

Token Flow Control in Data Center Networks

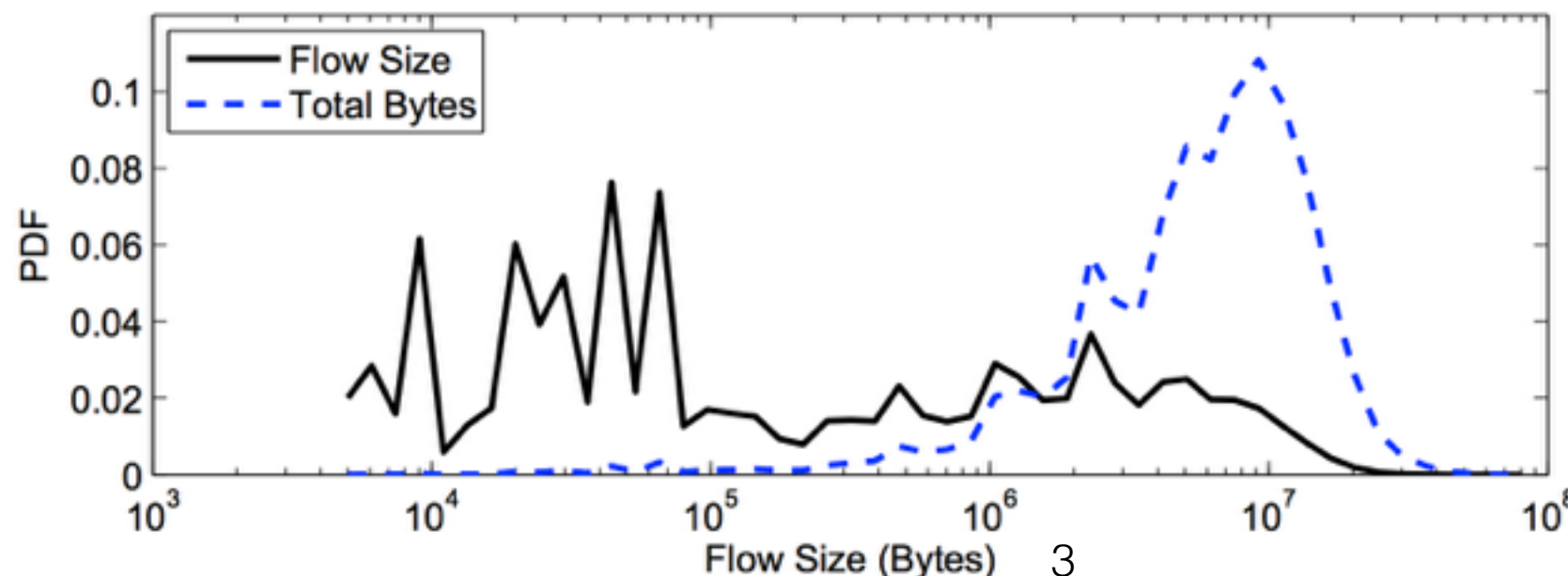
J. Zhang et al., Eurosys 2016

Outline

- Background
- Problem
- TFC Design
- Rethinking window-based CC for DCN

DCN flows' properties

- Bursty short flows (e.g. web search query)
 - size small (e.g. 0 - 100KB); latency-sensitive
- Large background flows (e.g. DB backup)
 - size large (e.g. > 10MB); throughput-demanding



*Statistics From
"DCTCP" paper*

DCN Network Properties

- High bandwidth (10Gbps -> 40Gbps -> 100+Gbps)
- Shallow buffered switch
- Small baseRTT (100 us -> 10 us)
 - $\text{instant_RTT} = \text{baseRTT} + \text{queueing delay}$
 - queueing delay increasingly dominant instant_RTT

Seemingly contradictory desired transport properties

- High throughput for Large flows
 - Fast Convergence; Necessary queueing
- Low latency for Short flows
 - Zero-loss; Little queueing

Goal of TFC transport

- Fast Convergence
 - quickly converge to a proper share of bandwidth
- Zero Packet Loss
 - specially with highly concurrent flows
- Low Latency (near-zero queueing)
 - desirable for small flows

Why existing works not enough

- TCP-variants and DCTCP (Window-based ones)
 - probe for available bandwidth and throttle if congestion detected
- RCP/D³/FastPass (Explicit rate calculation ones)
 - Unable to handle silent flows (connections held not closed and wait for data to resume)

