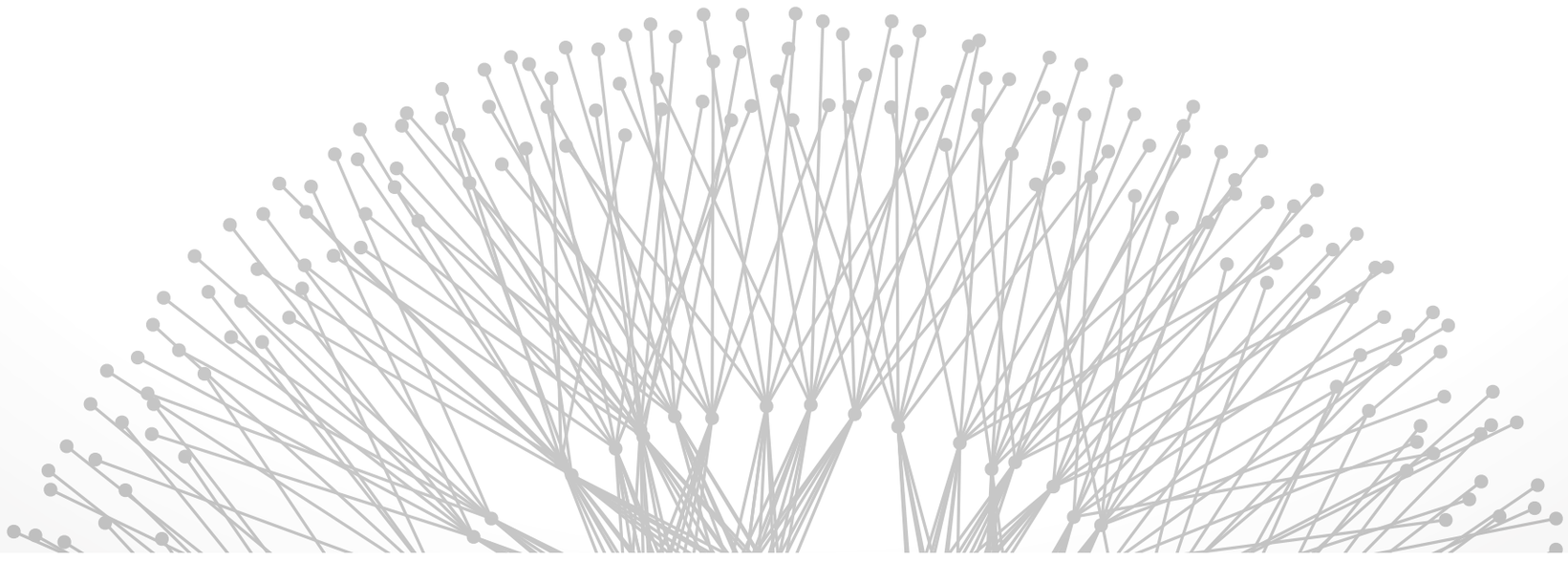
Congestion Control in Data Centers

TA: Chi-Yao Hong
CS 538 Sept 12 2013



Background: Data centers

Why data centers important?

Google | Data Centers

Data centers › Inside look › Locations

The Billion Dollar Data Centers

By: Rich Miller

April 29th, 2013

An overhead view of the server infrastructure in Google's data center in Council Bluffs, Iowa, where the company has invested more than $1 billion. (Photo: Connie Zhou for Google)

Microsoft now has one million servers – less than Google, but more than Amazon, says Ballmer

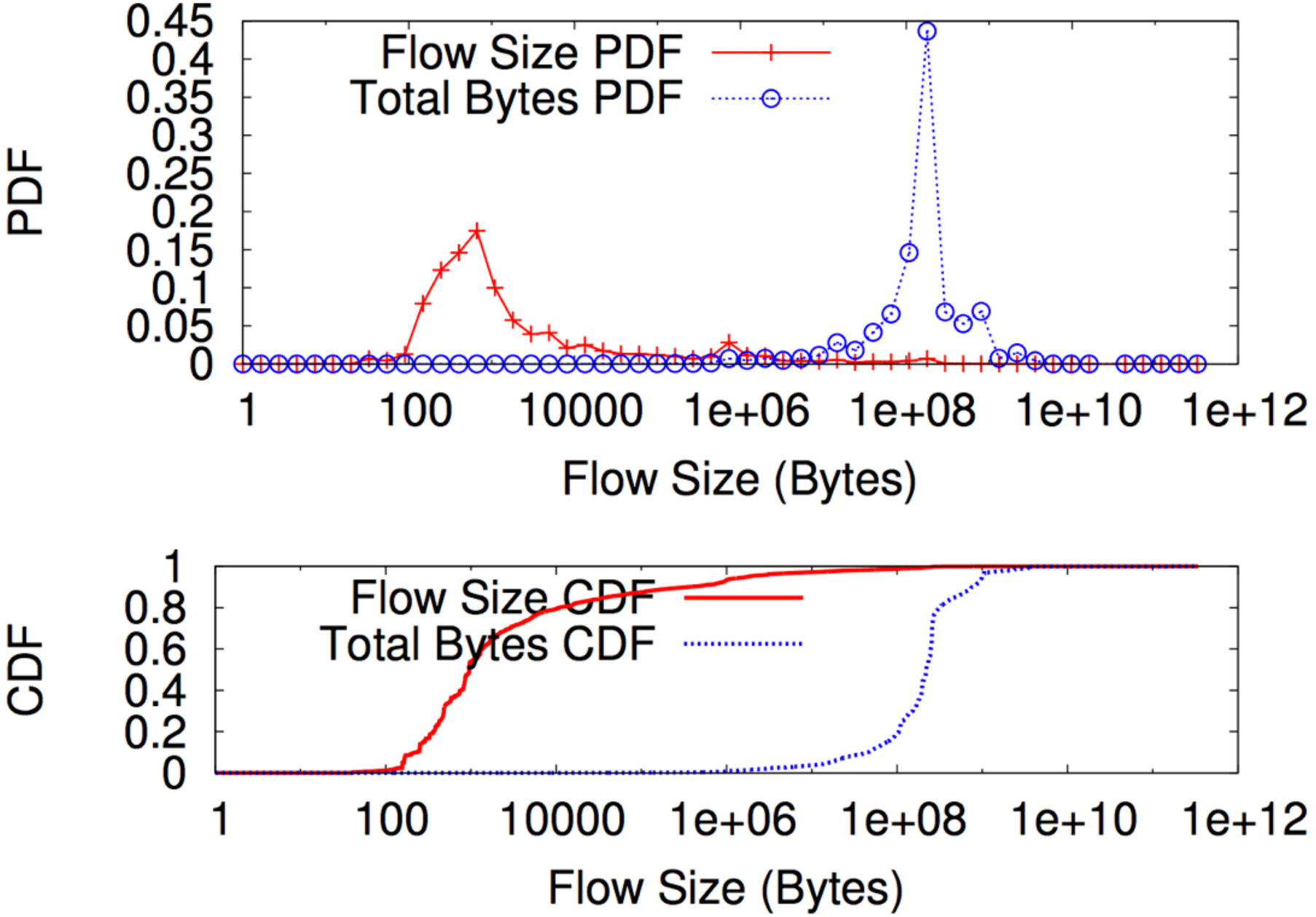
1 pm | 18 Comments

190 | 47 | +1 | Share

At Microsoft's 2013 Worldwide Partner Conference, CEO Steve Ballmer gave us a very interesting tidbit about the scale of Microsoft's server operations. "We have something over a million servers in our datacenter infrastructure."

to say that "Google is bigger" and "Amazon is a little bit such direct figures; in almost two decades, Google and gh figure on their server count — and now Ballmer is on

Data center traffic characteristics



[VL2, SIGCOMM'09]

What do we want?

