Advanced Topics in Congestion Control

EE122 Fall 2012

Scott Shenker

http://inst.eecs.berkeley.edu/~ee122/

Materials with thanks to Jennifer Rexford, Ion Stoica, Vern Paxson
and other colleagues at Princeton and UC Berkeley

New Lecture Schedule

- T 11/6: Advanced Congestion Control
- Th 11/8: Wireless (Yahel Ben-David)
- T 11/13: Misc. Topics (w/Colin)
  - Security, Multicast, QoS, P2P, etc.
- Th 11/15: Misc. + Network Management
- T 11/20: SDN
- Th 11/22: **Holiday!**
- T 11/27: Alternate Architectures
- Th 11/29: Summing Up (Final Lecture)

Office Hours This Week

- After lecture today
- Thursday 3:00-4:00pm

Announcements

- Participation emails:
  - If you didn't get one, please email Thurston.

- 128 students still haven't participated yet
  - Only seven lectures left
  - You do the math.

Project 3: Ask Panda

Some Odds and Ends about Congestion Control

Clarification about TCP “Modes”

- **Slow-start** mode:
  - CWND =+ MSS on every ACK
  - **[use at beginning, and after time-out]**
- **Congestion avoidance** mode:
  - CWND =+ MSS/(CWND/MSS) on every ACK
  - **[use after CWND>SSTHRESH in slow-start]**
  - **[and after fast retransmit]**
- **Fast restart** mode **[after fast retransmit]**
  - CWND =+ MSS on every dupACK until hole is filled
  - Then revert back to congestion avoidance mode

Delayed Acknowledgments (FYI)

- Receiver generally delays sending an ACK
  - Upon receiving a packet, sets a timer
    - Typically, 200 msec; at most, 500 msec
  - If application generates data, go ahead and send
    - And piggyback the acknowledgment
  - If the timer expires, send a (non-piggybacked) ACK
  - If out-of-order segment arrives, immediately ack
  - (if available window changes, send an ACK)
- Limiting the wait
  - Receiver supposed to ACK at least every second full-sized packet ("ack every other")
    - This is the usual case for "streaming" transfers

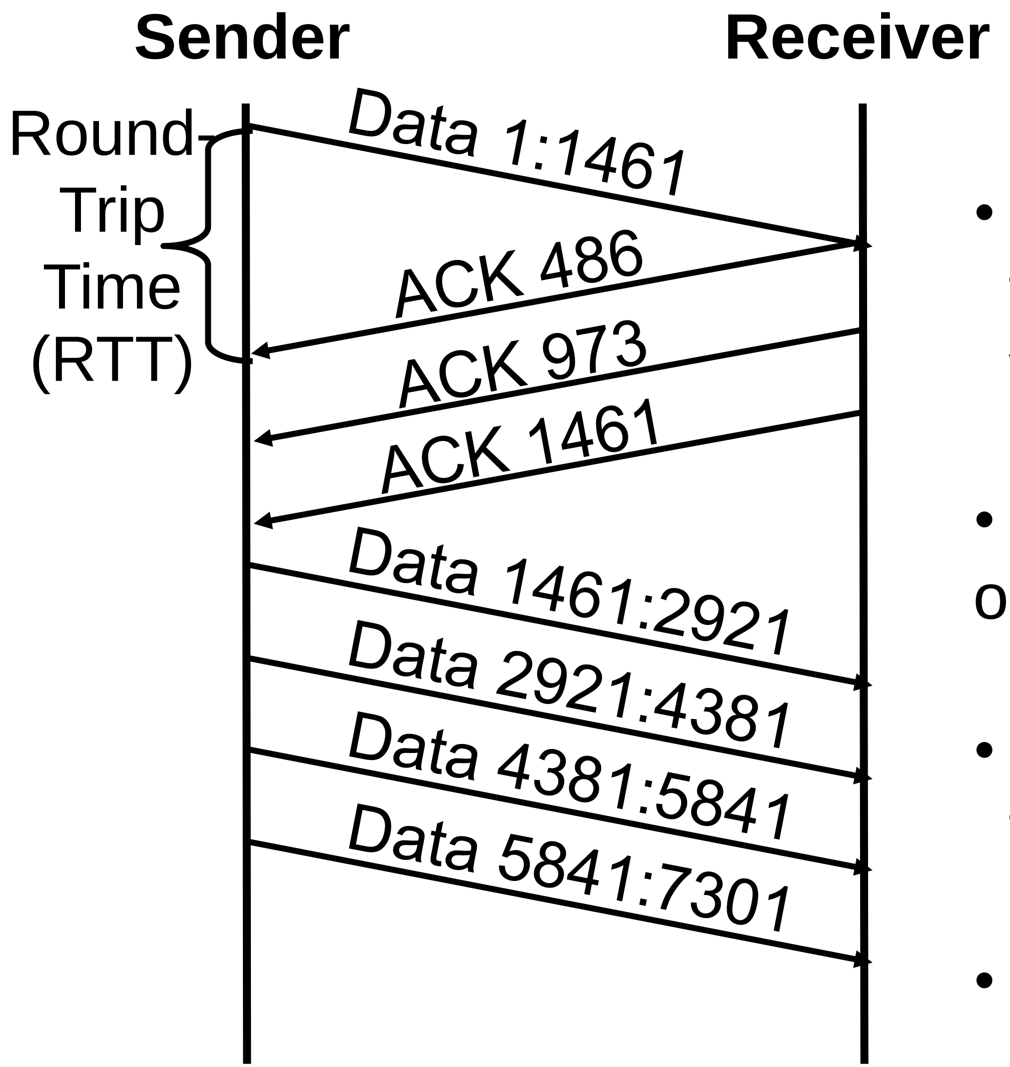
Performance Effects of Acking Policies

- How do delayed ACKs affect performance?
  - Increases RTT
  - Window slides a bit later $\Rightarrow$ throughput a bit lower

- How does ack-every-other affect performance?
  - ***If*** sender adjusts CWND on incoming ACKs, then CWND opens more slowly
    - In slow start, 50% increase/RTT rather than 100%
    - In congestion avoidance, +1 MSS / 2 RTT, not +1 MSS / RTT

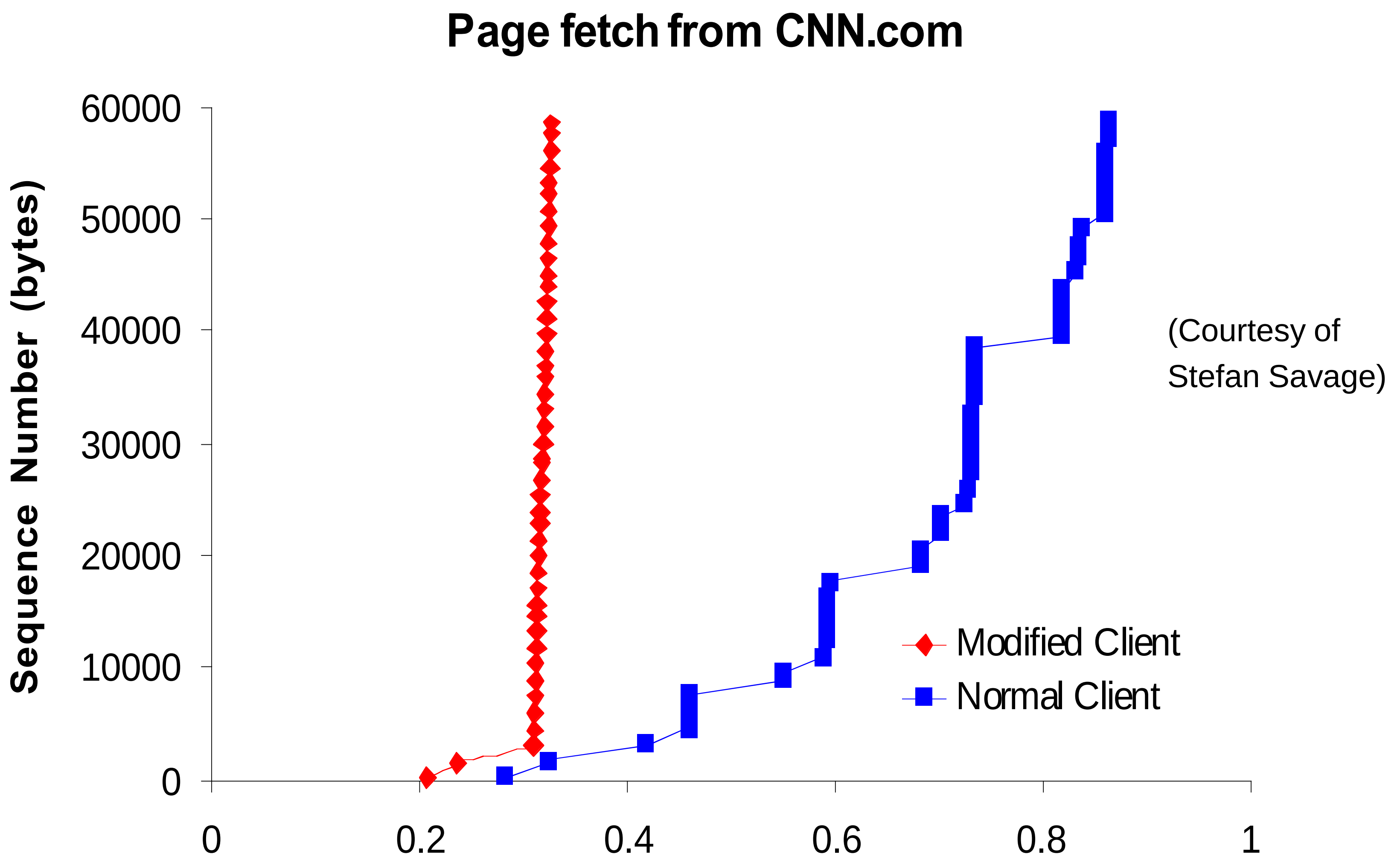
- What does this suggest about how a receiver might cheat and speed up a transfer?

