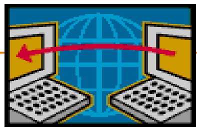


# Congestion Control

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

1

## Announcements

- Project 3 is out!

2

## A few words from Panda….

3

## Congestion Control Review

Did not have slides last time

Going to review key points

4

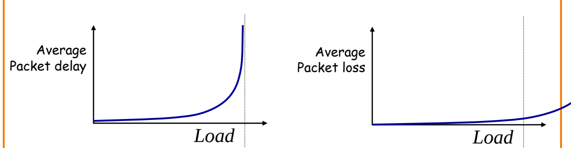
## Caveat: In this lecture

- Sometimes CWND is in units of MSS's
  - Because I want to count CWND in small integers
  - This is only for pedagogical purposes
- Sometimes CWND is in bytes
  - Because we actually are keeping track of real windows
  - This is how TCP code works
- Figure it out from context….

5

## Load and Delay

Typical queuing system with bursty arrivals




**Must balance utilization versus delay and loss**

6

## Not All Losses the Same

- Duplicate ACKs: isolated loss
  - Still getting ACKs
- Timeout: possible disaster
  - Not enough dupacks
  - Must have suffered several losses

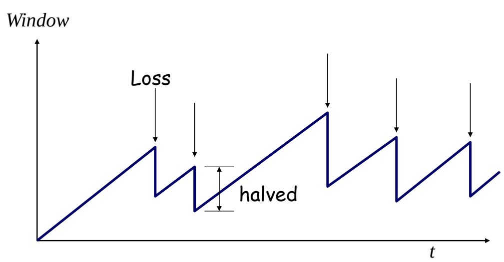
7

## AIMD

- Additive increase
  - On success of last window of data, increase by one MSS
- Multiplicative decrease
  - On loss of packet, divide congestion window in **half**

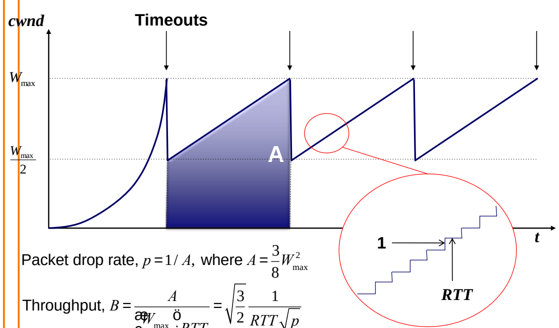
8

## Leads to the TCP "Sawtooth"




9

## Simple geometric analysis



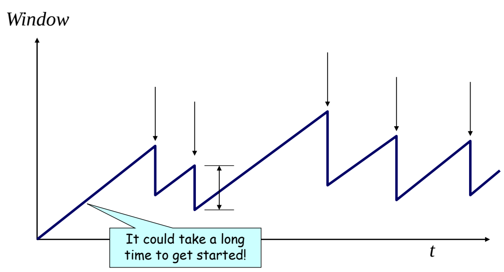

Packet drop rate, $p = 1/A$, where $A = \frac{3}{8} W_{max}^2$

Throughput, $B = \frac{A}{\left(\frac{W_{max}}{2}\right) RTT} = \sqrt{\frac{3}{2}} \frac{1}{RTT\sqrt{p}}$

10

## AIMD Starts Too Slowly!

Need to start with a small CWND to avoid overloading the network.




11

## "Slow-Start" Phase

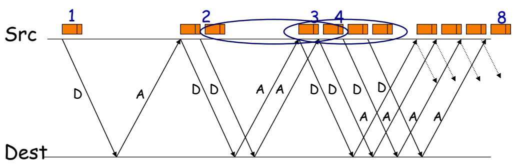
- Start with a small congestion window
  - Initially, CWND is 1 MSS
  - So, initial sending rate is MSS/RTT
- But want to increase quickly
  - Rather than just use additive increase….
  - ..we enter "slow-start" phase (actually "fast start")
- Sender starts at a slow rate (hence the name)
  - but increases exponentially until first loss

