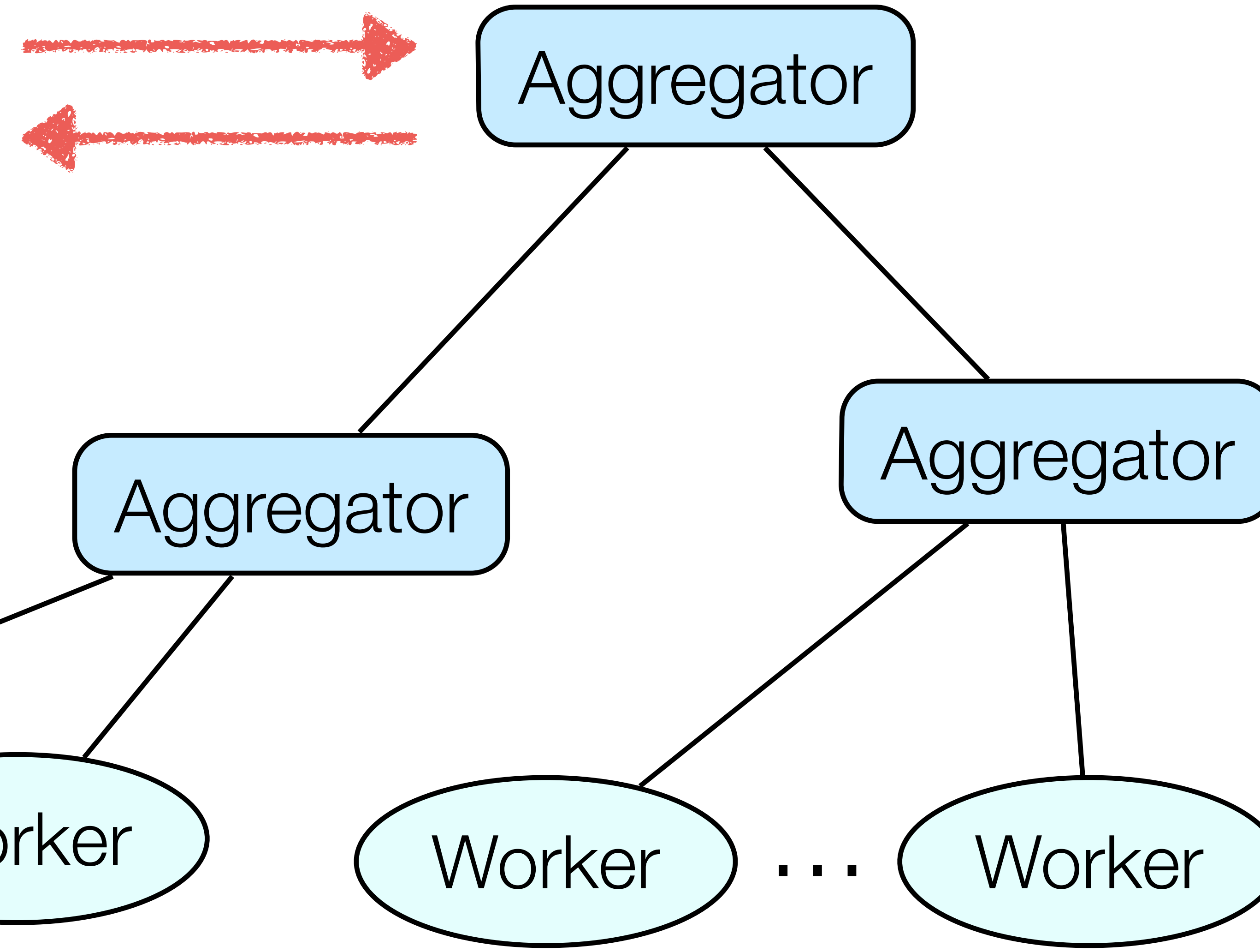
Load Balancing in Data Center Networks

Henry Xu

Computer Science
City University of Hong Kong

HKUST, March 2, 2015

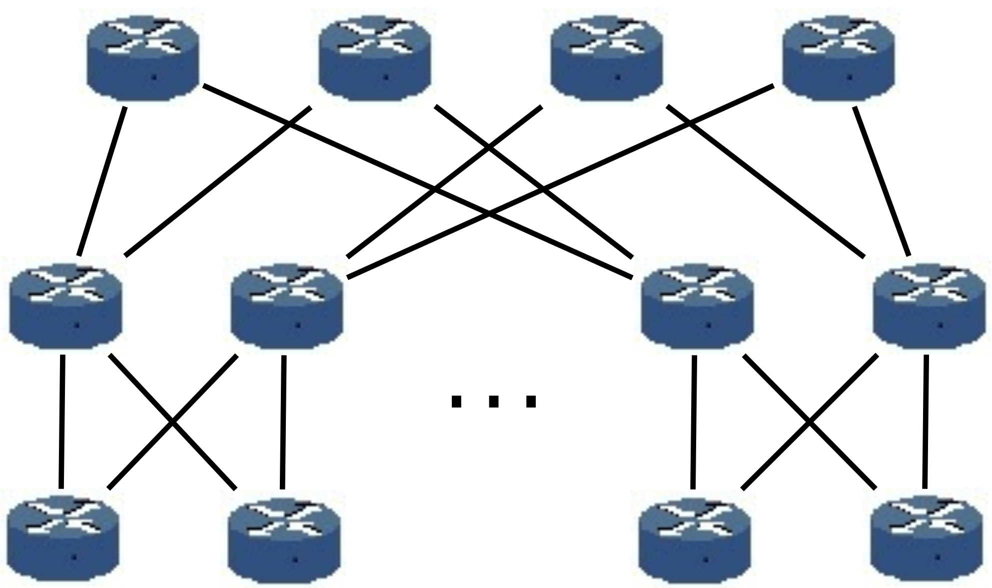
Background



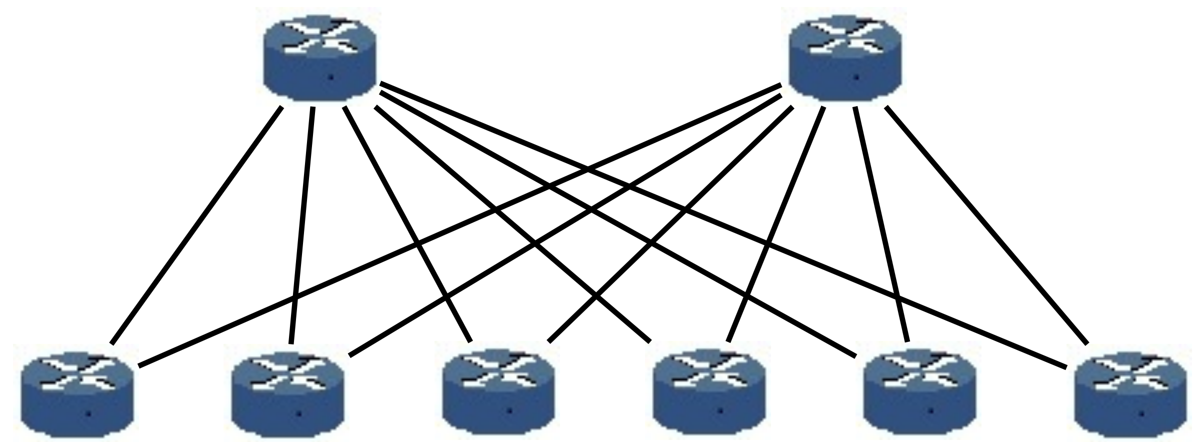
Low latency for a huge number of short query/response flows, especially the tail latency (e.g. 99-th)