ACK-splitting



- Rule: grow window by one full-sized packet for each valid ACK received
- Send **M** (distinct) ACKs for one packet
- Growth factor proportional to **M**
- What's the fix?

10 line change to Linux TCP



Problems with Current Approach to Congestion Control

Goal of Today's Lecture

- AIMD TCP is the conventional wisdom
- But we know how to do much better
- Today we discuss some of those approaches…

Problems with Current Approach?

- Take five minutes….

TCP fills up queues

- Means that delays are large for everyone

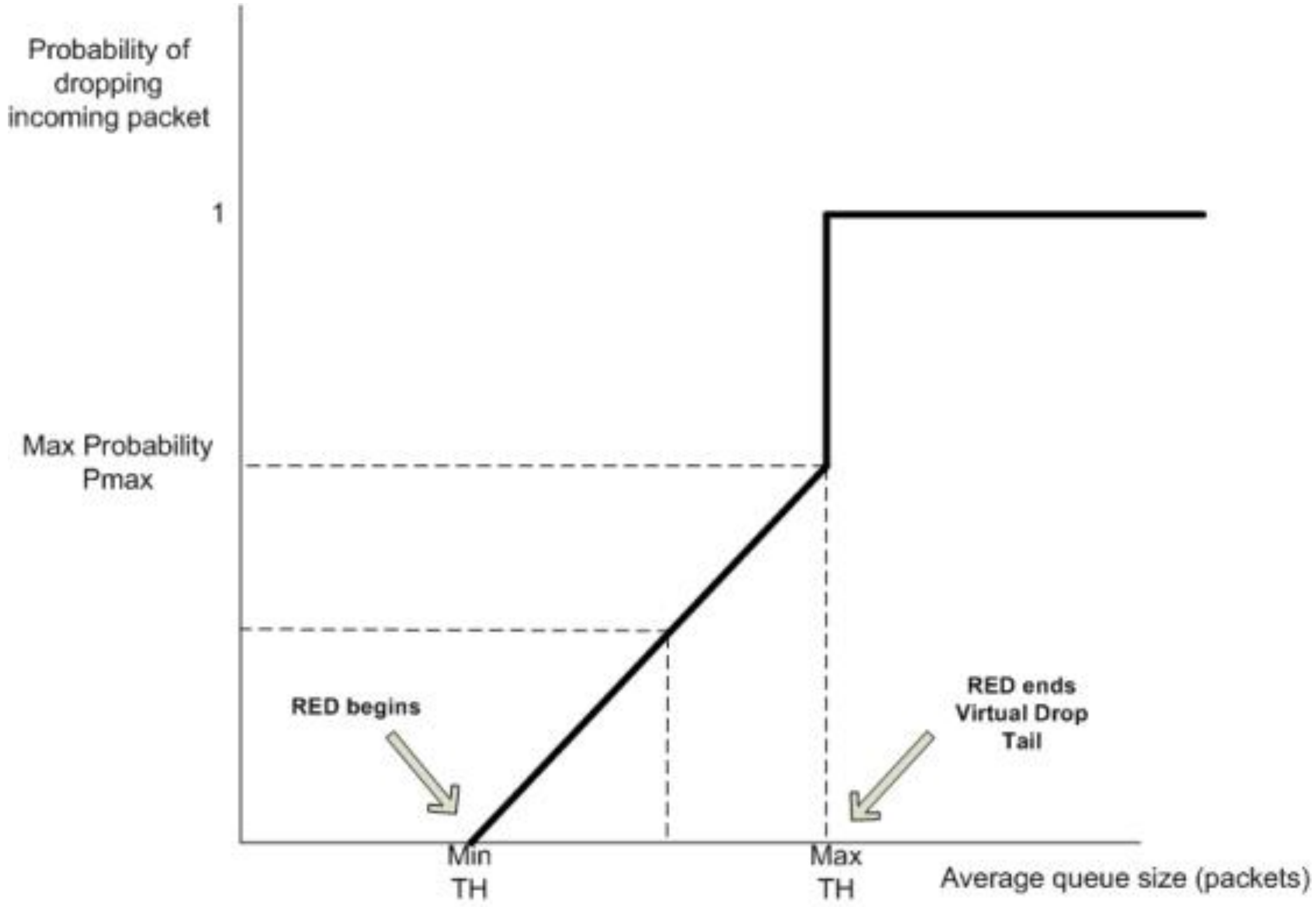
- And when you do fill up queues, many packets have to be dropped
  - *Not always, but it does tend to increase packet drops*

- Alternative: Random Early Drop (LBL)
  - Drop packets on purpose before queue is full

Random Early Drop (or Detection)

- Measure average queue size ***A*** with exp. weighting
  - Allows short bursts of packets without over-reacting
- Drop probability is a function of ***A***
  - No drops if ***A*** is very small
  - Low drop rate for moderate ***A***’s
  - Drop everything if ***A*** is too big

RED Dropping Probability



Advantages of RED

- Keeps queues smaller, while allowing bursts
  - Just using small buffers in routers can’t do the latter
- Reduces synchronization between flows
  - Not all flows are dropping packets at once

What if loss isn't congestion-related?

- Can use Explicit Congestion Notification (ECN)
- Bit in IP packet header (actually two)
  - TCP receiver returns this bit in ACK
- When RED router would drop, it sets bit instead
  - Congestion semantics of bit exactly like that of drop
- Advantages:
  - Doesn't confuse corruption with congestion
  - Doesn't confuse recovery with rate adjustment

How does AIMD work at high speed?

- Throughput = (MSS/RTT) sqrt(3/2p)
  - Assume that RTT = 100ms, MSS=1500bytes

- What value of p is required to go 100Gbps?
  - Roughly 2×10^{-12}
- How long between drops?
  - Roughly 16.6 hours
- How much data has been sent in this time?
  - Roughly 6 petabits

- These are not practical numbers!

Adapting TCP to High Speed

- One approach:
  - Let AIMD constants depend on CWND
- At very high speeds,
  - Increase CWND by more than MSS in a RTT
  - Decrease CWND by less than ½ after a loss
- We will discuss other approaches later…

High-Speed TCP Proposal

Bandwidth	Avg Cwnd w (pkts)	Increase a(w)	Decrease b(w)
1.5 Mbps	12.5	1	0.50
10 Mbps	83	1	0.50
100 Mbps	833	6	0.35
1 Gbps	8333	26	0.22
10 Gbps	83333	70	0.10

This changes the TCP Equation

- Throughput ~ $p^{-.8}$ (rather than $p^{-.5}$)

- Whole point of design: to achieve a high throughput, don't need such a tiny drop rate….

How “Fair” is TCP?

- Throughput depends inversely on RTT
- If open K TCP flows, get K times more bandwidth!
- What is fair, anyway?

What happens if hosts "cheat"?

- Can get more bandwidth by being more aggressive
  - Source can set CWND =+ 2MSS upon success
  - Gets much more bandwidth (see forthcoming HW4)
- Currently we require all congestion-control protocols to be "TCP-Friendly"
  - To use no more than TCP does in similar setting
- But Internet remains vulnerable to non-friendly implementations
  - Need router support to deal with this…

Router-Assisted Congestion Control

- There are two different tasks:
  - Isolation/fairness
  - Adjustment

Adjustment

- Can routers help flows reach right speed faster?
  - Can we avoid this endless searching for the right rate?

- Yes, but we won’t get to this for a few slides….

Isolation/fairness

- Want each flow gets its “fair share”
  - No matter what other flows are doing
- This protects flows from cheaters
  - ***Safety/Security issue***
- Does not require everyone use same CC algorithm
  - ***Innovation issue***

Isolation: Intuition

- Treat each “flow” separately
  - For now, flows are packets between same Source/Dest.
- Each flow has its own FIFO queue in router
- Service flows in a round-robin fashion
  - When line becomes free, take packet from next flow
- Assuming all flows are sending MTU packets, all flows can get their fair share
  - But what if not all are sending at full rate?
  - And some are sending at more than their share?

