Traffic Engineering in Packet/Circuit Hybrid Networks

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

Zhenzhen Yan

November 2013

Abstract

Scientific computing applications, used in fields such as high-energy physics, climate science, genomics, etc., generate large (tera- to peta-byte sized) data sets. To move these heavy-hitter datasets fast, supercomputing sites invest in high-end clusters that can sustain high-speed transfers. Such large-sized, high-speed transfers are called α flows. These flows can have adverse effects on packet delays of real-time flows as α flows being bursty in nature can cause router buffer buildups.

To enable the support of both α flows and real-time flows on the same network infrastructure while meeting quality-of-service (QoS) requirements of both types of flows, we propose a Hybrid Network Traffic Engineering System (HNTES). HNTES performs two tasks, 1) automatic identification of α flows at a core provider network's ingress routers, with no requirement of modifying end-user applications, 2) redirects these flows to traffic-engineered QoS-controlled virtual circuits.

This dissertation describes two versions of the HNTES design. The first design explored the possibility of implementing an online mechanism that identifies alpha flows from live traffic. But the design proved to be cost prohibitive. Our second design uses an offline approach by analyzing NetFlow reports of completed flows to determine the IP addresses of source-destination pairs that move large datasets at high rates. These extracted IP addresses are used to set firewall filters in ingress routers to capture future alpha flows for redirection. This solution is based on a hypothesis that alpha flows are repeatedly created by the same source-destination hosts. NetFlow data for a 7-month period was obtained from an ESnet router to test the hypothesis, which proved to be true. It showed that had HNTES been deployed at the start of this period, over 91% of data that appeared in bursts from

α flows would have been redirected. The final contribution of this thesis resulted from an experimental study of different scheduling and policing mechanisms implemented in routers. The objective was to find a suitable combination of mechanisms that achieved dual goals: reducing delay and jitter of real-time delay-sensitive flows, while at the same time allowing alpha flows to achieve high throughput. The best combination was found to be a no-policing, two-queue solution with priority and weighted fair queueing forms of packet scheduling. This result influenced the configuration of routers of a US backbone network provider.

Acknowledgements

I would like to express my deepest gratitude to my advisor, Prof. Malathi Veeraraghavan, for her consistent guidance, support and patience. Prof. Veeraraghavan has provided an excellent example for me in so many ways through her breadth and depth of knowledge, intellectual curiosity, critical thinking, and most importantly, through her research attitude. It has been an invaluable experience to learn so much from her.

Many thanks to Chris Tracy, Chin Guok, Brian Tierney, and all the ESnet engineers who helped me with data acquisition and testbed experiments.

I thank Mark Karol for his insights and advice throughout my doctoral study.

I thank Prof. Joanne Bechta Dugan, Mark Karol, Prof. Maite Brandt-Pearce, and Prof. Worthy Martin for serving on my defense committee and providing constructive comments.

I am grateful to the graduate students in our research group, Mark McGinley, Jie Li, Zhengyang Liu, Tian Jin, and Zhe Song, with whom I have had the honor of working.

I would like to thank my husband Jie Li, my parents Qingxu Yan and Ming Gao, my parents-in-law Wenmiao Li and Yuqing Zhu, and all of our family members for their love and support over the years.

This work was supported by the U.S. Department of Energy (DOE) grant DE-SC0002350, DE-SC0007341 and NSF grants OCI-1038058, OCI-1127340, and CNS-1116081. Support from our ESnet collaborator was critical to this work. The ESnet part was sponsored by the Director, Office of Science, Office of Basic Energy Sciences, of the U.S. DOE under Contract No. DE-AC02- 05CH11231. This research used resources of the ESnet ANI Testbed, which is supported by the Office of Science of the U.S. DOE under contract DE-AC02-05CH11231, funded through the American Recovery and Reinvestment Act of 2009. I would like to thank

DOE and NSF for funding this research.

Approval Sheet

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

Zhenzhen Yan

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

Malathi Veeraraghavan , Adviser

Joanne Dugan , Committee Chair

Mark Karol

Maite Brandt-Pearce

Worthy Martin

Accepted for the School of Engineering and Applied Science:

James Aylor, Dean, School of Engineering and Applied Science

November 2013

Contents

List of Tables

List of Figures

List of Abbreviations

ACK	acknowledgement
AFD	Approximate Fair Dropping
AFTES	Alpha-Flow Traffic Engineering System
AIMD	Additive-Increase/Multiplicative-Decrease
ANI	Advanced Networking Initiative
AQM	active queue management
BNL	Brookhaven National Laboratory
cwnd	congestion window
DCS	Dynamic Circuit Service
DOE	Department of Energy
DTP	Data Transfer Process
DYNES	Dynamic Network System
ESnet	Energy Sciences Network
FCFS	first-come first-serve
FMM	Flow-Monitoring Module
FT	File Transfers
GRE	Generic Routing Encapsulation
HNTES	Hybrid Network Traffic Engineering System
IDC	Inter-Domain Controller
IDCP	IDC Protocol
IM	Initialization Module

IntServ	Integrated Services
LDM	Unidata Local Data Manager
LIMAN	Long Island Metropolitan Area Network
LSP	Label-Switched Path
MFDB	Monitored-Flow Data Base
MPLS	MultiProtocol Label Switching
MSS	maximum segment size
NCAR	National Center for Atmospheric Research
NIC	network interface card
NICS	National Institute for Computational Sciences
OFAT	Offline Flow Analysis Tool
OOP	Out-of-Profile
OSCARS	On-Demand Secure Circuits and Advance Reservation System
PBR	Policy-based route
PI	Protocol Interpreter
PQ	Priority Queueing
QoS	Quality of Service
RCIM	Router-Control Interface Module
REN	Research-and-Education Network
RIB	Routing Information Base
RTT	round-trip time
SNMP	Simple Network Management Protocol
SOAP	Simple Object Access Protocol
SS	Scavenger Service
TCP	Transmission Control Protocol
UIM	User-Interface Module

VC	Virtual Circuit
WFQ	Weighted Fair Queueing
WNP	Well-Known Ports
WRED	Weighted Random Early Detection

Chapter 1

Introduction

1.1 Background and problem statement

Background The Internet carries traffic generated by a variety of applications. These include small data transfers, large file transfers, and real-time interactive audio/video flows. While large file transfers are throughput sensitive, interactive flows are latency sensitive. As communication link rates increase, the difference between the minimum and maximum rates of flows sharing a link increases. With compression technologies such as MPEG, high-quality audio/video can be supported at low rates. At the other extreme, scientific researchers invest in high-speed network interface cards for servers that are dedicated to file-transfer applications. Transfers at 90 Gbps have been demonstrated across a 100 Gb/s Ethernet network testbed supported by ESnet [2].

To avoid packet losses due to buffer overflows, routers increasingly use large-sized buffers. This phenomenon is referred to as bufferbloat [3]. But large buffers can cause increased latency for a delay-sensitive audio/video flow if a high-rate file transfer happens to occur simultaneously. Therefore, the question of how to support multiple types of services on the same network infrastructure, while meeting the disparate requirements of the services, has been a research problem of interest.

In the nineties, Integrated Services (IntServ) [4] was proposed to support a virtual-circuit service complementary to the connectionless service of the IP network. Virtual circuits were considered for delay-sensitive multimedia flows. However, this solution was not scalable to

large numbers of flows because of the challenges in implementing Quality of Service (QoS) mechanisms such as policing and scheduling on a per-flow basis.

In recent years, the Research-and-Education Network (REN) community that supports scientific high-performance computing has identified a potential solution to this problem of supporting multiple types of services on the same network. The concept is to identify "heavy-hitter" flows, i.e., flows that require more resources, and isolate them from general-purpose flows. Since the number of simultaneous heavy-hitter flows is typically small, QoS mechanisms are more easily applied to these flows than to the thousands of simultaneous multimedia flows on a shared link.

In trying to define thresholds for heavy-hitter flows, four dimensions: size (bytes), duration, rate, and burstiness are considered. Lan and Heidemann [5] gave names such as elephants and mice (size), tortoise and dragonfly (duration), cheetah and snail (rate), and porcupine and stingray (bustiness).

Ideally, an end-user application that is about to generate a heavy-hitter flow should signal control servers within the network with its resource requirements so that these control servers can configure paths and rate/buffer resources for the flow before it starts. But this mode of operation requires circuit or virtual-circuit (VC) services. Circuit/VC services have a call admission phase prior to data transfer. However, the Internet is connectionless, in that IP routers simply switch individual packets based on the destination IP address carried in packet headers. There is no admission control in connectionless networks. No state is maintained in IP routers for individual flows, which was key to scalability. However, as mentioned above, if there was built-in support for circuit/VC service within networks, it could be sparingly used just for heavy-hitter flows.

Today's IP routers support VC service with a technology called MultiProtocol Label Switching (MPLS) [6]. The REN community has exploited this complementary technology to offer Dynamic Circuit Services (DCS) on their deployed base of IP routers. Thus, an end-user application could signal control servers within networks to request a dynamically setup virtual circuit before sending data.

However there are two problems with this approach. First, DCS has not been deployed in all campus and regional REN networks. Backbone (core) RENs such as Department of Energy

(DOE)'s Energy Sciences Network (ESnet) [7] and Internet2 [8] have deployed DCS. Through an NSF-supported project called Dynamic Network System (DYNES) [9], 40 campuses and universities have recently added DCS. However, DCS is still not ubiquitous. The second problem is that end-user application software is implemented under an assumption that the network service is connectionless (specifically, IP-routed service). These software programs need to be modified to support VC service (i.e., to make requests for dynamic circuits prior to data transfer) and then deployed on campuses. This solution was attempted in projects such as Lambdastation [10], Terapaths [11], and CHEETAH [12], but practical difficulties of application upgrades and adoption by users hindered its deployment. This led us to pursue an intra-domain traffic-engineering solution because deployment of such a system would be entirely within a provider's control.

Problem statement Develop mechanisms for identifying heavy-hitter flows from the packet traffic entering a provider's network, and design solutions for traffic engineering and controlling the resources assigned to these flows. In this context, traffic engineering is the term used for path selection. This form of traffic engineering is referred to as tactical traffic engineering [13] in contrast to strategic traffic engineering, in which overall traffic matrix is considered while deciding paths between all source-destination pairs.

The research challenge of such an intra-domain traffic engineering system is to decide whether an online solution (upon flow arrival) is required or an offline solution (analyze completed-flow information and configure routers a priori for future flows) is sufficient. The online solution poses system design and implementation challenges. The total execution time of flow classification, circuit setup, and route configuration steps need to be significantly shorter than the flow duration. This requires the duration of a flow to be at least several minutes since the current circuit setup time itself is on the order of minutes. An offline solution is possible if identifiers, e.g., source and destination IP addresses, of heavy-hitters flows remain unchanged, and if the rate of arrival of new unexpected heavy-hitter flows is small.

This dissertation presents a new solution for intra-domain traffic engineering by exploring online and offline solutions, and considering different dimensions of heavy-hitter flows.

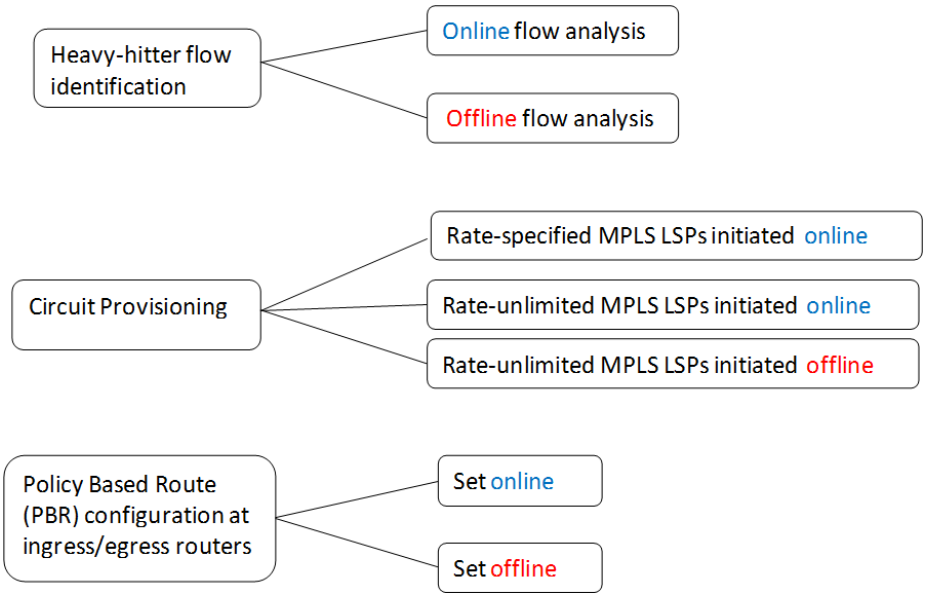


Figure 1.1: Three tasks executed by HNTES

1.2 Solution approach

We designed a network management system called a Hybrid Network Traffic Engineering System (HNTES) for deployment within a provider's network to address the above stated problem. The term "hybrid network" was used because the solution leverages both the IP-routed connectionless network and MPLS virtual-circuit connection-oriented network.

Broadly speaking, HNTES needs to perform three tasks: *heavy-hitter flow identification*, *circuit provisioning*, and *policy-based route (PBR) configuration*, as shown in Fig. 1.1. Two approaches were identified for *heavy-hitter flow identification*: online flow analysis and offline flow analysis. In online flow analysis, packet headers of newly arriving flows observed at an IP router (i.e., ingress router of a provider's network) are analyzed for live identification of heavy-hitter flows. In offline flow analysis, NetFlow reports created by IP routers from sampled packet-header data are analyzed to determine identifiers of completed heavy-hitter flows for use in future-flow traffic engineering.

For the second task, *circuit provisioning*, HNTES can either request the setup of rate-specified MPLS Label-Switched Paths (LSPs) online, or rate-unlimited MPLS LSPs online or offline. Specifying a rate for an MPLS LSP could reduce flow throughput. The work presented in Chapter 4 explores these different options for circuit provisioning.

The purpose of the third task, *PBR configuration*, is to set a policy-based route to filter out packets from a particular flow for handling with a separate set of procedures when compared to other packets. Default packet handling procedures are simple in IP routers. The destination IP address in an incoming packet is used to consult a forwarding table to determine the next-hop router toward which the packet is sent. For heavy-hitter flows, QoS mechanisms such as policing and scheduling would be applied and packets of these flows directed to possibly a different port than the one dictated by the default IP forwarding table. This task, PBR configuration, can be executed online (on flow arrival) or offline (a priori). If performed online, then the PBR can use source and destination IP addresses as well as source and destination TCP port numbers, while if it is performed offline, then only source and destination IP addresses can be used because many applications use ephemeral (not standardized static, referred to as "well-known") port numbers.

If all three tasks can be performed online (on flow arrival), this would be an ideal HNTES because only heavy-hitter flows would be subject to special treatment (sent to specific paths and assigned a specific set of resources). In Chapter 2, we set out to design and prototype such a system. At the end of Chapter 2, we discuss reasons why such an ideal HNTES is difficult to realize. In Chapter 3, we describe our work on a more practical offline HNTES solution, in which all three tasks are executed offline. There are costs to this solution, which are quantified through NetFlow data analysis.

1.3 Hypothesis

The hypothesis of this work is as follows: It is feasible to identify heavy-hitter flows within a provider's network and redirect them to traffic-engineered paths with separate resource allocations while meeting quality-of-service (QoS) requirements of both types of flows.

There are two research challenges in this hypothesis formulation. The first challenge lies in the phrase "within a provider's network." Traffic engineering of heavy-hitter flows would be easier to execute within provider networks if end-user applications offered an explicit indication prior to initiating a heavy-hitter flow. Since applications do not provide such indications, provider-deployed traffic engineering systems need to identify such flows

by examining packets entering a provider's network of IP routers at high speeds, which is challenging. The second challenge lies in the phrase "meeting quality-of-service (QoS) requirements" because certain types of flows do not have hard QoS requirements; instead these flows just want the highest possible throughput. In trying to allow these flows to enjoy highest possible throughput, other flows can be impacted adversely.

1.4 Dissertation organization

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

Chapter 2 presents a duration-based Hybrid Network Traffic Engineering System (HNTES) 1.0 solution for online identification and redirection of heavy-hitter flows. The system design and prototype implementation are described. The feasibility of deploying this solution is discussed at the end of the chapter.

Chapter 3 presents an algorithm for offline heavy-hitter flow identification. Unlike the online duration-based HNTES 1.0 design, this offline design is based on flow rates computed from NetFlow reports. The effectiveness of this algorithm is extensively evaluated through an analysis of 7 months of NetFlow data obtained from an ESnet provider edge router.

Chapter 4 describes our experimental evaluation of Quality-of-Service (QoS) mechanisms to achieve the dual goals of preventing heavy-hitter flows from adversely affecting delay-sensitive multimedia flows, while simultaneously allowing these flows to enjoy high throughput. The interaction between policing schemes on the ingress interfaces and scheduling schemes on the egress interfaces is studied through a set of experiments on a high-speed routed-network testbed.

Finally, conclusions and suggestions for future work are presented in Chapter 5.

1.5 Key contributions

A significant contribution is that our work led to a US-wide backbone network provider ESnet to change the quality-of-service configuration used in its IP routers.

The key contributions of this work are summarized as follows:

1. An understanding of practical considerations that limit feasible designs of traffic engineering systems that can manage multimedia flows with disparate characteristics while meeting the quality-of-service requirements of all flows.

2. A practical and effective Alpha-Flow Traffic Engineering System (AFTES) design, which was prototyped and evaluated. The term α flow is used to denote high-rate, large-sized flows.

3. Validation of a hypothesis that α flows are repeatedly sent by the same source-destination pairs. NetFlow reports for 7-months were collected from an operational large backbone (core) REN provider and analyzed. This work is published in a paper in the *IEEE 13th High Performance Switching and Routing (HPSR) 2012* [14].

4. The negatives of the offline AFTES solution were evaluated using 7-month NetFlow data collected from one ESnet router. Since the offline-set PBRs only include source and destination IP addresses, packets from general-purpose flows that share these addresses with α flows will also get redirected to the traffic-engineered, QoS-controlled paths established for α flows. Packets from these general-purpose flows could experience adverse effects such as additional delays caused by buffer build-ups. Our analysis showed that 90% of these general-purpose flow packets were from file-transfer applications, which are not as affected by α flows as packets from real-time applications. This work has been submitted in a paper to the *Journal of Network and Systems Management*. While this finding was true for the analyzed NetFlow data, it may not always hold true. It depends upon whether network administrators isolate α-flow generating clusters into their own subnets distinct from subnets that connect hosts/servers that execute general-purpose applications such as Web browsers/servers and Voice over IP. In most scientific data centers, this assumption of isolated subnets was true, and furthermore could be engineered if AFTES is deployed.

5. The interaction between policing schemes on ingress interfaces and scheduling schemes on egress interfaces of IP routers was studied in-depth. A scheduling-only mechanism with no policing was recommended to achieve the dual goals of preventing α flows

from adversely affecting delay-sensitive flows, while simultaneously maximizing α-flow throughput. This work was published in a paper in the *Sixth International Conference on Communication Theory, Reliability, and Quality of Service (CTRQ) 2013* [1]. An extended version of this work has been submitted in a paper to the *International Journal On Advances in Internet Technology*. An important result of this work is that ESnet modified its QoS configuration to create an additional virtual queue for α flows [15].

In addition, contributions were made to three other publications: 1) OWAMP (one-way ping) [16] delay data analysis was published in a paper in the *Proceedings of Optical Fiber Communication (OFC) 2011* [17]. 2) GridFTP log analysis was published in a paper in the *International Conference for High Performance Computing, Networking, Storage and Analysis 2012 (SC 2012)* [18]. 3) A comparative analysis of NetFlow data abstained from four ESnet routers was published in a paper in the *IEEE 14th High Performance Switching and Routing (HPSR) 2013* [19].

Chapter 2

A Hybrid Network Traffic Engineering System (HNTES) 1.0

2.1 Introduction

The concept of using dynamic circuit service for heavy-hitter flows strongly influenced our HNTES 1.0 design. The REN community had created standards, implemented circuit schedulers, and deployed a dynamic circuit service. Measurements showed that the circuit scheduler took on the order of 1 minute to create a circuit if one was requested for immediate use. This is because the scheduler batched requests and executed the configuration actions at the MPLS switches/IP routers only every minute. Knowledge of this circuit setup delay led us to focus on the "duration" dimension of flows when choosing heavy-hitters, since most flows are short-lived (last less than a few seconds).

Therefore the HNTES 1.0 design consisted of modules to perform three steps: (i) online detection of long-lived flows, (ii) online virtual circuit setup, and (iii) online PBR configuration.

Section 2.2 provides background and reviews related work. Section 2.3 describes HNTES 1.0 design, and the prototyping and evaluation efforts are described in Section 2.4. In Section 2.5, we discuss the feasibility of deploying HNTES 1.0 in provider networks, and conclude the chapter in Section 2.6.

2.2 Background and related work

With virtual-circuit technologies, such as MultiProtocol Label Switching (MPLS) [6], ESnet and other research and education network providers, such as Internet2 [20], GEANT2 [21], and JGN-X [22], offer a dynamic circuit service. An On-Demand Secure Circuits and Advance Reservation System (OSCARS) Inter-Domain Controller (IDC) [4] is used for circuit scheduling and provisioning. The basic interface to IDC requires an application to specify the circuit rate, duration, start time, and the endpoints in its advance-reservation request. These parameters are used for path computation in the call-admission/circuit scheduling phase, and for policing traffic in the data plane. In order to support inter-domain virtual circuits, Lake et al. from Internet2 and Robertson et al. from ESnet standardized the Inter-domain Controller (IDC) Protocol [23] in 2008.

Several traffic engineering solutions have been proposed based on this technology. For example, Lambdastation [10] is a deployed and demonstrated technique to redirect certain types of flows to virtual circuits. In this approach, applications signal a Lambdastation server, which is deployed in site networks, before initiating a long flow. The Lambdastation server issues a create-reservation request to the OSCARS IDC [24], which reserves a circuit, and at the time of the scheduled request, provisions the circuit. The virtual circuit endpoint is typically a customer or provider edge IP router within a site. Using policy-based route (PBR) configuration, which is a feature of a router that allows administrators to configure alternate routes other than the default IP route for packets belonging to specific flows, packets corresponding to the long flow are directed to the newly established virtual circuit. When the flow completes, the circuit is released.

The advantage of the HNTES approach over the application-triggered Lambdastation approach is that users do not need to modify their applications or run shell scripts that invoke circuit setup prior to application execution. The drawback however is that it is difficult to predict which flows are good candidates for virtual circuits when making the decision inside a transit provider network.

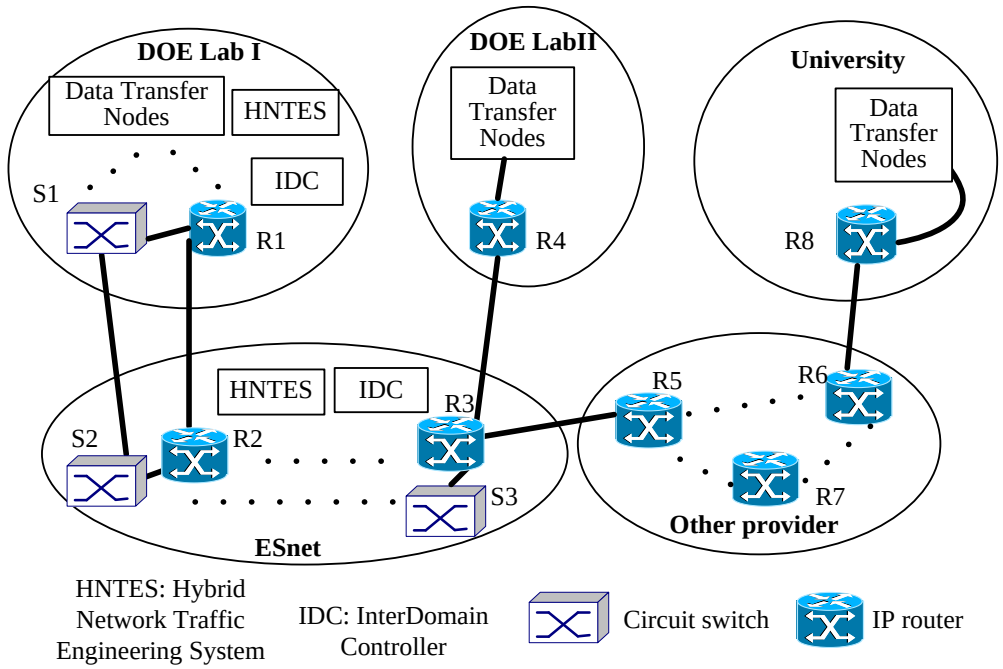


Figure 2.1: An example deployment scenario of HNTES

2.3 Hybrid Network Traffic Engineering System (HNTES) 1.0 design

2.3.1 Role of HNTES

We start with a big picture illustration of what role HNTES would play if deployed by providers who offer both IP-routed and virtual circuit (VC) services. In the example shown in Fig. 2.1, the DOE Lab I site network and ESnet are shown as offering both IP-routed and VC services, while DOE Lab II, other provider and the university network offer only IP-routed services. This reflects current-day reality. An IDC, which offers circuit scheduling, provisioning and release functionality, and a HNTES system are shown as being deployed in DOE Lab I and ESnet.

If a large dataset transfer is initiated from the data transfer nodes at DOE Lab II to the data transfer nodes at DOE Lab I, then the heavy-hitter flow will appear as IP-routed packets at the ESnet router R3. Without HNTES, packets from this flow would be forwarded with the IP-routed service via ESnet and DOE Lab I's network. With VC service, if offline flow analysis had identified this particular flow (e.g., if the source IP address and destination

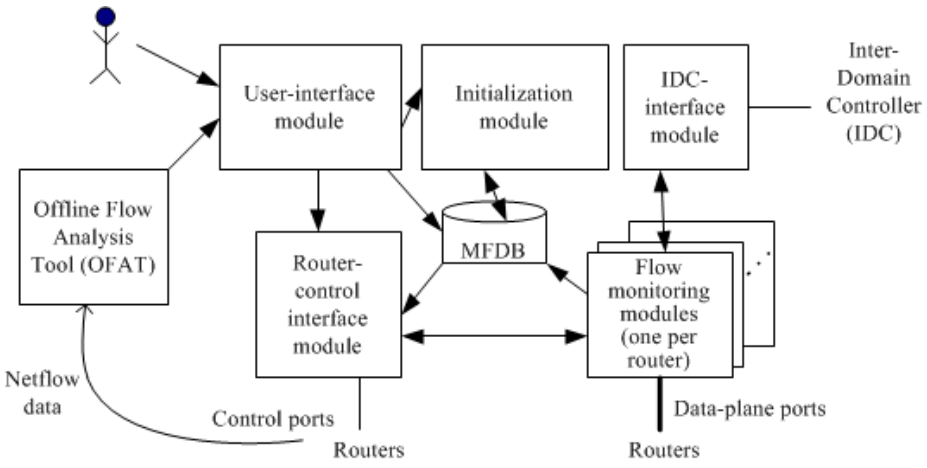


Figure 2.2: HNTES architecture

IP address had occurred often in prior NetFlow records), then the identifiers of this flow are entered into a database to be monitored. When packets from a monitored flow are captured, HNTES continues monitoring for a certain preconfigured duration. If packets continue to arrive for this flow, then it decides that the flow is a long flow, and requests an MPLS Label-Switched Path (LSP) between router R3 and router R2 passing through switches S3, S2 and other intermediate switches of ESnet. Finally, HNTES configures a policy-based route (PBR) in router R3 to forward packets from this flow to the newly established VC.

