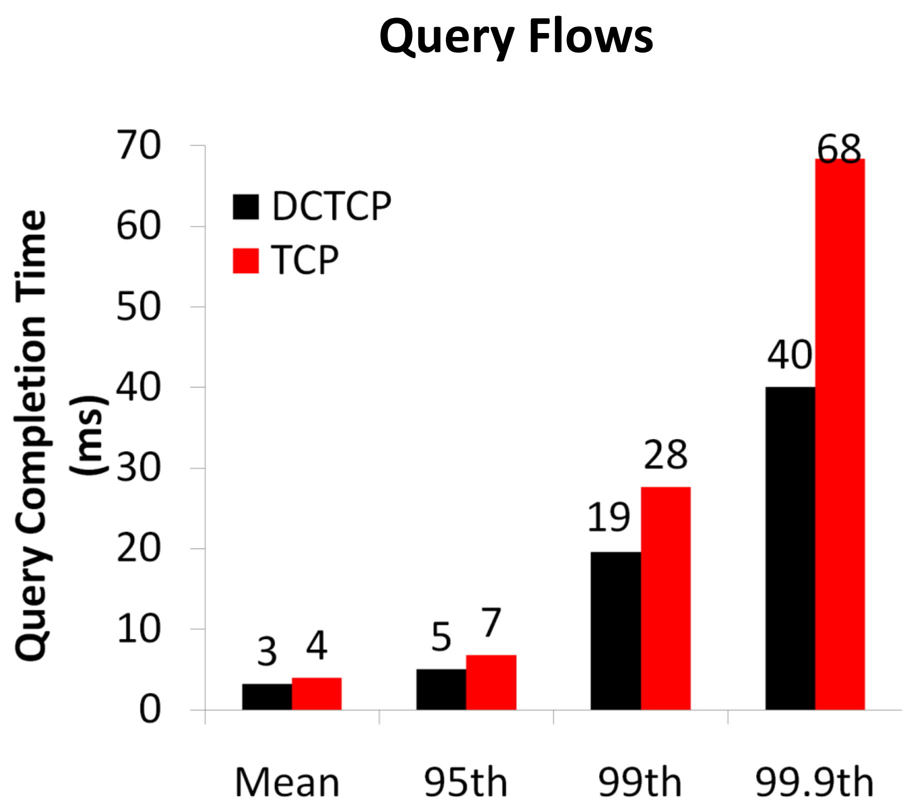
- Emulate traffic within 1 Rack of Bing cluster
  - 45 1G servers, 10G server for external traffic
- Generate query, and background traffic
  - Flow sizes and arrival times follow distributions seen in Bing
- Metric:
  - Flow completion time for queries and background flows.

