6.888
Lecture 5:
Flow Scheduling

Mohammad Alizadeh

Spring 2016

Datacenter Transport

Goal: Complete flows quickly / meet deadlines

Short flows

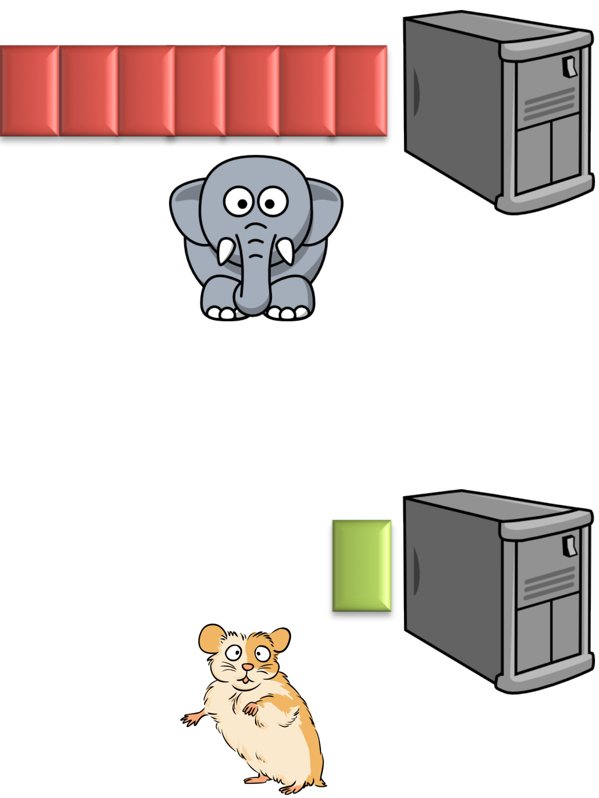
(e.g., query, coordination) → **Low Latency**

Large flows

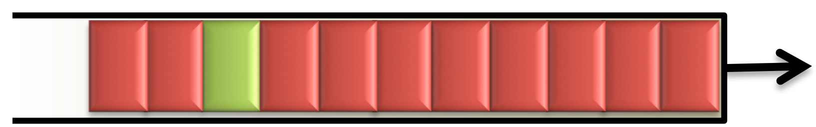
(e.g., data update, backup) → **High Throughput**

Low Latency Congestion Control (DCTCP, RCP, XCP, …)

Keep network queues small (at high throughput)



Implicitly prioritize mice

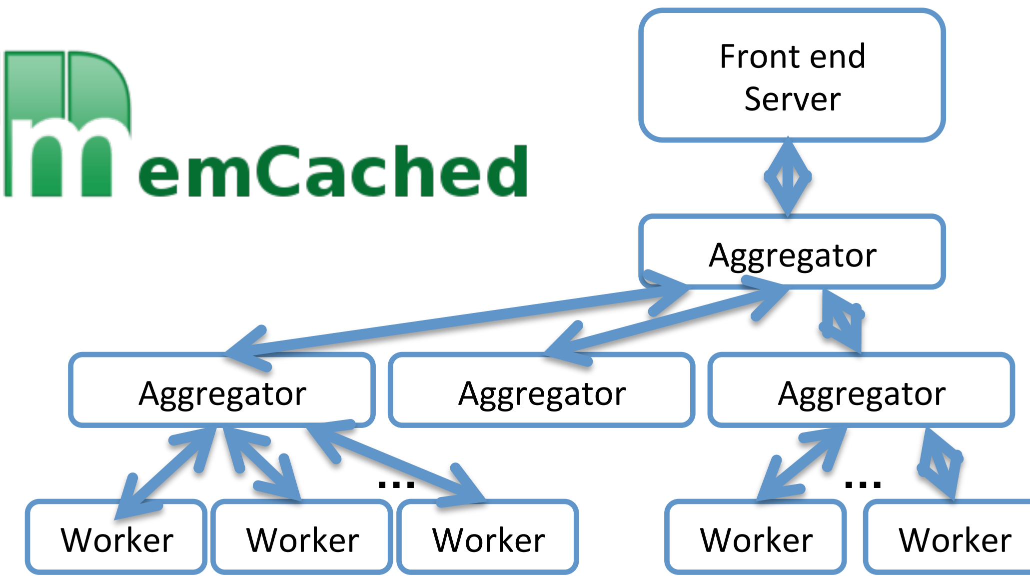


Can we do better?

The Opportunity

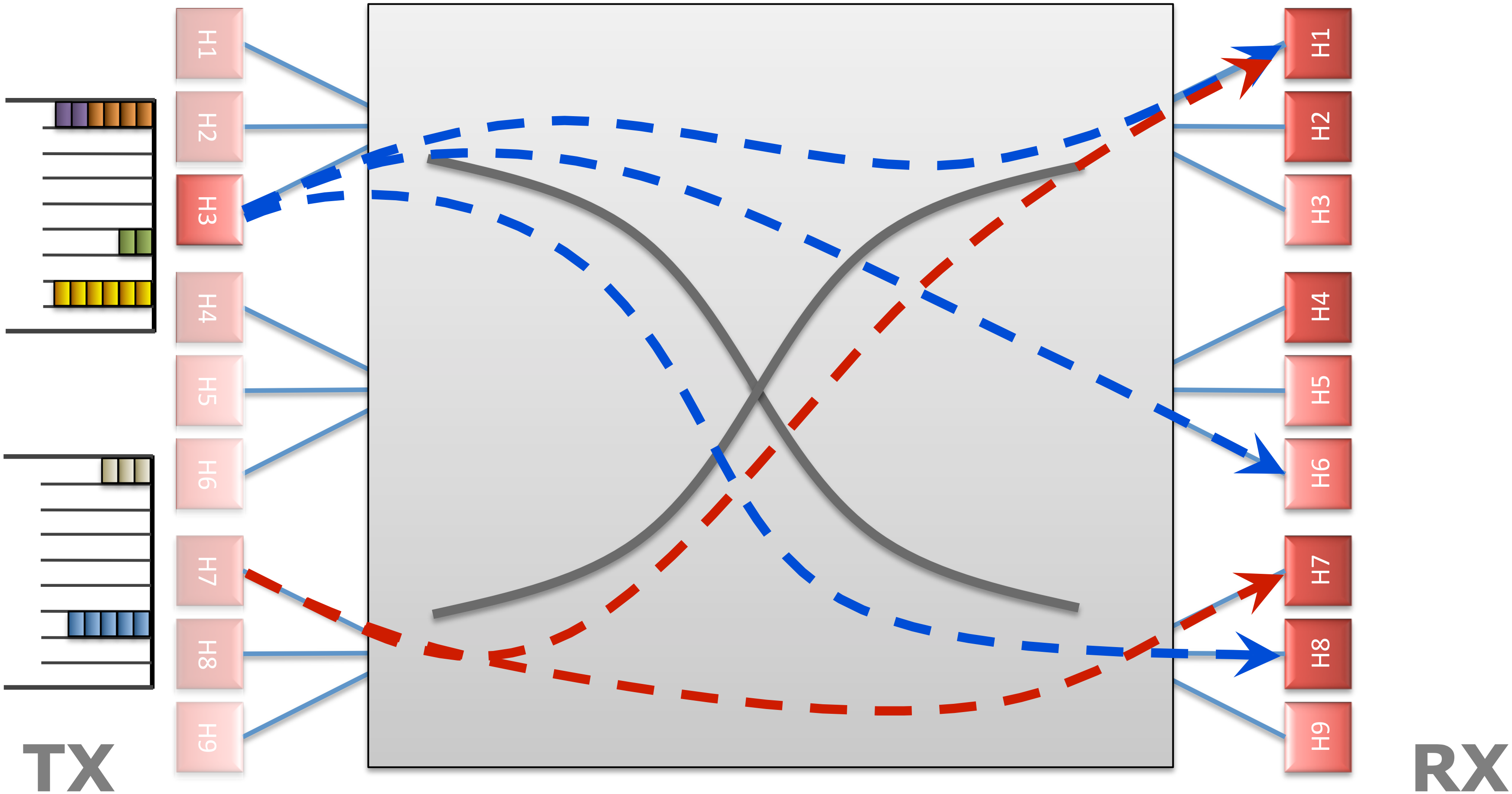
Many DC apps/platforms know flow size or deadlines in advance

