Toward Improving the Performance of Scientific and Desktop/Edge Applications

---

A Dissertation

Presented to

the Faculty of the School of Engineering and Applied Science

University of Virginia

In Partial Fulfillment

of the requirements for the Degree

Doctor of Philosophy (Computer Engineering)

by

Fatma Alali

May 2019

Approval Sheet

This dissertation is submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy (Computer Engineering)

Fatma Alali

This dissertation has been read and approved by the Examining Committee:

Prof. Malathi Veeraraghavan, Adviser

Prof. Alfred C. Weaver, Committee Chair

Prof. Ronald D. Williams

Prof. Ahmed Ibrahim

Prof. Rider W. Foley

Accepted for the School of Engineering and Applied Science:

Prof. Craig H. Benson, Dean, School of Engineering and Applied Science

May 2019

To my mother, Ghanima.

Abstract

The key protocols, TCP and IP, underlying the Internet were invented and introduced into ARPAnet, a precursor to the Internet, in the 1970s. The very success of these protocols has constrained the introduction of new high-throughput, low-latency and/or scheduled-delivery network services. While many green-field network designs have proven to be better suited for these types of high-performance services, these designs have been difficult to deploy incrementally into the Internet. Therefore, in this study, *we designed and evaluated new networking services, taking into account deployment constraints, so that these services can be introduced incrementally into different regions of the Internet for improved application performance.* This is an evolutionary approach to enabling services on deployed networks to improve application performance rather than a revolutionary "design-a-new-network" approach.

Given the large number of network technologies and even larger number of deployed networks, we selected networks within the following three categories: (i) datacenter networks, (ii) Wide-Area Networks (WANs), and (iii) Local-Area Networks (LANs). For each network type, we defined problems that address specific application needs, and proposed and evaluated our evolutionary solutions. For the datacenter networks, we tackled the problem of measuring congestion in InfiniBand production clusters, where congestion is known to reduce the performance of parallelized applications. For WANs, we proposed new network services to support high throughput for large data transfers, and scheduled delivery for delay-sensitive transfers. Finally, for LANs, we focused on the performance of Virtual Desktop (VD) applications. New methodologies are needed to evaluate advances in VD technologies, the results of which would allow for better engineering of VD deployments for improved

application performance.

In datacenter networks, low-latency communications are required for scientific, highly parallelized applications. Furthermore, predictable execution times are essential for workflow management. Since HPC clusters deployed by the scientific community use InfiniBand networks, our study targets these networks. Network congestion has been identified as one of the main reasons for performance variability of highly parallelized applications. Characterizing network congestion events will help network administrators identify bottlenecks, and accordingly deploy congestion-control solutions. We developed a methodology for measuring congestion and executed this methodology in a production, highly utilized, InfiniBand cluster called Yellowstone, in which congestion control is currently disabled due to a lack of proven techniques for countering congestion.

Methods for achieving high throughput across WANs are necessary for decreasing transfer times of large datasets. WAN provider links are often operated at low utilization levels, which leaves large unused capacity (headroom). Leveraging this observation, we propose using Software Defined Networking (SDN) controllers to support novel Static Headroom (SH) and Dynamic Headroom (DH) services. These services allow customers to fill the headroom and achieve high throughput without adversely affecting the provider's ability to meet its Best-Effort (BE) service-level agreements. Our solution calls for the use of lower-priority service for large data transfers, and is designed to operate in currently deployed networks with minimal changes.

To allow scheduled delivery of large datasets across WANs, we developed Calibers: Calendar and Large- scale Bandwidth Event-driven Simulations. Calibers leverages SDN-based network architectures and flow pacing algorithms. It uses techniques to intelligently and dynamically shape flows to maximize the number of flows that meet their deadlines while simultaneously improving network resource utilization.

LAN connections are used to offer users VDs from edge-cloud computers to user-owned I/O devices. Zero clients are custom, hardware units with no CPUs, which are designed to enable high-performance delivery of VDs to user-owned I/O devices. We measured the performance of VDs accessed through zero clients to study the feasibility of using this solution to provide desktop-PC experience for disadvantaged communities as a part of a

smart-city service. Current deployments of such services are constrained by the inability to run monitoring software packages at the zero clients to assess performance. Therefore, we introduced a new metric and methodology to measure VD performance based on analyzing network traffic, and conducted objective and subjective studies to explore the feasibility of such a solution to provide users a desktop-PC experience from edge clouds.

Acknowledgments

I would like to take this opportunity to thank many people without whom this research work would have not been possible. I would like to express my sincerest gratitude to my advisor Professor Malathi Veeraraghavan for her support and guidance. She dedicated her time for our success and always has faith in us.

My very profound gratitude to my greatest supporters, my family; my parents Ghanima and Hilal; my sisters: Aisha, Amina, Maryam, Asmaa, and their kids; and my brothers: Musab, Omar, and Abdullah. They all believed in me, supported me, and always been there for me. Their surprise-visits through the five years I spent in the program have always left me with joy and happiness. I am very grateful for the vacation days and long trips they took to spend a couple of days with me. My deepest gratitude is to my mother for her continuous prayers, support, and love.

I also cannot thank enough my friends — who lived in Charlottesville and who lived away in Kuwait — Deema, Rawan, Shaikha, Nadia, Asma, Maria, Ekram, and Ladan. I am very grateful to my cousin Latifa — who started the PhD journey with me— and her family for opening their home for me, for their hospitality, and being there whenever I struggled or felt overwhelmed by the graduate-student life. The cakes we ate, the TV shows we watched, and the funny stories we shared during the weekend nights have always helped to start the week with a positive mindset.

I would also like to thank my committee members, Prof. Ronald Williams, Prof. Alf Weaver, Prof. Rider Foley, and Prof. Ahmed Ibrahim for taking the time to serve on my proposal and defense committee and provide great suggestions, and feedback.

Many thanks to my fellow graduate students, Sherry, Sourav, Yizhe, Yuanlong, Xiang, Shuoshuo Reza, Elahe, and Fabrice, for their support, kind words, feedback, and friendship.

This work was supported by NSF grants CNS-1737453, ACI-1340910, CNS-1405171, CNS-1531065, CNS-1624676, and CNS 1528087; and the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, and Kuwait University.

Contents

List of Tables

List of Figures

List of Abbreviations

ACR	Absolute Category Rating
ALS	Advanced Light source
AQ	Audio Quality
ASICs	Application Specific Integrated Circuits
BE	Best-Effort
BECN	Backward ECN
CAIDA	Center for Applied Internet Data Analysis
CFDS	Centralized Filesystems and Data Storage
CIR	Committed Information Rate
CS	Class Selector
CSV	Comma Separated Values
CV	Coefficient of Variation
DAV	Data Analysis and Visualization
DSCP	Differentiated Services Code Point
ECN	Explicit Congestion Notification
EFs	Elephant Flows
FCP	Flow Control Packet
FDR	Fourteen Data Rate
FECN	Forward ECN

FITF	Forced Idle Time Fraction
FPGAs	Field Programmable Gate Arrays
HCAs	Host Channel Adapters
HOL	Head-of-Line
HPC	High Performance Computing
HSD	Honestly Significant Difference
ISPs	Internet Service Providers
LANs	Local-Area Network
LHC	Large Hadron Collider
LIDs	Local Identifiers
LJF	Longest Job First
LLR	Log-Likelihood Ratio
LMM	Linear Mixed effect Model
MOS	Mean Opinion Score
MPI	Message Passing Interface
NERSC	National Energy Research Scientific Computing Center
NSD	Network Shared Disk
OSCARS	On-Demand Secure Circuits and Advance Reservation System
PCoIP	PC over IP
PESQ	Perceptual Evaluation of Speech Quality
PIR	Peak Information Rate

PLR	packet loss rate
POLQ	Perceptual Objective Listening Quality
PPBP	Poisson Pareto Burst Process
QoE	Quality of Experience
RDMA	Remote Direct Memory Access
RDP	Remote Desktop Protocol
recv-PCoIP-fps	received PCoIP frame per second
RT	Response Time
RTT	Round-Trip Time
SDN	Software Defined Network
SJF	Shortest Job First
SLAs	Service-Level Agreements
SSV	session statistics viewer
TE	Traffic Engineering
ToR	Top-of-Rack
VD	Virtual Desktop
VD-DUT	Virtual Desktop Display Update Time
ViSQOL	Virtual Speech Quality Objective Listener
VM	Virtual Machine
VOQ	Virtual Output Queue
VQ	Video Quality
WANs	Wide-Area Networks

WSS Weighted Spectral Slope

Chapter 1

Introduction

This chapter gives a high-level overview of the research work presented in this dissertation. Section 1.1 provides the high-level background and motivation for the work. Section 1.2 defines the high-level problem statement, solution approach and hypothesis. The next four sections discuss problems that address specific application needs in the context of datacenter networks (Section 1.3), Wide-Area Networks (WANs) (Sections 1.4 and 1.5), and Local-Area Networks (LANs) (Section 1.6). The key contributions are listed in Section 1.7, and Section 1.8 presents the overall layout of the dissertation.

1.1 High-level background and motivation

The key protocols, TCP and IP, underlying the Internet were invented and introduced into ARPAnet, a precursor to the Internet, in the 1970s. Changes to these key protocols were implemented during the early development of the network in response to immediate pressing problems. For example, congestion control was developed due to the congestion collapses in the mid 1980s [6]. Other changes include addition of Domain Name System (DNS) and the deployment of the inter-domain routing protocol, Border Gateway Protocol (BGP) [7]. Large-scale changes were only feasible during the early stages. For instance, the reliable-transport feature was separated from the Internet Protocol (IP) and implemented in TCP. This change to the core network occurred in one day in 1983 where four hundred

nodes were switched to use TCP/IP. That was the last time such a switchover happened in one day [7].

No significant changes to the core protocols have occurred since 1993 [7] as the network grew in size and deployment of new protocols became difficult. Attempts have been made, but have not resulted in large-scale deployments. For example, Explicit Congestion Notification (ECN) was standardized, but it is not widely deployed; IPv6 and related protocols have been developed over the last two decades, but the current global IPv6 traffic remains low [8, 9]. The IPv6 traffic share at major Internet eXchange Points (IXPs) was reported to range between 1-2% in a 2017 study [8]. Google March 2019 statistics [10] show that 22.57% of user accesses to Google are via IPv6 addresses, and 30 countries have an IPv6 adoption rate greater than 15%. These studies indicate that a long technology-adoption phase was needed as IPv6 was standardized in the late 1990s.

Besides making core changes to the currently deployed network, other work targeted challenges, such as the introduction of Quality-of-Service (QoS) guaranteed delivery by designing a new networking technology called Asynchronous Transfer Mode (ATM). While ATM and other green-field network designs proved to be better suited for newer types of network services, these designs have been difficult to deploy incrementally into the Internet. For example, given the global scale of the Internet with its IP routers, it was not cost effective to switch over to ATM networking.

Despite these previous failures to incorporate new networking technologies into the Internet, in 2010, the National Science Foundation (NSF) launched a Future Internet Architectures (FIA) research program [11]. The concept of Named Data Network (NDN) was introduced by one of the FIA projects [12]. Ever since, many research studies have been conducted to develop solutions to support NDN networks.

In addition, there have been ongoing conversations within the networking research community about whether to support the development of clean-slate designs or incremental evolutionary approaches [13]. Our work does not compare these approaches; rather it just takes the second approach of designing solutions to improve existing network services while considering constraints imposed by the current deployment.

1.2 High-level problem statement and hypothesis

The problem statement of this study was to design and evaluate new networking services, while taking into account deployment constraints, so that these new services can be introduced incrementally into the Internet for improved application performance. This is an evolutionary approach to enabling services on deployed networks to improve application performance rather than a revolutionary green-slate "design-a-new-network" approach.

A general solution approach to tackle this problem consists of the following steps:

1. develop new metrics and methods if needed, and collect measurements to quantify application performance in deployed networks,
2. design and evaluate solutions in experimental, emulated, or simulated environments to improve application performance, and
3. finally, deploy the designed solutions.

In this PhD research, depending on the context, we focused on step 1 or 2 of this general solution approach, as step 3 is out-of-scope.

The overall *hypothesis* of this research is as follows: under low to moderate loads, it is feasible to introduce new network services with an evolutionary approach that can improve the performance of applications. This hypothesis is tested in different contexts, each of which addresses a specific application need for improved performance.

Given the large number of network technologies and even larger number of deployed networks, we selected four examples in the context of the following three types of networks: (i) datacenter networks, (ii) Wide-Area Networks (WANs), and (iii) Local-Area Network (LANs). Section 1.3 identifies an application problem in the usage of InfiniBand datacenter networks, and presents our solution for collecting measurements to quantify the extent of the problem (step 1 of the solution approach). Section 1.4 presents two new high-throughput services to support large data transfers across WANs; and Section 1.5 offers a solution for scheduled delivery across WANs (step 2 of the solution approach). Section 1.6 identifies a problem with how to collect measurements that quantify the performance of virtual-desktop services across LANs, and offers a solution (step 1 of the solution approach).

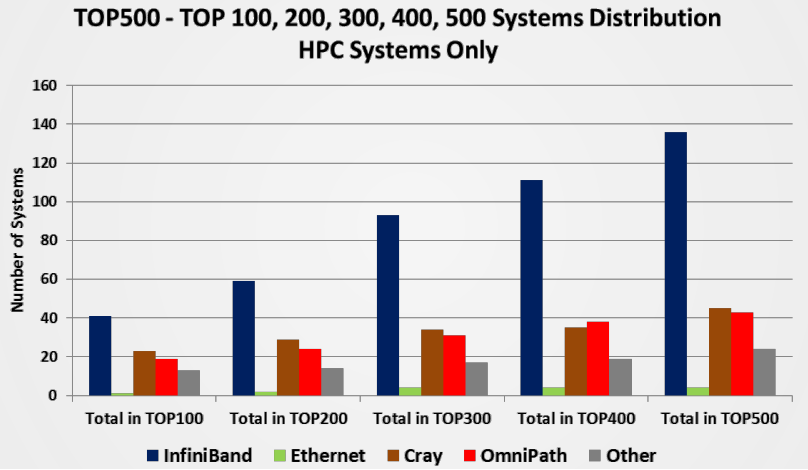


Figure 1.1: Cluster interconnect of the Top500 list including only HPC Systems (excluding Cloud, Hyperscale, and other types of systems) [1]

1.3 Datacenter network: InfiniBand measurement study

InfiniBand is a high-bandwidth, low-latency packet-switched networking technology designed for data centers and High Performance Computing (HPC) clusters. InfiniBand HPC clusters are widely deployed. In November 2018, InfiniBand was reported as the cluster interconnect of choice in 27% of the Top500 clusters [14], where the top three fastest systems use InfiniBand technology. Fig. 1.1 shows the interconnect of choice for HPC clusters in the Top500 list, where InfiniBand is seen to be used in a majority of the Top500 HPC systems.

After describing how InfiniBand achieves low latency, and why low latency and low execution-time variability are important to scientific applications, we describe how InfiniBand performs flow control and congestion control, both of which contribute to execution-time variability.

InfiniBand achieves low latency because the protocol layers from the transport layer to the physical layer are implemented in hardware. The transport layer is implemented in hardware in the Host Channel Adapters (HCAs) at the end nodes, while the other lower layers are implemented in all HCAs and switches. The features that specifically enable low-latency communications include: (i) `zero-copy`: allows memory-to-memory direct transfers without software based copying in end-host memory; (ii) `small switch buffers`: packets experience smaller queueing delays at switches, but packets can be held up at the sending

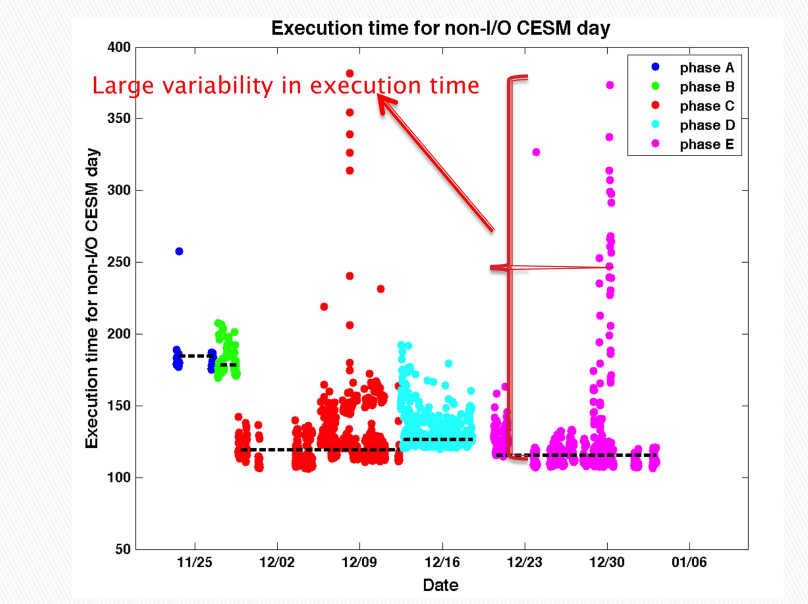


Figure 1.2: Execution time variation for a climate application running on an InfiniBand HPC cluster

ends of links if there are insufficient flow-control credits from the receiving ends; `message passing`: there is no concept of byte streams, and therefore, no overhead is incurred in tracking message boundaries within streams; and `kernel bypass`: allows user-level access to the HCA with no operating-system interrupts during data transfers.

Low latency and low execution-time variability are important to scientific applications, such as climate models. These applications run in HPC clusters and are highly parallelized to execute on thousand or more cores. Message Passing Interface (MPI) is the primary software library by which HPC applications portably pass messages between processes of a parallel program. MPI calls, such as MPI_Barrier or MPI_Waitall, are used for synchronization of the processing actions performed by different processes. When a process reaches a synchronization call, it has to wait for all the other processes to complete before proceeding. Such stalls in processing can noticeably increase the total execution time of an application.

Fig. 1.2 shows an example of execution time variability for the Community Earth System Model (CEMS) running on an HPC cluster. The high variability in execution time leads to unpredictable performance. When submitting a job to an HPC cluster, the user needs to estimate and specify the time required to run the job. If the job takes longer than the estimated time, the cluster scheduler can terminate the job before the job completes. On

the other hand, overestimating a job execution time degrades the cluster utilization and scheduler efficiency.

InfiniBand uses link-by-link flow control to avoid losses in switch buffers. A switch port is not allowed to transmit packets unless the corresponding receiving port has sufficient buffer space. The receiving port sends a Flow Control Packet (FCP) indicating how much space is left in its buffer. If the receiving buffer is full, the transmitting port has to wait until it receives an FCP from the receiver. While this mechanism avoids the end-to-end delays incurred with retransmissions in TCP, InfiniBand's link-by-link flow control mechanism could cause congestion to spread in the network, which in turn, could impact application performance.

Network congestion in HPC clusters has been identified as one of the main reasons for performance variability of highly parallelized applications [15, 16]. In order to improve the performance of MPI applications, achieve predictable performance, and increase cluster utilization, solutions are needed to address congestion. Following the high-level solution approach described in Section 1.2, measurements are needed to quantify congestion events in production HPC clusters and methods for handling congestion are required if the measurements show congestion to be an issue. In our research group, a Dynamic Congestion Management System (DCMS) [4] was proposed to avoid network congestion spread through a network to prevent victim flows (flows not contributing to the congestion event) from experiencing reduced throughput. When DCMS detects victim flows, it aggressively reduces the sending rates of congestion-contributing flows to allow victim flows to pass through congested ports. However, a methodology is needed to measure congestion to determine the extent of congestion in deployed InfiniBand networks (step 1 of the high-level solution approach). In most InfiniBand HPC clusters, congestion control has beed disabled because of its complexity and the lack of a proven study that provides guidance on how to set congestion-control parameters [17]. Hence, for this work, we answer the question of how to measure and characterize congestion events in InfiniBand HPC clusters given the deployment constraint of disabled congestion control.

1.4 WAN: High-throughput bulk data transfers

Large datasets are generated every year at an exponentially increasing rate. For example, the A Toroidal LHC ApparatuS (ATLAS) experiment, conducted on the Large Hadron Collider (LHC), located near Geneva, Switzerland, generates a 100-MB dataset each second, which adds up to about 1 PB each year [18]. The European Bioinformatics Institute (EBI) in the UK, which is one of the world's largest biology-data repositories (include genes, proteins, and small molecules datasets) store more than 200 PB of raw data and backups [19]. A total of 3.5 PB were downloaded from EBI datasets in 2016 [20]. These large datasets are transferred to supercomputing facilities and laboratories across the world for processing and analysis. Data backups are another source of large WAN transfers.

To accommodate the long-term growth in data volume, Internet Service Providers (ISPs) upgrade the capacity of their network links to maintain an average utilization level of 30%-40% [21]. ISP customers are limited to the sending rate specified in their access-link Service-Level Agreements (SLAs). For example, a customer may pay connectivity charges for a 1 GE port on the provider network, but have an SLA with Committed Information Rate (CIR) set to 200 Mbps and Peak Information Rate (PIR) set to 600 Mbps. The customer could have occasional, delay-tolerant large data transfers, e.g., backups. However, the customer would be limited to 600-Mbps throughput, while simultaneously intra-domain provider links would remain underutilized.

This work answers the question of how to allow WAN delay-tolerant transfers to achieve high throughput while allowing providers to leverage their underutilized links. By following the general solution approach outlined in Section 1.2, we first analyzed network traces collected by CAIDA [5] on several backbone links. Our data analysis showed that links are underutilized. We also ran simulations to determine whether occasional bulk data transfers that fill underutilized links could have adverse effects on background traffic, and found this to be the case. But with traffic engineering, i.e., controlling the paths taken by bulk transfers, the adverse impact on background traffic can be limited. Therefore, we designed headroom services that use traffic engineering to allow occasional, delay-tolerant data transfers to reach high throughput without impacting best-effort flows.

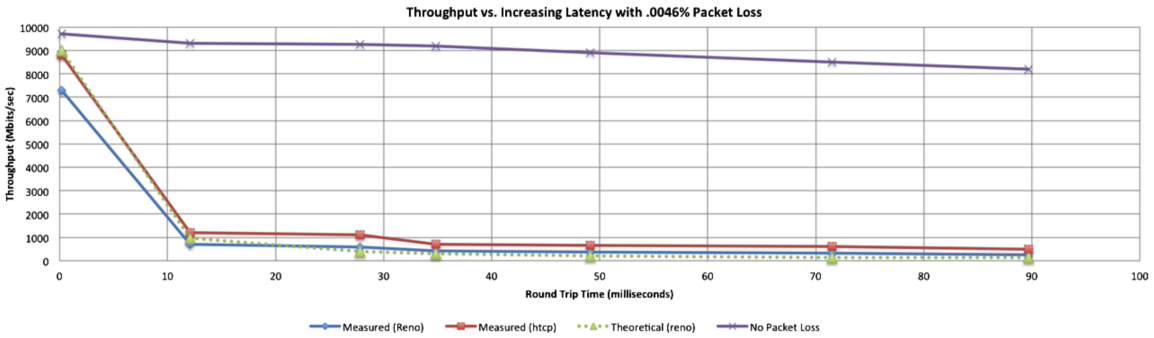


Figure 1.3: Achieved throughput with and without packet loss across different delays with 10 Gbps link [2]

1.5 WAN: Delay-sensitive data transfers

In contrast to delay-tolerant data transfers, delay-sensitive data transfers are required for another set of applications. Delay-sensitive data transfers require scheduled delivery, which is not offered by the best-effort IP services of the Internet. For example, researchers reserve computing resources in HPC systems for data analysis. Without scheduled-delivery services, it is difficult to estimate the time needed to transfer datasets from storage systems to the HPC sites. If a researcher overestimates this time, then the HPC cluster would be underutilized, and computing resources would be wasted. A study on four production systems (Intrepid at Argonne National Laboratory (ANL), Blue Horizon at the San Diego Supercomputing Center (SDSC), IBM SP2 at the Cornell Theory Center (CTC) and Linux cluster at High Performance Computing Center North, Sweden (HPC2N)) showed that on all four systems, it is common for a significant percentage of system resources (20% to 80% of nodes) to be idle while there are other jobs waiting in the queue. Such underutilization occurs because users overestimate job completion times [22].

TCP is the standard protocol used for reliable bulk-data transfers. TCP's typical "sawtooth" behavior, which results from the TCP sender increasing and decreasing its sending rate based on estimated path congestion, leads to a lack of predictability in meeting deadlines. Fig. 1.3 shows the achieved throughput of a transferred file under different network conditions. With no packet loss, the transfer experienced a throughput of 8 Gbps on a path with bottleneck link capacity of 10 Gbps and Round-Trip Time (RTT) of 90 ms. With a packet loss rate of 0.0046% and an RTT higher than 10 ms, the transfer achieved a

throughput around 1 Gbps, even with the HTCP transport protocol which is designed for paths with high bandwidth-delay products [23].

Another feature of TCP, fair sharing of link capacity, makes it difficult to support scheduled-delivery services on the Internet. This is because, given the goal of fair sharing, no single flow can be assigned a higher fraction of link capacity than other flows, and there is no admission control to limit the number of concurrent flows.

To answer the question of how to support scheduled delivery for transfers while maintaining high network utilization, we followed the general solution approach of Section 1.2. First, we collected measurements on the Energy Sciences Network (ESnet) testbed [24] by running experiments. The experimental study was conducted by our collaborators at the University of California, Davis, and ESnet. The testbed closely resembles ESnet's high-speed production network in both hardware and topology, as it is an overlay on the ESnet production WAN. An experiment was conducted with two transfers: (i) 5 GB file over a long RTT path, with a critical delivery window of 2-min, and (ii) 10 GB file over a shorter RTT path with a less strict deadline. Both file transfers shared a 500 Mbps bottleneck link. With HTCP's "fair" operation configured, the flows should complete in roughly the same time, as the differences in latency counteract the differences in file size. However, if the critical flow was paced to (assigned a sending rate of) 380 Mbps and the non-critical flow was assigned 120 Mbps, the critical flow will complete within its deadline, the available bottleneck bandwidth will then be relinquished, and the non-critical flow will still complete. Fig. 1.4 shows the results of the experiment. The critical deadline for the 5 GB file is shown with a vertical dashed line. Results for both the paced and unpaced scenarios of the experiment are shown. In both cases, the 5 GB and 10 GB files are transmitted simultaneously. In the unpaced case, much of the link capacity is wasted initially as the two hosts, located thousands of miles from the bottleneck, attempt to negotiate their equal share of the link. In the paced case, however, the link sharing is efficient and the critical-flow reached its destination within its deadline.

This experimental study addressed the first step (collecting measurements) in our general solution approach. In this research, we addressed the second step in our general solution approach, which is to design and evaluate a solution for offering a scheduled-delivery network service. For this new service, we developed scheduling algorithms to support deadline-based

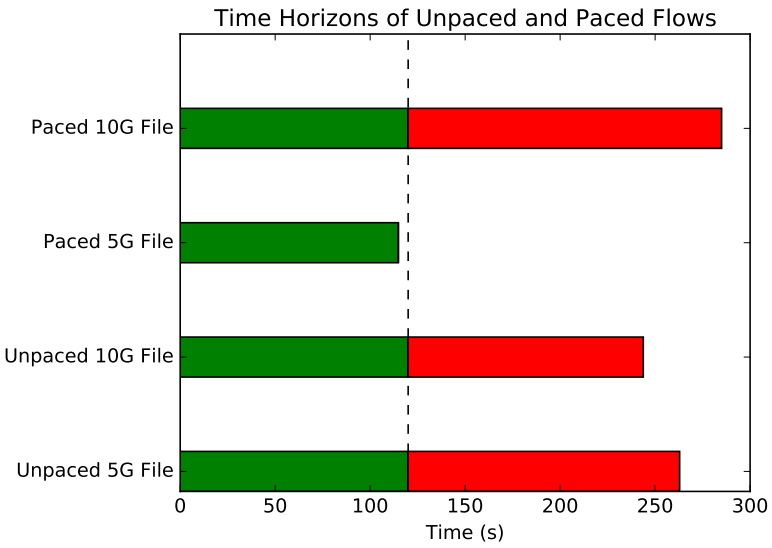


Figure 1.4: Flow completion times of two file transfers sharing a bottleneck link. The vertical line is the deadline for the 5 GB file [3]

delivery of files while maximizing network utilization. The solution works with current deployed network by leveraging Linux systems and router pacing algorithms that allow for the assignment of a specific rate to each flow. The proposed algorithms were evaluated with simulations.