Background

Data centers use multi-stage Clos topologies



Fat-tree



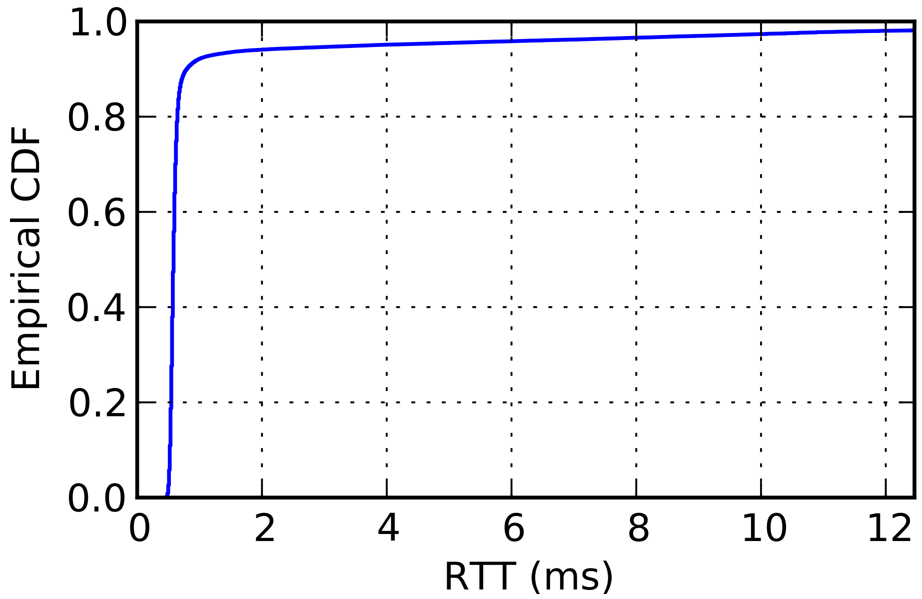
Spine-Leaf

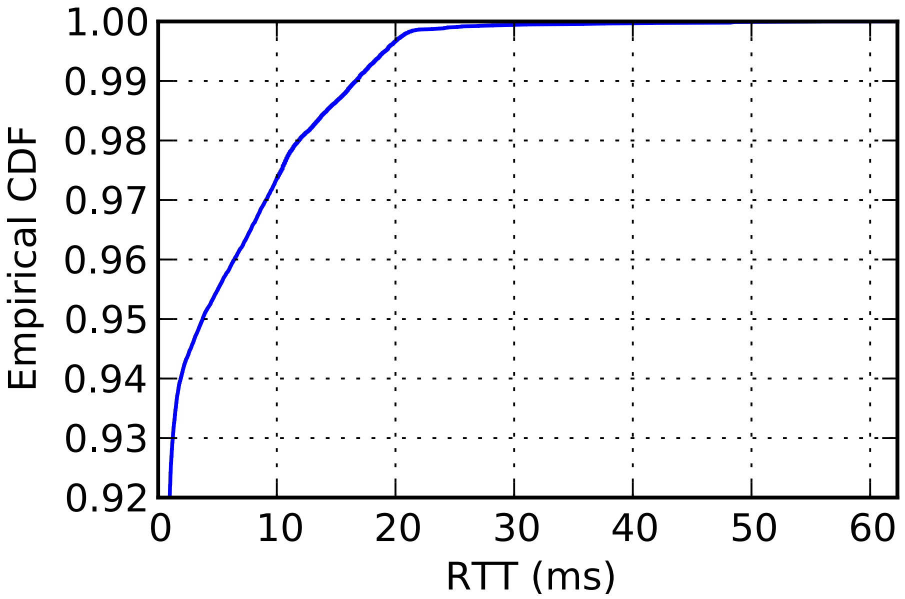
- Multiple equal-cost paths for a pair of hosts – How to load balance?
- Today's practice: ECMP, local, random – lots of problems

Background

- Current data center transport is ill-fitted for the task

RTT measurement in EC2 us-west-2c, 100K samples





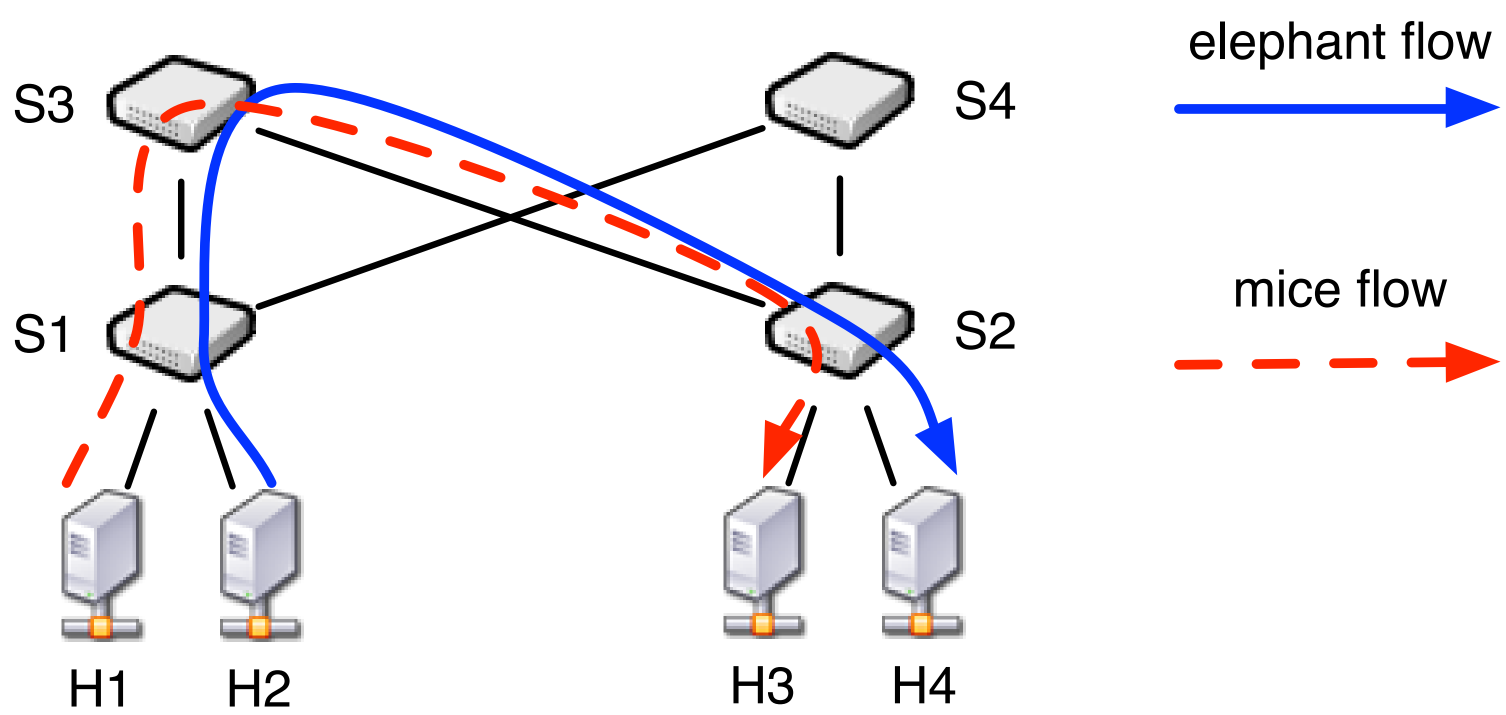
Mean RTT: 0.5ms

99-th RTT: 17ms

- Corroborated by measurements from many existing papers

Background

- Culprit: ECMP is static and agnostic to congestion



- Tail latency is even worse with elephants colliding on the same path due to ECMP

Our quest

- How can we improve load balancing in data center networks?
  - Scalable enough to handle millions of mice flows traversing numerous links
  - Smart enough to avoid congestion in the network dynamically

