Sorbonne Université

Master Thesis

Plasticine: A flexible buffer management scheme for data center networks

Vamsi Addanki *(3875663)*
Sorbonne Université

Supervisors:
Prof. Laurent Vanbever
ETH Zurich
Maria Apostalaki
ETH Zurich
Prof. Sebastien Tixeuil
Sorbonne Université

September 12, 2020

Contents

List of Figures

Acknowledgments

I would like to thank Prof. Laurent Vanbever for the thesis proposal and giving me the opportunity to work under his supervision. I would also like to thank Maria Apostalaki, a PhD student supervised by Prof. Laurent Vanbever, for her supervision and intellectual support. Finally, I would thank Sorbonne Université administration and academic supervisor Prof. Sebastien Tixeuil who enabled a path towards my research interests.

Work-place, team and contribution

My thesis work is carried as part of *Networked Systems Group (NSG)* lab at ETH Zurich, Switzerland. The lab is headed by Prof. Laurent Vanbever.

The work being presented in this thesis is a result of collective work among Maria Apostalaki, Myself, Prof. Laurent Vanbever and Prof. Manya Ghobadi. My specific contributions in this thesis are formulating the algorithm (with support from Maria Apostalaki), mathematical analysis of our solution, establishing it's bounds and evaluation.

Fig. 5.1, 3.1a, 3.1b, 2.1 are taken from Maria Apostalaki. Some of the content from the on-going work on writing a paper, has been re-used in this thesis.

Abstract

Network devices today share buffer across output queues to avoid drops during transient congestion with less buffer space, thus with a lower cost, per chip. While effective most of the time, this sharing can cause undesired interactions among seemingly independent traffic, especially in case of high load. As a result, low-priority traffic can cause increased packet loss to high-priority traffic, intra-DC traffic can impair WAN throughput and long flows can prevent the buffer for absorbing incoming bursts. The cause of this perhaps unintuitive outcome is today's buffer sharing techniques that are unable to guarantee isolation even to a small portion of the traffic, without statically allocating buffer space. To address this issue we designed Plasticine a novel buffer management scheme which offers strict isolation guarantees without keeping the buffer idle and is practical in today's hardware. We found that Plasticine: *(i)* significantly improves query completion times ($\approx>$ 30% compared to state-of-the-art solution) while achieving on-par throughput compared to convectional buffer management algorithms as well as TCP nuances; *(ii)* improves short-flow completion times; Our proposal is the first attempt to address the problems of bursts in today's data center networks from a buffer management perspective.

Chapter 1

Introduction

The internet is majorly a packet switched network and works as a best-effort service. As opposed to traditional circuit switched networks, transmission in packet switched network can happen at any time and does not require reservation of resources such as bandwidth. The incoming traffic at each hop in such networks is unpredictable and not necessarily bounded by the outgoing transmission capacity limits. This leads to transient congestion events and packet loss. As a result, network devices make use of buffer space to reduce packet drops during transient congestion. Further, in-order to maximize the output link utilization, numerous studies in the past discuss the appropriate buffer sizing per port [18] [12] [8] based on dynamics of the widely used *TCP* protocol.

Due to cost, unpredictable needs and implementation complexity modern devices resort to buffers that are shared among all ports [10, 1, 7, 18, 17]. This unavoidably requires a *Buffer Management* algorithm that splits the buffer among all the queues. A buffer management algorithm can be as conservative as diving the buffer into equal parts among all queues or as liberal as allowing every queue to use as much buffer as available. The former leads to under-utilization of buffer and the latter leads to unfair and over-utilization of buffer.

1.1 Good old Moore's law days and ATM

Decades ago, buffer space wasn't a scare resource with moore's law governing the relation between memory and capacity growth. A buffer management algorithm had four basic requirements for it's allocation scheme *(i)* fairness, *(ii)* maximize buffer utilization, *(iii)* adapt to load variations and *(iv)* isolation.

Fairness Sharing network resources fairly improves performance and enables QoS guarantees [25]. A buffer management algorithm should allocate buffer fairly to traffic classes of equal priority across all the ports. Additionally the algorithm should support weighted fair allocation among different traffic classes across all the ports.

Maximize buffer utilization A buffer management algorithm should not dissipate network resources and should allow for fully utilizing the available buffer space to *(i)* reduce

packet loss, *(ii)* maximize link utilization.

Adapt to load variations Changes in load conditions leads to growth in queue lengths. A buffer management algorithm, while aiming to maximize utilization, should also adapt to load conditions and be able to adjust the allocations across all the queues to accommodate the load variations on a given queue without losses until a steady state fair allocation is reached.

Isolation Satisfying loss requirements and proving certain isolation offers guarantees, deterministic and predictable performance. A buffer management algorithm should effectively isolate and guarantee buffer space for high priority traffic classes.

ATM networks allowed buffer management algorithm to easily achieve the above requirements mainly due to the design and available information in such networks.

- ATM networks used call admission control (CAC) that enabled deriving numerous parameters for a buffer management algorithm to consider such as buffer occupancy per connection, per loss priority etc. [16],
- Loss priorities were known prior and instantly available to a buffer management algorithm for each arriving cell. [14]
- Isolation was easily possible as a result of reservations. [16]

Such information is not available in today's IP networks and makes buffer management significantly challenging to satisfy the above key requirements. Furthermore, the traffic volume and diversity in traffic-mix have significantly changed over the years.

1.2 End of Moore's law and DCNs

Numerous observations from research and industry have been made in the wake of end of moore's law [23]. One such observation includes the mismatch in the growth of on-chip memory and transmission capacity. As a result, due to cost reasons, today's data center networks resort to shallow buffers. Additionally, traffic patterns in today's data center networks are highly unpredictable and have widely varying needs leading to two key requirements from a buffer management algorithm that are orthogonal to each other.

Minimize buffer utilization The traffic in data center networks is majorly comprised of short flows ($< 100KB$), latency and loss sensitive flows that are of highest priority [5] [21]. In order to reduce packet drops from such flows, the buffer needs to maintain enough headroom i,e. remaining space. As a result, the buffer allocation needs to be minimized and bounded such that low priority traffic and long flows are limited in the buffer they can use.

Maximize buffer allocation In addition to short-flows and deadline flows, data center traffic consists of micro-bursts which requires a large amount of buffer space for a short duration of time. In-order to accommodate bursts, the buffer allocation needs to be maximized such that a burst always finds enough buffer space avoiding packet drops.

Reducing and bounding the allocation leads to a higher available buffer for high priority flows. However, the reduced allocation itself limits the performance of high priority flows since they cannot make use of the actual remaining space.

While the recent advancements in congestion control aim at mitigating this issue, the problem still remains under high loads where the inter-flow arrival times reduces resulting in high arrival rates which cannot be controlled. Furthermore, congestion control algorithms have no means of feedback to indicate the overall buffer state in the network device. As a result, a buffer management algorithm is not allowed to allocate high amount of buffer to a single port, making a loop back into the orthogonal requirement problem i,e. minimizing and maximizing at the same time.

1.3 Buffer management: forgotten since 1997

Dynamic Threshold (DT) [9] that was initially proposed in 1997 is still the state-of-the-art and most commonly used buffer management algorithm by multiple vendors and operators such as Broadcom [10], Cisco [2], Yahoo [15], Alibaba [19]. In particular, DT allocates space per queue based on unused buffer space and a static parameter. As a result, DT does not control the absolute remaining buffer, meaning that its burst-tolerance and achieved throughput is dependent on the input traffic. Worse yet, DT treats buffer occupancy as a scalar unit, ignoring how it evolves. This can lead to extreme unfair scenarios, as miss-behaving traffic can easily monopolize the buffer.

An advanced version of Dynamic Threshold was also proposed inorder to serve multiple loss priorities [14], that was capable of allocating more buffer to highly loss sensitive traffic classes and less buffer to loss-tolerant traffic classes. The requirements for Dynamic Threshold for Multiple loss priorities: *(i)* Prior knowledge of loss priority, *(ii)* Number of packets belonging to a traffic class occupying buffer in a queue and *(iii)* Overall buffer occupancy of a traffic class across all the ports. As mentioned earlier, such information was available in ATM networks from CAC but IP networks do not provide with such information. The most commonly used version of Dynamic threshold is the one originally proposed for single loss priority making it an allocation scheme that is oblivious to traffic classes and network priorities.

Furthermore, in addition to the mentioned problems, there are certain fundamental problems with DT's allocation scheme and does not satisfy the needs of today's data center networks, which we will discuss in detail in Chapter. 2

So far the shortcomings of buffer management algorithms are well hidden behind poor monitoring. Instrumenting buffer occupancy to observe its workings in commodity devices is proven to be either too coarse-grained or unreliable. Indeed, a recent measurement study in Yahoo [15] only reports buffer occupancy per minute, while finer-grained measurements in Facebook, resulted in high rates (1%-100%) of missed intervals.

We believe it is high time to revisit buffer management for three main reasons. First, buffer cost starts to become an issue [18]. Second, buffer management algorithms that are used today were designed with assumptions about the network that do not hold anymore, e.g., infrequent bursts, single first-in-first-out queues, prior notion of loss priority and

resource reservations, etc. Finally, programmable devices make the first step towards accurate buffer monitoring by exposing buffer metrics, even though they keep the buffer management away from the programmer's hands.

1.4 Contributions

To address all the issues mentioned in the previous sections, we design Plasticine. Plasticine is a practical buffer management algorithm that achieves high-burst tolerance guarantees, without sacrificing throughput by dynamically allocating the buffer. Plasticine is also fair among queues while respecting their relative priority. Plasticine achieves these using three key insights. First, aggregated buffer occupancy needs to be bounded to be predictable. Second, per-port buffer allocation should be proportionate to the port's service rate, as we know from the long conversation about buffer sizing [13, 11, 18]. Third, the traffic of equivalent priorities needs to be treated fairly.

This thesis is organized as follows. In the next Chapter, we describe in detail the problems of state-of-the-art buffer management schemes and inefficiencies of congestion control. We give an overview of our proposal Plasticine in Chapter. 3 and analyze its key properties, bounds in Chapter. 4. In Chapter. 5, we evaluate Plasticine with simulations using a data center topology and realistic workloads. Chapter. 6 concludes this thesis.

Chapter 2

Background and Motivation

In this chapter, we first give a brief description of the context and architecture that we will refer to in the rest of this thesis. Later in this chapter, we discuss *(i)* The nature of DCN traffic and it's buffer requirements *(ii)* The role of congestion control, it's buffer requirements, where it shines and where it fails, *(iii)* Most common buffer-management schemes and inefficiencies. Overall the aim of this chapter is to showcase the problems of buffer-management that have been overlooked so far.

2.1 Preliminaries

Fig. 2.1 shows a simplistic architecture of a shared memory switch focusing on the components *(i)* ports, *(ii)* queues, *(iii)* enqueue, *(iv)* dequeue, *(v)* priority and *(vi)* shared memory.

Ports A port is a physical network interface card that receives and transmits packets on to the wire. It is characterized by it's transmission capacity i.e., the maximum rate at which it can transmit packets on to the wire. Routing tables determine the output port for an input packet. When the packet arrival rate at a port exceeds it's transmission capacity, packets are buffered in a queue.

Queues A port is equipped with a single or multiple queues. The commonly deployed queue structure in the network devices is a First-In-First-Out (*FIFO*) queuing discipline. *FIFO* queues are easy to implement in hardware as they only require a *push* at the tail and *pop* at the head operations. With a single queue per port, all the packets arriving at the output port are pushed into the same queue. For the sake of isolation, one could prefer a multiple queue per port architecture. Classification techniques (for example, based on traffic class mapping) determine the queue at a given output port for buffering a packet.

Enqueue Enqueue corresponds to a *push* operation for a queue. A switch's memory is limited and hence queues have thresholds i.e., the maximum number of packets buffered in a queue is limited by it's assigned threshold. Upon the arrival of a packet, the Traffic Manager (TM) decides whether it will be enqueued or dropped. To that end, TM compares the length of the queue that the packet is destined to with its assigned threshold. If the queue is shorter than this threshold, the packet is enqueued. To do so, the TM

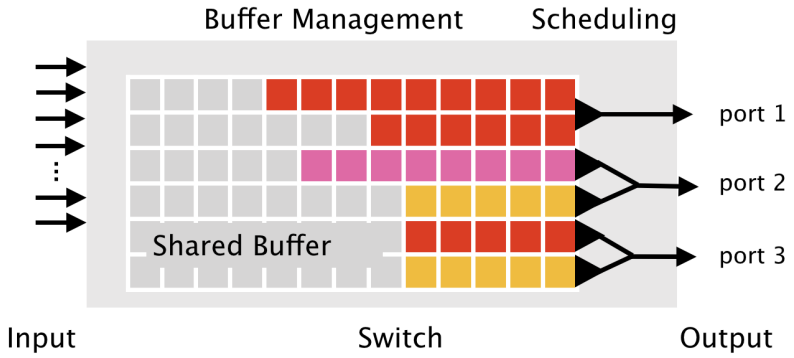


Figure 2.1: Input traffic is distinguished per priority (color). Each port has a different queue for each priority. Queues share the buffer at the device-level and the bandwidth at the port-level according to the buffer management and scheduling respectively.

stores the packet in the shared buffer until the packet is transmitted and performs a *push* operation, effectively adding a pointer to a linked list that corresponds to the queue.

