Finishing Flows Quickly with Preemptive Scheduling

Chi-Yao Hong Matthew Caesar Brighten Godfrey

University of Illinois at Urbana-Champaign

Goal

Low latency datacenter networks

Why do we even care?

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

[Speed matters; Greg Linden]

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]

Goal

Low latency datacenter networks

amazon.com

YAHOO!

Google

AOL

Firefox

Walmart

users really
respond to latency!

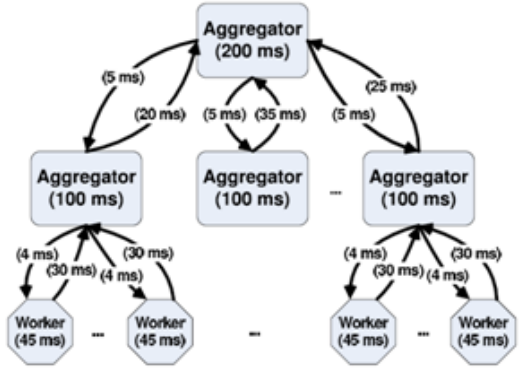
Low latency datacenter networks

Finish flows quickly

Users always want to get results earlier

Meet flow deadline

User-facing web applications have latency target



Partition-aggregate structure:
web search, recommendation systems, MapReduce/Dryad, social networks

D3: [Wilson, Ballani, Karagiannis, Rowstron; SIGCOMM'11]

Today's options for datacenter transport protocols

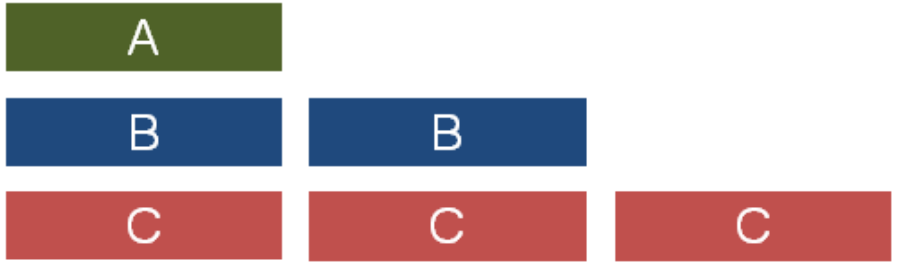
TCP ICTCP DCTCP XCP

Feature	Help reduce flow completion time?
High throughput & utilization	YES
Small queues & drops	YES
Fairness	NO!

In fact, fairness damages
flow completion time

Example

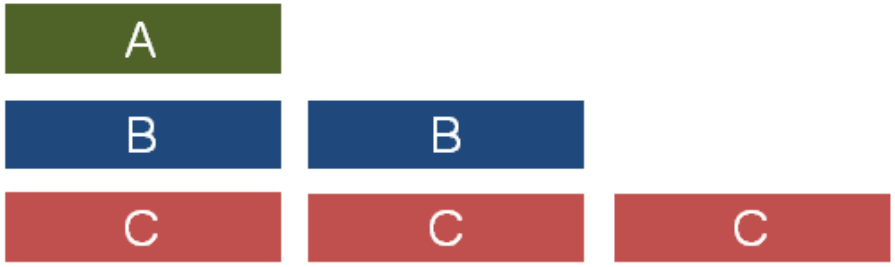
	Size
Flow A	1
Flow B	2
Flow C	3

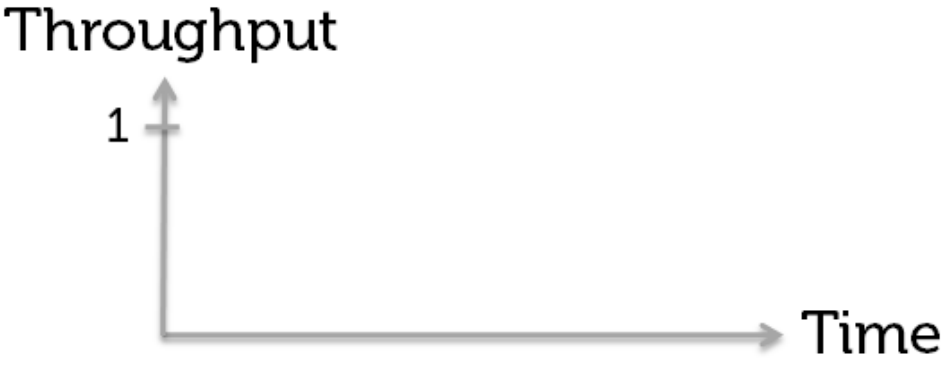


arrive at the same time

share the same bottleneck link

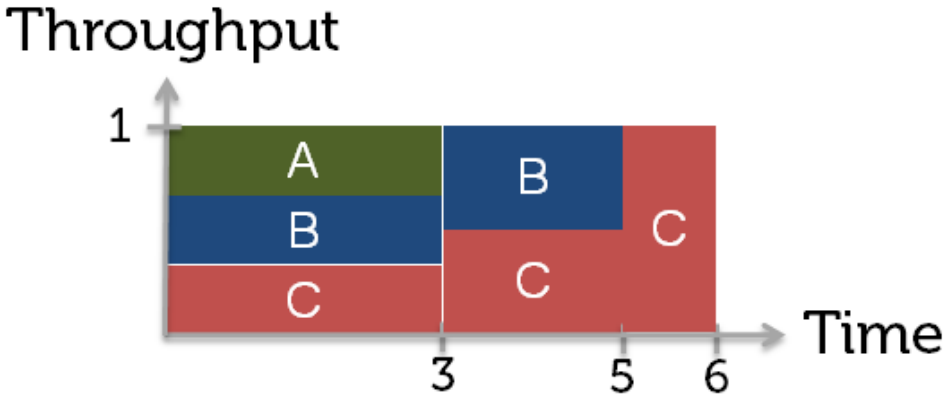
Example: Fair sharing





Example: Fair sharing

Fair sharing:
3, 5, 6
mean: 4.67

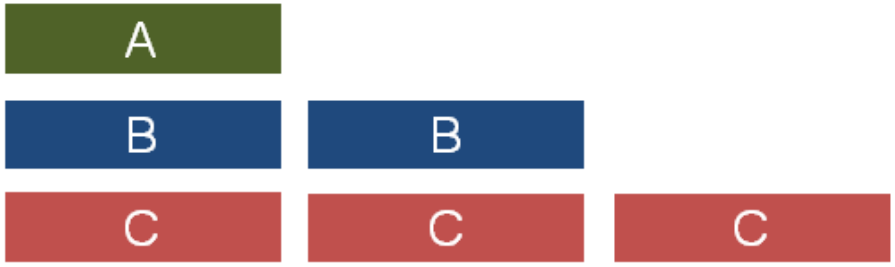


far from optimal!

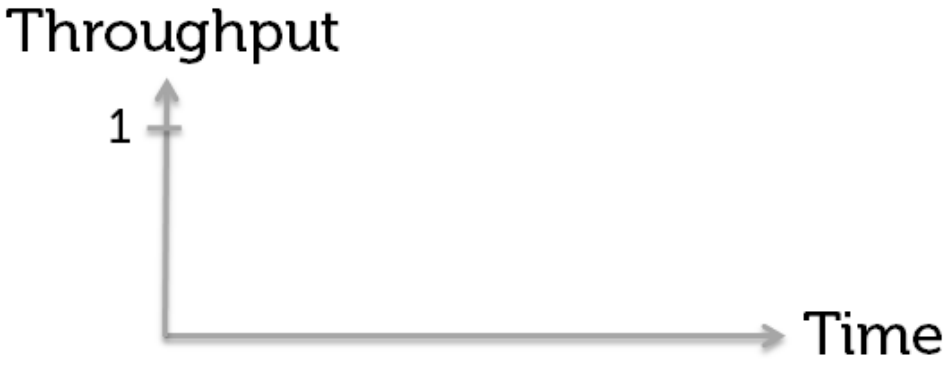
$$\text{mean flow completion time} = \frac{3+5+6}{3} = 4.67$$

Example: Relaxing fairness constraint

(A → B → C)



Fair sharing:
3, 5, 6
mean: 4.67

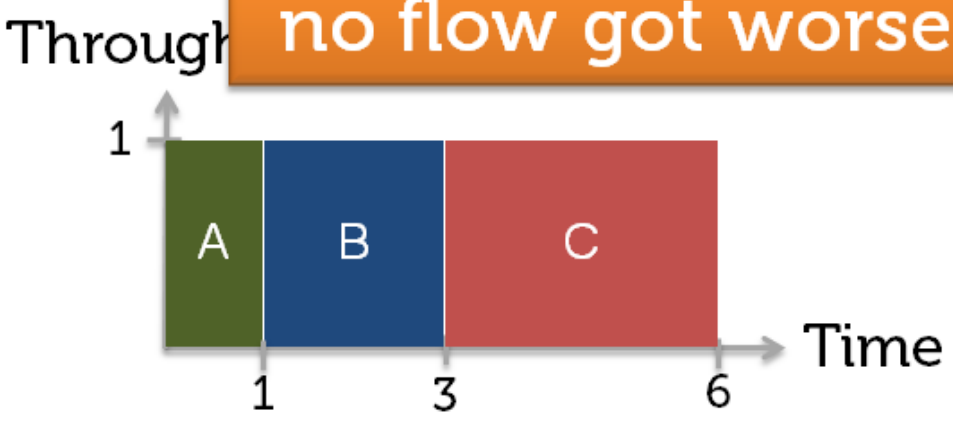


Example: Relaxing fairness constraint

(A → B → C)

29% saving in mean

It's still fair:
no flow got worse!



