

**ENERGY EFFICIENT QUALITY OF SERVICE
PROVISIONING IN DISAGGREGATED STORAGE SYSTEM**

A Dissertation
Submitted to
the Temple University Graduate Board

in Partial Fulfillment
of the Requirements for the Degree of
DOCTOR OF PHILOSOPHY

by
Joyanta Biswas
December, 2021

©

by

Joyanta Biswas

December, 2021

All Rights Reserved

ABSTRACT

With the emergence of high-performance storage devices and emerging remote access protocols such as NVMe over Fabrics (NVMe-oF), system throughput has increased several times. However, at the same time, aggregate throughput and uncertainty of traffic intensity can easily clog the network links and cause multiple QoS violations. Thus provisioning Quality of Services (QoS) becomes more challenging in such disaggregated storage systems, as it has to rely on both the storage and the network for performance. Besides that, energy management plays a crucial role in QoS provisioning, and provisioning needs to be managed cost-effectively (e.g., minimize energy consumption). Though most works mainly focus on the internal component of the storage and host servers, the network becomes a power-hungry resource as the legacy interconnect networks are being replaced with high-speed links (100 Gbps) that consumes several times more power than the conventional one. Also, at the same time, the increasing speeds generally result in much lower network utilization, which allows energy saving. From that motivation, we leverage the Low Power Idle (LPI) feature of Ethernet and propose an intelligent routing mechanism in order to maximize low power (or sleep) opportunities for the network interfaces while avoiding link congestion. In testing out our mechanisms, we have enhanced the popular *ns3* package for network modeling. Our intelligent routing mechanism serves the purpose of better performance and power tradeoff in networking components unless there is congestion in the storage server itself. To address that, we develop novel mechanisms for dynamically deciding when to move storage chunks (storage access unit) or alter the number of active chunk copies to alleviate congestion during high traffic episodes and enable traffic consolidation (and hence network energy savings) during low traffic periods. Increasing the number of active chunk copies (replication) or migrating storage chunks give the opportunity of performance enhancement during high load period and power-saving during the low period of activity. Using extensive simulations with

modified *ns3* network simulator, we provide deep insights into the migration vs. replication tradeoff. Though network congestion can be alleviated, tremendous throughput from different storage servers can still congest the TOR \iff Host link or TOR \iff Storage link in a shared environment. In that scenario, service differentiation is necessary, as different applications have different QoS requirements. In that context, we propose a QoS aware transport layer solution, which offers QoS differentiation without any compromise in overall system throughput.

ACKNOWLEDGEMENTS

This dissertation is dedicated specially to my late parents and also my loving and encouraging family members, friends, and colleagues who supported me a lot during my graduate study at Temple University.

TABLE OF CONTENTS

ACKNOWLEDGEMENT	vi
1 INTRODUCTION	1
1.1 Introduction and Motivation	1
1.2 Background	4
1.2.1 Low Power Idle	4
1.2.2 Replica Management in Distributed Storage	5
1.2.3 Network Congestion Management	7
1.2.4 Service Differentiation in Storage System	7
1.3 Goals and Objectives	8
1.4 Main Contributions	9
1.5 Outline	10
2 PERFORMANCE AND POWER TRADEOFF IN DATA CENTER NETWORK	12
2.1 Energy Management Basics	12
2.2 Energy Management Strategies	13
2.2.1 Dynamic Voltage and Frequency Scaling (DVFS)	13
2.2.2 VM Placement or Migration	13
2.2.3 Network Power Management	14
2.3 Energy Aware Intelligent Routing	16
2.3.1 Scalability of Control Mechanism	19
2.3.2 Energy Consumption of Backplane/Fabric	21
2.4 Results	22
2.4.1 Effects of Backplane Management	24
2.4.2 Comparison Between Fat-tree and Hypercube Network Topology	24
2.5 Related Works	25
2.6 Limitations	28

3	ADAPTIVE DATA CENTER NETWORK TRAFFIC MANAGEMENT FOR DISTRIBUTED HIGH SPEED STORAGE	29
3.1	Motivation and Observation	29
3.2	Proposed Methodology	31
3.2.1	Initial Placement of Chunks	32
3.2.2	Incremental Placement Optimization	34
3.3	Experimental Evaluation	38
3.3.1	Experimental Setup	38
3.3.2	Experimental Results and Discussion	41
3.4	Related Work and our Contribution	47
3.5	Discussion and Limitations	48
4	PROVISIONING DIFFERENTIATED QOS IN NVME OVER FABRICS (NVME-OF)	50
4.1	Introduction and Motivation	50
4.2	NVMe over Fabrics Internal	52
4.2.1	IO Flow in NVMe-OF	53
4.2.2	Priority Hint Passing	55
4.3	Variants of TCP and RDMA transport	57
4.3.1	TCP(Transmission Control Protocol)	57
4.3.2	RDMA (Remote Direct Memory Access)	58
4.4	Limitation of Existing Solutions	59
4.4.1	Priority Flow Control (PFC)	60
4.4.2	Enhanced Transmission Control (ETS)	60
4.5	Limitation of DCTCP and DCQCN:	61
4.5.1	Flow rate control	61
4.5.2	Limitations	63
4.6	QoS Aware TCP (QTCP)	64
4.6.1	QoS Specification	64
4.6.2	Quality Factor and Window Flow Control	65
4.7	Analytical Modeling of QTCP	66
4.7.1	A Simple Operational Model	66
4.7.2	Comparison of Model and Simulation	70
4.7.3	Convergence and Stability Analysis	70
4.8	Performance Evaluation of QTCP	72
4.8.1	Comparison with DCTCP Throughput	73
4.8.2	RTT Fairness	74
4.8.3	DCTCP Friendliness	75
4.8.4	Latency Comparison with DCTCP and D ² TCP	76
4.9	QoS Differentiation in DCQCN (QRDMA)	78
4.9.1	Effect of Different Incast Degree	79
4.10	Related Work	81

4.11	Limitations	82
5	CASE FOR IN-NETWORK QOS PROVISIONING FOR MIXED MEMORY AND STORAGE TRAFFIC IN NVME OVER FABRICS	83
5.1	Background and Motivation	83
5.1.1	Persistent Memory as Data Store	83
5.1.2	Latency and Throughput Tradeoff	85
5.2	Limitations in Existing Solutions	86
5.2.1	Variants to Address QoS Differentiation	86
5.2.2	Existing Proposals	88
5.2.3	Standalone Network QoS Management is Not Enough	88
5.3	Proposed Mechanism	89
5.4	Performance Evaluation	92
5.4.1	Coexistence of DCTCP and RDMA Traffic	92
5.4.2	QoS for Small Transfer Memory Traffic	96
5.4.3	Result and Discussion	96
6	DISCUSSION	100
6.0.1	Summary and Conclusions	100
6.0.2	Future Work	101
	REFERENCES	103

CHAPTER 1

INTRODUCTION

1.1 Introduction and Motivation

Storage access in the data center has always been remote, whether it is distributed across individual servers, concentrated in a storage area network tower, or managed by one or more storage servers. Detachment of the storage servers from the host end provides flexibility in the context of demand scaling [1, 2], data protection [3, 4], resource allocation [5, 6, 7], and high performance computing(HPC). However, at the same time, it imposes some unprecedented challenges: 1)With the increasing demand for high-performance computing (HPC), high-speed interconnection networks have become very common. An upgrade to the high-speed interconnect network increases the power consumption several times. For example, the power consumption of a 10 Gb/sec Ethernet can be anywhere between 2-10 times the power consumption of 1 Gb/sec Ethernet, depending on the number of ports and the technology used [8]. 2)As storage servers can be spawned over multiple locations and be shared amongst multiple applications, providing end-to-end quality of service(QoS) to different applications becomes difficult. Because when data is accessed over interconnect network, it requires coordination amongst different storage and network layer QoS features to guarantee end-to-end QoS.

Even though network power consumption has increased, the power

consumption can be reduced significantly with proper network energy management in place. Network power consumption goes up with the number of powered-up ports even when they are idle since the underlying link Phy is synchronous and constantly consumes power to keep transmit and receive sides in sync. Furthermore, the increasing speeds generally result in much lower network utilization. The deployment of higher-speed links is mainly motivated by technological availability, latency considerations, ability to handle highly bursty workloads, and much less by sustained high bandwidth demands. Such situations create an opportunity for power savings. During the low utilized period, the network ports can be switched to low power mode, though it requires intelligent routing, workload prediction scheme at the switch port end. Nevertheless, power reduction comes with a penalty in performance. For example, according to IEEE 802.3az standard, when switch port in idle mode, it takes 10's to 100's nanoseconds to switch back to the active mode (wake up latency) for 10 Gbps link [9], which adds in the transfer time latency ($TotalTransferTime = WakeUpLatency + TransferTime_{active}$). In order to manage a better performance and power tradeoff one has to design policies which takes into account any solution between two extremes 1) Switch port always active (Transfer time \approx network bandwidth) 2) Switch to low power mode when no packet to send (Transfer time \approx network bandwidth + WakeUpLatency).

In the context of power-saving, besides networking components like network switches or routers, storage servers also consume power when the corresponding storage server is active in serving the IO request. The situation becomes worse since in distributed storage. This is common practice to fragment the files into multiple small units, which is called **chunk**, and every chunk has multiple copies/replicas spread across multiple servers. As the primary goal of distributed storage is to offer high performance and high durability, the above mechanism increases the overall system throughput significantly. For instance, if multiple drives/storage servers host any single piece of data, all the storage drives/servers can operate in parallel to serve

requests from multiple applications destined for that specific data. Though the overall power consumption can be increased significantly as all the servers hosting chunks are active. Multiple copies also serve the goal of high durability since service providers can afford the simultaneous failure of one or two storage servers/drives since multiple copies exist of the same data piece. Consolidation of multiple requests to a single server can let other servers move to a low power state, but that violates the fundamental goal of a distributed storage system - high performance. So it requires an adaptive copy/replica management system, which can dynamically increase/ decrease the number of active chunks and reduce it when the workload drops below a threshold.

QoS Service Differentiation Parallel access to multiple copies can significantly increase the system throughput, but it has a negative impact. The interconnect network has a limited capacity, which can easily surpass by the tremendous throughput offered by the emerging remote access protocols such as NVMe over Fabrics (NVMe-oF)[10]. Traditionally, storage devices and the storage access protocols have been relatively slow and thus, the network latency in remote access storage has not been an issue. However, with NVMe-OF, network congestion is becoming a significant issue[11, 12, 13].¹ Though the Most recent HPC data center has moved to 100 Gbps interconnect links, this seems insufficient. Mellanox, NVMe-oF performance report shows that 4 NVMe SSDs can saturate 100 Gb/sec links in time, whereas 250 SATA HDDs are required to saturate the same links [14]. Moreover, with parallel access to multiple copies by different applications, the situation could worsen (e.g. incast collapse). Network congestion can significantly affect the overall performance since there will be packet drop, and as a consequence, there will be retransmission. Several works [15, 16, 17, 18] address this phenomenon, but the limitation is in offering service differentiation as these solutions offer

¹NVMe is a hardware-supported, low-latency storage access protocol that is becoming ubiquitous, and NVMe-oF is its extension that essentially handles NVMe requests over the network.

fairness amongst all application flows (equal bandwidth sharing). However, equal bandwidth sharing is not the expected behavior in most of the use cases. For example, several applications require high priority (mission-critical applications), compared to other assured service applications (e.g., email, backup, Etc.). So, it requires three-way coordination between the network layer protocols, storage layer protocol, and application's QoS specification.

1.2 Background

1.2.1 Low Power Idle

The energy-efficient Ethernet standard approved by IEEE in 2010 improves the Ethernet energy proportionality by defining a link sleep mode known as Low Power Idle (LPI). Although the standard defined the low-level mechanisms for switching to low power mode, the EEE standard does not define the strategy for deciding when to enter and leave the low-power mode. The proper configuration of EEE is critical for maximum energy saving and low-performance overhead [19] and the best setting depends on the distribution of the traffic on the network. Currently, vendors set all these parameters at default values without any knowledge of the workload. That often results in LPI not working well and perhaps cause the general perception that LPI does not work and should turn off. Previous studies show that the energy savings depend on the traffic pattern and network load [9, 20]. Some of the recent research has tried to understand the impact of EEE for different types of applications such as MapReduce [21], video streaming, scientific computing, Etc. For example, MapReduce has a specific traffic pattern during different phases that one could exploit for network energy management without introducing substantial latency. Since data centers comprise different applications with different workload patterns, an adaptive network traffic management scheme (e.g., traffic consolidation, batching) is required to increase sleep time opportunities in the network links.

Flow path selection and consolidation to maximize sleep opportunities can be pretty challenging in an extensive data center network because a workable scheme must simultaneously consider the availability of endpoint (or server) resources (CPU, memory, Etc.), local bandwidth demands at the switches, and end-to-end blocking/delay on the network paths. In a small data center, this can be accomplished via a central controller that has global network and endpoint visibility. However, such a scheme does not scale. An intelligent choice of flow paths through the DCN and their potential reorganization requires global visibility into the network not present in traditional networks. Although a software-defined network (SDN) [22] provides the same kind of visibility, there is still a matter of cost and other deployment issues associated with it.

1.2.2 Replica Management in Distributed Storage

As discussed in the introduction, All the HPC storage solutions like Amazon Dynamodb [23], Google Bigtable [2], Hadoop [24] has a common trend of fragmenting a file into multiple chunks and then distribute the chunks amongst multiple storage server in order to increase system throughput and durability. All the solutions have a different way of implementation to chunk the data and distribute the chunks. However, the central theme is common, distribution of data across different nodes available in the data center. Let's look at how the fragmentation and distribution is done in a very commonly used distributed file system HDFS. The overall diagram of the data distribution is shown in Figure 1.1. Here one large file *File.dat* splits into 4 chunks (A,B,C and D) of fixed size based on the configuration of the file system. Then the chunks are replicated three times (standard replication factor for most of the distributed systems). Then these multiple copies/replicas of every chunk are distributed amongst available data nodes (responsible for storing the chunks copy). So if any application wants to access file *File.dat*, it can read in parallel from all the four available data nodes; as a result,

throughput becomes four times higher theoretically. Namenode is responsible for keeping track of the chunk's status (location, utilization, etc) and sending that information to the application to read directly from all the data nodes that host the chunks of the specific file.

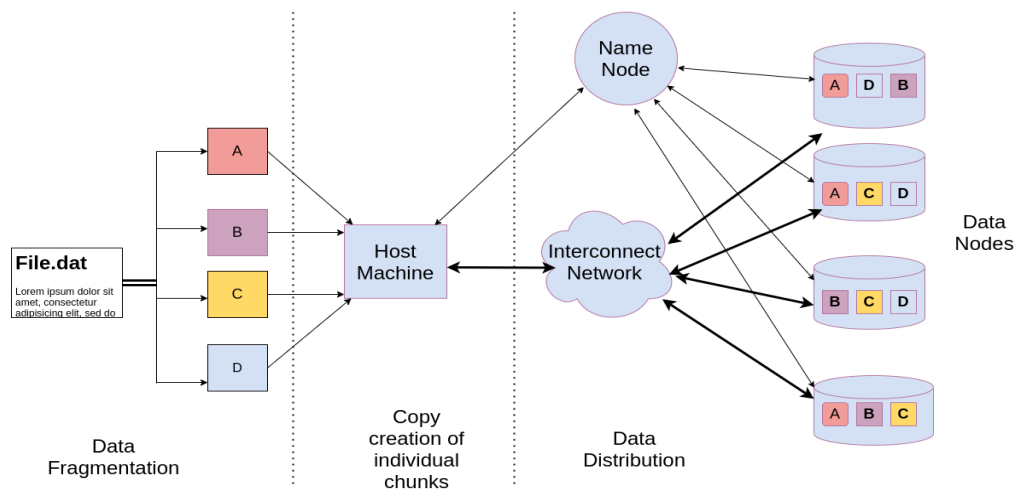


Figure 1.1: Data fragmentation, data replication and data distribution in HDFS system. Note that, the same principle can be applied to all other distributed storage solutions

Multiple copies can enhance the performance several times. It also offers performance enhancement when there is data sharing, as there are three copies located in three different locations, the overall load can be distributed among these three copies. Despite performance enhancement, this mechanism can increase the overall power consumption significantly because all the data nodes hosting that file copies should be active all the time, regardless of the popularity. Instead, if the file copies are not that popular, only one data node hosting one copy can be put active, whereas others can switch back to low power state mode. Besides that, without a proper network congestion control mechanism in place, performance can be severely affected. The main reason is, this architecture allows multiple high-performance storage devices to react simultaneously for even a single IO. As a result, collective throughput can surpass the interconnect network bandwidth, drop packets, which leads to

performance degradation. So network congestion management plays a crucial role in offering high performance in a distributed storage environment.

1.2.3 Network Congestion Management

Even though modern data centers use high bandwidth interconnect networks (~ 100 Gbps), packet drop and retransmission issues can cause under-utilization of the interconnect capacity. For example, when Transmission Control Protocol (TCP) is used in a 100 Gbps network, the throughput can even drop to ~ 60 Gbps! [25]. The main reasons are software overhead, congestion avoidance mechanism Etc. Different hardware-based solutions have been proposed [26, 27] to avoid software overhead. However, congestion and performance issues persist, as the primary reason hides inside the mechanism how TCP reacts to the congestion. Hardware-based solutions can accelerate the performance at the host or target end (in figure 1.1 **Host Machine** and **Data Node**, but not inside the interconnect network. Congestion in interconnecting networks mostly depends on how TCP controls its stream rate when a packet drop occurs (which signifies network congestion). Even though conventional TCP variants [28, 29] do not suit well in the HPC environment, TCP was redesigned later [15, 30, 31] to match the demand of different QoS sensitive applications. One issue that remains unresolved in TCP solutions is the differentiated treatment of different flows/applications. Provisioning differentiated services for applications is necessary because different applications have different QoS requirements. It is unexpected to distribute shared resources (e.g., network bandwidth) without considering the QoS demand of the applications.

1.2.4 Service Differentiation in Storage System

Storage protocols like NVMe offer differentiated service to some extent. For example, NVMe specification features a strict priority with weighted round-robin mechanism, defining queues of different priorities (e.g., Urgent, High,

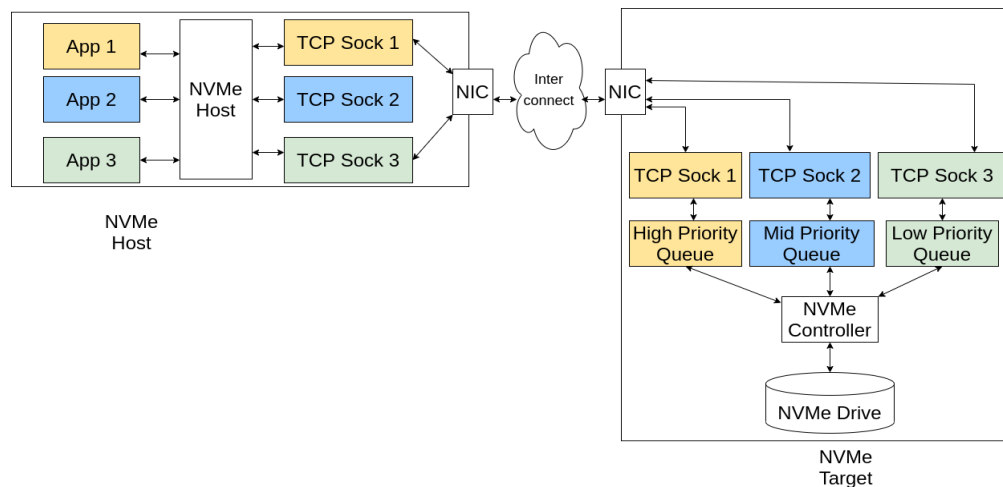


Figure 1.2: Though NVMe specification offers differentiated service, but when accessed using any fabric solution (TCP sock1/ TCP sock2), flow fairness dismisses the end to end differentiated service features goal

Medium, Low). It can assign different applications' IO requests to different queues. So, when NVMe is used as a standalone solution, differentiated service can be offered. However, the issue arises when NVMe is accessed remotely, as in that case, application and target NVMe has to communicate via a transport layer protocol (e.g. TCP) (Figure 1.2). Note that fabric solution offers two alternatives: TCP and RDMA (Remote Direct Memory Access), but TCP is ubiquitous and generally used as a transport. As discussed in the previous section (1.2.3), TCP lacks in offering differentiated service during the congestion episode, and in a distributed system network congestion becomes quite common, the end to end differentiated treatment remains unattainable.

1.3 Goals and Objectives

The main goals of this research work is:

1. Attaining better performance and power tradeoff in a data center environment. To accomplish this objective, mainly two components are considered:

- (a) **Network component:** Leverage LPI, and propose schemes that can increase the sleep time opportunities of the network links without any significant drop in the overall system throughput.
 - (b) **Storage Component:** Adaptively increase/decrease the chunk replicas so that during a high load period, more replicas are active to distribute the workload, whereas, during the lull activity period, the number of active replicas are reduced in order to save energy.
2. Provisioning end-to-end quality of services (QoS) differentiation in distributed storage environments. NVMe over Fabrics features need to be explored for this goal, as it offers a massive performance boost over other SCSI-based solutions. In order to achieve the end-to-end QoS goal, the transport layer needs to be enhanced to offer differentiated treatment and then map the well-defined NVMe priority to the appropriate transport layer priority.

1.4 Main Contributions

LPI feature can be used to implement energy-aware policies. To validate the effectiveness of those policies, it requires existing network simulators that are either designed for this purpose or would be easy to extend. Amongst many other simulators, NS3 [6] turns out to be a good candidate because of its flexible design and widespread familiarity in the research community. However, NS3 currently has little to offer in terms of energy management; it merely defines some parameters on energy consumption. To remove this deficiency, we have implemented an energy model for NS3 based on currently available network device energy management features in both inside-the-box fabrics (e.g., PCI-E) and outside-the-box fabrics (e.g., Ethernet) [32]. In our next work [33], we propose a lightweight mechanism to coordinate the actions of different controllers and show that such coordination can result in significant energy savings without needing VM migrations or shuffling of established flows.

[34] compares the mechanism proposed in [33] with established work and also provides a complete analysis of how the performance and energy consumption varies for different topologies and different configurations. In [35] we propose an adaptive chunk copy management scheme, which significantly improves the performance during the busy period of activity and consolidates chunks during the lull period to save energy. This work considers multiple factors that make this work noble: 1) chunk read to write ratio, 2) consistency issue, 3) tradeoff between copy replica vs. copy migration.

In the context of end-to-end QoS differentiation, we first enhance the existing NVMe storage stack in order to send hints from the host end to the target end. Because the IO request is generated from the host site and in a shared environment, QoS per IO basis is crucial. We leverage the NVMe admin command's spare bits to send the priority and use that priority for both the storage and network priority. Of Course, the priority hints need to be interpreted in terms of network and storage priority. We then propose a unified QoS aware transport layer solution [36], that works on both the fabrics alternatives (TCP and RDMA). Though the solution proposed in [36] works in a standalone TCP or RDMA environment, that does not work as expected when there is long transfer storage traffic and very small transfer memory traffic. We enhance the proposed solution and show the outcome in chapter 5.

1.5 Outline

The whole dissertation is organized as follows. Chapter ?? discusses the 1) basis of energy management, 2) proposed scheme which leverages the LPI feature, and 3) the details of an intelligent routing mechanism that enhances the sleep time opportunities in network links. This chapter also includes a detailed analysis of the proposed scheme and the effectiveness compared to existing solutions and different network topologies. Chapter 3 discusses in more detail the motivation regarding the adaptive copy management

mechanism, proposed mechanism, and experimental evaluation of the proposed mechanism. Chapter 4 includes the details of end-to-end QoS provisioning, and chapter 5 discusses some of the limitations of the previous QoS aware scheme and provides detailed analysis and workaround solutions for that. The dissertation ends at chapter 6 with a summary and future work direction.

CHAPTER 2

PERFORMANCE AND POWER TRADEOFF IN DATA CENTER NETWORK

2.1 Energy Management Basics

Different system components like CPU, storage media, or network adapters have notion of different operational states (a.k.a power states). Each of these states consumes a different amount of power, and based on that performance of these states varies. For example, PCIe defines four link power state levels: fully active state (L0), electrical idle or standby state (L0s), L1 (lower power standby/slumber state), L2 (low power sleep state), and L3 (link Off state). As links transition from L0 to L3 states, both power saving and exit latency (latency involved in becoming operational from sleep state) increases, which causes performance degradation. Advanced Configuration and Power Interface (ACPI)[37] is a standard that defines the power states, and allows the operating system to manage the performance and power trade off for the devices attached to the computer system.

2.2 Energy Management Strategies

2.2.1 Dynamic Voltage and Frequency Scaling (DVFS)

DVFS[38] is widely used to match system power consumption with required performance. The processor power consumption P_{whole} is the summation of static and dynamic power: $p_{whole} = p_{static} + p_{dynamic}$. The dynamic power ($p_{dynamic}$) is the main portion of the processor power dissipation. $p_{dynamic}$ can be expressed as following:

$$p_{dynamic} \propto C * V^2 * f \quad (2.1)$$

where C is the capacitance of the CMOS, V is the supplied voltage and f is the frequency at which the processor is driven at.

Since processors are responsible for almost $\approx 50\%$ of the overall system power consumption, regulating the supplied voltage (V) and the frequency (f) can significantly reduce the overall power consumption. The performance trade due to regulation is proportional to both V^2 and f (e.g. reducing frequency to half, can reduce both the performance and power consumption by 50%). DVFS has been considered as a method of reducing the overall power consumption. For example, [39], [40] leverage the DVFS feature and proposed intelligent task scheduling algorithm for a set of task with QoS requirement. The objective of their work, is to regulate the voltage and frequency as such, the power consumption can be minimized, and at the same time QoS being satisfied for all the tasks.

2.2.2 VM Placement or Migration

To save power at the endpoints, VM consolidation has also been considered as an efficient method in literature. Most of them [41],[42] have considered mainly VM placement and migration as optimization problems to reduce the server energy consumption without any non-local network side considerations.

Thus, the server energy savings may be accompanied with increased packet drop and jitter.

2.2.3 Network Power Management

Operating States of the Network Links

Nearly all of the network links are currently based on the serial links that use differential signaling, because such a technology can easily handle noise and does not suffer from cross-talk and clock skew issues. The links are then built using one or more such serial interfaces called “lanes”. This has led to the notion of repurposable Phy layer, i.e., the same basic Phy module that can support very different higher layer technologies including PCI-Express, Ethernet, Infiniband, Fiber Channel, etc. At a very low level, all the links possess three operating states:

L0: This is the normal operational state with highest power consumption, say, P_{L0} . P_{L0} does not have much dependence on the utilization since “filler” Idle messages are constantly exchanged whenever the link is idle even for very short periods. In other words, the link is always 100% busy.

L0s: This is a sleep state with entry and exit penalty typically in 10’s to 100’s ns range. In L0s, the power level P_{L0s} is around 40-50% of P_{L0} depending on the design [9]. There is usually a trade-off between transition latency and sleep power. L0s control applies independently to both sides of a bidirectional link. The link is kept “trained” during L0s and thus a part of the interface must stay awake.