2.3.2 HNTES architecture

Fig. 2.2 shows the HNTES software architecture and its interfaces. It interfaces with the Inter-Domain Controller (IDC) and IP routers. It consists of several modules as described below:

Offline Flow Analysis Tool (OFAT): This tool collects NetFlow data from the routers, analyzes the data to find long fat flows, and populates the Monitored-flow Data Base (MFDB) with the identifiers of these flows through the user-interface module (UIM). The circuit duration field is populated by OFAT.

User-Interface Module (UIM): The purpose of this module is to allow two types of users, human administrators and software systems, such as OFAT, to enter information about flows that should be redirected to virtual circuits whenever possible. We refer to these flows as monitored flows. This module supports a graphical interface for human users as well

Table 2.1: Monitored-Flow Data Base (MFDB) structure

Flow identifiers					Status	Layer-2 or Layer-3 circuit endpoints	Circuit duration	Circuit rate
Source IP address	Destination IP address	Protocol	Source port	Destination port				
Of monitored flow (not all fields are required for each flow)					Monitored Redirected Disabled	IP addresses and VLAN IDs		

as a programmatic interface for software, such as OFAT. Information entered through this module is saved in the MFDB. All users should be authenticated.

Monitored-Flow Data Base (MFDB): The structure of this database is as shown in Table 2.1. Flow identifiers are subsets of the 5-tuple: source IP address, destination IP address, protocol, source port, and destination port. The Status field shows whether the flow is currently just being monitored or whether it has already been redirected to a virtual circuit. It could be Disabled if the user who added this flow information decides to stop monitoring this flow. It is useful to store information for a disabled flow if there is a possibility that it will be monitored again in the future. The Layer-2 or Layer-3 circuit endpoints corresponding to each flow are computed by the initialization module. Finally, the circuit duration and rate desired for each flow should be precomputed and stored.

Router-Control Interface Module (RCIM): This module is triggered by the User-Interface Module to configure the router (through its control port) to mirror monitored flows to a data-plane interface leading to a server on which the Flow-Monitoring Module is executed. In other words, as packets are forwarded by the IP router to the normal data-plane interface as determined by the routing table, the packets from monitored flows are also mirrored to the Flow-Monitoring Module.

Flow-Monitoring Modules (FMM): One instance of this module will likely need to run per monitored router. When this module receives packets from a monitored flow, it continues monitoring for a certain preconfigured duration, and if packets continue to arrive for this flow, then it decides that the flow is a long flow. It then initiates the reservation and provisioning of a virtual circuit between the Layer-2 or Layer-3 circuit endpoints identified for that flow in the MFDB. It does this by sending a message to the IDC Interface Module. For heavy flow

volumes, a multiple node cluster implementation may be required for this module. These modules also update the MFDB entries based on observations of flow durations, and change the Status field of the flow. They initiate path termination by sending a message to the IDC Interface Module if the packets seen for a redirected flow start trickling down to a light level.

Inter-domain Controller (IDC) Interface Module (IDCIM): Upon receiving messages from the FMM, the IDCIM formulates IDC Protocol (IDCP) [23] Simple Object Access Protocol (SOAP) [25] messages and requests the reservation and provisioning of circuits and release of circuits. In a circuit request, it passes the IDC the circuit endpoints (referred to as Layer-2 information in IDCP) as well as the flow identifier (subset of 5-tuple, source IP address, destination IP address, source port number, destination port number, IP protocol type), referred to as Layer-3 information in IDCP. The IDC configures policy-based routes in the routers at the ends of the circuits after a circuit is provisioned or released.

Initialization module (IM): This module computes the endpoints of Layer-2 or Layer-3 circuits corresponding to each monitored flow. This information can be obtained from the IDC and/or routing information bases (RIBs) in routers. The initialization module also computes an appropriate value for the circuit rate desired for each monitored flow and enters this information in the MFDB.

2.4 HNTES1 prototyping and evaluation

2.4.1 Prototype implementation and testing

A prototype implementation of OFAT, UIM, MFDB, FMM, and IDCIM was completed. Of these modules, the UIM, MFDB, and FMM were tested on a DOE-funded wide-area state-of-the-art 100 Gbps hybrid network testbed called Advanced Networking Initiative (ANI) [26] tabletop testbed, while OFAT and IDCIM were tested on UVA servers. UVA's computing research clusters [27] were used for OFAT testing, and the MFDB was run on UVA's MySQL server. IDCIM was tested on our networking laboratory computers. A test IDC was made available to us by ESnet. This was used to test our IDCIM software.

The FMM uses the pcap library [28]. Both FMM and UIM are implemented in C++, MFDB in SQL, IDCIM in Java, and OFAT in the statistical programming R language. The IDCIM is a modified version of the OSCARS Java Client [24].

2.4.2 NetFlow analysis

Part of the task of classifying packets into flows is done by NetFlow [6], which is a feature available in most IP routers. NetFlow enables IP routers to collect a sample of packet headers, which carry the 5-tuple flow identification information described above. Each IP router's NetFlow system maintains a running set of flow reports. For each such report, it maintains the timestamp of the first and last packets as well as the total flow size. At the end of each active timeout interval, which is a configurable parameter and typically set to 60 seconds, the stored flow reports are exported from the IP router to a collector. Processing and maintaining volumes of NetFlow data can be both computationally and storage intensive, especially for routers with high-speed links. Therefore, packet sampling is used, e.g., in 2011, NetFlow was configured to sample 1-in-100 and 1-in-1000 packets in Internet2 routers and ESnet routers, respectively.

Through an analysis of NetFlow data obtained from Internet2 routers, a few scientific applications have been identified as generating long flows. For example, the Unidata Local Data Manager (LDM) [29] application is used by the Earth systems scientific community. NetFlow was configured in Internet2 routers to report out on a flow every 60 sec. Each Netflow file consists of flow information collected over a 5-minute interval. An examination of five consecutive 5-minute files showed multiple flow-export reports on the same flow. The source TCP port number was 388, which corresponds to the Unidata LDM application. Flows from consecutive 5-minute files were concatenated to determine the actual length of flows. Since a report was exported for a flow every 60 secs, to determine the true length of long flows, several 5-minute NetFlow files need to be analyzed. A whole day's NetFlow (5-min) files were downloaded for one core router.

For this flow, an entry was created in the MFDB using the source and destination IP addresses, protocol (6 for TCP) and source port 388. The status field was set to "Monitored," and the UIM invoked the RCIM to program the router to mirror packets of the flow to the

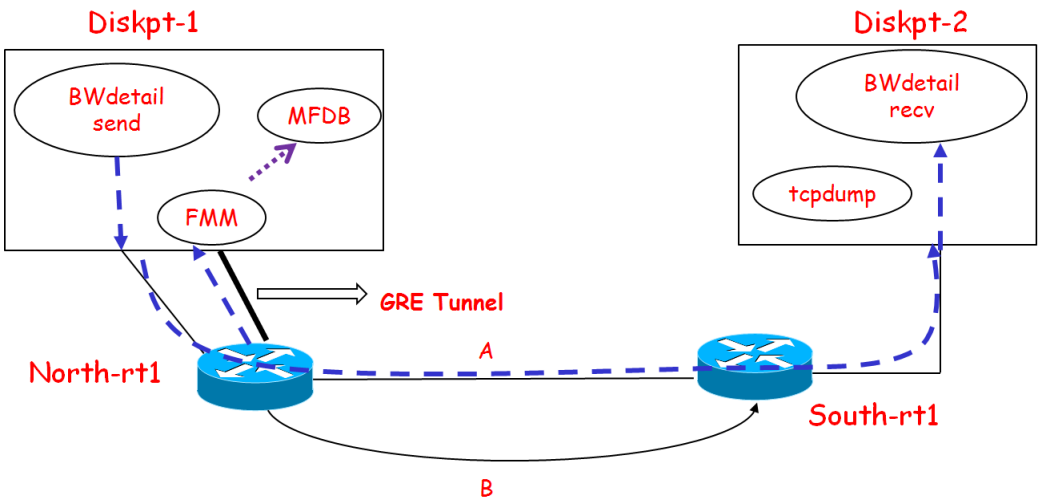


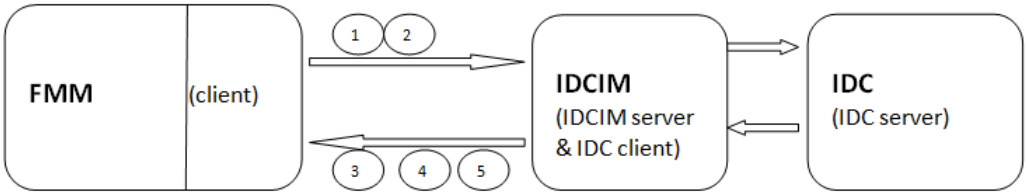
Figure 2.3: HNTES experiments on ANI tabletop testbed

FMM. Upon observing packets from this flow, the FMM initiated circuit setup through the IDCIM and changed the status of the flow in the MFDB to "Redirected."

2.4.3 HNTES experimental evaluation

Two sets of experiments were executed to test several HNTES modules. The *first* set of experiments were conducted on the ANI testbed. These experiments tested the functional operation of the FMM and MFDB modules. The test configuration is shown in Fig. 2.3. The `BWdetail` tool [30] was used to transfer files from host `Diskpt-1` to host `Diskpt-2`. The path passed through two IP routers (`North-rt1` and `South-rt1`). There were two paths (A and B) between the two routers. While both paths were single links in this experimental setup, Path A was viewed as emulation of an IP-routed path that could traverse multiple IP routers, and path B emulated a rate-guaranteed circuit for α flows. For testing purposes, the FMM and MFDB were modules installed on the same host `Diskpt-1` as the application software `BWdetail`, though in actual deployments, HNTES modules should be installed on servers that are directly connected to IP routers. There were two links that connected `Diskpt-1` with `North-rt1`, one of which was used for the `BWdetail` flows, the second (shown in bold black) was used for port-mirroring packets of α flows to the FMM. A Generic Routing Encapsulation (GRE) [31] tunnel is created to enable port mirroring.

The experiment consists of the following steps:



	Message type	parameters
from FMM to IDCIM	SetupCircuit (1)	5-tuple, rate, duration, circuit source, circuit destination
	ReleaseCircuit(2)	5-tuple
from IDCIM to FMM	CircuitSetupSuccessful (3)	5-tuple
	CircuitSetupFailed (4)	5-tuple
	CircuitReleased(5)	5-tuple

Figure 2.4: IDCIM interaction with FMM and IDC

- Enter the `BWdetail` flow's 5-tuple into the MFDB, and configure `North-rt1` to port mirror packets from this flow to the FMM on `Diskpt-1` through the GRE tunnel.
- Initiate the `BWdetail` file transfer, and check whether the FMM receives the packets from that flow. The FMM reads the flow status in the MFDB for the 5-tuple corresponding to the `BWdetail` flow. Both success and failure scenarios were tested.
- The PBR table of router `North-rt1` was configured manually to enable redirection of the `BWdetail` flow through path B. The `tcpdump` tool was used to verify that the flow redirection occurred correctly.

The *second* set of experiments tests the interaction of IDCIM with the FMM and the IDC as illustrated in Fig. 2.4. The IDCIM was executed on a host, `zelda2`, located in our networking laboratory at UVA, and a test IDC deployed by ESnet strictly for testing purposes was used for this set of experiments. The reason for using this test IDC is that the ANI Tabletop testbed did not have a deployed IDC.

When the port mirrored packets arriving at the FMM match a flow configured in the MFDB, the FMM sends a `SetupCircuit` message (type 1 in Fig. 2.4) with parameters consisting of the 5-tuple flow identifier, circuit rate, circuit duration, circuit source, and destination, to the IDCIM. Upon receiving this message, the IDCIM sends a `createReservation` message with the automatic signaling option (which indicates to the IDC that it should

automatically initiate path provisioning at the scheduled start time). Since this circuit reservation is for an already ongoing flow, the requested reservation is for an immediate start. When circuit setup is successful, a $PATH\_SETUP\_COMPLETED$ message is sent back to the IDCIM. After receiving this message, the IDCIM generates a `CircuitSetupSuccessful` message (type 3) with the 5-tuple flow identifier as a parameter for transmission to the FMM. When the circuit setup fails, a $PATH\_SETUP\_FAILED$ message is sent from the IDC server to the IDCIM, which then triggers the IDCIM to generate a `CircuitSetupFailed` (type 4) message to the FMM. The FMM continues its monitoring action for port-mirrored flows. If no packets are received for a redirected flow for a certain duration, which implies that the redirected flow terminated earlier than the predicted duration that was stored in the MFDB, a `ReleaseCircuit` (type 2) message is generated by the FMM and sent to the IDCIM. The latter sends a `TeardownCircuit` message to the IDC server. Upon receiving a $PATH\_TEARDOWN\_COMPLETED$ message, the IDCIM formulates a `CircuitReleased` message of type 5 and sends this to the FMM. In summary, the IDCIM operates as a "server" to the FMM client, while it is itself a "client" to the IDC server.

2.5 Discussion

Based on the results of our experiments and NetFlow data analysis, certain practical concerns were identified. First, the *Flow Monitoring Module (FMM) was deemed impractical* because it needs to process headers from all mirrored data packets in real time. As link speeds increase, such a task would require a high-performance computing cluster. Requiring the deployment of such clusters to serve one or more ESnet routers could be cost prohibitive.

To avoid the costs of analyzing all mirrored packets, we considered configuring the router to port mirror just control-plane packets. Control-plane communications are used to set up data-plane connections for actual user data transfer. For example, the GridFTP architecture consists of a client Protocol Interpreter (PI), one server PI at each end, and one or more Data Transfer Processes (DTPs). Transfers can occur in first-party mode or third-party mode. In first-party mode, the host on which the client PI is being run will typically also be running the DTP. If the server PI and server DTP are also run on the same host, the

source and destination IP addresses for the data connection will be the same as those of the control packets, which can be extracted from the mirrored control-plane packet headers. For third-party transfers, the data-connection IP addresses need to be extracted from commands such as PASV/SPAS and PORT/SPOR [32]. The port numbers of control-plane connections are often well-known, such as 2811 for GridFTP, but those of the data-plane connections are ephemeral. But through deep packet inspection of the PASV/SPAS and PORT/SPOR control-plane packets, the TCP port numbers of data-plane connections could be extracted. However, we learned that the Globus GridFTP implementation includes RFC 2228 [33], FTP Security Extensions, which requires control-plane packets to be *encrypted, making this scheme infeasible.* As a result, we decided to drop the FMM from the HNTES architecture. Without port-mirrored packet analysis, only offline flow-data analysis mechanisms are feasible to determine IP addresses of source-destination pairs that generate heavy-hitter flows.

The offline solution could work if a hypothesis that most heavy-hitter flows are repeatedly generated between the same source-destination pairs holds true. This allows for the identification of source-destination IP addresses of completed heavy-hitter flows for use in PBRs to redirect future heavy-hitter flows. Such a solution was explored, and is presented in the next chapter.

If the first task shown in Fig. 1.1 of Chapter 1 is offline, then the second task, i.e., *circuit provisioning, necessarily becomes offline too.* The implication of circuit provisioning being offline is that the circuits are static (i.e., not dynamically set up after a heavy-hitter flow is identified). Recall that HNTES operates within a provider's network and executes only intra-domain traffic engineering. Therefore the endpoints of a circuit are ingress and egress IP routers of the provider's network. Based on the source and destination IP addresses of completed heavy-hitter flows, the corresponding ingress and egress IP routers are identified from routing tables, and used for static circuit provisioning.

Given that the virtual circuits (MPLS LSPs) span between ingress and egress IP routers of a provider's network, it became unnecessary to assign single flows to circuits because this requires fine-grained resource control. Instead, in our new design, all heavy-hitter flows between the same ingress-egress ESnet routers would be redirected to the same virtual circuit. This coarse-grained approach is more practical since the number of virtual queues that can

be created within an egress interface's buffer is on the order of tens to hundreds. While the number of heavy-hitter flows is likely to be a small fraction of the total number of flows, it is possible to exceed this limit on the number of virtual queues if queues are configured for individual flows. Therefore this approach of *sharing a single virtual queue among multiple heavy-hitter flows* was adopted. The implications of this approach are studied in-depth in Chapter 4.

Finally, our focus on the duration dimension of flows in HNTES 1.0 could be changed to other flow dimensions because dynamic VC setup was no longer required. Therefore, we used a top-down approach to determine which of the four dimensions: duration, rate, size, and burstiness, was most important for traffic engineering. For successful sharing of a network infrastructure, one flow should not have adverse effects on another flow. Using this top-down approach (in contrast to the bottom-up approach of HNTES 1.0 where the need to use dynamic VCs drove our decision to consider the duration dimension), we ran experiments to determine which dimension was the most significant. Our answer was *rate in combination with size*, which are the dimensions used in the HNTES 2.0 design presented in the next chapter.

2.6 Conclusions

HNTES 1.0 is an online system to automatically identify heavy-hitter flows, request a dynamic virtual circuit (VC) from a circuit scheduler/provisioning system that sets up the VC and configures a policy based route in the ingress router to redirect packets from the identified heavy-hitter flow to the newly established VC. However, certain practical concerns were identified through NetFlow analysis and testbed experiments: (i) it is cost prohibitive to deploy Flow-Monitoring Modules (FMMs) that can keep pace with increasing backbone link rates; (ii) without online heavy-hitter flow identification, circuits need to be static; (iii) static circuits between ingress and egress router pairs can be shared among multiple heavy-hitter flows, and (iv) dimensions other than duration can be considered when determining heavy-hitter flows as VC setup delay is no longer an issue. These concerns led

to change in focus from an online HNTES design to an offline design. Our offline design is presented in the next chapter.

Chapter 3

Alpha Flow Traffic Engineering System (HNTES 2.0)

3.1 Introduction

In this chapter, we describe our design for an offline HNTES, HNTES 2.0. All three tasks described in Fig. 1.1 of Chapter 1 are performed in offline mode. As noted in Section 2.5, as the identification of heavy-hitter flows is done offline, i.e., on completed flows, the first task just yields source-destination IP address pairs. The second task is correspondingly offline, i.e., static virtual circuits are created between ingress-egress router pairs of the provider's network. The third task, Policy Based Route (PBR) configuration, is also executed offline. PBRs use just source and destination IP addresses to identify packets from potential heavy-hitter flows for redirection to traffic-engineered, QoS-controlled virtual circuits. Port numbers are not included in PBRs because this is an offline approach and port numbers can be ephemeral.

We also noted in Section 2.5 that without the constraint of VC setup delay, which is incurred if circuit provisioning is online, other dimensions of flows can be considered in deciding if a flow is a heavy-hitter or not. To select the appropriate dimension, we used a top-down approach to determine what types of heavy-hitter flows impact other flows. Our conclusion was that long-duration flows, if they are low-rate, e.g., the LDM port 388 flows, do not have adverse effects on other flows. By "low-rate," we mean the flow rate is a

small fraction of link capacity. Experiments showed that file-transfer flows could reach high rates, and given the nature of TCP, these flows generate packets in bursts. A large burst of packets can cause router buffers to fill up, which can cause increased delays for real-time flows. Section 3.2 describes one such experiment.

High-rate transfers that are of large-sized are particularly adverse to other flows as they have a higher potential for overlapping with real-time flows. Such high-rate, large-sized flows are referred to as α flows. Correspondingly, we changed the name of the system to Alpha-Flow Traffic Engineering System (AFTES). It is effectively a HNTES 2.0 system.

After presenting our motivation for choosing rate and size as the dimensions of interest for traffic engineering heavy-hitter flows in Section 3.2, related work is reviewed in Section 3.3. The basis for the AFTES solution is discussed in Section 3.4. In addition to its role in traffic engineering α flows, AFTES can be used to just obtain characteristics of α flows. Section 3.5 describes our methodology for characterizing α flows, and provides characteristics of α flows observed in the 7-month ESnet NetFlow data. AFTES evaluation (hypothesis testing) is presented in Section 3.6, and the chapter is concluded in Section 3.7.

3.2 Motivation

First, we conducted experiments on a high-speed network testbed to determine if a single high-rate TCP file-transfer flow could impact packet delays for a real-time flow (Section 3.2.1). *Second*, we obtained measurements on ESnet links to determine whether high rate spikes, such as those created in our experiment, occur in practice (Section 3.2.2). *Finally*, we analyzed file-transfer usage logs from GridFTP servers deployed in national scientific supercomputing centers to determine where actual file-transfer rates reach a high fraction of link capacity (Section 3.2.3).

3.2.1 An experiment conducted on a high-speed network testbed

We conducted an experiment on a high-speed DOE metropolitan-area IP-routed network testbed. Its high-end computing hosts were capable of sourcing/sinking data at multiple Gbps [1]. All links in this network were 10 Gbps Ethernet. Three flows were created

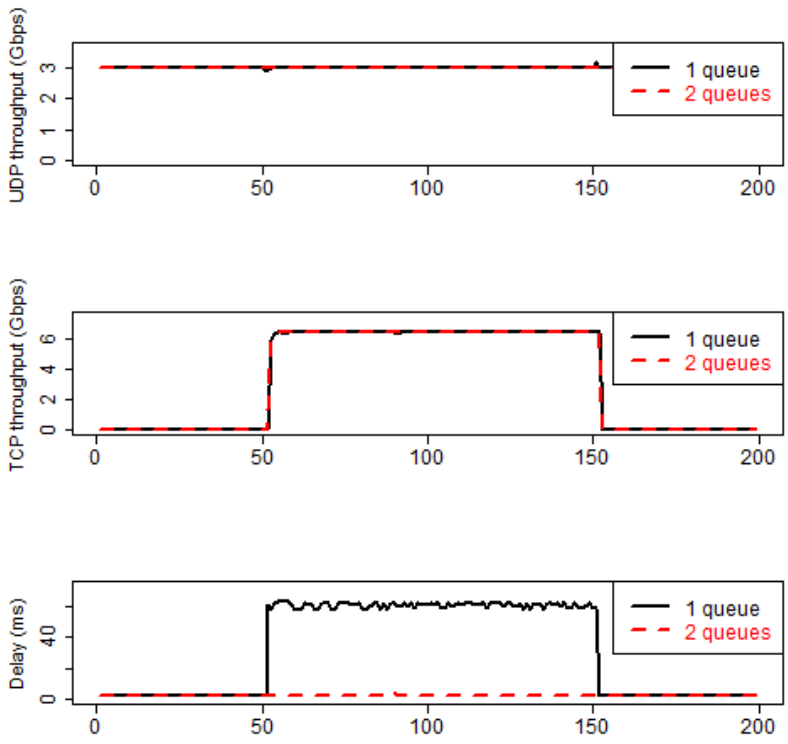


Figure 3.1: The x-axis is time measured in seconds; the top graph shows that the UDP-flow rate is 3 Gbps in both the 1-queue and 2-queues configurations; the middle graph shows the TCP flow throughput; the bottom graph shows the delays experienced in the ping application [1]

to compete for resources on a router's outgoing interface: (i) an *nuttcp* UDP flow that generated packets at 3 Gbps to emulate background traffic, (ii) an *nuttcp* TCP flow that was initiated at time 53 seconds after the start of the experiment, and (iii) a `ping` flow that sent requests every 1 sec to measure round-trip time, as shown in the three graphs of Fig. 3.1. In the 1-queue configuration, all three flows were served with best-effort IP service, i.e., their packets were buffered in the same output queue. The TCP flow throughput enjoyed more than 6 Gbps, but the ping flow delays increased from *2.3 ms* (when the TCP flow was absent) to *60.6 ms* when the TCP flow was initiated, and stayed high at 60.6 ms while the TCP flow was active. In the 2-queues configuration, the router directed the UDP-flow and ping-flow packets to one virtual queue and the TCP-flow packets to a separate virtual queue on the contending egress interface. Weighted-fair queueing (WFQ) was used with a 40-60 split between the first and second queues, but the transmitter was configured to operate

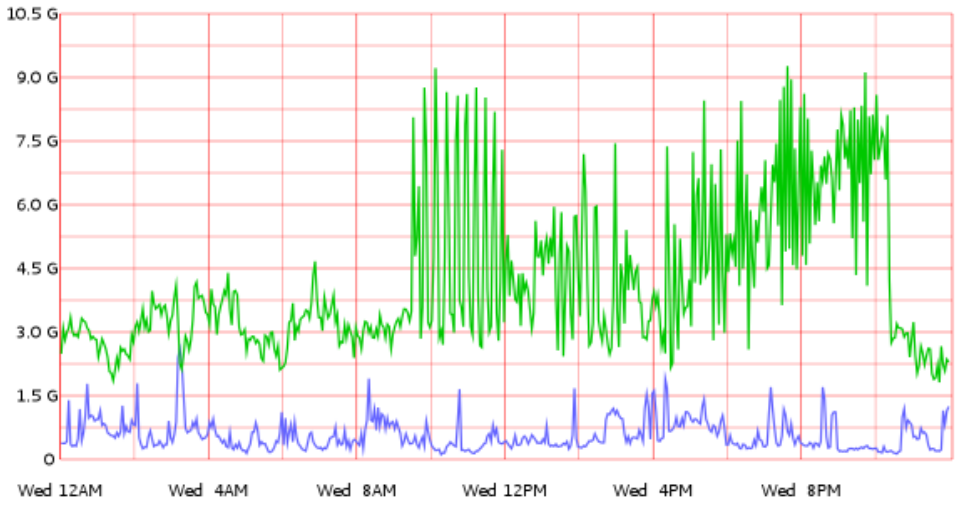


Figure 3.2: Utilization (b/s) of a 10 Gbps ESnet router interface observed on Jan. 16, 2013; the green line, showing bursts reaching over 9 Gbps, is the outgoing traffic from the ESnet router to a peering REN, while the lower-load blue line is the incoming traffic on the same interface

in a work-conserving mode, which means that if one of the virtual queues does not have packets, the other queue will be served even in excess of its rate allocation. This mode of operation allowed the TCP flow to still enjoy 6 Gbps, while the ping delay stayed low at *2.3 ms* even in the presence of the TCP flow. This experiment illustrates that high-rate flows can have adverse effects on delay-sensitive flows.

3.2.2 ESnet link usage measurements

While high-rate flows can be created on an experimental testbed to fill up a link as in our experiment, we checked actual traffic levels on ESnet interfaces to determine if aggregate rates ever approached link capacity. Fig. 3.2 plots Simple Network Management Protocol (SNMP) link utilization of an ESnet router interface. This data was collected on Jan. 16, 2013. The traffic level reached above 9 Gbps (the link's capacity was 10 Gbps). These sudden increases in traffic levels are most commonly caused by a single or few heavy-hitter flows. Research-and-Education Networks (RENs) such as ESnet are engineered to operate at 20-25% loads in order to absorb α-flow spikes [34]. Typically, traffic in both directions of the link shown in Fig. 3.2 have loads similar to that of the blue line (incoming direction) [35]. However, in spite of the capacity headroom, heavy-hitter flows can lead to such surges

in traffic that would result in buffer build-ups. The SNMP data are time-averaged over 30-sec intervals; on shorter timescales, buffers will fill up as the TCP flow rate will be 10 Gbps in short bursts because the sending host network interface card rate is 10 Gbps (this phenomenon is illustrated in Chapter 4).