Short flows

complete flows before their deadlines

Long flows

no deadline, but still preferable to finish earlier

Low latency is the key

Example: web-facing apps have strict latency requirements

amazon.com

Revenue decreased by 1% of sales
for every 100 ms latency

Low latency is the key

YAHOO!

400 ms slowdown resulted
in a traffic decrease of 9%

[Yslow 2.0; Stoyan Stefanov]

Google

100 ms slowdown reduces
searches by 0.2-0.4%

[Speed matters for Google Web Search; Jake Brutlag]

AOL

Users with lowest 10% latency viewed 50% more
pages than those with highest 10% latency

[The secret weapons of the AOL optimization team; Dave Artz]

mozilla Firefox

2.2 sec faster web response
increases 60 million more Firefox
install package downloads per year

[Firefox and Page Load Speed; Blake Cutler]

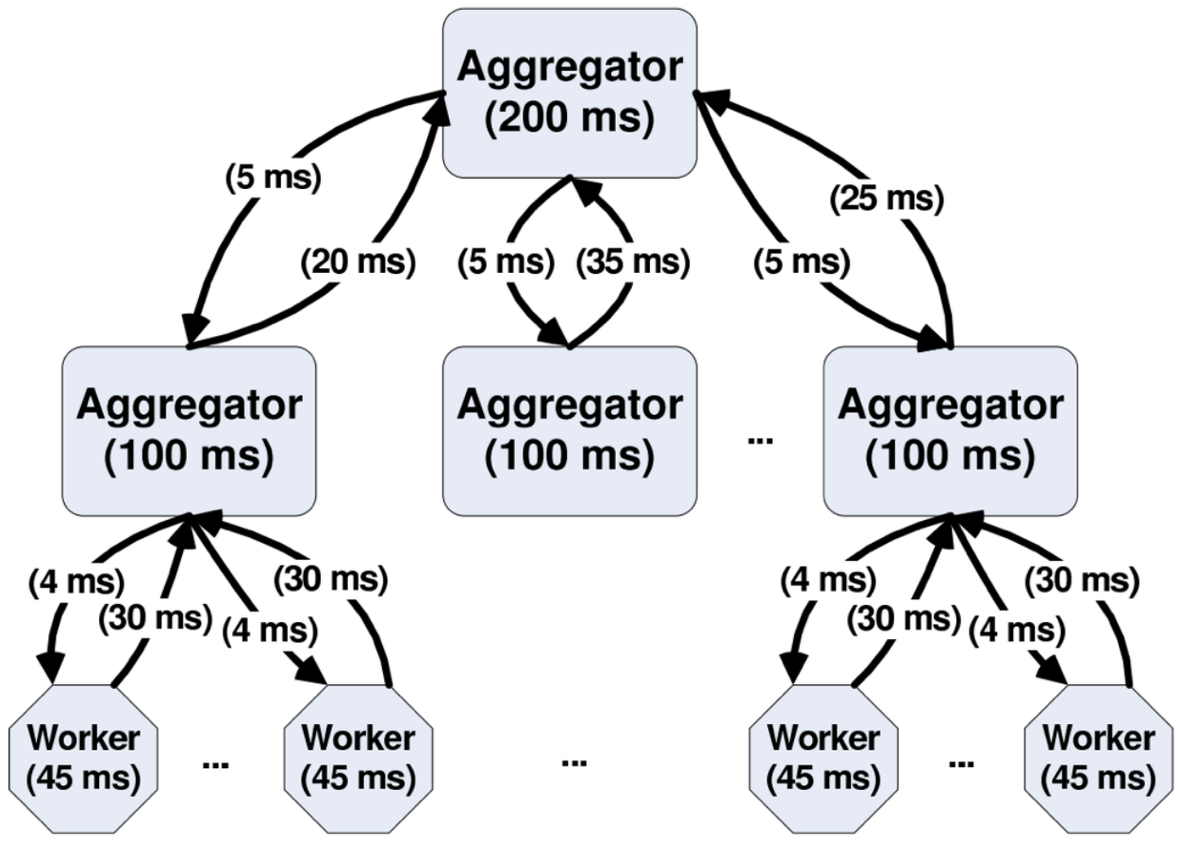
Walmart

Users with 0-1 sec load time have
2x conversion rate of 1-2 sec

[Is page performance a factor of site
conversion? And how big is it; Walmart Labs]

Improving latency in data centers

Server side optimization: Parallel computation



partition aggregate model

Improving latency in data centers

Network side optimizations

Physical interconnect

- Full-bisection bandwidth topology [Fat-tree, SIGCOMM'08] [VL2, SIGCOMM'09]
- Server-centric topology [BCube, SIGCOMM'09]
- Random graph [Jellyfish, NSDI'12]

Hybrid architecture

- add wireless [Flyways; SIGCOMM'11] [3D beamforming; SIGCOMM'12]
- add optical switching networks [OSA, NSDI'13]

Switch-side optimization

- detour [Zarifis; SIGCOMM'13 poster]

How does TCP congestion control perform in data centers?

3 impairments [DCTCP]

- Incast
- Queue buildup
- Buffer pressure

Incast

What is TCP Incast problem?