Dequeue Dequeue corresponds to a *pop* operation for a queue. Unlike enqueue, the dequeue rate is limited by a port's transmission capacity. In a single queue per port architecture, the dequeue rate of a queue equals the port's transmission capacity. In a multi queue per port architecture, the TM uses a scheduling algorithm to select a non empty queue of a port to perform dequeue operation. Nevertheless, the scheduling algorithm ensures that the total dequeue rate of all the queues of a port equals the port's transmission capacity (unless explicit reservations are made). The most commonly used scheduling algorithms are Round Robin (RR) and Strict Priority (SP). In RR, the dequeue operations happen in a cyclic manner from all the non-empty queues of a port. As a result, the transmission capacity of the port is effectively split equally among the active/non-empty queues. When using SP, each queue is assigned a priority (see next paragraph) and SP always selects the highest priority queue that is non-empty for a dequeue operation. SP offers certain isolation in terms of latency i.e., the queuing delays of high priority queues are not effected by low priority queues. A common mis-conception is that priority queuing isolates high priority traffic. We shall show later in this chapter how poor buffer management leads to cross-priority interference leading to high priority packet drops.

Priority In this work, we refer to priority only in two contexts *(i)* Buffer and *(ii)* Scheduling. Queues are assigned priorities and treated preferentially either for buffer or scheduling. The switch is often shared among multiple applications or tenants with different performance and/or isolation needs. As an intuition, a delay-sensitive application requires queue isolation and preferential dequeue to minimize queuing delays, whereas a loss-sensitive application requires more buffer to avoid losses. Similarly, one application might be more critical for the operator and need not be influenced by other traffic, essentially requiring stricter isolation guarantees. A high priority in the context of buffer

relates to higher thresholds for the corresponding queue. In the context of scheduling a priority relates to preferential dequeue operations from the corresponding queue. Often both the priority notions correlate with each other in network policies.

Shared Memory All the queues in a switch share a common memory space to buffer packets. Denote the queue length of a queue at port i with priority p as Q^i_p, and a total buffer space of B. The shared memory poses the following restriction i.e., the sum of all the queue lengths is less than or equal to the total buffer space.

$$\Sigma_i \Sigma_p Q^i_p <= B \tag{2.1}$$

In a nutshell, given the above restriction, the job of a buffer management algorithm is to control the thresholds for each queue so as to achieve the key requirements discussed in Chapter. 1. Eq. 2.1 makes it clear why a poor buffer management can lead to high-priority packet drops i.e., if low priority queues tend to occupy significantly large amount of buffer leaving no buffer for high priority, even though it is thought to be isolated due to priority queuing and SP scheduling.

2.2 Datacenter traffic and bursts

The data center traffic characteristics are quite different from internet traffic. Recent studies reveal measurements from Microsoft, Facebook, Google [5] [21]. The traffic can be classified in to three main categories *(i)* Short flows, *(ii)* Long flows and *(iii)* Bursts each having significantly different buffer requirements. Fig. 2.2 shows the flow-size distributions of Websearch [5], cache-follower and hadoop workloads [21].

Short flows A majority of flows in data center traffic are short flows. In particular, websearch workload consists of 70% of flows which are less than a 100KB in size. Short flows do not consume much buffer and a short flow completion time is desirable. Packet loss for such flows significantly impacts their flow completion times as a result of timeouts.

Long flows 90% of data bytes in the traffic comes from long flows while only 30% of the flows are long. Long flows add a significant buffer pressure and are tolerant to packet losses. The importance of long flows is their contribution to throughput and link utilization.

Bursts A burst is characterized by a very high arrival rate of packets at the switch for a short duration of time. Bursts originate mainly due to tcp-incast scenarios, common in query-response type of applications. Packet loss and timeouts in a query response significantly impacts the query completion times. As a result, due to the nature of bursts, they require a large amount of buffer space for a short duration of time.

Many ports using buffer is not uncommon A high resolution measurement study on Facebook data centers [24], shows that it is not uncommon that almost all the ports are simultaneously using the buffer. In particular, the study shows, hadoop workload forces towards extreme scenarios such as 100% of hotports in a switch. A large number of ports simultaneously using buffer puts a heavy burden on buffer management.

We will show in the next section, the effects of TCP and recently famed $DCTCP$ on the buffer utilization and their inefficiencies to guarantee burst absorption.

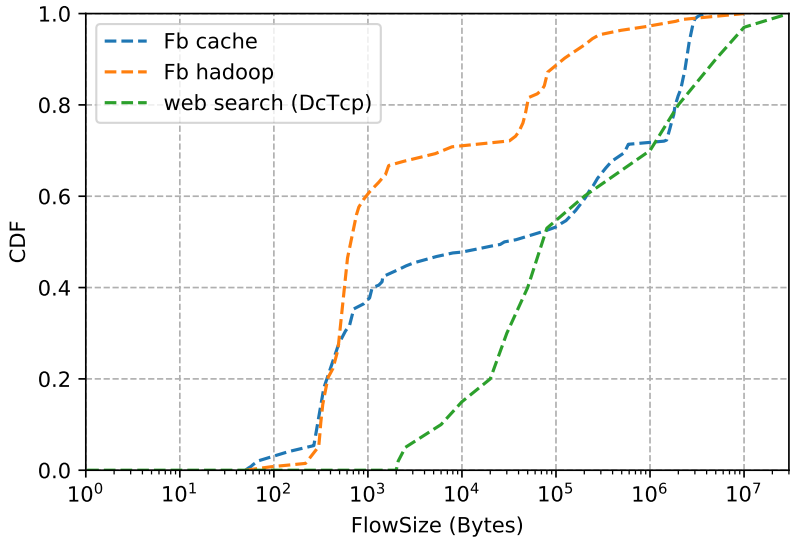


Figure 2.2: Workloads based on measurements from Microsoft and Facebook data center networks.

2.3 Buffer requirement

Buffer requirement is the minimum required buffer in-order to maximize link utilization and throughput. This requirement is heavily dependent on the dynamics of end-host congestion control. The widely used TCP utilizes as much buffer as allocated until a loss is detected i.e., when the buffer overflows, leaving no remaining buffer for important short-flows and bursts. Motivating on this observation, $DCTCP$ was recently proposed which exploits ECN marking to control the buffer utilization of long flows, effectively leaving headroom for short flows and bursts. $DCTCP$ gained fame as it offered a significant improvement over TCP in terms of buffer utilization, short flows and bursts performance. However, as we shall show in this section, $DCTCP$ in-itself cannot control the overall buffer occupancy in a switch and fails to differentiate between traffic classes. As a result, even $DCTCP$ is prone to poor burst absorption capabilities. As mentioned in Chapter 1 we bring to focus the long forgotten buffer management that inhibits burst absorption capabilities even with advanced congestion control algorithms such as $DCTCP$. Later in this work, using our proposed buffer management scheme (Sec. 3), we show a median of 60% improvement in burst absorption (Sec. 5) even with traditional TCP.

2.3.1 TCP

TCP is by default a loss based congestion control algorithm. As a result, the buffer utilization tends to maximum allocated buffer. It is well studied in the past and has been concluded that synchronized TCP flows require a buffer space equal to the Bandwidth Delay Product (BDP) to achieve full link utilization. Data centers have very short RTT values ranging from $100\mu s - 1ms$. As a result, the buffer requirement is significantly lower for example with 1Gbps links, $BDP \approx 25KB$. However, in-order to limit the buffer utilization by long flows to exact BDP, the buffer allocation needs to be set to BDP which inturn dramatically effects the performance of short and bursty flows. Clearly there is a need for differentiation between short and long flows in buffer allocation. $DCTCP$ gets around this problem by using marking schemes to limit the buffer utilization by long flows.

2.3.2 Data Center TCP *(DCTCP)*

A careful observation of traffic characteristic in data centers led to the proposal of $DCTCP$ [5]. In particular, $DCTCP$ exploits ECN marking to reduce the buffer utilization of long flows to a maximum of $K + N$ where $K = BDP/7$ is the marking threshold and N is the number of synchronized long flows. The marking threshold K is a network device configuration where every queue starts marking packets when it's length exceeds the marking threshold. Although $DCTCP$ offers a significant benefit over TCP, it has several downsides such as requirement of a full control over data center, coexistence with TCP, failure to sustain low buffer utilization under high-loads and concurrency.

Observation 1. *DCTCP uses a static K parameter that is determined based on transmission capacity C and round-trip time RTT and does not adapt to multi queue per port architecture. In a single queue per port scenario if a port consumes K+N buffer, the same port with capacity C with 2 queues consumes 2×(K+N) buffer.*

2.3.3 Buffer requirement VS utilization *(Ideal VS Real world)*

The purpose of this section is to bring to focus *(i)* the mismatch in the requirement and utilization of buffer *(ii)* how conventional buffer management limits burst-absorption.

Fig. 2 shows how $DCTCP$'s buffer utilization increases dramatically with increasing load. Although $DCTCP$ is believed to reduce buffer utilization, as load increases it does not satisfy the low buffer occupancy requirements anymore. As a consequence the original purpose of $DCTCP$ i.e., to maintain headroom for short flows and bursts doesn't hold well.

Observation 2. *As the load increases, even DCTCP tends to utilize more buffer, as much as TCP in extreme cases.*

One can argue that at high loads, the buffer allocation can be increased so as to accommodate bursts and short flows. However, the buffer space is limited and further

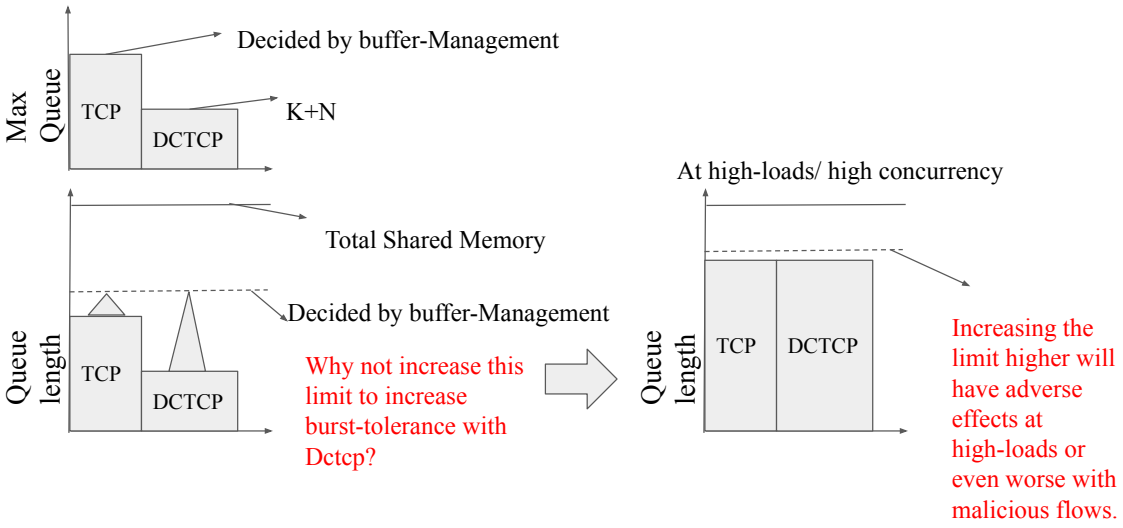


Figure 2.3: Mismatch in the objectives of buffer management that is oblivious to traffic types.

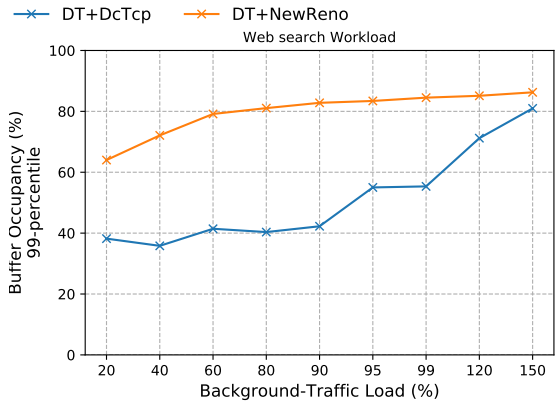


Figure 2.4: The variation of max buffer utilization of TCP and DCTCP using a leaf-spine topology and websearch workload. DT here is the conventional buffer management algorithm (See Sec. 2.4)

increasing or allowing queues to grow as much as possible, has adverse effects. such as unfair buffer utilization across ports, malicious flows monopolizing the buffer etc., .

Observation 3. *Based on Observation. 2 and in addition due to security and fairness concerns, conventional buffer management algorithms should not allocate large amount of buffer to a single queue.*

Finally, based on Observation. 2 and Observation. 3, buffer cannot be allocated arbitrarily large. However bursts require significantly large amount of buffer for a short duration of time. As a result, the burst absorption capabilities of buffer is severely ef-

fected, even in cases where enough remaining buffer space is available.

Observation 4. *Burst absorption even with DCTCP is limited due to conventional buffer management (Dynamic Threshold) which is not allowed to allocate more buffer.*

After understanding the limitations of congestion control in buffer occupancy and burst-tolerance, we were interested in investigating the buffer management itself i.e., the algorithm which assign thresholds to the queues. Indeed, the core of the problem of short flows and burst absorption lies in buffer management algorithm. Additionally, after Observation. 2, 4, 3 one could easily argue about separation of bursts into a different queue. However, this does not help either due to the inefficiencies of buffer management algorithms which will be made clear in the next section.