1.6 LAN: Virtual desktop measurement study

Virtual Desktop (VD) technologies have been developed in the past ten years. This computing paradigm includes four main parts: (i) an edge-cloud server to host the virtual desktops, (ii) an end-user device to access a virtual desktop, (iii) a LAN connecting a virtual desktop to the end-user device, and (iv) a remote desktop protocol to deliver the display content to the end-user video monitor, and receive keyboard and mouse input from the corresponding end-user devices. End-user devices could vary from thin clients to custom hardware units, called zero clients, which run remote desktop protocols, encryption and video decompression. Zero clients typically do not have general-purpose processors or operating systems, making them less vulnerable to cyberattacks. Latency considerations for VD applications made us focus on offering VD services at edge-cloud servers across LANs.

Zero clients have been in limited use today. Most zero-client deployments are in the health, business, or education sectors, where a specific set of applications are expected to be used. The use of zero clients is appealing for the potential of these devices to deliver high-performance computing experiences at favorable costs. Zero clients, with shared computing resources at servers, could be used to provide computing services to disadvantaged communities. These services can be provided as a part of a smart-city deployment as an effort to achieve community social inclusion. However, application performance when using zero clients and VD service has not been characterized before in experimental studies.

To address this question of how to design VD services over existing LANs (limited by technology and deployment constraints) to improve desktop application performance, we followed the general solution approach described in Section 1.2. In this work, we addressed step 1 of the general solution approach, i.e., measuring the performance of VD applications. Measuring the performance of VD applications delivered through zero clients is challenging. This is because zero clients do not include general-purpose CPUs, typically required for the installation and execution of performance-monitoring software packages. Running monitoring software packages at the edge-cloud servers alone is insufficient for accurate user-perceived performance measurement. Nevertheless, most of the computing tasks are performed on the edge-cloud server, where display pixels are encoded and sent to end-user devices over a LAN. Performance degradation could occur when the display-update pixels travel through the LAN. For example, Casas et al. [25] showed that a virtual desktop provided by XenDesktop and accessed by Citrix client running on a laptop suffered from an increase by 5x in the response time to change the display image when RTT increased to 50ms, indicating a significant network impact on performance. The Casas et al. study was conducted with laptops used as the end-user devices, which allowed for the execution of performance monitoring software. Therefore, methods are needed to quantify VD-application performance, while considering the constraints posed by the processor-less zero clients. Based on our measurements, we identified a need to carry out step-2 of the general solution approach, which is to develop methods to improve the design of VD services in order to offer users high-quality application performance at affordable costs.

1.7 Key Contributions

The key contributions of this work are as follows.

1. We developed a methodology for measuring congestion and executed this methodology in a production, highly utilized, InfiniBand cluster called Yellowstone. This work was published in the IEEE Network Traffic Measurement and Analysis Conference (TMA) [26].

2. We proposed two new Internet provider services to allow high-throughput large transfers: Static Headroom (SH) service, and Dynamic Headroom (DH) service. Large-transfers packets using these headroom services are tagged with a low-priority Class Selector (CS) to avoid adverse effects on other flows. This work was published in the IEEE 23rd Asia-Pacific Conference on Communications (APCC) [27].

3. We developed Calibers: Calendar and Large-scale Bandwidth Event-driven Simulation, for deadline-specific scientific large-data transfers. We proposed four heuristic algorithms and compared their performance. This work was published in the IEEE/ACM Innovating the Network for Data-Intensive Science (INDIS) workshop [3]. The work was then extended and published in a Elsevier Future Generation Computer System (FGSC) journal paper [28].

4. We conducted objective and subjective studies on desktop/edge computing for smart cities and communities. A new objective performance metric, Virtual Desktop Display Update Time (VD-DUT), was proposed. This work includes the first subjective, large-scale study on zero-client performance, where subjective measurements were correlated to objective measurements. This work is in preparation to be submitted to IEEE Access journal.

1.8 Dissertation Organization

This dissertation is organized into six chapters. Background, motivation, problem statement, and a summary of the key contributions are provided in this chapter.

Chapter 2 presents a measurement-based study on network congestion conducted in a production, highly utilized, InfiniBand 72-K core cluster called Yellowstone. A methodology was developed to measure congestion events in clusters with disabled congestion control. This methodology was used to characterize congestion events in Yellowstone, and our results and findings are presented and discussed.

Chapter 3 presents the design of static and dynamic SDN-enabled headroom services. A simulation study was conducted to compare the two services. Results, analysis, and advantages of these services are discussed in this chapter.

Chapter 4 describes the Calibers architecture and scheduling algorithms. Four heuristic-based algorithms are proposed, and a simulation study is presented in this chapter, along with results and analysis.

Chapter 5 presents objective and subjective studies on application performance over virtual-desktop services offered with edge-cloud servers and zero clients at user locations. The chapter describes the methodology used for each study and quantifies the correlation between the objective and subjective measurements.

Chapter 6 summarizes our work, discusses potential future work, and concludes the dissertation.

Chapter 2

A Measurement Study of Congestion in an InfiniBand Network

2.1 Introduction

InfiniBand was reported as the cluster interconnect of choice used in the majority of the Top500 HPC systems [1]. InfiniBand is a switched networking technology that is designed for lossless, low-latency communications. For highly parallelized MPI applications that are executed on HPC systems, communication delays can become the key determinant of application execution time when computing cores are not limited.

The InfiniBand link-by-link flow control, which is effective in preventing losses in switch buffers, has an insidious side effect of causing congestion to spread in the network. When an output port P1 of a switch becomes congested, the input-side buffer of another port P2 of the same switch could fill up. This will cause the port of the upstream switch connected to port P2 to be denied flow-control credits, thus effectively reducing the rate of the upstream port. Such a port is referred to as a "victim port" and flows passing through victim ports become "victim flows" (bulk-data flows suffer reduced throughput, and short-messages suffer increased delays).

Studies have shown that network effects, e.g., congestion, can increase variability of the total execution time for large core-count applications [15]. Therefore, many research papers, [4, 17, 29–36], have modeled, analyzed, and proposed solutions for InfiniBand congestion control, and evaluated these solutions using simulations or experiments on small testbeds.

In this chapter, we report on a measurement study of congestion in a production HPC cluster. To the best of our knowledge, no such measurement study has been reported in prior work. It is challenging to characterize congestion events because supercomputing centers typically disable congestion control in their InfiniBand clusters. The reason cited for disabling congestion control is that there is no proven study that provides guidance on how to set congestion-control parameters [17]. Therefore, a measurement study of congestion in a production network is not easy.

Specifically, our measurement-based study of congestion was carried out on a highly utilized 72K-core machine called Yellowstone. The Yellowstone network is a fat-tree topology, which consists of Top-of-Rack (ToR) switches, leaf switches, and spine switches. In addition to the main cluster of compute nodes, there is a disk I/O subsystem, and a data analysis and visualization subsystem. A methodology based on observing a port counter called `PortXmitWait` is proposed. For data analysis, a new metric called Forced Idle Time Fraction (FITF) is defined.

Our key contributions are as follows. (i) Our methodology and software offers a means for network administrators to obtain a conservative gauge of the level of congestion in a production network on which congestion control is disabled. (ii) We expected congestion to be predominantly in the disk I/O system, but found that ports in the compute-node cluster also suffered from congestion. In about 60% of the 100-ms intervals in which ToR-switch ports were observed, the ports were stalled waiting for flow-control credits. While in most 100-ms intervals, a port was denied flow-control credits for less than 10% of the interval, there were some instances in which a port was denied credit for significant portions (in the range 60-80%) of the 100-ms interval, with several intervals reaching 100%. Such long stalls in data transmission can impact the completion time of one or more MPI ranks adversely, which can increase execution time of highly parallelized communication-intensive applications that have `MPI_Barrier` calls for synchronizing MPI ranks. The material presented in this chapter

is an excerpt from our published work *A measurement study of congestion in an InfiniBand network* [26]

Section 2.2 provides background information on InfiniBand and Yellowstone. Section 2.3 describes our measurement study (methodology, data analysis, and metrics). The numerical results are presented in Section 2.4, and the impact of our findings is discussed in section 2.5. Section 2.6 reviews related work, and the chapter is concluded in Section 2.7.

2.2 Background

This section provides background information on InfiniBand networks, and describes the Yellowstone system.

2.2.1 InfiniBand

InfiniBand is a packet-switched networking technology designed for high-speed low-latency operation. Packet headers carry 16-bit source and destination Local Identifiers (LIDs), and packets are forwarded by switches with a destination-LID-based table lookup. A centralized subnet manager computes and downloads forwarding tables to the switches. To avoid packet loss, a link-by-link flow-control scheme is used. A transport-layer congestion-control mechanism includes actions at switches, receiving hosts, and sending hosts.

As mentioned in Section 1.3, in the link-layer protocol, a transmitter (switch port or host port) is not allowed to send out packets unless the corresponding receiving port has sufficient buffer space. The receiving port sends a FCP indicating how much space is left in its buffer. If the receiving buffer is full, the transmitter has to wait until it receives an FCP from the receiver.

Congestion control in InfiniBand is based on Explicit Congestion Notification (ECN). The mechanism used to detect congestion at a switch port is not defined in the InfiniBand specification, but rather, it is left up to the vendors. When congestion is detected on a port P, packets transmitted out on port P are marked by setting the Forward ECN (FECN) bit in the transport-layer header. The rate at which the packets are marked is controlled by a configurable parameter called the `Marking_Rate`. When a receiving host receives marked

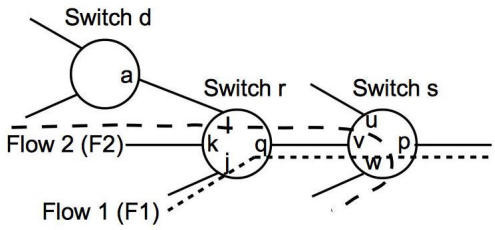


Figure 2.1: Illustrative InfiniBand Network [4]

packets for a flow, it sets a Backward ECN (BECN) bit in the acknowledgment (ACK) or other packets that are being sent in the opposite direction. When the sending host receives a BECN-marked packet for a particular flow, the sender reduces its sending rate according to a mechanism that dynamically adjusts inter-packet injection delay.

To understand how the link-by-link flow control mechanism causes the effects of a port's congestion to spread even to ports that have no shared flows with the congested port, consider the example shown in Fig. 2.1 [4]. Assume that port p of switch s becomes congested because the aggregate incoming rate of packets destined to port p exceeds the port capacity. This can happen when multiple high-throughput transfers (e.g., disk read/write and checkpointing) destined to the same switch port occur concurrently, as demonstrated in experiments in our prior work [4]. Now, assume flow F1 traverses ports j and q of switch r and ports v and p of switch s. When port p gets congested, the buffer on the incoming side of port v of switch s will start to fill up with flow-F1 packets. When this buffer fills up, the rate at which FCPs are generated by port v of switch s to port q of switch r, to offer the latter credits for packet transmission, will decrease. Effectively, the rate of port q of switch r is lowered. Now, consider flow F2, which traverses ports k and q of switch r and ports v and w of switch s. The rate of F2 will be reduced even though this flow does not traverse the congested port p. Flow F2 is a victim flow, and port q of switch r is a victim port. This example illustrates how the presence of the link-by-link flow control algorithm causes the effects of a congested port to spread to other parts of the network.

This type of spreading of the effects of a congested port does not occur in IP/Ethernet networks. This is because in IP/Ethernet networks, when a switch buffer is full, packets are simply dropped. There is no credit-based flow control mechanism, and therefore a link transmitter can freely send packets to a link receiver causing a buffer to overflow. This

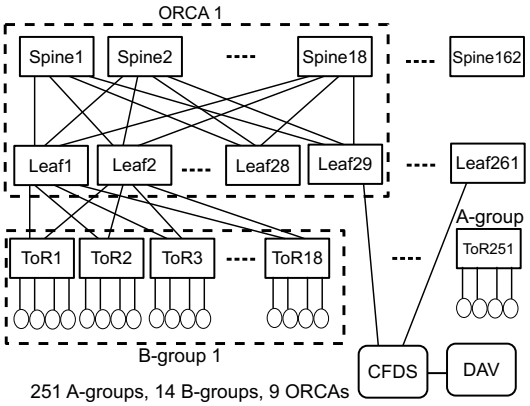


Figure 2.2: Yellowstone topology

approach has the advantage of not creating victim ports. Consider the example of flows F1 and F2 shown in Fig. 2.1. If the switches were Ethernet-based or IP routers, when port p of switch s becomes congested, its buffer will overflow, which causes F1 packets to be dropped. Flow F2, which is not destined to the congested port p, will not experience any dropped packets and hence will be unaffected, unlike in the InfiniBand network. However, if flow F2 also used port p, it would suffer packet losses. Therefore, there are victim flows in TCP/IP networks but not victim ports; the latter is worse as flows that do not even traverse the congested port become victimized.

Therefore, the work presented here is only relevant to InfiniBand-based HPC systems, which as noted in Section 2.1, is deployed widely in academic, research and government institutions. Commercial datacenters typically use Ethernet-based interconnects. The reason for this difference is that parallel programs written for scientific research typically use MPI and require low-latency communications, while commercial datacenters typically support applications that use TCP sockets for communications.

2.2.2 Yellowstone

Yellowstone is a high-performance IBM iDataPlex cluster, where the network interconnect is a Mellanox InfiniBand full fat-tree. All links carry 4-lane Fourteen Data Rate (FDR) (56 Gbps) signals.

Fig. 2.2 shows the Yellowstone topology. Each ToR switch has 36 ports, 18 of which are

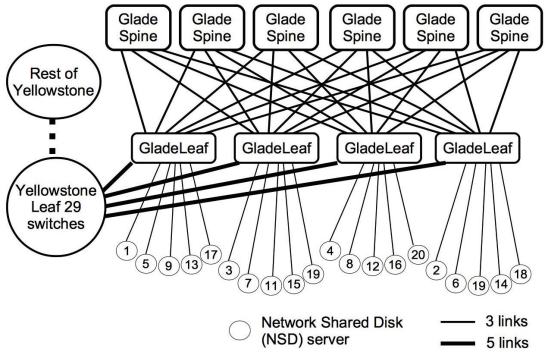


Figure 2.3: Glade subsystem topology

connected to compute hosts, and the remaining 18 are connected to leaf switches in ORCA[1] racks. Each ToR switch with its 18 compute hosts is called an A-group. The term B-group is used to represent a group of 18 A-groups. All ToR switches in a B-group are connected to the same two leaf switches of an ORCA rack, which consists of 29 leaf switches and 18 spine switches. For example, ToR1 to ToR18 of B-group 1 are connected to Leaf 1 and Leaf 2 of ORCA 1, Leaf 30 and Leaf 31 of ORCA 2, etc. In other words, all ToR switches of B-group 1 are connected to the first two leaf switches of each of the 9 ORCAs. Similarly, ToR19 to ToR36 of B-group 2 are connected to Leaf 3 and Leaf 4 of ORCA 1, Leaf 32 and Leaf 33 of ORCA 2, etc. Each leaf switch also has 36 ports, 18 of which are connected via downlinks to ToR switches, and the remaining 18 are connected to spine switches of the same ORCA rack to which the leaf switch belongs.

There is a disk I/O subsystem shown as Centralized Filesystems and Data Storage (CFDS) in Fig. 2.2. This system is named "Glade." The Glade leaf switches are connected to the 29^{th} leaf switch in each ORCA as shown in Fig. 2.3. There is also a Data Analysis and Visualization (DAV) subsystem, which is connected to the CFDS subsystem, as shown in Fig. 2.2.

Fig. 2.3 shows the details of the Glade disk I/O subsystem (marked as CFDS in Fig. 2.2). The Glade subsystem has 6 Glade spine and 4 Glade leaf switches, which are connected to GPFS Network Shared Disk (NSD) servers.

[1]ORCA does not appear to be an acronym.

2.3 Measurement study of congestion

This section describes the basis for our methodology, the data collection process executed on Yellowstone, our data analysis method, and the definition of a new metric used as a proxy measure for congestion.

2.3.1 Basis for methodology

InfiniBand switches implement two types of port counters among others, namely, `PortXmitCongTime`, and `PortXmitWait`. `PortXmitCongTime` is the amount of time a port has spent in a congested state, while `PortXmitWait` indicates the amount of time a port has data to send but lacks flow-control credits.

If an administrator disables congestion control, as is the case on the Yellowstone system, `PortXmitCongTime` counters will not register any values, and hence cannot be used to measure congestion. However, we contend that `PortXmitWait` counters can be used as a proxy indicator for congestion, and offer the following justification.

First, how does a switch decide that a port is congested? As noted in Section 2.2, the specific mechanism used to detect congestion is left up to vendor implementation. An approach proposed by Gran and Reinemo [32] for congestion detection is as follows: when the fill-ratio of an input-port Virtual Output Queue (VOQ)[2] holding packets destined to a particular output port exceeds a certain predefined level, the switch will consider the output port to be congested. The predefined level for fill ratio is related to the InfiniBand standard congestion-control parameter called `Threshold`. This parameter controls how quickly a switch reacts to congestion, with a value 15 indicating the fastest reaction to congestion onset, and a value 0 for disabled congestion control. Therefore, whether a switch declares a port to be in a congested state or not (i.e., whether or not packets sent on that port should be marked with FECNs) is dependent on these parameters. The `PortXmitCongTime` counter increases for ports declared to be in a congested state.

Second, if the `PortXmitWait` counter of an upstream port increases, does it necessarily mean that there is congestion on some corresponding downstream switch port? The answer

[2]VOQs are used to avoid the Head-of-Line (HOL) blocking problem.

is yes. The `PortXmitWait` counter of an upstream port (e.g., port q of switch r in Fig. 2.1) increases only when the transmitter has no flow-control credits from the receiver to send packets. This can only happen if the whole input-side buffer of the corresponding receiving port (e.g., port v of switch s) is full. But if the input-side buffer is full, it means that irrespective of the `Threshold` parameter setting, the fill ratio of at least one VOQ in this input-side buffer is guaranteed to have crossed the threshold by the time the downstream switch denies flow-control credits to the upstream switch. In other words, some downstream-switch port would have been declared as being congested before a corresponding `PortXmitWait` counter of an upstream switch port starts to increase.

Finally, consider the flipped question: does a downstream congested port necessarily cause the `PortXmitWait` counter of an upstream port to increase? The answer is no because the predefined level for the fill ratio of input-side VOQs of a downstream port is typically less than 100%, which means congestion would be declared even before the input-side buffer fills up completely, while `PortXmitWait` counter of the upstream port will not increase until the corresponding receiving port has no space in its input-side buffer. The implication is that an increasing `PortXmitWait` counter is a conservative monitor for congestion, not an overly optimistic one. In other words, there were likely more congestion events than reported here by a reading of the `PortXmitWait` counters. Therefore, applying our methodology, if large and/or frequent increases in `PortXmitWait` counters are observed, network administrators can be sure that there is network congestion.

In general, since InfiniBand switch buffer sizes are small, e.g., on the order of 64 KB, which is sufficient to hold just 32 frames [37], it is likely that soon after congestion is declared for a downstream-switch port, the `PortXmitWait` counter on at least one corresponding upstream port will increase. Our group previous paper [4] presented graphs that show the simultaneous increase in `PortXmitCongTime` counter of a downstream port and the `PortXmitWait` counter of an upstream port. A similar observation was made in an experimental study by Subramoni et al. [38], in which the `PortXmitWait` counter was seen to register increases when congestion was caused by an All-to-All Remote Direct Memory Access (RDMA) communications event.

2.3.2 Script implementation and execution for data collection

A Linux command called `perfquery` is available to read InfiniBand port counters. This command was used to read the `PortXmitWait` counter of switch ports at periodic intervals. Since the number of ports in Yellowstone is very large, and we had only limited CPU time to run this script, we decided to observe a set of randomly selected ports for short durations every hour. We wrote a shell script to first randomly select a switch port from across the whole Yellowstone topology, including the Glade and DAV subsystems, and then to issue the `perfquery` command to read the `PortXmitWait` counter of the selected port multiple times with a specified inter-query interval, before selecting the next port for observation.

The approximately 700 switches were divided into 6 sets, to allow for concurrent monitoring. Six instances of the script were executed, with each script running on a different host. The sets of switches are disjoint, and hence no script can randomly draw a port outside its domain.

The steps executed by the script are as follows: (i) Parse the script input arguments to obtain the lower and upper bounds of the assigned set of (approximately 115) switch LIDs. (ii) Select a switch LID at random (using uniform distribution) from the set of assigned LIDs, and select one of the switch ports also at random (using uniform distribution). (iii) Execute a command to check if the selected port is in an operational state. If not, select another switch and port. (iv) Reset the `PortXmitWait` counter for the selected port using the `perfquery` command with the reset (`-R`) flag. (v) Submit a sequence of 100 `perfquery` calls to read the `PortXmitWait` counter of the randomly selected port, with an inter-call time spacing of 100 ms (i.e., the process sleeps for 100 ms after each command completes). Each sequence of 100 calls is referred to as a *round*. In other words, each port was observed for approximately 10 sec. (vi) Append the results of each query into an open Comma Separated Values (CSV) file. (vii) After the 100th query, select another switch and port at random, and repeat steps (iii)-(v).

A `cron` job was used to run the 6 instances of the script (one corresponding to each set of approximately 115 switch LIDs) every hour. After 20 minutes of execution, the scripts terminate. To prevent excessive disk I/O, the results of each `perfquery` are appended to a

file that is stored on the local filesystem of the compute node on which the script is run. When the total size of the file exceeds a threshold, the file is copied to permanent storage.

This data collection process was executed for a period of 23 days in March 2016. The aggregate size of the CSV files was 2 GB, and each file had approximately 5M records[3].

2.3.3 Data analysis

Each row in the CSV file, which we refer to as a *record*, stores the parameters as well as results of one `perfquery` call. Record i in round r is defined to have the following fields:

$$\{t_r, s_r, p_r, t^q_{r,i}, t^p_{r,i}, W_{r,i}\} \tag{2.1}$$

where the first three parameters are common for a round r: t_r is the time instant when the 100-call `perfquery` round r was initiated, s_r is the switch LID (unique across the Yellowstone network), p_r is the port number on switch s_r, where $p_r \in \mathbf{P}_{s_r}$ and $\mathbf{P}_{s_r}$ is the set of all ports on switch s_r, and the remaining parameters are specific to a query i within round r: $t^q_{r,i}$ is the time recorded just before the i^{th} `perfquery` call of round r was issued, $t^p_{r,i}$ is the turnaround time of the i^{th} query of round r (i.e., the difference between the time recorded when the `perfquery` call returns and $t^q_{r,i}$), and $W_{r,i}$ is the `PortXmitWait` counter value of port p_r on switch s_r at time t, where t is estimated to be $(t^q_{r,i} + t^p_{r,i}/2)$ since the exact time when the switch s_r received the message to read the `PortXmitWait` counter is not directly measurable.

All records were merged into one CSV file, and a switch-port-classification script was executed to classify the switch LID into one of 7 categories (ToR, Leaf, Spine, GladeLeaf, GladeSpine, DAVSpine, and DAVLeaf) based on the role of the switch in the Yellowstone topology. The port number p_r in each record was used to determine whether the port was connected to an `up` or `down` link in the fat-tree topology of Yellowstone. For example, the `up` link of a ToR switch connects the ToR switch to a leaf switch, while the `down` link of a ToR switch connects the ToR switch to a host. This switch-port-classification script parsed the output of the `ibnetdiscover` tool to map LIDs and ports into the 7 switch categories

[3]The data is publicly available at this web site: http://pages.shanti.virginia.edu/HSN/tma17-data/

and `up/down` classifications, respectively, and then added two columns to each record in the merged CSV file indicating the switch category and `up/down` port classification.

The augmented record, denoted $\mathbb{R}_{r,i}$, in the merged CSV file has the following fields:

$$\mathbb{R}_{r,i} \triangleq \{\sigma_r, \lambda_r, t_r, s_r, p_r, t^q_{r,i}, t^p_{r,i}, W_{r,i}\} \tag{2.2}$$

where the first two fields are new relative to the fields in the original CSV files, as specified in (2.1). The first field σ is the switch category and has one of these 7 values: ToR, Leaf, Spine, GladeLeaf, GladeSpine, DAVSpine, and DAVLeaf, and the second field λ represents the port type as `up` or `down`.

2.3.4 Metrics

We define a new term *Forced Idle Time Fraction (FITF)*, represented by $F_r[i]$, as the fraction of time when a transmitter is made to wait for flow-control credits from the receiver between the i^{th} and $(i+1)^{th}$ `perfquery` calls within querying round r of a switch port. FITF vector $\mathbf{F}_r$ is defined as a vector with 99 entries corresponding to the 100 `perfquery` calls issued to a switch port in one *round*.

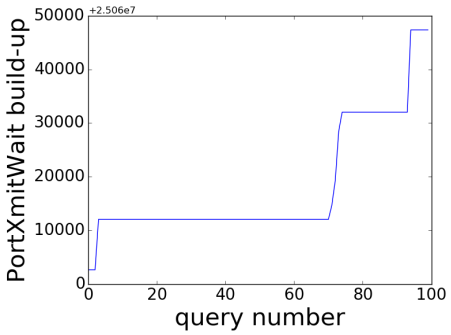
Consider two consecutive records $\mathbb{R}_{r,i+1}$ and $\mathbb{R}_{r,i}$ in the final CSV file that belong to the same round r. The i^{th} element of the vector $\mathbf{F}_r$ is given by:

$$F_r[i] = \frac{\tau\,(W_{r,i+1} - W_{r,i})}{(t^q_{i+1} + \frac{t^p_{i+1}}{2}) - (t^q_i + \frac{t^p_i}{2})}, 1 \leq i \leq 99 \tag{2.3}$$

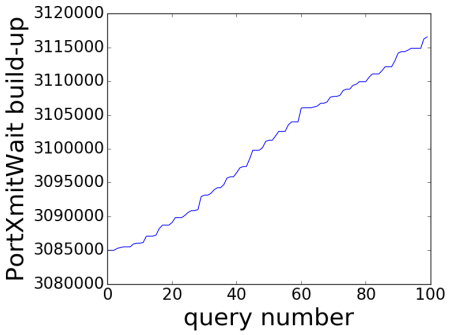
where τ corresponds a system tick, which is roughly 22 ns for an FDR switch port.

The $\mathbf{F}_r$ vectors from the various querying rounds of different switch ports are combined based on their σ switch category and λ port type to create a matrix $\mathbf{A}_{\sigma,\lambda}$ shown below:

$$\mathbf{A}_{\sigma,\lambda} = \begin{pmatrix} \mathbf{F}_1 \\ \mathbf{F}_2 \\ \vdots \\ \mathbf{F}_{k_{\sigma,\lambda}} \end{pmatrix} \tag{2.4}$$



(a) Switch category: ToR, port type: down



(b) Switch category: Spine, port type: down

Figure 2.4: `PortXmitWait` build-up

where the switches in all $k_{\sigma,\lambda}$ rounds belong to category σ, and the queried ports belong to the category λ. There are 11 (σ, λ) combinations **C**={(Spine, down), (Leaf, up), (Leaf, down), (ToR, up), (ToR, down), (DAVSpine, down), (DAVLeaf, up), (DAVLeaf, down), (GladeSpine, down), (GladeLeaf, up), (GladeLeaf, down)}. There are no (Spine, up), (DAVSpine, up), or (GladeSpine, up) possibilities since Spine is the top-level of the topology.