Our answer

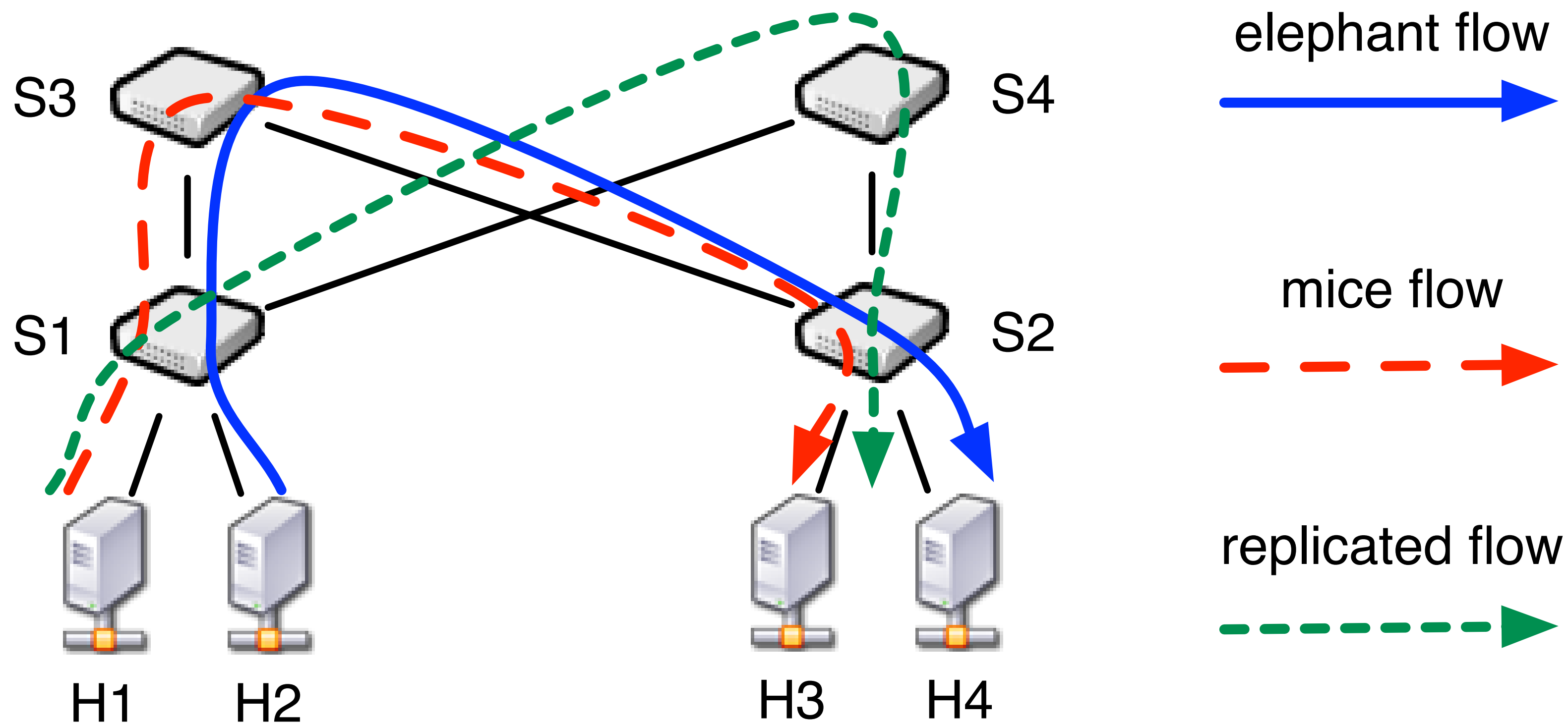
- Patch solution: RepNet
  - Application-layer transport that can be implemented today
  - INFOCOM 2014, under submission
- Fundamental solution: Expeditus
  - Distributed, congestion-aware load balancing protocol to replace ECMP
  - CoNEXT student workshop 2014 best paper, on-going work

Chapter I

RepNet

RepNet in a nutshell

- Replicate each mice flow to exploit multipath diversity



- No two paths are exactly the same – The power of two choices, M Mitzenmacher

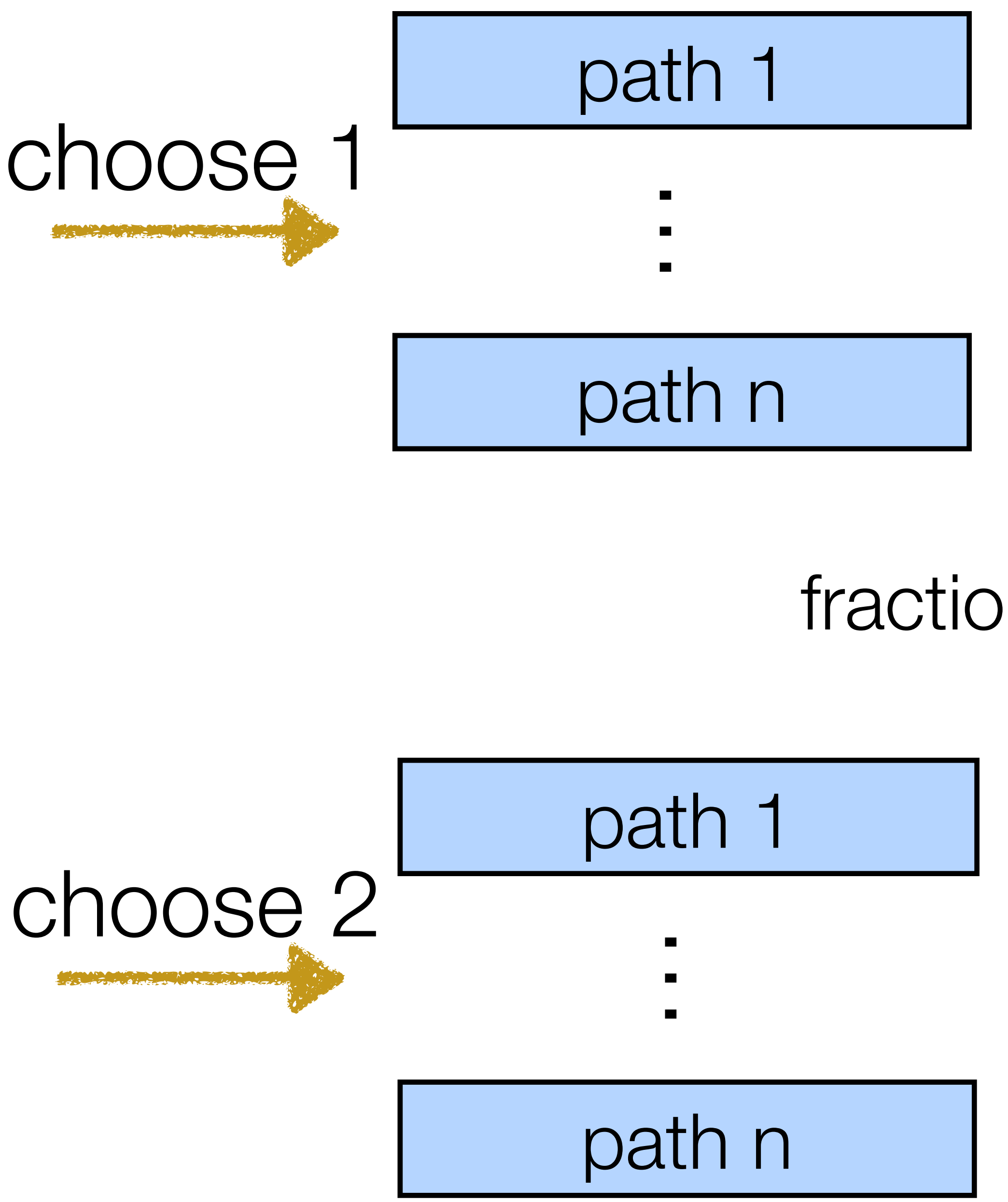
- Clos based topologies provide many equal-cost paths

RepNet’s design

- Which flows?
  - Less than 100KB, consistent with many existing papers
- When?
  - Always! (We’ll come back about the overhead issue)
- How?
  - RepFlow: replicate each byte of the flow
  - RepSYN: only replicate SYN packets and choose the quicker connection

Is RepNet effective?

Simplified queueing analysis



ρ

ρ

effective load: ρ

fraction of total bytes from mice (< 0.1)

