Characterization, Management, and Online Traffic Engineering of Heavy-Hitter Flows in Software Defined 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

Sourav Maji

August 2018

Approval Sheet

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

Sourav Maji

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

Malathi Veeraraghavan , Adviser

Joanne Bechta Dugan , Committee Chair

Alfred Weaver

Andreas Beling

Haiying Shen

Jordi Ros-Giralt

Accepted for the School of Engineering and Applied Science:

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

August 2018

Abstract

This dissertation describes advances made in the field of network management and high-performance networking. For network management, we designed and implemented an algorithm to reconstruct flows from NetFlow records collected at IP routers. We executed this system in a large Research and Education Network (REN), Energy Sciences Network (ESnet). We found that scientists move 100 GB to TB sized datasets at rates of 1 to 2.5 Gbps, and seldom use the network for transfers more than 10 hours. Our findings are useful for network planning and traffic engineering, and in improving user experience.

For high-performance networking we designed, implemented, and evaluated a high-speed Cheetah Flow Identification Network Function (CFINF). Two key features of the CFINF design are: (i) the ability to scale easily to higher levels of traffic utilization, and (ii) the flexibility for execution on general-purpose hardware. The system is designed with efficient data structures that are optimized to detect cheetah flows on a 10 Gb/s link that causes 1 M flows/min. With 10 CPU cores, CFINF can handle a 1-min 10-Gbps real Center for Applied Internet Data Analysis (CAIDA) traffic trace that contained 1.5M flows and 38M packets without loss. To improve efficiency, we ran CFINF in an 8-core configuration. However, with this configuration there were packet drops (max. rate of 0.036%).

To determine optimal values for CFINF parameters, we designed and implemented the Cheetah Flow Traffic Engineering System (CFTES). For an acceptable packet drop rate on a congested link, a high value of the rate threshold will result in few flows being redirected. A novel two-queue traffic redirection solution is presented that addresses the problem of packet reordering in TCP when a flow is redirected. We quantify a metric called burstiness and show that packet drop rate increases with increasing burstiness, even when the average

background rate is constant. The packet drop rate is also higher for high-RTT cheetah flows.

Finally, we propose a network service to diagnose throughput performance for large data transfers. Analysis of data-transfer logs, which were created by running experiments across Internet2, a US-wide REN, offer insights into the causes of poor performance.

Acknowledgments

I want to give my sincerest gratitude to my advisor, Professor Malathi Veeraraghavan, who has given me unconditional support throughout the years. She has guided me through this journey and made it possible to accomplish this achievement. I have learned many invaluable lessons from her passion for research and education, her curiosity for knowledge, and her great sense of responsibility for both her students and the projects in which she is involved. She has also helped me on making important decisions in my career and molded me into a better person.

I also want to thank Dr. Jordi Ros-Giralt, Alan Commike from Reservoir Labs and Chris Tracy from ESnet for their contributions and collaboration through multiple phases of this work. I truly appreciate the valuable feedback of Professor Naoaki Yamanaka and Professor Weiqiang Sun on my work.

I am immensely grateful to Professor Joanne Bechta Dugan, Professor Alfred Weaver, Professor Andreas Beling, and Professor Haiying Shen for serving on my proposal and defense committee, and for providing insight on how to improve the depth and breadth of this work.

I am grateful to Shrikant Ramamurthy for his guidance in my Ph.D life. He has been a benevolent mentor. I would like to thank other graduate students in our research group, Fatma, Xiaoyu, Molly, Yizhe, Yuanlong, Shuoshuo, Xiang, and Fabrice for always being helpful and cheering. I truly enjoyed the time we spent together in our lab and gatherings.

I wish to express my love and gratitude to my mom, dad, and my sister for their unwavering support and belief. My deepest gratitude to my dearest friends, Justin, Neel, and Tamal for their support and care. Special thanks to Siddhartha for motivating me to

embark on this journey.

Finally, this work was carried out under the sponsorship of NSF OCI-1038058, OCI-1127340, CNS-1116081, ACI-1340910, and DOE DE-SC0002350 and DE-SC0007341 grants. I thank the National Science Foundation and Department of Energy for funding this research.

Contents

List of Tables

List of Figures

List of Abbreviations

ASN	Autonomous System Number
API	Application Programming Interface
BDP	Bandwidth Delay Product
CAIDA	Center for Applied Internet Data Analysis
CFINF	Cheetah Flow Identification Network Function
CFTES	Cheetah Flow Traffic Engineering System
CFTMS	Cheetah Flow Throughput Monitoring System
DOE	Department of Energy
DNAC	Dynamic Network Acceleration for many-Core
DPI	Deep Packet Inspection
DYNES	Dynamic Network System
ESnet	Energy and Sciences Network
FS	Flow Set
HTCP	Hamilton TCP
ICMP	Internet Control Message Protocol
IP	Internet Protocol
IPFIX	IP Flow Information Export
ISP	Internet Service Provider
LRU	Least Recently Used
NAT	Network Address Translation
NFV	Network Function Virtualization
OSCARS	On-demand Secure Circuits and Advance Reservation System

NIC	Network Interface Card
O(1)	Order of 1
PPBP	Poisson Pareto Burst Process
PRCA	Performance Root Cause Analysis
RCA	Root Cause Analysis
REN	Research-and-Education Network
RIB	Routing Information Base
RQA	Router Query Agent
RTT	Round Trip Time
SDN	Software Defined Networking
SNMP	Simple Network Management Protocol
TAM	Traffic Analysis Module
TB	Tera Byte
TCP	Transmission Control Protocol
TEM	Traffic Engineering Module
TMS	Throughput Monitoring System

Chapter 1

Introduction

1.1 Background

The Internet's bandwidth has increased from its humble beginnings of a few Kbps in 1988 [1] to hundreds of Gbps in today's Research and Education Networks (RENs) (e.g, Internet2 [2] and ESNet [3]). Advances in the fields of fiber-optics, high-speed photo detectors, coherent communication technologies, and high-speed packet processors has contributed to an exponential growth of network capacity. Organizations upgrade their host hardware to leverage increased network capacity and applications use such host enhancements to fill the available network capacity. Along with the development of bandwidth demanding applications, the Internet is observing an exponential growth in the number of connected devices. The number of users and computers on the Internet has surpassed more than one billion [4] and that number continues to grow. Such a trend has resulted in widening the disparity of bandwidth used by applications. For example, web browsing and activities such as viewing emails require bandwidth from a few kbps, video streaming applications consume bandwidth in Mbps, whereas a large file transfer uses bandwidth in the range of Gbps. Large file transfer applications are massively bandwidth demanding and can rival the cumulative data rate from millions of connections. These applications generally transfer large amounts of data that can take the form of research datasets to virtual machine migration data. High bandwidth demanding applications unfairly compete for bandwidth and can trample over millions of low-rate connections. Helping the large number of small or low-rate connections

from being overrun by bandwidth intensive connections would result in improving quality of the Internet. This research explores the traffic characteristics of the Internet and proposes solutions for improving quality of service and network utilization.

With the advent of big-data in areas of research and industry, the frequency of large data movements has increased. Researchers execute experiments on large-scale systems that generate massive amounts of data. For example, the Conseil Européen pour la Recherche Nucléaire (CERNs) Large Hadron Collider generates 60 TB of physics data per year [5, 6], and the proposed Large Synoptic Survey Telescope is expected to produce 15 TB of data each day [7]. Scientific simulations executed in supercomputing facilities around the globe in the fields of High-Energy Physics, Genomics, and Climate Science generate multi-GB to TB size datasets. These large scientific datasets are moved across high-speed RENs between scientific laboratories and universities to enable collaboration. Data center facilities hosted by Google [8] and Microsoft [9] orchestrate networks to move large volumes of data across multiple data center sites.

Such large data movements result in large-sized flows called elephant flows. Elephant flows coexist with the vast number of small flows (also known as mice flows) generated from millions of connections. Statistically, the size of data flows in the Internet follows a Pareto distribution [10] with 80% of the data being transferred by just 20% of the flows. These elephant flows appear in the tail end of the Pareto distribution, and disrupt the small sized flows when sharing a link or path. Elephant flows can also exhibit a high-rate of data transfer. We define a high-rate elephant flow as a cheetah flow. Cheetah flows are detrimental to general Internet traffic. For example, cheetah flows have the potential to degrade service quality for real-time flows when data packets of a cheetah flow unfairly compete with packets from other flows for buffer space. Unfortunately, frequently aggressive variants of TCP like Hamilton TCP [11] are used to improve the throughput of cheetah flows, which cause unfairness in bandwidth distribution among smaller flows. The problem is compounded when more than one TCP stream is employed to transfer the large dataset.

Routers and switches in a path are equipped with buffers or queues that feed output ports to handle short periods of high traffic. The cheetah flows can generate long packet trains or bursts that have the potential of filling the switch port buffer. For small sized buffers,

this may result in packets being discarded from the cheetah flow as well as other traffic.. For large buffers, the effect increases the latency realized by packets of delay-sensitive flows, which can hurt a real-time connection. Addressing this requires the separation of cheetah flows from normal traffic.

1.2 Motivation

The motivation of this work stems from the existence of cheetah flows in RENs and commercial networks. A cheetah flow has detrimental effects on the general Internet traffic because it causes packet losses and increases the latency of delay sensitive flows. Therefore, network providers must be aware of the existence of cheetah flows in order to deploy solutions that provide better quality of service to their customers

Network data collected from an organization provides information on the characteristics of cheetah flows, such as flow size and flow duration. Knowledge about the network path traversed by such large flows is important in network capacity planning. Network paths that are frequently used for cheetah flows are prone to congestion events. Monitoring and studying the occurrence of cheetah flows will not only help providers plan for a network deployment that addresses current traffic needs, but also allows for growth in the future. Deployment costs are reduced as capacity is increased intelligently in segments of the network rather than increasing overall network capacity. We present a novel flow reconstruction algorithm that analyzes NetFlow records from Energy and Sciences Network (ESnet) to characterize cheetah flows.

Traffic engineering algorithms are used by most Internet Service Providers (ISPs) to operate network links at high utilization while maintaining high service quality. With the occurrence of cheetah flows in a network, the traffic engineering algorithms have to be informed about an arriving cheetah flow. Traffic engineering cheetah flows and redirecting such flows to separate links/queues not only improves the quality of service but also allows links to operate at higher utilization. Users of applications that generate cheetah flows have no incentive to notify the network of a large data transfer. Therefore, the network must employ traffic monitoring to detect a cheetah flow. For this reason, packet data is used to

identify a cheetah flow and then take actions that ameliorate the effects of a cheetah flow. Cheetah flow detection for the purpose of traffic engineering motivates the research and development of highly-scalable and simple algorithms, necessary for detecting a cheetah flow from millions of other flows. The rapid increase in compute power and reduction in the cost of processors have enabled network functions such as switching/routing to be performed in software. Network Function Virtualization is a new paradigm of executing networking software in general purpose servers. We present the design and implementation of a cheetah flow identification system that employs general purpose hardware and scales easily on increasing link capacity.

Networking research on cheetah flows lacks a common consensus of the definition of a cheetah flow based on flow metrics. We provide a method for setting a metric threshold that defines a cheetah flow. In our study of cheetah flows in ESnet, significant variation of data transfer throughput was observed. Variation in data transfer throughput results in uncertain job completion times when large datasets have to be moved as a part of a scientific workflow. Such variations occur when some transfers experience low-throughput, while other transfers enjoy high-throughput. Exploring the cause of low-throughput in data transfer will allow users to make amends to improve transfer throughput. The scope of this work includes characterizing cheetah flows, identifying cheetah flows for traffic engineering, and finally, investigating root causes for poor throughput performance in data transfers.

1.3 Problem statement

The research work explores the following three problems that arise from the presence of cheetah flows in a network. There are three main components to this work:

1. Analysis: The *first* problem was to identify the characteristics of cheetah flows in large networks. This led to the analysis of real-world network data to characterize network traffic. The solution was to analyze network monitoring data (i.e., data collected from the routers and switches in a large network) to characterize the size, duration, and rate of cheetah flows.

2. Implementation and experimentation: The *second* problem was to detect cheetah flows in real-time for the prospect of traffic engineering such flows. The work involved development of

novel algorithms that could be easily scaled and executed at high link speeds. The effects of a cheetah flow on small flows and delay sensitive flows are analyzed through experimentation.

3. Simulation: The *third* problem addresses a pragmatic approach to classify a cheetah flow through simulations. The trade-offs achieved in traffic engineering a cheetah flow were studied.

We used a combination of analytical, simulation, and experimental methods throughout this work.

1.4 Hypothesis formulation

As described in Section 1.2, to understand the nature of cheetah flows, we identified that ESnet provided network data that contained information required for flow reconstruction. In our previous work [12], we developed algorithms to identify cheetah flows from NetFlow records. The characteristics of these cheetah flows provided partial information on the datasets transferred. It is typical for more than one flow to be associated with a dataset transfer. Correspondingly our *first hypothesis* was that most data transfers occurs through multiple cheetah flows known as flowsets. Therefore, cheetah flows which were concurrent had to be a part of a transfer and these could be grouped to characterize the data transfer.

For our next contribution, the development and implementation of a cheetah flow detection system, we formulated a *second hypothesis*: Cheetah flows co-exist with millions of low-rate flows on a high speed link. In order to detect a cheetah flow in live network traffic we needed a relatively complex algorithm that could be executed on flexible/generic hardware and not require a dedicated ASIC for performance reasons. Hence our hypothesis was that an algorithm could be developed for detecting cheetah flows at high link rates, such that it could be easily scaled by adding more resources to monitor millions of flows arriving per minute.