L1: This is a much higher latency non-operational state with exit latency in 10’s of μ s range, but generally with a very low power consumption (e.g., 10% of P_{L0}) [9]. This state requires a handshake between transmit and receive sides and link retraining upon exit from low power mode. If either side refuses to go into L1, L1 will not be entered. The training symbols are not exchanged (like L0s) during L1 and thus link retraining is required on wake up, which makes the exit latency quite high.

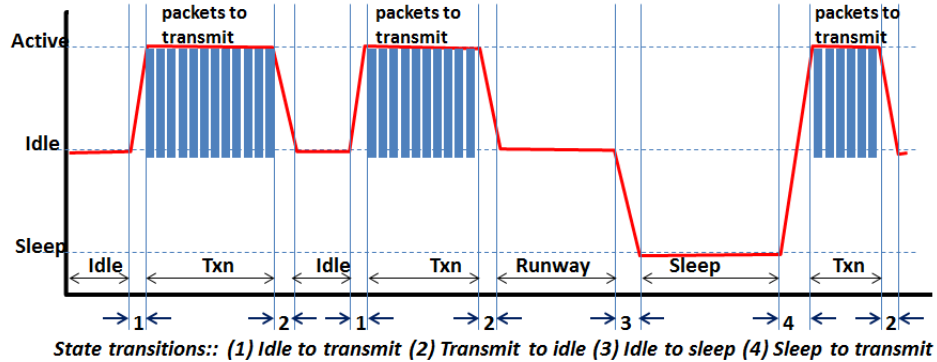


Figure 2.1: TOS packet transmit model

Low Power Idle (LPI)

we propose two types of LPI models: Transmitter Only Sleep (TOS) model and (ii) Transmitter Receiver Sleep (TRS) model as described below. Basically, TOS is based on the availability of hardware level L0s sleep state and TRS is based on handshaking based L1 state, as stated earlier.

TOS Model: The TOS model is mainly driven by the hardware, hence it can apply energy management method at fine time grain for example gaps between successive packets on a link. Low power consumption of L0s state is offset by the very low exit latency to move from sleep to active state and resume transmission. The packet-transmit diagram for TOS is given in Fig 2.1. During active transmission periods, the device is in active state S_{active} and hence consumes active power. During the idle (no transmission) period, the link steps down to sleep state after some “runway” interval, as discussed by Kant in [43]. We have followed the same “runway” concept both in case of TOS and TRS, to avoid unnecessary delay introduced due to transition to sleep states. Dynamic runway to adapt to the network state is left for future studies.

TRS Model: Compared to the TOS, the TRS model works at a coarser time grain and uses L1 sleep state. The runway for TRS is kept larger than TOS because of its higher exit latency. In the TRS model, each device is

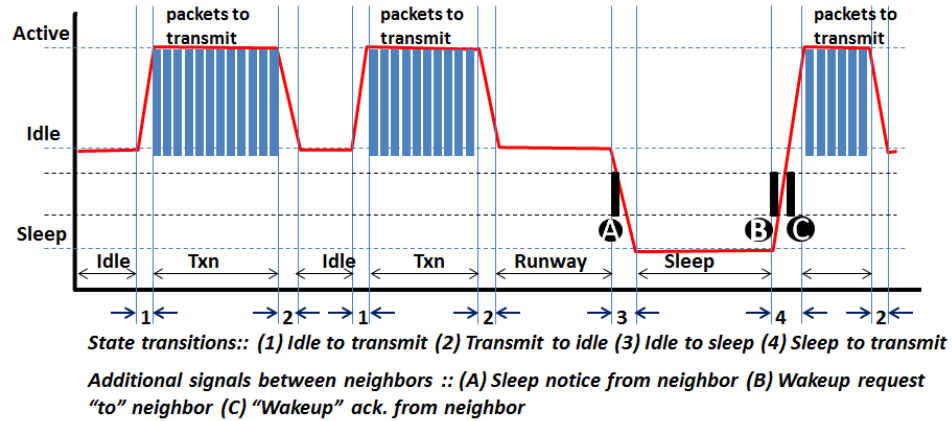


Figure 2.2: TRS packet transmit model

aware of its neighboring devices' energy state. Before switching to deep sleep state, an idle device has to inform its neighbor of its transition to sleep state. Likewise, if the neighbor of a device is in sleep state, the device has to wake up the sleeping neighbor, before forwarding the network traffic. This adds a level of management burden on both the transmit and the neighboring receive devices. This overhead involves - additional queue depth to buffer packets until the neighbor wakes up and acknowledges its active status, and an inter-device handshaking through packets exchange (sleep packet and wake up packet). The TRS state transition model and the packet transmission modes with the overlay of inter-device wake-up packets is shown in Fig 2.2. We study this model along with its effect on latency, queue depth and opportunities to extend the sleep state. This mechanism can save more energy than TOS when the link utilization is low, as we have discussed in the results.

2.3 Energy Aware Intelligent Routing

In order to utilize the LPI feature of Ethernet, it is required to consolidate the traffic into fewer links, so that the rest of the link's corresponding port can be put into low power state to reduce power consumption. In that path, we

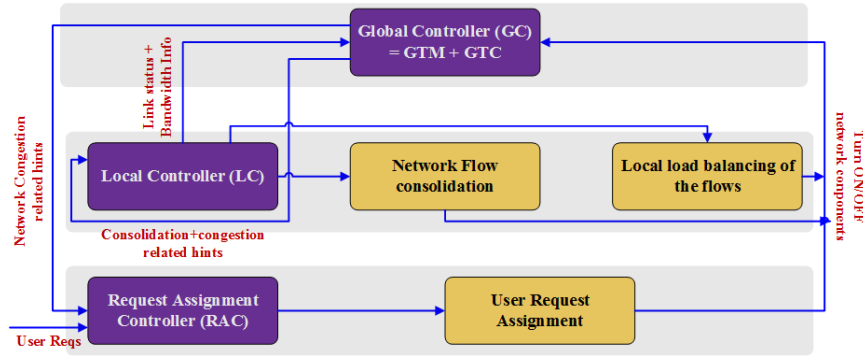


Figure 2.3: Interaction between various controllers

proposed a probabilistic routing scheme, which adjust the number of links dynamically based on the network utilization. Network control in our model is effected by three entities: Global Controller (GC), Local Controller (LC) and the topology aware Request Assignment Controller (RAC). The functionality of GC is actually separated into two entities, namely, Global Traffic Monitor (GTM) and the Global Traffic Consolidator (GTC). The former does the monitoring and hinting, whereas the latter does consolidation of ongoing flows by reshuffling, as explained before. Since we have not used the GTC in this work, so henceforth we are going to use the term GC and GTM interchangeably.

When multiple controllers are involved, it is crucial to build a consistent strategy that needs to be followed by the controllers to avoid any contradicting actions [44]. In terms of control, GTM merely provides hints to LCs and RAC for traffic consolidation and external request placements respectively. It is reasonably active in collecting up to date information, so that it can provide timely hints to RAC and LCs. The main challenge for the GTM is to build *consistent strategies* for the LCs and RAC in order to maximize sleep opportunities and to avoid network congestion.

Each LC periodically collects low-level network statistics (For example, link utilization, number of active flows etc.) and sends it to GTM. GTM builds a global view of the network based on those accumulated intelligence.

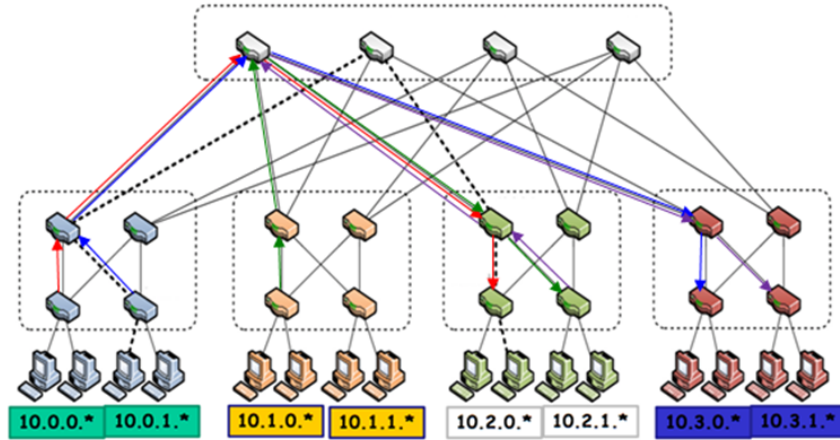


Figure 2.4: An illustration of fat-tree network. The flows are mostly

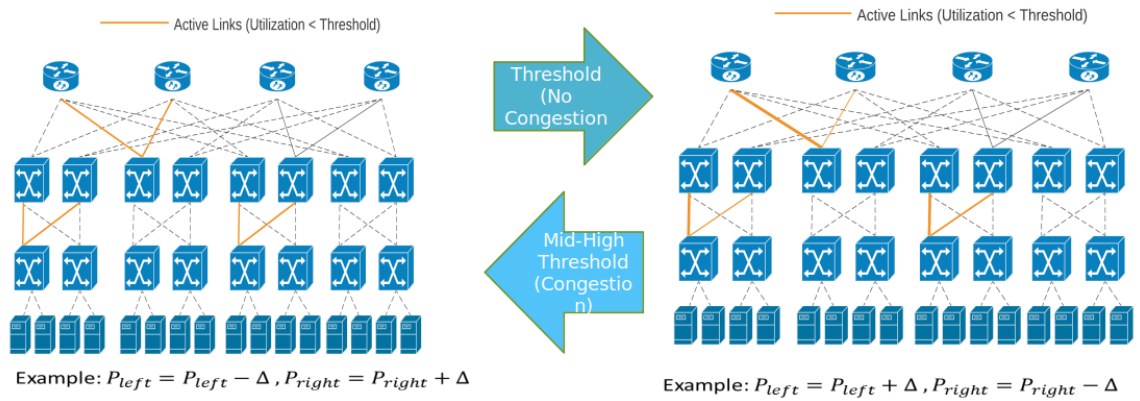


Figure 2.5: An illustration of our scheme. When the utilization is beyond threshold, the probability to the left increases, otherwise decreases.

Depending on the global state of the network, the GTM sends back very little hints that can help LCs to make routing decisions. By adopting this simple information exchange mechanism, it is possible to keep the GTM scalable even with reasonably large networks.

The LCs are responsible for routing of a new incoming flow and can only do local consolidation of the flows. For example, in the fat-tree network shown in Fig. 2.4, an edge switch has two links towards the aggregation switch. From a power management perspective, it is better to concentrate traffic on one of them (if possible), so that the other link can have longer idle duration and

hence better chance to save power by going into the sleep mode. It is clear that if each LC greedily focuses traffic on one of the links, this may result in congestion or suboptimal traffic distribution at higher levels. The purpose of GTM is to provide hints to LCs to avoid this situation.

The overall scheme works as shown in Figure 2.5. When the overall utilization is below some threshold, the probability of forwarding the traffic to the left links increases by Δ . When the LC can sense utilization going up, then the the probability is decreased by Δ . In general for a switch (LC in our case) with $n + 1$ possible links, the probability factors are calculated as follows. Assume that the probability factors for the $n + 1$ links are P_1, P_2, \dots, P_{n+1} respectively from left to right. Thus if P_1 is incremented (or decremented) by some amount, then probability factors for all the other corresponding links are calculated by solving the above expressions:

$$\begin{aligned} P_2 + P_3 + \dots + P_{n+1} &= 1 - P_1 \\ P_i &= \zeta^{i-1} (1 - P_1) \quad \forall 2 \leq i \leq n + 1, 0 \leq \zeta \leq 1 \end{aligned} \quad (2.2)$$

where ζ is a system constant. The idea is that the probability factors of the links decrease exponentially from left to right. Thus the traffic is mostly consolidated at the left-most links, which provides enough opportunity for the links towards the right to sleep.

2.3.1 Scalability of Control Mechanism

The amount of information exchanged between LCs and GC increases both with the size of the data center and the dynamism of the traffic in terms of congestion episodes. It is important to note that each switch handles the congestion locally in its outgoing links; only the congestion in the down-links requires exchange of hints between LCs and GC. However, the congestion in the downlink means that a lot of data headed to the corresponding rack. If this happens frequently or persistently, it means that there is deficiency in the hints are provided to RAC by GC, rather than a scalability issue. In this regard

we note that the congestion notification to the GC is an event driven process, rather than a periodic one. In the fat-tree topology, as the size of network (i.e., the parameter k) increases, so does the number of alternative links, which is likely to reduce congestion episodes. Nevertheless, as the network size increases, the GC must keep track of more information and provide hints to more switches.

Suppose that at some point in time, the fraction X links have exceeded the higher threshold. The corresponding LCs will then continue sending the congestion notification until the congestion reaches down to the lower threshold. From the definition of Fat-tree, in a K -array Fat-tree, we have $K^2/4$ core switches, each connecting to all K pods. This gives $K^3/4$ downlinks from core switches. Each of the K pods has $K/2$ aggregate switches, each with $K/2$ downlinks, or a total of $K^2/4$ downlinks per pod from aggregate switches. The number of edge switches is same as aggregate switches, each connecting to $K/2$ "hosts", which yields another $K^3/4$ downlinks. Thus we have a total of $3.K^3/4$ downlinks.

In real data centers, the "host" of a pure fat-tree structure can be considered as a rack connected via "top-of-the-rack" (ToR) switch to the edge switch on one side and the rack servers on the other. Thus a pure fat-tree structure beyond the rack level has $N = K^3/4$ racks and $3N$ downlinks. A standard rack can typically hold 42 1U servers, and as many as 96 blade servers. A rack typically also has two uplinks to load balancing and reliability, but let's ignore that for simplicity. We also do not consider congestion on the links going to individual servers in order to keep focus on the fat-tree which lives above the rack.

Now to get some insight into the request handling capability required by the GC, we need to choose the fraction of congested links, say X , realistically. In a pathological scenario, it is possible that $X = 1$, i.e., every link is congested. This is completely unreasonable since real data center networks are designed with enough capacity to make packet drops and congestion a very infrequent occurrence. We assume that it is adequate to be prepared to handle, on the

average, one congested link in every downlink set of a switch, i.e., $X = 2/K$. (Note that this assumption correctly accounts for the fact that congestion becomes less likely in larger network due to more alternate links). Thus, the required request handling capability required by the GC, say R , is given by:

$$R \leq (3.K^3/4) * X = 3.K^2/2 = 6N/K \quad (2.3)$$

Now consider a case of $K = 32$, which yields $N = K^3/4 = 8K$. With 40 servers per-rack, this corresponds to a very large data center with 320K servers. For this, $R = 1.5K$. Suppose that the GC recomputes hints every 100 ms and send them out to individual switches to mitigate congestion. Such responsiveness should suffice for most interactive applications. Now, receiving, processing, and responding to 1.5K congestion notification requests in 100 ms should be possible for high end server with 16 cores. The networking requirements are also very modest – 15K packets/sec ingress and outgress, each perhaps only 100-200 bytes in size and should not stress the minimum 10 Gbps capability that one would expect to have here. As to the storage requirements, maintaining $3N$ or 24K different congestion entries is quite small and could well be mostly residing in processor caches (L1-L3).

A truly large data center may require breaking up GC into per-pod GCs coordinated by a global GC, but we do not consider such extreme cases here.

2.3.2 Energy Consumption of Backplane/Fabric

In the basic switch fabrics, all the incoming and outgoing net-devices are connected through the device backplane as shown in Fig 2.6. The switch backplane consumes more power compared to the ports, connected to it. The switch fabric periodically checks the activity of it's connected network devices. Based on the inactivity interval, the fabrics can switch to low power (SLEEP) state. While in SLEEP state, the penalty for processing the first packet would be $t_0 + t_{wakeup}$, where t_{wakeup} is the wakeup penalty and in the order of 100s of μs (C6). Because of the high cost of fabric exit latency, the fabric wake time suppresses the net device wake up time.

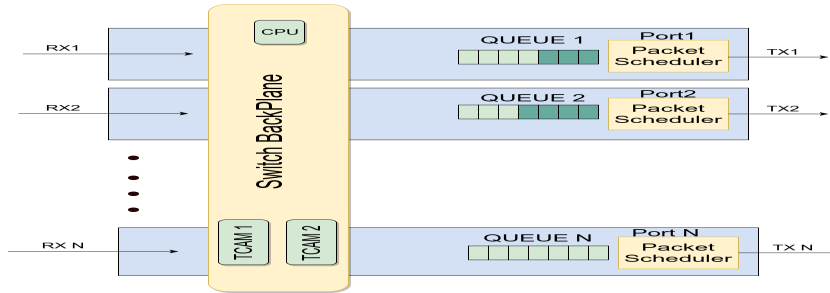


Figure 2.6: Switch Fabrics

2.4 Results

We compare our result with default ECMP routing, with no power management scheme. We show the result in figure 2.7. It is clear that the energy management techniques like TOS and TRS can save a significant percentage of power, regardless of routing techniques (probabilistic routing and ECMP). The best power saving can be achieved by probabilistic routing with TRS. The power savings with PR + TRS as compared to the baseline is 40%, although they come at the cost of increased delay. The PR + TOS mechanism also provides great power savings, but with a smaller increase in delay ($16 \mu\text{s}$). While the delay increase is significant in percentage terms, it is important to keep in mind that the end to end network delays are in 100 of μs range, and the impact of additional $16 \mu\text{s}$ delay on the performance depends on the nature of the application. While some HPC applications can be very latency sensitive, most are unlikely to be affected by a small increase in networking delays. The additional delays are a result of additional queuing delay due to traffic consolidation and is unavoidable. Also TOS provides better power and performance trade-off than TRS in case of ECMP. The percentage of power saving and delay increase with ns3 ECMP routing and TOS (ns3+TOS) is 35.1 % and 2.8%, whereas with TRS the values are 29 % and 71.9% respectively. The reason behind higher energy consumption and higher delay in TRS is the frequent transition between active and deep sleep state. TOS does not show the same trend, because of smaller wake up delay and no association of

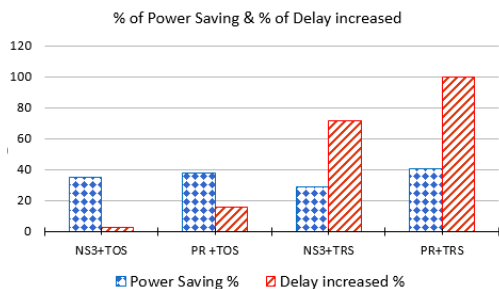


Figure 2.7: Comparison of Different Scheme

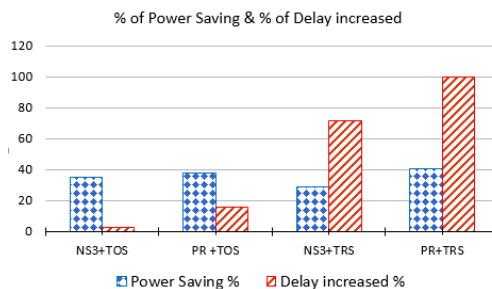


Figure 2.8: Comparison With Elastic Tree

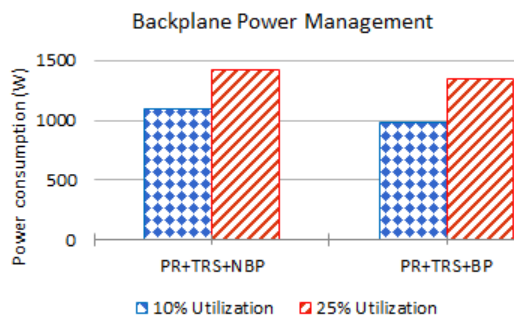


Figure 2.9: Power with BP management

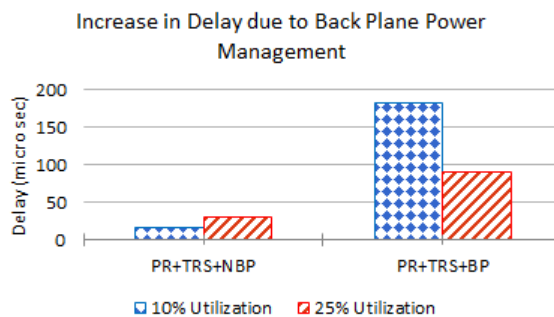


Figure 2.10: Delay with BP management

request-acknowledgment message exchange mechanism (Section 2.2.3).

We next compare the probabilistic routing (PR) with a deterministic routing (DR) scheme in Figure 2.8. DR is similar to greedy approach proposed by the Elastic Tree [45], except that the priority of the links are changed dynamically. The GC provides congestion hints to the LC and based on that the priority is changed. In DR routing scheme, traffic is forwarded to the highest priority links up to some threshold. When this threshold exceeds, the remaining new flows are forwarded to the next highest priority link, and so on. In our experiment, we assume the maximum threshold of DR to be 0.8.

2.4.1 Effects of Backplane Management

Fig 2.9-2.10 show effect of backplane (BP) management, on power consumption and network delay. From Fig 2.9 we can observe that with 10% utilization (10 %), TRS+PR can reduce the power consumption by 10.3% using the the backplane power management. However, the delay due to BP increases significantly as shown in Fig. 2.10, especially in case of low utilization. This is because the back plane gets much opportunity to switch back to sleep state, while causing more flow delay. In case of higher utilized network, the backplane gets less opportunity to switch into deep sleep state. So, the delay impact is less severe. From Fig. 2.10 we can observe that with backplane management, the average delay in case of 25% utilization($117.111\mu s$) is less than the average delay associated at 10% utilization ($181.811\mu s$).

2.4.2 Comparison Between Fat-tree and Hypercube Network Topology

Although fat-tree is the most popular data center network topology, other topologies are also used, particularly in HPC data centers. These include hypercube, toroid and related topologies. Here we compare hypercube and fat-tree with respect to our probabilistic routing (PR) mechanism. To make the hypercube and fat tree infrastructures comparable, we equate the bisection bandwidth of the two while selecting the k for fat-tree and the dimension d for hypercube. The bisection bandwidth for fat tree is $(k^3)/8$. The bisection bandwidth for d -dimensional hypercube is 2^{d-1} . Therefore,

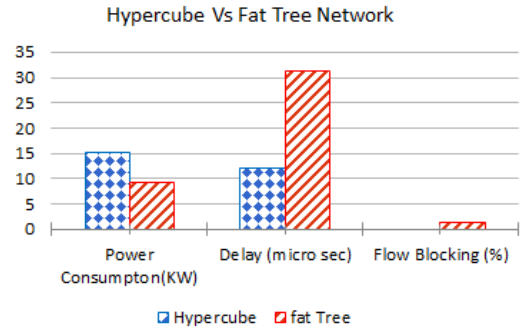


Figure 2.11: Comparison of Fat tree and Hyper-cube

$$(k^3/8) = 2^{d-1} \Rightarrow k = \sqrt[3]{2^{d+2}} \quad (2.4)$$

Based on the above equation, we create a 7-dimension hypercube topology with one host per switch, which results in an interconnection network of 128 hosts. We also use $k = 8$ for fat-tree to accommodate 128 hosts.

Fig. 2.11 compares power, delay, and blocking probability for the two networks. It is clear that hypercube yields higher power than the fat-tree but also has correspondingly lower delay. It also provides better congestion management and flow blocking. Hypercube consumes 72% more power than the fat tree topology, but with a significantly lower average delay. The average delay with the fat-tree is almost 263% higher, and the flow blocking is 4.6%. With detailed observation we found that for fat-tree topology, the congestion is dominant in the link from the core to the aggregate switch, which is the key reason for higher flow blocking.

2.5 Related Works

We classify related work into multiple categories.

Low Power Idle: Mostowfi [46] studied EEE LPI operations that put the device into two states - deep sleep and fast wake. Thaenchakun et al [47] proposed energy saving model using a control plane that utilizes an energy aware routing protocol. They showed that a combined strategy for routing protocol and energy aware path augmenting solution can provide good energy savings. Nedevschi et al. [48] proposed a buffer and burst scheme that shapes traffic to alternate active and idle periods, thereby providing increased opportunities to sleep. The authors also proposed a scheme where rate of operation of network links is dynamically adapted to the arrival rate of packets. Abts, Dennis and Marty [49] discussed link adaptation to dynamically reduce the link speed for less energy consumption using a central controller. They do so use adaptive link rate (ALR), which is a predecessor to LPI mechanism for

EEE that attempts to switch PHY(s) at run time (as opposed to simply the initialization time). It turns out that even with Rapid PHY Selection (RPS) it takes time to change the speed of the link and there is a huge overhead of changing the link speed.

Traffic Consolidation: The better outcome through LPI is possible with minimization of the number of active links and switches, in certain cases. The works in [50][51] have tried to implement different intelligent mechanisms to achieve the goal of traffic consolidation. All the solutions consist of a centralized optimizer, which tracks the traffic statistics and redirects the traffic according to the optimization solution. For example, Wang [51] discusses the correlation of peak workloads among different flows while consolidating. In our work, centralized optimizer uses topology statistics for both the traffic merging and request aggregation in a coordinated manner to avoid the conflicts in agenda.

Endpoint/VM Consolidation: To save power at the endpoints, VM consolidation has also been considered as an efficient method in literature. Most of them [41],[42] have considered mainly VM placement and migration as optimization problems to reduce the server energy consumption without any non-local network side considerations. Thus, the server energy savings may be accompanied with increased packet drop and jitter.

Request Assignment Controller: In [52], the authors present an energy aware request assignment controller that distributes the tasks on VMs based on their speed and energy efficiency. In their model the migration decisions are based on the vCPU units demanded by an application and the available capacity of the host and of the other servers in the cluster. Authors in [53] schedule the tasks on VMs that are normalized based on system and application level resource management. Their scheduling and migration of VMs is based on the vCPU units demanded by an application and the available capacity of the host and of the other servers in the cluster.

Coordinated Energy Management: A coordination between VM consolidator and network is essential to address the increased packet drop

and jitter problem, as we have explored in this work. In our work, we perform the endpoint consolidation by aggregating the external request to the target servers. There are several previous works on server-network coordination like [54][55][56][57]. The primary concern of the [56] and [57] is to place VMs close to each other, those having more mutual communication. That reduces the path length, in other word the delay and the bandwidth requirement at the higher layer of the data center. [54] has extended the work of [51], by considering VM utilization and correlation analysis for both the VMs and flows - in a coordinated way. In [55], the optimization problem of VM placement and flow routing has been represented as a unified optimization problem and tries to solve it. Doing all these approaches, both the probability of network congestion and energy consumption of the overall system (includes both end points and DCN) can be reduced significantly.

Since VM migration can be expensive both in terms of latency and network traffic [41][58], we do not depend mainly on VM migration in our work. Instead, our main goal is to provide hints to local controller and the Request Assignment Controller to work in sync. Besides that, we work on the minimal assumptions: the affinity of VMs or the correlation of loads between mutual flows and VMs are unknown in our case. In reality, these metrics would be very uncertain, so VM consolidation/migration and flow routing based on that statistics might not bring expected results.

Network Energy Model Simulator: The existing work on energy modeling in NS3 is quite limited. Hu [59] presented the first framework for incorporating the energy model in NS3. Tapparello [60] used the same model from Hu to design an energy harvesting framework. Our model is built over these basic frameworks while enhancing the energy metrics captured and granularity of data collected. We believe that these granular data and enhanced energy metrics will be of interest for future research studies and aid in understanding the underlying mechanism of device energy consumption.

