Congestion Control in Data Center

Animesh Trivedi
Fall 2019, Period 2

Structure

Core - data centers

- Data center network: architectures
- Data center network: flow scheduling
- **Data center network: congestion control** 👈
- Software defined networking
- Network virtualization
- Network function virtualization
- Programmable data plane
- In-network computing

Agenda

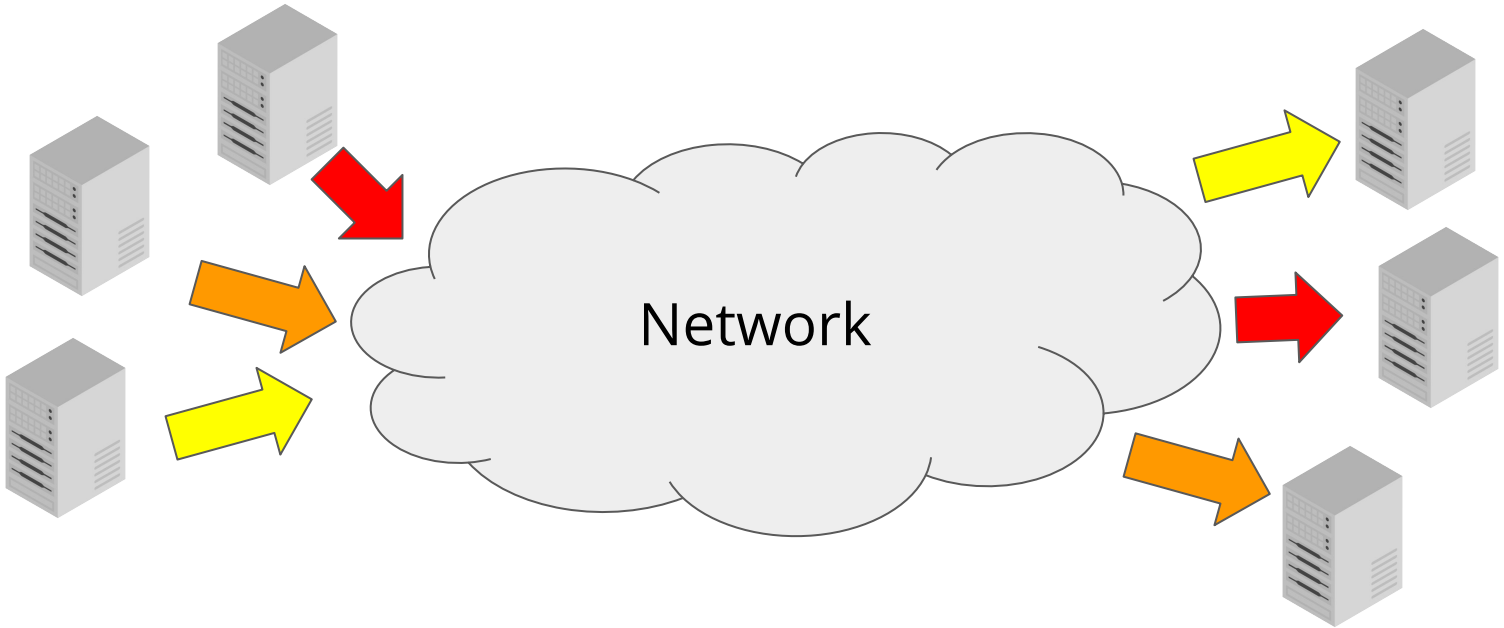
- TCP in data center
  - What is special about data centers
  - The Incast problem
  - Data center TCP (DCTCP)
  - Challenges for TCP
- New non-TCP Transports
  - Infiniband /RDMA networking
  - DC-QCN (Microsoft)
  - TIMELY (Google)
- Summary
- Research directions

What is Unique about Data Center Transport

- Network itself
  - High speed (100+ Gbps), low latency (10-100s of microseconds)
  - No centralized control point
- Diverse applications and workloads
  - Large variety in performance requirements
- Traffic patterns
  - Recall - mouse and elephant flows
  - Scatter gather, broadcast, multicast
- Built out of commodity components
  - Shared switches and their resources (memories, queues, ports)
  - No expensive or customized hardware

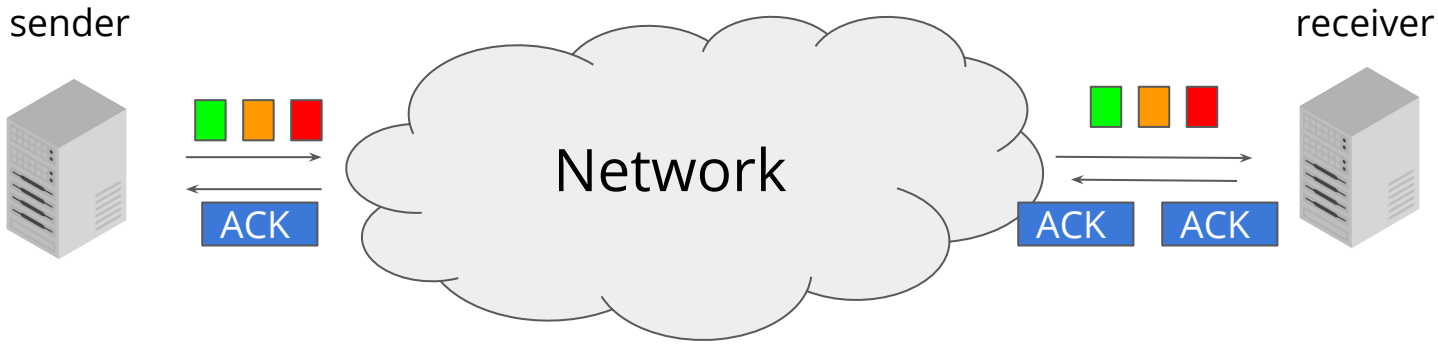
What do we mean when we say congestion control?

Congestion Control



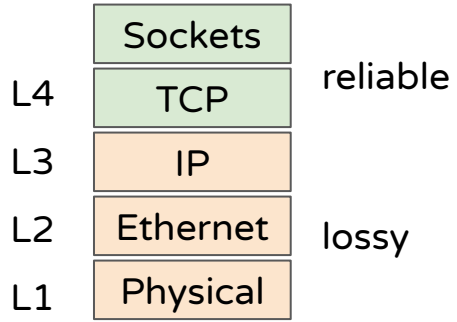
Not to over run network capacity - How do we do it today?

TCP Protocol



The **transport layer** in the network model

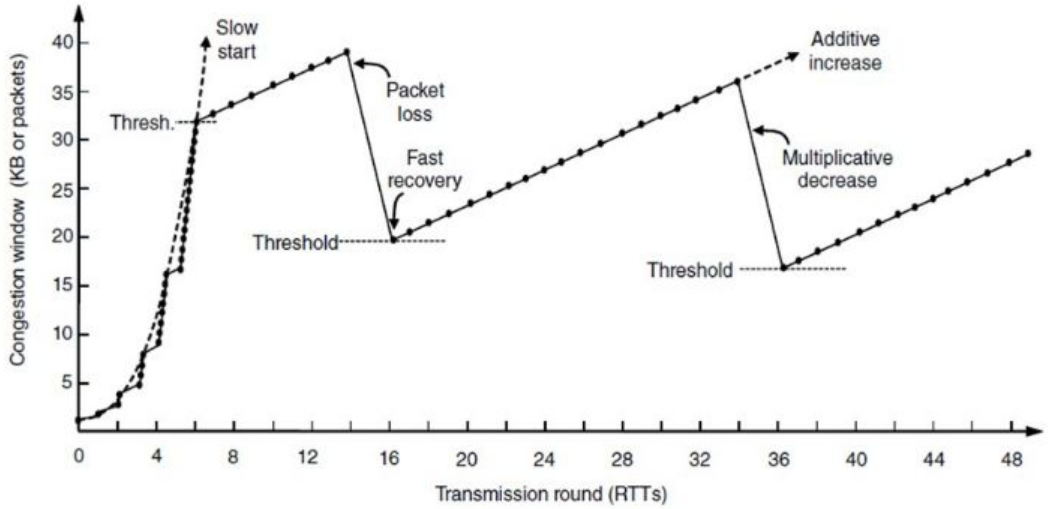
- reliable, in-order delivery using acknowledgements
- make sure not to <u>overrun</u> the receiver (**receiving window, RW**) and the network (**congestion window, CW**)
- what can be sent = **minimum of (RW, CW)**



TCP Congestion Control

TCP has a congestion control mechanism

- Additive increase (ACK)
- Multiplicative decrease (loss)
- Fast recovery (recover)



Computer Networks, Fifth Edition by Andrew Tanenbaum and David Wetherall, © Pearson Education-Prentice Hall, 2011

What could possibly go wrong here?

TCP Incast Problem

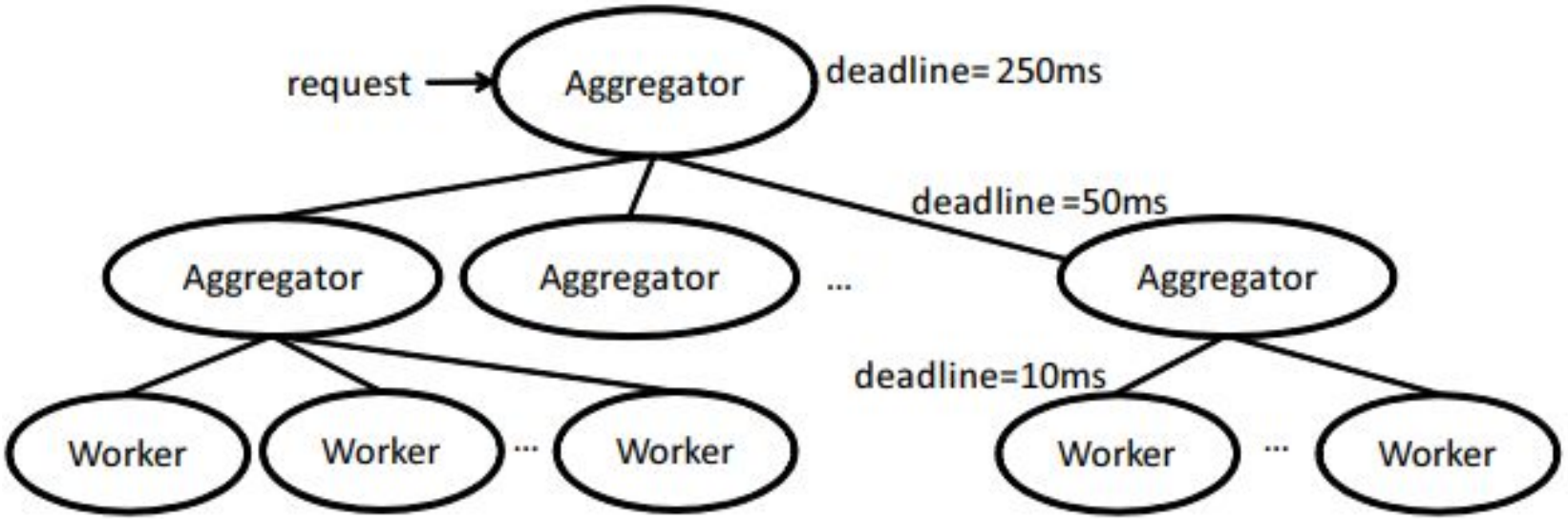
A data center application (storage, cache, data processing - MapReduce) run on multiple servers

They use scatter-gather (or partition-aggregation) work patten

- a client sends a request to a bunch of servers for data [**scatter**]
- all servers respond the the client [**gather**]