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

Baseline

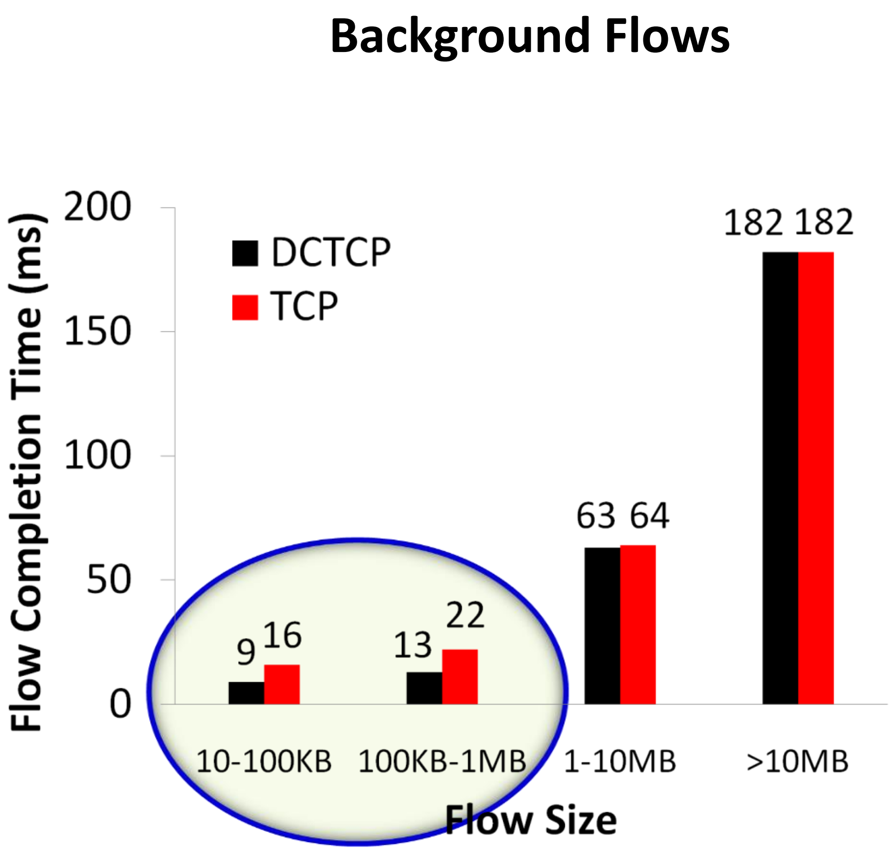


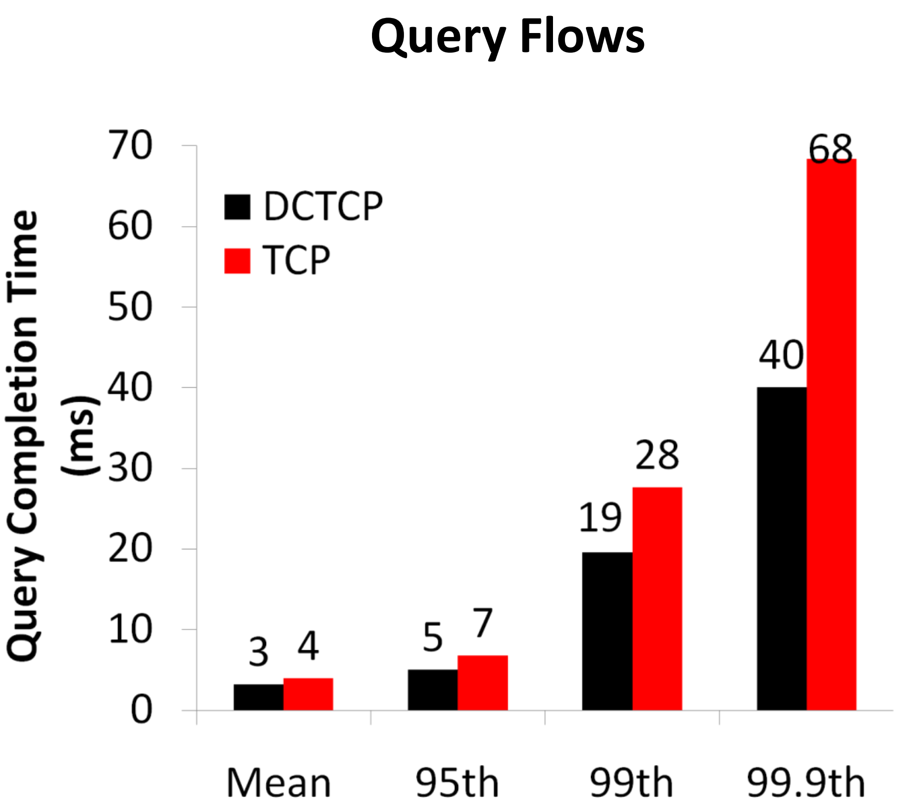
Background Flows
Flow Completion Time (ms)
200
150
100
50
0
DCTCP
TCP
9
16
13
22
63
64
182
182
10-100KB
100KB-1MB
1-10MB
>10MB
Flow Size



Query Flows
Query Completion Time (ms)
70
60
50
40
30
20
10
0
DCTCP
TCP
3
4
5
7
19
28
40
68
Mean
95th
99th
99.9th

