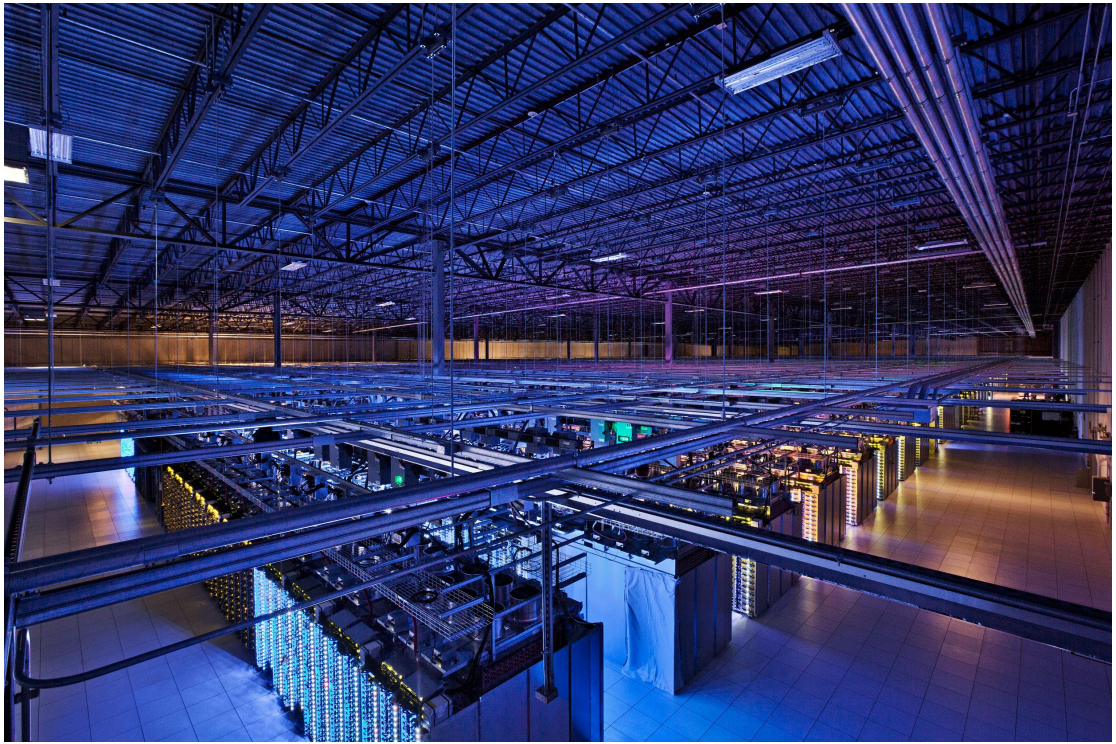
Congestion Control in Datacenters

Ahmed Saeed

What is a Datacenter?

- Tens of thousands of machines in the same building (or adjacent buildings)
- Hundreds of switches connecting all machines

