Congestion Control in Data Center Networks

Lecture 8

Subir Varma

Data Centers and Data Center Networks (DCN)

- Modern data centers are created by interconnecting together a large number of commodity servers and their associated storage systems, with commodity Ethernet switches.

- They create a “warehouse-scale” computing infrastructure that scales “horizontally”; that is, we can increase the processing power of the data center by adding more servers as opposed to increasing the processing power of individual servers (which is a more expensive proposition).

- The job of a DCN in the data center architecture is to interconnect the servers in a way that maximizes the bandwidth between any two servers while minimizing the latency between them.

- The DCN architecture should also allow the flexibility to easily group together servers that are working on the same application, irrespective of where they are located in the DCN topology.

DCN Characteristics

From the congestion control point of view, DCNs have some unique characteristics compared with the other types of networks that we have seen so far:

- The round trip latencies in DCNs are extremely small, usually of the order of a few hundred microseconds, as opposed to tens of milliseconds and larger for other types of networks.
- Applications need very high bandwidths and very low latencies at the same time.
- There is very little statistical multiplexing, with a single flow often dominating a path.
- To keep their costs low, DCN switches have smaller buffers compared with regular switches or routers because vendors implement them using fast (and expensive) static random-access memory (SRAM) to keep up with the high link speeds. Moreover, buffers are shared between ports, so that a single connection can end up consuming the buffers for an entire switch.
- To communicate at full speed between any two servers, the interconnection network provides multiple paths between them, which is one of the distinguishing features of DCNs.

Problems with TCP Performance in DCNs

- Loss based TCP requires large buffers; indeed, the buffer size should be greater than or equal to the delay bandwidth product of the connection to fully use the full bandwidth of the link (see Lecture 2)

- End-to-end delays of connections under TCP are much larger than what can be tolerated in DCNs. The delays are caused by TCP’s tendency to fill up link buffers to capacity to fully use the link.

- How to load balance the traffic among multiple paths in a DCN?

- The Incast problem is a special type of traffic overload situation that occurs in data centers. It is caused by the way in which jobs are scheduled in parallel across multiple servers, which causes their responses to be synchronized with one another, thus overwhelming switch buffers.

Types of DCN Congestion Control Algorithms

1. Algorithms that retain the end-to-end congestion control philosophy of TCP:

- Data Center TCP (DCTCP), Deadline-Aware Data Center TCP (D2TCP), and High bandwidth Ultra Low Latency (HULL), fall into this category of algorithms.
- They use a more aggressive form of a Random Early Detection (RED) like Explicit Congestion Notification (ECN) feedback from congested switches that are then used to modify the congestion window at the transmitter.
- These new algorithms are necessitated due to the fact that Normal RED or even the more effective form of Active Queue Management (AQM) with the Proportional-Integral (PI) controller is not able to satisfy DCNs' low latency requirements.

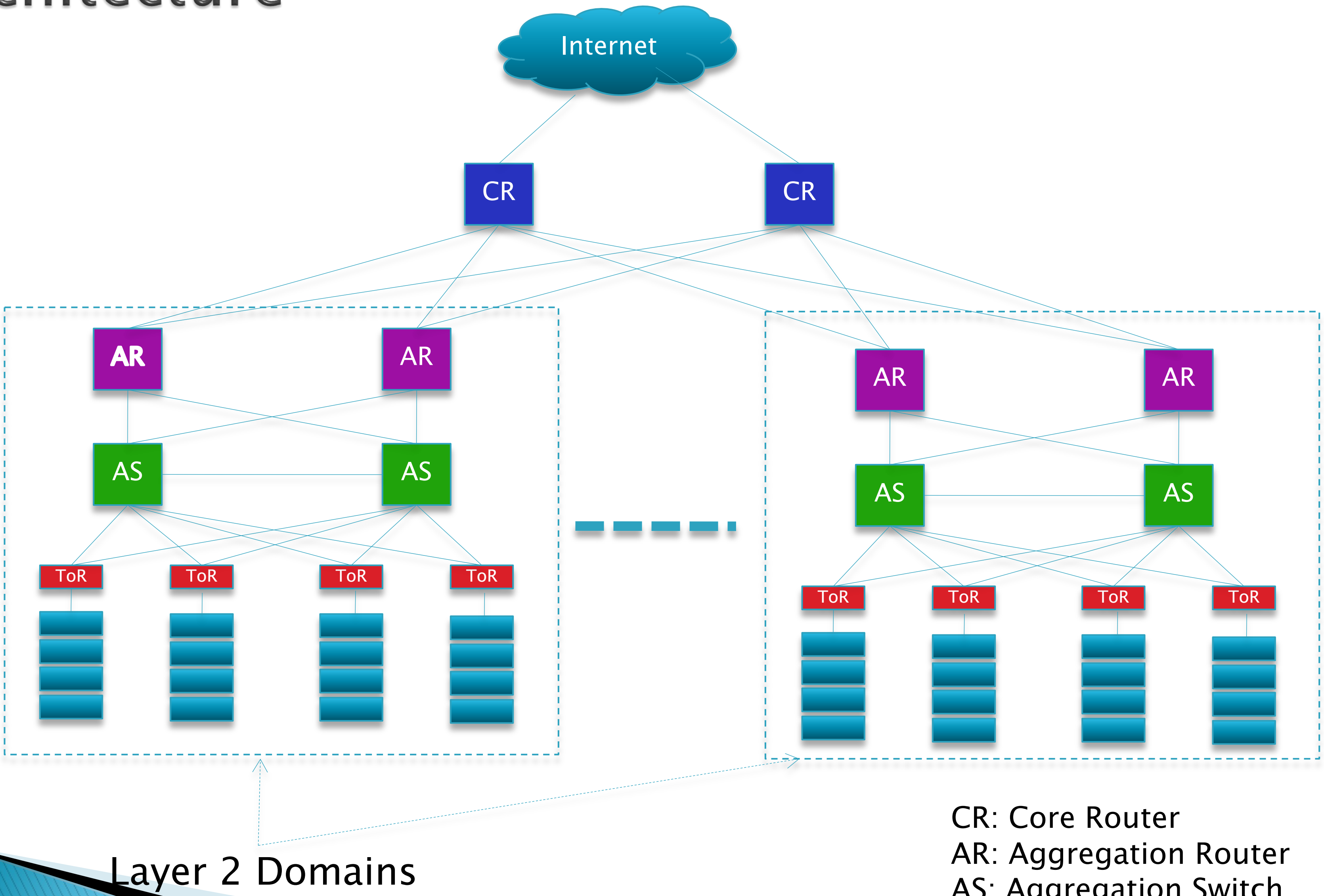
Types of DCN Congestion Control Algorithms

2. Algorithms that depend on in-network congestion control mechanisms

- DCN congestion control protocols such as D3, Preemptive Distributed Quick Flow Scheduling (PDQ), DeTail, and pFabric fall in this category.
- They use additional mechanisms at the switch, such as bandwidth reservations in D3, priority scheduling in pFabric or packet by packet load balancing in DeTail.
- The general trend is towards a simplified form of rate control in the end systems coupled with greater support for congestion control in the network because this leads to much faster response to congestion situations.
- This is a major departure from the legacy congestion control philosophy of putting all the intelligence in the end system.

Data Center Interconnect Architecture

Traditional Tree Type Data Center Network Architecture



CR: Core Router
AR: Aggregation Router
AS: Aggregation Switch
ToR: Top of Rack Switch

Traditional DCN

- A typical data center consists of thousands of commodity servers that are connected together using special type of Ethernet network called an Inter-connection Network.