A *third hypothesis* was formulated for finding a solution to traffic engineer cheetah flows. The routers and switches should have some form of rule based traffic steering mechanisms to separate cheetah flows from other traffic. There should be a method to enter flow identifiers for such rule based filtering. Such a method is applicable to recent router technology.

The routers should also support scheduling algorithms on their egress interfaces that are work-conserving in nature. We used this hypothesis to traffic engineer cheetah flow packets into a low-priority scheduling queue, and analyzed its benefits and trade-offs. Since a cheetah flow was redirected to a low-priority queue in a strict priority system, it would lead to queue starvation. Our hypothesis was that since cheetah flows are rare and background traffic load exhibits only periods of high utilization, the low priority queue will be served. Hence queue starvation will not arise.

1.5 Dissertation organization

This dissertation is organized into six chapters. The motivation of this work, relevant background work, and a summary of this work's key contributions are provided in this chapter.

In Chapter 2, the size, duration, and rate of cheetah flows are characterized. A novel method is provided to reconstruct cheetah flows from passive network measurement data (i.e., NetFlow records). The NetFlow records were collected from 66 routers in ESnet over four months. From these NetFlow, records we reconstructed cheetah flows and grouped single flows to characterize big-data movement. The statistics of data transfer sizes, duration, and rates were analyzed. These statistics provide useful information for network planning and traffic engineering of data transfers.

In Chapter 3, we investigate cheetah flows that cause increased packet delays and losses for other flows. We develop and implement a high-speed *Cheetah Flow Network Identification Function* (CFINF), which identifies cheetah flows using a new metric known as a *short-duration* flight rate. We provide a set of highly scalable algorithms that can detect cheetah flows among millions of flows traversing a high-speed link. These algorithms run on a x86-processor-based platform. Such design principles can be applied to the development of other high-speed packet processing network functions.

In Chapter 4, a simulation-based study is performed to gain insights into the choice of a cheetah flow threshold. A *Cheetah Flow Traffic Engineering System* (CFTES) is presented

that dynamically computes a cheetah flow threshold using information from the background traffic. The CFTES complements the development of the CFINF.

In Chapter 5, we propose an approach to identification of the throughput performance variation in large data transfers. Through a set of experiments, we show the feasibility of a throughput monitoring service, and identify the causes of low-throughput.

Finally, this work's contributions are summarized, future work is discussed, and the dissertation is concluded in Chapter 6.

1.6 Key contributions

The key contributions are as follows:

1. A measurement-based study to characterize big-data movement in a REN is performed. A flow reconstruction algorithm from prior work [12] is used to identify cheetah flows from NetFlow records. Parallel TCP connections, where each connection is an cheetah flow, are used for large dataset transfers to increase throughput. Our key contribution is to provide a technique for aggregating parallel cheetah flows into flowsets to identify a data transfer. Analysis of such data transfers are conducted to understand the data movement in a large REN such as ESnet. Statistics of the size, duration, and rate of such data movements are characterized, which provides valuable information for network planning and traffic engineering. This work has been published in *IEEE European Conference on Networks and Communications* [13].

2. We present a novel cheetah flow identification algorithm that detects cheetah flows online from millions of flows traversing a network link every minute. The algorithms are executed on x86 processors offering flexibility in programming, which is a fundamental tenet in Network Function Virtualization for packet processing. Efficient data structures and flow-based load balancing address the requirements of high-speed packet processing and scalability, respectively. We develop and implement a cheetah flow identification system and evaluate the accuracy of the system with respect to compute power. The purpose of detecting a cheetah flow is to separate it from other traffic. A traffic engineering system is proposed that involves cheetah flow detection and subsequent interaction with an Software

Defined Network (SDN) controller and a switch to redirect the cheetah flow. This work has been published in *IEEE Conference on Network Function Virtualization and Software Defined Networks* [14]. This work led us to co-author another paper with our collaborators, and has been published in *IEEE High Performance Extreme Computing Conference* [15].

3. An interesting problem of determining a cheetah flow threshold was discovered in our implementation of the CFINF. Since link utilization characteristics vary between networks and also in time, it is clear that automated mechanisms are necessary to set the cheetah flow threshold. Through in-depth simulation we describe a solution of computing a cheetah flow threshold using the nature of the background traffic. An effective traffic engineering policy of cheetah flow redirection is evaluated to achieve high cheetah flow throughput for an acceptable percentage of packet drops. This work was accepted in *IEEE European Conference on Networks and Communications* [16].

Chapter 2

A measurement-based study of big-data movement

2.1 Introduction

In this work, we analyzed NetFlow records collected in an operational research-and-education network across which large scientific datasets are moved routinely, reconstruct individual cheetah flows from the NetFlow records, and assemble parallel flowsets from cheetah flows. Bulk of this material is presented from our published work *A measurement-based study to characterize big-data movement* [13]

Our *findings* are as follows. The top 1% of flowset sizes were in the hundreds of GBs to low TBs range, 95% of flowsets had rates less than 2.5 Gbps, and 99% of flowsets had durations shorter than 4 hours. Median flowset rate increases and rate variance decreases with increasing number of per-flowset component flows. Such findings are useful for network planning, traffic engineering, and for improving user performance, since large dataset transfers are among the most demanding of network applications.

Researchers use supercomputing centers to execute large-scale, highly parallelized simulations and big-data analytics, in fields such as High-Energy Physics, Genomics, and Climate Science. Multi-GB to TB datasets are generated from these applications. Research-and-Education Networks (RENs), which connect national laboratories (labs) and universities

where most scientific researchers are based, have high-rate links to enable fast transfers of these datasets.

Throughput is an important measure as it determines the time needed for the dataset transfer. Even low loss rates, on the order of 0.005%, can have a significant negative impact on throughput on high Bandwidth-Delay Product (BDP) paths because of TCP's congestion control algorithm [17]. To counter this effect and achieve high throughput, file-transfer applications used by the scientific community have file segmentation and reassembly features to use parallel TCP connections, and concurrency features to run multiple application threads on multi-core end hosts [18].

Problem statement Determine the characteristics of big-data movement from passive data measurements collected at provider IP routers. Routers are configured to save flow records using tools such as NetFlow. Since large datasets are often transferred using parallel (concurrent) TCP connections, records about multiple flows must be aggregated in order to accurately characterize the size, rate, and duration of such transfers.

Solution approach We define a *parallel flowset* as a group of flows between a source host and a destination host that occur concurrently. A method for reconstructing parallel flowsets from NetFlow records is designed and applied to data collected by a US-wide backbone network provider, ESnet [3]. In prior work [19], we developed a method for reconstructing single cheetah flows from NetFlow records, and presented characteristics of cheetah flows observed at four ESnet routers for a 7-month period in 2011. In this work, we developed and implemented a new algorithm for reconstructing parallel flowsets, each consisting of multiple flows, from the single cheetah-flow characteristics obtained from prior work. We then executed our software on NetFlow records collected from June-Sept. 2014 at all 66 ESnet routers. We present the size, rate and duration characteristics of these parallel flowsets as being representative of large scientific dataset movement.

Novelty To the best of our knowledge, we are the first researchers to characterize large datasets that are moved with parallel TCP flows. Other studies have characterized individual cheetah flows [20, 21], but have not reconstructed parallel flowsets. Given the scientific community's use of file-transfer applications that use parallel connections and concurrent

threads, reconstruction of parallel flowsets is essential for gaining an understanding of big-data movement.

Contributions Our method for reconstructing parallel flowsets, and the characteristics of large scientific dataset transfers, are the primary contributions of this work. ESnet offers high-speed network access to US Department of Energy (DOE) labs, some of which operate large scientific supercomputing centers. Therefore, ESnet is one of the best networks in the US for observation of scientific big-data movement.

Impact Several interesting phenomena were observed in this flowset reconstruction study that show how this work can impact network planning, traffic engineering, and help improve user experience. *First*, all parallel flowsets observed in/out of five major DOE labs over a 4-month period, were shorter than 10 hours, except three flowsets, which were of lengths 13 hours, 16 hours, and 33 hours. This observation is consistent with anecdotal information that scientists use overnight shipping services when the network is likely to be slower. *Second*, in RENs, while aggregate traffic volumes are typically low, the need to support large-dataset transfers is the primary reason for upgrading link speeds to avoid bottlenecks on end-to-end paths. Our software offers administrators a tool for understanding their users' big-data movement needs for future network and Data Transfer Node (DTN) capacity planning. *Third*, throughput differences observed between short-RTT paths and long-RTT paths makes the case for using rate-guaranteed dynamic Layer-2 services [22], such as those offered by Internet2 and ESnet, to avoid packet losses. *Fourth*, our work is useful for traffic engineering applications, such as offloading cheetah flows to optical circuits [23]. *Fifth*, user experience can be improved by using our software to identify low-throughput paths for initiating diagnostics. *Finally*, in a broader context, commercial cloud computing providers can use this technique for learning the networking needs of their large-usage customers for planning, traffic engineering, and trouble-shooting.

Section 2.2 describes our methodology for reconstructing parallel flowsets. Characteristics of big-data movement across ESnet are presented in Section 2.3. Section 2.4 reviews related work. Section 2.5 presents our conclusions.

2.2 Solution Approach

There are four steps in our solution: (i) reconstruct cheetah flows from NetFlow records, (ii) determine the set of cheetah flows to and from each lab from the per-router cheetah-flow data, (iii) reconstruct parallel flowsets from the cheetah-flows data, and (iv) characterize the size, rate (throughput), and duration of flowsets. Results for the last step are provided in the next section.

Reconstruct cheetah flows from NetFlow records We define cheetah flows as flows that send more than 1 GB in a 1-minute (min) period within their lifetimes based on our prior work [19]. NetFlow records, collected in IP routers with or without packet sampling, provide packet counts, byte counts, timestamps, etc., on a per-flow basis. Fioreze et al. showed that for cheetah flows, we can trust the size numbers determined from NetFlow records even with 1-in-1000 packet sampling [20]. An active timeout interval parameter configured in NetFlow determines the maximum length of each NetFlow record. Therefore, cheetah flows that last longer than the active timeout interval should be reconstructed by concatenating information from multiple records.

Determine per-lab sets of cheetah flows NetFlow records include source and destination Autonomous System Numbers (ASNs), which is useful information as it provides us the end points of flows (IP addresses were anonymized for privacy reasons, and only REN traffic was included). The ASNs were saved in the per-router cheetah-flow data, and used to aggregate sets of cheetah flows to and from each lab.

Reconstructing parallel flowsets from flows Flow k is represented by the following tuple:

$$\{s_k, d_k, x_k, y_k, z_k, f_k, l_k, o_k, a_k, b_k\} \tag{2.1}$$

where s_k: source IP address, d_k: destination IP address, x_k: source port number, y_k: destination port number, z_k: protocol number, f_k: UTC timestamp of the first packet of the flow, l_k: UTC timestamp of the last packet of the flow, o_k: cumulative number of bytes, a_k: source ASN, and b_k: destination ASN.

Using the notation in Table 2.1, the main steps of the algorithm are listed below:

Table 2.1: Notation (© 2015 IEEE)

i	per-day index
j	flowset-identifier (ID) index
k	cheetah-flow index
p	parallel FlowSet (FS) index
ω_j	flowset identifier
τ	maximum time gap between start times of flows in a FS
$\mathbf{F}_i$	set of cheetah flows
$\mathbf{W}_i$	set of unique flowset IDs amongst all flows $k \in \mathbf{F}_i$
$\mathbf{A}_{ij}$	set of cheetah flows that share a flowset ID; $\omega_j \in \mathbf{W}_i$
$\mathbf{B}_{ijp}$	set of cheetah flows in a single parallel flowset p; $\mathbf{B}_{ijp} \subseteq \mathbf{A}_{ij}$
N_{ijp}	Number of component flows in parallel flowset
S_{ijp}	Size of parallel flowset
D_{ijp}	Duration of parallel flowset
R_{ijp}	Rate (throughput) of parallel flowset

No. of FSs with different N_{ijp} values	Lab 1	Lab 2	Lab 3	Lab 4	Lab 5
2-flow FSs	132 | 1364	1989 | 5669	172 | 974	112 | 835	57 | 2155
4-flow FSs	67 | 226	82 | 237	60 | 215	35 | 3010	14 | 5399
8-flow FSs	526 | 41	2 | 33	46 | 10	6 | 651	3 | 1268
16-flow FSs	0 | 1	0 |10	0 | 3	3 | 14	8 | 46
FSs with other values for N_{ijp}	98 | 87	132 | 622	321 | 229	24 | 131	77 | 1649
Total no. of FSs	823 | 1719	2205 | 6571	427 | 1431	180 | 4641	159 | 10517
No. of DHs | No. of SHs of FSs	66 | 46	883 | 264	245 | 288	56 | 52	194 | 56

Table 2.2: Parallel FlowSets (FSs) observed during June-Sept. 2014; To-Lab | From-Lab (except last row) (© 2015 IEEE)

1. From each day's set of cheetah flows, $\mathbf{F}_i$, determine the set of unique flowset identifiers, $\mathbf{W}_i$.

2. Divide each set $\mathbf{F}_i$ into mutually disjoint sets $\mathbf{A}_{ij}$, each of which consists of all cheetah flows in day i that share a flowset identifier $\omega_j \in \mathbf{W}_i$.

3. Divide each set $\mathbf{A}_{ij}$ into mutually disjoint subsets $\mathbf{B}_{ijp}$, each of which consists of flows in the same parallel flowset p.