More broadly, a client-facing query might have to collect data from many servers

TCP Incast Problem

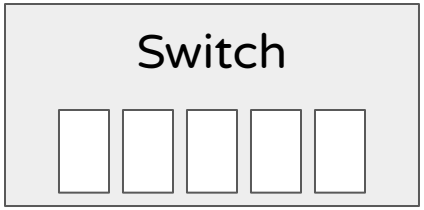


Alizadeh et al., Data Center TCP (DCTCP), SIGCOMM 2010

From a Switch Point of View

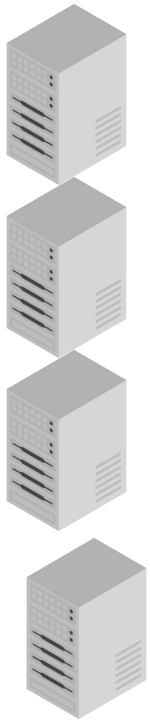
client

Queue capacity 5 packets

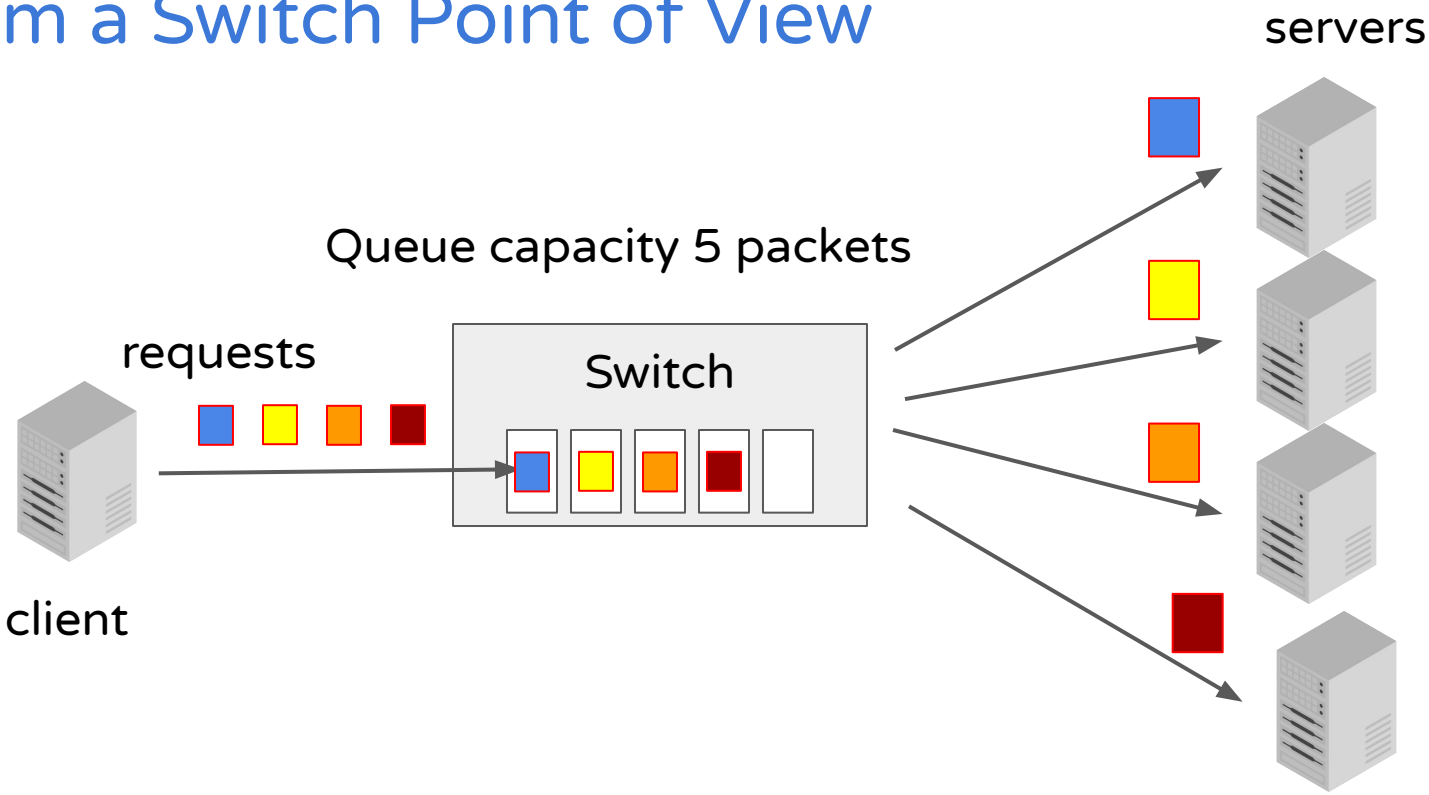


Remember from lecture 1, we are using commodity off-the-shelves switches

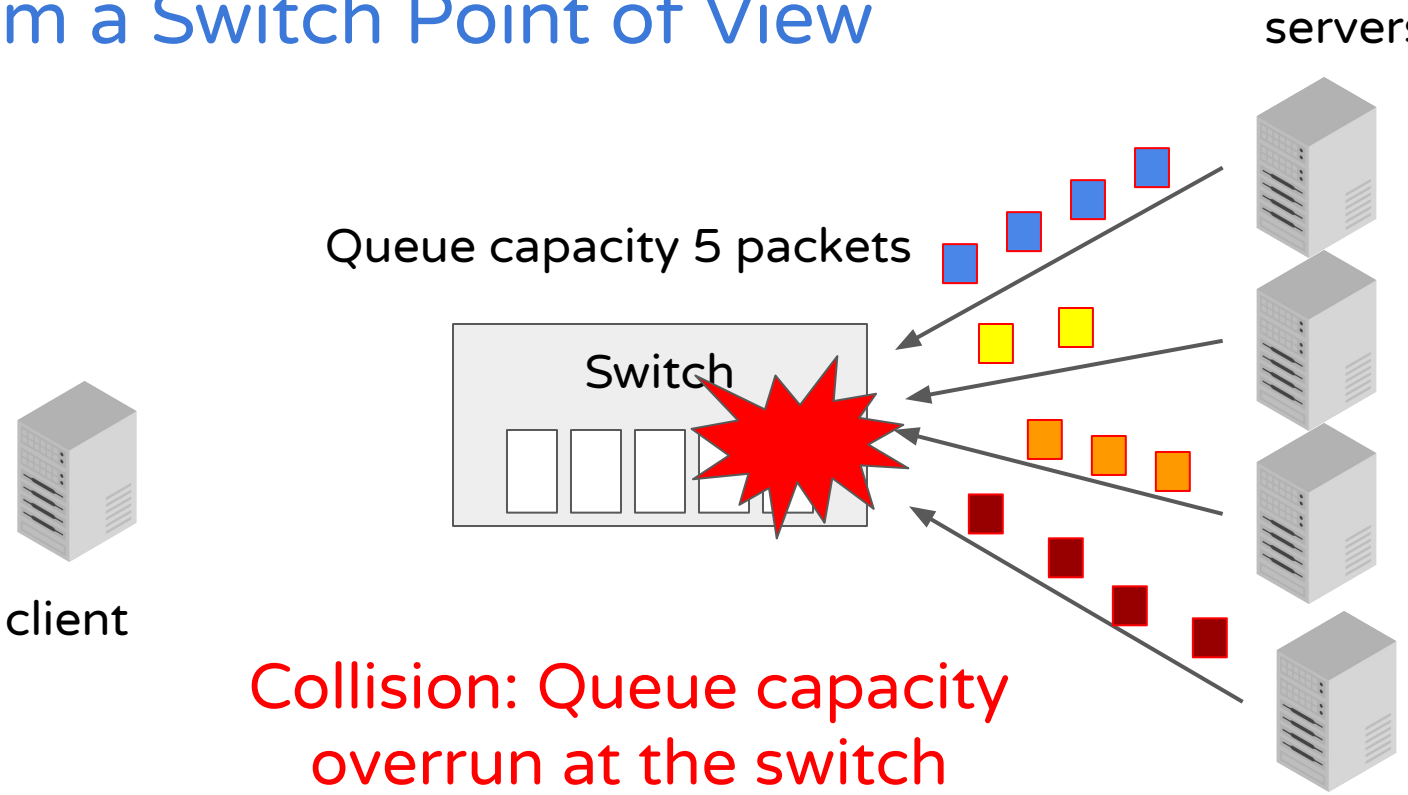
servers



From a Switch Point of View

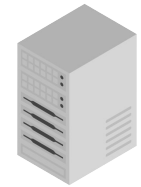


From a Switch Point of View



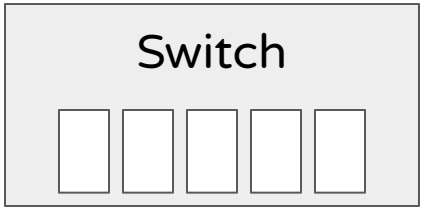
Collision: Queue capacity overrun at the switch

From a Switch Point of View

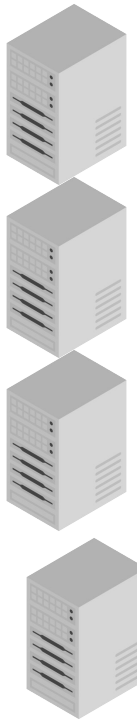


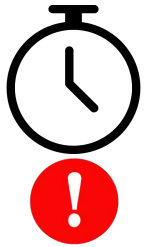
client

Queue capacity 5 packets



servers

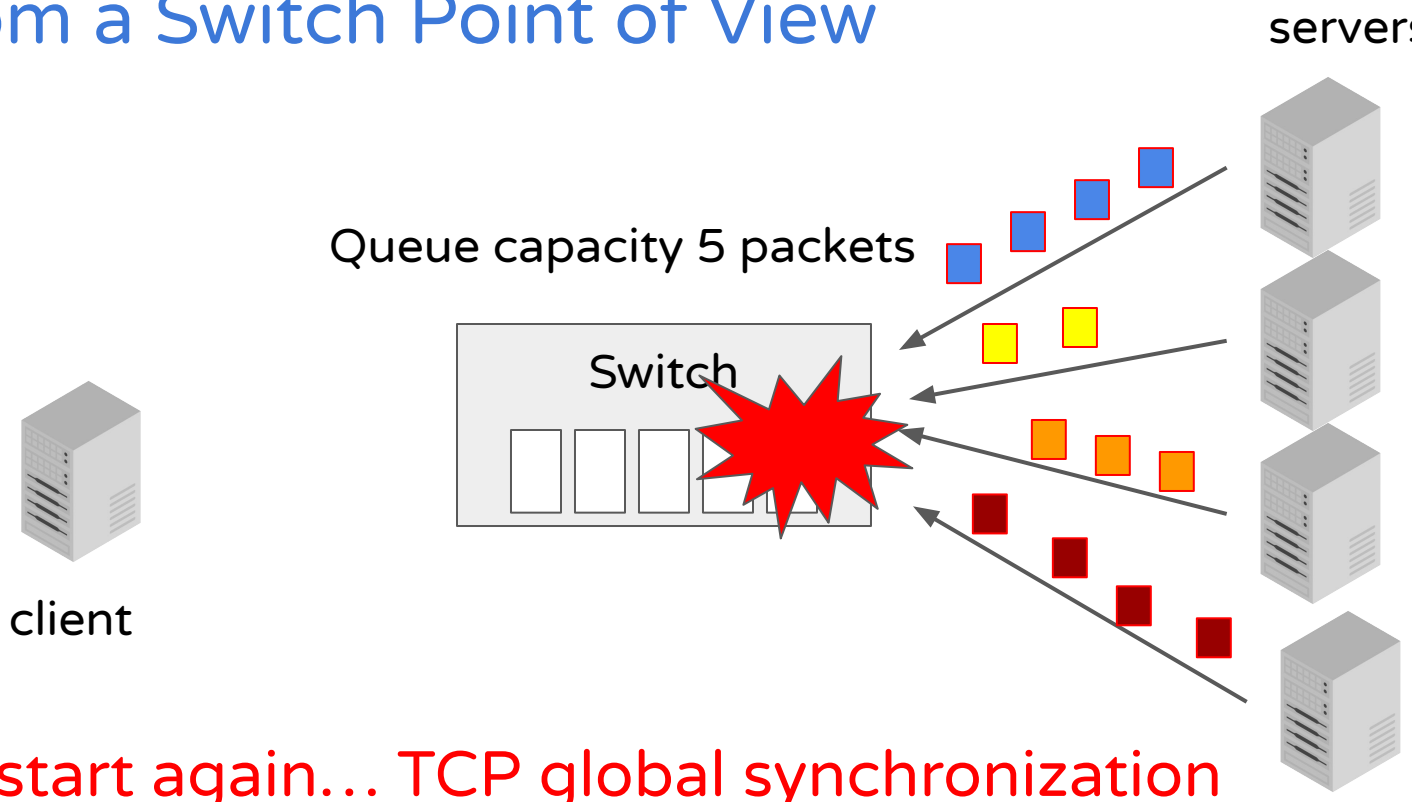




Typically ~100ms

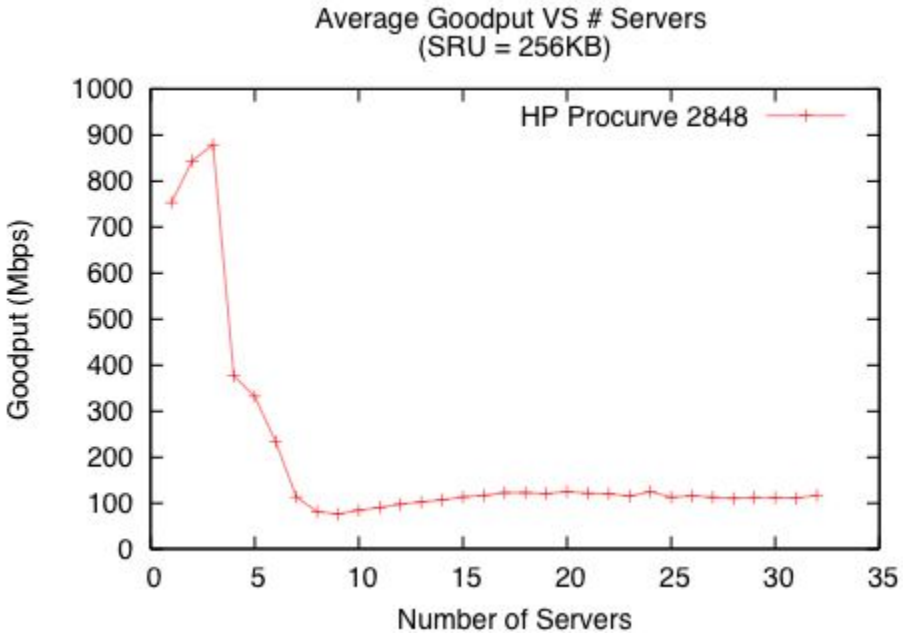
After a timeout, the servers will realize that the packets have been lost

From a Switch Point of View



We start again… TCP global synchronization

TCP Incast Problem



Phanishayee et al., *Measurement and Analysis of TCP Throughput Collapse in Cluster-based Storage Systems*, FAST 2008 [https://www.cs.cmu.edu/~dga/papers/incast-fast2008/].

TCP Incast

Packet drops due to the capacity overrun at shared commodity switches

- can lead to TCP global synchronization and even more packet losses
- the link remains idle (hence, lost capacity and poor performance)
- first discussed in Nagle et al, *The Panasas ActiveScale Storage Cluster*, SC 2004

Some potential solutions

- use lower timeouts
  - can lead to spurious timeouts and retransmissions
  - high operating system overheads
- other variants of TCP (SACKs, RENO) : improve the performance but cannot avoid the basic phenomenon of TCP Incast
- large switch buffer - helps to push the collapse point further, but expensive

Can we do better?