- Key/value stores
- Data processing
- Web search



What You Said

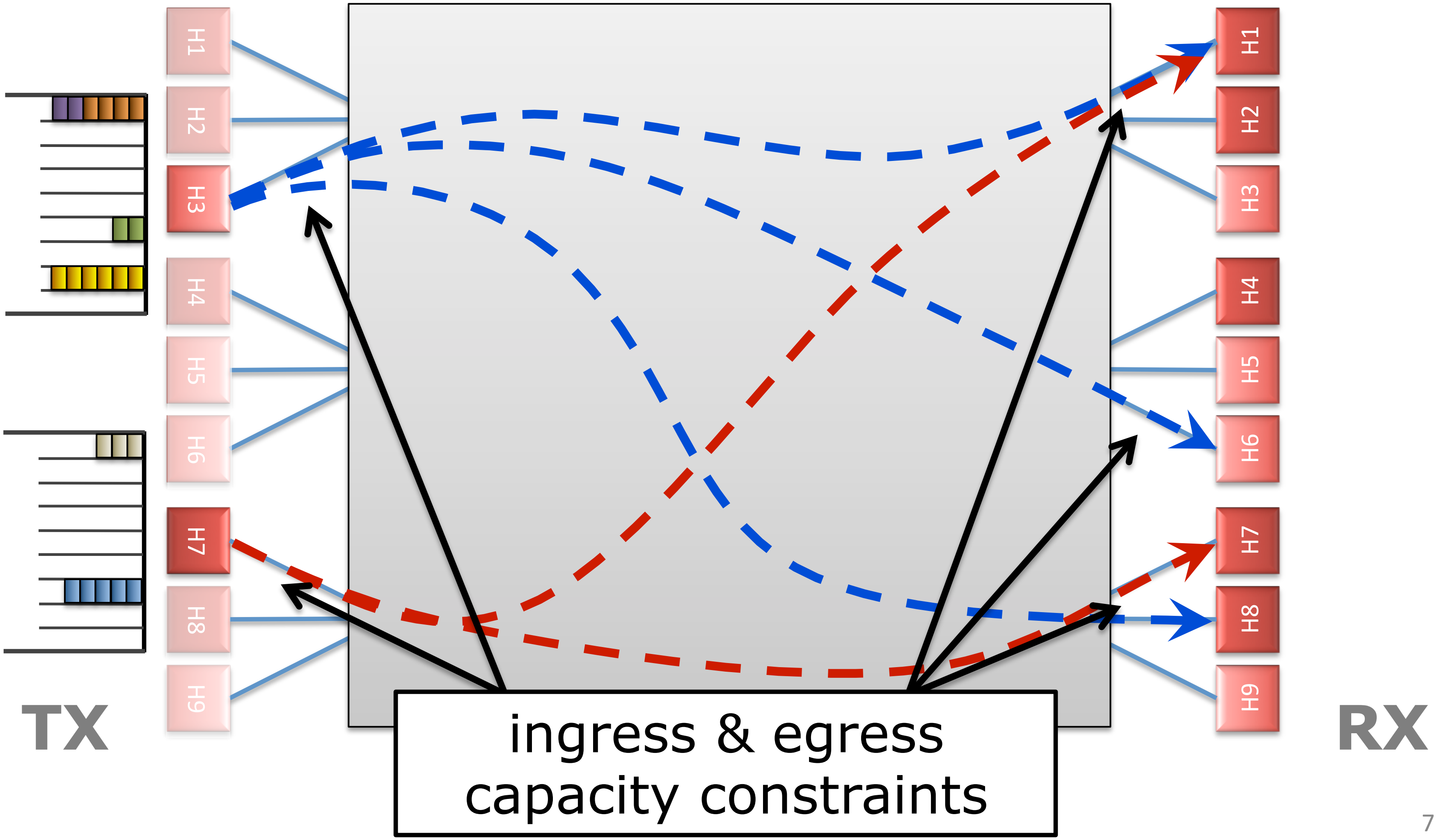
Amy: *“Many papers that propose new network protocols for datacenter networks (such as PDQ and pFabric) argue that these will improve "user experience for web services". However, none seem to evaluate the impact of their proposed scheme on user experience... I remain skeptical that small protocol changes really have drastic effects on end-to-end metrics such as page load times, which are typically measured in seconds rather than in microseconds.”*


H1
H2
H3
H4
H5
H6
H7
H8
H9
TX
RX

DC transport = Flow scheduling on giant switch

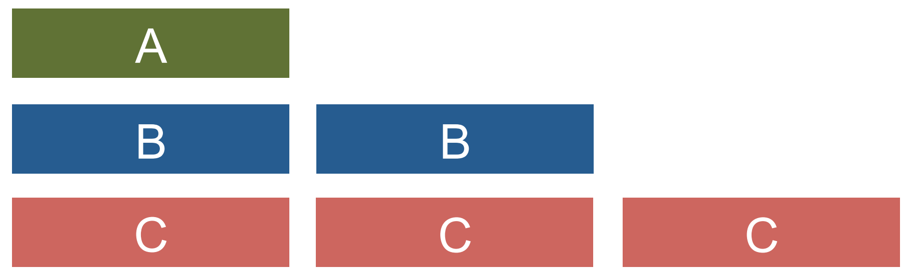
Objective?

- Minimize avg FCT
- Minimize missed deadlines



Example: Minimize Avg FCT

	Size
Flow A	1
Flow B	2
Flow C	3



arrive at the same time

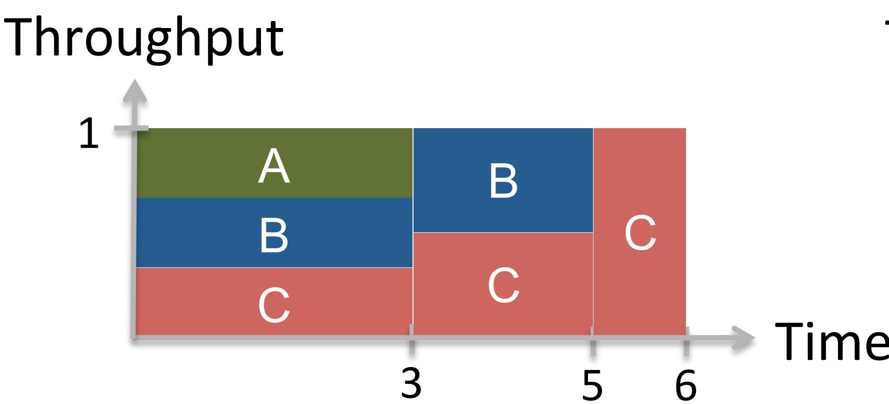
share the same bottleneck link

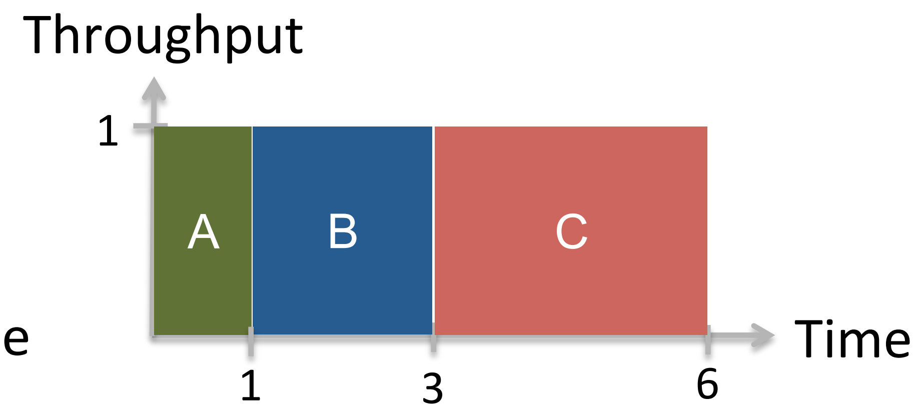
✧ Adapted from slide by Chi-Yao Hong (UIUC)