4. Compute size, duration, and rate of each flowset $p \in \mathbf{B}_{ijp}, \forall \omega_j \in \mathbf{W}_i$.

5. Concatenate flowsets that cross midnight boundaries.

For each day's set of cheetah flows, $\mathbf{F}_i$, the *first step* is to find the corresponding set of unique flowset identifiers, $\mathbf{W}_i$. The flowset identifier ω_j is defined as the combination of source IP address and destination IP address.

The *second step* divides the set of cheetah flows $\mathbf{F}_i$ into mutually disjoint sets $\mathbf{A}_{ij}$, where the number of subsets is $|\mathbf{W}_i|$. Flow $k \in \mathbf{A}_{ij}$ if $\{s_k, d_k\} = \omega_j$, where $\omega_j \in \mathbf{W}_i$.

In the *third step*, first order the flows in each set $\mathbf{A}_{ij}$ by sorting on the first-packet timestamp (f_k in (2.1)) from the earliest timestamp to the latest timestamp. The ordered set of flows are $k_1, k_2, \cdots, k_{|\mathbf{A}_{ij}|}$. Next, divide each set $\mathbf{A}_{ij}$ into disjoint subsets $\mathbf{B}_{ijp}$, such that a consecutive set of flows $\{k_n, k_{n+1}, \cdots, k_{n+u}\} \in \mathbf{B}_{ijp}$ iff

$$\begin{aligned} f_{k_{n+u}} - f_{k_n} &\leq \tau \\ f_{k_n} - f_{k_{n-1}} &> \tau \qquad \text{for } n \neq 1 \\ f_{k_{n+u+1}} - f_{k_{n+u}} &> \tau \quad \text{for } n+u \neq |\mathbf{A}_{ij}| \end{aligned} \tag{2.2}$$

In other words, the first sampled-packet timestamps of all flows in a flowset should be within a time interval of length τ. When a file-transfer application that creates multiple parallel flows is initiated, data transfer should start almost simultaneously on all flows. A time gap τ is permitted because even if all flows of a flowset start within the same second, there can be gaps in the first-packet timestamps of the individual flows due to NetFlow packet sampling.

The *fourth step* is to determine the size, rate and duration of each flowset. The flowset size is simply the sum of the sizes of its component flows. Duration is computed by finding the time difference between the last-packet timestamp of the last flow and the first-packet timestamp of the first flow in each parallel flowset. Rate (throughput) is computed by dividing flowset size by flowset duration.

A parallel flowset p, that occurred on day i with flowset-ID ω_j, and consisting of flows $\{k_n \cdots, k_{n+u}\} \in \mathbf{B}_{ijp}$, is characterized by

$$\begin{aligned} \text{Size } S_{ijp} &= \sum_{k \in \mathbf{B}_{ijp}} o_k \\ \text{Duration } D_{ijp} &= l_{k_{n+u}} - f_{k_n} \\ \text{Rate } R_{ijp} &= \frac{S_{ijp}}{D_{ijp}} \end{aligned} \tag{2.3}$$

Finally, the last timestamps l_k of the flows constituting each flowset are checked. If these values are within one-minute before midnight for a particular flowset, then the flowset identifier is used to check the next-day's flowsets to determine whether the flowset crossed the midnight boundary. If so, the two flowsets are concatenated.

2.3 Data transfer characterization in ESnet

Our starting dataset consisted of NetFlow records collected from all ESnet routers over a 4-month time period, June-Sept. 2014. The method described in Section 2.3 (the time gap τ was set to 10 sec) was applied to these NetFlow records, and one set of parallel FlowSets (FSs) was reconstructed for each ESnet customer site (DOE labs and supercomputing centers) with a public ASN. Results are presented for sites with the largest number of parallel FSs, but site names have been anonymized as Lab 1, Lab 2, etc., due to privacy considerations.

Section 2.3.1 provides numbers of parallel FSs, and numbers of source and destination hosts that engage in large-dataset transfers. Section 2.3.2 provides statistics on the sizes, rates, and durations of datasets moved with parallel FSs. Section 2.3.3 describes an analysis of rates of FSs between two pairs of source-destination labs, one across a short-RTT path, and another across a long-RTT path.

2.3.1 Numbers of parallel FlowSets (FSs)

Table 2.2 lists the numbers of parallel FSs observed to and from each lab for different values of N_{ijp}, the number of component flows. For example, there are 132 2-flow FSs destined to hosts in Lab 1, and 1364 2-flow FSs sourced from hosts in Lab 1. Popular values for N_{ijp}

Sizes in GiB					
Measure	Lab 1	Lab 2	Lab 3	Lab 4	Lab 5
Median	31.8 | 24.7	3.2 | 4.1	14.9 | 15.4	16.8 | 105.2	9.8 | 19.8
Mean	80.4 | 69.1	3.9 | 5	70.1 | 50.5	86.9 | 206.3	536.4 | 21.1
99%	435.7 | 479.3	13.3 | 23	389.5 | 512.5	520.9 | 1608	8524.6 | 80.4
Max.	1128.5 | 1306	26.4 | 112.5	704.5 | 2469.1	651.9 | 26521.2	26521.2 | 1758.6
Rates in Gb/s					
Median	1.7 | 0.5	0.5 | 0.5	0.4 | 0.4	0.5 | 1.2	1.4 | 0.8
Mean	1.5 | 0.7	0.6 | 0.6	0.5 | 0.5	1 | 1.4	1.4 | 1
95%	2.3 | 2.3	1.3 | 1.2	0.9 | 0.9	2.3 | 2.2	2.0 | 2.4
Max.	7.4 | 5.4	4.3 | 9.7	2.1 | 3	7.4 | 7	2.7 | 6.9
Durations in sec except for the Max. row					
Median	149.1 | 241.7	53.8 | 70.7	251.4 | 233.4	348.8 | 736.7	84.4 | 179.1
Mean	1098 | 954.9	76.5 | 87.6	545.5 | 732.6	1158.5 | 1391.8	4297 | 256.6
99%	11854 | 13664.8	321.9 | 400.8	6799.2 | 11524.6	8936.6 | 11130.1	5514 | 1602.4
Max. in hr	6.6 | 6.9	0.3 | 1.1	4.4 | 13.8	3.2 | 33	33 | 16

Table 2.3: Statistics for parallel FSs observed during June-Sept. 2014; To-Lab | From-Lab (© 2015 IEEE)

No. of FSs	Lab 1	Lab 2	Lab 3	Lab 4	Lab 5
Single flows	9165 | 13041	14132 | 34268	6293 | 51214	3483 | 55745	2604 | 60027

Table 2.4: Single flows (FSs) observed during June-Sept. 2014; To-Lab | From-Lab

appear to be 2, 4, 8, and 16. In general, the number of parallel FSs sourced at a lab host was greater than the number of parallel FSs destined to a lab host. This is because many external scientists use DOE lab supercomputers to run their simulations/analytics, and then download relevant datasets.

The last row of Table 2.2 shows the number of hosts in each lab that were destinations of FSs and the number of the hosts that were sources of FSs.

We also studied the statistics of size, duration, and rate of flows not grouped into flowsets. These flows are called single flows. Table 2.4 lists the numbers of single flows observed to and from each lab. Again, the number of flows emanating from a lab (i.e., downloads) is greater than the number of flows sending data to a lab (i.e., uploads). Similar to the statistics of parallel flowsets, the distribution of size, duration, and rate of single flows are also right skewed with mean values larger than the median values.

2.3.2 Size, rate, and duration characteristics

Table 2.3 shows independent statistics for the size, rate, and duration characteristics of the FSs to/from the same five labs as those shown in Table 2.2. Mean values are typically larger than median values because sizes, rates, and durations are all right skewed. The minimum size of FSs is 2 GB, given that an FS has at least two flows, and our definition of cheetah flows requires each component flow of an FS to send at least 1 GB.

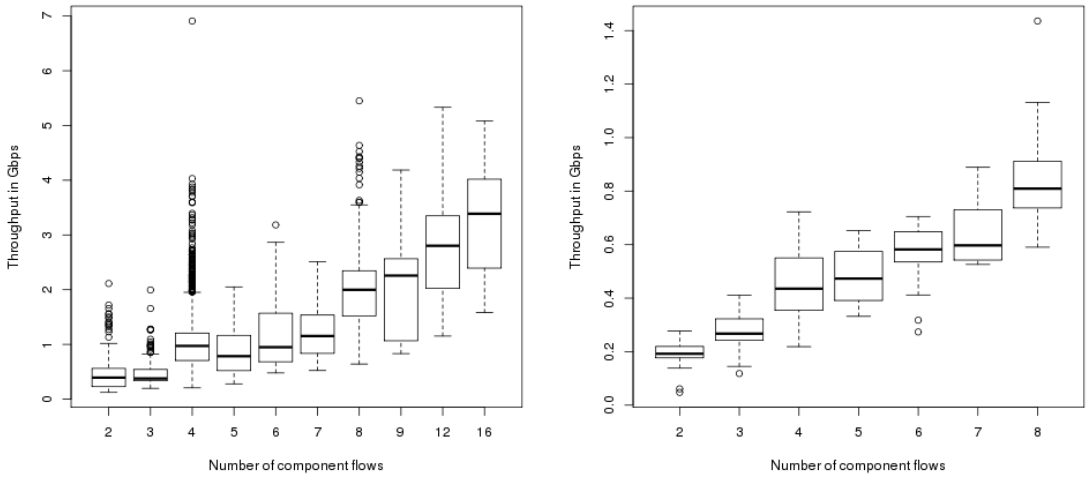
The largest transfer observed was from Lab 5 to Lab 4, and was of size 26.5 TiB (1 Tebibyte = 2^{40} bytes). This FS consisted of 16 parallel flows, lasted 33 hours (it was the longest FS as seen in the last row of Table 2.3), and transferred data at a rate of 1.93 Gbps. The next maximum-sized FS observed across the five labs was from Lab 3. It moved 2469.1 GiB, had 4 parallel flows, lasted 13.8 hrs, and transferred data at a rate of 407 Mbps. Only a handful of FSs were larger than 1 TiB. Considering that mean rates in some labs are only 0.5 Gbps range, a 1 TiB transfer could take close to 5 hours.

The sizes of datasets transferred from labs are generally greater than sizes of datasets transferred to labs for Lab 2, Lab 3 and Lab 4. This is because these Labs have supercomputing facilities that are used by other DOE lab and university-based scientists.

The highest-rate FS (a 16-flow parallel FS) at 9.7 Gbps, transferred 112.53 GiB, and lasted for 1.66 mins. The next two highest-rate FSs were both around 7.4 Gbps. One had 4 parallel flows, lasted 2.1 mins and transferred 108 GiB, while the other had 8 parallel flows, lasted 5.3 mins, and moved 275.14 GiB. These high rates are outliers. As seen in Table 2.3, 95% of FS rates were 2.3 Gbps or lower, which are typical disk I/O rates [24].

Interestingly as noted in Section 2.1, duration appears to be the deciding factor for whether users choose the network or express shipping services. Given that overnight delivery services are available, we observed only a few FSs that lasted more than 10 hours. Given disk access rate are in the range 2.5-3 Gbps for sequential file-systems, this combination of rate and duration constraints limits the size of datasets moved across the network. As mentioned earlier, the longest transfer lasted 33 hours (which was clearly an outlier), but the second longest transfer lasted 16 hours, had 2 parallel flows, and moved 1.54 TiB, and experienced a throughput of only 240 Mbps. This transfer occurred on a high-RTT path.

2.3.3 Comparison of FS rates on same paths



(a) Short-RTT path between two DOE labs (b) Long-RTT path between two DOE labs

Figure 2.1: Boxplots showing variability in FS rates (© 2015 IEEE)

While the previous section discussed rates of all FSs into and out of a lab, in this section, we describe our analysis of variability in rate for FSs between the same source and destination ASNs. While the source and destination hosts may not be the same, the bottleneck link rate and RTT of all compared FSs are the same.

Fig. 2.1 shows boxplots for the rates of FSs, split by the number of component flows, between two source-destination ASN pairs.

First, we observe that median rate increases with the number of component flows, making a case for using parallel TCP connections. The increased throughput is likely due to faster recovery from packet losses, and higher DTN-resource usage by concurrent server threads. Second, we observe that at least on this compared pair of paths, rates are higher on the short-RTT path than on the long-RTT path. But we refrain from drawing broad conclusions about packet loss rates as we have not yet conducted a systematic study of all FS rates to characterize dependence on RTT.

There are more than 10 FSs for each number of component flows on the two paths used in this study. For the short-RTT path, we found an interesting decrease in the ratio of inter-quartile range (3rd quartile - 1st quartile) to the median. The numbers of 2-, 3-, 4-,

and 8-component FSs were 214, 224, 4572, and 1139, respectively. The numbers of FSs were smaller, though all were larger than 10, for the other numbers of component flows. The IQR-to-median ratio (a measure of variance) values for the 2-, 3-, 4-, and 8-component FSs were 0.84, 0.54, 0.51, 0.41, respectively. This observation shows that rate variance is smaller for FSs with larger numbers of component flows. The sample sizes for the long-RTT path FSs were smaller in size, ranging from 18 to 43.

2.4 Related work

General-purpose Internet measurements papers [25, 26] analyze network traffic for various purposes, such as the identification of top users of network bandwidth, e.g., video flows, peer-to-peer flows, or Web traffic. Elephant flows are characterized from traffic traces collected by CAIDA [27] in a recent paper [21]. The definition used for elephant flows in this paper, as flows whose size exceeds the mean plus three times the standard deviation of sizes of all flows, is different from ours, which sets fixed thresholds for size and rate. Sampled NetFlow data can be used to estimate flow sizes only when such flows have a high rate. From previous experiments we verified that flow sizes can be estimated with a good level of accuracy when we define a cheetah flow as one that transfers 1 GB of data in 1 min. Therefore, our analysis is of a certain class of high-rate elephants.