3.2.3 GridFTP file transfer log analysis

Since we found that network link utilization could sometimes reach close to 100%, we were interested in analyzing usage logs from actual scientific data transfers to check if the throughput of such transfers reached significant fractions of link capacity. Toward this end, we obtained GridFTP usage logs from the National Center for Atmospheric Research (NCAR). This set of logs showed metadata about file transfers from NCAR to the National Institute for Computational Sciences (NICS) [36] for the period 2009-2011. The set consisted of metadata, such as transfer size and transfer duration, for $52,454$ transfers. Often scientists move lots of files because their simulation programs or experiments create many files. Scripts were used to have GridFTP move all files in one or more directories. The GridFTP usage logger logs information for each file. Each such file movement is referred to as a "transfer," and "sessions" consist of one-or-more transfers executed in batch mode by an automated script. A configurable parameter, g, was used to set the maximum allowed gap between the end of one transfer and the start of the next transfer within a session (logs do not show whether transfers were part of session or not).

Table 3.1 shows the size and duration statistics about *sessions*, and throughput statistics about *transfers*, assuming g is 1 min. The maximum transfer throughput observed was 4.23 Gbps, which means this single transfer utilized more than 40% of the link bandwidth. This throughput value is determined by dividing transfer size by transfer duration; it therefore represents an average rate, not the instantaneous rate. The actual packet transfer pattern would have been bursty since the underlying transport protocol was TCP. This particular transfer must have occurred from a host with a 10 Gbps network interface card (NIC) since Ethernet NICs support one of four rates: 10 Mbps, 100 Mbps, 1 Gbps or 10 Gbps. Therefore, we can surmise that within the lifetime of this transfer, there would have been bursts of packets that were transmitted at 10 Gbps. Since most REN link capacities were 10 Gbps

Table 3.1: NCAR-NICS sessions and transfers; $g = 1$ min

Characterization of session sizes, in MB

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
8,793 (bytes)	5,808.7	70,708.4	263,771.4	320,600	2,873,868.5

Characterization of session durations, in s

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
0.05	210.5	1,445	4,039	5,261	48,420

Characterization of transfer throughput, in Mbps

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
2.1 (bps)	298	468.3	506.1	682.2	4,227

during the 2009-2011 period, such bursts could have caused buffer buildups in the presence of background traffic that was typically in the 2-3 Gbps range.

Other interesting numbers shown in Table 3.1 are the maximum size (over 2.8 TB) and maximum duration (over 13 hours). These numbers show that indeed large datasets are moved by scientists using these computing sites.

3.3 Related Work

As mentioned in Section 2.2, Lan and Heidemann [5] identify four dimensions of flows: size (bytes), rate, duration, and burstiness. Further, four methods are used to define the thresholds between the two classes for each of these dimensions. In the first method, the numbers corresponding to the mean + 3 standard deviations of the sampled data are used as the thresholds, based on earlier work by Sarvotham et al. [37]. In the second method, the top 1% of all flows is used to set the thresholds. In the third method, the thresholds are cutoff points in a heavy-tailed distribution where the cutoff is determined using the *aest* method [38]. In the fourth method, the thresholds are set so that 50% of all traffic is carried by the heavy-hitters (elephants, cheetahs, etc.). The goals of the Lan and Heidemann study were to characterize flows along the four dimensions and to study correlations between different flow types, while the purpose of Sarvotham's study was to create a framework for modeling network traffic.

Papagiannaki et al. [39] have the same stated goal as ours of identifying elephant flows for traffic engineering. Their threshold for elephant flows is varied based on load. They conclude that single-feature methods lead to the detection of short-lived elephant flows, while a two-feature method that uses a latent-heat metric to take into account persistence makes it feasible to identify only long-lived elephant flows that are better candidates for traffic engineering. For their analysis purposes, Wallerich et. al [40] classified a flow as an elephant if in any time bin (which was chosen to be 1 min) in its lifetime, the number of bytes sent is in the top ranks of all flows. Other such papers are surveyed by Callado, et al., in a 2009 paper [41].

These prior definitions of heavy-hitter flows are based on relative values among all flows. In contrast, our definition of α flows uses a fixed threshold for bytes observed within a fixed duration. Only flows that exceed the threshold are likely to have adverse effects on real-time flows. Using relative values may result in the classification of certain flows as elephant or α flows, even if their rates/sizes are not large enough to have adverse effects on other flows. Therefore our work uses absolute thresholds. These threshold values are based on an analysis of actual scientific data transfers logged by GridFTP servers [18].

There are several papers proposing methods for identifying large flows or high-rate flows with new router hardware. These include ElephantTrap [42], RATE [43], CATE [44], an FPGA-based cache solution [45], and a Grid flow real-time detector for 1 Gbps links [46]. Also Hohn and Veitch [47] proposed a scheme for finding the spectral density, distribution of the number of packets per flow, and showed why alternate sampling techniques were needed to obtain this second-order statistic about flows. Given our focus on designing network management systems and not new router hardware, our scheme relies on the built-in NetFlow system supported in most deployed provider routers.

Kamiyama and Mori propose a short-timeout method to identify high-rate flows [48] and elephant (large) flows [49] with low false-positive and false-negative rates, but not to determine the flow rates or sizes. Zhang, Fang and Zhang [50] proposed a Bayesian single sampling method to identify high-rate flows, but again not to characterize their sizes/rates.

Duffield, Lund and Thorup [51] had a goal of finding information about flows in unsampled packets using information in sampled packets. In contrast, our goal is more specific to

characterizing α flows. Given the higher rate of sampling of these flows, our method will result in higher accuracy but is not as general in its scope [51].

Fioreze et al. [52] proposed redirecting elephant flows (which they defined as long in duration and big in volume) to optical circuits in hybrid IP/optical networks. The focus of this work was on evaluating the impact of sampling rates.

The automatic identification of flows have been proposed in the HELIOS low-power optical circuit switches to complement electronic packet-switched networks within datacenters [53–55]. In these hybrid networks such as the Helios architecture [53], elephant flows could be identified for automatic redirection to optical circuits. By strategically placing an online HNTES on links to filesystem servers, such flows could be identified live and redirected.

Other papers of relevance are on flow classification algorithms [56, 57]. Flow classification methods are grouped in [57] into three categories: (i) port based, (ii) payload based, and (iii) flow statistics based. Our solution falls in the last category. In addition, machine learning techniques are used because port and payload based methods have drawbacks caused by port masquerading and payload encryption, respectively. Scientific data transfer applications, such as GridFTP, use ephemeral ports and control-plane packet encryption, which make these methods unsuitable for our purposes. While machine learning methods can be explored for our problem, they are likely to be more complex than our solution of using flow statistics, which is sufficiently effective as shown in Section 3.6.

3.4 Basis for AFTES solution

This section provides the basis for the AFTES solution. *First*, an architectural overview of AFTES is provided. *Second*, several hypotheses that underpin the AFTES solution are presented. If these hypotheses are true, AFTES would be highly effective for α-flow identification. Results of the hypothesis testing will be presented in a later section. *Finally*, an experimental study was conducted on a ESnet path to test if NetFlow reports can be used to identify α flows because ESnet configures its routers to sample 1-in-1000 packets for NetFlow report creation.

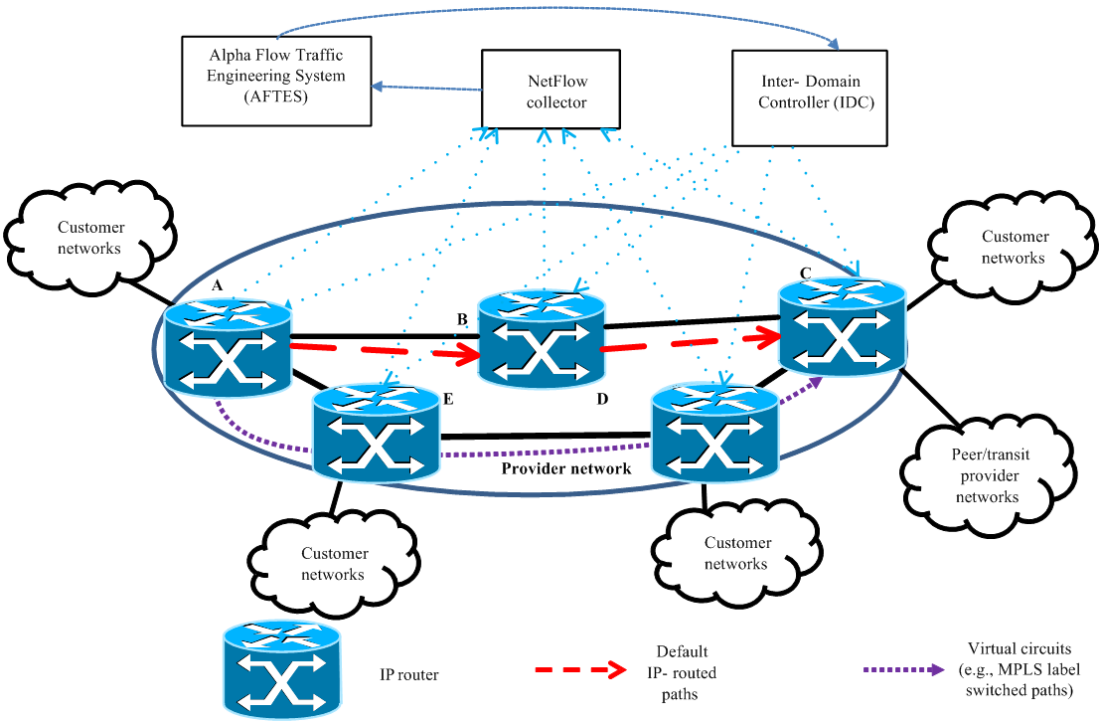


Figure 3.3: Illustration of the role of Alpha Flow Traffic Engineering System (AFTES)

3.4.1 Overview of AFTES architecture

Consider the example provider network shown in Fig. 3.3. Default IP-routed paths from router A to router C in the example provider network are shown with red dashed lines. *AFTES* is a network management software system that would be run on an *external* server as shown in Fig. 3.3. AFTES interacts with two external systems, a *NetFlow collector*, and an *Inter-Domain Controller (IDC)*. The role of a NetFlow collector is explained in Section 2.4.2, and the purpose of the IDC is described in Section 2.2. As mentioned earlier, MultiProtocol Label Switching (MPLS) is used for dynamic virtual circuit service. AFTES leverages this service for α flows. The setup phase in VC networking offers the opportunity for traffic engineering α flows along paths distinct from the default IP-routed paths if needed (e.g., for load balancing).

AFTES operations are grouped into two phases:

α-flow address prefix identification Periodically AFTES obtains NetFlow reports from the NetFlow collector (as shown in Fig. 3.3), and analyzes these reports to identify the source

and destination IP address prefixes of completed α flows. Details regarding the thresholds used in determining which flows are α flows are provided in the next section. Assuming that AFTES runs on a nightly basis, it creates a list of α *prefix IDs,* which are source-destination address prefixes of α flows observed during the day, to store in a set $\mathbf{F}_i$, where i is a time (e.g., per-day) index. These identifiers are referred to as prefix IDs because address prefixes rather than complete addresses may be used (a /24 subnet identifier is an address prefix that represents a group of 255 /32 IPv4 addresses). To keep the set $\mathbf{F}_i$ from becoming too large, address prefix pairs for which α flows have not been observed within an aging interval (e.g., 30 days) will be deleted.

Configuring routers for future α-flow redirection The source-destination IP address prefix pairs in $\mathbf{F}_i$ are used to set firewall filter rules (previously referred to as policy-based routes) at each ingress router I to separate out packets of future α flows and redirect them to traffic-engineered, QoS-controlled virtual circuits. Like HNTES 1.0, AFTES is designed for intra-domain usage by any single provider. The technological solution of carrying IP packets over MPLS label switched paths (LSPs) for segments of an end-to-end path is leveraged by AFTES. On each day i, AFTES determines the egress router E corresponding to each new α prefix ID stored in set $\mathbf{F}_i$, and sends requests to the IDC for an LSP, if one does not already exist between the ingress router I and egress router E. The IDC executes three steps: (i) sets up the LSP between the ingress router I and egress router E, (ii) configures QoS mechanisms, and (iii) configures a rule in the firewall filter at the ingress router to identify packets corresponding to α prefix IDs, and direct them to the virtual queue served by the MPLS LSP. If an LSP already exists between I and E corresponding to a new α prefix ID in $\mathbf{F}_i$, only steps (ii) and (iii) are required. Incoming flows on day i whose source and destination addresses match one of the α prefix IDs in the firewall filter $\mathbf{F}_i$ will be automatically classified as α flows by the router and directed to the virtual queue for the corresponding MPLS LSP.

In *summary*, the AFTES design uses an *offline* approach, in which α prefix IDs are determined through a posteriori analysis, and used to configure virtual circuits, QoS mechanisms, and firewall filters in routers so that future α flows can be identified and isolated

from general-purpose traffic.

3.4.2 Hypotheses

Several hypotheses underpin this AFTES design:

1. NetFlow data can be used to estimate the size of α flows in spite of packet sampling (e.g., 1-in-1000).

2. Most high-speed data transfer nodes have static well-known IP addresses, and α flows are repeatedly created between the same source-destination subnets. Since scientists typically execute simulations on the same supercomputing centers, it is likely that their data transfers occur repeatedly between the same two clusters.

3. New α flows can appear on any given day, for example, when a new high-speed data transfer node is added at a supercomputing data center, or when a new scientist joins a project. The use of /24 subnet addresses instead of /32 complete IPv4 addresses (a subnet is a grouping of IP addresses) in the firewall filters can reduce the number of undirected first-time α flows. An α flow generated by host with a previously unseen /32 address, but one that belongs to a /24 subnet address for which a firewall filter rule has been configured, will be automatically redirected as it will match the rule.

4. The number of α prefix identifiers for which firewall filters are configured is not large, making it manageable from an operational point of view.

5. The percentage of real-time flow packets that share α-flow prefix IDs is small as we do not expect many audio/video interactive applications to be run on hosts located in the same subnet as the clusters dedicated for handling file transfers from supercomputing centers.

6. A majority of α flows are created by large file transfers from well-equipped servers that are referred to as "data doors." These are servers dedicated to execute file-transfer applications, and are deployed in most scientific supercomputing centers and clusters dedicated for scientific research at university laboratories.

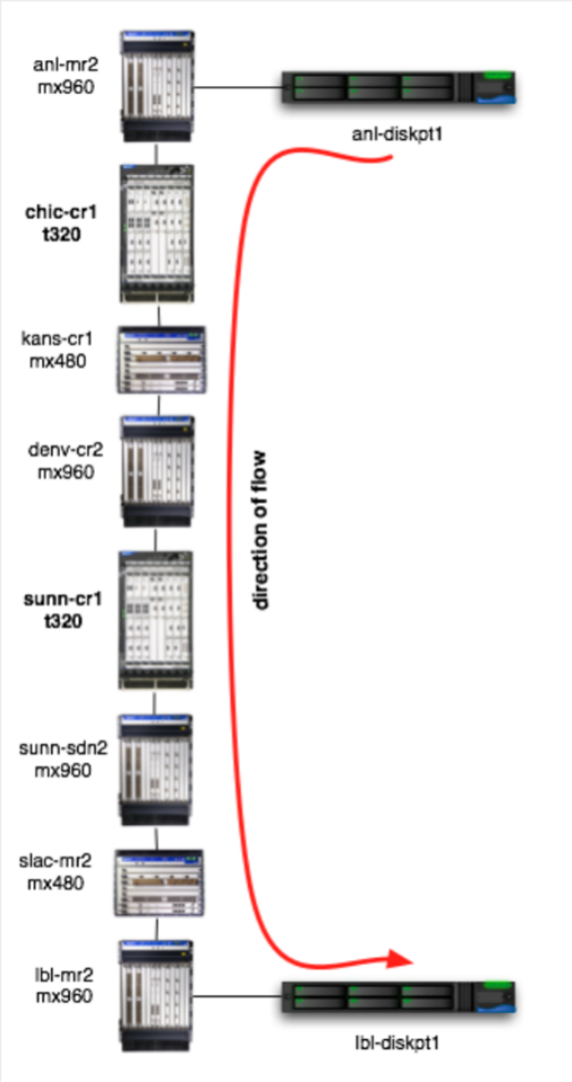


Figure 3.4: NetFlow data usability experiment

If these hypotheses prove to be true, the offline prefix identifier based AFTES solution will be highly effective in identifying and directing future α flows to traffic-engineered, QoS-controlled virtual circuits.

3.4.3 NetFlow data usability

To test the hypothesis that NetFlow data is sufficient to identify α flows in spite of the low packet sampling rate, an experimental study was conducted on a path consisting of eight (operational) ESnet IP routers.

Table 3.2: Size-accuracy ratio for files for different sizes (sample size: 50)

	NetFlow records obtained from Chicago ESnet router		NetFlow records obtained from Sunnyvale ESnet router	
	Mean	Standard deviation	Mean	Standard deviation
100 MB	0.949	0.2780	1.0812	0.3073
1 GB	0.996	0.1708	1.032	0.1653
10 GB	0.990	0.0368	0.999	0.0252

As is shown in Fig. 3.4, two high-end hosts, `anl-diskpt1` located at Argonne National Laboratory, Chicago, and `lbl-diskpt1` located at Lawrence Berkeley Laboratory, Berkeley, were used to run a GridFTP server and GridFTP client, respectively. There were eight ESnet IP routers on the path between these hosts, with 10 Gb/s links between the routers. This experimental setup was such that high rates of data transfer could be sustained as the disks in the end hosts had multi-Gbps access rates and the bottleneck link rate on the end-to-end path was 10 Gbps.

GridFTP transfers of known sizes were executed between the server and client, and NetFlow data was collected from two transit routers, `chic-cr1` and `sunn-cr1`. From the GridFTP logs stored at the server, the TCP ports of the data connections were obtained. All flow records corresponding to the GridFTP transfers were filtered out using the five-tuple identifier, and the size of each transfer was estimated using a 1000 factor multiplier on the total size reported by the flow records for the data connection, since the ESnet NetFlow sampling rate is 1-in-1000. A `size-accuracy ratio` is defined to be the ratio of the NetFlow estimated size and actual file size. For each file size (100 MB, 1 GB, and 10 GB), multiple runs were executed since the packet sampling at the router makes the size-accuracy ratio a random variable. The results obtained are shown in Table 3.2. For all three file sizes, the sample mean shows a size-accuracy ratio close to 1, and more interestingly, the standard deviation is smaller for larger files (0.28 for 100 MB files to 0.04 for 10 GB files).

These findings show that NetFlow data can be used to detect large-sized high-rate flows. Thus our first hypothesis proved to be true. Since NetFlow data is based on random packet sampling, and large-sized flows across high-rate paths have more packets within NetFlow active timeout intervals, the probability of packets from these flows being captured is higher

than those from small-sized flows, or large-sized flows across low-rate paths.

3.5 Using AFTES to characterize α flows

While the main purpose of AFTES is to find source-destination IP addresses of completed α flows and use these addresses to set firewall filters for future α-flow redirection to virtual circuits, AFTES can also be used to simply characterize α flows.

This section first describes our methodology for characterizing α flows (Section 3.5.1). This methodology was implemented using the R statistical programming language. Results from running this software on NetFlow data collected over a 7-month period (May-Nov. 2011) from an ESnet router are presented next (Section 3.5.2).

The specific α-flow characteristics computed include the maximum per-day total number of bytes sent in α intervals by any single source-destination host/subnet pair, a measure of persistency of α-flow generation by the same source-destination pairs, the cumulative per-day number of bytes sent between all source-destination pairs that generate α flows, the total amount of time when there were one or more active α flows on a router interface, and the percentage of the total traffic represented by α flows.

3.5.1 Methodology

Two terms, *5-tuple flow* and *prefix flow* [40], are used in characterizing α flows. A *5-tuple flow* consists of all packets arriving with the same 5-tuple values {source IP address, destination IP address, source port number, destination port number, protocol type} with no consecutive inter-packet gaps greater than some fixed time threshold. The fixed time threshold phrase is required because TCP and UDP port numbers for the same source-destination hosts are reused at some point in time. A *prefix flow* consists of all packets arriving within an aggregation interval (some fixed duration) that have the same source and destination address prefixes (e.g., /24 prefixes).

Tables 3.3 and 3.4 define the terminology and notation used in this work. Table 3.3 first defines α *NetFlow reports* as reports whose total bytes exceeds a threshold H. Next, Table 3.3 defines an α *flow* as a 5-tuple flow for which there is at least one α NetFlow report.

Table 3.3: Terminology

Term	Meaning
α NetFlow report	a NetFlow report in which the byte threshold H is exceeded
α flow	a 5-tuple flow (see Section 3.5.1 for definition) for which there is at least one α NetFlow report
α-interval of an α flow	interval between first-packet and last-packet timestamps in each α NetFlow report of the α flow
α prefix identifiers (IDs)	prefix identifiers (source/destination address prefixes) of α flows
α prefix flow	a prefix flow (see Section 3.5.1 for definition) in which all its packets belong to α flows that share its prefix identifier
α-bytes of an α prefix flow	sum of bytes reported in its constituent α NetFlow reports
α-time of an α prefix flow	the total time within each aggregation interval in which at least one of the constituent α flows experienced an α-interval.

Table 3.4: Sets created for the i^{th} aggregation interval

Symbol	Set	No. of elements	Elements of the set	Attributes of an element				
				Identifier	α-bytes	Start and end time	α-time	No. of α NetFlow reports aggregated
$\mathbf{R}_i$	Set of α NetFlow reports	m_i	$\mathbf{r}_{ij}$	ω_{ij}	β_{ij}	(s_{ij}, e_{ij})	NA	NA
$\mathbf{P}_i$	Set of α prefix flows	d_i	$\mathbf{p}_{il}$	ζ_{il}	η_{il}	NA	μ_{il}	g_{il}

Thus, an α flow may have some α NetFlow reports and some non-α NetFlow reports. An α flow can have one or more *α-intervals*, each of which is a time period in which the α flow sent packets whose aggregate size exceeded H bytes within a NetFlow active timeout period (denoted τ). Finally the source and destination address prefixes of α flows are referred to as *α prefix identifiers*.

The next three terms in Table 3.3 define *α prefix flows* and their characteristics, *α-bytes* and *α-time*. A prefix flow is an *α prefix flow* if its source/destination address prefix is an α prefix identifier. Two characteristics of an α prefix flow, *α-bytes* and *α-time*, are used to represent the aggregate bytes and total time observed in α-intervals of α flows that shared the prefix flow's identifier.

Table 3.4 lists notation for parameters of α NetFlow reports and α prefix flows, which are used to compute α-bytes and α-time for α prefix flows. In day i, the set of NetFlow reports $\mathbf{R}_i$ has m_i α NetFlow reports. Each report $\mathbf{r}_{ij}$ is itself a set consisting of several parameters, such as the number of bytes β_{ij}, source/destination address prefixes denoted as the identifier, ω_{ij}, and start time s_{ij} and end time e_{ij}, which represent the UTC timestamps of the first and last packets in the NetFlow report, respectively. The number of bytes in all NetFlow reports in set $\mathbf{R}_i$ are lower-bounded by H, i.e.,

$$\beta_{ij} \geq H,\ 1 \leq j \leq m_i \tag{3.1}$$

The next row in Table 3.4 represents the parameters of α prefix flows. On each day i, a set $\mathbf{P}_i$ of α prefix flows is created. Each element in this set is itself a set consisting of an identifier ζ_{il}, which is the source/destination address prefix pair, and other parameters as shown in Table 3.4.

Below, the α-bytes and α-time of each α prefix flow are characterized. For each α prefix flow $\mathbf{p}_{il}$, the α-bytes, η_{il} (see Table 3.4), is determined as follows:

$$\eta_{il} = \sum_{j=1}^{g_{il}} \beta_{ij}, \text{s.t.}, j\ \epsilon\ \mathbf{J}_{il},\ 1 \leq l \leq d_i \tag{3.2}$$

where $\mathbf{J}_{il} = \{j_1, j_2, \cdots, j_{g_{il}}\}$, a set of indices selected from report set $\mathbf{R}_i$ such that the

source/destination address prefixes in identifiers ω_{ij_a} are equal to ζ_{il} for $1 \leq a \leq g_{il}$ and $1 \leq j_a \leq m_i$ (see Table 3.4).

To determine α-time of $\mathbf{p}_{il}$, the following procedure is used. If there are multiple α flows that share the prefix ID of an α prefix flow $\mathbf{p}_{il}$, these flows could have overlapping α-intervals. Such overlapping intervals should be counted only once. The α-intervals of constituent α flows in an α prefix flow $\mathbf{p}_{il}$ are divided into two sets: $\mathbf{O}_{il}$ consisting of x_{il} overlapping α-intervals, and $\mathbf{N}_{il}$ consisting of y_{il} non-overlapping α-intervals. A new set of non-overlapping intervals $\mathbf{M}_{il}$ of size u_{il} is derived from $\mathbf{O}_{il}$ as follows: from a contiguous set of overlapping α-intervals within set $\mathbf{O}_{il}$, a new interval is created for set $\mathbf{M}_{il}$ with the earliest start time, s^e_{iv}, and the latest end time, e^l_{iv}, $v\ \epsilon(1, u_{il})$. The *α-time*, μ_{il} (see Table 3.4), is then computed as

$$\mu_{il} \triangleq \sum_{v=1}^{u_{il}} (e^l_{iv} - s^e_{iv}) + \sum_{u=1}^{y_{il}} (e_{iu} - s_{iu}),\ 1 \leq l \leq d_i \tag{3.3}$$

3.5.2 Characterizing α flows

ESnet NetFlow data was collected from an ESnet provider edge router for seven months: May 1 - Nov. 30, 2011 (214 days). In ESnet routers, NetFlow was configured to sample 1-in-1000 packets, and the *active timeout interval* was set to 60 seconds. Flow information was collected for the incoming side of all inter-domain interfaces. Flow tools [58], and custom Perl and R [59] programs are used to analyze the data.

The following parameter values were used: τ, NetFlow active timeout interval, was 1 minute, aggregation interval for creating prefix flows was 1 day, and H, the α NetFlow report threshold, was 1 GB. A new symbol I is used to represent the period of the analyzed NetFlow data, which was 214 days. Two types of prefix identifiers were used: (i) /32 source and destination IP addresses, and (ii) /24 source and destination subnet IDs. The aging parameter used to delete entries from the firewall filter was varied to study its impact.

As explained in Section 3.5.1, sets $\mathbf{R}_i$, $1 \leq i \leq I$, were created from the daily set of NetFlow reports using the α-flow criterion. From this filtered set of α NetFlow reports, per-day α prefix flow sets $\mathbf{P}_i$, $1 \leq i \leq I$, were created. The following characteristics of these sets are described:

Table 3.5: Data for individual α prefix flows (the per-day sets for the whole 214-day period are sorted by α-bytes or α-time)

	Percentiles	100%	90%	80%	70%	60%	50%
Sorted on α-bytes (GB)	/24	2603.58	44.43	16.94	8.5	5.34	3.46
	/32	2060.29	12.54	5.48	3.5	2.44	1.95
Sorted on α-time (min)	/24	544.64	21.27	8.16	4.54	2.66	1.88
	/32	544.64	6.16	2.76	1.78	1	0.99

- Data for individual α prefix flows
- A measure of persistence
- Cumulative per-day data
- Relative values of α flows when compared to all traffic

Individual α prefix flows A total of 125 unique /24 α prefix IDs and 1548 unique /32 α prefix IDs were observed in the 214-day NetFlow reports. In other words, 125 source-destination subnets and 1548 source-destination hosts generated α flows in the observed 214-day period. Not all α prefix IDs made an appearance every day. A matrix consisting of α prefix IDs as rows and the 214 days as columns was sparse, where each matrix entry consists of two tuples: {α-bytes, α-time}.