TFC: Token Flow Control

- Explicit Window-based transport protocol
 - Each switch assign congestion window to each active flow
 - The minimum congestion window along the path of a flow will be carried back to the sender by ACKs

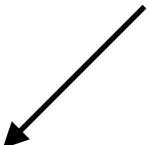
Challenges to tackle

- Near-zero queueing need exclude buffer space
- Get # active flows (SYN may be inaccurate)
- Work-conserving with multiple bottlenecks
- Window less than one (highly concurrent flows)

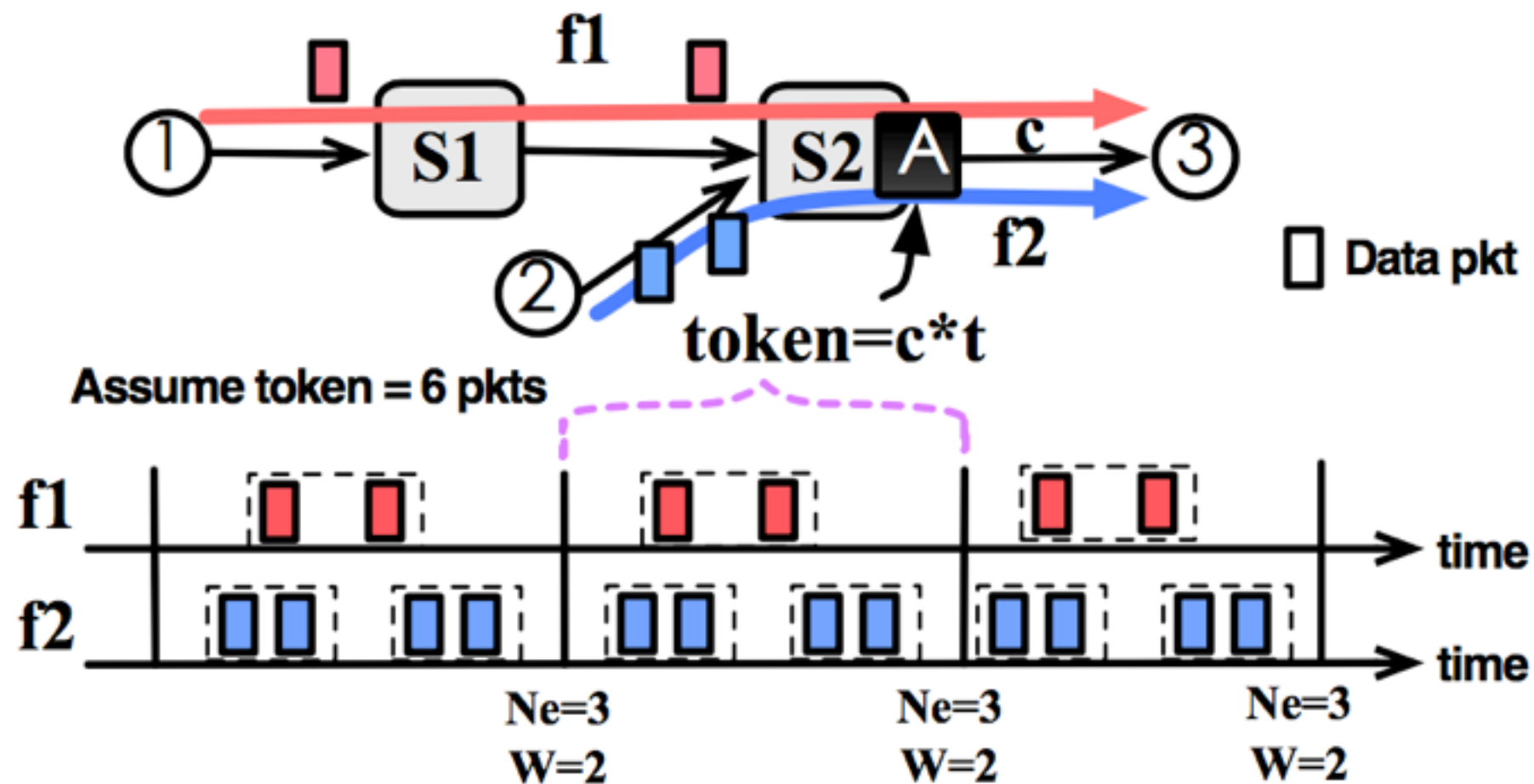
TFC Design

- Divide time into slots $[1, 2, \dots, n, \dots]$
- Token **$T[n]$** : the amount of data that can be injected to saturate the link in time slot n . (*Resource*)
 - Fast convergence to shared bandwidth
- # Effective flows **$E[n]$** : number of full-window-size data that are injected by all the passing flows in time slot n . (*Consumer*)
 - Ignore silent flows