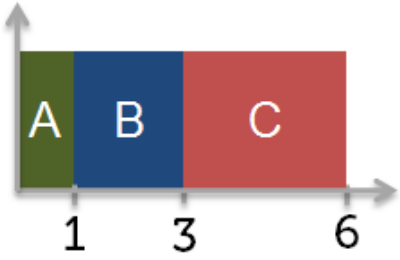
mean flow completion time = $\frac{1+3+6}{3}$ = 3.33

Fair sharing:
3, 5, 6
mean: 4.67

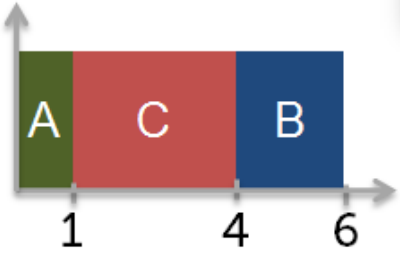
(A → B → C):
1, 3, 6
mean: 3.33

Order matters

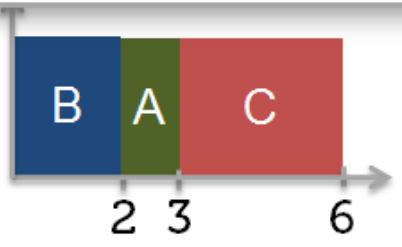
Fair sharing:
3, 5, 6
mean: 4.67



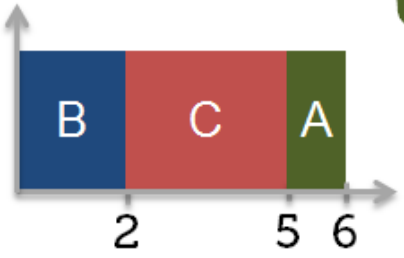
PDQ(A → B → C): 1, 3, 6
mean: 3.33 ✔



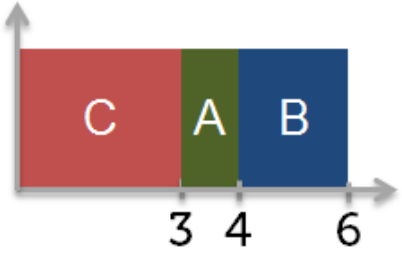
PDQ(A → C → B): 1, 6, 4
mean: 3.67



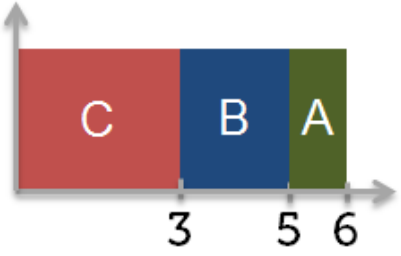
PDQ(B → A → C): 3, 2, 6
mean: 3.67



PDQ(B → C → A): 6, 2, 5
mean: 4.33



PDQ(C → A → B): 4, 6, 3
mean: 4.33



PDQ(C → A → B): 6, 5, 3
mean: 4.67 ✘

Relaxing fairness helps

Order matters!

Our solution

Forget about fairness –
let's relax fairness constraints!

Preemptive Distributed Quick flow scheduling

Pretty Damn Quick!

Chi-Yao Hong, Matthew Caesar, Brighten Godfrey University of Illinois at Urbana-Champaign

PDQ: Idea

Scheduling flows based on flow criticality

//

relative priority of flows;
transmission order

PDQ: Two primitives

Preemptive scheduling

Less-critical flows yield to critical flows

PDQ: Two primitives

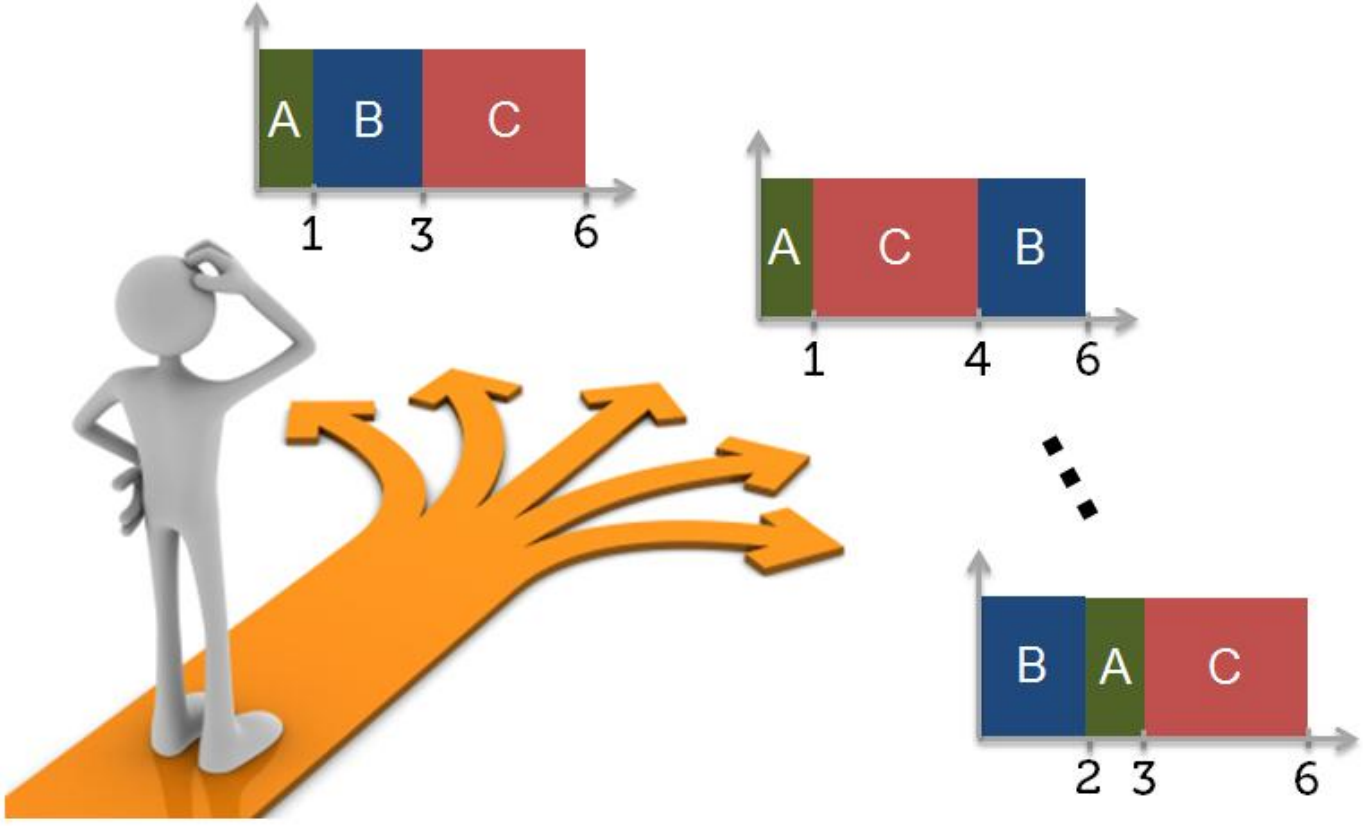
Preemptive scheduling

Less-critical flows yield to critical flows

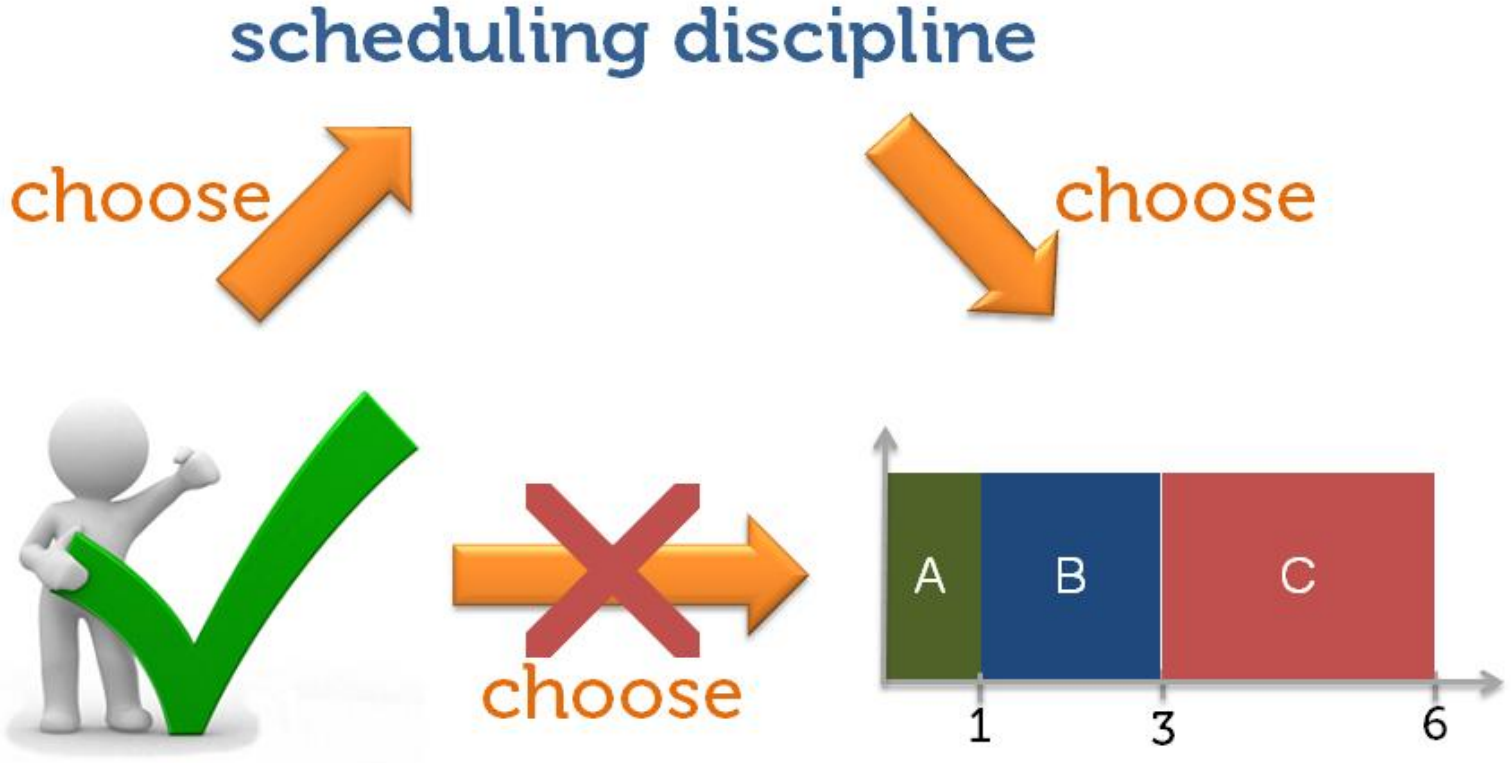
Dynamic scheduling

Flow criticality may change over time

How to choose flow criticality?