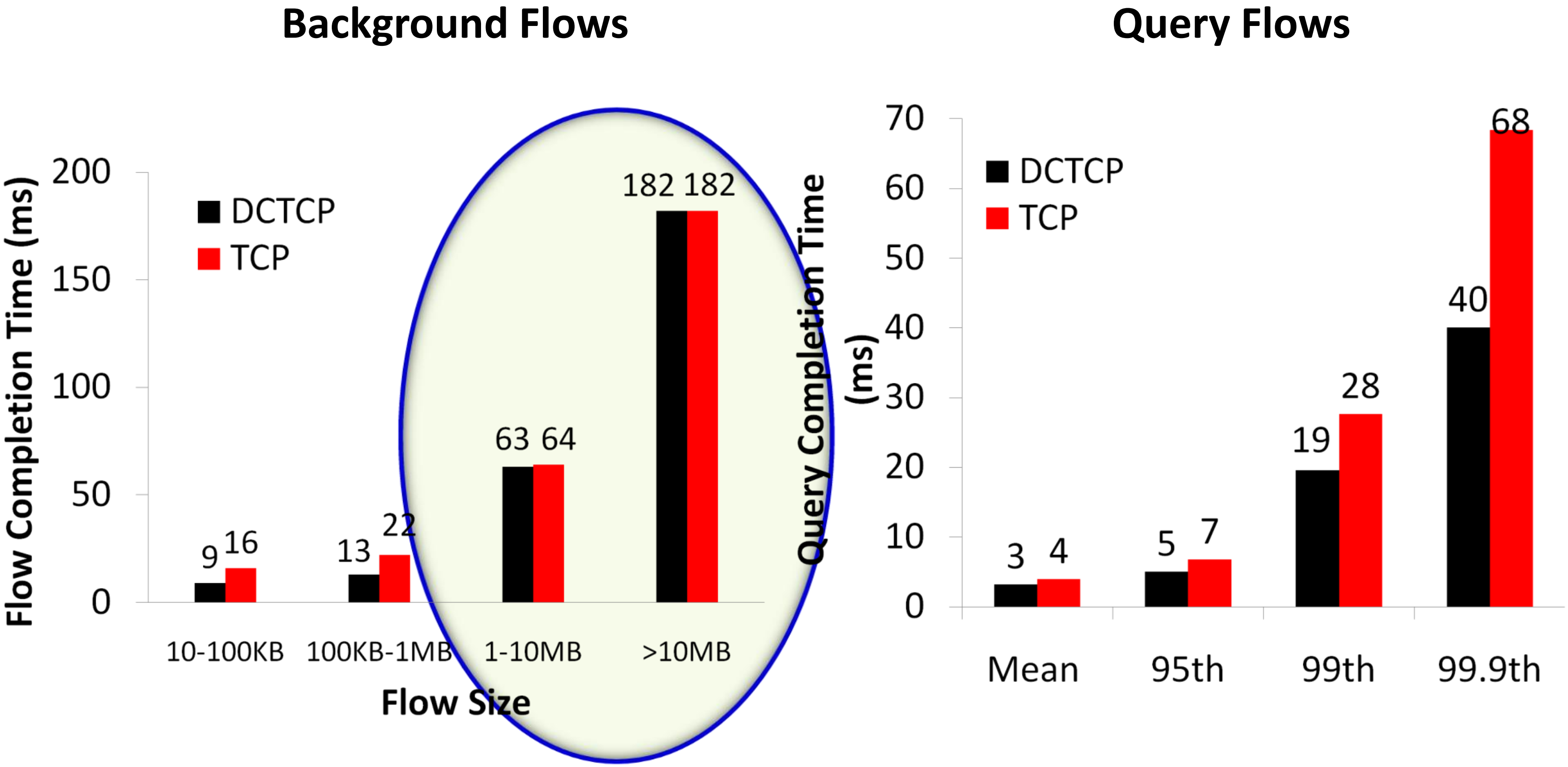
Baseline





✓ Low latency for short flows.