TFC Design

- # Effective flows : $E[n] = \sum_f \frac{t}{rtt_f}$
time slot length
- UpdateWindow : $W[n+1] = \frac{T[n]}{E[n]}$

TFC basic model



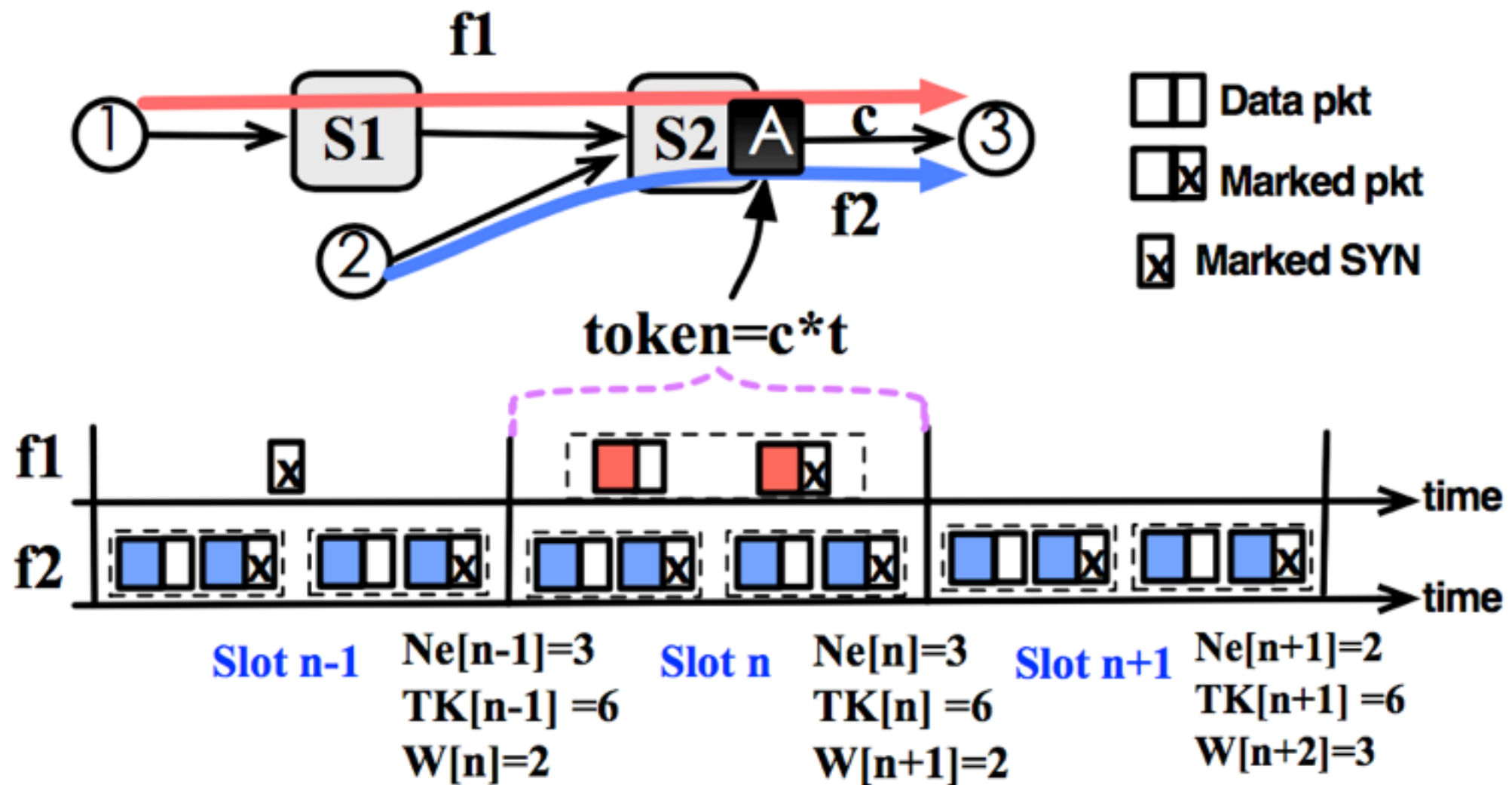
Two flows, f_1 and f_2 , passing port A. Assume $rtt_1 = 2 \times rtt_2$ and the token value is 6 packets. Since there are three effective flows in a time slot, the congestion window is 2 packets for each flow.