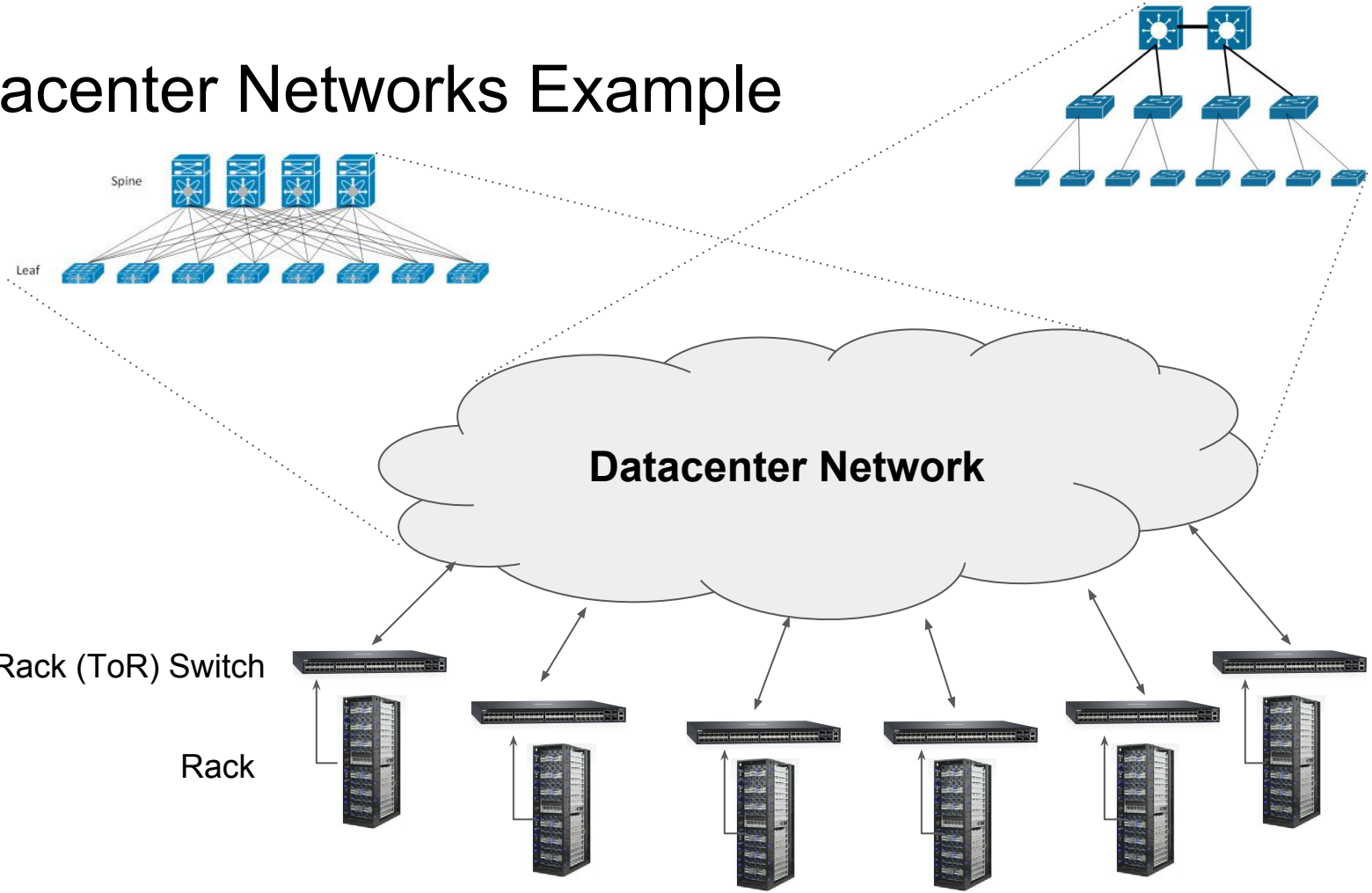


What is a Datacenter?

- A closer look ...

- Virtual reality tour available
  https://youtu.be/zDAYZU4A3w0

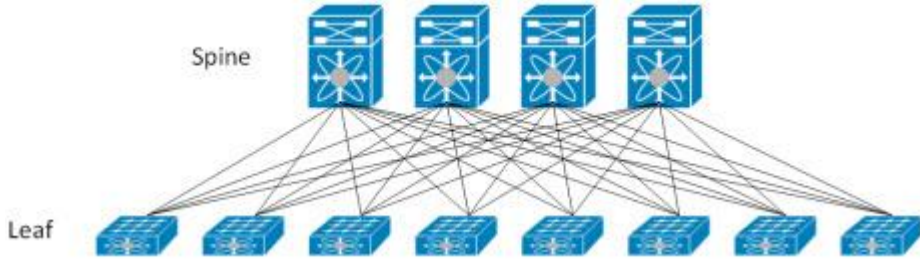
Datacenter Networks Example



Datacenter Goals

- Mesh connectivity
- High utilization
- Resilience
- Low latency

- Picture of an example of tree datacenter network connections which provides redundancy and allows for easier scalability



How is a Datacenter Network different from any other network?

- **Low-cost switches**
  Not a lot of memory for queues (shallow buffers)
- **Clear classes of traffic based on known applications**
  Throughput sensitive large flows and delay sensitive short flows (Data copy load vs. a search query)
- **Better capacity planning**
  The perk of owning all nodes is having clear estimates of how much resources are needed

Datacenter Networks Traffic

- What goes through a datacenter network?
  - Videos, emails, search queries, web crawling data, instant messages, ... (data)
  - Web indexing, search queries, machine learning models, ... (data intensive applications)
- Some data centers are dedicated for the applications
- Applications are usually MapReduce-like to improve programs scalability
  - MapReduce is a programming model that breaks any task into subtasks:
    - **Map step** sends subtasks to worker nodes to solve parts of the task
    - **Reduce step** collects the answers of the subtasks and computes the solution for the task
- Mapreduce traffic requires low latency for fast computation

- Benson, Theophilus, Aditya Akella, and David A. Maltz. "Network traffic characteristics of data centers in the wild." *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*. ACM, 2010.
- Dean, Jeffrey, and Sanjay Ghemawat. "MapReduce: simplified data processing on large clusters." *Communications of the ACM* 51.1 (2008): 107-113.