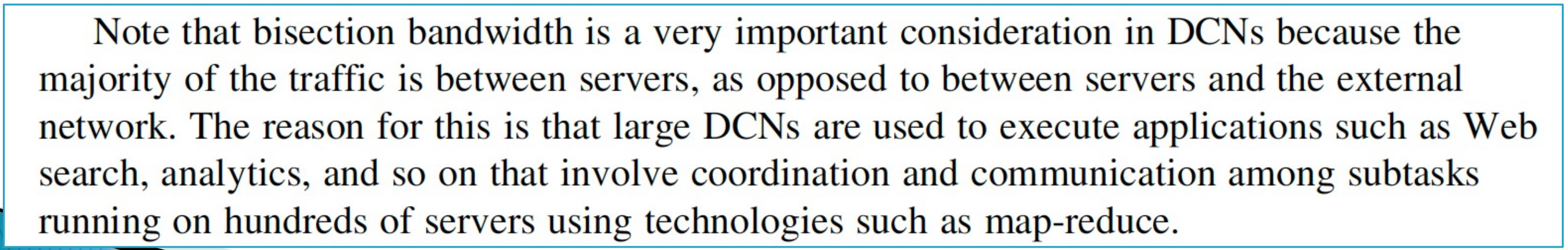
- Servers are arranged in racks consisting of 20 to 40 devices, each of which is connected to a Top of Rack (ToR) switch. Each ToR is connected to two aggregation switches (AS) for redundancy, perhaps through an intermediate layer of L2 switches.

- Each AS is further connected to two aggregation routers (ARs), such that all switches below each pair of ARs form a single Layer 2 domain, connecting several thousand servers.

- Servers are also partitioned into Virtual Local Area Networks (VLANs) to limit packet flooding and Address Resolution Protocol (ARP) broadcasts and to create a logical server group that can be assigned to a single application. ARs in turn are connected to core routers (CRs) that are responsible for the interconnection of the data center with the outside world.

Problem with Traditional DCNs

- Lack of bisection bandwidth:
  - Bisection bandwidth is defined as the maximum capacity between any two servers. Even though each server may have a 1-Gbps link to its ToR switch and hence to other servers in its rack, the links further up in the hierarchy are heavily oversubscribed.
  - For example, only 4 Gbps may be used on the link between the ToR and AS switches, resulting in 1:5 oversubscription when there are 20 servers per rack, and paths through top-most layers of the tree may be as much as 1:240 oversubscribed.
  - As a result of this, designers tend to only user servers that are closer to each other, thus fragmenting the server pool (i.e., there may be idle servers in part of the data center that cannot be used to relieve the congestion in another portion of the system).

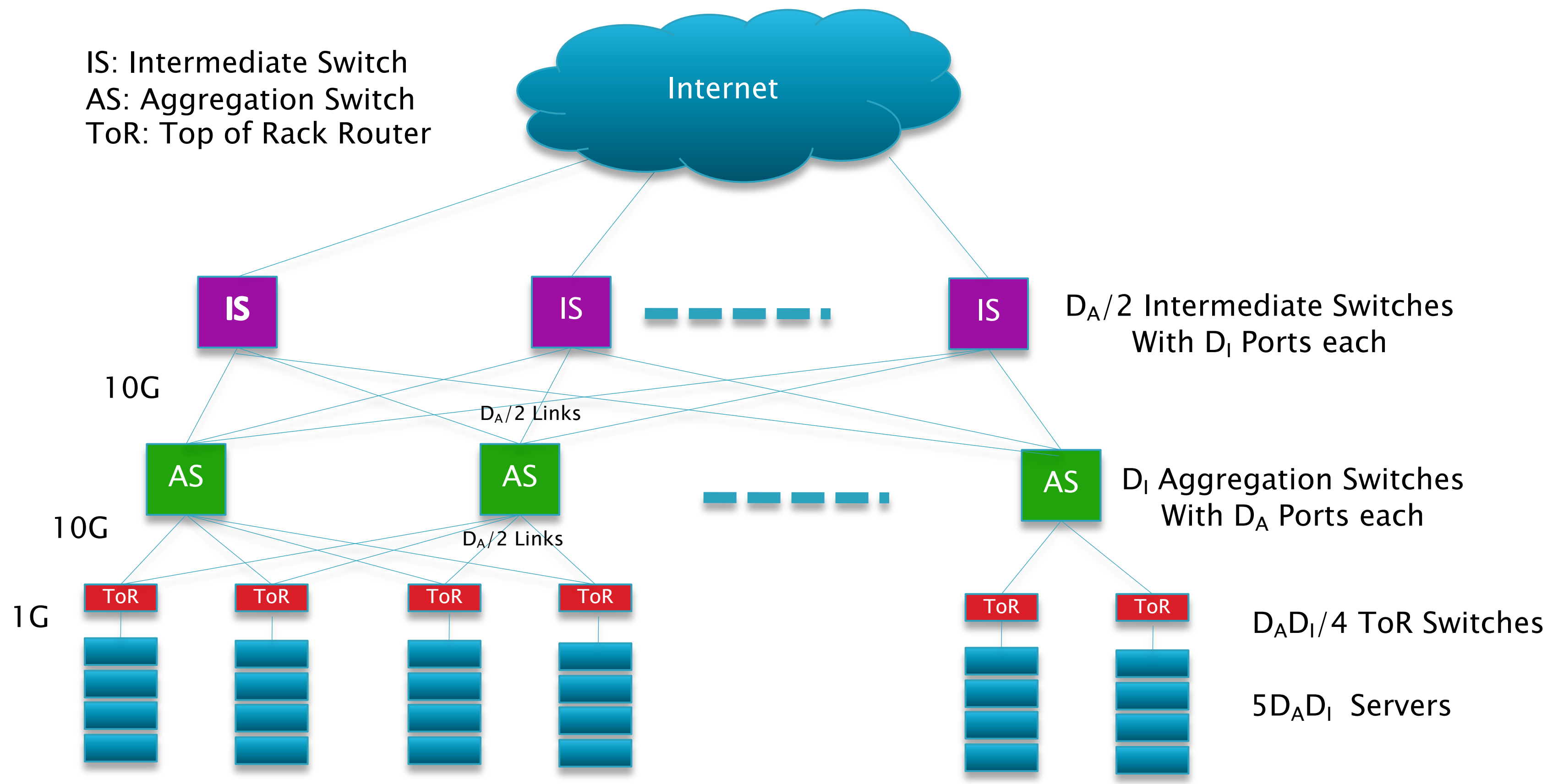

Note that bisection bandwidth is a very important consideration in DCNs because the majority of the traffic is between servers, as opposed to between servers and the external network. The reason for this is that large DCNs are used to execute applications such as Web search, analytics, and so on that involve coordination and communication among subtasks running on hundreds of servers using technologies such as map-reduce.

Problem with Traditional DCNs

- Configuration complexity:
  - Adding additional servers to scale the service outside the domain requires reconfiguration of IP addresses and VLAN trunks because IP addresses are topologically significant and are used to route traffic to a particular Layer 2 domain. As a result, most designs use a more static policy whereby they scale up by adding servers idle within the same domain, thus resulting in server under-utilization.

- Poor reliability and redundancy:
  - Within a Layer 2 domain, the use of Spanning Tree protocol for data forwarding results in only a single path between two servers. Between Layer 2 domains, up to two paths can be used if Equal Cost Multi-Path (ECMP) routing is turned on.

