# The TCP Outcast Problem

## Exposing Unfairness in Data Center Networks

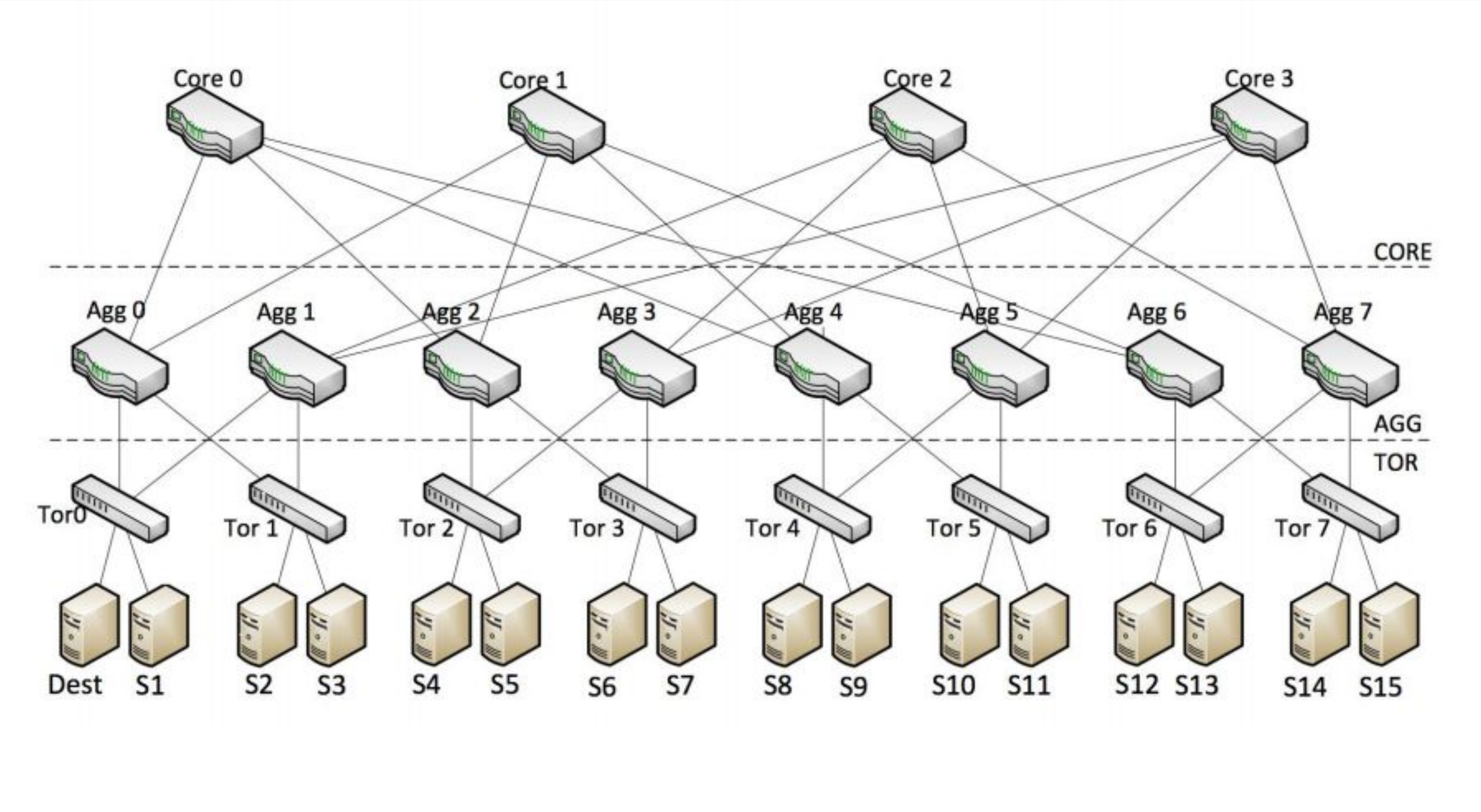
# Data Center is a Shared Environment

- Enterprises host applications on data centers
    - Cost cutting, high reliability, increased efficiency
- Data center is a shared environment
    - Multiple applications
    - Applications belonging to multiple tenants
- State of resource sharing
    - CPU/memory usually strictly sliced