What is congestion?

$$\Sigma r > C$$

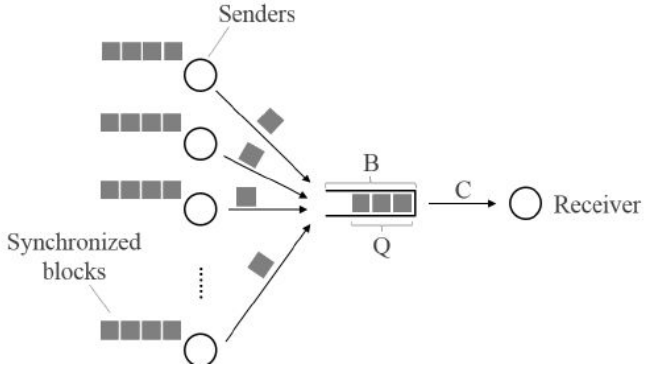
Overutilization of network resources which is problematic because:

Results in long queues in routers which cause

- Excessive delays in some cases
- Buffer overflows in other cases

What to consider in CC for Datacenter Networking?

- **Incast**
  Shallow buffers + Mapreduce = overflowing queues due to a large number of small flows



- **Queue buildup**
  If we only use loss, long flows eat up queues till a loss occurs which also results in long queues for shorter flows

Mohammad Alizadeh, Albert Greenberg, David A. Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. 2010. Data center TCP (DCTCP). In *Proceedings of the ACM SIGCOMM 2010 conference* (SIGCOMM '10).

Loss as signal for congestion

- Datacenters require high throughput at low latency and loss-based CC fails both
- If memory available in switches allows for long queues
  - Waits too long to limit cwnd resulting in
    - Large delays
    - Favors large flows as short flows might not even have space in the queue
- If memory available in switches allows for only short queues
  - Acts too aggressively by halving cwnd at every loss
    - Degrades performance for flows that could have been fine with cwnd-1 instead of cwnd/2
- **Note that we have problems only when queues are nearly full**

Random Early Drop (RED)

- RED attempts to keep queues short by randomly dropping packets before the queue is full
- RED has two thresholds for queue length in the switch
  - Low threshold at which packets are marked at random
  - High threshold at which all packets are marked
  - Setting the parameters is a “black art”
- RED attempts to avoid TCP global synchronization
  - TCP global synchronization happens with tail drop in switches when due to drops all TCP connections going through that queue back off at the same time and the growth of their windows synchronizes (inefficient because network remains under utilized while all flows ramp up their windows)
  - RED drops a packet at random to avoid synchronization

If Loss doesn’t work anymore, then what?

