Data Center TCP (DCTCP)

Mohammad Alizadeh, Albert Greenberg, David A. Maltz,
Jitendra Padhye, Parveen Patel, Balaji Prabhakar,
Sudipta Sengupta, Murari Sridharan

SIGCOMM 2010

Remember this name if you want to work in DCN :P
We call him big M (will introduce small M in the following lectures…)

Outline

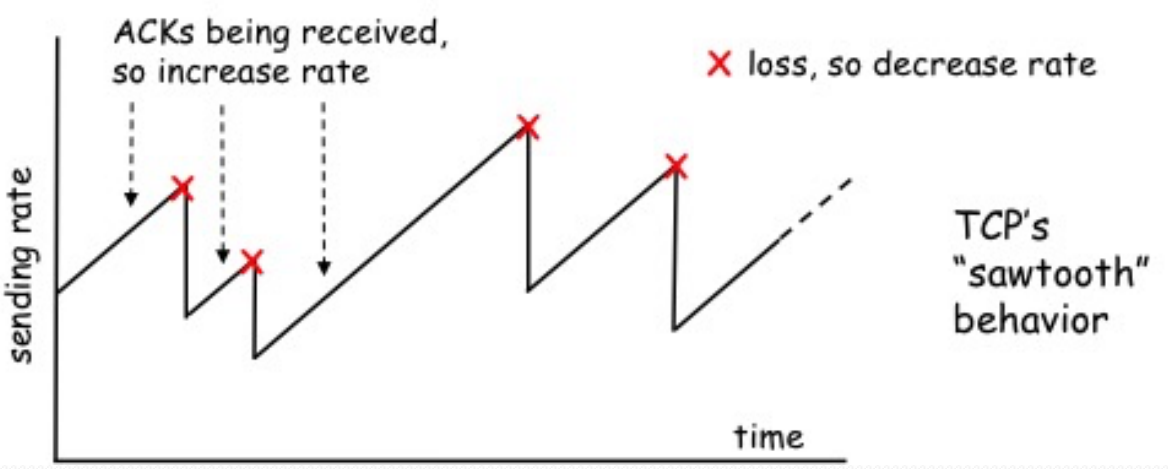
- **Background (Not in the paper…)**
- DCTCP
- Implementation and evaluation
- Review

TCP Before DCTCP

- Function of Transport:
  - Reliable Connection
  - In-order delivery
  - Flow Control
  - **Congestion Control**

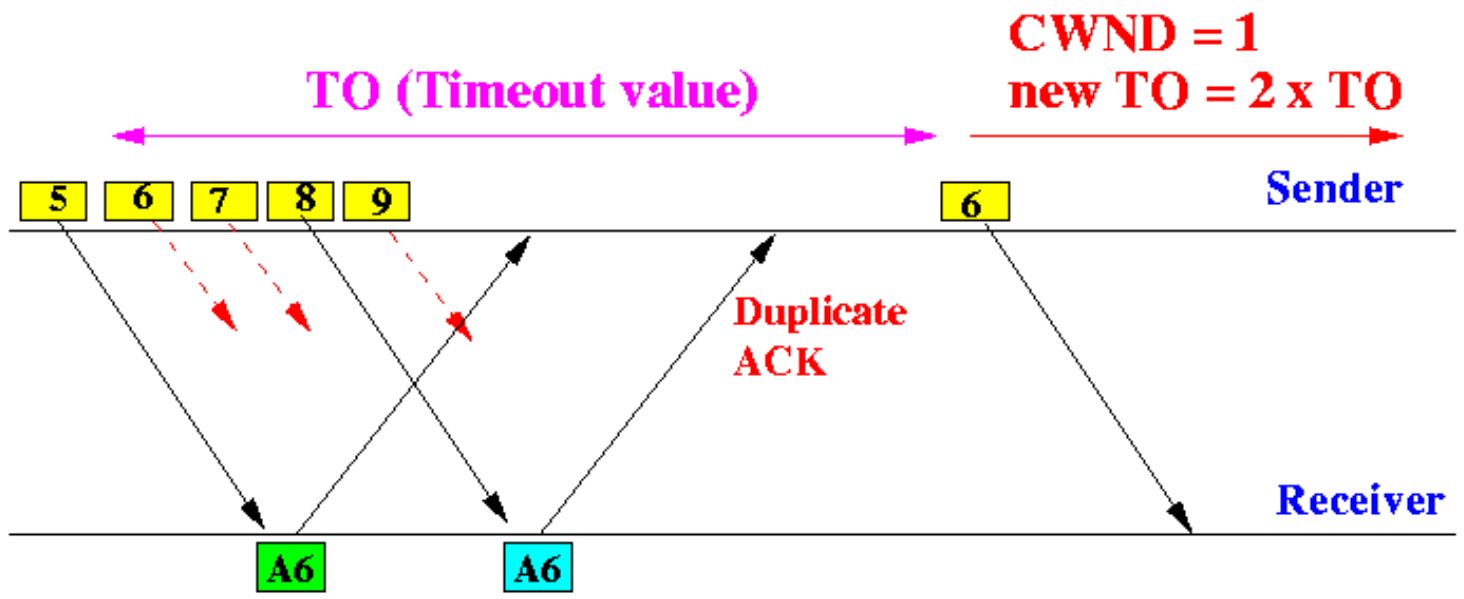
How to Control Congestion?

- **No Loss - Additive Increase**
  - cwnd = cwnd+1
- **Loss - Multiplicative Decrease**
  - cwnd = cwnd/2



Loss?

- Duplicate ACK
- Timeout (millisecond level)



How to Control Congestion?

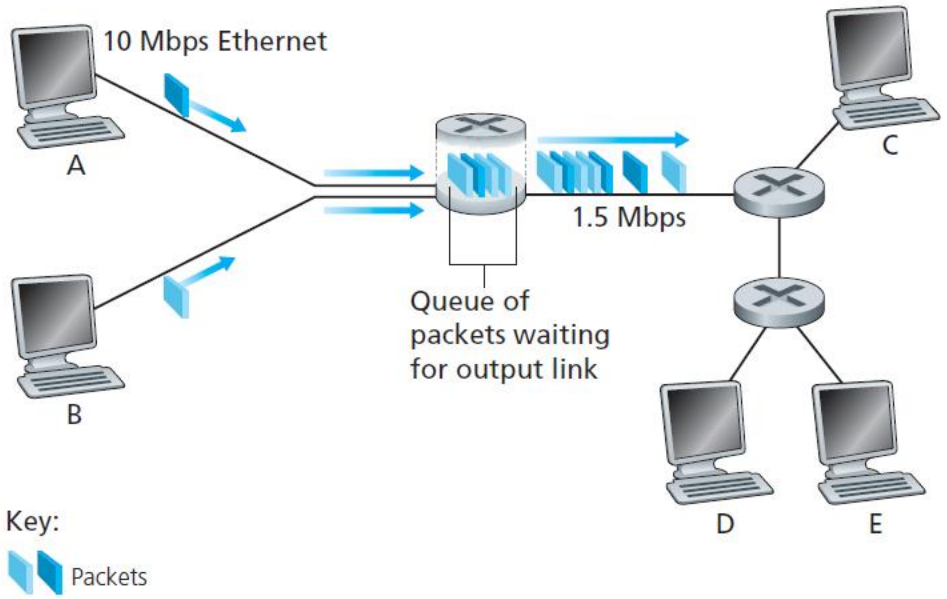


Figure 1.12 ♦ Packet switching

High throughput: The switch has something to send when it has a change (No empty queue for a long time)

Low lantecy: The queue should be small

Can TCP Work in DCN?