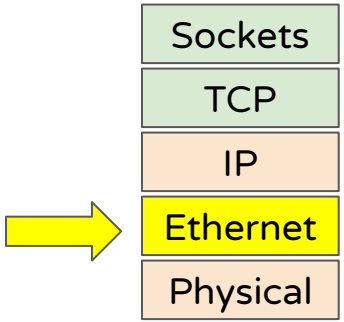
The basic challenge is that there are _only_ limited number of things we can do once a packet is dropped

- various acknowledgements schemes
- various timeouts based optimizations

Whatever clever way you can come up with - imagine deploying that with multiple workloads, flow patterns, and switches …

Can we try to avoid dropping packet in the first place?

Ethernet Flow Control Mechanisms



Pause Frame (IEEE 802.3x)

An overwhelmed Ethernet receiver/NIC can send a “pause” ethernet frame to the sender

Upon receiving the pause frame, the sender stops transmission for a certain duration of time

Limitations:

- designed for end-host NIC (memory, queue) overruns, not switches
- blocks all transmission at the Ethernet-level (port-level, not flow-level)

Priority-based Flow Control (PFC, IEEE 802.1Qbb)

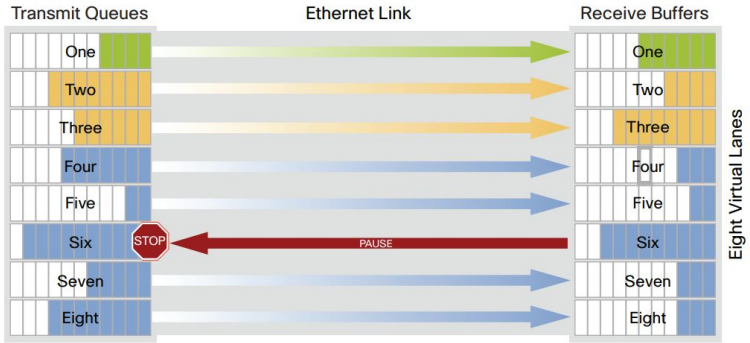
Enhancement over Pause Frames

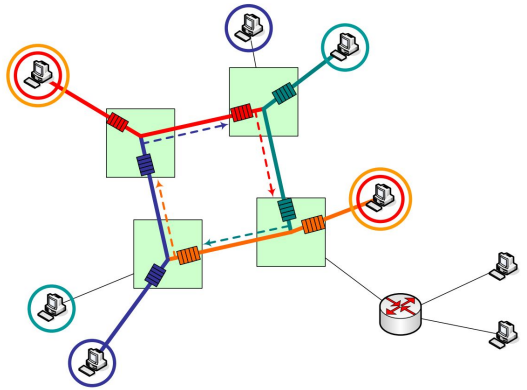
8 virtual traffic lanes, and one can be selectively stopped

Timeout is configurable

Limitations:

- only 8 lanes
- deadlocks in large networks
- unfairness (victim flows)





https://www.cisco.com/c/dam/en/us/solutions/collateral/data-center-virtualization/ieee-802-1-data-center-bridging/at_a_glance_c45-460907.pdf

Data center TCP (DCTCP), SIGCOMM 2010

DCTCP

TCP-alike congestion protocol

The basic idea: pass information about **switch queue building** to to senders

- from **where** to pass information?
- **how** to pass information?

At the sender: re-act to this information by slowing down the transmission

- by **how** much?
- how **many times**?

Explicit Congestion Notification (ECN)

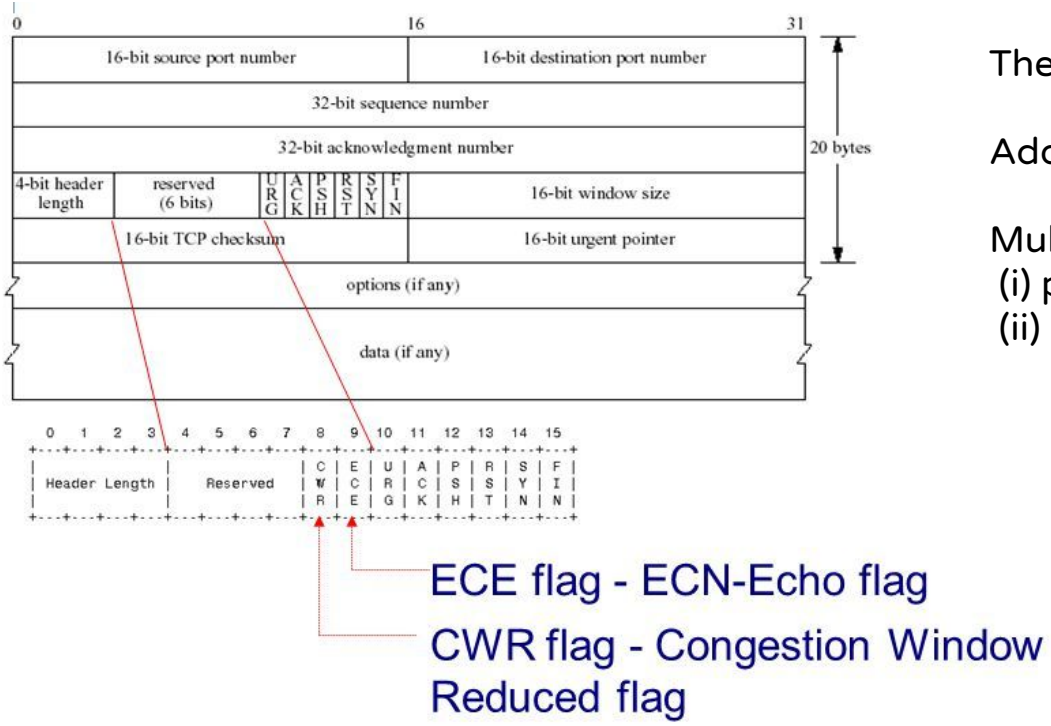
ECN is a standardized way of passing “the presence of congestion”

- part of the IP packet header, uses 1 bits (capability,ack) and 1 bit (indicate, yes/no for congestion) (2 bits in total)
- supported by all commodity switches

Logic: For a queue size of “N” when the queue occupancy go beyond “K”, then mark the passing packet’s ECN bit as set

- there are more sophisticated logics (Random Early Detection, RED) that can *probabilistically* mark packets (see later)

The ECN Bit Location



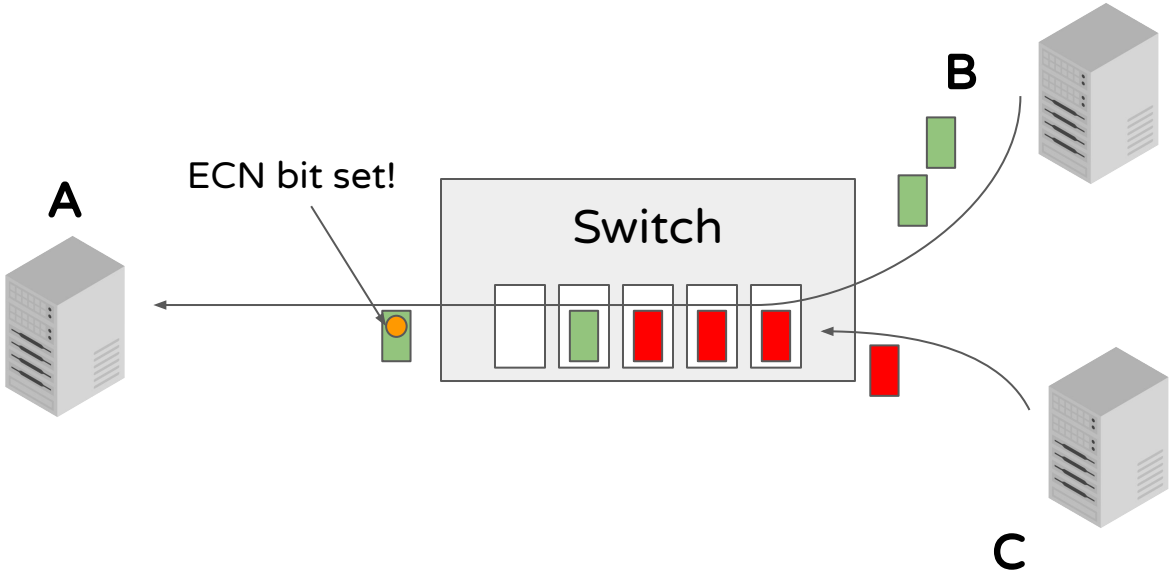
The TCP congestion window logic:

Additive Increase : W -> W + 1 / RTT

Multiplicative Decrease: W -> W/2
(i) packet loss
(ii) a packet received with ECN

https://tools.ietf.org/html/rfc3168 and https://en.wikipedia.org/wiki/Explicit_Congestion_Notification

ECN Bit



Assuming that "**B**" is sending TCP data packets to "**A**"

At some time, "**C**" also starts to send packets, and the queue is getting full

The switch starts to mark packets with "ECN"

How does "B" get to know there was congestion at the switch?

DCTCP Basic Idea

1. Simple marking at the switch : after threshold “K” start marking packets with ECN (instantaneous vs average marking) - uses instantaneous for fast notification

2. **ECN** receiver : mark ACKs with ECN, until the sender ACKs back (the 2nd bit, the CWR flag)
   DCTCP receiver : _only_ mark ACKs corresponding to the ECN packet

3. Sender’s Congestion Control: α (alpha) : estimation of packets that are marked in a running window

DCTCP Congestion Window Calculations

$$\text{each RTT : } F = \frac{\#\text{marked ECN ACKs}}{\#\text{Total Acks}}$$

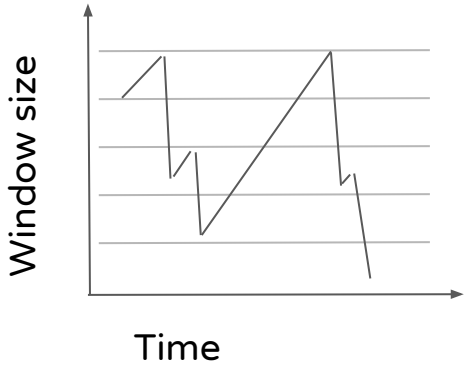
$$\alpha \leftarrow (1 - g)\,\alpha + g \times F \quad \text{(running estimate)}$$

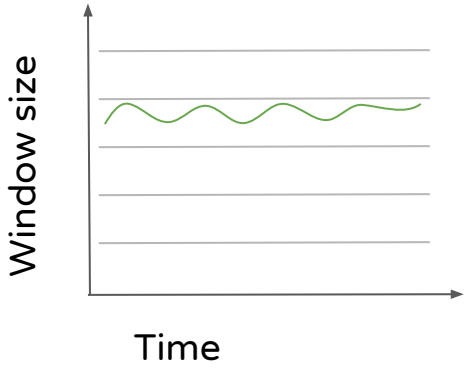
Weightage for past vs present measurements
$0 < g < 1$

$$\text{Congestion window } (cnwd) \leftarrow cnwd \times (1 - \alpha/2)$$

DCTCP vs TCP Example

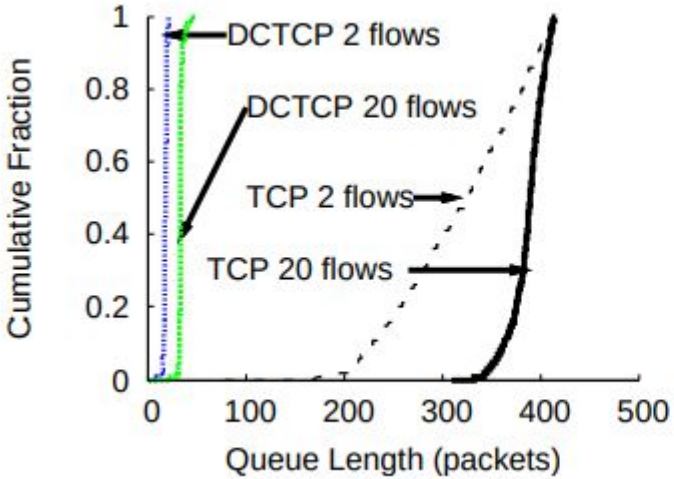
ECN Marks on ACKs	TCP	DCTCP
0 1 1 0 0 0 1 0 0 1	Cut window by **50%** (every time)	Cut window by **40%**
0 0 0 0 0 0 0 0 0 1	Cut window by **50%**	Cut window by **10%**



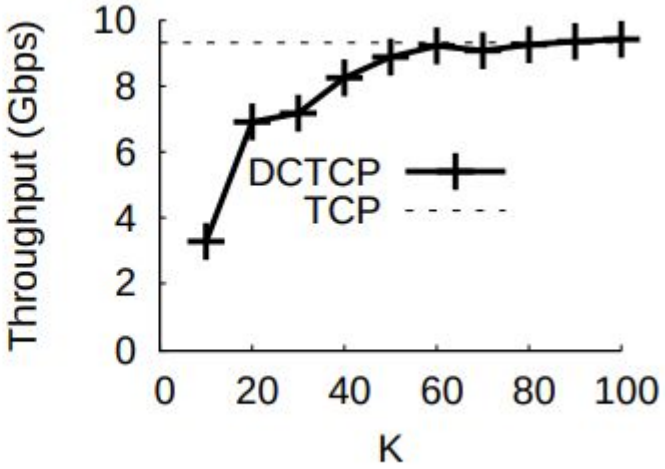


DCTCP - Performance with “K” Marking

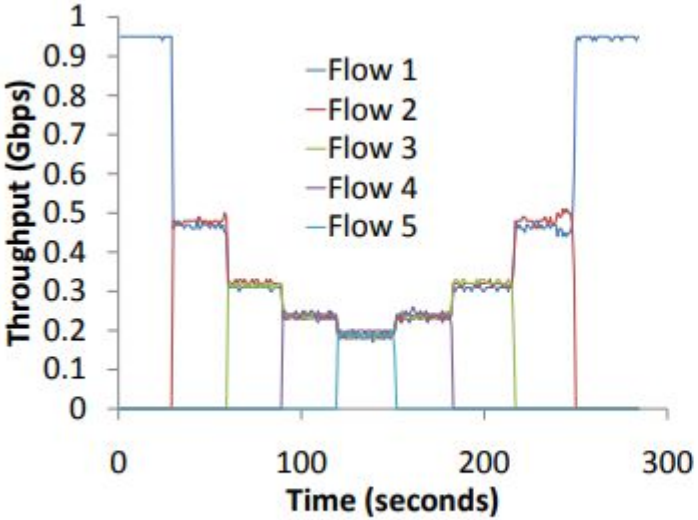
1 Gbps, same bandwidth but lower switch occupancy