2.4 Numerical results

Section 2.4.1 shows examples of how the `PortXmitWait` counter grows within a 10-sec interval for two ports. Section 2.4.2 presents a comparison of FITF across the 11 switch-port categories. Section 2.4.3 presents an analysis of the rounds in which FITF was non-zero (without considering outliers), and Section 2.4.4 discusses the outlier FITF values.

2.4.1 Examples illustrating `PortXmitWait` growth

Fig. 2.4 shows two examples of how `PortXmitWait` counter value increases within one querying round. Even though the counter was reset at the beginning of each round, the `PortXmitWait` value returned from the first `perfquery` call was not zero because there was a time interval between the resetting action and when the call to read the counter was issued. In Fig. 2.4a, for example, the counter stays unchanged at 25072028 until the 71^{st} query, and then increases to 25092026, where it stays unchanged until the 94^{th} query (the values shown on the y-axis of Fig. 2.4a should be added to 2.506e7, as shown at the top of the axis,

Table 2.1: Total number of querying rounds for each switch category and port type

Port type	Switch category						
	Spine	Leaf	ToR	DAV-Spine	DAV-Leaf	Glade-Spine	Glade-Leaf
down	63129	63527	60217	1422	1192	1455	1165
up	NA	63679	60131	NA	1133	NA	1172

to obtain the actual `PortXmitWait` counter readings). On the other hand, Fig. 2.4b shows that the counter value increases almost continually within each 100-ms interval.

2.4.2 Zero vs. non-zero FITF values

Table 2.1 shows the number of querying rounds $k_{\sigma,\lambda}$ for each combination of switch category σ and port type λ. Since the number of ToR switches is roughly equal to the number of leaf switches, the number of querying rounds for these two types of switches were roughly the same. There are much fewer DAV and Glade switches, and hence there were fewer querying rounds for these switches.

Fig. 2.5 shows a stacked bar-plot for the percentage of zero and non-zero FITF values for each switch category and port type. In the ToR switches, the number of non-zero FITF values is greater than the number of zero FITF values for both up and down port types. For example, a ToR switch down port was queried in 60217 rounds, which means the `PortXmitWait` counter value growth over 100 ms was observed 60217×99 times, which is roughly 6M. In these 6M observations, we found that the port was forced to wait for credits (which is an indication of congestion) in 60% of these 100-ms intervals. The implication of a ToR switch downlink port being held up waiting for credits is that the corresponding HCA buffer was full.

Fig. 2.5 shows that for GladeLeaf switches, there is more congestion on the uplink ports than on the downlink ports. A possible explanation for congestion on the uplink ports is as follows. The Glade disk I/O subsystem shown in Fig. 2.3 offers users multiple filesystems such as `scratch` and `home`. The `scratch` filesystem is used for temporary data storage. Users move data from the `scratch` filesystem to the `home` filesystem for permanent storage. These data transfers could cause congestion on the uplink GladeLeaf ports, as the

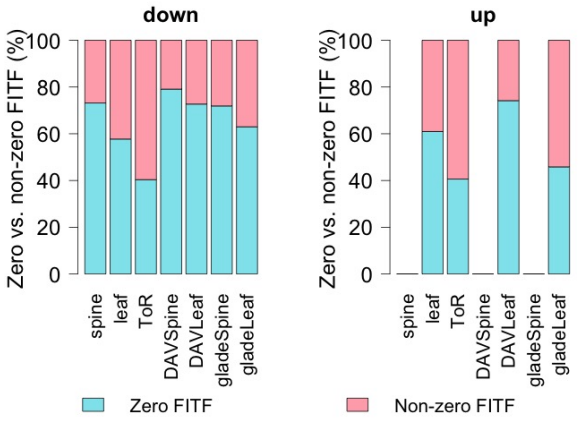


Figure 2.5: The percentage of zero and non-zero FITF values for each switch category and link (port) type

two filesystems are likely to be connected to nodes served by different GladeLeaf switches, which would necessitate transfers through GladeSpine switches.

Fig. 2.5 shows that on DAVLeaf switches, congestion occurs at equal rates on downlinks and uplinks. One DAVLeaf switch is connected to one GladeLeaf switch, via 6 links, and these links are classified as downlinks on both switches. Uplinks connect DAVLeaf switches to DAVSpine switches. Since users move data for analysis from the Glade disk I/O nodes into the DAV nodes, and store analysis results back into the Glade disk I/O nodes, traffic flows both ways on the DAV-to-Glade links.

We expected congestion to be predominantly in the Glade disk I/O subsystem switches, but the results in Fig 2.5 show that ToR switch ports experience congestion even though they primarily serve compute nodes. This could be the effect of cascading rate reductions caused by the link-by-link flow control procedure. Victim ports could occur anywhere in the network far from a congested port.

2.4.3 Non-zero FITF values

To gain some insights into the fraction of each 100-ms interval that a port was denied credits, we undertook a study of just those rounds in which the FITF value was non-zero for at least one 100-ms interval. We refer to these rounds as *non-zero rounds*.

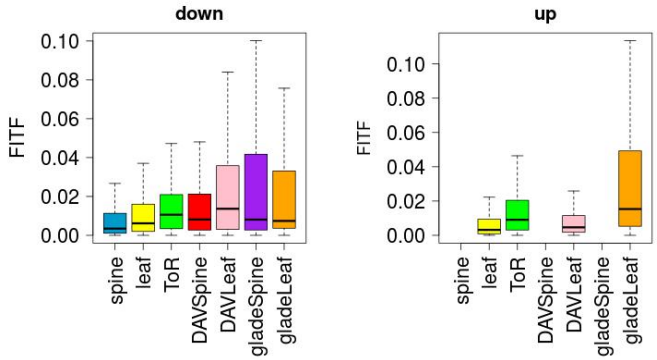


Figure 2.6: The maximum FITF value per non-zero round for each switch category and link type (outliers removed)

Fig. 2.6 shows a boxplot of the maximum non-zero FITF values across all non-zero rounds of all switch ports belonging to a particular switch category and port type. Using (2.4), a maximum FITF was computed for each row of the matrix $\mathbf{A}_{\sigma,\lambda}$, i.e., the maximum value across the elements of vector $\mathbf{F}_r$, $1 \leq r \leq k_{\sigma,\lambda}$. From Fig. 2.6, in which outliers were removed, we conclude that in most of the 10-sec observation rounds, a port was denied credits in less than 10% of a 100-ms interval.

The plot in Fig. 2.6 shows that the Glade subsystem switch ports had the highest variability in maximum FITF. This is because bulk data transfers do not occur continuously, but, when they do occur, these transfers can cause congestion, and increase the forced idle time significantly. For example, the maximum FITF across all querying intervals of GladeLeaf downlink ports was 85%. The DAV leaf switches also experienced a high variability due to the bulk data transfers that occur occasionally. The ToR switches have more variability when compared to the compute subsystem leaf and spine switches.

Fig. 2.7 shows a boxplot of the average FITF across non-zero rounds for all switch ports belonging to a particular switch category and port type. The averaging was done across only the non-zero FITF values. The ToR switches experience more variability in average FITF when compared to the spine and leaf switches. This is an interesting result as it is the opposite of our expectation that leaf and spine switch links would have more congestion due to oversubscription at the higher levels of the fat tree.

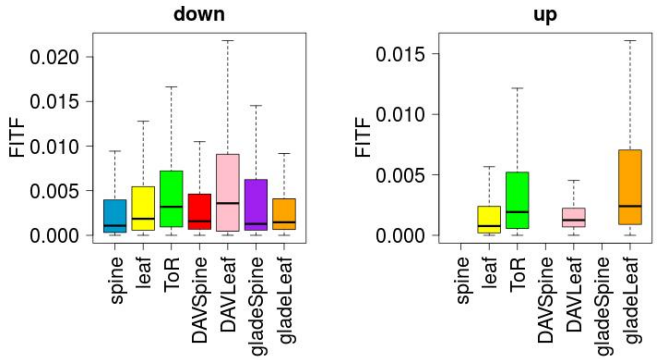


Figure 2.7: The average FITF value per non-zero round for each switch category and link type (outliers removed)

Table 2.2: Details about outliers left out of the boxplot in Fig. 2.6 and Fig. 2.7

Port type		Switch category						
		Spine	Leaf	ToR	DAV-Spine	DAV-Leaf	glade-Spine	glade-Leaf
down	min	0.027	0.037	0.047	0.049	0.086	0.102	0.078
	max	0.865	1.179	0.879	0.432	0.409	0.807	0.852
	Number of querying rounds with non-zero maximum FITF	2765	40258	51814	565	550	806	627
	Number of outliers	670	2771	3185	56	43	119	81
	Number of FITF values ≥ 1	none	37	none	none	none	none	none
up	min	NA	0.022	0.046	NA	0.026	NA	0.116
	max	NA	1.163	1.158	NA	0.424	NA	0.959
	Number of querying rounds with non-zero maximum FITF	NA	47167	44807	NA	959	NA	1132
	Number of outliers	NA	3751	2561	NA	111	NA	128
	Number of FITF values ≥ 1	none	23	6	none	none	none	none

Table 2.3: Example of two consecutive records belong to the same querying round r of a down Leaf port

i	t^q	t^p	W	W_{i+1}-W_i	$(t^q_{i+1} + \frac{t^p_{i+1}}{2}) - (t^q_i + \frac{t^p_i}{2})$	FITF
1	1.45640989347e+18	18350267	2426695474	6656811	124199424	1.179
2	1.4564098936e+18	24841529	2433352285			

2.4.4 Outlier FITF values

The key finding presented in this section is that there were some 100-ms intervals in which the port was completely stalled, i.e., the transmitter was prevented from sending data due to a lack of flow-control credits for the whole 100-ms interval. Occurrences of these

extreme cases (outliers) were not included in the boxplots of Fig. 2.6 and 2.7 to allow for a better visualization of the differences between the switch-port categories. Nevertheless, these outliers are important because highly parallelized communication-intensive applications with MPI synchronization calls, e.g., climate-science applications that use 1000 cores or more, will experience significant increases in their execution delays if such long stalls occur. An *outlier* is defined as a data point that lies 1.5 × IQR above the 75% number or below the 25% number, where IQR is Inter-Quartile range (difference between the 25% and 75% values).

Table 2.2 shows the outliers statistics for the different switch ports categories. For example, there were 44807 querying rounds of ToR switch uplink ports that had a non-zero maximum FITF, and out of the 44807 querying rounds, there was a total of 2561 outliers. The `max` rows in Table 2.2 for both downlink and uplink ports show FITF values that are greater than 1, which represents 100%. For an explanation of how this is possible, we provide the details of one example query.

Consider the example query shown in Table 2.3. The rows correspond to the values of two consecutive records $\mathbb{R}_{r,1}$ and $\mathbb{R}_{r,2}$ per (2.2). To find FITF, the difference between the `PortXmitWait` counter readings W in the two consecutive records is computed, and the difference is divided by the time difference between the *estimated* times at which the counter was read. As shown in (2.3), the exact time at which the port counter is read cannot be determined accurately. Instead, the time is estimated by halving the query turnaround times $t^p_{r,1}$, and $t^p_{r,2}$. If the first query took a shorter time reaching the switch, and the second query took a longer time reaching the switch than estimated by the halving operation, then the time between the two consecutive readings of the port counter could be longer than the estimated time difference. Such an occurrence would result in an estimate of FITF that is greater than 1. Effectively, such FITF values should be interpreted as the port being stalled, waiting for flow-control credits, during the entire 100-ms interval.

2.5 Discussion: Impact of findings

In this section, we describe the potential value of our findings for InfiniBand network operations and for scientific applications. Specifically, the results could be interesting both

to InfiniBand network administrators, and to scientific researchers who run applications on InfiniBand HPC systems.

Network operations As stated in Section 2.1, most InfiniBand HPC administrators do not enable congestion control due to a perceived lack of proven methods for setting parameters. But if congestion control is disabled, network administrators are blind as to whether or not congestion is occurring, because the `PortXmitCongTime` counters will not register any values. Our proposed method for using `PortXmitWait` counters as a proxy, and our software for collecting the data and analyzing the collected data to derive FITF values, can thus be used by administrators to gain insights into the state of their networks. Further, administrators could correlate FITF values with application execution time. Such a correlation study would show whether or not high variability in application execution can be attributed to network congestion. If a high correlation is observed, network administrators could then enable congestion control.

Administrators can test the various solutions (for setting congestion-control parameters) that have been proposed in research literature (reviewed in Section 2.6). For example, our research group designed, implemented and evaluated a scheme called Dynamic Congestion Management System (DCMS) [4]. DCMS was designed to detect if a victim flows were created because of a congestion event and modify congestion parameters to dissipate the congestion event rapidly. Recall from Section 2.2 that the switch has a parameter called `Marking_Rate`. The key idea of DCMS is to dynamically adjust this parameter based on whether-or-not victim flows are being created by a congestion event. This solution was found to be effective through an experimental evaluation. For example, the evaluation results showed that a victim flow's throughput decreased from 8 Gbps to 2.67 Gbps because of congestion at a switch port that the victim flow did not even traverse. However, when DCMS detected the congestion and took action by adjusting the switch `Marking_Rate`, the congestion-causing flows dropped their sending rates, and the victim flow throughput rebounded back to 8 Gbps.

Applications We offer three examples of the impact of our FITF-related findings on scientific-research applications. *First*, Petrini et al. [39] presented evidence that even

frequent, short-duration congestion events can have a detrimental impact on applications that have fine-grained communications (frequent short message exchanges). In fact, our results show that in Yellowstone, while FITF values were of short duration, i.e., less than 10% (see Fig. 2.6), the network link stalls occurred quite frequently (Fig. 2.5 showed that the `PortXmitWait` counter registered increases in a significant percentage of the observed 100-ms intervals).

Second, we experimented with the climate-science High-Order Method Modeling Environment (HOMME) application, and found that it issues a synchronization MPI call 140K times in one 90-core run (which is a small configuration; in actual runs by scientists, 1000s of cores are used for each run). If messages sent by one source MPI rank took longer to reach their destination MPI ranks (e.g., these messages were affected by one of the outlier congestion events reported in Table 2.2), then all MPI ranks will be delayed when an `MPI_Barrier` or `MPI_Waitall` call is encountered. This could lead to a significant increase in the total execution time of the application. Furthermore, since synchronization calls cause MPI ranks to stop processing and wait until the slowest MPI rank completes, communication delays can result in under-utilization of compute nodes.

Third, Subramoni et al. [38] stated that their proposed technique for reducing "the amount of network contention observed during the All-to-All/FFT operations" resulted in a "9% improvement in the communication time of P3DFFT at 512 processes."

In *summary*, our methodology, new metric, data collection and analysis software, and specific numerical findings for Yellowstone, are all useful contributions to the InfiniBand provider and user community.

2.6 Related work

As noted in Section 2.1, there have been many studies of InfiniBand congestion control [4, 17, 29–36], but none of these papers measured congestion on a production InfiniBand network as we have done. Some papers [33, 38, 40] used `PortXmitWait` as an indicator of congestion, which is also the basis of our FITF metric.

Papers that suggested mechanisms for seting congestion-control parameters include work by Pfister et al. [17], Gusat et al. [29], and Gran et al. [31]. All three studies used simulations to characterize the effects of various parameter settings on application metrics.

Several papers, such as VOQsw [41], dFtree [42], vFtree [43], BBQ [44], Flow2SL [35], and pFtree [45], proposed using InfiniBand virtual lanes to combat congestion.

Other papers proposed new/enhanced congestion control mechanisms, unconstrained by the InfiniBand standard. Yan et al. [30] proposed a Power Increase and Power Decrease (PIPD) function for controlling sending rate. Michelogiannakis et al. [46] proposed a Channel Reservation Protocol (CRP) to prevent congestion. Russell et al. [47] developed Red and Green light-Based Congestion Control (RGBCC), which is less sensitive to small changes in parameters when compared to the standard InfiniBand congestion control mechanism. Liu et al. [48] proposed improvements to the InfiniBand standard by adding a Link Bandwidth Availability Report (LABR).

Finally, modifications were proposed to job schedulers, such as SLURM, to take into account network conditions [40] when assigning MPI ranks to nodes. The work by Bhatele et al. [16], which demonstrated the impact of neighborhood jobs on application performance in Cray networks offers a good model for a similar study in InfiniBand networks.

2.7 Summary and Conclusions

This chapter presented a 23-day measurement study of congestion on a production, highly utilized, 72K-core InfiniBand cluster called Yellowstone. We proposed a methodology based on reading the `PortXmitWait` counter of ports, and a new metric called Forced Idle Time Fraction (FITF). While congestion is likely caused by bulk data flows in the disk I/O and Data Analysis and Visualization subsystems of Yellowstone, our findings were that ports of even ToR switches that serve compute nodes suffered from such flow-control related stalls. In about 60% of the 100-ms intervals in which ToR-switch ports were observed, the ports had to wait for flow-control credits, but most of these transmission stalls were shorter than 10 ms. Prior work showed that short but frequent congestion events are detrimental to applications with fine-grained communications. Also, some congestion events lasted for

significant portions (in the range 60-80%) of the 100-ms intervals, with several events even reaching 100%. Such events can significantly increase execution time of applications that have MPI synchronization calls.

Chapter 3

SDN-Enabled Headroom Services for High-Speed Data Transfers

3.1 Introduction

WAN links are typically operated at 30-40% average utilization levels [21]. This leaves a large headroom as illustrated in Fig. 3.1. There are three reasons for such low-utilization operation: (i) to accommodate long-term growth in the traffic volume, (ii) to handle the extra load created by the traffic that is rerouted onto a link in response to failures in other links, and (iii) to support Elephant Flows (EFs), which are typically high-rate large-sized dataset transfers. Examples include inter-datacenter movement of user files, backups for disaster recovery, and scientific data movement for high-performance computing.

While, in theory, EFs should be able to take advantage of headroom, in practice, customer access-link SLAs are limiting factors. For example, a customer may pay connectivity charges for a 1 GE port on the provider network, but have an SLA with CIR set to 200 Mbps and PIR set to 600 Mbps. In this case, the customer EFs would be limited to 600-Mbps throughput, while simultaneously intra-domain provider links would remain underutilized.

To exploit the headroom with customer EFs, we propose two new Internet provider services: Static Headroom (SH) service, and Dynamic Headroom (DH) service. To avoid adverse effects on other flows, we propose that EF-packets using these headroom services be

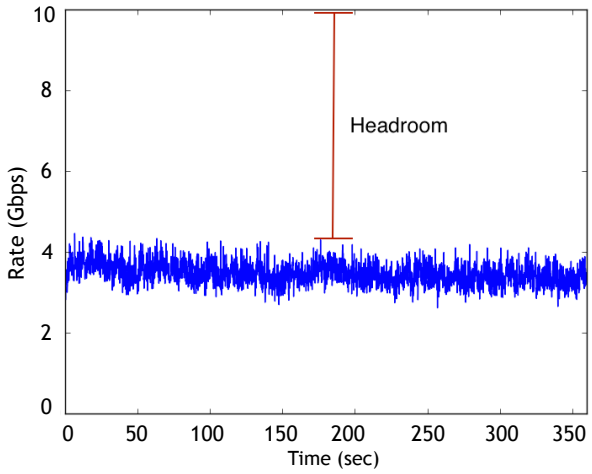


Figure 3.1: 2014 packet trace collected by CAIDA [5] on a 10 Gbps link

tagged with the low-priority CS1 DiffServ Differentiated Services Code Point (DSCP) [49]. The CS1 service class is lower than the standard best-effort CS0 class, i.e., CS1 packets are sent to a lower-priority queue. Thus, even if EFs fill the headroom on provider links, best-effort flows will not suffer increased packet losses or packet delays.

To implement SH and DH services, providers would need to offer a new type of SLA to their customers in which the aggregate traffic rate can reach the access-link capacity as long as packets in excess of CIR are tagged with CS1 DSCP. With our proposed headroom services, in the example described above, customer EFs could enjoy up to 1-Gbps throughput. Such increased EF-throughput is the *benefit of SH/DH services to customers*.

The *benefit of SH/DH services for the provider* is that the link headroom can be utilized without adversely affecting the provider's primary best-effort service, or the network's ability to absorb additional traffic resulting from reroutes caused by link failures. Thus, both providers and customers can benefit from our proposed SH/DH services.

The difference between SH and DH services is that the former requires a Software Defined Network (SDN) controller in only customer networks (to handle the marking of EF-packets with the CS1 DSCP), while the latter requires the additional deployment of an SDN controller in provider networks. The advantage gained is that the provider-network SDN controller can compute a path with maximum available headroom between its own ingress and egress routers to offer customer EFs better throughput.

Analytical and simulation based comparisons of Best-Effort (BE), SH and DH services for EFs were conducted. The *key findings* are as follows:

- The advantage of SH over BE service is that, with SH service, EF-induced packet loss in BE traffic is eliminated. The higher the general-purpose IP traffic burstiness level, the bigger this advantage.

- The advantage gained with DH relative to SH is higher average EF throughput. This advantage is higher the greater the non-uniformity in EF traffic patterns. Under high non-uniformity, the DH gain factor in average EF throughput relative to SH was a factor of 2.4 on a 6-node simulated network.

- A combination of SH and DH services is recommended because only SH service is suitable for short-duration high-rate flows as DH service incurs the cost of path-setup delay; on the other hand, DH service allows simultaneous EFs to be routed on different paths, which could result in higher EF throughput.

The material presented in this chapter is an excerpt from our published work *SDN-enabled headroom services for high-speed data transfers* [27] ©2017 IEEE. Section 3.2 reviews related work. The proposed novel SH and DH services are described in Section 3.3. Section 3.4 presents results of our simulation study comparing BE, SH and DH services, and the chapter is concluded in Section 3.5.

3.2 Related Work

The Microsoft SWAN [50] and Google B4 [21] network architectures were designed to increase the utilization of their dedicated inter-datacenter links. Since there was full control of all end-to-end resources, these solutions focused on orchestrating data transfers to increase link utilization. There are other solutions for increasing utilization of dedicated inter-datacenter links [51, 52]. Our headroom services based solution is different from these prior solutions because it requires an inter-domain solution involving customer and provider networks with benefits to both.

Solutions [53, 54] proposed the use of the customer access link for EFs during off-peak hours. Consider SLAs in which the provider charges the customer based on its 95 percentile usage level within a set time period T (usually a month). With this SLA, even a few EFs during peak hours can increase the 95 percentile bandwidth significantly, thus increasing the cost to the customer. Therefore, these solutions proposed postponing delay-tolerant data transfers (EFs) to off-peak hours so that the 95 percentile usage level is lower. The disadvantage of these solutions is that large data transfers incur a waiting delay, while no such delays are incurred in our solution.

Solutions [55–60] proposed dynamic configuration of paths, but unlike our DH service, these solutions require advance knowledge of available bandwidth. Our DH service does not require SDN controllers to estimate headroom capacity since EFs only affect each other and not the general-purpose IP traffic, which gets higher priority.

Our solution is most similar to Internet2 QBSS [61] and SIFT [62]. For example, SIFT identifies large flows and redirects them to a lower-priority queue to reduce packet delay for small flows. While these solutions demonstrate the benefits of directing EFs to lower-priority queues, they do not address the practical aspects of how the SIFT solution can be deployed to the benefit of both providers and customers in the multi-domain Internet, as we do. Additionally, our proposal for the DH service is new relative to this prior work.

3.3 SDN-enabled SH and DH Services

The subsections below describe the two variants of the proposed headroom service.

3.3.1 Static Headroom (SH) service

Fig. 3.2 shows an example Tier-2 provider network in which two queues have been configured on an output port of a router, and packets marked with the CS1 DSCP are sent to the lower-priority queue. This two-queue configuration is illustrated in only one router port for space reasons, but SH service requires this configuration in all routers.

Fig. 3.2 also shows two methods by which the customer-network SDN controller can determine the flow IDs of EFs. In the first method, the file-transfer application running on

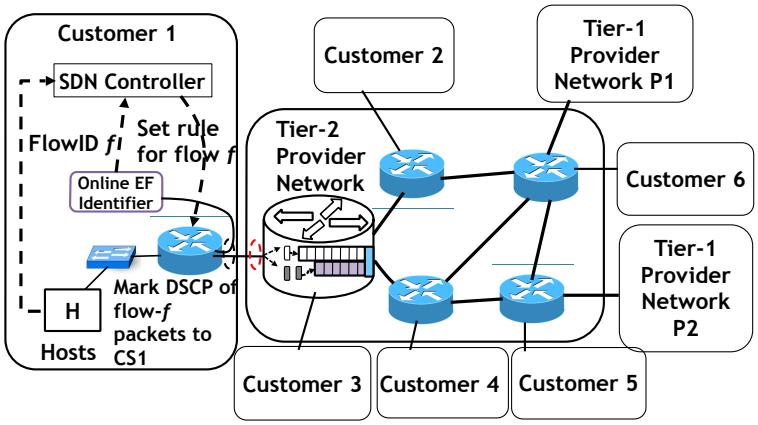


Figure 3.2: Static Headroom (SH) service architecture

hosts explicitly provides the flow ID f to the SDN controller before starting the EF. In the second method, the customer deploys an online EF identifier that identifies EFs from live traffic mirrored to it from the router, and provides the flow IDs to the SDN controller. The SDN controller can then set a filter rule to match all packets corresponding to the EF ID, and mark the DSCP field in the IP headers to CS1.

A typical customer SLA is as follows: $\{CIR, PIR\}$ where $PIR < C$, where C is the access-link capacity. Since PIR is smaller than the link capacity C, without SH service, the customer's EFs can only enjoy throughput up to PIR. Furthermore, traffic sent at rates in the range $\{CIR, PIR\}$ could cause packet losses and/or delays to other customers' traffic in the provider network. Our proposed SH service differs from current-SLA based BE service in two ways: (i) $PIR = C$, and (ii) EFs sent at rates above CIR are tagged with CS1 DSCP. With the provider's configuration of routers to treat these packets with lower-priority than BE packets, a customer's SH-service EF will not cause adverse effects on other general-purpose flows.

If the Tier-2 provider network also has an SLA with its Tier-1 provider wherein $PIR < C$, then the new SH-service based SLA could be negotiated between the Tier-2 provider and its Tier-1 provider. If, for example, two or more simultaneous EFs from different customers of the Tier-2 provider network shown in Fig. 3.2 are routed to the same egress link to Tier-1 Provider Network P1, then both flows would have packets tagged with CS1 DSCP. Therefore the SDN controller in Tier-2 provider network (not shown in Fig. 3.2) is not required to

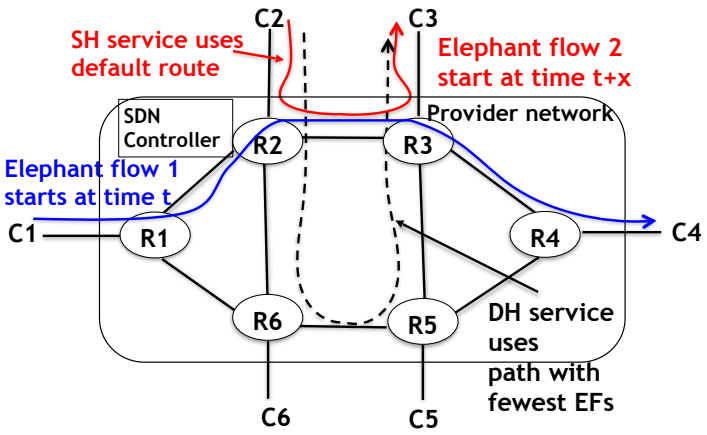


Figure 3.3: Dynamic Headroom (DH) service EF-flow routing

execute DSCP packet marking as was needed in customer networks. Since EFs typically use TCP, the standard TCP congestion control algorithms would enable sharing of the lower-priority queue bandwidth offered to the headroom-service flows.