GridFTP [28] has features that support parallel TCP connections and concurrent server threads to enable fast data movement. The use of parallel TCP connections allows for quicker recovery from packet losses, and the use of multiple GridFTP server threads in multi-core DTNs allows separate files from a dataset to be moved simultaneously. Therefore, while characterizing individual dataset transfers, we should reconstruct parallel flowsets rather than just single flows. The bbftp [29] application also supports such features. Our work reconstructs parallel flowsets regardless of the application.

In this work, we reconstructed parallel flowsets from NetFlow records collected by ESnet routers. Other industry standards for flow-level monitoring include IPFIX [30] and OpenFlow statistics collection [31]. As our work shows the value of collecting statistics about cheetah

flows for network planning and traffic engineering, it could inform the design of new network monitoring techniques to better incorporate features targeted for cheetah flows.

2.5 Conclusions

To characterize the size, rate, and duration of large dataset transfers, this work identified the importance of reconstructing parallel flowsets from NetFlow records collected at IP routers. Since high-performance file-transfer applications use parallel TCP connections for large dataset transfers, characterizing single elephant flows alone will not provide network administrators and users an accurate picture of big-data movement. Upon applying our method to NetFlow records from a large research-and-education network, we found that today's scientists move 100 GB to TB sized datasets at rates of 1 to 2.5 Gbps, and seldom use the network for transfers more than 10 hours. We found that the median rate of flowsets increases and rate variance decreases with the number of per-FS component flows.

In the next two chapters, we explore a class of elephant flows that has detrimental effect on real-time traffic. We present a traffic engineering solution, and describe its deployment in an enterprise network.

Chapter 3

A High-Speed Cheetah Flow Identification Network Function (CFINF)

3.1 Introduction

This chapter presents the design, implementation, and evaluation of a Cheetah Flow Traffic Engineering System. As mentioned in Section 1.2 cheetah flows are high-rate flows that can cause increased packet delays and losses in other flows. Therefore, to improve quality of service and increase link utilization cheetah flows must be isolated from the general traffic. The first step towards isolating a cheetah flow is to detect the cheetah flow. We present the design of a Cheetah Flow Traffic Engineering System (CFTES) that is composed of a Cheetah Flow Identification Network Function (CFINF) and an SDN controller. In the chapter, we describe the development and implementation of a High-Speed Cheetah Flow Network Identification Function. We provide a set of highly scalable algorithms that can detect a cheetah flow from millions of flows traversing a high-speed link per minute. We used a commodity x86 processor to execute these algorithms for a flexible design. A significant portion of this chapter is an excerpt from our published work *A high-speed cheetah flow identification network function (CFINF)* [14]

Cheetah flows are high-rate flows that can fill up buffers associated with switch/router ports, which can have detrimental effects on other flows. For routers with large buffers [32], cheetah flows can increase packet delays for real-time flows. For switches with small buffers, as is the case with some enterprise and datacenter switches [33], cheetah flows can cause packet losses. This is especially true for TCP flows across large Bandwidth-Delay Product (BDP) paths. To achieve high flow throughput, the TCP-sender and receiver buffer sizes need to be larger than BDP. For example, a TCP flow on a 1-Gbps, 100-ms end-to-end path has a BDP of 12.5 MB, which is larger than buffer sizes of low-end switches. When the TCP-sender and receiver buffers are larger than the switch buffer size, the TCP sender could create high-speed flights of packets that can cause buffer overflows.

To avoid the detrimental effects of cheetah flows (shortened to "cheetahs") on other traffic, we propose a new network function called Cheetah Flow Identification Network Function (CFINF), and a Cheetah Flow Traffic Engineering System (CFTES) to redirect identified cheetahs in real-time to a separate queue for isolation from other traffic. We define a new term called *T-duration flight* to refer to a set of packets of a flow that arrive within a duration of length T[1], and the corresponding term *flight-rate*, which is determined by dividing the cumulative size of packets in the flight by T. The CFINF stores per-flow information to determine flight-rates, which are then compared to a set rate threshold R. A flow that has even a single flight whose rate exceeds R is classified as a cheetah. The 5-tuple identifier of a flow classified as a cheetah is immediately sent to the CFTES, which in turn communicates with the Software Defined Network (SDN) controller to set filter rules in the router for redirection of cheetah-flow packets to a separate queue. The rate threshold R is computed periodically based on monitored background traffic characteristics, and modified if needed.

We leverage the flexibility of the Network Function Virtualization (NFV) paradigm to implement CFINF. In the NFV paradigm, data-plane functions are implemented in software and executed on commodity hardware [36] in contrast to traditional data-plane function implementations that use dedicated hardware. For example, the NetFlow/IPFIX [30]

[1]Our usage of the term "flight" is different from prior usage where the inter-arrival times between packets are considered in determining which packets belong to the same flight [34, 35].

implementations built into dedicated-hardware based routers are not scalable enough to support the type of short-duration flight-rate monitoring required for CFINF.

Prior work focused on identifying elephant (defined as high-rate large-sized) flows [37–42]. These papers offer algorithms to identify flows with the highest rates (relative measure) not the specific flows whose absolute flow rates exceed a threshold. But if the absolute rate of a flow is not high enough to fill the headroom in a link, the flow is not a threat to other traffic, and hence should not be redirected. Furthermore, our method does not attempt to determine whether a flow is an elephant, i.e., whether it is large-sized. Instead the focus of our algorithm is to determine whether a flow is a cheetah as soon as possible so that it can be isolated from other non-cheetah flows.

In designing CFINF, the main challenge lies in identifying cheetahs in real-time on a high-speed link that carries millions of simultaneous flows. We implemented and evaluated the CFINF algorithms on a high-performance, multi-core x86-processor based commodity server [43].

Our main *findings* are as follows: (i) When our CFINF was run on a 10-core configuration, it could handle a packet trace collected live on a 10-Gbps link at San Jose by the Center for Applied Internet Data Analysis (CAIDA) [44] without dropping any packets. This trace had 1.5M flows and 38M packets. (ii) Flow-filtering based on maximum packet length offers an effective means for lowering costs without sacrificing performance. For example, with an 8-core configuration, the median number of packets dropped by CFINF decreased from 17K to 2.8K when the flow-filter packet-length threshold L was set to 600B (i.e., flows that did not have any packets longer than L were dropped without processing), while accuracy did not suffer much (the median number of identified cheetahs dropped from 2354 to 2116).

The rest of the chapter is organized as follows. The CFINF algorithms and CFTES are described in Section 3.2. Section 3.3 describes a data analysis of several 1-min CAIDA traces to determine whether packet-length based filtering would be beneficial. Section 3.4 presents results of our CFINF evaluation on the commodity x86 server. Experimental studies of cheetah flows are presented in Section 3.5 to shed light on the types of cheetahs. Section 3.6 reviews related work. Our conclusions are presented in Section 3.7.

3.2 Cheetah Flow Traffic Engineering System

Section 3.2.1 defines cheetah flows, Section 3.2.2 describes an example deployment of CFINF and CFTES, and Section 3.2.3 explains the CFINF algorithms.

3.2.1 Definition

A flow f is classified as a cheetah by CFINF if

$$\frac{\sum_{i=1}^{i=n} s_i}{T} > R \tag{3.1}$$

where R is the configured rate threshold, T is the flight duration, s_i is the size of packet i of the flight under observation, and n is the number of packets in the flight under observation. If $(\tau_1, \tau_2, \cdots, \tau_n)$ represent the packet-arrival timestamps of the n packets in the flight, then $\tau_n - \tau_1 \geq T$ and $\tau_{n-1} - \tau_1 < T$.

The rate threshold R is set as follows:

$$R = \alpha_{bg}(C - \overline{r}_{bg}) \tag{3.2}$$

where α_{bg} is a function of the variability in background traffic rate, C is link capacity, and $\overline{r}_{bg}$ is the average background traffic rate, with the averaging interval set to the same duration T used for computing flight rates. The operator-selected value of α_{bg} is inversely proportional to the variability in background traffic rate. If the background traffic rate is almost constant, then the R threshold for declaring a flow as a cheetah can be close to the headroom in link utilization.

3.2.2 CFTES

Fig. 4.1 shows a deployment scenario for CFINF and CFTES. All packets sent by router R1 on its link to R2 are mirrored to the CFINF. Utilization of customer-to-provider links are typically high, at least during peak hours, and therefore deploying a CFINF/CFTES to monitor/redirect cheetahs on such links could improve performance for all flows.

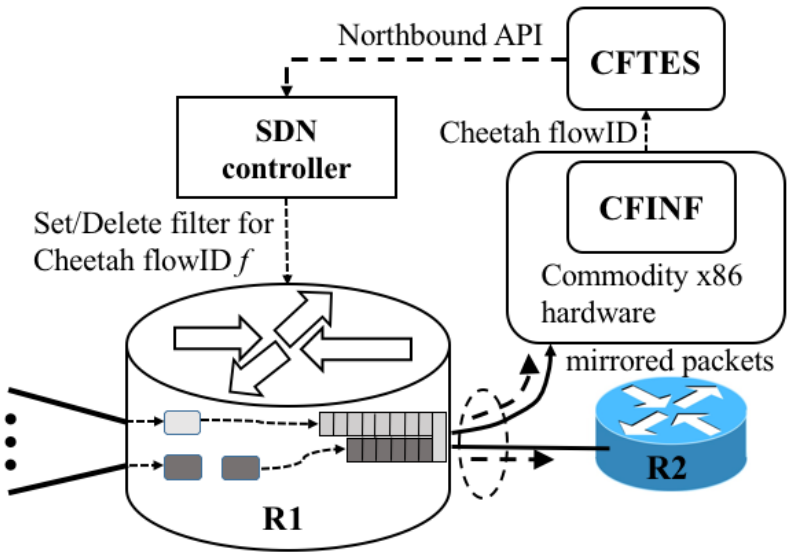
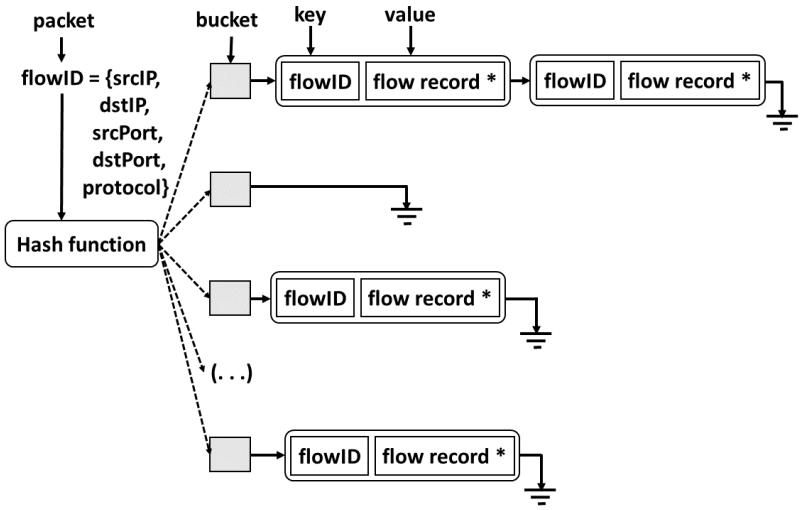


Figure 3.1: CFINF and CFTES deployment scenario (© 2017 IEEE)

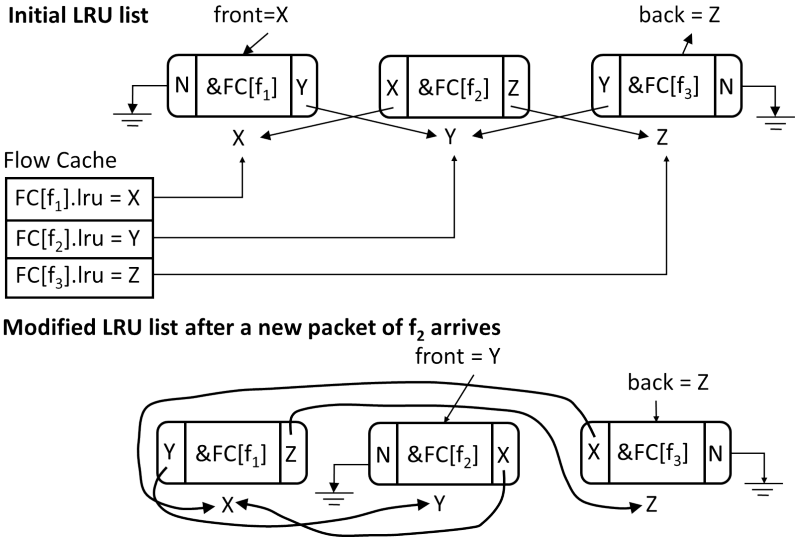
When the CFINF identifies a cheetah, the CFINF sends the cheetah flow identifier (flowID is the 5-tuple: source and destination IP addresses, source and destination port numbers, and protocol number) to CFTES, which then sends the received cheetah flowID immediately to an SDN controller via its northbound API. The functionality associated with the SDN-controller's northbound protocol could be complex. Therefore, the CFTES performs this function, which then allows the CFINF to execute its primary task of high-speed packet processing. The CFINF-CFTES interface can be simple, e.g., cheetah flowIDs can be passed via a lock-free circular buffer if the CFTES is run on a separate core of the CFINF server. Upon receiving a message via its northbound API, the SDN controller sets a filter rule (e.g., OpenFlow match operation) in router R1 to direct all subsequent packets of the cheetah flow to a separate queue on the monitored link (e.g., OpenFlow action).