Max-Min Fairness

- Given set of bandwidth demands r_i and total bandwidth C, max-min bandwidth allocations are:

$$a_i = \min(f, r_i)$$

where f is the unique value such that Sum(a_i) = C

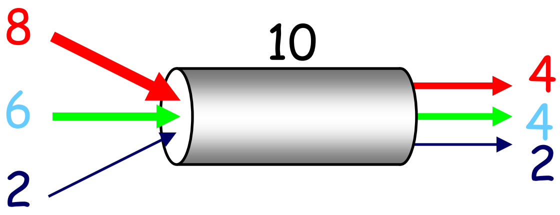
- This is what round-robin service gives
  - if all packets are MTUs
- Property:
  - If you don't get full demand, no one gets more than you
  - Use it or lose it: you don't get credit for not using link

Example

- Assume link speed C is 10mbps
- Have three flows:
  - Flow 1 is sending at a rate 8mbps
  - Flow 2 is sending at a rate 6mbps
  - Flow 3 is sending at a rate 2mbps
- How much bandwidth should each get?
  - According to max-min fairness?

Example

- C = 10; r_1 = 8, r_2 = 6, r_3 = 2; N = 3
- $C/3 = 3.33 \rightarrow$
  - Can service all of r_3
  - Remove r_3 from the accounting: $C = C - r_3 = 8$; $N = 2$
- $C/2 = 4 \rightarrow$
  - Can't service all of r_1 or r_2
  - So hold them to the remaining fair share: $f = 4$



f = 4:
min(8, 4) = 4
min(6, 4) = 4
min(2, 4) = 2

Fair Queuing (FQ)

- Implementation of round-robin generalized to case where not all packets are MTUs

- Weighted fair queueing (WFQ) lets you assign different flows different shares

- WFQ is implemented in almost all routers
  - Variations in how implemented
    - Packet scheduling (here)
    - Just packet dropping (AFD)

Enforcing fairness through dropping

- Drop rate for flow i should be $d_i = (1 - r_{fair}/r_i)_+$
- Resulting rate for flow is $r_i(1-d_i)=MIN[r_i,r_{fair}]$
- Estimate r_i with "shadow buffer" of recent packets
  - Estimate is terrible for small r_i, but $d_i = 0$ for those
  - Estimate is decent for large r_i, and that's all that matters!
- Implemented on much of Cisco's product line
  - Approximate Fair Dropping (AFD)

With Fair Queueing or AFD Routers

- Flows can pick whatever CC scheme they want
  - Can open up as many TCP connections as they want
- There is no such thing as a “cheater”
  - To first order…
- Bandwidth share does not depend on RTT
- Does require some complication on router
  - But certainly within reason

FQ is really “processor sharing”

- PS is really just round-robin at bit level
  - Every current flow with packets gets same service rate
- When flows end, other flows pick up extra service
- FQ realizes these rates through packet scheduling
  - AFD through packet dropping
- But we could just assign them directly
  - This is the Rate-Control Protocol (RCP) [Stanford]
    - Follow on to XCP (MIT/ICSI)

RCP Algorithm

- Packets carry “rate field”
- Routers insert “fair share” f in packet header
  - Router inserts FS only if it is smaller than current value
- Routers calculate f by keeping link fully utilized
  - Remember basic equation: $Sum(Min[f,r_i]) = C$

Fair Sharing is more than a moral issue

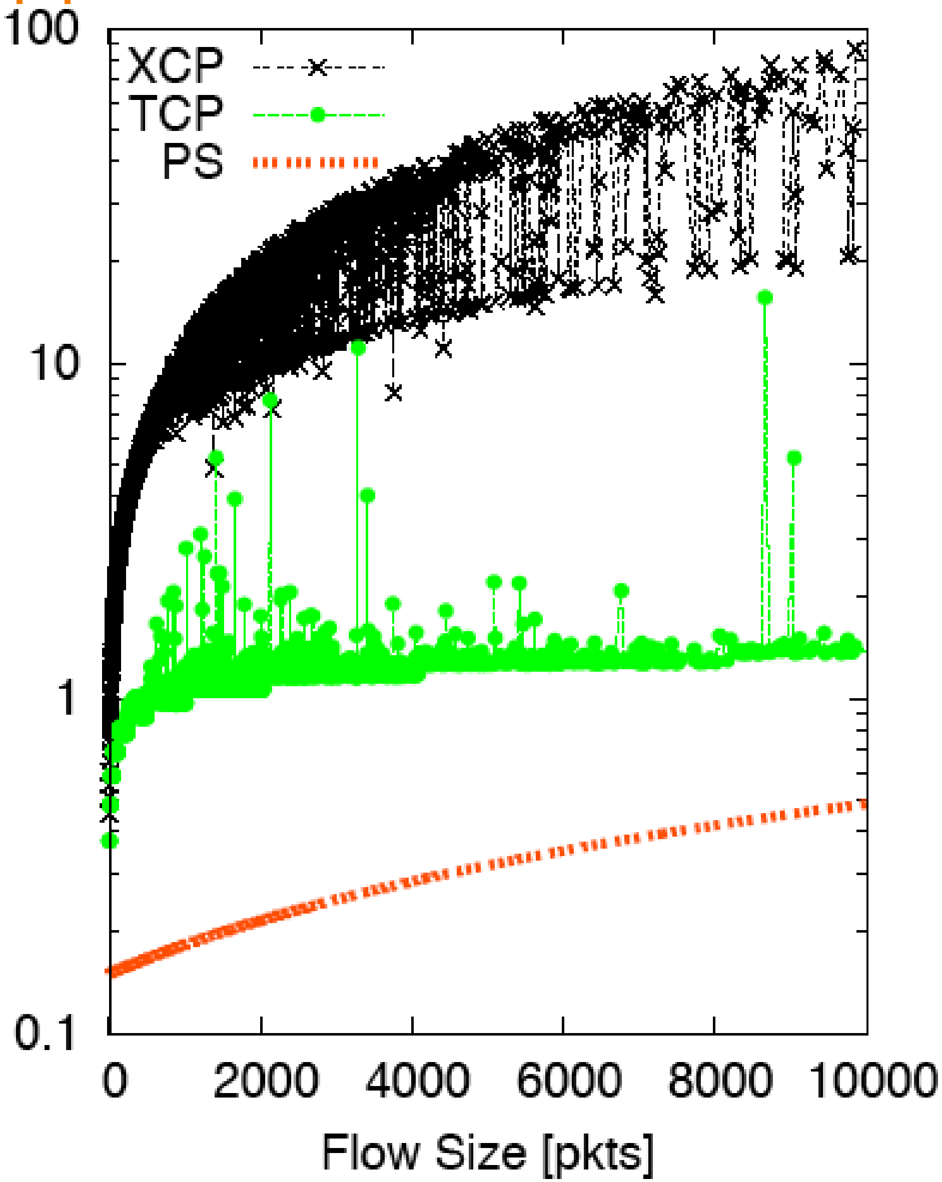
- By what metric should we evaluate CC?

- One metric: average flow completion time (FCT)

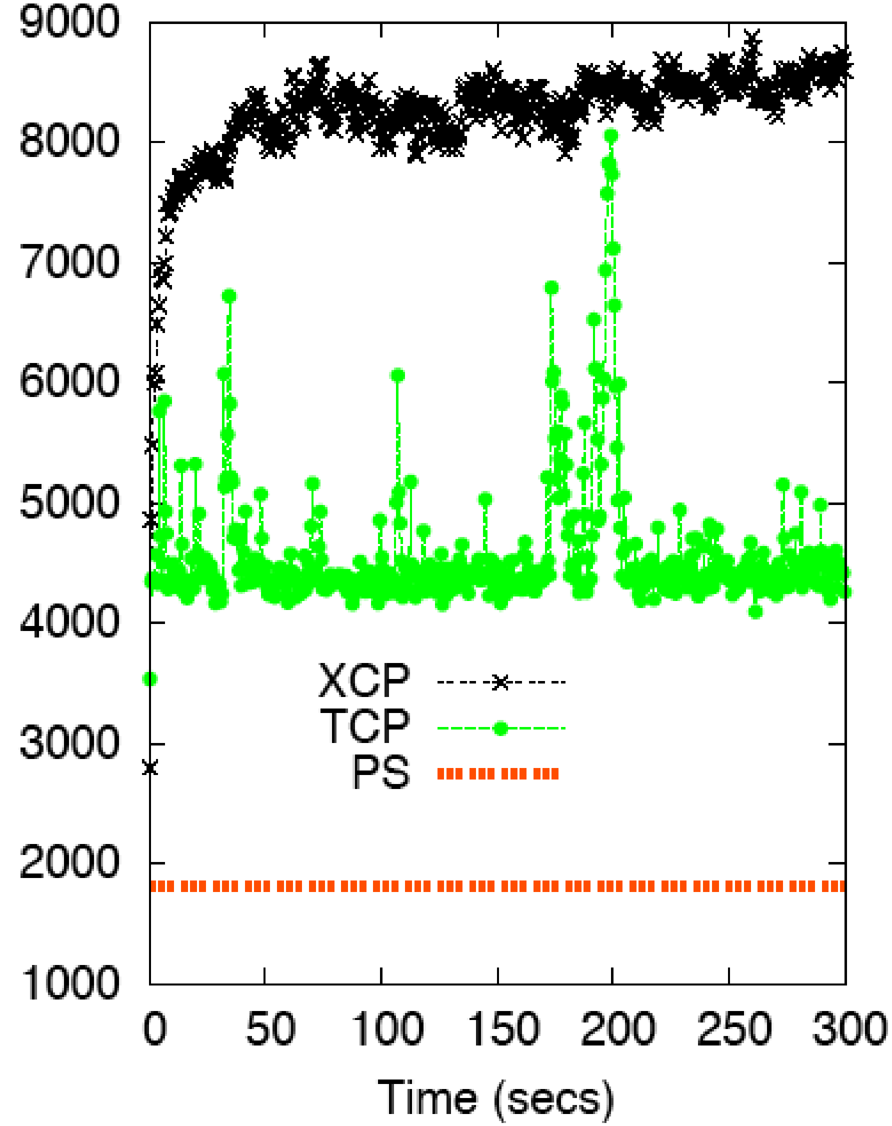
- Let's compare FCT with RCP and TCP
  - Ignore XCP curve….