12

## Slow Start in Action

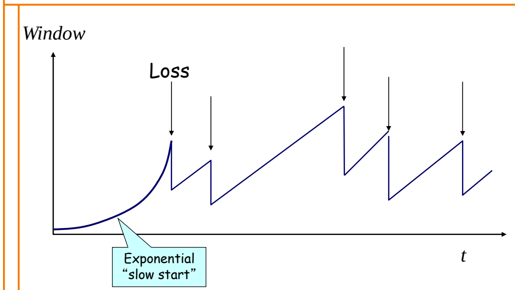
Double CWND per round-trip time

Simple implementation:
on each ack, CWND += MSS





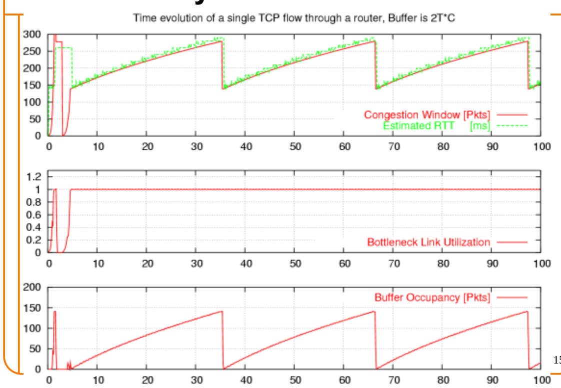

## Slow Start and the TCP Sawtooth




Why is it called slow-start? Because TCP originally had no congestion control mechanism. The source would just start by sending a whole window's worth of data.



## What is really looks like…






## Congestion Control Details



## Increasing CWND

- **Increase by MSS for every successful window**
- Increase a fraction of MSS per received ACK
- # packets (thus ACKs) per window: CWND / MSS
- Increment per ACK:

CWND += MSS / (CWND / MSS)

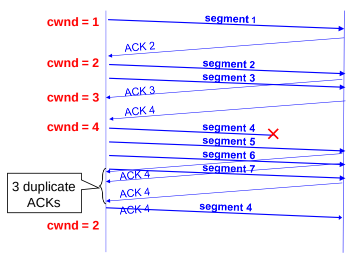
- Termed: Congestion Avoidance
  - Very gentle increase



## Fast Retransmission

- Sender sees 3 dupACKs
- Multiplicative decrease: CWND halved

## CWND with Fast Retransmit






## Loss Detected by Timeout

- Sender starts a timer that runs for RTO seconds
- **Restart timer whenever ack for new data arrives**
- If timer expires:
  - Set **SSTHRESH** ← CWND **/ 2** ("Slow-Start Threshold")
  - Set **CWND** ← MSS
  - Retransmit **first** lost packet
  - Execute Slow Start until **CWND** > **SSTHRESH**
  - After which switch to Additive Increase



## Summary of Decrease

- Cut CWND half on loss detected by dupacks
  - "**fast retransmit**"
- Cut CWND all the way to 1 MSS on **timeout**
  - Set ssthresh to cwnd/2
- Never drop CWND below 1 MSS

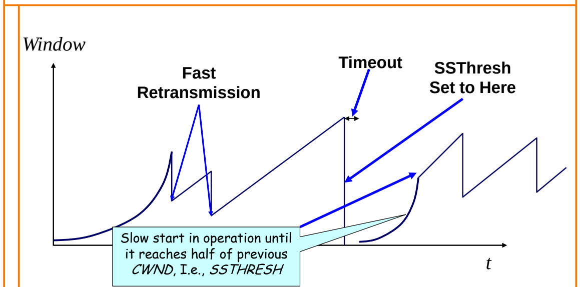


## Summary of Increase

- "Slow-start": increase cwnd by MSS for each ack
- Leave slow-start regime when either:
  - cwnd > SSThresh
  - Packet drop
- Enter AIMD regime
  - Increase by MSS for each window's worth of acked data



## Repeating Slow Start After Timeout




Slow-start restart: Go back to CWND of 1 MSS, but take advantage of knowing the previous value of CWND.