The VL2 Interconnection Network



VL2: CLOS Network type architecture

- Each ToR switch is still connected to two AS switches using 1-Gbps links, but unlike the Tree Architecture, each AS switch is connected to every other AS switch in 2-hops, through a layer of intermediate switches (ISs) using 10 Gbps or higher speed links.
- As a result, if there are n IS switches, then there are n paths between any two ASs, and if any of the IS fails, then it reduces the bandwidth between 2 AS switches by only 1/n.
- For the VL2 network shown in prior page, the total capacity between each layer is given by $D_I D_A/2$ times the link capacity, assuming that there are D_I AS switches with D_A ports each.
- Also note that the number of ToR switches is given by $D_A D_I/4$, so that if there are M servers attached with a 1-Gbps link to the ToR and the links between the AS and IS are at 10 Gbps, then equating the total bandwidth from the AS to the ToR switches to the total bandwidth in the opposite direction, we obtain

$$M \times \frac{D_A D_I}{4} = 10 \times \frac{D_A D_I}{2} \quad i.e. \quad M = 20$$

VL2 (cont)

- The VL2 architecture uses Valiant Load Balancing (VLB) among flows to spread the traffic through multiple paths. VLB is implemented using ECMP (Equal Cost Multi Path) forwarding in the routers.

- VL2 enables the system to create multiple Virtual Layer 2 switches (hence the name), such that it is possible to configure a virtual switch with servers that may be located anywhere in the DCN.

- All the servers in Virtual Switch are able to communicate at the full bisection bandwidth and furthermore are isolated from servers in the other Virtual Switches.

Internal Routing within VL2

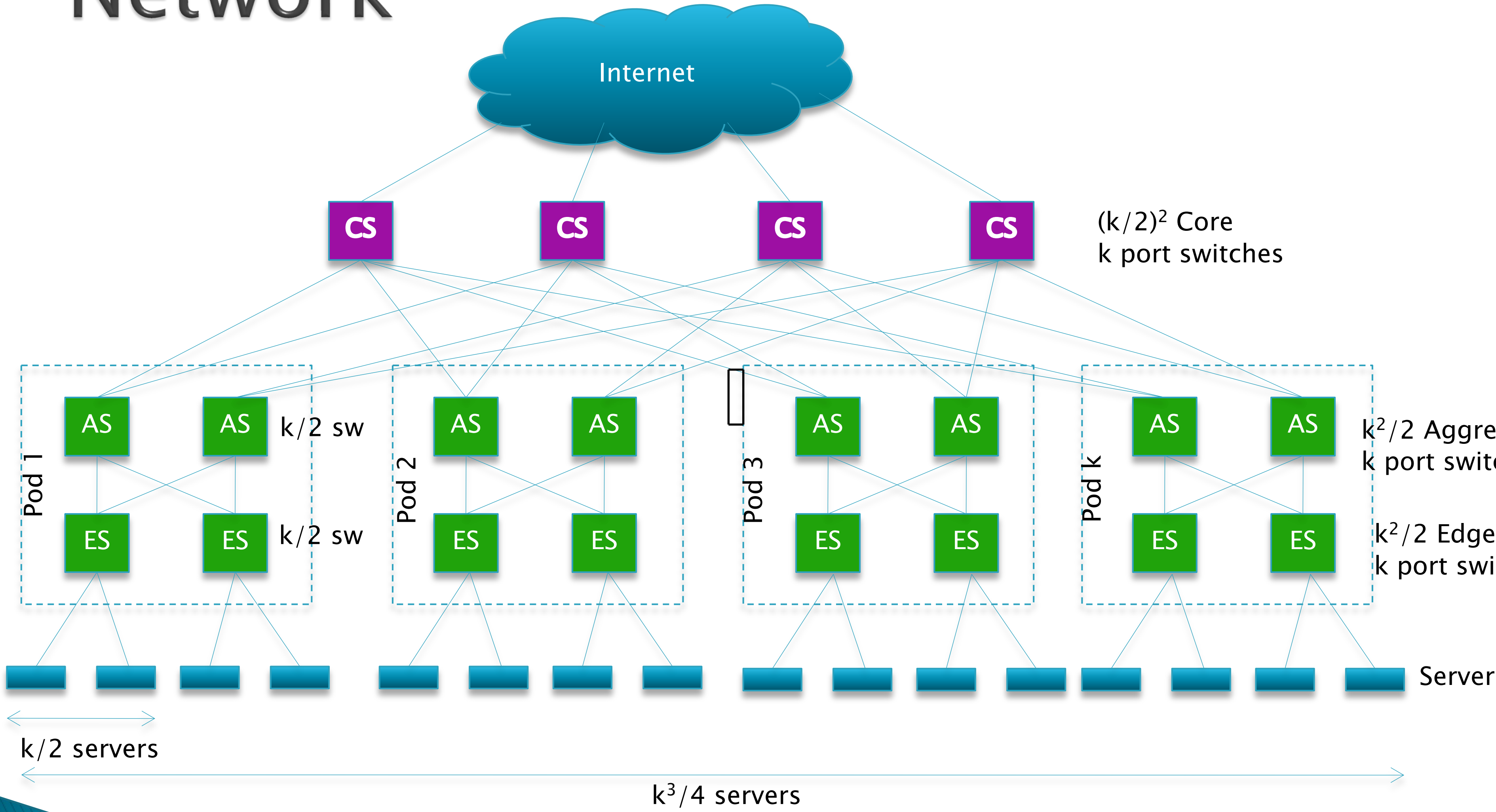
- To route packets to a target server, the system uses two sets of IP addresses. Each application is assigned an Application Specific IP Address (AA), and all the interfaces in the DCN switches are assigned a Location Specific IP Address (LA) (a link-state based IP routing protocol is used to disseminate topology information among the switches in the DCN).

- An application's AA does not change if it migrates from server to server, and each AA is associated with an LA that serves as the identifier of the ToR switch to which it is connected. VL2 has a Directory System (DS) that stores the mapping between the AAs and LAs.

- When a server sends a packet to its ToR switch, the switch consults the DS to find out the destination ToR and then encapsulates the packet at the IP level, known as tunneling, and forwards it into the DCN.

- To take advantage of the multiple paths, the system does load balancing at the flow level by randomizing the selection of the Intermediate Switch used for the tunnel.