When a cheetah flow ends, the CFINF will eventually evict the flow from its flow cache, and send the corresponding flowID to CFTES. This flowID will be passed by CFTES to the SDN controller for deletion of the filter rule in router R1.

We demonstrated the benefits of such cheetah flow redirection in a previous paper [45].



(a) Flow cache lookup using a hashing function



(b) Least Recently Used (LRU) list of flowIDs used to manage eviction of entries from flow cache

Figure 3.2: Performance optimization techniques used for high-speed operation of CFINF (© 2017 IEEE)

3.2.3 CFINF

Table 3.1 presents the notation used in this chapter. The flow-cache entry $FC[f]$ consists of the timestamps of the first and last observed packets of a flow (τ_1, τ_p), packet count p, cumulative byte count b, flow state ($state$), and location in a Least Recently Used (LRU) flow list (lru).

When a packet arrives, if its flowID is not in the flow cache, a new flow entry is created (if there is no space in the flow cache, an LRU-based flow eviction algorithm is executed).

Algorithm 1: Handle incoming packets (© 2017 IEEE)

Input : packet P
Initialization : $t \leftarrow$ arrival time instant
$f \leftarrow$ flow ID of P
1 **if** $FC[f] == NULL$ **then**
2 **Algorithm 2** ▷ New flow
3 **else**
4 **Algorithm 3** ▷ Existing flow

For each subsequent packet arrival, the last-packet timestamp, packet count, and byte count fields of the corresponding flow-cache entry are updated. The cheetah flow decision shown in (3.1) is made when p, the number of received packets for the flow reaches n, the number of packets required for a T-duration flight. If the CFINF decides that a flow is a cheetah, the CFINF sends the flowID to CFTES as described in Section 3.2.2.

If, on the other hand, the CFINF decides that a flow is not a cheetah, the flow entry is removed from the flow cache so that a new T-duration flight of packets can be assembled in the flow cache for that flow. For example, if a large-BDP TCP flow was in its Slow Start phase when its first packet was captured, the computed rate would be diluted if the flow was retained in the flow cache accumulating packets past the flight duration. Therefore, it is better to remove the flow-cache entry and start a new entry for the next flight of packets. The implication of this design is that a flow-cache entry could be created and deleted multiple times within the lifetime of a flow, with the T-duration flight rate being checked each time.

Details of the CFINF algorithms are as follows.

Table 3.1: Notation (© 2017 IEEE)

Packet $P : \{\tau, f, l\}$	{arrival time instant, flow identifier (flowID), and length}
Flow cache entry $FC[f]$: $\{\tau_1, \tau_p, p, b, state, lru\}$	{first packet timestamp, last packet timestamp, packet count, byte count, flow state, location in LRU list}
Algorithm (configurable) parameters: $\{L, S, T, R\}$	{packet length threshold, post-lookup sampling probability, flight duration, rate threshold}
d and r	duration and rate of a flow

Algorithm 1 extracts the five tuples from the IP header of each received packet P to form a flowID f. It then either creates a new flow entry (Algorithm 2) or updates an existing flow entry (Algorithm 3) in the flow cache FC.

Algorithm 2: Add new flow entry for packet P (© 2017 IEEE)

```
1  if protocol-drop check returns positive then
2      drop P and return;
3  l ← length of packet P;
4  if l < L then
5      drop P and return;
6  generate a random number r from X ~ U(0,1);
7  if r > S then
8      drop P and return;
9  if FC is full then
10     Algorithm 4                          ▷ Evict a flow entry to create space
11 Initialize flow entry and add flowID to LRU list:
12     FC[f].τ1 ← t; FC[f].τp ← t; FC[f].p ← 1; FC[f].b ← l;
13     FC[f].state ← monitored;
14     add flowID f to LRU list; update LRU list front variable;
15     FC[f].lru ← location in LRU list;
```

For high-speed implementations, a hashing function is used for O(1) flow-cache lookup. Fig. 3.2a illustrates the hashing procedure. For flows that hash to the same bucket, a linear search is carried out in the list corresponding to that bucket. Each entry consists of the flowID key, and flow record value, which is a pointer to its flow-cache location.

Missing in this algorithm is a pre-lookup packet sampling operation. Our algorithm for cheetah flow identification is based on a comparison of the rate of a flow with an absolute threshold, and therefore, it is important to capture all packets of a monitored flow. To handle high packet arrival rates, other optimizations are built into the later steps of our algorithm to control the number of entries in the flow cache, but the system resources should be sized to allow the initial flow-cache lookup to occur at an unsampled rate.

Algorithm 2 is executed to create a new flow entry, i.e., when the received packet is the first packet of a flight. In order to limit the number of flows being monitored, three tests are applied to drop packets that are not likely to be part of cheetah flows. *First*, packets for certain protocols, e.g., ICMP, are dropped, as these flows are not likely to be cheetahs. *Second*, packets whose size (length) is shorter than a threshold L are dropped. For TCP flows, the SYN segments are short packets, and hence will be dropped if L is not 0. However, subsequent data packets are likely to have large-sized frames and therefore new flow entries will be created for TCP flows. Traffic analysis, which is presented in a later section, shows

that the number of flows to be managed in the flow cache decreases significantly when this operation is executed since all packets of many real-time flows are short. *Third*, a packet sampling operation (controlled by parameter S) is applied. As described in Algorithm 1, once a flow entry exists in the flow cache, all packets of that flow are captured, but this sampling step is used to limit the number of monitored flows. The parameters L and S can be set to 0 and 1, respectively, if the CFINF processing rate is fast enough to keep up with packet arrival rate.

For a packet that survives the three drop steps described above, a new flow cache entry is created. If the flow cache is full (lines 9-10), a flow entry is evicted according to a procedure described in Algorithm 4. Lines 11-15 show how the fields of the new flow-cache entry are initialized, and how the LRU-list is updated. The flow state is set to `monitored` to accumulate packets for a T-duration flight.

Fig. 3.2b shows that the LRU list is doubly linked. Each entry holds (i) the location of the flow-cache entry for flowID f (shown as $\&FC[f]$), (ii) a pointer to the previous entry in the list, and (iii) a pointer to the next entry in the list. When a new flow entry is created in the flow cache, a new entry is created in the LRU list, and its location is placed in the `front` variable of the LRU list. Further, the location of the flowID entry in the LRU list is written into the last element, location-in-LRU-list (lru), of the flow-cache entry. This structure makes the process of evicting a flow entry an efficient O(1) operation.

Algorithm 3 is executed when the arriving packet already has an existing entry in the flow cache. Line 2 shows that the last packet timestamp, packet count, and cumulative byte count fields of the flow entry are updated. The LRU-list related actions are executed in lines 3-4. Fig. 3.2b illustrates an example with an initial state of the LRU list, and how pointers are modified when a packet from flow f_2 arrives and the `front` variable of the LRU list is modified to point to Y, which is the location of the entry for flow f_2 in the LRU list.

If the flow is in the `monitored` state, i.e., a T-duration flight of packets is being assembled, its flow duration is first computed (line 7). If the flight duration T has been crossed (line 8), the flow rate is computed (line 10) and compared with the rate threshold R (line 11). The flow is classified as a cheetah flow if the rate threshold is crossed, and hence its state is

Algorithm 3: Update flow entry for P & check if cheetah (© 2017 IEEE)

```
1                          ▷ Update fields of the flow entry for the arriving packet;
2  FC[f].τ_p ← t; FC[f].p ← FC[f].p + 1; FC[f].b ← FC[f].b + l;
3  use FC[f].lru location to update LRU list entry for f;
4  update LRU list front variable;
5  if FC[f].state == monitored then
6      ▷ Determine flow duration and flow rate;
7      d ← FC[f].τ_p − FC[f].τ_1;
8      if d > T then
9          ▷ Check if flow is a cheetah;
10         r ← FC[f].b/d                          ▷ Compute flight rate;
11         if r > R then
12             FC[f].state ← cheetah;
13             send f to CFTES to set filter rule;
14         else
15             ▷ delete entry and update LRU list;
16             delete LRU list entry located at FC[f].lru;
17             update LRU list front variable;
18             delete flow entry FC[f];
```

Algorithm 4: Evict a flow entry (© 2017 IEEE)

```
1  A ← &FC[f] from location stored in LRU-list back variable;
2                          ▷ Address of flow-cache entry;
3  delete LRU list entry that was stored in back variable;
4  update LRU list back variable;
5  if A.state == cheetah then
6      send f to CFTES to delete filter rule;
7  delete flow entry at address A of flow cache to free-up space;
```

modified to `cheetah` and the flowID is sent to CFTES (lines 12-13).

If the rate of the T-duration flight of packets is below the rate threshold R, the flow removed from the cache for reasons explained at the start of this section. Lines 16-18 deal with the LRU-list updates required when a flow is removed from the flow cache.

Implicit in Algorithm 3 is the fact that once a flow is identified as a cheetah, the flow rate is not computed again as the flow has already been redirected. When the cheetah flow ends, the position of its flowID will move to the end of the LRU list, at which point, the flow entry will be evicted by Algorithm 4.

Algorithm 4 evicts a flow from the flow cache thereby creating space to add a new flow entry. The LRU-list `back` variable has the location of the last node of the LRU list. Line 1 shows that from this location, the address A of the flow-cache entry of the flow corresponding to

the last element of the LRU list is determined. Lines 3-4 delete the last LRU-list node and update the LRU-list `back` variable. If the flow corresponding to the entry stored at A in the flow cache was a cheetah, a message is sent to delete the filter rule that was used for redirection (line 6), before the flow entry itself is deleted from the flow cache to free up space (line 7).

3.3 Traffic Trace Analysis

The Center for Applied Internet Data Analysis (CAIDA) [44] collects unsampled 1-hour packet traces once a quarter (every 3 months) on 10 Gbps links. Each 1-hour trace is saved in 61 files, with each file storing packet headers from roughly 1 min of traffic (the extra file occurs due to timing alignment issues). Each 1-min file is approximately 2.4 GB.

We downloaded these traces and used them in experimental tests of our CFINF implementation as will be described in the next two sections. But in this section, we describe three types of analyses that we ran on these traces.

First, Section 3.3.1 describes our analysis of a 1-min CAIDA trace to evaluate the efficiency of the hashing function used for flow-cache lookup. To determine an appropriate value for the rate threshold of the CFINF algorithms, we analyzed rates of flows in six 1-hour CAIDA traces. This analysis is presented in Section 3.3.2. Finally, Section 3.3.3 describes an analysis of the same six 1-hour CAIDA traces to determine whether there is a performance benefit in using the length-based packet filter of Algorithm 2.

3.3.1 Evaluation of hashing algorithm

As described in Section 3.2.3, for high-speed flow-cache lookups, a hashing function is used. Specifically, the Bernstein hash function [46] is used. An ideal hash function should map every key to a different bucket, and there should be no unused buckets. This is not always as the case as illustrated in Fig. 3.2a, where multiple keys hash to the same bucket and there are unused buckets. It is therefore advisable to test a particular hash function with sample data. Specifically, a 1-min CAIDA packet trace was used (June 2014 trace collected at 13:00 hours UTC from the equinix-sanjose direction A monitor).

Fig. 3.3 shows the histogram of the number of flow entries per bucket after hashing 38,166,365 packets (1,788,675 flows). More than 900 K buckets did not have a flow entry, and more than 700 K buckets contained only one flow entry. The maximum number of flow entries appearing in one bucket was 9, and there were just two such occurrences. The percentages of flowIDs hashing to buckets with one, two, three, and four entries were 43.2%, 36%, 15.2%, and 4.3% respectively. Therefore, only a small percentage of flows (1.3%) required linear searches in a list with more than four entries. In summary, for the type of

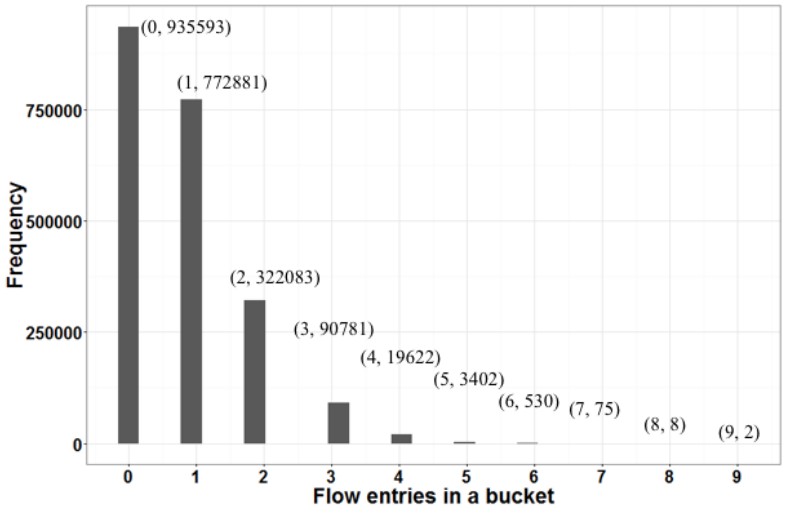


Figure 3.3: Histogram of number of flow entries in one bucket

flowIDs observed on the link monitored by CAIDA, the Bernstein hash function performs well.

3.3.2 Flow-rate analysis

Since Algorithm 3 computes the rate of a flow over an observation duration T, in this analysis, for each flow in the traffic trace, we computed the rate of the flow in each non-overlapping discrete time interval of duration T within its lifetime (if CFINF is run with no packet filtering or post-lookup packet sampling, these are the rates that CFINF would compute). For example, if a flow is 5-sec in duration, and T is set to 1 sec, then five rate values are computed for this flow. We use the term *flow-window rate* to describe this metric. The total number of time windows is $\sum_{f=1}^{f=F} D_f$ where F is the total number of flows in the trace, and D_f is the duration of flow f expressed as a multiple of T.