- **DCN is different from Internet**

	DCN	Internet
Ownership	Single administrative domain	Many
Scale	Short, within DC	Large, globally
Topology Structure	Structured, regular	Policy driven (AS)
Bandwidth	High (100-800 Gbps)	Low (~Mbps)
Latency	**Ultra low (µs–low ms)**	Higher and variable (ms–100+ ms)
Traffic Pattern	**Server-to-server**/East-west **Long tail distribution**	Client-to-server/North-south
Switch	**Shallow Buffer**	Deep Buffer

Long Tail Distribution

- **Few large flows dominate the bandwidth**
  - High throughput
- **Many small flows**
  - Low latency
  - Tail latency (90^{th}/99^{th} percentile)

Can TCP Work in DCN?

Packet loss is too late as it makes TCP react only when queue is full, which causes short flows to suffer long latency!

Can TCP Work in DCN?

- **DCN is different from Internet**

	DCN	Internet
Ownership	Single administrative domain	Many
Scale	Short, within DC	Large, globally
Topology Structure	Structured, regular	Policy driven (AS)
Bandwidth	High (100-800 Gbps)	Low (~Mbps)
Lantecy	**Ultra low (µs–low ms)**	Higher and variable (ms–100+ ms)
Traffic Pattern	**Server-to-server**/East-west **Long tail**	Client-to-server/North-south
Switch	**Shallow Buffer**	Deep Buffer

Loss Causes Large Tail Latency

- Why packet loss hurts tail latency for short flows?
  - Short flows have few packets -> Not enough Dup ACK
  - Then Timeout ☹ ~ milliseconds
  - DCN RTT: Ultra low (μs)
- Will packet loss hurts the performance of large flows?

Can TCP Work in DCN?

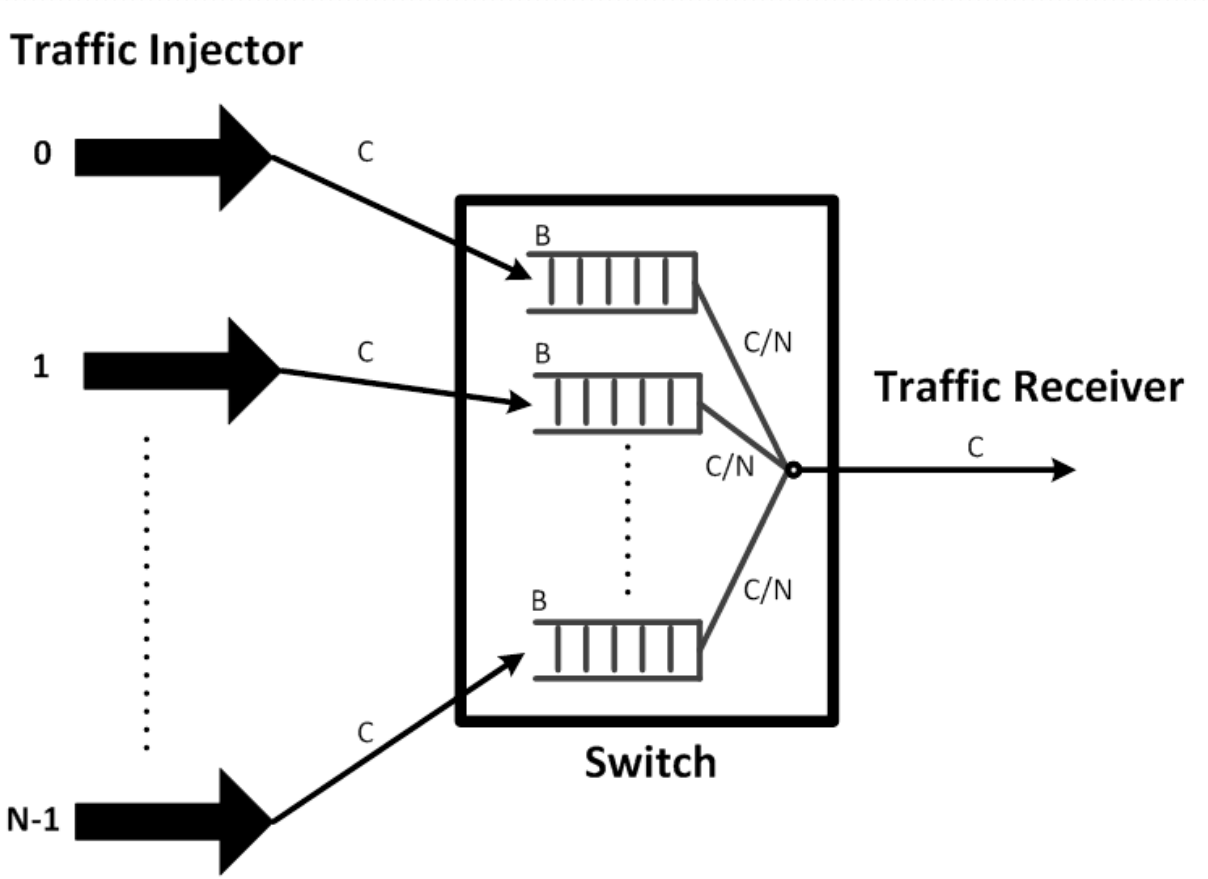
If drop triggers timeout, it causes very poor tail latency

Can TCP Work in DCN?

- **DCN is different from Internet**

	DCN	Internet
Ownership	Single administrative domain	Many
Scale	Short, within DC	Large, globally
Topology Structure	Structured, regular	Policy driven (AS)
Bandwidth	High (100-800 Gbps)	Low (~Mbps)
Lantecy	**Ultra low (µs–low ms)**	Higher and variable (ms–100+ ms)
Traffic Pattern	**Server-to-server**/East-west **Long tail**	Client-to-server/North-south
Switch	**Shallow Buffer**	Deep Buffer

Incast


Traffic Injector
0
1
N-1
C
C
C
B
B
B
C/N
C/N
C/N
Traffic Receiver
C
Switch

Incast

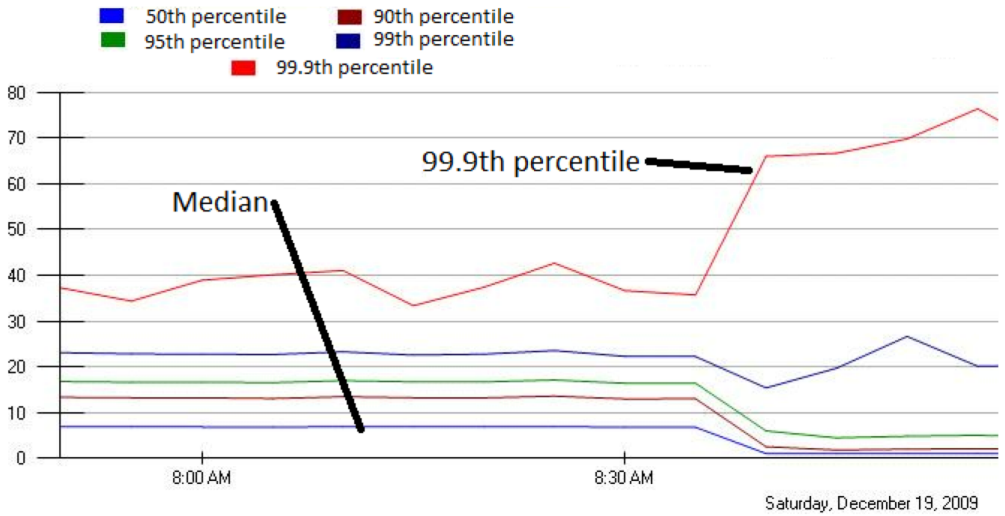


Figure 8: Response time percentiles for a production application having the incast traffic pattern. Forwarded requests were jittered (deliberately delayed) over a 10ms window until 8:30am, when jittering was switched off. The 95th and lower percentiles drop 10x, while the 99.9th percentile doubles.