3.3.2 Dynamic Headroom (DH) service

To offer DH service, a provider would need to deploy an SDN controller, which is added complexity. The customer network SDN controller would need to notify the provider SDN controller of an incoming EF. The provider SDN controller can then create flow-specific forwarding entries in the routers on the path with the most available headroom. Since the goal of these headroom services is to fill the excess capacity of intra-provider network links, such custom routing could yield throughput benefits for customer EFs and increased usage for providers.

The provider-network SDN controller would then set OpenFlow (or equivalent) flow-matching rules in the routers on the selected path for the EF identifier (ID: five tuple of source and destination IP addresses, source and destination port numbers and protocol), and direct packets from this EF to the appropriate output port. As the packets of the EF would be tagged as CS1, these packets would be queued in the low-priority queue at the output port.

Fig. 3.3 illustrates the potential gain using DH service with an example. Elephant flow 1 starts at time t from customer C1 to customer C4, while EF 2 starts from C2 to C3 at time

$t + x$. If both flows are routed on the default paths, then they will share the headroom on the link between routers R2 and R3. On the other hand, with DH service, EF2 could be custom routed on the path R2-R6-R5-R3. If all EFs are considered equally capable of filling the headroom, then a simple SDN controller can just track the number of EFs sharing a link. More sophisticated versions of DH service could allow customers to provide maximum rate information for EFs, which would then allow for even further improvements in EF throughput.

3.4 Evaluation

Section 3.4.1 describes an analytical single-link model of SH service, the purpose of which was to understand the impact of link utilization on average EF throughput. Section 3.4.2 describes an NS3 packet-level simulation study to compare BE and SH services. Section 3.4.3 describes a Matlab based flow-level simulation comparison of SH and DH services. In all models, we assumed that the EF-arrival process was Poisson with parameter λ, and that EF sizes fit a Pareto distribution with shape α and scale x_m, which is the minimum value of file size.

3.4.1 Impact of utilization on throughput

Analytical model A provider network is modeled as a single link with headroom capacity C_{hr}. Given our EF-related assumptions of a Poisson arrival process and Pareto file sizes, we can use the M/G/1 queueing model to determine the average number of concurrent EFs competing for the provider-link headroom. The mean number of jobs in an M/G/1 queueing system is given by:

$$E[N] = \frac{\rho}{2} + \frac{\rho + \lambda^2\sigma^2}{2(1-\rho)} \tag{3.1}$$

where ρ is traffic load ($\rho = \lambda/\mu$), μ is mean service time (EF duration), and σ^2 is service-time variance. We set $\alpha > 2$, and therefore, the mean and variance of service time are given by:

$$1/\mu = \frac{\alpha x_m}{(\alpha-1)C_{hr}};\ \sigma^2 = \frac{x_m^2\alpha}{(\alpha-1)^2(\alpha-2)C_{hr}^2} \tag{3.2}$$

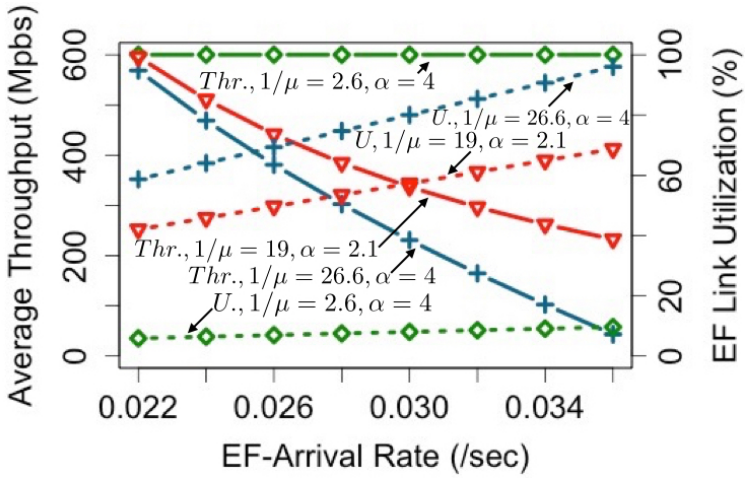


Figure 3.4: Analytical model of SH service; (α, x_m); x_m unit: GB

Since TCP allows EFs to simultaneously share the server (i.e., link headroom), the average EF throughput is:

$$\text{Average EF Throughput} = \begin{cases} \dfrac{C_{hr}}{E[N]}, & \text{if } E[N] > 1 \\ C_{hr}, & \text{otherwise} \end{cases} \tag{3.3}$$

Input parameters were set as follows: (i) $C_{hr} = 600$ Mbps; (ii) EF-arrival rate λ was increased from 0.022/sec to 0.036/sec in steps of 0.002; and (iii) Three sets of file-size distribution parameters were chosen such that $(1/\mu, \alpha)$ are: (i) {2.6 sec, 4}, (ii) {26.6 sec, 4}, and (iii) {19 sec, 2.1}, respectively.

Fig. 3.4 shows average (EF) throughput (left-hand y-axis, solid lines) and EF link utilization (right-hand y-axis, dashed lines) under increasing traffic load. When the EF-arrival rate is 0.034 sec and mean service time is 26.6 sec, link utilization (which is same as traffic load ρ) is 0.92. At this operating point, the average number of concurrent EFs would be 5.86, which makes the average EF throughput about 102 Mbps. At the same EF-arrival rate, if the mean service time is only 2.6 sec, utilization is only 0.1, and consequently EFs enjoy throughput close to the link headroom of 600 Mbps.

To illustrate an intermediate operating point, we lowered α to 2.1, and mean service time to 19 sec. For the EF-arrival rates shown on the x-axis, utilization is lower. For example, when the EF-arrival rate is 0.034 sec, link utilization is only 0.65, and the average EF

throughput was correspondingly higher.

3.4.2 Comparison of BE and SH services

Simulation setup *Topology:* the provider network was modeled as a two-router dumbbell with multiple host nodes simulating customer networks connected to each provider node (router). Both the inter-router provider link rate and the access link rate were set to 1 Gbps.

Host configuration: The TCP buffer size and Ethernet-layer traffic-control queue size were set to be large enough to ensure that each EF could fill the headroom in the inter-router link. *Router configuration:* to support priority queuing, we used the NS3 `pfifo` model in the routers. This model has 3 queues, each of which can grow up to a size specified by the `pfifo` attribute `Limit`. The `Limit` parameter represents the total buffer size for a port, which any single queue is allowed to consume. To avoid EF packets completely filling the buffer, we modified the NS3 code to allow for a static partitioning of the port buffer between the CS0 and CS1 queues. We assumed a port buffer size of 10000 packets in our BE-service simulation, and in the SH-service simulation, we assumed a size of 5000 packets for each of the two queues for a fair comparison.

Workload: host nodes generated EFs and general-purpose IP traffic. (i) EF: size parameters α and x_m were set to 4 and 1.5 GB, respectively. Mean EF size was 2 GB. The EF-arrival rate λ was varied from 0.083/sec to 1/sec. (ii) General-purpose IP traffic trace: was generated using an NS3 model [63] of a Poisson Pareto Burst Process (PPBP) [64]. This model has 4 parameters: mean burst length, T_{on}, and Hurst parameter, H, were held constant at 0.2 and 0.7, respectively, while the other two parameters, burst constant bit rate and mean number of bursts, were varied as shown in Table 3.1. The goal was to vary `burstiness`, which was defined as the Coefficient of Variation (CV) of the number of bytes in 200-ms intervals. Thus, if B_{200} represents the number of bytes in each 200-ms interval, `burstiness` is $SD(B_{200})/Mean(B_{200})$, where SD is standard deviation computed across the simulation duration. Given that WAN link utilization averages between 30-40% [21], we chose the PPBP parameters such that regardless of `burstiness`, the average throughput of the general-purpose IP traffic trace was around 400 Mbps.

Metric: Packet loss rate in the general-purpose IP traffic was used to evaluate the performance. A total of 30 EFs were generated in each simulation run. The average packet loss rate across 30 runs per configuration is reported.

Table 3.1: PPBP parameters and `burstiness`

Burst constant bit rate r	Mean number of bursts λ_p	`Burstiness`
1 Mbps	2000	0.073
50 Mbps	40	0.298
110 Mbps	18	0.432
200 Mpbs	10	0.572

Results When SH service was used for EFs, general-purpose IP traffic suffered no packet losses because of the use of a lower-priority queue for EF packets. However, when EFs used BE service, there were packet losses in the general-purpose IP traffic. Fig. 3.5 plots the average packet loss rate in the general-purpose IP traffic against EF-arrival rate for different values of `burstiness` in the general-purpose IP traffic. While the TCP senders of EFs will adjust their sending rates based on available bandwidth, due to round-trip delays in making these adjustments, packet loss rate will be larger during high burstiness periods in the general-purpose IP traffic. For example, when the general-purpose IP traffic `burstiness` was 0.572, packet loss rate was 0.8% even when the EF-arrival rate was only 1 per 12 sec (x-axis value = -2.5).

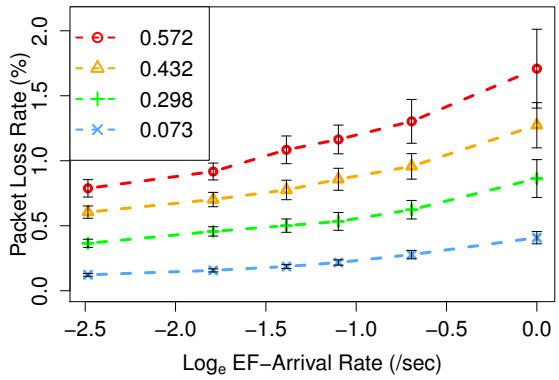
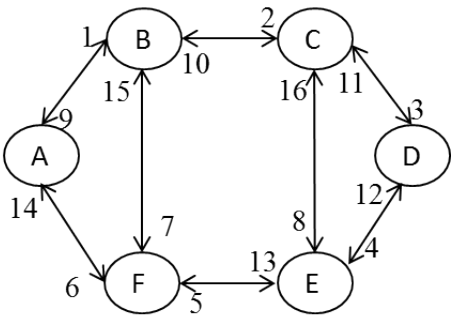
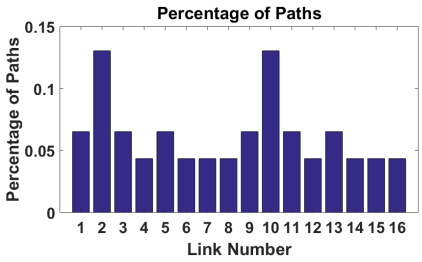


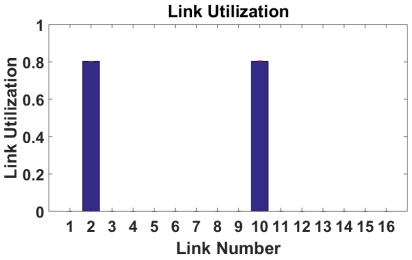
Figure 3.5: Average (with 95% confidence-interval error bars) general-purpose IP packet-loss rate when EFs use BE service; under SH, this packet loss rate is 0



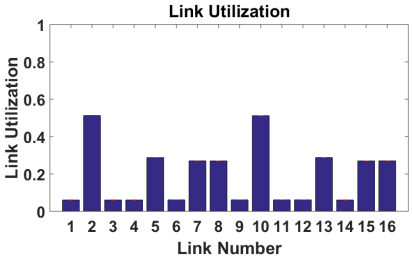
(a) Simulated network



(b) P_i for all links i



(c) SH: $w = 1$ and $\lambda = 0.1$



(d) DH: $w = 1$ and $\lambda = 0.1$

Figure 3.6: DH-service simulation

3.4.3 Comparison of SH and DH services

Simulation setup Given the high execution time of packet-level simulation, a flow-level simulation was conducted where a simplified assumption of equal sharing of link capacity by concurrent flows was made. Only EFs were generated, i.e., there was no background traffic.

Topology: Fig. 3.6a shows the network topology. The numbers denote links, e.g., 5 denotes the link from router E to router F, while 13 denotes the link from router F to route E. *Path computation for SH-service:* EFs were routed on default IP paths, and these paths were assumed to use the top set of links in the network. For example, flows from router A to router D were routed on links 1 ($A \rightarrow B$), 2 ($B \rightarrow C$), and 3 ($C \rightarrow D$). Let $N_i^{s,d}$ represent the number of source-destination paths that traverse link i, and $\mathbf{L}$ represents the set of links in the network. The percentage of paths that use link i is given by:

$$P_i = \frac{N_i^{s,d}}{\sum_{1 \leq i \leq |\mathbf{L}|} N_i^{s,d}} \tag{3.4}$$

For example, link 1 carries packets for three source-destination pairs $(A - B)$, $(A - C)$

and $(A-D)$, while link 2 carries packets for the most number (6) of source-destination pairs, $(A-C)$, $(A-D)$, $(B-C)$, $(B-D)$, $(B-E)$, and $(F-C)$. Fig. 3.6b shows these percentages. *Path computation for DH-service:* in the DH-service simulation, the number of ongoing EFs was maintained for each link. For a newly arriving EF, the maximum number of ongoing EFs across all links of each potential path was computed, and the path with the smallest value was selected. Thus, EF load was spread to more links.

Workload: (i) The source and destination: for each EF were determined using the following rules. An EF was assigned nodes B and C as source and destination, respectively (or vice versa), with probability $1/30 + 14w/30$, while the probability of any other of the 28 node pairs being selected was $1/30 - w/30$. The weight w was varied from 0 to 1, in steps of 0.1. When w is 0, all node pairs are equally likely to be selected, and hence the traffic pattern would be uniform. When w is 1, all EF transfers will occur between nodes B and C, with no EFs between other pairs. Link 2 $(B \rightarrow C)$ and Link 10 $(C \rightarrow B)$ were chosen for this worst-case non-uniform traffic pattern because these links had the highest percentage of source-destination paths as seen in Fig. 3.6b. The impact of the degree of non-uniformity on the relative performance of DH and SH was determined in this simulation study. (ii) The EF-arrival process parameter: λ was varied from 0.01/sec to 0.1/sec, in steps of 0.01. (iii) The EF-size parameters: α and scale x_m were set to 4 and 15 GB, respectively. With these parameters, the mean flow size was about 20 GB.

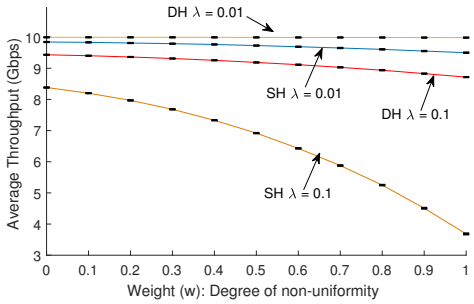
Metrics: In each simulation run, 100,000 EFs were generated, and 30 runs were executed for each combination of parameter settings. The two output metrics of interest are average (EF) throughput and DH gain factor, which is a measure of the improvement in average (EF) throughput with DH relative to SH service.

Results Before presenting simulation results for the metrics, using a third metric, link utilization, we demonstrate why DH service has an advantage over SH service.

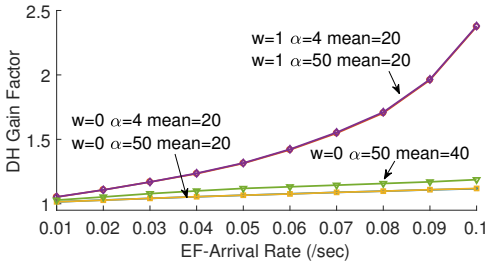
Utilization of link i is:

$$U_i = \frac{\sum_{k=1}^{k=K} \delta_{i,k}}{K} \tag{3.5}$$

where $\delta_{i,k} = 0$ if the number of EFs $M_{i,k}$ traversing link i at the end of timeslot k, $1 \leq k \leq K$ is zero, and 1 otherwise. Since EFs can arrive and depart in the middle of a 1-sec timeslot



(a) Impact of non-uniformity measure on average EF throughput



(b) DH Gain Factor

Figure 3.7: Comparison of SH and DH services; α: shape; mean (GB)

(which is the granularity used for statistics collection), the computed utilization is an approximation.

Figs. 3.6c and 3.6d show link utilization for simulation runs under SH and DH service, respectively. In both runs, w was set to 1, which means the traffic pattern was highly non-uniform (all EFs were between routers B and C) and λ was set to 0.1/sec. Fig. 3.6c shows that with SH service, as default routing is used for EF paths, links 2 and 10 were highly utilized, while all other links were unutilized. Fig. 3.6d shows that with DH service, as EFs are routed on other paths, all links are utilized. Given this spreading of EF load, average EF throughput is better with DH service than with SH service. To prevent thrashing at very high loads, a cut-off threshold should be used to limit the number of hops on which EFs are routed. This condition was not encountered for the simulated loads.

Next, we quantify the two main metrics, average EF throughput and DH gain factor. Fig. 3.7a compares the average EF throughput of SH and DH services under increasing levels of non-uniformity in the traffic matrix. The error bars, which represent 95% confidence intervals for the average EF throughput computed over 30 runs, were very small, and hence not easily visible in Fig. 3.7a.

Even under uniform traffic pattern (when $w = 0$), when the EF-arrival rate was 0.1/sec, the average EF throughput was higher with DH service when compared to SH service. This is because some links carry a greater percentage of source-destination paths, which leads to overloaded links under SH, and hence a lower average EF throughput. With DH service, the SDN controller can spread concurrent EFs to other paths.

As the degree of non-uniformity increases to $w = 1$, the advantage of DH over SH becomes even more obvious. For example, under an EF-arrival rate of 0.1/sec, with the uniform traffic pattern, the average EF throughput for SH and DH are 8.46 and 9.45 Gbps, respectively, while in the extreme non-uniform setting (when all EFs are between routers B and C), the the average EF throughput for SH and DH are 3.68 and 8.71 Gbps, respectively. With SH service, each EF lasts longer as there are more concurrent EFs, and hence each EF receives lower throughput.

Finally, consider the metric *DH Gain Factor*, which is defined as the ratio of average EF throughput under DH service to that under SH service. This factor is plotted against EF-arrival rate in Fig. 3.7b. At higher EF arrival rates and under non-uniform traffic patterns, the DH gain factor is higher, reaching about 2.4 (= 8.71 Gbps/3.68 Gbps) at the highest load. The Pareto distribution shape parameter, α, does not have a significant impact, but if the mean EF size is larger, e.g., 40 GB instead of 20 GB, the DH gain factor is higher as seen in Fig. 3.7b. This is because EF durations will be longer, and hence there is a higher probability of overlapping EFs under the SH service.

In *summary*, the higher the load, or greater the degree of non-uniformity of traffic patterns, the greater the average EF throughput gain when using DH relative to SH service.

3.5 Conclusions

This work addressed the question of whether provider-link headroom can be used effectively by customer EFs without adversely affecting best-effort traffic and the ability of provider networks to absorb extra traffic during failures. By marking EF-packet headers with a lower-priority service class, providers can offer a new Static Headroom (SH) service to allow customers to fully leverage their access-link capacities. When EFs use SH service instead of BE service, EF-induced packet losses in general-purpose IP traffic are eliminated. Our simulation study showed that if the burstiness level of the general-purpose IP traffic was 0.572, then even a low EF-arrival rate of 0.083/sec can cause a 0.8% packet loss rate if BE is used for EFs. A second simulation study showed that with Dynamic Headroom (DH)

service, wherein a low-utilization path is computed for EFs, the average EF throughput gain factor of DH over SH was a factor of 2.4 under a highly non-uniform traffic pattern.

Chapter 4

Calibers: A Bandwidth Calendaring Paradigm For Science Workflows

4.1 Introduction

Scientific analysis in experiments such as high-energy physics or climate modeling, usually involve extremely complex workflows to ensure successful and reliable results. These workflows include a number of tasks, involve multiple actors, software and infrastructures, that work together as a workflow from data generation to delivery. For example, in the Advanced Light source (ALS), data is generated from multiple detectors which is then collected on the National Energy Research Scientific Computing Center (NERSC) via high-speed network connections. It is imperative that the data is delivered in a timely manner, with minimum loss, such that further computations can be performed using supercomputing resources that have to be *a priori* reserved. In order that the supercomputing resources are maximally utilized, this requires the network service to allow deadlines for large data transfers.

There are two approaches to ensure predictable performance and scheduled delivery for data transfers. One approach is to use advanced reservations of links, such as On-Demand

Secure Circuits and Advance Reservation System (OSCARS) or open NSA [65], that allow setting up circuits of specified capacities between routers. Advanced reservation schemes require additional time to setup circuits, are only associated with WAN border routers and are difficult to automate due to required user knowledge, network topology and request details. Furthermore, applications do not generate traffic all the time which leads to wasted reserved capacity.

The second approach is to run the network at a low utilization and use standard TCP. New TCP protocols, such as HTCP [23] and BBR-TCP [66], can efficiently adapt to the bottleneck capacity. When multiple competing flows are involved, they *equally* split the bottleneck capacity. However, even with the new TCP algorithms, sustained bottlenecks lead to unpredictable throughput performance and difficulties in arbitrarily splitting bottlenecked bandwidths among competing flows. Finally, as the growth in data transfer volume outpaces the increase in the data link rates, running the network at low utilization is not cost effective [67].

To help accelerate the effort to run the network at high utilization and enable scheduled data transfers, network automation through SDN are being advanced to control network traffic depending on data demand. In principle, SDN allow individual switches to be managed and controlled following centralized traffic engineering principles [21]. Furthermore, SDN switches provide the ability to pace traffic at ingress of the network. These features in addition to TCP protocol, or the pacing algorithm at the source nodes [68], together provide the necessary tools to dynamically allocate bandwidth to flows for meeting deadlines while ensuring the network operates at high utilization [69, 70].

This study aims to implement a centralized traffic engineering approach and control distributed agents at the edge (ingress point of the network) to dynamically pace flow for meeting transfer deadlines, while achieving high network utilization. The dynamic pacing algorithm is able to analyze traffic patterns and follow a rolling horizon model to pace flows at appropriate rates to optimize network performance and meet deadlines. A significant portion of this chapter is an excerpt from our published work *Calibers: A bandwidth calendaring paradigm for science workflows* [28].

Following are the main contributions of this work:

1. We describe an architecture that implements bandwidth calendaring for scientific workflows. The architectures leverage SDN switches that can pace flows at the ingress point. The architecture implements a central controller with distributed agents at the edge of the network that monitor flow performance and implement dynamic flow pacing set by the controller.

2. We propose different heuristic algorithms based on combining two orthogonal principles (i) local vs global optimization and (ii) Shortest Job First (SJF) vs Longest Job First (LJF). We perform a preliminary performance comparison of these algorithms with respect to a performance metric efficacy that is defined as the ratio of the request success rate to the wasted bandwidth. Our results show that simple heuristics, that optimize locally on the most bottlenecked link can perform almost as well as heuristics that attempt to optimize globally.

The remainder of this chapter is organized as follows. In Section 4.2, we present the architecture of Calibers in a software defined network. In Section 4.3 we present work on a dynamic flow pacing algorithm and present preliminary simulation results. In Section 4.5, we present the related work followed by conclusions in Section 4.6.

4.2 Architecture

Fig. 4.1 shows Calibers architecture. Each site has a workflow orchestrator that sends transfer requests to Calibers and receives flow pacing updates. Calibers consists of the following components:

- A REST/JSON API: Orchestrators use this API to schedule file transfers. They provide source, destination, file size, deadline and maximum I/O rate of the endpoints of the transfer.

- An event publisher: Allows orchestrators to obtain real-time information on the maximum rate the network has allocated to data transfers without experiencing network congestion.

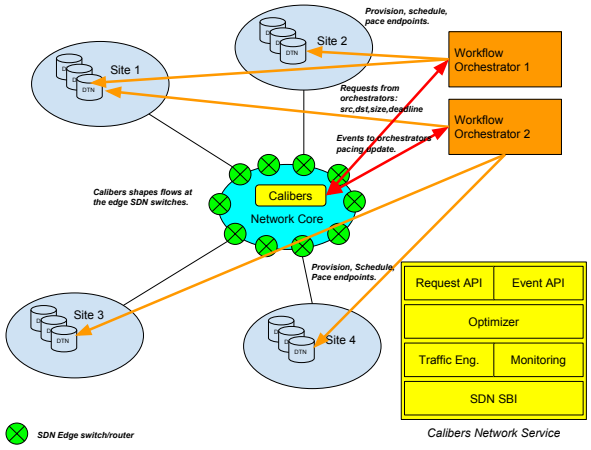


Figure 4.1: Calibers architecture illustrating the various components.

- SDN-based rate shaper: This component enforces flows to not exceed their bandwidth allocation.

- Calibers optimization algorithm: This dynamically adjusts the maximum rate of each flow, ensuring that all flows are on track to meet their respective deadlines. This in turn increases network utilization and maximizes the request admission rate.

Calibers is an experimental network service that makes several assumptions to realize its goals. These assumptions are not always true in a production environment. For example, Calibers assumes that endpoints are sufficiently provisioned with I/O, networking and processing resources; and Calibers is given a minimum guaranteed capacity on the overall network and therefore prevents it to be impacted by non Calibers traffic.

When the flow pacing rate is dynamically changed at the network edge, it may result in packet loss which may result in throughput loss. Note that TCP algorithms such as HTCP and BBR-TCP can quickly adapt to these changes. Additionally, new source pacing algorithms based on model predictive control [71] can also be used to ensure that the source quickly adapts to edge pacing rates. In this work, we focus on the scheduling algorithms, refer to our work [3] for more details on the prototype of Calibers, pacing techniques, and experimental results.

Table 4.1: Notation used.

Symbol	Description
t_{now}	Current scheduling epoch
U	Set of requests
u_i	User i request which is defined by 5 tuples $(IP_{src}, IP_{dst}, S, t^d_{f_i})$ where IP_{src} and IP_{dst} are the source and destination IP addresses, S is the data size, and t_d is the deadline
F	The set of the currently scheduled flows
f_i	flow i $\in F$
p_i	A path p_i is sequence of links corresponds to f_i source and destination pair
$R^{min}_{f_i}$	For flow f_i, this is the minimum rate that will guarantee that the deadline is met. This is $S/(t_d - t_{now})$
$R^{alloc}_{f_i}$	The rate allocated for flow f_i
$R^{slack}_{f_i}$	The slack rate assigned to flow f_i, $R^{slack}_{f_i} = R^{alloc}_{f_i} - R^{min}_{f_i}$
L	The set of the links in the network
l_i	Link i in the network, $l_i \in L$
l^C_i	Link l_i capacity
l^F_i	List of flows span link l_i
$R^{resid}_{l_i}$	The residual capacity for link l_i. $R^{resid}_{l_i} = l^C_i$ - $\sum_{\forall f_i \in l^F_i} R^{alloc}_{f_i}$
$l_{bottleneck}$	The link with the flow that has the maximum t_c
$t^c_{f_i}$	The completion time of flow f_i, which depends on $R^{alloc}_{f_i}$

4.3 Scheduling Algorithm for Dynamic Pacing

The notations used are shown in Table 4.1. The objective of the scheduler is to decrease the number of rejected data transfer requests while increasing the network utilization. This can be achieved by minimizing the sum of the completion time of all flows. This will push the completion time of all flows to the left (considering a time line), which increases the link utilization and frees up resources to accommodate future requests. The objective function is given by

$$\min \sum_{f_i \in U} t^c_{f_i} \quad \text{subject to}$$

$$\sum_{f_i \in l^F_i} R^{alloc}_{f_i} \leq l^C_i \forall l_i \in involved\_links$$

$$t^c_{f_i} \leq t^d_{f_i} \forall f_i \in involved\_flows$$

where $involved\_links$ is the set of links that the new requests traverse, and $involved\_flows$ is the set of flows that span the $involved\_links$. As the number of requests and the size of the network increases, the solution to the objective function is not guaranteed to converge. Hence, we propose four heuristic schedulers that also aim to minimize the sum of the completion time of the flows.