Flow Completion Time: TCP vs. PS (and XCP)

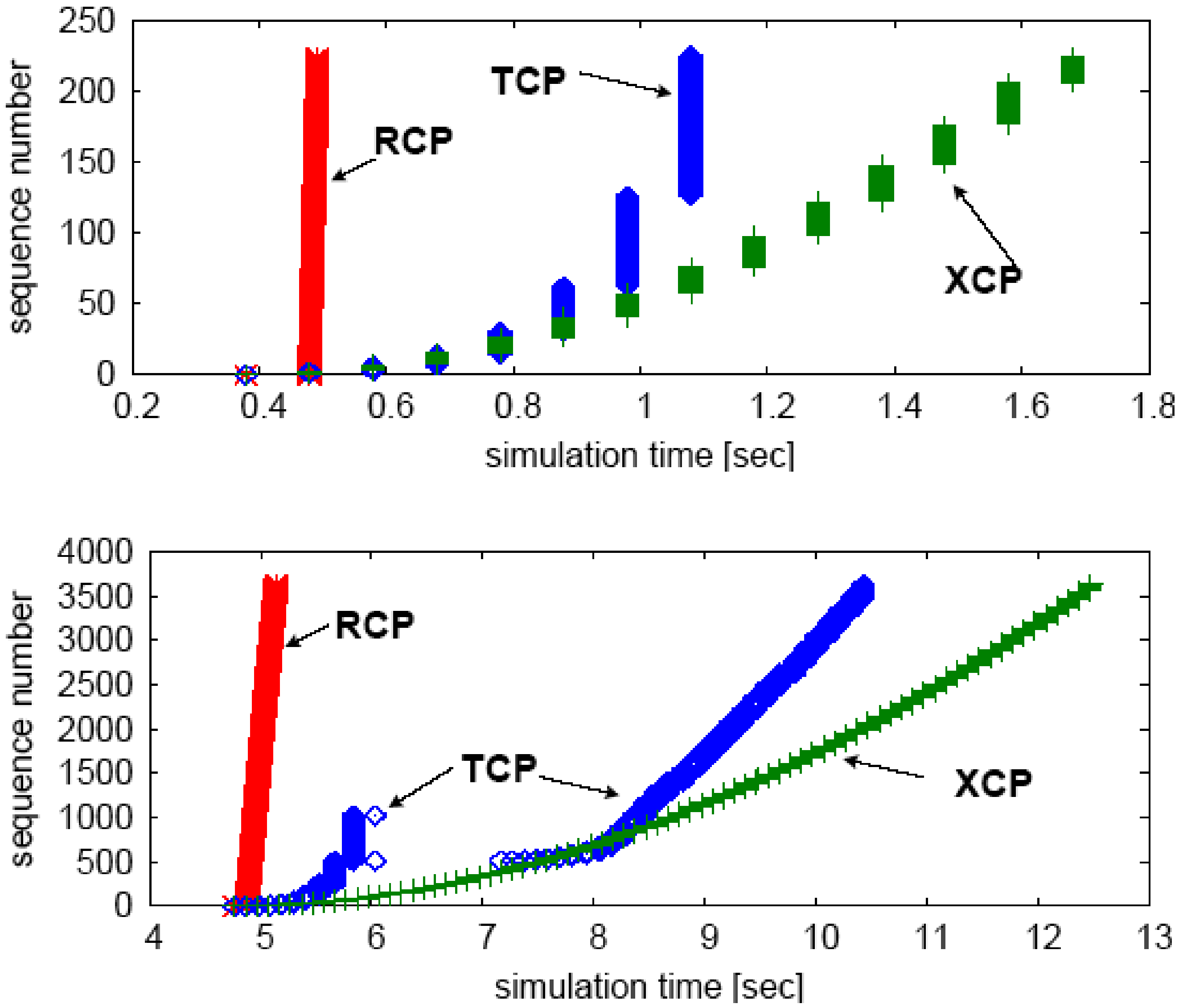
Flow Duration (secs) vs. Flow Size



Active Flows vs. time



Why the improvement?



RCP (and similar schemes)

- They address the "adjustment" question
- Help flows get up to full rate in a few RTTs
- Fairness is merely a byproduct of this approach
  - One could have assigned different rates to flows

Summary of Router Assisted CC

- Adjustment: helps get flows up to speed
  - Huge improvement in FTC performance
- Isolation: helps protect flows from cheaters
  - And allows innovation in CC algorithms
- FQ/AFD impose “max-min fairness”
  - On each link, each flow has right to fair share

Why is Scott a Moron?

Or why does Bob Briscoe think so?

Giving equal shares to “flows” is silly

- What if you have 8 flows, and I have 4…
  - Why should you get twice the bandwidth?
- What if your flow goes over 4 congested hops, and mine only goes over 1?
  - Why not penalize for using more scarce bandwidth?
- And what is a flow anyway?
  - TCP connection
  - Source-Destination pair?
  - Source?

flow rate fairness
dismantling a religion

<draft-briscoe-tsvarea-fair-01.pdf>

status: individual draft
final intent: informational
intent next: tsvwg WG item after (or at) next draft

Bob Briscoe
Chief Researcher, BT Group
IETF-68 tsvwg Mar 2007

Charge people for congestion!

- Use ECN as congestion markers
- Whenever I get ECN bit set, I have to pay $$$
- No debate over what a flow is, or what fair is…
- Idea started by Frank Kelly, backed by much math
  - Great idea: simple, elegant, effective
  - **Never going to happen…**

Datacenter Networks

What makes them special?

- Huge scale:
  - 100,000s of servers in one location
- Limited geographic scope:
  - High bandwidth (10Gbps)
  - Very low RTT
- Extreme latency requirements
  - With real money on the line
- Single administrative domain
  - No need to follow standards, or play nice with others
- Often “green field” deployment
  - So can “start from scratch”…

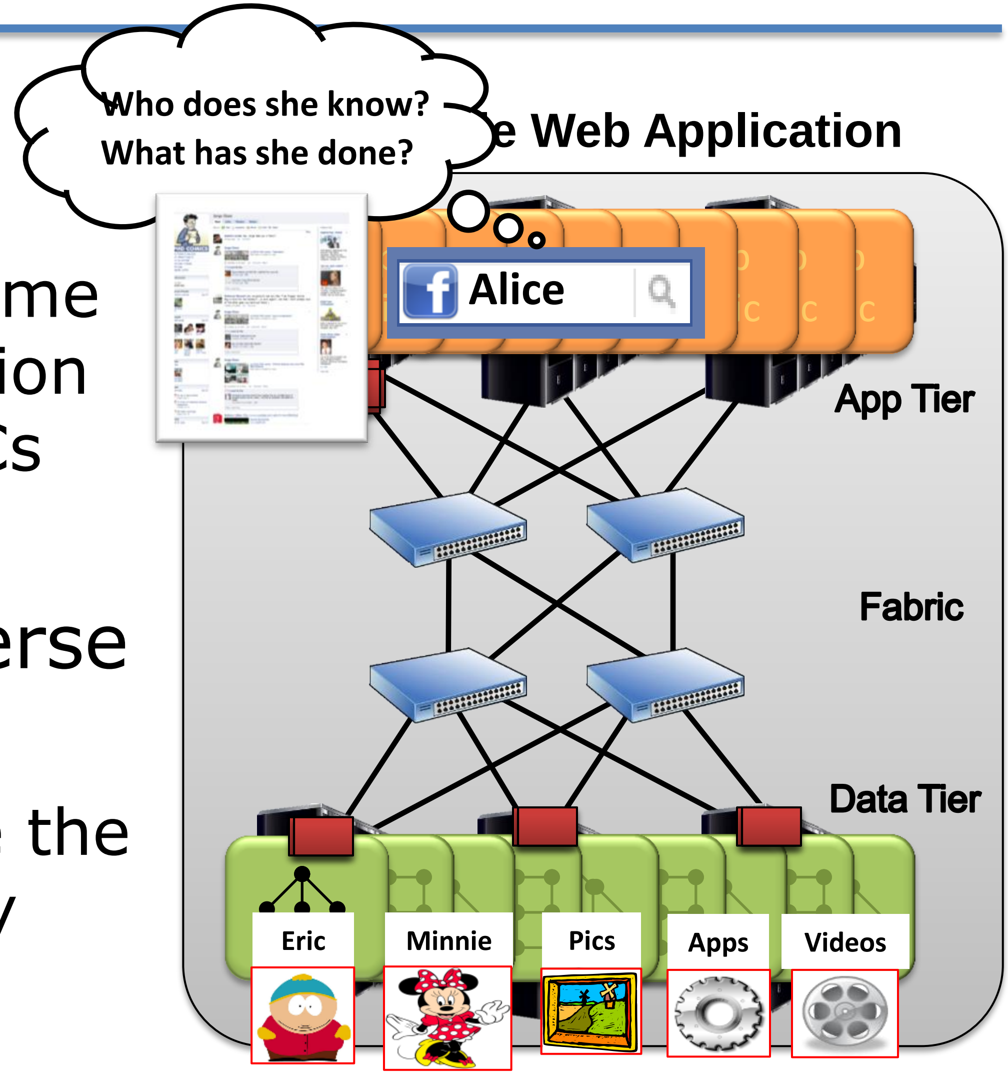
Deconstructing Datacenter Packet Transport