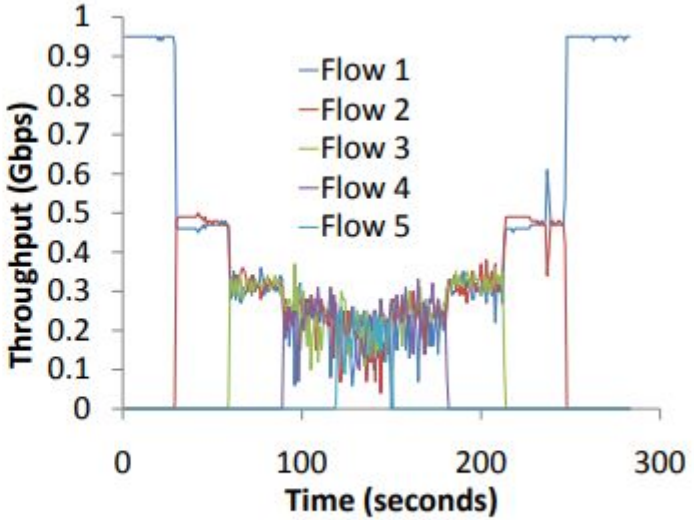
At 10 Gbps, after certain “K” threshold, the same bandwidth



DCTCP - Convergence to Fair Performance

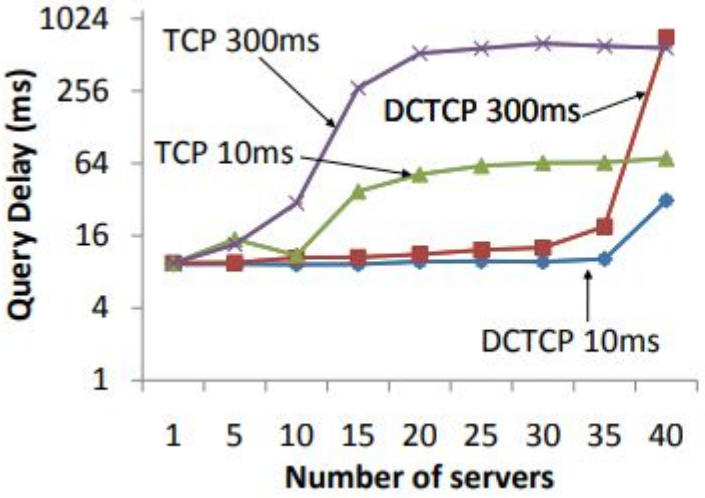


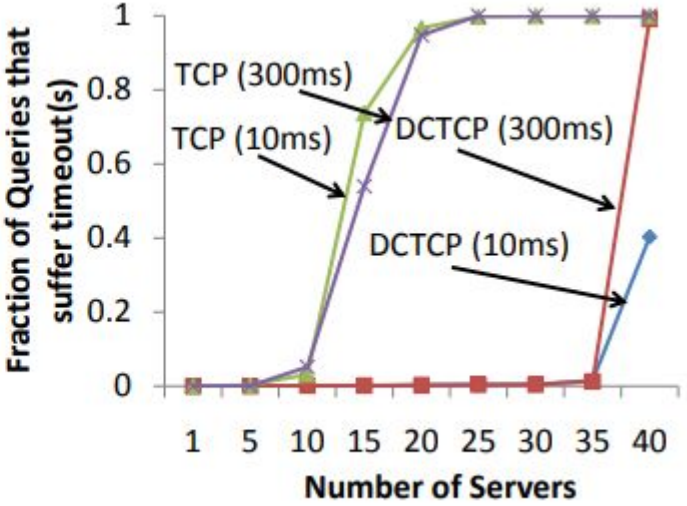
(a) DCTCP



(b) TCP

What about Incast?





Better performance than TCP upto a point where (#35) where not even a single packet can pass from the switch

DCTCP has very low “lost” packets in comparison to TCP

DCTCP is not alone

Multiple projects to improve the basic TCP for data centers

- Deadline aware DCTCP (D2TCP)
- D3: Allocate deadline-proportional bandwidth in routers
- Multipath TCP (mTCP)
- TCP-BOLT…

The general idea is to improve the performance of the TCP protocol for various workloads simultaneously to provide : low latency, high bandwidth, high link utilization, good performance for small bursty flows …

Do we really need TCP

- TCP state machine processing is CPU heavy
  - Network are getting fast, CPU is not
  - Very interesting research topic
- TCP is defined for point-to-point communication, not group communication which is common in data center
- TCP is defined with minimum possible assumptions about the network, like Internet
  - Most of the TCP mechanisms are for "*what to do after a packet loss*"

Can we think of a better way to build a transport network?

Congestion “Avoidance” for **non-TCP Transport** Networks

What are non-TCP Transport Networks

Borrow ideas from High-performance community

Remote Direct Memory Access (**RDMA**) is an end-host API

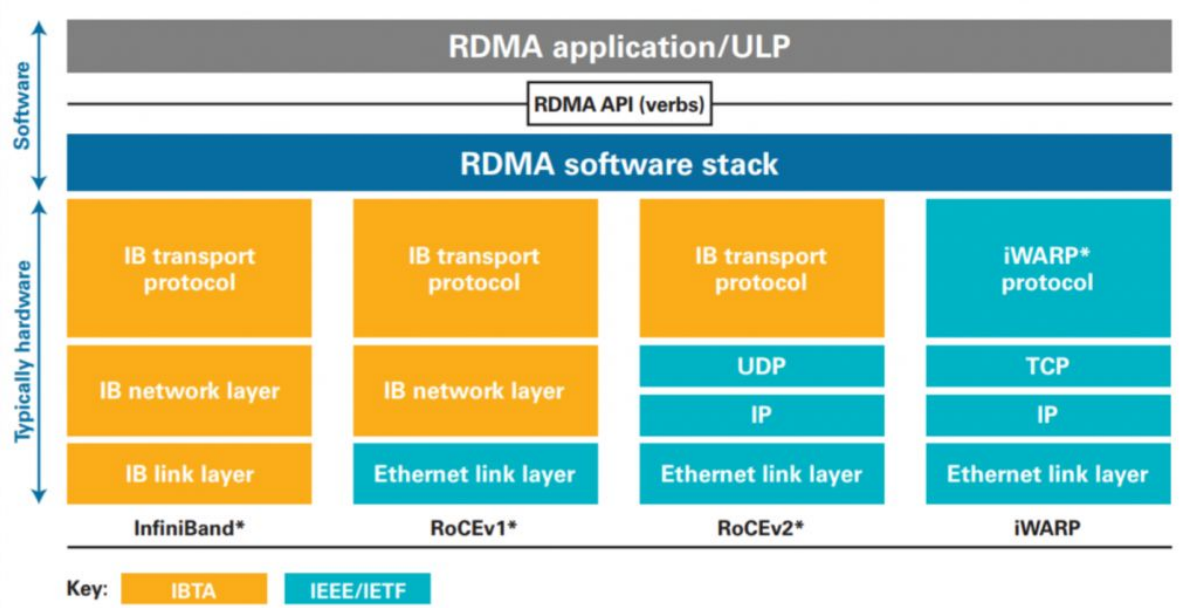
- send/recv network operations (you know it!)
- one-sided remote memory access (just like DMA)
- more sophisticated operations also possible (atomics, locks)

The API is

- operation and message oriented on TX/RX queues
- various reliable/unreliable connectivity options

The use of RDMA in data center applications is a very active research area

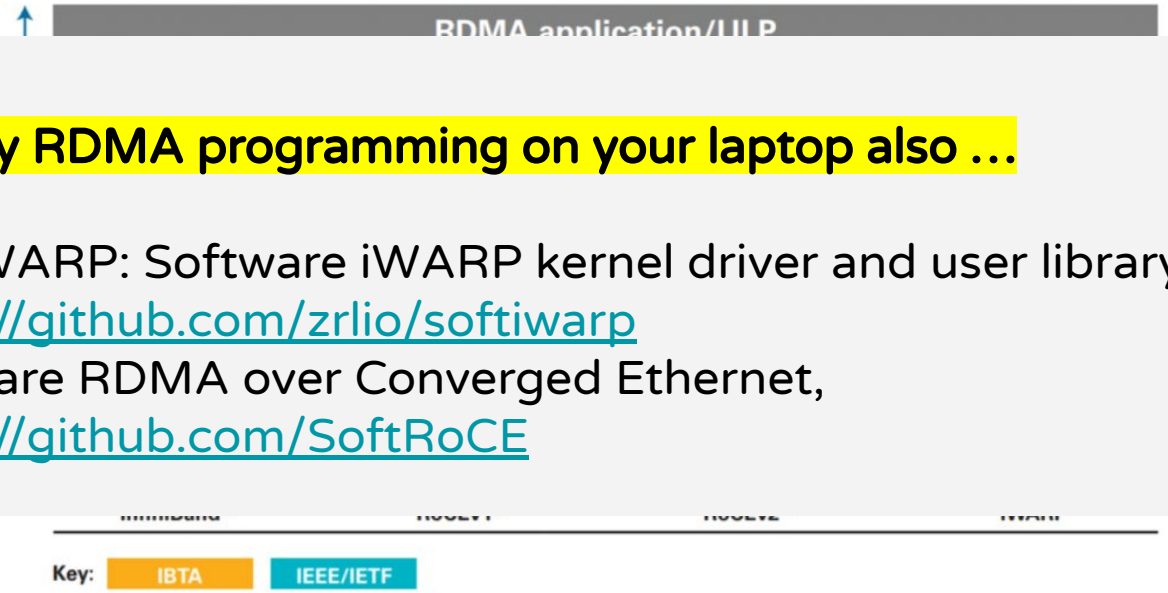
RDMA Enabled Transports



A Survey of End-System Optimizations for High-Speed Networks, ACM Computing Surveys (CSUR) Surveys Homepage archive Volume 51 Issue 3, July 2018. https://dl.acm.org/citation.cfm?doid=3212709.3184899

Image reference: https://fakecineaste.blogspot.com/2018/02/

RDMA Enabled Transports



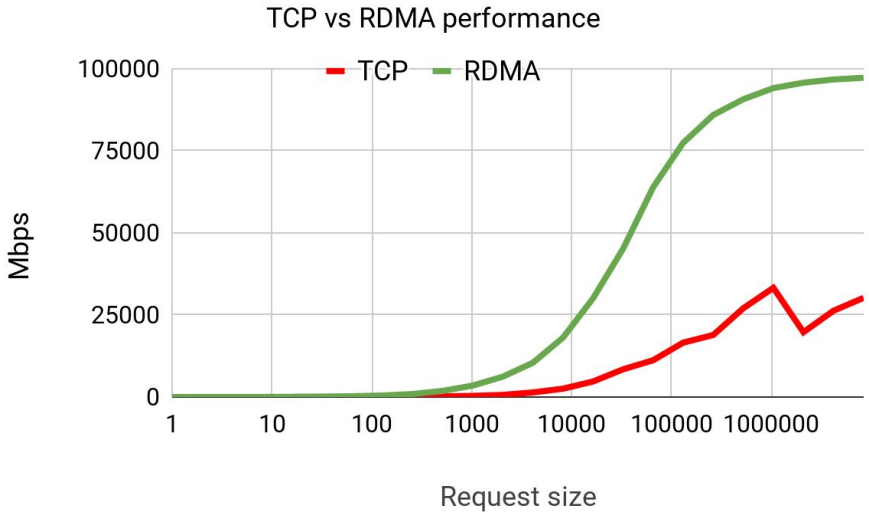
You can try RDMA programming on your laptop also …

1. SoftiWARP: Software iWARP kernel driver and user library for Linux https://github.com/zrlio/softiwarp
2. Software RDMA over Converged Ethernet, https://github.com/SoftRoCE

A Survey of End-System Optimizations for High-Speed Networks, ACM Computing Surveys (CSUR) Surveys Homepage archive Volume 51 Issue 3, July 2018. https://dl.acm.org/citation.cfm?doid=3212709.3184899