2.4 Buffer management

2.4.1 Dynamic Threshold *(DT)*

DT [9] allows each queue to grow up to a dynamically-assigned threshold. It computes this threshold for each queue as the product of the unused buffer with a predefined parameter α_p specified for a priority as shown in Eq.2.3 and Fig. 2.5a illustrates the relationship between the α_p parameter, the thresholds computed by DT and the unused/remaining buffer.

$$Threshold = \alpha_p \cdot (RemainingBuffer) \tag{2.2}$$

$$T_p(t) = \alpha_p \cdot (B - Q(t)) \tag{2.3}$$

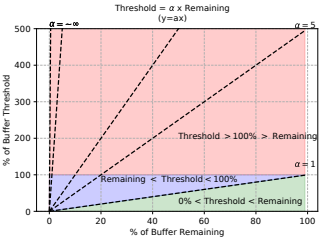
$$\frac{dT_p(t)}{dt} = -\alpha_p \cdot \frac{dQ(t)}{dt} \tag{2.4}$$

B : Total buffer space of the device
$Q(t)$: Total buffer occupancy at time t

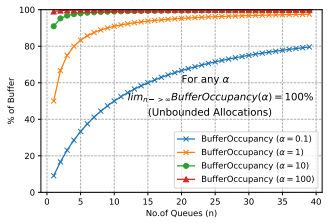
DT is the current default approach, used by multiple vendors, including Broadcom [20]. Its main strength is that it is adaptive to the load. Indeed, buffer allocation by DT is inversely related to the buffer utilization, in other words, directly proportional to remaining buffer. Changes in load conditions leads to changes in queue lengths which in-turn leads to threshold changes of each queue along a straight line corresponding to it's α_p as shown in Fig. 2.5a.

While the operators can define a different α value per queue, they often resolve in defaults or arbitrary selected values since there is no way to optimally configure it. For instance, Yahoo mentions they use $\alpha = 8$ while Cisco has $\alpha = 14$ [15], [2].

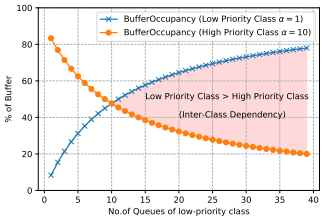
Observe from Eq. 2.3 and Eq. 2.4 that, the threshold calculations are continuous in time. The threshold is effected by total buffer occupancy i.e. the buffer occupancy of all the queues, including the queue under consideration. This dependency of threshold on total buffer occupancy leads to two distinct states, a *Steady-State* when the load conditions are stable. Thus, the queue occupancies are lower than or equal to a threshold and a *Transient-State* when load conditions change drastically, leading to a potential mismatch.



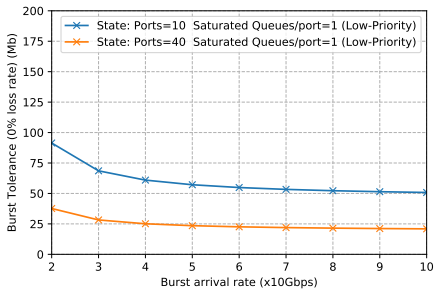
(a) *DT − Algorithm*, linear relationship between the instantaneous buffer-allocation (Threshold) per queue, the remaining buffer and α parameter.



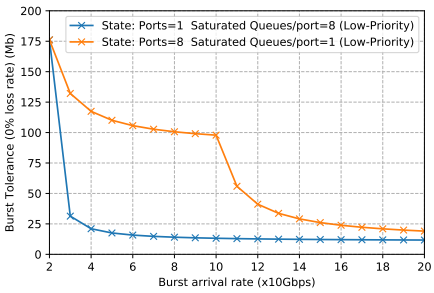
(b) *Classless* Example showing the Unbounded evolution of total buffer allocations/steady-state occupancy by n queues for different α parameter values.



(c) *Classful* Example showing how buffer-allocations for queues of one priority class depend on other priority classes leading to high inter-priority-class dependancy.



(d) Example showing how unbounded-allocations and inter-class-dependence affects burst-tolerance. A comparison of burst-tolerance in a buffer state with 10 saturated queues vs 40 saturated queues.



(e) Example showing how burst-tolerance is effected due to the inability of *DT* to differentiate between two different states with same number of queues but with different aggregate service capacities.

Figure 2.5: DT algorithm and it's inefficiencies

2.4.2 DT's inefficiencies

While dynamic, DT allocates the buffer assuming a best case scenario, namely infrequent changes in load conditions, a single first-in-first-out queue per port. As a result, DT remains incompatible for today's data center networks as we will show in this section. In

particular the incompatibility of DT in today's networks arises where devices have multiple queues per port, multiple priority classes defined by operators, varying scheduling policies and high frequency of micro-bursts (leading to high frequency changes in load conditions).

Unbounded Buffer Allocation The first version of DT was proposed as a *classless* buffer management scheme with single FIFO queue per port where it is desirable to have dynamic allocations which sum up to full utilization of buffer when many queues are saturated. However, the *classful* extension of DT follows the same fundamental algorithm with a difference that each priority class can be mapped to an α_p leading to proportionate buffer allocation based on priority. In Fig. 2.5b, we show the evolution of buffer occupancy with number of queues (n) of a single-class (*classless*) for different values of chosen α_p parameter. The total buffer occupancy in a *classless* scheme is given by $(B - \frac{B}{(1+n \cdot \alpha_p)})$. It can be observed that for any chosen α_p parameter, the buffer occupancy tends to B i.e 100% of buffer, which leads to uncontrollable and unbounded buffer allocations. As a result of applying the same fundamental allocation scheme for multiple priorities (*classful*), different priorities are inter-dependent on the occupancies of each other. In Fig. 2.5c, we show an example with 2 priority classes where one is of high-priority ($\alpha_p = 10$) and the other of low-priority ($\alpha_p = 1$). As the number of saturated queues of low-priority class increase, the buffer-allocation for queues of high-priority class are heavily impacted.

The consequence of unbounded and inter-class-dependent buffer allocations, in Fig 2.5d we show an example with 10 ports and 40 ports each with a single saturated queue of a low-priority class ($\alpha_p = 1$) showing how the burst-tolerance (assuming that the burst packets are sent to a queue of high-priority class $\alpha_p = 10$) is affected.

We argue that, in today's networks, especially considering micro-burst events which are often mapped to queues of high-priority class in buffer management schemes, an unbounded and class-inter-dependent buffer allocation scheme cannot provide guaranteed services.

DT is oblivious to service capacity The threshold or in other words buffer allocation by DT algorithm is only based on unused buffer space, which makes DT a scalar and single-dimensional algorithm. Although such an algorithm was proposed based on single FIFO queue per port assumptions, various vendors such as cisco, broadcom have adopted the same algorithm for multiple queue per port switch models. As an example, it can be observed from Eq. 2.3 that, DT allocates same buffer space to a queue with service capacity C and another queue with $C/8$. When multiple queues share the capacity of the output port, the service capacity of the queue becomes dependent on the scheduling policies. Under such circumstances, *DT* is oblivious to service capacity. To illustrate the problem in relation to today's networks, in Fig. 2.5e we show an example with 2 different states which are indeed equivalent for DT resulting in same buffer allocation in both states. It can be observed from Fig. 2.5e, how burst-tolerance is heavily impacted in the case with 1 port and 8 queues/port $1p8q$ (low-priority class) due to low rate of change of occupied buffer by queues of low-priority class ($\frac{dQ_L(t)}{dt}$). It can be observed from Eq. 2.4 how the rate of change of threshold is related to the rate of change of total occupied buffer space. Given such a behavior, we argue that DT being oblivious to service

capacity, cannot provide any guaranteed services for high-priority classes.

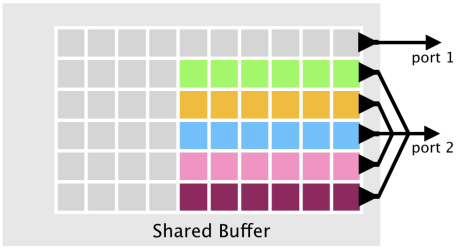
In the next Chapter we propose Plasticine suitable today's data center networks, intuitively solving the inefficiencies of DT.

Chapter 3

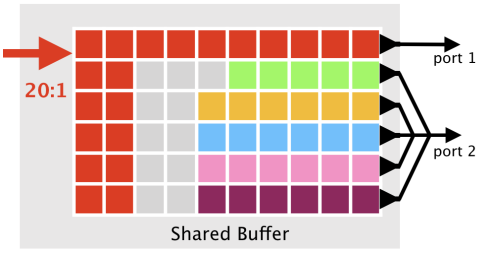
Plasticine Overview

To address the shortcomings of DT, we design Plasticine a novel priority-based buffer management scheme designed to offer isolation, effectively minimizing cross-priority side effects and providing strict burst-tolerance guarantees. Next, we describe the key insights behind Plasticine's design.

Plasticine is a priority-based scheme. Plasticine manages the buffer at the priority level in two ways. First, Plasticine decides the threshold of each queue considering its priority and the aggregated buffer usage of the priority. To that end, Plasticine requires the operator to define an α parameter per priority as opposed to an α parameter per-queue that DT requires. This is an intuitive configuration as traffic that belongs to the same priority will have similar requirements from the buffer. For example, some priority might be able to tolerate loss but not delay (low α), while another might display micro-bursting behavior (high α). Second, Plasticine distinguishes priorities to those that require strict isolation guarantees (isolation priorities) and those that share the buffer fairly, proportionately to their α values (regular priorities). Distinguishing priorities based on their isolation requirements allows Plasticine to find a trade-off between burst-tolerance and high buffer utilization. For example, the operator can allow both *(i)* highly bursty traffic, such as Map-Reduce jobs, to use the buffer during an incast as if they were alone



(a) Plasticine allocates less space per queue (6 packets) if those share a port's service rate.



(b) The burst is fully absorbed despite the low dequeue rate as there is remaining buffer.

Figure 3.1: Plasticine allocates buffer per queue proportionately to its service rate, thus keeping the buffer ready for a burst.

in it (without statically allocating it); and *(ii)* storage traffic to use just enough buffer to facilitate high throughput whenever possible.

Plasticine offers isolation per priority. PLASTICINE dynamically bounds buffer usage per priority to prevent any priority to monopolize the buffer. If the buffer utilization per priority is unbounded, as it is for DT, any single priority can exhaust buffer resources simply by employing multiple queues, as was illustrated in Fig. 2.5b. Unlike DT that would uncontrollably allocate space for each new queue of the yellow priority, PLASTICINE splits a dynamically-bounded priority-share among queues of that priority. Effectively, it allows the critical red-priority queue to allocate 30 packets regardless of the number of queues the yellow priority has in the buffer, as shown in Fig. 3.1b. Note, though, that no allocation is static. Namely, if the red-queue does not use/need its maximum buffer occupancy, the yellow priority will get more. Thus, PLASTICINE's dynamic allocation cannot be achieved by conventional techniques such as statically allocating space to each queue (complete partitioning) or to a group of queues (application pools) or by statically configuring a different ratio of α values. All of the aforementioned alternatives would lead to buffer under-utilization.

Plasticine makes bursts first-class citizens in the buffer. PLASTICINE allows the operator to provide strict burst-tolerance guarantees, to isolation priorities while not starving the rest of the traffic. To that end, PLASTICINE continually guards the buffer occupancy to ensure that *(i)* there is always enough unoccupied buffer space; and/or *(ii)* adequate aggregate dequeue rate to accommodate a burst. Note that the two conditions need not both hold. On the one hand, an incoming burst can be absorbed regardless of the free buffer space at its arrival iff the aggregate dequeue rate of the allocated buffer space is at least as high as the enqueue rate. On the other hand, if one or a few ports consume the buffer, thus the aggregated dequeue rate is low, the unoccupied buffer needs to be sufficient to accommodate an incoming burst. Thus, PLASTICINE would limit each queue to 6 packets, as shown in Fig. 3.1b as a penalty to their low dequeue rate. As a result, when the same 20:1 burst arrives, instead of being heavily dropped as in DT, it will be entirely absorbed by the buffer, as shown in Fig. 3.1b.

Chapter 4

Plasticine Analysis

A general analysis of PLASTICINE algorithm is described in this chapter. In particular, steady-state, transient-state, the key properties, guidelines to determine parameters and the guarantees are described analytically.

4.1 Assumptions

The analysis is based on a fluid model where packet (bits) arrivals and departures are assumed to be fluid and deterministic. A switch with arbitrary number of ports with arbitrary number of queues per port is assumed. In particular, each port has only one queue per priority.

B : Total shared buffer space of switch.
$Q(t)$: Instantaneous occupied buffer space at time t
α_p : Quantification of priority level, a parameter for buffer-management algorithm.

4.2 Plasticine

PLASTICINE works based on two-levels of hierarchy i.e priority and groups. The general notion of priority remains same. In addition, PLASTICINE requires the priorities to be divided in-to groups, *shared* and *isolated*. A *shared* group contains priorities which share the buffer fairly proportionate to their alpha values. An *isolated* group is achieved simply by creating a group consisting of single priority.

PLASTICINE buffer-management algorithm requires an α_p parameter per priority. The buffer-allocation is based on threshold calculations per queue. In particular, the threshold of a queue at port i, of priority p and belonging to a group g is calculated by PLASTICINE algorithm as,

$$Threshold = (alpha) \times (FairShare) \times (Norm.DequeueRate) \times (remaining) \quad (4.1)$$

i.e,

$$T_p^i(t) = \alpha_p \cdot \beta_{g(p)}(t) \cdot \gamma_p^i(t) \cdot (B - Q(t)) \tag{4.2}$$

where, $\beta_{g(p)}(t)$ is the inverse of the total number of congested queues in group $g(p)$ (to which the priority p belongs to) at time t and $\gamma_p^i(t)$ is the normalized dequeue rate (or normalized service capacity) of the queue at time t.