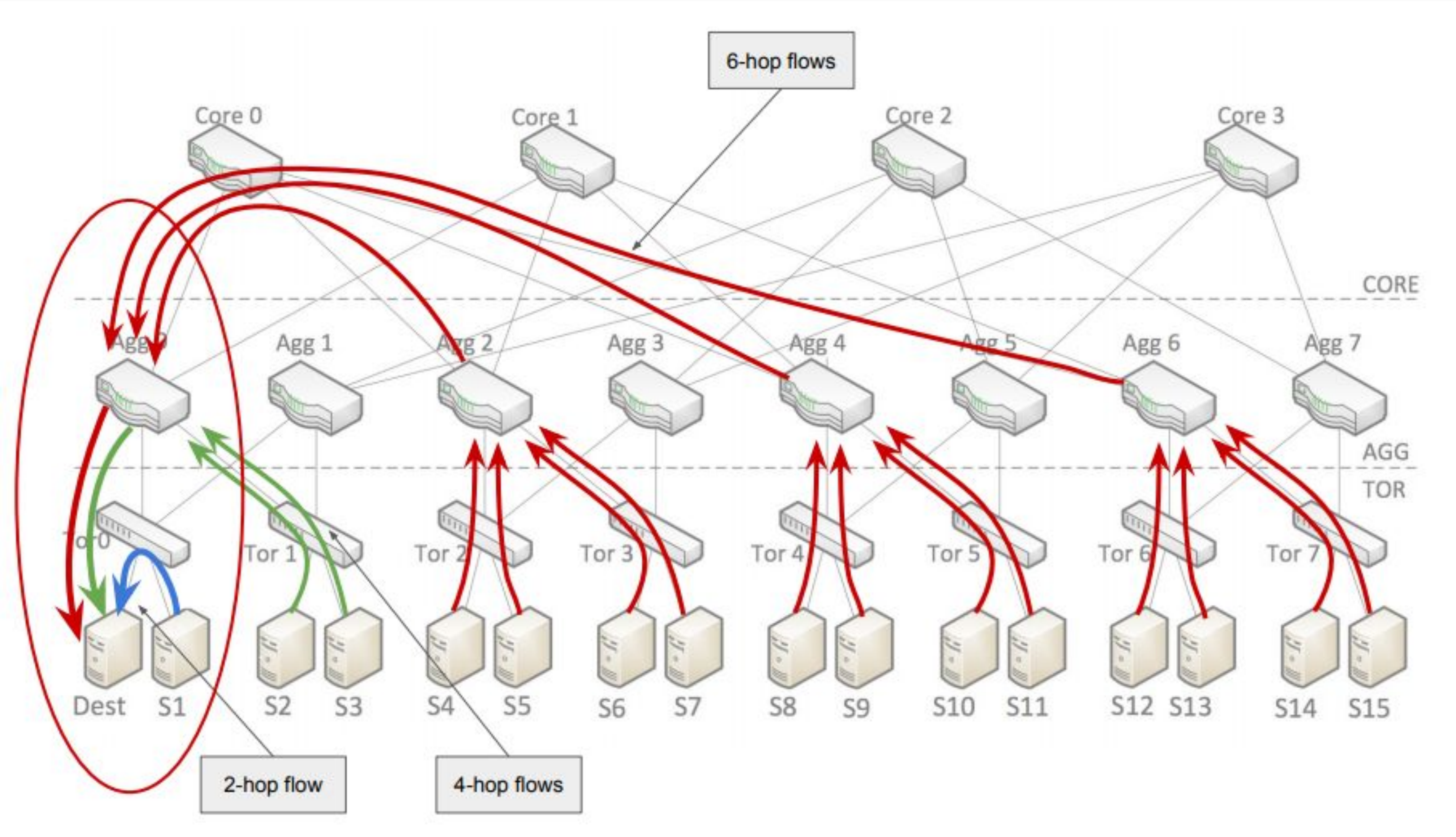
# Network Sharing in Data Centers

- Network sharing largely laissez-faire
    - Recent proposals (eg. SecondNet, Oktopus) call for per-tenant slicing
    - Still at research level though
- Today, flows in data centers compete via TCP
- Ideally, TCP should achieve true fairness
    - Flows get fair share of bottleneck link capacity
- In practice, TCP exhibits RTT-bias
    - Throughput is inversely proportional to RTT

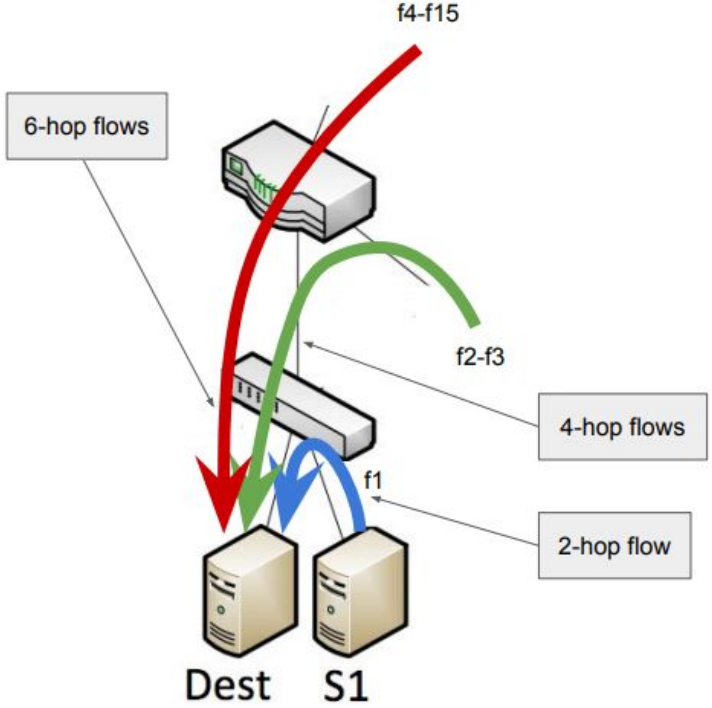
# Experimental Setup with K=4 Fat-Tree Topology