Mohammad Alizadeh, Shuang Yang, Sachin Katti,
Nick McKeown, Balaji Prabhakar, Scott Shenker

Stanford University U.C. Berkeley/ICSI

Transport in Datacenters

- Latency is King
  - Web app response time depends on completion of 100s of small RPCs
- But, traffic also diverse
  - Mice AND Elephants
  - Often, elephants are the root cause of latency



Transport in Datacenters

- Two fundamental requirements
  - **High fabric utilization**
    - Good for all traffic, esp. the large flows
  - **Low fabric latency (propagation + switching)**
    - Critical for latency-sensitive traffic

- Active area of research
  - DCTCP[SIGCOMM'10], D3[SIGCOMM'11]
    HULL[NSDI'11], D^2TCP[SIGCOMM'12]
    PDQ[SIGCOMM'12], DeTail[SIGCOMM'12]

vastly improve performance, but fairly complex

pFabric in 1 Slide

Packets carry a single priority #

- e.g., prio = remaining flow size

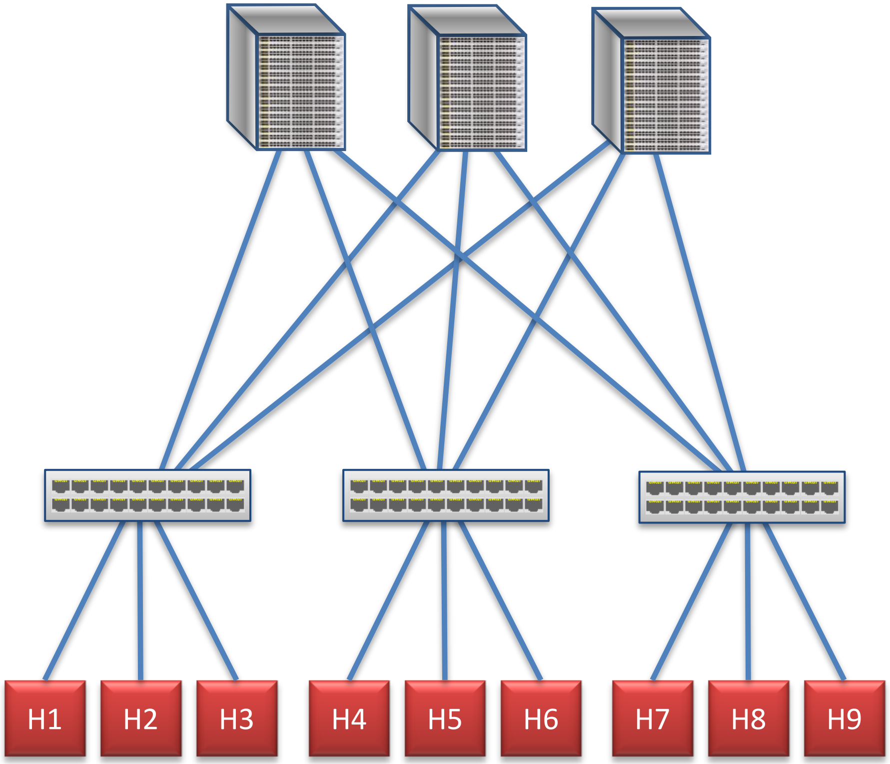
pFabric Switches

- Very small buffers (e.g., 10-20KB)
- Send highest priority / drop lowest priority pkts

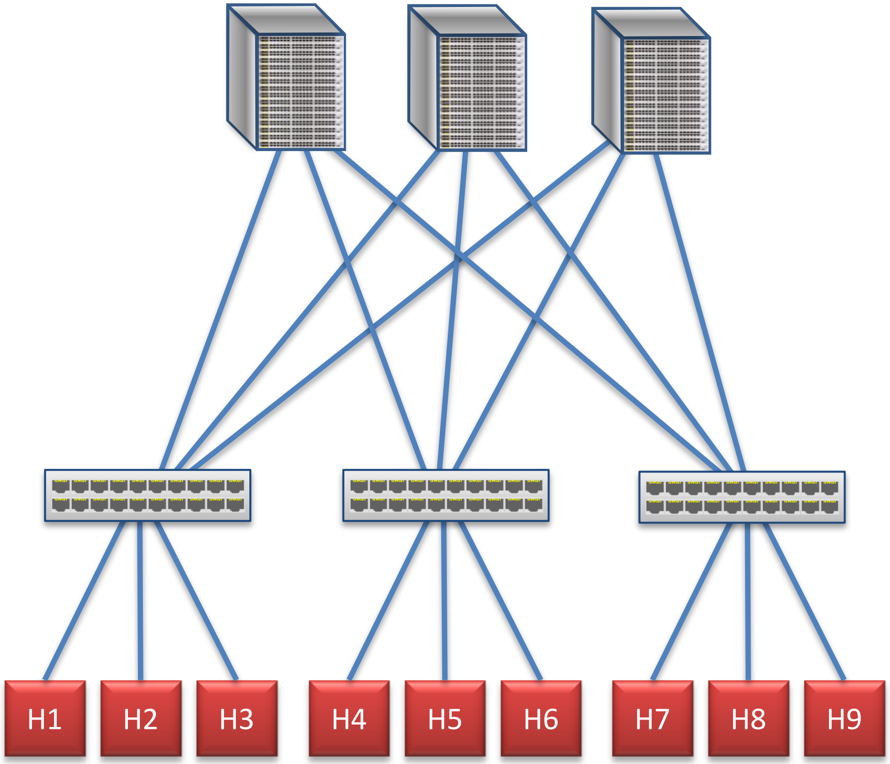
pFabric Hosts

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

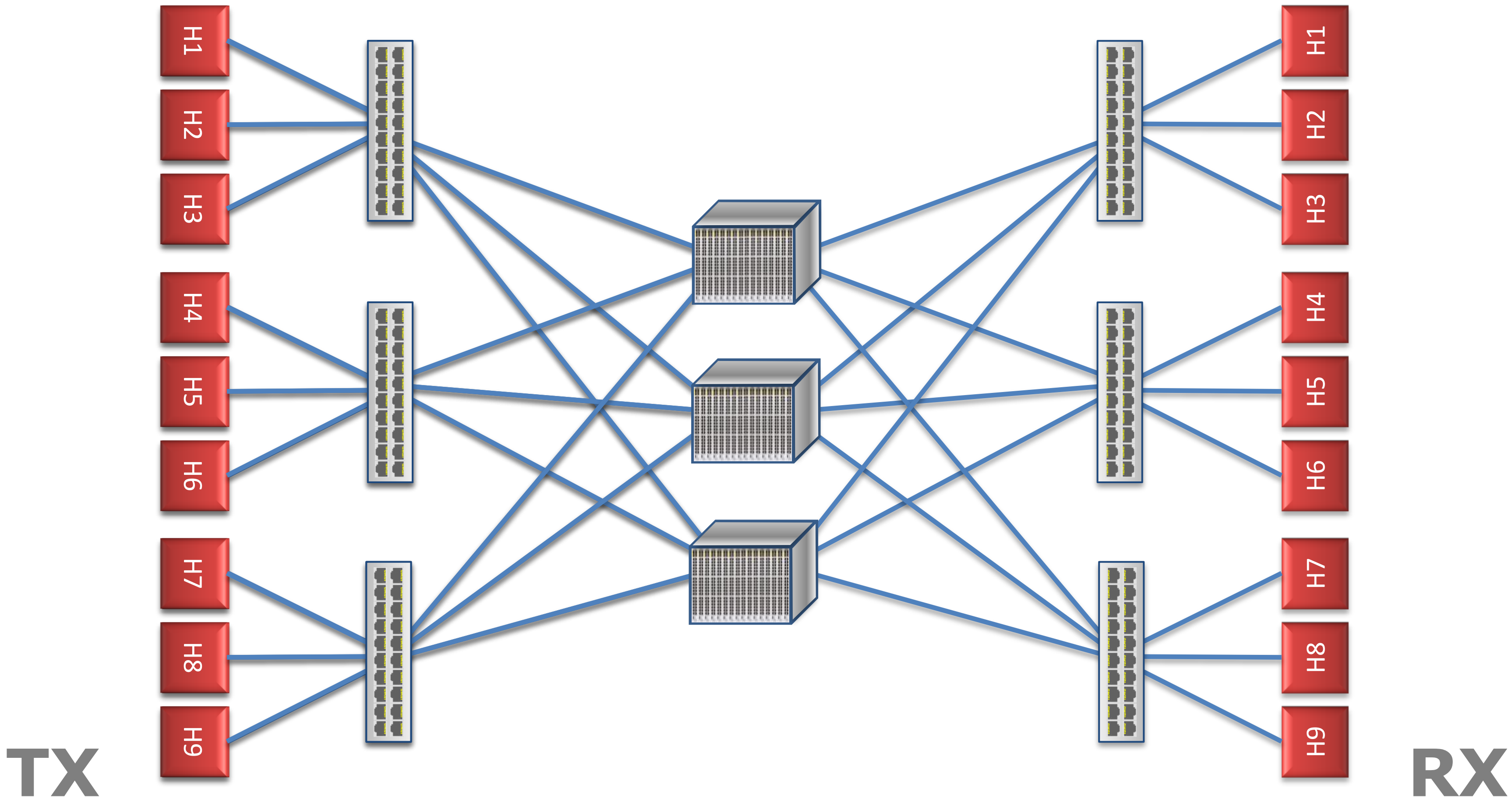
DC Fabric: Just a Giant Switch!