- Synchronized flows overflow the switch buffer

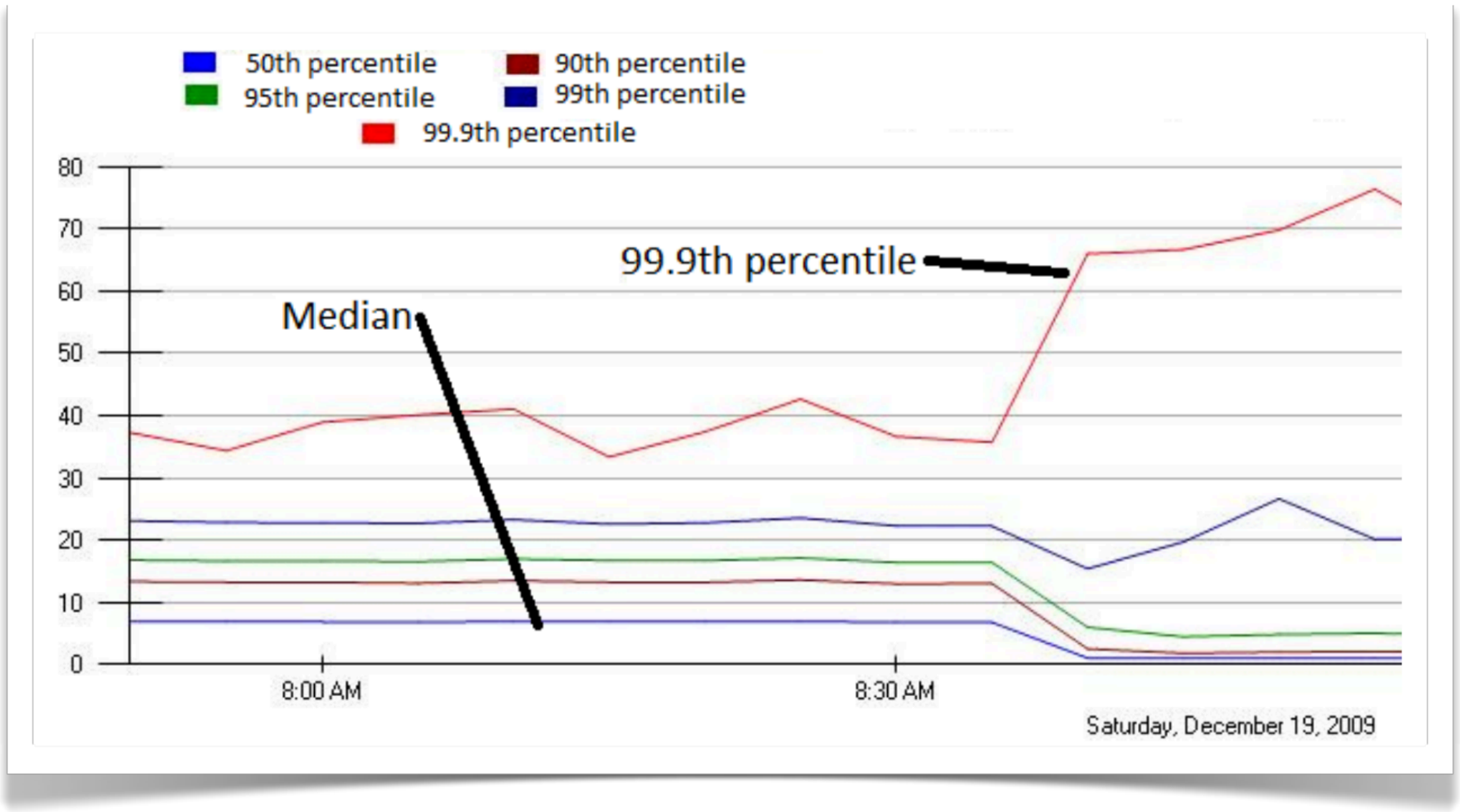
Causes?

- (Barrier) synchronized many-to-one traffic pattern
- Short flows (10s KB to 100s KB)
- Small queue buffer (4 to 8 MB shared memory)
- Large default RTO (300 ms)

Fixing TCP Incasts

- Use larger switch buffers
- Decrease RTOmin
- Desynchronize flows (random delay ~10ms)

Query completion time [ms]



Queue buildup and buffer pressure

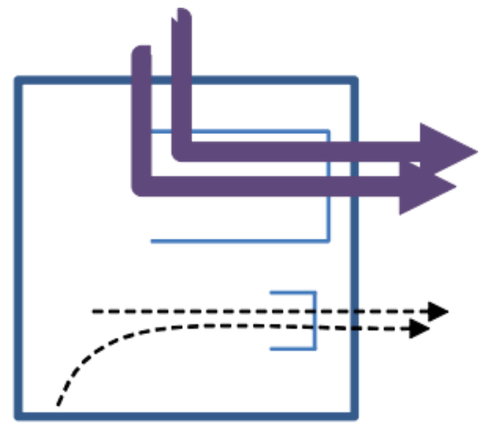
Causes: Long TCP flows occupy switch buffer

Queue buildup: short flow experiences increased delay

90%: RTT < 1ms --- (Bing's DC)

10%: 1 ms < RTT < 15 ms

Buffer pressure: 4 MB shared memory, i.e., how much buffer per port is not a constant



Many solutions to Incast do not apply here...

DCTCP

[Alizadeh et al., SIGCOMM'10]

(adapted from Alizadeh's slides)

DCTCP: Two goals

Goal #1: Low latency and high burst tolerance

- Ensuring low queue occupancy

Goal #2: Still having high throughput for long flows

- Using most of the network bandwidth

Achieve either goal is not hard; what's hard is to achieve both

Explicit Congestion Notification

Switches mark packet's ECN bit *before* buffer overflows

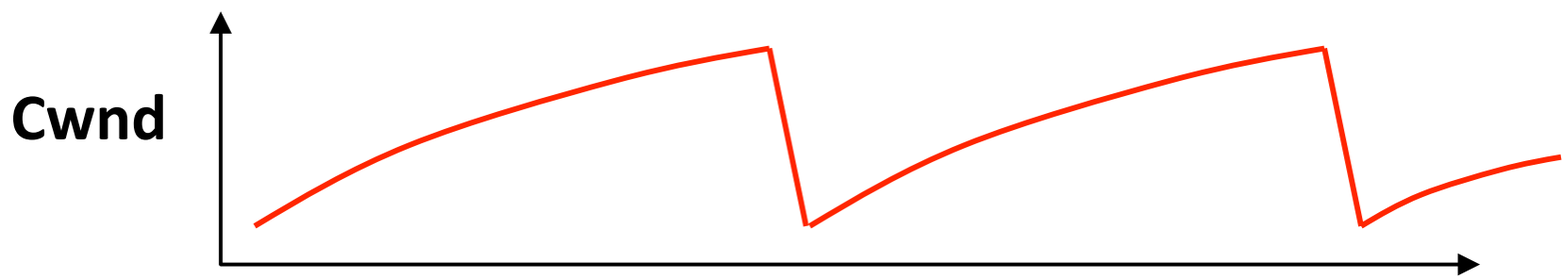
TCP sender treats ECN signals as if a single packet is dropped — but packets are not actually dropped

More useful for short flows — avoid packet drop, therefor avoid RTO timeout.

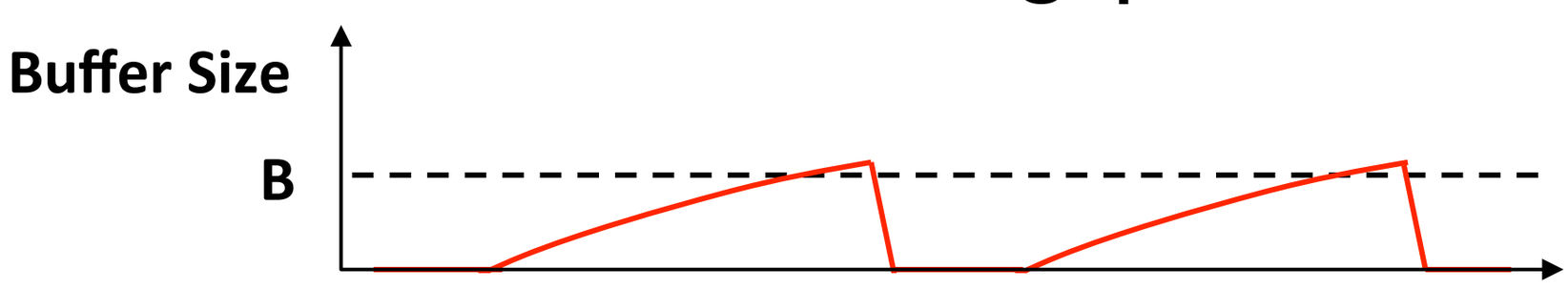
Well supported by today's commodity switches and end-hosts