$(1 + \epsilon)\rho$

$(1 + \epsilon)\rho$

effective load: $(1 + \epsilon)^2 \rho^2$

Packet-level NS-3 simulations

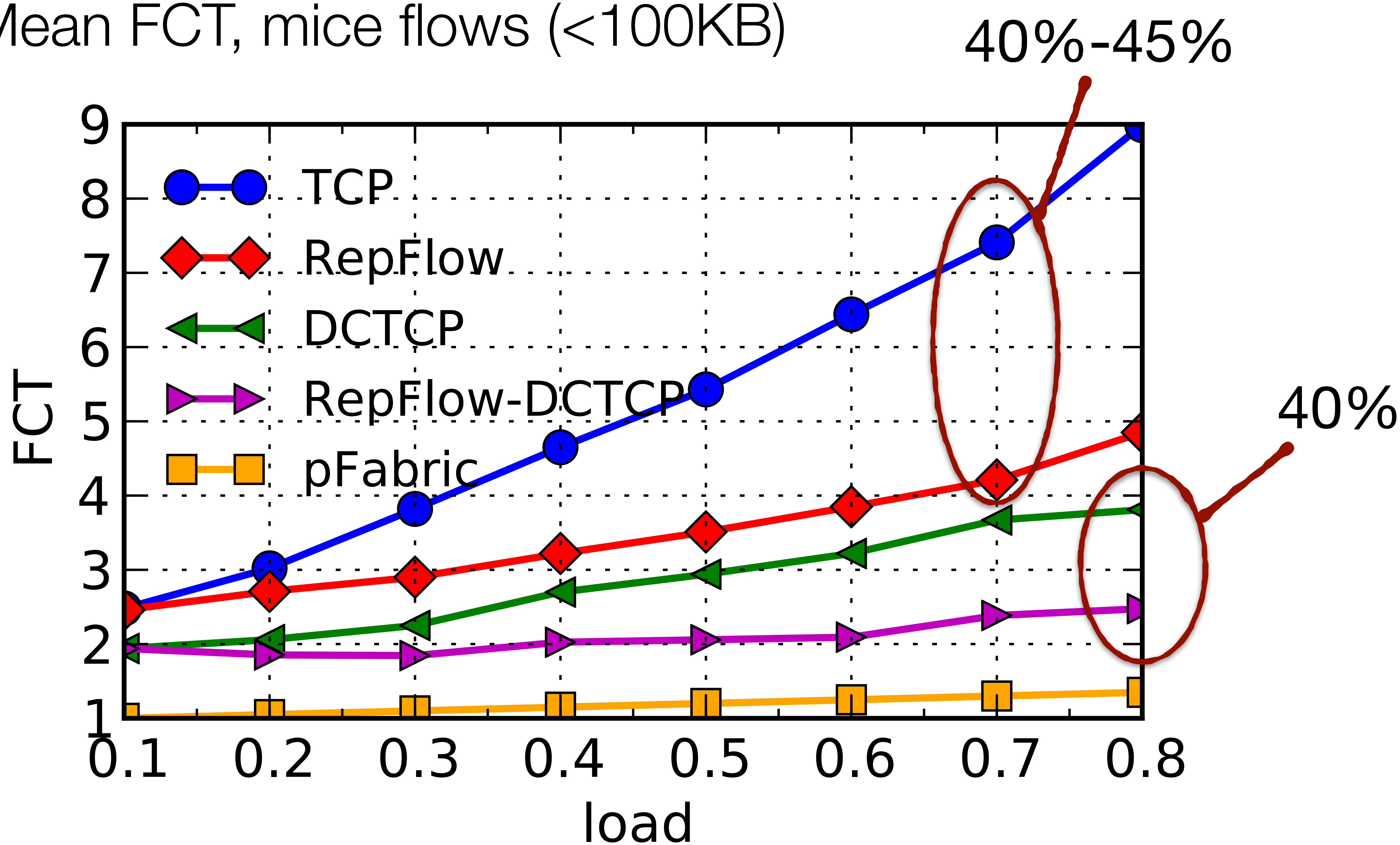
- Topology: 16-pod 1Gbps fat-tree, 1,024 hosts
- Traffic pattern: Poisson, random src/dst, 0.5s worth
- Flow size distribution:
  - Web search cluster from DCTCP paper
    - >95% bytes are from 30% flows large than 1MB
  - Data mining cluster from VL2 paper (not shown here)
    - >95% bytes are from 3.6% flows large than 35MB

Benchmarks

- TCP: TCP NewReno, initial window 12KB, DropTail queues with 100 packet buffer
- RepFlow
- DCTCP: source code from authors of D2TCP
- RepFlow-DCTCP: RepFlow on top of DCTCP
- pFabric: state-of-the-art, near-optimal FCT with priority queueing, source code obtained from authors

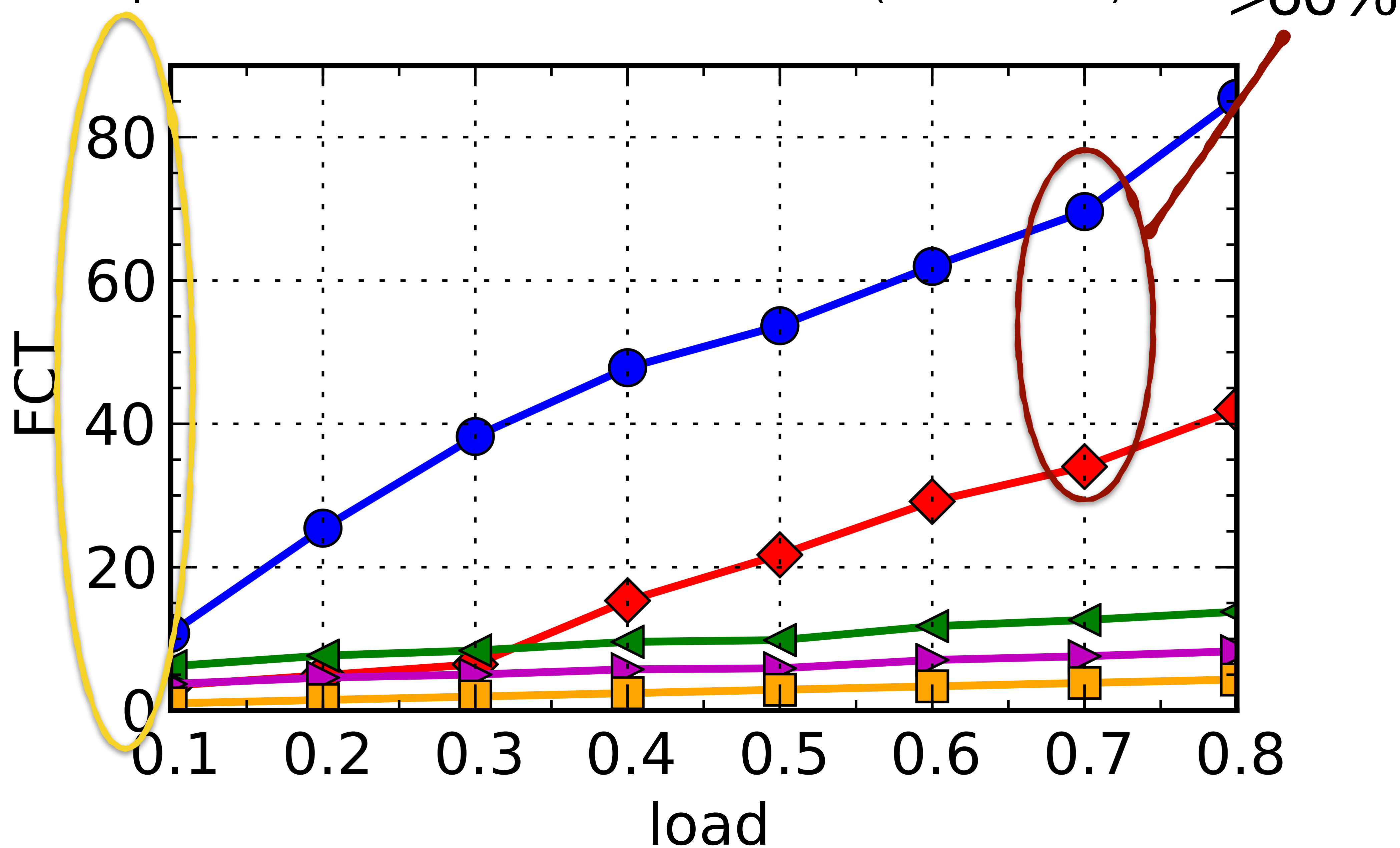
Results [1/4]

- Mean FCT, mice flows (<100KB)



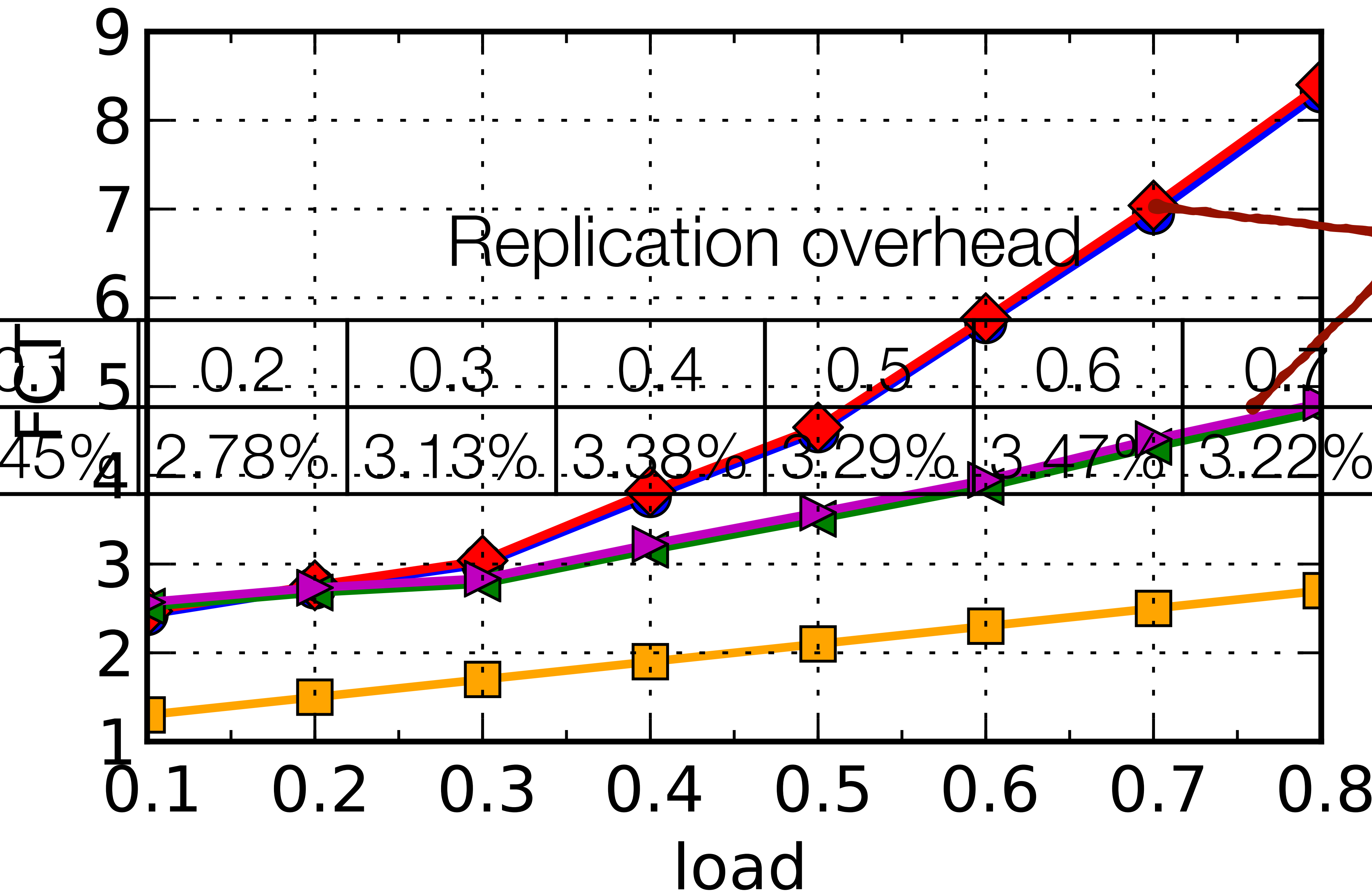
Results [2/4]