The scheduler operates at fixed discrete epochs with the following assumptions: (i) each link has a free capacity l_i^C to be used by the scheduler, (ii) start time for each data transfer request is immediate, i.e., the scheduler does not support advance reservation, and (ii) the scheduler updates the network status periodically every scheduling interval (epoch).

The scheduling problem is divided into to sub-problems: (i) *new flow*: when a new request arrives, how to decide whether to accept or reject the request? and (ii) *completed flow*: when a request completes, how to distribute the free capacity among the ongoing flows? We study four heuristic algorithms by combining two concepts: (i) global and local optimization and (ii) Shortest Job First (SJF) and Longest Job First (LJF).

In the global approach, the scheduler consider all the flows when distributing any residual capacity. On the other hand, the local approach focus on the bottleneck links in the network and distribute the residual capacity by reallocating locally only the flows that span the bottleneck link(s). The LJF and SJF are known concepts where longest jobs are favored with LJF and shortest jobs are favored when SJF is used. Those concepts are used by both the global and local scheduler in the following way. When the scheduler decides (locally or globally) which flows should be considered when distributing the residual capacity, SJF or LJF will be used to decide the order in which the flows will be assigned the residual capacity.

The scheduler keeps track of multiple parameters as shown in Table 4.1. One of the most important parameter the scheduler uses to make decision is $R_{f_i}^{min}$, which is the minimum required rate to ensure the flow f_i will not miss its deadline. The pseudo-code of both the global and local-approach is provided in Alg. 1. Both schedulers use the same approach when a new request arrives, however, they differ in the way the capacity is redistributed when a request completes.

4.3.1 Approach 1: Global Optimization

Sub-problem 1: *new flow*, when a new flow f_i corresponding to request u_i arrives, the scheduler computes $R_{f_i}^{min}$, and checks if the residual bandwidth in the flow path $R_{p_i}^{resid}$ is greater than $R_{f_i}^{min}$, it assign $R_{p_i}^{resid}$ to the new flow as shown in line 2-8 of Alg. 1. The scheduler gives the maximum available rate to the new request instead of giving it $R_{f_i}^{min}$ for two reasons: (i) to increase the link utilization, and (ii) to complete the file transfer as soon as possible in order to free up the resources to accept future requests. If $R_{f_i}^{min}$ is not available, the scheduler move to the second phase, which is pacing other flows in order to accept the new flow.

The pacing phase is shown in line 10-26 of Alg. 1, where for each link l_i in the path p_i of the flow f_i, the scheduler finds the list of flows l_i^F span link l_i. SJF or LJF concepts are used to decide which flow(s) of the list l_i^F to pace (slow down). When using SJF, the scheduler favors short flows with longest flows being slowed down first and vice versa. The scheduler paces the first flow in the sorted list by taking its slack rate $R_{f_j}^{slack}$ and assigns it to the new flow. If the first flow slack ($R_{f_j}^{slack}$) in the sorted list is less than the new flow required minimum rate ($R_{f_i}^{min}$), then the scheduler takes the slack rate of the second flow in the sorted list until the sum of the slack rates is equal to the new flow required minimum rate, or until there are no more flows in the sorted list. Hence, the request will be rejected because even with pacing, $R_{f_i}^{min}$ cannot be assigned to the new flow.

Sub-problem 2: *completed flow*, at the beginning of each epoch, the scheduler checks if a flow has completed by checking the flow completion time t_c. The flow completion time is a dynamic parameter, that changes based on the allocated rate ($R_{f_i}^{alloc}$). For all the flows completed at the scheduling epoch, the scheduler traverse the path of each completed flow and finds the set of other flows, that span the links in the path (involved_flows). After finding all involved flows, the scheduler now has a global view and starts distributing the residual capacity using SJF or LJF concepts. For example, if SJF is used, the scheduler finds the flow with the shortest completion time and assign to it the residual bandwidth available in its path then move to the next shortest flow and so on. The procedure of the global optimization approach is shown in Alg. 1, line 40-45.

Algorithm 1: Dynamic Pacing Scheduler

Input : U
remove_completed_flows(t_now)
foreach $u_i \in U$ **do**
  $R_{p_i}^{resid} = min(R_{l_i}^{resid}\ \forall l_i \in p_i)$
  if $R_{p_i}^{resid}$ ¡ $R_{f_i}^{min}$ **then**
    pace(f_i)
  else
    $R_{f_i}^{alloc} = min\{R_{f_i}^{max}, R_{p_i}^{resid}\}$
    return success
Function *pace(f_i)*
  $\forall l_i \in p_i$
  $R_{temp} = 0$
  $R_{temp}^{p_i} = []$
  found = False
  involved_flows = l_i^F
  sorted_involved_flows = sort the list in ascending (if LJF) or descending (if SJF) order based on t_c
  foreach $f_j \in sorted\_involved\_flows$ **do**
    $R_{temp} = R_{temp} + R_{f_j}^{slack}$
    if $R_{temp} \geq R_{f_i}^{min}$ **then**
      found = True
      add R_{temp} to $R_{temp}^{p_i}$
      break
  if *found = True* **then**
    $R_{f_i}^{alloc} = min(R_{temp}^{p_i})$
    return success
  else
    return reject
Function *remove_completed_flows()*
  involved_flows = []
  involved_links = []
  foreach $f_i \in F$ **do**
    if $t_c = t_{now}$ **then**
      $\forall l_j \in p_i$
      remove f_i from l_j^F and F
      add l_j to involved_links
      add to l_j^F involved_flows
    if *local-sched* **then**
      local_reshape(involved_links)
  if *global-sched* **then**
    global_reshape(involved_flows)
Function *global_reshape(involved_flows)*
  $R_{f_i}^{alloc} = R_{f_i}^{min}\ \forall f_i \in$ involved_flows
  sorted_involved_flows = sort the list in ascending (if SJF) or descending (if LJF) order based on t_c
  foreach $f_i \in sorted\_involved\_flows$ **do**
    $R_{p_i}^{resid} = min(R_{l_i}^{resid}\ \forall l_i \in p_i)$
    $R_{f_i}^{alloc} = R_{f_i}^{alloc} + R_{p_i}^{resid}$
Function *local_reshape(involved_links)*
  find $l_{bottleneck}$
  $R_{f_i}^{alloc} = R_{f_i}^{min}\ \forall f_i \in l_{bottleneck}$
  involved_flows = $l_{bottleneck}^F$
  sorted_involved_flows = sort the list in ascending (if SJF) or descending (if LJF) order based on t_c
  foreach $f_i \in sorted\_involved\_flows$ **do**
    $R_{p_i}^{resid} = min(R_{l_i}^{resid}\ \forall l_i \in p_i)$
    $R_{f_i}^{alloc} = R_{f_i}^{alloc} + R_{p_i}^{resid}$

4.3.2 Approach 2: Local Optimization

As mentioned earlier, the same approach is used when a new request arrives. However, the local scheduler takes a different approach when a flow completes.

Sub-problem 2: *completed flow*, at the beginning of each epoch, the scheduler checks if a flow has completed. The scheduler finds the bottleneck link ($l_{bottleneck}$) from the paths of all completed flows. The link which has a flow with the maximum $t_{f_i}^c$, is the link that will stay busy the longest, hence, is the bottleneck link that might cause future requests to be rejected. If multiple links have a flow with the same maximum $t_{f_i}^c$, then the link

with the highest average $t^c_{f_i}$ (without considering the maximum $t^c_{f_i}$) is decided to be the bottleneck link. By freeing up only the bottleneck link the probability of accepting flows in the future increases. The scheduler considers only the flows spanning the bottleneck link when distributing the residual capacity, which in contrast to the global approach, where scheduler considers all flows spanning all links of all completed flows paths. The procedure of the local optimization approach is shown in Alg. 1, line 46-53.

4.3.3 Algorithm complexity

The complexity of the worst-case scenario for new flows is $O(FL)$ for both the global and local optimization approaches as both algorithms follow the same approach when a new request arrives. When a new request arrives, the algorithm finds the new flow path, then it goes through all the flows traversing each link of the path to find if a residual bandwidth is available. The worst-case is when the path of the new request includes all the links (L), and each link is traversed by all the flows (F).

In the reshape phase when a flow is completed, both the local and global optimization approaches have a worst-case complexity of $O(F^2L)$. The worst-case scenario in the global approach is when the completed flow path consists of all the links of the network (L), which means all the flows (F) are considered for reshaping. Hence, for each flow f_i in F, each link $l_i$in the path p_i of f_i is visited, and each flow traverses l_i is visited to find the residual bandwidth (if any), which yields to a complexity of $O(F^2L)$.

In the local optimization approach, even though only the bottleneck link is considered, the worst-case scenario occurs when the bottleneck link is traversed by all the flows on the network (F). Thus, all the flows will be considered for reshaping which would result on the same complexity of the global optimization. However, as the size of network increases, the probability of having all the flows (or large number of flows) traversing the bottleneck link is lower compared to the global approach where all the flows traversing the paths (many links) of the completed flow(s) are considered.

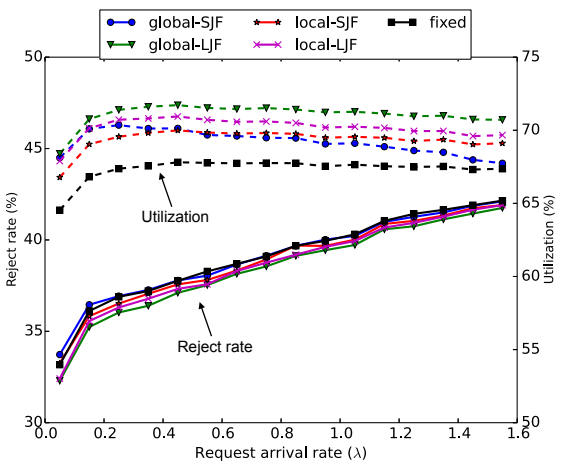
4.4 Simulation Analysis

Flow-level simulation was conducted to evaluate the performance of the four schedulers: (i) local-SJF, (ii) local-LJF, (iii) global-SJF, and (iv) global-LJF. The simulator was written using Python and for each simulation setup, 10 runs were executed where in each run 30k requests were generated.

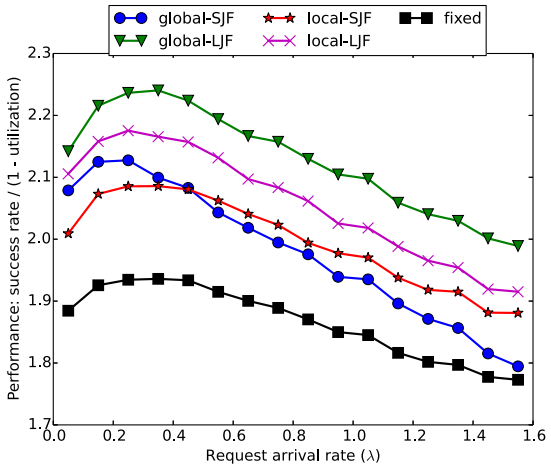
4.4.1 Simulation Setup

Network Google's inter-data center network G-scale [21] with 12 nodes and 19 links was used to evaluate. The link capacity l_i^C was set to 10 Gbps for all links in the network. ***Workload*** Requests were generated as follow: (i) Request inter-arrival time was modeled with an exponential distribution with arrival rate λ varying between 0.05 to 1.6 with step of 0.1, i.e. the mean inter-arrival time between requests varies from 20 sec to 0.625 sec. (ii) Request deadline time was modeled following an exponential distribution with average deadline (t_d) of 1 hour. (iii) As the file size is related to the deadline, it was modeled as follows [70]. First, a transfer rate (i.e. $R_{f_i}^{min}$) is sampled following an exponential distribution with average rate of 100 Mbps. Next the file size was computed as transfer rate $\times$ t_d. This results in a product distribution with a mean file size of 45 GB. (iv) Source and destination pairs were picked uniformly.

Metrics Three performance metrics were measured: (i) Network utilization is computed by measuring the link utilization per second for each link in the network $l_i^{utilization}(t)$, then taking the average utilization for each link across the entire simulation time ($\forall l_i \in L$, $L_i^{utilization} = \text{mean}(l_i^{utilization}(t))$). Finally, the network utilization is measured as the average of $L_i^{utilization}$ for all the links in the network (network utilization = mean$[L_i^{utilization}\ \forall l_i \in L]$). (ii) Reject rate which is defined as the number of rejected requests divided by the total number of requests. (iii) Performance index is defined as the ratio of the request success rate to the wasted bandwidth (Performance index = $\frac{success\ rate}{(1-Utilization)}$).



(a) Reject rate and network utilization



(b) Performance index

Figure 4.2: Performance comparison of the four algorithms for the G-scale network.

4.4.2 Results

Figure 4.2 shows the performance of five schedulers for the G-scale network with mean file size of 45 GB (transfer rate of 100 Mbps) and an epoch of 3 minutes. The four heuristic schedulers are compared against a fixed scheduler which assigns R_{min} for each flow and does not dynamically change any flow rate.

Figure 4.2a shows that the reject rate increases as the request arrival rate increases. Also, the global-LJF has the highest utilization while the fixed scheduler has the lowest utilization because the fixed scheduler assigns R_{min} for each flow and does not utilize the residual bandwidth.

Figure 4.2b shows the performance difference between the five schedulers where the fixed scheduler is achieving the lowest performance. For example, with the low arrival rate of 0.05, the global-LJF scheduler outperformed the fixed scheduler by 20% (2.16-1.80)/1.80).

Many observations can be made on the performance of the four heuristic schedulers, *first*, while the request arrival rate increases, the performance increases until the arrival rate reaches 0.4 where the performance starts decreasing. This is because at a low arrival rate, the number of requests arriving is low, therefore, the link utilization is low. However, as the arrival rate increases, the reject rate increases, which yields to performance degradation.

Second, the global-LJF has a slightly higher performance compared to the local-LJF. The global scheduler consider all the flows when distributing the residual bandwidth. On the other hand, the local scheduler consider only the flows spanning the bottleneck link, which results on smaller number of considered flows. If the considered flows cannot utilize the residual bandwidth (because of other bottleneck links in their paths), then the residual bandwidth is wasted, hence, the performance decreases. With the simulated network, the performance difference between global-LJF and local-LJF is negligible, where global-LJF outperformed local-LJF by only 3%. This shows that redistributing the capacity only in the bottleneck link of the path of the completed flow, is enough to perform as good as when considering all flows traversing the path.

Third, by comparing global-SJF against global-LJF, or local-SJF against local-LJF, we can conclude that LJF schedulers have a better performance. LJF reduces the makespan time of all flows by assigning more rate to the flows with the longest completion time. This results on freeing up the links faster to accommodate future requests. On the other hand, SJF frees up some capacity of the link earlier than LJF but the makespan of the flows stays the same. Also, since SJF favors the flows with the lowest completion time when redistributing the residual capacity, then the probability of the flow finishing during the epoch is higher compared to if LJF is used. Therefore the flow will finish during the epoch and the capacity used by the completed flow will be wasted as no reshaping is done within the epoch. To further explore the performance difference between SJF and LJF, we executed both LJF and SJF schedulers with a small epoch duration of 1-sec. The results showed a negligible performance difference between SJF and LJF. SJF assigns more bandwidth to the shortest flows, which could result on the completion of the shortest flows at the beginning of the epoch. However, since the epoch is small (1-sec), only a fraction of the bandwidth could be wasted because in the next epoch (after 1-sec) the scheduler will redistribute the residual bandwidth, which increases the performance of SJF.

In summary, the local optimization approach performs as good as the global optimization. The performance when using SJF versus LJF is negligible in short epoch duration (e.g., 1-sec), but the performance improvement shows with longer epoch duration, therefore, we choose local-LJF as the best heuristic for scheduling compared to the other three heuristics.

4.5 Related Work

The over-arching goal of this work is to enable deadline-aware network service, while ensuring high network utilization. We leveraged SDN with the ability to perform dynamic traffic pacing at the network edge. There are a number of recent studies with similar goals. In the following paragraphs, we review the related work and point out the key differences from our work.

There has been a number of prior studies on flow pacing [72–74]. Broadly speaking, flow pacing can be performed at the source host or at the edge where the access network connects to the core network. The former is referred to as host pacing, or more commonly TCP pacing, while the later is referred to as edge pacing and can be performed by the network service provider [74]. In this study, we propose using edge pacing both at the end hosts and network ingress switches.

SDN networks allow dynamic and centralized Traffic Engineering (TE) via flow pacing. B4 [21] presents Google's effort in leveraging SDN to centralized TE and drive links to near full utilization. As similar study SWAN [50] also improves network utilization of inter-DC WAN by scheduling the service traffic in a centralized manner. However, all of these studies do not consider deadline associated with their transfers. The study in Tempus [69] considered deadlines and developed an optimization framework to maximize the fraction of transfer delivered before deadline, ensuring fairness among all requests. This work, however also does not guarantee meeting deadlines.

In a recent study [70], deadlines have been investigated in the context of inter-data center data transfers. Building on a deadline aware network abstraction (DNA) where transfer deadlines can be specified, the study proposes AMEOBA which uses traffic shaping at the source to meet data transfer deadlines. While this study is the most similar to ours, there are a few key differences: we considered edge pacing and we showed that simple dynamic pacing algorithms that optimizes locally on the most bottleneck link perform as well more complex algorithms that attempt to optimize globally.

4.6 Conclusions

TCP congestion avoidance algorithms, while performing well at maximizing network utilization, cannot provide the desired behavior for workflow orchestration. In particular, TCP relies on network characteristic, such as RTT and packet retransmission, and ignore flow needs. As result some flows may go faster than they need, while others may go slower than they should, such to meet deadlines and maximize resource utilization. However, the performance of modern TCP, in conjunction with the ability of SDN to implement centralized traffic engineering, allows Calibers to optimize network utilization to provide predictable performance. Our preliminary study on dynamic pacing algorithms suggests that simple heuristics that optimize locally on the most congested link can perform almost as well as attempts to optimize globally.

Chapter 5

A Study on Virtual Desktop Service using Edge Clouds for Smart Communities

5.1 Introduction

Virtual desktop technologies have been growing in popularity as several remote desktop protocols, such as VNC, Microsoft Remote Desktop Protocol (RDP), and TeamViewer, have been developed to support this paradigm. Custom hardware units, called zero clients, have also been implemented to run remote desktop protocols. These hardware units are built with Application Specific Integrated Circuits (ASICs) or Field Programmable Gate Arrays (FPGAs) with limited or no CPU functionality. In principle, the zero client acts as a peripheral to its associated server with minimal local processing support beyond decryption and decoding. In practice, some local processing support or thin client functionality may be provided.

Zero client hardware has primarily been developed for enterprise applications that require many terminals with centralized control. Thus, zero clients are used in schools, hospitals, libraries, and data entry business environments [75–78]. Zero clients have also been used for high performance graphics applications to allow multiple users to share the more expensive

processing engines [79]. The graphics application frequently exploits co-location of the zero clients with the graphics servers usually connected through a LAN or other reliable network interface. Zero clients also provide secure access for data center management, usually through a zero client switch.

The introduction of edge cloud or fog computing creates a new opportunity to expand the range of potential applications for zero clients into personal or home computing (PC). The use of zero clients with edge clouds is appealing for the potential to provide high performance computing experiences at favorable costs. A zero client computing approach could enable sharing hardware and software licensing costs among many users. Each user or household would only require a zero client having network access to the edge cloud. Zero clients are less expensive than standard personal computers, and this hardware combined with access to edge servers may be a more affordable option than a PC for households without computers.

Thin clients have been marketed as affordable computing platforms, citing similar benefits [80–83]. However, consumer thin clients compromise performance and offer a lower quality computing experience for multimedia usage [84, 85]. In contrast, zero clients are designed to deliver the full computer performance of the associated server and can offer high-quality computing experiences that include support for graphic intensive applications. Moreover, the zero client approach enhances security because zero clients neither run a standard OS nor expose a CPU on which attackers can install malicious software. However, using a zero client requires an active, reliable, and high speed network for interaction between the servers and the virtual desktops. The requirement for zero client connectivity can impose a mobility challenge as network access must be available to support operation with the zero client. Fortunately, edge cloud computing is being introduced precisely to provide reliable, high speed, low latency network connections, and the so-called “mobile” edge clouds are planned to provide similar access for a mobile networking environment.

Even though virtual desktop technologies have evolved significantly over the past ten years, commercial zero clients are still limited to business and health sectors today. For example, North Kansas City Hospital deployed 700 zero clients allowing staff members to use any nearby “PC” in the hospital throughout the day to access centrally secured medical data, instead of having one dedicated PC per staff member [78]. In the health or business

sector, employees tend to mostly use a specific set of applications. On the other hand, a full PC experience is expected for residential use. As edge cloud computing becomes available, it is important to understand the viability for zero client use in this environment and what computing experience can be supported over edge clouds.

Previous work [86, 87] on virtual desktops performance was conducted with methods that depend on monitoring the performance at the end-user devices, which is not feasible with zero clients. Other proposed methods depend on monitoring the performance at the server [88–90] by monitoring CPU utilization, monitoring the display buffer to detect if a task has completed, or running commercial PC benchmarks on the server. Monitoring at the server does not consider the involvement of the network to transmit the display to the end-user devices. Some works have been done to measure VD performance by analyzing the network traffic [87, 89, 91], however, the network traffic was analyzed only to measure the video quality.

Therefore, in this chapter, we introduce performance measurement methods for zero clients that depend on analyzing the network traffic between the zero client and the server, and include not only video quality measurements, but also responsiveness as perceived by the end user. These methods do not require running software-monitoring packages at the end user device, or collect measurements at the server. We provide the results of our study on the zero client computing approach for a residential use-case to address the following concerns: (i) How to measure the zero client performance without the ability to run measurement software programs at the end-user device (ii) What is the impact of the network on the zero client performance (iii) Do objective measurements reflect user Quality of Experience (QoE) as determined through subjective measurement studies.

Objective and subjective measurements were obtained to evaluate the performance of the zero client for residential usage. Objective measurements were defined within the measurement categories of: response time, video quality, and audio quality. For response time, we defined a new metric, Virtual Desktop Display Update Time (VD-DUT), which depends on analyzing the network traffic between a server and a zero client. Video quality was measured by analyzing the network traffic and by capturing frame per second rate. Audio was captured using a hardware device connected to the zero client to measure the

performance. The subjective measurement study involved 115 participants. For both the objective and subjective studies, four activities were performed to evaluate the performance: browsing 2D images, exploring 360-degree images, watching a video, and participating in a video-conference call.

We found that the network packet loss rate has a different impact on the zero client performance based on the application, where video applications experienced the highest impact. By performing statistical analysis on the subjective measurements, we found that at 0.5% packet loss rate and higher, the subjects' quality of experience values had no statistically significant difference for all the tested applications (except the video), indicating that the subjects interpreted the quality at 0.5% packet loss rate and higher in a similar way. A strong correlation between objective and subjective measurements was found. However, as packet loss rate increased, the rate at which the objective measurements changed was different in compare to the rate at which the subjective measurements changed.

A secondary observation from this work suggests that commodity data center servers are unlikely to be adequate for use in zero client personal computing. This observation is based on the recognition that zero client performance critically depends upon server performance, and typical data center servers are optimized for serving applications rather than full desktop experiences.

Our main contributions are as follows: (i) Defined a new metric, Virtual Desktop Display Update Time (VD-DUT), to measure zero client responsiveness. (ii) Conducted the first, large-scale subjective study on zero client performance with 115 participants. (iii) Quantified the correlation between objective and subjective metrics.

A significant portion of this chapter is submitted for publishing in IEEE Access journal Section 5.2 describes the zero client computing approach and the challenges of collecting measurements. Section 5.3 describes the objective evaluation approach including metrics, setup, and applications; and Section 5.4 describes the subjective evaluation approach. Section 5.5 presents the results of both the objective and subjective studies and quantifies their correlation to one another. After reviewing related work in Section 5.6, the chapter is concluded in Section 5.7.

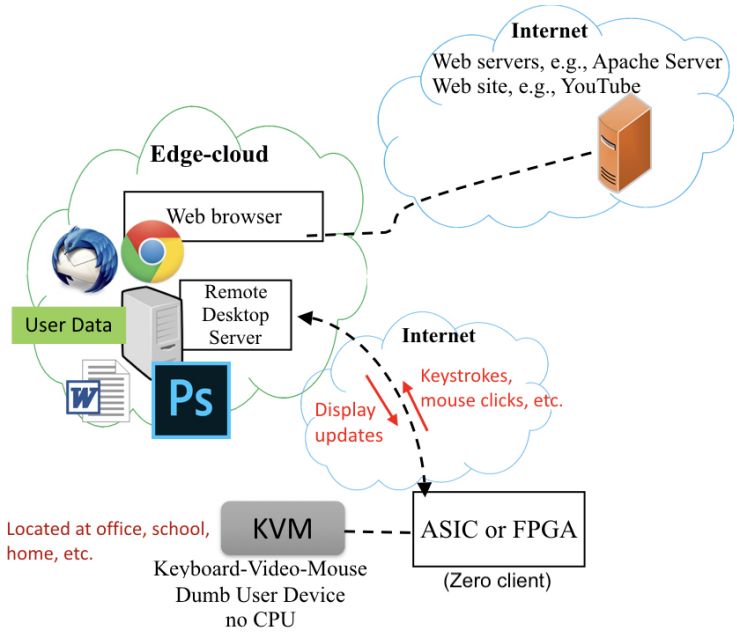


Figure 5.1: Zero client computing approach

5.2 Zero client computing approach

Fig. 5.1 illustrates the zero client computing approach. In this approach, virtual desktops are hosted on an edge cloud or directly by local/enterprise machines. The zero client uses custom hardware to drive user devices such as Keyboard Video Mouse (KVM) terminals. The zero client runs remote desktop protocols with encryption and video encoding/decoding at the user end. Supporting these protocols without exposing a general-purpose CPU in the user-owned and operated end device reduces costs while also reducing exposure to cyberattack.

PC over IP (PCoIP) [92] is a high-performance display protocol used to deliver virtual desktops to end-user devices (e.g., zero clients). PCoIP compresses, encrypts, and encodes the desktop for transmission over the IP network. Only display pixels and control signals (i.e. mouse clicks) are sent over the network with all the processing being executed on a remote desktop server. This protocol has a hardware implementation (the zero client).