How to choose flow criticality?



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 priority

Any criticality-based scheduling

OK, theoretically they are optimal
but in practice…

Roadmap

Challenges

- Computing schedule across large network
- Ensuring seamless flow switching

Moving to actual datacenter settings

- Inaccurate flow information
- Fairness
- Scalability
- Failure Resilience
- Multipath
- …

Challenge: Scheduling requires centralized computations

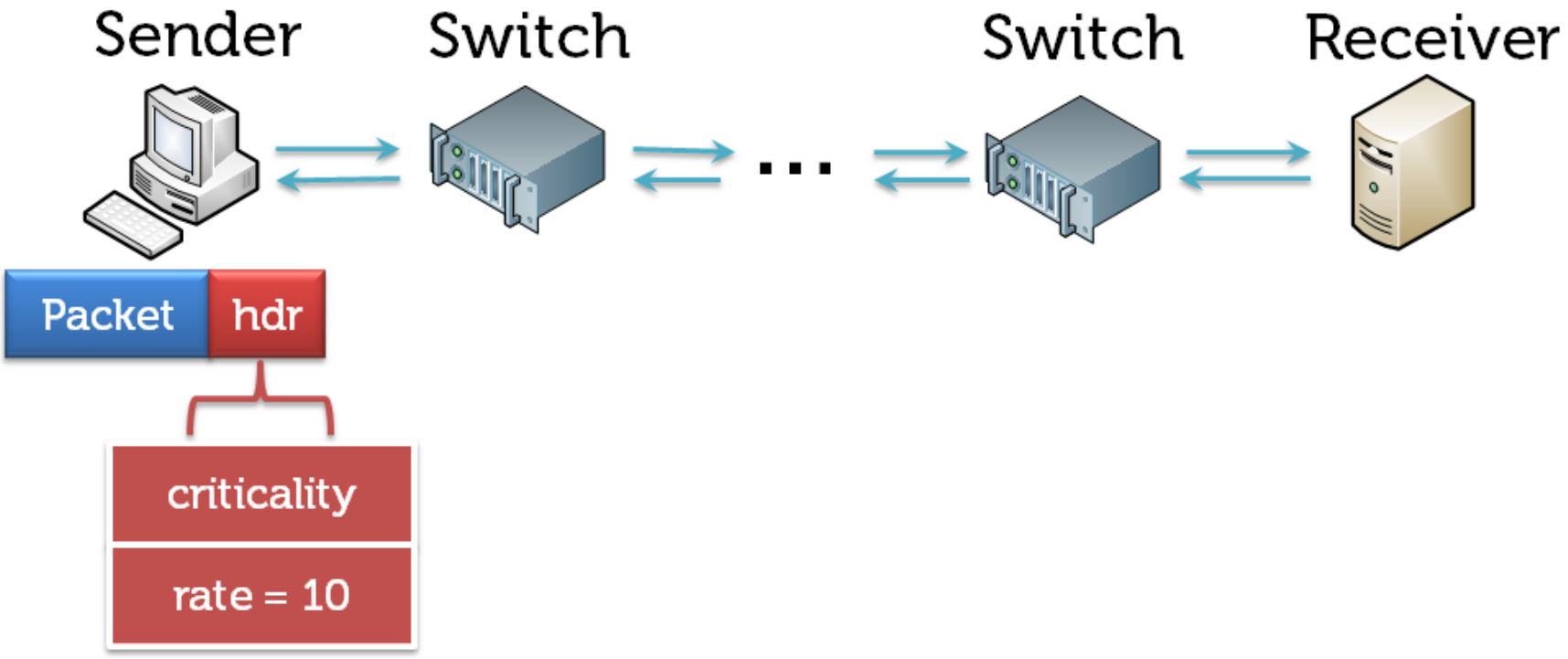
Centralized coordinator fails to scale

- Single point of failure
- Congestive hot-spot
- High flow initialization overhead *esp. for short flows*

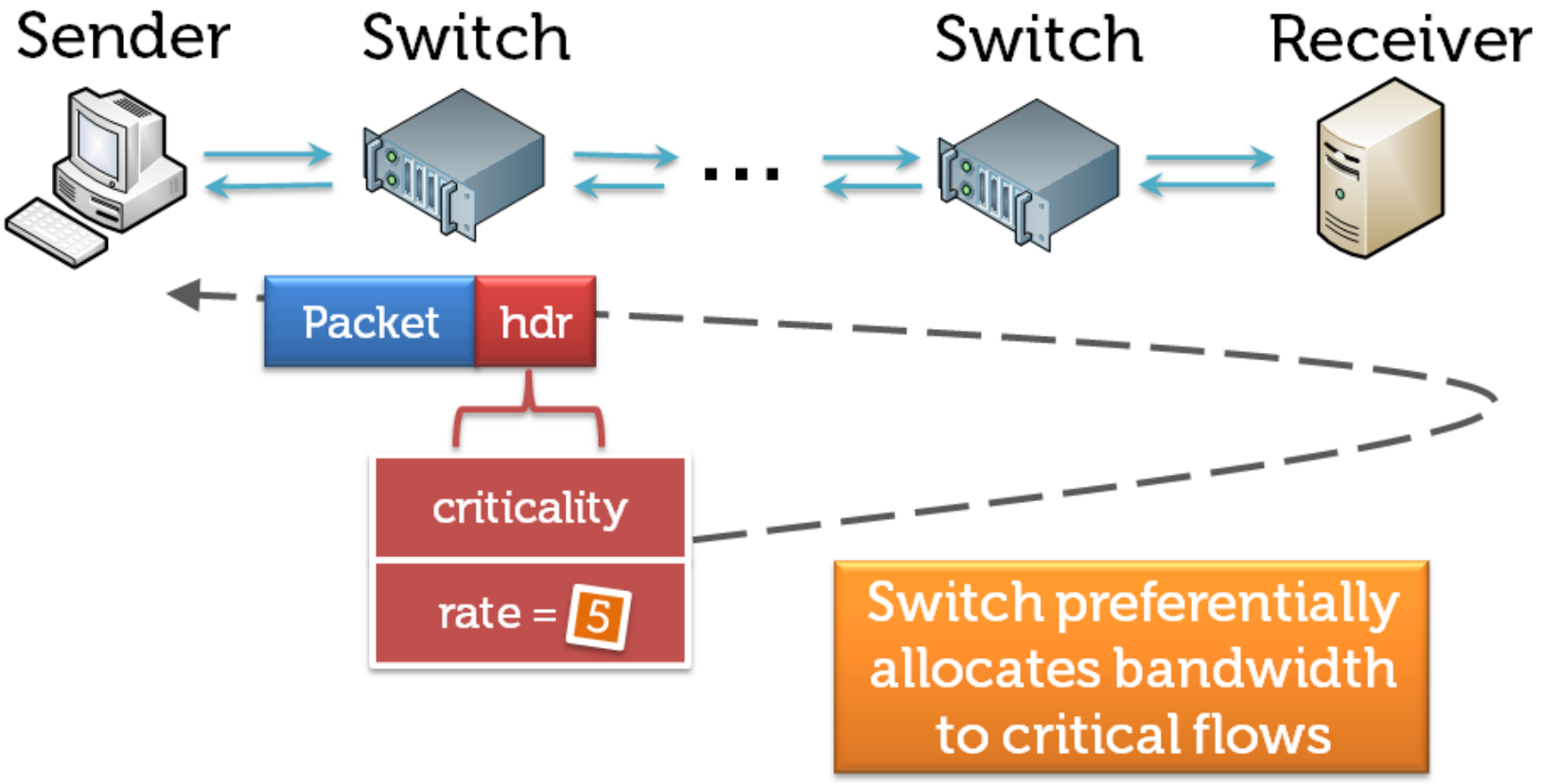
Fully distributed solution

- PDQ switches collaboratively control flow schedule by tagging packet headers
- No deadlock
- Bounded convergence time
- Use only FIFO tail-drop queue
- Some computation required at the switch per flow