Buffer requirements in TCP

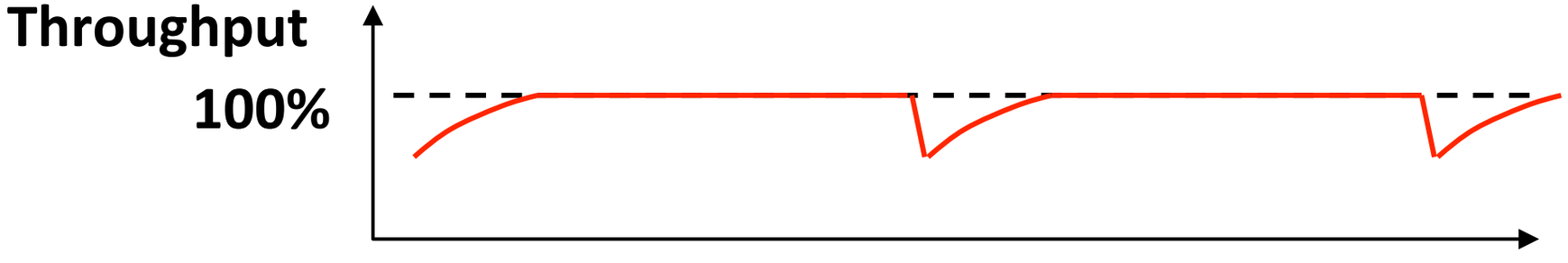
TCP sawtooth behavior:



Small buffer leads to low throughput:



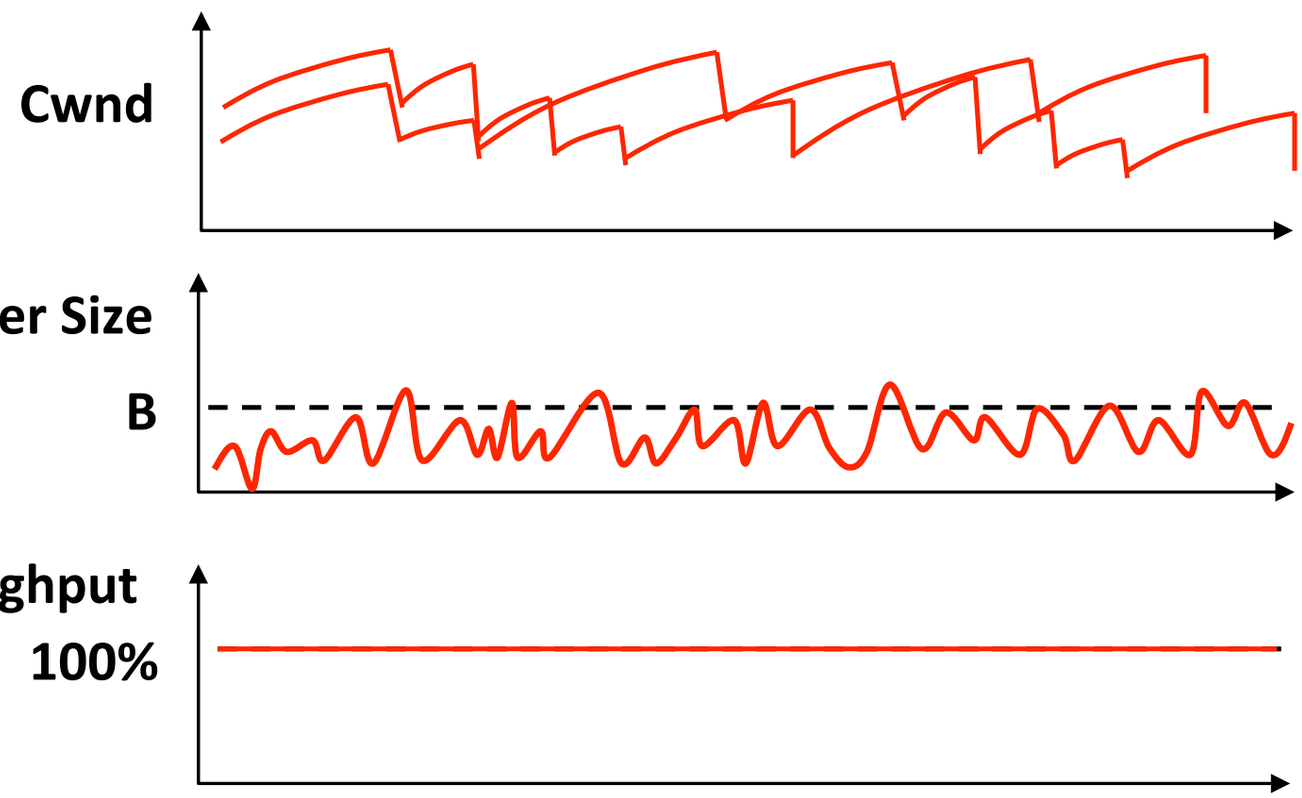
A single flow needs *C* x *RTT* buffer for 100% throughput



Buffer requirements in TCP

For large # of flows: $C \times RTT / \sqrt{N}$ is enough

[Appenzeller et al; SIGCOMM'04]



But low statistical multiplexing in data center networks

- 75th percentile: 2 long flows per server

DCTCP: Two Key ideas

1. React in proportion to the extent of congestion, not its presence

ECN Marks	TCP	DCTCP
1011110111	cut window by **50%**	cut window by **40%**
0000000001	cut window by **50%**	cut window by **5%**

2. Mark based on instantaneous queue length

   - Fast feedback to better deal with bursts

DCTCP Algorithm

Switch side:

- mark packet iff queue length > K

Sender side:

- maintain running avg of fraction of marked pkts

In each RTT:

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

- adaptive window decreases: $Cwnd \leftarrow (1 - \frac{\alpha}{2})Cwnd$

Why does it work?

Small buffer occupancies

→ bursts fit

→ low queueing delay

Aggressive marking when queue buffer builds up

→ fast react before packet drops

Adaptive window reduction

→ high throughput

Discussion

- DCTCP mitigates three impairments. Does this give you optimal latency in data center networks?

- Can we use DCTCP in wide area networks?

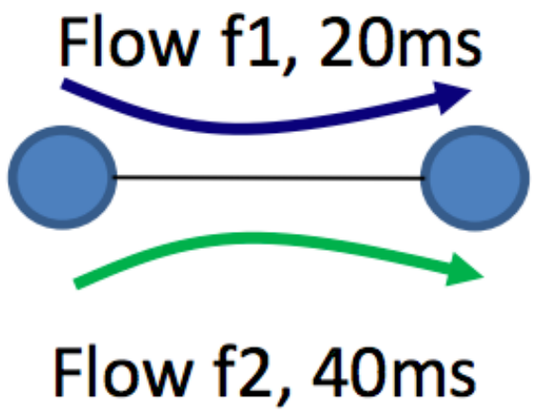
- Can we use other switch features to improve the performance?

  - Alok Tiagi's point

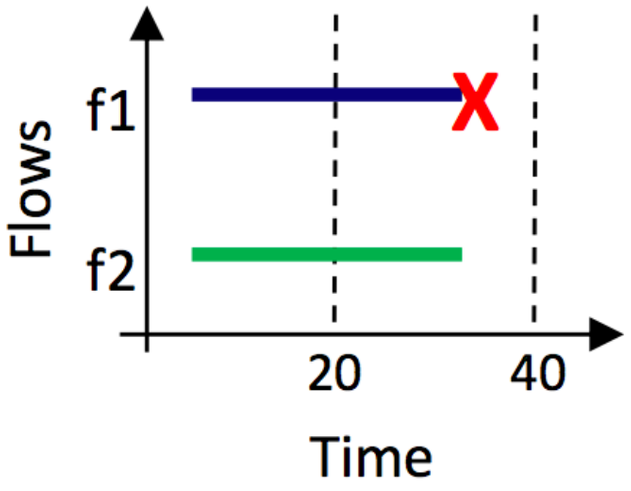
Can we finish flows even faster?