The data (α-bytes and α-time) corresponding to α prefix flows $\mathbf{p}_{il}$, $1 \leq l \leq d_i$ and $1 \leq i \leq I$, for 6 different percentile values are presented in Table 3.5. For example, $\max_{1\leq i\leq I}(\max_{1\leq l\leq d_i} \eta_{il})$ for the /24 α prefix IDs is 2.6 TB. In the other days, within that 214-day period, α flows corresponding to one α prefix ID transferred 2.6 TB within α-intervals in one day. The longest α-time from among all prefix flows (/24 or /32) observed in the 214-day period was 544.64 min (i.e., 9.08 hours of α intervals within one 24-hour period). This is significant because during α intervals, packets from real-time audio-video flows could have suffered increased delays. These 100 percentile values are not typical. As seen in Table 3.5, 90% of α prefix flows have a much smaller α-time, i.e., 21.27 mins for /24 sets and 6.16 mins for /32 sets.

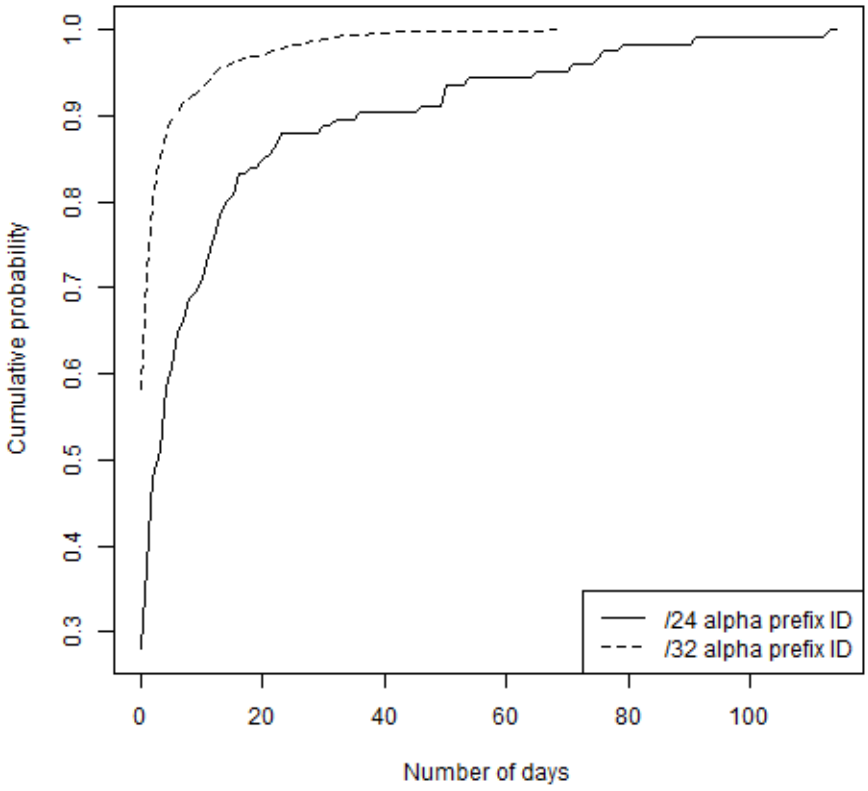


Figure 3.5: Cumulative probability of the number of days, $N_u, u \in \mathbf{U}$, in which a unique α prefix ID u made an appearance

A measure of persistence The number of days in which an α prefix ID makes an appearance is characterized as a measure of persistence. Let $\mathbf{U}$ represent the set of unique α prefix identifiers that appeared in the 214-day observation period (determined from the sets $\{\zeta_{il}, 1 \leq l \leq d_i \text{ and } 1 \leq i \leq I\}$). For each $u \in \mathbf{U}$, N_u is defined as the number of days in which $\{\eta_{il} > 0 \text{ for } \zeta_{il} = u, 1 \leq l \leq d_i \text{ and } 1 \leq i \leq I\}$.

Cumulative probability plots of N_u for the /24 and /32 cases are shown in Fig. 3.5. The maximum number of days (out of 214) in which a /24 α prefix ID appeared was 114. The maximum number of this persistency measure for a /32 α prefix ID was 68. Of the /24 and /32 unique α prefix IDs, 21.6% and 4.5% appeared more than 15 days, respectively, and 39.2% and 10.34% appeared more than 7 days, respectively. These numbers show that some source-destination host/subnet pairs repeatedly generate α flows. This is consistent with

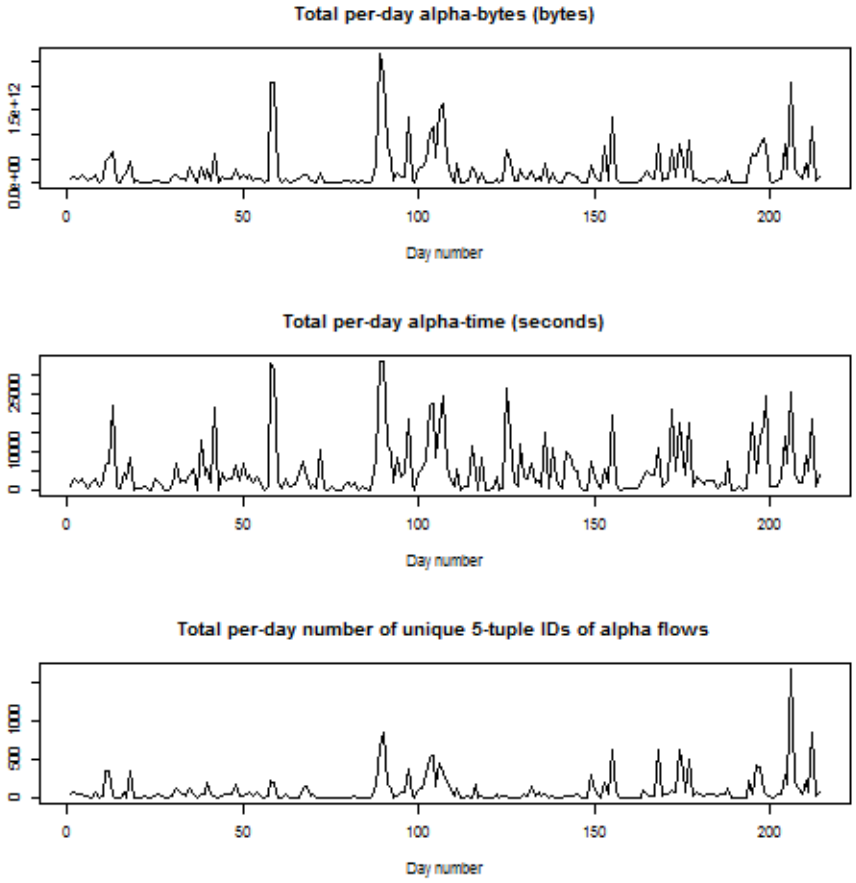
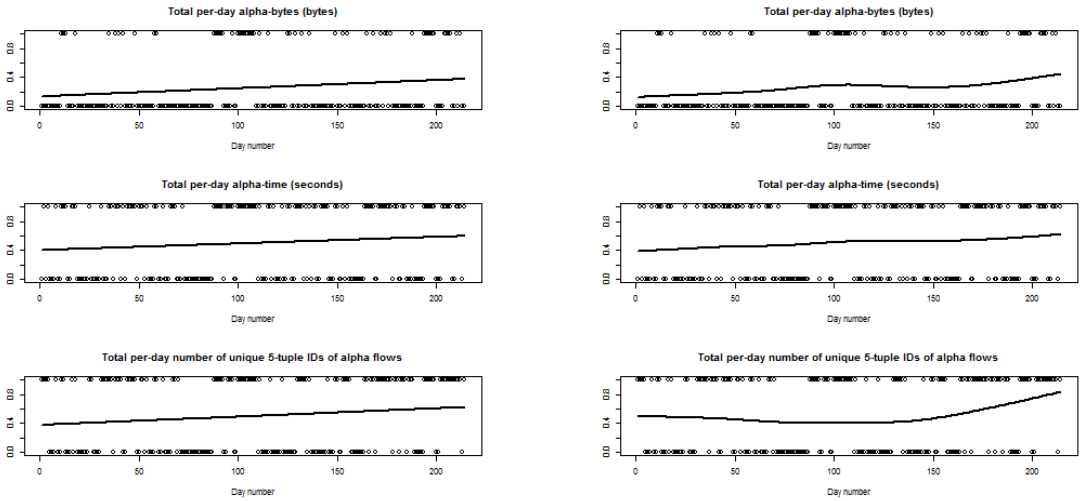


Figure 3.6: Cumulative (across all α prefix IDs) per-day data

our second hypothesis.

Cumulative per-day data The total α-bytes ($\sum_{l=0}^{l=d_i} \eta_{il}$), total α-time ($\sum_{l=0}^{l=d_i} \mu_{il}$), and total number of unique 5-tuple IDs among the α flows, for each day i, for $1 \leq i \leq I$, are plotted in Fig. 3.6. Both the total α-bytes and total α-time peaked on Jul. 28, 2011 (day number 89). On this day, there were 2.65 TB transferred in α-intervals, and 9.28 hours of α-time. It was noted earlier (Table 3.5) that most α prefix flows have small α-times, i.e., 90% of the /24 sets and /32 sets have per-day numbers less than 21.27 mins and 6.16 mins, respectively. However, the cumulative data analysis illustrates that in 10% of the days, the total amount of α-time (i.e., the sum of all unique α-intervals) was more than 4.1 hours, which means during these intervals, α flows could cause increases in the delays experienced by real-time audio-video flow packets (this particular router has only one outgoing link to another ESnet

router on which all these α flows are being carried). The total number of unique 5-tuple IDs among all α flows peaked at 1662 on Nov. 22, 2011 (day number 206).



(a) df = 2 (b) df = 4

Figure 3.7: Binary quantized cumulative data plus a smoothing spline function

In the plots of Fig. 3.6, one can observe increasing trends in α-flow activity over the 7-month period. For example, it appears that there were more days with α flows in the later months of the observed time period than in earlier months. In order to study this trend for the three measures plotted in Fig. 3.6, a coarse binary quantization was applied before fitting the points with a smoothing spline function. Specifically, the median value for each of the three measures was chosen as the threshold for a 0/1 quantization of the observed values, and two different values were used for the number of degrees of freedom (df), 2 and 4, in the smoothing spline function. Lower values of df creates greater smoothing at a cost of accuracy of fit. The increasing trends seen in the smoothed spline plots of Fig. 3.7 are likely due to increasing sizes of scientific data sets, and frequency of transfers.

Relative values of α flows when compared to all traffic Table 3.6 shows the portions of the total traffic constituted by α-bytes on a monthly basis for the observed time period. The total traffic was determined from SNMP link usage data collected by ESnet [35]. The percentages for all seven months were less than 2%. Prior work cite the 50-20 rule [60], in which 20% of the flows contribute 50% of the packets. As noted in Section 3.3, the criterion

used for classifying a flow report as belonging to an α flow did not rely on a volume threshold such as the 50% number used by Lan and Heidemann [5]. Instead the threshold was fixed at 1 GB in a time period less than equal to 1 min. This threshold was high enough that the byte ratio is small at 2%. An analysis of actual GridFTP usage transfer statistics for two source-destination pairs [18] shows that the percentage of scientific transfers that would create flows exceeding this threshold is small. This high threshold was selected because of the light link loads observed with SNMP data, which indicates that unless a sizeable burst occurs in a short duration the flow is not likely to cause adverse effects on real-time flows, and hence should not be labeled an α flow. Even though the percentage of α bytes was small, the high-rate/large size of α flows can have adverse effects as illustrated in Section 3.2.

Table 3.6: Percentage of α-bytes in total traffic for /24 case

Month (2011)	May	Jun	Jul	Aug	Sep	Oct	Nov
α-bytes (TB)	4.2	7.8	7.9	10.9	5.2	6.9	10.3
Total (TB)	625.9	533.7	692.3	640.6	740.6	1101.8	869.9
Percentage	0.67%	1.46%	1.14%	1.7%	0.7%	0.63%	1.18%

3.6 AFTES evaluation

This section evaluates the AFTES design by testing the remaining hypotheses of Section 3.4. *First*, we computed the percentage of α-bytes that would have been redirected to virtual circuit had AFTES been deployed on ESnet during the May-Nov. 2011 period. While this percentage, termed effectiveness, was higher when /24 subnet IDs were used in firewall filters than when /32 complete host addresses were used (91% vs. 82%), there were a cost to using /24 prefix IDs in that β-flow (non-α) packets that shared the same prefix IDs as α flows would also get directed to the traffic-engineered paths/queues set up for α flows. This analysis is presented in Section 3.6.1. In a *second* analysis, this cost was quantified, using a measure of the percentage of β-flow packets that were attributable to file transfer applications (the assumption being that α-flow bursts were less adverse to file-transfer flows than to real-time delay-sensitive flows). Section 3.6.2 presents this analysis. *Finally*,

the hypothesis that α flows primarily originate from "data doors," well-equipped nodes dedicated for large high-speed data transfers, was tested and found to be true, as presented in Section 3.6.3.

3.6.1 Effectiveness of AFTES

This analysis measures the effectiveness of the offline approach proposed for AFTES. *Effectiveness* is defined as the percentage of α-bytes that would have been redirected to traffic-engineered paths and isolated from the general traffic had AFTES been deployed.

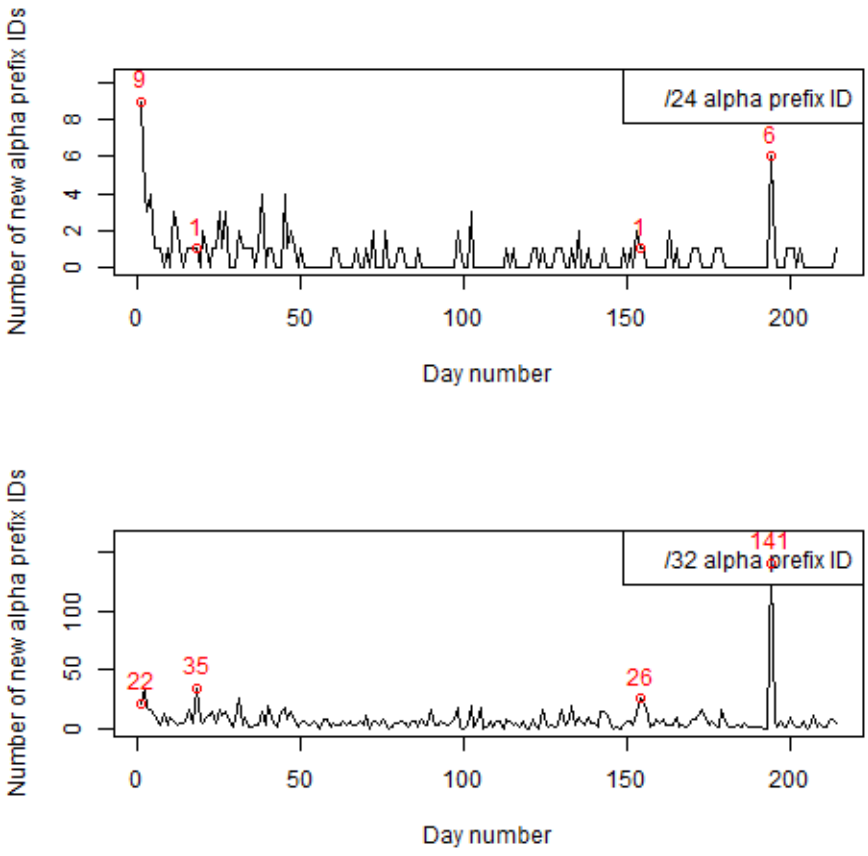


Figure 3.8: Number of new α prefix IDs per day

But before presenting our effectiveness results, the number of **new** α prefix IDs that appeared each day is plotted. Fig. 3.8 shows the number of new α-flow associated prefix identifiers appearing each day in the 214-day observed period. The aging parameter A was set to 214 days (i.e., firewall filter rules corresponding to α flows were never deleted).

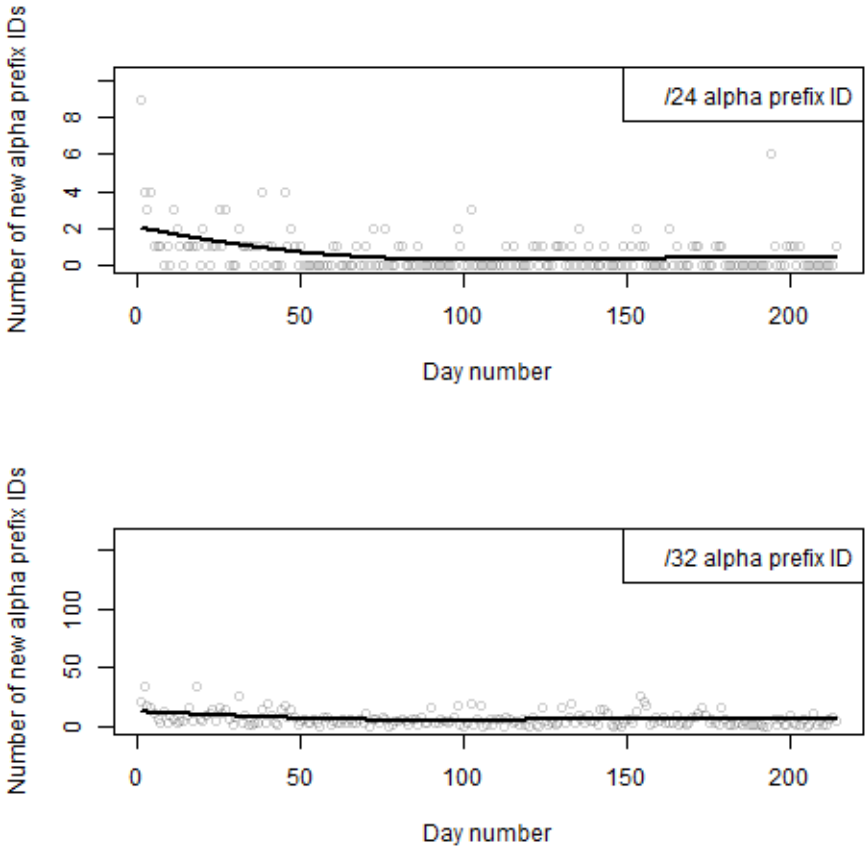


Figure 3.9: Number of new α prefix IDs per day with smoothing spline function (df=4)

The numbers would be greater if the aging parameter was set to a smaller time period to prevent the firewall filter from growing too large. The dependence on the aging parameter is presented later. Overall, the trend in the new α prefix IDs graph is downward as seen with the smoothing spline function (degrees of freedom set to 4) in Fig. 3.9. For example, on day 1, there were nine /24 new α prefix IDs but after day 45, in 94.7% of the days, there were only 0 or 1 new α prefix IDs. This confirms our third hypothesis that the number of new α prefix IDs is small. The implication is that the size of the firewall filter needed to support traffic engineering for α flows may not be significant, since modern routers allow for very large numbers of firewall filter rules. However, there can be days, such as Nov. 10th, 2011, when α flows were observed corresponding to 6 new /24 prefix IDs, and 141 new /32 prefix IDs. This can happen when there are new installations of high-speed data transfer nodes or when new scientists access existing data transfer nodes.

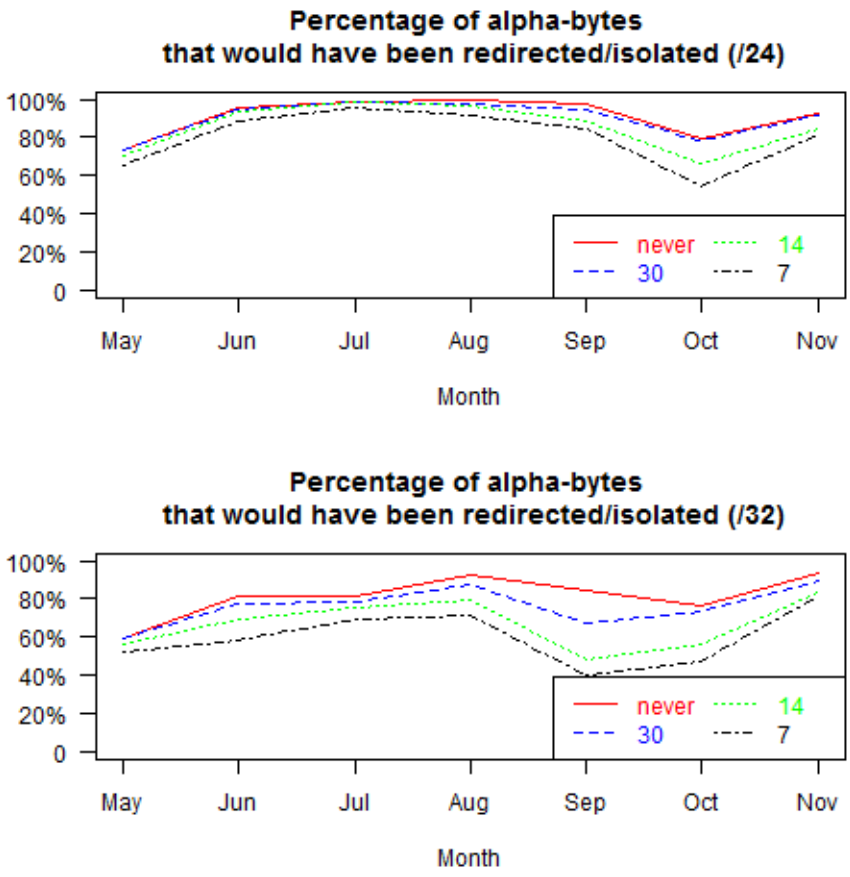


Figure 3.10: Effectiveness of AFTES for different values of the aging parameter

Fig. 3.10 shows the *effectiveness* of AFTES; specifically it shows the percentage of α-bytes that would have been redirected and isolated, on a per-month basis.

Table 3.7 shows the aggregate percentage of α-bytes that would have been directed across the whole observed time period of 214 days corresponding to different values of the aging parameter, A. Usage of /24 prefix IDs was more effective than the /32 prefix IDs (the negative aspect of this finding is quantified in the next section). This is to be expected since data transfer nodes are often cluster computers with IP addresses in the same subnet. With new installations of high-speed data transfer nodes, previously unseen /32 prefix identifiers would have been covered by /24 identifiers. If rules are never deleted from the firewall filter, for the /24 case, 92% of α-bytes would have been isolated from β flows across the 7 months.

The negative of never deleting firewall filter rules is that the firewall filter keeps growing as shown in Fig. 3.11, though the absolute size was small relative to what modern routers

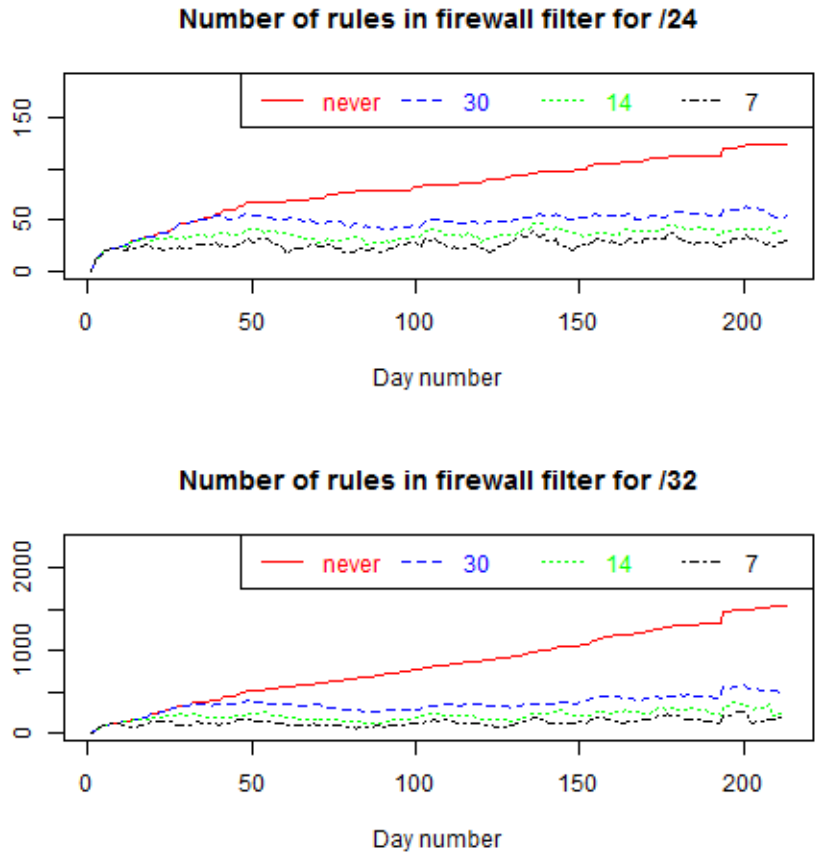


Figure 3.11: Growth of firewall filter for different values of the aging parameter

can support. But for sustainability, firewall filter rules corresponding to prefix IDs with no α flow appearances should be deleted. The use of an aging parameter of 30 days appears to be a good compromise. The firewall filter size levels off as seen in Fig. 3.11, while at the same time, the effectiveness measure does not decrease significantly. The aggregate percentage across 7 months decreases from 92% to 91% for the /24 case, when the aging parameter was dropped from 214 to 30 days, as seen in Table 3.7. This confirms our fourth hypothesis that the firewall rules is manageable from an operational point of view.

3.6.2 Quantifying the cost of AFTES on β flows

The cost of using /24 source and destination subnet identifiers rather than the more constrained /32 host identifiers in firewall filters is that β flows that share the same source/destination address prefixes as α flows will also get redirected to the traffic-engineered

Table 3.7: Percentage of α-bytes that would have been redirected and isolated across whole 214-day period

Aging parameter (days)	/24	/32
7	82%	67%
14	87%	73%
30	91%	82%
214	92%	86%

paths/queues meant to handle α flows. Packets from these β flows could suffer increased delays as they would be queued in the same buffers as those used for α flows (see Section 3.2). The purpose of this analysis was to characterize these unfortunate β flows. The term "hapless β flows" is used for these flows that share α prefix identifiers, and get traffic engineered on to the same paths and queues as α flows.

The method used was to start with the set of non-α NetFlow reports (flow reports in which the size threshold H is not exceeded) for each aggregation interval i, and then apply *three filters* **in sequence**. *First*, non-α NetFlow reports corresponding to α flows were identified. As noted in Section 3.4 and per the definitions in Table 3.3, a 5-tuple flow is classified as an α flow even if it has just one α NetFlow report. Other NetFlow reports corresponding to this 5-tuple flow may have sizes that are below the H threshold. Therefore, the first step was to identify the subset of non-α NetFlow reports that belong to α flows. The remaining non-α NetFlow reports were from hapless β flows.