Example: Minimize Avg FCT

Fair sharing:
3, 5, 6
mean: 4.67

Shortest flow first:
1, 3, 6
mean: 3.33



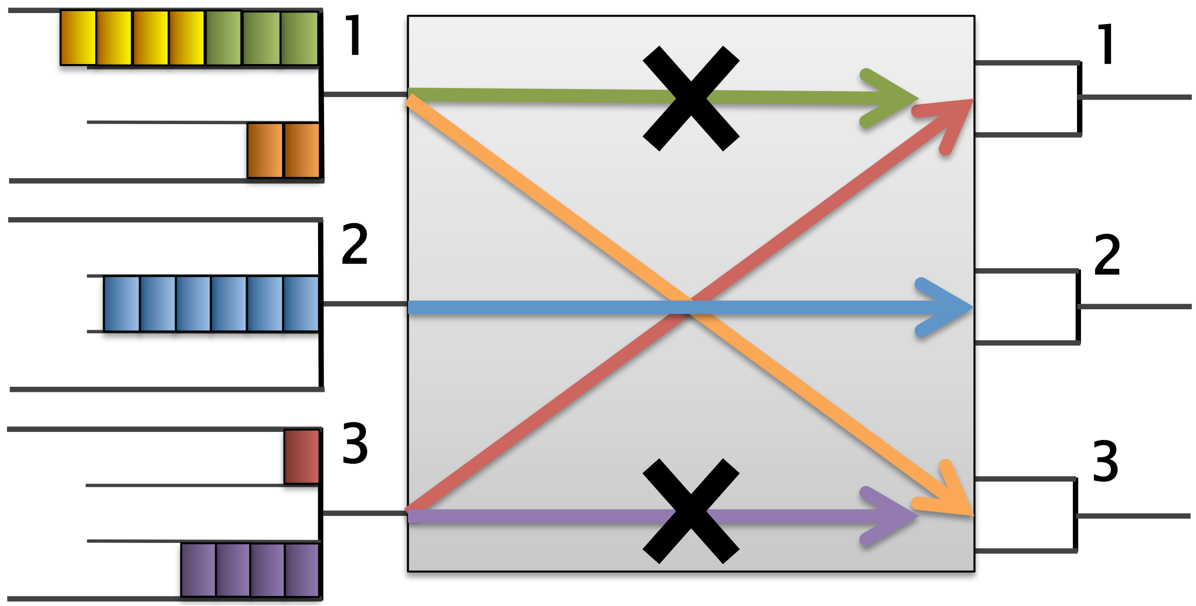


✧ Adapted from slide by Chi-Yao Hong (UIUC)

Optimal Flow Scheduling for Avg FCT

NP-hard for multi-link network [Bar-Noy et al.]

- Shortest Flow First: **2-approximation**



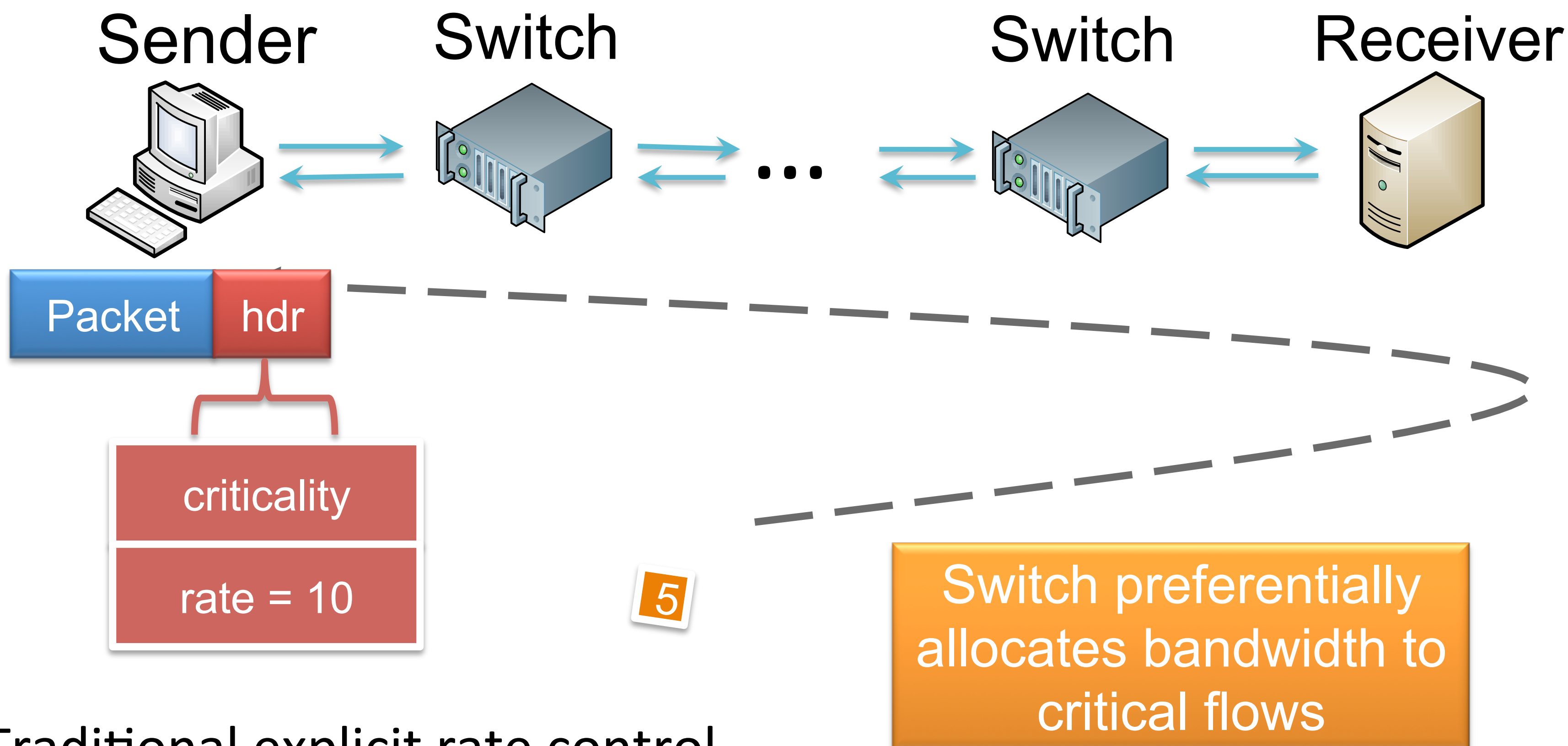
How can we schedule flows based on flow criticality in a distributed way?

Some transmission order

PDQ

✧Several slides based on presentation by Chi-Yao Hong (UIUC)

PDQ: Distributed Explicit Rate Control

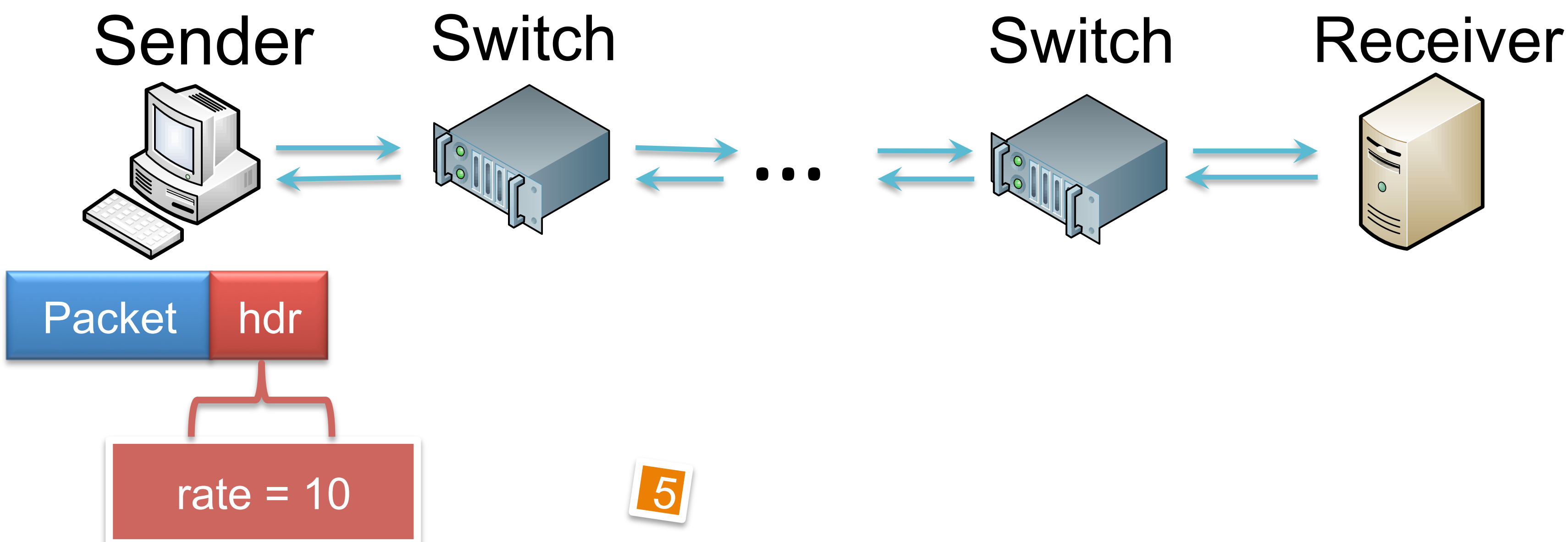


Traditional explicit rate control

Fair sharing (e.g., XCP, RCP)