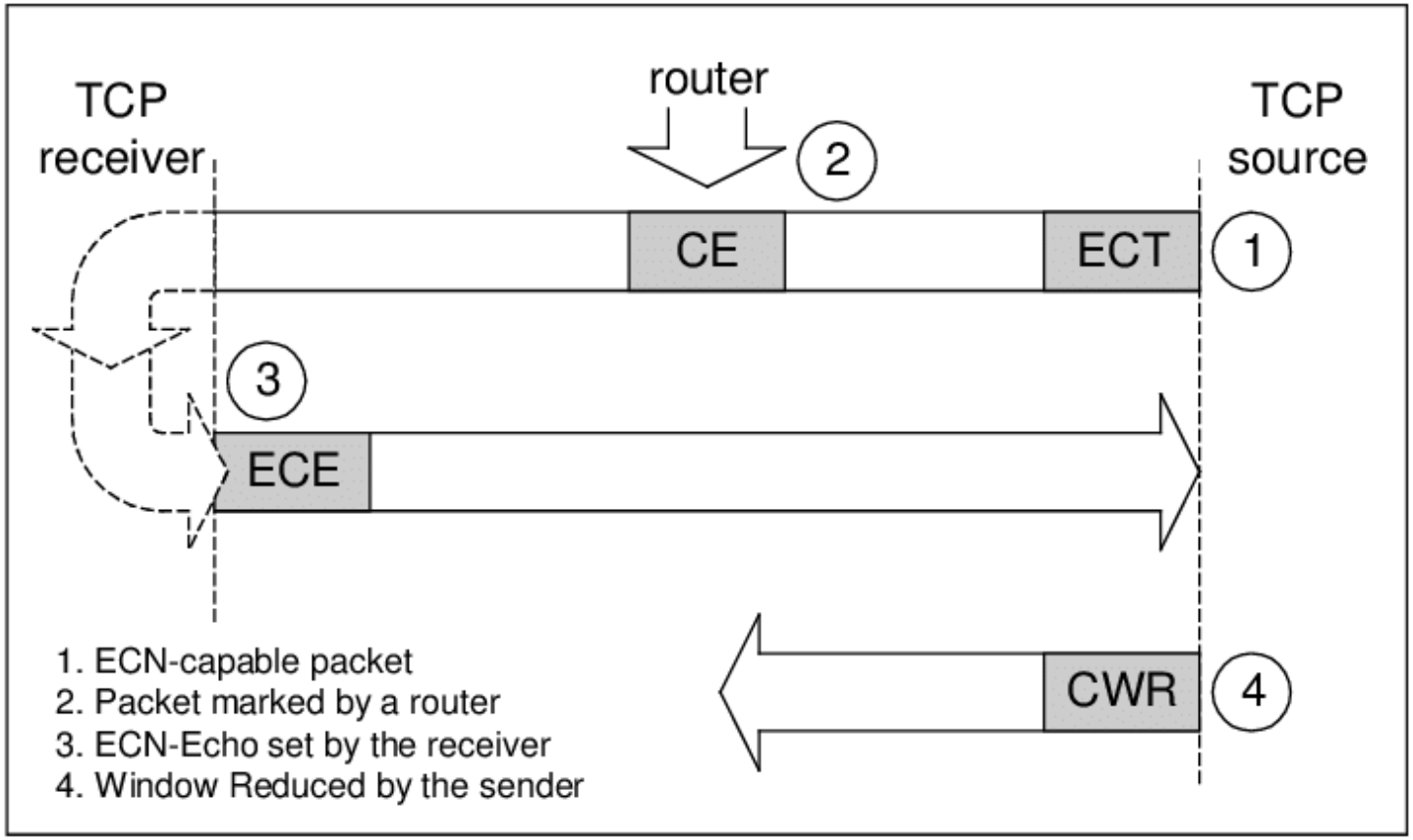
Can TCP Work in DCN?

Incast makes the tail latency problem even severe!

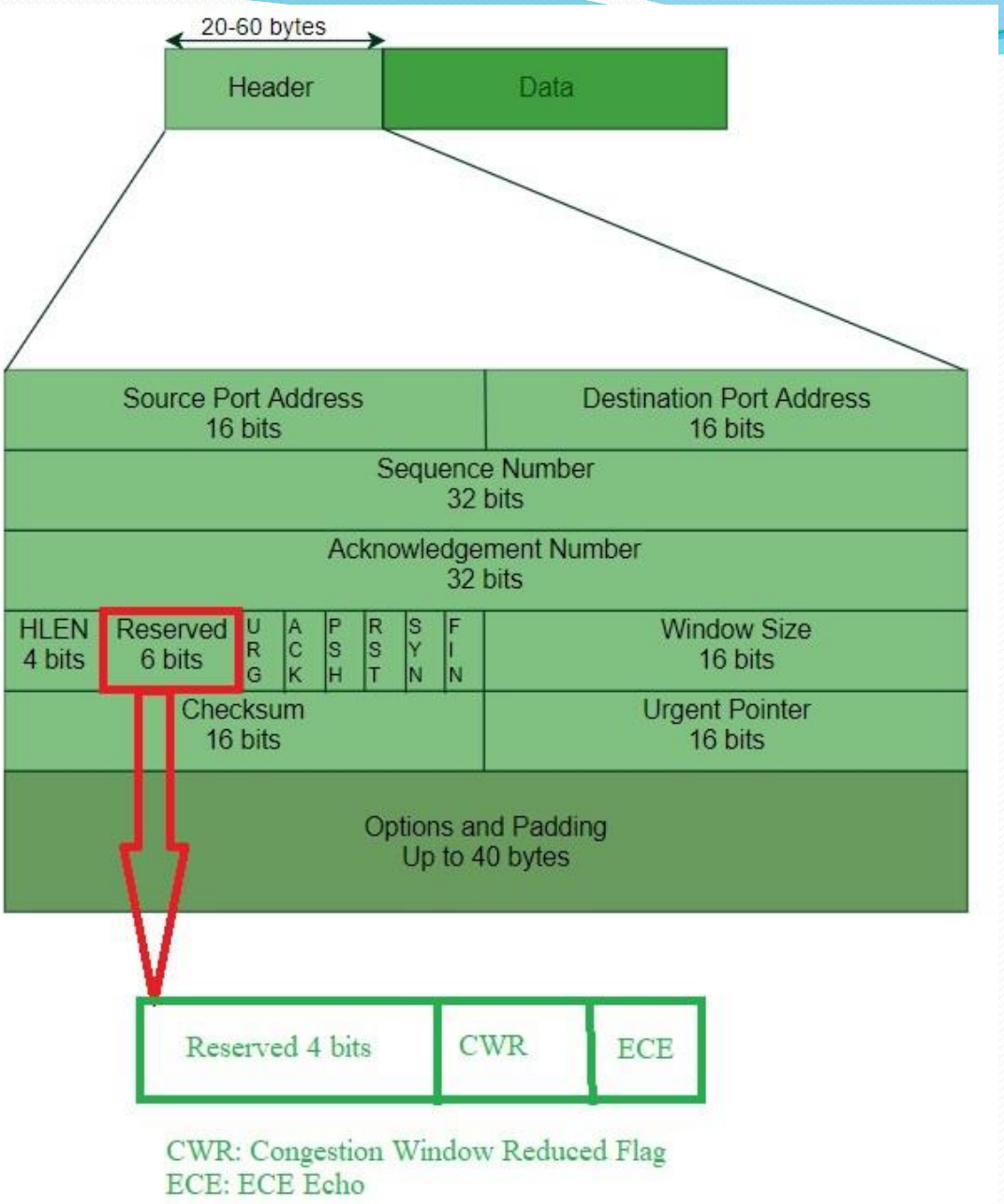
Some Improvements!