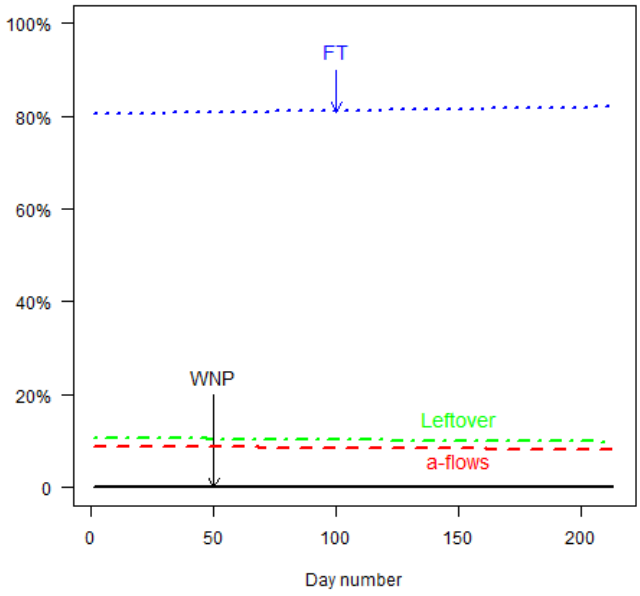
The *second* filter was used to identify NetFlow reports from file transfer applications with the assumption that the delay impact on such flows caused by bursts from α flows is not as critical as on real-time flows. The criteria used for a NetFlow report to qualify as being from a file transfer application were that (a) the number of bytes/packet should be at least 1000, (b) the total byte count should be greater than a threshold G, where $G < H$, and (c) there is at least one other NetFlow report within the aggregation interval that shares the same source and destination IP addresses, protocol type, and at least one of the two port numbers as the candidate NetFlow report. Typically NetFlow reports from file transfer applications will have multiple packets in spite of the low sampling rate (e.g., 1-in-1000), and most of these packets will be maximum-sized packets (Ethernet's Maximum Transmission

Unit size is 1500 B). The second criterion ensures that the flow corresponding to the flow report was generating a sizeable amount of data. The last criterion was applied because parallel TCP streams are used by scientific data transfer applications such as GridFTP. For example, 8-stream transfers were common [18]. Flows corresponding to these 8 streams typically shared the source and destination IP addresses, protocol type (TCP), and either the source port number or destination port number, with the other port number being different for each of the 8 streams. Therefore, due to the low sampling rate, even if there was just one NetFlow report from one stream, it is likely that there were NetFlow reports from the other streams.

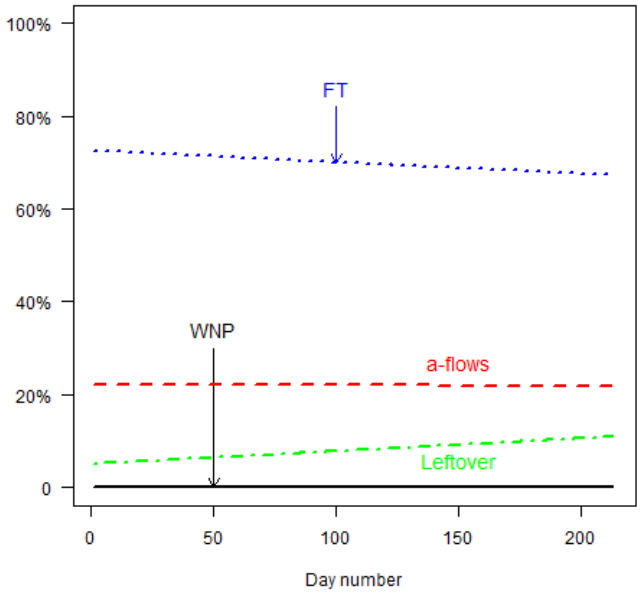
The *third* filter isolated NetFlow reports for flows with well-known port numbers; specifically ssh, http, imap, smtp, ssmtp, http, https, nntp, imap, imaps, imap4ssl, unidata, rtsp, rsync, sftp, bftp, ftps, pop3, and sslpop. Some file transfers used scp, which would appear as ssh flows, but such flows would have been filtered out into the first or second categories leaving behind only those ssh flows corresponding to interactive applications such as SecureCRT because of the sequential application of these filters.

After the sequential application of the three filters on the set of non-α NetFlow reports for each aggregation interval i, the remaining (unclassified) NetFlow reports were grouped together as "Leftover." The set of non-α NetFlow reports corresponding to hapless β flows were thus divided into three groups: *File Transfers (FT)*, *Well-Known Ports (WNP)*, and *Leftover*.

Our analysis focused on *packets* instead of bytes for these three groups of hapless β-flow Netflow reports. This is because for smaller sized flows, the size accuracy ratio of multiplying the byte count by 1000 (because of the 1-in-1000 packet sampling) will be lower than for large-sized flows [14]. Specifically, for the sampled set of packets from the hapless β flows, the percentages represented by packets from each of the three groups were computed. Across the whole 214-day period, most of the hapless β flow packets belonged to the first group (file transfers). For the /24 case, 89.37% of the hapless β-flow packets were classified as being from file transfer flows, and for the /32 case, the percentage was 88.77% with the G threshold set to 10 MB (recall the H threshold was 1GB). Only a small percentage of the hapless β-flow packets belonged to the second group, i.e., from non-file transfer flows

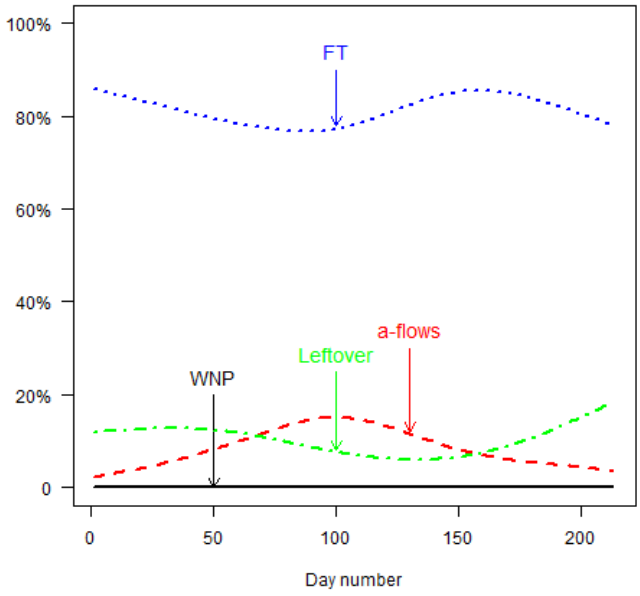


(a) /24 prefix ID case

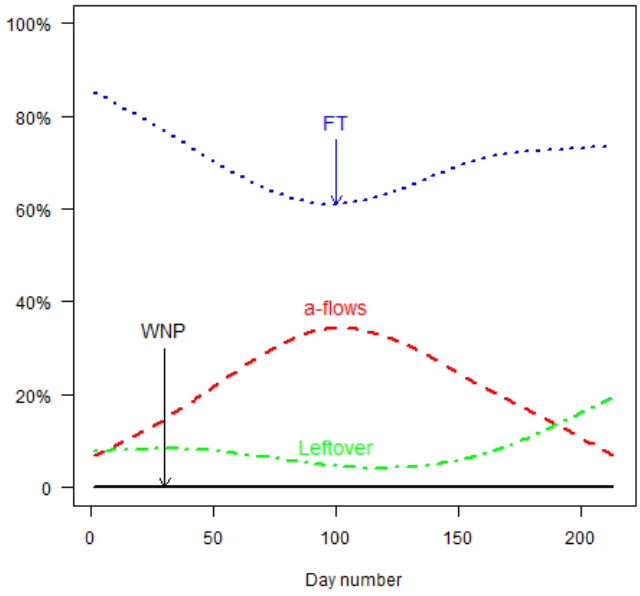


(b) /32 prefix ID case

Figure 3.12: Percentages of four groups of hapless β flow packets with smoothing spline function (df=2)



(a) /24 prefix ID case



(b) /32 prefix ID case

Figure 3.13: Percentages of four groups of hapless β flow packets with smoothing spline function (df=4)

with well-known port numbers: for the /24 this number was 0.026% for both the /24 and /32 cases. The third group (leftover) percentage was small (e.g., 10.6% for the /24 case). These numbers confirm our fifth hypothesis that the number of β flows that share the same source-destination subnets with α flows and are adversely affected is small.

The per-day percentages for these three groups of hapless β flow packets are plotted using smoothing spline functions (degrees of freedom set to 2 and 4), in Figs. 3.12 and 3.13. For both /24 and /32 cases, the WNP group percentages were almost 0.

In *summary*, the above analysis shows that most β flows that shared α prefix identifiers were from file transfer applications, and not from interactive applications such as VoIP and Web browsing. This is likely because in most supercomputing centers, high-end data transfer nodes are set up as clusters in their own subnets. Therefore the cost of using prefix IDs instead of 5-tuple flow IDs for traffic engineering α flows appears to be low.

Furthermore, we recommend the use of /24 address prefixes rather than /32 addresses because the effectiveness measure was higher for the former, and the negative effect on β flows was small. It is better to have a small percentage of β flows be impacted by α flows (which will happen at a higher rate with /24 prefixes) than to miss α flows, which will happen at a higher rate with /32 addresses, and have those α flows impact the much larger proportion of β flows on the default IP-routed paths.

3.6.3 Data-door analysis

Our last hypothesis, presented in Section 3.4, was that α flows are created by large file transfers from well-equipped nodes that are referred to as "data doors." In order to test this hypothesis, the domain names/IP addresses of data doors in major high performance computing facilities (e.g., rftpexp.rhic.bnl.gov) were obtained from their corresponding Web sites, and these IP addresses were matched against the α prefix identifiers found through the above described NetFlow data analysis. Our findings support the hypothesis.

Absolute and relative measures of the daily number of α prefix IDs matched with data door address prefixes are shown in Fig. 3.14. In 96 (44.9%) days out of 214, all α prefix identifiers matched with those of data doors, and in 175 (81.8%) days out of the 214 days, the matched rate was more than 80%.

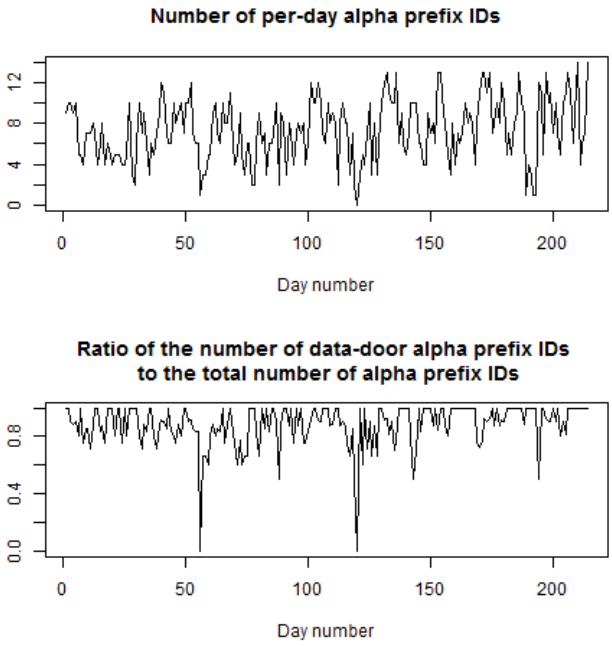


Figure 3.14: A measure of the ratio of α prefix IDs from/to data doors

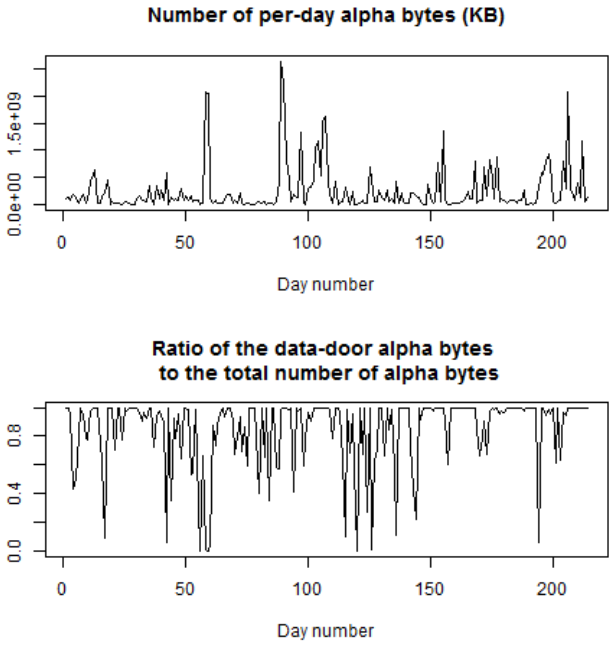


Figure 3.15: A measure of the ratio of α-bytes from/to data doors

Absolute and relative measures of the per-day α-bytes from/to data doors are shown in Fig. 3.15. In 148 (69.2%) days out of 214, more than 90% of the α-bytes were from/to data

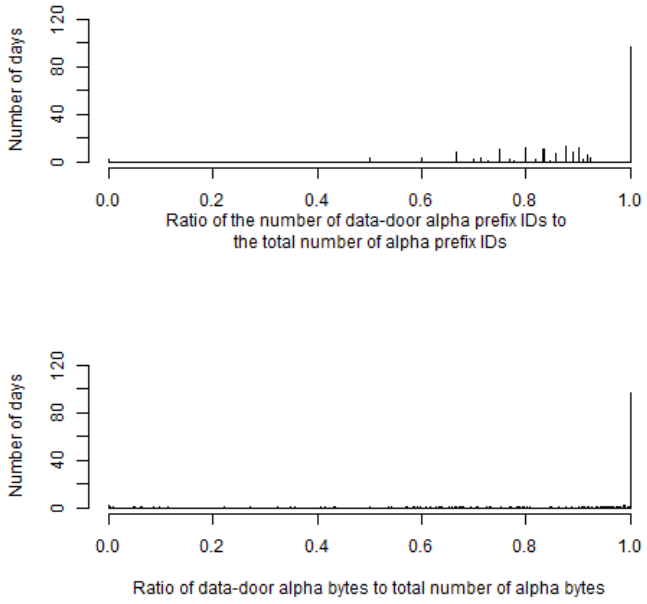


Figure 3.16: Histograms of the two data-door ratio measures

doors.

Histograms of the two ratios presented above are shown in Fig. 3.16. The high probability mass at a ratio of 1 indicates that for most days, the α flows came from data doors. In *summary*, it appears that indeed most α prefix flows were from/to high-end servers that are designed specifically for handling data transfers.

3.7 Conclusions

In this chapter, an offline mechanism was presented for determining prefix identifiers of α flows, which are caused by high-rate, large-sized data transfers that are typically created in research-and-education networks by scientific researchers. Such an offline scheme is developed for a network management system called Alpha Flow Traffic Engineering System (AFTES) for intra-domain traffic engineering. Prefix identifiers (IDs) (source and destination address prefixes) of persistent α flows are used to set firewall filters to redirect future α flows to traffic-engineered paths, and to isolate their packets to separate queues. These actions are performed to reduce adverse effects that α flows could have on packets from real-time

flows, such as increased packet delay and delay variance. The effectiveness of this offline mechanism was evaluated through an analysis of 7 months of NetFlow data obtained from an ESnet router. For this data set, 91% of bytes generated by α flows in bursts would have been directed with /24 based prefix IDs and an aging interval of 30 days. An analysis of β-flow packets that share α-flow prefix IDs showed that close to 90% of these packets were from file-transfer applications, which are not as affected by α flows as packets from real-time applications.

Chapter 4

Quality of Service (QoS) provisioning for α flows

4.1 Introduction

In Chapter 3, we presented an overall architecture for an intra-domain traffic engineering system called Alpha Flow Traffic Engineering System (AFTES) that performs two tasks: (i) analyzes NetFlow reports offline to identify α flows, and (ii) configures the ingress routers for future α-flow redirection to traffic-engineered Quality-of-Service (QoS)-controlled paths. Chapter 3 focused on the first aspect, and this chapter focuses on the second aspect of AFTES by addressing the question of how to achieve α-flow redirection and isolation to traffic-engineered paths.

As introduced in Section 2.2, The Inter-Domain Controller (IDC) requires an application to specify the circuit rate, duration, start time, and the endpoints in its advance-reservation request. The specified rate is used both for (i) path computation in the call-admission/circuit-scheduling phase and (ii) policing traffic in the data plane. If the application requests a high rate for the circuit, the request could be rejected by the OSCARS IDC due to a lack of resources. On the other hand, if the request is for a relatively low rate (such as 1 Gbps), then the policing mechanism could become a limiting factor to the throughput of α flows, preventing TCP from increasing its sending rate.

The purpose of this chapter is to evaluate the effects of different scheduling and policing mechanisms to achieve two goals: (i) reduce delay and jitter of real-time sensitive flows that share the same interfaces as α flows, and (ii) achieve high throughput for α-flow transfers.

Our *key findings* are as follows:

1. With the current widely deployed best-effort IP-routed service, which uses first-come-first-serve (FCFS) packet scheduling on egress interfaces of routers, the presence of an α flow can increase the delay and jitter experienced by audio/video flows.

2. This influence can be eliminated by configuring two virtual queues at the contending interface and redirecting identified α flows to one queue (α queue), while all other flows are directed to a second queue (β queue).

3. The policer should not be configured to direct out-of-profile packets of an α TCP flow to a different queue from its in-profile packets. When packets of the same TCP flow are served from different queues, packets can arrive out of sequence at the receiver. Out-of-sequence arrivals triggers TCP's fast retransmit/fast recovery congestion algorithm, which causes the TCP sender to lower its sending rate resulting in degraded throughput.

4. An alternative approach to dealing with out-of-profile packets is to probabilistically drop a few packets using Weighted Random Early Detection (WRED), and to buffer the remaining out-of-profile packets in the same queue as the in-profile packets. This prevents the out-of-sequence problem and results in a smaller drop in α-flow throughput when compared to the separate-queues approach.

5. The no-policing scheme is preferred to the policing/WRED scheme because AFTES redirects α flows within a provider's network, which means that these flows will typically run TCP and are not rate-limited to the circuit rate. If an end application requested a circuit explicitly, then it can be expected to use traffic control mechanisms, such as Linux `tc`, to limit the sending rate. But with AFTES, the end application is not involved in the circuit setup phase, and therefore the applications are likely to be running unfettered TCP. Under these conditions, when buffer occupancy builds up, packets will be deliberately dropped in the policing/WRED scheme, leading to poor

performance. Furthermore, if there are two simultaneous α flows, the probability of buffer buildups increases, which in turn increases the dropped-packet rate and lowers throughput.

6. The negatives of partitioning rate/buffer space resources between two queues were studied. Our conclusions are that close network monitoring is required to dynamically adjust the rate/buffer space split between the two queues as traffic changes, and the probability of unidentified α flows should be reduced whenever possible to avoid these flows from becoming directed to the β queue.

Section 4.2 provides background and reviews related work. Section 4.3 describes the experiments we conducted on a high-speed testbed to evaluate different combinations of QoS mechanisms and parameter values to achieve our dual goals of reduced delay/jitter for real-time flows and high throughput for α flows. Our conclusions are presented in Section 4.4.

4.2 Background and Related Work

The first two topics, Transmission Control Protocol (TCP) and QoS support in state-of-the-art routers, provide relevant background information. The last topic, QoS mechanisms applied to TCP flows, covers related work.

Transmission Control Protocol (TCP) TCP was first proposed by Cerf and Kahn in 1974 [61]. It is a reliable transport protocol that requires the receiver to send acknowledgements (ACKs) back to the sender to confirm receipts of segments (term used at the TCP layer for packets). The goal of a TCP sender is to maximize its own throughput while being fair to other TCP flows. The TCP sender starts out slowly by sending only one segment and awaiting an ACK. When the ACK arrives, it increases its sending window (also called congestion window) size by a maximum segment size (MSS), which correspondingly allows the TCP sender to send two segments. This congestion control scheme is described as Additive-Increase/Multiplicative-Decrease (AIMD), because the TCP congestion window is increased additively every round trip time (RTT), but decreased by a multiplicative factor if

congestion is detected by a packet loss. Packet loss events are triggered by the retransmission timer maintained at the sender (one at a time for each TCP connection while awaiting an ACK for a segment) times out (called an RTO). When an RTO occurs, the TCP sender drops the congestion window to 1 MSS, which means its sending rate drops to the minimum level where it sends one segments and awaits an ACK.

In most current implementations of TCP, e.g., TCP Reno and H-TCP, a fast retransmit/fast recovery algorithm is added in order to allow TCP to react faster to network congestion. When the sender receives three duplicate ACKs, the sender assumes that a packet was lost because the receiver triggers a duplicate ACK for every new packet received out of sequence. The sender interprets the reception of triple-duplicate ACKs to mean that congestion level is not that high because packets are getting through to the receiver, while that one packet for which triple-duplicate ACKs are received must have been lost (cumulative ACKs are used in TCP, which means that the requested ACK number is the sequence number of the next expected segment). The TCP sender just halves the congestion window (instead of dropping it to 1 MSS as happens with an RTO). This allows the TCP sender to ramp back up the congestion window to the point at which it is no longer waiting for ACKs, i.e., it is constantly sending segments as ACKs are streaming back in.

If TCP segments get delayed due to buffer buildups within routers, RTT increases, which means ACKs arrive back at the sender at a slower rate, which effectively lowers the TCP-segment sending rate. This should relieve buffer buildup causing RTT to drop and the rate to pick back up. This process is referred to as self-clocking in the TCP context.

QoS support in state-of-the-art routers Multiple policing, scheduling and traffic shaping mechanisms have been implemented in today's routers. We review the particular mechanisms used in ESnet, and hence in our experiments. For scheduling, two mechanisms are used: Weighted Fair Queueing (WFQ) and Priority Queueing (PQ) [62]. With WFQ, multiple traffic classes are defined, and corresponding virtual queues are created on egress interfaces. Bandwidth can be strictly partitioned or shared among the virtual queues. WFQ is combined with PQ as explained later. On the ingress-side, policing is used to ensure that a flow does not exceed its assigned rate (set by the IDC during call admission). For

example, in a single-rate two-color (token bucket) scheme, the average rate (which is the rate specified to the IDC in the circuit request) is set to equal the generation rate of tokens, and a maximum burst-size is used to limit the number of tokens in the bucket. The policer marks packets as *in-profile* or *out-of-profile*. Three different actions can be configured: (i) discard out-of-profile packets immediately, (ii) classify out-of-profile packets as belonging to a `Scavenger Service (SS)` class, and direct these packets to an `SS` virtual queue, or (iii) drop out-of-profile packets according to a WRED profile, but store remaining out-of-profile packets in the same queue as in-profile packets. For example, the drop rate for out-of-profile packets can be configured to increase linearly from 0 to 100 for corresponding levels of queue occupancy.

QoS mechanisms applied to TCP flows Many QoS provisioning algorithms that involve some form of active queue management (AQM) have been studied [63–67]. Some of the simpler algorithms have been implemented in today's routers, such as RED [63] and WRED [65], while other algorithms, such as Approximate Fair Dropping (AFD) [67], have been shown to provide better fairness. An analysis of the configuration scripts used in core and edge routers of ESnet shows that these AQM related algorithms are not enabled. This is likely due to the commonly adopted policy of overprovisioning (an Internet2 memorandum [34] states a policy of operating links at 20% occupancy). Nevertheless, providers have recognized that in spite of the headroom, an occasional α flow can spike to a significant fraction of link capacity (e.g., our GridFTP log analysis showed average flow throughput of over 4 Gbps across 10-Gbps paths [18]). When the flow throughput averaged across its lifetime is 4 Gbps, there can be short intervals in which the flow rate spiked to values close to link capacity.

4.3 Experiments

A set of experiments were designed and executed to determine the best combination of QoS mechanisms with corresponding parameter settings in order to achieve our dual goals of reduced delay/jitter for real-time traffic and high throughput for α flows. For the first goal, we formulated a hypothesis as follows: a scheduling-only no-policing scheme that isolates

α-flow packets into a separate virtual queue is sufficient to keep non-α flow delay/jitter low. For the second goal, we experimented with different QoS mechanisms and parameter settings.

Experiment 1 was designed to understand the two modes for sharing link rate (strictly partitioned and work conversing), and to determine the router buffer size. *Experiment 2* tests the above-stated hypothesis for the first goal. *Experiments 3 and 4* studied two different mechanisms, using a separate scavenger-service (SS) queue vs. using WRED, for handling the out-of-profile packets identified by ingress-side policing, and compared results with a no-policing approach. We concluded that the WRED scheme was better, but it was outperformed by the no-policing scheme. *Experiment 5* was designed to check if the policing/WRED scheme had a fairness advantage over the no-policing scheme. We found that since neither of the two policed α flows honored their assigned rates (which should be expected for HNTES-redirected flows), under the no-policing scheme the TCP flows adjusted their sending rates and had no packet losses, while the deliberate packet losses introduced in the policing/WRED scheme lowered throughput for both flows, and furthermore resulted in lower fairness because of a difference in RTTs, even though this difference was small. In *Experiment 6*, we characterized the impact of QoS provisioning under changing traffic conditions, and compared two versions of TCP: Reno [68] and H-TCP [69]. In the presence of an α flow that uses up its whole α-queue rate allocation, if the background traffic is more than the rate allocated to the β queue, the latter will suffer from more losses than if there had been no partitioning of resources between the two queues. This implies a need for closer monitoring of traffic and dynamic reconfiguration of the rate/buffer allocations to the two queues. However, since two rare events, an α flow and an increased background load, have to occur simultaneously, the probability of this scenario is low. H-TCP is better than Reno for high-speed transfers, but from the perspective of the impact on other flows, we did not see a significant difference in our tested scenarios. *Experiment 7* was designed to study the effects of an unidentified α flow being directed to the β queue. Here again, if there was no simultaneous α flow directed to the α queue when the unidentified α flow appeared, then the impact will be the same as without partitioning. However, if this combination of rare events occurs jointly, then given that the β queue has only a partition of the total interface

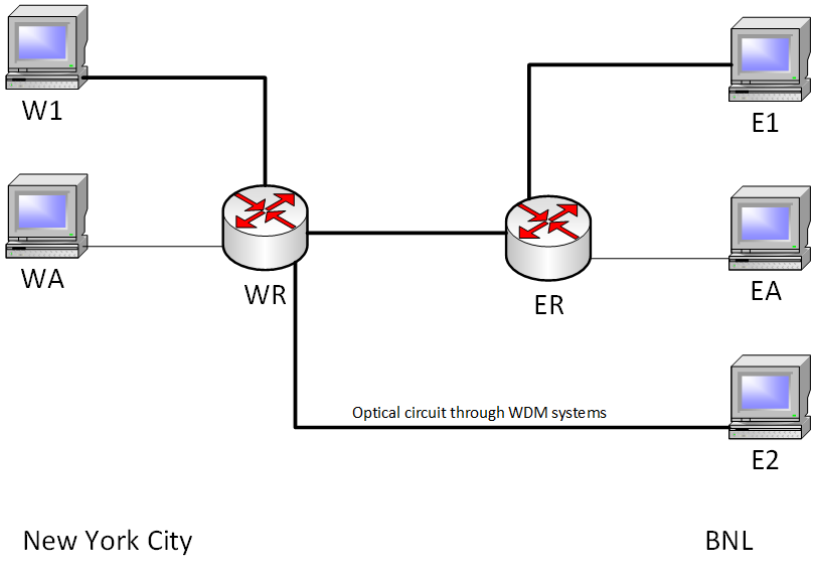


Figure 4.1: Experiment setup

rate/buffer space, the impact on delay-sensitive flows will be greater than if there had been no partitioning.

Section 4.3.1 describes the experimental setup, the experimental methodology, and certain router configurations that are common to all the experiments. The remaining subsections describe the seven experiments.

4.3.1 Experimental Setup

The experimental network setup is shown in Figure 4.1. It was called the Long Island MAN (LIMAN) testbed, and was supported by ESnet as a DOE-funded testbed for networking research. The high-performance hosts, `W1` (West 1), `E1` (East 1), and `E2` (East 2), were Intel Xeon Nehalem E5530 models (2.4GHz CPU, 24GB memory) and ran Linux version 2.6.33. The application hosts, `WA` (West App-host) and `EA` (East App-host), were Intel Dual 2.5GHz Xeon model and ran Linux 2.6.18. The routers, `WR` (West Router) and `ER` (East Router), were Juniper MX80's running Junos version 10.2. The link rates were 10 Gbps from the high-performance hosts to the routers, 1 Gbps from the application hosts to the routers, and 10 Gbps between the routers.