2.6 Limitations

The limitation of our approach is, when there are no alternate links to forward the traffic, i.e. the congestion is at TOR \iff Storage Server link, then our proposed scheme will not work. The only way to minimize the QoS violation in that case, is load balancing the requests into multiple storage chunk (storage access unit) copies or migrate the heavily used chunks to other underutilized location. This issue motivates us to our next work - "Adaptive Data Center Network Traffic Management for Distributed High Speed Storage".

CHAPTER 3

ADAPTIVE DATA CENTER NETWORK TRAFFIC MANAGEMENT FOR DISTRIBUTED HIGH SPEED STORAGE

3.1 Motivation and Observation

Storage systems are evolving rapidly from the slow spinning magnetic media to high-speed flash technologies (i.e., SSDs) and even higher speed technologies NVM technologies (e.g., PCM, MRAM, etc.), which can rival DRAM access latency. Alongside there have been rapid developments in storage access protocols that are much leaner and lower in latency (e.g., the NVMe protocol that is rapidly becoming ubiquitous) [2, 61]. This emerging high-speed storage will invariably be accessed over the data center network by many hosts regardless of how it is deployed (from fully centralized to fully distributed per server storage). With distributed applications deployed

in VMs/containers on different hosts and accessing large amounts of data, remote storage access is a norm rather than an exception. The net result of these trends is that storage systems can drive tremendous throughput, although this typically happens only for brief periods. Furthermore, with storage device latency going into 10’s of microseconds or less, the network latency becomes a significant part of the end to end latency and needs to be managed carefully, especially during congestion periods. Thus an intelligent management of network congestion during burst periods becomes important. Since the congestion management generally requires spreading out the data (to spread out the network traffic), we also need to consider the fact that scattered data becomes undesirable during low traffic periods (which are dominant) as it interferes with the ability to achieve low network latency and exploit power-saving techniques (e.g., sleep modes) for links with low utilization. Thus, an intelligent mechanism needs to manage data location (by copying or moving) in a way to both avoid the congestion and to avoid keeping it scattered throughout the network.

The dynamic data placement to achieve the congestion mitigation vs. traffic consolidation tradeoff requires that the storage be divided up into “chunks” of a suitable size and virtualized so that it is possible to move these chunks dynamically without any impact on the applications in terms of addressing or accessing the data they need. Storage virtualization is a very

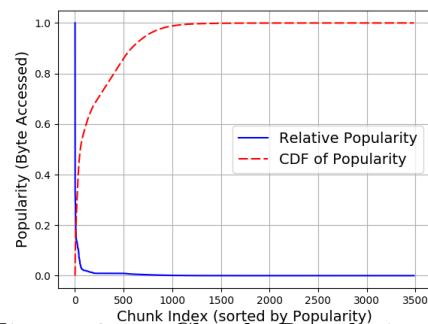


Figure 3.1: Chunk Popularity based on Temple University cluster data traces, 11% of the LBA (logical block addresses) are responsible for 80% of the traffic

mature technology and is used extensively. It may be performed by the host, a virtualization appliance, or the switch. While each mechanism has its pros and cons in terms of network impact and delays, we do not delve into those details here.

Despite virtualization, chunk movement without allowing ongoing transactions to complete can lead to lost or out of order network packets [62, 63]. This mobility limitation tends to make the problem of data movement/copy management more challenging, as discussed later in the dissertation. Besides that, it is also worth noting that the storage access pattern is generally highly skewed, which means that a small fraction of chunks will account for a large percentage of accesses [64, 65]. This is illustrated in Fig. 3.1 for our university traffic by plotting the distribution of chunk popularity. It turns out that approximately 11% of the LBA (logical block addresses) are responsible for 80% of the traffic. In fact, the popularity distribution often turns out to be similar to Zipf. Obviously, only the highly popular chunks need to be moved or copied to deal with congestion; however, since highly popular chunks are likely to be accessed from multiple hosts, managing their mobility becomes quite challenging.

3.2 Proposed Methodology

We assume a traditional data center network architecture organized as a "fat tree" illustrated in Fig. 2.4. We further assume that all storage is virtualized with the chunk size of 4MB. Although any storage chunk can reside on any storage server, it is desirable to keep the chunks close to the applications that access them the most while still consolidating the traffic under normal conditions. This can be done initially by solving an optimization problem [66, 67], which we discuss in section 3.2.1. Solving such an optimization problem requires knowledge of various access types and intensities, which may be available based on historical behavior, however, the access patterns are likely to change over time. Thus in addition to the initial placement, we also need a mechanism for continuous monitoring of the entire network and occasional incremental optimization that moves or changes the copies of a certain number of chunks. We expect the need for incremental optimization to arise only occasionally and likely involve only a small fraction of all the

chunks.

3.2.1 Initial Placement of Chunks

For initial placement, we formulate an optimization problem where x_{un} is a decision variable which is 1 if either the application or the chunk u are assigned to node n and 0 otherwise. Suppose r_i and r_o are the cost of accessing a chunk that is within and outside the node respectively, i.e. $r_o > r_i$. To linearize the problem, we also introduce an auxiliary decision variable e_{uv} , which is 1 if and only if the application u and chunk v are placed on different nodes. Let w_{uv} denote the weight (or relative intensity) of accessing chunk- v from application- u and I_u the IO demand by the application or chunk u . We could then define our *Traffic-aware Application-Chunk Placement (TACP)* as follows:

$$\text{Min} \quad \sum_{(u \in \mathcal{A} \vee v \in \mathcal{C})} w_{uv} e_{uv} r_o + w_{uv} (1 - e_{uv}) r_i \quad \text{s.t.} \quad (3.1)$$

$$\sum_{n=1}^{\mathcal{N}} x_{un} = 1 \quad \forall u \in \mathcal{A} \vee \mathcal{C} \quad (3.2)$$

$$\sum_{u \in (\mathcal{A} \vee \mathcal{C})} I_u x_{un} \leq \mathcal{L} \quad \forall n \quad (3.3)$$

$$w_{uv} = 0 \quad \{u, v\} \in \mathcal{A} \vee \{u, v\} \in \mathcal{C} \quad (3.4)$$

$$e_{uv} \geq x_{un} - x_{vn}, \quad e_{uv} \geq x_{vn} - x_{un}, \quad (3.5)$$

$$e_{uv} \geq x_{un} + x_{vn} - 1 \quad \forall u, \forall v, \forall n \quad (3.6)$$

$$e_{uv} \in \{0, 1\}, \quad x_{uv} \in \{0, 1\} \quad \forall u, \forall v \quad (3.7)$$

Here constraint in eqn(2) states that every application and chunk copy is assigned to some node. Equation(3) states that the overall IO rate of the node (denoted \mathcal{L}) is respected. The constraint in eqn(4) states that there is no IO between two applications or two data chunks. Constraints in (5)-(6) ensure that if $e_{uv} = 1$, then u and v are placed on different nodes.

The above formulation can be used at multiple levels in a real network where a *node* could represent pod in entire fat-tree, rack within a pod, and

servers within a rack. During the initial placement problem, the TACP is solved twice to determine in which *pods* and then which *nodes* inside that pod the application/chunks are assigned.

Theorem 1 *The problem TACP is NP-hard.*

Proof 1 *This can be proved by reduction from the Minimum K-cut problem (MKP) in a graph representing flows between nodes.*

Because of the NP-hard nature of the TACP problem, we propose the following heuristics to solve it efficiently. We solve the problem in two stages. First, we assign apps/chunk-copies to a set of virtual nodes, each of which corresponds to a real node and has a specified IO capacity (i.e., every node is assumed to have the same IO capacity). Next, we map the virtual nodes to the real nodes of the network.

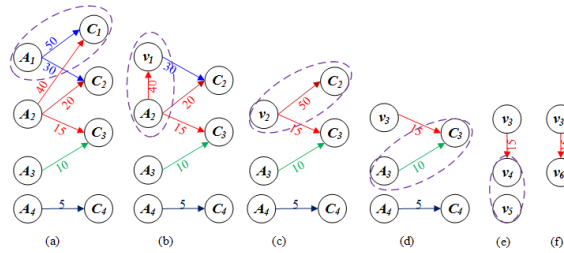


Figure 3.2: Assigning apps/chunk-copies to the virtual nodes

In the first step, we construct an affinity graph Fig. 3.2(a) with application and data nodes (A_i 's and C_j 's) and weighted edges showing traffic demand from A_i 's to C_j 's. We then greedily choose the edge with the highest weight (i.e. $A_1 \rightarrow C_1$ in Fig. 3.2(a)) and merge its vertices to create a new composite vertex, and direct all incoming and outgoing vertices from/to the merged node to the composite vertex. Fig. 3.2(b) shows this with composite vertex v_1 . The composite node corresponds to application A_1 and chunk C_1 being placed on the same node. The composite vertex is attempted to be merged further so long as the co-location does not exceed the capacity of a virtual node. E.g., in Fig. 3.2(d) merging v_3 and C_3 would exceed the virtual node capacity, and thus at this point, the algorithm looks to a distinct merger (to create another

virtual node). The end result is two virtual nodes v_3 and v_4 that cannot be merged further.

At this point, there may be some disconnected components that are merged/packed into at most N virtual nodes using the typical bin-packing solution [68]. Thus v_3^* and v_6^* become two virtual nodes after the first step.

In the second step, the virtual nodes are assigned to the pods. For this, we start with the n virtual nodes from step 1, and if $n < N$, expand them to N virtual nodes by inserting $N - n$ dummy nodes. We then recursively apply Kernighan-Lin (K-L) graph bi-partitioning algorithm that minimizes the cut weights across two partitions. The K-L algorithm divides these N v-nodes into two partitions each one with a size of $N/2$ v-nodes (assuming N is even). The K-L algorithm is again applied to these two partitions if they have non-zero virtual nodes. This process is repeated until each partition has more k virtual nodes. These partitions can now be thought of individual pods, where the virtual nodes are assigned from left-to-right based on their app/chunk loads (from maximum to minimum).

Theorem 2 *Assuming N is even, the K-L algorithm will be called at most $(\frac{N}{k} - 1)$ times.*

Proof 2 *The K-L algorithm divides the virtual nodes into two equal-size partitions. Thus if we start with N virtual nodes, then from the formulas of G.P. series it is easy to verify that after $(\frac{N}{k} - 1)$ steps we will get the partitions with the size of k .*

Henceforth the term “**node**” explicitly refers to a “**rack**”, since all our strategies beyond this point are based on the rack-level estimates.

3.2.2 Incremental Placement Optimization

Since the application behavior and hence the network traffic will vary over time, it is essential to monitor the traffic constantly and incrementally make

adjustments to the position or number of copies of highly used chunks.¹

To facilitate chunk migration/replication, we assume that we initially deploy k copies (e.g., $k = 3$) for every chunk across the storage servers. These copies include the initial number of copies needed in the initial optimization (the active copies) and a few extra copies (inactive copies). The additional copies, if any, are placed according to some fair distribution rule, for which we use a simple hash-based approach described below. The purpose of the extra copies is to allow for inexpensive copy activation when required by the high demand without creating substantial additional network traffic. All inactive copies are synchronized opportunistically (to minimize activation cost) but active copies are kept entirely consistent.

In addition, the GC maintain a bitmap $BM(c)$ for each chunk c where each entry is primarily k bits long (we use $k = 3$). Thus if $BM(c, i)$, $i \in 0..k - 1$ is 1(0), then a predefined hash function $h_i(c)$ for this position provides the location for the i th active(inactive) copy of chunk c . The GC also stores the activation ordering of any chunk using $k - 1$ additional bits which it uses during copy deactivation (thus each BM entry costs $2k - 1$ bits for any k). The LCs do not require access to this bitmap. The mechanism will, however, require communication between the GC and the virtualization engine during the time of migration/replication, but detailed consideration of the overheads of this interaction is beyond the scope of this work.

We assume that the episodes involving the arrival of large traffic bursts and their subsequent dissipation are infrequent, as observed from several traffic traces. Therefore, the frequency of update of BM is expected to be relatively low. Nevertheless, in very large data centers, the maintenance of a centralized data structure such as BM may be undesirable. This aspect will be explored in our future research.

¹It is also possible to occasionally re-run the initial optimization but this may not be desirable if it leads to substantial data movement.

Copy Activation and Deactivation

For both replication and migration of chunks, we use the following mechanism based on the traffic monitoring (in bytes/sec) by the LC. For replication, the LC watches the outgoing traffic of the node (this is primarily the read traffic), and for migration, the incoming traffic (this is primarily the write traffic). When this traffic overshoots a certain high cutoff threshold τ_h , the LC reports the event to the GC along with the top P highly used chunk numbers with their respective utilization.

Suppose that the current utilization of the i -th chunk is U_i . Then we have the following situations:

Replication: Suppose that chunk i has n active replicas with evenly distributed load among them. Then, creating another copy (replica) of that chunk will reduce their estimated load to approximately $\frac{n}{n+1}U_i$.

Migration: Migrating the chunk from node m to p will reduce load U_i from node m and increase U_i at destination node p . For any chunk with multiple active copies in the system, the incremental optimization migrates only the copy from the node that has raised the event, while keeping the locations of the remaining active copies being unchanged.

Based on this approximation the GC selects chunk from the list in descending order of their utilization at the nodes and activates the replicas at some other node that (a) has an inactive replica of that chunk, (b) can accept a certain amount of additional load (i.e. $\frac{n}{n+1}U_i$ in case if replicating the i -th chunk) and (c) has the least load (preferably preoccupied) satisfying (a) and (b). The GC continues to activate replicas until the expected load of the congested node goes below the normal limit of τ_{n2} . In case there is no node that satisfies these conditions, that chunk is not replicated or migrated and GC moves to the next chunk in the sorted list.

When the traffic surge fades away, our scheme needs to deactivate some copies activated during congestion so that the overhead of maintaining the consistency among the replicas is avoided. A migration also involves a copy

creation initially followed with new traffic directed to the copy. Only when the ongoing traffic dies down, the original copy is deactivated. Deactivating extra copies when the traffic subsides can be rather tricky. Deactivating it too early after the traffic burst subsides could hurt in terms of delay since there still might a built-up backlog that must be cleared.

There could be several policies for copy deactivation. Each of them has its advantages and downsides. It may be difficult to find a single copy with low utilization under our traffic forwarding mechanism. So infrequently, the LC updates the GC with the set of Q chunks having low utilization. From the list, GC selects only the chunks with multiple active copies. It then deactivates the last created copy of the chunk and shifts its load equally to other active copies. (Note that no deactivation will happen if the remaining copies cannot handle the extra load).

Concurrency Control

Since we use multiple copies as a way of relieving congestion, we must address the multi-copy consistency issues as well for chunks that are shared across applications and thus could get access requests for them asynchronously from multiple applications. As usual, this can be achieved in two ways: (a) via locking or pessimistic concurrency control (PCC) [69] or (b) via optimistic concurrency control (OCC) which rolls back any conflicting transactions [70]. It is well known that the OCC works better in environments with low contention for the resources. Assuming that the chunk sharing is not prevalent, the contention is expected to be low, and hence we choose to implement the OCC. In OCC, the resource access by a transaction must still be monitored; however, the requester can proceed without locking. When the transaction is ready to commit, a check is made if any other transactions have committed since this transaction started. If so, the transaction is rolled back and may retry after some delay to avoid the conflict. The transaction also checks for conflict upon arrival and backs off if another transaction has already started.

Conflicts are still possible since there is a small gap between determining that there is no conflict and actually recording that the transaction has started.

It is important to note that our concurrency control operates at the storage level and is only concerned with individual read and write operations. Additional concurrency control may be applied at a higher level (e.g., for database transactions running on top of the storage system).

3.3 Experimental Evaluation

3.3.1 Experimental Setup

To comprehensively evaluate the network congestion control mechanism we have enhanced the popular NS3 network simulation package to include the modeling of chunk based storage access, concurrency control, and traffic control via local controllers (LC) and global controller (GC). We built a small fat tree infrastructure for $k = 4$ with 100 Gb/s network links in the core and the aggregate layer and 10 Gb/s at the access or edge level.

For the storage, we consider device characteristics similar to those for currently available low-latency NVMe SSDs. However, we do not consider or simulate the complexities of the storage systems here – assuming that the storage device utilization is kept reasonably low, the effect of queuing for storage devices is not important to our study. For routing, we use the Equal-Cost Multi-Path Routing (ECMP) with a point to point connections. Also to avoid acting upon intermittent short-lived spikes, we perform *exponential smoothing* of the congestion metric. Some of the key parameters for the experimental setup are given in Table 3.1.

Application and Data Chunk Model

In our system, we have M application instances and N data chunks. We define the data chunk access intensities following a Zipf distribution over the N ($N > M$) unique chunks with the decay factor $\alpha = 0.8$. We then

Table 3.1: Simulation Configuration Values

Parameters	Values	Parameters	Values
Fat Tree Size (k)	4	Mid Cut-off (τ_{n2})	55%
Zipf dist. (α)	0.8	Normal Cut-off (τ_{n1})	45%
#application instances	47	Default chunk size	256K
#unique data chunks	1500	SSD Read Latency	25 μ s
#servers per rack	14	SSD Write Latency	100 μ s
High Cut-off (τ_h)	75%	Smoothing factor (β)	0.85

randomly map the applications to the chunks. Based on the mapping between an application to the set of chunks, we determine the application frequencies. Without any sharing on the chunks across applications, the application set should also follow a Zipf distribution. Additionally, we share each chunk across a small set of applications.

Copy Activation Overhead

As stated earlier we can either create copies proactively or reactively. Under *proactive control*, we decide m alternative locations per chunk using m hash functions and create copies in advance, which are then synchronized in the background. In *reactive control*, we create copies as needed (i.e., at the time of congestion). Additionally, under replication, as we keep multiple copies of chunks active and share them across a small number of applications, we use Optimistic Concurrency Control (OCC) model to keep the copies consistent.

Chunk size and Metadata Overhead of our Scheme

The suitable chunk size is configured by the virtualization layer and it involves balancing the overhead of metadata maintenance (which prefers large sizes) vs. the overhead of moving chunks and controlling false data sharing (which prefers small chunks). For our mechanism, smaller chunks are preferred, and the chosen size of 256KB is unlikely to be burdensome in most situations because of the highly skewed nature of storage accesses. Note that our mechanism only needs to store the chunk id, offset, and three active locations

of that chunk which results in a few bytes (32 bytes) per chunk. This has a miniscule storage overhead (0.0125%).

As described in section 3.2.2, for the bitmap GC uses only 5 bits per chunk (for 3 extra copies), hence for a petabyte of storage with 256K chunk size, the bitmap will require 2GB of memory space. GC also stores the node level utilization for all the nodes to decide the best-fit location for the chunk while replication or migration. LCs do keep track of the chunks inside a single node which constitutes of chunk number, server number, and utilization. We make the assumption that at most 10% of the data is active at any point in time for which LC requires an in-memory monitoring structure. Given the chunk usage are roughly balanced across the pods in the data center (but there could be local imbalance inside a pod) each LC will require 0.5GB to 1.5GB of memory for monitoring.

Application Traffic Generation Model

We have largely used synthetic traffic for our evaluation as it allows for an easy change in the traffic parameters. We generate traffic based on the statistical characteristics of real block device access traces collected from both Temple University Cluster² and [71]. Analysis of the trace file in section ?? shows several periods of high activity, which we term as traffic “burst”, and develop our traffic generation model around these traces.

In our model, network flow duration follows Pareto distribution with shape=3.5 and mean as 3.6 ms. The bandwidth of each flow follows a uniform distribution with mean 1Gbps and a range from 500 Mbps to 1.5Gbps. To avoid the remote communication latency we keep the bisection bandwidth of the network topology adequate. We measure performance as – *the average and the maximum observed latency faced by any application issuing remote storage access request.*

²<https://drive.google.com/file/d/1gSCToOck4oigxLd8jKO3LRLIMWobl9C>

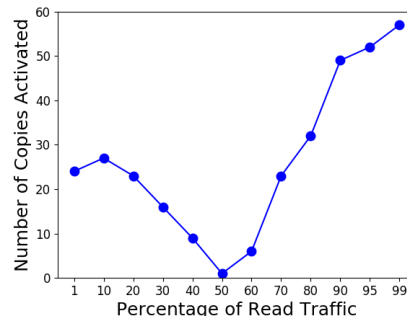
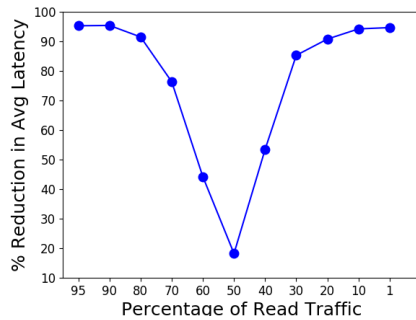


Figure 3.3: Avg. Latency for Random vs optimized placement

Figure 3.4: No. of copies activated with adaptive control

3.3.2 Experimental Results and Discussion

Optimized vs. Random Initial Placement

We start with a comparison of our optimized initial placement (*OIP*) against a uniform random placement (*URP*) of both the applications and the chunks. Under 20% utilization, the *OIP* only occupies 8 out of 16 available nodes (racks). In contrast, the *URP* involves all nodes as expected. We use the same traffic for both the cases and compare the performance of both the placements.

Fig. 3.3 shows the comparison of average latency under changing read fraction (95% to 1%). With bidirectional traffic, the network link gets congested at the extreme points where the traffic is either read or write dominated. At 95% read or 99% write traffic, we achieve the maximum reduction (approximately 95%) in average latency with *OIP*. Also *OIP* does not have any packet delivery issues whereas *URP* fails to deliver approximately 1% of the packets by the end of the simulation due to long delays. As long as the resources are available, *OIP* tries to consolidate the applications with their associated data chunks and accommodates them in the same pod (perhaps on the same node) to reduce packet propagation delay. Although mixed read/write traffic results in lower congestion due to bidirectional links, *OIP* can still achieve 18% reduction in average latency at 50% read traffic.

Impact of Incremental Optimization

Number of copies activated vs. read fraction under different policies

Fig. 3.4 shows the number of additional copies that are activated (and eventually deactivated with replication) over time. Across all the cases, in the worst case, we only activate a maximum of 50 additional copies out of all 1500 chunks. However, copy deactivation in our system follows a conservative approach. This sluggish deactivation might not have any impact on read dominated traffic, but it could increase concurrency control traffic when we replicate under write-heavy traffic.

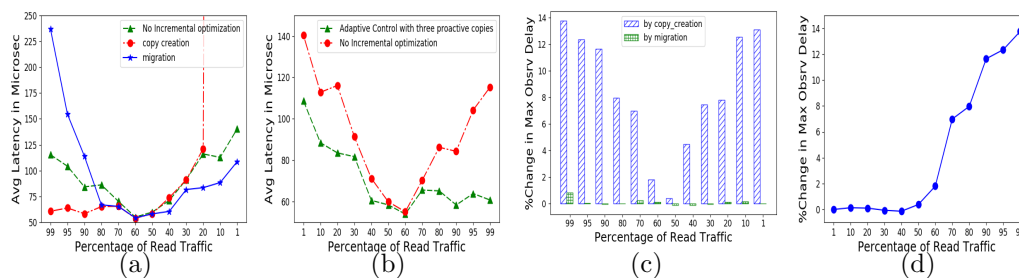


Figure 3.5: (a,b) Average latency and (c,d) maximum observed latency reduction for various chunk management mechanisms

Change in average delay with changing read fraction The following scenario can be treated as the best case for the incremental optimization where we create inactive copies *proactively* and perform *lazy synchronization*. We create inactive copies of working set on all the nodes. Hence during congestion, the traffic that we need to send to activate those inactive copies is mostly very small. Also, assuming copies everywhere ("*copies everywhere*") gives us the flexibility to activate the replica on any node. Here in the current experiment independent of the ongoing traffic characteristics (read/write), we either only migrate or replicate. Fig. 3.5(a) shows the average latency with and without the incremental optimization following the *OIP*. With read dominated traffic, the incremental optimization (which creates copies of the busy chunks) achieves approximately 47% improvement over *no opt*. This improvement results from the enhanced parallelism due to the extra copies. However,

with write dominated traffic, extra copies cost additional synchronization traffic which actually increases the delay significantly as compared to *no opt*. Instead, a migration under write-heavy traffic achieves a 22% latency reduction. Obviously, migration is not helpful under read-dominated traffic. It follows that we need an adaptive mechanism that can track the congestion and the nature of the traffic (e.g., read fraction) and accordingly decides whether to copy or move chunks. So we introduce adaptive control based on the read fraction. The system decides whether we copy or move the busy chunks. Fig. 3.5(b) compares the average latency of this mechanism against *no opt*. When the traffic is mostly read; the system tries to reduce the congestion by replicating but when the traffic largely write dominated, the system reduces the consistency control overhead by migrating the busy chunks. Please note when there is not much traffic ongoing in either direction (read% 50-60%), neither copy nor migration is beneficial as the congestion in both the directions becomes insignificant.

Change in maximum observed latency with changing read fraction

Fig. 3.5(c) depicts the change in maximum observed latency that supports our findings from incremental optimization (Fig 3.5(a)). The maximum observed latency for any flow decreases by 14% compared to *no-opt* as we replicate due to parallel request service. However, when we migrate, the maximum observed latency remains identical to that for *no opt*. So by migrating, we do not lose anything in terms of maximum observed latency, whereas copy creation always reduces the maximum observed latency. Fig. 3.5(d) confirms our findings from Fig. 3.5(b) when tested under adaptive control and compared against *no opt*. It shows improvement in maximum observed latency is achieved through replication under read dominated traffic.

Due to the performance advantage of the adaptive mechanism over simple migration/replication, we use the *adaptive mechanism* for the rest of our experiments.

From the perspective of efficient copying/migration, we would like to have

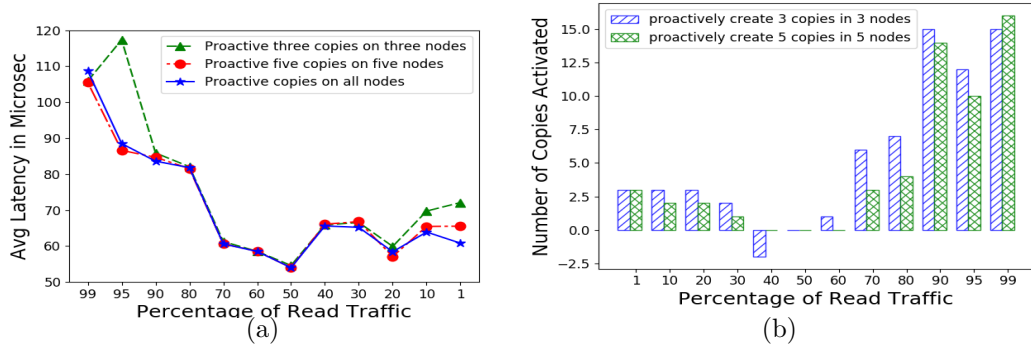


Figure 3.6: Comparing (a) Average delay and (b) number of copy activated under limited copy approach over copy everywhere

a copy of each chunk available at each node. However, this would require an enormous amount of extra storage. Hence to keep the number of copies limited, we tested our system with three (*three copies*) and five (*five copies*) proactively created copies of all the chunks. We compare the performance of these cases against *copies everywhere*. Fig. 3.6(a) shows that *three copies* are enough to achieve significant performance improvement. With mostly read traffic, no significant performance improvement is observed beyond *three copies*. However, with write traffic, increasing the number of inactive copies can provide better location choices for migration, but at the cost of increased consistency control overhead.