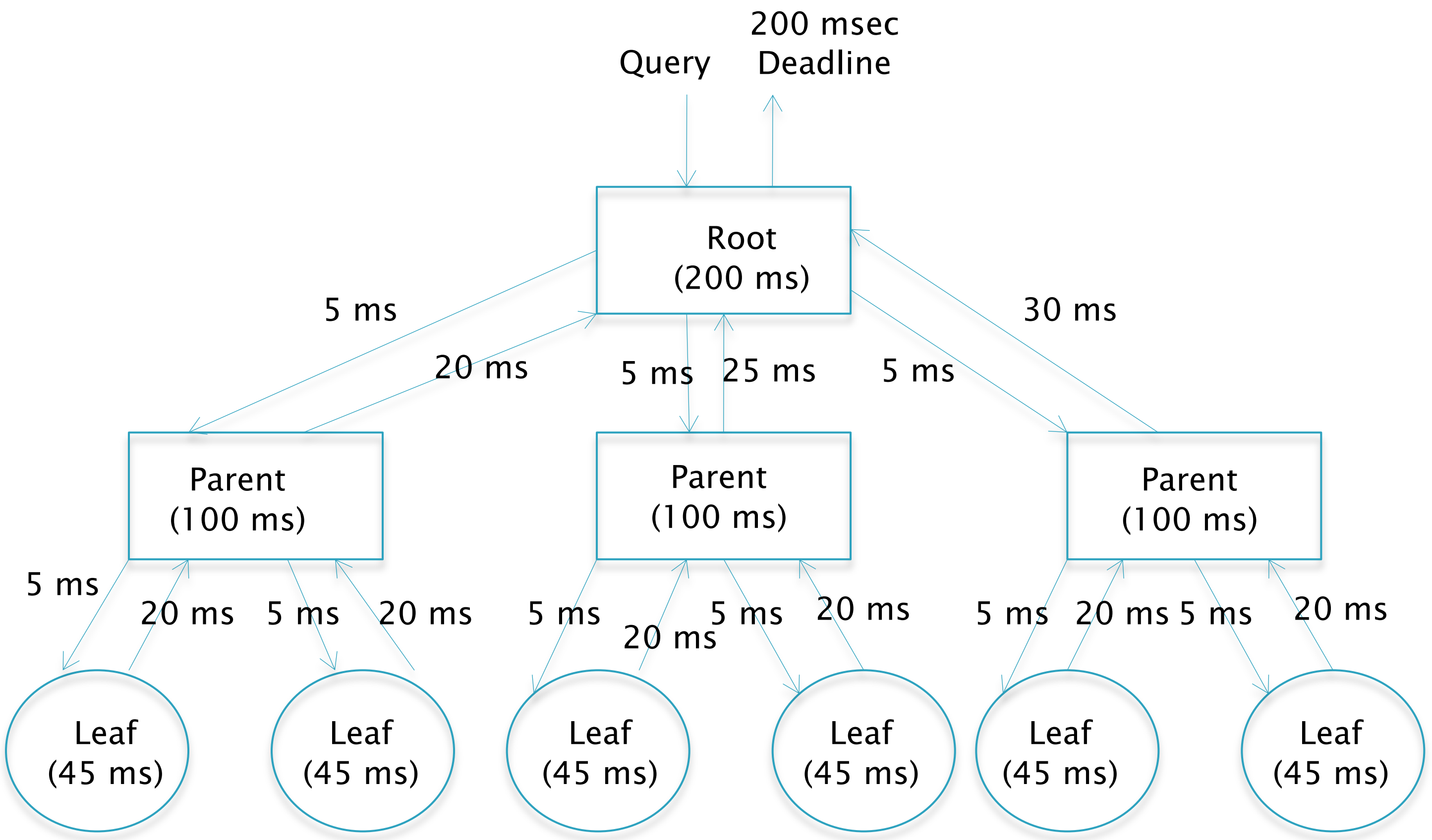
The Fat Tree Interconnection Network



Fat Tree

- Fat Tree is also CLOS based
- The network is built entirely with k-port 1-Gbps switches and is constructed around k pods, each of which has two layers of switches, with (k/2) switches in each layer.
- The bottom layer of switches in a pod is connected to the servers, with (k/2) servers connected to each switch. This implies that the system with k pods can connect
  (k/2) switches/pod * (k/2) server facing ports per switch * k pods = (k^3/4) servers per network.
- Similarly, it can be shown that there are $k^2/4$ core switches per network, so that the number of equal-cost multipaths between any pair of hosts is also given by $k^2/4$.
- Because the total bandwidth between the core switch and aggregation switch layers is given by $k^3/4$, it follows that the network has enough capacity to support a full 1-Gbps bisectional bandwidth between any two servers in the ideal case.

Map-Reduce type Data Model with Deadlines



Map Reduce Type App

- Each of the nodes represents the processing at a server node; the connector between nodes stands for the transmission of a flow across the interconnection network.
- A typical online transaction is generated when a user enters a query into a search engine. A very large data set is required to answer the query, which is spread among thousands of servers in the DCN, each with its own storage.
- The way a query progresses through a DCN is shown in the figure:
  - The query arrives at the root node, which broadcasts it to down to the next level,
  - These in turn generate their own queries one level down until the leaf nodes holding the actual data are reached.
  - Each leaf node sends its response back to its parent, which aggregates the replies from all its child nodes and sends it up to the root.
- Furthermore, answering a query may involve iteratively invoking this pattern; one to four iterations are typical, but as many as 20 may occur.
- Studies have shown that the response needs to be generated within a short deadline of 230 to 300 ms.

Map Reduce Type App

- The propagation of the request down to the leaves and the responses back to the root must complete within the overall deadline that is allocated to the query; otherwise, the response is discarded.
- To satisfy this, the system allocates a deadline to each processing node, which gets divided up into two parts:
  - The computation time in the leaf node and the communication latency between the leaf and the parent.
  - The deadlines for individual computational nodes typically vary from about 10 to 100 ms.
- The deadlines for communications between hosts cannot exceed tens of milliseconds if the system is to be able to meet the overall job deadline.

Data Center TCP
DCTCP

DCTCP Objectives

Objectives

- Minimization of flow latency is a very important requirement in DCNs.

- At the same time, other flows in the data center are throughput intensive. For example, large transfers that update the internal data structures at a server node fall into the latter category; hence, throughput maximization and high link utilization cannot be ignored as the congestion control objective.

- To meet the requirements for such a diverse mix of short and long flows, switch buffer occupancies need to be persistently low while maintaining high throughput for long flows.

- Two ways to control queueing latency:
  - Delay-based protocols such as TCP Vegas: Problem with these – they rely on a very accurate measurement of RTT which is a problems in a DCN environment because the RTTs are extremely small, of the order of a few hundred microseconds.
  - AQM algorithms: These use explicit feedback from the congested switches to regulate the transmission rate, RED being the best known member of this class of algorithms.

    However RED and and even PI controllers do not work well in environments where there is low statistical multiplexing and the traffic is bursty, both of which are present in DCNs.

DCTCP Operation

1. At the switch: An arriving packet at a switch is marked with Congestion Encountered (CE) codepoint if the queue occupancy is greater than a threshold K at its arrival. A switch that supports RED can be reconfigured to do this by setting both the low and high threshold to K and marking based on instantaneous rather than average queue length.

2. At the receiver: A DCTCP receiver tries to accurately convey the exact sequence of marked packets back to the sender by ACKing every packet and setting the ECN-Echo flag if and only if the data packet has a marked CE codepoint. This algorithm can also be modified to take delayed ACKs into account while not losing the continuous monitoring property at the sender.

3. At the sender: The sender maintains an estimate of the fraction of packets that are marked, called α, which is updated once for every window of data as follows:

   $$\alpha \leftarrow (1-g)\alpha + gF$$

   where F is the fraction of packets that were marked in the last window of data and $0<g<1$ is the smoothing factor.
   Note that because every packet gets marked, α estimates the probability that the queue size is greater than K.

$$\boxed{\alpha = P(b > K)}$$

DCTCP Operation

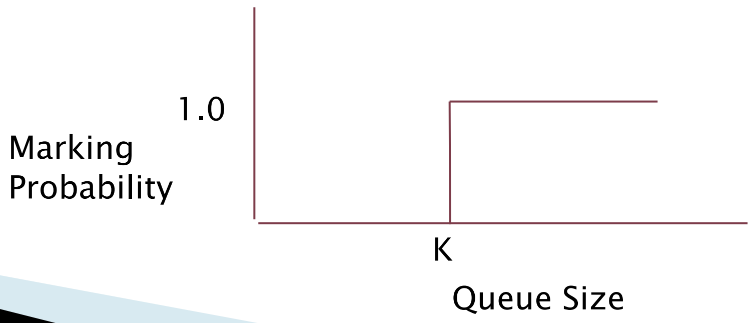
4. Features of TCP rate control such as Slow Start, additive increase during congestion avoidance, or recovery from lost packets are left unchanged. However, instead of cutting its window size by 2 in response to a marked ACK, DCTCP applies the following rule once every RTT:

$$W \leftarrow W + 1 \quad \text{if none of the packets are marked in a window}$$

$$W \leftarrow W\left(1 - \frac{\alpha}{2}\right) \quad \text{if one or more packets are marked per window}$$

5. Thus, a DCTCP sender starts to reduce its window as soon as the queue size exceeds K (rather than wait for a packet loss) and does so in proportion to the average fraction of marked packets. Hence, if very few packets are marked then the window size hardly reduces, and conversely in the worst case if every packet is marked, then the window size reduces by half every RTT (as in Reno).