- 99-th percentile FCT, mice flows (<100KB)



Results [4/4]

- Mean FCT, elephant flows (>=100KB)



Replication overhead

0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8
3.45%	2.78%	3.13%	3.38%	3.29%	3.47%	3.22%	3.27%

no impact

Is RepNet *really* effective?

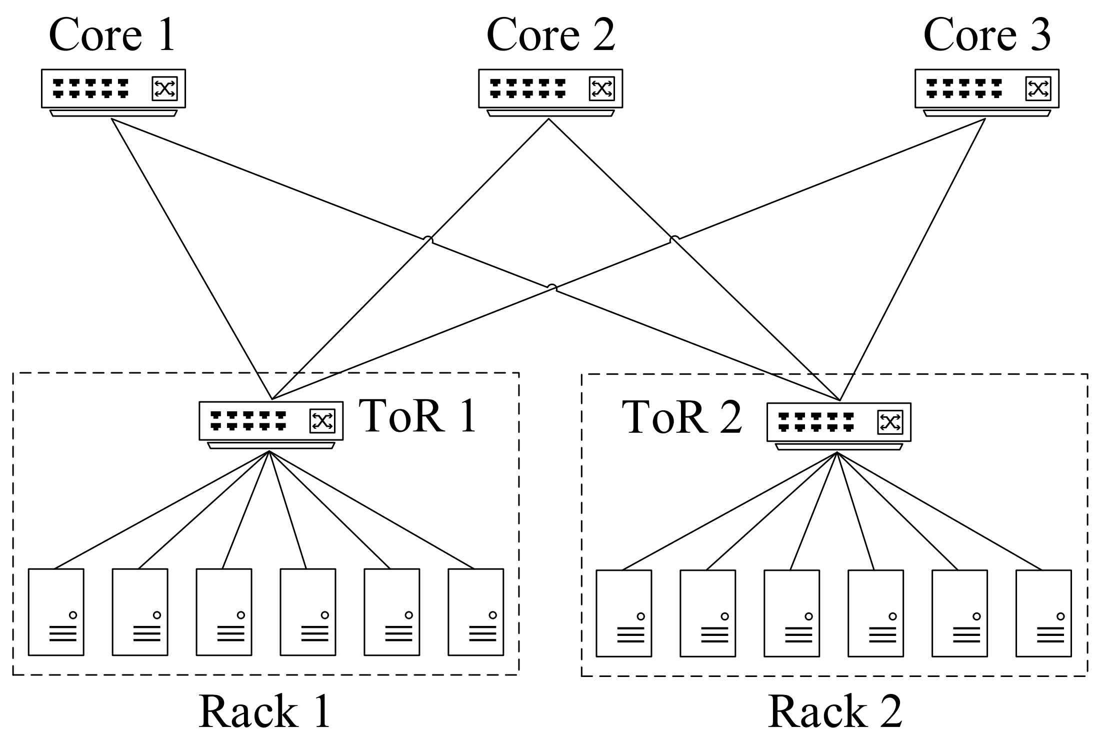
Implementation

- Based on **node**, a highly scalable platform for real-time server-side networked applications
  - Single-threaded, non-blocking socket, event driven
  - Widely used in industry for both front-end and back-end

Implementation

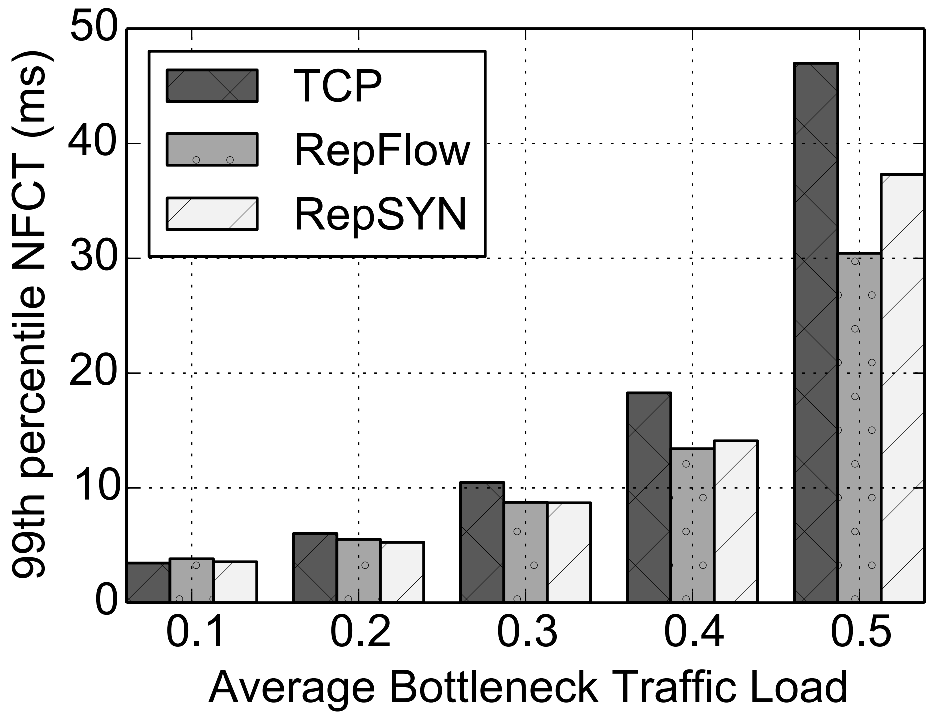
- A module `RepNet` based on `Net`, the standard library for non-blocking sockets
- Applications only need to change one line:
  - `require(‘net’) -> require(‘repnet’)`
- `RepNet.Socket`: a single socket abstraction for applications while having two TCP sockets
- `RepNet.Server`: functions for listening for and managing both replicated and regular TCP connections