A case for unfair sharing

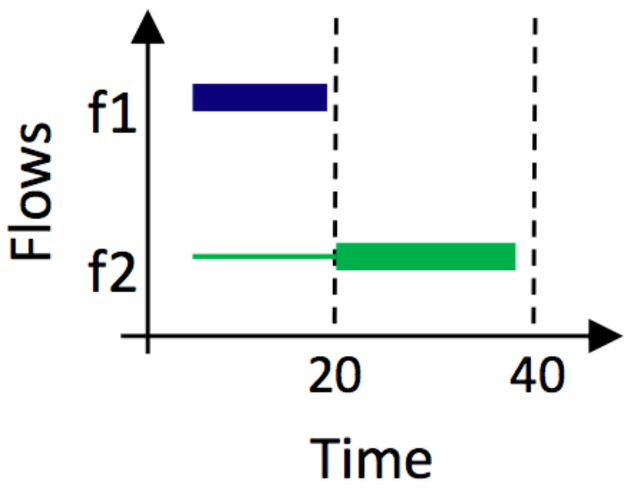
Scenario



Fair sharing



deadline aware



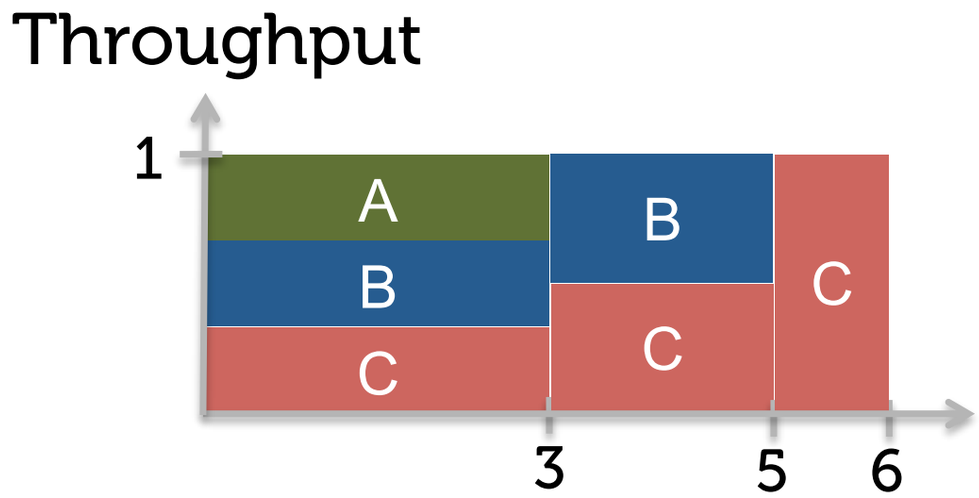
Flow f1 misses its deadline

(incomplete response to user)

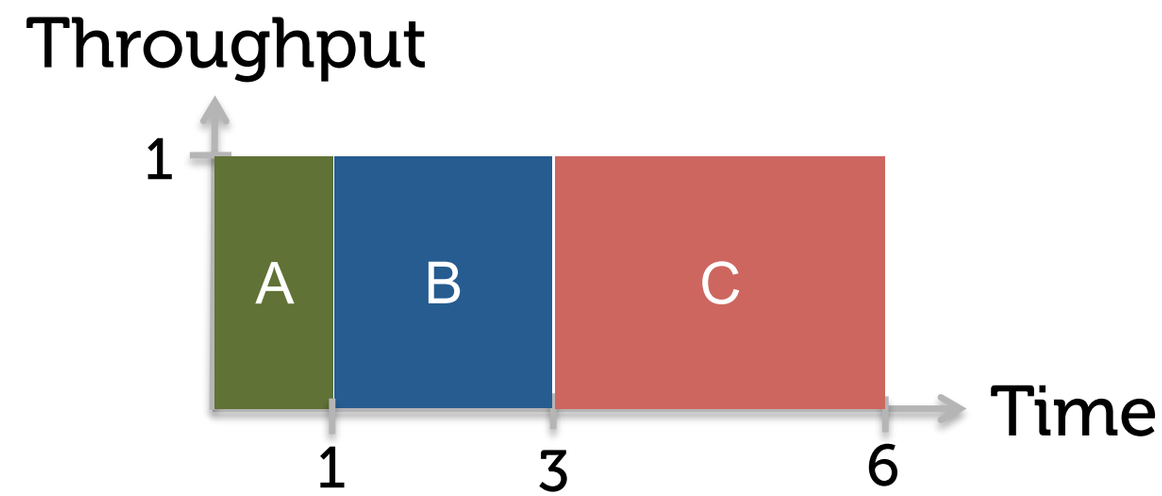
Another case for unfair sharing

Scenario

Flow (A, B, C)
with size (1, 2, 3)
no deadline



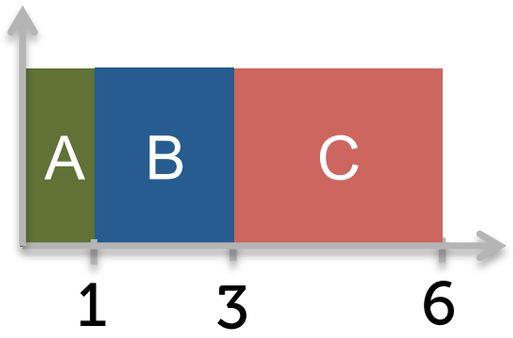
mean flow completion time = $\frac{3+5+6}{3}$ = 4.67



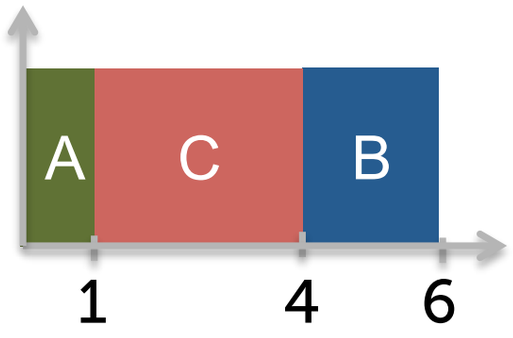
mean flow completion time = $\frac{1+3+6}{3}$ = 3.33