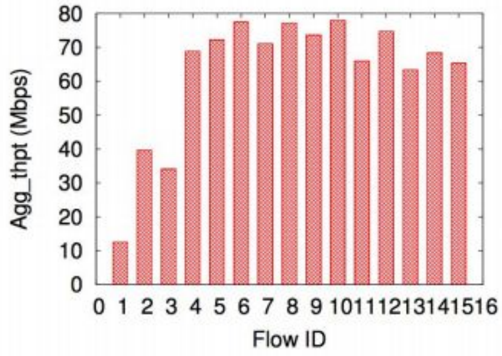


# Traffic Pattern

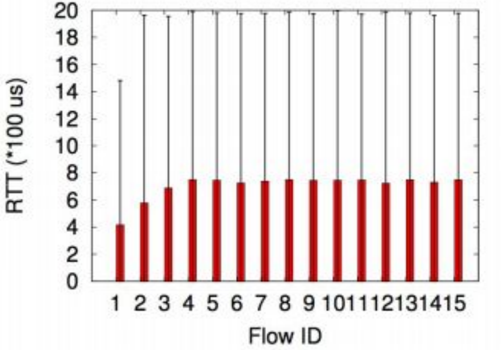


# Finding Unfairness



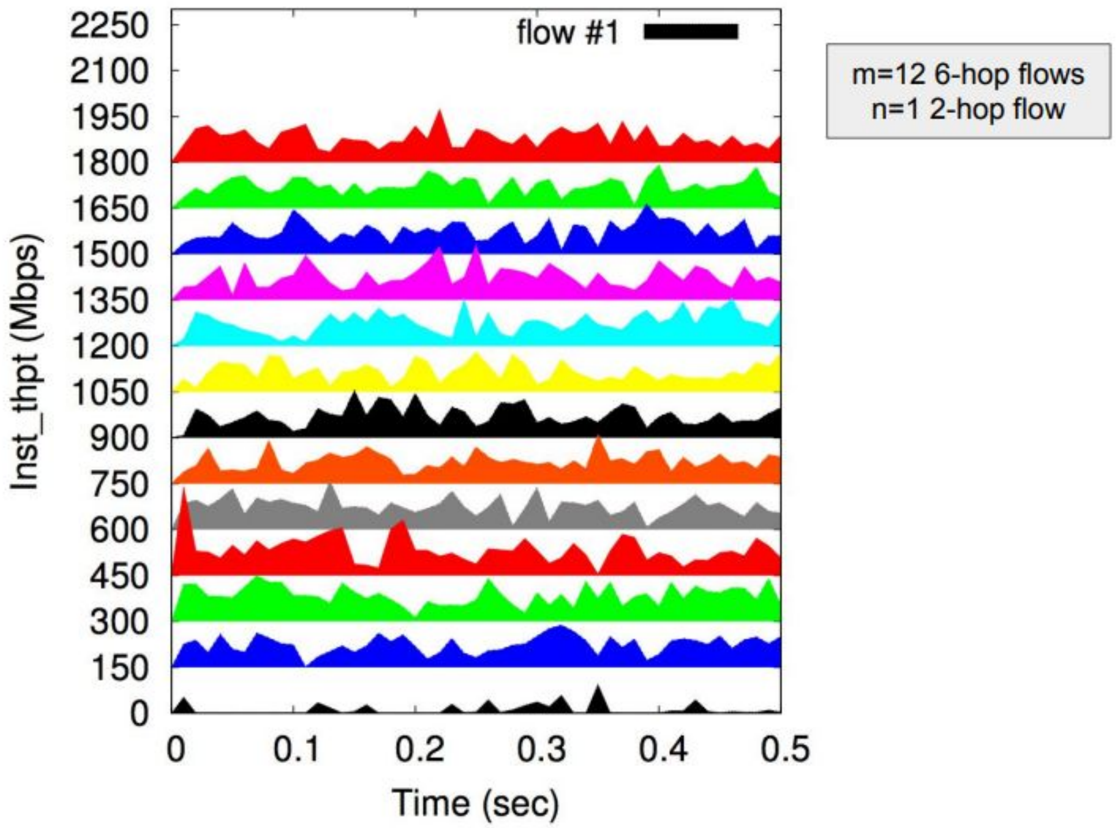




(a) Aggregate throughput




(b) Average RTT

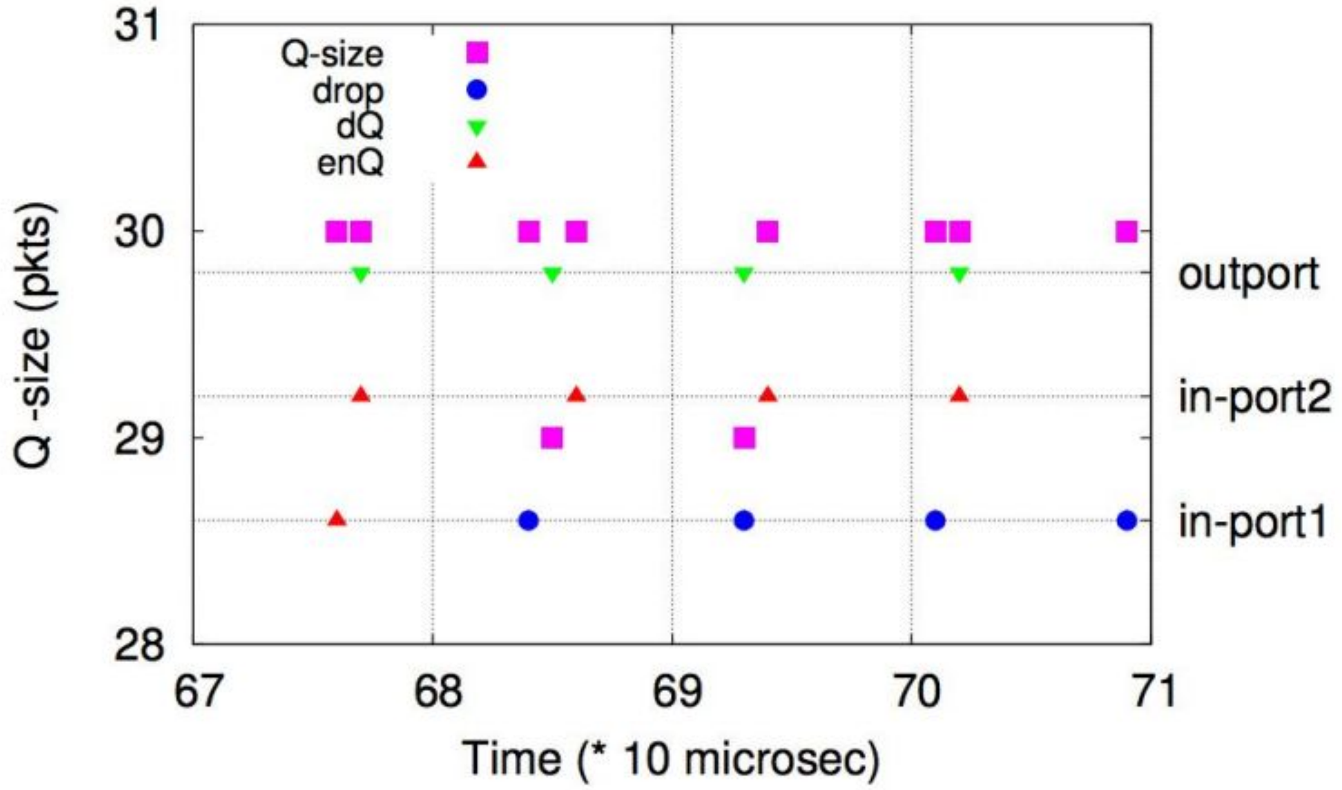