Measuring the performance of a zero client is challenging because measurement software cannot be installed and run on the zero client itself, since it has no standard operating system that allows running user applications. Also, performance measurement at the edge

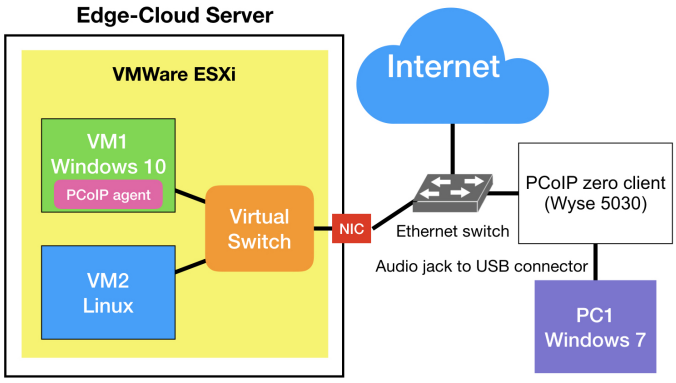


Figure 5.2: Setup

cloud (i.e. the remote desktop server) may not provide a faithful representation of the zero client performance. For example, consider a zero client connected via the network to a virtual desktop running on a Virtual Machine (VM) hosted on a server, measuring the response time by AppTimer at the VM as done by vSIP study [88], will measure the time to launch an application when a display physically connected to the server rather than the time to see the application window at a remote zero client. With this technique, the time needed for the VM to open and display the application window at the server is measured. However, this time will not include the time to compress the display pixels, the time to send the display pixels over the network, the time for the zero client to receive, decode, and paint the display pixels. Therefore, running measurement software tools at the server could lead to inaccurate results. Since monitoring applications cannot be run on the zero client, new monitoring approaches are needed.

5.3 Objective evaluation approach

An objective evaluation of the zero client computing model was conducted by measuring the performance of the system while running different applications. User input was emulated using `Autoit` [93], which is a scripting language used to automate Windows GUI input by simulating keystrokes, mouse movements and clicks. Measurements were obtained from within the edge-cloud host (server), from packet traces between the edge-cloud host and the zero client, and by capturing the audio output of the zero client.

5.3.1 Setup

Fig. 5.2 shows the setup used to run experiments and obtain measurements. An ASUS STRIX laptop was configured to operate as the edge-cloud server with VMWare ESXi 6.5.0, 16 GB RAM, four Intel i7 2.80GHz multithreading CPUs, and a 1 Gbps Network Interface Card (NIC). Two VMs were created within the server: (i) VM1 with Windows 10 OS, and (ii) VM2 with Ubuntu 16.04. The two VMs were connected via a virtual switch within the server. Port mirroring cannot be configured within an ESXi virtual switch, so we configured the virtual switch using the promiscuous mode to allow VM2 to receive all packets. This configuration supported monitoring and processing of VM1-client network traffic. We used the Wyse 5030, which supports PCoIP and equipped with Teradici TERA2321 chip, as the zero client. Both the server and the zero client were connected via an Ethernet switch with 1 Gbps ports. We used a USB audio capture device to obtain the audio output from the zero client. The audio jack from the zero client was connected through USB to a PC (PC1) running Audacity to record the audio from the zero client.

5.3.2 Metrics

Different metrics were used to measure the performance of the system. The metrics can be divided into 3 categories:

- Response Time (RT)
- Video Quality (VQ)
- Audio Quality (AQ)

Response Time We used the slow-motion technique proposed by Nieh at al. [94] to measure the system response time. The (emulated) user initiates a single task and then waits for the server to perform the task. The user further waits for the response to be received and painted on the display before the user initiates another task. This technique allows for the network packets associated with each task to be identified within the traffic trace captured between the server and client. For example, Fig. 5.3 shows the network traffic from a server to a zero client with a 20-sec gap between the first and second task and a 40-sec gap between

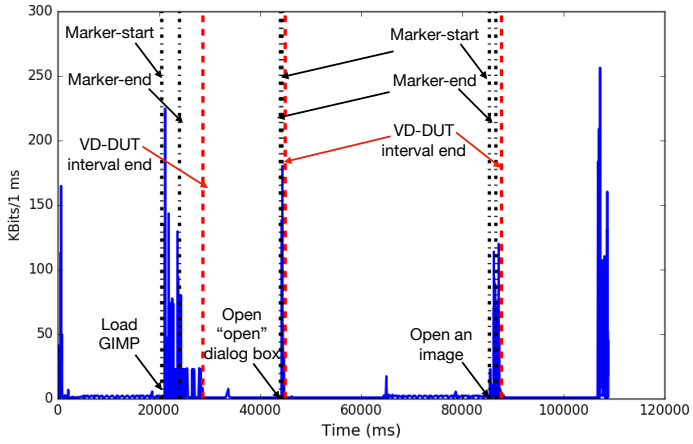


Figure 5.3: Network traffic capture from server to the client when three tasks were performed

the second and the third tasks. Response times are computed from the time instants of traffic spikes. RT metrics include: RT-Autoit, RT-Marker and VD-DUT.

RT-Autoit is measured at the VM running in the edge-cloud host using an `Autoit` built-in function to detect when a task has completed. For example, `Autoit` uses the `WinWaitActive()` function to detect the rendering of a window for a launched application. This function locks the execution thread until `Autoit` detects that the application window has appeared on the display. When `Autoit` calls the `WinWaitActive` function, it is checking the VM frame buffer at the server to check if the application window has appeared on the display. Our traffic analysis showed that this does not include the time taken to send all the display updates to the client and the time taken by the client to process and draw the display.

RT-Marker is measured using the VDBench [95] method. This method sends a marker packet (UDP packet to a predefined port) before performing a task. When `Autoit` detects that the task has been performed, another marker packet is sent to indicate the end of the task. From the collected network trace, RT-Marker is obtained by computing the time between the two marker packets. Since there is background traffic between the edge-cloud and zero client to support the protocol, we need a threshold (τ) to identify the end of a display update.

VD-DUT, our newly defined metric, also sends a marker packet to indicate the start time of a task. However, instead of relying on an end marker packet, the VD-DUT computation

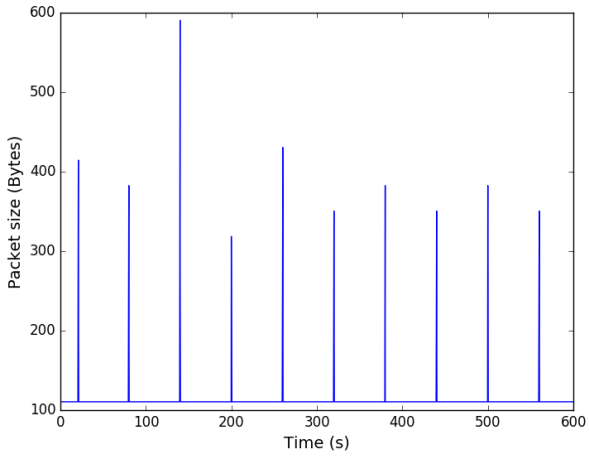


Figure 5.4: Packet size from the server to the client under idle condition with no display updates

method finds the last display update packet in the traffic trace.

Example: Fig. 5.3 shows the network traffic trace from a server to a client when the following tasks were performed: (i) sleep for 20 sec, (ii) send a start marker packet, (iii) load GIMP application (which is a photo editor), (iv) send an end marker packet, (v) sleep for 20 sec, (vi) send a start marker packet, (vii) open the "Open" dialog box, (viii) send an end marker packet, (ix) sleep for 20 sec, (x) choose a picture, (xi) send a start marker packet, (xii) click on "open" button to load the picture, (xiii) send an end marker packet, (xiv) sleep for 20 sec. Fig. 5.3 shows that, for task 1 (loading GIMP), even after the end marker packet was sent, more packets were sent from the server to the client. Hence, to consider all the sent display updates when computing response time, we used a simple heuristic to decide which packets were part of the display update and should be considered when computing VD-DUT.

To determine an appropriate value for τ for VD-DUT computation, we examined the network traffic from the server to the client. We found two types of packets: (i) periodic, small PCoIP communication packets, and (ii) display update packets. To understand the characteristics of the small periodic packets, we characterize the traffic from the server to the client under idle condition.

Fig. 5.4 shows a 10-min snippet of a collected packet trace showing packets sent from the server to the client under idle condition. The horizontal line in the plot represents small

continuous 110-byte packets sent with a short inter-arrival time (on the order of hundreds of milliseconds). On the other hand, each vertical line represents two larger packets (of size 200 - 600 bytes) sent back-to-back approximately every 1 minute (the two packets overlap in the plot). In the illustrated 10-min packet trace, there were 1417 packets of size 110-byte, and only 20 packets with a size larger than 110-byte. Therefore, we built our heuristic based on the smaller packets by assuming that any packet with a size greater than 110-byte could be a display update packet.

The threshold to measure VD-DUT was defined based on the inter-arrival time of packets with a size greater than 110-byte. If the inter-arrival time between two consecutive, larger than 110-byte packets (p_i and p_{i+1}) is greater than 500 ms, then the second packet p_{i+1} is not part of the display update and VD-DUT is determined from the time at which the start marker packet was sent to the time at which p_i packet was sent.

We chose 500 ms as our threshold after examining the inter-arrival time between the large packets (>110-byte) in many captured packet traces between the edge-cloud host and the zero client. We know that large packets sent between the start and end marker packets were task-related packets since we only send a start marker packet after performing a task at the VM running in the edge-cloud host. We found the mean inter-arrival time between these large display update packets was 5 ms, the 99th percentile was 175 ms, and the maximum inter-arrival time was 490 ms. Since the inter-arrival time between the large packets under idle condition was approximately 1-min, we chose a number larger than 490 ms but smaller than 1-min. Specifically we chose 500 ms. To conduct a sensitivity analysis, we redid the analysis with 1-sec, but found the same results, therefore, we chose 500 ms

VD-DUT has its limitations in measuring the display time because it does not consider: (i) the time from a mouse click until the packet carrying the mouse click is sent, (ii) the time taken to send the mouse click from the zero client to the server, and (iii) the time the zero client takes to receive, decode, and paint the display. The second time can be considered by adding half the Round Trip Time to VD-DUT, however, the first and third times are difficult to measure because the zero client is a commodity hardware with no general purpose CPU to run any monitoring software within it.

VD-DUT can be broken down into the server processing time (or RT-Autoit), transmission

time, and retransmission time as shown in (5.1). T_{trans} is computed by dividing the total display update transmitted bytes by the link rate (1 Gbps).

$$\text{VD-DUT} = T_{proc} + T_{trans} + T_{retrans} \tag{5.1}$$

VQ To measure the video quality, we used two metrics: (i) received PCoIP frame per second (recv-PCoIP-fps), and (ii) video quality measured following Nieh et al. [94] equation (5.2) (slow-mo-VQ). Frame per second rate is a good representation of the video quality because PCoIP adjusts fps based on the network condition. To obtain fps data, running software at the zero client to capture fps data is not feasible given the zero client limitations. However, PCoIP has a session statistics viewer (SSV) tool which collects fps data of the zero client. On the other hand, slow-mo-VQ can be generally used regardless of the remote desktop protocol and it does not depend on a specific tool. It depends on analyzing the network traffic, where P in (5.2) represents the tested video. Slow-mo-VQ compares the slow-motion video to the regular speed video to quantify how many frames were dropped or not transmitted by looking at the total bytes transferred and the time required to play the video. A video is first played back at 1 fps and a network trace is captured. The video is then replayed at regular speed over various network conditions. The video playback at the low fps rate is used to establish a reference data transfer rate (total data transferred divided by the total playback time) corresponds to a perfect video playback with no dropped frames. Then, compare the data transfer rate of the regular fps video to the reference transfer rate, where the data transfer rate is assumed to reflect the video quality.

$$\text{slow-mo-VQ} = \frac{\frac{Data\ Transferred(P)\ /\ Playback\ Time(P)}{Ideal\ FPS(P)}}{\frac{Data\ Transferred(\text{slow-mo})\ /\ Playback\ Time(\text{slow-mo})}{Ideal\ FPS(\text{slow-mo})}} \tag{5.2}$$

AQ A survey on perceptual quality assessment for audio/visual services [96] showed that the video is the dominant factor that decides QoE when watching videos, however, in a video-conference call, the dominant factor is the audio quality. Therefore, for video playback testing, only the video quality was measured, and for the video-conference call, only audio quality was considered when measuring the performance.

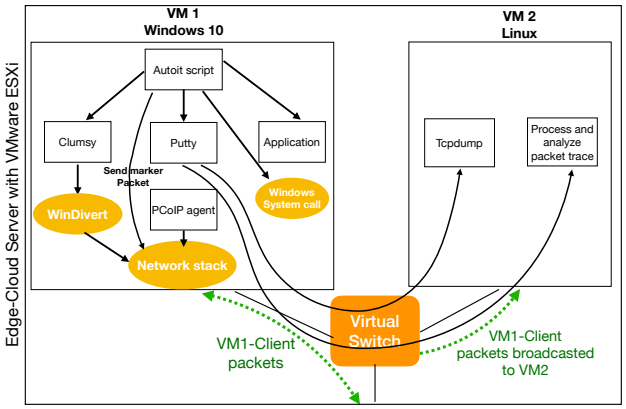


Figure 5.5: Application automation process

To measure the audio quality, we used three objective audio/speech evaluation metrics: (i) Weighted Spectral Slope (WSS) [97], (ii) Log-Likelihood Ratio (LLR) [98], and (iii) Virtual Speech Quality Objective Listener (ViSQOL) [99]; which are signal-based, full reference metrics. WSS and LLR were not developed with VoIP in mind, on the other hand, ViSQOL was designed to be a general objective speech quality metric with a focus on VoIP. ViSQOL was developed as an alternative to the commercial, industry-standard speech quality metrics: Perceptual Evaluation of Speech Quality (PESQ) [100] and Perceptual Objective Listening Quality (POLQ) [101]. Both WSS and LLR measure the distance between the reference signal and the degraded signal. ViSQOL uses the Neurogram Similarity Index Measure (NISIM) to measure the similarity between the two signals and map the result to a value within the range of 1 to 5.

5.3.3 Applications automation

Fig. 5.5 shows how different components were used to automate and execute the experiments, and how these components interacted with each other. VM1 has five main components: (i) `Autoit` script, which emulates different user activities, (ii) the application, which is being tested and controlled via an `Autoit` script, (iii) Clumsy [102], which is a network emulation tool used to change the network conditions within the VM, (iv) Putty, to allow VM1 to remotely access VM2 and initiate the network traffic monitoring script, and (v) PCoIP agent, which is used to enable the remote desktop access.

Four activities were considered for the performance evaluation study: 2D image viewing via Windows `Photos`, video playback via `MPlayer`, 360-degree image exploring via `Chrome`, and video-conference call via `Skype`. Five packet loss rate (PLR) values of 0%, 0.5% 3%, 5%, and 10% were used for the evaluation.

2D image viewing and 360-degree image exploring

Each activity was automated by an `Autoit` script as follow: (i) the `Autoit` script starts by changing the network configuration with Clumsy, which leverages Windows WinDivert package that allows user layer applications to manipulate sent and received network packets. (ii) After setting up the network, the `Autoit` script initiates tcpdump at VM2 via an ssh connection with Putty. (iii) The `Autoit` script: sends a UDP start marker packet, (iv) controls the application to perform a specific task (e.g., open an image or play a video), (v) waits for the task to be executed using Windows system calls to check the status of the task, (vi) then sends an end marker packet, (vii) stops clumsy, (viii) stops the packet capture at VM2 via Putty, and (ix) initiates an analyzing script at VM2, which parses the captured packet trace, finds the total bytes, finds VD-DUT and saves the results..

For the 2D image viewing, six images were used with a high resolution between 1024x1545 - 5719x3803, and unique pixels in the range 309K - 877K. During this test, all the images were pre-loaded in the RAM by opening them before taking measurements. In each run, all the six images were viewed one by one with a 20-sec gap in between. Each run was repeated 70 times, the geometric mean of VD-DUT and total sent bytes of the six images were computed, then the arithmetic mean values were computed across runs.

For the 360-degree image exploring, we used a web-hosted 360-degree image tool to explore the images, instead of running a 360-degree image exploring tool locally in the edge-cloud server. Our edge-cloud host is not equipped with a virtualization-supported GPU, which would impact the performance of the application profoundly. Since our goal is not to evaluate the computing resources of the edge-cloud, but rather to measure the responsiveness of the system, we used commercial-cloud 360-degree image viewing software rather than running our local copy of the software. Three images were used and explored via

Pano2VR running in a web server [1]. In each run, for each image, the `Autoit` script would open a new tab in a Chrome web-browser, visit the image URL, drag-and-drop to change the scene, zoom-in, and zoom-out, where every two tasks were separated by a 20-sec gap and repeated 90 times. For each run, VD-DUT was computed for each task performed on each image, the geometric mean of all images and all tasks was computed for each run, and then the arithmetic mean was computed across runs.

Video

To obtain video measurements, the following steps were followed: (i) at VM1, the `Autoit` script initiates tcpdump at VM2, (ii) plays a video via MPlayer at 1 fps rate , (iii) after the video finishes, the script logs the time that was taken to play the video and stops tcpdump at VM2, (iv) configures the network with Clumsy, (v) initiates tcpdump at VM2, (vi) starts SSV at VM1 to capture fps, (vii) starts the video at the regular fps rate, (viii) after the video finishes, the script logs the time that was taken to play the video and stops tcpdump at VM2, and (ix) executes the analysis script at VM2 to parse the collected packet traces by extracting total bytes and use it to compute the video quality via (5.2). Steps (iv-viii) are repeated for the different network conditions. Each run was repeated for 75 times and mean values were reported. A 36-sec video from the animated movie "Zootopia" with 23.9 fps rate was used.

Skype

Evaluating Skype performance involves performance metrics that do not require collecting network packet traces. The audio coming from the AUX jack of the zero client was captured and processed to measure the call audio quality. PC1, which is shown in Fig. 5.2, was used to emulate the other end of the Skype call and record the audio output of the zero client. A USB audio capture device was used to capture the audio output of the zero client. The captured audio was forwarded to PC1 which runs `Audacity` to record the audio.

AQ metrics require comparing the recorded audio file to a reference file. To obtain a reference file, a Skype call between two PCs was performed. An audio file was fed to

[1] https://rodedwards.com/interactive-files/Chatsworth_House/index.html

Skype at one PC, and the audio output of the second PC was recorded. The recorded audio file represented the reference audio. The above steps were repeated four times with two audio files with male and female speakers from ITU-P.862 conformance data (`u_am1s03` and `u_af1s02`). Each audio file was recorded twice to account for the variability of Skype calls. Four reference audio files were obtained from the above experiment and were used as the reference to compute the audio quality metrics (`u_af1s02_f_ref1`, `u_af1s02_f_ref2`, `u_am1s03_m_ref1`, and `u_am1s03_m_ref2`)

The following steps were taken to conduct the zero client Skype experiment: (i) a Skype call was initiated from PC1 (Fig. 5.2) to VM1 in the edge-cloud host. (ii) A master `Autoit` script at VM1 was executed. (iii) The master script starts by configuring the network using Clumsy, then connects to PC1 via Microsoft RDP. (iv) The master script then starts another `Autoit` script we developed (play-and-record script) at PC1, (v) The script at PC1 starts recording by running `Audacity` and then instantly feeds an audio file to the Skype call (`u_am1s03` or `u_af1s02`). (vi) After the audio file ends, the `Autoit` script at PC1 stops recording and exports the recorded audio as a `wav` file. (vii) Finally, the script analyzes the collected audio file by computing the AQ metrics. Each run consists of repeating the above steps five times back-to-back for each network condition. Five runs were executed before changing the network condition to give PCoIP enough time to adapt to the network changes since the used audio files have a length that varies between 7-10 sec. The experiment was repeated 50 times for each audio file (`u_am1s03` and `u_af1s02`), and the two references corresponding to each audio file were used in each run to compute the AQ metrics.

5.4 Subjective evaluation approach

We conducted studies (i) to evaluate users' subjective experiences with the zero client computing approach, and (ii) to quantify the relationship between objective and subjective measurements. A total of 115 participants (66 males and 49 females) at the University of Virginia completed the subjective study in the fall of 2018. Participants rated their experiences using the Mean Opinion Score (MOS) with a 5-point Absolute Category Rating (ACR) scale following ITU-T Recommendation P.800 and P800.1 [103,104]. Each participant

was asked to assess the QoE for each application on a scale from 1 (bad) to 5 (excellent). The same applications were used to evaluate the performances in both the objective and subjective studies.

5.4.1 Setup

Two testing stations (cubicles) were configured with identical keyboards, mice, and monitors (Dell LCD) with 1680x1050 resolution. A Wyse 5030 zero client was used in one cubicle, and an LG CBV42-B PCoIP zero client was used in the other. Both zero clients are equipped with the same Teradici TERA2321 PCoIP processor.

The arrangement for this study was similar to that used for the objective study shown in Fig. 5.2. This study includes an additional VM (VM3) with the same configurations as VM1. Also, the LG PCoIP zero client was connected to the physical Ethernet switch shown in the setup. VM2 was not used to collect packet traces during this subjective study. Subjective and objective experiments were executed during different time periods using the same arrangement.

5.4.2 Methodology

Upon arrival, each participant was seated in one of the cubicles and directed to click on an application icon located in the middle of the Desktop. The application was our master script to select study applications and collect user input. Participants first read the informed consent agreement and, if they agreed to the terms, they were directed to a short survey that captured information about their computer experience. The actual test started by asking the participant to execute three activities in sequence.

Four activities (applications) were used to evaluate the experience: (i) image viewing via Windows `Photos`, (ii) 360-degree image exploring via `Chrome`, (iii) video playback via Windows `Movies & TV`, and (iv) a video-conference call via `Skype`. The first two activities were used to collect data related to the responsiveness of the system. The other two activities collected data related to the audio and video qualities. During each activity, PLR was changed to test the performance under different network conditions as emulated by Clumsy.

Watching a video: Each participant was asked to watch a 36-sec video three times with the PLR changed for each iteration. Each participant was asked to rate each viewing iteration based on the quality of the video and audio without considering the content.

Image viewing: Each participant was asked to view the same six images that were used for the objective experiment and rate the quality of each image without considering the image content. Each participant was asked to look at each of the images three times with the PLR changed for each of the views.

360-degree image exploring: Each participant was asked to explore three 360-degree images. Every 15 seconds during the exploration, a window appeared asking the participant to rate the responsiveness of the system and the quality of the images. A new PLR value was established before each image was presented for exploration.

Skype: Each participant was asked to join a 2-min video call via Skype with one of two research assistants. Every 40 seconds during the call, a window appeared asking the participant to rate the call quality. Call variability was limited during the test by having the research assistance receiving the call always sitting in the same room, using the same laptop and connected to the same network. We controlled the call conversation by asking the participant to play the "20 questions" game. The participant would think of a person or item, and the research assistant would have 20 questions to ask to identify the person or item. We chose this interaction instead of using a written script because we wanted the participant to focus on the call quality rather than reading a script. The research assistant asked the questions to the participant to ensure that the participant was listening and paying attention to the audio quality.

Upon completion of each activity, the participant was invited to continue with the next activity or opt out. Thus, participants had the option of rating their experiences for one, two or three activities. The activities were automated using an `Autoit` script. Participant interaction with the study personnel was not necessary, and this helped to reduce any influence of study personal over participant ratings. The automation also maintained test consistency and controlled the testing time. The automation script performs the following tasks: (i) it shows the participant a dialog box to describe the activity, it runs the tested application (e.g., visits the 360-degree image URL, or opens the directory that includes the

2D images that need to be explored), it interrupts the test at specific time intervals to ask about user experience, and it changes the network configuration.

5.4.3 Data analysis approach

MOS values for each combination of applications and PLR values were computed and reported in Sec. 5.5.2. To study the subjective measurements, we conducted a pairwise t-test to check if the MOS values across different PLR values are significantly different. We used the t-test to test the null hypothesis that there is no difference between the mean values (MOS) across PLR.

Because of the repeated measures in our study where the same subject rates the experience under different network conditions, a subject dependency is expected. The subject dependency in the results occurred because each participant provided experience ratings for 3 PLR values; the subject might have rated the experience with a 3% PLR while recalling the previous experience with a 0% PLR baseline. To account for the subject dependency in our study, we performed further analysis using a Linear Mixed effect Model (LMM). LMM has two types of effects: fixed and random. We used PLR as our fixed effect and the subject as our random effect ($QoE \sim PLR + subject$). We conducted Tukey's Honestly Significant Difference (HSD) test to check if the differences between the groups (PLR) were significant. HSD adjusts the p-value based on the total number of pairwise comparisons. It is very conservative with respect to Type I error (rejecting the null hypothesis when it is true). We used R package "lme4" version 1.1-19 to fit the data to LMM.

5.5 Results

5.5.1 Objective evaluation results

Image viewing Fig. 5.6 shows the mean of the different RT metrics. In the figure, when the packet loss rate increases, VD-DUT increases. On the other hand, both RT-Marker and RT-Autoit remained constant when the packet loss rate increased because both of these timers are based on monitoring the server frame buffer, hence, network activities would

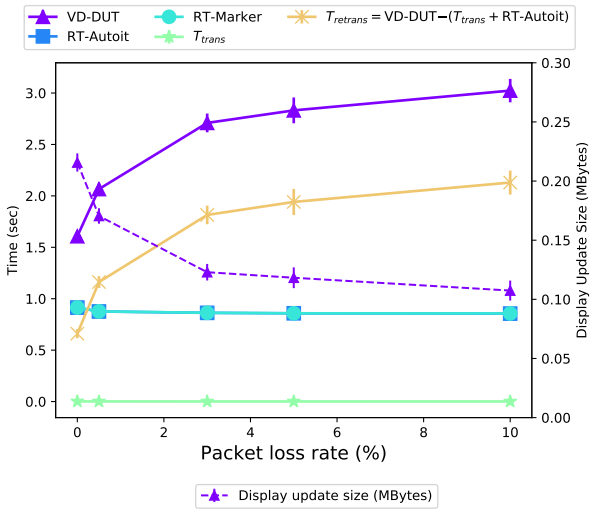


Figure 5.6: Response Time breakout for image viewing. RT-Autoit and RT-Marker are overlapping in the plot

not impact the timers. RT-Autoit and RT-Marker could be affected by the processing time within the server, e.g., if many applications are sharing the VM CPU resources, then RT-Autoit and RT-Marker could log increases in the time.

Since processing time (RT-Autoit) and transmission delay (T_{trans}) are almost constant under different PLR conditions, and by using (5.1), we could conclude that the increase of the delay in VD-DUT is due to the retransmission time. This time also includes the time of processing the packets at the client and determining the missing display pixels.

We also note from the plot that VD-DUT is increasing even though the total sent bytes are decreasing as the packet loss rate increases. It is expected that if the total bytes decreases, VD-DUT will decrease since fewer packets are sent. However, VD-DUT is increasing because of the extra time required to retransmit the packets. Also, the total number of the transmitted bytes is also decreasing due to PCoIP decreasing the display resolution when detecting the high packet loss rate. Therefore, users would notice an increase in display update time and decrease in the display resolution with high packet loss rate even with a simple 2D image viewing activity.

360-degree image exploring Fig. 5.7 shows mean values of VD-DUT and display update size. Similar to the 2D image viewing, VD-DUT for the 360-degree images increases as the packet loss rate increases. However, the increase rate was lower compared to the 2D image

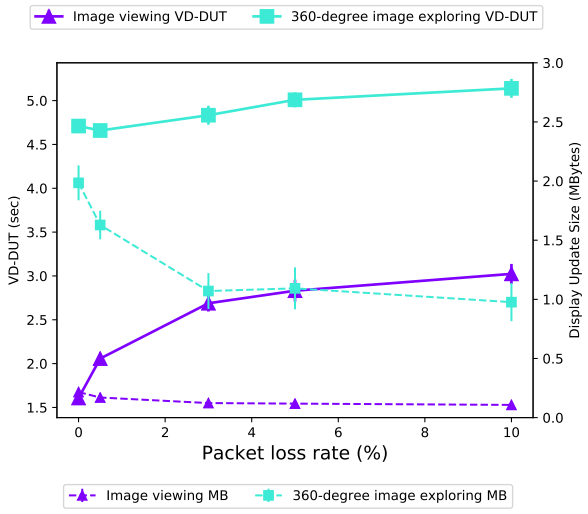


Figure 5.7: 360-degree image exploring

viewing (9.1% increase rate between 0 to 10% packet loss rate for the 360-degree images, whereas for 2D images the increase rate was 46.7%). This behavior could be due to the nature of the 360 images in which the display is changing rapidly; hence PCoIP is not taking time to retransmit the lost packets, as with the 2D images, once we display an image, the display stays unchanged for some time.