29% saving in mean

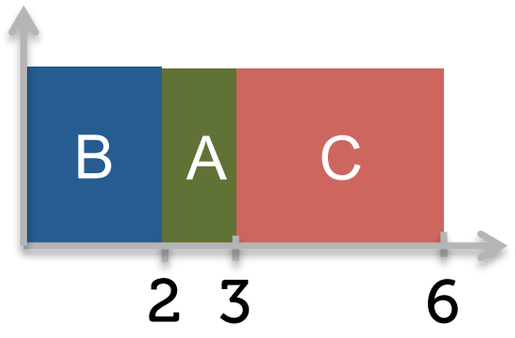
Order matters



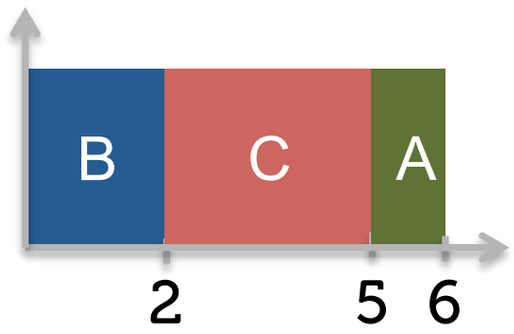
mean: 3.33 ✔



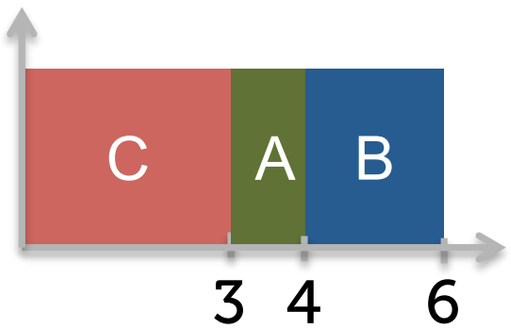
mean: 3.67



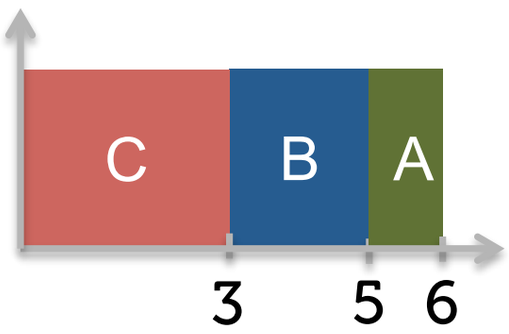
mean: 3.67



mean: 4.33



mean: 4.33



mean: 4.67 ✘

Relaxing fairness constraints help

Order matters

PDQ: Preemptive Distributed Quick flow scheduling

[Hong et al.; SIGCOMM'12]

Pretty Damn Quick

PDQ: Idea

plug in any desirable value

Scheduling flows based on flow criticality

//

relative priority of flows;
transmission order

PDQ: Two primitives

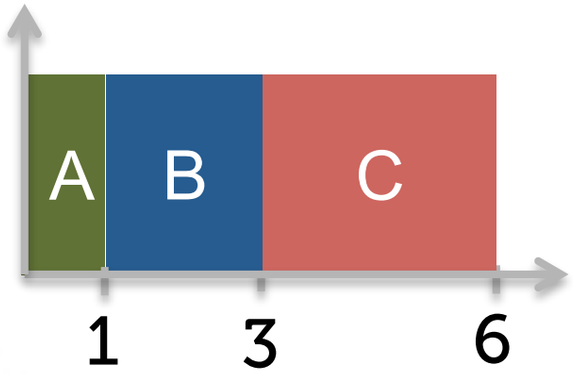
Preemptive scheduling

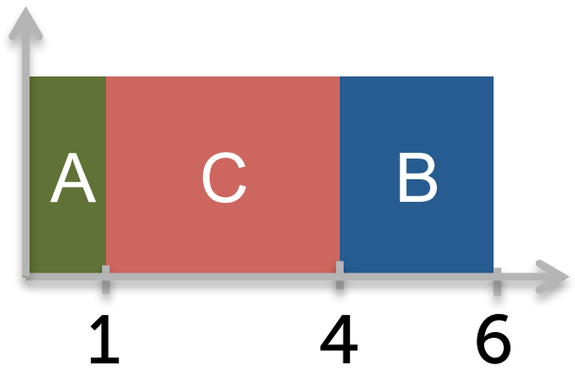
Less-critical flows yield to critical flows

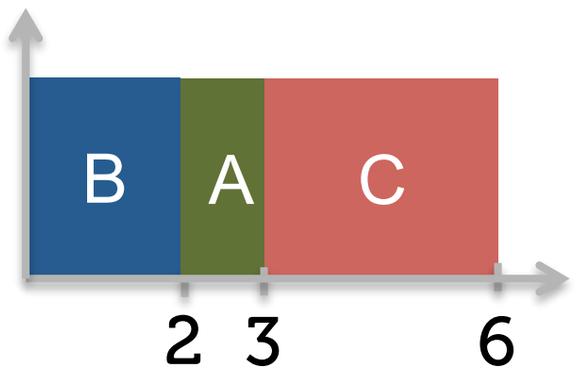
Dynamic scheduling

Flow criticality may change over time

How to choose flow criticality?







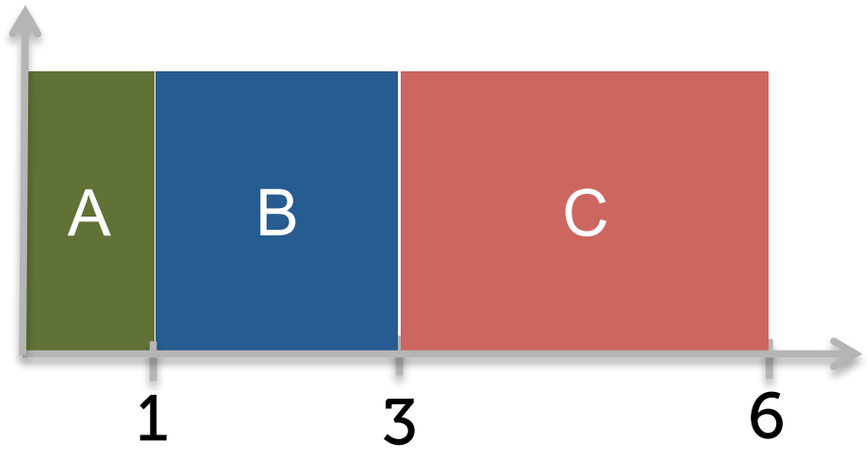
How to choose flow criticality?

Scheduling discipline

choose

choose

choose



PDQ's scheduling disciplines

EDF (Earliest Deadline First)

Optimal for satisfying flow deadlines

EDF + SJF

EDF if there's deadline; give preference to deadline flows

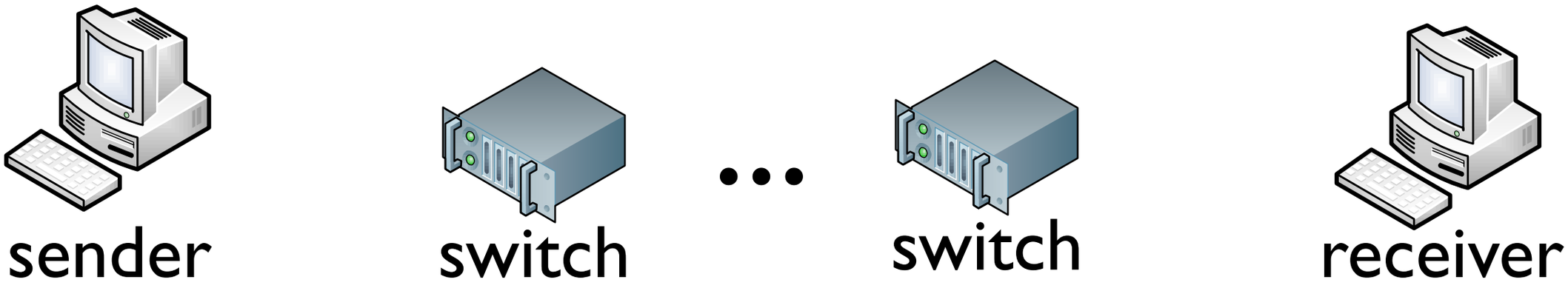
SJF (Shortest Job First)