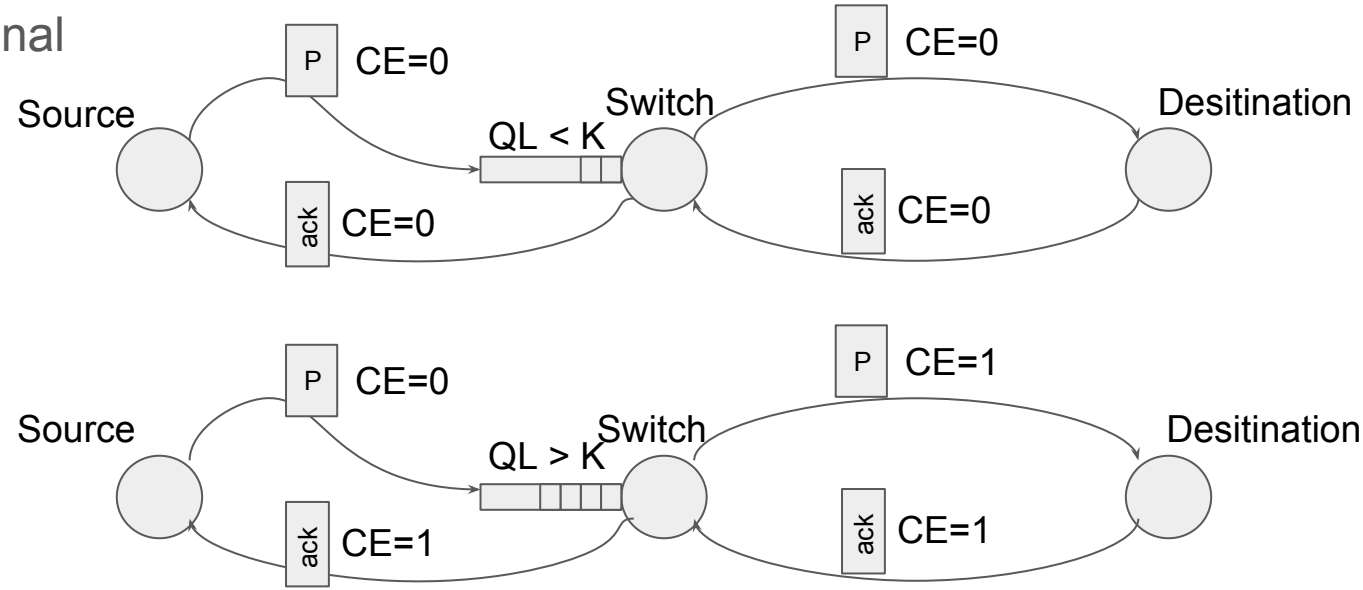
*Recall that the bottom line goal is to have **short queues at the bottleneck***

- Switches can signal senders to allow them to react to congestion using **Explicit Congestion Notifications (ECN)-> DCTCP**

- End nodes can measure delay using large delays as indication of congestion **(Delay Based) -> Timely**

Explicit Congestion Notification (ECN)

- Basic idea: Report congestion severity to source so that sources react to congestion based on its severity
- Every switch has a threshold K on queue length to set Congestion Encountered signal



Random Early Marking (RED+ECN)

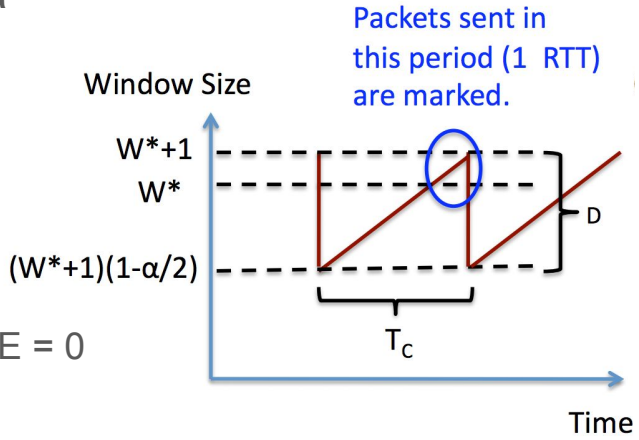
- RED drops a packet that could have been deliver which is wasteful
- RED+ECN marks the packet rather than dropping it
    - The sender treats marked packets as if the packet was dropped by the packet is not wasted
    - Strong assumption that the sender and receiver will understand the signal but valid in data centers
- TCP treats reaction is still severe
    - Drops window to half when just dropping it to three quarters could have done the job

DCTCP

- DCTCP keeps a signal (α) indicating how congested a path is

$$\alpha = (1-g)\,\alpha + g\,F$$

  - F is the ratio of packets with CE = 1 in the last window
  - g is a weight given to new samples over past samples
  - α is 1 when all packets have CE = 1 and 0 if all packets have CE = 0



- DCTCP reacts proportional to congestion:
  - When CE is 0 DCTCP acts like TCP CUBIC
  - When CE is 1 congestion window is reduced proportional to α