PDQ: Fully distributed design

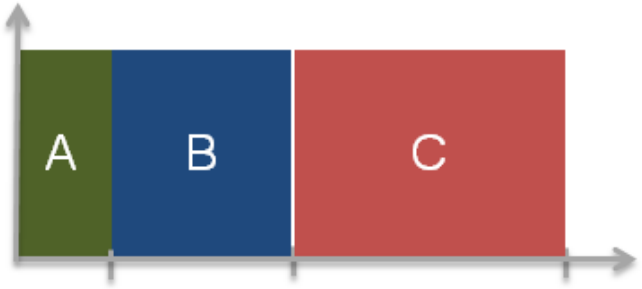


PDQ: Fully distributed design

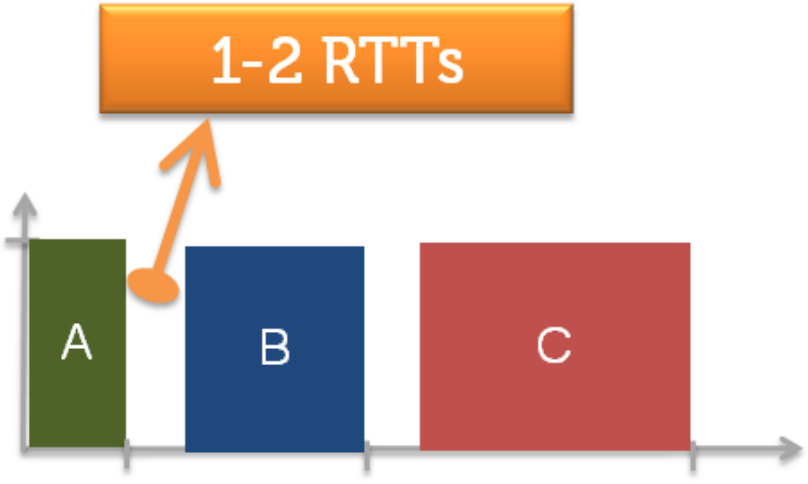


Challenge:
Low utilization during flow switching

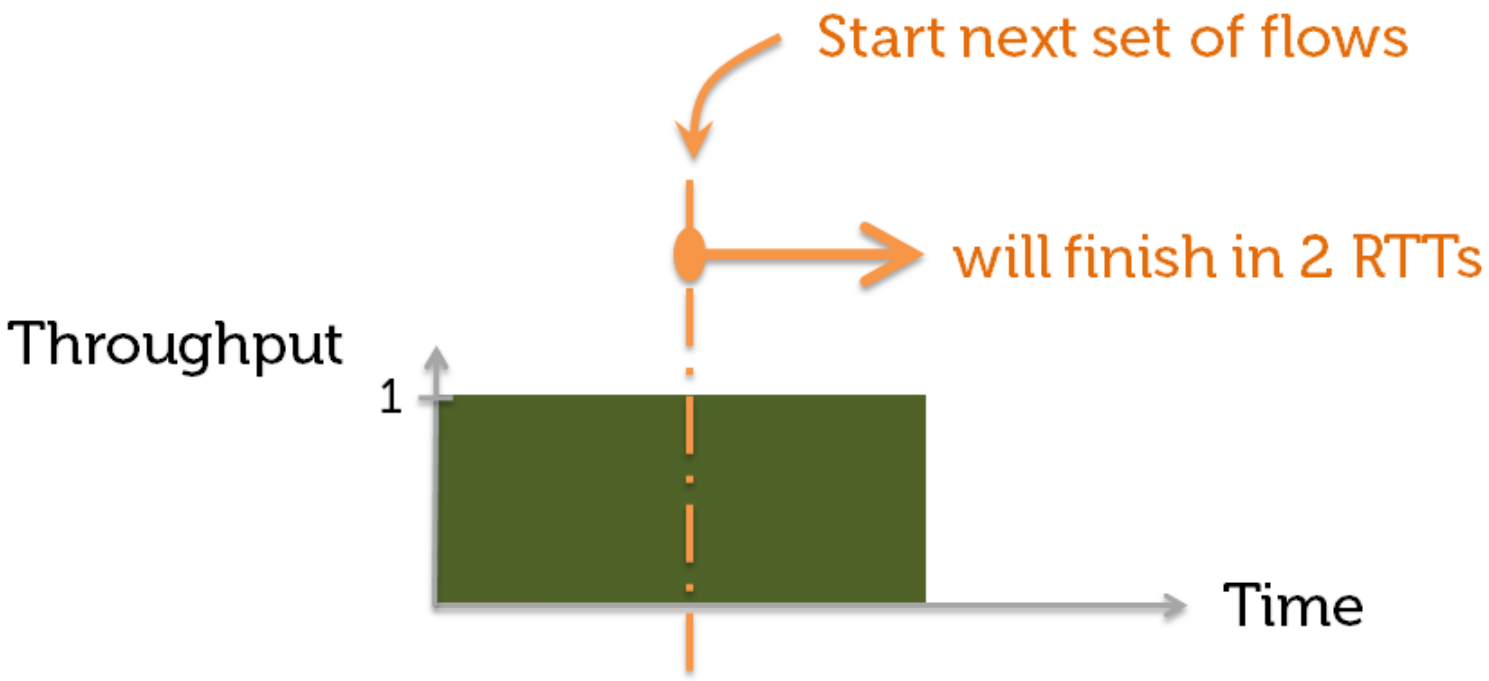
Goal:



Practice:

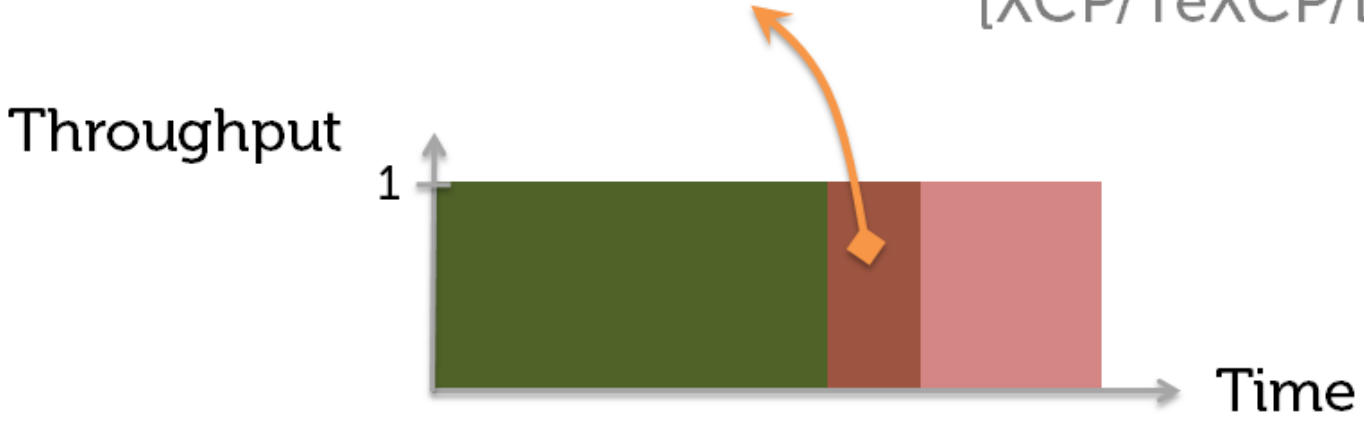


Early Start: Seamless flow switching

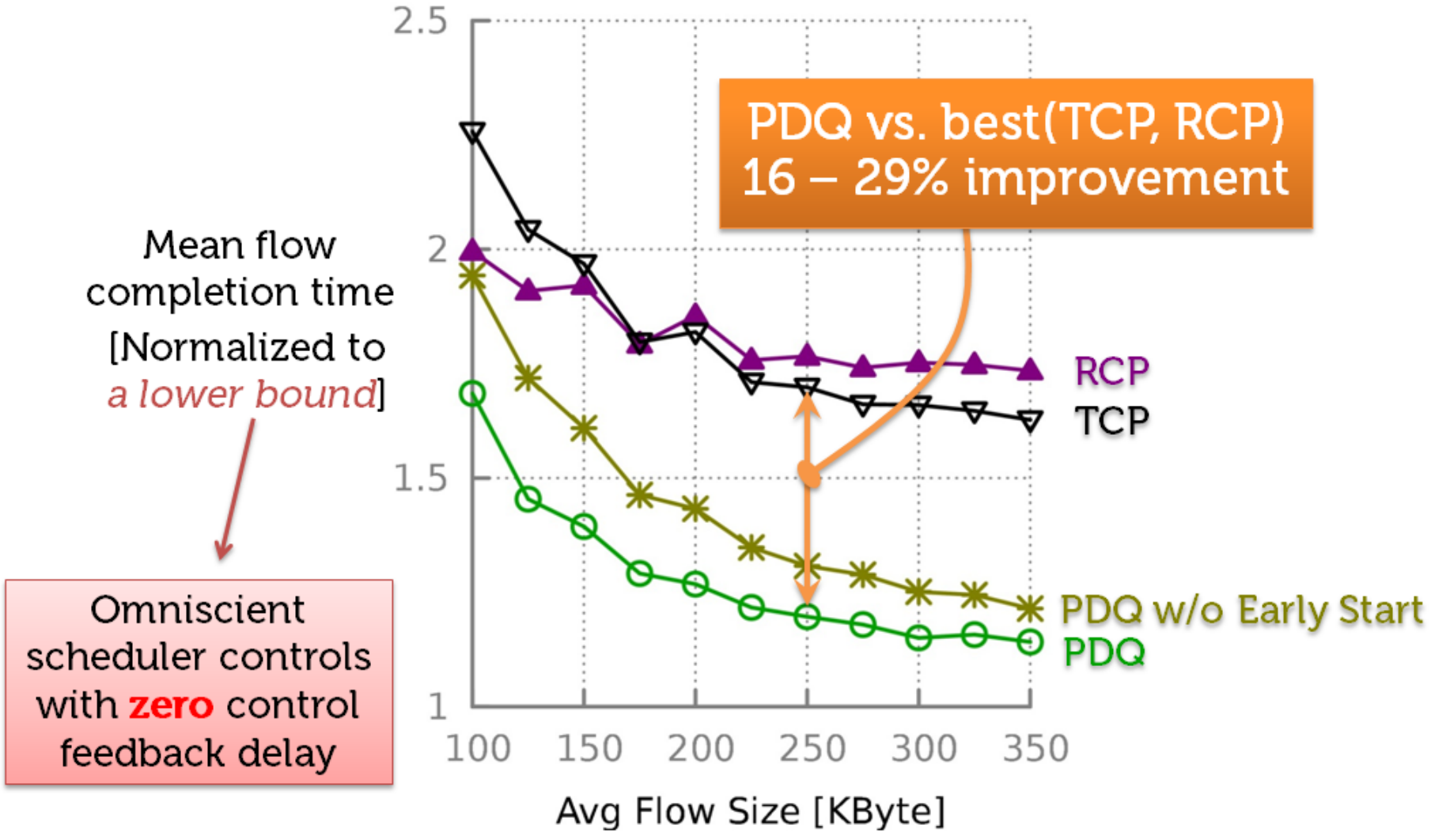


Early Start: Seamless flow switching

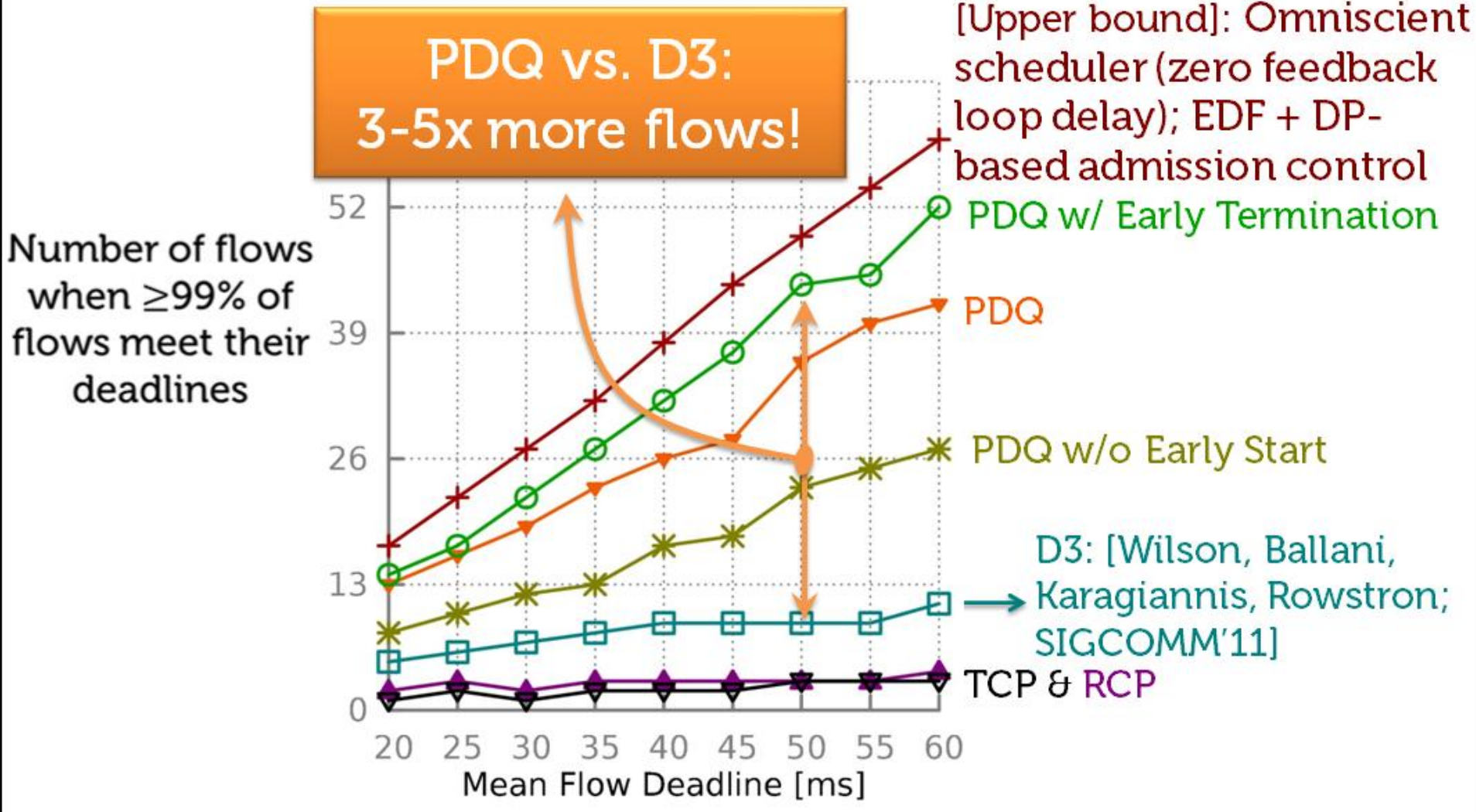
Solution:
rate controller at switches
[XCP/TeXCP/D3]



Mean flow completion time: PDQ vs. Alternatives



Number of "supported" deadline flows: PDQ vs. Alternatives



works well in a "nice" environment, but...

- Topology and Scalability: works well with scale?
- Sending Pattern: works well beyond query aggregation?
- Traffic Pattern: works well in real datacenter workloads?
- Resilience to error: what if packet gets lost or flow information is inaccurate?
- Multipath: does PDQ benefit from multipath forwarding?
- ✔ Flow size estimation: What if estimation fails?
- ✔ Fairness: can long flows starve?

In all of the tested scenarios, PDQ provides significant advantages

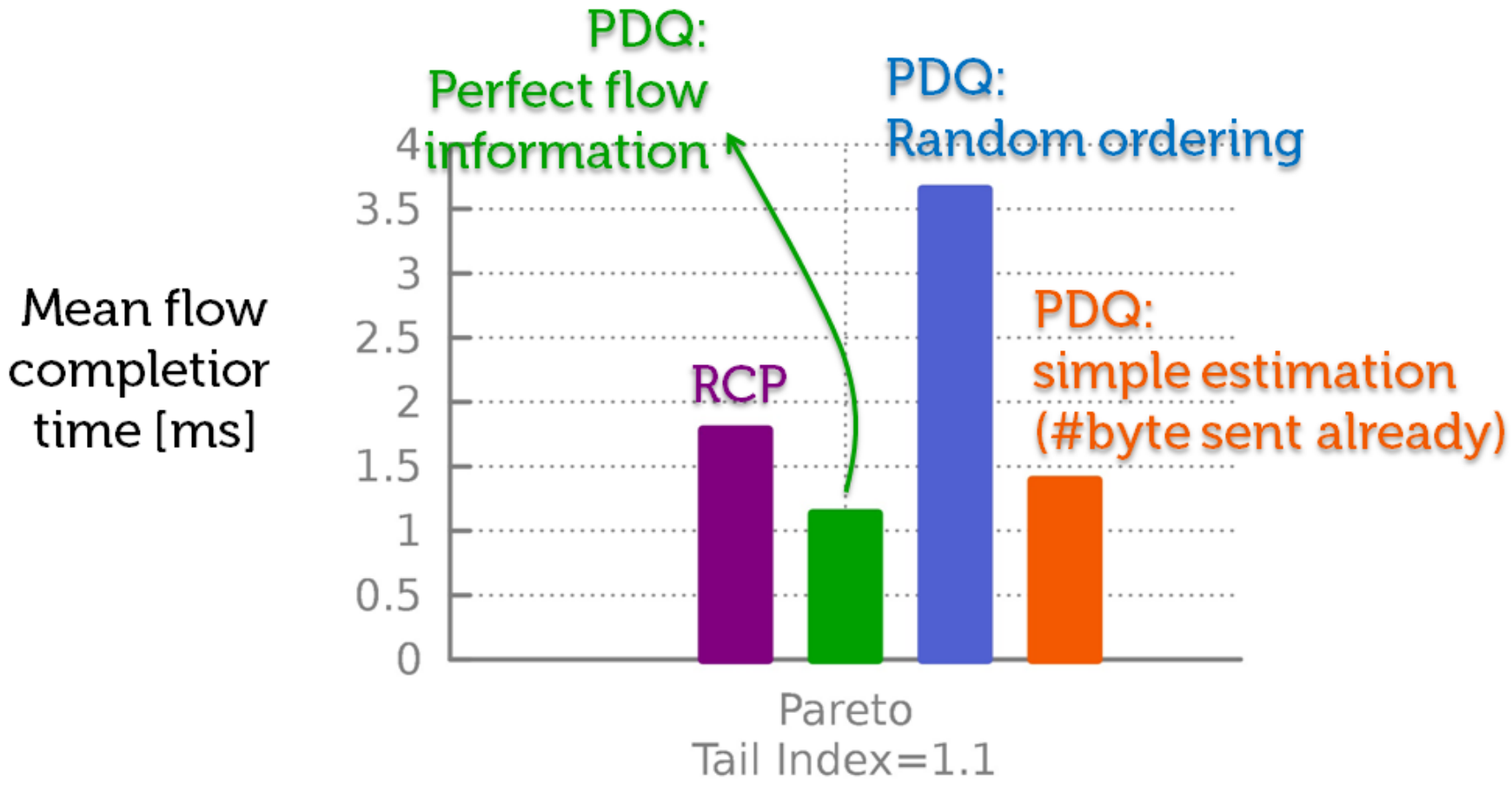
Flow size information

For many data center applications, flow size can be precisely known at flow initialization time