Observe that, $\beta_{g(p)}(t)$ remains same for all the priorities belonging to a group and can be expressed as $\beta_G(t)$ where G denotes the group $g(p)$.

Here after for simplicity $\omega_p^i(t)$ is defined as,

$$\omega_p^i(t) = \alpha_p \cdot \beta_G(t) \cdot \gamma_p^i(t) \tag{4.3}$$

Key properties of Omega:

$$\sum_i \sum_{p \in G} \omega_p^i(t) = \beta_G(t) \cdot \sum_i \sum_{p \in G} \alpha_p \gamma_p^i(t) \tag{4.4}$$

If, all the queues of all priorities of group G share same alpha parameters (say α_G) and have same normalized dequeue rate at time t (say γ_G), then Eq. 4.4 reduces to,

$$\sum_i \sum_{p \in G} \omega_p^i(t) = \alpha_G \cdot \gamma_G \tag{4.5}$$

Further, if $\gamma_G = 1$,

$$\sum_i \sum_{p \in G} \omega_p^i(t) = \alpha_G \tag{4.6}$$

In general there exists a limit given by,

$$\sum_i \sum_{p \in G} \omega_p^i(t) \leq max(\alpha_p) = \alpha_{max}^G \tag{4.7}$$

We will see later, how these properties enable Plasticine to achieve certain isolation and burst-tolerance guarantees.

4.3 Steady-State Analysis

In this subsection, it is assumed that load-conditions remain stable and a steady-state of buffer is achieved. Following this assumption, all the equations in this subsection are expressed without the time variable. Under this state, the queue lengths remain stable at less than or equal to the corresponding threshold. For simplicity, it is assumed that all the queues-lengths are at their respective thresholds. Then the total buffer occupancy can be expressed as,

$$Q = \sum_i \sum_p Q_p^i \tag{4.8}$$

From the assumption that the queue-lengths are equal to their thresholds, using Eq. 4.2 and Eq. 4.3,

$$Q = \sum_i \sum_p \omega_p^i \cdot (B - Q) \tag{4.9}$$

Solving for $Q(t)$ gives,

$$Q = \frac{B \sum_i \sum_p \omega_p^i}{1 + \sum_i \sum_p \omega_p^i} \tag{4.10}$$

where ω_p^i is given by Eq. 4.3
Using, Eq. 4.10, the remaining buffer space $B - Q(t)$ can be expressed as,

$$Remaining = \frac{B}{1 + \sum_i \sum_p \omega_p^i} \tag{4.11}$$

Under steady-state, from Eq. 4.11 and Eq. 4.2, the threshold of a queue at port i and of priority p is given by,

$$T_p^i = \frac{B \cdot \omega_p^i}{1 + \sum_i \sum_p \omega_p^i} \tag{4.12}$$

Key properties of Remaining Buffer space:
Using the notion of groups, Eq. 4.11, can be expanded as,

$$Remaining = \frac{B}{1 + \sum_g \sum_i \sum_{p \in g} \omega_p^i} \tag{4.13}$$

Using the maximum limit of ω property from Eq. 4.7,

$$Remaining \geq \frac{B}{1 + \sum_g \alpha_{max}^g} \tag{4.14}$$

This key property shows how the remaining buffer space and buffer-occupancy are bounded.

For example, lets consider, there exists two priorities p_0 and p_1 with alpha parameters α_0 and α_1, each of which belongs to a separate group. The remaining buffer space, irrespective of the number of queues and their dequeue rates, Eq. 4.14 can be expressed as,

$$Remaining \geq \frac{B}{1 + \alpha_0 + \alpha_1} \tag{4.15}$$

4.4 Transient-State Analysis

Given a steady-state of buffer assuming that all the queue lengths are controlled by a threshold, when traffic to empty queues appear, load conditions change. The new queues

increase in length creating changes in the remaining buffer. As a result, the thresholds and queue lengths under go a transient state. Due to the appearance of new queues, ω_p^i of some of the existing queues get affected due to the changes in β_p (number of queues belonging to a group) and γ_p^i (normalized dequeue rate). Let C_e denote the set of queues whose ω_p^i gets affected and C_{ne} denote the set of queues whose ω_p^i does not get affected. Note that the ω_p^i values of C_e only reduce. (It is not possible that ω_p^i increases due the appearance of a new queue). For simplicity lets denote the queue at port i and of priority p with ordered pairs as (i, p). The set of ordered pairs of existing queues is denoted as S_{old}. The ordered pairs of new queues that trigger transient state are denoted as S_{new}. Observe that $S_{old} = C_{ne} \cup C_e$.

The arrival rate of traffic at each new queue is denoted by r and the arrival process is fluid and deterministic. At $t = 0$,

$$T_p^i(0) = \frac{\omega_p^i \cdot B}{1 + \sum_{\forall(i,p)\in S_{old}} \omega_p^i} \tag{4.16}$$

$$Q_p^i(0) = \begin{cases} \frac{\omega_p^i \cdot B}{1 + \sum_{\forall(i,p)\in S_{old}} \omega_p^i} & \text{, for } \forall(i,p) \in S_{old} \\ 0 & \text{, for } \forall(i,p) \in S_{new} \end{cases} \tag{4.17}$$

At $t = 0^+$, ω_p^i of C_e change and remain same for the entire duration of transient state. At the same time, the ω_p^i of C_{ne} remain unchanged. Hence such changes are assumed to happen and the time variable is dropped for ω_p^i in the equations.

From Eq. 4.2, the rate of change of thresholds and queue lengths can be expressed as follows,

$$\frac{dT_p^i(t)}{dt} = -\omega_p^i \cdot \sum_{\forall(i,p)\in S_{old}\cup S_{new}} \frac{dQ_p^i(t)}{dt} \tag{4.18}$$

$$\frac{dQ_p^i(t)}{dt} = \begin{cases} -\gamma_p^i & \text{, if } Q_p^i(t) > T_p(t) \text{ and } \forall(i,p) \in S_{old} \\ max[-\gamma_p^i, min[\frac{dT_p(t)}{dt}, r - \gamma_p^i]] & \text{, if } Q_p^i(t) = T_p(t) \text{ and } \forall(i,p) \in S_{old} \\ r - \gamma_p^i & \text{, if } Q_p^i(t) < T_p(t) \text{ and } \forall(i,p) \in S_{new} \end{cases} \tag{4.19}$$

It can be proved by contradiction that $\frac{dT_p^i(t)}{dt} \leq 0 < r - \gamma_p^i$. Solving Eq. 4.18 and Eq. 4.19 for $t = 0+$,

$$\left(\frac{dT_p^i(t)}{dt}\right)_{(t=0+)} = -\omega_p^i \cdot \left(\sum_{\forall(i,p)\in S_{old}} max[-\gamma_p^i, \frac{dT_p(t)}{dt}_{(t=0+)}]\right) - \omega_p^i \cdot \sum_{\forall(i,p)\in S_{new}} (r - \gamma_p^i) \tag{4.20}$$

Recall that $S_{old} = C_e \cup C_{ne}$. All the queues belonging to C_e, will experience a change in their ω_p^i values at $t = 0^+$ resulting in their queue-lengths greater than threshold. As

a result, the rate of change of their queue lengths is their corresponding dequeue rates. Eq. 4.20 can then be expanded as,

$$\left(\frac{dT_p^i(t)}{dt}\right)_{(t=0+)} = -\omega_p^i \cdot \left(\sum_{\forall(i,p)\in C_e} -\gamma_p^i\right) - \omega_p^i \cdot \left(\sum_{\forall(i,p)\in C_{ne}} max[-\gamma_p^i, \frac{dT_p(t)}{dt}_{(t=0+)}]\right) - \omega_p^i \cdot \sum_{\forall(i,p)\in S_{new}} (r - \gamma_p^i) \tag{4.21}$$

From Eq. 4.21, arrival rate of traffic in new queues i.e r can be expressed as,

$$r = \frac{\sum_{\forall(i,p)\in S_{new}\cup C_e} \gamma_p^i}{\sum_{\forall(i,p)\in S_{new}} 1} - \frac{\frac{dT_p^i(t)}{dt}_{(t=0+)} + \omega_p^i \cdot \left(\sum_{\forall(i,p)\in C_{ne}} max[-\gamma_p^i, \frac{dT_p(t)}{dt}_{(t=0+)}]\right)}{\omega_p^i \cdot \sum_{\forall(i,p)\in S_{new}} 1} \tag{4.22}$$

By applying summation across $\forall(i,p) \in C_e$ over Eq. 4.21 (will be seen later how this will be useful), r can be expressed as,

$$r = \frac{\sum_{\forall(i,p)\in S_{new}\cup C_e} \gamma_p^i}{\sum_{\forall(i,p)\in S_{new}} 1} - \frac{(\sum_{i,p\in C_{ne}} \frac{dT_p^i(t)}{dt}) + \left(\sum_{\forall(i,p)\in C_{ne}} max[-\gamma_p^i, \frac{dT_p(t)}{dt}]\right) \cdot \sum_{\forall(i,p)\in C_{ne}} \omega_p^i}{(\sum_{\forall(i,p)\in C_{ne}} \omega_p^i) \cdot (\sum_{\forall(i,p)\in S_{new}} 1)} \tag{4.23}$$

Now it can be observed that the value of r influences for all $\forall(i,p) \in C_e$, $\left(\frac{dT_p^i(t)}{dt}\right)_{(t=0+)}$. In other words, the value of r influences the total i.e $\sum_{\forall(i,p)\in C_e} \left(\frac{dT_p^i(t)}{dt}\right)_{(t=0+)}$ which is the aggregate rate at which thresholds drop for the non affected set of queues i.e C_{ne}.

4.4.1 Case-1

In this case, the arrival rate r is such that, the queues belonging to C_{ne} are able to reduce in length exactly tracking the changes in their thresholds. As a result their queue-lengths remain equal to the threshold throughout the transient state i.e,

$$\left(\frac{dT_p^i(t)}{dt}\right)_{(t=0^+)} \geq -\gamma_p^i \tag{4.24}$$

leading to,

$$\sum_{\forall(i,p)\in C_{ne}} \left(\frac{dT_p^i(t)}{dt}\right)_{(t=0+)} \geq \sum_{\forall(i,p)\in C_{ne}} -\gamma_p^i \tag{4.25}$$

Using Eq. 4.24 and Eq. 4.25 in Eq. 4.23, the condition on r can be expressed as,

$$r \leq \frac{\sum\limits_{\forall(i,p)\in S_{new}\cup C_e} \gamma_p^i}{\sum\limits_{\forall(i,p)\in S_{new}} 1} + \left(\sum_{\forall(i,p)\in C_{ne}}^{*} \gamma_p^i\right) \cdot \frac{1 + \sum\limits_{\forall(i,p)\in C_{ne}} \omega_p^i}{(\sum\limits_{\forall(i,p)\in C_{ne}}^{*} \omega_p^i) \cdot (\sum\limits_{\forall(i,p)\in S_{new}} 1)} \tag{4.26}$$

Note that in Eq. 4.23, we deliberately apply summation over $\forall(i,p) \in C_{ne}$ which can be a null set. If $C_{ne} = \phi$, by applying summation over $\forall(i,p) \in C_e$ in Eq. 4.21, r condition can be expressed as,

$$r \leq \frac{\sum\limits_{\forall(i,p)\in S_{new}} \gamma_p^i}{\sum\limits_{\forall(i,p)\in S_{new}} 1} + \left(\sum_{\forall(i,p)\in C_e} \gamma_p^i\right) \cdot \frac{1 + \sum\limits_{\forall(i,p)\in C_e} \omega_p^i}{(\sum\limits_{\forall(i,p)\in C_e} \omega_p^i) \cdot (\sum\limits_{\forall(i,p)\in S_{new}} 1)} \tag{4.27}$$

For generalization, observe the "*" over the summation terms in Eq. 4.26. Here after, the convention follows that, where ever "*" appears, it means that, the summation is deliberate and can be interchanged between $\forall(i,p) \in C_{ne}$ and $\forall(i,p) \in C_e$ if $C_{ne} = \phi$. All the other summations have usual meaning.