- React earlier than loss! (buffer full)
- ECN: Explicit Congestion Notification
  - Send Explicit Mark
  - cwnd = cwnd/2 if receiving a mark
- RED: Random Early Detection

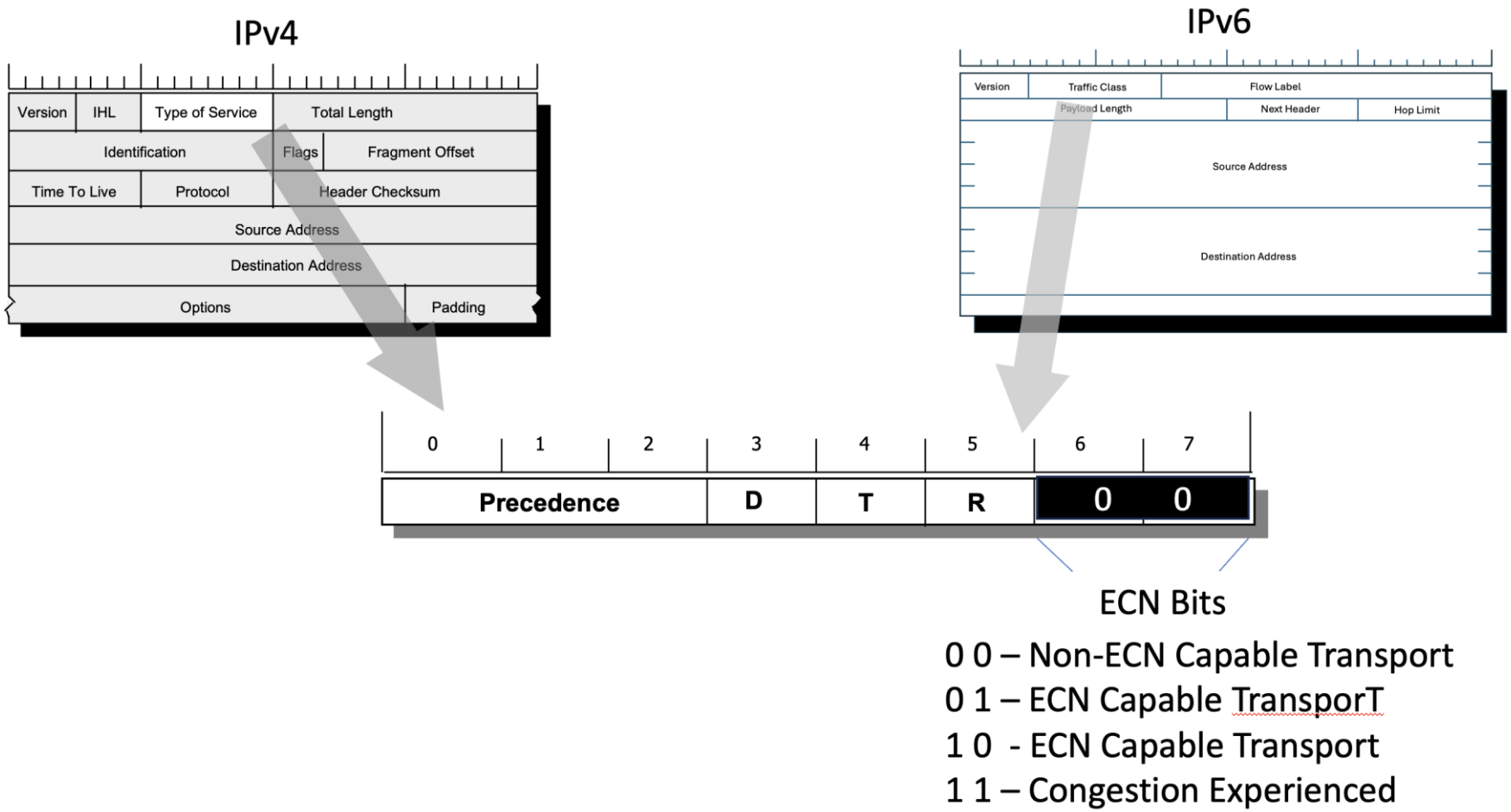
ECN


TCP receiver
router
TCP source
2
CE
ECT
1
3
ECE
CWR
4
1. ECN-capable packet
2. Packet marked by a router
3. ECN-Echo set by the receiver
4. Window Reduced by the sender

ECN


20-60 bytes
Header
Data
Source Port Address
16 bits
Destination Port Address
16 bits
Sequence Number
32 bits
Acknowledgement Number
32 bits
HLEN
4 bits
Reserved
6 bits
URG
ACK
PSH
RST
SYN
FIN
Window Size
16 bits
Checksum
16 bits
Urgent Pointer
16 bits
Options and Padding
Up to 40 bytes
Reserved 4 bits
CWR
ECE
CWR: Congestion Window Reduced Flag
ECE: ECE Echo