Contrast with Traditional Explicit Rate Control

Traditional schemes (e.g. RCP, XCP) target fair sharing



- Each switch determines a "fair share" rate based on local congestion: R 🡐 R - k*congestion-measure
- Source use smallest rate advertised on their path

Challenges

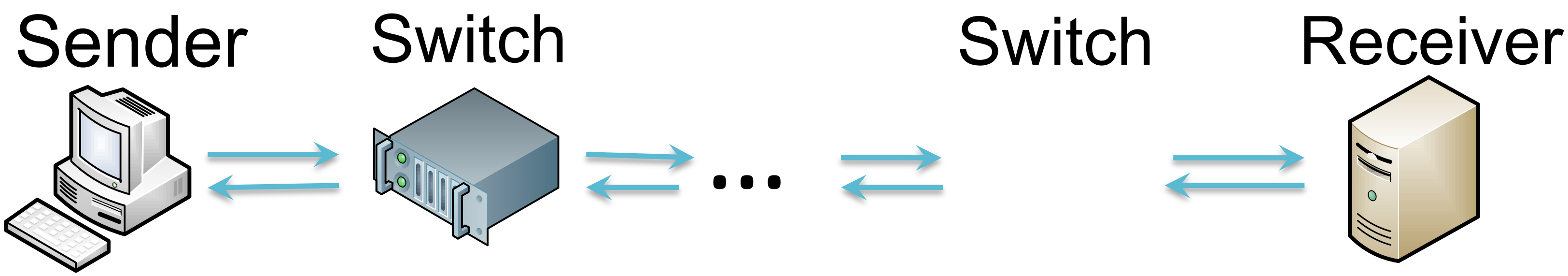
PDQ switches need to agree on rate decisions

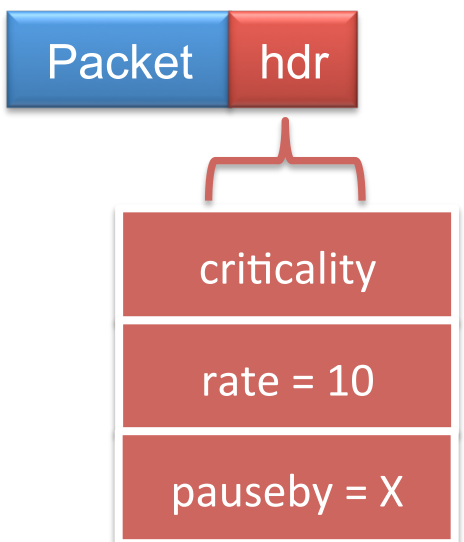
Low utilization during flow switching

Congestion and queue buildup

Paused flows need to know when to start

Challenge:
Switches need to agree on rate decisions





What can go wrong without consensus?

How do PDQ switches reach consensus?

Why is “pauseby” needed?

What You Said

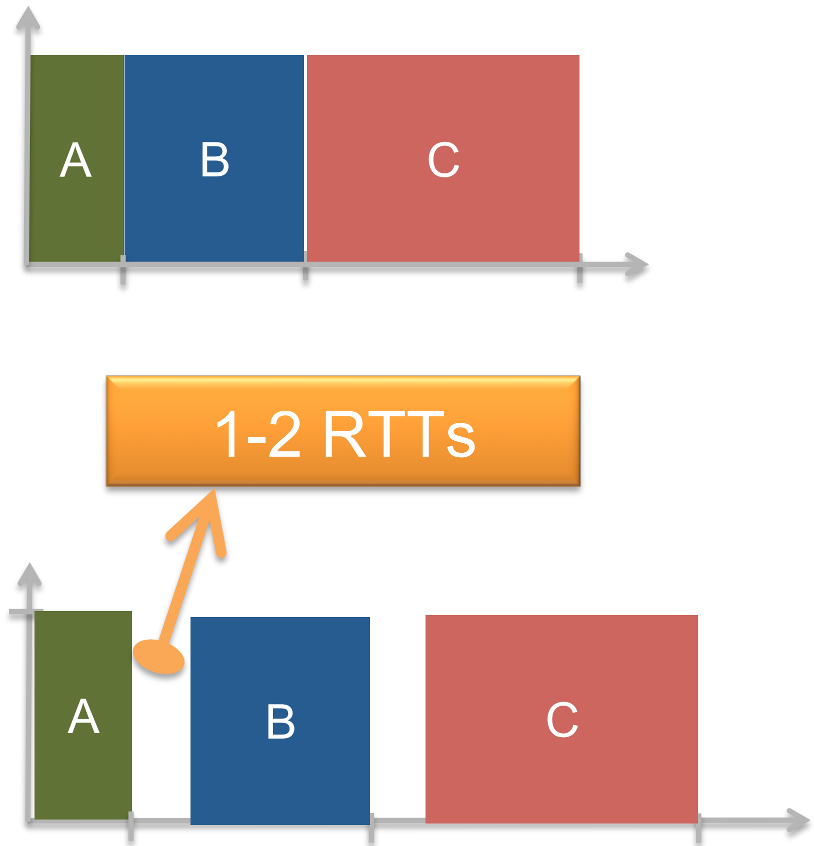
Austin: *"It is an interesting departure from AQM in that, with the concept of paused queues, PDQ seems to leverage senders as queue memory."*

Challenge:
Low utilization during flow switching

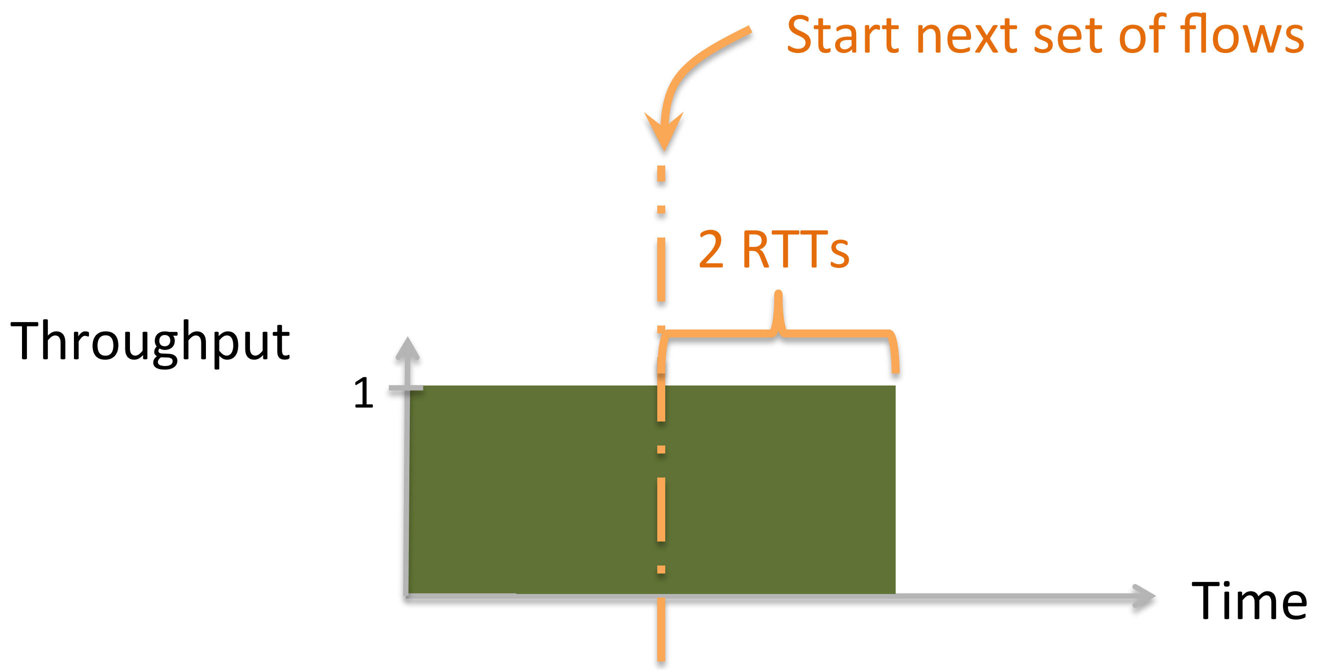
Goal:

How does PDQ avoid this?