Image reference: https://fakecineaste.blogspot.com/2018/02/

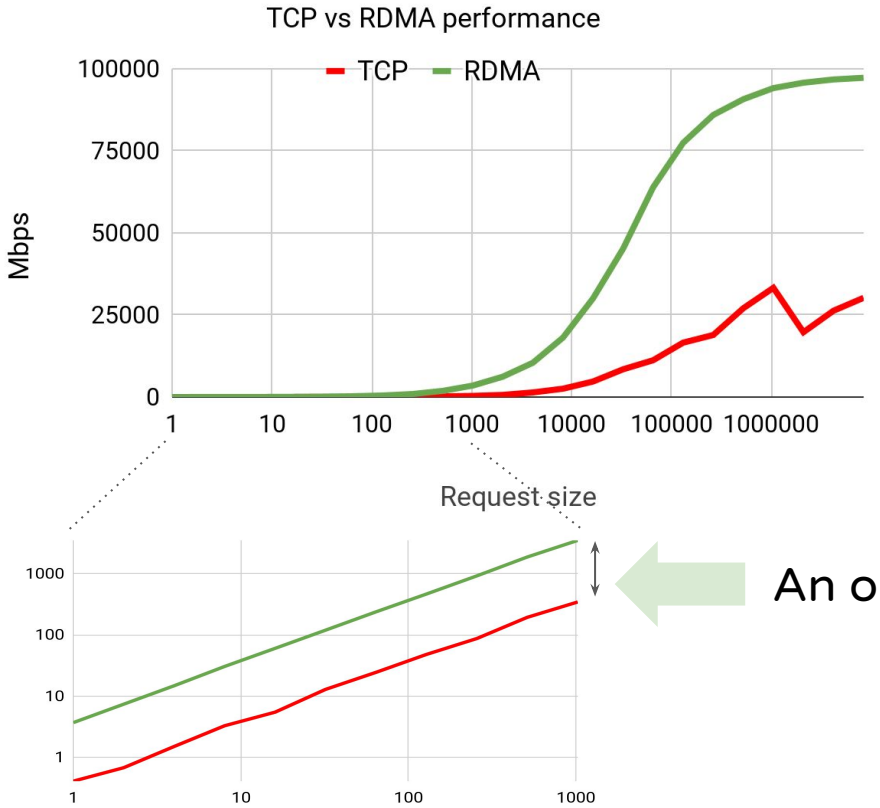
RDMA Performance



97.2 Gbps

33.2 Gbps

RDMA Performance

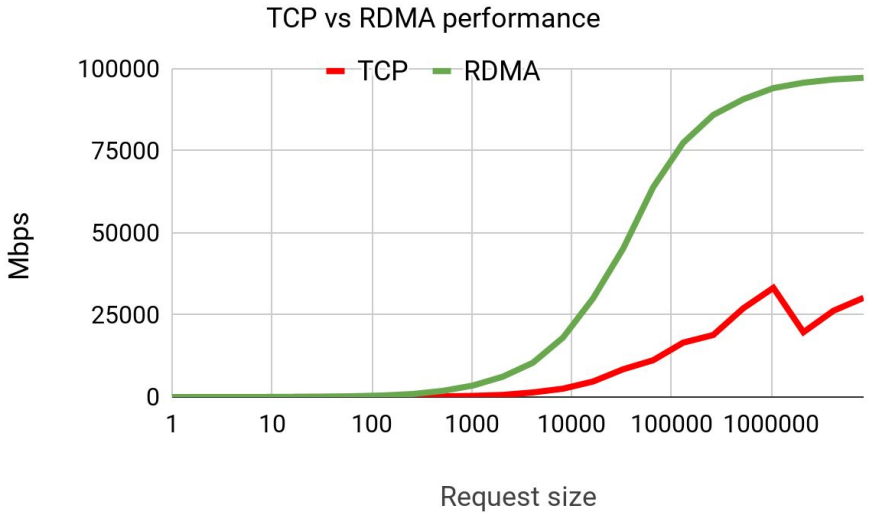


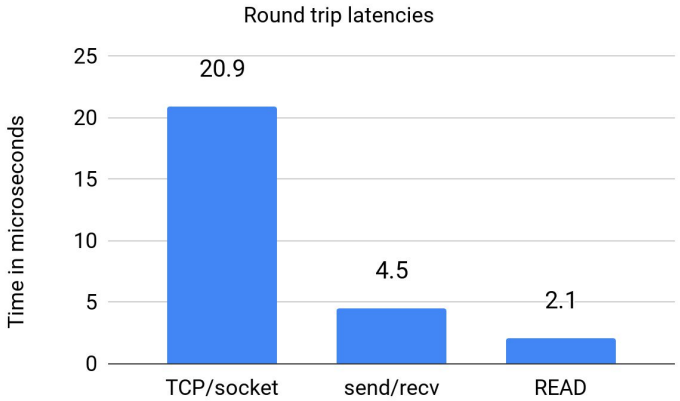
97.2 Gbps

33.2 Gbps

An order of magnitude gap for small requests

RDMA Performance





4.6x

10x

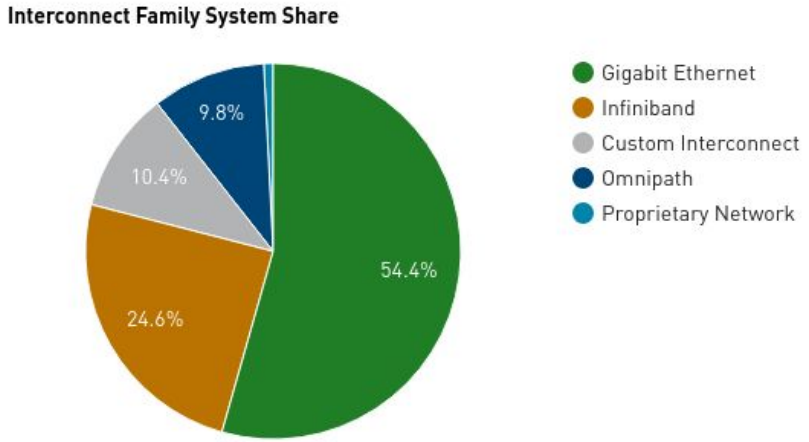
InfiniBand (IB)

Very successful network design for delivering high-performance

41 out of top 100 supercomputers use it

Is designed for ultra low latencies (< 1 usec) and 100/200 Gbps bandwidths

Whole network stack is optimized for delivering performance



Uses **virtual lanes** and **credit based flow control** for loss-less packet delivery!

https://www.slideshare.net/masonmei37/designing-cloud-and-grid-computing-systems-with-infiniband-and-highspeed-ethernet

Virtual Lanes (VLs)

Each switch has multiple virtual channels per physical channel /port

- between 2 to 16
- each virtual channel contains separate buffer space and flow control
- two types of packets within the link layer, **management and data packets**

A head or sender must acquire two resources before forwarding / sending

- a virtual channel on the next switch (including buffer space)
- channel bandwidth at the switch
- encoded in **Credits**

https://www.slideshare.net/masonmei37/designing-cloud-and-grid-computing-systems-with-infiniband-and-highspeed-ethernet
http://www.ieee802.org/1/files/public/docs2014/new-dcb-crupnicoff-ibcreditstutorial-0314.pdf

Credit-based Flow Control

Uses a credit-based flow control mechanism per virtual lane

Each receiving end of a link supplies credits to the sending device on the link to specify the amount of data that can be received without loss of data.

Switch keeps count of number of free buffers per downstream switch (credits)

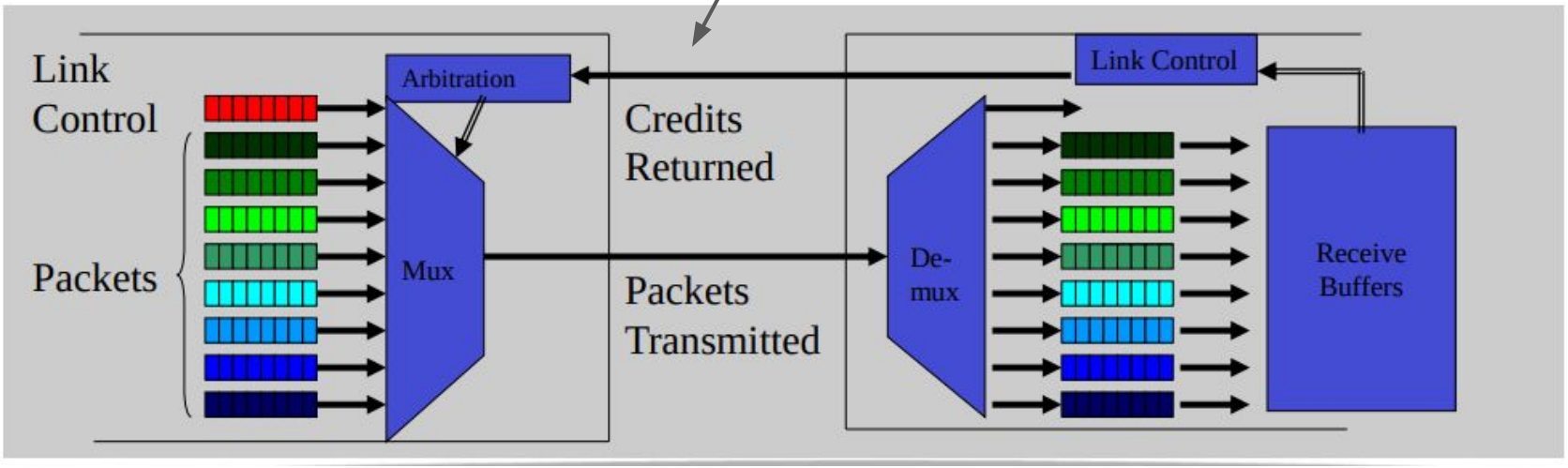
Counter decreased when sending at downstream switch

Stop sending when counter reaches zero

Downstream switch sends back signal to increment credit counter when buffer is freed (forwarding)

InfiniBand

Separate lanes for management and credit passing (Ethernet does not have that!)



https://www.hpcadvisorycouncil.com/pdf/Intro_to_InfiniBand.pdf

InfiniBand Overview

Infiniband delivers very high performance, but

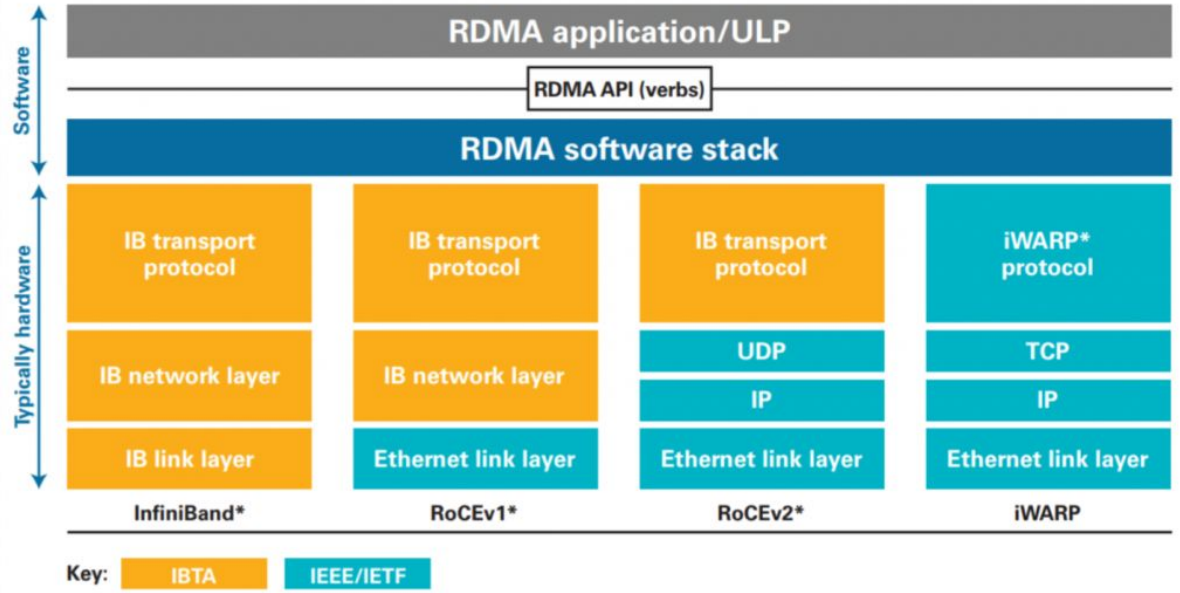
- different (!compatible) network and link layers
- different networking interface and API
- different naming and addressing mechanisms
- different tools
- different cables and NICs

But commodity data center uses Ethernet, and IP - we know how to use them from last 30 years

Hard to convince datacenter operators to deploy another technology - complexity management

https://www.informatix-sol.com/docs/EthernetvInfiniBand.pdf

RDMA Enabled Networks



A Survey of End-System Optimizations for High-Speed Networks, ACM Computing Surveys (CSUR) Surveys Homepage archive Volume 51 Issue 3, July 2018. https://dl.acm.org/citation.cfm?doid=3212709.3184899

Image reference: https://fakecineaste.blogspot.com/2018/02/

RDMA on Ethernet (RoCE)

RoCE uses the very simple IB transport layer (L4) over …

- **v1** used IB networking (L3) on Ethernet (L2) (deprecated)
- **v2** replaces IB networking (L3) with IP and UDP
  - IP for routing, and UDP for ECMP

That means, L2 must provide a reliable packet delivery

Lossless or converged-enhanced Ethernet

- can use PFC, but poor performance, deadlocks, congestion spreading
- only port/priority-based, we need more fine-grained (**per-flow**)

Congestion Control Large Scale RDMA Deployments, SIGCOMM 2015 [DC-QCN]

DC-QCN

DCQCN is a **rate-based**, **end-to-end, lossless** congestion management protocol (mix of PFC, DCTCP, and QCN)

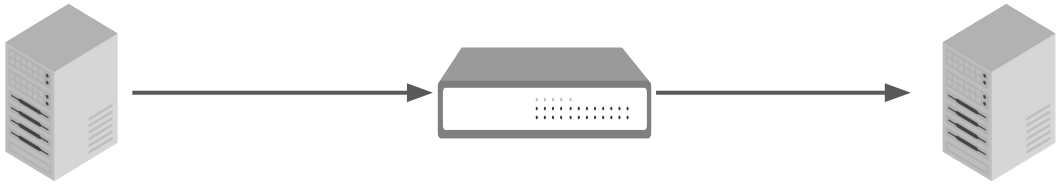
PFC for zero-drop network - STOP, RESUME frames to immediately stop TX

What is QCN? Quantized Congestion Notification (802.1Qau)

- **L2 flow-level** congestion mechanism
- on each packet arrival a congestion metric (quantized value) is calculated
- the switch probabilistically sends this value back to the sender - who can adjust its flow rates accordingly (similar to 802.1Qau)

Why QCN alone is not sufficient?