Host `W1` and router `WR` were physically located in New York City, while the East-side hosts and routers, and host `E2`, were physically located in the Brookhaven National Laboratory

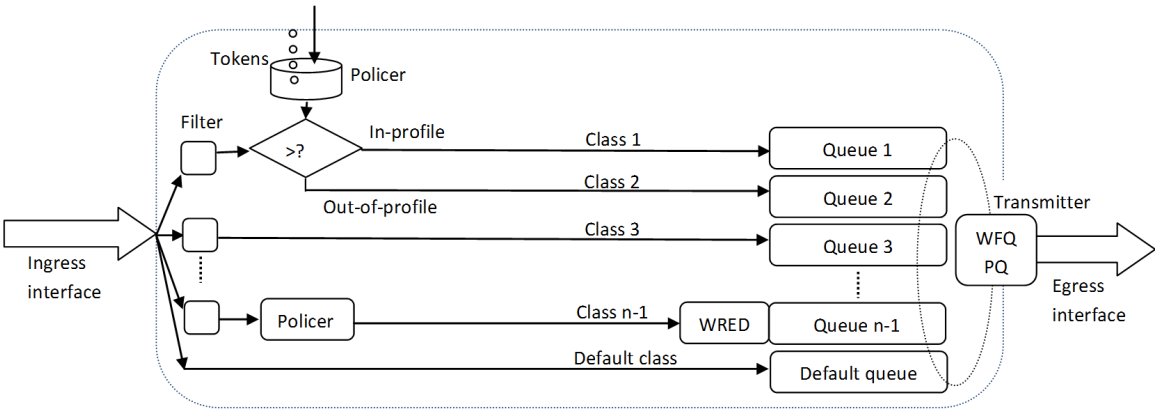


Figure 4.2: Illustration of QoS mechanisms in a router

(BNL) in Long Island, New York. Host `E2` was connected to router `WR` via a circuit provisioned across the Infinera systems of the underlying optical network as shown in Figure 4.1.

Each experiment consists of four steps: (i) plan the applications required to test a particular QoS mechanism, (ii) configure routers to execute the selected QoS mechanisms with corresponding parameter settings based on the planned application flows, (iii) execute applications on end hosts to create different types of flows through the routers, and (iv) obtain measurements for various characteristics, e.g., throughput, packet loss, and delay, from the end-host applications as well as from packet counters in the routers.

A preliminary set of experiments were conducted to determine the specific manner in which the egress-side link capacity was shared among multiple virtual queues. Theoretically the transmitter can be strictly partitioned or shared in a work-conserving manner. If strictly partitioned, then even if there are no packets waiting in one virtual queue, the transmitter will not serve packets waiting in another queue. In this mode, each queue is served at the exact fractional rate assigned to it. In contrast, in the work-conserving mode the transmitter will serve additional packets from a virtual queue that is experiencing a higher arrival rate than its assigned rate if there are no packets to serve from the other virtual queues. The buffer is always strictly partitioned between the virtual queues in the routers used in our experiments.

Figure 4.2 illustrates how a combination of QoS mechanisms was used in our experiments. *First*, incoming packets are classified into multiple classes based on pre-configured firewall filters, e.g., α-flow packets are identified by the source-destination IP address prefixes and

classified into the α class. *Second*, packets in some of these classes are directly sent to corresponding egress-side virtual queues, while flows corresponding to other classes are subject to policing. A single-rate token bucket scheme is applied. If an arriving packet finds a token in the bucket, it is marked as being in-profile; otherwise it is marked as being out-of-profile. *Third*, for some policed flows, in-profile and out-of-profile packets are sent to separate egress-side virtual queues, while packets from other policed flows are subject to WRED before being buffered in a single virtual queue. On the egress-side, each virtual queue is assigned a priority level, a fractional allocation (expressed as a percentage) of link capacity, and a fractional allocation of the buffer. As noted in the previous paragraph the buffer allocation is strictly partitioned while the transmitter is shared in work-conserving mode. *Fourth*, the WFQ scheduler decides whether a virtual "queue is in-profile or not," by comparing the rate allocated to the queue and the rate at which packets have been served out of the queue. *Finally*, the PQ scheduler selects the queue from which to serve packets using their assigned priority levels, but to avoid starvation of low-priority queues, as soon as a large enough number of packets are served from a high-priority queue to cause the status of the queue to transition to out-of-profile, the PQ scheduler switches to the next queue in the priority ordering. When all queues become out-of-profile, it starts serving packets again in priority order. It is interesting that while the *policer* is flagging *packets* as in-profile or out-of-profile on a per-flow basis, the *WFQ scheduler* is marking *queues* as being in-profile or out-of-profile.

4.3.2 Experiment 1

Purpose and execution

The goals of this experiment were to (i) determine the router buffer size, (ii) determine the default mode used in the routers for link capacity (rate) sharing (between the two options of strict partitioning and work-conserving), and (iii) compare these two modes. Correspondingly, three scenarios were tested with different router configurations. To control rate and buffer allocations, the router software required the configuration of a virtual queue on the egress interface, even if it was just a single queue to which all flows were directed. In

scenario 1, by modifying the buffer allocation for the virtual queue, router buffer size was determined. In scenario 2, by modifying the rate allocation, the default mode for capacity sharing was determined. Finally, in scenario 3, the router was explicitly configured to operate in the two different modes for comparison.

As per our execution methodology, the first step was to plan applications. For the first two scenarios, we planned to use two UDP flows created by the `nuttcp` application, and a "ping" flow to send repeated echo-request messages and receive responses. The purpose of the ping flow was to measure round-trip delays, and the UDP flows were used to fill up the router buffer. Only one UDP flow was required for the third scenario. Hosts `W1` and `E2` were used to generate the two UDP flows, both of which were destined to host `E1`. Different hosts were used to achieve high transfer rates. The ping flow sent messages from host `WA` to host `EA`. Therefore, contention for buffer and bandwidth resources occurrred on the link from router `WR` to router `ER`.

Our next step was to configure the routers. A single virtual queue was configured on the output interface from `WR` to `ER`, and all application flows were directed to this queue. In scenario 1, the whole link capacity was assigned to the virtual queue, but the buffer allocation was changed from 20% to 100%. In scenario 2, the assigned rate was varied from 1% to 100%, while the buffer allocation was set to 100%, and in scenario 3, the rate and buffer allocations were set to 20%, and the capacity sharing mode was explicitly configured.

Next, we executed the experiments corresponding to the scenarios. For the first two scenarios, each `nuttcp` application was initiated with the sending rate set to 7 Gbps, resulting in a total incoming rate of 14 Gbps in order to fill up the buffer of the 10 Gbps `WR`-to-`ER` interface. Due to the resulting packet losses, `nuttcp` at the receiving host `E1` reported rates of approximately 5 Gbps for each UDP flow. In scenario 3, the sending rate of the single UDP flow was set to 3 Gbps. This was sufficient given the 20% rate allocation to the configured virtual queue on the `WR`-to-`ER` link in this scenario. In all three scenarios, the UDP flows and ping flow were run for 60 seconds.

Finally, for the first and third scenarios, round-trip time (delay) measurements were obtained from the ping application on the `WA` host. For the second scenario, router counters for outgoing packets on the `WR`-to-`ER` link were read in order to find the number of packets

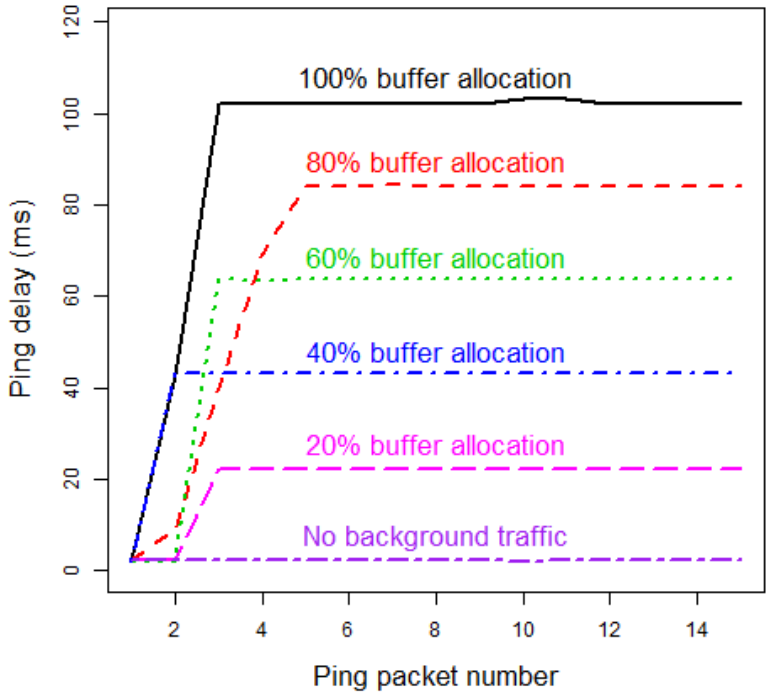


Figure 4.3: Experiment 1 scenario 1 results: Ping delay for different buffer allocations (rate allocation was 100%)

transmitted within 60 seconds under different rate allocations.

Results and discussion

Router buffer size The ping packet delay measured in scenario 1 is plotted against the ping packet number, which is effectively the same as time, in Figure 4.3. With increasing time, the ping delay increases gradually because the `nuttcp` UDP packets start filling the buffer partition allocated to the virtual queue on the `WR`-to-`ER` interface. The minimum ping delay (2.1 ms) was observed when there were no UDP flows, i.e., there was no background traffic. The maximum delay (102 ms) was observed when the buffer allocation for the virtual queue was 100%.

In the various plots of Figure 4.3, the buffer allocations for the virtual queue are indicated. When the buffer allocation was limited to 20%, the delay was only 22.2 ms, while when the buffer allocation was set to 100%, the ping delay was higher because the whole buffer had

Table 4.1: Experiment 1 scenario 2: packet counter values observed at router `WR` for its `WR`-to-`ER` interface

Link rate allocation	1%	20%	40%	60%	80%	100%
Number of packets transmitted on the `WR`-to-`ER` link	51924370	51536097	52755553	52669911	52786301	52637553

filled up. Recall that the aggregate arrival rate of packets destined for the `WR`-to-`ER` link was 14 Gbps, while the outgoing link rate was only 10 Gbps.

Based on these observations, the buffer size for the `WR`-to-`ER` egress interface can be computed as follows:

$$10\ Gbps \times (102 - 2.1)\ ms = 125\ MB \tag{4.1}$$

Default mode for link capacity sharing From the experiments conducted in Scenario 2, the router counters for the `WR`-to-`ER` were recorded, and are shown in Table 4.1. The reported packets were almost the same for all values of the link capacity allocation. Recall that the buffer allocation was set to 100% for this scenario. In other words, even if only a 1% rate was assigned to the virtual queue in which packets from all three flows were held, the virtual queue was served at 100% capacity. This result verifies that the default mode of operation for the tested router is the work-conserving mode.

Comparison of the two rate sharing modes Two rate sharing strategies, strictly partitioned and work conserving, were compared in Scenario 3. Figure 4.4 shows the ping delay results under these two configurations. In the strictly partitioned configuration, ping delays built up to 102 ms. Recall that for scenario 3, the virtual queue rate and buffer allocations were set to 20%, which was confirmed as follows:

$$R = \frac{125\ MB \times 0.2}{(102 - 2.1)\ ms} = 2\ Gbps \tag{4.2}$$

Under the work-conserving configuration, ping delay was only 2.1 ms (the round-trip time with no background traffic). Recall that the UDP flow sending rate was 3 Gbps in this

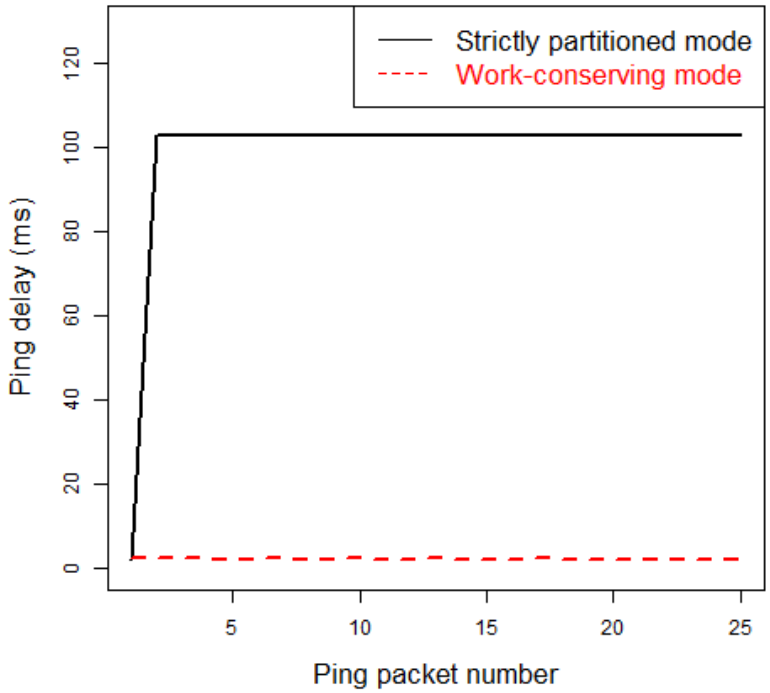


Figure 4.4: Experiment 1 scenario 3: Results comparing the two rate sharing modes (rate and buffer allocation was 20%)

scenario, while the rate allocation was only 2 Gbps. Yet there was no queue buildup in the buffer, which means the egress interface was served at a rate greater than 3 Gbps. Thus, in the work-conserving mode, virtual queues that have packets are served with excess capacity, if any.

4.3.3 Experiment 2

Purpose and execution

The goals of this experiment were to (i) determine whether α flows have adverse effects on real-time flows, and (ii) determine whether a scheduling-only no-policing solution of α-flow isolation to a separate virtual queue is sufficient to meet the first goal of keeping non-α flow delay/jitter low.

The first step was to plan a set of applications. We decided to use four `nuttcp` TCP flows and a ping flow. The TCP version used was H-TCP [69] because it is the recommended

option to create high-speed (α) flows [70]. Two of the TCP flows carried data from host `E2` toward host `W1`, while the other two TCP flows were from `E1` to `W1`. The ping flow was from `EA` to `W1`. Therefore, in this experiment, contention for buffer and bandwidth resources occurred on the link from router `WR` to host `W1`. Although the high-performance host `W1` was the common receiver for all flows, there was no contention for CPU resources at `W1` because the operating system automatically scheduled the five receiving processes to different cores.

The second step was to configure the routers. For comparison purposes, this experiment required two configurations: (i) `1-queue`: a single virtual queue was defined on the egress interface from `WR` to `W1`, and all flows were directed to this queue, and (ii) `2-queues`: two virtual queues (α queue and β queue) were configured on the egress interface from `WR` to `W1`, and WFQ scheduling was enabled with the following rate (and buffer) allocations: 95% for α queue and 5% for β queue. The priority levels of the α and β virtual queues were set to medium-high and medium-low, respectively. In the 2-queues configuration, two additional steps were required. A firewall filter was created in router `WR` to identify packets from TCP (α) flows using their source and destination IP addresses. A class-of-service configuration command was used to classify these packets as belonging to the α class and to direct packets from these flows to the α queue on the egress interface from `WR` to `W1`. By default, all other packets were directed to the β queue, which means that packets from the ping flow were sent to the β queue.

In the third step, the applications were executed as follows. The four TCP flow execution intervals were: (0, 200), (20, 160), (40, 140), and (60, 120), respectively, while the ping flow was executed from 0 to 200 seconds.

Finally, throughput measurements as reported by each `nuttcp` sender were collected, as were the delays reported by the ping application.

Results and discussion

The top graph in Figure 4.5 illustrates that the scheduling-only no-policing solution of configuring two virtual queues on the shared egress interface and separating out the α flows into their own virtual queue leads to reduced packet delay/jitter for the β flow. In the 1-queue configuration, the mean ping delay was 60.4 ms, and the standard deviation was

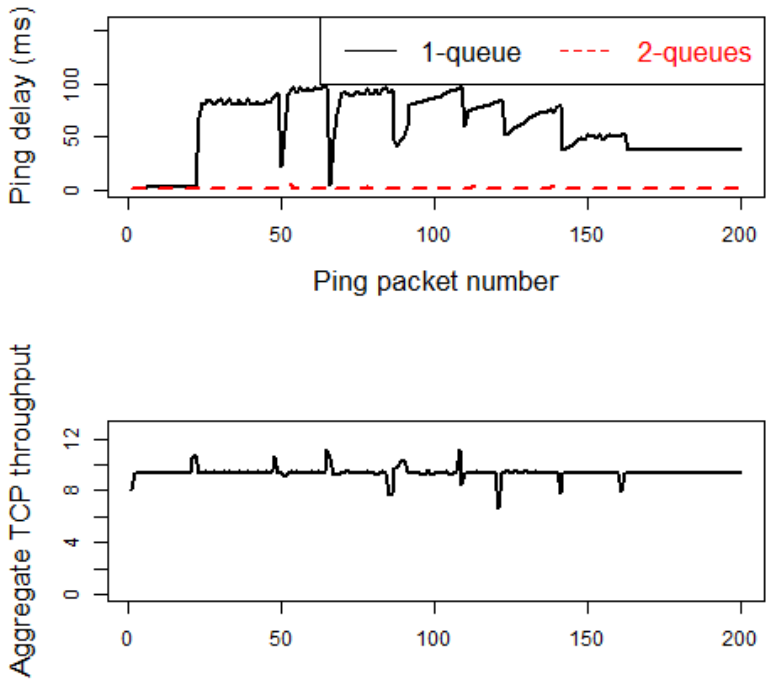


Figure 4.5: Experiment 2: Top graph shows the delays experienced in the ping flow under 1-queue and 2-queues configurations; bottom graph shows the aggregate TCP flow throughput

29.3 ms, while in the 2-queues configuration, the mean ping delay was only 2.3 ms, with a standard deviation of 0.3 ms.

In the 2-queues case, since the rate of the ping flow was much lower than the 5% allocated rate for the β queue, the β queue was in-profile, and hence the ping-application packets were served immediately without incurring any queueing delays.

The bottom graph in Figure 4.5 shows the aggregate throughput of the four TCP flows. A comparison of this throughput graph with the top ping-delay graph shows the following: (i) when the aggregate TCP throughput increased from 9.4 Gbps to 10.7 Gbps at time 22, and the ping delay increased from 3 ms to 82 ms. The `nuttcp` application reports average throughput on a per-sec basis. Therefore, while the total instantaneous throughput cannot exceed 10 Gbps (link rate), the sum of the per-sec average throughput values for the four TCP flows sometimes exceeds 10 Gbps, (ii) when the aggregate TCP throughput dropped from 10.6 Gbps to 9.3 Gbps at time 49, the ping delay dropped from 92 ms to 22

ms, correspondingly, and (ii) throughput drops at 85, 121, 141, and 161 sec coincided with ping-delay drops.

4.3.4 Experiment 3

Purpose and execution

The goals of this experiment were to (i) compare a 2-queues configuration (scheduling-only, no-policing) with a 3-queues configuration (scheduling and policing), and (ii) compare multiple 3-queues configurations with different parameter settings.

As per our execution methodology, the first step was to plan applications. To study the behavior of the QoS mechanisms, one `nuttcp` TCP flow and one `nuttcp` UDP flow (background traffic) were planned. The UDP flow carried data from host E2 toward host W1, while the TCP flow was from E1 to W1. Contention for buffer and bandwidth resources occurred on the link from router WR to host W1.

In the second step, the router `WR` was configured with the following QoS mechanisms. The 2-queues configuration was the same as in experiment 2 (no-policing), except that both queues were given equal weight in sharing the rate and buffer (50% each). For the 3-queues configurations, the allocations for the three queues (α, β, and SS) to which in-profile TCP-flow packets, UDP and ping packets, and out-of-profile TCP-flow packets, were directed, respectively, are shown in Table 4.2. The priority levels of these three virtual queues were medium-high, medium-low, and low respectively. The policer was configured to direct in-profile TCP-flow packets ($\leq$ 1 Gbps and burst-size $\leq$ 31 KB) to the α queue, and out-of-profile packets to the SS queue.

In the third step, experiment execution, the UDP flow rate was varied from 0 Gbps to 3 Gbps in a particular on-off pattern as shown in the top graph of Figure 4.6, and the TCP flow was executed for the whole 200 sec. Finally, the same performance metrics were collected as in experiment 2.

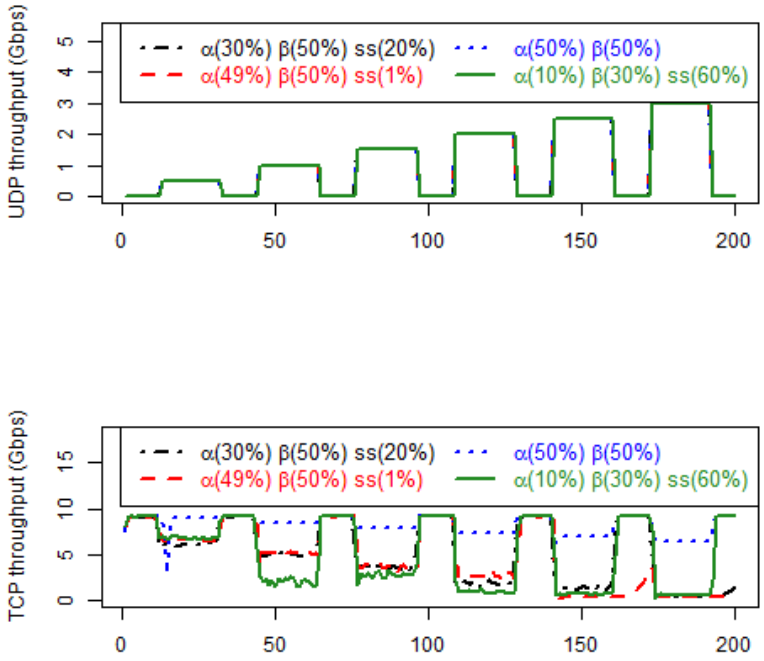


Figure 4.6: Experiment 3: The x-axis is time measured in seconds; the top graph shows the on-off mode in which the UDP rate was varied; the lower graph shows the TCP flow throughput under the four configurations.

Table 4.2: Experiment 3: α-flow throughput under different background loads (UDP rate) and QoS configurations

UDP rate (Gbps)	α-flow throughput (Gbps)			
	Percentages for 2-queues (α, β) and 3-queues (α, β, SS) configurations			
	(50,50)	(49,50,1)	(30,50,20)	(10,30,60)
0	9.12	9.09	9.07	9.12
0.5	8.92	6.62	6.06	6.83
1	8.43	5.22	5	2.12
1.5	7.94	3.78	3.67	2.82
2	7.44	2.7	1.93	0.92
2.5	6.95	0.33	1.38	0.69
3	6.46	0.34	0.38	0.61

Results and discussion

Figure 4.6 shows the TCP throughput under the four configurations (one 2-queues and three 3-queues) for different rates of the background UDP flow. When the UDP flow rate

was non-zero, since some of the plots overlap, we have summarized the mean TCP-flow throughput in Table 4.2. When there was no background UDP traffic, the throughput of the TCP flow was around 9.1 Gbps for all four configurations as seen in the first row of Table 4.2. As the background traffic load was increased, the throughput of the TCP flow in all the 3-queues configurations dropped more rapidly than in the 2-queues configuration, e.g., when the background UDP-flow rate was 3 Gbps, the TCP throughput was in the 300-610 Mbps range for the 3-queues configurations, while the TCP throughput was 6.5 Gbps for the 2-queues scenario (see last row of Table 4.2).

In addition to explaining the first and last rows of Table 4.2, we provide an explanation for the drop in TCP-flow throughput in the last column of the row corresponding to UDP rate of 1 Gbps, which highlights the importance of choosing the WFQ allocations carefully.

Explanation for the first row of Table 4.2: The explanation for the TCP-flow throughput when there was no background traffic is straightforward in the 2-queues configuration. As there were no packets to be served from the β queue and the transmitter was operating in a working-conserving manner, the β queue's 50% allocation was used instead to serve the α queue, and correspondingly the TCP flow enjoyed the full link capacity.

The explanation for the TCP-flow throughput values observed in the 3-queues configurations requires an understanding of the packet arrival pattern to the policer (see Figure 4.2) and the rate at which packets leave the policer. When TCP-flow throughput was almost the line rate (over 9 Gbps), then the rate at which in-profile packets left the policer was almost constant at 1 Gbps. This is because the token generation rate was 1 Gbps and packet inter-arrival times were too short for a significant collection of tokens in the bucket. Therefore, in an almost periodic manner, every tenth packet of the TCP flow was marked as being in-profile and sent to the α queue and the remaining 9 packets were classified as out-of-profile and sent to the *SS* queue. Given that in all the 3-queues configurations, the α queue was assigned at least 10% of the link rate/buffer space, the WFQ scheduler determined that the α queue was in-profile, and the PQ scheduler systematically served 1 packet from the α queue followed by 9 packets from the *SS* queue thus preserving the sequence of the TCP-flow packets. In the (49,50,1) configuration, 9 packets were served out of the SS queue in sequence even though the queue was out-of-profile after the first

packet was served. This is because there were no packets in the β queue and none in the α queue given the policer's almost-periodic direction of 1-in-10 packets to this queue. Since no packets were out-of-sequence or lost, the TCP-flow throughput remained high at above 9 Gbps in all the 3-queues configurations.

Explanation for the last row of Table 4.2: When there was background `nuttcp` UDP traffic at 3 Gbps, in the 2-queues configuration, it is easy to understand that the `nuttcp` TCP flow was able to use up most of the remaining bandwidth, which is the line rate minus the rate of background `nuttcp` UDP flow, and hence the TCP-flow throughput was about 6.5 Gbps.

The explanation for the low `nuttcp` TCP throughput in the 3-queues configurations is that the opposite of the systematic behavior explained above for the first row occurred here. When the incoming packet rate to the policer was lower than the line rate, the token bucket had an opportunity to collect a few tokens. Therefore, when TCP-flow packets arrived at the policer, a burst of them was classified as in-profile (since for every token present in the bucket, one packet is regarded as being in-profile), and sent to the α queue. These were served in sequence, but because the transmitter had to serve the β queue (for the UDP flow), the pattern in which the policer sent packets to the α queue and *SS* queue is unpredictable and involved bursts. This resulted in TCP segments arriving out-of-sequence at the receiver (as confirmed with `tcpdump` and `tcptrace` analyses presented in the next section). Out-of-sequence arrivals trigger TCP's Fast retransmit/Fast recovery algorithm, which causes the sender's congestion window to halve resulting in lower throughput.

Explanation for the last-column entry in the row corresponding to 1 Gbps in Table 4.2: The TCP-flow throughput dropped much faster from 6+ Gbps to 2.12 Gbps when UDP rate increased from 0.5 to 1 Gbps in the (10,30,60) 3-queues configuration than in the other two 3-queues configurations. This is explained using the above-stated reasoning that when the TCP-flow packets do not arrive at close to the line rate, the inter-packet arrival gaps allow the token bucket to collect a few tokens, making the policer send bursts of packets to the α queue. In this (10,30,60) configuration, after serving only one packet from each burst, the WFQ scheduler found the α queue to be out-of-profile since its allocation was only 10% or equivalently 1 Gbps. This led to a greater number of out-of-sequence arrivals at the TCP

receiver than in the other two 3-queues configurations, and hence lower throughput.