Simulation results in Alizadeh et. al. show that DCTCP achieves its main objective of reducing the size of the bottleneck queue size; indeed, the queue size with DCTCP is 1/20th the size of the corresponding queue with TCP Reno and RED.



DCTCP Analysis

The following non-linear, delay-differential equations describe the dynamics of W(t), α(t), and the queue size at the switch, q(t):

$$\frac{dW}{dt} = \frac{1}{T(t)} - \frac{W(t)\alpha(t)}{2T(t)} p(t - T(t))$$

$$\frac{db}{dt} = N\frac{W(t)}{T(t)} - C$$

$$\frac{d\alpha}{dt} = \frac{g}{T(t)}[p(t - T(t)) - \alpha(t)]$$

Here p(t) indicates the packet marking process at the switch and is given by

$$p(t) = 1_{\{Q(t) > K\}}$$

Using the approximation T = D + K/C (where D is the round trip propagation delay), it was shown by Alizadeh et al. that this fluid model agrees quite well with simulations results.

Stability Condition: $g \in (0, 1]$ and $(CD + K)/N > 2$

To attain 100% throughput, the minimum buffer size K at the bottleneck node is given by

$$K > 0.17CD$$

Contrast this with TCP Reno

DCTCP Analysis

Let $S(W_1,W_2)$ be the number of packets sent by the sender while its window size increases from W_1 to $W_2 > W_1$. Note that this takes (W_2-W_1) round trip times because the window increases by at most 1 for every round trip. The average window size during this time is given by $(W_1 + W_2)/2$, and because the increase is linear, it follows that

$$S(W_1, W_2) = \frac{W_2^2 - W_1^2}{2} \tag{9}$$

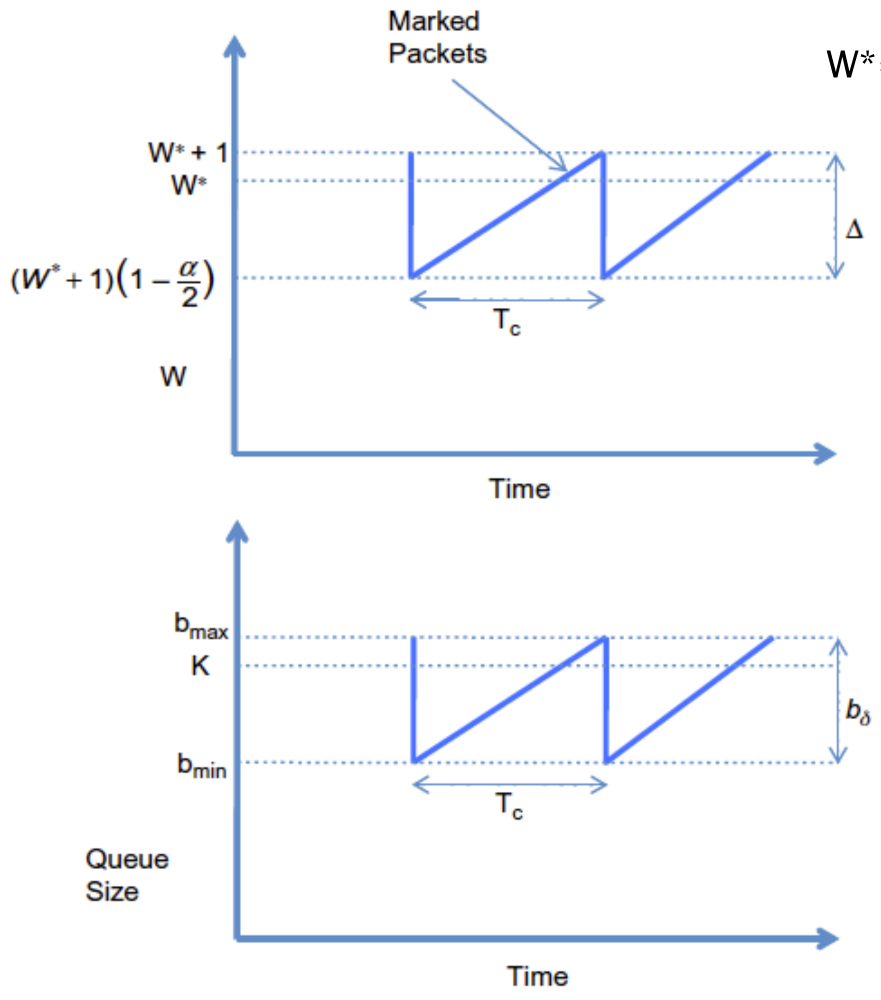
The steady-state fraction of marked packets is given by

$$\alpha = \frac{S(W^*, W^* + 1)}{S((W* + 1)(1 - \alpha/2), W* + 1)} \tag{10}$$

By plugging equations 9 into 10 and simplifying, it follows that

$$\alpha^2\left(1 - \frac{\alpha}{4}\right) = \frac{2W^* + 1}{(W* + 1)^2} \approx \frac{2}{W*} \text{ so that } \alpha \approx \sqrt{\frac{2}{W*}} \tag{11}$$

Window size and bottleneck queue dynamics for DCTCP



W*=(CD+K)/N

When the window increases to W*=(CD+K)/N, t hen the queue size reaches K (because CD+K is the maximum number of packets that can be in transit + waiting for service for a buffer size of K and a single source), and the switch starts to mark packets with the congestion codepoint. However, it takes one more round trip delay for this information to get to the source, during which the window size increases to W*+1.

Steady state fraction of marked packets is given by:

$$\alpha = \frac{S(W^*, W^* + 1)}{S((W^* + 1)(1 - \alpha/2), W^* + 1)}$$

DCTCP Analysis

To compute the magnitude of oscillations in the queue size, we first compute the magnitude of oscillations in window size of a single connection, which is given by

$$\Delta = (W^* + 1) - (W^* + 1)\left(1 - \frac{\alpha}{2}\right) = (W^* + 1)\frac{\alpha}{2}$$

Because there are N flows, it follows that the oscillation in the queue size b_δ, is given by

$$\begin{aligned} b_\delta &= N\Delta = N(W^* + 1)\frac{\alpha}{2} \approx N\sqrt{\frac{W^*}{2}} \\ &= \sqrt{\frac{N(CD + K)}{2}} \end{aligned}$$