# Instantaneous Throughput



# Port Blackout : Reason for TCP Outcast

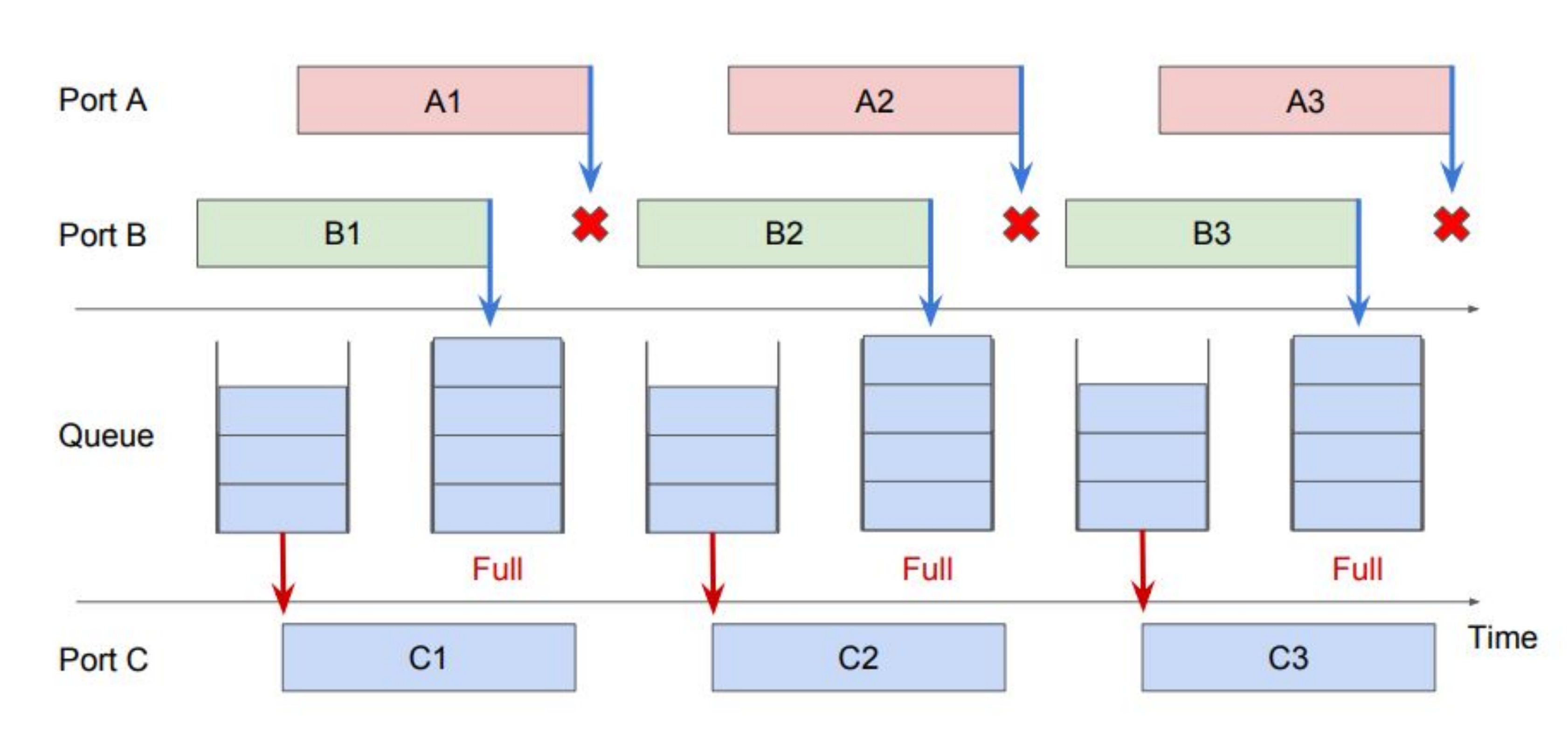
- Port blackout can happen to any input port
- It happens for small intervals of time
- But port blackout has more catastrophic impact on throughput of fewer flows!