Get active flow amount

1. each flow send a small 1byte signal packet along with sending data.
 - bandwidth waste
2. each switch calculates : divide the bytes passed in one slot by the window obtained in last slot.
 - window may not be the bottleneck value

Combing both, each flow mark one packet for each slot

TFC model



Senders mark the first packet of each full window of data packets to facilitate switches to measure $E[n]$.

Duration of a time slot: t

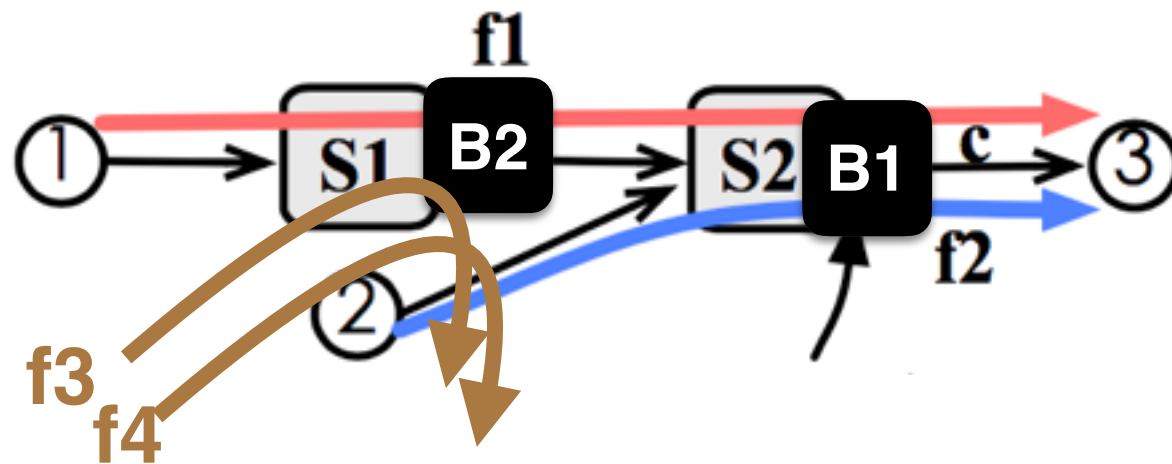
- If too large
 - could not update the congestion window according to the number of new/finished flows
- If too small
 - the number of effective flows will fluctuate dramatically, thus window sawtooth will be severe

baseRTT of any flow under multi-rooted tree topology

Effect on $T[n]$ and $E[n]$

- baseRTT for calculating $T[n]$
 - Zero-queue BDP estimation
- RTT (with queueing delay) for measuring $E[n]$
 - Due to the counting of marked packet scheme

Deal with under-utilization



- Cause: multiple bottlenecks; each switch maintains a single window value for all passing flows
- Solution:

$$T[n] = c \times rtt_b[n] \times \frac{\rho_0}{\rho[n]}$$

expected utilization

actual utilization

Implementation

- Endhost: GNU/Linux kernel 2.6.38.3. two reserved bits in the *flags* field to mark the first packet in each full window.
- Switch: NetFPGA to compute $T[n]$, measure $E[n]$ and update $W[n+1]$. most overhead in division.

Evaluation

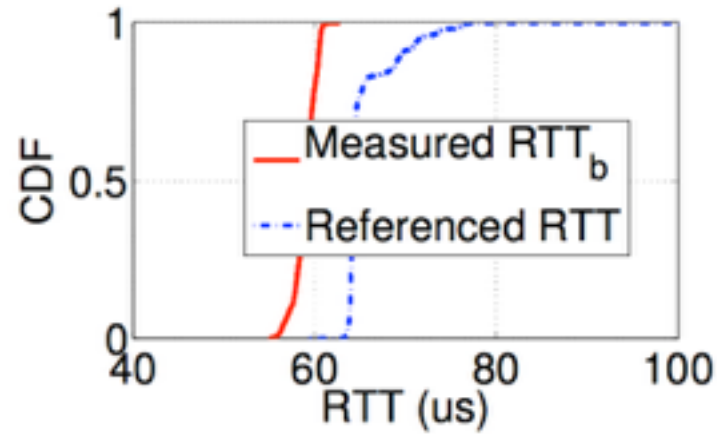


Figure 6: Measured rtt_b .