$$cwnd = cwnd\,(1 - \alpha/2)$$

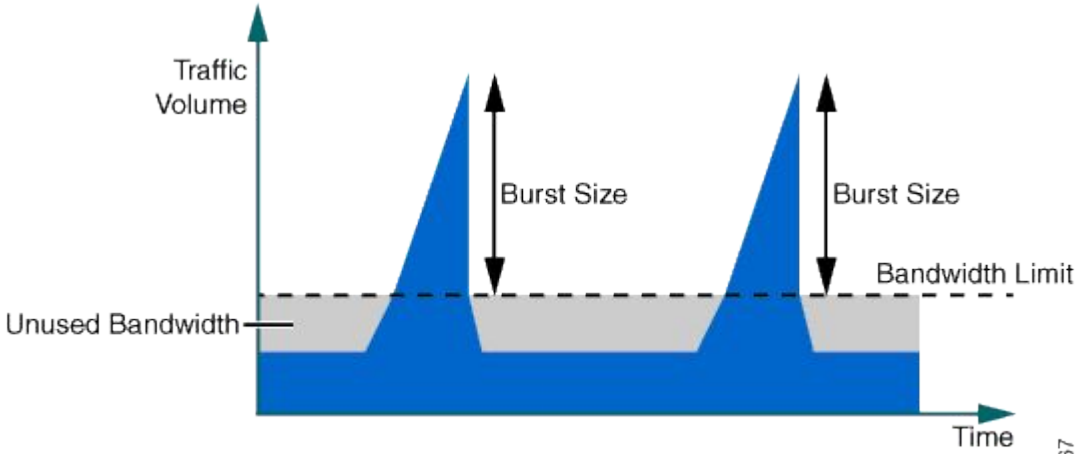
  - cwnd will decrease as in normal TCP when α = 1

DCTCP Impact

- **Incast**
  If each of the flows forming incast traffic has more than one packet, DCTCP helps improve performance. Otherwise, it is still problematic.
- **Queue Buildup**
  DCTCP prevents a few large flows from eating up a majority of the queue. This allows bursty flows to encounter shorter queues. It also reduces latency observed in the network.

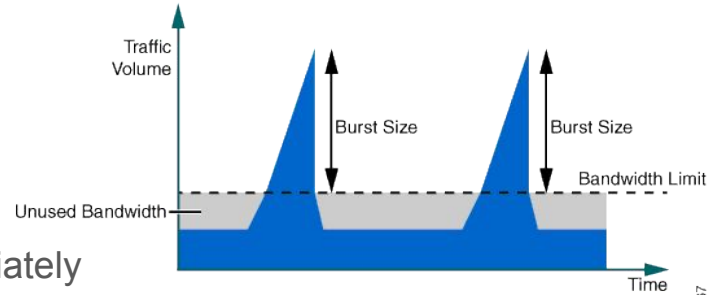
Bursts in Networks

For any traffic with a bandwidth limit (e.g. limited cwnd), traffic seen by the switch looks like below



On average the limit is maintained, but there are spikes which can overwhelm the switch's buffer.

Bursts in Networks (Cont'd)

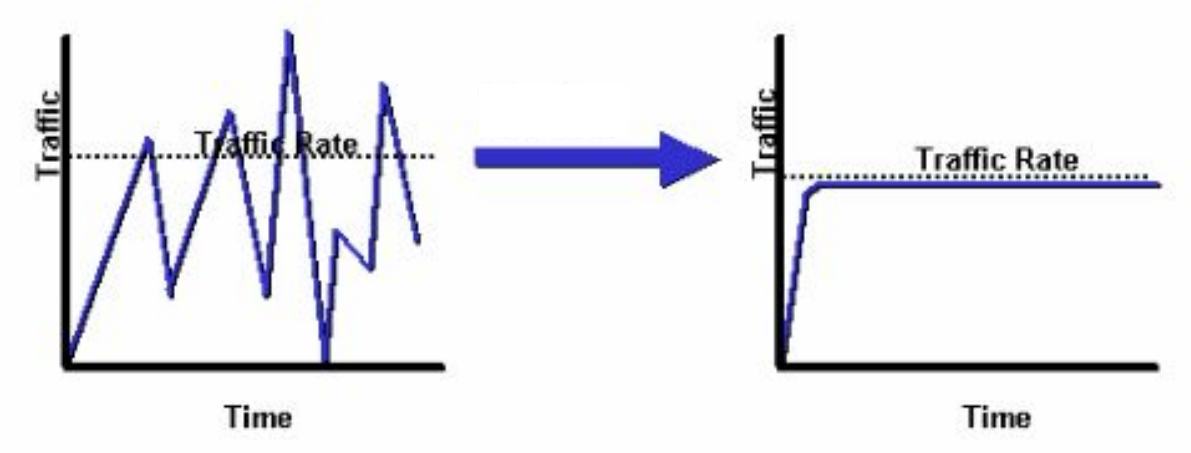


- Consider how window-based scheduling works
  - Send all available packets within a window size immediately
  - What if the window available is less than 1 packet?
    - Packets are queued till the window is large enough for a complete packet

- This means that relying on typical TCP windowing behavior can cause unnecessary drops when switches have short queues