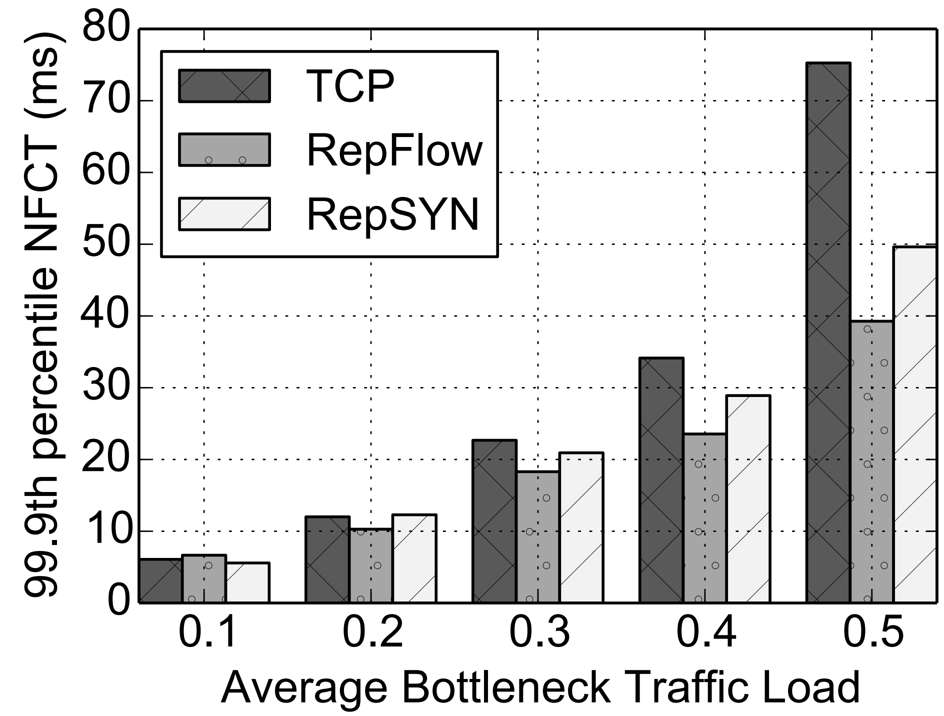
Testbed evaluation



- Pronto 3295 switches. 1Gbps links. Oversubscribed at 2:1
- Ping RTT 178us across racks
- Flow size distribution from the DCTCP paper

Testbed evaluation





- RepFlow and RepSYN significantly improve tail FCT when load is high in an oversubscribed network
- RepFlow is more beneficial

More results

- Application level performance using RepNet
- Mininet emulation with a 6-pod fat-tree
- All source code and experiment scripts are online
    - https://bitbucket.org/shuhaoliu/repnet
    - https://bitbucket.org/shuhaoliu/repnet_experiment

Recap

- Takeaway: *RepNet is a practical and effective application layer low latency transport*
- Open-source implementation and experimental evaluation
- Patch solution, short-term

Chapter II

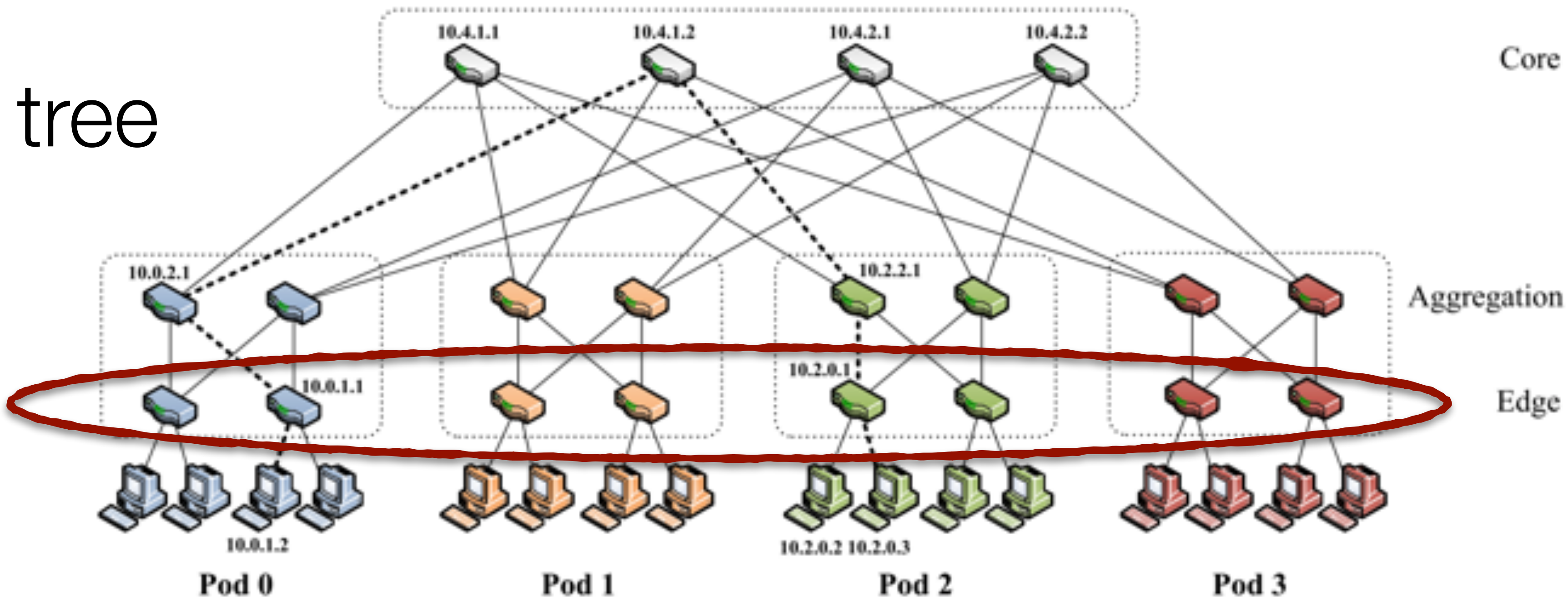
Expeditus

How to build a distributed congestion-aware load balancing protocol, for a large-scale data center network?

- Naive solution: track congestion information for all possible paths
- This simply can't scale

Per-path isn't scalable

k-pod fat tree



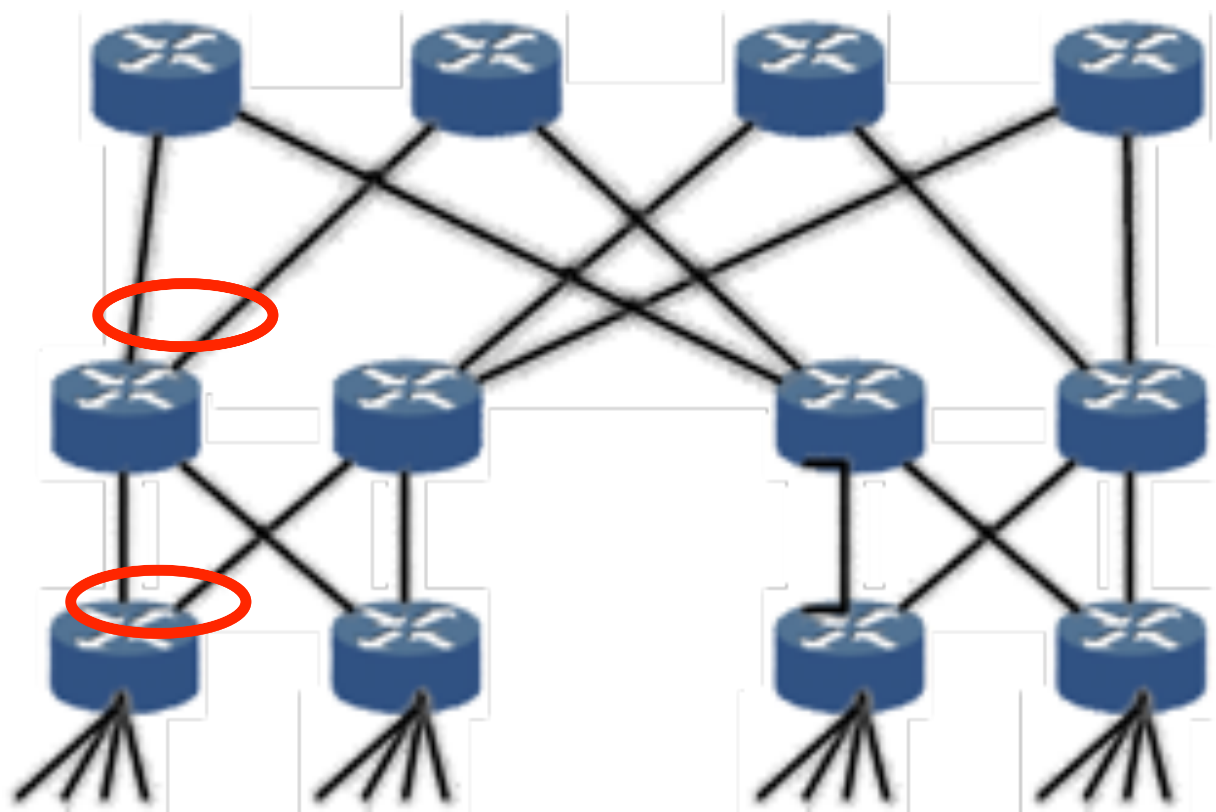
- $k^2/4$ paths between edge switches of distinct pods
- An edge switch talks to $k^2/2-k/2$ edge switches in distinct pods
- Each edge switch needs to track $O(k^4)$ paths!