Optimal for minimizing mean flow completion time

Policy-based

Assignment that reflects business priorities

PDQ Algorithm



- sender appends flow criticality on packet header
- switch preferentially allocates bandwidth to flows and tag flow sending rate on packet header
- sender sends with rate given by packet header

pFabric: Minimal Near-Optimal Datacenter Transport

[Alizadeh et al.; SIGCOMM'13]

(based on Alizadeh's slides)

pFabric in 3 sentences

- Packets carry a single priority number
- Switches use very small buffer (10-20 KB per port) and send highest priority / drop lowest priority packets
- Hosts send/retransmit aggressively with a minimal rate control to prevent congestion collapse

Why it works

Buffers are very small (~1 BDP)

- e.g., C=10Gbps, RTT=15us → BDP = 18.75 KB

Worst-case: ~300 packets (with minimal size of 64 B)

- 51.2 ns to find the highest/lowest priority of at most ~300 numbers
- binary tree implementation takes log2(300)=9 clock cycles
- current ASICs clock cycle = 1-2 ns

Minimal rate control

Flow starts at line rate

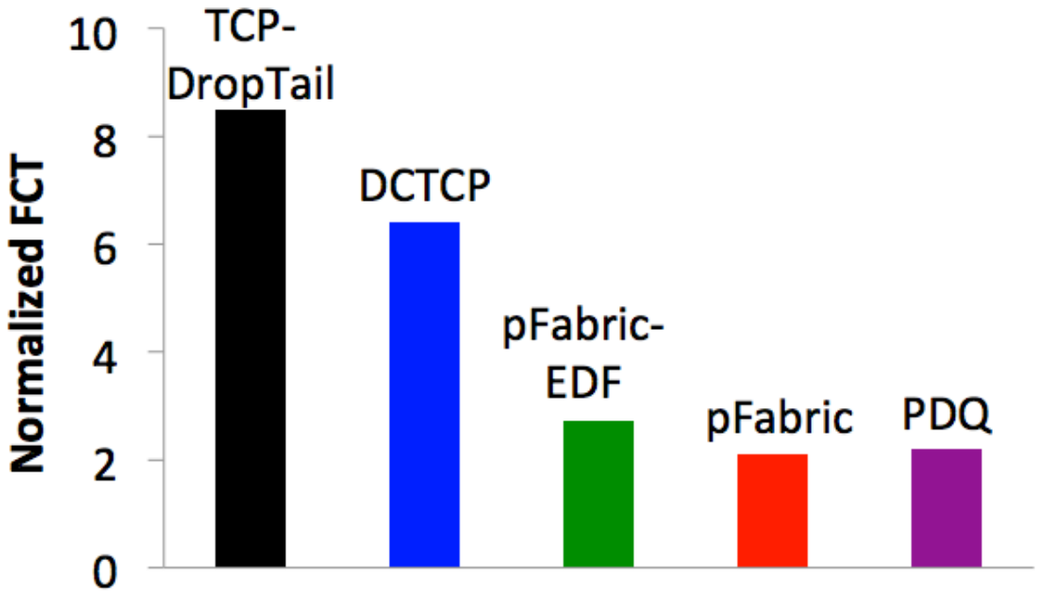
Additive increase for every ACK

No fast retransmits, no dupACKs detection

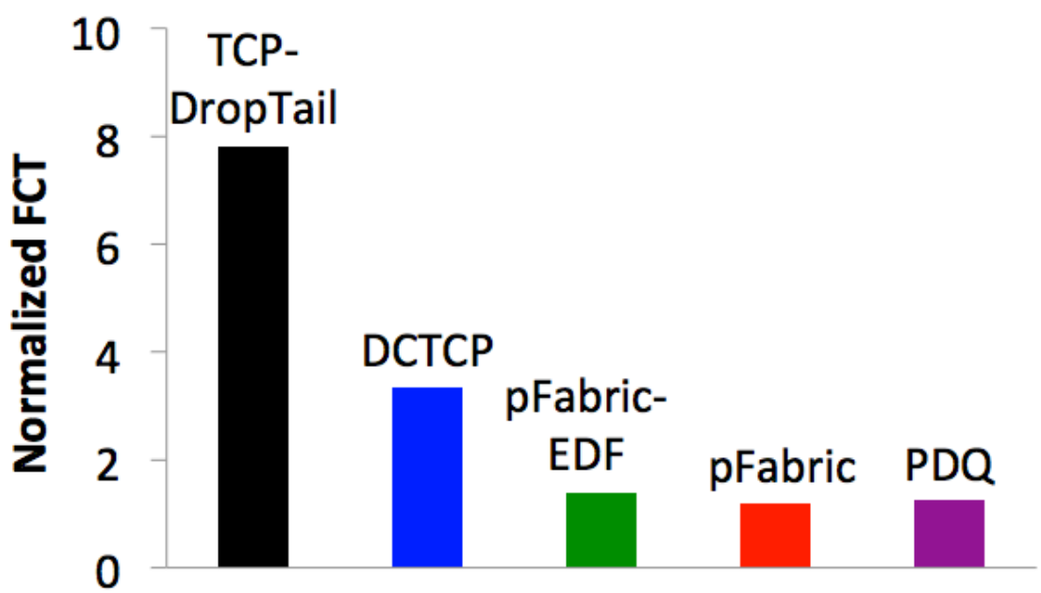
Timeout = 3 times fabric RTT

If timeout too many times, enter probe mode (sending only probe packet with 1-byte payload) and resume when it receives ACK