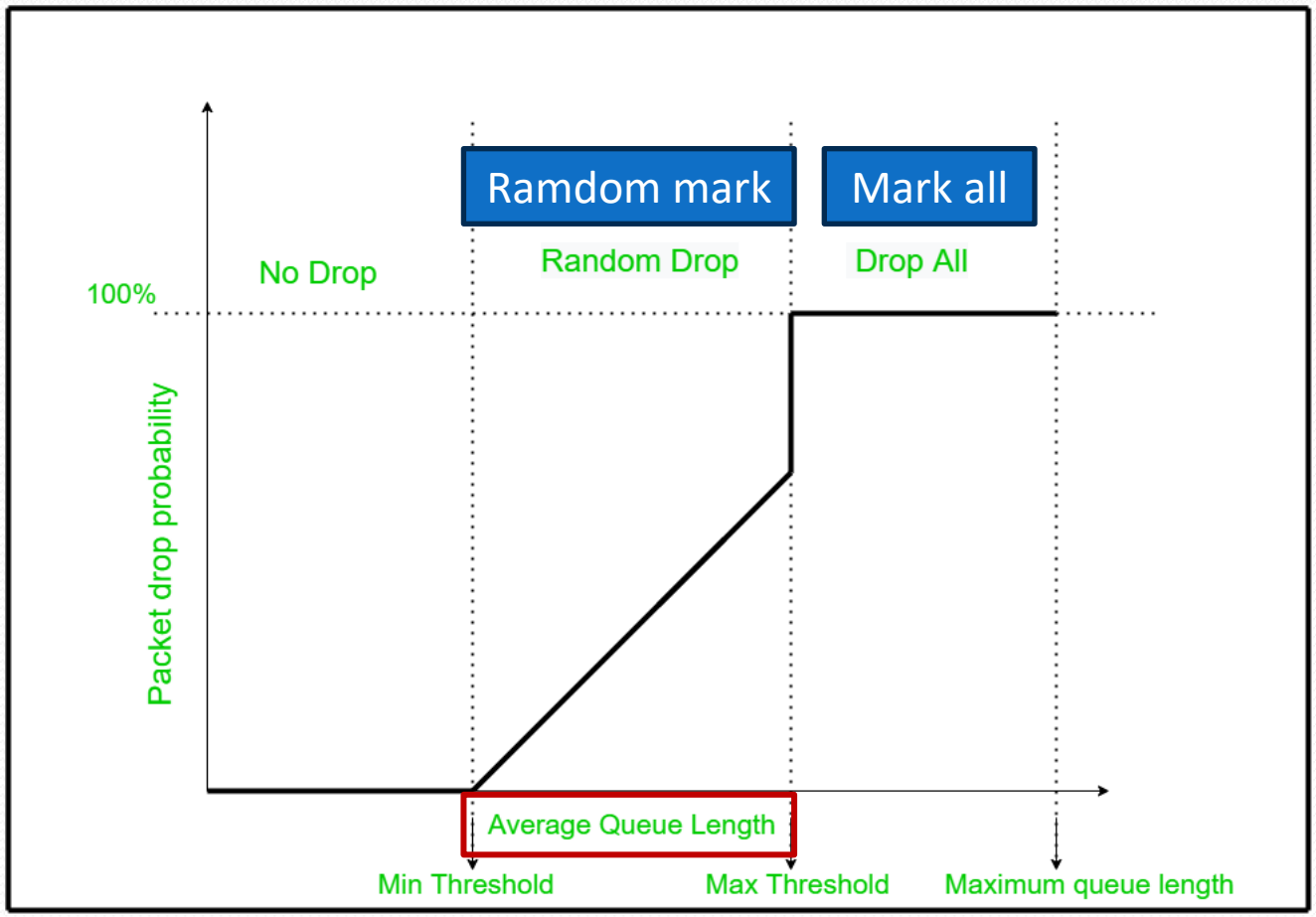
ECN


IPv4
Version
IHL
Type of Service
Total Length
Identification
Flags
Fragment Offset
Time To Live
Protocol
Header Checksum
Source Address
Destination Address
Options
Padding
IPv6
Version
Traffic Class
Flow Label
Payload Length
Next Header
Hop Limit
Source Address
Destination Address
0
1
2
3
4
5
6
7
Precedence
D
T
R
0
0
ECN Bits
0 0 – Non-ECN Capable Transport
0 1 – ECN Capable TransporT
1 0 - ECN Capable Transport
1 1 – Congestion Experienced

ECN

S. No.	ECT	CE	Codepoint	Sent From	To
1.	0	0	non-ECT	any	any
2.	0	1	ECT(1): ECN Capable Transport	sender	receiver
3.	1	0	ECT(0): ECN Capable Transport	sender	receiver
4.	1	1	CE: Congestion Experienced	router	receiver

RED


Ramdom mark
Mark all
No Drop
Random Drop
Drop All
100%
Packet drop probability
Average Queue Length
Min Threshold
Max Threshold
Maximum queue length

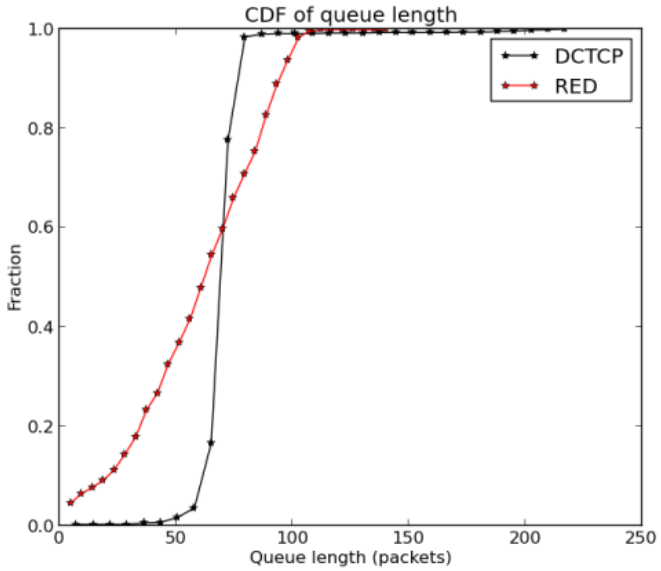
Still Not Good Enough for DCN!

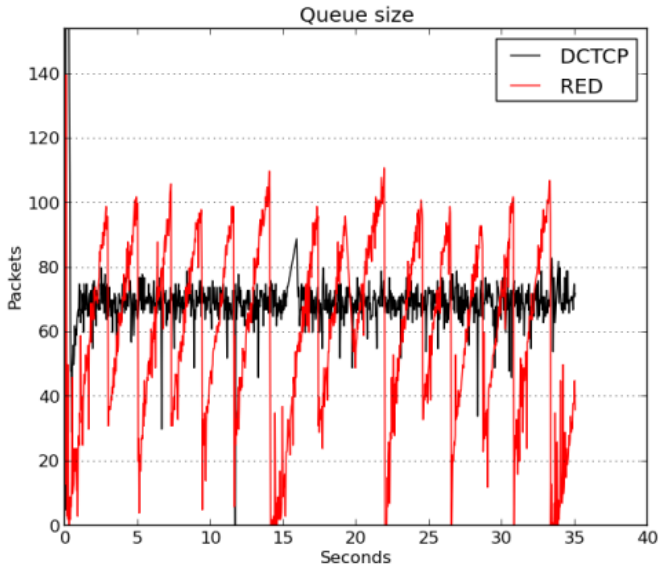
- cwnd = cwnd/2 is too aggressive…
  - Lose too much throughput
- Avg. Queue Length/probabilistic: Not sensitive to **bursty traffic**
  - **Incast**
  - TCP Segmentation Offload (TSO), or Large Send Offload (LSO)

What does DCN require?

- High throughput
- Low latency
- Burst tolerance

Keep the Queue **Low** and **Stable**!