## More Advanced Fast Restart

- Set ssthresh to cwnd/2
- Set cwnd to cwnd/2 + 3
  - for the 3 dup acks already seen
- Increment cwnd by 1 MSS for each additional duplicate ACK
- After receiving new ACK, reset cwnd to ssthresh

## Example

- Consider a TCP connection with:
  - MSS=10bytes
  - ISN=100
  - CWND=100 bytes
  - Last ACK was for seq # 110
    - i.e., receiver expecting next packet to have seq. no. 110
- Packets with seq. no. 110 to 200 are in flight
  - What ACKs do they generate?
  - And how does the sender respond?



## History

- ACK 110 (due to 120) cwnd=100 dup#1
- ACK 110 (due to 130) cwnd=100 dup#2
- ACK 110 (due to 140) cwnd=100 dup#3
- RXMT 110 ssthresh=50 cwnd=80
- ACK 110 (due to 150) cwnd=90
- ACK 110 (due to 160) cwnd=100
- ACK 110 (due to 170) cwnd=110 xmit 210
- ACK 110 (due to 180) cwnd=120 xmit 220



## History (cont'd)

- ACK 110 (due to 190) cwnd=130 xmit 230
- ACK 110 (due to 200) cwnd=140 xmit 240
- ACK 210 (due to 110 rxmit) cwnd=ssthresh=50 xmit 250
- ACK 220 (due to 210) cwnd=60
- …..



**Why AIMD?**

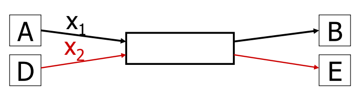


## Four alternatives

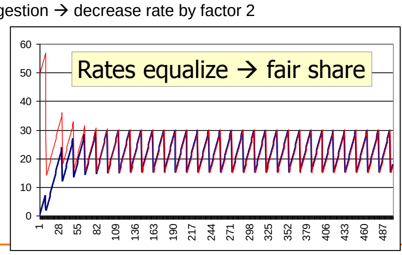
- AIAD: gentle increase, gentle decrease
- AIMD: gentle increase, drastic decrease
- MIAD: drastic increase, gentle decrease
  - too many losses: eliminate
- MIMD: drastic increase and decrease



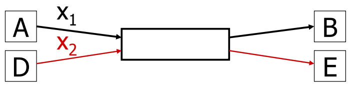
## AIMD Sharing Dynamics



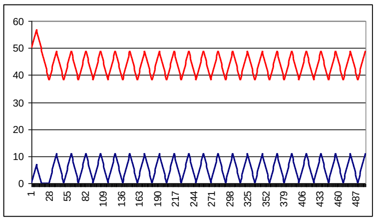

- No congestion → rate increases by one packet/RTT every RTT
- Congestion → decrease rate by factor 2



## AIAD Sharing Dynamics




- No congestion → x increases by one packet/RTT every RTT
- Congestion → decrease x by 1






## Other Congestion Control Topics



## 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 really)
- Alternative: Random Early Drop (LBL)
  - Drop packets on purpose before queue is full
  - Set drop probability D as a function of queue size
  - Keep queue average small, but tolerate bursts



## What if loss isn't congestion-related?

- Can use Explicit Congestion Notification (ECN)
- Bit in IP packet header, that is carried up to TCP
- When RED router would drop, it sets bit instead
  - Congestion semantics of bit exactly like that of drop
- Advantages:
  - Don't confuse corruption with congestion
  - Don't confuse recovery with rate adjustment



## How does this 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 $2 \times 10^{-12}$
- 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: once speed is past some threshold, change equation to $p^{-.8}$ rather than $p^{-.5}$
  - Let the additive constant in AIMD depend on CWND
  - At very high speeds, increase CWND by more than MSS
- We will discuss other approaches next later…

## How "Fair" is TCP?

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

37

## 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…

38

## Router-Assisted Congestion Control

- There are two different tasks:
  - Isolation/fairness
  - Adjustment
- Isolation/fairness:
  - We would like to make sure each flow gets its "fair share"
  - This protects flows from cheaters
    - ***Safety/Security issue***
  - No longer requires everyone use same CC algorithm
    - ***Innovation issue***
- Adjustment:

39

## 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?

40

## Max-Min Fairness

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

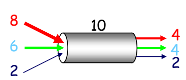
$$a_i = \min(f, r_i)$$

- where f is the unique value such that $\mathrm{Sum}(a_i) = C$
- 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