Design

- One-hop congestion information collection
  - Each edge and aggr switch maintains congestion information for k ports in k-pod fat-tree
- Two-stage path selection

One-hop info collection

- Northbound congestion information can be obtained by polling buffer occupancy of egress ports

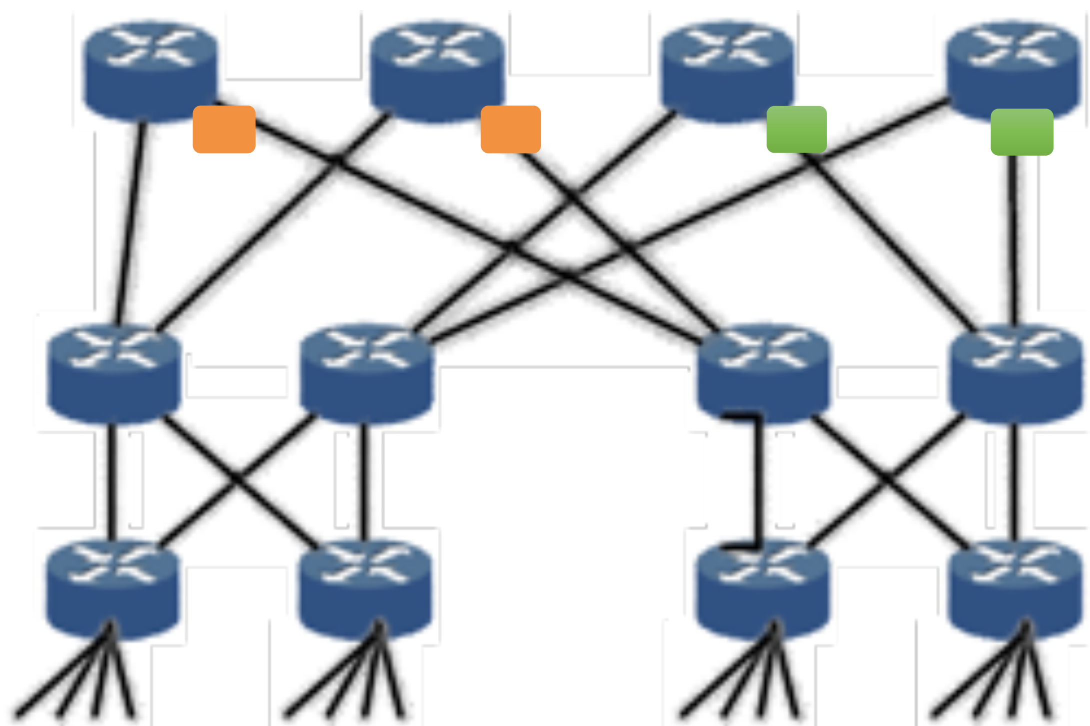


One-hop info collection

- Southbound congestion information needs to be transmitted by piggybacking in packets

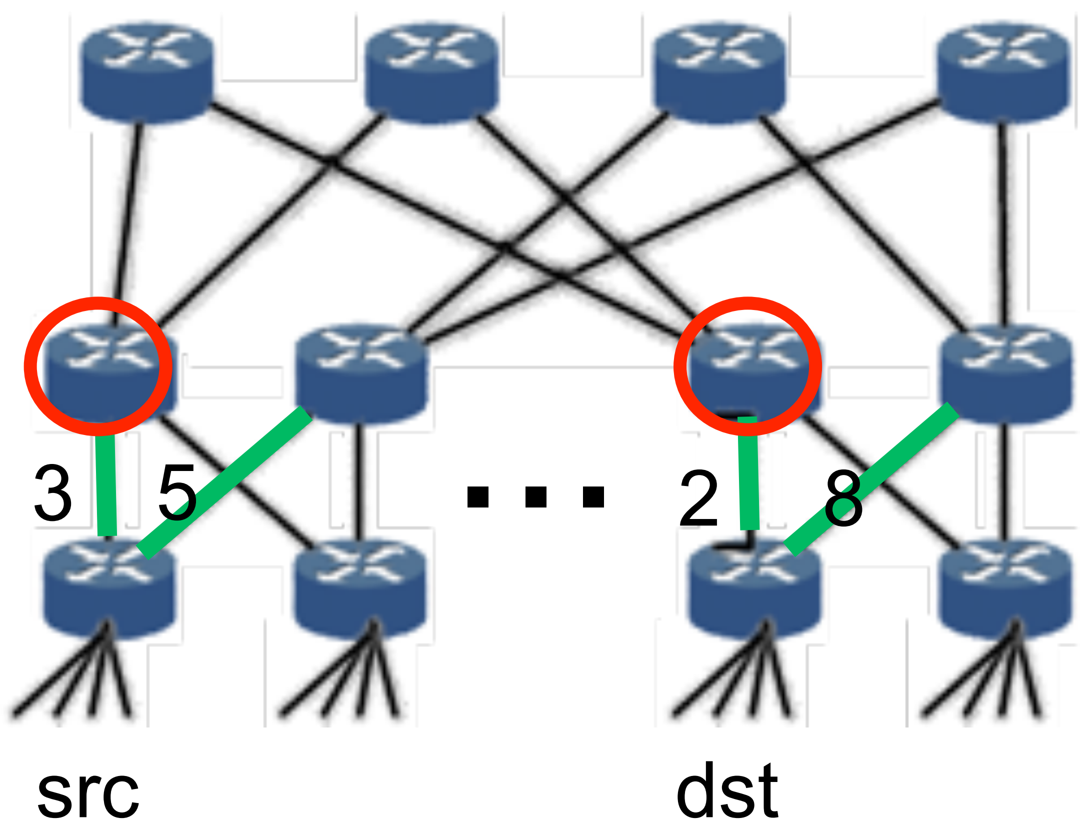
Aggr switches collect congestion information coming from core switches

Edge switches collect congestion information coming from aggr switches



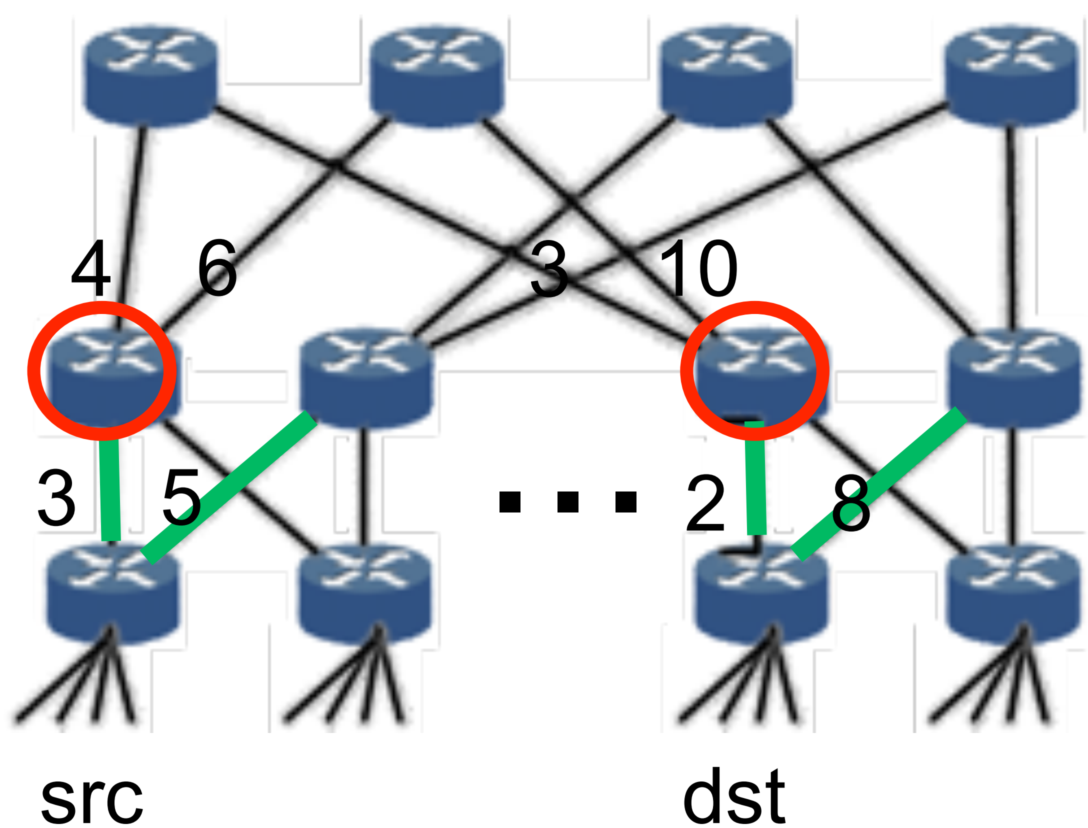
Two-stage path selection

- SYN packet carries congestion information at source edge switch to destination edge switch
- Destination edge switch chooses the aggr switch with the least combined congestion at the first and last hop