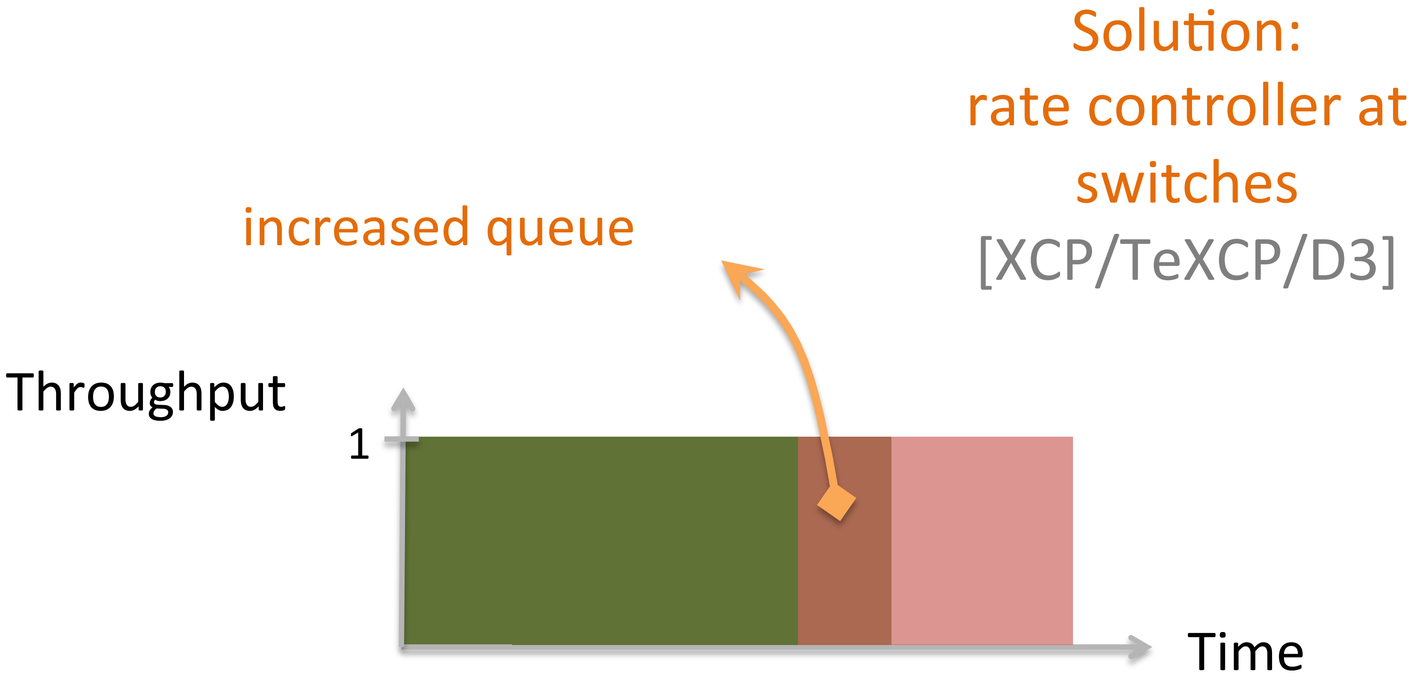
Practice:



Early Start:
Seamless flow switching



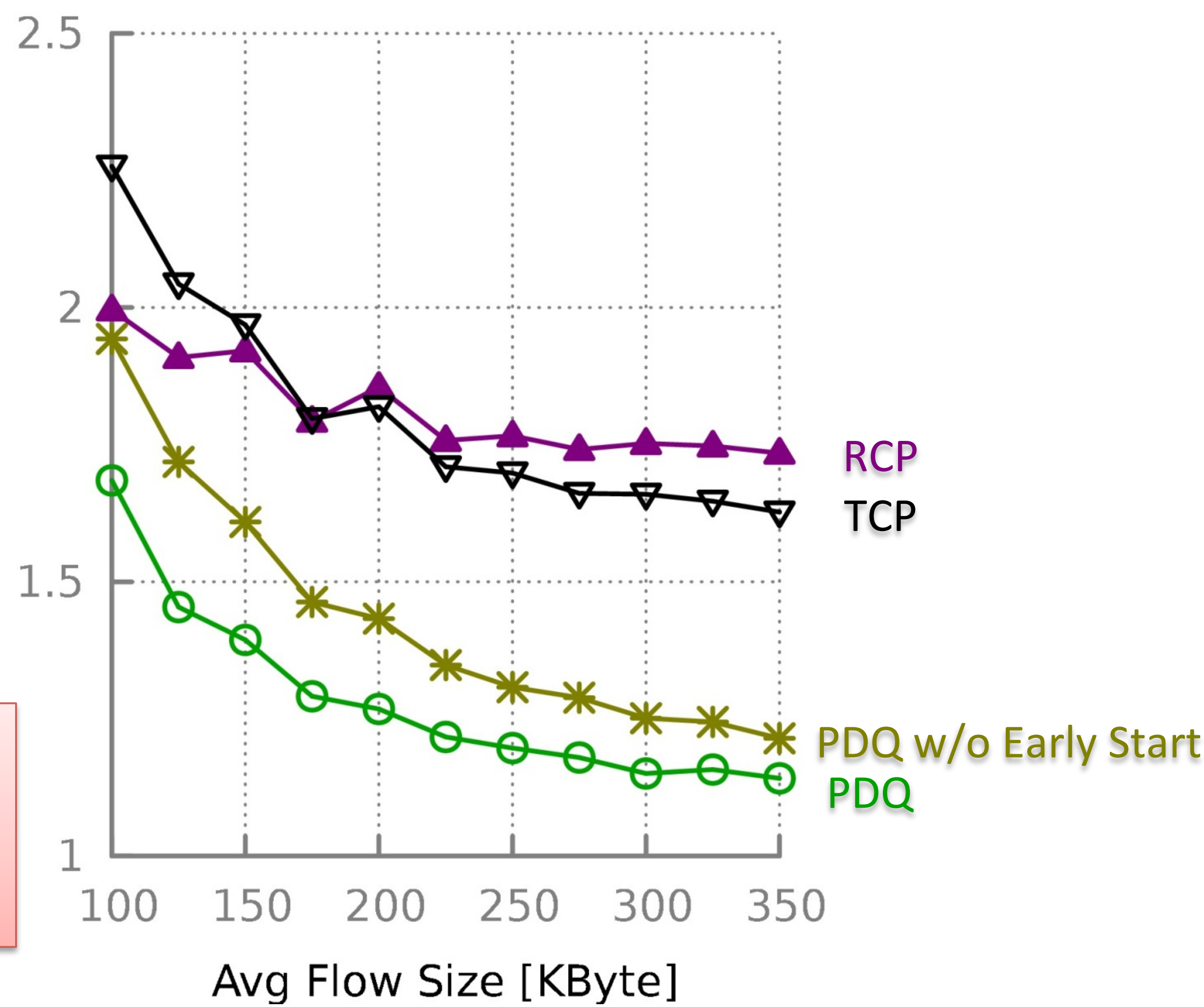
Early Start:
Seamless flow switching



Discussion

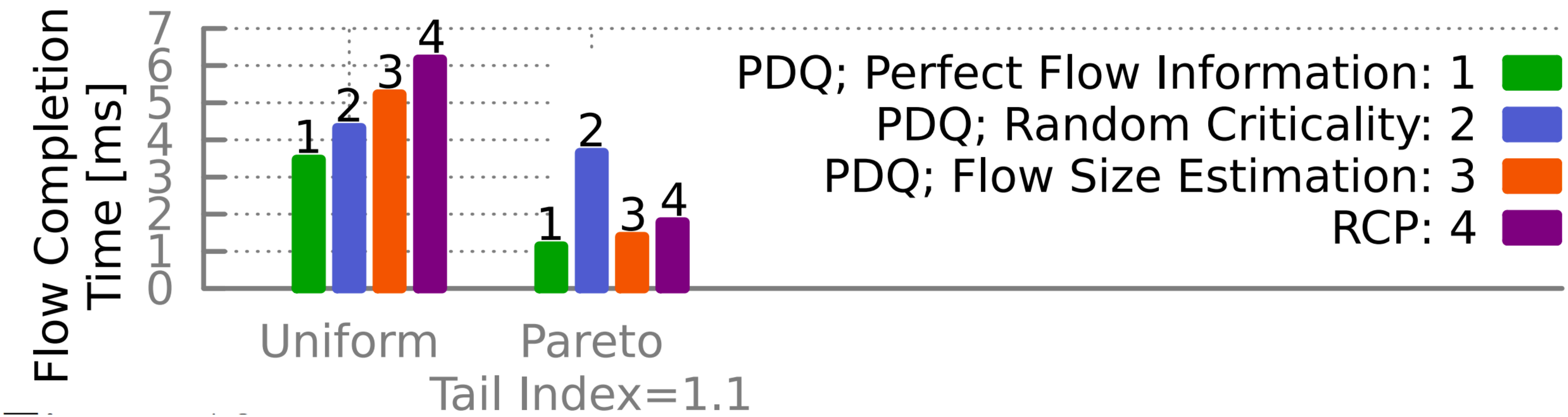
Mean FCT

Mean flow completion time [Normalized to *a lower bound*]



Omniscient scheduler controls with **zero** control feedback delay

What if flow size not known?