Substituting Eq.4.24 and Eq.4.25 in Eq. 4.21 and using the result in Eq. 4.19 gives,

$$\left(\frac{dT_p(t)}{dt}\right)_{(t=0^+)} = \frac{-\omega_p^i \cdot \left(\sum\limits_{\forall(i,p)\in C_e} -\gamma_p^i + \sum\limits_{\forall(i,p)\in S_{new}} (r - \gamma_p^i)\right)}{1 + \sum\limits_{\forall(i,p)\in C_{ne}} \omega_p^i} \tag{4.28}$$

$$\left(\frac{dQ_p^i(t)}{dt}\right)_{(t=0^+)} = \begin{cases} \dfrac{-\omega_p^i \cdot \left(\sum\limits_{\forall(i,p)\in C_e} -\gamma_p^i + \sum\limits_{\forall(i,p)\in S_{new}} (r - \gamma_p^i)\right)}{1+\sum\limits_{\forall(i,p)\in C_{ne}} \omega_p^i} & \text{, for } \forall(i,p) \in C_{ne} \\ -\gamma_p^i & \text{, for } \forall(i,p) \in C_e \\ r - \gamma_p^i & \text{, for } \forall(i,p) \in S_{new} \end{cases} \tag{4.29}$$

These differential equations will be valid as long as $Q_p^i(t) = T_p^i(t)$ for $\forall(i,p) \in C_{ne}$ && $Q_p^i(t) \geq T_p^i(t)$ for $\forall(i,p) \in C_e$ && $Q_p^i(t) < T_p^i(t)$ for newly created queues i.e $\forall(i,p) \in S_{new}$. Solving these equation, using the initial conditions, Eq. 4.16 and Eq. 4.17 leads to,

$$T_p^i(t) = \frac{\omega_p^i \cdot B}{1 + \sum\limits_{\forall(i,p)\in S_{old}} \omega_p^i} - \frac{\omega_p^i \cdot \left(\sum\limits_{\forall(i,p)\in C_e} -\gamma_p^i + \sum\limits_{\forall(i,p)\in S_{new}} (r - \gamma_p^i) \right) \cdot t}{1 + \sum\limits_{\forall(i,p)\in C_{ne}} \omega_p^i} \tag{4.30}$$

$$Q_p^i(t) = \begin{cases} \dfrac{\omega_p^i \cdot B}{1+\sum\limits_{\forall(i,p)\in S_{old}} \omega_p^i} - \dfrac{\omega_p^i \cdot t \cdot (\sum\limits_{\forall(i,p)\in C_e} -\gamma_p^i + \sum\limits_{\forall(i,p)\in S_{new}} (r - \gamma_p^i))}{1+\sum\limits_{\forall(i,p)\in C_{ne}} \omega_p^i} & \text{, for } \forall(i,p) \in C_{ne} \\ \dfrac{\omega_p^i \cdot B}{1+\sum\limits_{\forall(i,p)\in S_{old}} \omega_p^i} - \gamma_p^i \cdot t & \text{,for } \forall(i,p) \in C_e \\ (r - \gamma_p^i) \cdot t & \text{, for } \forall(i,p) \in S_{new} \end{cases} \tag{4.31}$$

As we can observe from Eq. 4.30 and Eq. 4.31, the new queues will grow in length without dropping packets upto a time say $t1_p^i$ when the threshold equals the queue length. It is considered that $t1_p^i$ denotes the time at which a new queue of priority p at port i first touches the threshold. The transient state continues after $t1_p^i$ until all the queues achieve a steady state occupancy. By equating Eq. 4.30 and Eq. 4.31 for the case of $\forall(i,p) \in S_{new}$, $t1_p^i$ can be obtained as,

$$t1_p^i = \frac{\omega_p^i \cdot B \cdot (1 + \sum\limits_{\forall(i,p)\in C_{ne}} \omega_p^i)}{(1 + \sum\limits_{\forall(i,p)\in S_{old}} \omega_p^i) \cdot ((r - \gamma_p^i) \cdot (1 + \sum\limits_{\forall(i,p)\in C_{ne}} \omega_p^i) + \omega_p^i \cdot (\sum\limits_{\forall(i,p)\in C_e} -\gamma_p^i + \sum\limits_{\forall(i,p)\in S_{new}} (r - \gamma_p^i)))} \tag{4.32}$$

4.4.2 Case-2

In this case, the arrival rate r is such that, the queues belonging to C_{ne} are unable to reduce in length in accordance with the changes in their thresholds. As a result their queue-lengths remain greater that the threshold throughout the transient state i.e,

$$\left(\frac{dT_p^i(t)}{dt} \right)_{(t=0^+)} < -\gamma_p^i \tag{4.33}$$

leading to,

$$\sum_{\forall(i,p)\in C_{ne}} \left(\frac{dT_p^i(t)}{dt} \right)_{(t=0+)} < \sum_{\forall(i,p)\in C_{ne}} -\gamma_p^i \tag{4.34}$$

Using Eq. 4.33 and Eq. 4.34 in Eq. 4.23, the condition on r can be expressed as,

$$r > \frac{\sum\limits_{\forall(i,p)\in S_{new}\cup C_e} \gamma_p^i}{\sum\limits_{\forall(i,p)\in S_{new}} 1} + \left(\sum_{\forall(i,p)\in C_{ne}}^{*} \gamma_p^i\right) \cdot \frac{1 + \sum\limits_{\forall(i,p)\in C_{ne}} \omega_p^i}{(\sum\limits_{\forall(i,p)\in C_{ne}}^{*} \omega_p^i) \cdot (\sum\limits_{\forall(i,p)\in S_{new}} 1)} \tag{4.35}$$

if $C_{ne} = \phi$, then r the above condition is expressed as,

$$r > \frac{\sum\limits_{\forall(i,p)\in S_{new}} \gamma_p^i}{\sum\limits_{\forall(i,p)\in S_{new}} 1} + \left(\sum_{\forall(i,p)\in C_e} \gamma_p^i\right) \cdot \frac{1 + \sum\limits_{\forall(i,p)\in C_e} \omega_p^i}{(\sum\limits_{\forall(i,p)\in C_e} \omega_p^i) \cdot (\sum\limits_{\forall(i,p)\in S_{new}} 1)} \tag{4.36}$$

Following similar procedure as in *Case-1*, the equations for *Case-2* can be easily determined. Finally, the time $t1_p^i$ at which one of the queues that belong to S_{new} touches it's threshold can be expressed as,

$$t1_p^i = \frac{\omega_p^i \cdot B}{\left(1 + \sum\limits_{\forall(i,p)\in S_{old}} \omega_p^i\right) \cdot \left((r - \gamma_p^i) + \omega_p^i \cdot \left(\sum\limits_{\forall(i,p)\in S_{old}} -\gamma_p^i + \sum\limits_{\forall(i,p)\in S_{new}} (r - \gamma_p^i)\right)\right)} \tag{4.37}$$

4.5 How it all relates to Burst-Tolerance

Denote Burst-Tolerance for a queue of priority p at port i as $Burst_p^i$ can be defined as

$$Burst_p^i = r \cdot t1_p^i \tag{4.38}$$

where r is the arrival rate of traffic.

Then, the maximum burst that can pass without experiencing drops is given by $Burst_p^i$. Say an operator specifies $Burst_p^i$ i.e r and $t1_p^i$ to be guaranteed to pass at all times. How can α_p be optimized?

Given an arrival rate r, at a given state of buffer, whether r satisfies *Case-1* or *Case-2*, observe that $t1_p^i$ of *Case-1* is always greater than *Case-2*.

Let α_L denote the maximum alpha value for the queues of S_{old}. Additionally, for simplicity and to aim at worst-case, it is assumed that, α_L is the maximum alpha value

for both C_{ne} and C_e. Then from the properties of Omega, for a burst than happens on a queue at port i and of priority p, for *Case-1* and *Case-2*, from Eq. 4.32 and Eq. 4.35, the conditions in terms of ω_p^i can be expressed as,

$$\alpha_L \begin{cases} \leq \dfrac{1}{\left(\dfrac{\sum\limits_{\forall(i,p)\in S_{new}} 1}{\overset{*}{\sum\limits_{\forall(i,p)\in C_{ne}}} \gamma_p^i}\right) \cdot \left(r - \left(\dfrac{\sum\limits_{\forall(i,p)\in S_{new}\cup C_e} \gamma_p^i}{\sum\limits_{\forall(i,p)\in S_{new}} 1}\right)\right) - 1 \cdot !(C_{ne==\phi})} & \textit{Case-1} \\ > \dfrac{1}{\left(\dfrac{\sum\limits_{\forall(i,p)\in S_{new}} 1}{\overset{*}{\sum\limits_{\forall(i,p)\in C_{ne}}} \gamma_p^i}\right) \cdot \left(r - \left(\dfrac{\sum\limits_{\forall(i,p)\in S_{new}\cup C_e} \gamma_p^i}{\sum\limits_{\forall(i,p)\in S_{new}} 1}\right)\right) - 1 \cdot !(C_{ne==\phi})} & \textit{Case-2} \end{cases} \tag{4.39}$$

Let α_H denote the alpha parameter for the new queues. For a burst with arrival rate r upto time t to pass, it is required that $t <= t1_p^i$. Then, whether α_L is determined based on the above guideline or chosen based on steady-state allocations, α_H can be expressed as,

$$\alpha_H \geq \frac{1}{\gamma_p^i \cdot \beta_{g(p)}^i} \cdot \frac{t \cdot (r - \gamma_p^i) \cdot (1 + \alpha_L)}{B - t \cdot \left(\sum\limits_{\forall(i,p)\in C_e} -\gamma_p^i + \sum\limits_{\forall(i,p)\in S_{new}} (r - \gamma_p^i)\right)} \tag{4.40}$$

Note that, in Eq. 4.40, when the denominator is less than or equal to 0, α_H has no meaning.

Chapter 5

Evaluation

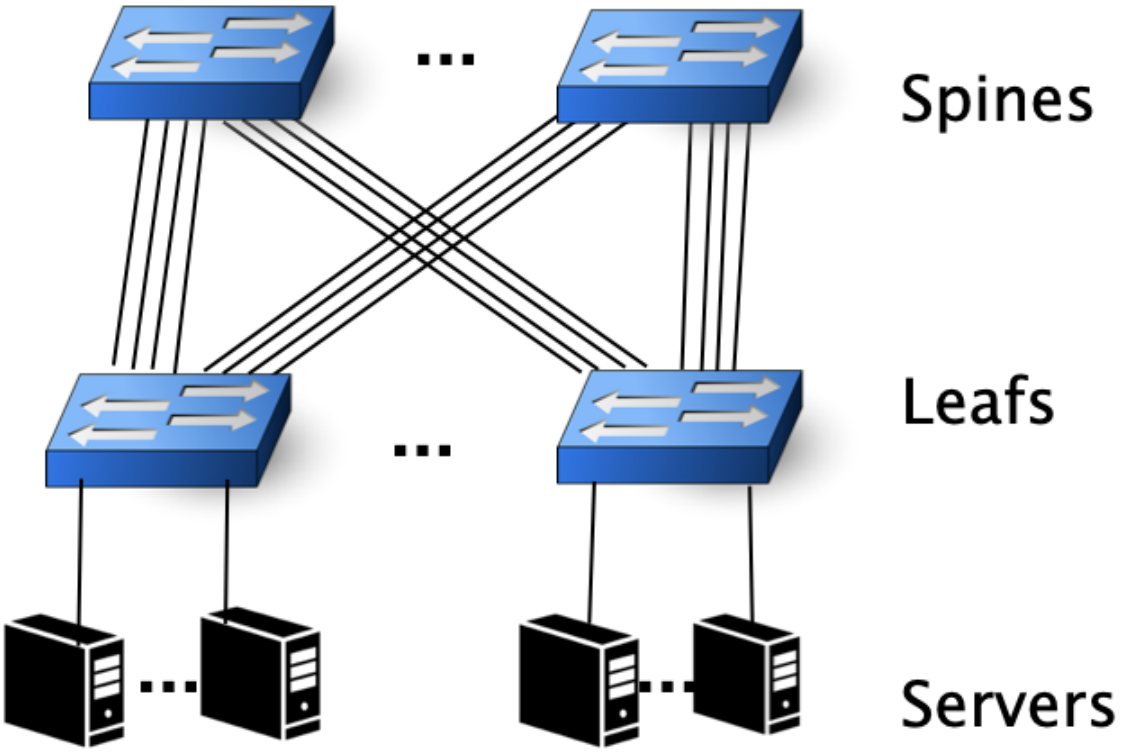


Figure 5.1: Topology

In our evaluation we want to answer four main questions. First, how does Plasticine compares against different buffer management schemes? Second, can the benefits of Plasticine be achieved if DCTCP is in place? Third, is Plasticine's performance impacted by traffic of mixed CC?

We found that,

- PLASTICINE achieves lower flow completion times and on-par throughput with conventional buffer management schemes that utilize more buffer.
- Advanced TCP versions such as DCTCP that require end-hosts collaboration, only alleviate buffer utilization problems but are heavily impacted by tcp-incast problems due to inefficiencies of underlying buffer-management schemes.
- PLASTICINE is not effected by CC and allows for co-existence without any impact on burst tolerance.

5.1 Methodology

Our evaluation is based on ns3 [3] simulations. We implemented DCTCP based on the online available ns2 code which was used in the evaluation of DCTCP. Our implementation of different buffer-management in ns3 is made available online.

We report 5 performance metrics throughout our evaluation *(i)* Burst Absorption: The number of queries that experience zero loss *(ii)* Average query completion times *(iii)* 99%-ile flow completion time *(iv)* Average and 99%-ile buffer utilization *(v)* Average aggregate throughput of uplinks in our topology. All the reported metrics are an average over 10 simulations.

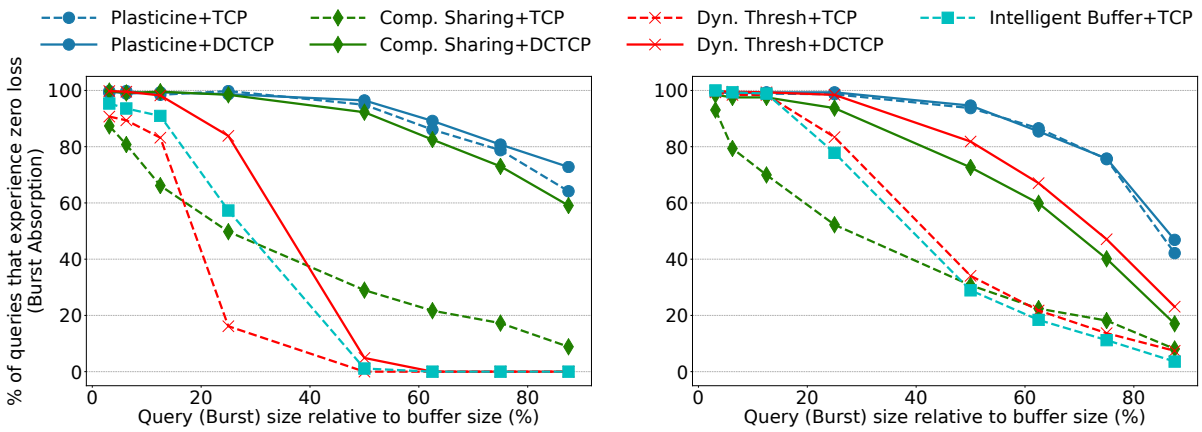
5.2 Topology

We evaluate PLASTICINE's performance in a Leaf Spine topology [4] shown in Fig. 5.1. Our topology consists of 2 leaves, 2 spines with 4 links between each leaf↔spine and 40 servers connected to each leaf *(oversubscription of 5)*. ECMP is used to load-balancing traffic across uplinks.