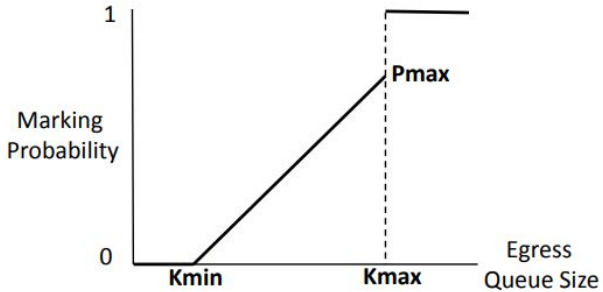
DC-QCN Basic Idea



Sender or
Reaction Point (**RP**)

Congestion Point or
CP

Receiver or
Notification Point (**NP**)

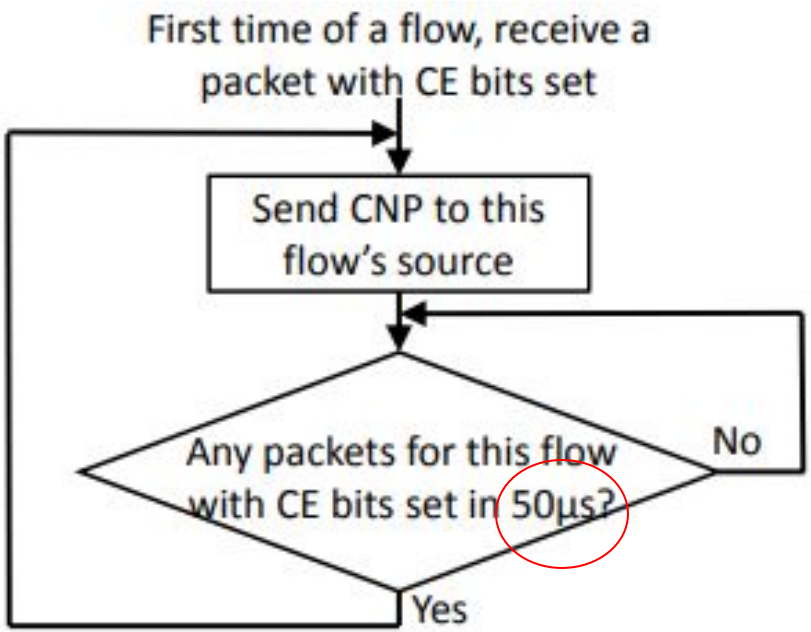


CP runs a *similar* mechanism as DCTCP : *probabilistically* mark ECN-bit

NP sends back a special **Congestion Notification Packet** (CNP) when it receives ECN-marked packets

RP rate adjusts the speed of sending **flow** in a *similar* manner as QCN

Receiver's Algorithm

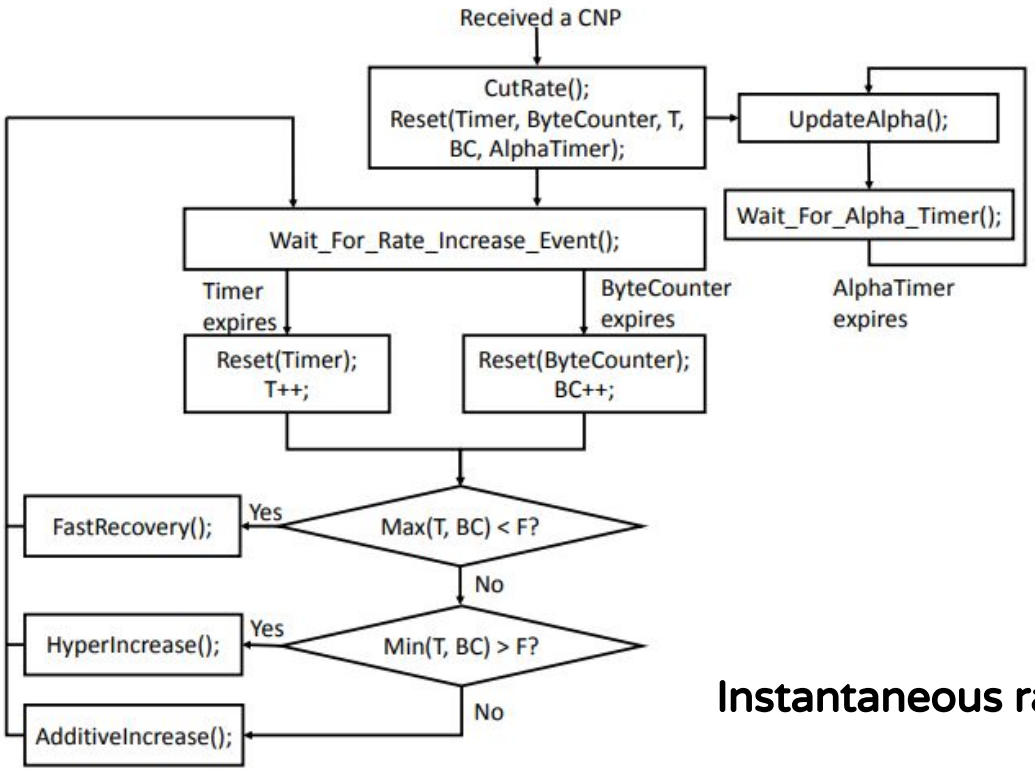


If a new marked packet arrives

- CNP is sent in last 50 usec
  - then no sending
- CNP is not sent in last 50 usec
  - send a new CNP

CNP sending is an expensive operation, so at most one CNP per "N" microseconds

Sender's Algorithm



R_T = target rate, R_C = current rate

$$\begin{aligned} R_T &= R_C, \\ R_C &= R_C(1-\alpha/2), \\ \alpha &= (1-g)\alpha + g, \end{aligned} \tag{1}$$

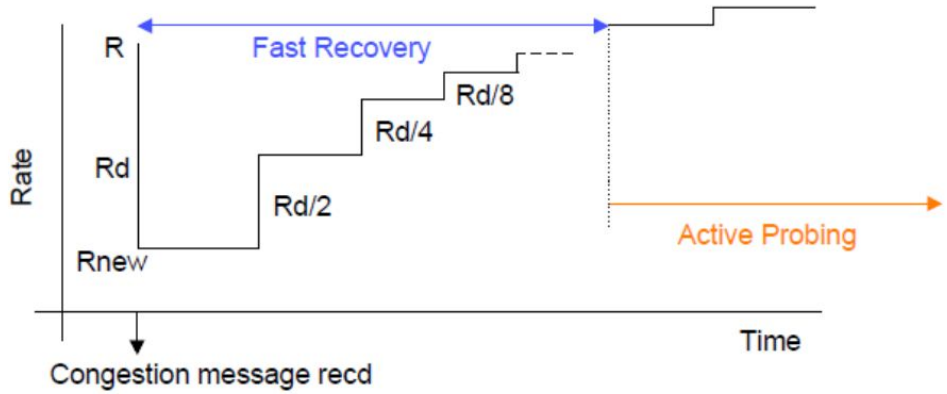
$$\alpha = (1-g)\alpha, \tag{2}$$

$$R_C = (R_T + R_C)/2, \tag{3}$$

$$\begin{aligned} R_T &= R_T + R_{AI}, \\ R_C &= (R_T + R_C)/2, \end{aligned} \tag{4}$$

Instantaneous rate based adjustment (no windows!)

Example Rate Calculation



$$\begin{aligned} R_T &= R_C, \\ R_C &= R_C(1-\alpha/2), \\ \alpha &= (1-g)\alpha + g, \end{aligned}$$

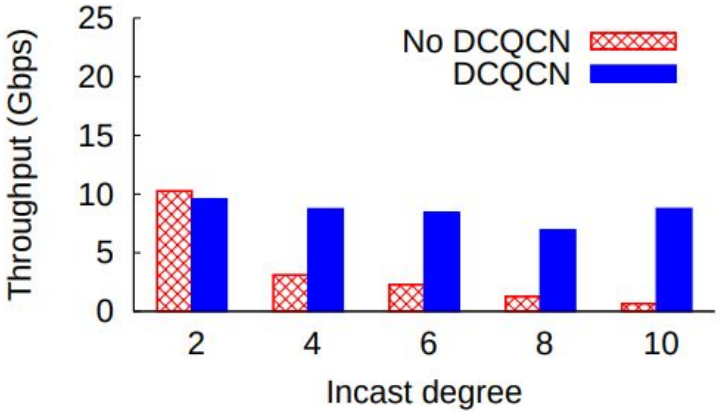
$$R_C = (R_T + R_C)/2,$$

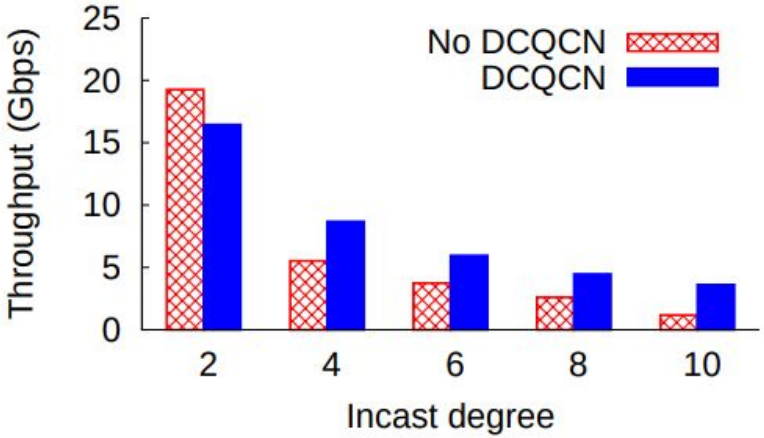
$$\begin{aligned} R_T &= R_T + R_{AI}, \\ R_C &= (R_T + R_C)/2, \end{aligned}$$

Continuous update if no CNP arrived: $\alpha = (1-g)\alpha,$

https://community.mellanox.com/s/article/understanding-dc-qcn-algorithm-for-roce-congestion-control

TCP Incast - 10% Percentile Performance





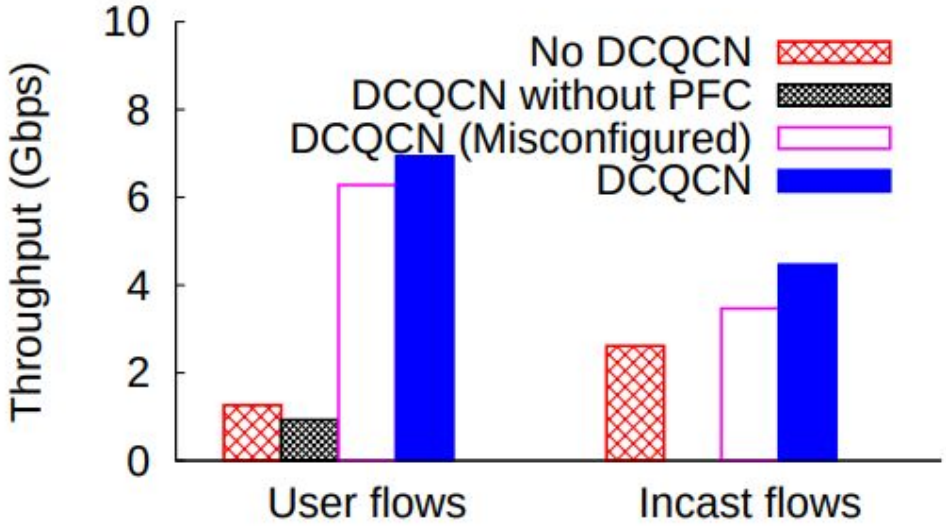
User flows performance which get caught up in the Incast pattern

- No-DCQCN used pause frames
- Port-wide blocking

Incast flows performance

- gradual degradation of performance
- fair sharing (10% is the same as the median, see the paper)

TCP Incast with DCQCN



8:1 incast performance for user and incast flows

- PFC alone is not sufficient to help incast flows
- Misconfiguration of PFC with ECN is a concern (ECN -> then PFC)
- DCQCN helps to deliver the best of the bunch performance

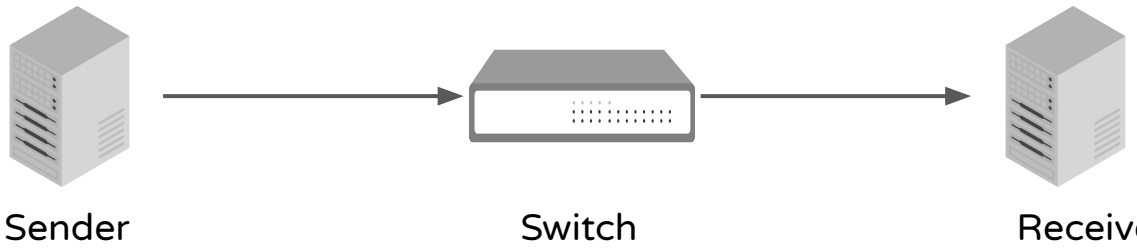
DC-QCN Recap

An end-to-end congestion control scheme for lossless RoCE v2 networks

Flow-based congestion management

It is rate based congestion approach (rather than a window-based (e.g., TCP))

It uses *switch queue occupancy* as the key indicator of congestion. Can we use something else?