Baseline

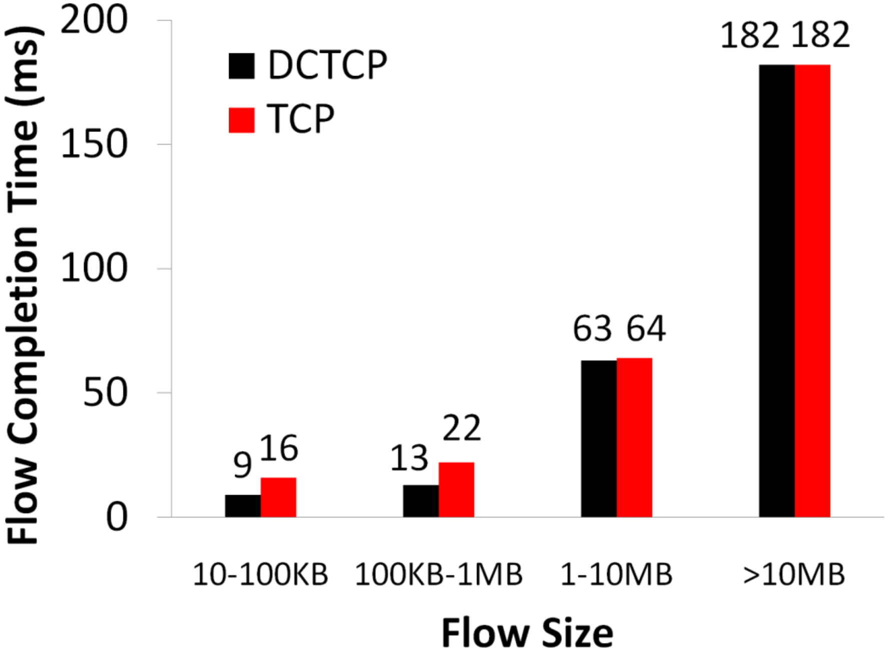


✓ Low latency for short flows.

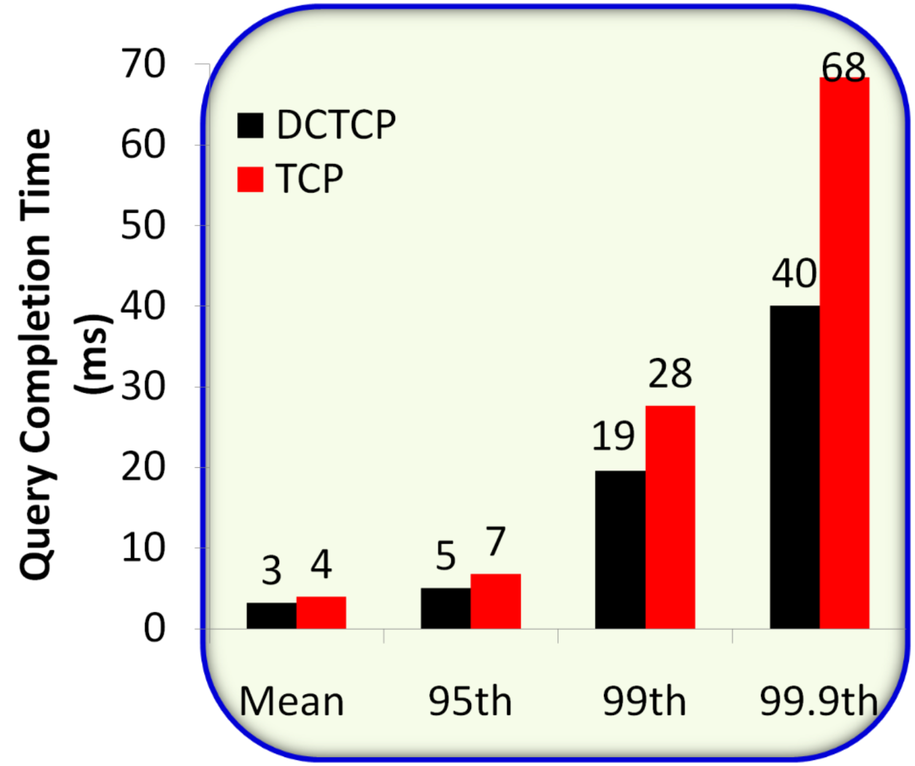
✓ High throughput for long flows.