Traffic mix We use two traffic distributions namely web search and query traffic. In the case of web search we used flow size distribution from [5] and tuned the mean of poisson inter-arrivals such that a certain load is achieved. We set the load to 90% over the uplinks in our topology (Fig. 5.1). We refer to this type of traffic as Background traffic. In the case of queries we assume that a query arrives at each server according to a poisson process with mean 1 query/second. Each query consists of a server attached to a leaf requesting *Query-Size* file from all the servers connected to the other leaf. Each request is then responded by 40 servers each transmitting $\frac{1}{40}$ of the file. All the query responses start with open tcp window leading to 1 : 40 worst-case incast scenario. A query is completed when the requester receives *Query-Size* file. We refer to this type of traffic as foreground traffic.

Priorities and α_p We consider two different cases in our evaluation *(i)* Switches have single queue per port and packets do not carry priority tags. *(ii)* Packets are marked with priority tags at the sender and such a tag is used for in-network priorities and queuing.

We use two α_p values, $\alpha_L = 0.5$ and $\alpha_H = 20$ for low priority and high priority respectively. In the case with single queue per port, α_L is used for all the queues in the



(a) Single priority queue per port

(b) Multiple priority queues per port

Figure 5.2: % of queries that experience zero loss (representing burst absorption). *(i)* PLASTICINE bridges the gap between DcTcp and Tcp *(ii)* PLASTICINE achieves $\approx 5\%$ and $\approx 60\%$ gain with DcTcp and Tcp respectively *(iii)* The performance degrade with DcTcp from single queue to multiple queue per port scenarios show how DcTcp fails to adapt it's K parameter, allowing for more buffer utilization leading to low burst absorption.

switch. In the case with in-network priorities, the highest priority queue at each port is configured with α_H and all the other queues are configured with α_L.

Additionally in the case of PLASTICINE, foreground traffic is mapped to the highest priority with $\alpha_H = 20$ and to a group with index 0. Background traffic flows are mapped to $\alpha_L = 0.5$ and to a group with index 1. All the queues are DropTail expect in the evaluations with *DCTCP*. In the evaluations with *DCTCP*, all queues are RED with min and max threshold set to 20 ($K = 20$) for 1Gbps links following the recommendations in [5]. TCP *minRTO* is set to $100ms$.

5.3 Results

5.3.1 Burst Absorption

We first evaluate the burst absorption capabilities by measuring the percentage of queries in our traffic-mix that experience zero loss. A query response from multiple servers creates an in-cast scenario leading to a burst at the switch. In Fig. 5.2, we vary the burst sizes and measure the burst absorption capabilities of different buffer management schemes using TCP and DCTCP. We observe that PLASTICINE achieves the best burst absorption in both single and multiple queues per port scenarios. From Fig. 5.2a, we observe that PLASTICINE achieves over 70% higher burst absorption compared to *DT* with TCP and DCTCP. PLASTICINE limits the buffer occupancy of long flows to a maximum of $\alpha \cdot \frac{B}{1+\alpha} \approx$ 33% of total buffer where $\alpha = 0.5$. Additionally, PLASTICINE prioritizes bursts enabling high burst absorption with predictable performance. We notice that a significant drop in

performance of DT is mainly due to it's inability to prioritize bursts and allocate more buffer.

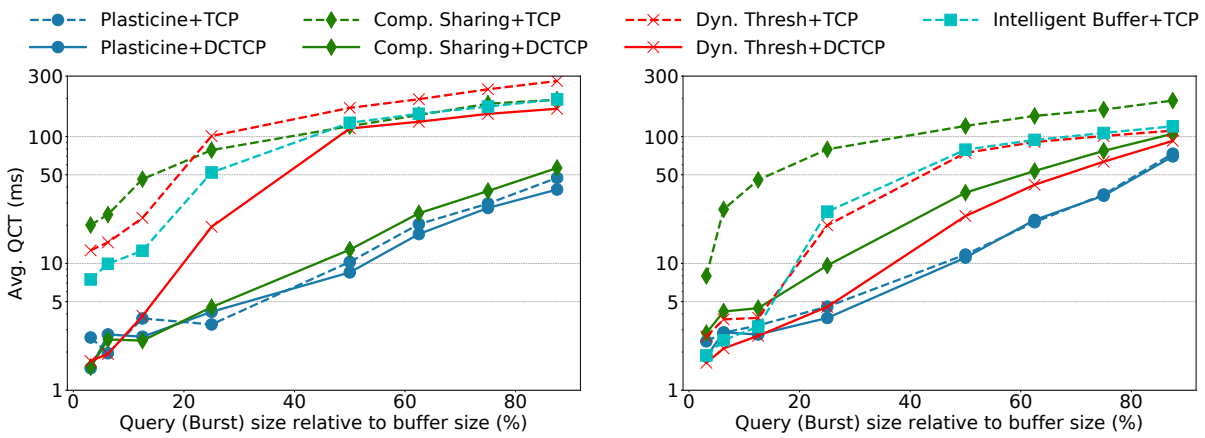
We then evaluate in a scenario where end-hosts add priority tags to packets. In particular, query traffic (foreground) flows carry highest priority tag and background traffic flows carry 4 low priority tags chosen uniformly at random by the end-hosts. All the switches are equipped with multiple queues per port, where traffic is split in to queues based on priority tags. DT is configured with a high α_H value for highest priority queue so as to allocate as much buffer as available for bursts. In this scenario, we are interested in the performance gain obtained from isolating the bursts to a separate queue. We observe from Fig. 5.2b that Plasticine still achieves the best burst absorption, 30% higher than DT and 35% higher than CS with DCTCP. Enabling packet priority tags, and priority queuing increased the burst absorption capabilities of DT. Given that bursts are prioritized in all the algorithms, our result in Fig. 5.2b focuses on the abilities of different buffer management schemes in *(i)* limiting long flows and *(ii)* providing more buffer to bursts. We observe that even after prioritizing bursts, $DT's$ burst absorption is still 60% and 30% lower than Plasticine with TCP and DCTCP respectively.

Plasticine achieves better QCTs: Our previous results on burst absoption showed that Plasticine achieves the best burst absorption compared to the state-of-the-art solutions. We are now interested in the real impact of burst absorption on average Query Completion Times (QCT). The QCT is the time difference between the receipt of a query request and the arrival of complete $QuerySize$ data at the receiver. We observe from Fig. 5.3a that Plasticine achieved $\approx 115ms$ lower QCTs than DT with DCTCP and $\approx 18ms$ lower QCTs than CS with DCTCP. Further even when bursts are prioritized in a multi queue per port architecture, we observe from Fig. 5.3b that Plasticine achieved $\approx 21ms$ lower QCTs. Indeed it is clear that, better burst absorption leads to better query completion times.

The case of Complete Sharing: Certain control over queue lengths without the need for buffer management can be achieved with DCTCP as can be seen from Fig. 5.5 where the average buffer occupancy with *Complete Sharing* is around 20% with DCTCP and 80% with TCP. As complete sharing is analogous to a case with no buffer management, this result shows the capability of DCTCP to control queue lengths. In this case, simply leaving the buffer with no buffer management could enable high burst absorption. However, the burst tolerance of *Complete Sharing* with TCP is poor and especially leads to throughput loss as can be seen in Fig. 5.5. Nevertheless, Plasticine achieves 10% higher burst absorption than *Complete Sharing* by *(i)* dynamically limiting the buffer occupancy of long flows, *(ii)* prioritizing bursts and reducing the allocation for long flows if needed for a short time to provide further more buffer to bursts.

5.3.2 Flow completion times (FCT) for short flows

We consider flows which are less than a 100KB in size as short flows. FCT for such flows is mainly effected by *(i)* queuing delay, *(ii)* packet loss, eventually timeouts and *(iii)* number of round trips spent before completion. In the presence of bursts and long flows,



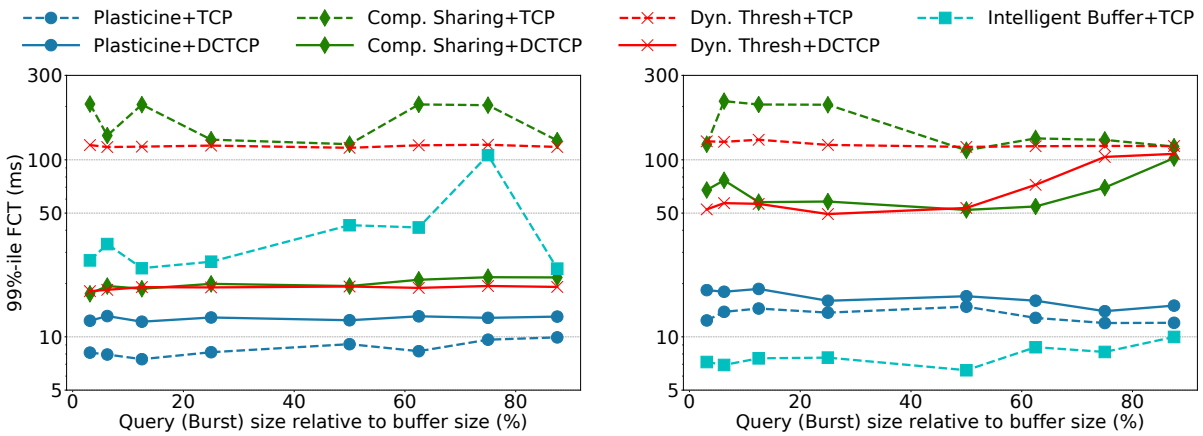
(a) Single priority queue per port

(b) Multiple priority queues per port

Figure 5.3: Average Query completion times, showing PLASTICINE outperforms in both single and multi queue architectures.

we first evaluated the performance of short flows in a single queue per port architecture. We observe from Fig. 5.4a that 99 percentile flow completion times are significantly lower with PLASTICINE in comparison to other buffer management schemes. In particular, PLASTICINE achieved ≈ 10 times lower FCTs than the state-of-the-art Dynamic Threshold (DT) when TCP is used as congestion control. The poor performance with DT is due to the presence of bursts that severely effect short flows. With DT, bursts and long flows consume majority of a queue's allocated buffer leaving little to no buffer for short flows. As a result, short flows experience timeouts even if the overall buffer has remaining space (see Fig. 5.5). As expected, DCTCP improved FCTs for DT, however still ≈ 2 times higher than PLASTICINE. On the other hand, the FCTs of PLASTICINE with TCP and DCTCP show a counter-intuitive result. First, PLASTICINE achieves lower FCTs with both TCP and DCTCP due to actively limiting long flows buffer occupancy and prioritizing short flows. Second, DCTCP under high loads has queue lengths frequently exceeding the marking threshold. As a result, short flows are prone to excessive marking feedback resulting in a relatively slower completion of flows. Nevertheless the difference is not significant.

Next, we observe the effect on short flow completion times when the queuing interference from bursts is removed. Here, bursts are marked prior and hence are separated to a high priority queue. Our evaluation in a single queue per port showed that short flows experience timeouts due to the presence of bursts. However we show in Fig. 5.4b that multi queue architecture and separation of bursts to a dedicated queue only worsened the flow completion times, expect in the case of PLASTICINE. Multiple queues share both buffer and dequeue rate at the same port. PLASTICINE pro-actively controls the threshold to minimize queuing delays caused by long flows, thanks to the dequeue rate factor in it's allocation. As discussed in Sec. 2, none of the current buffer management and congestion control mechanisms are capable of such adaptive buffer allocation based



(a) Single priority queue per port

(b) Multiple priority queues per port

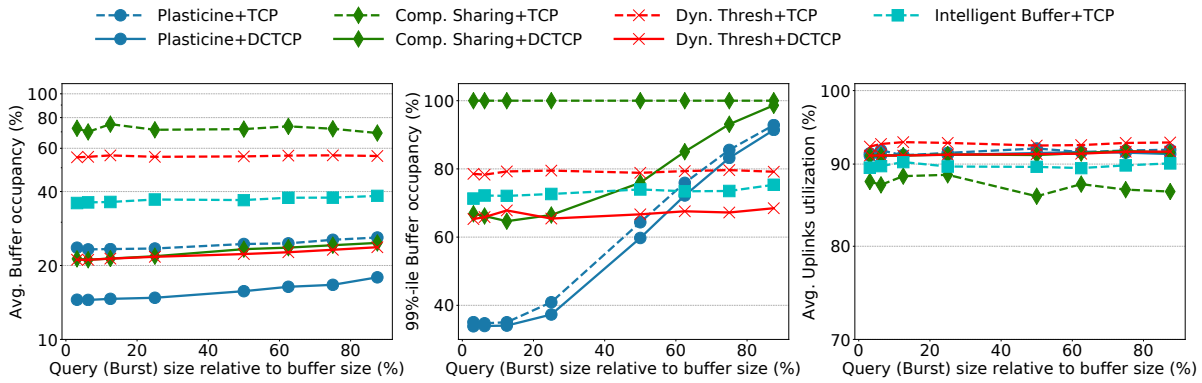
Figure 5.4: 99%-ile FCT for short flows ($< 100KB$) from background traffic. *(i)* In single queue per port, bursts do not significantly impact short flows 99%-ile FCTs as can be seen from relatively no variation *(ii)* In multi queue per port, DT and CS experience a shoot up in FCTs as the bursts start to interfere where the interference is due to drastic reduction in threshold of other queues caused by high priority queue occupancy by bursts. *(iii)* PLASTICINE offers best FCT performance for short flows.

on dequeue rates. Finally, we see from Fig. 5.4b and Fig. 5.4a that PLASTICINE achieved significantly lower flow completion times for short flows, $\approx 10\times$ with TCP and $\approx 2\times$ with DCTCP, irrespective of single or multiple queues per port sharing the buffer.