# Port Blackout in NS-2




Timeline of port blackout when queue is almost full

# Port Blackout

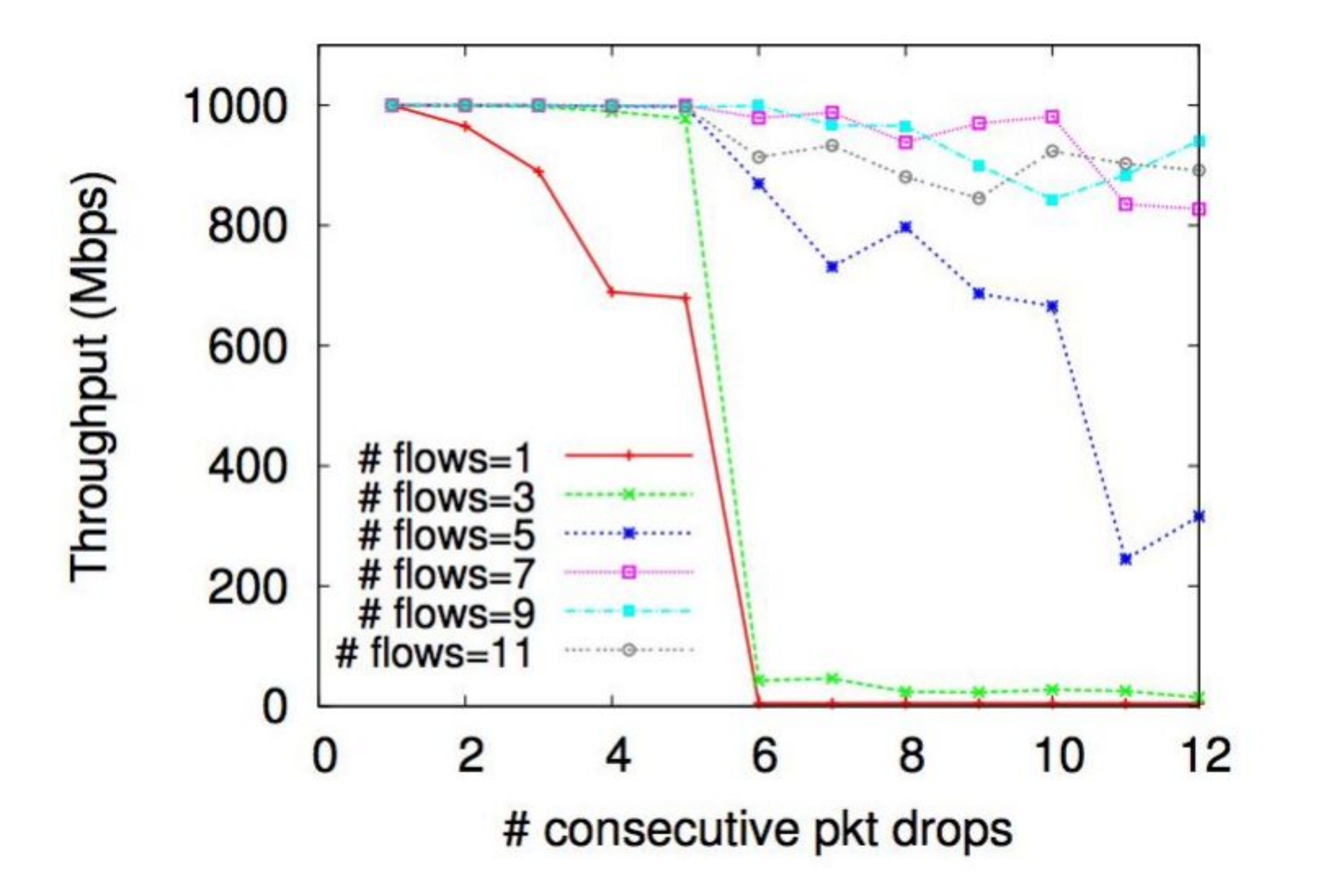



1) Packets are roughly of same size
2) Packets are sent back-to-back with similar inter frame spacing and hence predictable timing