DCTCP Analysis

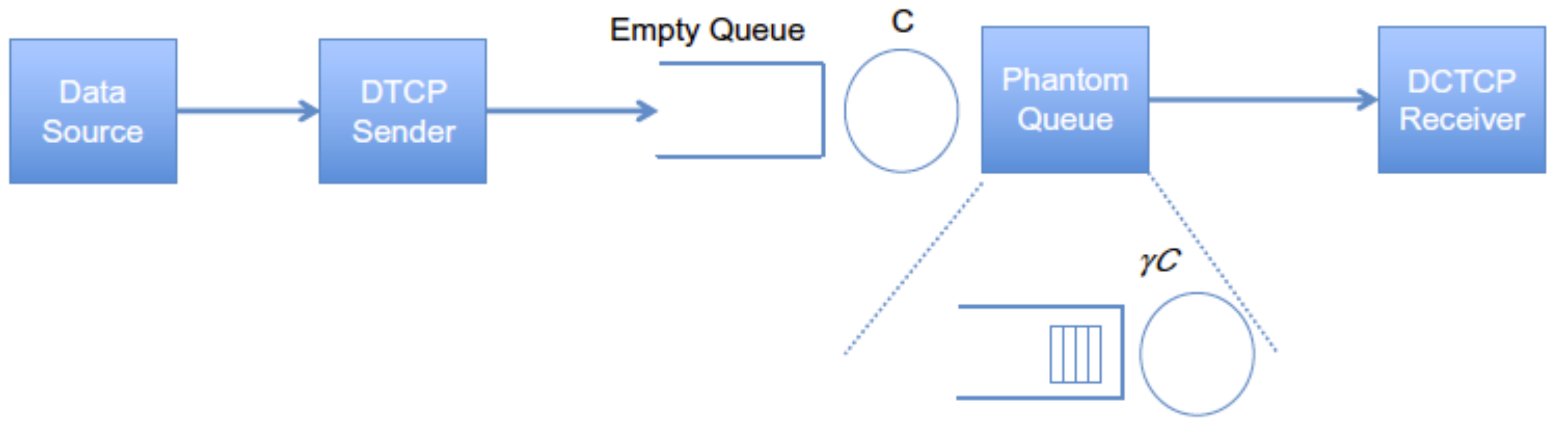
- It follows that the amplitude of queue size oscillations in DCTCP is $O(\sqrt{CD})$ which is much smaller than the oscillations in TCP Reno, which are *O(CD)* .
- This allows for a smaller threshold value K, without the loss of throughput. Indeed, the minimum value of the queue size b_{min}, can also be computed and is given by

$$
\begin{aligned}
b_{min} &= b_{max} - b_{\delta} \\
&= K + N - \sqrt{\frac{N(CD+K)}{2}}
\end{aligned}
$$

To find a lower bound on K, we can minimize this equation over N and then choose K so that this minimum is larger than zero, which results in

$$K > \frac{CD}{7} = 0.14CD$$

Implementation of the Phantom Queue



Phantom queue operating at 10% lower capacity

It is possible to reduce the flow latencies across a DCN even further than DCTCP by signaling switch congestion based on link utilizations rather than queue lengths

Phantom Queue with DCTCP

- Consider a simulated queue called Phantom Queue (PQ), which is a virtual queue associated with each switch egress port, and in series with it.
- Note that the PQ is not really a queue because it does not store packets; however, it is simply a counter that is updated as packets exit the link to determine the queuing that would have occurred on a slower virtual link (typically about 10% slower).
- It then marks the ECN for packets that pass through it when the simulated queue is above a fixed threshold. The PQ attempts to keep the aggregate transmission rate for the congestion-controlled flows to be strictly less than the link capacity, which keeps the switch buffers mostly empty.
- The system also uses DCTCP as its congestion control algorithm to take advantage of its low latency properties.
- To further reduce the queue build-up in the switches, HULL also implements Packet Pacing at the sender. The pacer should be implemented in hardware in the server Network Interface Card (NIC) to smooth out the burstiness caused by the data transfer mechanism between the main memory and the NIC.
- As a result of these enhancements, HULL has been shown achieve 46% to 58% lower average latency compared with DCTCP, and 69% to 78% slower 99th percentile latency.

Deadline Aware Congestion Control Algorithms

Does DCTCP Meet Deadlines?

- Flows in DCNs come with real-time constraints on their completion times, typically of the order of tens of milliseconds.
- One of the issues with DCTCP is that it does not take these deadlines into account; instead, because of its TCP heritage, it tries to assign link bandwidth fairly to all flows irrespective of their deadlines.
- As a result, it has been shown that as much as 7% of flows may miss their deadlines with DCTCP.
- Two algorithms that take deadlines into account
  - Deadline Aware Datacenter TCP (D^2TCP)
  - D^3

D²TCP

The basic idea behind D²TCP is to modulate the congestion window size based on both deadline information and the extent of congestion. The algorithm works as follows:

- As in DCTCP, each switch marks the CE bit in a packet if its queue size exceeds a threshold K. This information is fed back to the source by the receiver though ACK packets.

- Also as in DCTCP, each sender maintains a weighted average of the extent of congestion α, given by

  $$\alpha \leftarrow (1-g)\alpha + gF$$

  where F is the fraction of marked packets in the most recent window and g is the weight given to new samples.

- D²TCP introduces a new variable that is computed at the sender, called the deadline imminence factor d, which is a function of a flows deadline value, and such that the resulting congestion behavior allows the flow to safely complete within its deadline.

- Define T_c as the time needed for a flow to complete transmitting all its data under a deadline agnostic behavior, and δ as the time remaining until its deadline expires. If Tc >δ, then the flow should be given higher priority in the network because it has a tight deadline and vice versa. Accordingly, the factor d is defined as

$$d = \frac{T_c}{\delta}$$

D²TCP: Computing d

$$d = \frac{T_c}{\delta}$$

- Note that δ is known at the source.
- To compute T_c, consider the following: Let X be the amount of data (in packets) that remain to be transmitted and let W_m be the current maximum window size.
- Recall from the DCTCP analysys that the number of packets transmitted per window cycle N is given by

$$N = S\left((W_m + 1)\left(1 - \frac{\alpha}{2}\right), W_m + 1\right) = \frac{W_m^2}{2}\alpha\left(1 - \frac{\alpha}{4}\right) = \sqrt{\frac{W_m^3}{4} - \frac{W_m}{4}}\text{packets/cycle}$$

so that $N \approx \frac{W_m^{1.5}}{2}$

- Hence, the number round trip latencies M in a window increase decrease cycle is given by $M = \frac{X}{N} \approx \frac{2X}{W_m^{1.5}}$
- The length τ of a cycle is given by $\tau = TW_m\frac{\alpha}{2} \approx T\sqrt{\frac{W_m}{2}}$
- It follows that Tc is given by $T_c = M\tau = \frac{X\sqrt{2}}{W_m}T$

so that

$$d = \frac{X\sqrt{2}}{W_m\delta}T$$

D²TCP

$$\alpha = P(b > K)$$

$$d = \frac{T_c}{\delta}$$

- Based on α and d, the sender computes a penalty function p applied to the window size, given by

$$p = \alpha^d \tag{23}$$

The following rule is then used for varying the window size once every round trip

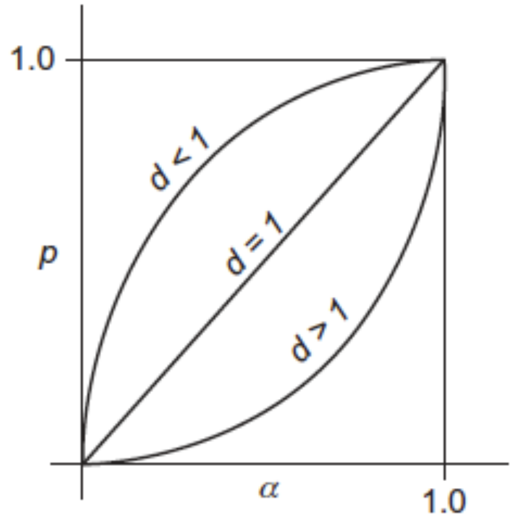
$$W \leftarrow \begin{cases} W+1, & if\ p = 0 \\ W(1-\frac{p}{2}), & if\ p > 0 \end{cases}$$

- When α=1, then p=1, and the window size is halved just as in regular TCP or DCTCP.

- For 0<α<1, the algorithm behaves differently compared with DCTCP, and depending on the value of d, the window size gets modulated as a function of the deadlines.

D²TCP Correction Function

$$W \leftarrow \begin{cases} W+1, & \text{if } p=0 \\ W(1-\frac{p}{2}), & \text{if } p>0 \end{cases}$$



$$\alpha = P(b > K)$$

$$d = \frac{T_c}{\delta}$$

$$p = \alpha^d$$

- When d=1, then p = α, so that the system matches DCTCP.
- If d>1, it follows that the time required to transmit the remaining data is larger than allowed by the deadline; hence, the flow should be given higher priority by the network. The equation $p = \alpha^d$ rings this about by reducing the value of p for such a flow, resulting in a larger window size.
- Conversely, d<1 leads to a smaller window size and hence lower priority for the flow.
- Hence, the net effect of these window change rules is that far-deadline flows relinquish bandwidth so that near-deadline flows can have greater short-term share to meet their deadlines.