41

## 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

42

## 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)

43

## With FQ 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 complication on router
  - Cheating not a problem, so there's little motivation
  - But WFQ is used at larger granularities

44

## FQ is really "processor sharing"

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

45

## 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

46

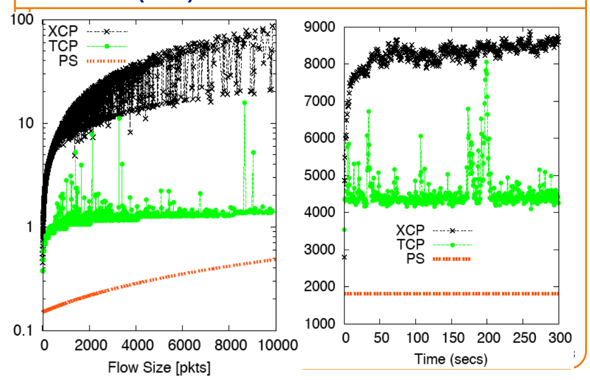
## Fair Sharing is more than a moral issue

- 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….

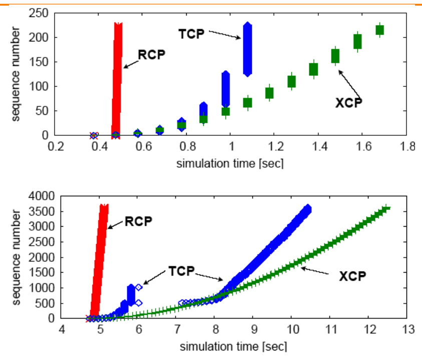
47

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

Flow Duration (secs) vs. Flow Size   # Active Flows vs. time



## Why the improvement?




## 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 shouldn't you be penalized for using more scarce bandwidth?
- And what is a flow anyway?
  - TCP connection
  - Source-Destination pair?
  - Source?



## Charge people for congestion!

- Use ECN as congestion markers
- Whenever I get ECN bit set, I have to pay $$$
- Now, there's 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
  - 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

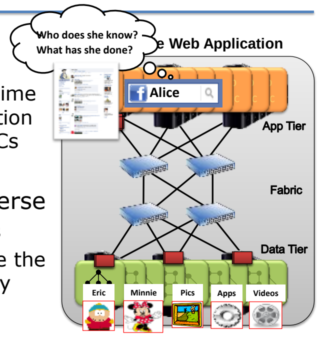
**Mohammad Alizadeh**, Shuang Yang, Sachin Katti, Nick McKeown, Balaji Prabhakar, Scott Shenker

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$^2$TCP[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**

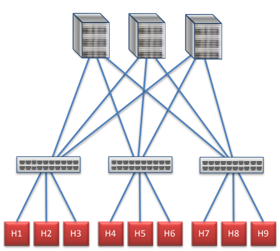
- Very small buffers (e.g., 10-20KB)
- Send highest priority / drop lowest priority pkts