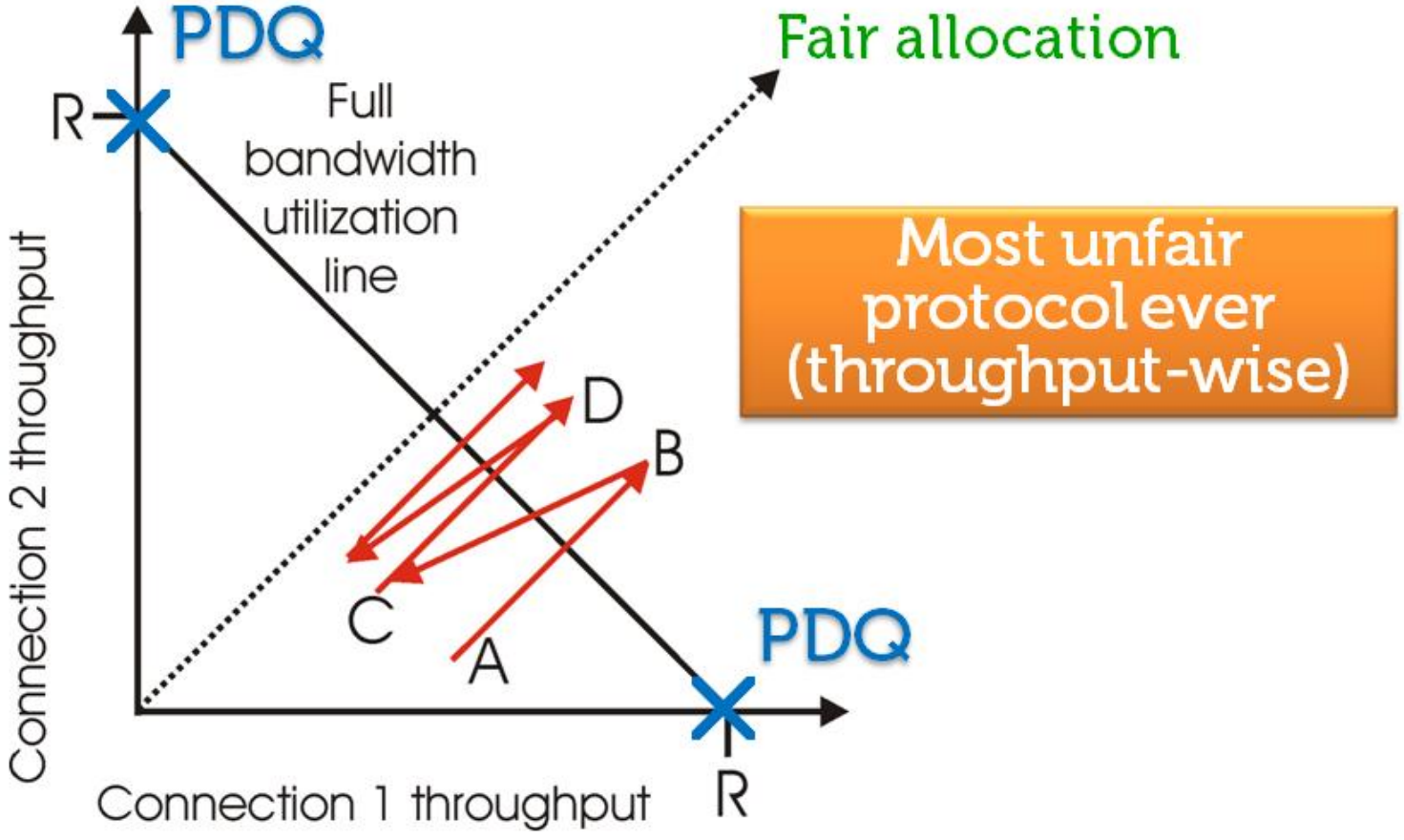
[Wilson, Ballani, Karagiannis, Rowston; SIGCOMM'11]

Otherwise…

Simple flow size estimation works well



Fairness



[Chiu and Jain; Computer Networks and ISDN Systems, 1989]

≥99% of jobs complete faster under SJF than under fair sharing

[Bansal, Harchol-Balter; SIGMETRICS'01]

Assumption: heavy-tailed flow distribution

For datacenter topologies

We found that 85-95% of flows complete faster under PDQ than under RCP

Topologies: Fat-tree, BCube, Jellyfish

Shorten flow completion time tail

Aging: Increase criticality
based on waiting time

Conclusion

Finish flows faster

Improved mean FCT by 16-30% over TCP, RCP and D^3

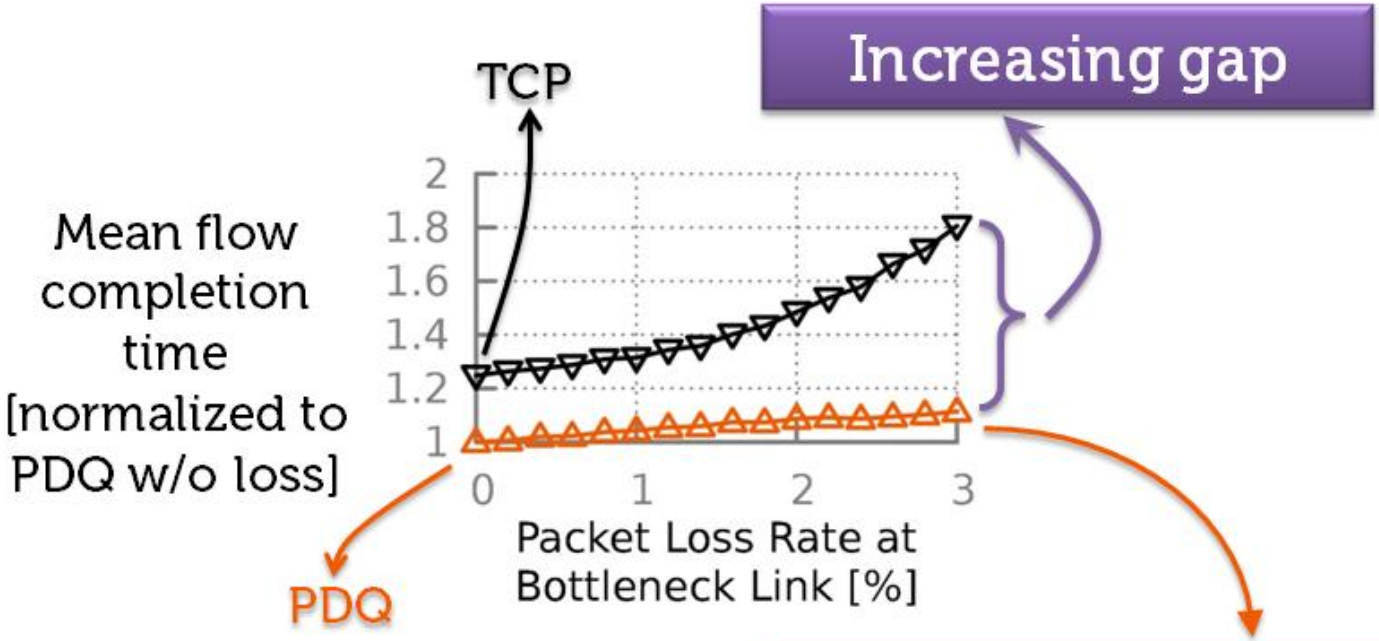
Satisfy flow deadline

3x as many flows as D^3 while meeting flow deadlines

(Multipath) PDQ

Backup slides

Resilience to packet loss



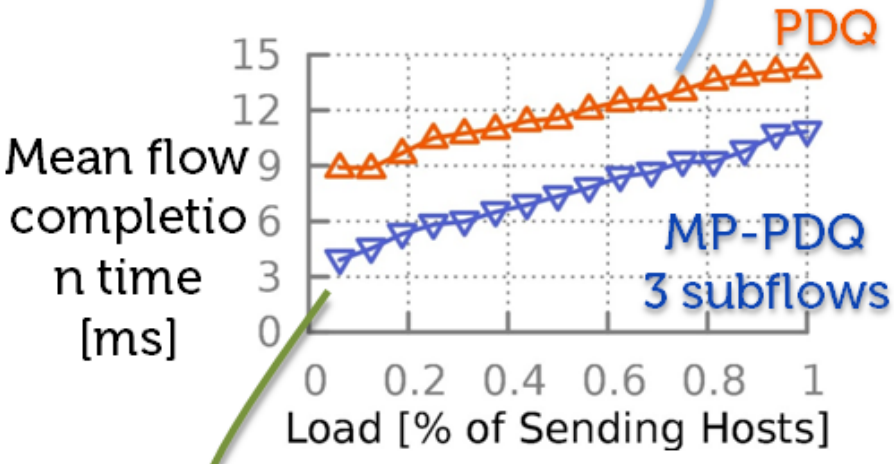
When overall loss rate ~6%, mean flow completion time increased only 12%