Skype Fig. 5.8 shows Skype performance measured using different AQ metrics. In general, AQ metrics values were decreasing as PLR increased. The changing rate in the audio quality when PLR increased from 0% to 10%, was 47.04% for ViSQOL, 4.59% for WSS, and 7.65% for LLR. Both LLR and WSS showed low-value changes when PLR increased. On the other hand, ViSQOL showed a wider range of values when PLR increased. At PLR of 0%, ViSQOL was computed to be 3.2, which is approximately the average ViSQOL value based on the ViSQOL range (ViSQOL was designed to have a maximum value of 5 and a minimum of 1). This indicates that video-conference calls via zero clients have an average quality under the ideal network condition.

Video Fig. 5.9 shows the video quality measured via slow-mo-VQ and recv-PCoIP-fps rates, where both metrics were found to be decreasing as PLR increased. At PLR 3, 5, and 10% the changes in recv-PCoIP-fps and slow-mo-VQ were minimal compared to the changes between 0, 0.5 and 3%. The video quality dropped when PLR changed from 0% to 10%

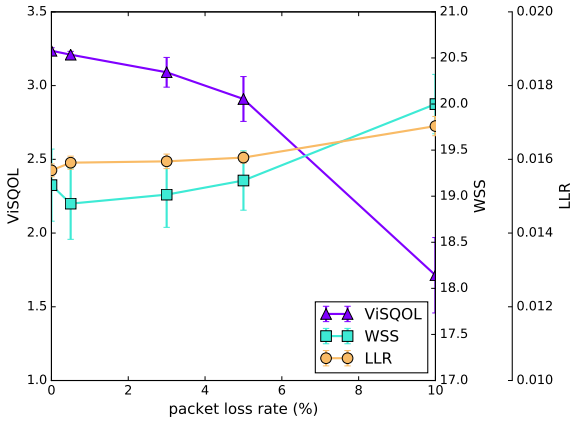


Figure 5.8: Skype AQ measurements obtained via three different metrics

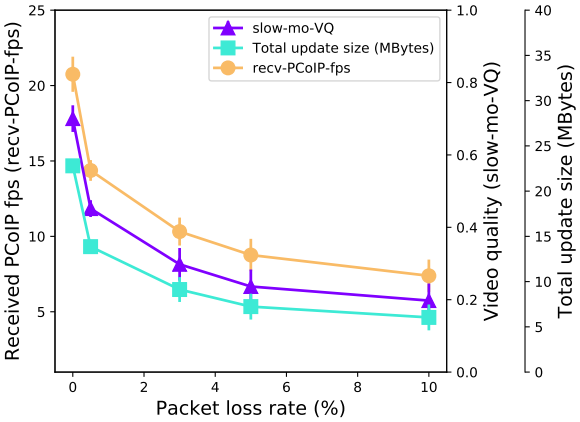


Figure 5.9: Video quality across different PLR

at a rate of 64.4% for recv-PCoIP-fps and 71.8% for slow-mo-VQ, where both show a high drop rate when PLR increased. At a 0% PLR, the video reached 20.75 recv-PCoIP-fps rate, which is less than the original video rate (23.9), similarly, slow-mo-VQ achieved 70%. The amount of received data decreased to 6 MB (note that the bytes were captured after the packets were dropped by the network emulator). The low amount of received data implies many frames were dropped or partially dropped (some pixels were dropped which makes constructing the frame difficult) which explains the very low recv-PCoIP-fps rate of 7.4 at 10% PLR.

Table 5.1: Total number of collected QoE values for each activity and packet loss rate value

	packet loss rate (PLR) %				
Activity	0	0.5	3	5	10
Image viewing	75	31	44	44	31
360-degree image exploring	58	24	34	34	24
Video playback	66	28	38	38	28
Skype	53	26	27	27	26

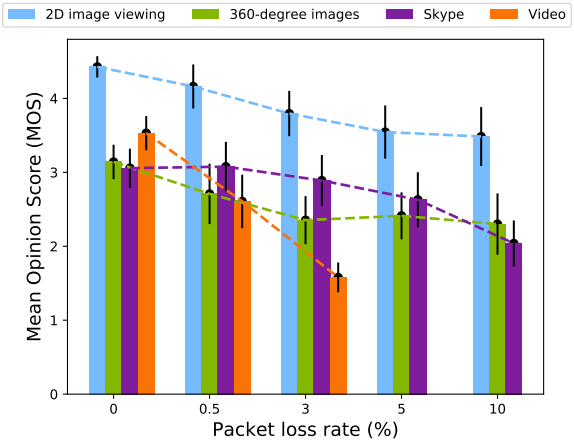


Figure 5.10: MOS values of different applications with 95% confidence interval

5.5.2 Subjective evaluation results

Table 5.1 shows the total number of collected QoE values corresponding to each tested PLR case. The 0% PLR case has a higher number of collected QoE values because both stations had 0% PLR as an initial condition, whereas the other PLR values were divided to be tested between the two stations. The minimum number of collected QoE values is 24 (ITU recommends that a minimum of 15 participants are required to evaluate image quality on a screen [105]).

MOS analysis Fig. 5.10 shows MOS values for different applications with 95% confidence intervals (CI). Image viewing is the only application that achieved MOS value higher than 4 with 0% PLR (MOS=4.43). Even with the high PLR of 10%, image viewing MOS value did not drop below 3. This indicates that when browsing through images, participants were less sensitive to decreased-resolution, still images. For 360-degree images, MOS with an ideal network condition (0% PLR) has a lower rating (MOS=3.31) compared to the 2D image viewing. The highest drop rate of MOS occurred when PLR increased from 0 to 3%. After

Table 5.2: T-test pairwise p-value for different applications

	0	0.5	3	5
0.5	1.99e-01			
3	6.74e-04	1.07e-01		
5	2.74e-06	6.95e-03	2.25e-01	
10	7.73e-06	6.13e-03	1.69e-01	7.85e-01

(a) Image Viewing

	0	0.5	3	5
0.5	9.28e-01			
3	4.53e-01	4.69e-01		
5	5.70e-02	8.59e-02	3.13e-01	
10	1.25e-05	1.08e-04	1.26e-03	2.37e-02

(b) Skype

	0	0.5	3	5
0.5	0.075699			
3	0.000317	0.188981		
5	0.000837	0.272655	0.810676	
10	0.000231	0.141992	0.834638	0.659491

(c) 360-degree image

	0	0.5
0.5	1.02e-05	
3	1.18e-19	8.65e-06

(d) Video playback

3% PLR, MOS value continued to decrease but at a lower rate.

For Skype, it is interesting to note that even with the very high PLR value of 10%, MOS value did not drop below 1. This could be due to: (i) participants expecting video-conference calls to have poor quality, and (ii) audio quality being the dominant factor when evaluating video-conference calls as shown by a previous study [96]. More bandwidth is assigned to the audio channel during video-conference calls, and the impact of 10% PLR was not as high as we expected.

The MOS for video playback showed the most dramatic decrease as it dropped to 1.58 with 3% PLR. Video applications require much data because the display changes rapidly; the objective measurements showed that the video has the highest number of sent bytes among the applications. Therefore, video playback should be the most sensitive application to PLR compared to the other tested applications. We did not collect QoE values beyond 3% PLR because the MOS value had dropped to 1.58.

T-test analysis Table 5.2 shows the p-value for a pairwise t-test conducted on the different PLR values across each application with a cutoff of 0.05. Highlighted table cells have p-values < 0.05 indicating a significant difference between the two PLR values. For image viewing, we cannot reject the null hypothesis for the consecutive pairs (0,0.5), (3,5), and (5,10). We could also see in Fig 5.10 that the decrease rate was low, i.e., MOS values were close. These results were expected considering the activity of browsing through still images. Participants could sense a difference between no packet loss (0%) and packet loss (3, 5, or 10%), but they could not necessarily sense a difference among the PLR values higher than 0%.

In contrast, for the interactive Skype application, the t-tests among the groups of PLR showed more significant differences. By comparing 0, 0.5, 3, and 5% loss rate values, we cannot conclude that participants noticed a difference in the performance. However, with the higher PLR of 10%, we can conclude that the QoE rating is significantly different compared to the lower PLR of 0, 0.5, 3 and 5%. The results from 360-degree image exploration allow rejection of the null hypothesis only when comparing 0% PLR to the other PLR values, and we can conclude that the participants had a better experience with no packet loss rate compared to any tested PLR case. Fig 9 supports these results as the QoE mean values (MOS) of 3, 5, and 10% PLR, are close. On the other hand, the video playback t-test results showed a significant difference between the QoE values across the different PLR cases with a very low p-value between 0, and 3% (1.18e-19).

LMM with HSD test analysis For different applications, Fig. 5.11 shows the 95% CI of the difference between the mean values for each PLR pair. The mean values were obtained using the LMM, and CI lines were computed by applying an HSD test between the mean values for each group (PLR case). In the plots, if CI includes zero within its range, then we cannot conclude that the two means are statically significantly different because there is a chance that the difference between the two mean values of a PLR pair is zero. Fig. 5.11 shows more conservative results compared to the t-test results reported in Table 5.2. For the video, the HSD test on the fitted LMM showed similar results to the t-test such that there were significant differences between the mean across all PLR pairs. For the other three applications, there were statically significant differences between the mean values when PLR

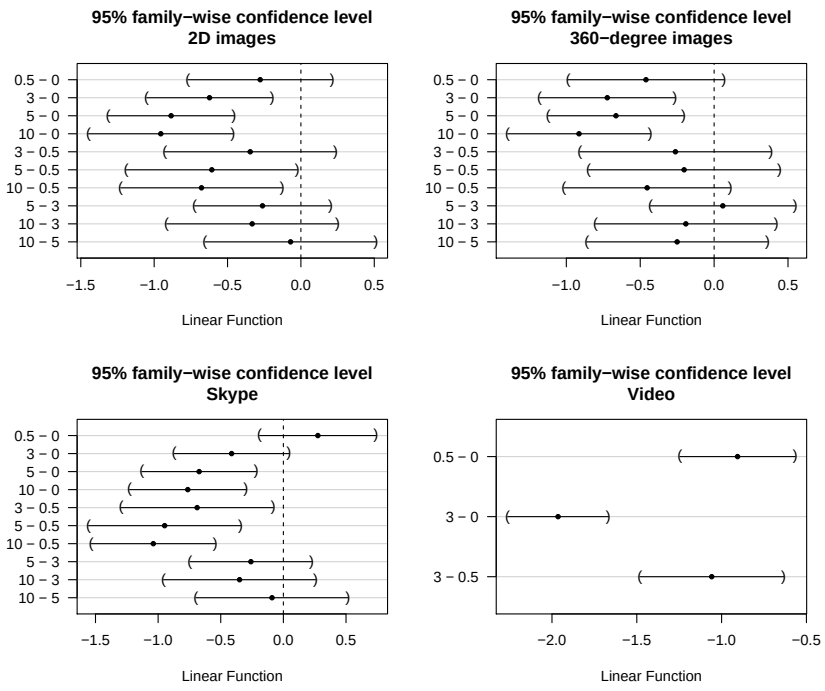


Figure 5.11: Pairwise 95% confidence interval of the difference between every two mean values across PLR using a Linear Mixed-effect Model with HSD test

was 0% and when PLR was greater than 0%. Even though for the other PLR pairs, we cannot reject the null hypothesis, Fig5.11 shows that the difference between MOS values across many pairwise PLR groups is near-marginal significance (for some CI, only a small part of the line crossed 0).

Subjective and objective correlation To study the correlation between the objective (VD-DUT, VQ, AQ) and subjective MOS values, we used Pearson's correlation coefficient (r). For the tested applications, r values were found to be as follows: image viewing (r = -0.990), 360-degree image (r=-0.733), Skype-LLR (r=-0.963), Skype-WSS (r=-0.946), Skype-ViSQOL (r=0.974), video-recv-PCoIP-fps (r=0.987), video-slow-mo-VQ (r=0.986). These results imply strong correlations between the objective metrics and the subjective MOS values. However, the rates of increase/decrease in the objective and the corresponding subjective metric were different when PLR increased.

Table 5.3 shows the slope of a linear model for each application and method (LMM was used to fit the subjective results). For image viewing, both objective and subjective metrics have approximately the same absolute line slope indicating that PLR impacted

Table 5.3: The slope of fitted linear models of each application for both objective and subjective measurements

App	**Method**	**slope**
Image viewing	Objective-VD-DUT	0.1260
	Subjective	-0.1041
360-degree image	Objective-VD-DUT	0.0483
	Subjective	-0.0866
Skype	Objective-ViSQOL	-0.1494
	Objective-WSS	0.0961
	Objective-LLR	0.0001
	Subjective	-0.0979
Video	Objective-recv-PCoIP-fps	-2.9060
	Objective-slow-mo-VQ	-0.1102
	Subjective	-0.6565

both metrics at the same rate. For Skype, subjective measurements, ViSQOL, and WSS have approximately the same increase/decrease rate caused by PLR increase. On the other hand, LLR has a minimal slope value, indicating a minimum impact of PLR on the performance (i.e., LLR underestimated the effect of PLR compared to the subjective results). For 360-degree image exploring, the subjective metric (MOS) was decreasing faster than the rate at which the objective metric (VD-DUT) was increasing indicating a different impact by PLR (i.e., the subjective metric was more sensitive to PLR changes compared to the objective metric (VD-DUT)). For the video analysis, we only used the objective data points collected at PLR of 0, 0.5, and 3% since we only collected subjective data at these three PLR values. Comparing the objective recv-PCoIP-fps and slow-mo-VQ to the subjective QoE results, the subjective metric was decreasing when PLR increased at a rate higher than slow-mo-VQ decreasing rate, and at a rate lower than recv-PCoIP-fps decreasing rate. In conclusion, nevertheless, objective metrics showed a decreasing trend on the performance as PLR increased (which matches the subjective results), some objective metrics underestimated or overestimated the impact of PLR on the performance.

5.6 Related Work

Many studies have been conducted on objective measurements to quantify virtual desktop performance and/or to evaluate new solutions for virtual desktops. Nieh et al. [94] proposed

a methodology to measure the performance of thin clients via slow-motion techniques based on monitoring network traffic. Packet traces are collected in an ideal environment as a baseline and then compared against packet traces collected under different network conditions and server loads. Slow-motion techniques have been used by other researchers as well [87, 89, 91, 95, 106].

VDBench [95] is a thin-client benchmarking tool that uses slow-motion techniques. Realistic loads of multiple applications were generated to compare different remote desktop protocols by measuring video quality and application response time under various server loads and network conditions. CloudRank-V [107] is another benchmarking tool that uses network traces to find response time (latency). A method to generate complex workloads was proposed by mixing nine applications, and the maximum number of VMs the server can execute before user performance degradation is noticeable was determined. Our objective study is similar to VDBench and CloudRank-V, where we measured response time based on analyzing network traffic by defining our own metric VD-DUT (VDBench response time is based on when the action is performed at the server side, on the other hand, CloudRank-V threshold to define start and end of display updates was not stated). Despite the previous work on measuring the performance of virtual desktops, to the best of our knowledge, this is the first study on virtual desktops accessed through zero clients as studies used thin clients or other computing platforms.

VNCPlya [108] and DeskBench [86] are VD benchmarking tools that measure application response time based on the status of the display buffer. When a specific task is performed (e.g., opening a text editor), the display buffer is captured and used as a reference. To measure the response time, the tool performs the same specific reference task, and then keeps on comparing the display buffer to the reference until they match (DeskBench uses a hash function of the display buffer instead of using the image itself). Pandey et al. [109] proposed a framework to facilitate VD benchmarking and allow adaptation to changes in VDI software architecture. Song et al., [87] presented FastDesk, a remote desktop system for multi-tenants, which was evaluated by measuring CPU utilization, response time and video quality for different applications. However, all previous works rely on running a piece of code on the client to monitor the display buffer or capture mouse clicks and display changes,

which are not feasible in a zero client setup.

Sui et al. [88] evaluated their proposed virtual scheduler for interactive performance (vSIP) at the server side without considering the remote desktop protocol nor the client end-device. The evaluation metrics included video quality measured by the rate of dropped frames at the server side, cold and warm launch time of applications, and Web page loading time. Server-side measurements were also obtained in Zhou et al. study [90]. Our goal is to evaluate the end-user device performance; thus, server-side measurements are not sufficient.

Some studies focused only on video quality and developed methods to measure it as the only metric [89, 90, 106, 110, 111]. For example, Laine et al. [110] used displayed image frames, the frame rate and play duration as performance metrics. Yu et al. [106] used a modified slow-motion video quality metric.

Subjective assessments have been used to measure user quality of experience [88, 112–118]. The number of participants on the assessments varied between 10 to 40 and Mean Opinion Score (MOS) was used to evaluate the performance. These studies focused only on the quality of video or gaming experience [119], other applications in addition to video were considered in our subjective study.

Casas et al. [25] undertook a subjective study with 52 participants to measure QoE using Citrix technologies. Each participant performed several tasks (text editing, drag and drop, scroll down, and Web browsing) and evaluated the experience using MOS under different RTTs. The authors characterized traffic by collecting packet traces, measured the response time, and compared the response time when using Citrix with response time when running the applications on a desktop. Our work is similar to Casas et al. as different applications were considered. However, our study focused on zero client performance and in addition to the subjective study we conducted a correlation study on the subjective and objective results. Also, we conducted a Skype subjective conversation-opinion test on VDs specifically with zero clients setup.

In the VDpilot study [120], virtual desktop performance was evaluated by 38 participants to asses QoE of using some applications via a virtual desktop compared to being in a lab to access the applications via a physical desktop in the lab. Five applications were evaluated, where the participants used their own devices to access the virtual desktops over a WAN

connection. In contrast, our study focused on the zero client performance and also quantified the impact of the network on the QoE via subjective and objective studies.

Previous work on subjective Voice over IP (VoIP) performance evaluation were conducted with audio or video files fed into the call, without interaction with a person on the other side of the call [121–125]. This test is defined as the listening quality test [103], which achieve less degree of realism compared to the more complex conversation-opinion test. Very few studies have been conducted using conversation opinion test. For example, Cano et al. [126] conducted a subjective study on Skype performance with real calls, but for a different purpose of evaluating four different VoIP applications. Khitmoh et al. [127] and Daengsi [128] performed subjective studies on VoIP service where every two subjects had a 3-5 minute conversation on a controlled laboratory setup to develop a model for VoIP quality evaluation for the Thai language.

5.7 Conclusions

Objective and subjective measurements were obtained to evaluate zero client performance in which four activities were performed: (i) image viewing, (ii) 360-degree image exploring, (iii) video playback, and (iv) video-conference calling. Many objective metrics were used: Virtual Desktop Display Update Time (VD-DUT), Audio Quality (AQ) metrics, and Video Quality (VQ) metrics. VD-DUT, our newly defined metric, is measured by analyzing the network traffic between the zero client and the edge cloud; and it was used to measure the system responsiveness for the first two activities. Methodologies to measure AQ and VQ were also described, and experiments were conducted to measure the video quality and the audio quality of a Skype call. The first large-scale subjective study on zero client performance was conducted with 115 participants at the University of Virginia, in which Mean Opinion Score values were collected and analyzed. The network conditions were altered during the objective and subjective studies by increasing the packet loss rate (PLR).

The PLR impact on the zero client performance varied based on the application. By analyzing the objective measurements, we found that VD-DUT increased and both AQ and VQ decreased when PLR increased, as expected. The video experienced the highest

impact from the PLR as the video quality (measured by recv-PCoIP-fps) decreased by a rate of 71.8% when the PLR changed from 0% to 10%. Statistical analysis conducted on the subjective measurements showed that the MOS values at 0.5% PLR and higher were not statistically significantly different, implying that the subjects interpreted the quality at 0.5% PLR and higher in a similar way. Video is the exception, where MOS values across different PLRs were significantly different and the impact of PLRs on the subjects' experiences was the highest as MOSs dropped from 3.53 (at 0% PLR) to 1.58 (at 3%), decreasing at a rate of 55.2%. A strong correlation between the objective and subjective measurements was found but the speed at which the objective and subjective measurements fell with increasing PLR differed depending on the application.

Chapter 6

Conclusions and Future Work

In this chapter, we first summarize the work presented in this dissertation and draw four key conclusions, and then discuss potential future work to advance our current research.

6.1 Summary and Conclusions

We designed and evaluated new networking services, taking into account deployment constraints, so that these services can be introduced incrementally into different regions of the Internet for improved application performance. Given the large number of network technologies and even larger number of deployed networks, we selected networks within the following three categories: (i) datacenter networks, (ii) Wide-Area Networks (WANs), and (iii) Local-Area Networks (LANs). For each network type, we defined problems that address specific application needs, then proposed and evaluated our evolutionary solutions.

Chapter 2 presented a measurement study of congestion on a production, highly utilized, 72K-core InfiniBand cluster called Yellowstone. In datacenter networks, low-latency communications are required for scientific, highly parallelized applications. We developed a methodology for measuring congestion and executed this methodology in Yellowstone. The measurement study consists of a 23-day data collection phase in which port counters of the Yellowstone switches were read multiple times every hour to check for stalls during which the port is unable to send data due to a lack of flow-control credits. A total of 30M data records were obtained and analyzed. Our *conclusion* is that congestion occurs in production clusters

where we found that in a significant number of the 100-ms intervals over which a port counter was observed, there were transmission stalls. Out of 6M observations of Top-of-Rack (ToR) switch uplink ports, we found that the port was forced to wait for credits in 60% of these 100-ms intervals. Such transmission stalls could increase application execution time, and also decrease cluster utilization.

Chapter 3 described our proposed SDN-enabled headroom services: (i) Static Headroom (SH) and (ii) Dynamic Headroom (DH) services to allow customers to fill provider-link headroom with Elephant Flows (EFs) without adversely affecting the provider's ability to meet its Best-Effort (BE) service-level agreements, and ability to absorb extra traffic load created during failure recovery periods. Our solution calls for the use of lower-priority service for EFs. We used simulations to compare SH service with BE service, and DH service with SH service. When EFs are sent on BE service, they could cause packet losses in general-purpose IP traffic, especially when the burstiness of the latter is high, while with SH service, this packet loss rate is reduced to 0. While DH service requires the added complexity of a provider SDN controller, the ability to dynamically route EFs on lower-utilized links results in higher average EF throughput. The higher the non-uniformity (from a node-pair perspective) in network traffic, the greater the DH gain factor. Therefore, we *conclude* that under low-to-moderate traffic loads, provider-link headroom can be used effectively by customer elephant flows without adversely affecting best-effort traffic and the ability of provider networks to absorb extra traffic during failures.

Chapter 4 presented Calibers, our developed solution to allow scheduled delivery across WANs. Calibers uses shaping, metering, and pacing at the edge of networks and end-systems to allow scheduled delivery. We introduced four scheduling algorithms by combining two principles: (i) local and global optimization, and (ii) Shortest Job First and Longest Job First. We compared the performance of these algorithms using simulations. We *concluded* that a simple heuristic, which optimizes usage locally on the most bottlenecked link, can perform almost as well as heuristics that attempt global optimizations.

Chapter 5 presented our measurement studies conducted to evaluate desktop application performance when using zero-client based virtual-desktop services. We defined a new metric, Virtual Desktop Display Update Time (VD-DUT), which depends on analyzing the network

traffic between the zero client and the edge cloud, and used this metric to characterize system responsiveness. Both objective and subjective measurement studies were conducted, the results were analyzed, and the correlation between the two measurements was quantified. During the objective and subjective studies, network conditions were altered by changing the emulated packet loss rate (PLR). We *concluded* that PLR impact on the zero-client performance varied considerably based on the application. Video applications experienced the highest impact of packet losses, and we found a strong correlation between objective and subjective measurements.

6.2 Future Work

This work can be extended in the following directions:

1. The InfiniBand measurements study can be extended by undertaking a thorough simulation study to understand the conditions (combinations of network traffic) under which congestion occurs, and to quantify the relationship between Forced Idle Time Fraction (FITF), a measure of congestion, and application performance.

2. For large data transfers using our new headroom services across WANs, a simulation study can be conducted to evaluate a merged solution of SH and DH services that leverages knowledge of network utilization and file sizes.

3. For Calibers, the scheduling algorithms can be prototyped and integrated with the prototyped architecture, and then evaluated experimentally. The work assumed prior knowledge of background traffic. Hence, methods to predict background traffic can be investigated.

4. For the virtual-desktop measurements study, our newly proposed metric can be used to execute a scalability study, in which both network and computing resources are considered. A small experimental study is required to first characterize and model the network traffic and computing-resource usage. A subsequent simulation study can be carried out to quantify scalability.

Bibliography

[1] Mellanox Technologies. Interconnect Your Future, Enabling the Best Datacenter Return on Investment. https://www.mellanox.com/related-docs/solutions/hpc/TOP500-January-2019.pdf, 2019.

[2] Eli Dart, Lauren Rotman, Brian Tierney, Mary Hester, and Jason Zurawski. The science DMZ: A network design pattern for data-intensive science. In *Proc. of SC'13*, pages 85:1–85:10. ACM, 2013.

[3] Fatma Alali, Nathan Hanford, Eric Pouyoul, Mariam Kiran, Dipak Ghosal, Rajkumar Kettimuthu, and Ben Mackcrane. Calibers: A Bandwidth Calendaring Paradigm For Science Workflows. In *Innovating the Network for Data Intensive Science (INDIS) 2017*. SC17 Workshop, 2017.

[4] Fabrice Mizero, Malathi Veeraraghavan, Qian Liu, Robert Russell, and John Dennis. A Dynamic Congestion Management System for InfiniBand Networks. *Supercomputing frontiers and innovations*, 3(2), 2016.

[5] CAIDA. http://www.caida.org/.

[6] Van Jacobson and Michael J. Karels. Congestion avoidance and control. In *Proceedings of the Sigcomm '88 Symposium*, pages 314–329, 1988.

[7] Mark Handley. Why the internet only just works. *BT Technology Journal*, 24(3):119–129, 2006.

[8] Enric Pujol, Philipp Richter, and Anja Feldmann. Understanding the share of IPv6 traffic in a dual-stack ISP. In *International Conference on Passive and Active Network Measurement*, pages 3–16. Springer, 2017.

[9] Amogh Dhamdhere, Matthew Luckie, Bradley Huffaker, Ahmed Elmokashfi, Emile Aben, et al. Measuring the deployment of IPv6: topology, routing and performance. In *Proceedings of the 2012 Internet Measurement Conference*, pages 537–550. ACM, 2012.

[10] Google IPv6 Statistics. https://www.google.com/intl/en/ipv6/statistics.html#tab=ipv6-adoption.

[11] National Science Foundation. Future Internet Architectures (FIA). https://www.nsf.gov/pubs/2010/nsf10528/nsf10528.pdf, 2010.

[12] Lixia Zhang, Deborah Estrin, Jeffrey Burke, Van Jacobson, James D Thornton, Diana K Smetters, Beichuan Zhang, Gene Tsudik, Dan Massey, Christos Papadopoulos, et al. Named data networking (NDN) project. *Relatório Técnico NDN-0001, Xerox Palo Alto Research Center-PARC*, 157:158, 2010.

[13] Jennifer Rexford and Constantine Dovrolis. Future Internet architecture: clean-slate versus evolutionary research. *Communications of the ACM*, 53(9):36–40, 2010.

[14] Top500 List Statistics. https://www.top500.org/statistics/list/, November 2018.