Entries in Table 3.2 show the number of time windows in which the rate of some flow exceeded the rate threshold R. A single flow could be counted multiple times in any of the entries in Table 3.2 since each flow has multiple flow-window rates, and more than one of these flow-window rates could exceed R.

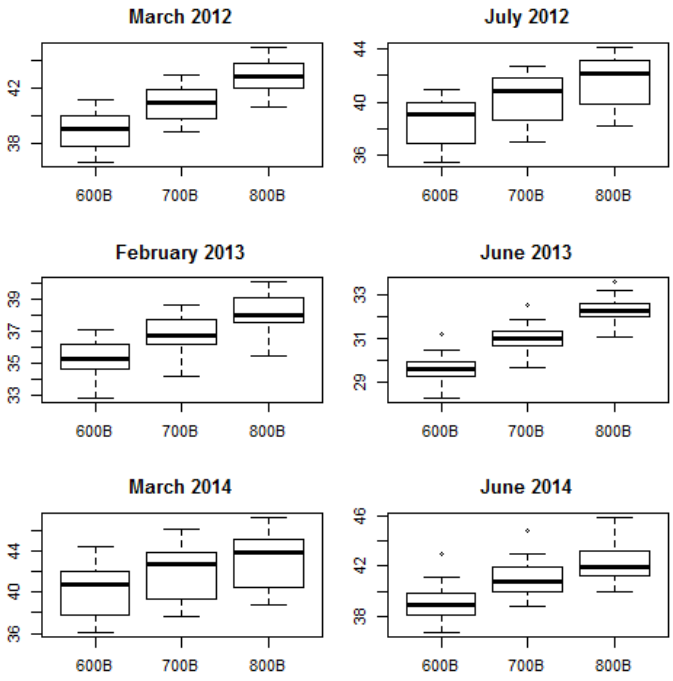
Rate threshold R (Mbps)	50	100	200	500	1000
`June 2014`	4761	755	124	6	0
`March 2014`	1949	432	96	2	0
`June 2013`	1469	468	129	21	0
`Feb 2013`	2105	1099	88	3	0
`July 2012`	1639	395	97	3	0
`March 2012`	1276	324	97	15	0

Table 3.2: Number of times a flow-window rate $> R$

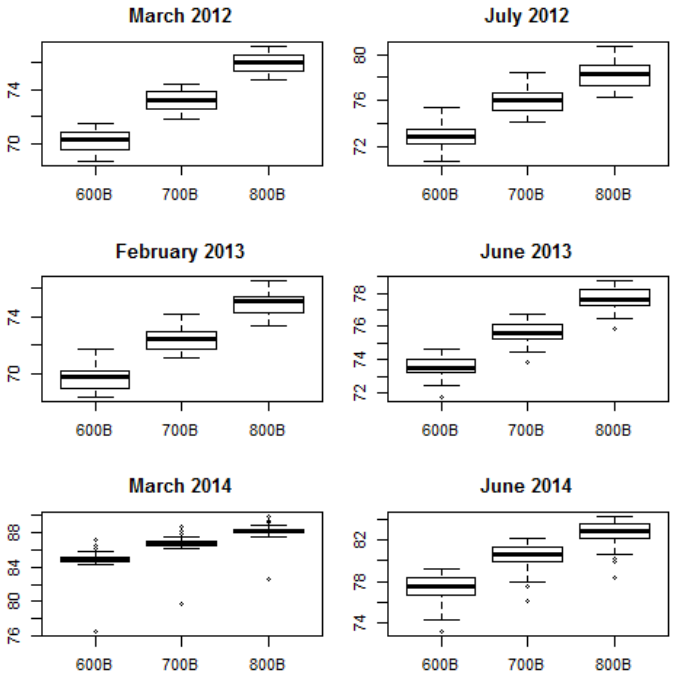
Our analysis shows that the flow-window rate exceeded a rate threshold of 5% of the link capacity, i.e., 500 Mbps, a maximum of 21 times in the June 2013 trace, and there were no time windows in which any flow's rate exceeded 10% of the link capacity (1 Gbps). We used 50 Mbps as the rate threshold R in our experimental testing of CFINF given the number of time windows observed with this setting.

3.3.3 Impact of packet-length based filtering

For CFINF performance optimization, Algorithm 2 includes three methods to drop packets as a way of limiting the number of entries in the flow cache. One of these methods was filtering out small packets, and initializing a flow-cache entry only for packets longer than the packet-length threshold, L. This type of length-based packet filtering, if done without post-lookup packet sampling, will not affect flow-rate computation of potential cheetah flows. This is because while the TCP SYN segment will be dropped, the first data packet (typically, maximum-sized) will be captured by CFINF, which will then create a flow-cache entry. Recall that once a flow-cache entry is created, all subsequent packets of the flow will be processed by CFINF. Thus, dropping the SYN segment will not hurt the flow-rate computation; to the contrary, the computed flow rate will be more accurate for high-RTT flows.



(a) Percentage of packets dropped



(b) Percentage of flows dropped

Figure 3.4: Potential for computational savings offered by length-based packet filtering (© 2017 IEEE)

This section describes an analytical study of CAIDA packet traces to determine the potential savings in terms of computational effort required of CFINF when length-based packet filtering is executed. Two questions were asked: (i) what is the percentage reduction in the number of packets, and (ii) what is the percentage reduction in the number of flows? The question of whether the accuracy of cheetah-flow identification is compromised with this filtering is addressed in Section 3.4.

The analysis program implemented the relevant features of Algorithms 1 and 2, i.e., the program maintained a database with just flowIDs, and dropped a packet whose length was less than the threshold L only if the corresponding flowID was not present in the database. However, this analysis program did not implement flow eviction, which means the reported percentage of packets dropped would be lower that in the real CFINF implementation. Inspite of this simplification, we found that significant savings in computational effort are possible with this type of filtering.

Specifically, six different one-hour packets traces collected over a three-year period by CAIDA were analyzed. All traces were collected by the same `equinix-sanjose` monitor on a 10 Gbps link. Each trace consists of 61 one-minute packet capture files.

Fig. 3.4a shows the percentage of packets dropped for different values of the packet-length threshold L: 600B, 700B and 800B. The percentage of packets dropped was computed for each one-min file, and the statistics across the 61 one-min files collected in a quarter, e.g., March 2012, are plotted in the corresponding boxplot. With an 800B threshold, across all 366 (6×61) one-min files, we found that, in the worst case, the maximum percentage of packets left behind would have been 68.9% (this number was computed for a one-min trace from June 2013).

Fig. 3.4b shows the percentage of flows that would have been dropped for different values of the threshold L. With an 800B threshold, across all 366, one-min files, we found that, in the worst case, the percentage of flows left behind was 26.6% (this number was computed for a one-min trace from Feb 2013). This is a significant reduction in the number of flows handled.

In *summary*, packet length based filtering is a promising approach for CFINF performance optimization. The impact on accuracy of cheetah-flow identification will be presented in the

next section.

3.4 High-speed CFINF Implementation and Evaluation

We implemented CFINF in a high-performance, multicore commodity x86 hardware system called R-Scope [43]. The R-Scope system has two Intel Xeon E5-2670 processors running at 2.50 GHz for a total of 20 physical cores (without hyperthreading). It has four high-speed Solarflare SFC9100 optical network adapters, which support high-performance NIC features such as packet coalescence, receive-side scaling, kernel bypass, and polling-mode operation. A software layer called Dynamic Network Acceleration for many-Core (DNAC) interfaces with these high-speed NICs, and offers CFINF a set of library functions for efficient packet handling. DNAC performs flow-preserving load balancing, i.e., it sends all packets of a flow to the same core. Therefore, the CFINF instances running on separate cores execute independently without any interaction between the instances. This design feature ensures scalability of CFINF to higher link speeds by using off-the-shelf commodity systems with more processing cores.

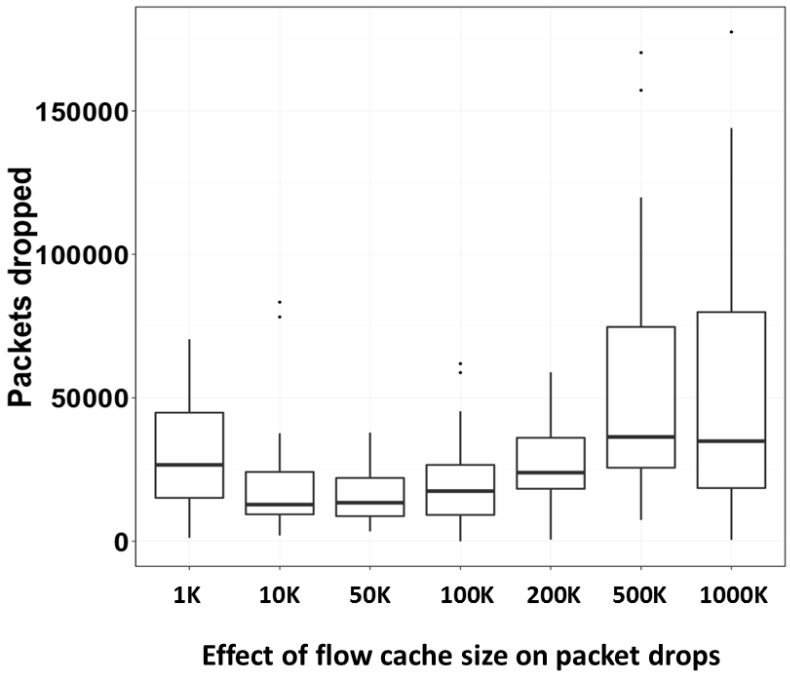
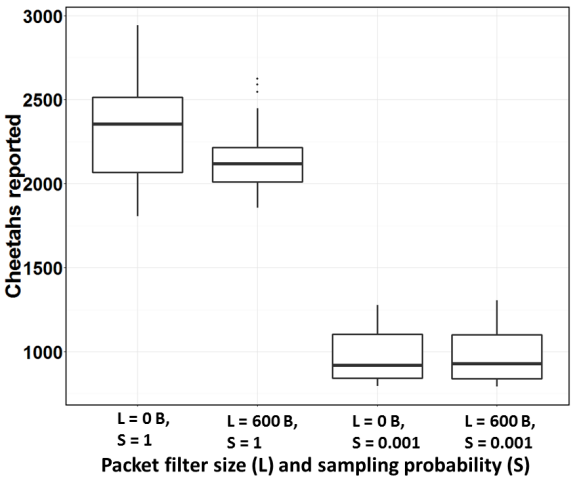
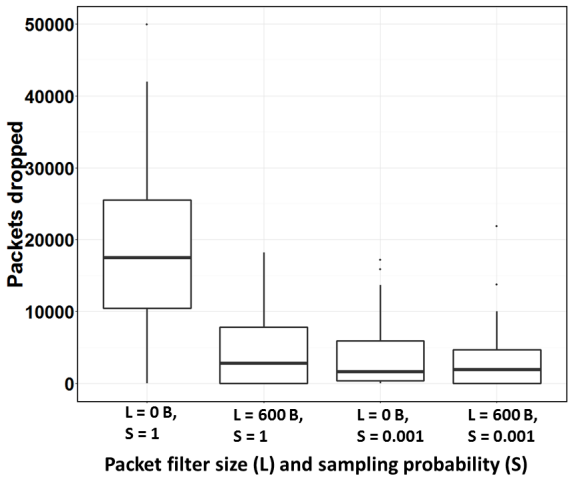


Figure 3.5: CFINF performance as a function of flow-cache size (© 2017 IEEE)



(a) Number of reported cheetahs



(b) Number of packets dropped

Figure 3.6: Impact of length based filtering and sampling; T = 100 ms; R=50 Mbps (© 2017 IEEE)

Test setup: A traffic-generator host and the CFINF host were interconnected via two 10-Gbps links. The Linux `tcpreplay` utility was executed on the traffic-generator host to play out a 1-min CAIDA packet trace (which had 38 M packets, 1.5 M flows, and an aggregate size of 28 GB). Each instance of the `tcpreplay` utility could only generate packets at 2.5 Gbps, but using multiple instances and switch multicasting, we were able to create a setup in which the CFINF host received packets at 5 Gbps on each of its two ports, for a total of 10 Gbps.

Four instances of CFINF were run to handle packets received on each of the two ports for a total of 8 cores. The rate threshold R was set to 50 Mbps, and the flight duration T was set to 100 ms. Four combinations of the packet filtering threshold, L, and sampling

probability, S, were tested.

Two output metrics were collected: number of reported cheetahs, and number of dropped packets. The number of packets dropped is measured by a packet drop counter at each R-Scope interface, and reported out by DNAC. Packets are dropped when there is no space left in the DNAC queues that hold arriving packets while waiting to be read by the CFINF instances.

Determine flow-cache size: The cache size was varied from 1 K to 1 M flow entries in a series of experiments. For each setting, 20 runs were executed. In each run, the total number of packets received by R-Scope was about 138 M, and due to the high entropy of source and destination IP addresses in the trace, packets were evenly processed by the 8 cores.

Fig. 3.5 shows that the median number of dropped packets follows a U shaped curve with increasing flow-cache size. With a small flow cache, the rate of evictions was high, and the extra compute cycles spent in flow-cache management led to dropped packets. When the flow-cache size was larger than the processor cache, the required, but slower, main-memory (RAM) operations caused the dropped-packet percentage to increase. As the best performance was obtained with a flow-cache size of 50K, this setting was used in subsequent experiments.