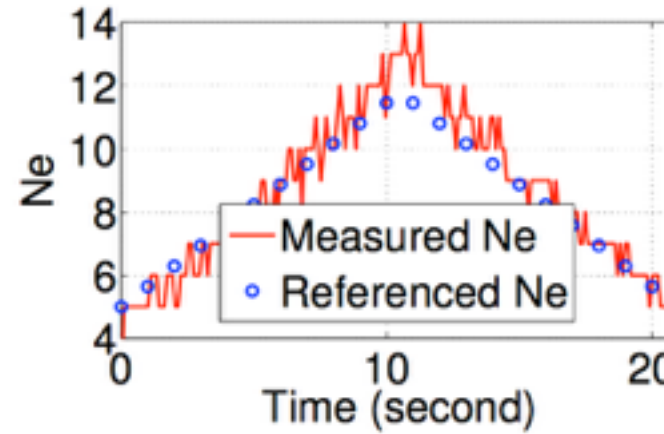


Figure 7: Accuracy of N_e with inactive flows.

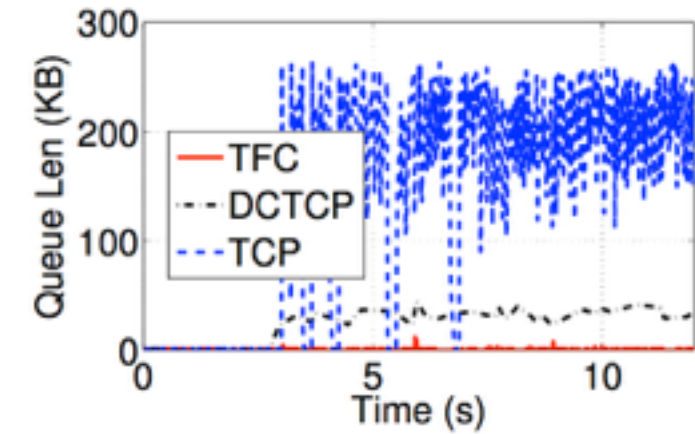
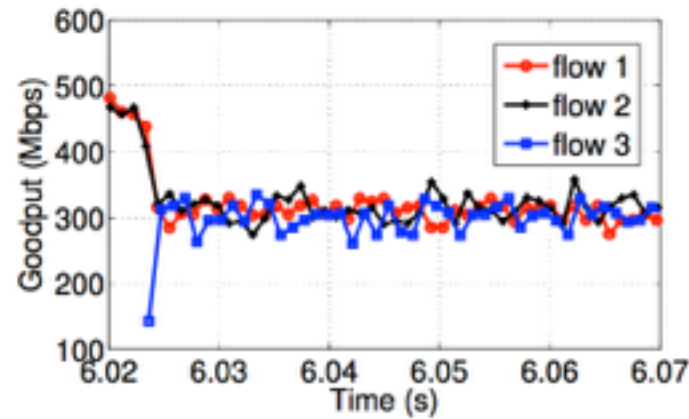
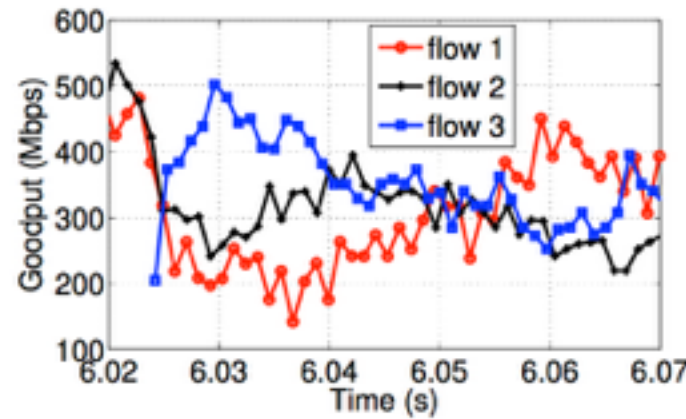


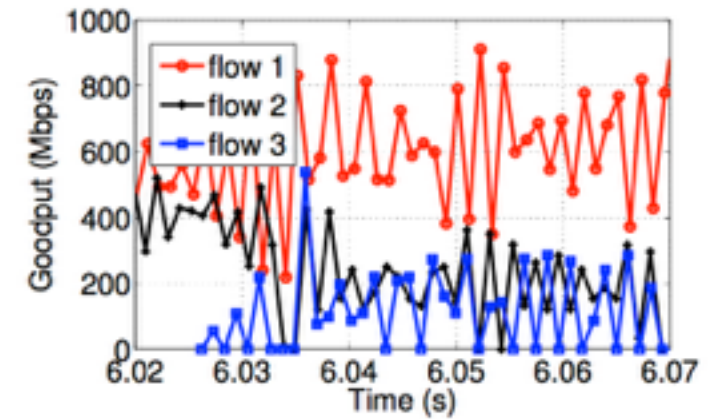
Figure 8: Queue length.