Vamanan et al. [3] simulated D²TCP and showed that the algorithm reduces the fraction of missed deadlines by 75% when compared to DCTCP and 50% compared with D³. It is also able to coexist with TCP flows without degrading their performance.

Multipath TCP
MPTCP

MPTCP

- One of the features that differentiates a DCN from other types of networks is the presence of multiple paths between end points. In order to realize the full bisection bandwidth between the end points, the system should be able to spread its traffic among the multiple paths while at the same time providing congestion control along each of the paths.

- Traditional Way:
  - Do Load Balancing on a per flow basis, with each flow mapped randomly to one of the available paths. This is usually done with the help of routing protocols, such as ECMP, that use a hash of the address, port number and other fields to do the randomization.
  - However, randomized load balancing cannot achieve the full bisectional bandwidth in most topologies because often a random selection causes a few links to be overloaded and other links to have little load.

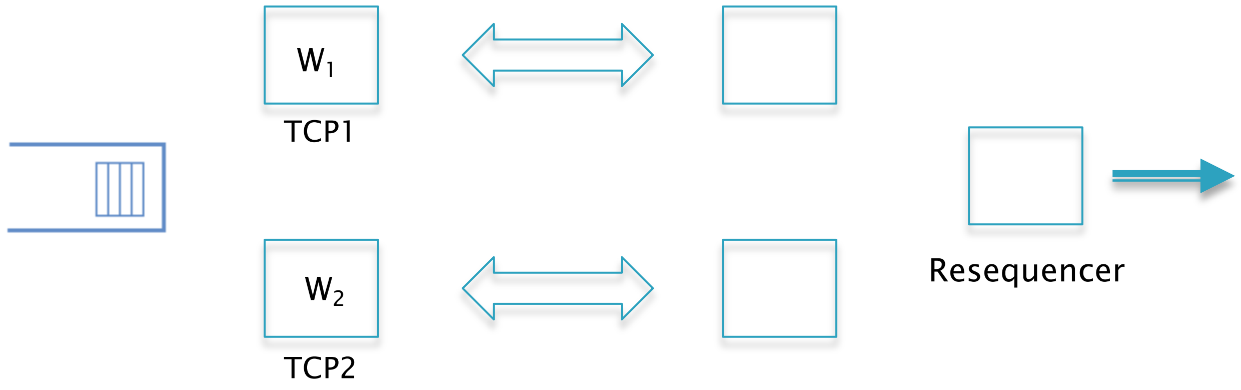
- MPTCP combines a better way of doing Load Balancing along with Congestion Control.
  - It uses a simple yet effective mechanism to link the congestion control dynamics on the multiple sub-flows, which results in the movement of traffic away from more congested paths and on to less congested ones.

MPTCP Operation

- MPTCP support is negotiated during the initial SYN exchange when clients learn about additional IP addresses that the server may have. Additional sub-flows can then be opened with IP addresses or in their absence by using different ports on a single pair of IP addresses. MPTCP then relies on ECMP routing to hash the sub-flows to different paths.

- After the multiple sub-flows have been established, the sender's TCP stack stripes data across all the sub-flows. The MPTCP running at the receiver reconstructs the receive data in the original order. Note that there is no requirement for an application to be aware that MPTCP is being used in place of TCP.

- Each MPTCP sub-flow have its own sequence space and maintains its own congestion window so that it can adapt independently to conditions along the path.



$$W_T = W_1 + W_2$$

MPTCP Window Increment Decrement Rules Version 1: Static Splitting

- Recall that for AIMD(a,b) system the max Window Size is given by

$$W_m = \sqrt{\frac{2a}{(2b - b^2)}\frac{1}{p}}$$

- If we choose a = $1/n^2$, and assuming equal T, results in an equilibrium window size of W/n for each subflow.

- However, this algorithm can result in suboptimal allocations of the subflows through the network because it uses static splitting of the source traffic instead of adaptively trying to shift traffic to routes that are less congested.

MPTCP Window Increment Decrement Rules Version 2: Dynamic Splitting

Adaptive shifting of traffic to less congested routes can be done by the using the following rules:

$$W_r \leftarrow W_r + \frac{1}{W_T} \text{ on every ACK, and} \tag{28}$$

$$W_r \leftarrow \frac{W_T}{2} \text{ on detecting packet loss} \tag{29}$$

- Each subflow increases its window by (W_r/W_T) per roundtrip, so the the total across all flows will increase by 1 per RTT
- Consider the case when the packet drop rates are not equal. The window increment and decrement amounts are the same for all paths; hence, it follows that the paths with higher drop rate will see more window decreases, and in equilibrium, the window size on these paths will go to zero.

This is a problem

MPTCP Window Increment Decrement Rules Version 2: Dynamic Splitting

Assuming that each of the paths have the same packet loss rate p and using the argument that in equilibrium, the rates of increases and decreases of the window size must balance out,
it follows that

$$\left[\frac{W_r}{T_r}(1-p)\right]\frac{1}{W_T} = \left[\frac{W_r}{T_r}p\right]\frac{W_T}{2}$$

so that

$$W_T = \sqrt{\frac{2(1-p)}{p}} \approx \sqrt{\frac{2}{p}}$$

- It follows that the total window size W_T, is the same as for the case when all the traffic was carried on a single path.
- Hence, unlike the case of static splitting, the adaptive splitting rule does not use more bandwidth by virtue of the fact that it is using multiple paths.

MPTCP Window Increment Decrement Rules Version 3: Improved Dynamic Splitting

- MPTCP Version 2 leads to the situation where there is no traffic directed to links with higher packet drop rates.
- If there is no traffic going to a path for a subflow, then this can be a problem because if the drop rate for that link decreases, then there is no way for the subflow to get restarted on that path.
- To avoid this, it is advisable to have traffic flow even on links with higher loss rates,

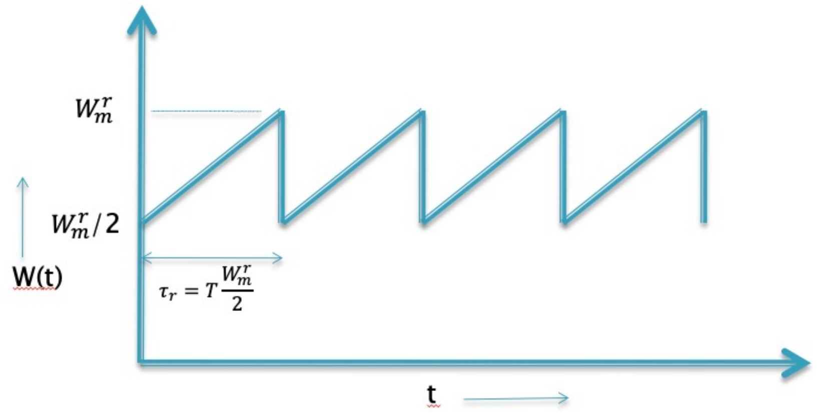
Modified increment-decrement rule

$$W_r \leftarrow W_r + \frac{a}{W_T} \text{ on every ACK and}$$

$$W_r \leftarrow \frac{W_r}{2} \text{ on detecting packet loss}$$

Note that this causes a decrease in the rate at which the window decreases for higher loss rate links, and the factor a increases the rate of increase.