Many-a-times the system fails to find a new destination for a busy chunk due to IO rate limitations at nodes hosting the inactive copies. Note that we determine the locations of the inactive copies in advance, and in a highly dynamic system, it is very likely that the action taken for one busy chunk can influence the migration/replication decision of another busy chunk. So if we now compare the number of inactive copies that are successfully activated during the bursts Fig. 3.6(b), fewer chunks are activated with the limited number of copies than for *copies everywhere*. For example with 90% read traffic, we activate 34 total copies with *three copies*, 35 with *five copies* and 49 with *copies everywhere*. We do see a few exceptions; e.g., at 40% read, *three copies* creates two more copies than *copies everywhere*, yet the latency is 2% less in the latter case. This suggests that as the system fails

to find a destination for a busy chunk, it chooses a less important chunk. Creating copies of such chunks requires more migrations/replications to bring the congestion down.

Since keeping *three copies* is enough to handle congestion, we use this configuration for further experiments.

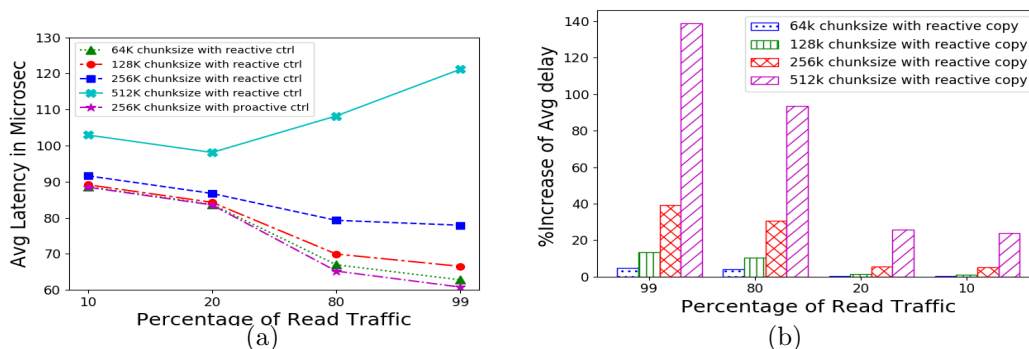


Figure 3.7: (a) Average delay with proactive and reactive control and (b) Increase in delay during bursts over proactive approach

Impact on Average delay under reactive copy creation with varying chunk size

Figs. 3.7 show the impact of increasing chunk size on average latency with inline copy creation (as opposed to our proactive method). In Fig. 3.7(a) we show the average latency by varying the chunk size from 64k to 512k. We compare the performance against our proactive approach with *three copies* (*default chunk size*).

As stated above, our default chunk size is 256k. We perform the test with four different read proportions consisting of either read or write-heavy traffic. With read dominated traffic, the delay increases with the chunk size. At 99% read, compared to the proactive approach, the reactive method with a chunk size of 512k increases overall latency by 99%. This latency increase is due to the copy creation traffic that puts additional load on read path itself. For the same reason, with the write-heavy traffic, the increase in delay due to

reactive copy creation decreases significantly. With 90% write traffic, we see an increase of 0.2%, 1%, 5%, 23% in delay with chunk size 64, 128, 256, and 512k respectively. Fig. 3.7(b) zooms into the bursts period, and the plot shows the increase in delay over proactive copy which confirms the same. Here due to bigger chunk size, the delay increases by 140% at 99% read fraction, which is more than the increase in overall delay.

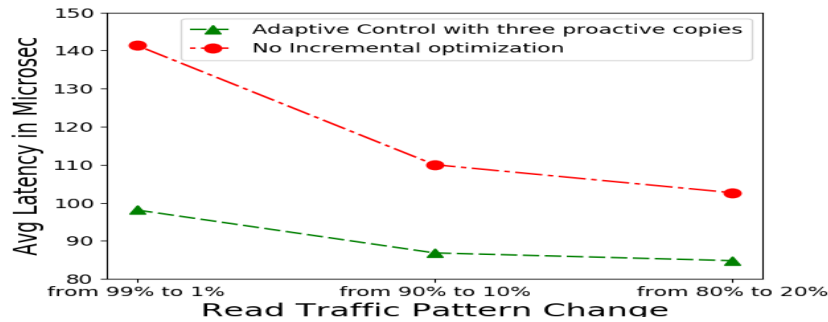


Figure 3.8: Avg latency vs. read fraction

Thus our findings from Fig. 3.6 and Figs. 3.7 indicate that with write-heavy traffic, one can afford to have reactive copies as long as the chunk size is small to moderate. With limited copies, activation may fail due to the shortage of resources at those nodes. Reactive copies will reduce the concurrency control traffic and might provide better locations for migration. In contrast, with read oriented traffic, it is beneficial to create copies proactively.

Adapting to Traffic type and Intensity Changes

Finally, we tested our system under not just change in traffic intensity, but traffic where both the intensity and the read-write ratio changes. Fig. 3.8 shows the change in average delay when both traffic type and intensity changes. Note that though some works [72, 73, 74] report the performance enhancement with respect to the number of replica increases/decreases, we do replica migration to overcome the consistency traffic issue. We use three cases where traffic type changes from (1) 99% to 1% (2) 90% to 10% and (3) 80% to 20% read,

with traffic intensity increases as before. We use *three copies* and compare the performance against *no opt.* Please note that in all our previous experiments, the read fraction was constant throughout a simulation and only varied across different runs. However, in this experiment, the read fraction of the traffic changes within a single run to test the adaptability. We found that our mechanism adapts nicely by both migrating and replicating chunks based on the change in traffic intensity and type. Also, we achieve 30% improvement over no optimization as we move from completely read to write heavy traffic.

3.4 Related Work and our Contribution

Several works focus on performance aware data placement and data replication from different viewpoints. In particular, references [75, 76] concentrate on placement in a cache tiering environment to achieve performance in terms of in-storage access latency, but the *network bottleneck* issue is overlooked. References [77, 78] focus on a distributed file system and database respectively and propose replication with *static workload estimate*. However, the authors do not consider the overhead due to *consistency control* inside the workload estimates.

The authors in [79, 80, 81, 54, 55, 56, 57] focus on the locality-aware data placement in a distributed storage system. References [79, 80, 81] mainly consider distributed big data applications (e.g, Hadoop and distributed database). In [79], the author uses the correlation between the number of accesses and concurrent accesses to find popularity. In [80] the author works on the same basis (chunk’s popularity), but the popularity is determined by correlation analysis between access frequency and age of the file. The work in [81] additionally considers the host’s capability for hosting the replicas while making the placement decisions. References [54, 55, 56, 57] discuss network-aware end-point consolidation, with references [56] and [57] place VMs with more mutual communication close to one another to reduce delay. In [54], authors have extended the work of [51] by considering VM utilization and

correlation analysis for both the VMs and flows in a coordinated manner. Scheduling the data access can also improve the performance [82], but in a large distributed storage system this would introduce *additional overhead* to manage the coordination.

Locality aware data replication/placement has also been studied in Wide Area Network [83, 84]. GlobeDB [83] uses clustering of data chunks based on the read and write amount conducted by each server and place each chunk cluster to any single server based on a cost function, which considers three metrics: read, write and consistency traffic. Reference [84] studies different heuristics to solve the NP-hard chunk placement problem.

Contrary to the above approaches, in this work, we propose (a) a dynamic data-chunk placement scheme that balances both congestion and consolidation depending on change in traffic bursts, (b) adapts to traffic behavior (read/write) by using either replication or migration to mitigate congestion; (c) incrementally reacts to any perturbation in traffic; and (d) finally addresses all the overheads due to replication, migration, consistency control.

3.5 Discussion and Limitations

In this work, we proposed an adaptive mechanism to decide whether to migrate, activate an inactive copy, or reduce the number of active copies to handle changes in the network traffic due to variations in network traffic generated by high-speed storage devices that are becoming the norm in the data center. We discuss the tradeoff between copy and migration, in the context of read or write dominated traffic environment. Through extensive simulations, we show that it is possible to achieve near 47% improvement in average delay with purely read traffic. However, for chunks that are mostly written, the best strategy is to migrate them out of the congested node and place them at some low utilized node.

While our proposed solution increases overall performance, our solution

still lacks in providing differentiated treatment for different applications, while preserving the overall system performance. The demand of differentiated treatment arises when there is resource constraint (i.e. network bandwidth, storage servers), and the resources are being shared amongst several applications, having different QoS requirements (e.g. Service level agreement). This limitation drives us for the next work, which is provisioning end to end Quality of Services for data center applications. As we target to provide end to end QoS for different applications, we need to validate our work on any standard network storage protocol stack. For that purpose, we choose nvme over fabrics (NVMe-OF) protocol, and proposed our solution based on the features/API available and discuss the enhancement we made on that NVMe-OF protocol.

CHAPTER 4

PROVISIONING

DIFFERENTIATED QOS IN

NVME OVER FABRICS

(NVME-OF)

4.1 Introduction and Motivation

Traditionally, storage devices and the storage access protocols have been rather slow and thus the network latency in remote access storage has not been an issue. However, with the emergence of high performance storage devices and emerging remote access protocols such as NVMe over Fabrics (NVMe-oF)[10], network congestion is becoming a significant issue[11, 12, 13].¹ For example, even a cheap consumer SSD can drive $\sim 25\text{-}35$ Gb/sec of throughput. The Mellanox NVMe-oF performance report shows that 4 NVMe SSDs can saturate 100 Gb/sec links in time, whereas 250 SATA HDDs are required to saturate the same links [14]. Thus, the end-to-end quality of service (QoS) is

¹NVMe is a hardware-supported, low-latency storage access protocol that is becoming ubiquitous, and NVMe-oF is its extension that essentially handles NVMe requests over the network.

becoming essential for storage access, and the network becomes a crucial piece of it.

NVMe-oF establishes the host to target connection and thus needs an underlying transport layer such as TCP or RDMA (remote direct memory access). Data centers currently use both Layer2 (L2) and Layer3 (L3) switches; therefore, an end-to-end QoS is best implemented at the transport level. This may, in turn, use features provided by lower layers, if available, but we focus on the transport layer only in this work. Two examples of lower level QoS features are (a) the data center bridging (DCB)[85], which includes the priority flow control (PFC) [86] and enhanced transmission selection (ETS), and (b) IP level DSCP (differentiated services code point)[87]. While DCB is becoming more generally available in data center switches, it must be augmented with additional mechanisms to provide an end-to-end solution. The DSCP mechanism is not very useful because (a) it is defined in terms of packet losses, which are highly undesirable in data centers, and (b) it is a hop-by-hop control, typically implemented in the routers[88].

For NVMe-oF transport, we shall consider both TCP and RDMA transports. TCP is predominant in data centers; however, because of its kernel-based and software-centric implementation, it results in high latencies. RDMA, in contrast, is largely supported by the hardware, is much leaner, and also avoids kernel transitions. RDMA is essential for access to remote persistent memory (PM) and to other emerging high-speed technologies where low latency is crucial[89].

Because of the undesirability of packet losses, we shall focus on lossless versions of TCP where the congestion control is initiated before any loss can occur. A popular and widely implemented version in this regard is the Data Center TCP (DCTCP)[15], which relies on the ECN (explicit congestion notification) [90] mechanism for congestion feedback, but in a lossless way. In particular, if the switch buffer occupancy exceeds some threshold much below the actual buffer size, the congestion is indicated, and the source then aggressively reduces the flow so that the congestion is contained quickly

and decisively. The ECN mechanism involves two bits; the Congestion Encountered (CE) bit in a packet is set when it encounters congestion on some switch in the path. When the packet reaches the receiver, and the CE bit is set, the receiver sets the ECN-echo (ECE) bit in a reverse packet (such as ACK) to inform the source of the congestion. However, DCTCP does not provide differentiated service during congestion. Thus a goal of this work is to propose a DCTCP-like mechanism called *Quality Aware TCP (QTCP)*² that allows differentiated treatment to TCP connections belonging to different QoS classes that happen to pass through a bottleneck link.

The novelty of this work is to develop QTCP and QRDMA *specifically targeted for handling NVMe-oF related storage traffic flows inside the confines of the data center*. Although it is possible to use them elsewhere in the data center as well, that aspect is beyond the scope of this work. Therefore, we do examine the coexistence of QTCP/QRDMA with flows using the regular (non-differentiated) versions of these protocols.

4.2 NVMe over Fabrics Internal

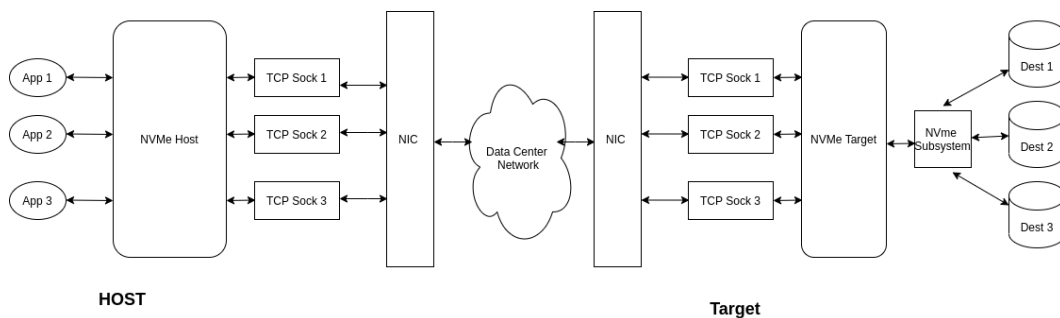


Figure 4.1: NVMe over Fabrics Internal

Figure 4.1 shows the internal architecture of nvme over fabrics protocol. NVME over Fabrics mainly consists of two modules: Host side NVMe and

²We have name conflict with another work named QTCP (<https://ieeexplore.ieee.org/document/8357943>), which we discuss in related work section

Target Side NVMe. These two modules can communicate through fabrics, and the fabrics solution could be either TCP or RDMA. In the target side NVMe, there is NVMe subsystem, which virtualizes the nvme devices to the host application. For example, in the diagram, NVMe subsystem is responsible to advertise the NVME devices (Disk 1, Disk 2 and Disk 3) to the host applications. NVMe subsystem implements the QoS features proposed in nvme specification. Some of the NVMe specified QoS features are (Zoned namespace, Queue arbitration etc.)

4.2.1 IO Flow in NVMe-OF

NVMe IO request response works in a doorbell mechanism. The application puts a request into one specific submission queue. When the IO is done by the nvme drive, then it puts the request into a completion queue and generate a interrupt into the host system, so the specific application is informed about the status of IO and can fetch the data from the completion queue.

QoS of single IO Flow

In NVMe, QoS of a particular IO can be achieved by providing relatively higher or lower priority compared to other IOs. Though there are certain ways to attain that, we leverage only one mechanism, *Queue Arbitration Mechanism*.

As discussed above, each of the IO request is put into any one of the available submission queue. NVMe offers the developer to implement their own version of queue arbitration algorithm or to use the default one. The default queue arbitration mechanism in nvme is Weighted Round Robin with Urgent Priority Class (Figure 4.2).

The arbitration mechanism works as following.

- There are mainly three types of queues: Admin Queue, Urgent Queues, Priority Queues.
- Admin queue gets the highest priority over any other queues. Admin

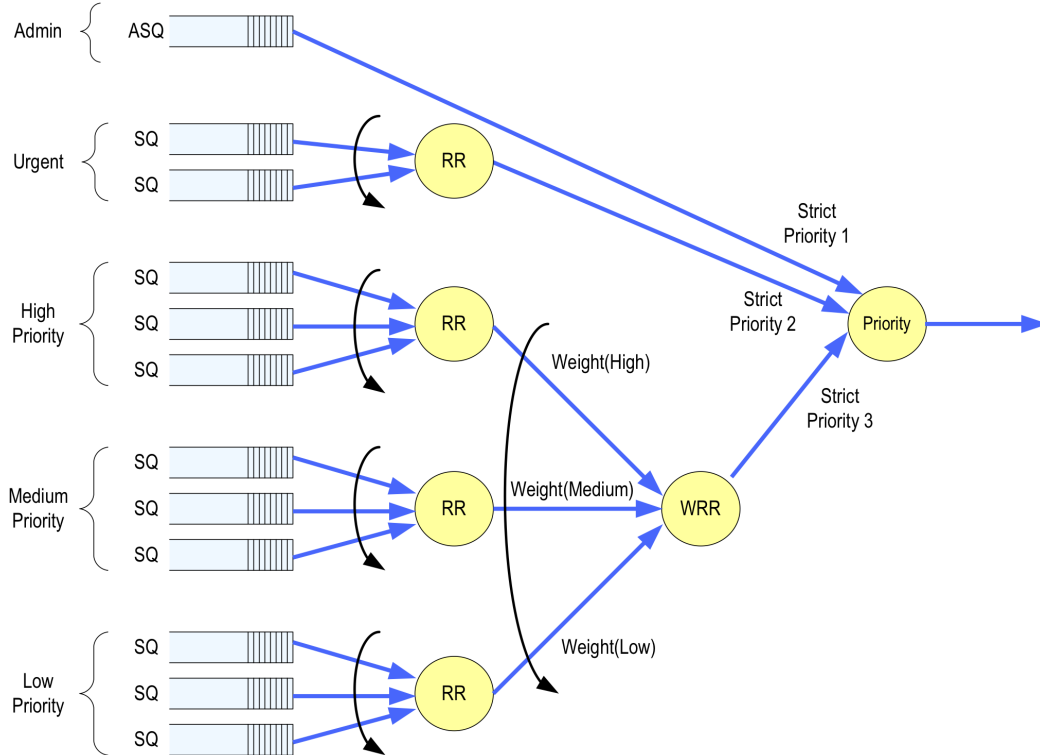


Figure 4.2: Weighted Round Robin with Urgent Priority Class.

queue is mainly used admin commands. Some of the task that admin command does: Create and Delete I/O Submission and Completion Queue, Get and Set Features, Namespace Management etc. Until there is any entry in the Admin queue, no other queues will be served.

- If there is no entry in the Admin queue, the urgent queues will be served. If there are multiple urgent queues, then all the urgent queues will be served in a round robin manner.
- If there is no entry in both Admin and any of the urgent queues, then the Priority queues will be served in a weighted round robin manner. Priority queues can be classified into three classes, low, medium and high. The weights of different priority queues can be set by the admin command.

End to end QoS of single IO Flow

As per discussion, it is obvious that IO QoS can be implemented using the existing queue arbitration feature, but that doesn't guarantee end to end QoS for single IO flow. The QoS that can be achieved only confined inside the NVMe target side, and in order to attain end to end QoS (Host application \Leftrightarrow target NVMe Drive), we need to leverage the transport QoS, and map appropriately to NVMe target QoS specification. In order to do that we need to do the following:

1. (Step 1) Transfer the priority hint (P_{io}) from the host to the target side NVMe layer.
2. (Step 2) Upon receiving the priority hint P_{io} , target NVMe should:
 - (a) Put the IO request to the appropriate submission queue,
 - (b) When the data is ready, map P_{io} to the appropriate transport layer priority and sent the data to the host.

Please note that, **transport layer priority is mainly required when the interconnect link is congested. When there is no congestion, means the NVMe throughput is less than the network capacity, and there is no need to prioritize individual flows.** In the following sections, we discuss elaborately about step 1 and step 2.

4.2.2 Priority Hint Passing

There are mainly two ways to sent the priority hints down to the NVMe drive layer. 1) Making a separate admin command 2) Attach the priority into the spare bits of the IO request. In this work we followed the No 2 approach, since sending out separate admin command per IO might increase the overall message passing overhead. Even if we send opportunistic admin command, we have to come up with a new admin command set, which requires significant effort to do.

The fundamental element of the IO request is called Capsule, and each of IO request is sent in form of Capsule. Each capsules mainly have the following information: 1) Submission Queue ID (qid), 2) Completion Queue Id, 3) Namespace ID. The submission queue id is 16 bits long. Though using 16 bits, the NVMe subsystem can manage upto 65536 queues, in reality NVMe drives usually doesn't provide that huge number of queues. The maximum number of queues vary from 64 to 128, which can easily fit within 7 bits. So we leverage the most significant 3 bits to send the priority to the target NVMe side. The overall priority sending mechanism works as follows:

- Before requesting any IO, the application requests the NVMe subsystem to create a submission queue, where it can send the IO requests. It sends an admin command, and the qid for any admin request is 1. So we mask the most significant 3 bits of the queue id (qid) with the priority (P_{IO}) and attach to the capsule.
- Upon receiving any capsule, NVMe target Unmask the first 3 bits of the qid to get the priority (P_{IO}), create a submission queue with that specified priority, response back the queue id, and store P_{IO} in order to use that for transport layer priority (if any).

As discussed earlier, when the IO is complete, the NVMe subsystem sends the data using any of the transport/fabric solution available. The available fabric solution for NVMe-OF is TCP and RDMA. NVMe subsystem uses the TCP/RDMA socket to send the data to the application end. From now on, we will focus our discussion in the fabric part. We will discuss the available options/variants of TCP and RDMA, which can be an candidate for 1)high performance storage system 2)at the same time can gurantee end to end QoS differentiation for different applications.

4.3 Variants of TCP and RDMA transport

4.3.1 TCP(Transmission Control Protocol)

As the name implied, TCP provides rate control mechanism in order to ensure integrity of the data being communicated over a network. In summary, the TCP works as follows. Before sending any data the transmitter side has to make a agreement to the receiver side, and starts sending data in batch (i.e. Frame). The receiver side needs to acknowledge the sender side about previous batch (F_{n-1}) within a specific session/timeout period (Timed out session), so that transmitter can send the next batch of data (F_n). If the timeout expires, then the transmitter resends the batch F_{n-1} . For the brevity, we exclude the details of TCP mechanism.

TCP has well defined congestion control mechanism, compared to the other counterpart UDP (User Datagram Protocol). As we discussed in section 4.2.1, we only consider to gurantee QoS when there is network congestion, so we are focusing the congestion control features available in TCP. TCP congestion control mechanism can be classified into two categories:

Reactive Congestion Control Mechanism

Reactive congestion control means triggering the congestion avoidance mechanism, when the transmitter can sense there is a missing frame ACKs, or timeout occurs. These can happen when there is a network congestion, and the switch or target end buffer is filled with the packets, so the frame get dropped. TCP CUBIC [28], TCP Vegas [91], TCP Westwood [29], TCP RENO [92] all can be classified into this category. What is different amongst these mechanism is the way they manage to module their congestion window (*cwnd*) upon detecting congestion (missing ACKs, timeout). The main issue with the reactive congestion control is, it waits until there is a packet drop due to congestion, and then halve the congestion window size (*cwnd*). In a high performance environment, this causes loss in performance, as 1) wait until

timeout to be expired 2) plus retransmit the same frame.

Proactive Congestion Control Mechanism

To avoid the highly undesired packet losses in a data center network, the congestion must be triggered when the queue length at a switch along the path or at the receiver exceeds some value K that is substantially less than the actual buffer size. This feedback can occur through the standard ECN (explicit congestion notification) mechanism which operates as follows: The network switches mark the congestion encountered (CE) bit in a packet when a queue length threshold K is exceeded. When the endpoint receives the packet, it sends the ECE (ECN-echo) message back to the sender which reduces its window. From latency perspective also, K needs to be sufficiently small for a data center environment regardless of the available buffer size. Also, in order to limit the tail latency, it is more appropriate to use tail drop (i.e., all packets that exceed the threshold K) instead of RED (random early drop) beyond K . DCTCP[15] is candidate for proactive congestion control solution that leverages ECN mechanism and suggested by SNIA [93] to be used as TCP variant when TCP is used as a fabric in NVMe over Fabrics environment.

4.3.2 RDMA (Remote Direct Memory Access)

RDMA stands for remote direct memory access. RDMA enables the NIC (Network interface controller) to directly access the target host memory, thus offloading the task of CPU to access the memory. All the processing of the packets like serialization, deserialization, encapsulation etc are done in hardware. Note that, **RDMA is a mechanism and whereas TCP is a transport layer protocol**. So RDMA can use both TCP and UDP for the transport. For example, iWARP [26] is a RDMA protocol which uses TCP for the transport. But due to performance and deployment issue in iWARP [94], this is not usually implemented. In most of the data center RDMA over Converged Ethernet (ROCe) is used, which uses UDP as the

underlying transport.

Little history of RDMA Though TCP is ubiquitous, RDMA was originally developed for the InfiniBand (IB) technology and implemented entirely in hardware for low-latency lossless transfers. It has been successfully transported to Ethernet. In particular, RDMA over Converged Ethernet (RoCEv1) [95] is an Ethernet layer protocol that allows IB traffic to be Ethernet (L2) compatible. Its successor, RoCEv2[27], replaces the IB header with the UDP and IPv4/IPv6 header, eliminating the need for the IB stack entirely. It also allows RDMA to utilize the feature of both L2 (Ethernet) and L3 (IP). Data Center Quantized Congestion Notification (DCQCN) [17] is an end-to-end congestion control protocol that works with ROCEv2, and leverages both the PFC and ECN for congestion management. Like DCTCP, DCQCN also lacks differentiated service during congestion. Our work addresses this limitation and introduces a *Quality Aware RDMA (QRDMA)* mechanism similar to QTCP, but in the context of RDMA.

4.4 Limitation of Existing Solutions

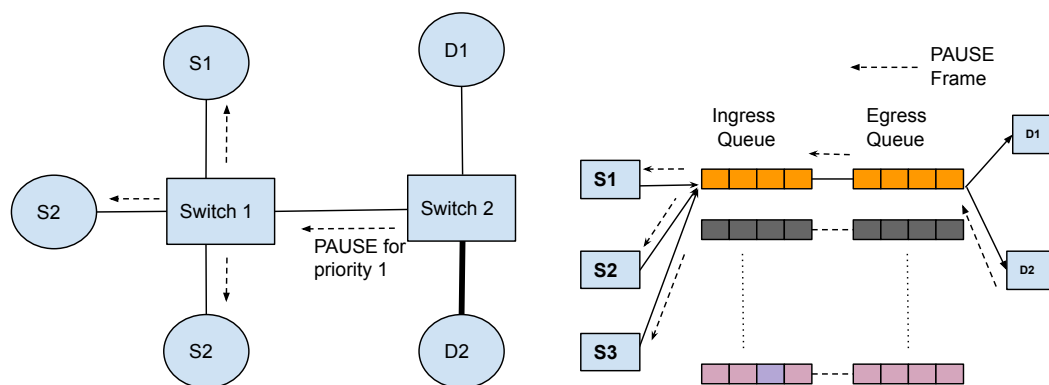


Figure 4.3: PFC Fairness issue, some flow may suffer (S1-D1) even if the congestion is not it's access path.