Why does flow size estimation (criticality = bytes sent) work better for Pareto?

Other questions

Fairness: can long flows starve?

> 99% of jobs complete faster under SJF than under fair sharing
>
> [Bansal, Harchol-Balter; SIGMETRICS'01]
> Assumption: heavy-tailed flow distribution

Resilience to error: what if packet gets lost or flow information is inaccurate?

Multipath: does PDQ benefit from multipath?

pFabric

pFabric in 1 Slide

Packets carry a single priority #

- e.g., prio = remaining flow size

pFabric Switches

- Send highest priority / drop lowest priority pkts
- Very small buffers (20-30KB for 10Gbps fabric)

pFabric Hosts

- Send/retransmit aggressively
- Minimal rate control: just prevent congestion collapse

Main Idea:
Decouple scheduling from rate control

pFabric Switch

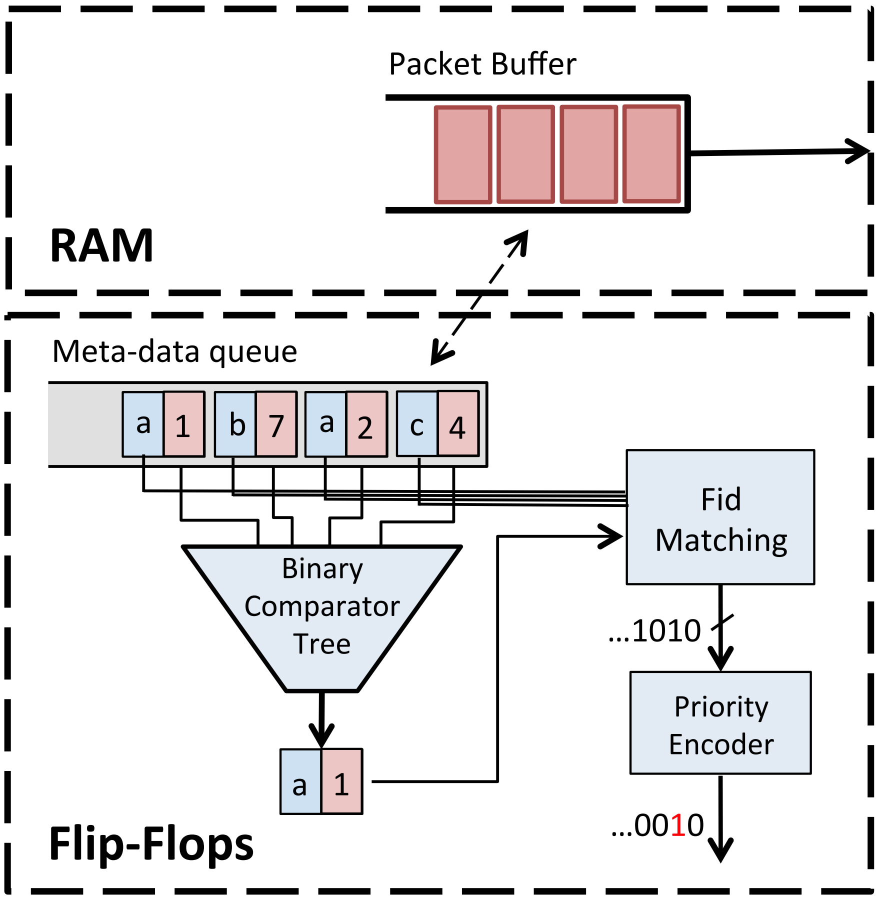
Boils down to a sort

- Essentially unlimited priorities
- Thought to be difficult in hardware

Existing switching only support 4-16 priorities

pFabric queues very small

- 51.2ns to find min/max of ~600 numbers
- Binary comparator tree: 10 clock cycles
- Current ASICs: clock ~ 1ns



pFabric Rate Control

Minimal version of TCP algorithm

1. Start at line-rate
   - Initial window larger than BDP
2. No retransmission timeout estimation
   - Fixed RTO at small multiple of round-trip time
3. Reduce window size upon packet drops
   - Window increase same as TCP (slow start, congestion avoidance, …)
4. After multiple consecutive timeouts, enter "probe mode"
   - Probe mode sends min. size packets until first ACK

What about queue buildup?

Why window control?

Why does pFabric work?

Key invariant:

At any instant, have the highest priority packet (according to ideal algorithm) available at the switch.

Priority scheduling

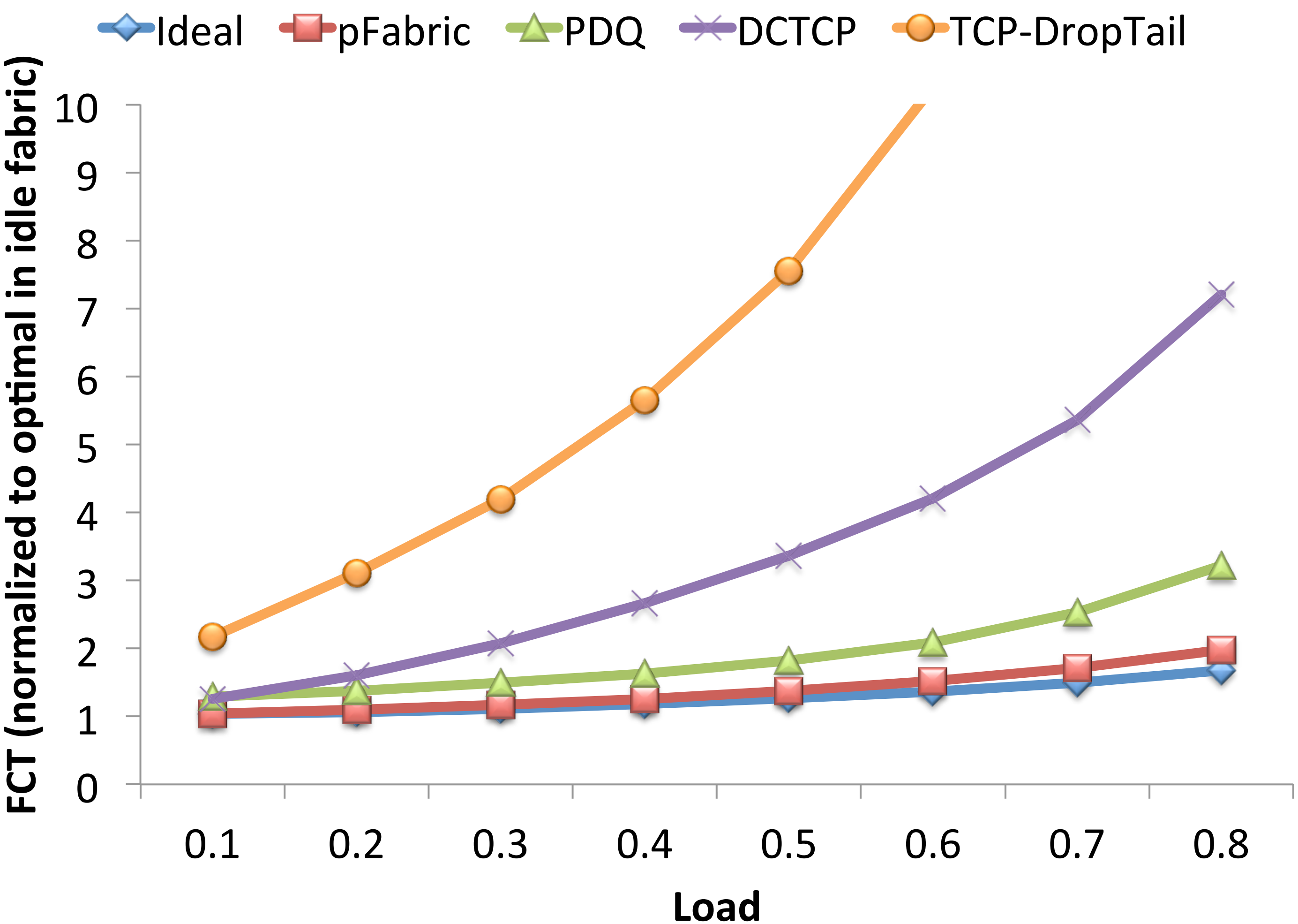
- High priority packets traverse fabric as quickly as possible

What about dropped packets?