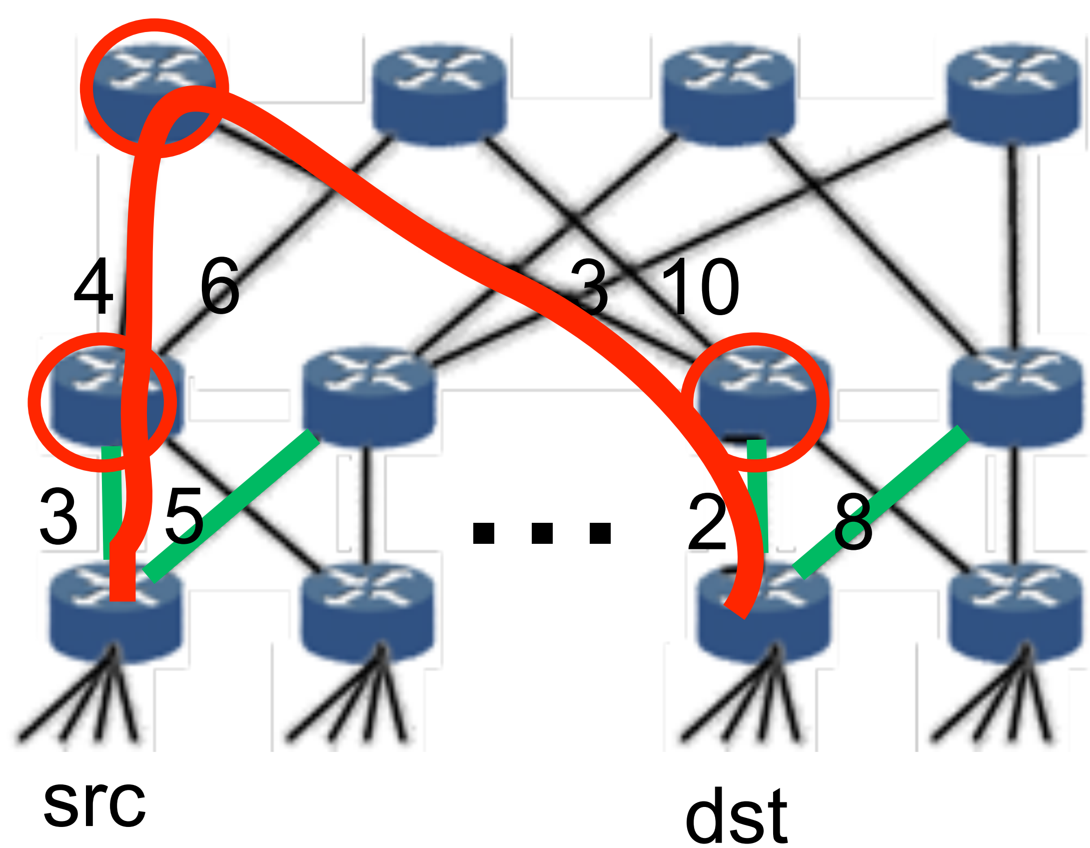
Two-stage path selection

- SYN-ACK packet carries congestion information at destination aggr switch to source aggr switch
- Source aggr switch chooses the core switch with the least combined congestion at the second and third hops



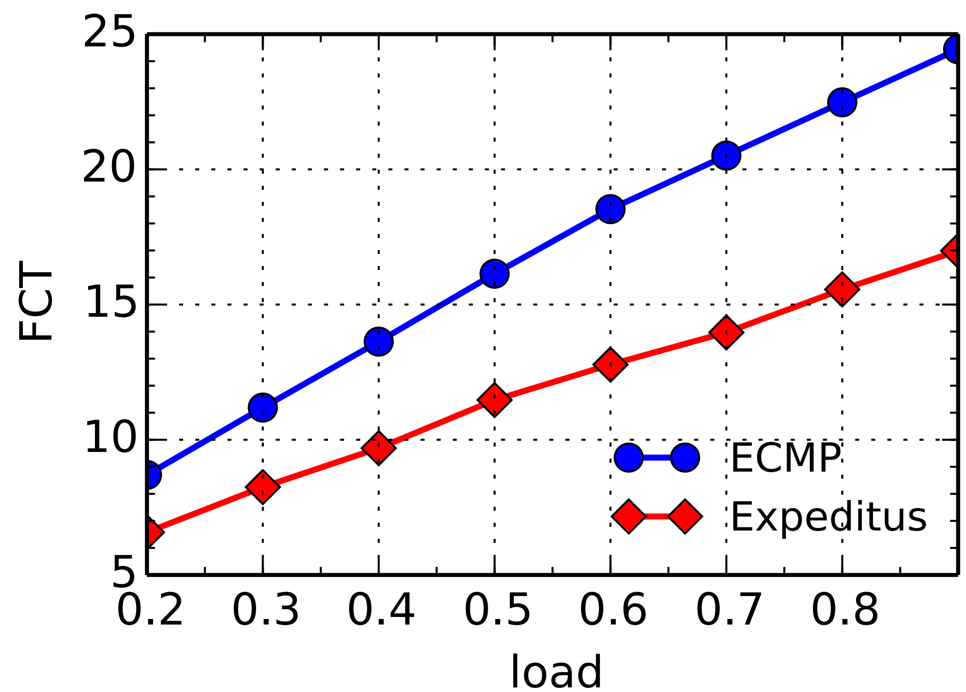
Two-stage path selection

- Assemble a complete path based on selected aggr and core switches, store in host's flow routing table
- IP-in-IP encapsulation to enforce source routing

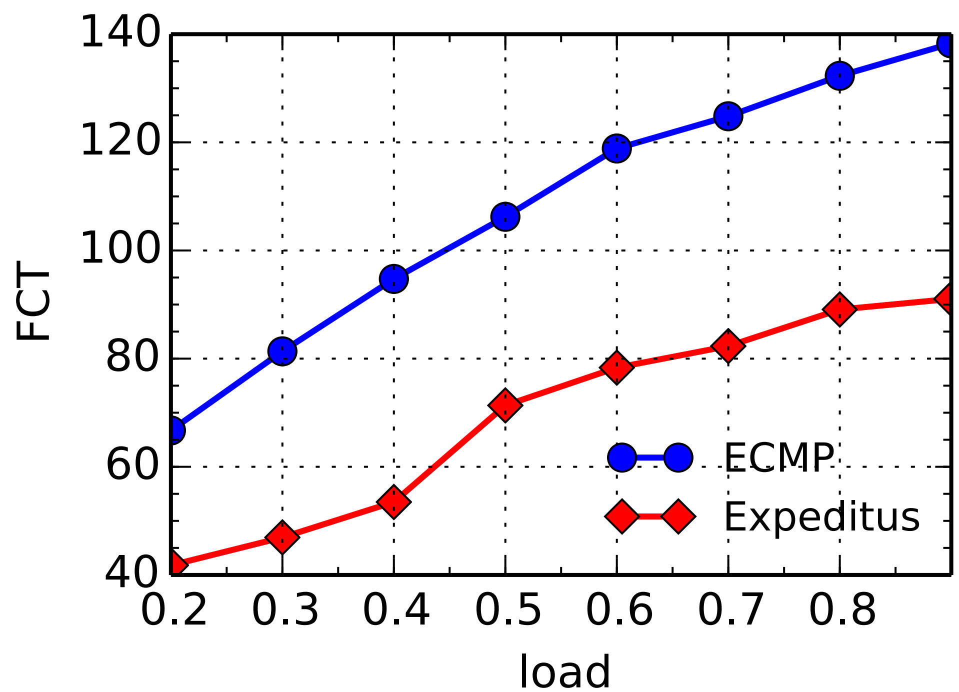


Preliminary evaluation

- NS-3 simulation with a 16-pod fat-tree (1,024 hosts), oversubscribed at 4:1, DCTCP flow size distribution



mean FCT



99th percentile FCT

Implementation–on-going

- Click software router implementation (together with CONGA)
- Experiments on a fat-tree on Emulab
  - 20 PCs with 5 NICs as Expeditus switches
  - 16 PCs with 1 NICs as hosts
  - https://www.emulab.net/showproject.php3?pid=expeditus

Related work

- Reducing (tail) latency in data center networks is an important problem
  - Reduce queue length: *DCTCP (2010), HULL (2012)*
  - Prioritize scheduling: *D^3 (2011), PDQ (2012), DeTail (2012), pFabric (2013)*
- They all require **modifications to end-hosts and/or switches**, making it difficult to deploy in reality

Thank you!

Henry Xu
City University of Hong Kong