(a) TFC



(b) DCTCP



(c) TCP

Figure 10: Convergence rate.

Evaluation

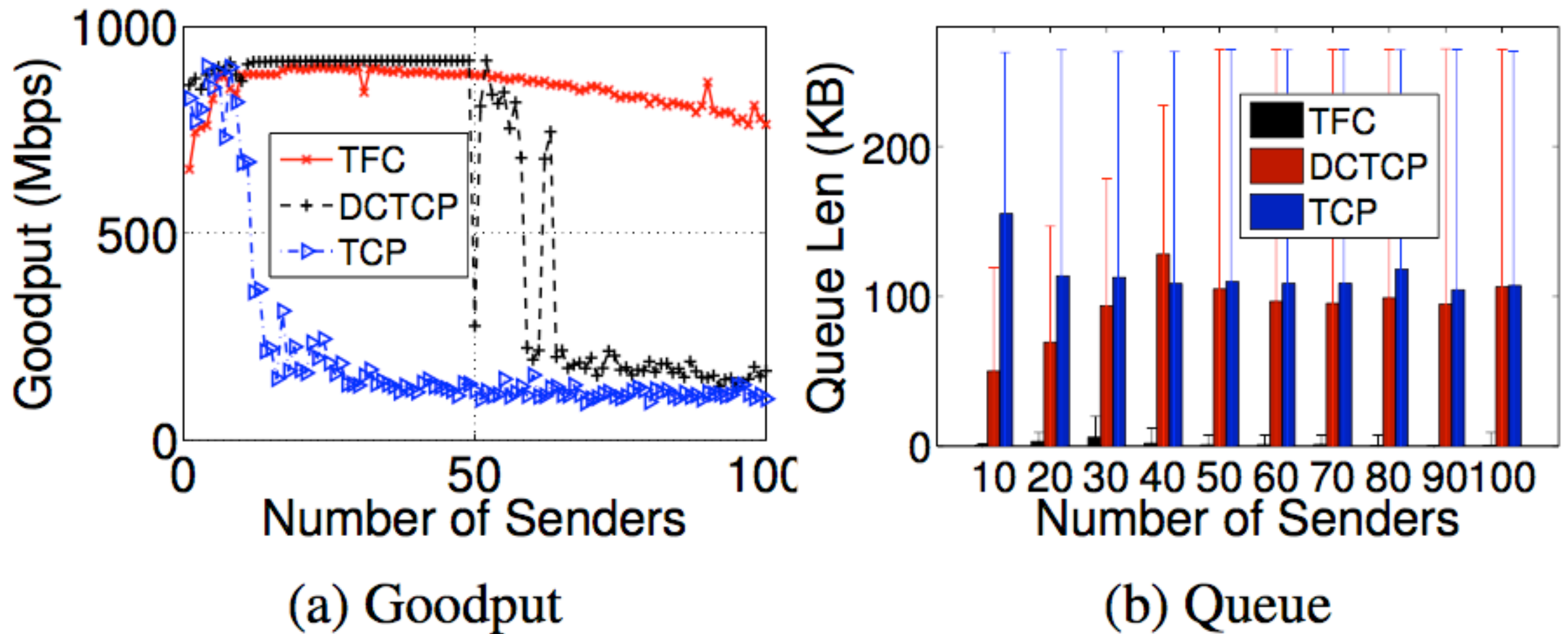


Figure 12: Incast: goodput and queue length.