**pFabric Hosts**

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

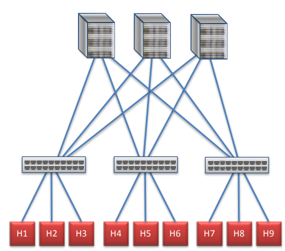


## DC Fabric: Just a Giant Switch!

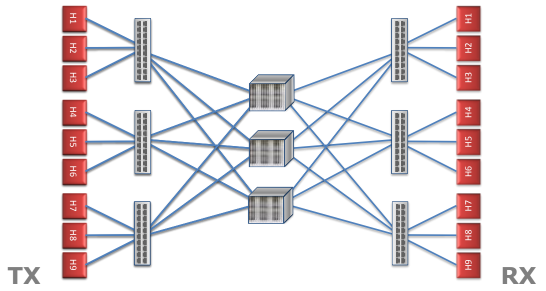





## DC Fabric: Just a Giant Switch!

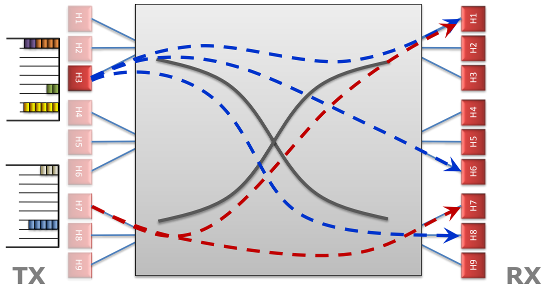


## DC Fabric: Just a Giant Switch!





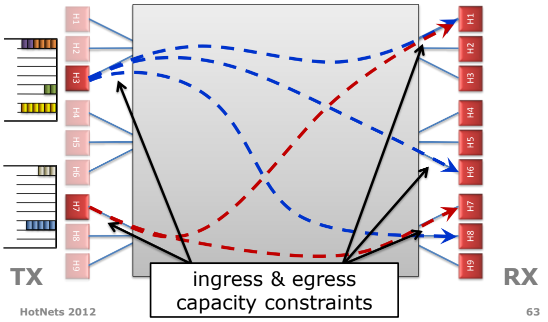

## DC Fabric: Just a Giant Switch!






DC transport = Flow scheduling on giant switch

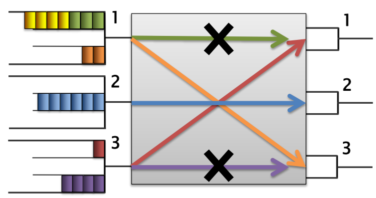
Objective?
- Minimize avg FCT






## "Ideal" Flow Scheduling

Problem is NP-hard ☹ [Bar-Noy et al.]
- Simple greedy algorithm: **2-approximation**



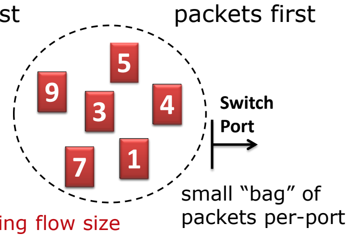



# pFabric Design



## pFabric Switch

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




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**

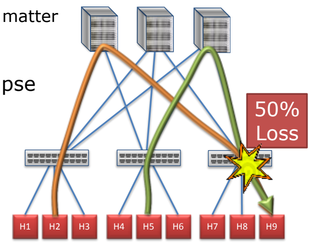
- Worst-case: Minimum size packets (64B)
  - 51.2ns to find min/max of ~300 numbers
  - Binary tree implementation takes 9 clock cycles
  - Current ASICs: clock = 1-2ns



# 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.

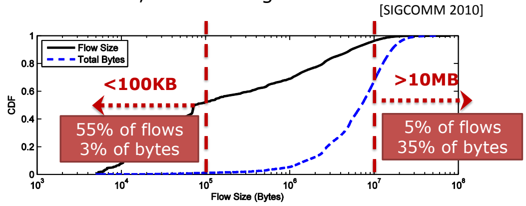
- **Priority scheduling**
  - ➢ High priority packets traverse fabric as quickly as possible
- **What about dropped packets?**
  - ➢ Lowest priority → not needed till all other packets depart
  - ➢ Buffer larger than BDP → more than RTT to retransmit



# Evaluation

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

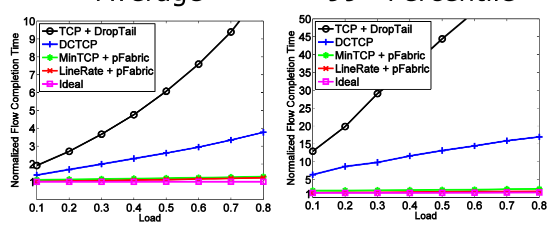
* From Alizadeh et al. [SIGCOMM 2010]





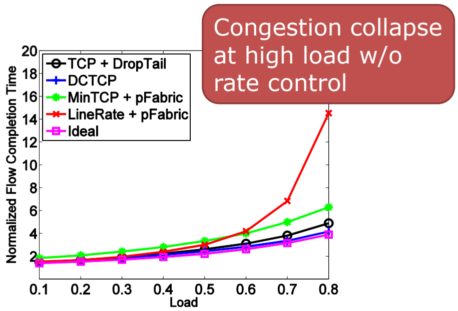

# Evaluation: Mice FCT (<100KB)

Average | 99th Percentile




Near-ideal: almost no jitter

## Evaluation: Elephant FCT (>10MB)






## 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