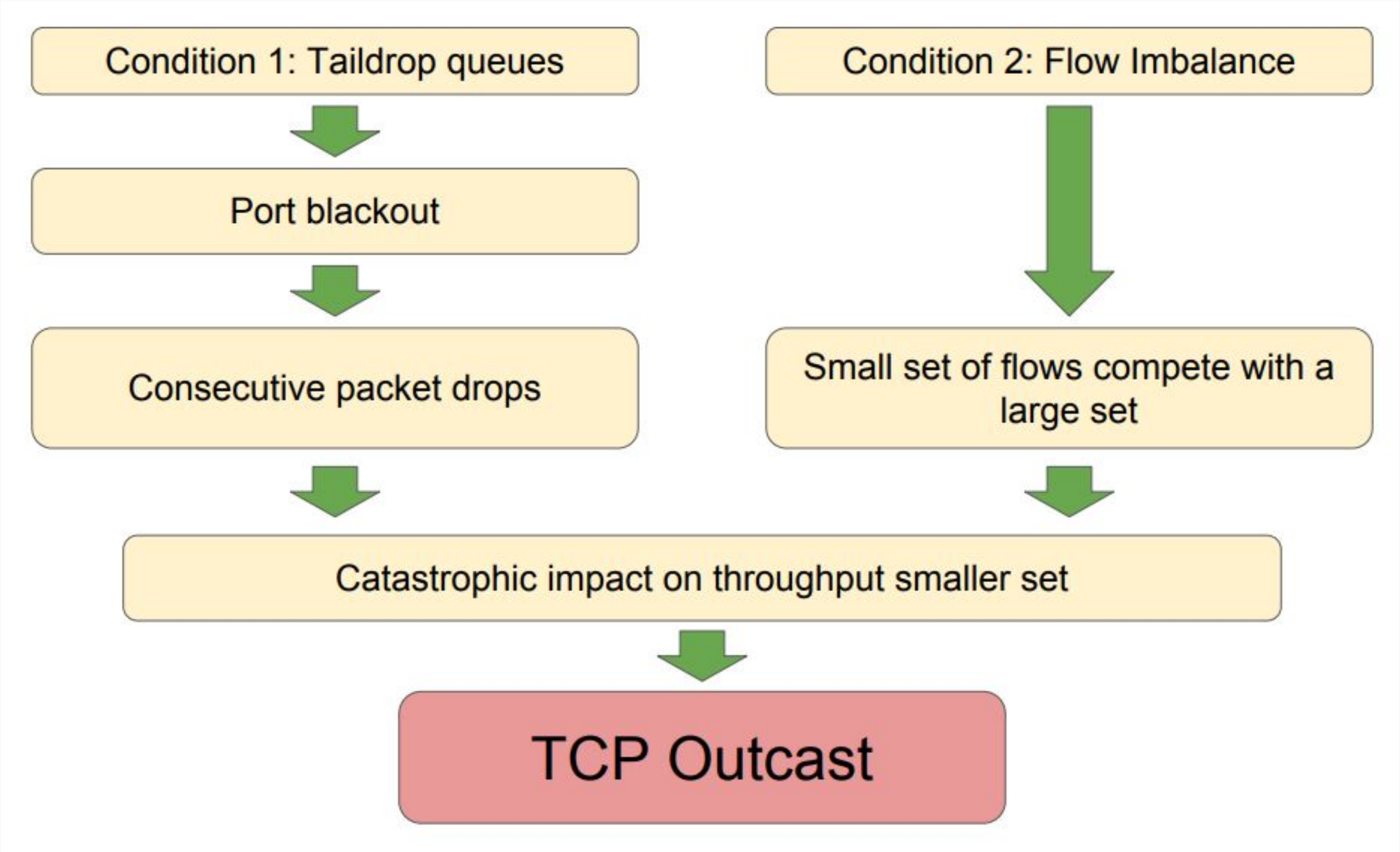
# Impact of Port Blackout on TCP

- Simulated port blackout in NS-2

- Queue toggles between ON and OFF modes
  - In ON stage, drops k successive packets
  - In OFF stage, allows all packets



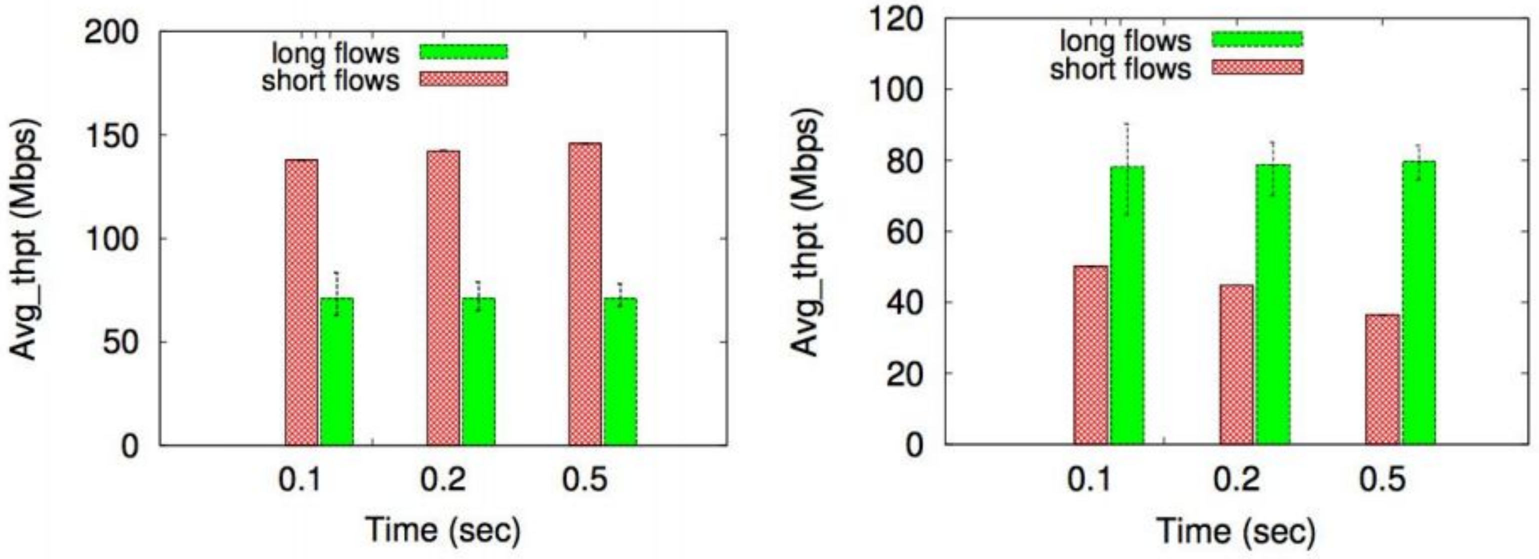
# Conditions for TCP Outcast



# Solutions for Outcast problem

- Random Early Discard (RED)
    - Retains RTT bias (does not give true fairness)
    - Switch based solution
- TCP Pacing
    - Helps to some extent
    - But, does not work perfectly
    - End-host based solution