[15] Torsten Hoefler, Timo Schneider, and Andrew Lumsdaine. The impact of network noise at large-scale communication performance. In *Parallel & Distributed Processing IPDPS,IEEE Intl. Symp. on*, pages 1–8. IEEE, 2009.

[16] Abhinav Bhatele, Kathryn Mohror, Steven H. Langer, and Katherine E. Isaacs. There goes the neighborhood: Performance degradation due to nearby jobs. In *Proceedings of the Intl. Conf. on High Perf. Computing, Networking, Storage and Analysis*, SC '13, 2013.

[17] G Pfister, M Gusat, W Denzel, D Craddock, N Ni, W Rooney, T Engbersen, R Luijten, R Krishnamurthy, and J Duato. Solving hot spot contention using InfiniBand architecture congestion control. In *High Perf. Interconnects for Dist. Comp., Research Triangle Park,2005*.

[18] DOE/FERMI National Accelerator Laboratory. The sensitive giant at CERN, ATLAS effort emphasizing people skills. https://www.eurekalert.org/features/doe/2004-03/dnal-tsg032604.php, 2004.

[19] European Bioinformatics Institute (EMBL-EBI). European Bioinformatics Institute - Cambridge, Scientific Report 2017. https://www.ebi.ac.uk/sites/ebi.ac.uk/files/groups/external_relations/EMBL-EBI_Scientific_Report_2017_DIGITAL_LR.pdf, 2017.

[20] European Bioinformatics Institute (EMBL-EBI). The European Bioinformatics Institute Impact Report 2016. https://www.ebi.ac.uk/sites/ebi.ac.uk/files/groups/external_relations/Documents/EMBL-EBI_Impact_Report_2016_web.pdf, 2016.

[21] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, et al. B4: Experience with a globally-deployed software defined WAN. *ACM SIGCOMM Computer Communication Review*, 43(4):3–14, 2013.

[22] Shuangcheng Niu, Jidong Zhai, Xiaosong Ma, Mingliang Liu, Yan Zhai, Wenguang Chen, and Weimin Zheng. Employing checkpoint to improve job scheduling in large-scale systems. In *Workshop on Job Scheduling Strategies for Parallel Processing*, pages 36–55. Springer, 2012.

[23] D. Leith and R. Shorten. H-TCP: TCP Congestion Control for High Bandwidth-Delay Product Paths, IETF Internet Draft draft-leith-tcp-htcp-03, June 2007.

[24] ESnet testbed. http://es.net/network-r-and-d/experimental-network-testbeds/100g-sdn-testbed/.

[25] Pedro Casas, Michael Seufert, Sebastian Egger, and Raimund Schatz. Quality of experience in remote virtual desktop services. In *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on*, pages 1352–1357. IEEE, 2013.

[26] ©2017 IEEE. Reprinted, with permission, from, Fatma Alali, Fabrice Mizero, Malathi Veeraraghavan, and John M Dennis. A measurement study of congestion in an InfiniBand network. In *Network Traffic Measurement and Analysis Conference (TMA), 2017*, pages 1–9. IEEE, 2017.

[27] ©2017 IEEE. Reprinted, with permission, from, Fatma Alali, Xiao Lin, Malathi Veeraraghavan, Naoaki Yamanaka, and Weiqiang Sun. SDN-enabled headroom services for high-speed data transfers. In *Communications (APCC), 2017 23rd Asia-Pacific Conference on*, pages 1–6. IEEE, 2017.

[28] Fatma Alali, Nathan Hanford, Eric Pouyoul, Mariam Kiran, Dipak Ghosal, Rajkumar Kettimuthu, and Ben Mackcrane. Calibers: A Bandwidth Calendaring Paradigm For Science Workflows. In *Special Issue on "Innovating the Network for Data Intensive Science - INDIS 2017" in Future Generation Computer Systems. Pending.* Elsevier, 2018.

[29] M. Gusat, D. Craddock, W. Denzel, T. Engbersen, N. Ni, G. Pfister, W. Rooney, and J. Duato. Congestion control in InfiniBand networks. In *High Perf. Interconnects (HOTI), 13th Symp. on*, pages 158–159, 2005.

[30] Shihang Yan, Geyong Min, and I. Awan. An enhanced congestion control mechanism in InfiniBand networks for high performance computing systems. In *Adv. Info. Networking and App.(AINA) 20th Intl. Conf. on*, 2006.

[31] E.G. Gran, M. Eimot, S.-A. Reinemo, T. Skeie, O. Lysne, L.P. Huse, and G. Shainer. First experiences with congestion control in InfiniBand hardware. In *Parallel Dist. Proc. (IPDPS), IEEE Intl. Symp. on*, 2010.

[32] Ernst Gunnar Gran and Sven-Arne Reinemo. InfiniBand Congestion Control: Modelling and Validation. In *Proceedings of the 4th Intl. ICST Conf. on Simulation Tools and Techniques*, SIMUTools '11, 2011.

[33] Wei Lin Guay, Sven-Arne Reinemo, Olav Lysne, and Tor Skeie. dFtree: A fat-tree routing algorithm using dynamic allocation of virtual lanes to alleviate congestion in InfiniBand networks. In *Proc. of the First Intl. Workshop on Network-aware Data Management*, NDM '11.

[34] E.G. Gran, S.-A. Reinemo, O. Lysne, T. Skeie, E. Zahavi, and G. Shainer. Exploring the scope of the InfiniBand congestion control mechanism. In *Parallel Dist. Proc. Symp. (IPDPS), IEEE 26th Intl.*, 2012.

[35] Jesus Escudero-Sahuquillo, Pedro J. Garcia, Francisco J. Quiles, Sven-Arne Reinemo, Tor Skeie, Olav Lysne, and Jose Duato. A new proposal to deal with congestion in InfiniBand-based fat-trees. *Journal of Parallel and Dist. Comp.,2014.*

[36] Nathan R Tallent, Abhinav Vishnu, Hubertus Van Dam, Jeff Daily, Darren J Kerbyson, and Adolfy Hoisie. Diagnosing the causes and severity of one-sided message contention. In *ACM SIGPLAN Notices*, number 8, pages 130–139. ACM, 2015.

[37] HyunBean Yi, SungJu Park, Misook Kim, and Kiman Jeon. An efficient buffer allocation technique for virtual lanes in InfiniBand networks. In *Proceedings of the 2Nd International Conference on Human.Society@Internet*, HSI'03, pages 272–281, Berlin, Heidelberg, 2003. Springer-Verlag.

[38] H Subramoni, A Venkatesh, K Hamidouche, K Tomko, and DK Panda. Impact of InfiniBand DC transport protocol on energy consumption of all-to-all collective algorithms. In *IEEE 23rd Annual Symp. on High-Performance Interconnects*, pages 60–67. IEEE, 2015.

[39] Fabrizio Petrini, Darren J Kerbyson, and Scott Pakin. The case of the missing supercomputer performance: Achieving optimal performance on the 8,192 processors of ASCI Q. In *Supercomputing, ACM/IEEE Conf.,2003*.

[40] Hari Subramoni, Devendar Bureddy, K Kandalla, K Schulz, B Barth, J Perkins, M Arnold, and Dhabaleswar K Panda. Design of network topology aware scheduling services for large InfiniBand clusters. In *IEEE Intl. Conf. on Cluster Computing (CLUSTER)*, pages 1–8. IEEE, 2013.

[41] M.E. Gomez, J. Flich, A. Robles, P. Lopez, and J. Duato. VOQSW: a methodology to reduce hol blocking in InfiniBand networks. In *Parallel and Dist. Proc. Symp., 2003. Proceedings. Intl.*, page 10, April 2003.

[42] Wei Lin Guay, Sven-Arne Reinemo, Olav Lysne, and Tor Skeie. dftree: A fat-tree routing algorithm using dynamic allocation of virtual lanes to alleviate congestion in infiniband networks. In *Proceedings of the First Intl. Workshop on Network-aware Data Management*, NDM '11, 2011.

[43] Wei Lin Guay, B. Bogdanski, S.-A. Reinemo, O. Lysne, and T. Skeie. vFtree - a fat-tree routing algorithm using virtual lanes to alleviate congestion. In *Parallel Dist. Proc. Symp. (IPDPS), IEEE Intl.*, 2011.

[44] Pedro Yebenes Segura, Jesus Escudero-Sahuquillo, Crispin Gomez Requena, Pedro Javier Garcia, Francisco J Quiles, and Jose Duato. BBQ: a straightforward queuing scheme to reduce HoL-blocking in high-performance hybrid networks. In *European Conf. on Parallel Processing*, pages 699–712. Springer, 2013.

[45] Feroz Zahid, Ernst Gunnar Gran, Bartosz Bogdanski, Bjørn Dag Johnsen, and Tor Skeie. Partition-aware routing to improve network isolation in InfiniBand based multi-tenant clusters. In *Cluster, Cloud and Grid Computing (CCGrid), 2015 15th IEEE/ACM Intl. Symp. on*, pages 189–198. IEEE, 2015.

[46] George Michelogiannakis, Nan Jiang, Daniel Becker, and William J. Dally. Channel reservation protocol for over-subscribed channels and destinations. In *Proceedings of the Intl. Conf. on High Performance Computing, Networking, Storage and Analysis*, SC '13, 2013.

[47] Q. Liu and R. D. Russell. RGBCC: A New Congestion Control Mechanism for InfiniBand. In *Parallel, Distributed, and Network-Based Processing (PDP), 2016 24th Euromicro Intl. Conf. on* , 2016.

[48] Qian Liu, Robert D Russell, and Ernst Gunnar Gran. Improvements to the InfiniBand Congestion Control Mechanism. In *High-Performance Interconnects (HOTI), 2016 IEEE 24th Annual Symp. on*, pages 27–36. IEEE, 2016.

[49] J. Babiarz, K. Chan, and F. Baker. Configuration Guidelines for DiffServ Service Classes. RFC 4594 (Informational), August 2006.

[50] Chi-Yao Hong, Srikanth Kandula, Ratul Mahajan, Ming Zhang, Vijay Gill, Mohan Nanduri, and Roger Wattenhofer. Achieving high utilization with software-driven WAN. In *ACM SIGCOMM*, volume 43, pages 15–26. ACM, 2013.

[51] L. Tong, X. Zheng, Y. Xia, and M. Li. Delay Tolerant Bulk Transfers on Inter-Datacenter Networks. In *2016 IEEE Globecom Workshops*, 2016.

[52] Y. Wu, Z. Zhang, C. Wu, C. Guo, Z. Li, and F. C. M. Lau. Orchestrating Bulk Data Transfers across Geo-Distributed Datacenters. *IEEE Transactions on Cloud Computing*, 5(1):112–125, Jan 2017.

[53] C. Shi, M. H. Ammar, and E. W. Zegura. iDTT: Delay Tolerant Data Transfer for P2P File Sharing Systems. In *2011 IEEE Global Telecommunications Conference - GLOBECOM 2011*, Dec 2011.

[54] N. Laoutaris, G. Smaragdakis, R. Stanojevic, P. Rodriguez, and R. Sundaram. Delay-Tolerant Bulk Data Transfers on the Internet. *IEEE/ACM Transactions on Networking*, 21(6):1852–1865, Dec 2013.

[55] I. Bueno, J. I. Aznar, E. Escalona, J. Ferrer, and J. A. Garcia-Espin. An OpenNaaS Based SDN Framework for Dynamic QoS Control. In *2013 IEEE SDN4FNS*, pages 1–7, Nov 2013.

[56] A. V. Akella and K. Xiong. Quality of Service (QoS)-Guaranteed Network Resource Allocation via Software Defined Networking (SDN). In *2014 IEEE 12th International Conference on Dependable, Autonomic and Secure Computing*, pages 7–13, Aug 2014.

[57] Cosmin Caba and José Soler. SDN-Based QoS Aware Network Service Provisioning. In *International Conference on Mobile, Secure and Programmable Networking*, pages 119–133. Springer, 2015.

[58] N. F. Huang, I. J. Liao, H. W. Liu, S. J. Wu, and C. S. Chou. A dynamic QoS management system with flow classification platform for software-defined networks. In *2015 8th International Conference on Ubi-Media Computing (UMEDIA)*, pages 72–77, Aug 2015.

[59] Y. Wang, C. Q. Wu, and A. Hou. On Periodic Scheduling of Bandwidth Reservations with Deadline Constraint for Big Data Transfer. In *2016 IEEE 41st Conference on Local Computer Networks (LCN)*, pages 224–227, Nov 2016.

[60] S. Tomovic and I. Radusinovic. Fast and efficient bandwidth-delay constrained routing algorithm for SDN networks. In *2016 IEEE NetSoft Conference and Workshops (NetSoft)*, pages 303–311, June 2016.

[61] Stanislav Shalunov and Benjamin Teitelbaum. Qbone scavenger service (QBSS) definition. Technical report, Internet2, 2001.

[62] Konstantinos Psounis, Arpita Ghosh, and Balaji Prabhakar. SIFT: A low-complexity scheduler for reducing flow delays in the Internet. Technical report, 2004.

[63] Doreid Ammar, Thomas Begin, and Isabelle Guerin-Lassous. A new tool for generating realistic Internet traffic in ns-3. In *Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques*, pages 81–83, 2011.

[64] Ronald G. Addie, Timothy D. Neame, and Moshe Zukerman. Performance evaluation of a queue fed by a Poisson Pareto burst process . *Computer Networks*, 40(3):377–397, 2002.

[65] ESnet. OSCARS: On-demand Secure Circuits and Advance Reservation System `https://www.es.net/engineering-services/oscars/`.

[66] Neal Cardwell, Yuchung Cheng, C Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. BBR: Congestion-based congestion control. *Queue*, 14(5):50, 2016.

[67] DOE. DOE Network 2025: Network Research Problems and Challenges for DOE Scientists Workshop. 2016.

[68] Jim Gettys and Kathleen Nichols. Bufferbloat: Dark Buffers in the Internet. *Queue*, 9(11):40:40–40:54, November 2011.

[69] Srikanth Kandula, Ishai Menache, Roy Schwartz, and Spandana Raj Babbula. Calendaring for Wide Area Networks. In *Proceedings of the 2014 ACM Conference on SIGCOMM*, SIGCOMM '14, pages 515–526, New York, NY, USA, 2014. ACM.

[70] Hong Zhang, Kai Chen, Wei Bai, Dongsu Han, Chen Tian, Hao Wang, Haibing Guan, and Ming Zhang. Guaranteeing Deadlines for Inter-Data Center Transfers. *IEEE/ACM Trans. Netw.*, 25(1):579–595, February 2017.

[71] David Fridovich-Keil, Nathan Hanford, Margaret Chapman, Claire Tomlin, Matthew Farrens, and Dipak Ghosal. A Model Predictive Control Approach to Flow Pacing for TCP. In *55th Annual Allerton Conference on Communication, Control, and Computing*, 2017.

[72] Amit Aggarwal, Stefan Savage, and Thomas Anderson. Understanding the performance of TCP pacing. In *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1157–1165. IEEE, 2000.

[73] D Wei, Pei Cao, S Low, and Caltech EAS. TCP pacing revisited. In *Proceedings of IEEE INFOCOM*, 2006.

[74] Hassan Habibi Gharakheili, Arun Vishwanath, and Vijay Sivaraman. Comparing edge and host traffic pacing in small buffer networks. *Computer Networks*, 77:103 – 116, 2015.

[75] VMware. Key Considerations in Choosing a Zero Client Environment for View Virtual Desktops in VMware Horizon. https://www.vmware.com/content/dam/digitalmarketing/vmware/en/pdf/techpaper/vmware-top-five-considerations-for-choosing-a-zero-client-environment.pdf, 2014.

[76] Teradici PCoIP. Moulton College deploys Teradici PCoIP Hardware Accelerator with Zero Clients to deliver flawless video performance while sharply decreasing the management burden for its endpoints. http://www.teradici.com/docs/default-source/resources/case-studies/moulton_college_case_study.pdf, 2016.

[77] Teradici PCoIP. Lewiston Library offers patrons quiet, high-performance zero clients and virtual desktops. https://www.teradici.com/docs/default-source/resources/case-studies/cs_cityof-lewiston-case-study.pdf, 2014.

[78] Teradici PCoIP. North Kansas City Hospital delivers secure, point-of-care computer access anytime, anywhere. http://www.teradici.com/docs/default-source/resources/case-studies/nkch_case_study.pdf, 2014.

[79] Teradici PCoIP. PCoIP Zero Clients, Hardware Accelerators, & GPUs Match Graphic-Intensive Workloads for 2D/3D CAD. https://www.teradici.com/docs/default-source/resources/case-studies/cs_construction_industry_case_study.pdf, 2015.

[80] Douglas Brinkley. Thin-clients in the Classroom; Software Compatibility and a Survey of Systems . In Thomas Reeves and Shirley Yamashita, editors, *Proceedings of E-Learn: World Conference on E-Learning in Corporate, Government, Healthcare, and Higher Education 2006*, pages 383–390, Honolulu, Hawaii, USA, October 2006. Association for the Advancement of Computing in Education (AACE).

[81] Niraj Tolia, David G Andersen, and Mahadev Satyanarayanan. Quantifying interactive user experience on thin clients. *Computer*, (3):46–52, 2006.

[82] Technology Advice for Small Businesses. How thin and zero clients save money. https://www.techadvisory.org/2017/07/how-thin-and-zero-clients-save-money/, 2017.

[83] Michael Remer. Reduce Your I.T. Costs With Thin Clients. https://www.businessmagazinegainesville.com/reduce-your-i-t-costs-with-thin-clients/, 2017.

[84] Business Value. The Pros and Cons of Thin Clients. https://www.houkconsulting.com/2016/06/pros-cons-thin-clients/, 2009.

[85] Andrew Wood. Seven deadly sins when deploying thin clients. https://www.astroarch.com/tvp_strategy/seven-deadly-sins-when-deploying-thin-clients-1544/, 2009.

[86] Junghwan Rhee, Andrzej Kochut, and Kirk Beaty. DeskBench: Flexible virtual desktop benchmarking toolkit. In *Integrated Network Management, 2009. IM'09. IFIP/IEEE International Symposium on*, pages 622–629. IEEE, 2009.

[87] Tao Song, Jiajun Wang, Jiewei Wu, Ruhui Ma, Alei Liang, Tao Gu, and Zhengwei Qi. FastDesk: A remote desktop virtualization system for multi-tenant. *Future Generation Computer Systems*, 81:478–491, 2018.

[88] Yan Sui, Chun Yang, Ning Jia, and Xu Cheng. vSIP: virtual scheduler for interactive performance. In *Proceedings of the ACM International Conference on Computing Frontiers*, pages 222–231. ACM, 2016.

[89] Biao Song, Mohammad Mehedi Hassan, Yuan Tian, M Shamim Hossain, and Atif Alamri. Remote display solution for video surveillance in multimedia cloud. *Multimedia Tools and Applications*, 75(21):13375–13396, 2016.

[90] Yuezhi Zhou, Wenjuan Tang, Di Zhang, and Yaoxue Zhang. Software-defined streaming-based code scheduling for transparent computing. In *Advanced Cloud and Big Data (CBD), 2016 International Conference on*, pages 296–303. IEEE, 2016.

[91] Md Abu Layek, TaeChoong Chung, and Eui-Nam Huh. Adaptive desktop delivery scheme for provisioning quality of experience in cloud desktop as a service. *The Computer Journal*, 59(2):260–274, 2016.

[92] PCoIP. https://www.teradici.com/what-is-pcoip.

[93] Autoit. https://www.autoitscript.com/site/.

[94] Jason Nieh, S Jae Yang, and Naomi Novik. Measuring thin-client performance using slow-motion benchmarking. *ACM Transactions on Computer Systems (TOCS)*, 21(1):87–115, 2003.

[95] Alex Berryman, Prasad Calyam, Matthew Honigford, and Albert M Lai. VDBench: A benchmarking toolkit for thin-client based virtual desktop environments. In *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, pages 480–487. IEEE, 2010.

[96] Junyong You, Ulrich Reiter, Miska M Hannuksela, Moncef Gabbouj, and Andrew Perkis. Perceptual-based quality assessment for audio–visual services: A survey. *Signal Processing: Image Communication*, 25(7):482–501, 2010.

[97] Dennis Klatt. Prediction of perceived phonetic distance from critical-band spectra: A first step. In *Acoustics, Speech, and Signal Processing, IEEE International Conference on ICASSP'82.*, volume 7, pages 1278–1281. IEEE, 1982.

[98] R Viswanathan, John Makhoul, and William Russell. Towards perceptually consistent measures of spectral distance. In *Acoustics, Speech, and Signal Processing, IEEE International Conference on ICASSP'76.*, volume 1, pages 485–488. IEEE, 1976.

[99] Andrew Hines, Jan Skoglund, Anil Kokaram, and Naomi Harte. ViSQOL: The virtual speech quality objective listener. In *Acoustic Signal Enhancement; Proceedings of IWAENC 2012; International Workshop on*, pages 1–4. VDE, 2012.

[100] Antony W Rix, John G Beerends, Michael P Hollier, and Andries P Hekstra. Perceptual evaluation of speech quality (PESQ)-a new method for speech quality assessment of telephone networks and codecs. In *Acoustics, Speech, and Signal Processing,*

2001. Proceedings.(ICASSP'01). 2001 IEEE International Conference on, volume 2, pages 749–752. IEEE, 2001.

[101] ITU-T Recommendations P. 863: Assessment, Perceptual Objective Listening Quality. 2011.

[102] Clumsy 0.2. https://jagt.github.io/clumsy/index.html.

[103] ITU-T Recommendations P. 800: Methods for subjective determination of transmission quality. 1996.

[104] ITU-T Recommendations P. 800.1: Mean opinion score (MOS) terminology. 2016.

[105] ITU-R Recommendation BT.500-13: Methodology for the subjective assessment of the quality of television pictures. 2012.

[106] Weiren Yu, Jianxin Li, Chunming Hu, and Liang Zhong. Muse: a multimedia streaming enabled remote interactivity system for mobile devices. In *Proceedings of the 10th International Conference on Mobile and Ubiquitous Multimedia*, pages 216–225. ACM, 2011.

[107] Lin Cai, Zhen Jia, Yong Qi, and Lei Wang. CloudRank-V: A Desktop Cloud Benchmark with Complex Workloads. In *High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing (HPCC_EUC), 2013 IEEE 10th International Conference on*, pages 415–421. IEEE, 2013.

[108] Nickolai Zeldovich and Ramesh Chandra. Interactive Performance Measurement with VNCPlay. In *USENIX Annual Technical Conference, FREENIX Track*, pages 189–198, 2005.

[109] Atul Pandey, Lan Vu, Vivek Puthiyaveettil, Hari Sivaraman, Uday Kurkure, and Aravind Bappanadu. An automation framework for benchmarking and optimizing performance of remote desktops in the cloud. In *High Performance Computing & Simulation (HPCS), 2017 International Conference on*, pages 745–752. IEEE, 2017.

[110] Sanna Laine and Ismo Hakala. H. 264 QoS and application performance with different streaming protocols. In *Proceedings of the 8th International Conference on Mobile Multimedia Communications*, pages 32–38. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2015.

[111] An Chan, Kai Zeng, Prasant Mohapatra, Sung-Ju Lee, and Sujata Banerjee. Metrics for evaluating video streaming quality in lossy IEEE 802.11 wireless networks. In *INFOCOM, 2010 Proceedings IEEE*, pages 1–9. IEEE, 2010.

[112] Ricky KP Mok, Edmond WW Chan, and Rocky KC Chang. Measuring the quality of experience of HTTP video streaming. In *Integrated Network Management (IM), 2011 IFIP/IEEE International Symposium on*, pages 485–492. IEEE, 2011.

[113] Tasnim Abar, Asma Ben Letaifa, and Sadok El Asmi. Objective and subjective measurement QOE in SDN networks. In *Wireless Communications and Mobile Computing Conference (IWCMC), 2017 13th International*, pages 1401–1406. IEEE, 2017.

[114] BP Bondzulic, BZ Pavlovic, VS Petrovic, and MS Andric. Performance of peak signal-to-noise ratio quality assessment in video streaming with packet losses. *Electronics Letters*, 52(6):454–456, 2016.

[115] Federica Battisti, Marco Carli, and Pradip Paudyal. QoS to QoE mapping model for wired/wireless video communication. In *Euro Med Telco Conference (EMTC), 2014*, pages 1–6. IEEE, 2014.

[116] Zhengfang Duanmu, Kai Zeng, Kede Ma, Abdul Rehman, and Zhou Wang. A quality-of-experience index for streaming video. *IEEE Journal of Selected Topics in Signal Processing*, 11(1):154–166, 2017.

[117] Zhenjie Deng, Yanwei Liu, Jinxia Liu, Xu Zhou, and Song Ci. QoE-Oriented Rate Allocation for Multipath High-Definition Video Streaming Over Heterogeneous Wireless Access Networks. *IEEE Systems Journal*, 2015.

[118] Jeevan Pokhrel, Bachar Wehbi, Anderson Morais, Ana Cavalli, and Eric Allilaire. Estimation of QoE of video traffic using a fuzzy expert system. In *Consumer Communications and Networking Conference (CCNC), 2013 IEEE*, pages 224–229. IEEE, 2013.

[119] Michael Jarschel, Daniel Schlosser, Sven Scheuring, and Tobias Hoßfeld. An evaluation of QoE in cloud gaming based on subjective tests. In *Innovative mobile and internet services in ubiquitous computing (imis), 2011 fifth international conference on*, pages 330–335. IEEE, 2011.

[120] Prasad Calyam, Alex Berryman, David Welling, Mohan Saravanan, Rajiv Ramnath, and Jay Ramanathan. VDPilot: feasibility study of hosting virtual desktops for classroom labs within a federated university system. *IJCC*, 3(2):158–176, 2014.

[121] Wen-Hui Chiang, Wei-Cheng Xiao, and Cheng-Fu Chou. A performance study of VoIP applications: MSN vs. skype. *Proceedings of MULTICOMM*, pages 13–18, 2006.

[122] Gao Lisha and Luo Junzhou. Performance analysis of a P2P-based VoIP software. In *Telecommunications, 2006. AICT-ICIW'06. International Conference on Internet and Web Applications and Services/Advanced International Conference on*, pages 11–11. IEEE, 2006.

[123] Prasad Calyam, Eylem Ekici, Chang-Gun Lee, Mark Haffner, and Nathan Howes. A GAP-Model based framework for online VVoIP QoE measurement. *Journal of Communications and Networks*, 9(4):446–456, 2007.

[124] Nate Kushman, Srikanth Kandula, and Dina Katabi. Can you hear me now: it must be BGP. *ACM SIGCOMM Computer Communication Review*, 37(2):75–84, 2007.

[125] Mohammad Goudarzi, Lingfen Sun, and Emmanuel Ifeachor. Modelling speech quality for NB and WB SILK codec for VoIP applications. In *Next Generation Mobile Applications, Services and Technologies (NGMAST), 2011 5th International Conference on*, pages 42–47. IEEE, 2011.

[126] Maria-Dolores Cano and Fernando Cerdan. Subjective QoE analysis of VoIP applications in a wireless campus environment. *Telecommunication Systems*, 49(1):5–15, 2012.

[127] Nalakkhana Khitmoh, Pongpisit Wuttidittachotti, and Therdpong Daengsi. A subjective VoIP quality estimation model for G. 729 based on native Thai users. In *Advanced Communication Technology (ICACT), 2014 16th International Conference on*, pages 48–53. IEEE, 2014.

[128] Therdpong Daengsi, Pongpisit Wuttidittachotti, Chai Wutiwiwatchai, Apiruck Preechayasomboon, and Saowanit Sukparungsee. VoIP Quality of Experience: A proposed subjective MOS estimation model based-on Thai users. In *Ubiquitous and Future Networks (ICUFN), 2013 Fifth International Conference on*, pages 407–412. IEEE, 2013.