Evaluation



(a) Web search workload



(b) Data mining workload

Software Defined Transport

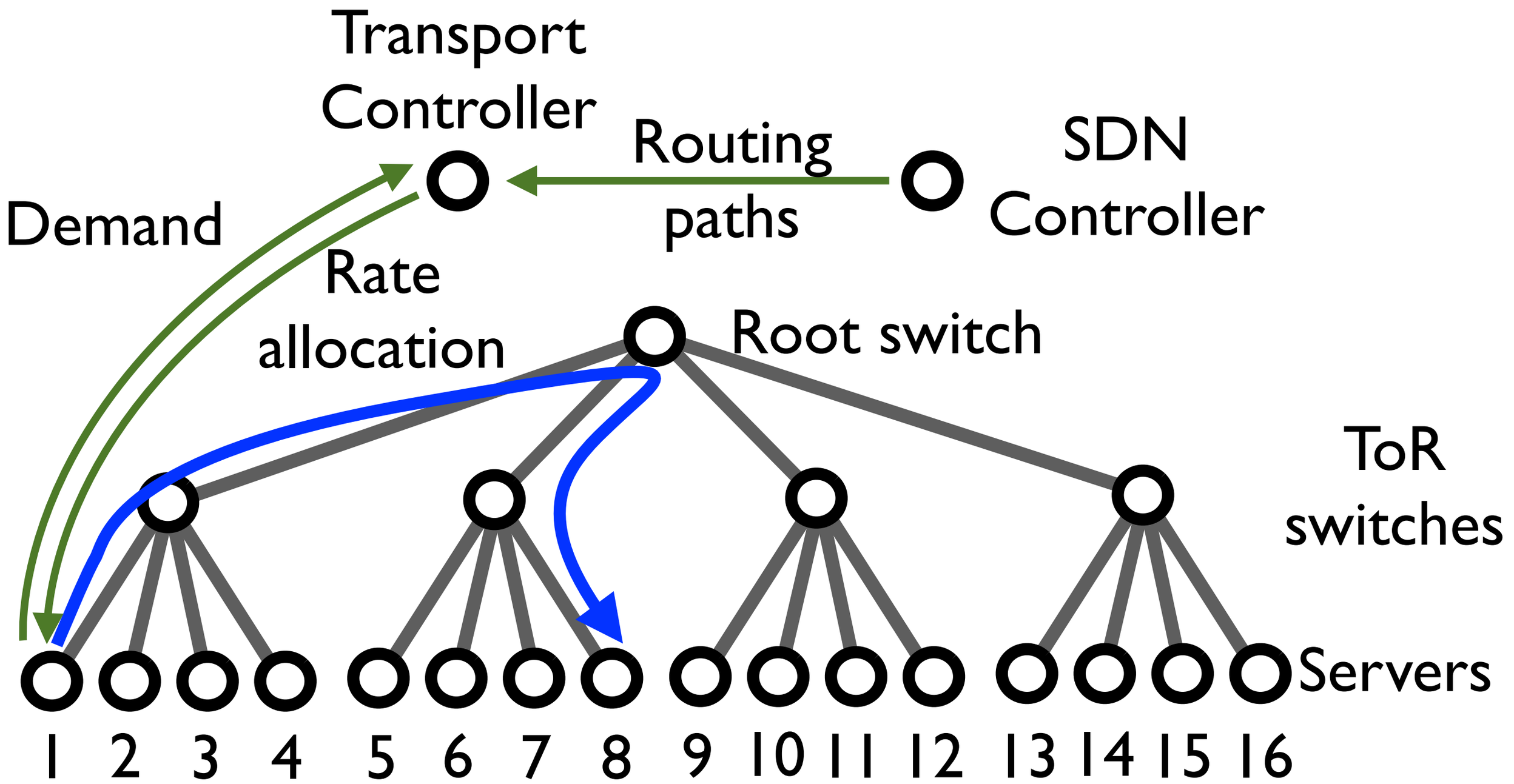
Goal

A flexible and deployable congestion control protocol that supports a wide range of transport policies:

- Weighted max-min fairness
- Flow prioritization
- Application-aware scheduling (e.g., job-level allocation)

... without modifying switches!

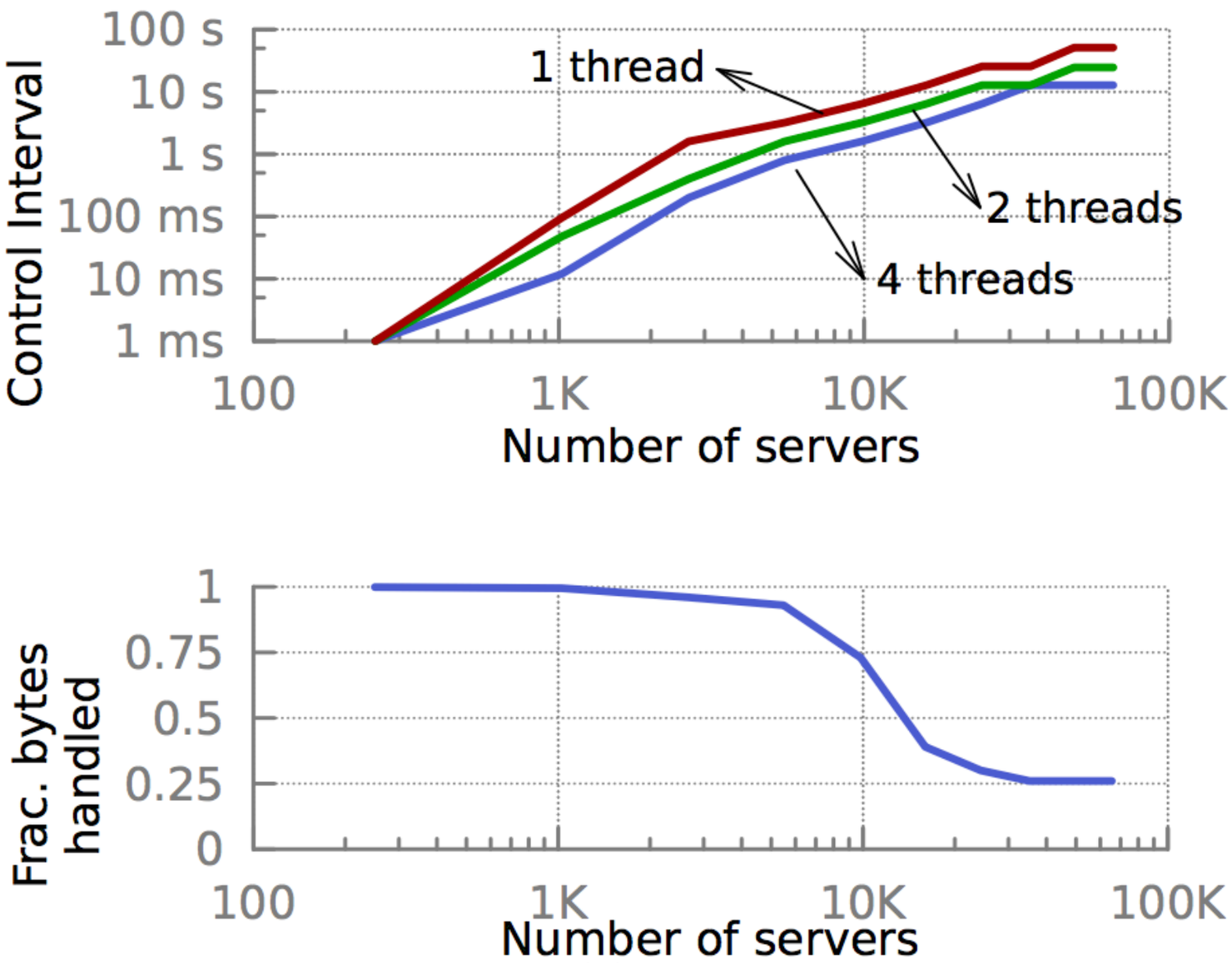
Design


Transport Controller
Routing paths
SDN Controller
Demand
Rate allocation
Root switch
ToR switches
Servers
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

Scalable rate allocation

- We handle short, transient flows without the controller
- A multi-threaded algorithm to simulate the fluid-level forwarding behavior on every network link
  - Each link is a thread
  - Based on input flow rate, derives the output flow rates and signals the allocation to downstream neighboring links
  - Using per-link dirty bit to avoid unnecessary checking (without placing mutex)

Evaluation


Control Interval
100 s
10 s
1 s
100 ms
10 ms
1 ms
1 thread
2 threads
4 threads
100
1K
10K
100K
Number of servers
Frac. bytes handled
1
0.75
0.5
0.25
0
100
1K
10K
100K
Number of servers

Project idea

Announcements

Announcements

Assignment 1 due next Tuesday

Next week reading: Congestion control in the network