Number of reported cheetahs: Fig. 3.6a shows statistics for the number of reported cheetahs across 30 runs for each setting of the packet-filtering length threshold L, and post-lookup sampling probability S. Filtering without sampling ($L = 600$B; $S = 1$ setting) does not unduly affect the accuracy of cheetah flow identification (the median is 2354 without filtering and 2116 with filtering alone). However, when sampling was introduced, $S = 0.001$, the median number of reported cheetahs dropped significantly to 919. This result appears to indicate that post-lookup packet sampling is not a good means to optimize CFINF performance.

Percentage of dropped packets: Fig. 3.6b shows the number of packets dropped by CFINF. With 8 cores, the maximum percentage of packets dropped is only 0.036% since the total number of packets is 38M. When the number of cores was increased to 10, in 6 out of 30 runs, there were no packet drops.

Next, consider the effects of filtering and sampling. Fig. 3.6b shows that the median number of dropped packets falls by 83% with filtering. With filtering, there were no eviction events as most flow entries are removed (by Algorithm 3) at a frequency of 100 ms, which creates space in the flow cache. Without filtering, there were 2.13M evictions (by Algorithm 4). Recall from Fig. 3.6a that the number of reported cheetahs was not affected significantly by filtering alone.

Therefore, we conclude that length-based packet filtering is a good solution to improve efficiency. Sampling offers a slight reduction in the number of packets dropped, but recall that its impact on the accuracy of the number of cheetahs reported was significant.

3.5 Experimental Studies of Cheetah Flows

Two sets of experimental studies were conducted. First, we ran experiments to test the basic functionality of CFINF: identifying a cheetah flow in real-time and redirecting it to a CF queue. This experiment demonstrates the value offered by CFTES by comparing the impact of the cheetah flow on a delay-sensitive flow, with and without redirection. These experiments are described in Section 3.5.1.

Next, Section 3.5.2 describes a series of experiments that we undertook in two testbeds to study how various parameters of a potential cheetah flow can determine whether or not the flow has adverse affects on other flows.

3.5.1 Illustration of the value offered by CFTES

Fig. 3.7 shows the experimental setup used for testing the basic functionality and illustrating the value of CFTES. The setup is a ProtoGENI slice consisting of five bare-metal hosts (Host 1 to Host 5) located in the University of Kentucky testbed. All hosts have a single 4-core CPU (Intel Core 2 Quad CPU Q6600 @ 2.40GHz) and 8 GB of RAM. Each host has five 1 GigE Network Interface Cards (NICs). The `eth0` NICs on hosts are used for remote login, while the other NICs are used in the data-plane of the experiment. Host 1 was used to send a `ping` flow, Host 2 was used used to replay a real packet trace collected by the

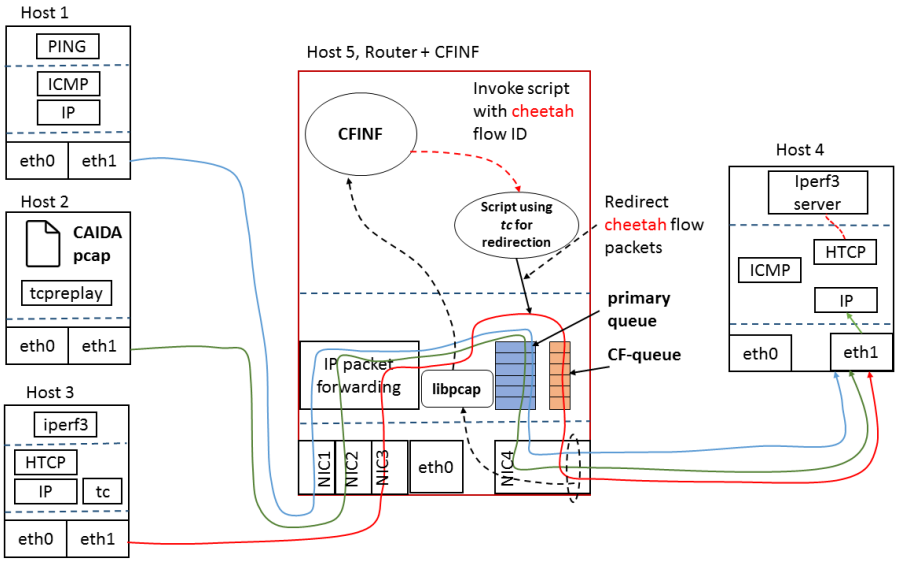


Figure 3.7: Example use of CFTES; CF: Cheetah Flow

CAIDA to emulate background traffic, and Host 3 was used to generate a cheetah flow with `iperf3`. All flows were destined to Host 4.

Host 5 was configured to serve as an IP router and forward packets between NICs 1 through 4 as illustrated in Fig. 3.7. While in real usage, the CFINF would be executed in a host distinct from the router, in this experiment, the CFINF was executed on Host 5, as shown in Fig. 3.7. As part of CFTES initialization, two output queues —a primary queue and an Cheetah-Flow (CF) queue —were created for the output port NIC4 using the Linux `traffic control (tc)` utility. The `tc` configuration specifies rates for the two queues but allows for bandwidth borrowing between the queues, i.e., if one queue has no packets, the scheduler will send packets from the other queue. The default option was to send all packets to the primary queue.

As shown in Fig. 3.7, `libpcap` is used to emulate the port mirroring operation described in Section 3.2 in which all packets sent out on the access link are copied to CFINF. In this experiment, NIC4 emulates the access link. The CFINF rate threshold R is set to 100 Mbps and the duration threshold T is set to 1 sec. The length threshold $\mathbb{L}$ was set to 0, and post-lookup sampling probability S was set to 1 (which means no packets were dropped).

The CAIDA packet trace has only TCP and IP headers. The Ethernet header and payloads were stripped out before the packet trace was saved at CAIDA. We used a `python`

packet manipulation module called `scapy` to add dummy payloads to each IP packet in the trace using the packet length specified in the IP header. In order to adapt the 10 Gbps CAIDA trace to our 1 Gbps experiment, packet inter-arrival times were increased by a factor of 10. We used `tcprewrite` to add source and destination MAC addresses to each packet, and the destination IP address of each packet was changed to the IP address of the `eth1` interface of Host 4 so that the kernel-level IP packet-forwarding table of Host 5 (router emulator) required just one entry. This enabled the packet forwarding software to keep up with packets at 1 Gbps speed. The `tcpreplay` tool was used to replay the modified CAIDA trace.

To emulate an cheetah flow, a high-rate `iperf3` HTCP flow was initiated on Host 3. Using `tc`, the sending rate on `eth1` NIC was set to 800 Mbps.

With CFINF running, the `ping` flow was initiated, and the modified CAIDA packet trace was replayed. CFINF captured packets from all three flows and ran its algorithms. When the duration threshold (1 sec) was crossed by the `iperf3` flow, the flow was identified by CFINF as a cheetah flow. CFINF then invoked a script specifying the cheetah flowID as an input parameter. The script used the `tc` utility to set a filter rule for subsequent packets of the cheetah flow to be enqueued in the CF queue.

Fig. 3.8 shows the `ping` delay in two cases: with redirection of the cheetah flow (in blue) and without redirection of the cheetah flow (in red). *Without redirection*, the `ping`-flow packets experienced increased queuing delays (from 5ms to over 13.8 ms) as the cheetah flow and background traffic filled up the primary queue. The reason why ping-flow packets experienced lower delay (5 ms) soon after initially experiencing 13.8 ms is because the buffer filled up and the cheetah flow experienced packet losses causing its sending rate to drop. But after the ping delay drops to 5 ms, it starts growing again as the cheetah flow ramps up its sending rate.

In the run *with cheetah-flow redirection*, the `ping` flow initially experienced an equally large delay (16.8 ms) because CFINF waits for 1 sec to determine whether a flow is a cheetah before redirection. However, the effect of redirection can be seen in Fig. 3.8 shortly after 1 sec. As the cheetah-flow packets were redirected to a separate CF queue, the `ping` flow delay dropped to an average of 2 ms. This method of online cheetah-flow detection and traffic

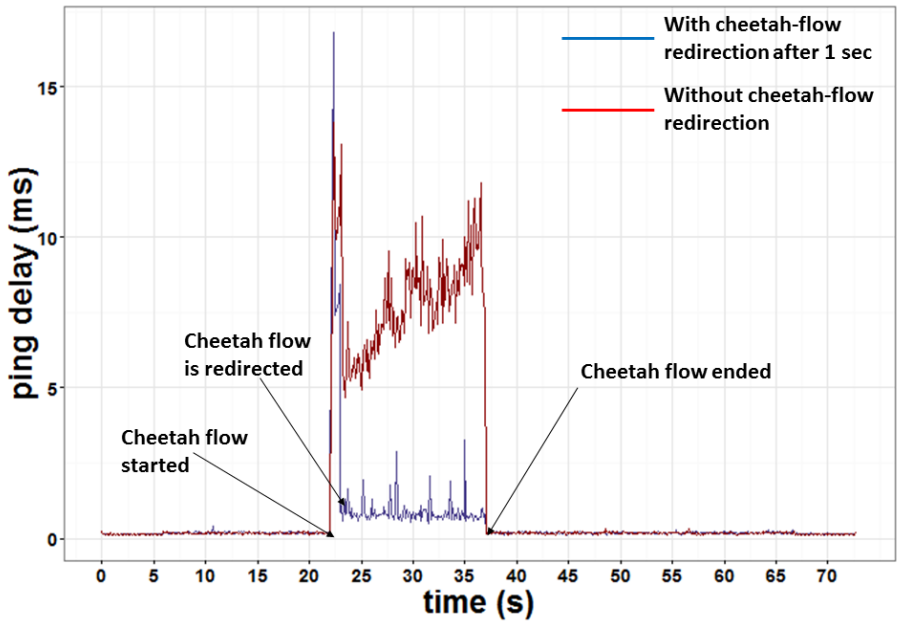


Figure 3.8: Illustration of the use and value of CFTES; isolation of CF-flow packets causes the ping flow to enjoy low latency

engineering prevents delay-sensitive flows (as exemplified by the ping flow) from experiencing high latency.

3.5.2 Impact of different parameters on cheetah flow behavior

Two intra-rack experimental setups were used: (i) an InstaGENI setup, and (ii) Chameleon [47] setup. The topology was similar to the one shown in Fig. 3.7 except that a top-of-rack (TOR) Ethernet switch (instead of Host 5) connected the four hosts, Host 1 through Host 4. All links in the InstaGENI setup were 1 GigE, while in the Chameleon setup, all links were 10 GigE. Using UDP flows and a ping flow, we determined the switch buffer size to be about 2 MB in the InstaGENI TOR switch, and 5 MB in the Chameleon TOR switch.

A ping flow was sent from Host 1 to Host 4, the CAIDA trace, used as background traffic, was replayed from Host 2 to Host 4, and an `iperf3`-HTCP flow, emulating a cheetah, was sent from Host 3 to Host 4. On the InstaGENI setup, the modified CAIDA 1 Gbps trace was replayed with the multiplier factor set to 1. On the Chameleon Cloud setup, the CAIDA trace was replayed without a rate change. The round-trip time (RTT) between Host 3 and Host 4 on InstaGENI and Chameleon Cloud were 0.348 ms and 0.237 ms, respectively.

Two key parameters on the path of the HTCP flow, bottleneck link rate and RTT, were controlled using `tc` and `netem`, respectively. One other parameter, TCP buffer size (the same value was used for send- and receive-side buffers), was controlled for the HTCP flow. Output metrics, collected from `iperf3` logs, include (i) per-sec HTCP-flow throughput, (ii) per-sec packet retransmissions, and (iii) per-sec congestion window size.

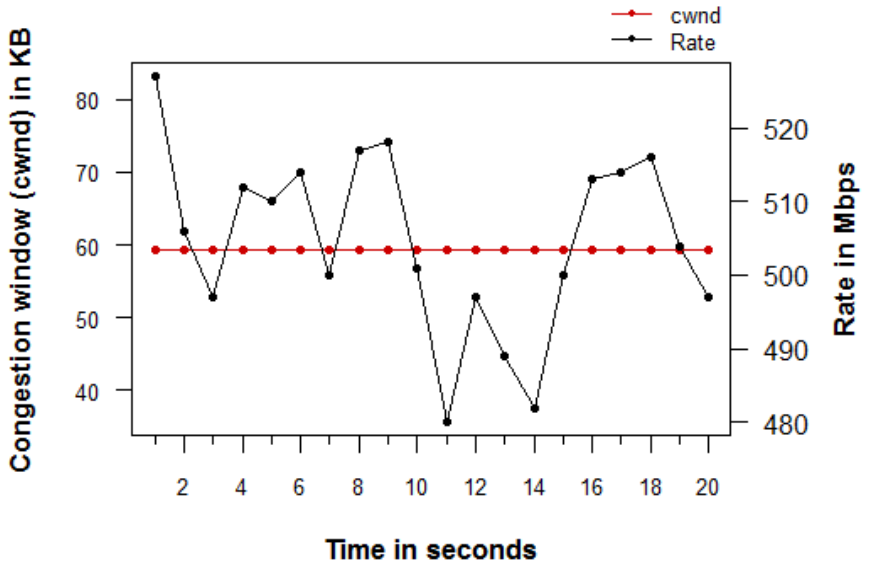
The five experimental cases that were executed are described in Table 3.3. A 1-min CAIDA trace was used in each experiment, and the HTCP flow was started at a random time after the CAIDA-trace replay was initiated.