In *summary*, the higher the background traffic load, the lower the `nuttcp` TCP-flow packet arrival rate to the policer, the larger the inter-arrival gaps, the higher the number of collected tokens in the bucket, and the larger the number of in-profile packets directed to the α queue. If the WFQ allocation to the α queue is insufficient to serve in-profile bursts, packets from the α queue and *SS* queue will be intermingled resulting in out-of-sequence packets at the receiver. This fine point notwithstanding, the option of directing out-of-profile packets from the policer to a separate queue appears to be detrimental to α-flow throughput. We conclude that the second goal of high α-flow throughput cannot be met with this policing approach. In the next experiment, a different mechanism for dealing with out-of-profile packets was tested.

4.3.5 Experiment 4

Purpose and execution

The goal of this experiment was to compare the approach of applying WRED to out-of-profile packets rather than redirecting these packets to a scavenger-service queue as in experiment 3. The planned applications were the same as in experiment 3, i.e., to generate one `nuttcp` TCP flow and one `nuttcp` UDP flow.

The next step was router configuration. Four configurations are compared as shown in Table 4.3. In the fourth option, Out-of-Profile (OOP) packets are dropped probabilistically at the same rate as the fraction of α-queue occupancy. In other words, if the α queue has 50% occupancy, then 50% of the OOP packets are dropped on average. The policing rate and burst size settings were the same as in Experiment 3.

Both the TCP and UDP flows were executed for 200 sec, but unlike in experiment 3, the rate of the UDP flow was maintained unchanged at 3 Gbps for the whole time period. Finally, in addition to the previously used methods of obtaining throughput reports from `nuttcp`, two packet analysis tools, `tcpdump` and `tcptrace`, were used to determine the number of out-of-sequence packets at the receiver. Additionally, to find the number of lost packets, a counter was read at router `WR` for the `WR`-to-`W1` link before and after each application run.

Table 4.3: Experiment 4: QoS configurations; OOP: out-of-profile

Configuration	Policing	WFQ allocation `2-queues`:(α,β) `3-queues`:(α,β,`SS`)	WRED
`2-queues`	None	(60,40)	NA
`3-queues + policing1`	OOP to `SS` queue	(59,40,1)	NA
`3-queues + policing2`	OOP to `SS` queue	(20,40,40)	NA
`2-queues + policing + WRED`	WRED	(60,40)	Drop prob. = queue occ.

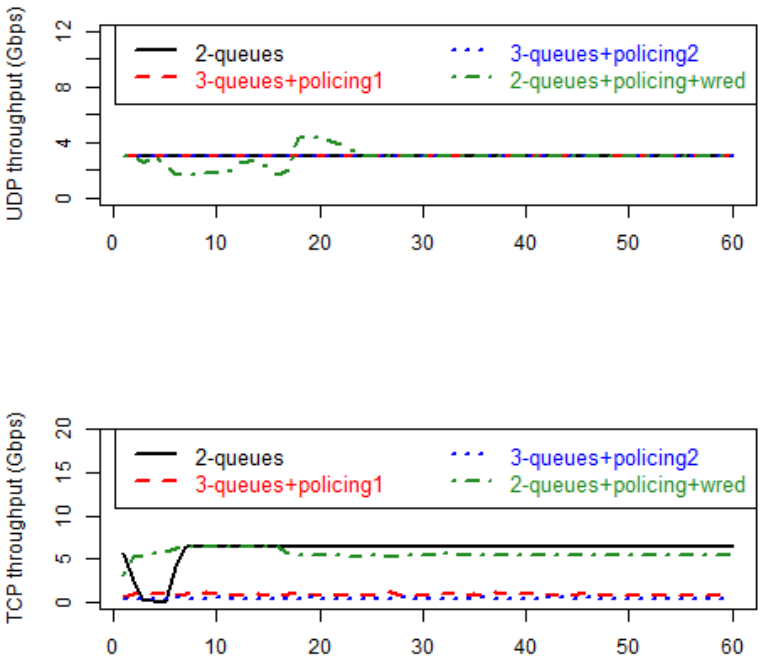


Figure 4.7: Experiment 4: The x-axis is time measured in seconds; the top graph shows the on-off mode in which the UDP rate was varied; the lower graph shows the TCP flow throughput under the four configurations.

Results and discussion

The lower graph in Figure 4.7 and Table 4.4 show that the TCP-flow throughput is highest in the 2-queues (no-policing) scenario, with the WRED option close behind. The policing with

WRED option performs much better than the options in which out-of-profile (OOP) packets are directed to an *SS* queue. In the WRED-enabled configuration, the TCP flow experiences a small rate of random packet loss, as shown in Table 4.4, while in 3-queues configurations, there were much higher numbers of out-of-sequence packets. The out-of-sequence packets in the WRED-enabled configuration result from the 15 lost packets, and are not independent events.

Table 4.4: Experiment 4: number of out-of-sequence packets and lost packets for different QoS settings

Measure	`2-queues`	`3-queues+ policing1`	`3-queues+ policing2`	`2-queues+ policing+wred`
Average throughput	6 Gbps	0.92 Gbps	0.47 Gbps	5.6 Gbps
Num. of out-of-sequence packets at the receiver	4076	8812	7199	15
Num. of lost packets at the `WR`-to-`W1` router link	5050	0	0	15

Surprisingly, even though the number of out-of-sequence packets was larger for the `3-queues+policing1` configuration, the throughput was higher in that configuration. This implies that fewer number of the out-of-sequence packets caused triple-duplicate ACKs in the first case. But this pattern is likely to change for repeated executions of the experiment.

Finally, Figure 4.7 shows that in the `2-queues` (no-policing) configuration, there was degradation of throughput soon after the flow started. Also, Table 4.4 shows a loss of 5050 packets (the 4076 out-of-sequence packets were related to these losses). Using `tcptrace`, we found that these losses occurred at the start of the transfer. This is explained by the aggressive growth of the congestion window (`cwnd`) in H-TCP, which uses a short throughput probing phase at the start. During the 1st second, the throughput of the TCP flow averaged 5.7 Gbps. The 5050 lost packets occurred in the 2^{nd} second. These losses occurred in the `WR` router buffer on its egress link from `WR` to `W1`. If H-TCP increased its `cwnd` to a large enough value to send packets at an instantaneous rate higher than 7 Gbps, then given the presence of the UDP flow at 3 Gbps, the α queue would fill up. From experiment 1, we determined that the particular router used as `WR` has a 125 MB buffer. Since the buffer is

shared between the α and β queues in a strictly partitioned mode with the 60-40 allocation, the α queue has 75 MB, which means that if the H-TCP sender exceeds the 7 Gbps rate by even 600 Mbps, the α queue will fill up within a second. Inspite of this initial packet loss, the `2-queues` no-policing configuration achieves the highest throughput. In the next experiment, we consider the question of whether the use of policing and WRED has a fairness advantage when multiple α flows share a queue.

4.3.6 Experiment 5

Purpose and execution

The goal of this experiment was to understand how two α flows compete for bandwidth under different 2-queues configurations: without policing (`2-queues`), and with policing and WRED (`2-queues+policing+WRED`). In a first scenario, the α flows had similar round-trip times (RTTs), while in a second scenario, the RTTs differed significantly. We expected a fairness advantage for the policing/WRED scheme, but found the opposite. This is because neither of the two policed α flows honored their assigned rates, and while under the no-policing scheme the TCP flows adjusted their sending rates and had no packet losses, the deliberate packet losses in the policing/WRED scheme lowered throughput and resulted in a lower fairness. Thus, the no-policing configuration out-performs the configuration with policing and WRED from both throughput and fairness considerations when neither flow honors the policed rate.

The first step was to choose applications. Two `nuttcp` TCP flows were planned. The first TCP flow (`TCP1`) was from host `E2` to host `W1`, and the second TCP flow (`TCP2`) was from host `E1` to host `W1`. The RTTs were similar but not exactly the same. The RTT was 1.98 ms on the `E2-to-W1` path and 2.23 ms on the `E1-to-W1` path, because the latter path passes through an additional router, `ER`.

The router configurations were as follows. In the `2-queues` configuration, packets from both TCP flows were directed to an α queue, with the rate and buffer allocations set to (60,40) for the α and β queues, respectively. In the `2-queues+policing+WRED` configuration, the policing rate/burst size settings were the same as in Experiment 3, and Out-of-Profile

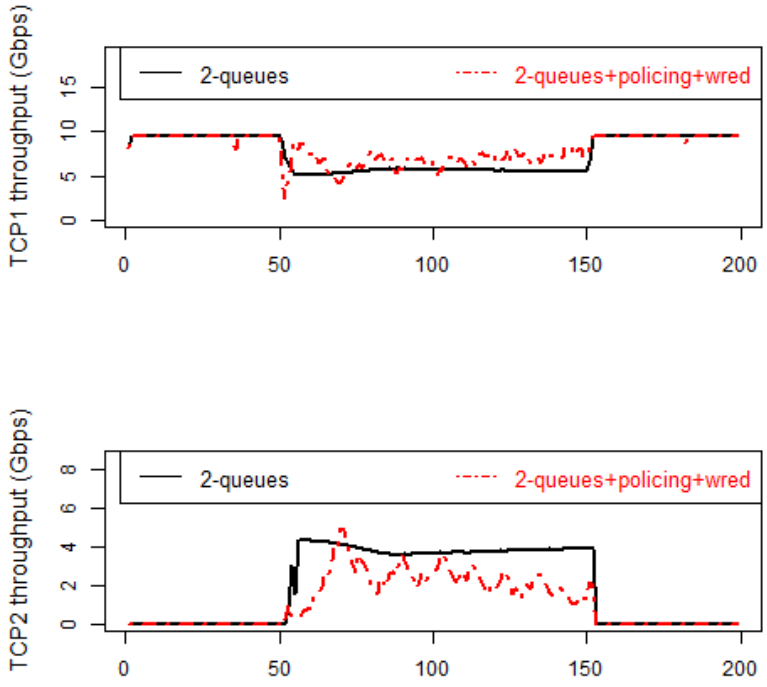


Figure 4.8: Experiment 5: Throughput of two TCP flows under two QoS configurations (similar RTTs)

(OOP) packets were dropped probabilistically with the same settings as in Experiment 4 ([0,100] drop probability corresponding to [0,100] buffer occupancy.

`TCP1` and `TCP2` execution intervals were (0, 200) and (51, 151), respectively. In the different-RTTs scenario, the RTT of `TCP2` was increased by 50 ms using `tc`. Finally, throughput and retransmission data were collected every second by `nuttcp` at the senders.

Results and discussion

Experimental results are presented for the similar-RTT and different-RTTs scenarios.

Similar-RTT scenario Figure 4.8 shows the throughput of the two TCP flows when they compete for the bandwidth and buffer resources of the α queue. In the `2-queues` configuration, the throughput of `TCP1` was approximately 9.4 Gbps for the first 50 seconds, but dropped to 7.1 Gbps at $t = 51$, since `TCP2` was initiated then. In the 52^{nd} second, both flows suffered packet losses, with `TCP1` requiring 2418 retransmissions and `TCP2` requiring 3818

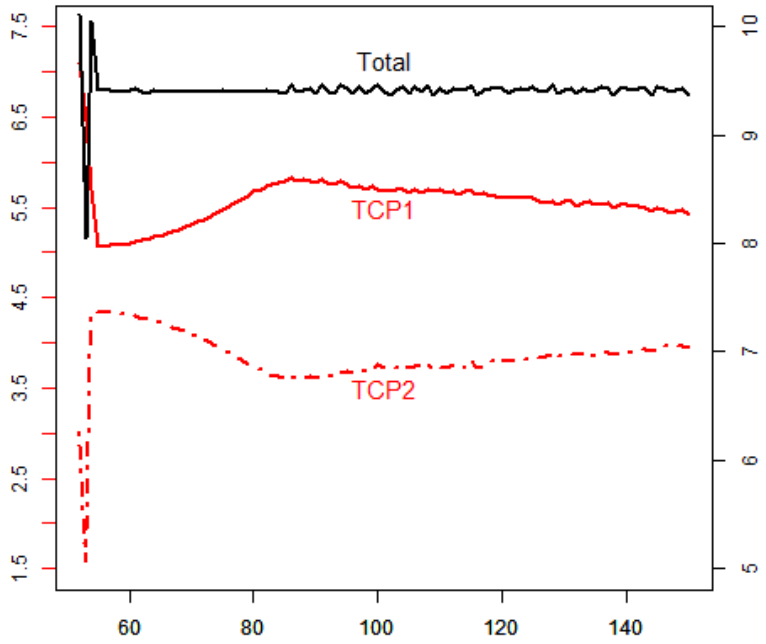


Figure 4.9: Experiment 5: Throughput of two TCP flows, and their total throughput in the `2-queues` configuration (similar RTTs)

retransmissions. Since the sum of the rates of the flows exceeded 10 Gbps, it caused losses and retransmissions in the 52^{nd} sec. After the 52^{nd} second, there were no retransmissions on either flow. The per-second throughput recorded by `nuttcp`, from $t = 51$ to $t = 151$ during which both TCP flows were active, is shown in Figure 4.9. As the buffer filled up and queueing delays increased, TCP acknowledgments (ACKs) would have been delayed causing RTT for `TCP1` to increase. This decreased the effective sending rate (`cwnd`/RTT). No losses occurred in the rest of the experiment because as sending rates increased in one or both flows, the buffer filled up delaying packets, and hence increasing RTT, which in turn caused the sending rate to drop thus reducing buffer buildups. This oscillatory behavior can be observed in the throughput sum plot of Figure 4.9. The higher rate of `TCP1` could be because of the slightly lower RTT for this flow when compared to that of TCP2.

Next, consider the throughput values of `TCP1` and `TCP2` in the `2-queues+policing+WRED` configuration shown in Figure 4.8. From $t = 51$, `TCP1` suffered losses and its throughput

Table 4.5: Experiment 5: retransmissions and throughput of 2 TCP flows for the policing/WRED configuration (similar RTTs)

Time (s)	TCP	Retrans-missions	Throughput (Gbps)		
			Min	Median	Max
51 - 53	TCP1	227	2.56	4.84	6.31
	TCP2	32	0.46	0.5	1.03
54 - 69	TCP1	14	4.35	7.37	8.98
	TCP2	0	0.47	1.78	4.98
70 - 151	TCP1	65	4.26	6.91	8.47
	TCP2	3	1.02	2.26	4.26

dropped steadily until it reached 4.35 Gbps, while `TCP2` throughput kept increasing until it reached 4.98 Gbps at $t = 69$. The reason why `TCP1` throughput dropped is because of the policing limit of 1 Gbps. Packets exceeding this rate were marked as out-of-profile. Since `TCP1` rate was 9.4 Gbps at $t = 50$ just before `TCP2` was started, its sending rate was well above the policing rate of 1 Gbps. Subsequent to reaching this almost balanced throughput level at $t = 69$, losses, and hence retransmissions, were observed on both flows, but there were more losses in `TCP1` (see Table 4.5) because its rate was higher.

The key difference between the `2-queues` and `2-queues+policing+WRED` configurations is that there were no losses in the former configuration after $t = 53$, while in the latter configuration both flows kept experiencing packet losses. This is because in the second configuration, as both flows exceeded the policing limit of 1 Gbps, a few packets were marked as out-of-profile in both flows. Recall that under WRED packets are dropped probabilistically at a rate equal to buffer occupancy, and since the buffer will sometimes have packets, losses are inevitable in the `2-queues+policing+WRED` configuration. When losses occurred under the `2-queues+policing+WRED` configuration, the slight edge in RTT for `TCP1` may account for its higher throughput when compared to `TCP2`. `TCP1` maintained an average rate of 6.86 Gbps from $t = 70$ to $t = 151$ when `TCP2` was terminated, at which point `TCP1` recovered its rate to 9.4 Gbps. The `TCP2` average throughput from $t = 70$ to $t = 151$ was smaller at 2.35 Gbps. A loss detected with triple duplicate ACKs results in a halving of `cwnd`, which is equivalent to halving the sending rate. `TCP1` operated in a higher range of `cwnd` values when compared to `TCP2`.

Using Jain's fairness index [71],

$$f(x) = \frac{(\sum_{i=1}^{n} x_i)^2}{n \cdot \sum_{i=1}^{n} x_i^2} \quad x_i \geq 0 \tag{4.3}$$

and average throughput values across the $t = 51$ to $t = 151$ time range, we computed the fairness values to be 0.97 and 0.8 for the `2-queues` and `2-queues+policing+WRED` configurations, respectively. This does not imply that the former is a more fair configuration; it is just that in this experiment, given that both TCP flows did not honor the policing limit, policing caused packet losses, and recovery from packet losses was slower for the longer-RTT path even if the RTT difference was small. Without policing, there were no deliberate packet drops in the `2-queues` configuration; instead the TCP senders self-regulated their sending rates. When the rates were high, buffer occupancy grew, but this caused RTT to increase, which, in turn, caused a lower sending rate.

In *summary*, this experiment showed that policing will result in decreased throughput for TCP based α flows when two or more such flows occur simultaneously. In Experiment 4, policing with WRED did not impact throughput significantly but there was only one TCP based α flow, unlike in this experiment.

Different RTTs Figure 4.10 shows the throughput of the two TCP flows with different RTTs. During the 100-second period when both TCP flows were active, the throughput of the two TCP flows and their total throughput are plotted in Figure 4.11. The throughput of `TCP1` dropped from 9.1 Gbps at $t = 50$ to 7.1 Gbps at $t = 51$, since `TCP2` was initiated at time 50. In the 57^{th} second, when `TCP2` built up its rate to 2.93 Gbps, which made the sum of the rates exceed 10 Gbps, both flows suffered packet losses, with `TCP1` requiring 3315 retransmissions and `TCP2` requiring 4118 retransmissions. After the 57^{th} second, there were no retransmissions on either flow. Since the RTT of `TCP2` was increased by 50 ms, it took 6 sec to reach the time instant when losses occurred unlike in the similar-RTT scenario in which both flows experienced losses in 2 sec. In the second after the losses, `TCP1` recovered its throughput back to 9.38 Gbps, while `TCP2` throughput decreased from 2.92 Gbps to 11 Mbps.

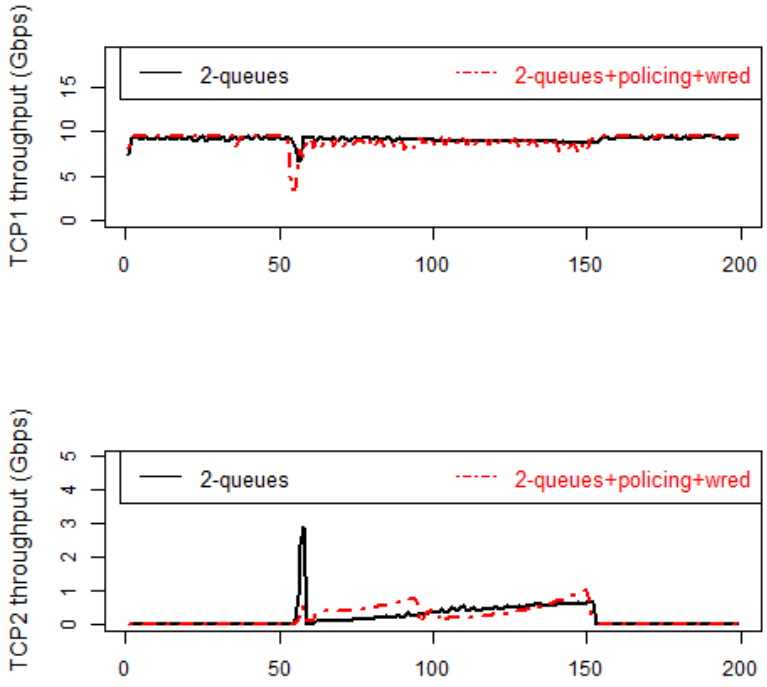


Figure 4.10: Experiment 5: Throughput of two TCP flows under two QoS configurations (different RTTs)

Table 4.6: Experiment 5: retransmissions and throughput of 2 TCP flows for the policing-WRED configuration (different RTTs)

Time (s)	TCP	Retrans-missions	Throughput (Gbps)		
			Min	Median	Max
51 - 58	TCP1	140	3.51	7.6	9.41
	TCP2	1	0.003	0.086	0.54
59 - 151	TCP1	107	7.61	8.81	9.35
	TCP2	4	0.056	0.42	0.98

Next, we repeated the experiments with the policing and WRED configuration. The retransmissions and throughput of the two TCP flows are shown in Table 4.6. `TCP1` experienced losses even after the initial set of losses unlike in the `2-queues` configuration. Consequently, TCP1's average throughput was lower in the `2-queues+policing+WRED` configuration (8.98 Gbps) than in the `2-queues` configuration (9.1 Gbps), while TCP2's average throughput was higher (0.43 Gbps vs 0.41 Gbps). Jain's fairness index value for

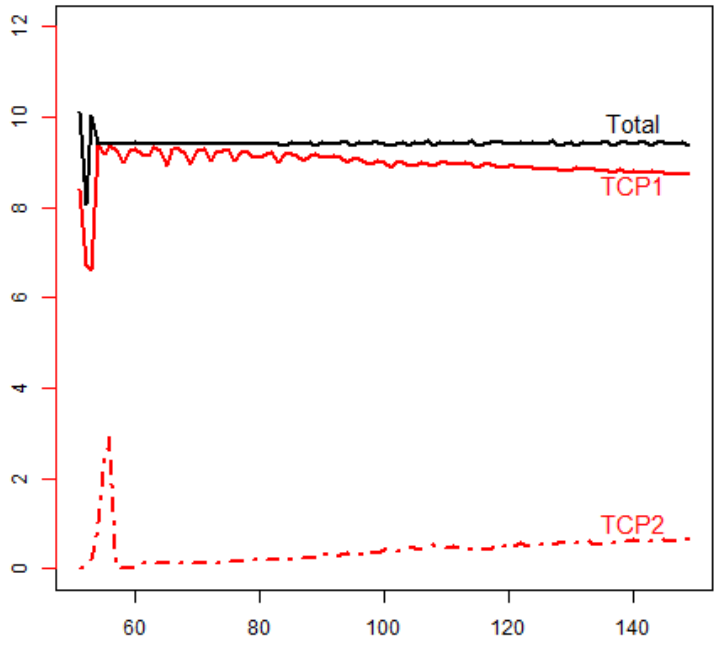


Figure 4.11: Experiment 5: Throughput of two TCP flows, and their total throughput in the `2-queues` configurations (different RTTs)

throughput of the two TCP flows was comparable under the two configurations (0.546 and 0.551 under the `2-queues` and `2-queues+policing+WRED` configurations, respectively). The RTT difference was the dominant reason for the unfair treatment of `TCP2`, not the QoS configuration.

4.3.7 Experiment 6

Purpose and execution

The goals of this experiment were to (i) identify the impact of QoS provisioning under changing traffic conditions, and (ii) compare two versions of TCP: Reno and H-TCP. In the first part, we studied the effect of enabling QoS control, specifically, the `2-queues` configuration, on changing traffic patterns. For example, what is the impact of background traffic increasing to 3 Gbps when the β queue to which background traffic was directed was allocated only 20% of the rate/buffer capacity on a 10 Gbps link (based on previous

traffic measurements). As α flows occur infrequently, most of the time, service quality for the background traffic would be unaffected, but if this surge in background traffic occurred within the duration of an α flow, there could potentially be higher losses and delays in the background traffic than if QoS mechanisms had not been enabled.

As mentioned in Section 4.3.3, the TCP version used in our experiments was H-TCP, the recommended option for high-speed networks [70]. However, although computers dedicated for high-speed transfers are likely to be configured to use H-TCP, as the most widely used TCP version is still TCP Reno, we undertook a comparative experiment.

Three applications were planned for this experiment: one `nuttcp` TCP flow (from host `E1` to `W1`), one `nuttcp` UDP flow (from host `E2` to `W1`) and one ping flow (from host `EA` to `W1`). In the router configuration step, two queues were configured: a β queue for the background UDP flow and the ping flow, and an α queue for the TCP flow. The rate/buffer allocation (the same percentage was used for both resources) for the β queue was varied from 20% to 60% in 10% increments, and the allocations for the α queue were set correspondingly. The applications were executed as follows: UDP flow and ping flow in the time interval (0, 200), and the TCP flow in the interval (53, 153). Two rates were used for the UDP flow: 2 Gbps and 3 Gbps.

Results and discussion

Goal 1 Table 4.7 shows the UDP-flow loss rate and ping delay under different rate/buffer allocations for the β queue in the `2-queues` configuration. Before the TCP flow was initiated (the first 53 seconds) and after the TCP flow ends (the last 47 seconds), even if the rate of the UDP flow exceeded the allocated rate for the β queue (i.e., 20% allocation when the UDP-flow rate was 3 Gbps), the UDP flow experienced no losses, and the ping delay remained at around 2.26 ms, which implies that there was no buffer buildup in the β queue. This is because the transmitter was operating in work-conserving mode, which allowed it to serve packets from the β queue as the α queue was empty.

During the time interval (53-153) when the TCP flow was active, with a 20% rate/buffer allocation for the β queue, a 2 Gbps UDP flow suffered a 5% packet-loss rate, and the ping delay was 103 ms, which means the β queue was full. When the UDP-flow rate was

Table 4.7: Experiment 6: UDP-flow loss rate and ping delay

	β queue rate and buffer allocation	UDP rate (Gbps)	UDP flow average packet loss rate before, during, and after the TCP flow			Average ping delay (ms) before, during, and after the TCP flow		
			t $\in$ (0-52)	t $\in$ (53-153)	t $\in$ (154-200)	t $\in$ (0-52)	t $\in$ (53-153)	t $\in$ (154-200)
Reno	20%	2	0	5.03%	0	2.25	103	2.26
	30%	2	0	0	0	2.3	2.25	2.25
	20%	3	0	39.33%	0	2.26	103	2.27
	30%	3	0	4.57%	0	2.27	104	2.31
	$\geq$ 30%	2 or 3	0	0	0	2.26	2.26	2.26
H-TCP	20%	2	0	5.3%	0	2.27	103	2.27
	30%	2	0	0	0	2.27	2.26	2.25
	20%	3	0	39.3%	0	2.26	103	2.27
	30%	3	0	4.67%	0	2.28	104	2.29
	$\geq$ 30%	2 or 3	0	0	0	2.26	2.27	2.27

increased to 3 Gbps, while the β-queue allocation was held at 20% (to model changing traffic conditions), the UDP-flow packet loss rate increased to about 39%, and the ping delay remained at 103 ms. Such a significant loss rate and increased packet delay would not have occurred had separate QoS classes not been created and the buffer not been divided. When the UDP-flow rate increased, the TCP-flow rate would have decreased as it would also have suffered losses. In the `2-queues` configuration, the TCP flow suffered no losses for both the combinations described above: 20% β queue allocation with 2 Gbps UDP-flow rate, and the 20%-3 Gbps combination. This is because the TCP flow was directed to the α queue, which had its own large (80%) buffer/rate allocation.

Goal 2 The numbers in Table 4.7 show that there was no difference between H-TCP and Reno with regards to the impact of the TCP flow on the UDP and ping flows. Furthermore, Table 4.8 shows that the TCP flow enjoyed the same rate for most of its duration. When the background UDP-flow rate was 2 Gbps, the TCP-flow throughput was 7.45 Gbps, and when the UDP-flow rate was 3 Gbps, the TCP-flow throughput was correspondingly lower at 6.45 Gbps, irrespective of β-queue rate/buffer allocation. The only difference observed between Reno and H-TCP was in the TCP-flow's behavior in the first few seconds as shown in Table 4.9. Recall the TCP flow was started at $t = 53$. With Reno, the number of

retransmissions that occurred in the early seconds drops as the β-queue buffer allocation was increased (and the α-queue size, to which the TCP flow was directed, correspondingly decreased). With smaller α-queue sizes, it appears that the TCP sender starts reducing its sending rate sooner, and hence there were fewer losses and retransmissions. We expected H-TCP to suffer more losses in the initial few seconds as it is more aggressive in increasing its sending window, but this was not observed. Both adjusted their sending rates and experienced no losses after the initial set of losses shown in Table 4.9.