MPTCP Window Increment Decrement Rules Version 3: Formula for W_r



M_r: Multiple of RTTs contained in an increase−decrease cycle
N_r: Number of packets transmitted an increase−decrease cycle for the rth subflow

Because the window size increases by aW_r/W_T for every RTT and the total increase in window size during a cycle is $W_r/2$, it follows that

$$M_r = \frac{W_r/2}{aW_r/W_T} = \frac{W_T}{2a}$$

so that

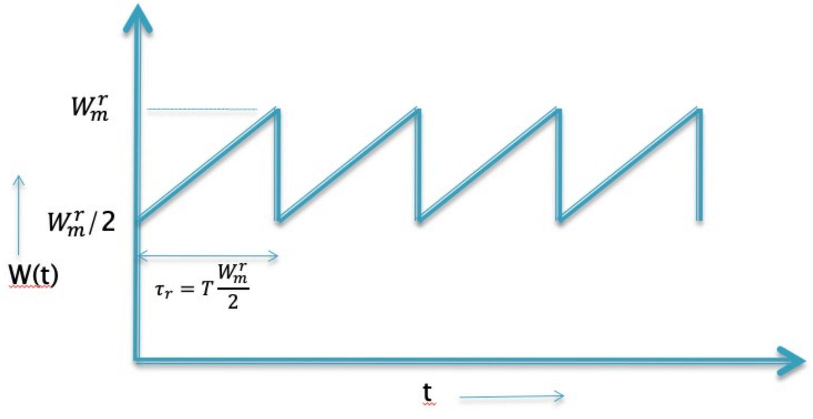
$$\tau_r = M_r T = \frac{W_T T}{2a}$$

Also

$$N_r = \frac{1}{T}\int_0^{\tau_r} W(t)dt = \frac{1}{T}\left[\frac{\tau_r W_r^m}{2} + \frac{\tau_r W_r^m}{4}\right] = \frac{3W_T W_r^m}{8a}$$

Equating N_r to $1/p_r$, it follows that $$\frac{3W_T W_r^m}{8a} = \frac{1}{p_r}$$

MPTCP Window Increment Decrement Rules Version 3: Formula for W_r



M_r: Multiple of RTTs contained in an increase−decrease cycle
N_r: Number of packets transmitted an increase−decrease cycle for the r^th subflow

$$\frac{3W_T W_r^m}{8a} = \frac{1}{p_r}$$

It follows that $W_T = \sum_s W_s^m = W_r^m \sum_s \frac{p_r}{p_s}$

Substituting for W_T back into prior equation, it follows that

$$W_r^m = \sqrt{\frac{8a}{3}} \frac{1/p_r}{\sqrt{\sum_s 1/p_s}}$$

This algorithm allocates a nonzero window size to flows with larger packet drop rates.

MPTCP Window Increment Decrement Rules Version 4: Flows with Differing RTTs

- The window increment decrement rules are effective in guiding traffic toward links with lower loss rates; however, they do not work very well if the paths have differing round trip delays.
- This is because the path with the higher round trip delay will experience a lower rate of increase of window size, resulting in a lower throughput even if the packet loss rates are the same.

Impose the condition $$\sum_r \frac{W_r}{T_r} = \max_r \frac{W_r^{TCP}}{T_r}$$

Where W_r^{TCP} is the window size attained by a single-path TCP experiencing path r's loss rate. This implies:

1. The multipath flow takes at least as much capacity as a single path TCP flow on the best of the paths, and

2. The multipath flow takes no more capacity on any single path (or collection of paths) than if it was a single path TCP using the best of those paths.

MPTCP Window Increment Decrement Rules Version 4

$$W_r \leftarrow W_r + \min\left(\frac{a}{W_T}, \frac{1}{W_r}\right) \text{ on each ACK}$$

$$W_r \leftarrow \frac{W_r}{2} \qquad \text{on packet drop}$$

where a is given by

$$a = W_T^m \frac{\max_r \frac{W_r^m}{T_r^2}}{\left(\sum_r \frac{W_r^m}{T_r}\right)^2}$$

The difference between this algorithm and the Version 3 is that window increase is capped at $1/W_r$, which means that the multipath flows can take no more capacity on any path than a single-path TCP flow would.

Version 3

$$W_r \leftarrow W_r + \frac{a}{W_T} \text{ on every ACK and}$$

$$W_r \leftarrow \frac{W_r}{2} \text{ on detecting packet loss}$$

MPTCP Window Increment Decrement Rules Version 4: Derivation of *a*

Using the same notation as before, note that the window size for the r^th^ subflow increases by $\min(\frac{aW_r}{W_T}, 1)$ in each round trip. Hence, it follows that

$$M_r \min\left(\frac{aW_r^m}{W_T^m}, 1\right) = \frac{W_r^m}{2} \text{ so that}$$

$$M_r = \begin{cases} \frac{W_T^m}{2a} & \text{if} \quad aW_r^m < W_T^m \\ \frac{W_r^m}{2} & \text{otherwise} \end{cases} \tag{42}$$

It follows that

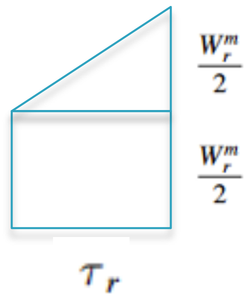
$$\tau_r = M_r T_r \quad \text{and} \quad N_r = \frac{3}{4T_r} W_r^m \tau_r \tag{43}$$

Using the deterministic approximation technique, it follows that

$$\frac{3}{4T_r} W_r^m \tau_r = \frac{1}{p_r} \tag{44}$$

Using the deterministic approximation technique, it follows that

$$\frac{3}{4T_r} W_r^m \tau_r = \frac{1}{p_r}$$



MPTCP Window Increment Decrement Rules Version 4: Derivation of *a*

$$\boxed{\sum_r \frac{W_r}{T_r} = \max_r \frac{W_r^{TCP}}{T_r}}$$

Note that the window size for the rth sub-flow increases by $\min(\frac{aW_r}{W_T}, 1)$ on each roundtrip. Hence it follows that

$$M_r \min\left(\frac{aW_r^m}{W_T^m}, 1\right) = \frac{W_r^m}{2} \text{ so that}$$

$$M_r = \begin{cases} \frac{W_T^m}{2a} & \text{if} \quad aW_r^m < W_T^m \\ \frac{W_r^m}{2} & \text{otherwise} \end{cases}$$

$$\boxed{\frac{3}{4T_r} W_r^m \tau_r = \frac{1}{p_r} \qquad \tau_r = M_r T_r}$$

Since $\tau_r = M_r T_r \quad \text{and} \quad N_r = \frac{3}{4T_r} W_r^m \tau_r$ it follows that $\frac{3}{4T_r} W_r^m \tau_r = \frac{1}{p_r}$

Since $W_r^{TCP} = \frac{1}{T_r}\sqrt{\frac{8}{3p_r}}$ it follows that $\left(\sum_r \frac{W_r^m}{T_r}\right)^2 = \max_r \frac{1}{T_r^2} * \frac{8}{3p_r}$

Substituting for p_r and τ_r from prior page, we finally obtain

$$\left(\sum_r \frac{W_r^m}{T_r}\right)^2 = \max_r \frac{2}{T_r^2} W_r^m \tau_r = \max_r \frac{2}{T_r^2} W_r^m T_r \frac{W_T^m}{2a},$$

from which $a = W_T^m \frac{\max_r \frac{W_r^m}{T_r^2}}{\left(\sum_r \frac{W_r^m}{T_r}\right)^2}$ follows

M_r: Multiple of RTTs contained in an increase–decrease cycle
N_r: Number of packets transmitted an increase–decrease cycle for the rth subflow

Further Reading

- Chapter 7 of Internet Congestion Control