Why I present this paper

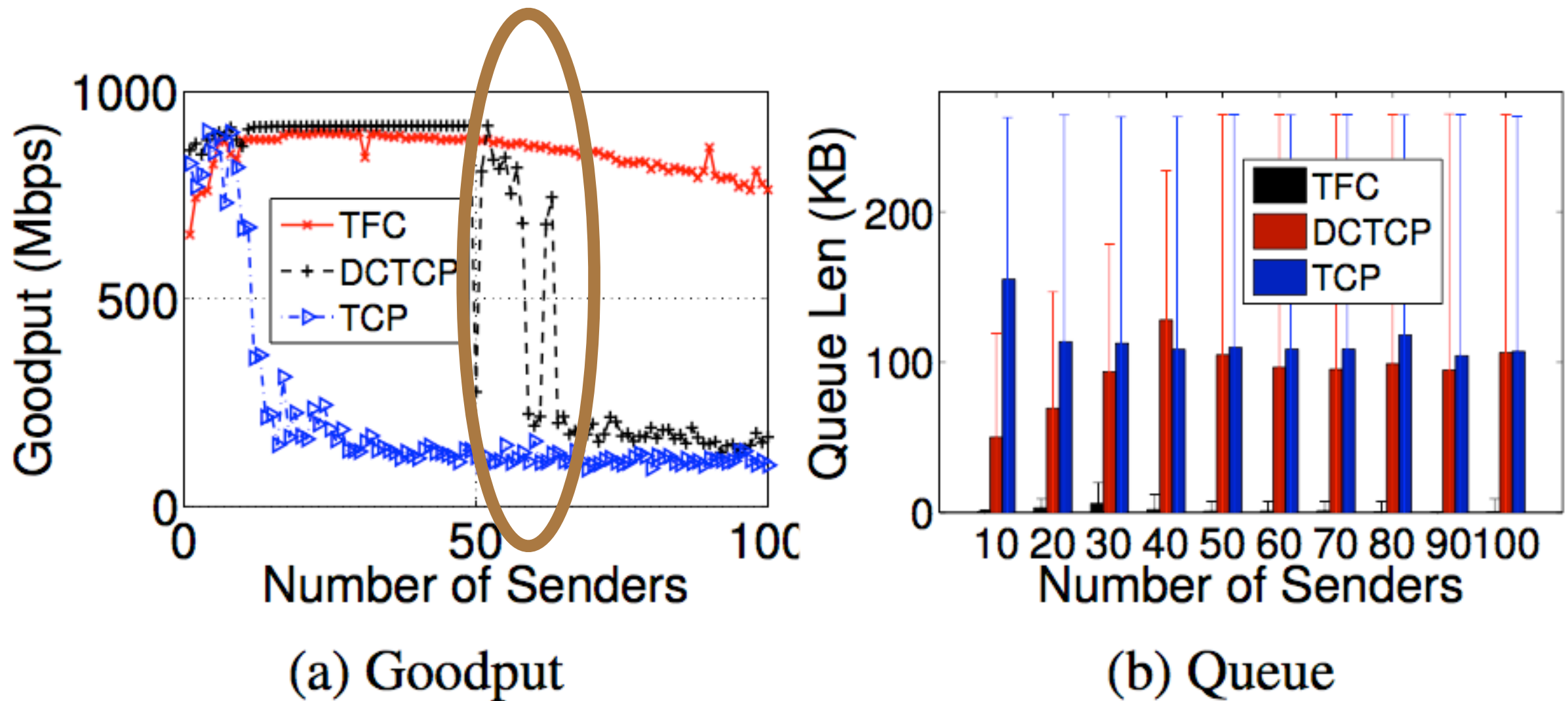
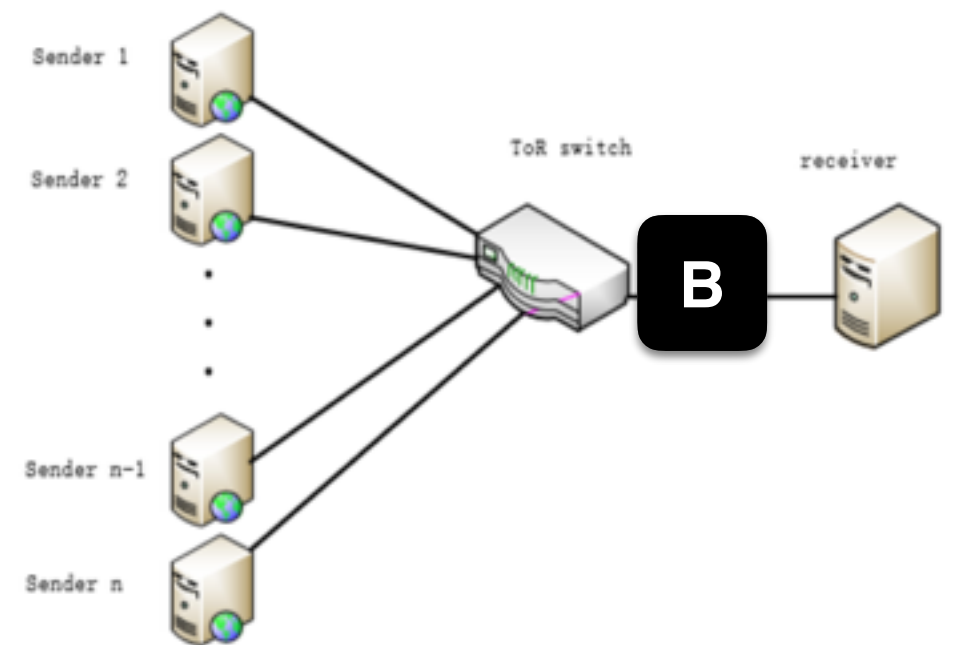