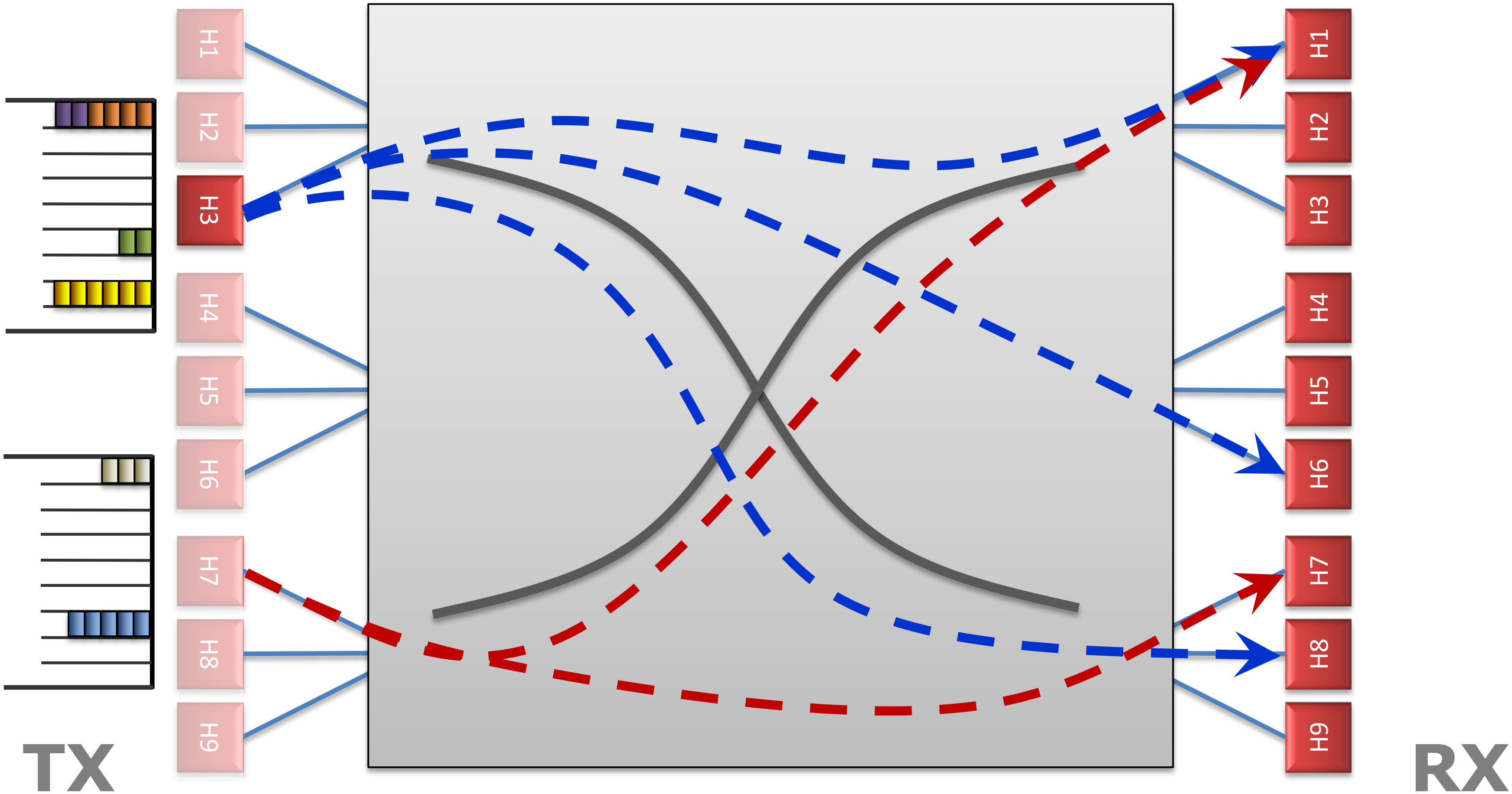
DC Fabric: Just a Giant Switch!



DC Fabric: Just a Giant Switch!



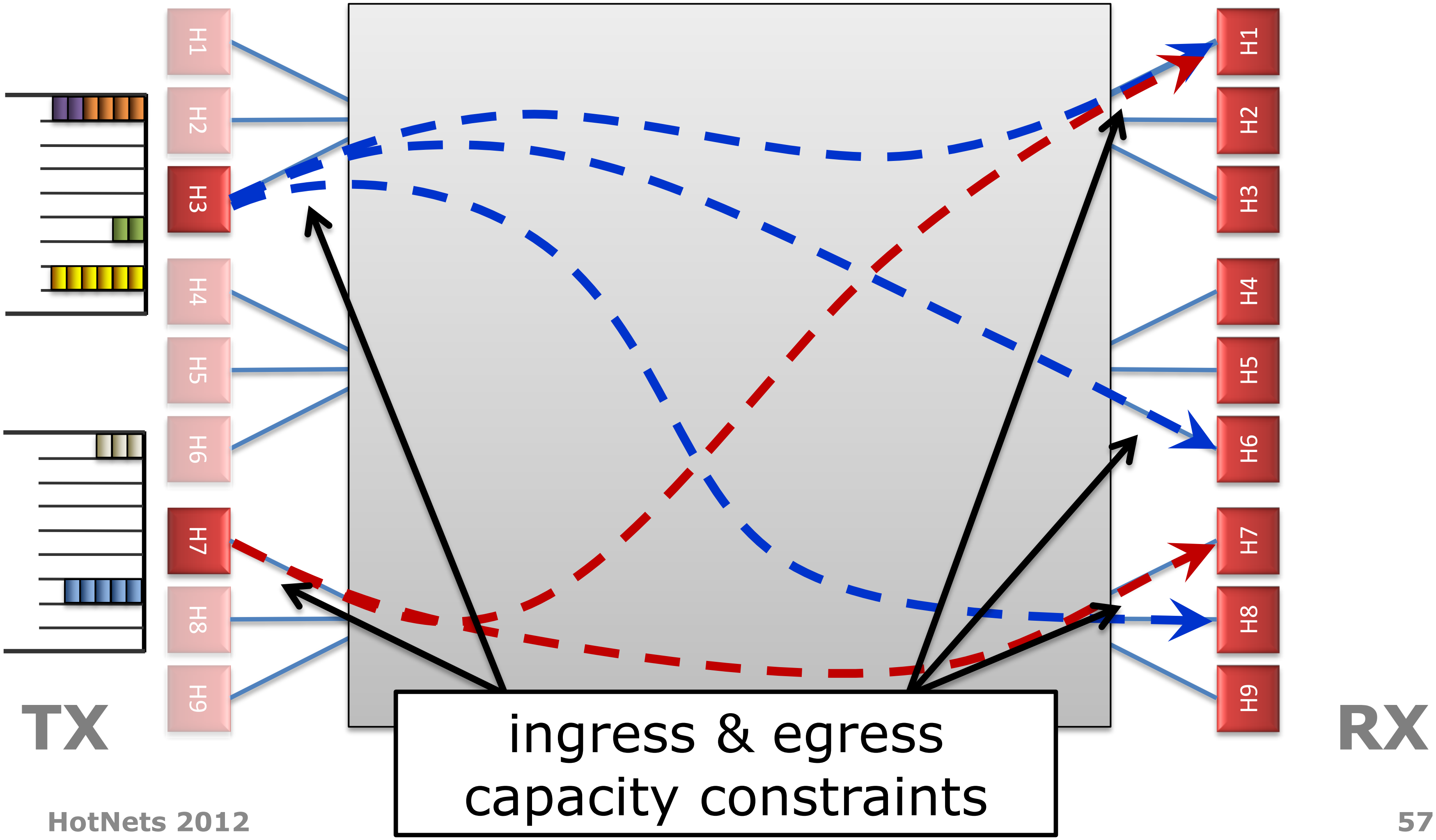
DC Fabric: Just a Giant Switch!



DC transport =
Flow scheduling
on giant switch

Objective?