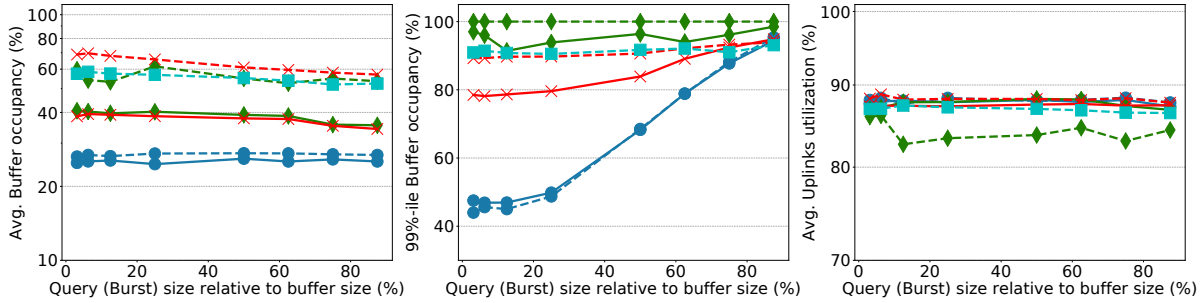
5.3.3 Buffer Occupancy and throughput

Finally, after comparing the performance metrics, we now compare the overall buffer occupancies and the achieved throughput. Indeed, a high burst absorption can be achieved by overly dropping long flows causing throughput loss. Hence we are interested in understanding whether PLASTICINE's performance gain costs any throughput loss. Additionally we also measure buffer occupancies which explains the underlying reasons for the performance of different buffer management algorithms shown in previous sections.

Plasticine achieves on-par throughput: All the performance gains shown in the previous sections are achieved without any significant loss in throughput as seen in Fig. 5.5. In particular, all the algorithms achieve similar throughput except *Complete Sharing* with TCP which experiences significant throughput loss. It is an expected behavior with CS as there is no control over buffer allocation and can lead to unfair buffer utilization and consequently throughput loss. However we notice that, CS with DCTCP achieves on-par throughput. This is due to the fact that DCTCP controls the buffer utilization and under a controlled setup, there is simply no necessity for a buffer management scheme to maintain fairness in buffer utilization. We show in the next section how CS falls back to poor performance with *(i)* traffic-mix consisting both TCP and DCTCP *(ii)* presence of



(a) Single priority queue per port



(b) Multiple priority queues per port

Figure 5.5: Buffer Occupancy comparison in a pure DC traffic-mix setup.

single UDP or malicious flow.

Plasticine efficiently reduces average buffer utilization We observe from Fig. 5.5 that with TCP, PLASTICINE enabled $\approx 30\%$ lower average buffer utilization compared to other buffer management schemes. As mentioned earlier, PLASTICINE limits the utilization of long flows to a maximum of $\alpha \cdot \frac{B}{1+\alpha} \approx 33\%$ of buffer where $\alpha = 0.5$. We observe that the bound holds well, as can be seen from average buffer occupancies and 99-%ile buffer occupancy at close to zero burst size. Further, DCTCP reduced the average buffer utilization for all the schemes. Finally, we observe that PLASTICINE strategically allocates the 99-%ile buffer that enables high burst absorption and better flow completion times. On the other hand, we observe that *DT* and *CS* over allocate in average and in 99%-ile even for small burst sizes, with out any significant benefit in such over allocations.

Overall, PLASTICINE *achieves high Burst Absorption, improves Query Completion Times and short-Flow Completion times, reduces average buffer utilization and still achieves on-par throughput.*

5.3.4 TCP and DCTCP interactions

We evaluate PLASTICINE performance in a mixed congestion control traffic pattern. Particularly, we generate the same traffic pattern as in the previous sections with a change that, a flow is either DCTCP or TCP chosen uniformly at random. At all the network devices, there exists a mechanism which splits DCTCP and TCP flows into two separate queues, ECN enabled and tail drop queues respectively. The aim of this evaluation is to find whether mixed CC effects the performance of PLASTICINE and how other buffer management schemes work in this scenario.

We find that, although *DCTCP* consumes less buffer, TCP flows tend to overuse the buffer in case of *DT* and even worse in case of Complete Sharing as observed in Fig. 5.6. PLASTICINE dynamically allocates buffer to TCP flows to achieve high throughput. At the same time, bounds the overall allocations leaving enough buffer for bursts and short flows. Finally, Fig. 5.6 shows that PLASTICINE is unaffected even when TCP flows coexist with DCTCP.

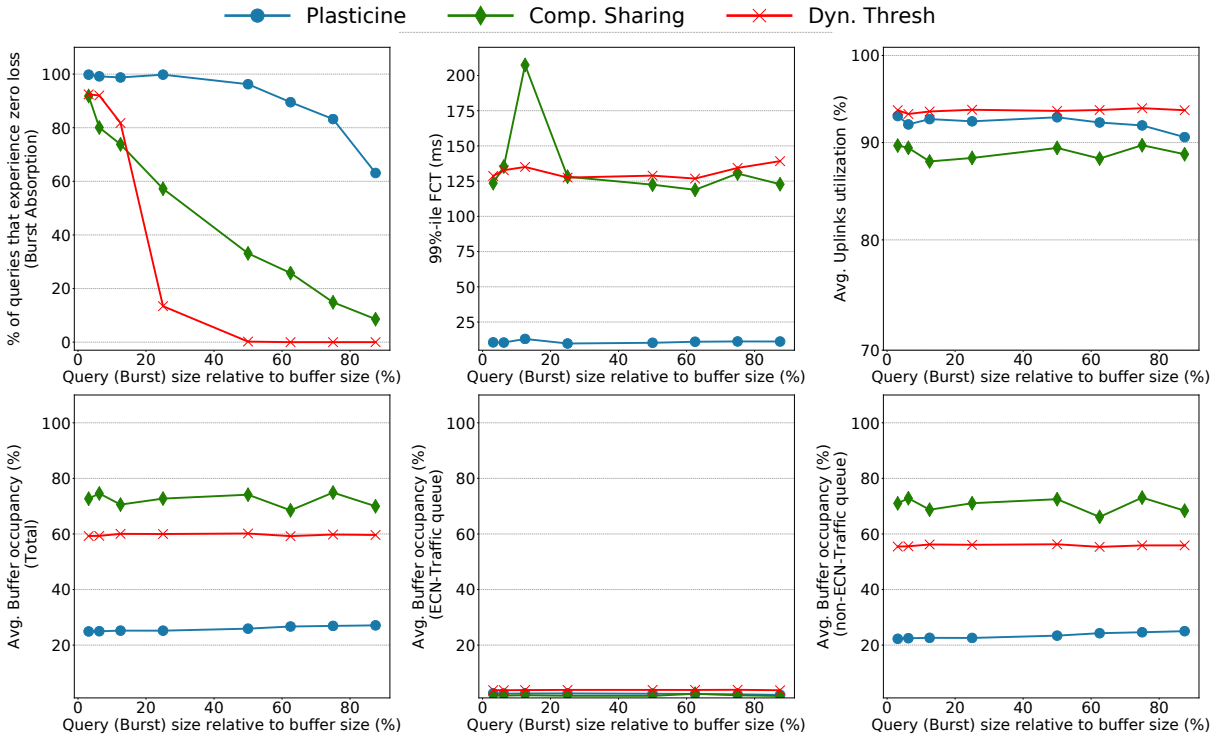


Figure 5.6: Evaluation using TCP and DCTCP mixed traffic pattern showing how burst absorption and flow completion times are severely affected in the case of Dynamic Threshold and Complete Sharing while PLASTICINE successfully limits TCP buffer consumption enabling high burst absorption and low flow completion times for short flows

5.3.5 WAN and DC traffic interactions

Finally, we evaluate our algorithm PLASTICINE in a setup where wide-area-network (WAN) and data center (DC) network traffic coexist[1]. In particular, such a traffic-mix majorly has a single point of congestion at the top of the rack switch. We set the intra-datacenter RTT to $200\mu s$ and WAN RTT to $10ms$. All the links in the network are 1Gbps except the links connecting to WAN which are 10Gbps. We use *DCTCP* in all the evaluation from here. WAN and DC are split into separate queues at the switches and a marking threshold of 200 (and buffer management $\alpha = 0.75$) and 20 packets (and buffer management $\alpha = 1$) is set for each queue respectively. A high alpha value $\alpha = 1024$ is used for all short and bursty flows in the case of PLASTICINE. We evaluate in two scenarios where WAN to DC traffic ratio is 5:1 and 1:5 with an overall load of 80% on the uplinks. We use weighted round-robin scheduling for the two queues whose weights are set to the same ratio of traffic mix. Finally, the foreground query traffic is part of data center traffic i.e., rack-to-rack. In this setup, we are interested in the abilities of buffer-management to sustain good performance for data center traffic while not losing throughput of WAN. In other words, isolation of the two traffic type. We use the same performance metrics as in the previous sections to compare different buffer management schemes.

Variation across Burst sizes

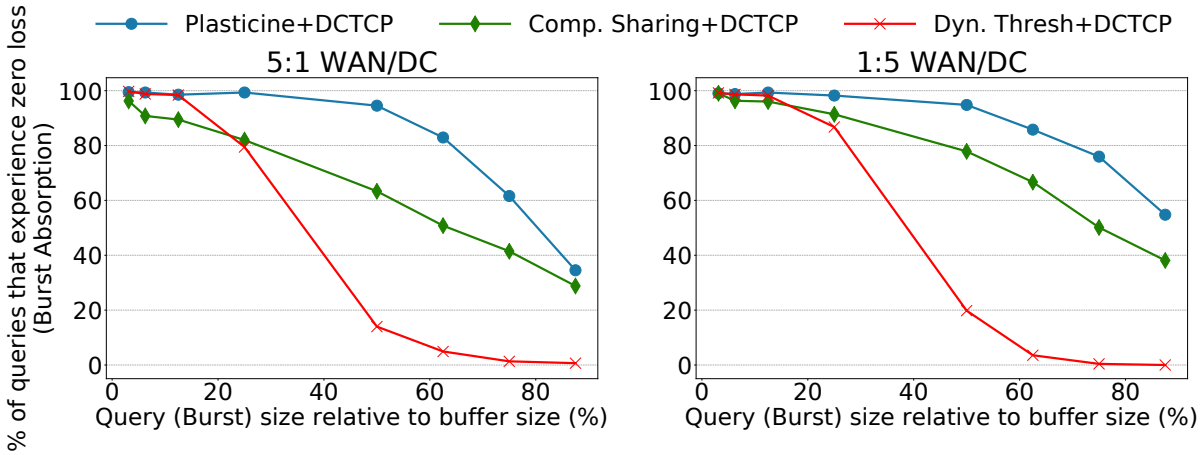


Figure 5.7: % of queries that experience zero loss i.e., burst absorption in WAN+DC traffic-mix showing how *DT* and *Comp.Sharing* suffer from the presence of WAN traffic while PLASTICINE still achieves high burst-absorption with a $40+\%$ gain.

First, we observe from Fig. 5.7 that PLASTICINE achieves significantly high burst ab-

[1]We only evaluate using DCTCP as congestion control, 1Gbps bottleneck and 10ms WAN RTT i.e., 1.25MB of BDP. We use a total buffer size of 1MB. In this case, traditional TCP performs poorly as Buffer $<$ BDP even for single port. However DCTCP enables lower buffer requirement with a marking threshold of $\approx 200KB$ in this case. For higher capacity links and today's available buffer sizes often experience Buffer $<$ BDP and how to reduce the buffer requirement is a hot-topic in research [22] which falls under congestion-control research and is out of scope of this thesis.

sorption ≈ 40%. greater than *Complete Sharing (CS)* and ≈ 80% greater than *Dynamic Threshold (DT)*. In particular, *CS* and *DT* perform much worse than in the pure DC scenario (Fig, 5.2a 5.2b). On the other hand, PLASTICINE performs well effectively isolating different traffic types. PLASTICINE dynamically allocates buffer to WAN traffic and bounds them to a maximum of $\alpha \times \frac{B}{1+\alpha} \approx 40\%$ of buffer, where $\alpha = 0.75$. At the same time PLASTICINE bounds the long flow buffer occupancy of data center traffic to a maximum of $\alpha \times \frac{B}{1+\alpha} \approx 50\%$ of buffer, where $\alpha = 1$. Finally, PLASTICINE allocates as much buffer as need to short and bursty flows enabling a high burst absorption without interference from WAN traffic.