Before jumping into the limitation of TCP and RDMA transport variants

(DCTCP and DCQCN) in offering differentiated QoS, we discuss the limitations of the well established QoS features available in Ethernet (L2) and IP (L3) layers. These will bolster our claim of putting the QoS differentiation mechanism in transport layer (L4), rather than Ethernet and IP layer.

4.4.1 Priority Flow Control (PFC)

Priority Flow Control (PFC) provides ROCEv2 a lossless fabrics solution by preventing the flow packets to enter into the Ingress NIC/switch port, so that the buffer overflow can be minimized. When the ingress port of PFC enabled NIC or switches exceeds some threshold, a PAUSE message is sent to neighbour transmitter to stop sending any packets until any RESUME message is received. The main issue with PFC is, it operates on the port and priority basis. That can lead to unfairness to the same priority class traffic. For example, all traffic injected by source $S1 - S3$, belong to the same priority class. But the destination of the flows are different ($S1$ sends traffic to $D1$ and $S2 - S3$ sends traffic to $D2$). Now if there is a congestion at $D2$, then $S1$ should not get affected, but as PFC works in the port and priority level, $S1$ will encounter PAUSE in it's transmission. This effect is known as cascade effect of the PAUSE frame. It is obvious that some end point traffic control management is required alongside PFC in order to address this issue.

4.4.2 Enhanced Transmission Control (ETS)

Enhanced Transmission Control (ETS) enables traffic prioritization by leveraging the Differentiated Service Code Point (DSCP) bits in the IP header. ETS allows to assign relative weights to different traffic class priority queues and implements traffic prioritization by weighted round robin (WRR) arbitration. As ETS too has no notion of end point rate control, this may cause switch buffer overflow, and as a result packet drop occurs, which leads to packet re-transmission after timeout (e.g. TCP timeout and re-transmission). In a data center environment packet drop should be avoided, as these could

cause additional delay, so multiple QoS violation may occurs during the burst episode.

The main reason why network layer solutions like PFC and ETS can't guarantee lossless transmission is, none of them coordinate with the end point transmission layer protocol (TCP or ROCEv2), in order to modulate the flow rate during the congestion episode. For example, In case of PFC, when multiple same priority flows coexist and transmit through the same port, there is no way to differentiate the flows. Though in an attempt to mitigate this issue, congestion control mechanisms have been proposed for RoCE (e.g., DCQCN [17] and Timely [96]) which reduce the sending rate on detecting congestion, but are not enough to eradicate the need for PFC. And the same conclusion can be drawn in case of ETS. So to devise a proper lossless transmission protocol, we need to 1) coordinate with the transmission layer protocols in order to modulate the flow rate 2) design proactive congestion control mechanism.

4.5 Limitation of DCTCP and DCQCN:

4.5.1 Flow rate control

Both DCTCP and DCQCN control flow rate based on the congestion feedback (ECN and CNP).

DCTCP: DCTCP uses an exponentially smoothed version of the ECN feedback to modulate the congestion window (cwnd) for reducing the amount of traffic into the network. The underlying control parameter is the fraction of acknowledgements in a window that arrive with the ECE bit set. Consider I competing TCP connections (or flows). Let $f_i(n)$ denote this fraction for i th flow during its n th window. Then we can obtain an exponentially smoothed version of this quantity over successive windows, popularly denoted as α_i , as follows:

$$\alpha_i(n) = (1 - g)\alpha_i(n - 1) + gf_i(n - 1) \quad (4.1)$$

where $0 < g < 1$ is a smoothing constant (independent of the flow id i). DCTCP reduces the window per RTT in proportion to the latest estimate of α_i such that in the limiting case of $\alpha_i = 1$, the window is halved. That is, the window control follows the following rule:

$$W_i(n) = W_i(n-1) \left(1 - \frac{\alpha_i}{2}\right) \quad (4.2)$$

DCQCN:

For RDMA, DCQCN is considered as the popular transport protocol. DCQCN requires the PFC data center Ethernet feature, which is a L2 QoS capability to guarantee lossless transport for flows over a link. PFC cannot distinguish between traffic of same priority class, going to different destinations. Thus the congestion along a given destination path will generate a PAUSE frame, which causes blocking of all the incoming flow packets belonging to the same priority class, even if the destination is different and there is no congestion at the corresponding path (Discussed in section 4.4.1). DCQCN addresses these issues by leveraging both the PFC and ECN mechanism. However, the congestion feedback mechanism is different in DCQCN, since the underlying ROCEv2 utilizes connection less protocol i.e. UDP, so TCP like ACK feedback per frame transmission is not implemented. As a result, congestion feedback mechanism in ROCEv2 somehow needs to be request agnostic. Upon receiving a ECN marked packet, the receiver NIC sends ROCEv2 standardized congestion notification packet (CNP) to the sender NIC in N microsecond time interval, until the receiver NIC continues receiving ECN marked packets [17]. When the flow i sender gets a CNP, it reduces its current rate (RC_i) and updates the value of the rate reduction factor, α_i , like DCTCP, and remembers the current rate as target rate (RT_i) for later recovery, i.e.

$$RT_i(n) = RC_i(n-1) \quad (4.3)$$

$$RC_i(n) = RC_i(n-1) \left(1 - \alpha_i/2\right) \quad (4.4)$$

The value of $RT_i(n)$ is used for flow rate increasing during the non congestion episode. Rate increase has two main phases; fast recovery, and additive increase. During the fast recovery phase, the flow rate RC_i is rapidly increased towards the fixed target rate for F successive iterations, as follows:

$$RC_i(n) = \left\{ RT_i(n-1) + RC_i(n-1) \right\} / 2 \quad (4.5)$$

Fast recovery is followed by an additive increase, where the current flow rate slowly approaches to the target rate, and the target rate is increased by an additive constant R_{AI} , which is summarized as follows:

$$RT_i(n) = RT_i(n-1) + R_{AI} \quad (4.6)$$

$$RC_i(n) = \left\{ RT_i(n-1) + RC_i(n-1) \right\} / 2 \quad (4.7)$$

where R_{AI} is a additive constant (independent of the flow id i).

4.5.2 Limitations

Despite all the pros, both DCTCP and DCQCN were not designed for any service differentiation and thus effectively results in equal division of the available bandwidth amongst the existing flows during the congestion episodes. As a result, in the presence of congestion, a high priority flow may suffer more compared to other low priority flows if both of them react to the same congestion event similarly. That means, regardless of the QoS management in NVMe storage protocol (e.g. queue arbitration, block layer flow control etc), the end to end QoS objective remains unattainable.

Virtualization: Virtualization has become a common practise in data center environment. Virtualization might cause high priority flow to be placed in a more distant location, whereas low priority or no priority (assured service) flows closer to the same destination. So the delay encountered between successive ECN or CNP will be lower in case of low priority flows and grab the bandwidth more quickly. In TCP, this phenomenon is also known as *RTT Fairness*.

We propose a solution called QoS Aware TCP (QTCP) to address the QoS issue in the transport layer. In the next section, I talk in details about QTCP scheme.

4.6 QoS Aware TCP (QTCP)

We assume a fixed set of I QoS classes, with *a persistent TCP connection per class*. We assume that the applications are classified into one of these classes and thus each application uses a specific class throughout its lifetime, which is our notion of a “flow”. (It is possible that long running applications have different phases with distinct behavior, and thus the same TCP connection could potentially require different treatment across phases, but we do not consider such situations.)

4.6.1 QoS Specification

In general, each class has a specified tail latency objective and a minimum bandwidth objective, e.g., at least 200 Mb/sec with 90 percentile latency of 100 μ s. We consider two regimes of operation:

1. The bottleneck link has enough bandwidth to accommodate the average bandwidth demand of all the flows. Short-term congestion can occur in this case and can seriously affect the latencies achieved by various classes. Thus the objective of congestion control is entirely to satisfy the latency requirements that we assume are specified in terms of tail latencies.

2. The bottleneck link is lacking capacity to satisfy the minimum bandwidth of some of the flows. In this case, it is necessary to ensure that various classes get a predefined fraction of the bottleneck link capacity. We assume that the total available bandwidth of the bottleneck link is known here.

We further *assume that all tail latencies are specified using the same percentile value, e.g., 90 percentile*. If originally the latency is specified differently (e.g., 99 percentile), we then need some way to estimate the

corresponding 90 percentile value. For example, if the mean and variance of the latency distribution is known, we can use Chebychev inequality ?? to estimate the tail latency. That is, for a random variable X with given expected value $E[X]$ and standard deviation σ_X , we have:

$$Pr[|X - E[X]| \geq \delta\sigma_X] \leq 1/\delta^2 \text{ for any } \delta > 1 \quad (4.8)$$

The reason for assuming the same percentile is that it enables us to control the window size based directly on the ratios of achieved and target latency values.

The BW based control is a little more complex. The problem is that the "desired bandwidths" must be limited if the total offered traffic exceeds the bottleneck link capacity C . Thus, there are two situations to consider for each class i :

- No congestion: Desired BW = Offered load of the class i
- Congestion: Desired BW = $C \times$ Desired BW ratio for class i

4.6.2 Quality Factor and Window Flow Control

We now define a measure called *quality factor*, and denoted as Q_i for class i . Let L_{ia} and L_{it} denote, respectively, the actual and target tail latencies for class i . We express Q_i as a ratio of the two, with actual latency smoothed over time. That is,

$$Q_i = L_{it}/L'_{ia} \text{ where } L'_{ia} = (1 - \gamma)L_{ia} + \gamma L'_{ia}, i = 1..K \quad (4.9)$$

where γ is the smoothing factor. Q_i is a dimensionless number, ranging from 0 to some maximum value limited by the admission control. If $Q_i > 1$, class i has a slack (i.e., its window can be squeezed), and if $Q_i < 1$, then class i has deficit, and its window needs to be increased. Since we assume that each class uses a separate TCP connection, the window for each class is controlled

independently based on its Q factor. A similar Q_i can be defined for bandwidth centric control, i.e.,

$$Q_i = \lambda'_{ia}/\lambda_{it} \text{ where } \lambda'_{ia} = (1 - \gamma)\lambda_{ia} + \gamma\lambda'_{ia}, i = 1..K \quad (4.10)$$

where we have reversed the ratio, to keep the same sense for Q_i factor ($Q_i > 1$ means that we have slack, and $Q_i < 1$ means that we have deficit).

In addition to the quality factor, we also need to make use of the congestion feedback returned by the standard ECN mechanism. The window modulation for different QTCP flows is done based on both the value of α_i (probability of congestion a.k.a. percentage of packets are ECN marked) and quality factor metric Q_i . The overall window modulation mechanism for a single flow i :

$$W_i(n) = \begin{cases} W_i(n) + 1, & \text{No ECN} \\ W_i(n)(1 - \frac{\alpha_i}{2}), & \alpha_i \geq 0 \text{ and } Q_i > 1 \\ W_i(n)(1 - \frac{\alpha_i}{2})Q_i, & \alpha_i \geq 0 \text{ and } Q_i \leq 1 \end{cases} \quad (4.11)$$

The above scheme works as follows. When there is no congestion indication, then congestion window for each flows would increase by 1 per RTT. In case there is a congestion indication (marked ECN ACKs) in the previous RTT, then the window will be updated in proportion to α and Q_i . As discussed earlier, $Q_i > 1$ means, the flows has not satisfied the QoS requirement yet, so the modulation would only be based on the α . When $Q_i \leq 1$, the flow has already met the QoS demand, so it is okay to back off from the assigned bandwidth resources to make room for others.

4.7 Analytical Modeling of QTCP

4.7.1 A Simple Operational Model

The essential aspects of QTCP can be captured via a *discrete time model* (DTM). Our proposed DTM model considers the change in flow rate from one update interval to the next. The behavior may also be approximated via a

fluid flow model (FFM) as in the analysis of DCTCP in [15]. Note that unlike DCTCP, where only one flow can be analyzed in isolation, our model must analyze a coupled system of equations involving all classes. Both models have their strengths and weaknesses. The main issue with DTM is that it considers the behavior of TCP only at certain discrete points and thus cannot model the intra-update state. On the other hand, the main issue with FFM is that it incorrectly assumes infinitesimal control of state and because of that cannot easily handle the notion of state during the update. In particular, the FFM in [15] uses an average value of RTT to refer to the last RTT cycle. We focus on DTM only in our work.

For the DTM, we continue to use n as the current “time-slot” or current update duration. Consider $i = 1..I$ active connections, each belonging to a distinct class, so the notion of class and flow can be used interchangeably. Let C denotes the capacity of the bottleneck link used by these classes with $C_i(n)$ as the share of class i at slot n , with $\sum_{i=1}^I C_i = C$. Let $R_i(n)$ denote the round-trip time (RTT), and $q_{aj}^{(i)}(n)$ denote the queue length of class j at the *bottleneck egress port of the switch as seen by an arriving class i packet*. Furthermore, let $p_i(n)$ denote the event that the switch queue is already at or beyond the threshold K when a class i packet arrives, and thus this packet has its CE bit set. Note that in the current window, the relevant event is from the last window. That is,

$$e_i(n) = I_{\sum_{j=1}^I q_{aj}^{(i)}(n-1) \geq K} \quad (4.12)$$

In general, each arriving class i may see a different distribution of packets in the queue; however, since we assume that the switch uniformly marks packet of any class that sees a “full” queue, the dependence of $q_{aj}^{(i)}(n)$ is likely to be weak, if any. Therefore, we henceforth assume that such a dependence does not exist, and denote the queue full event as simply $e(n)$. However, for a DTM, we need not observe the individual events but rather the probability of the queue being full, henceforth denoted as $p(n)$. We can estimate this as follows:

$$p(n) = \begin{cases} 1 - \frac{(k-1)}{B(n-1)}, & \text{if } B(n-1) \geq K \\ 0, & \text{if } B(n-1) < K \end{cases} \quad (4.13)$$

where $B(n-1) = \sum_{i=1}^I q_{ai}(n-1)$. The overall latency $L_{ai}(n)$ observed by an arriving class i packet is given by $L_{ai}(n) = d_i + q_{ai}(n)/C_i(n)$ where d_i is the baseline delay independent of queuing (including send/receive processing delay, link propagation delay, switch processing delay, and transmission time of one arriving request). We assume that the feedback packets (ACKs) do not face any significant queuing delay, and thus $R_i(n) = d'_i$ where d'_i is the backwards delay. For simplicity we will assume that $d'_i = d_i$ for all i .

We assume that suitable admission control is in place so that the *average* total offered traffic to the bottleneck link is always strictly less than the link bandwidth C . In other words, we assume that the packets cannot build up at the transmit nodes indefinitely. Thus, the congestion is a result of the burstiness in individual class traffic, including the overlaps in high traffic periods of various classes such that the link capacity is temporarily exceeded but no packet is ever dropped either in the switches or at the transmitter. That is, the long-term throughput of the system equals the offered load.

Throughput Ratios: Class i is targeted to get the given BW ratio of r_i relative to class 1 (i.e., $r_1 = 1$). That is, $C_i = C.r_i/\sum_i r_i$ and is independent of slot. Since no packets are lost, the actual throughput can be estimated from the number of packets transmitted $W_i(t - \bar{R}_i)$ in the last update interval $R_i(t - \bar{R}_i)$, i.e., $\lambda_i(n) = W_i(t - \bar{R}_i)/R_i(t - \bar{R}_i)$. Therefore, $Q_i(n) = \lambda_i(n)/C_i$.

Queuing Latency: We assume that the classes are ordered according to an importance score, with class 1 being the most important. The queuing latency target L_{it} for these classes must be chosen sufficiently large to be realizable, particularly since the switch is assumed to use FCFS (first come first serve) scheduling for all packets. Though One could use multi-class open-system queuing formulae [97] to estimate range of values to use. The actual congestion $L_i(n) = d + q_{ai}(t - \bar{R}_i)/C_i(t - \bar{R}_i)$ where $C_i(n) = W_i(n)/R_i(n)$.

Therefore, $Q_i(n) = L_i(n)/L_{it}$.

We could then write the equations for all quantities. In the following we assume that the bandwidth, transferred data and throughput are in the units of packets rather than bytes. Based on the discussion above, we first restate the basic quantities below.

$$C_i(n) = \begin{cases} C \cdot r_i / \sum_i r_i, & \text{Throughput Control} \\ \frac{W_i(n-1)}{R_i(n-1)} & \text{Latency Control} \end{cases} \quad (4.14)$$

$$Q_i(n) = \begin{cases} \frac{C_i(n)R_i(n-1)}{W_i(n-1)} & \text{Throughput Control} \\ \frac{d_i + q_{ai}(n-1)/C_i(n-1)}{L_{it}} & \text{Latency Control} \end{cases} \quad (4.15)$$

$$\alpha_i(n) = \alpha_i(n-1) + \gamma[p(n) - \alpha_i(n-1)] \quad (4.16)$$

The order of calculation is as follows: We first estimate $p(n)$, i.e., whether ECN was received in the last update event, based on the queue length in the previous slot ($q_{ai}(n-1)$). This, in turn, is used to compute the fraction of BW given to each flow in the current slot, $C_i(n)$, and from there the quality factor $Q_i(n)$. We next update α , which in turn provides all the parameters required to update the packet transferred $W_i(n)$ and the update interval ($R_i(n)$) for the current slot. This is then used to estimate $q_{ai}(n)$, the queue length for the current slot, so that the temporal evolution can continue. Thus,

$$W_i(n) = \begin{cases} W_i + 1, & \alpha_i \leq \epsilon \\ W_i[1 - \frac{\alpha_i(n)}{2}], & \alpha_i > \epsilon \ \& \ Q_i(n) > 1 \\ W_i[1 - \frac{\alpha_i(n)}{2}]Q_i(n), & \alpha_i > \epsilon \ \& \ Q_i(n) < 1 \end{cases} \quad (4.17)$$

$$R_i(n) = 2d_i + B/C; q_{ai}(n) = \max[0, q_{ai} + W_i(n) - C_i R_i(n)] \quad (4.18)$$

where we have used $C(n-1)$ in the last equation, since the known drainage rate is $C(n-1)$ during the n th slot.

One could seek the ‘‘steady state’’ from these equations by considering the case where the W_i and R_i do not change from slot to slot. However, it is

clear from equation (4.17) that this system does not have any fixed point. The equations above assume that there are always enough packets available at the transmitter so that it can fill whatever the W_i is in each slot. We can extend the model further by including a packet generation process and keeping track of untransmitted packets for each class i , henceforth denoted as $U_i(n)$. The actual window size for class i , henceforth denoted as $W'_i(n)$, is then the minimum of the computed window size $W_i(n)$ (from $W'(n-1)$ using equation like(4.17)). That is,

$$M = \inf_{(\sum_{m=1}^{M'} G_i^{(m)}) > R(n-1)} (M') \quad (4.19)$$

$$U_i(n) = U_i(n-1) - W'_i(n-1) + M - 1 \quad (4.20)$$

$$W'(n) = \min[W(n), U_i(n)] \quad (4.21)$$

4.7.2 Comparison of Model and Simulation

We validate the analytical modeling of QTCP with our ns3 implementation in Fig. 4.4 with 3 applications. We set the bottleneck bandwidth C at 10 Gbps; the applications are injecting traffic at a rate of 3, 6 and 9 Gbps respectively. We assume the threshold K to be 140 ($K \sim 0.17Cd$, where C = Bottleneck capacity, d = propagation delay), which is also used in the DCTCP paper [15].

From Fig. 4.4 we can observe that our analytical model shows similar behavior in terms of throughput of the individual applications, as compared to the ns3 simulations. Thus this validates that our analytical model closely approximates the behavior that is obtained from the simulations.

4.7.3 Convergence and Stability Analysis

We next conduct the convergence and stability analysis of the developed analytical model in presence of 3 applications.

Fig. 4.5 shows the variation of Q_i with RTT slots, where the RTT and γ are assumed to be $248\mu s$ and 0.02 respectively. From this figure we can observe

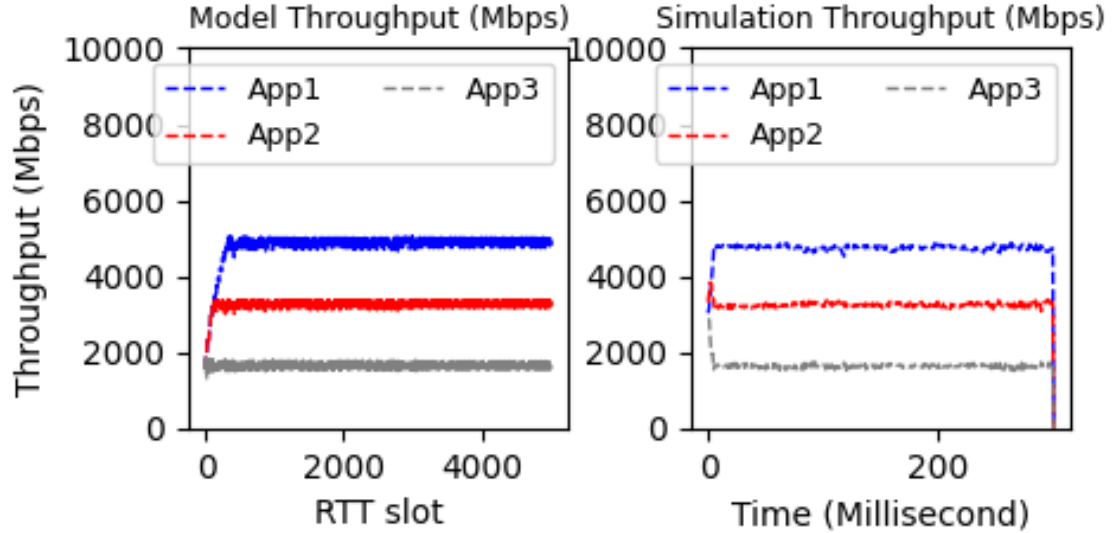


Figure 4.4: Comparison of throughput between model and simulation

that the Q_i 's of all three applications converge to approximately 1 within 100 RTT slots. Because of the target throughput based window modulation, the actual throughput of the applications reaches close to the target throughput, which makes the quality factors close to unity.

Fig. 4.6 shows the effect of different RTT values on the convergence time. As expected the convergence time increases with the increase in RTT values. In a data center environment, the RTT is relatively small (~ 150 - $200 \mu s$) [31]; thus, the convergence time will be fairly quick.

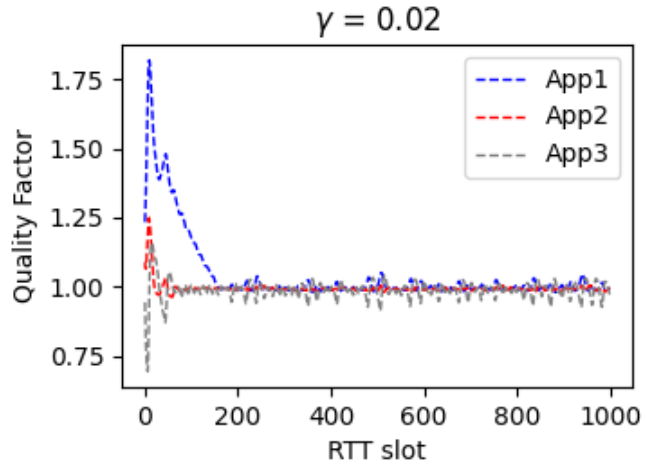


Figure 4.5: Variation of quality factor per RTT that the quality factor of the applications will converge to 1.

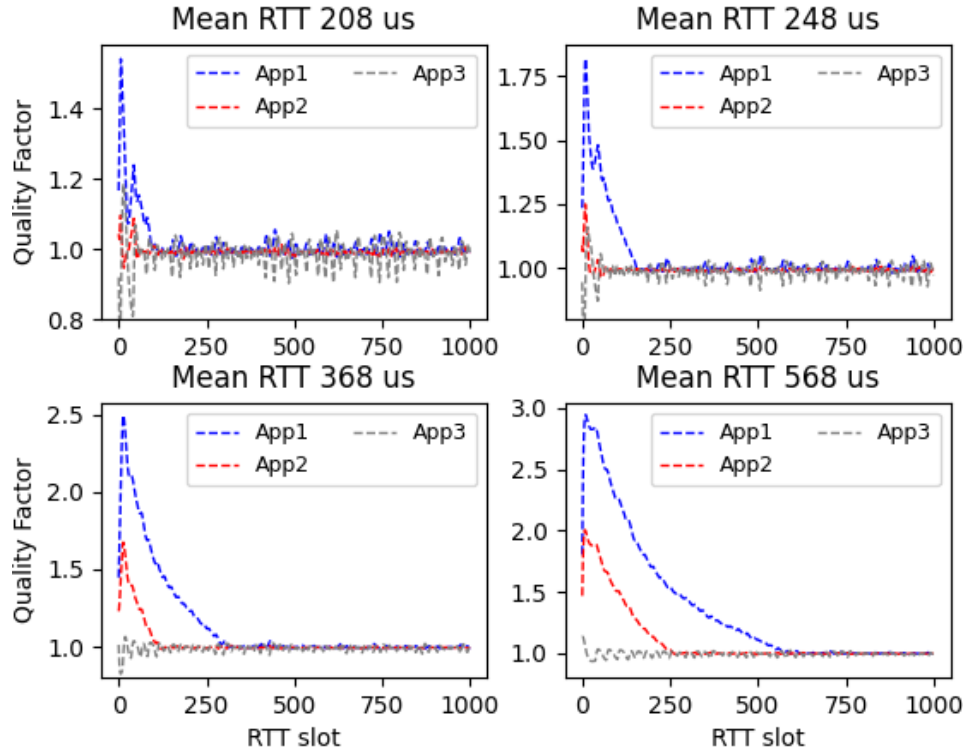


Figure 4.6: QTCP convergence with different RTT

4.8 Performance Evaluation of QTCP

We comprehensively evaluate the QTCP mechanism using the popular ns3 network simulation package. For this, we started with the detailed DCTCP implementation in ns3 (which closely follows the RFC 8259) and also implemented the proposed QTCP mechanism. In spite of existence of numerous data center network topologies, most data centers still use the fat-tree topology as illustrated in Fig. 2.4. Note that regardless of the size of network, the fat-tree topology always has 3 levels of switches (edge, aggregation, and core), which means that no request/response ever travels more than 6 hops. Thus the network in Fig. 2.4 is adequate for exhibiting QTCP performance, even though it is quite small. We used 10 Gb/s links for simulation efficiency, but similar results apply to the more contemporary 100 Gb/s links.