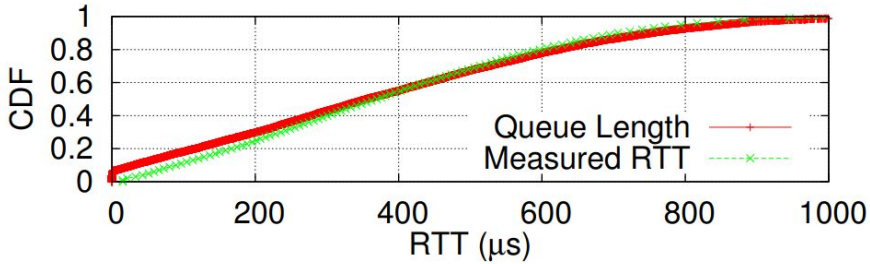
TIMELY: RTT-based Congestion Control for the Datacenter, SIGCOMM 2015

TIMELY

Uses Round Trip Time (RTT) as the indication of congestion signal

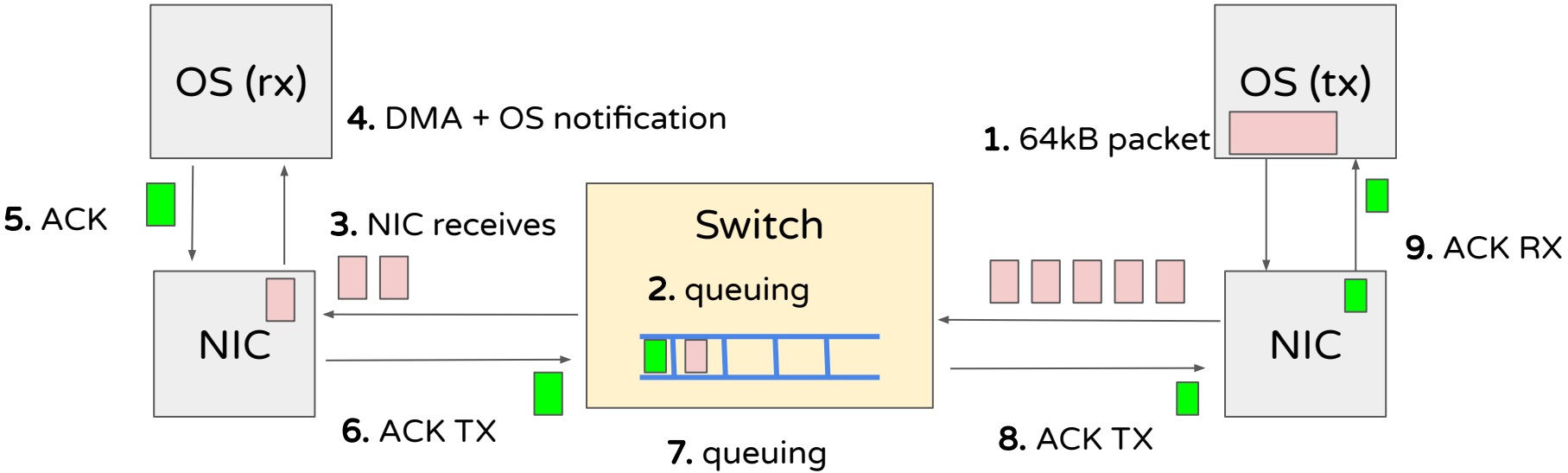
RTT is a multi-bit signal indicating end-to-end congestion throughout the network - *no explicit switch support required to do any marking*

RTT covers ECN signal completely, but not vice versa!



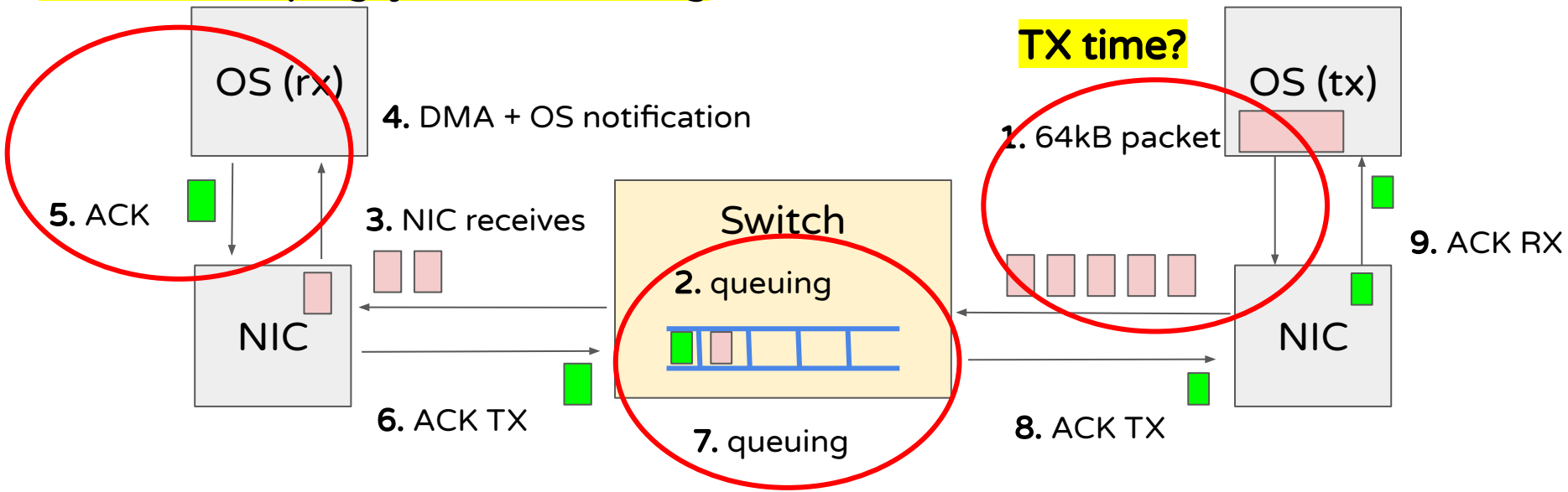
However, getting RTT right is challenging. Why?

RTT Calculation Challenges



RTT Calculation Challenges

OS timestamping, jitter, scheduling



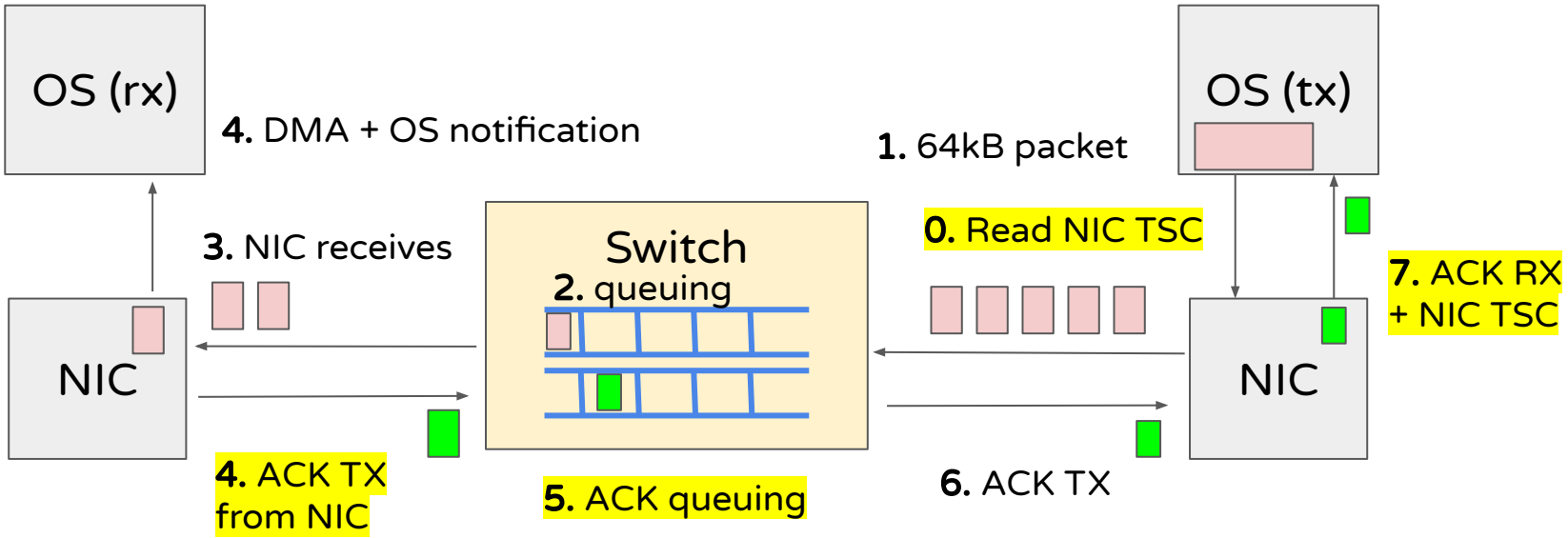
Reverse congestion

RTT Calculation Support from NICs/Switches

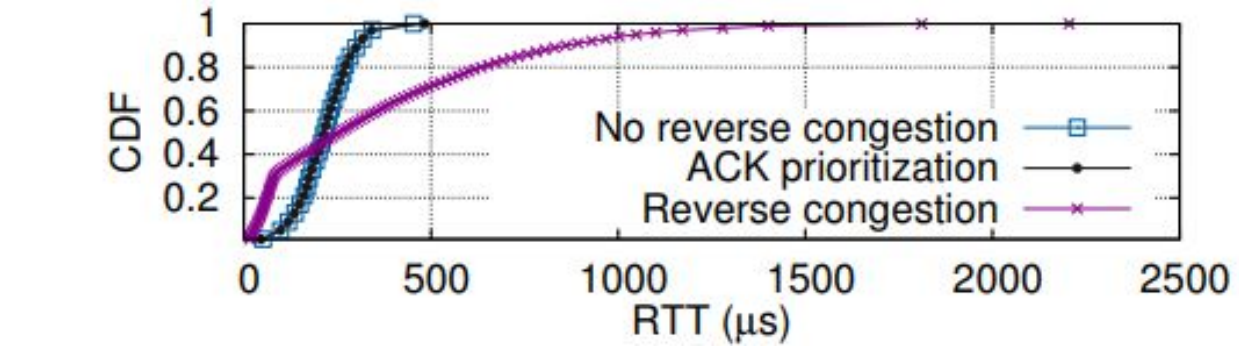
TIMELY assumes that

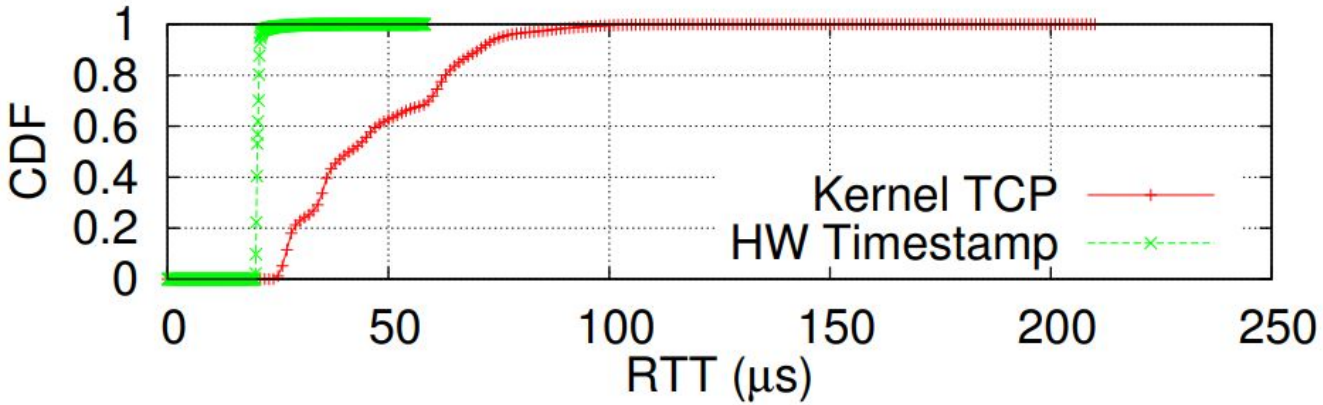
1. The TX NIC can generate completion timestamps so that OS knows when a transmission finished
2. The RX NIC can generate ACKs in hardware without any OS involvement
3. At switches ACKs go through a high priority separate queue

TIMELY RTT Calculation Challenges



Can We Measure RTT Accurately? **YES**





RTT Calculation

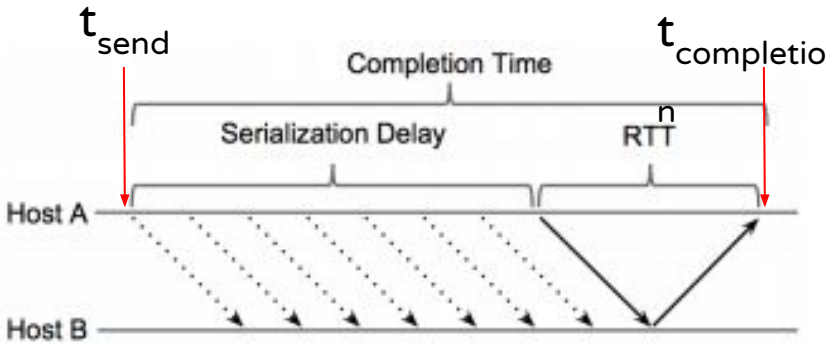
RTT =

1. serialization

2. wire time (dotted lines)

3. ACK write time (solid lines)

4. general queuing delay



$$RTT = \mathrm{t_{completion}} - \mathrm{t_{send}} - \frac{\text{seg. size}}{\text{NIC line rate}}$$

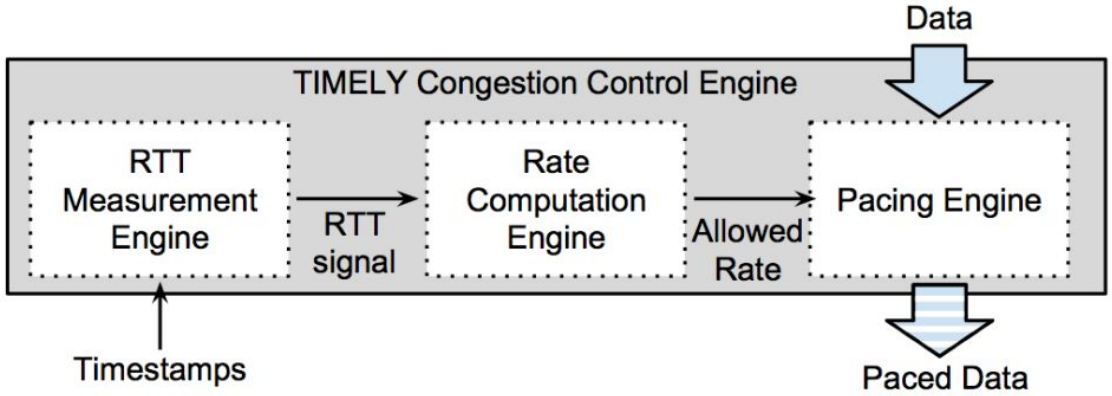
TIMELY

Independent of the transport used - assumes an ACK based protocol (**TCP?**)

Receiver must generate ACKs for new data (there are variants to this)

Key concept here is : absolute RTTs are not required, only the **_gradient_**

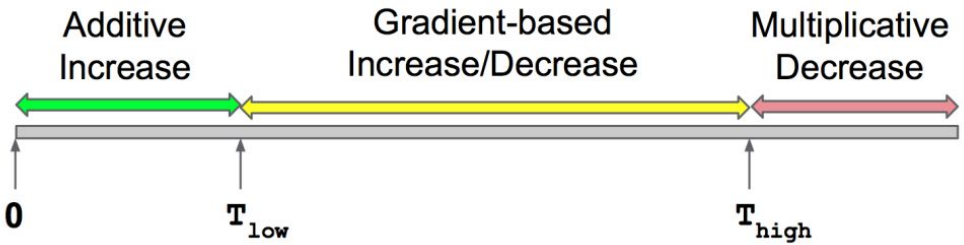
Rising RTT-> queue building, decreasing RTT -> queue depleting



TIMELY Congestion Management

Algorithm 1: TIMELY congestion control.