# RED and TCP Pacing



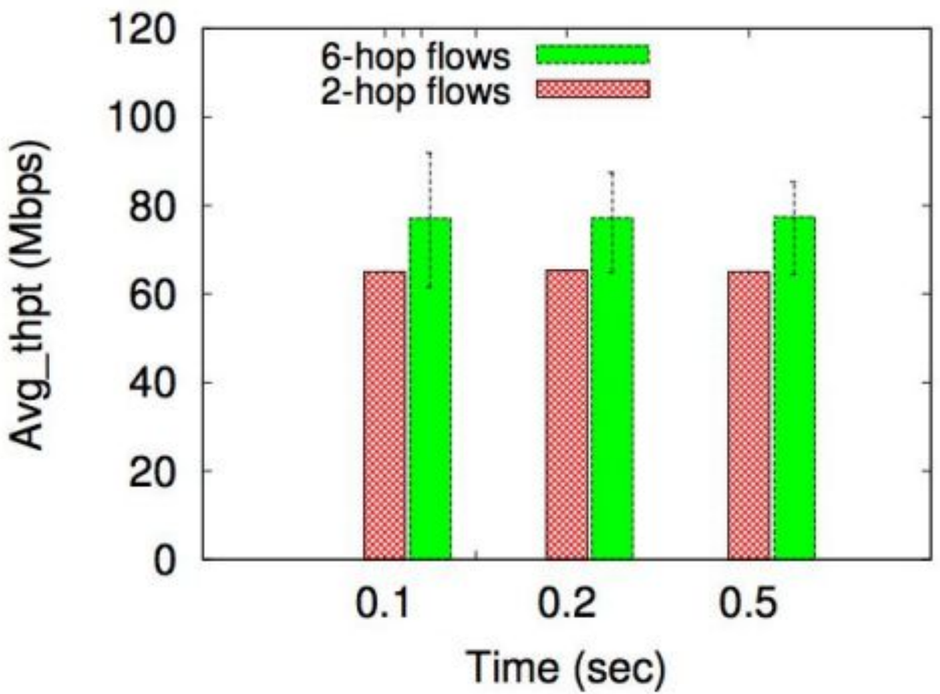

(a) RED

(c) TCP pacing

m=12 n=1

# Stochastic Fair Queuing (SFQ)

- Explicitly enforce fairness among flows
- Limited by number of supported classes