Baseline

Background Flows



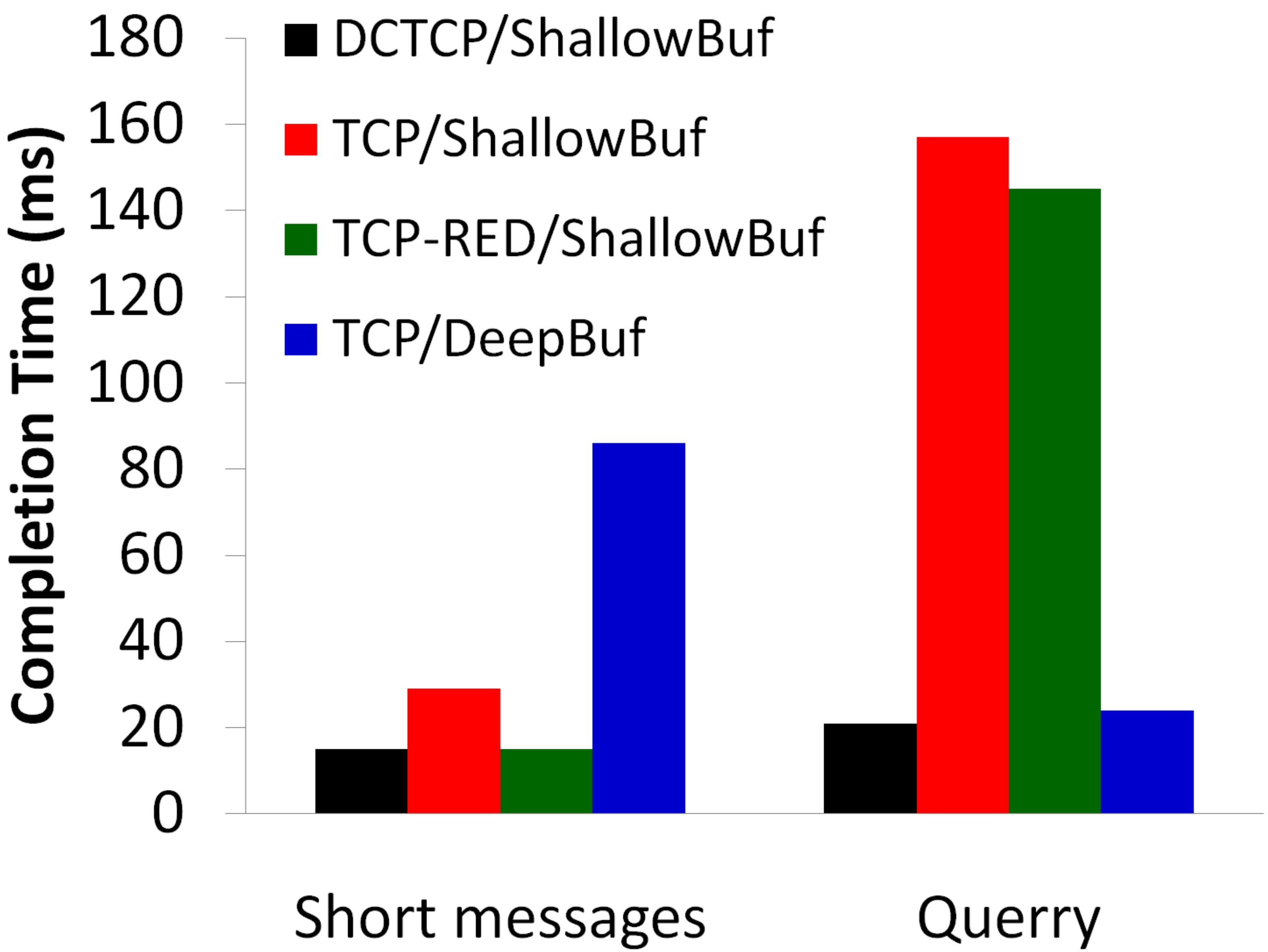
Query Flows



- ✓ Low latency for short flows.
- ✓ High throughput for long flows.
- ✓ High burst tolerance for query flows.