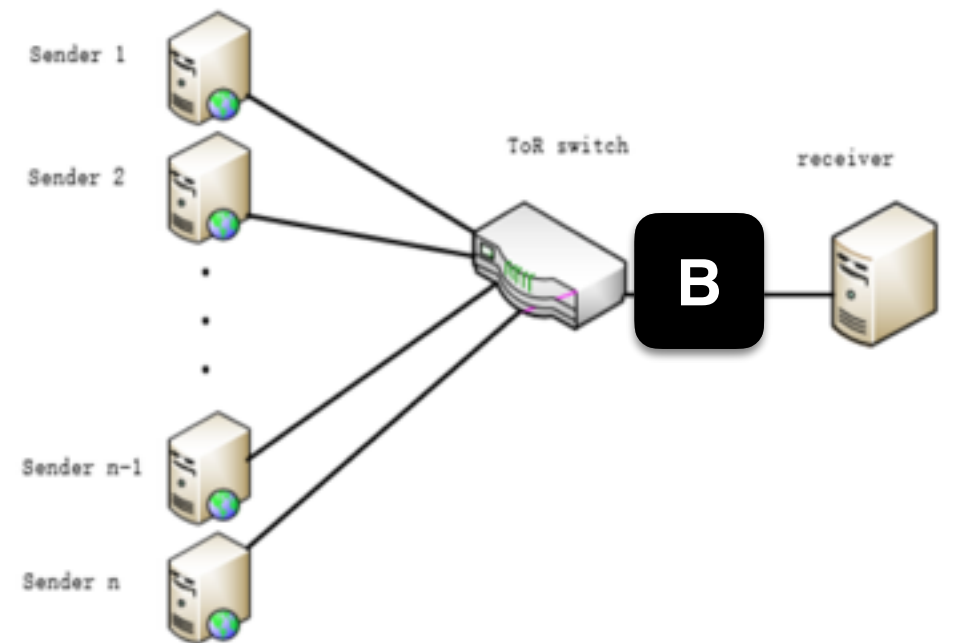
Figure 12: Incast: goodput and queue length.

Consider this setting



- Bottleneck link bandwidth: 40 Gbps
- baseRTT: 10 μ s
- zero-queue BDP = $40 \text{ Gbps} * 10 \mu\text{s} = 400000 \text{ bits} = 50000 \text{ Bytes}$

Suppose MTU = 1500 Bytes



- zero-queue BDP = $50000 / 1500 = 33$ MTU-size packets
- With DCTCP, if marking threshold $K=40$ MTU-size packets, then the Capacity of the bottleneck link is $33+40 = 73$ MTU-size packets

Window less than MTU

- Here, the link can support no more than 73 of active flows. If $\# \text{active flows} > \text{capacity}$, some may get window less than one MTU even by injecting a single MTU-size data packet, then wait for connection timeout.
- Possible solutions:
 1. set MTU smaller by $\text{MTU} = \text{BDP} / E[n]$
 2. allow window to be less than one MTU, set a high resolution timer $\text{RTT} / \text{Window}$ to pace packet

Window less than MTU

- TFC solves this problem at switches enlightened by the traffic shaping mechanism, token bucket algorithm.
- Each switch maintains a counter for a port, which represents how many data can be sent. The counter will increase as time elapses.

Window less than MTU

- On a marked ACK packet arrival, if the carried congestion window is smaller than MTU and the counter value is larger than MTU, then the carried congestion window in the ACK header will be modified to MTU and the counter decreases by one MTU.
- Otherwise, the ACK will be put into a delay queue to wait for a large enough counter.

Limitation of TFC

- Not supported by commodity switch
- Unfairness issue
- Fault tolerance (marked packets dropped due to hardware bug)

Window-based CC for high-speed, ultra-low latency, ECN-enabled, Small BDP network

Discussion