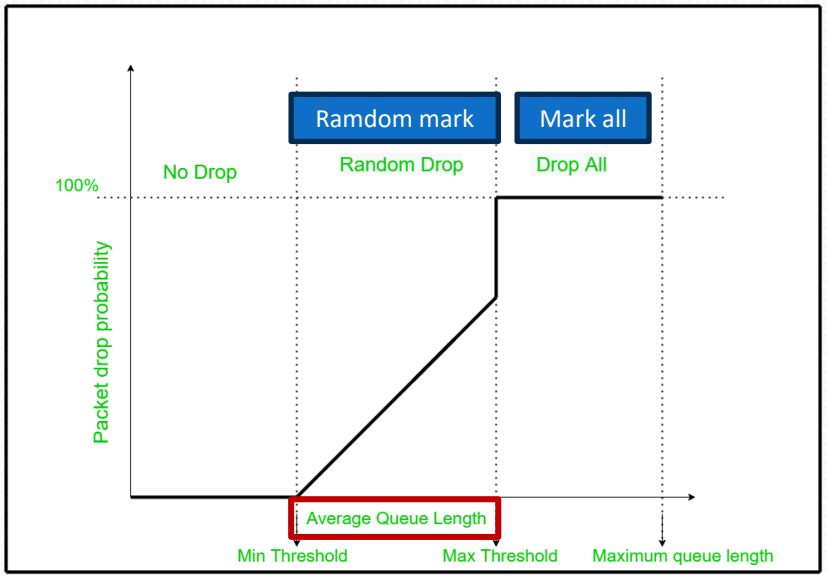
Outline

- Background
- **DCTCP**
- Implementation and evaluation
- Review

Changes to ECN/RED

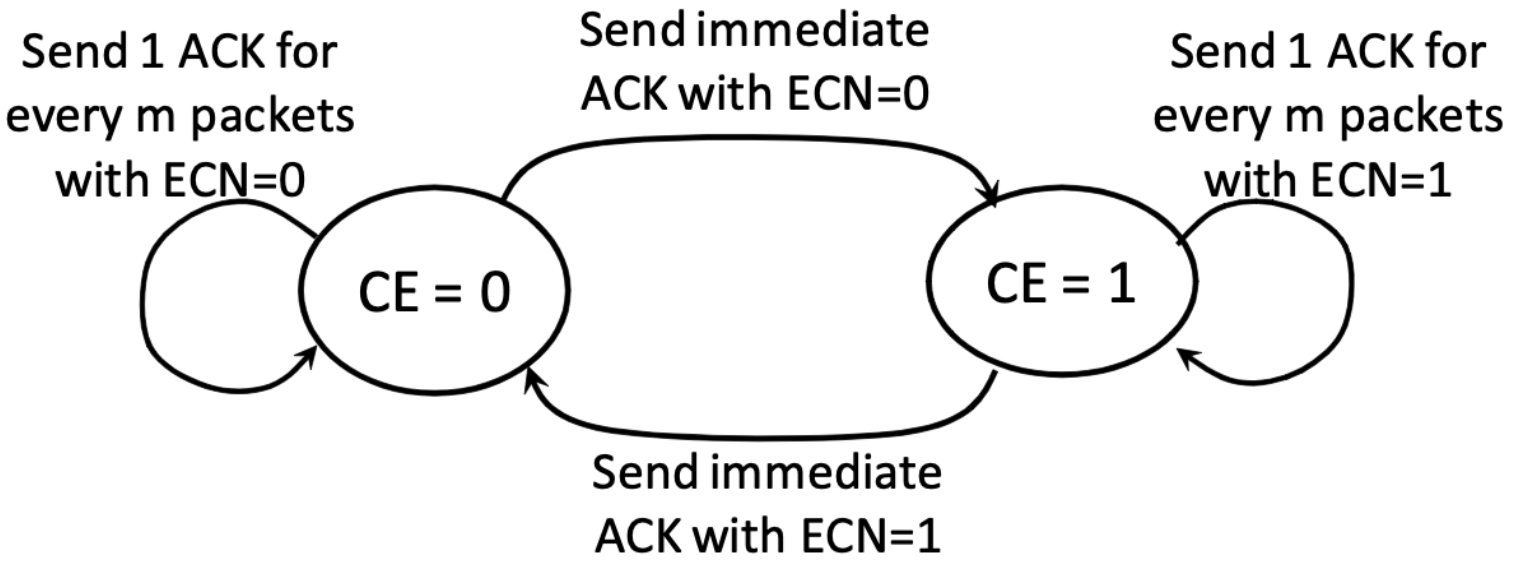
- At the switch side (these changes are supported by all commodity switches)
- **Set Pmin = Pmax = K**
- Marking based on **instantons queue length** not average queue length
- **Good Bursty Tolerance**
- $\boldsymbol{K = \lambda \times C \times RTT}$
- $\boldsymbol{\lambda = \frac{1}{7}}$

Analysis of DCTCP: Stability, Convergence, and Fairness, SIGMETRICS 11



Changes to TCP (Receiver Side)

- Still use accumulative ACK: one cumulative ACK for every m consecutively received packets
- Since the sender knows many packets each ACK covers, it can exactly reconstruct the number of marks seen by the receiver.



Change to TCP (Sender Side)

- The sender maintains an estimate of the fraction of packets that are marked

$$\alpha \leftarrow (1-g) \times \alpha + g \times F,$$

- Updated once for every window of data (roughly one RTT)
- F is the fraction of packets that were marked in the last window of data

Change to TCP (Sender Side)

$$cwnd \leftarrow cwnd \times (1 - \alpha/2).$$

- When α is near 0 (low congestion), the window is only reduced slightly to keep throughput
- When α is near 1 (high congestion), DCTCP fallbacks to original aggressive window cut 1/2

Insights Behind

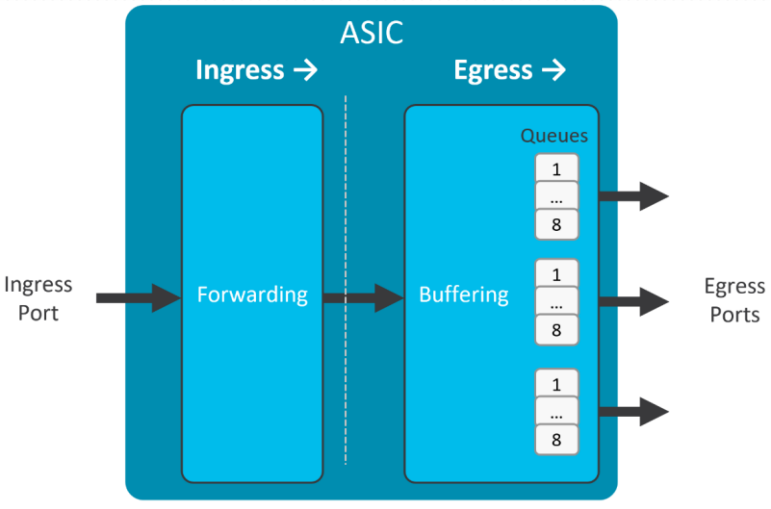
- Original TCP: **0/1 control**
  - Congested: cut the window into half
  - Not congested: keep the window
- DCTCP: **Continuously/fine-grained control**
  - Cut the windows based on the **degree** of congestion

Benefit of DCTCP - Queue Buildup

- DCTCP senders start reacting **as soon as queue length on an interface exceeds K**. This reduces queueing delays on congested switch ports, which **minimizes the impact of long flows on the completion time of small flows**.
- More buffer space is available as **headroom** to absorb transient micro-bursts.

Benefit of DCTCP - Buffer Pressure

- DCTCP also solves the buffer pressure problem because a congested port's queue length does not grow exceedingly large.
- Therefore, in **shared memory switches**, a few congested ports will not exhaust the buffer resources harming flows passing through other ports.



Benefit of DCTCP - Incast

- Cannot completely solve incast problem
  - Extreme Case: If the number of small flows is so high that even 1 packet from each flow is sufficient to overwhelm the buffer on a synchronized burst
- In most cases, flows have several RTTs
- Because DCTCP starts marking early (and aggressively – based on **instantaneous queue** length), DCTCP sources receive enough marks during the first one or two RTTs to tame the size of follow up bursts.

Some Numbers to Remember

- K = **20 packets** for 1Gbps network
- K = **65 packets** for 10Gbps
- Gap between theory and practical system
  - Theory assumes packet goes out in an average way
  - Due to LSO, packet **burst out** from an interface

Summary

TCP	DCTCP
binary signal	multi-bit signal
loss-based	ECN-based
cwnd = cwnd/2	cwnd = cwnd (1 - α/2)
queue oscillation large	queue stable

Outline

- Background
- DCTCP
- **Implementation and evaluation**
- Review

Implementation

- Fully implemented in Linux kernel
  - sysctl -w net.ipv4.tcp_congestion_control=dctcp
  - sysctl -w net.ipv4.tcp_ecn_fallback=0 (optional)
- ns3 simulation from myself
  - https://github.com/snowzjx/ns3-ecn-sharp
  - https://github.com/snowzjx/ns3-ecn-sharp/blob/master/src/internet/model/tcp-dctcp.cc

Evaluation

- Main Evaluation
  - Throughput & Queue length
  - Fairness & Convergence
  - Multi-bottleneck Network
- Microbenchmarks
  - Incast
  - Queue Buildup

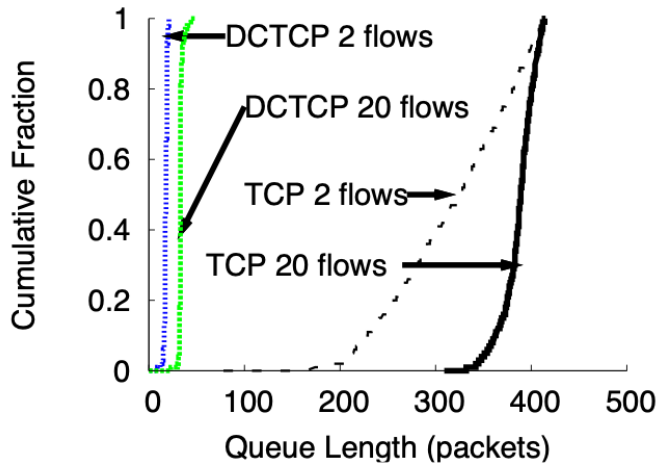


Figure 13: Queue length CDF (1Gbps)