- Lowest priority → not needed till all other packets depart
- Buffer > BDP → enough time (> RTT) to retransmit

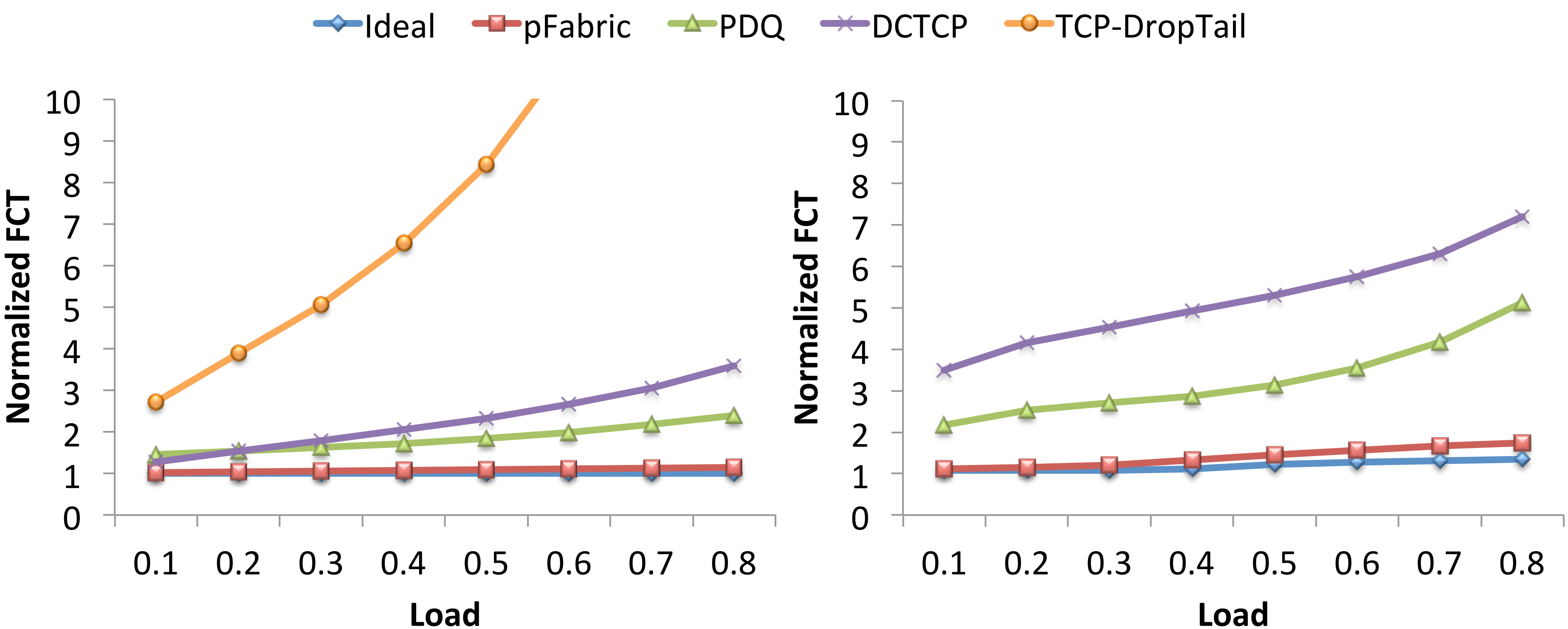
Discussion

Overall Mean FCT

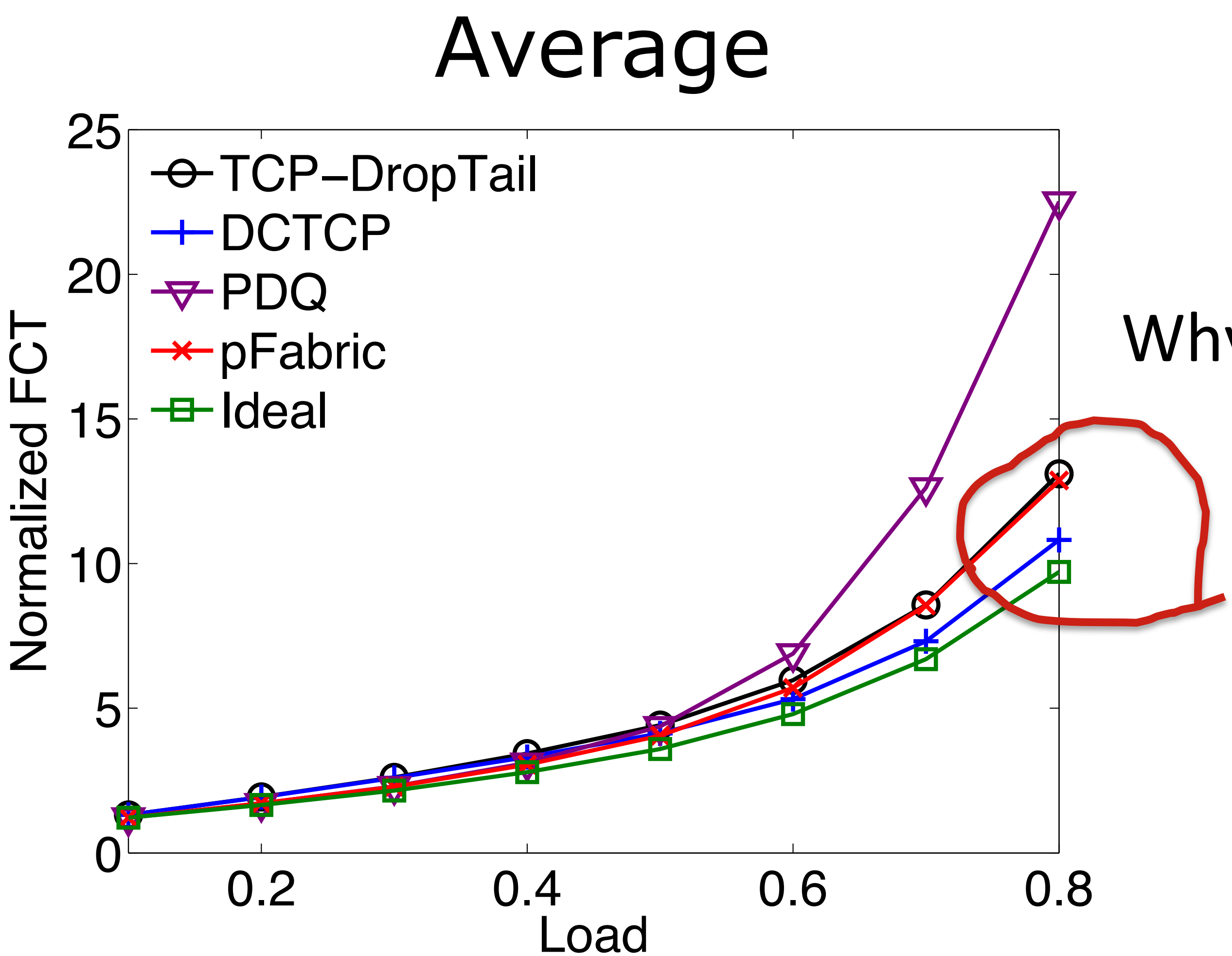


Mice FCT (<100KB)