Scaled Background & Query

10x Background, 10x Query



Scalability

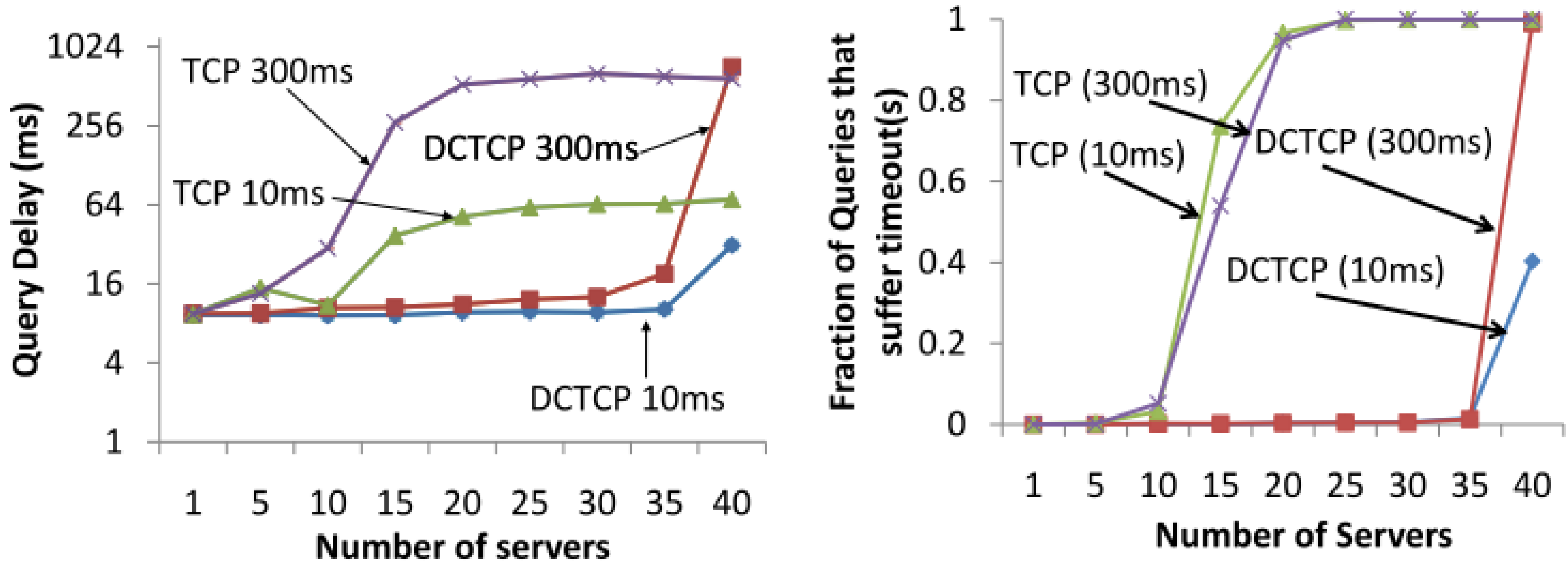


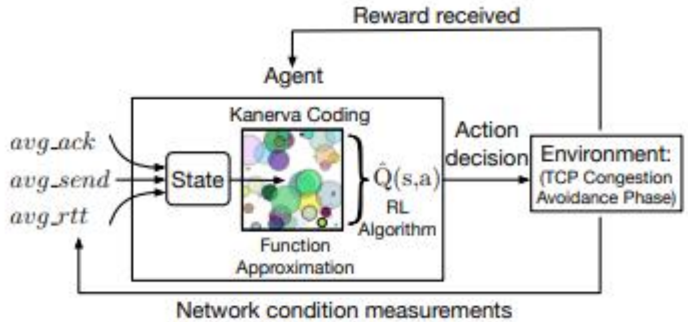
Figure 18: DCTCP performs better than TCP and then converges at 35 senders (log scale on Y axis; 90% confidence intervals for the means are too small to be visible).

QTCP: Adaptive Congestion Control with Reinforcement Learning

Wei Li, Fan Zhou, Kaushik Chowdhury, and Waleed Meleis

Next generation network access technologies and Internet applications have increased the challenge of providing satisfactory quality of experience for users with traditional congestion control protocols.
Efforts on optimizing the performance of TCP by modifying the core congestion control method depending on specific network architectures or apps do not generalize well under a wide range of network scenarios.
This limitation arises from the rule-based design principle, where the performance is linked to a pre-decided mapping between the observed state of the network to the corresponding actions. Therefore, these protocols are unable to adapt their behavior in new environments or learn from experience for better performance.
We address this problem by integrating a reinforcement-based Q-learning framework with TCP design in our approach called QTCP.
QTCP enables senders to gradually learn the optimal congestion control policy in an on-line manner.
QTCP does not need hard-coded rules, and can therefore generalize to a variety of different networking scenarios.
QTCP outperforms the traditional rule-based TCP by providing 59.5% higher throughput while maintaining low transmission latency.