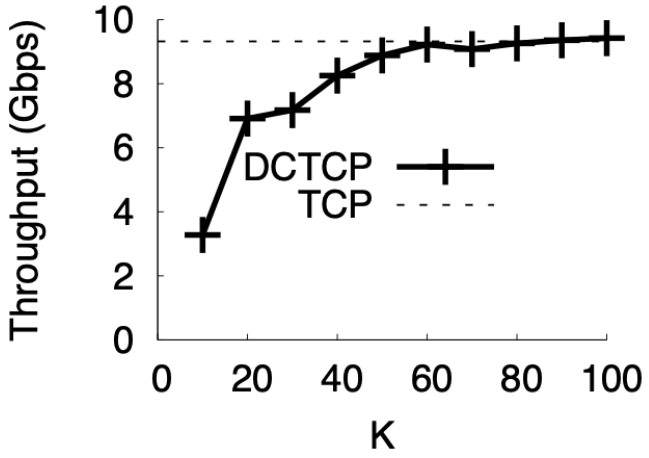
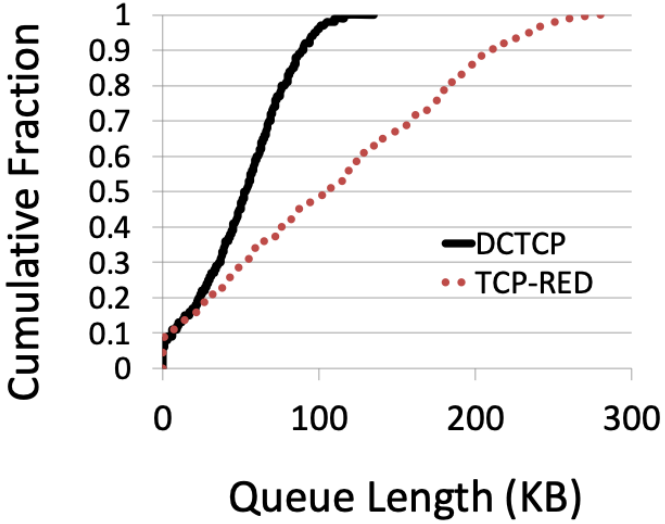
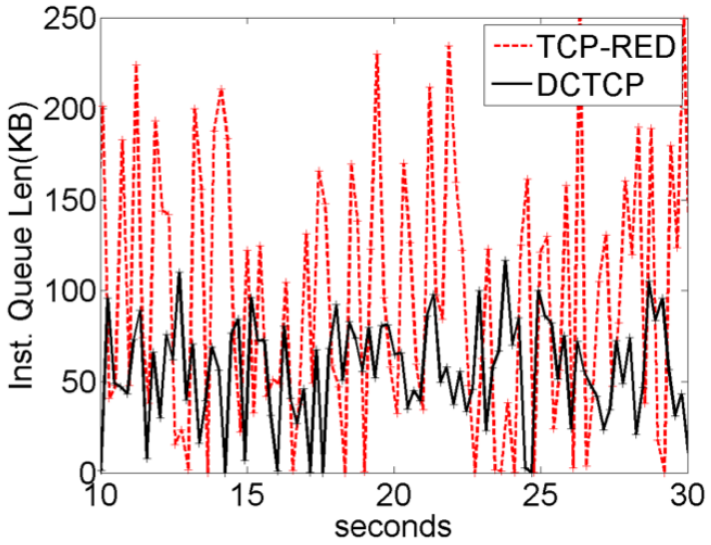


Figure 14: Throughput (10Gbps)





(a) CDF of queue length (b) Time series of queue length

Figure 15: DCTCP versus RED at 10Gbps

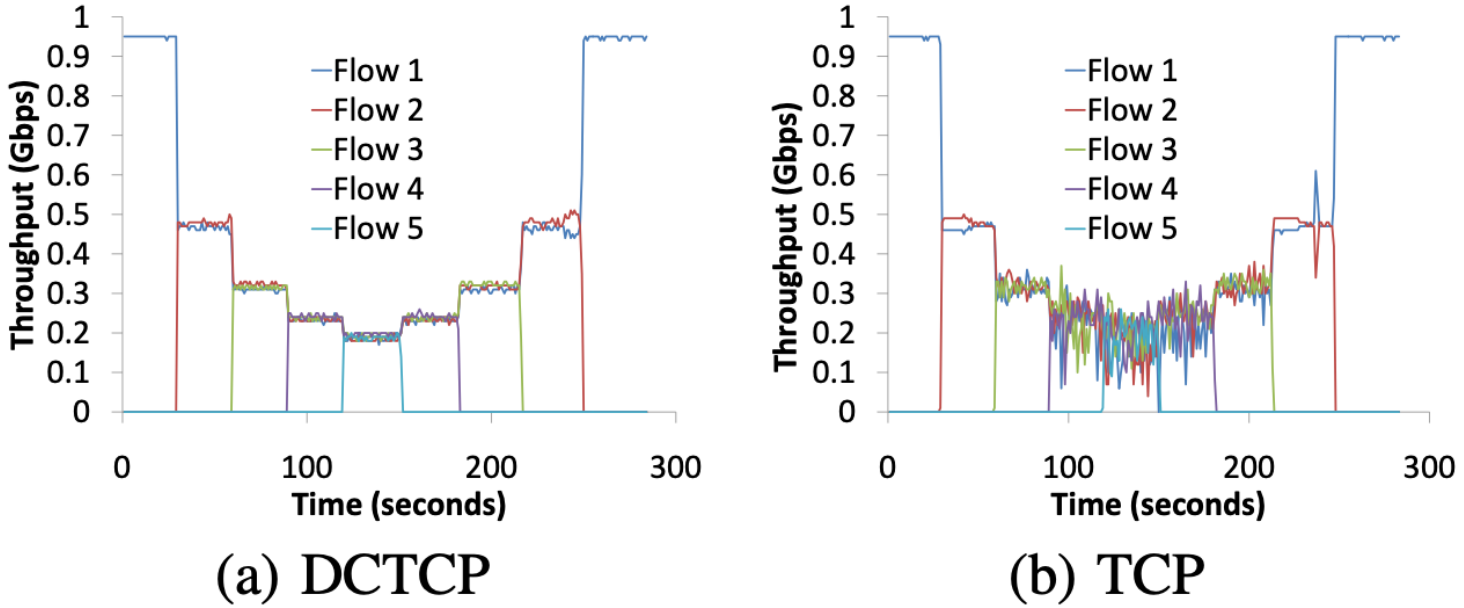


Figure 16: Convergence test

Multi-bottleneck Network

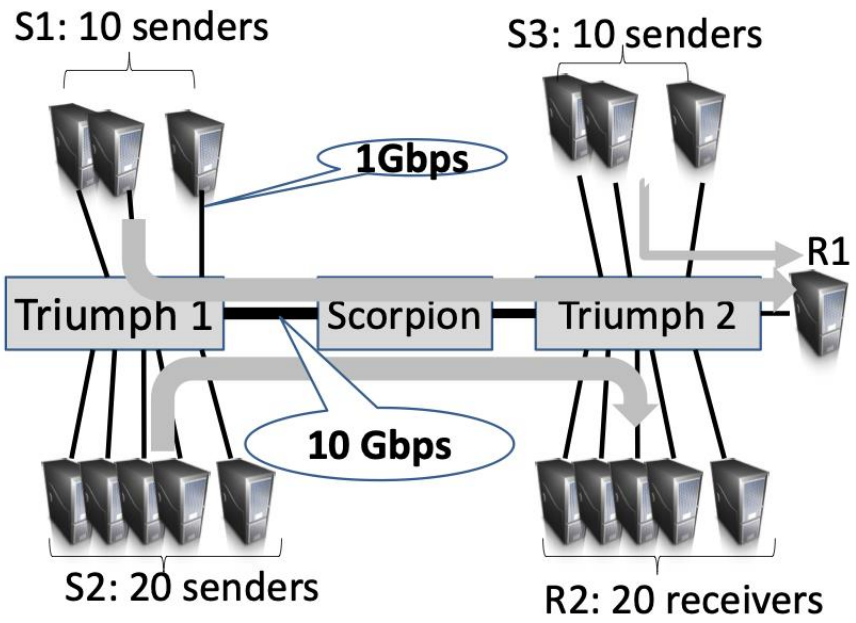


Figure 17: Multi-hop topology

S1 -> R1
S3 -> R1
S2 -> R2

2 bottlenecks:

- The 10Gbps link between Triumph 1 and the Scorpion -> 30Gbps:10Gbps oversubscribed
- The 1Gbps link connecting Triumph 2 to R1 -> 20Gbps:1Gbps oversubscribed

Multi-bottleneck Network

- Each sender in S1 gets 46Mbps and S3 gets 54Mbps throughput, while each S2 sender gets approximately 475Mbps
  - 1Gbps / 20 senders = 50Mbps
  - (10Gbps – 0.5Gbps) / 20 senders = 475Mbps

- Within **10% of their fair-share throughputs.**

Incast

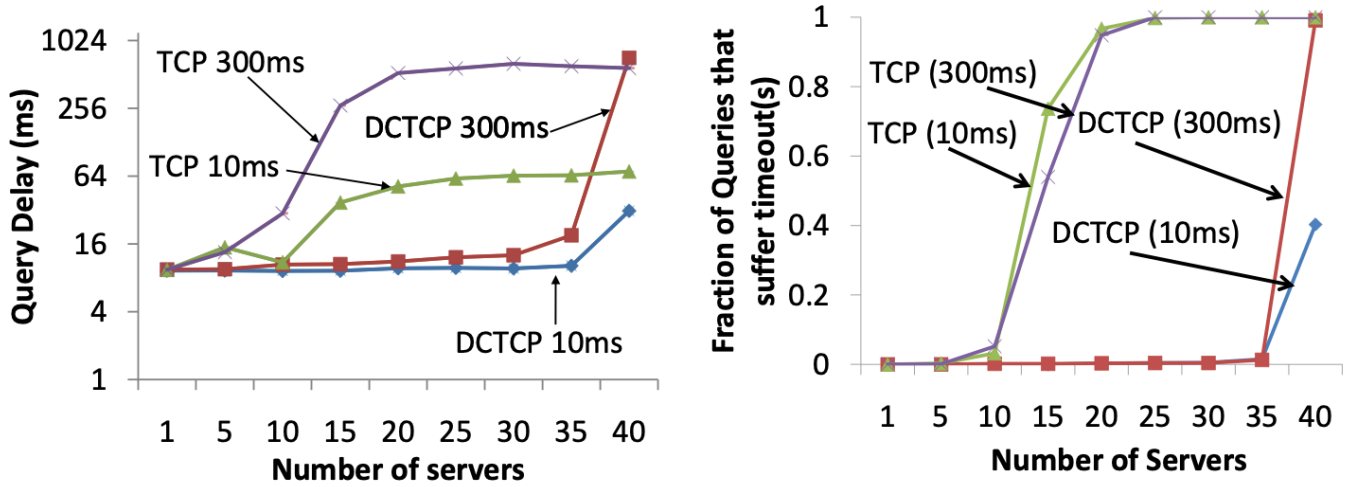


Figure 18: DCTCP performs better than TCP and then converges at 35 senders (log scale on Y axis; 90% confidence intervals for the means are too small to be visible).

Incast

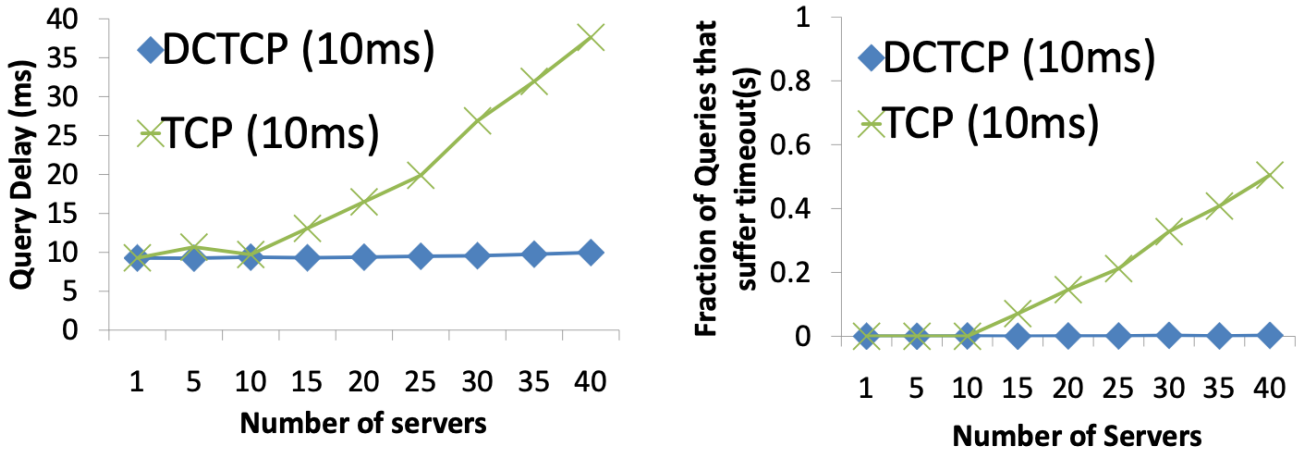


Figure 19: Many-to-one: With dynamic buffering, DCTCP does not suffer problems at even high load.

Queue Buildup

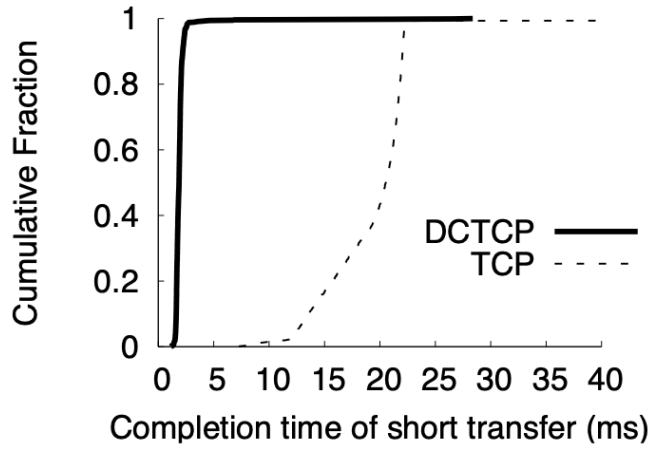


Figure 21: Short transfers see low delay with DCTCP.

A common way to monitor queue length: Use a short flow to **probe**

Outline

- Background
- DCTCP
- Implementation and evaluation
- **Review**

Review

- TCP cannot simultaneously achieve high throughput/low latency and burst tolerance in DCN
- DCTCP employs two key mechanisms
  - Change probabilistic marking into deterministic marking to achieve good burst tolerance
  - Change 0/1 control into continuously control to keep the queue low and stable
- Evaluation shows DCTCP can achieve good performance:
  - High throughput/Low latency
  - Good fairness/Fast convergence
  - Reduce the performance degradation caused by incast (Not completely solved)