Multipath PDQ

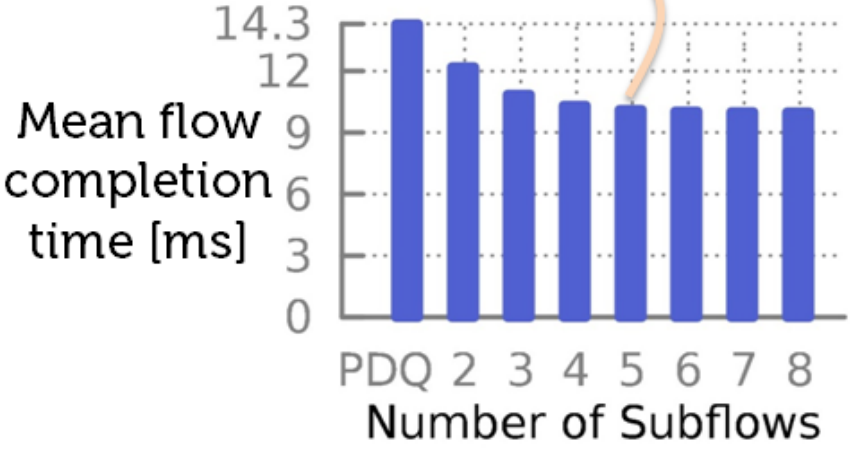
- Design goal: Increasing transmission reliability and exploring path diversity
- *Sender* splits a PDQ flow into equal-size subflows, and periodically shifts the load from the paused subflows to the sending one with minimal remaining load
- *Receiver* maintains per-flow buffer to resequence out-of-order packet arrival
- *Switch* requires zero modification from standard PDQ

Multipath PDQ evaluation

Under heavy loads, MP-PDQ allows flows have higher sending rate



5 subflows to reach its full potential



Under light loads, MP-PDQ exploits more under-utilized links

Formal properties

Deadlock-freedom

Deadlock: two (or more) flows are paused and are each waiting for the other to finish

- "*Hold and Wait*" won't happen in PDQ
  - A flow is accepted *only* after *every* switches along the path accepts the flow

Bounded convergence time

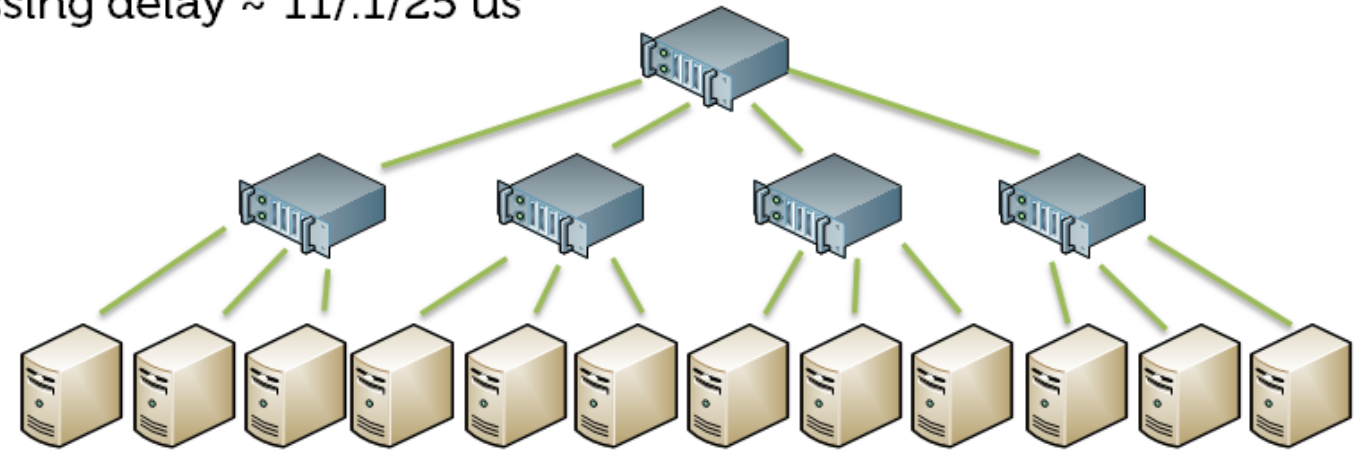
For stable workloads, PDQ will converge to the equilibrium state in at most Pmax+1 RTTs

Default evaluation setting

Packet Level Simulation
discrete events, drops, timeouts, retransmissions

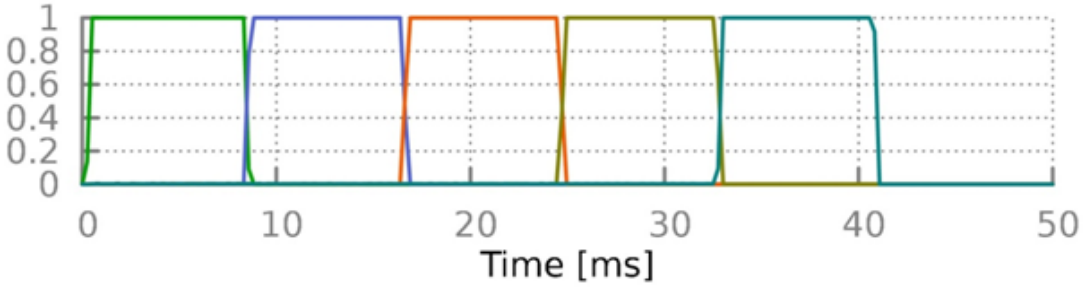
Query Aggregation Scenario
a number of senders initiate flows
at the same time to the same receiver

Three-level Tree
1Gbps links. Per-hop transmission/
propagation/processing delay ~ 11/.1/25 us

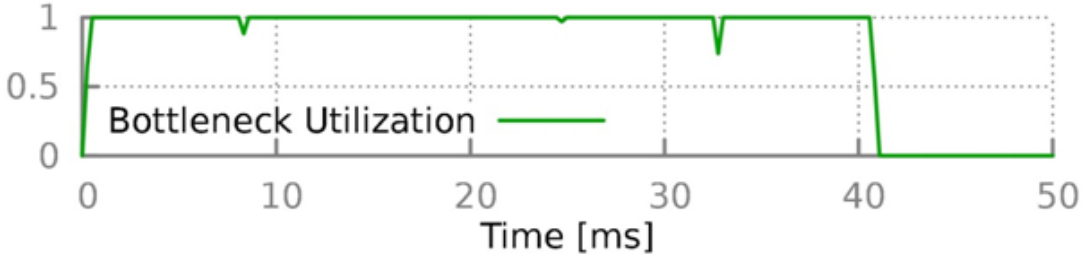


Intuition: PDQ performs well under congestion control metrics

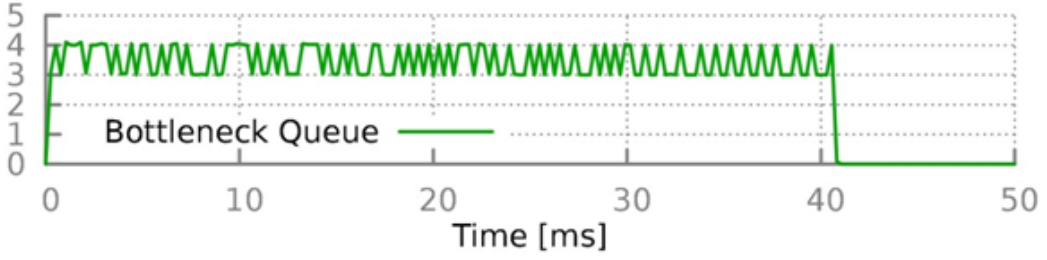
Flow Throughput [Gbps]

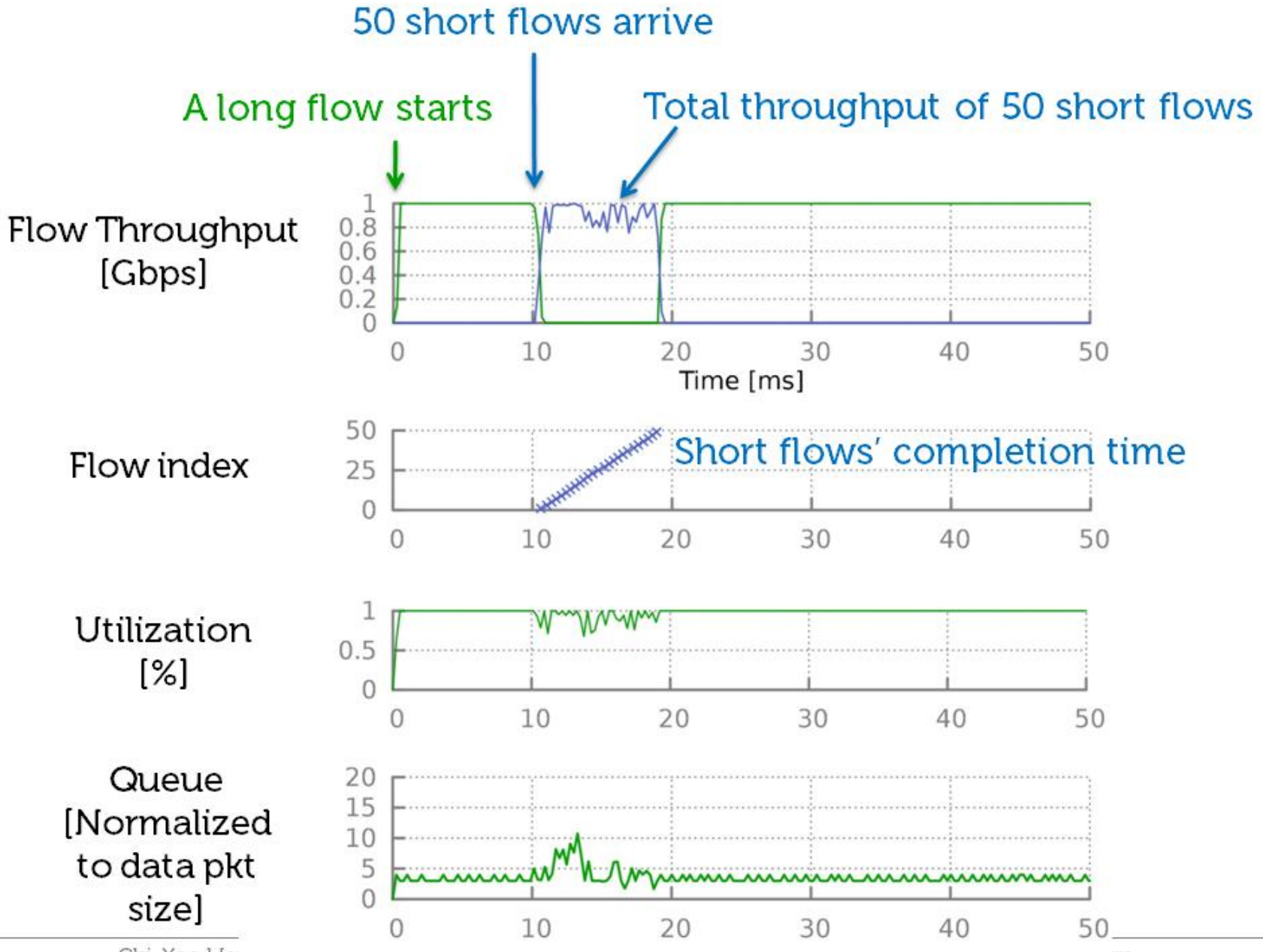


Utilization [%]