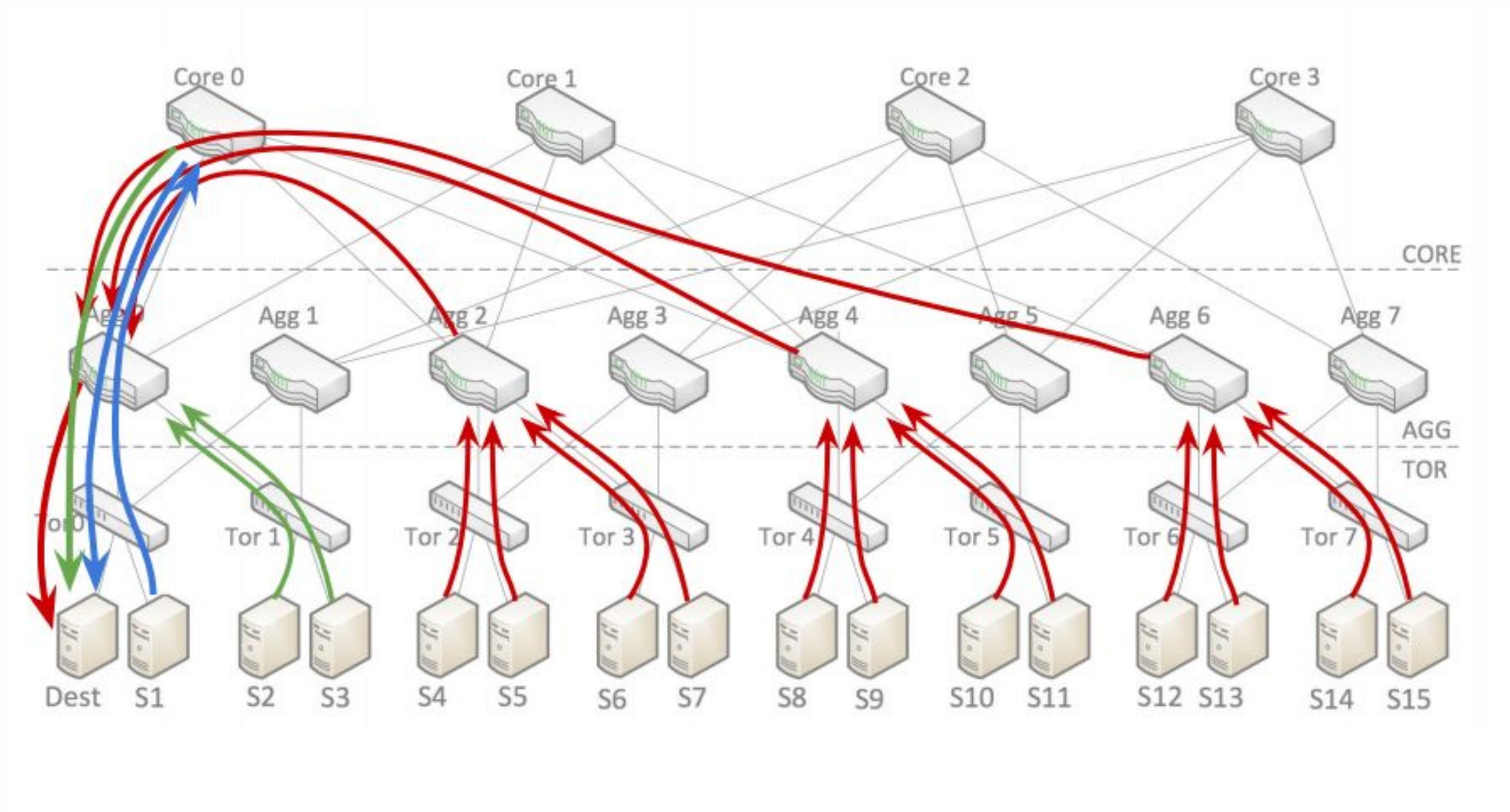



(b) SFQ

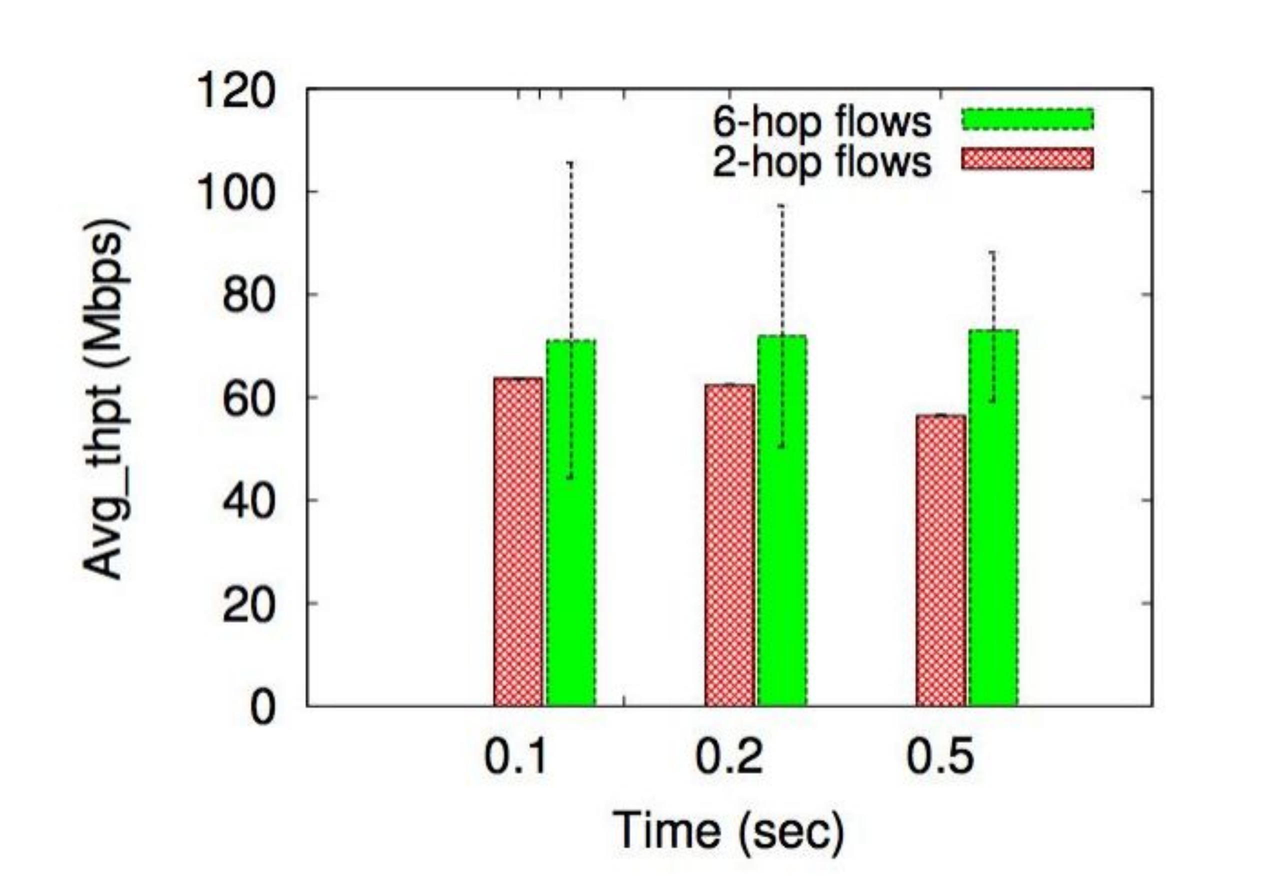
# Equal Length Routing

- All flows are routed through a core switch
- Uses ECMP (for randomization)
- Requires topology without oversubscription
  - Perfect fat-tree for example
- Allows better interleaving of packets to alleviate port blackout
- Flow bundles balanced on input ports

# Equal Length Routing



# Equal Length Routing



# Why does unfairness matter?

- In multi-tenant clouds, some tenants get better performance than some other
- Unfairness may cause straggler problems in MapReduce-type applications
  - One delayed flow effects overall job completion time
  - May cause increased memory requirements for applications (e.g. during merge sort)

# Conclusions

- A new problem called TCP Outcast problem
- Conditions for TCP Outcast:
    - Switches with taildrop queues (leads to port-blackout)
    - A small set of flows share a bottleneck link with a large set of flows
- Solutions such as SFQ and equal length routing help restore fairness