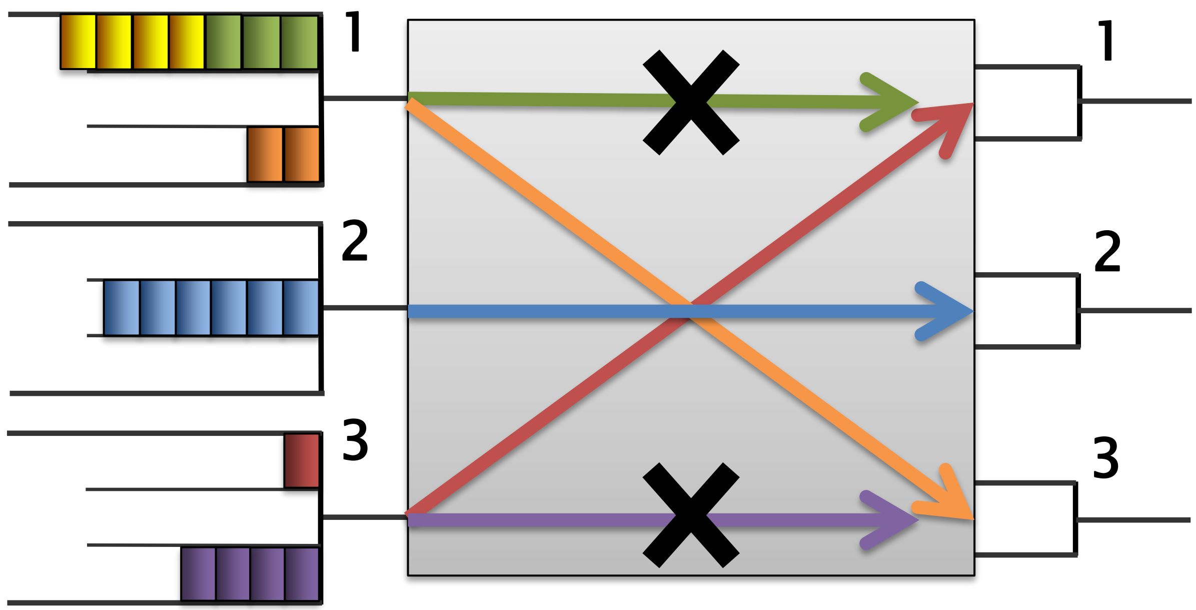
➢ Minimize avg FCT



"Ideal" Flow Scheduling

Problem is NP-hard ☹ [Bar-Noy et al.]

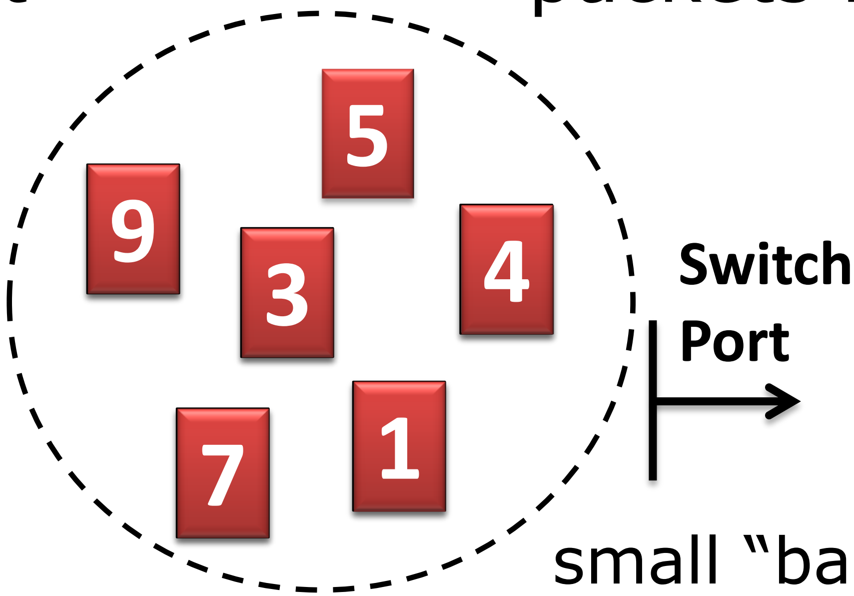
– Simple greedy algorithm: **2-approximation**



pFabric Design

pFabric Switch

- **Priority Scheduling**
  send higher priority packets first
- **Priority Dropping**
  drop low priority packets first



small "bag" of packets per-port

prio = remaining flow size

Near-Zero Buffers

- Buffers are very small (~1 BDP)
  - e.g., C=10Gbps, RTT=15μs → BDP = 18.75KB
  - Today's switch buffers are 10-30x larger

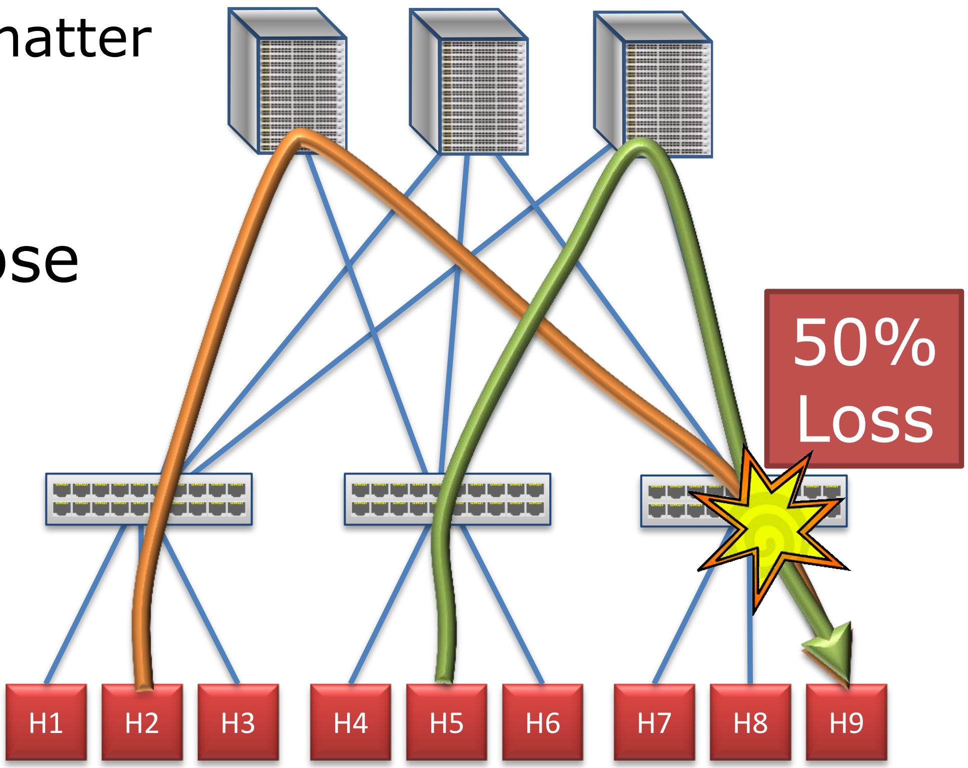
Priority Scheduling/Dropping Complexity

- Worst-case: Minimum size packets (64B)
  - 51.2ns to find min/max of ~300 numbers
  - Binary tree implementation takes 9 clock cycles
  - Current ASICs: clock = 1-2ns

pFabric Rate Control

- Priority scheduling & dropping in fabric also simplifies rate control
  - Queue backlog doesn't matter

One task:
Prevent congestion collapse when elephants collide



pFabric Rate Control

- Minimal version of TCP
  1. Start at line-rate
     - Initial window larger than BDP
  2. No retransmission timeout estimation
     - Fix RTO near round-trip time
  3. No fast retransmission on 3-dupacks
     - Allow packet reordering

Why does this work?

Key observation:

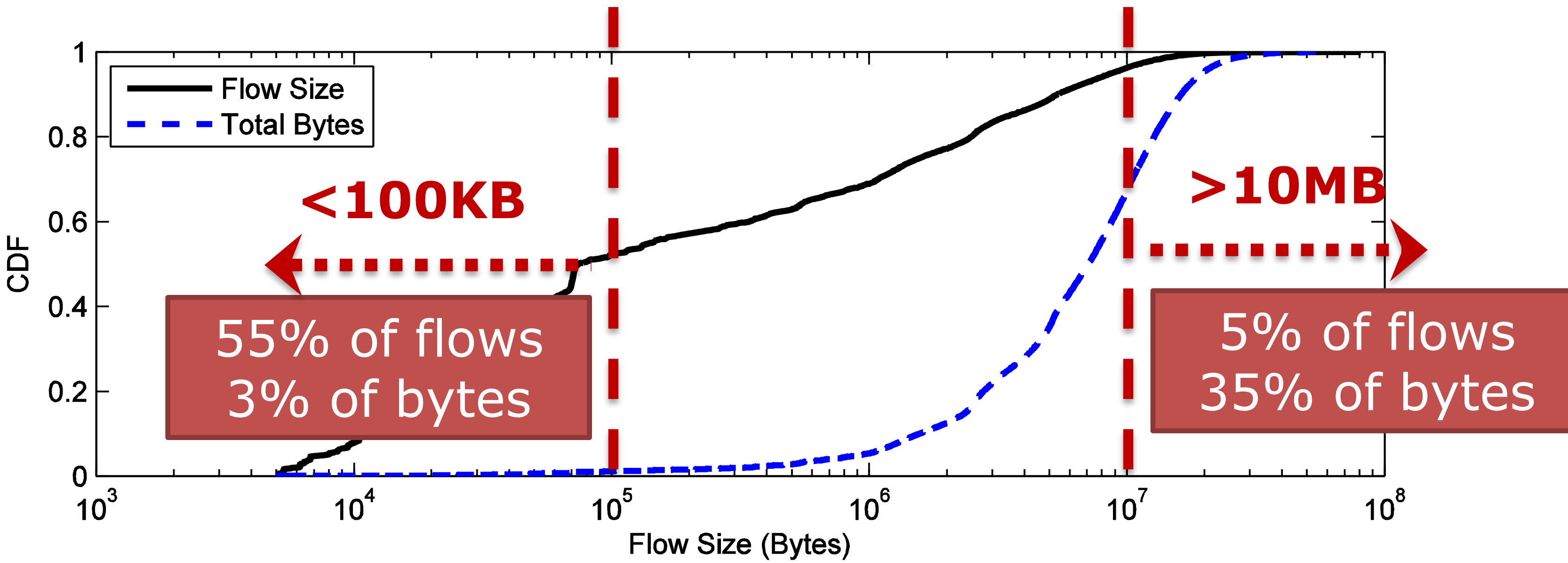
Need the highest priority packet destined for a port **available at the port** at any given time.

- **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 larger than BDP → more than RTT to retransmit

Evaluation

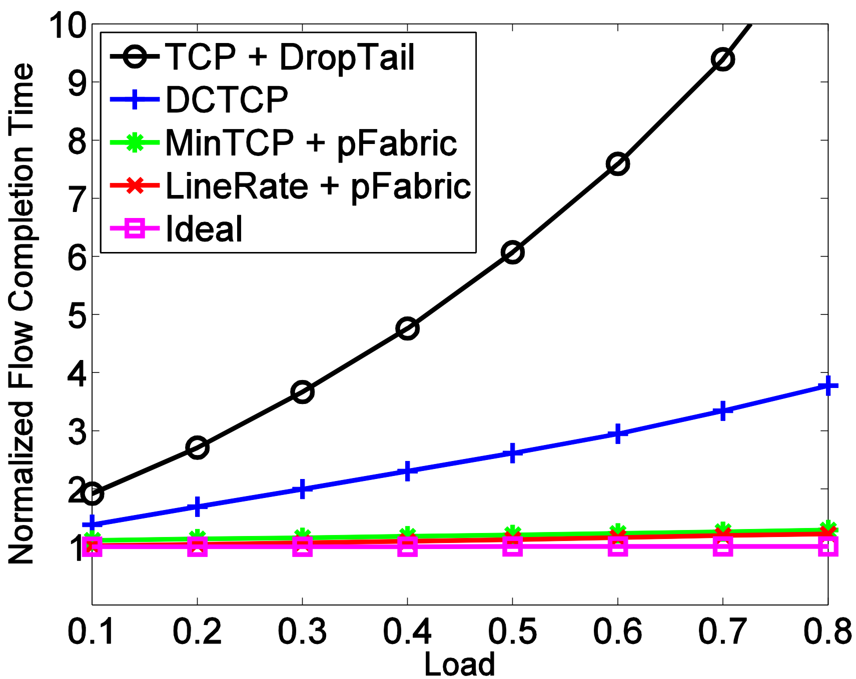
- 54 port fat-tree: 10Gbps links, RTT = ~12μs
- Realistic traffic workloads
  - Web search, Data mining

* From Alizadeh et al. [SIGCOMM 2010]

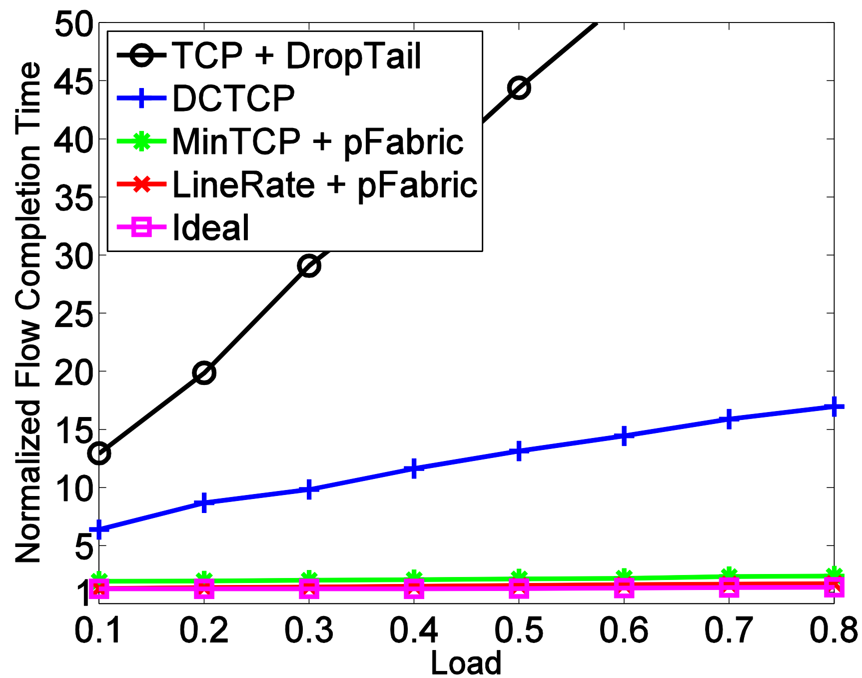


Evaluation: Mice FCT (<100KB)

Average



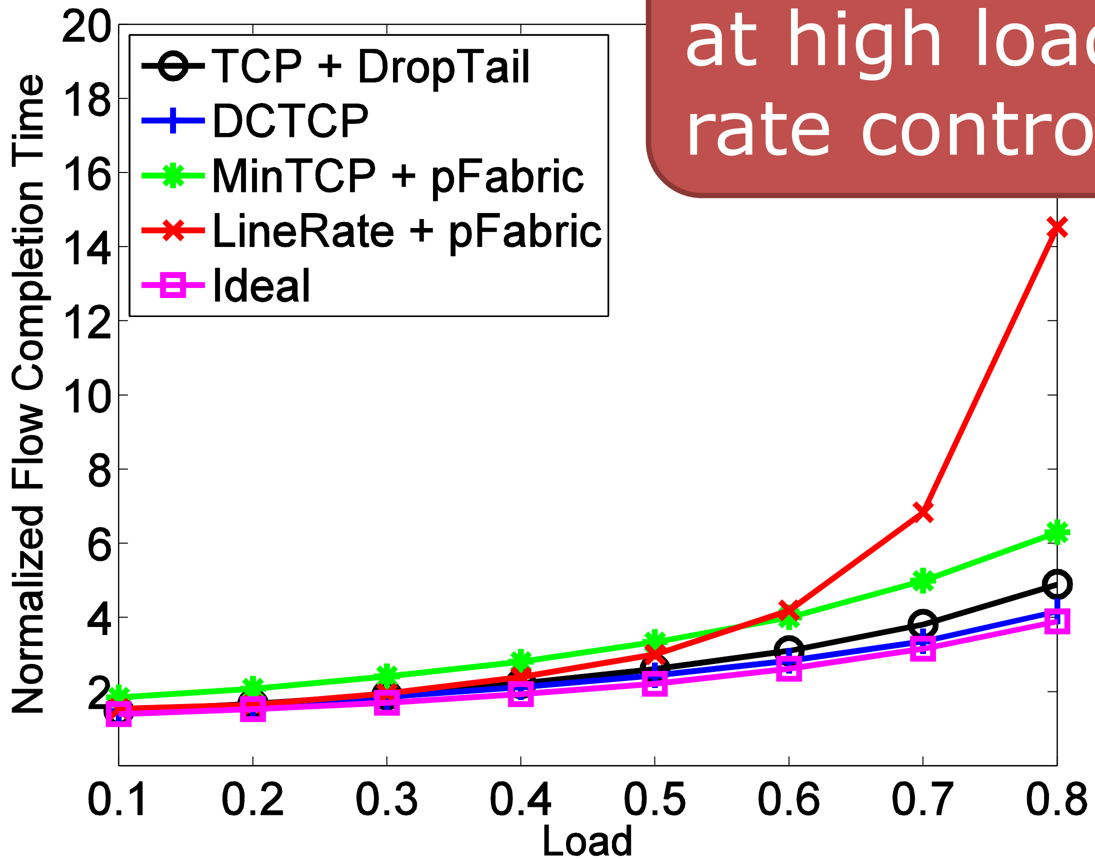
99^{th} Percentile



Near-ideal: almost no jitter

Evaluation: Elephant FCT (>10MB)

Congestion collapse at high load w/o rate control



Summary

pFabric's entire design:
Near-ideal flow scheduling across DC fabric

- **Switches**
  – Locally schedule & drop based on priority
- **Hosts**
  – Aggressively send & retransmit
  – Minimal rate control to avoid congestion collapse