Average — 99th Percentile



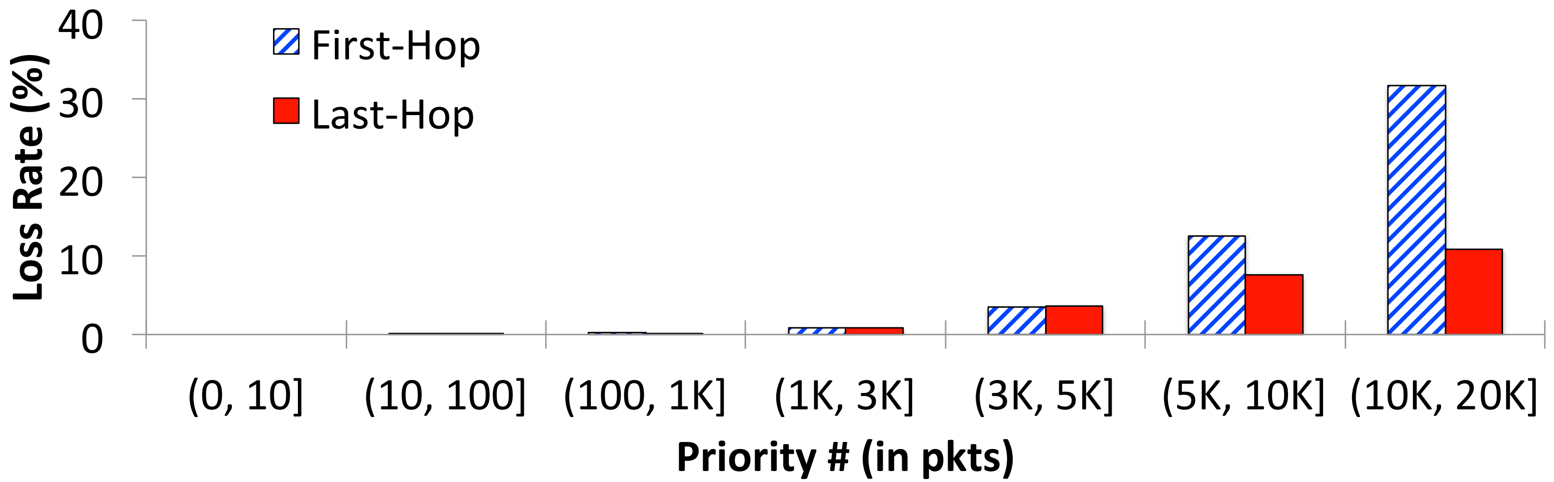
Elephant FCT (>10MB)



Why the gap?

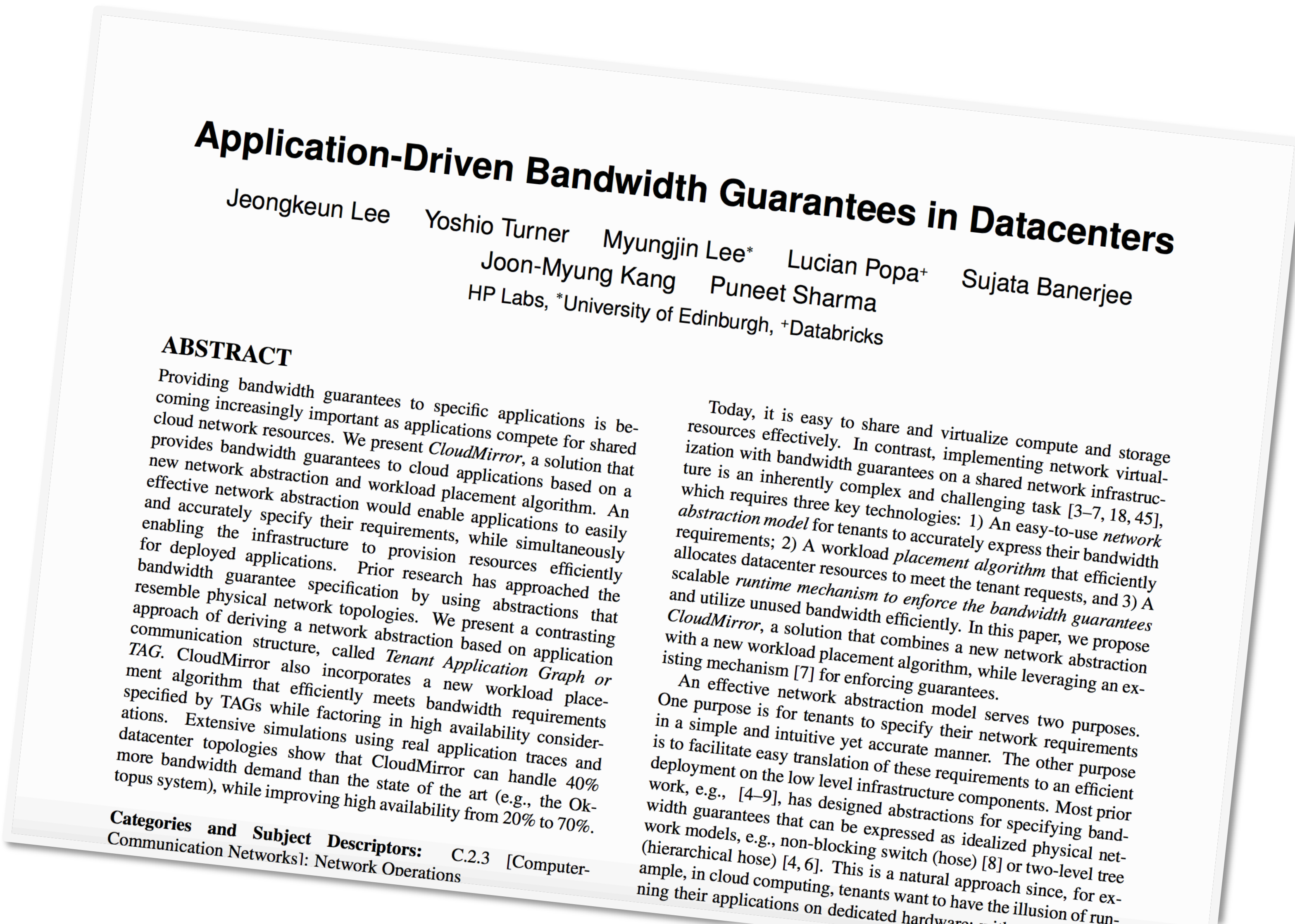
Loss Rate vs Packet Priority
(at 80% load)

* Loss rate at other hops is negligible



Almost all packet loss is for large (latency-insensitive) flows

Next Time:
Multi-Tenant Performance Isolation



Application-Driven Bandwidth Guarantees in Datacenters

Jeongkeun Lee Yoshio Turner Myungjin Lee* Lucian Popa+ Sujata Banerjee
Joon-Myung Kang Puneet Sharma
HP Labs, *University of Edinburgh, +Databricks

ABSTRACT

Providing bandwidth guarantees to specific applications is becoming increasingly important as applications compete for shared cloud network resources. We present *CloudMirror*, a solution that provides bandwidth guarantees to cloud applications based on a new network abstraction and workload placement algorithm. An effective network abstraction would enable applications to easily and accurately specify their requirements, while simultaneously enabling the infrastructure to provision resources efficiently for deployed applications. Prior research has approached the bandwidth guarantee specification by using abstractions that resemble physical network topologies. We present a contrasting approach of deriving a network abstraction based on application communication structure, called *Tenant Application Graph or TAG*. CloudMirror also incorporates a new workload placement algorithm that efficiently meets bandwidth requirements specified by TAGs while factoring in high availability considerations. Extensive simulations using real application traces and datacenter topologies show that CloudMirror can handle 40% more bandwidth demand than the state of the art (e.g., the Oktopus system), while improving high availability from 20% to 70%.

Categories and Subject Descriptors: C.2.3 [Computer-Communication Networks]: Network Operations

Today, it is easy to share and virtualize compute and storage resources effectively. In contrast, implementing network virtualization with bandwidth guarantees on a shared network infrastructure is an inherently complex and challenging task [3–7, 18, 45], which requires three key technologies: 1) An easy-to-use *network abstraction model* for tenants to accurately express their bandwidth requirements; 2) A workload *placement algorithm* that efficiently allocates datacenter resources to meet the tenant requests, and 3) A scalable *runtime mechanism to enforce the bandwidth guarantees* and utilize unused bandwidth efficiently. In this paper, we propose *CloudMirror*, a solution that combines a new network abstraction with a new workload placement algorithm, while leveraging an existing mechanism [7] for enforcing guarantees.

An effective network abstraction model serves two purposes. One purpose is for tenants to specify their network requirements in a simple and intuitive yet accurate manner. The other purpose is to facilitate easy translation of these requirements into an efficient deployment on the low level infrastructure components. Most prior work, e.g., [4–9], has designed abstractions for specifying bandwidth guarantees that can be expressed as idealized physical network models, e.g., non-blocking switch (hose) [8] or two-level tree (hierarchical hose) [4, 6]. This is a natural approach since, for example, in cloud computing, tenants want to have the illusion of running their applications on dedicated hardware