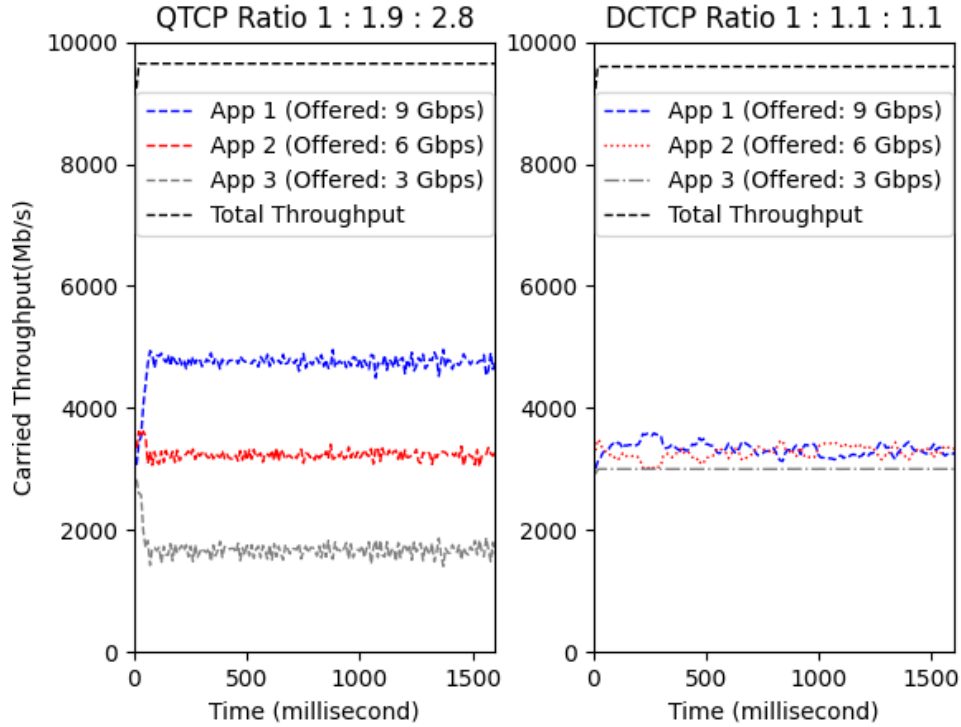


Figure 4.7: Comparison of bandwidth distribution between QTCP (left) and DCTCP (right). The bottleneck bandwidth is kept as 10 Gbps. The attained throughput ratio of QTCP is approximately equal to the target throughput ratio, 1.0 : 2.0: 3.0

4.8.1 Comparison with DCTCP Throughput

Fig. 4.7 shows the comparison between QTCP and DCTCP in presence of 3 applications carrying load of 3, 6, and 9 Gbps respectively; thus the accumulated offered load (i.e. 18 Gbps) exceeds the bottleneck link capacity of 10 Gbps. The ratio of offered load is 1.0: 2.0: 3.0. In case of DCTCP, we can observe equal sharing of the bandwidth between the 3 applications in Fig. 4.7(b), and the carried throughput ratio is 1: 1.1: 1.1. In case of QTCP, the carried throughput ratio is almost exactly equal to that of the target ratio. Thus QTCP can achieve the desired QoS objective of allocating the bottleneck link bandwidth in the desired proportions to different classes. Also notice that, the overall throughput in case of QTCP is almost equal to DCTCP (9.63 Gbps compared to 9.57 Gbps in DCTCP).

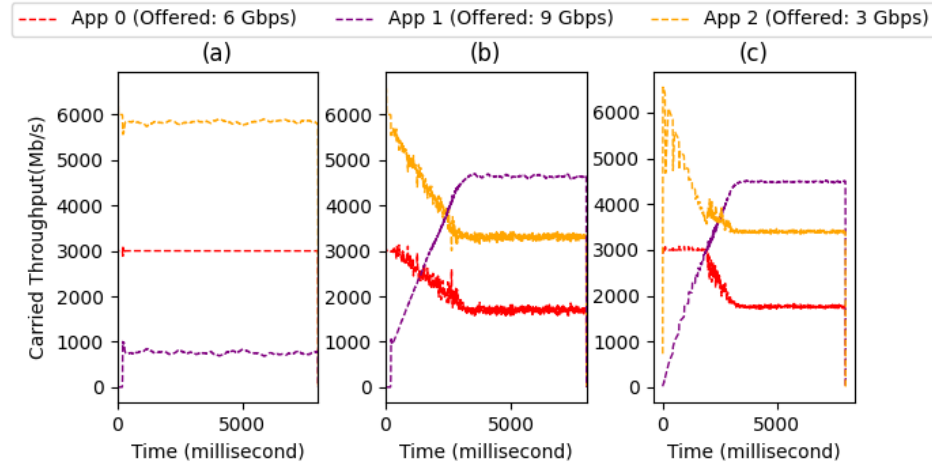


Figure 4.8: Comparison of bandwidth distribution between (a) DCTCP and (b)-(c) QTCP for different RTTs. In (a) and (b), the RTT of 9Gbps application is higher as compared to other comparatively low priority applications. In (c), all the three applications are put into different fat tree PoDs. In both the cases, bandwidth distribution using QTCP is approximately equal to the target bandwidth ratio.

4.8.2 RTT Fairness

We define RTT fairness as the extent to which we can achieve the target throughput ratios under differing RTTs of the flows. For this, we generate inter-pod and intra-pod flows with different target ratios. Fig. 4.8 shows the results for both DCTCP and QTCP. The average RTT of high QoS Application (9 Gbps) is approximately $1800 \mu\text{s}$, whereas the low QoS applications (i.e. 3 Gbps and 6 Gbps) have a RTT of $350 \mu\text{s}$. It is clear that DCTCP has a bias against flows with longer RTTs, as flows with shorter RTTs grab bandwidth more quickly.

However, QTCP is not affected; it still offers the desired throughput ratios and the same overall throughput as DCTCP (9.6 Gbps). In QTCP, although the flows with shorter RTTs grab window more quickly, the quality factor forces the low QoS flows to make room for the high QoS flows, regardless of the RTT. Fig. 4.8 compares the results when a congestion occurs in several places, rather than only at the TOR (Top of the Rack) layer. 4.8(a) shows the result when the congestion is only at the TOR switch, whereas 4.8(b) shows

the case of congestion in both aggregation and TOR layers. In both the cases, the bandwidth distribution ratio is close to the target throughput ratio.

4.8.3 DCTCP Friendliness

The estimation of available bandwidth may be inadequate in a dynamic environment where new non-QTCP applications may start or stop at any time. Therefore, we also consider an alternate mechanism where the QTCP itself continuously adjusts the bottleneck bandwidth by monitoring the impact of any interfering traffic. For this, we assume that the bottleneck bandwidth (λ) is known initially (given or estimated by explicit methods like [30]). If an interfering flow alters this value, each QTCP flow estimates it as shown in Algorithm 1. Here $target_i$ is the QoS requirement of flow i when there is congestion in the network, and $actual_i$ is the estimated throughput till that point. Since quality factor Q_i quickly converges to close to 1 (section 4.7.3), its perturbation by more than σ is considered as a signal of a new interference or disappearance of the interference. We then change the target bandwidth $target_i$ by the amount called “factor” and change the window size accordingly. Algorithm 1 describes the modified QTCP scheme which triggers in presence of interfering flows. $ratio_i$ defines the fraction of bottleneck bandwidth assigned), Q_i defines the current quality factor measured, W_i is the current congestion window size for flow i . λ is the bottleneck bandwidth measured/estimated. Also for our experiment we choose σ as 5% (i.e. 0.05).

Fig. 4.9 (a) shows the DCTCP friendliness results. When the DCTCP flow leaves (after about 1500 milliseconds of simulation time), the QTCP flows manage to grab the bandwidth resources according to the QoS specified ratio (1 : 2 : 3). In Fig. 4.9(b) we simulate the scenario where the DCTCP flow enters and leaves during the simulation; in this scenario also we observe that QTCP adjusts the remaining bandwidth among the active flows. We simulate with multiple DCTCP flows and get the expected result as 4.9(a) and 4.9(b) in all the cases. Another solution to ensure the DCTCP friendliness could

Algorithm 1 Modified QTCP Scheme

```

 $W_i = W_i(1 - \alpha/2)$ 
factor = 0
if ( $Q_i - 1.0 > \sigma$ ) then
    factor =  $-\sigma$ 
else if ( $Q_i - 1.0 < \sigma$ ) then
    factor =  $\sigma$ 
end if
 $\lambda = (\lambda + \text{factor}) \times \text{target}_i$ 
 $\text{target}_i = \text{ratio}_i \times \lambda$ 
 $Q_i = \text{target}_i / \text{actual}_i$ 
if ( $Q_i - 1.0 > \sigma$ ) then
     $W_i = W_i \times Q_i$ 
end if

```

be to reserve the bandwidth explicitly for the interfering flows as suggested in [98]. However, this will result in network under-utilization when there is no interfering DCTCP flow.

4.8.4 Latency Comparison with DCTCP and D²TCP

We now consider the case of latency sensitive traffic, where the QoS is defined in terms of target latency. For the experiments, we categorized applications into four classes; 3 out of 4 classes have latency requirements of 5366, 6604, 7832 μs respectively, whereas class 4 is best effort and has no QoS requirements. The mean transfer size we choose is 2MB. The flow arrival rate follows Poisson distribution and average utilization is 80%.

Fig. 4.10 shows the comparison between QTCP, DCTCP and D²TCP for latency sensitive applications. We simulate both the normal and stress situations to show the effectiveness of our scheme. Fig. 4.10(a) depicts the normal situation where the high priority flows (i.e. applications 1 and 2) generate packets at a frequency lower than that of others; in our simulations

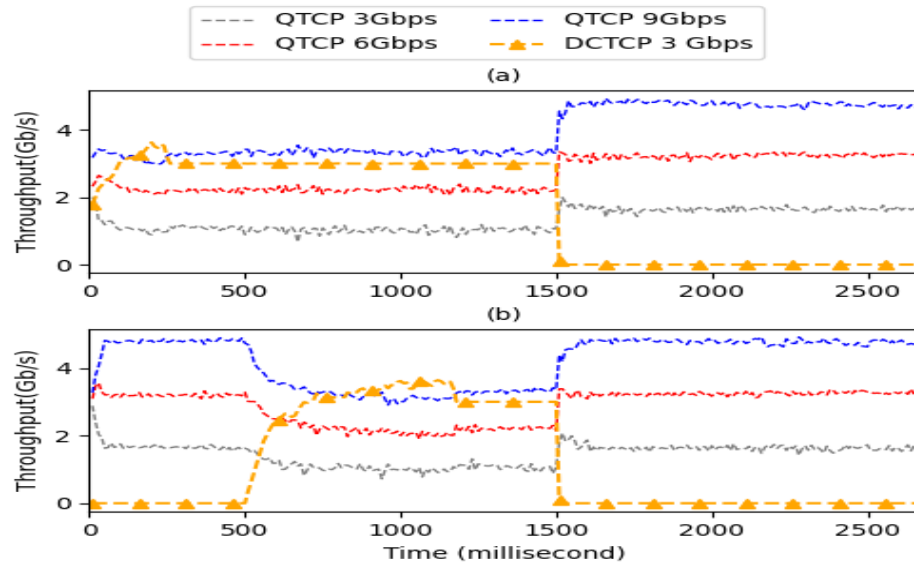


Figure 4.9: Illustration of DCTCP friendliness of QTCP

their overall generated traffic is 10% and 20% respectively. In case of QTCP, almost all the applications are able to meet their target latency, whereas in DCTCP $\sim 5\text{-}8\%$ packets miss their deadlines. In 4.10(b) we simulate a more challenging situation, where each class contributes to 25% of the overall load. As compared to DCTCP, in case of QTCP the fraction of traffic with missed target latency reduces from $\sim 30\text{-}65\%$ to $\sim 15\text{-}18\%$. Notice that in Fig. 4.10, we do not report the latency statistics for application 4, as it is of best effort type.

Fig. 4.10 also compares the performance of QTCP with D^2 TCP. In the normal scenario, D^2 TCP does not miss any deadlines. However, in a stress situation QTCP yields fewer deadline misses. As compared to D^2 TCP, QTCP reduces the percentage of missed deadlines from $\sim 35\%$ to $\sim 18\%$ for application 1, whereas for the other applications the performance of both the schemes are almost identical.

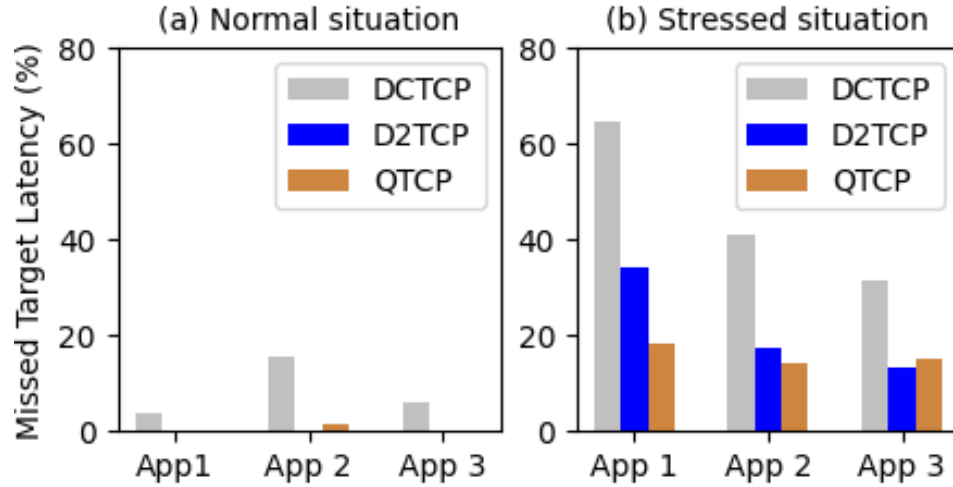


Figure 4.10: Comparison of QTCP, DCTCP and D²TCP

4.9 QoS Differentiation in DCQCN (QRDMA)

Similar to DCTCP, in case of DCQCN too the available bandwidth is distributed equally among all the existing flows. Thus, to implement QoS differentiation in RDMA, we develop the scheme QRDMA which essentially uses the same mechanism as developed for QTCP in section 4.7.

In particular, we control the target rates to provide the desired ratios for different classes. Fig. 4.11 shows the comparison between DCQCN and QRDMA with three applications; the simulation environment is exactly same as that of section 4.8. The attained throughput ratio of the three applications is 1.0: 1.8: 2.6, which is also approximately equal to the target throughput ratio (1.0: 2.0 :3.0). This shows that the QRDMA scheme also achieves expected service differentiation similar to QTCP.

Fig. 4.12 compares the performance of DCQCN and QRDMA with latency sensitive applications in a stressed scenario as in Fig. 4.10(b). From Fig. 4.12 we can observe that as compared to DCQCN, QRDMA demonstrates fewer deadline misses. In fact, as compared to DCQCN, QRDMA reduces the deadline misses by $\sim 43-80\%$. However, while comparing Fig. 4.10(b) and Fig. 4.12, we can observe that QRDMA yields higher deadline misses as compared to QTCP. This is mainly because the implementation of DCQCN

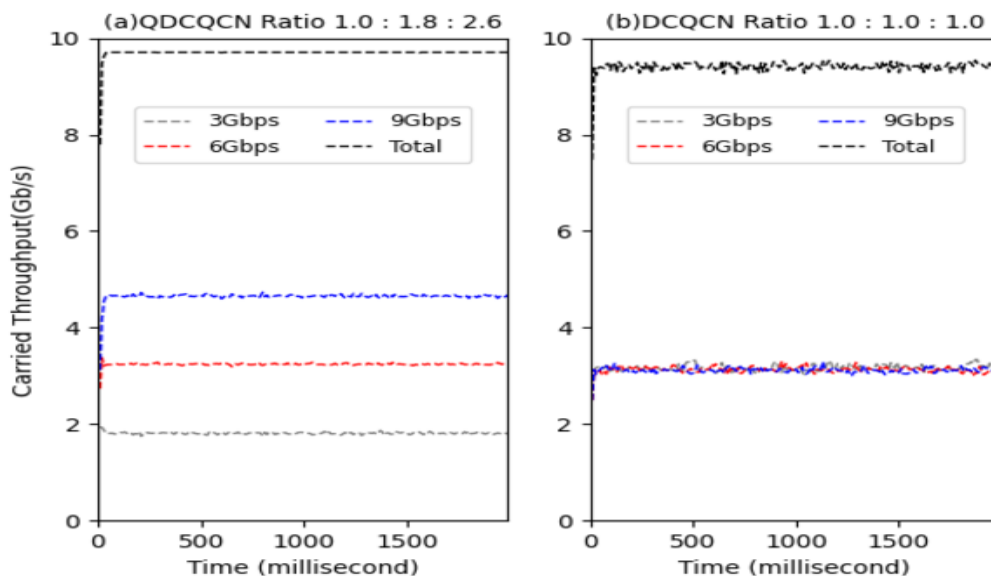


Figure 4.11: Bandwidth Distribution in QRDMA (left) and DCQCN (right). Bottleneck bandwidth is 10 Gbps and attained carried throughput ratio of QRDMA is approximately equal to the target throughput ratio, 1.0 : 2.0: 3.0.

(on which QRDMA is implemented) requires careful parameter tuning for the best performance. In our implementation of QRDMA, we have used the default parameter settings as suggested in [17]. Parameter tuning for QRDMA is beyond the scope of this work and is left for future work.

4.9.1 Effect of Different Incast Degree

Single I/O request from a data center application might comprise of response flows from multiple storage servers. These multiple parallel flows can also be generated from complex interplay between data center applications. For example, in case of big data applications like Hadoop, or Spark, during the shuffle/reduce phase, several concurrent

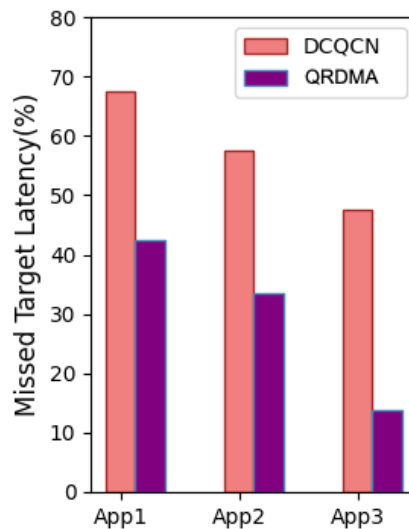


Figure 4.12: Deadline misses in case of DCQCN and QRDMA

flows are generated. These concurrent independent flows can cause buffer overflow all of a sudden, and thus decrease application level throughput far below the link bandwidth. This phenomenon is known as *incast problem*. To avoid long latencies due to the buffer-bloat problem, the idea in DCTCP/DCQCN is to provision rather small packet buffers in the data center switches and control the traffic injection aggressively so the queues do not build up much beyond the point where ECN is triggered. However a switch queue to which a storage server is connected could still be overwhelmed due to incast problem and result in packet losses due to physical buffers being overrun. In this section, we discuss how a QRDMA works for different incast degree.

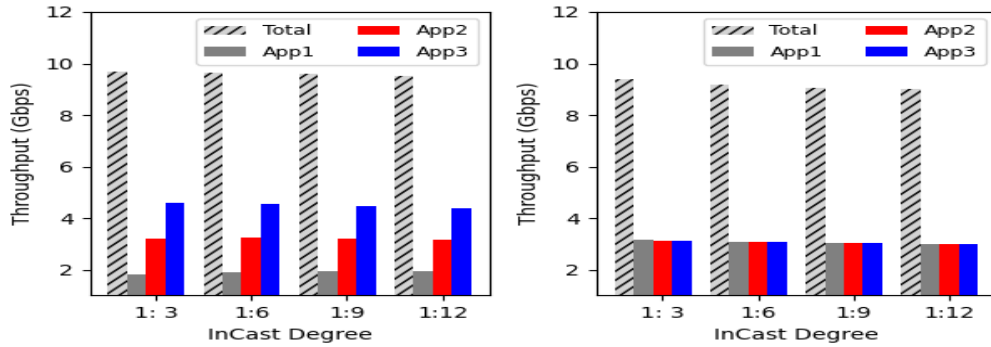


Figure 4.13: Comparison of bandwidth distribution between QRDMA (left) and DCQCN (right) for different incast degrees.

Fig. 4.13 shows the result for different incast degrees. In case of QRDMA, we observe a slight improvement ($\sim 3\%$) in terms of the overall throughput. Since the endpoint has a notion of quality factor in QRDMA, it can modulate the data rate in a more controlled way, as compared to DCQCN. We also notice that irrespective of the incast degree, QRDMA achieves QoS differentiation for different applications.

The incast issue can happen with TCP as well, and the experiments (not reported here) show we achieve similar QoS differentiation with QTCP as compared to DCTCP.

4.10 Related Work

Although the main goal of conventional transport layer solutions (either TCP or ROCEv2) is fairness (equal sharing of bottleneck bandwidth) amongst different flows, some variants address the differentiated treatment. To the best of our knowledge, no work addresses the QoS issue in data center RDMA transport. But unlike QoS aware RDMA, several works have addressed the issue of QoS aware differentiated flow control in the TCP context. Homa [99], L²DCT [100], D²TCP [16], PDQ [101], D³ [102] consider QoS in terms of individual flow completion time (i.e. deadline). Homa addresses head-of-the-line (HoL) blocking issue posed by TCP streams. They leverage in-network queue priority to provide low latency QoS to the small messages (99 percentile latency of 10 μ s). L²DCT and D²TCP modulate TCP congestion window for different flows based on the QoS parameter provided. One of the key issue with these schemes is that the administrators need to have prior knowledge about the network delay and RTT in order to set the QoS parameters, whereas in the case of QTCP we just need to specify the relative bandwidth ratio of different flows. PDQ proposes distributed scheduling algorithm, where the switches coordinate among themselves to schedule the high priority flow earlier (i.e. flow with critical deadline). It requires specialized switches and extensions to the protocol header to convey the QoS hints. D³ is another deadline-aware TCP variant, however, D³[102] requires specialized switches and is not feasible for a ubiquitous solution. D³ also requires centralized control, so scalability might get affected badly by the communication overhead. Learning based approaches [103, 104] for optimal tcp performance can enhance the TCP performance, but what lacks in these solutions is: offering QoS differentiation and based on Loss based solutions. Besides that, using learning approach in the data center context is impractical, as data center comprises multiple applications, and it would be highly inaccurate to learn any workload pattern by any means of machine learning algorithm and apply corresponding optimal parameter setup.

4.11 Limitations

Though our solution works for TCP and RDMA in isolation, it is very common data center network consists of mixture of protocol specific traffics. For example, the ECN marking threshold paramter K is different in case of DCTCP and DCQCN. Also DCTCP requires both minimum and maximum probaility (P_{MIN} and P_{MAX}) for ECN marking to be equal 1.0, whereas DCQCN demands P_{MIN} and P_{MAX} to be different for the optimal performance.

Another limitation of our scheme is in offering ultra low latency QoS to the small transfer memory traffic. Though in the evaluation for latency qos sensitive applications, we assume transfer size of 2 MB, which is standard block (a.k.a chunk) size for big data applications, but data center network is also utilized for small transfer memory traffic, which can stuck behind the long transfer traffic packets (HOL phenomemon), and can drop the overall performance significantly. In the next chapter, we talk about the dynamics for mixture of different types of traffic and the enhancement in our scheme to overcome the issues described here.

CHAPTER 5

CASE FOR IN-NETWORK QOS PROVISIONING FOR MIXED MEMORY AND STORAGE TRAFFIC IN NVME OVER FABRICS

As discussed in the previous chapter, our scheme still lacks providing QoS for small transfer memory traffic and mixture of storage traffic (mainly TCP and RDMA), In this chapter, we discuss elaborately the limitations and work around solutions, that gurantee the QoS differentiation in mixed memory and storage traffic environment.

5.1 Background and Motivation

5.1.1 Persistent Memory as Data Store

Traditionally data management system uses solid state drive or magnetic disk to store data. The main reason is data persistence, performance, capacity

and cost. Most system uses DRAM as a cache, to enhance the performance, but DRAM can't be used as data store media due to its volatile nature, limited capacity and high cost. Persistent memory reduces this gap by offering non-volatility, higher performance and capacity, makes it an alternative for the data store solution. For example, Amazon ElastiCache ¹ and Oracle Persistent Memory Database leverage persistent memory (pmem) technology for data storage and database solutions respectively. pmem can be used both with storage and memory semantics. With storage semantics, the access sizes will typically be large (e.g., 4KB), and thread requesting pmem access will be switched out until the response is received. With a memory model, the access sizes are typically 1-2 cachelines, and the CPU will stall until the data is received. Although pmem as a data store can provide high performance as compared to a SSD when accessed locally, this may not be the case when accessed remotely due to network latency [105, 106]. In fact, since the network carries both storage and pmem traffic, without a differentiation, the network latency could dominate the pmem access latency even with the storage semantics. With memory semantics, remote pmem access is unlikely to be useful unless the network latency can be controlled strictly.

Persistent Memory over Fabrics: Persistent Memory over Fabrics (PMoF) [89, 107] is the new disaggregated storage trend which targets to use the remote persistent memory efficiently. The main benefit of using PMoF are mainly 1) High availability - storing data in both local and remote PM to avoid single point of failure and 2) Shared memory - multiple applications can share the data and coordinate operations via the remote memory pool. PMoF mainly responsible for the CPU load and store operations, whose transfer size is usually very tiny in size (2-4 cacheline).

In the conventional data center network this small transfer might experience a huge latency, which causes huge performance drop, as CPU stalls during the whole time.

¹<https://aws.amazon.com/elasticache/>

PMoF uses RDMA, and the underlying transport layer protocol could be TCP (iWARP) or UDP (ROCEv2), though ROCEv2 is recommended and generally used considering the low latency it offers compared to hardware based TCP solution iWARP. In our work, we assume PMoF using ROCEv2 as a transport. One of the main goal of our work is to guarantee QoS for the PM traffic when it coexists with other storage traffic.

5.1.2 Latency and Throughput Tradeoff

Lets assume a simple switch/router with single ingress and egress port, and has a buffer of size n packets, If any incoming packet P arrives in the switch and there are n packets buffered in the switch, then P will be dropped and retransmission occurs. Now there are 2 applications using the switch as a route on their way to the destination. Application 1 is high throughput QoS application, which usually performs online analytics and transfer size is usually large, and Application 2 is latency sensitive application, which requests are usually tiny, (1-2 packets), which performs inter process communication kind of stuff. Now, application 1 and 2 expect the following behavior from the buffered system:

- Any packet/stream belongs to Application 2 expects the switch buffer to be empty when arrived, otherwise this application will encounter head of the line blocking (HOL) either by Application 2 packets, or packets generate by itself earlier and have not served yet, or both. So to guarantee low latency QoS, the buffer utilization needs to be way below 100%, and as a result the overall system throughput should be low.
- Application 1 expects the switch buffer to filled (n packets) all the time, since there is a propagation delay associated with each transfer of packet/stream, so the buffered system will be underutilized if Application 1 sends single packet or frame at a time.

So, as a result, in order to satisfy QoS for both the throughput and latency

sensitive applications, the appropriate action should be maintaining the queue depth of the buffer as such, the objective of both Application 1 and 2, can be guaranteed.