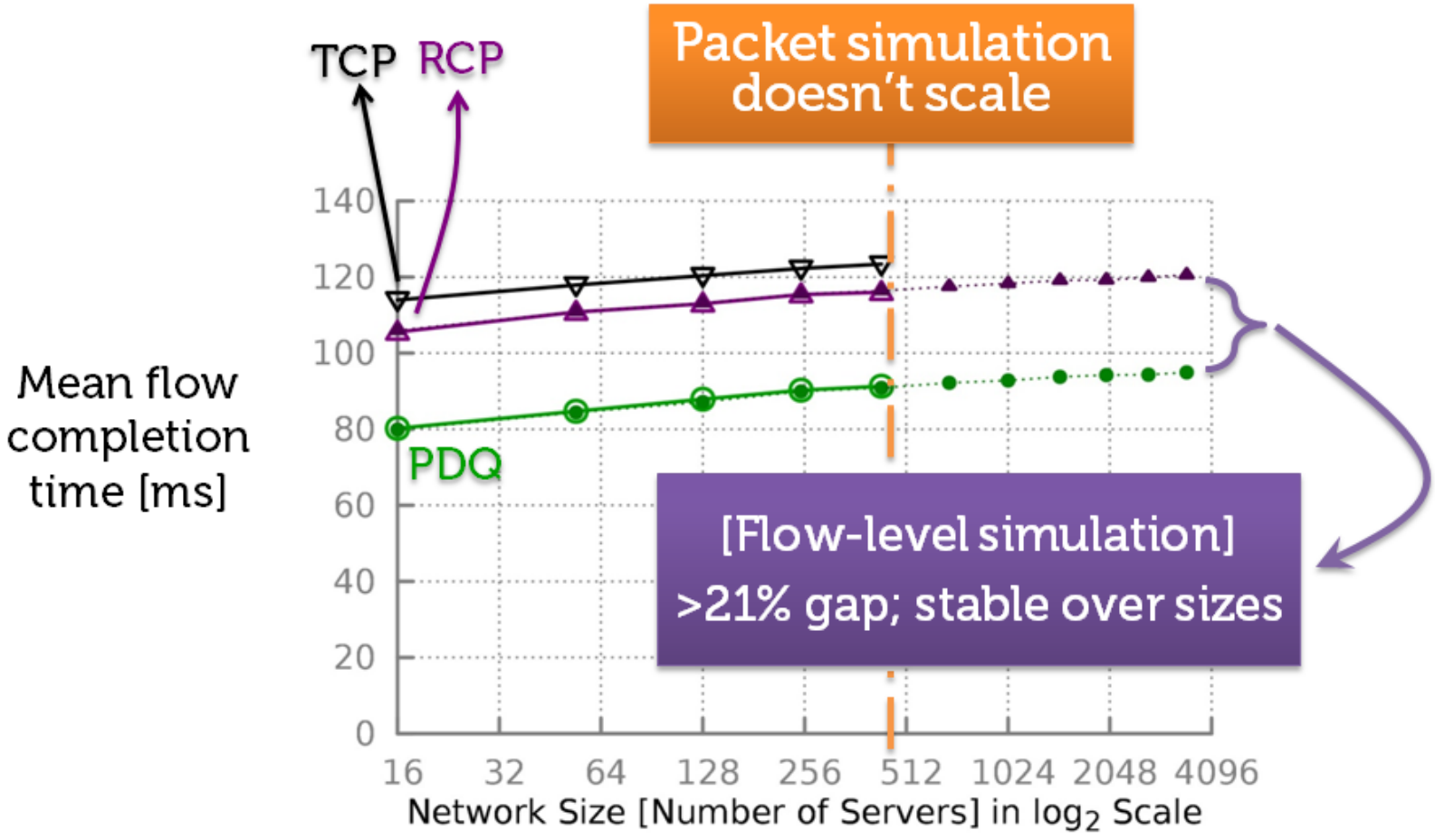


Queue [Normalized to data pkt size]



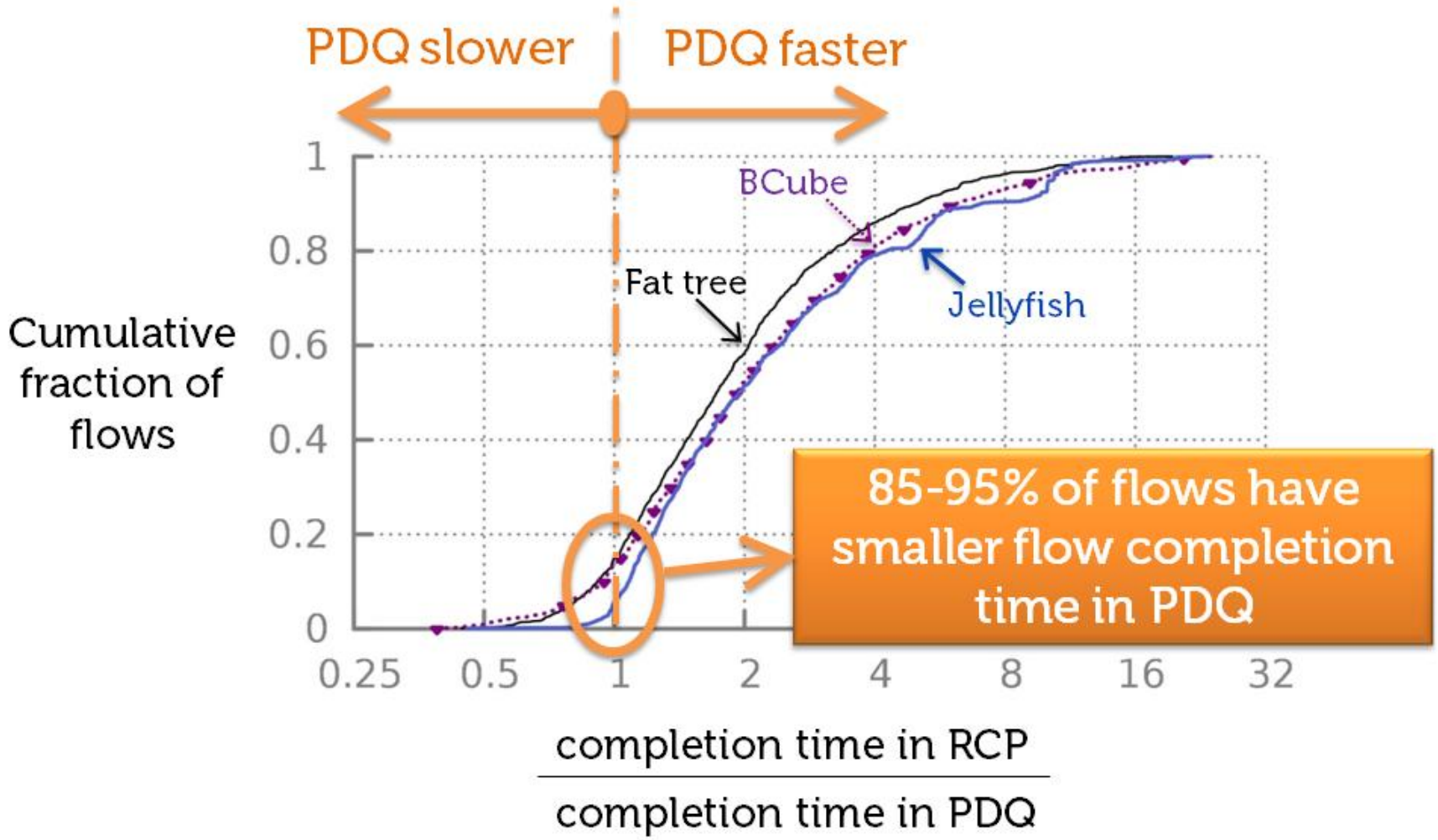

50 short flows arrive
A long flow starts
Total throughput of 50 short flows
Flow Throughput [Gbps]
1
0.8
0.6
0.4
0.2
0
0
10
20
30
40
50
Time [ms]
Flow index
50
25
0
Short flows' completion time
Utilization [%]
1
0.5
0
Queue [Normalized to data pkt size]
20
15
10
5
0

Scalability



Fat tree topology [Al-Fares, Loukissas, Vahdat; SIGCOMM'08]

Fairness



[~128 servers, random permutation traffic]

Shorten flow completion time tail

Aging: Increase criticality based on waiting time