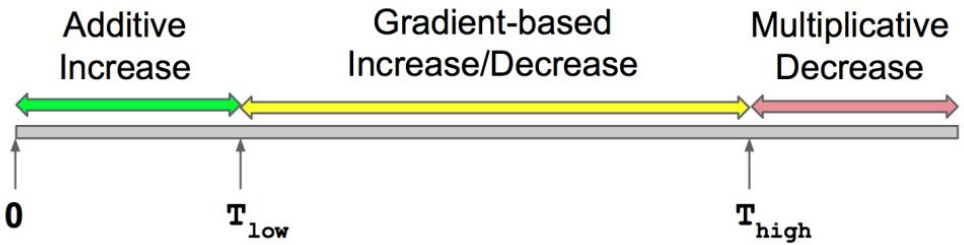
Data: new_rtt
Result: Enforced rate
new_rtt_diff = new_rtt - prev_rtt ;
prev_rtt = new_rtt ;
rtt_diff = (1 - α) · rtt_diff + α · new_rtt_diff ;
▷ α: EWMA weight parameter
normalized_gradient = rtt_diff / minRTT ;
if *new_rtt* < T_{low} **then**
  rate ← rate + δ ;
  ▷ δ: additive increment step
  return;
if *new_rtt* > T_{high} **then**
  rate ← rate · $\left(1 - \beta \cdot \left(1 - \frac{T_{high}}{new\_rtt}\right)\right)$;
  ▷ β: multiplicative decrement factor
  return;
if *normalized_gradient* ≤ 0 **then**
  rate ← rate + N · δ ;
  ▷ N = 5 if gradient<0 for five completion events (HAI mode); otherwise N = 1
else
  rate ← rate · (1 - β · normalized_gradient)



TIMELY Congestion Management

Algorithm 1: TIMELY congestion control.

Data: new_rtt
Result: Enforced rate
new_rtt_diff = new_rtt - prev_rtt ;
prev_rtt = new_rtt ;
rtt_diff = (1 - α) · rtt_diff + α · new_rtt_diff ;
▷ α: EWMA weight parameter
normalized_gradient = rtt_diff / minRTT ;
if *new_rtt* < T_{low} **then**
  rate ← rate + δ ;
  ▷ δ: additive increment step
  return;
if *new_rtt* > T_{high} **then**
  rate ← rate · $\left(1 - \beta \cdot \left(1 - \frac{T_{high}}{new\_rtt}\right)\right)$;
  ▷ β: multiplicative decrement factor
  return;
if *normalized_gradient* ≤ *0* **then**
  rate ← rate + N · δ ;
  ▷ N = 5 if gradient<0 for five completion events (HAI mode); otherwise N = 1
else
  rate ← rate · (1 - β · normalized_gradient)

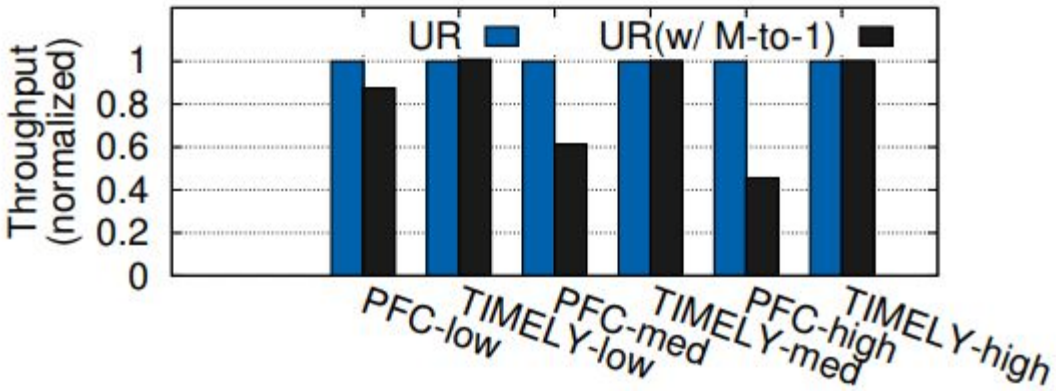


Incast Experiment

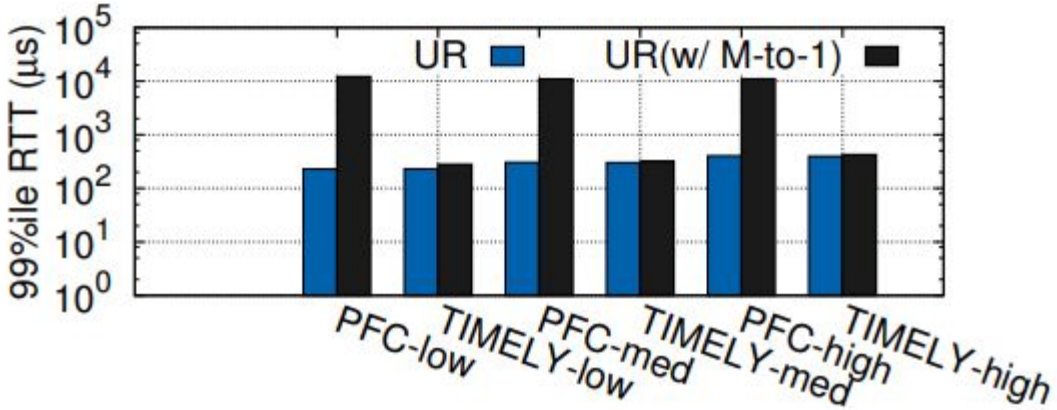
40-to-1 pattern with with 3 flavors of uniform random background traffic:

(a) Normalized throughput

(b) 99-percentile RTT



(a)



(b)

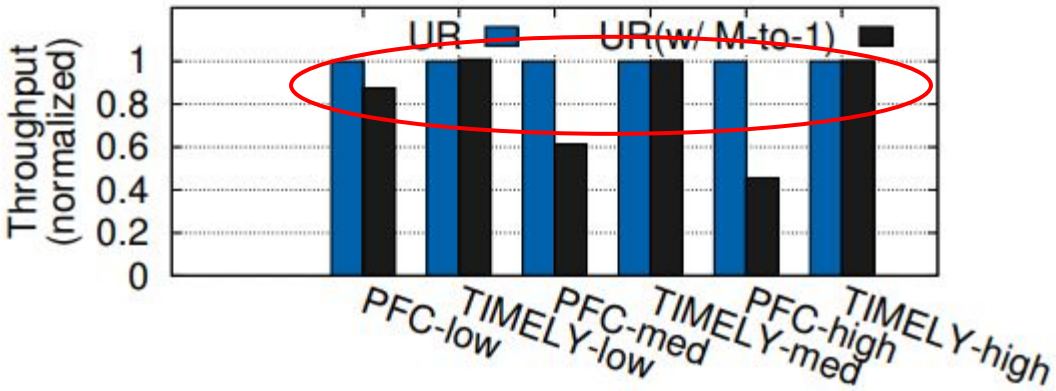
Incast Experiment

40-to-1 pattern with with 3 flavors of uniform random background traffic:

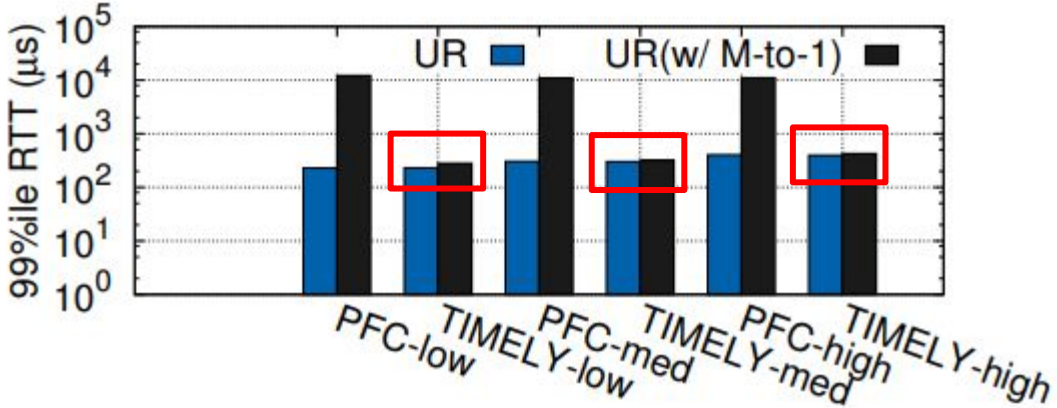
(a) Normalized throughput

(b) 99-percentile RTT

Results: TIMELY throughput and latency are the same as the background traffic w/o reduction

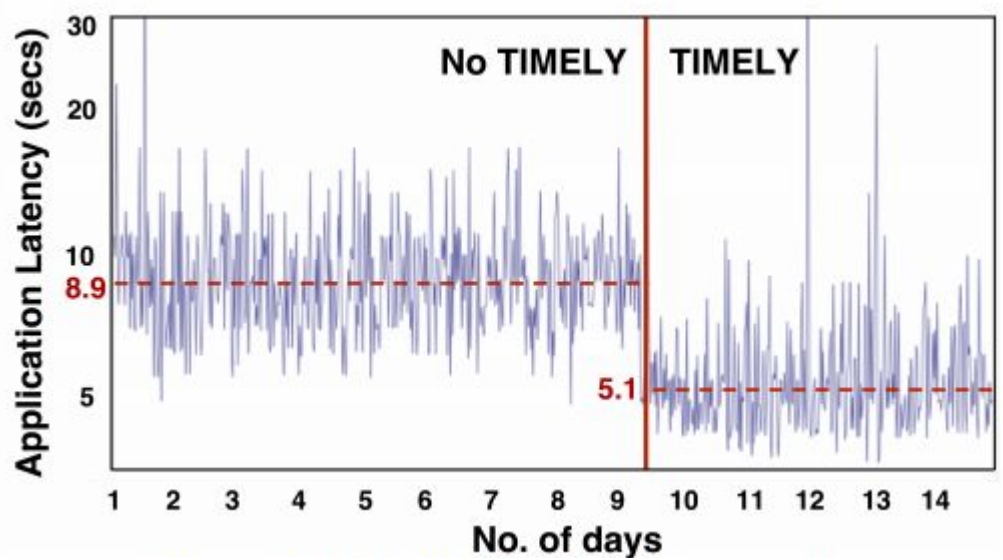


(a)



(b)

Application-Level Benchmark



A (unknown) RPC latency of data center storage benchmark

Summary

- PFC is used for zero-loss networks
- TCP based congestion control mechanisms
  - Window based
  - DCTCP uses ECN markings
- Non-TCP transport
  - InfiniBand: a credit-based lossless link layer
  - DCQCN: uses “queue occupancy” as the congestion signal
  - TIMELY: uses “RTT” as the congestion signal
- There are many follow up research and variants of this work - very active field of research

Potential Project/Thesis/Survey Topics

- Literature survey for TCP optimizations in data centers
  - End host (new abstractions, APIs, low-level OS details)
  - In network (switch programming, distributed algorithms, new mechanisms)
- Literature survey on congestion mechanisms in data centers
- Benchmark InfiniBand performance on DAS
  - Learn RDMA programming and benchmarking
- Build your congestion and flow control mechanism and benchmark it
  - Lossy or lossless - you decide!
  - Learn how to program NIC for your transport
- Compare DCTCP, TIMELY, DCQCN (modeling + evaluation)

Further Reading

Build a scheduler for transmission to manage congestion

- Fastpass: a centralized "zero-queue" datacenter network, SIGCOMM ‘14
- Finishing Flows Quickly with Preemptive Scheduling. In SIGCOMM ‘12
- Universal Packet Scheduling, NSDI 2016.

More mechanisms for congestion control

- ECN or Delay: Lessons Learnt from Analysis of DCQCN and TIMELY, CoNEXT 2016
- PCC: Re-architecting Congestion Control for Consistent High Performance, NSDI 2015
- HPCC: high precision congestion control, SIGCOMM ‘19