5.2 Limitations in Existing Solutions

5.2.1 Variants to Address QoS Differentiation

In this section, we talk about D²TCP [16], L²DCT [100], QTCP and QRDMA. Then we discuss how QTCP and QRDMA outperform others in terms of QoS differentiation. This gives the justification why QTCP and QRDMA are used besides in network queuing. At the same time, we discuss though qos differentiation can be achieved for long living throughput sensitive flows, no guarantee for the small transfer PM traffic. And the reasons are described in the next section:

Connection-less Protocol

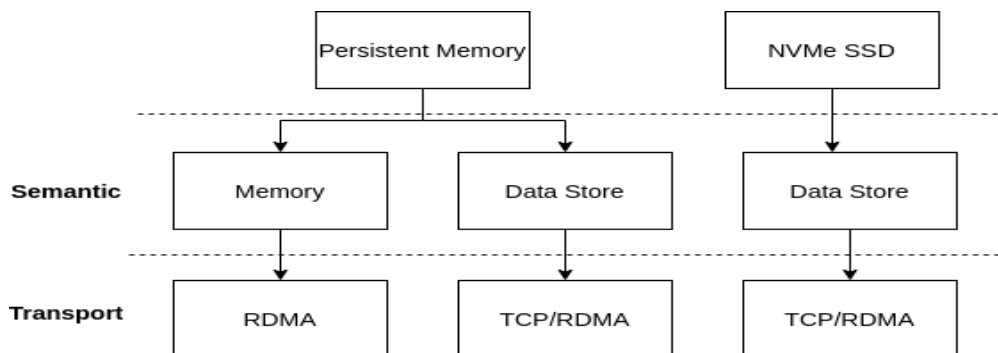


Figure 5.1: Semantics and transport type for different applications using remote persistent memory or NVMe SSD.

Small transfer PM traffic can't afford the additional delay introduced by the TCP protocol overhead (slow start, ACKs etc.), because CPU stalls the whole time, which might lead to CPU under-utilization and drop in performance. That's why small transfer PM traffic mainly uses UDP. Please note that,

DCQCN also uses UDP as underlying transport layer protocol, but DCQCN addresses these issues by leveraging both the PFC and ECN mechanism. However, the congestion feedback mechanism is different in DCQCN, since the underlying ROCEv2 utilizes connection less protocol i.e. UDP, so TCP like ACK feedback per frame transmission is not implemented. As a result, congestion feedback mechanism in ROCEv2 somehow needs to be request agnostic. Upon receiving a ECN marked packet, the receiver NIC sends ROCEv2 standardized congestion notification packet (CNP) to the sender NIC in N microsecond time interval, until the receiver NIC continues receiving ECN marked packets [17]. Besides that DCQCN[17] needs a lot of parameter tuning for the superior performance, which seems rather impractical in a real environment.

Head of the Line (HOL) Blocking

Switch buffer plays an important role for the overall performance. Switch buffer is shared amongst multiple outgoing ports, and when the buffer occupancy by the packets exceeds the capacity, the packet is dropped. If any mechanism puts the buffer occupancy low then there would not be packet drop, but the throughput would be low. In order to address the latency and throughput trade-off, DCTCP RFC suggests marking threshold of $K > (RTT \times C)/7$, where C is the link capacity in packets per second. For example, if we have 10Gbps link and RTT of 20 μ S, then the threshold needs to be put 150. As a result, PM traffic which has high latency QoS requirement, might encounter queuing delay imposed by ~ 150 packets. So PM packets need to be treated differently than other storage traffic. Please note that even though PM traffic class are given strict priority over others, HOL blocking can still occur since, 1) PM packet can stuck behind other PM packets, 2) Switch serves packet in a non-preemptive manner.

5.2.2 Existing Proposals

It's true that there are numerous works which address the QoS of the small transfer flows, but those are not in the context of memory transfer flows. For example HOMA [18] is a receiver driven feedback based mechanism, which piggyback the receiver buffer status to the sender, so that the sender can schedule it's flow accordingly. One of the main issue with this per packet based feedback mechanism is, if a small transfer consists of multiple packets, then packet P_i has to wait until acknowledgement of packet P_{i-1} doesn't arrive. [108, 109] uses in network priority queues, but their proposal needs to implement some of the scheduling policy in the switch ASIC, which make it hard to use the solution ubiquitously. On the other hand, our solution doesn't require any additional hardware upgrade or anything. Our solution can work with slight modification at the kernel level, and small configuration at the switch end.

5.2.3 Standalone Network QoS Management is Not Enough

One might argue instead of transport layer QoS differentiation mechanism, why not leverage the in network scheduling mechanism to achieve the goal. For example Cisco switches offers the opportunity to implement different packet scheduling policies like weighted round robin, strict priority etc such that different class of traffic can be treated differently. In the next subsection we discuss the limitation and drawback of implementing these mechanisms.

Deployment Difficulty

One step of implementing QoS management in eligible switches, is to filter the packets based on five tuples (source, destination, protocol, source port, destination port), and then assign a priority class. In a distributed environment, these decisions are pretty dynamic, and hard to decide statically

to filter and assign the priority class to any flow. For example, service provider might want to deploy n additional servers for performance enhancement, then it requires to change the QoS configuration for all the switches connected with those n servers. Due to this hassle, in network qos is not usually implemented. Instead it's much more easier and convenient to impose QoS at the sender or receiver end (flow rate control), which [100], [16] has followed.

Packet Drop

Switch packet drop occurs when buffer is overwhelmed with the incoming packets. And in data center environment, we can't afford packet drop, as re-transmission and timeout mechanism could cause multiple QoS violation. So, we need to put the appropriate flow rate control mechanism by the upper layer (i.e. Transport (L4) layer), so that switch buffer doesn't get overwhelmed very frequently. PFC, a layer 2 flow rate control mechanism was developed to address this buffer overloading problem, but PFC cannot distinguish between traffic of same priority class, going to different destinations. Thus the congestion along a given destination path will generate a PAUSE frame, which causes blocking of all the incoming flow packets belonging to the same priority class, even if the destination is different and there is no congestion at the corresponding path. So, in order to let the IP (L3) layer QoS policies (strict priority, WRR etc) to work properly, there needs a coordination mechanism with Layer 4 (Transport) in place, such that packet drop can be minimized.

5.3 Proposed Mechanism

We propose a mechanism which leverages existing QoS differentiation features both in IP layer and Transport layer. It is clear from the above discussion, best QoS treatment can be achieved for the small transfer memory traffic if any incoming packet, 1) has a buffer reservation in switch such that packets never get dropped, and 2) those packets should be given the highest

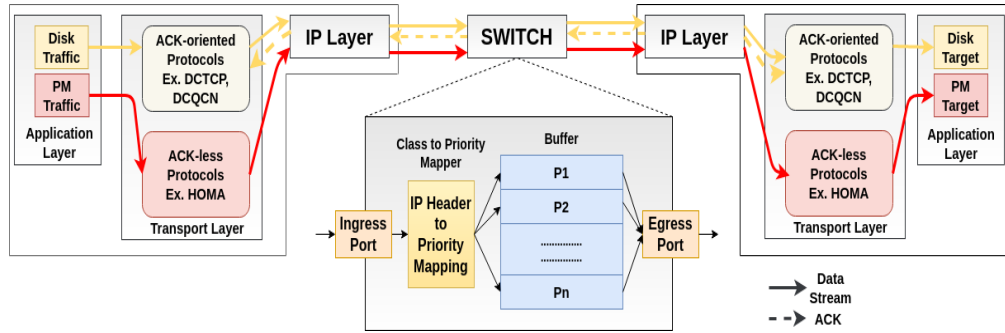


Figure 5.2: Our proposed mechanism, solid line represents the data stream and the dotted line represents the ACKs corresponding to the data stream. Note that the connection oriented protocol like DCTCP or DCQCN leverages the ECN enabled ACK feedback for the flow control, such that switch buffer doesn't get overwhelmed and drop any packet. Whereas small transfer PM over fabrics traffic doesn't follow any feedback based flow rate control. We also show details of in-network packet scheduling mechanism. Priority classes (a.k.a priority queue) (P_1, P_2, \dots, P_n) are predefined. Traffic class marking in the IP header is done at the end point (Transport Layer), and the job of Class to Priority Mapper is to forward the traffic into different priority queues. priority. On the other hand, medium and large size transfer which uses mainly TCP (i.e. DCTCP or variants of DCTCP) or RDMA (i.e. DCQCN), should target to control the buffer occupancy at a point where a better throughput and latency trade-off can be achieved.

In summary, If the buffer size is B packets, best QoS treatment for latency and throughput (memory and Storage traffic respectively) sensitive traffic can be achieved if there is a buffer reservation P for memory traffic, and transport layer flow control targets to keep buffer size within $B - P$ packets. Besides that, while scheduling, the memory traffic should be treated as highest priority (e.g. strict priority).

We show the details of in network packet scheduling mechanism in Figure 5.1. The strict priority classes (P_1, P_2, \dots, P_n) are predefined. Priority assignment to any individual packet is done based on the protocol number attached in the packet header. By doing that, we avoid the deep packet inspection overhead (i.e. analyzing the packet payload for traffic classification), performed on each packet. Traffic classification can also be done using

VXLAN mechanism, but it adds up delay due to packet encapsulation and decapsulation mechanism by VTEP [110]. If the overhead is negligible, VXLAN mechanism can be used instead. In any case, packet marking needs to be done at endpoint (Transport Layer). In the diagram, only P_n buffer is used for the large transfer traffic, and the ECN marking is done based on the buffer occupancy in P_n . The RED threshold K , refers to the buffer size after which ECN marking should be triggered, which needs to be some fraction of $B - P$, such that better latency and throughput trade-off can be achieved. In order to provision qos differentiation for large transfer traffics, we follow the transport layer flow rate control mechanism proposed in our previous work [36]. For the small transfer flows, we utilize in-network strict priority queues. Since the network switches needs to filter the packet and assign priority of single packet arrives in, the packets require to get marked using priority specific values at the host/transmitter end. Also, we should not treat/tag large flows as highest priority flows, because that could cause starvation to other flows. We propose marking the packets in IP header, and filter the packets based on the IP specific value at the switch. Figure 5.4 shows the marking mechanism at the transmitter end. We focus on the protocol number attribute of IP header (Figure 5.3). Note that, in a IP header protocol number is 8 bits long, and it actually tells the network devices which transport layer this packet belongs to. For example, for TCP and UDP, the protocol number is 6 and 17 respectively. According to RFC decimal value 144-252 are still unassigned. For our work, we choose protocol value 144, if this flow belongs to PM traffic. For that, we need to make changes in the TCP/IP layer of the stack, which we show in figure 5.4. Inside the network device, we filter IP packet with protocol number value 144, and give the highest priority (strict priority) over other transport protocol specific flows. So the overall architecture works as follows: In the network devices, different priority queues are predefined for the small transfer traffic, so that it can avoid HOL blocking, and QoS differentiation can be achieved for different latency sensitive small transfer traffic (e.g. memory traffic, IPC traffic etc). For the throughput sensitive large transfer traffic, we leverage the

0-3	4-7	8-15	16-31	
Version	IHL	TOS	Length	
Identification			Flags	Fragment Offset
TTL		Protocol	Checksum	
Source Address				
Destination Address				

Figure 5.3: Structure of the IP header (20 bytes). Switches maps the priority based on the **protocol** value embedded in the IP header by the end point. We keep remaining fields in the IP header unchanged.

ECN based mechanism, to modulate the flow rate at the transport end so that, 1) switch buffer occupancy can be minimized to avoid packet drop and 2) QoS differentiation for throughput sensitive traffic can be achieved.

5.4 Performance Evaluation

5.4.1 Coexistence of DCTCP and RDMA Traffic

In this section, we discuss the bandwidth sharing for mixture of DCTCP and QCDCN traffic. For this experiment, we have DCTCP and QCDCN traffic, destined for different destination, but share a bottleneck bandwidth. Figure 5.5 shows the topology for this experiment.

Default behavior for bandwidth sharing

Fig 5.7 shows the bandwidth distribution result when DCTCP and DCQCN has a shared bottleneck link. For this experiment, we choose optimal parameter (KMIN, KMAX, PMAX, etc) setting for DCQCN, and see the outcome in presence of DCTCP traffic. In this experiment, initially DCQCN grabs most of the link bandwidth, but as traffic amount increases (initially DCTCP goes through slow start phrase) with time, DCQCN drops in performance. The reason hides behind the mechanism how DCTCP and DCQCN flows react during the congestion avoidance phrase. In case of

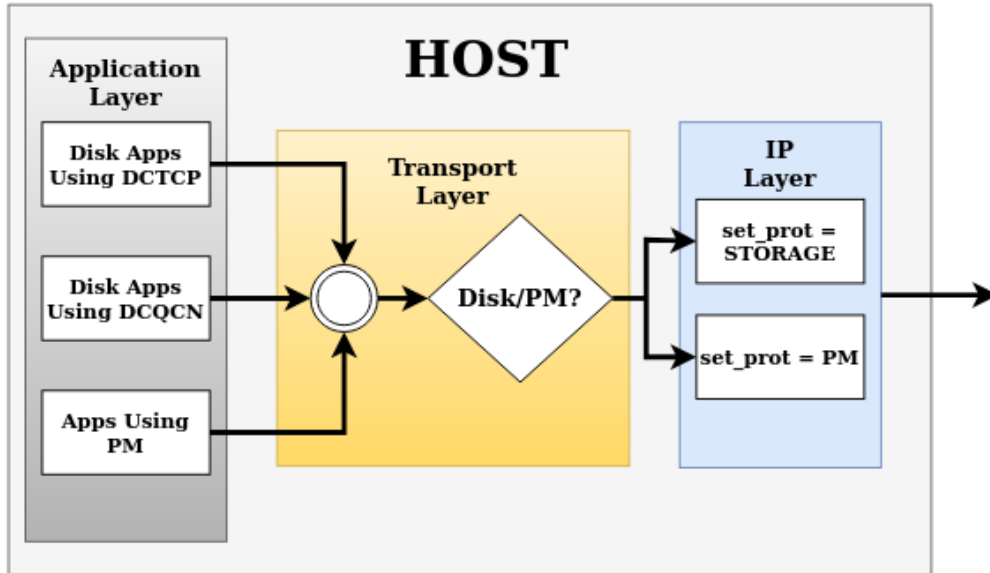


Figure 5.4: Marking mechanism at the host end. Application layer of the host machine provides hints to the IP layer via the transport layer (e.g. using socket priority), about the type of the flow. Based on the type, IP layer set the protocol number in the generated packet header and inject the IP packet into the network.

DCTCP, if there is no ECN in the last window of transfer, congestion window size increase by 1 Frame. In terms of rate increase if we define it as R_{AI} , during the congestion avoidance phrase, current rate RC_i would be $RC_i = RC_i + R_{AI}$. On the other hand, in case of DCQCN, RC_i changes as follows:

$$RT_i(n) = RT_i(n-1) + R_{AI} \quad (5.1)$$

$$RC_i(n) = \left\{ RT_i(n-1) + RC_i(n-1) \right\} / 2 \quad (5.2)$$

Note that, initially $RT_i(n) = RC_i(n)$, and during the Fast Recovery (a.k.a congestion avoidance phrase), the current rate RC_i would be approximately $\frac{RC_i + RC_i + R_{AI}}{2} = RC_i + \frac{R_{AI}}{2}$, which is less than the DCTCP case. As a result, though initially DCQCN flows grab much of the bandwidth, during the Fast recovery DCTCP rate increase is higher than the DCQCN, as a result DCTCP flows gradually end up grabbing most of the bandwidth and stabilizes. Fig 5.6 shows a plot of this phenomenon and our experiment

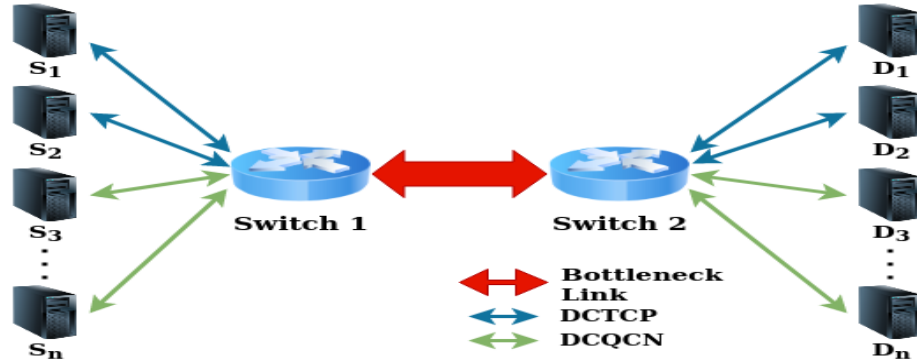


Figure 5.5: Topology for the experimental setup. Each of the link capacity is 100 Gbps, and bottleneck link is between Switch 1 and Switch 2. Each of the switch has both ECN and PFC enabled. Each of the host (S_1, S_2, \dots, S_n) and target (D_1, D_2, \dots, D_n) machine has RDMA capable NIC installed.

in Fig 5.7 exhibits similar behavior. Another reason for this asymmetrical bandwidth distribution could be generation of PAUSE frame. But during our experiment, PFC frame was never triggered, since ingress queue depth was below threshold due to the optimal parameter settings for DCQCN.

Bandwidth shaping with quality factor notion

In this section, we discuss how notion of quality factor can be used to address the issue discussed in previous section. In this experiment, we apply the quality factor specific changes for only DCTCP, and run the same experiment. From the result, it is clear that DCTCP and DCQCN flows get equal bandwidth sharing when bottleneck link is congested. Since, DCTCP doesn't react on the PAUSE frame generated by L2 ingress ports, the quality factor helps in controlling the DCTCP flows in a way such that ingress port doesn't get overwhelmed with its packet. Note that, in this manner, *quality factor*

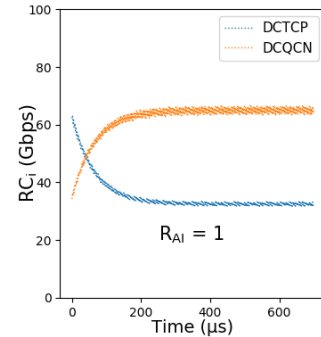


Figure 5.6: Evolution of current rate RC_i for both DCTCP and DCQCN. Initially RC_i of DCQCN is higher than DCTCP, which reduces over time.

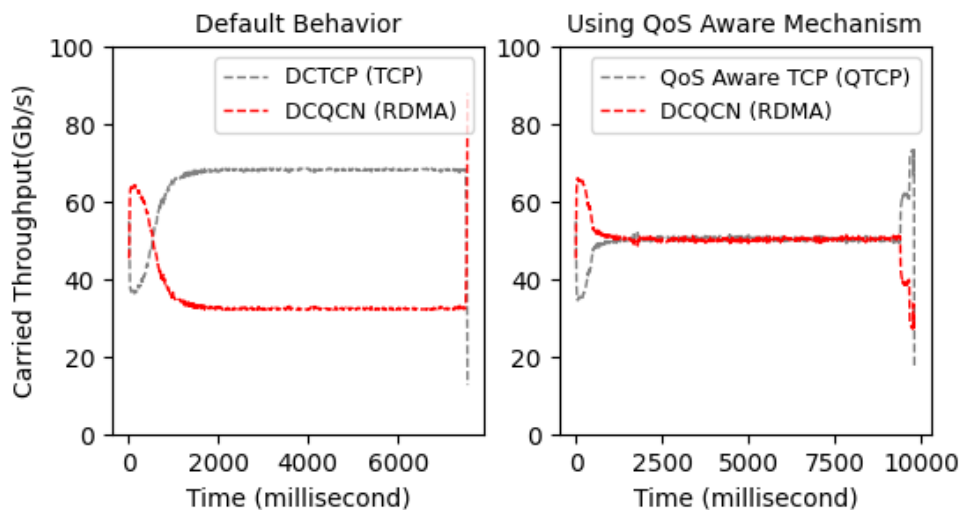


Figure 5.7: Results of bandwidth sharing with or without QoS aware flow rate control. Left shows the default behavior when DCTCP and DCQCN coexists in a environment and no QoS aware mechanism applied. Right diagram shows the result when QoS aware mechanism is applied. Note that, TCP can be used for RDMA, but for our experiment, we assume DCQCN is used for RDMA.

can also be used to provide RDMA (DCQCN) traffic

more priority over the TCP traffic or vice versa. Note that original DCQCN suggests bandwidth reservation for the TCP traffic, in order to provide fairness for DCQCN flows [17]. But the problem is two fold 1) During non-congested period, the outgoing links could be underutilized, 2) Optimal parameter tuning in order to get DCTCP and DCQCN flow fairness is pretty difficult to ensure and could be error prone. Instead we are proposing a mechanism, which works at the end point (transport end) and much easier to implement.

Notion of quality factor can also be used to differentiate qos for individual flows. For example, in Figure 5.8, we set the relative ratio of flows as 1.0: 1.5: 2.0: 3.0. Though we see qos differentiation to some extent within the QTCP flows, this is quite different than the target ratio. Recall the ECN parameter setup are in favor of optimal performance of DCQCN, not DCTCP. In case of standalone DCTCP flows, the ratio is approximately equal to the target throughput ratio. Figure 5.8(b) shows some instability in bandwidth sharing,

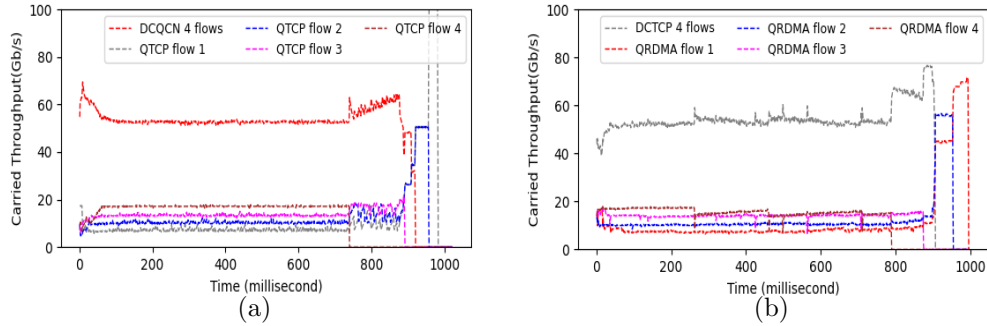


Figure 5.8: (a) shows the individual flow bandwidth distribution for DCTCP, when all the DCQCN flows shared the rest bandwidth equally. (b) shows the reverse case, DCQCN flows are having different bandwidth sharing, and all the DCTCP flows are having same bandwidth sharing.

we need to explore more to figure out the actual reason behind that.

As this is clear from the above discussion, quality factor notion can be used to provide differentiated qos to different application having large transfer storage flows, we will classify QTCP and QRDMA flows as storage traffic onwards.

5.4.2 QoS for Small Transfer Memory Traffic

Traffic Generation Model

For the PM traffic model, we experiment with different intensity of the PM access. The PM access size is uniformly distributed among 1-2 cacheline (128-256 bytes). Inter request arrival interval follows poisson distribution, and the average is determined as such PM traffic consists of 10%,12% and 20% of the total load (bottleneck bandwidth which is 100 Gbps in this experiemnts).

5.4.3 Result and Discussion

Though using the notion of qualify factor, it is possible to impose qos differentiation for storage traffic, the issue of QoS for small transfer memory transfer can not be resolved using the same mechanism. From figure 5.9(a) one thing is obvious, using only our proposed qos aware solution, the latency

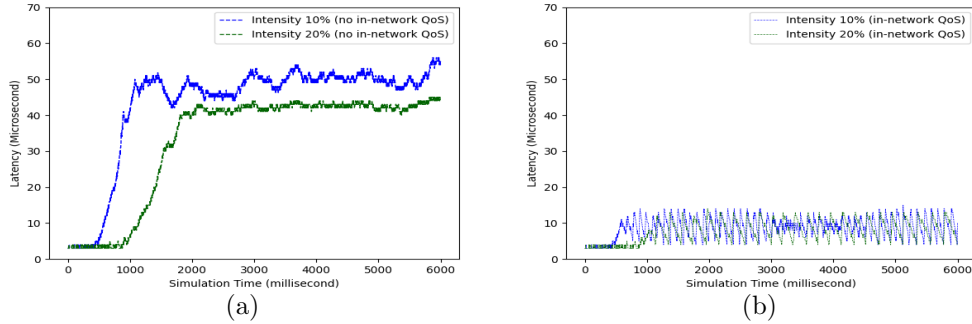


Figure 5.9: Latency distribution of PM traffic with/without in-network priority management with respect to different intensity. (a) Shows the result when our scheme only uses transport layer QoS. (b) shows the result when our solution involves coordination between our proposed Transport layer QoS and existing IP layer QoS as discussed in section 5.3.

distribution is around $40-55\mu\text{S}$, based on the intensity of the PM traffic, which is pretty high for memory traffic. According to SNIA, the core requirement² for networked PM is 100 nanoseconds for read operations and 500 nanoseconds for write operations, compared to that QoS requirement, $40-55\mu\text{S}$ is too high. Our PM traffic follows the ROCEv2 protocol as proposed. Though the topology we worked on, each hop adds $1\mu\text{S}$ of service delay for each packets, and the PM located 3 hops away, the best latency that can be offered for a single PM access is $\sim 3\mu\text{S}$ (considering only the transfer and service time when there is no queue in the network). But the result we get using only transport layer QoS, is $40-55\mu\text{S}$, which is too high than the QoS requirement specified. Also the PM latency increases when the PM load is low (10% compared to 20%), since PMOF follows connectionless protocol, and all the other storage flows follow connection oriented protocol (QTCP and QRDMA), network buffer consists of more PM traffic with high utilization.

Figure 5.9(b) shows the result, when we apply the 1)coordination mechanism between the transport layer qos and the existing IP layer QoS we proposed in section 5.3, and 2) in network priority packet filtering using the IP QoS. The latency distribution follows almost same trend regardless of the

²<https://www.snia.org/sites/default/files/ESF/Extending-RDMA-for-Persistent-Memory-over-Fabrics-Final.pdf>

utilization. The latency distribution falls between $4 - 12\mu\text{S}$, which closely match with the ultra low latency requirement PMOF requires. **In summary, we achieve these ultra low latency QoS requirement:**

1. Marking the PM packets. The marking is done by:
 - (a) Committing minimal changes in the legacy TCP/IP stack
 - (b) Marking targets conventional IP packets and changes one attribute of the packet header, which is *ProtocolNumber*
2. Instruct the network devices to priority filter the packets based on one attribute (*ProtocolNumber*) of the IP packet header. Note that, filtering can be done using the location of the PM source or destination (source ip, and destination ip), source port or destination port, or the protocol number. Using the protocol number as filter makes PM traffic management less error prone and more straight forward.

Recall that, Solution proposed by HOMA [18], requires specilized NIC design and besides that it doesn't offer in-network priority. We overcome these two limitations with a simple extension of our qos aware scheme. Another limitation of the existing feature was protocol friendliness to offer QoS differentiations. We also overcome that, and we disccus it in the next section.

QoS differentiation For Storage Traffic

From figure 5.10 shows the storage bandwidth distribution in presence of PM traffic. With 10% of the PM traffic, the storage gets almost 85.5% of the total bandwidth, and with 20%of the PM traffic, the storage gets almost 74.2% of the total bandwidth. Now for this experiment we used the updated scheme we used to support the DCTCP friendlines discussed in Algorithm 1, as because the persistent memory can grab some of the provisioned bandwidth (100 Gbps), and PM protocol follows connectionless schema.