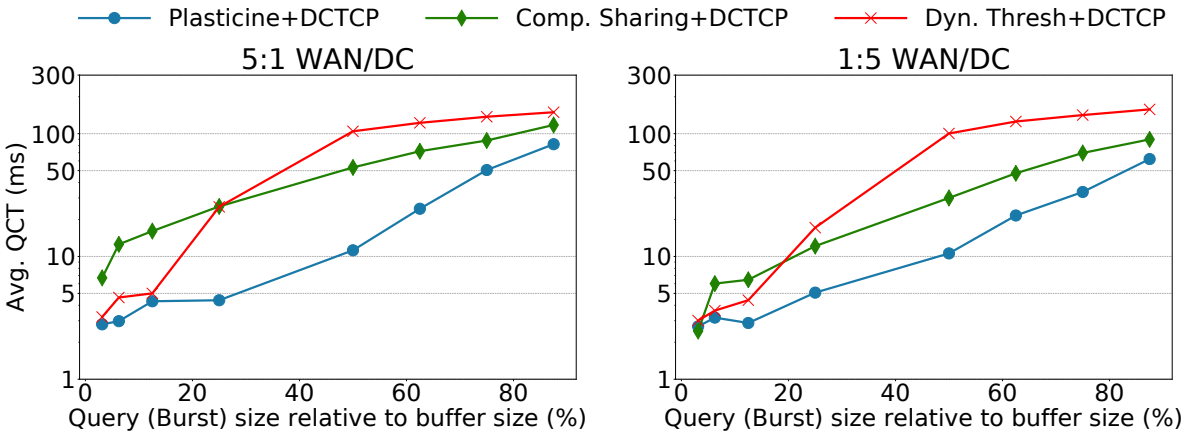


Figure 5.8: Average Query Completion Times in WAN+DC traffic-mix. PLASTICINE achieves $40ms$ and $30ms$ lower query completion times in 5:1 and 1:5 Wan, DC traffic ratios.

Analogous to burst absorption, similar performance order is seen in Query completion times in Fig. 5.8. High burst absorption enables better query completion times. As a result, PLASTICINE achieves $40ms$ and $100ms$ lower query completion times compared to *CS* and *DT* respectively with 5:1 WAN/DC traffic mix. In the scenario with lower volume of WAN traffic, PLASTICINE still outperforms convention buffer management algorithms with $30ms$ and $100ms$ lower query completion times compared to *CS* and *DT* respectively.

We observe from Fig. 5.9 that PLASTICINE outperforms in flow completion times for short flows in data center traffic. In particular *CS* performs very poorly in flow completion times although it performs well in burst absorption and query completion times. This is due to the fact that *CS* allows each queue to take up as much buffer as available and has no performance guarantees.

Finally, we show the buffer occupancy and achieved uplink utilizations in Fig. 5.10. We observe that PLASTICINE enables a lower average buffer utilization on one hand and dynamically takes more 99%-ile buffer to accomodate more buffer on the other hand. In particular, we observe that although *CS* performs good enough in burst absorption and query completion times, it experiences throughput loss when WAN traffic dominates in the traffic-mix (5:1 WAN/DC).

To conclude, PLASTICINE effectively isolates the interactions between WAN and DC

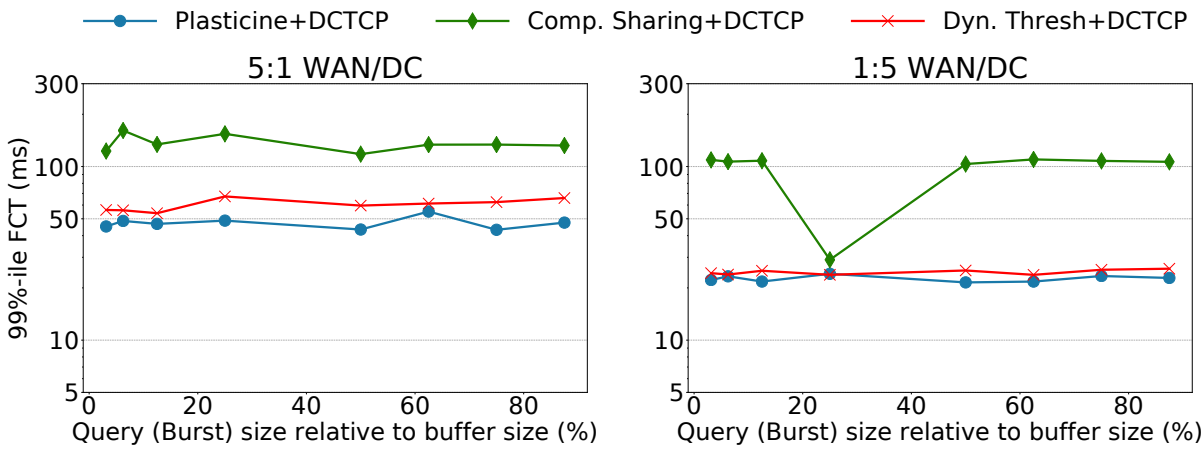


Figure 5.9: 99%-ile flow completion times for short-flows ($< 100KB$) in a WAN+DC traffic-mix.

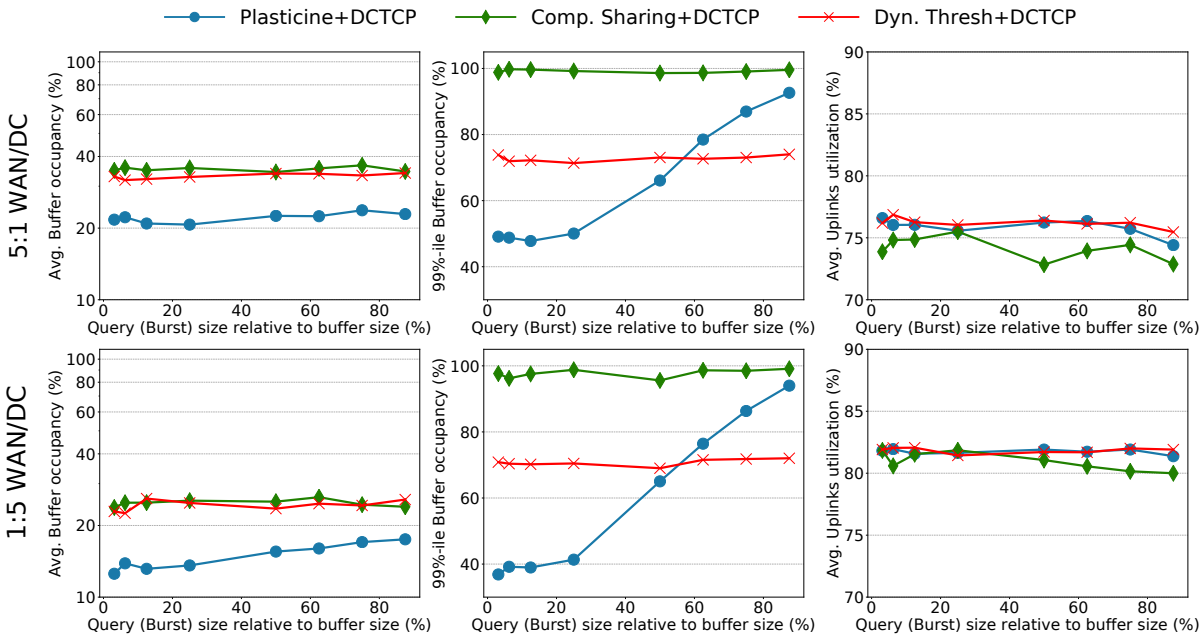


Figure 5.10: Buffer occupancy and throughput comparisons in WAN+DC traffic-mix. PLASTICINE successfully reduces the average buffer utilizations, strategically utilizes the 99%-ile buffer space as the burst sizes increase and achieves on-par throughput in both 5:1 and 1:5 WAN/DC traffic mix ratios.

traffic, achieves better performance for data center flows at the same time not losing throughput in WAN.

Chapter 6

Conclusion

Today's data center networks host applications whose workloads and aggregate traffic patterns differ widely from internet traffic. In this aspect, numerous efforts in the recent past were in the congestion control algorithms to reduce the buffer pressure. However buffer problems still persist. In this thesis, we revisit the long forgotten buffer management which is considered as a black box now. We show the inefficiencies of state-of-the-art buffer management scheme *Dynamic Thresholds* and emphasize that the available buffer managements do not address the needs of today's data center networks. We then proposed a novel buffer management algorithm PLASTICINE, analyze it in detail and establish it's theoretical bounds. An implementation of PLASTICINE on barefoot tofino (programmable switch) will be presented in [6] and is out of scope of this thesis. Finally, we evaluate our proposed algorithm PLASTICINE in a pure Data Center (DC) traffic, mixed congestion control DC traffic, WAN and DC mixed traffic patterns, using simulations. Our evaluation shows that PLASTICINE outperforms the state-of-the-art solutions with over $\approx 70\%$ gain in some cases and a minimum of 5% gain in all cases. Overall PLASTICINE enables high burst absorption, better flow completion times, offers flexibility in isolating traffic classes while still achieving on-par throughput.

Bibliography

[1] Input and output queueing. https://cs.nyu.edu/ anirudh/lectures/lec12.pdf.

[2] Nexus 9000 architecture. https://www.ciscolive.com/c/dam/r/ciscolive/apjc/docs/2018/pdf/BRKDCT-3640.pdf.

[3] Ns3 network simulator. https://www.nsnam.org/.

[4] Alizadeh, M., and Edsall, T. On the data path performance of leaf-spine datacenter fabrics. In *2013 IEEE 21st annual symposium on high-performance interconnects* (2013), IEEE, pp. 71–74.

[5] Alizadeh, M., Greenberg, A., Maltz, D. A., Padhye, J., Patel, P., Prabhakar, B., Sengupta, S., and Sridharan, M. Data center tcp (dctcp). *ACM SIGCOMM computer communication review 41*, 4 (2011), 63–74.

[6] Anonymous. To be revealed in upcoming months (2020).

[7] Appenzeller, G., Keslassy, I., and McKeown, N. *Sizing router buffers*, vol. 34. ACM, 2004.

[8] Beheshti, N., Ganjali, Y., Ghobadi, M., McKeown, N., and Salmon, G. Experimental study of router buffer sizing. In *Proceedings of the 8th ACM SIGCOMM conference on Internet measurement* (2008), pp. 197–210.

[9] Choudhury, A. K., and Hahne, E. L. Dynamic queue length thresholds for shared-memory packet switches. *IEEE/ACM Transactions On Networking 6*, 2 (1998), 130–140.

[10] Das, S., and Sankar, R. Broadcom smart-buffer technology in data center switches for cost-effective performance scaling of cloud applications. *Broadcom White Paper* (2012).

[11] Dhamdhere, A., and Dovrolis, C. Open issues in router buffer sizing. *ACM SIGCOMM Computer Communication Review 36*, 1 (2006), 87–92.

[12] Ganjali, Y., and McKeown, N. Update on buffer sizing in internet routers. *ACM SIGCOMM Computer Communication Review 36*, 5 (2006), 67–70.

[13] Ganjali, Y., and McKeown, N. Update on buffer sizing in internet routers. *ACM SIGCOMM Computer Communication Review 36*, 5 (2006), 67–70.

[14] Hahne, E. L., and Choudhury, A. K. Dynamic queue length thresholds for multiple loss priorities. *IEEE/ACM Transactions On Networking 10*, 3 (2002), 368–380.

[15] He, Y., Batta, N., and Gashinsky, I. Understanding switch buffer utilization in clos data center fabric.

[16] Krishnan, S., Choudhury, A. K., and Chiussi, F. M. Dynamic partitioning: A mechanism for shared memory management. In *IEEE INFOCOM'99. Conference on Computer Communications. Proceedings. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. The Future is Now (Cat. No. 99CH36320)* (1999), vol. 1, IEEE, pp. 144–152.

[17] Mathis, M., and McGregor, A. Buffer sizing: a position paper.

[18] McKeown, N., Appenzeller, G., and Keslassy, I. Sizing router buffers (reduxß). *ACM SIGCOMM Computer Communication Review* (Oct. 2019).

[19] Miao, R., Li, B., Liu, H. H., and Zhang, M. Buffer sizing with hpcc.

[20] Opsasnick, E. Buffer management and flow control mechanism including packet-based dynamic thresholding. *US patent US7953002B2*.

[21] Roy, A., Zeng, H., Bagga, J., Porter, G., and Snoeren, A. C. Inside the social network's (datacenter) network. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication* (2015), pp. 123–137.

[22] Saeed, A., Gupta, V., Goyal, P., Sharif, M., Pan, R., Ammar, M., Zegura, E., Jang, K., Alizadeh, M., Kabbani, A., et al. Annulus: A dual congestion control loop for datacenter and wan traffic aggregates. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication* (2020), pp. 735–749.

[23] Theis, T. N., and Wong, H.-S. P. The end of moore's law: A new beginning for information technology. *Computing in Science & Engineering 19*, 2 (2017), 41–50.

[24] Zhang, Q., Liu, V., Zeng, H., and Krishnamurthy, A. High-resolution measurement of data center microbursts. In *Proceedings of the 2017 Internet Measurement Conference* (2017), pp. 78–85.

[25] Zhou, Y. Resource allocation in computer networks: Fundamental principles and practical strategies.