- We describe QTCP, a Q-learning based congestion control protocol that automatically learns the effective strategies for adjusting the cwnd to achieve high throughput and low delay in an on-line manner.
- This fundamentally changes the design of previous NewReno-like TCP variants that require fixed, manually selected rules.



Solution framework of our RL-based TCP congestion control design.

- The framework of QTCP is shown above. The learning agent (sender) interacts with the network environments and keeps exploring the optimal policy by taking sequential actions (e.g., varying the cwnd) given feedback as it works to achieve its desired goal, i.e., large throughput and low latency.
- Like any typical RL problem, QTCP consists of the following elements:

• States: defined as informative perceptions or measurements that an agent can obtain from the outside environment. Here, the state is a unique profile of the network conditions evaluated through selected performance metrics.

• Actions: chosen by an agent at each time step, after perceiving its current state, according to a policy. In the context of congestion control, the action is the decision to increase, decrease, or leave unchanged the current cwnd.

• Reward: this reflects the desirability of the action picked. As we describe below, the reward is further specified by the value of a utility function, which is computed based on the measurement of flow throughput and latency. Higher throughput and lower latency translates into a higher utility value and vice-versa .

• Training algorithm: The purpose of the training algorithm is to learn the optimal policy to select certain action for each state. This is the central module of QTCP as it is responsible for developing the congestion control strategies.

- In general, QTCP works by checking the values of selected state variables and passing these state values to the currently trained policy to generate an action to adjust cwnd. Then QTCP observes the new state and the reward and uses them as an input to the training algorithm that evaluates and improves the cwnd changing policies.

States

- three state variables because they are significantly affected by network congestion and can be seen as efficient "congestion probes". For example, avg_send characterizes the traffic sending rate at the sender side and avg_ack reflects the real goodput measured at the receiver side. If there is no congestion, then ideally avg_send should be equal with avg_ack. On the other hand, avg_send < avg_ack indicates a high possibility of congestion and the sender should slow down its sending rate.
- three state variables described as followings:

- avg_send: the average interval between sending two packets.
- avg_ack: the average interval between receiving two consecutive ACKs.
- avg_rtt: the average RTT.

We calculate avg_send by taking the average of several packetsending intervals in a time window (one RTT) to reduce the estimation bias.

The avg_send and avg_rtt are calculated in a similar way.

All values are represented in milliseconds

Actions

TABLE : *cwnd* modification options

Change in *cwnd*	Extent of change (bytes)
Increase	10
Decrease	-1
No change	0

selection of actions(3) is the key to the QTCP's performance. An action specifies how QTCP should change its cwnd in response to variations in the network environments.
The first action increases the cwnd by 10 bytes.
The second action reduces the size of cwnd by 1 byte making it possible to reduce the congestion issue in the network flow and
the last action does nothing to the size of cwnd letting the cwnd remains the same as before.
The reason why we assign a relatively large value, i.e., 10, to increase the size of cwnd and at the same time assign a relatively small value, i.e., 1, to reduce the size of cwnd is that we intend to encourage the agent to quickly increase the cwnd to utilize the bandwidth while still offering an option to decrease the sending rate when necessary when applying our learning algorithm to TCP congestion control, our learning agent makes the action decisions in every tinterval time interval, allowing the action taken in previous state have enough time to occur on the network flow which also allows the agent accurately measure resulted throughput and RTT since it takes certain time for the sender to count ACKs received (the ACKs are used to measure the throughput and RTT by our learning agent) with respect to those latest sent packets.

Utility Function and Reward Definition

- A utility function specifies the objective of QTCP. The goal of the QTCP is to find the decision policy of cwnd that can maximize the value of the utility function for each sender. While QTCP can take a variety of different objectives, we choose proportional fairness and define the utility function as follows:- $Utility = \alpha \times \log(\text{throughput}) - \delta \times \log(\text{RTT})$

Conclusions

- 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.**
  - ✓ **Based on mechanisms already available in Silicon.**

Discussion

- What if traffic patterns change?
  - E.g., many overlapping flows

- What do you like/dislike?