Table 4.8: Experiment 6: TCP-flow throughput for most of the duration

Background (UDP) rate	TCP throughput	
	Reno	H-TCP
2 Gbps	7.45 Gbps	6.45 Gbps
3 Gbps	7.45 Gbps	6.45 Gbps

Table 4.9: Experiment 6: TCP-flow retransmissions in its first few seconds (the flow was started at $t = 53$)

UDP-flow rate	β-queue rate/ buffer alloc.	Time	Number of retx pkts
Reno			
2 Gbps	30%	$t = 54$	6624
	40%	$t = 54$	5811
	50%	$t = 53$	4327
	60%	$t = 54$	2645
3 Gbps	30%	$t = 54$	7673
	40%	$t = 54$	6970
	50%	$t = 53$	5137
	60%	$t = 54$	3495
H-TCP			
2 Gbps	30%	$t = 54$	6008
	40%	$t = 54$&55	4322 & 298
	50%	$t = 53$	3825
	60%	$t = 54$	3966
3 Gbps	30%	NA	0
	40%	$t = 54$	1423
	50%	NA	0
	60%	$t = 54$	3528

4.3.8 Experiment 7

Purpose and execution

As described in Section 4.1, AFTES uses an offline approach by analyzing NetFlow reports of completed flows to determine source-destination addresses of α flows, and then uses these addresses to configure firewall filters in ingress routers of a provider's network to redirect packets of future α flows to traffic-engineered QoS-controlled paths. With this scheme, an α flow between a new source-destination pair will not be identified as such until its NetFlow reports are analyzed, which most likely will occur after the flow completes. Such unidentified α flows will be directed to the β queue in a `2-queues` configuration. Since in such a configuration, buffer resources are partitioned between the β queue and α queue, the purpose of this experiment was to study the impact of such unidentified α flows.

Three `nuttcp` flows were planned for this experiment: a UDP flow from `E2` to `W1`, TCP flow `TCP1` from `E2` to `W1`, and a second TCP flow `TCP2` from `E1` to `W1`. In addition, a ping flow was executed from from `EA` to `W1`. Two router configurations were used in this experiment: (i) `1-queue`: a single virtual queue was defined on the egress interface from `WR` to `W1`, and all flows were directed to this queue, and (ii) `2-queues`: two virtual queues (α queue and β queue) were configured on the egress interface from `WR` to `W1`, and WFQ scheduling was enabled with the following rate (and buffer) allocations: 60% for α queue and 40% for β queue.

The execution intervals of the flows, ping, UDP, `TCP1`, and `TCP2` were (0,200), (0, 200), (42, 101), and (22, 162), respectively. The rate of the UDP flow was set to 3 Gbps. We assumed `TCP1` to be the unidentified α flow, which was hence directed to the β queue, while TCP2 was assumed to be an α flow from a previously seen source-destination pair, and hence directed to the α queue. The ping and UDP flows were directed to the β queue. Measurements were collected from the `nuttcp` and ping applications.

Results and discussion

The throughput of the two TCP flows and the ping delays are shown in Figure 4.12. In the `1-queue` configuration, during the 60 seconds when both TCP flows were active, `TCP1`

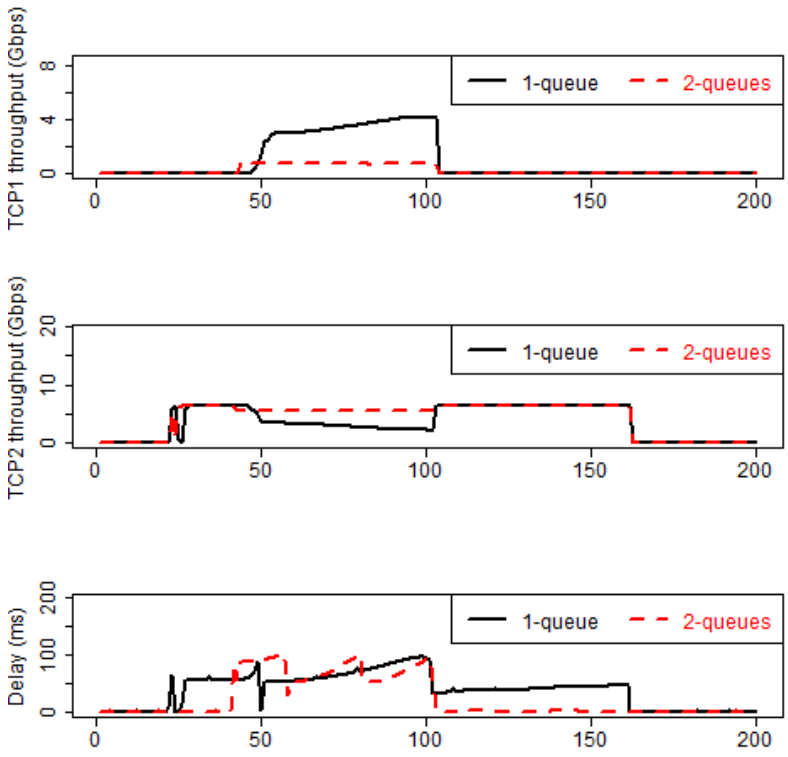


Figure 4.12: Experiment 7: The impact of an unidentified α flow with and without HNTES

throughput kept increasing to 4.16 Gbps, while `TCP2` throughput kept decreasing from 6.5 Gbps to 2.26 Gbps. This is because the RTT was slightly lower for `TCP1` as discussed earlier.

In the `2-queues` configuration, `TCP1` throughput was only 1 Gbps. This is because the β queue allocation was 40% of the link rate/buffer, of which 3 Gbps was used by the UDP flow, and `TCP2` was actively consuming the 60% allocation of the α queue. The mean throughput of the new α flow (TCP1) in the `1-queue` case was 3.2 Gbps, while it was only 0.8 Gbps under the `2-queues` configuration. In other words, the presence of HNTES and the corresponding `2-queues` configuration had an adverse effect on the unidentified α flow, though as shown in our prior work, most α-flow generating source-destination pairs send repeated α flows [14].

Consider the impact of the unidentified α flow on the ping flow. In the `1-queue` configuration, the ping delay was around 2.3 ms until `TCP2` was initiated at $t = 22$, at which instant the ping delay surged up to 65.9 ms as seen in Figure 4.12 because of the buffer build-up from TCP2 packets. Since H-TCP is aggressive in increasing its sending rate, in

Table 4.10: Experiment 7: TCP-flow retransmissions and ping delays

(sec, no. of `TCP1` retx)	(sec, no. of) `TCP2` retx)	(sec, ping delay (ms))
`1-queue configuration`		
NA	(23, 3267)	(24, 2.25)
(50, 955)	(50, 568)	(50, 4.7)
`2-queues configuration`		
NA	(22, 8074)	(22, 2.3)
(44, 1855)	(44, 0)	(44, 87.3)
(60, 7)	(60, 0)	(60, 30.2)
(82, 7)	(82, 0)	(83, 48.5)

its 2^{nd} second ($t = 23$), there were 3267 packet drops as shown in Table 4.10. With all these losses, ping delay correspondingly dropped down to 2.25 ms at $t = 24$. However the delay quickly increased back to the 56 ms range peaking at 88.7 ms at $t = 49$. As shown in Table 4.10, it took a few seconds after `TCP1` was initiated for both `TCP1` and `TCP2` to experience packet losses causing ping delay to drop back down to 4.7 ms at $t = 50$. Beyond this time instant, neither TCP flow suffered losses with both adjusting their sending rates based on received acknowledgments and ping delay peaked at 91.5 ms at $t = 101$ when `TCP1` ended. The ping delay dropped to 34 ms and increased to 47.9 ms at which point it dropped to 2.3 ms at $t = 162$ when `TCP2` ended.

In the `2-queues` configuration, the ping delay stayed around 2.3 ms even after `TCP2` was initiated as seen in Figure 4.12 (because `TCP2` was directed to a different queue), but increased to 87.3 ms when `TCP1` was initiated at $t = 43$ (since `TCP1` representing an unidentified α flow was directed to the same queue as the ping flow). `TCP2` suffered no losses after the initial losses of 8074 packets in its first second. On the other hand, `TCP1` suffered losses not only in its first second (1855 losses), but again at $t = 60$ and $t = 82$. During these seconds, ping delay dropped correspondingly from 103 ms at $t = 59$ to 30.2 ms at $t = 60$, and from 103 ms at $t = 82$ to 48.5 ms at $t = 83$. These results illustrate that the smaller buffer allocation for the β queue can have a negative effect on real-time flows when an unidentified α flow appears.

In summary, QoS partitioning does have negative effects when mismatched with traffic

as shown in Experiment 6, and when α flows are undetected and hence handled by the partition set aside for β flows. Nevertheless, the benefits of QoS partitioning as illustrated in the first five experiments outweigh these costs.

4.4 Conclusions

The contribution of this work is to determine the best QoS mechanisms for the virtual circuits used in this application. Our findings are that a no-policing, two-queues (one for α flows and one for β flows) solution with weighted fair queueing and priority queueing is both sufficient and the best for this application. It allows for the dual goals of reduced delay/jitter in real-time flows, and high-throughput for α flows, to be met.

We studied two types of policing schemes for handling out-of-profile packets: redirection to a (third) scavenger-service (SS) queue and Weighted Random Early Detection (WRED) in which out-of-profile packets are either dropped probabilistically according to some profile or held in the same queue as in-profile packets. The WRED scheme was better than the SS-queue scheme because the latter caused out-of-sequence arrivals at the receiver, which triggered TCP congestion control mechanisms that led to lower throughput. However, the no-policing solution was better than the policing/WRED solution because in this application flows are not likely to honor the circuit rates and therefore deliberate packet drops are inevitable in the policing/WRED solution causing lowered throughput. The negatives of partitioning rate/buffer space resources between two queues were studied. Our conclusions are that close network monitoring is required to dynamically adjust the rate/buffer space split between the two queues as traffic changes, and the probability of unidentified α flows should be reduced whenever possible to avoid these flows from becoming directed to the β queue.

Chapter 5

Conclusions and Future Work

We first summarize the work presented in this dissertation and draw three key conclusions, and then suggest future work items to extend this research.

5.1 Summary and conclusions

In this work, we developed a Hybrid Network Traffic Engineering System (HNTES) that can automatically identify and redirect heavy-hitter flows within a core network in order to mitigate the adverse effects heavy-hitter flows can have on delay-sensitive flows. The HNTES system consists of modules to perform three steps: (i) heavy-hitter flow identification, (ii) circuit provisioning, and (iii) policy-based route (PBR) configuration at the ingress and egress routers. Each of these steps can be performed online (on flow arrival) or offline (a priori).

Chapter 2 presented a HNTES 1.0 design in which these three steps are performed online. With regards to the first step of heavy-hitter flow identification, first, we need to define the dimension on which a flow is measured to determine whether or not it is a heavy hitter. Flows have four dimensions: duration, size, rate, duration and burstiness. The HNTES 1.0 design uses the duration dimension. Since the ESnet-deployed dynamic circuit scheduler takes on the order of 1 minute to create a circuit, only long flows were considered to be suitable for hybrid network traffic engineering. Based on the results of our experiments and NetFlow data analysis, certain practical considerations were identified,

1) it is cost prohibitive to deploy this online HNTES because it requires online analysis of packet headers, which is difficult to implement in high-speed networks, 2) deep packet inspection of only control-plane packets to extract data-connection source and destination address and port numbers is infeasible because control-plane packets are often encrypted. These concerns made us switch from the online solution to an offline solution for the first step (heavy-hitter flow identification), which necessitated the execution of steps (ii) and (iii), circuit provisioning and PBR configuration, to be offline as well. Therefore, instead of dynamic circuits (set up after flow arrival) circuits are provisioned a priori between ingress and egress routers. Such a full mesh of circuits is feasible because these are virtual circuits, which means bandwidth and buffer resources can be shared among them, unlike time- and frequency-division multiplexed circuits. Effectively, the fine-granularity approach in HNTES 1.0 (one circuit per flow) was replaced by coarse-granularity virtual circuits (shared by multiple flows between same ingress-egress router pairs). Therefore, our *first conclusion* is that online HNTES designs may not be feasible in high-speed networks, and that long-duration flows whose rates are low typically do not have adverse effects on other flows, and hence do not need to be sent to special traffic-engineered paths.

In Chapter 3, new dimensions were used in determining whether or not a flow was a heavy-hitter. Based on an experiment conducted on a high-speed network testbed, ESnet link usage measurements, and science data-transfer log analysis, rate and size were selected as the dimensions of interest for traffic engineering heavy-hitter flows (α flows). An offline mechanism was designed for determining prefix identifiers of α flows that are typically created in research-and-education networks by scientific researchers who move large datasets between well-equipped data-transfer nodes. The offline scheme was implemented in a network management system called Alpha Flow Traffic Engineering System (AFTES). Prefix identifiers (IDs) of completed α flows are used to set firewall filters to redirect future α flows to traffic-engineered paths, and to isolate their packets to separate queues. Through an analysis of 7 months of NetFlow data obtained from an ESnet router, the offline mechanism was found to be highly effective in that 91% of bytes generated by α flows in bursts would have been directed had AFTES been deployed at the start of this period. Also, our analysis showed that most of the general-purpose flows that were redirected to the same paths as α

flows were from file-transfer applications, which are not as affected by α flows as packets from real-time applications. Our *second conclusion*, drawn from this AFTES design and evaluation work, is that α flows are created repeatedly between the same high-end data transfer nodes, which makes the offline solution useful for deployment.

Chapter 4 studied (experimentally) different scheduling and policing mechanisms to achieve two goals: (i) reduce delay and jitter of real-time delay sensitive flows that share the same interfaces as α flows, and (ii) achieve high throughput for α-flow transfers. Two types of policing mechanisms for handling out-of-profile packets were studied: redirection to a third scavenger-service (SS) queue and Weighted Random Early Detection (WRED). The WRED scheme was better than the SS-queue scheme because the latter caused out-of-sequence arrivals at the receiver, which triggered TCP fast retransmit mechanism and led to lower throughput. However, the no-policing solution was better than the policing/WRED solution because TCP flows are bursty, which means router buffers can start filling up, and with WRED, packets will be probabilistically dropped based on buffer occupancy. In TCP, packet drops lead to reduced throughput because the congestion control algorithm halves the sending rate when drops are detected. The WRED effects were observed experimentally; they occur because the AFTES solution is intra-domain, which means end-user applications are not part of the decision process to use a virtual circuit for a portion of the end-to-end path. Without knowing that their flows are being policed, end applications cannot control their sending rates, which are determined by the TCP congestion-control mechanisms. Our *third conclusion*, drawn from this experimental study, is that a no-policing, two-queues (one for α flows and one for β flows) solution with weighted fair queueing and priority queueing is both sufficient and the best for this application. It allows for the dual goals to be met.

5.2 Future Work

This work can be extended in the following directions:

The *alpha*-flow identification algorithm was evaluated using 7 months of NetFlow data obtained from an ESnet router. The evaluation could be applied to NetFlow data collected from multiple ESnet routers, and routers from other core networks.

While it is cost prohibitive to execute packet header analysis from port-mirrored packet traces as the HNTES 1.0 experience showed, an online scheme may still be feasible with real-time analysis of NetFlow data. Routers can be configured with interface cards that can export NetFlow data every 10 sec, though these cards are expensive too.

The experiments we conducted to study the effects of scheduling and policing mechanisms on TCP flows were useful but could be generalized. Theoretical and/or simulation models can be created and analyzed to characterize the impact of QoS provisioning schemes on TCP throughput.

Bibliography

[1] Z. Yan, M. Veeraraghavan, C. Tracy, and C. Guok. On how to provision Quality of Service (QoS) for large dataset transfers. In *Proceedings of the Sixth International Conference on Communication Theory, Reliability, and Quality of Service (CTRQ)*, Apr. 21-26, 2013.

[2] B. Tierney. ESnet and ANI Testbed Update. DOE PI Meeting, Mar. 2011.

[3] V. Cerf, V. Jacobson, N. Weaver, and J. Getty. Bufferbloat: what′s wrong with the Internet? *Communications of the ACM*, 55(2):40–47, 2012.

[4] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin. Resource ReSerVation Protocol (RSVP). RFC 2205, Sep. 1997.

[5] K. Lan and J. Heidemann. A measurement study of correlations of Internet flow characteristics. *Computer Networks*, 50(1):46–62, 2006.

[6] Multiprotocol Label Switching (MPLS). http://www.cisco.com/en/US/products/ps6557/products_ios_technology_home.html.

[7] ESnet. `http://www.es.net/`.

[8] Interoperable Ondemand Network (ION). `http://spaces.internet2.edu/download/attachments/10659/Internet2+ION+Service+Definition+v1.0.pdf?version=1&modificationDate=1249591387079`.

[9] Development of Dynamic Network System (DYNES). `http://www.internet2.edu/ion/dynes.html`.

[10] The Lambda Station Project. http://www.lambdastation.org/.

[11] TeraPaths: Configuring End-to-End Virtual Network Paths with QoS Guarantees. `https://www.racf.bnl.gov/terapaths/`.

[12] Circuit Switched High-speed End-to-End Transport Architecture (CHEETAH). `http://www.ece.virginia.edu/cheetah/`.

[13] D. Awduche and B. Jabbari. Internet traffic engineering using multi-protocol label switching (MPLS). *Computer Networks*, 40(1):111–129, 2002.

[14] Z. Yan, C. Tracy, and M. Veeraraghavan. A hybrid network traffic engineering system. In *Proc. of IEEE 13th High Performance Switching and Routing (HPSR) 2012*, Jun. 24-27 2012.

[15] C. Guok. Proposed QoS parameters for esnet5. Obtained from ESnet.

[16] OWAMP: One-Way Ping. http://www.internet2.edu/performance/owamp.

[17] M. Veeraraghavan and Z. Yan. Interaction between applications and the network. In *Optical Fiber Communication Conference and Exposition and the National Fiber Optic Engineers Conference (OFC/NFOEC) 2011*, pages 1–3, Mar. 2011.

[18] Z. Liu, M. Veeraraghavan, Z. Yan, C. Tracy, J. Tie, I. Foster, J. Dennis, J. Hick, Y. Li, and W. Yang. On using virtual circuits for GridFTP transfers. In *The International Conference for High Performance Computing, Networking, Storage and Analysis 2012 (SC 2012)*, pages 81:1–81:11, Nov. 10-16, 2012.

[19] T. Jin, C. Tracy, M. Veeraraghavan, and Z. Yan. Traffic engineering of high-rate large-sized flows. In *Proc. of IEEE 14th High Performance Switching and Routing (HPSR) 2013*, Taipei, Taiwan, Jul. 8-11 2013.

[20] Internet2. http://www.internet2.edu/.

[21] GEANT2. http://www.geant2.net/.

[22] JGN2plus. http://www.jgn.nict.go.jp/english/index.html.

[23] A. Lake, J. Vollbrecht, A. Brown, J. Zurawski, D. Robertson, M. Thompson, C. Guok, E. Chaniotakis, and T. Lehman. Inter-domain Controller (IDC) Protocol Specification, May 2008.

[24] On-Demand Secure Circuits and Advance Reservation System (OSCARS). http://www.es.net/OSCARS/docs/index.html.

[25] Simple Object Access Protocol (SOAP). http://www.w3.org/TR/2007/REC-soap12-part1-20070427/.

[26] Advanced Networking Initiative (ANI). http://www.er.doe.gov/ascr/ProgramDocuments/ARRA/ANIDesignDoc.pdf.

[27] UVa Alliance for Computational Science and Engineering. http://uvacse.virginia.edu/.

[28] The Packet Capture library (PCAP). http://www.tcpdump.org/pcap3_man.html.

[29] Unidata Local Data Manager (ULDM). http://www.unidata.ucar.edu/software/ldm/.

[30] BWdetail. http://www.ece.virginia.edu/cheetah/software/software.html.

[31] D. Farinacci, T. Li, S. Hanks, D. Meyer, and P. Traina. Generic Routing Encapsulation (GRE). RFC 2784, Mar. 2000.

[32] GT 5.2.1 GridFTP: User's Guide. http://www.globus.org/toolkit/docs/5.2/5.2.1/gridftp/user/gridftpUserGuide.pdf.

[33] M. Horowitz and S. Lunt. FTP Security Extensions. RFC 2228 (Proposed Standard), 1997.

[34] R. Vietzke. Internet2 Headroom Practice. `https://wiki.internet2.edu/confluence/download/attachments/17383/Internet2+Headroom+Practice+8-14-08.pdf\?version=1"`, Aug. 15 2008.

[35] ESnet Graphite. `https://stats.es.net/graphite/`.

[36] National Institute for Computational Sciences. http://www.nics.tennessee.edu.

[37] S. Sarvotham, R. Riedi, and R. Baraniuk. Connection-level analysis and modeling of nework traffic. In *ACM SIGCOMM Internet Measurement Workshop 2001*, pages 99–104, Nov. 2001.

[38] M. Crovella and M. Taqqu. Estimating the heavy tail index from scaling properties. *Methodology and Computing in Applied Probability*, 1:55–79, 1999.

[39] K. Papagiannaki, N. Taft, S. Bhattacharyya, P. Thiran, K. Salamatian, and C. Diot. A pragmatic definition of elephants in Internet backbone traffic. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*, pages 175–176, 2002.

[40] J. Wallerich, H. Dreger, A. Feldmann, B. Krishnamurthy, and W. Willinger. A methodology for studying persistency aspects of Internet flows. *ACM SIGCOMM Communication Review*, 35(2), 2005.

[41] A. Callado, C. Kamienski, G. Szabo, B. Gero, J. Kelner, S. Fernandes, and D. Sadok. A survey on Internet traffic identification. *Communications Surveys Tutorials, IEEE*, 11(3):37 –52, Jul. 2009.

[42] Y. Lu, M. Wang, B. Prabhakar, and F. Bonomi. ElephantTrap: A low cost device for identifying large flows. In *15th Annual IEEE Symposium on High-Performance Interconnects (HOTI 2007)*, pages 99–108, 2007.

[43] M. Kodialam, T. Lakshman, and S. Mohanty. Runs based traffic estimator (rate): a simple, memory efficient scheme for per-flow rate estimation. In *Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2004)*, volume 3, pages 1808–1818, 2004.

[44] F. Hao, M. Kodialam, T. Lakshman, and H. Zhang. Fast, memory-efficient traffic estimation by coincidence counting. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, volume 3, pages 2080–2090, 2005.

[45] M. Zadnik, M. Canini, A. Moore, D. Miller, and W. Li. Tracking elephant flows in Internet backbone traffic with an FPGA-based cache. In *International Conference on Field Programmable Logic and Applications (FPL 2009)*, pages 640–644, 2009.

[46] J. Paisley and J. Sventek. Real-time detection of grid bulk transfer traffic. In *Network Operations and Management Symposium (NOMS 2006)*, pages 66–72, Apr. 2006.

[47] N. Hohn and D. Veitch. Inverting sampled traffic. *IEEE/ACM Transactions on Networking*, 14(1):68–80, 2006.

[48] N. Kamiyama and T. Mori. Simple and accurate identification of high-rate flows by packet sampling. In *25th IEEE International Conference on Computer Communications (INFOCOM 2006)*, pages 1–13, 2006.

[49] T. Mori, M. Uchida, R. Kawahara, J. Pan, and S. Goto. Identifying elephant flows through periodically sampled packets. In *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement (IMC '04)*, pages 115–120, New York, NY, USA, 2004. ACM.

[50] Y. Zhang, B. Fang, and Y. Zhang. Identifying high-rate flows based on bayesian single sampling. In *2010 2nd International Conference on Computer Engineering and Technology (ICCET)*, volume 1, pages 370–374, 2010.

[51] N. Duffield, C. Lund, and M. Thorup. Estimating flow distributions from sampled flow statistics. *IEEE/ACM Transactions on Networking*, 13(5):933–946, 2005.

[52] T. Fioreze, L. Granville, A. Pras, A. Sperotto, and R. Sadre. Self-management of hybrid networks: Can we trust NetFlow data? In *IFIP/IEEE International Symposium on Integrated Network Management (IM '09)*, pages 577–584, 2009.

[53] N. Farrington, G. Porter, S. Radhakrishnan, H. Bazzaz, V. Subramanya, Y. Fainman, G. Papen, and A. Vahdat. Helios: a hybrid electrical/optical switch architecture for modular data centers. In *ACM SIGCOMM Computer Communication Review*, volume 40, pages 339–350. ACM, 2010.

[54] G. Wang, D. Andersen, M. Kaminsky, K. Papagiannaki, T. Ng, M. Kozuch, and M. Ryan. c-Through: Part-time optics in data centers. In *ACM SIGCOMM Computer Communication Review*, volume 40, pages 327–338. ACM, 2010.

[55] K. Chen, A. Singla, A. Singh, K. Ramachandran, L. Xu, Y. Zhang, X. Wen, and Y. Chen. OSA: An optical switching architecture for data center networks with unprecedented flexibility. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pages 18–18, 2012.

[56] N. Brownlee and K. Claffy. Understanding Internet traffic streams: dragonflies and tortoises. *Communications Magazine, IEEE*, 40(10):110 – 117, Oct. 2002.

[57] T. Nguyen and G. Armitage. A survey of techniques for Internet traffic classification using machine learning. *IEEE Communications Surveys and Tutorials*, 10(4):56–76, 2008.

[58] flow-tools. `http://www.splintered.net/sw/flow-tools/docs/flow-tools.html`.

[59] The R Project for Statistical Computing. `http://www.r-project.org/`.

[60] K. Thompson, G.J. Miller, and R. Wilder. Wide-area Internet traffic patterns and characteristics. *IEEE Network*, 11(6):10 –23, Nov./Dec. 1997.

[61] V. Cerf and R. Icahn. A protocol for packet network intercommunication. *ACM SIGCOMM Computer Communication Review*, 35(2):71–82, 2005.

[62] J. Kurose and K. Ross. Computer networks: A top down approach featuring the internet. Pearson Addison Wesley, 2010.

[63] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, Aug. 1993.

[64] D. Lin and R. Morris. Dynamics of random early detection. In *ACM SIGCOMM Computer Communication Review*, volume 27, pages 127–137, Oct. 1997.

[65] WRED. `http://www.cisco.com/en/US/docs/ios/11_2/feature/guide/wred_gs.html"`.

[66] R. Guérin, S. Kamat, V. Peris, and R. Rajan. Scalable QoS provision through buffer management. In *ACM SIGCOMM Computer Communication Review*, volume 28, Oct. 1998. pp. 29-40.

[67] R. Pan, L. Breslau, B. Prabhakar, and S. Shenker. Approximate fairness through differential dropping. *ACM SIGCOMM Computer Communication Review*, 33(2):23–39, Apr. 2003.

[68] K. Fall and S. Floyd. Simulation-based comparisons of Tahoe, Reno, and Sack TCP. *Comput. Commun. Rev.*, 1996.

[69] D. Leith and R. Shorten. H-TCP: TCP for high-speed and long-distance networks. In *Protocols for Fast Long Distance Networks Workshop (PFLDnet)*, Feb. 16-17, 2004.

[70] http://fasterdata.es.net/host-tuning/linux/.

[71] R. Jain, D. Chiu, and W. Hawe. A quantitative measure of fairness and discrimination for resource allocation in shared computer systems. *Technical Report TR-301, DEC Research*, Sep. 1984.