Packet Pacing

Basic idea



Packet Pacing (Cont’d)

- Basic idea: put packets in a priority queue based on the time of transmission of their last byte, dequeue packets when their deadline arrives

- Useful beyond cwnd<1: consider the low-cost switches with small queues
  - Bursts vs paced packets

- Pacing is CPU intensive as it requires a busy loop to check the priority queue for packets, however, its overhead can, sometimes, be justified by gains in bandwidth

Delay (multi-bit) vs ECN (single bit)

- Consider the following cases
  - Multiple switches congested in the network
    - ECN gives the same indicator as if one switch is congested
    - Delay will be proportionally inflated to the queue build in a packet's path
  - Consider a bursty link
    - Packets belonging to the same burst will have the same ECN marking
    - Different packets from the same burst will encounter slightly different RTT
- RTT to be highly correlated with queue length as opposed to fraction of packets with ECN marking
  - because "RTT is more expressive"

Delay-based CC

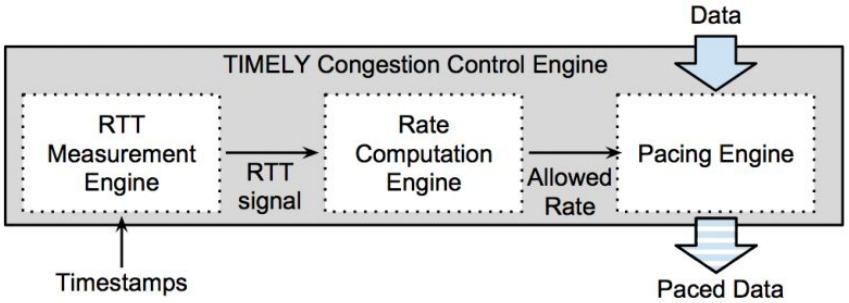
- Basic idea: Large RTT indicates congestion as it reflects delay inflated by network queuing

- Old idea: Can you guess why it is more challenging than in Wide Area Networks?
  - Different paths
  - Delayed acks
  - Processing delays

- Datacenter networks have very low RTT << msec
  - NICs can generate ACKs removing processing delays
  - New NICs can estimate such low RTTs efficiently through hardware timestamping

Mittal, Radhika, et al. "TIMELY: RTT-based Congestion Control for the Datacenter." *ACM SIGCOMM Computer Communication Review*. Vol. 45. No. 4. 2015.

Timely: RTT-based congestion control algorithm



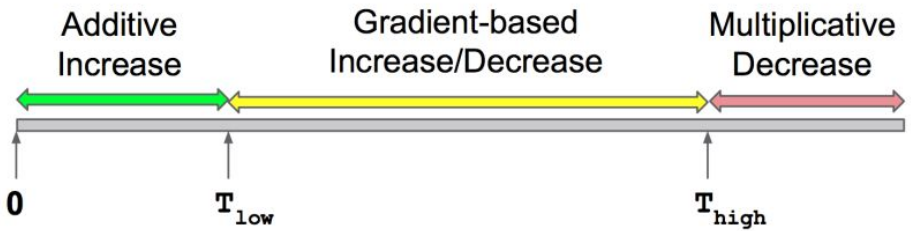
- RTT measurement engine: measures RTT based on timestamped packets, their ACKs, and their rate
- Based on estimated RTT a rate is calculated for each flow
- Packets for each flow are paced out based on the calculated rate

Rate Calculation

- What do we need to perfectly control queue length?
  - To detect and attempt to drain the queue faster than it drains without intervention which is not possible in modern networks
    - We need at least an RTT to detect it which can be as low as 50 microseconds but the queue length is in the range of a few microseconds

- We need a proxy of that queue buildup: ***Direction of Change of Delay***
  - Proxy on the direction of queue build up

- Basic idea:



Timely Algorithm

- Keep track of trend of the direction of change of RTT
  - Use Exponential Weighted Moving Average (EWMA) filter on difference between RTTs for consecutive packets
- When gradient is negative
  -> Network has capacity
  -> Additive increase
- When gradient is positive
  -> Queue build up
  -> Multiplicative decrease proportional to normalized gradient

Timely Algorithm (Cont'd)

- What behavior will this algorithm maintain?
  - Gradient close to zero
  - Queues that are very small

- Gradient close to zero
  - If below zero add packets and build queues
  - If queues build up back off

- Queues that are very small
  - If RTT reaches T_{high} even if gradient is at zero will back off