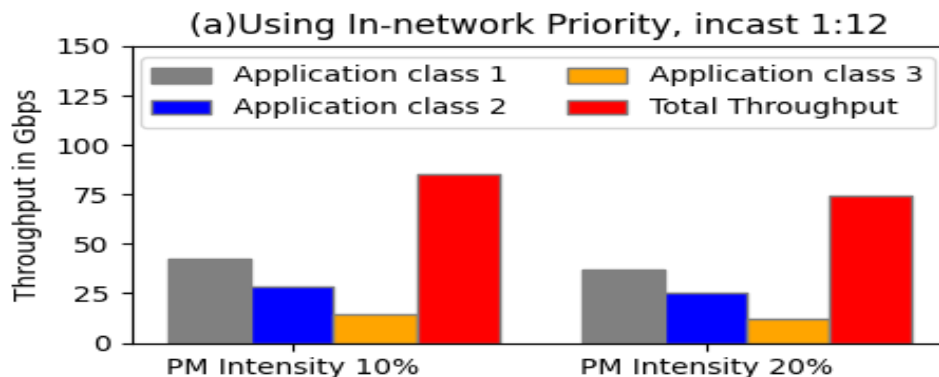


Figure 5.10: Storage Bandwidth Distribution for Different Class of Applications in different PM traffic intensity.

From figure 5.10 this is clear that, though bandwidth provisioned for storage traffic is 100 Gbps, amongst which some fraction of bandwidth was taken by the high priority PM traffic flows, the rest of the bandwidth was distributed according to the QoS ratio specified. For example, in case of 10% intensity, the effective bandwidth for the storage traffic was 90 Gbps, and the collective carried throughput is close to this 90 Gbps ($\sim 85.54\%$). And the same trend we see in case of 20% intensity of PM traffic (collective carried throughput $\sim 74.3\%$). We didn't run our experiments for higher intensity of PM traffic, since local PM device is responsible in serving most of memory access request [107], and 20% seems pretty reasonable traffic amount responsible for remote PM access.

CHAPTER 6

DISCUSSION

6.0.1 Summary and Conclusions

In this dissertation, we have studied the data center network energy management and proposed a lightweight mechanism that involves coordination among three types of controllers: local controller at each switch, a global network controller, and an energy aware user request assignment controller. Our mechanism is feedback based mechanisms that monitors network traffic activity (Local Controller), periodically sends the information to the decision maker (Global Controller and Request Assignment Controller), and sends back the decision to the action taker(Local Controller). Our mechanism can save power upto 40%, with a tiny penalty in the performance ($\sim 16\mu S$). Though Fat tree is very commonly used in data center, for HPC environment Hypercube is more appropriate. We compare the performance and power metrics of these two topologies and find that fat-tree can provide low power consumption compared to Hypercube, but in terms of performance (i.e.latency) Hypercube outperform Fat tree.

Next, we discussed an adaptive mechanism to decide whether to migrate, activate an inactive copy, or reduce the number of active copies to handle changes in the network traffic due to variations in network traffic generated by highspeed storage devices that are becoming the norm in the data center.

Through extensive simulations, we show that it is possible to achieve near 47% improvement in average delay with purely read traffic. However, for chunks that are mostly written, the best strategy is to migrate them out of the congested node and place them at some low utilized node.

Next, we proposed an QoS aware transport layer solution which works both NVMe fabrics solution TCP and RDMA. Our solution can be implemented using ECN-capable switches and does not need any specialized hardware. We propose concept of Quality Factor (QF), which works above the rate control mechanisms proposed for TCP and RDMA, and gurantee QoS differentiation amongst multiple flows in a distributed manner. Our solution offers backward compatility, means it can coexist with the TCP variant DCTCP and still can manage QoS differentiation amongst the flows that uses our proposed QoS aware solution. Besides DCTCP friendliness, we also show the effectiveness of our scheme to achieve RTT fairness issue. We also discuss the limitation of ethernet based QoS proposals (PFC and ETS), and the limiatation of our scheme in offering QoS in a mixed workload environment (more specifically in offering ultra low latency QoS requirement of small transfer remote access memory traffic). We leverage in-network flow priority mechanism to address ultra low latency QoS issue, and propose a coordination between TCP and IP layer to differentiate the memory traffic so that traffic belongs to this class get expedited treatment at the switch end, regardless of any other class of traffic.

6.0.2 Future Work

We can extend our existing works in several ways, as our work mainly focuses on QoS differentiation in the presence of long transfer storage flows and tiny transfer memory flows. We provide the highest priority (strict priority) to the small transfer flows and instruct the intermediate switches to follow the strict priority rule. In the future, we want to explore the cases when a network consists of several classes of small transfer flows and how to assign the priorities to meet the QoS demand for all the flows. That requires extensive

analysis of the network workload pattern and dynamic priority assignment of the existing flows. This workload analysis is computationally expensive and requires implementing a distributed analytical model to create no hot spots anywhere in the overall system. Recent storage and NIC technologies offer features to offload CPU and storage controller tasks using the native resources. These solutions are IPU (Infrastructure Processing Units), Active Storage, smart switches (Barefoot). With these hardware-based solutions, computations become fast. These devices can perform traffic categorization, intelligent routing, and traffic analysis relatively faster without creating a bottleneck at the computational resources like the CPU. With these emerging technologies becoming ubiquitous in data centers soon, we can extend our QoS aware solution by introducing real-time traffic analytics to take QoS aware decisions more accurately.

REFERENCES

- [1] J. Varia, S. Mathew *et al.*, “Overview of amazon web services,” *Amazon Web Services*, vol. 105, 2014.
- [2] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, “Bigtable: A distributed storage system for structured data,” *ACM Trans. Comput. Syst.*, vol. 26, no. 2, jun 2008. [Online]. Available: <https://doi.org/10.1145/1365815.1365816>
- [3] K. Kapusta and G. Memmi, “Data protection by means of fragmentation in distributed storage systems,” in *2015 International Conference on Protocol Engineering (ICPE) and International Conference on New Technologies of Distributed Systems (NTDS)*, 2015, pp. 1–8.
- [4] C. Ruan and V. Varadharajan, “Data protection in distributed database systems,” in *Foundations of Intelligent Systems*, M.-S. Hacid, N. V. Murray, Z. W. Raś, and S. Tsumoto, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 191–199.
- [5] A. Gulati, I. Ahmad, C. A. Waldspurger *et al.*, “Parda: Proportional allocation of resources for distributed storage access.” in *FAST*, vol. 9, 2009, pp. 85–98.

- [6] H. Goudarzi and M. Pedram, “Maximizing profit in cloud computing system via resource allocation,” in *2011 31st International Conference on Distributed Computing Systems Workshops*. IEEE, 2011, pp. 1–6.
- [7] L. M. Haji, S. Zeebaree, O. M. Ahmed, A. B. Sallow, K. Jacksi, and R. R. Zeabri, “Dynamic resource allocation for distributed systems and cloud computing,” *TEST Engineering & Management*, vol. 83, pp. 22 417–22 426, 2020.
- [8] Sohan *et al.*, “Characterizing 10 gbps network interface energy consumption,” in *IEEE LCN*, 2010, pp. 268–271.
- [9] Christensen *et al.*, “Ieee 802.3 az: the road to energy efficient ethernet,” *IEEE Communications Magazine*, vol. 48, no. 11, 2010.
- [10] “Nvm express over fabrics revision 1.1,” <https://nvmexpress.org/wp-content/uploads/NVMe-over-Fabrics-1.1-2019.10.22-Ratified.pdf>, 2019.
- [11] Z. Guz, H. H. Li, A. Shayesteh, and V. Balakrishnan, “Nvme-over-fabrics performance characterization and the path to low-overhead flash disaggregation,” in *ACM SYSTOR*, 2017.
- [12] —, “Performance characterization of nvme-over-fabrics storage disaggregation,” *ACM Trans. Storage*, 2018.
- [13] M. Ray, J. Biswas, A. Pal, and K. Kant, “Adaptive data center network traffic management for distributed high speed storage,” *IEEE LCN*, 2019.
- [14] R. Davis, “The network is the new storage bottleneck,” <https://www.datanami.com/2016/11/10/network-new-storage-bottleneck/>, 2016.
- [15] M. Alizadeh, A. Greenberg, D. Maltz, J. Padhye, P. Pate, B. Prabhakar, S. Sengupta, and M. Sridharan, “Data center tcp (dctcp),” in *ACM SIGCOMM*, 2010.

- [16] B. Vamanan, J. Hasan, and T. Vijaykumar, “Deadline-aware datacenter tcp (d2tcp),” in *ACM SIGCOMM*, 2012.
- [17] Y. Zhu, H. Eran, D. Firestone, C. Guo, M. Lipshteyn, Y. Liron, J. Padhye, S. Raindel, M. H. Yahia, and M. Zhang, “Congestion control for large-scale rdma deployments,” ser. SIGCOMM ’15. Association for Computing Machinery, 2015.
- [18] B. Montazeri, Y. Li, M. Alizadeh, and J. Ousterhout, “Homa: A receiver-driven low-latency transport protocol using network priorities,” in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, 2018, pp. 221–235.
- [19] S. Herrera-Alonso, M. Rodriguez-Prez, M. Fernandez-Veiga, and C. Lopez-Garca, “Optimal configuration of energy-efficient ethernet,” *Computer Networks*, 2012.
- [20] P. Reviriego, J. Hernandez, D. Larrabeiti, and J. A. Maestro, “Burst transmission for energy-efficient ethernet,” *IEEE Internet Computing*, July 2010.
- [21] R. F. e Silva and P. M. Carpenter, “Energy efficient ethernet on mapreduce clusters: Packet coalescing to improve 10gbe links,” *IEEE/ACM Transactions on Networking*, Oct 2017.
- [22] M.-K. Shin, K.-H. Nam, and H.-J. Kim, “Software-defined networking (sdn): A reference architecture and open apis,” in *2012 International Conference on ICT Convergence (ICTC)*, 2012, pp. 360–361.
- [23] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, “Dynamo: Amazon’s highly available key-value store,” *ACM SIGOPS operating systems review*, vol. 41, no. 6, pp. 205–220, 2007.

- [24] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, “The hadoop distributed file system,” in *2010 IEEE 26th symposium on mass storage systems and technologies (MSST)*. Ieee, 2010, pp. 1–10.
- [25] M. Hock, M. Veit, F. Neumeister, R. Bless, and M. Zitterbart, “Tcp at 100 gbit/s tuning, limitations, congestion control,” in *2019 IEEE 44th Conference on Local Computer Networks (LCN)*, 2019, pp. 1–9.
- [26] M. J. Rashti, R. E. Grant, A. Afsahi, and P. Balaji, “iwarp redefined: Scalable connectionless communication over high-speed ethernet,” in *2010 International Conference on High Performance Computing*, 2010, pp. 1–10.
- [27] “Infiniband architecture specification release 1.2.1 annex a17: Rocev2,” <https://cw.infinibandta.org/document/dl/7781>, 2014.
- [28] S. Ha, I. Rhee, and L. Xu, “Cubic: A new tcp-friendly high-speed tcp variant,” p. 6474, 2008.
- [29] M. Gerla, M. Sanadidi, R. Wang, A. Zanella, C. Casetti, and S. Mascolo, “Tcp westwood: congestion window control using bandwidth estimation,” in *GLOBECOM’01. IEEE Global Telecommunications Conference (Cat. No.01CH37270)*, vol. 3, 2001, pp. 1698–1702 vol.3.
- [30] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, “BBR: congestion-based congestion control,” *ACM Queue*, vol. 14, no. 5, pp. 20–53, 2016.
- [31] G. Zeng, W. Bai, G. Chen, K. Chen, D. Han, and Y. Zhu, “Combining ecn and rtt for datacenter transport,” in *APNet*, 2017.
- [32] S. Sondur, M. Ray, J. Biswas, and K. Kant, “Implementing data center network energy management capabilities in ns3,” in *2017 Eighth International Green and Sustainable Computing Conference (IGSC)*, 2017, pp. 1–8.

- [33] M. Ray, S. Sondur, J. Biswas, A. Pal, and K. Kant, “Opportunistic power savings with coordinated control in data center networks,” in *Proceedings of the 19th International Conference on Distributed Computing and Networking*, ser. ICDCN '18. New York, NY, USA: Association for Computing Machinery, 2018. [Online]. Available: <https://doi.org/10.1145/3154273.3154328>
- [34] J. Biswas, M. Ray, S. Sondur, A. Pal, and K. Kant, “Coordinated power management in data center networks,” *SUSCOM*, vol. 22, pp. 1 – 12, 2019.
- [35] M. Ray, J. Biswas, A. Pal, and K. Kant, “Adaptive data center network traffic management for distributed high speed storage,” in *2019 IEEE 44th LCN Symposium on Emerging Topics in Networking (LCN Symposium)*, 2019, pp. 166–174.
- [36] J. Biswas, J. Gupta, K. Kant, A. Pal, and D. Minturn, “Provisioning differentiated qos for nvme over fabrics,” in *2021 IEEE 46th Conference on Local Computer Networks (LCN)*, 2021, pp. 154–161.
- [37] L. Brown, “Acpi in linux,” in *Linux Symposium*, vol. 51, 2005.
- [38] P. Arroba, J. Moya, J. L. Ayala, and R. Buyya, “Dvfs-aware dynamic consolidation of virtual machines for energy efficient clouddata centers,” *Concurrent and Computation: Practice and Experimence*, 2010.
- [39] T. Gurout, T. Monteil, G. Da Costa, R. Neves Calheiros, R. Buyya, and M. Alexandru, “Energy-aware simulation with dvfs,” *Simulation Modelling Practice and Theory*, vol. 39, pp. 76–91, 2013.
- [40] S. Wang, Z. Qian, J. Yuan, and I. You, “A dvfs based energy-efficient tasks scheduling in a data center,” *IEEE Access*, 2017.
- [41] Verma *et al.*, “pmapper: power and migration cost aware

- application placement in virtualized systems,” in *ACM/IFIP/USENIX International Conference on Middleware*, 2008, pp. 243–264.
- [42] Liu *et al.*, “Greencloud: A new architecture for green data center,” in *ACM ICAC-INDST*, 2009, pp. 29–38.
- [43] K. Kant, “Multistate power management of communications links,” in *Proc. of COMSNET*, 01 2011.
- [44] Raghavendra *et al.*, “No power struggles: Coordinated multi-level power management for the data center,” in *ACM ASPLOS*, 2008, pp. 48–59.
- [45] Heller *et al.*, “Elastictree: Saving energy in data center networks,” in *USENIX Conference on Networked Systems Design and Implementation*, 2010.
- [46] Mostowfi *et al.*, “A simulation study of energy-efficient ethernet with two modes of low-power operation,” *IEEE Communications Letters*, vol. 19, no. 10, pp. 1702–1705, 2015.
- [47] C. Thaenchaikun *et al.*, “Augmenting the energy-saving impact of ieee 802.3az via the control plane,” in *IEEE ICCW*, 2015, pp. 2843–2849.
- [48] Nedeveschi *et al.*, “Reducing network energy consumption via sleeping and rate-adaptation,” in *USENIX NSDI*, 2008, pp. 323–336.
- [49] Abts *et al.*, “Energy proportional datacenter networks,” *SIGARCH Comput. Archit. News*, vol. 38, no. 3, pp. 338–347, Jun. 2010.
- [50] Al-Fares *et al.*, “Hedera: Dynamic flow scheduling for data center networks,” in *USENIX NSDI*, 2010, pp. 19–19.
- [51] X. Wang *et al.*, “Carpo: Correlation-aware power optimization in data center networks,” in *IEEE INFOCOM*, 2012, pp. 1125–1133.
- [52] Gao *et al.*, “Energy-aware load balancing in heterogeneous cloud data centers,” in *ACM ICMSS*, 2017, pp. 80–84.

- [53] Paya *et al.*, “Energy-aware load balancing and application scaling for the cloud ecosystem,” *IEEE Transactions on Cloud Computing*, 2015.
- [54] K. Zheng *et al.*, “Joint power optimization of data center network and servers with correlation analysis,” in *IEEE INFOCOM*, 2014, pp. 2598–2606.
- [55] H. Jin *et al.*, “Joint host-network optimization for energy-efficient data center networking,” in *IEEE IPDPS*, 2013, pp. 623–634.
- [56] Meng *et al.*, “Improving the scalability of data center networks with traffic-aware virtual machine placement,” in *IEEE INFOCOM*, 2010, pp. 1154–1162.
- [57] D. Li *et al.*, “Joint power optimization through vm placement and flow scheduling in data centers,” in *IEEE IPCCC*, 2014, pp. 1–8.
- [58] Travostino *et al.*, “Seamless live migration of virtual machines over the man/wan,” *Future Gener. Comput. Syst.*, vol. 22, no. 8, pp. 901–907, 2006.
- [59] H. Wu, S. Nabar, and R. Poovendran, “An energy framework for the network simulator 3,” in *SimuTools*, 2011.
- [60] C. Tapparello, H. Ayatollahi, and W. B. Heinzelman, “Energy harvesting framework for network simulator 3,” in *ENSsys@SenSys*, 2014.
- [61] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, “Dynamo: Amazon’s highly available key-value store,” in *Proceedings of Twenty-First ACM SIGOPS Symposium on Operating Systems Principles*, ser. SOSP ’07. New York, NY, USA: Association for Computing Machinery, 2007, p. 205220. [Online]. Available: <https://doi.org/10.1145/1294261.1294281>

- [62] X. Li and C. Qian, “Low-complexity multi-resource packet scheduling for network function virtualization,” in *2015 IEEE Conference on Computer Communications (INFOCOM)*, 2015, pp. 1400–1408.
- [63] Q. Wang, G. Shou, Y. Liu, Y. Hu, Z. Guo, and W. Chang, “Implementation of multipath network virtualization with sdn and nfv,” *IEEE Access*, vol. 6, pp. 32 460–32 470, 2018.
- [64] L. Zhang, Y. Deng, W. Zhu, J. Zhou, and F. Wang, “Skewly replicating hot data to construct a power-efficient storage cluster,” *Journal of Network and Computer Applications*, vol. 50, pp. 168–179, 2015. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1084804514001362>
- [65] D. Park, B. Debnath, Y. Nam, D. H. C. Du, Y. Kim, and Y. Kim, “Hotdatatrap: A sampling-based hot data identification scheme for flash memory,” in *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, ser. SAC ’12. New York, NY, USA: Association for Computing Machinery, 2012, p. 16101617. [Online]. Available: <https://doi.org/10.1145/2245276.2232034>
- [66] L. Liu, H. Wang, X. Liu, X. Jin, W. B. He, Q. B. Wang, and Y. Chen, “Greencloud: A new architecture for green data center,” in *Proceedings of the 6th International Conference Industry Session on Autonomic Computing and Communications Industry Session*, ser. ICAC-INDST ’09. New York, NY, USA: Association for Computing Machinery, 2009, p. 2938. [Online]. Available: <https://doi.org/10.1145/1555312.1555319>
- [67] J. Xu and J. A. B. Fortes, “Multi-objective virtual machine placement in virtualized data center environments,” in *2010 IEEE/ACM Int’l Conference on Green Computing and Communications Int’l Conference on Cyber, Physical and Social Computing*, 2010, pp. 179–188.

- [68] M. R. Garey and D. S. Johnson, *Computers and Intractability; A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1990.
- [69] X. Song and J. Liu, “Maintaining temporal consistency: pessimistic vs. optimistic concurrency control,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 7, no. 5, pp. 786–796, 1995.
- [70] H. T. Kung and J. T. Robinson, “On optimistic methods for concurrency control,” *ACM Trans. Database Syst.*, vol. 6, no. 2, pp. 213–226, Jun. 1981.
- [71] A. Verma, R. Koller, L. Useche, and R. Rangaswami, “Srcmap: Energy proportional storage using dynamic consolidation,” in *8th USENIX FAST*, Berkeley, CA, USA, 2010, pp. 20–20.
- [72] X. Wang, S. Yang, S. Wang, X. Niu, and J. Xu, “An application-based adaptive replica consistency for cloud storage,” in *2010 Ninth International Conference on Grid and Cloud Computing*, 2010, pp. 13–17.
- [73] Q. Wei, W. Lin, and Y. K. Leong, “Adaptive replica management for large-scale object-based storage devices.”
- [74] J. Liao, L. Li, H. Chen, and X. Liu, “Adaptive replica synchronization for distributed file systems,” *IEEE Systems Journal*, vol. 9, no. 3, pp. 865–877, 2015.
- [75] Z. Yang *et al.*, “Autotiering: Automatic data placement manager in multi-tier all-flash datacenter,” in *2017 IEEE 36th (IPCCC)*, 2017, pp. 1–8.
- [76] T. Wang, J. Wang, S. N. Nguyen, Z. Yang, N. Mi, and B. Sheng, “Ea2s2: An efficient application-aware storage system for big data processing in heterogeneous clusters,” in *2017 26th (ICCCN)*, 2017, pp. 1–9.

- [77] J. Xiong, J. Li, R. Tang, and Y. Hu, “Improving data availability for a cluster file system through replication,” in *2008 IEEE International Symposium on Parallel and Distributed Processing*, 2008, pp. 1–8.
- [78] E. Zamanian, C. Binnig, and A. Salama, “Locality-aware partitioning in parallel database systems,” in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, 2015, pp. 17–30.
- [79] Ananthanarayanan *et al.*, “Scarlett: Coping with Skewed Content Popularity in MapReduce Clusters,” *EuroSys ’11*, pp. 287–300, 2011.
- [80] C. L. Abad, Y. Lu, and R. H. Campbell, “DARE: Adaptive data replication for efficient cluster scheduling,” *IEEE ICC*, pp. 159–168, 2011.
- [81] Xie *et al.*, “Improving MapReduce performance through data placement in heterogeneous Hadoop clusters,” *IPDPSW 2010 IEEE International Symposium on*, vol. 9, pp. 29–42, 2010.
- [82] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, “Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling,” in *ACM EuroSys*, 2010, pp. 265–278.
- [83] S. Sivasubramanian, G. Alonso, G. Pierre, and M. van Steen, “Globedb: Autonomic data replication for web applications,” in *Proceedings of the 14th International Conference on World Wide Web*, 2005.
- [84] M. Karlsson and C. Karamanolis, “Choosing replica placement heuristics for wide-area systems,” in *24th International Conference on Distributed Computing Systems, 2004. Proceedings.*, 2004.
- [85] “Addressing and data center bridging (dcb),” <https://1.ieee802.org/dcb/>.
- [86] IEEE, “802.1qbb priority-based flow control,” <https://1.ieee802.org/dcb/802-1qbb/>, 2011.

- [87] F. Baker, D. L. Black, K. Nichols, and S. L. Blake, “Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers,” RFC 2474, 1998. [Online]. Available: <https://rfc-editor.org/rfc/rfc2474.txt>
- [88] J. Polk, F. Baker, and M. Dolly, “A Differentiated Services Code Point (DSCP) for Capacity-Admitted Traffic,” RFC 5865, 2010. [Online]. Available: <https://rfc-editor.org/rfc/rfc5865.txt>
- [89] S. Scargall, “Remote persistent memory,” in *Programming Persistent Memory*. Springer, 2020, pp. 347–371.
- [90] K. Ramakrishnan, S. Floyd, D. Black *et al.*, “The addition of explicit congestion notification (ecn) to ip,” 2001.
- [91] L. S. Brakmo, S. W. O’Malley, and L. L. Peterson, “Tcp vegas: New techniques for congestion detection and avoidance,” ser. SIGCOMM ’94, 1994, p. 2435.
- [92] V. Kokshenev and S. Suschenko, “Tcp reno congestion window size distribution analysis,” in *Information Technologies and Mathematical Modelling*, A. Dudin, A. Nazarov, R. Yakupov, and A. Gortsev, Eds., 2014, pp. 205–213.
- [93] “What nvme tcp means for networked storage,” <https://www.snia.org/sites/default/files/ESF/SNIA-NVMe-TCP-Final.pdf>, 2019.
- [94] Y. Zhang, Y. Tan, B. Stephens, and M. Chowdhury, “Rdma performance isolation with justitia,” 2019.
- [95] “Infiniband architecture specification release 1.2.1 annex a16: Roce,” <https://cw.infinibandta.org/document/dl/7148>, 2010.

- [96] R. Mittal, T. Lam, N. Dukkipati, E. Blem, H. Wassel, M. Ghobadi, A. Vahdat, Y. Wang, D. Wetherall, and D. Zats, “Timely: Rtt-based congestion control for the datacenter,” in *Sigcomm '15*, 2015.
- [97] K. Kant, *Introduction to computer system performance evaluation*. McGraw-Hill, 1992.
- [98] C. Guo, H. Wu, Z. Deng, G. Soni, J. Ye, J. Padhye, and M. Lipshteyn, “Rdma over commodity ethernet at scale,” in *ACM SIGCOMM*, 2016.
- [99] B. Montazeri, Y. Li, M. Alizadeh, and J. Ousterhout, “Homa: A receiver-driven low-latency transport protocol using network priorities,” in *ACM SIGCOMM*, 2018.
- [100] A. Munir, I. A. Qazi, Z. A. Uzmi, A. Mushtaq, S. N. Ismail, M. S. Iqbal, and B. Khan, “Minimizing flow completion times in data centers,” in *IEEE INFOCOM*, 2013.
- [101] C.-Y. Hong, M. Caesar, and P. B. Godfrey, “Finishing flows quickly with preemptive scheduling,” in *ACM SIGCOMM*, 2012.
- [102] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron, “Better never than late: Meeting deadlines in datacenter networks,” in *ACM SIGCOMM*, 2011.
- [103] M. Hemmati, A. Yassine, and S. Shirmohammadi, “An online learning approach to qoe-fair distributed rate allocation in multi-user video streaming,” in *2014 8th International Conference on Signal Processing and Communication Systems (ICSPCS)*, 2014, pp. 1–6.
- [104] W. Li, F. Zhou, K. R. Chowdhury, and W. Meleis, “Qtcp: Adaptive congestion control with reinforcement learning,” *IEEE Transactions on Network Science and Engineering*, vol. 6, no. 3, pp. 445–458, 2019.
- [105] A. Kalia, D. Andersen, and M. Kaminsky, “Challenges and solutions for fast remote persistent memory access,” in *Proceedings of the 11th*

- ACM Symposium on Cloud Computing*, ser. SoCC '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 105119. [Online]. Available: <https://doi.org/10.1145/3419111.3421294>
- [106] Y. Shan, S.-Y. Tsai, and Y. Zhang, “Distributed shared persistent memory,” in *Proceedings of the 2017 Symposium on Cloud Computing*, ser. SoCC '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 323337. [Online]. Available: <https://doi.org/10.1145/3127479.3128610>
- [107] P. Grun, S. Bates, and R. Davis, “Persistent memory over fabrics,” *SNIA Persistent Memory summit, San Jose, CA*, 2018.
- [108] P. X. Gao, A. Narayan, G. Kumar, R. Agarwal, S. Ratnasamy, and S. Shenker, “phost: Distributed near-optimal datacenter transport over commodity network fabric,” in *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies*, 2015, pp. 1–12.
- [109] W. Bai, L. Chen, K. Chen, D. Han, C. Tian, and H. Wang, “Information-agnostic flow scheduling for commodity data centers,” in *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. Oakland, CA: USENIX Association, May 2015, pp. 455–468. [Online]. Available: <https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/bai>
- [110] M. Mahalingam, D. G. Dutt, K. Duda, P. Agarwal, L. Kreeger, T. Sridhar, M. Bursell, and C. Wright, “Virtual extensible local area network (vxlan): A framework for overlaying virtualized layer 2 networks over layer 3 networks.” *RFC*, vol. 7348, pp. 1–22, 2014.