InstaGeni 1 Gbps testbed; b_r = 350 Mbps					Chameleon 10 Gbps testbed; b_r = 3.2 Gbps			
Cases	B	RTT_{CF}	R_{CF}	loss	B	RTT_{CF}	R_{EF}	loss
Case 1	160 KB	0.3	800 Mbps	no	1.2 MB	0.3	8 Gbps	yes
Case 2	2 MB	0.3	800 Mbps	yes	10 MB	0.3	8 Gbps	yes
Case 3	10 MB	0.3	100 Mbps	no	20 MB	0.3	1 Gbps	no
Case 4	40 MB	100	800 Mbps	yes	400 MB	100	8 Gbps	yes
Case 5	40 MB	100	100 Mbps	no	400 MB	100	1 Gbps	no

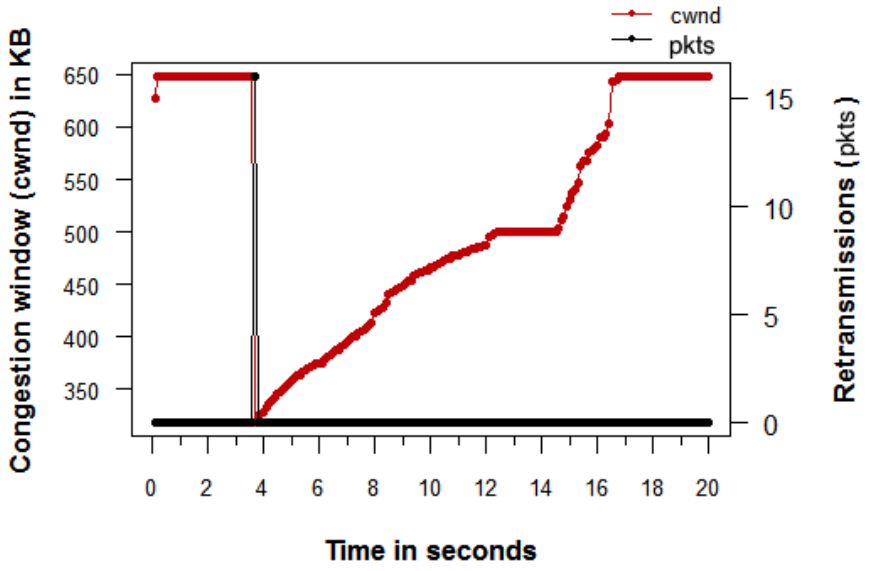
Table 3.3: Parameters used in different experimental runs, and observed packet losses; TCP buffer is denoted as B, RTT is in ms and background traffic rate is b_r

Low-RTT cases: In the first three cases, no additional delay was added to the HTCP-flow path, and therefore RTT was approximately 0.3 ms. Two values of bottleneck link rate were emulated using `tc` at Host 3.

Consider the difference between cases 1 and 2 in the InstaGENI experiments. The TCP buffer size in case 1 was smaller than the switch buffer size (which was 2 MB), while in case 2, the TCP buffer size was matched to the switch buffer size. The HTCP-flow path bottleneck rate was high (800 Mbps) in both cases, and the background traffic rate was 350 Mbps (CAIDA trace is bursty; this 350 Mbps number was computed over the whole 1-min range). The HTCP sender congestion window (`cwnd`), as reported on a per-sec basis by `iperf3`, stayed unchanged at 59.4 KB, as seen in Fig. 3.9a, and there were no packet losses. The right-hand side y-axis shows that the HTCP-flow per-sec throughput (rate), as reported by `iperf3`, showed some variation, presumably as the switch buffer filled and caused increased RTT, but the rate was approximately 500 Mbps. Recall that the background traffic rate was 350 Mbps, and link capacity was 1 Gbps.



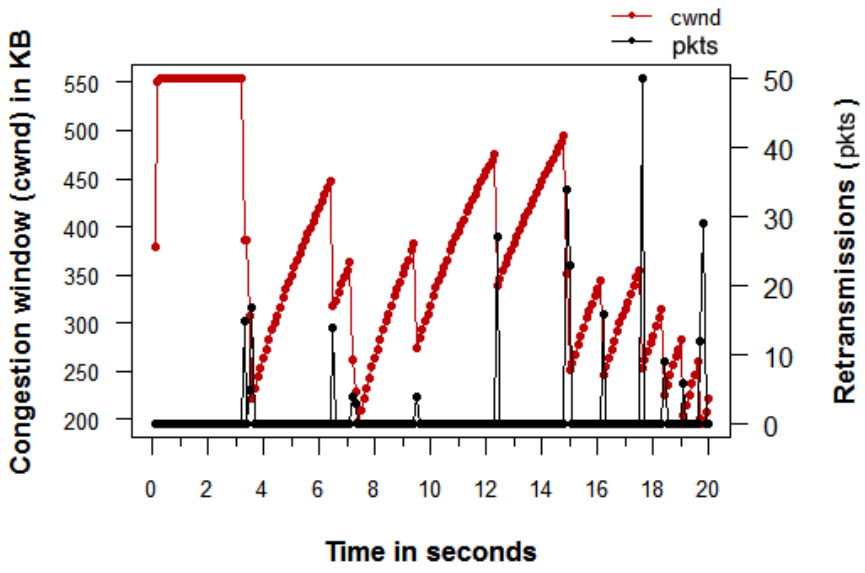
(a) Case 1: Small TCP buffer; InstaGENI; EF rate: 800 Mbps



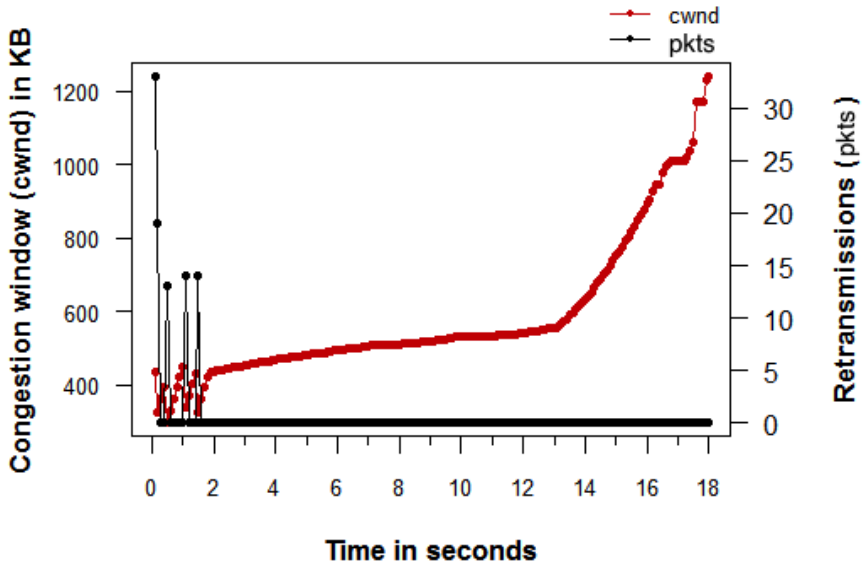
(b) Case 1: Small TCP buffer; Chameleon; EF rate: 8 Gbps

Figure 3.9: Plots from `iperf3` logs for low-RTT path experiments (cases 1-2) for small TCP buffer

In contrast, in case 2, since `cwnd` was allowed to grow to a large value (TCP buffer size was 2 MB), the burst of traffic transmitted by the HTCP sender caused the switch buffer (also 2 MB) to overflow, since CAIDA traffic is bursty, and hence there were resulting packet losses as seen on the right-hand y-axis of Fig. 3.10a. Correspondingly, cwnd also showed increases and decreases (see left-hand y-axis of Fig. 3.10a).



(a) Case 2: Large TCP buffer; InstaGENI; EF rate: 800 Mbps



(b) Case 2: Large TCP buffer; Chameleon; EF rate: 8 Gbps

Figure 3.10: Plots from `iperf3` logs for low-RTT path experiments (cases 1-2) for large TCP buffer

These two cases offer an interesting challenge for rate-threshold setting for CFINF. In both cases, the HTCP flow registered a rate of more than 500 Mbps in the first sec. However, in case 1, no packet losses were caused because the HTCP sender was not allowed to increase its `cwnd` to a point close to the switch buffer size, while in the case 2, this was allowed by the high value set for the TCP buffer size. If a flow is declared a cheetah only if it causes packet

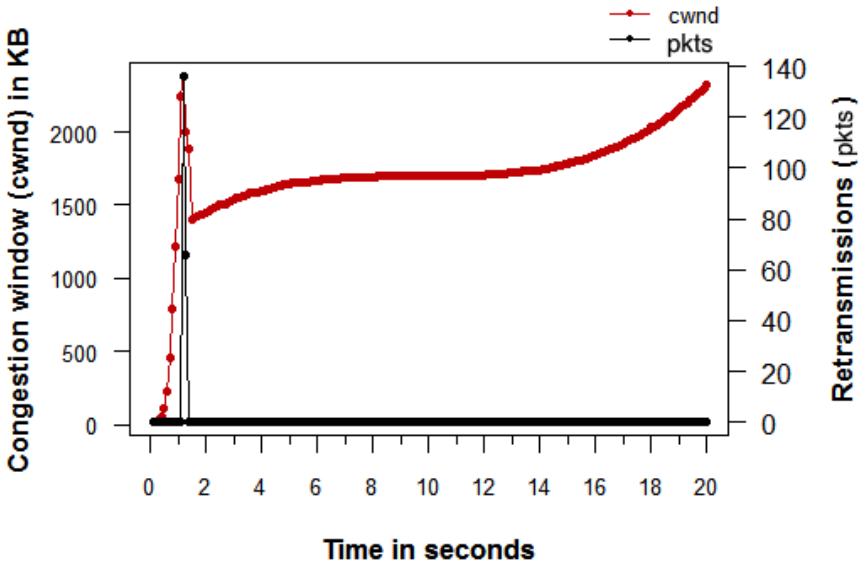
losses, arguably, the HTCP flow in case 1 is not a cheetah flow. But, it is safe to say that a flow capable of reaching 500 Mbps on a 1 Gbps link with a 350 Mbps background traffic rate, is likely to cause packet losses sooner or later since the background traffic could drop allowing the HTCP sender to start sending at a higher rate and then when the background traffic increases again, there will be packet losses before the HTCP sender can reduce its rate. In summary, by choosing a rate threshold for a cheetah flow that allows some (say 20%) headroom when added to the background traffic, packet losses can be prevented.

Table 3.3, and Figs. 3.9b and 3.10b, show the results for a similar pair of experiments on the Chameleon testbed. In case 1, there were packet losses but because the link rate was so high (10 Gbps), it took a while for `cwnd` to build back up even with HTCP's aggressive rate-increase algorithm.

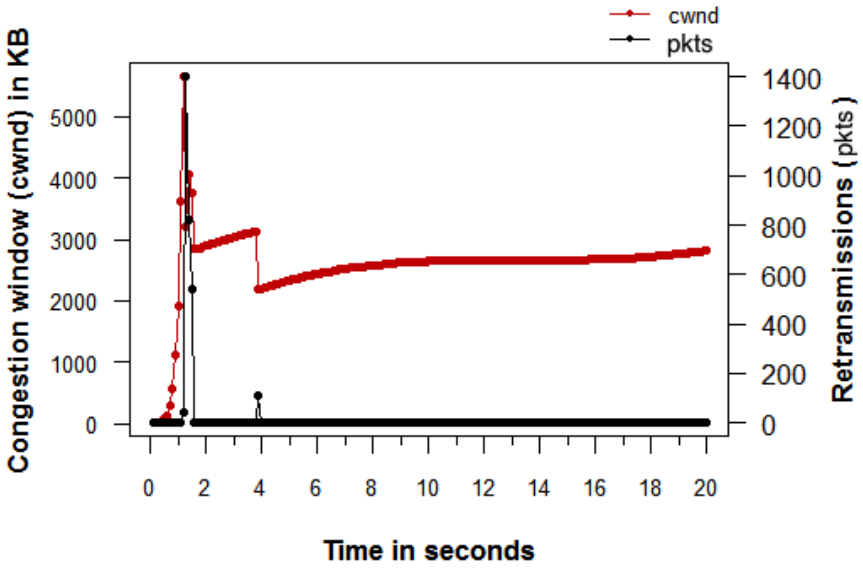
Case 3 shows a run in which the HTCP flow was rate limited to 100 Mbps on InstaGENI and to 1 Gbps on Chameleon, and since this flow rate when added to the corresponding background traffic rate was well below the link capacity, there were no packet losses, even though the TCP buffer size was larger than the switch buffer size in both cases.

High-RTT cases: A second set of intra-rack experiments was run with an emulated delay of 100 ms added to the HTCP-flow path. In case 4, the HTCP-flow path rate was high (800 Mbps in InstaGENI and 8 Gbps in Chameleon), and the TCP buffer size was larger than switch buffer size. For such a high-BDP path, where both rate and RTT are high, the TCP buffer size needs to be large to achieve high throughput. But when the TCP buffer size is larger than the switch buffer size, packet losses occur more easily since a whole `cwnd` worth of data could be sent in a burst by the HTCP sender. Results for the InstaGENI and Chameleon experiments are shown in Figs. 3.11a and 3.11b, respectively. In case 5, since the path is rate limited to a low value (100 Mbps in InstaGENI and 1 Gbps in Chameleon), even though the TCP buffer size is large, `cwnd` does not grow larger than the switch buffer size and there are no packet losses, as seen in Fig. 3.12.

In *summary*, SNMP-reported packet losses should be used to adjust the rate threshold because on high-RTT, high-rate paths, users would have likely set large TCP buffer sizes to achieve high throughput, and these buffer sizes are likely to exceed switch buffer sizes.



(a) InstaGENI; Large TCP buffer, EF rate: 800 Mbps



(b) Chameleon; Large TCP buffer; EF rate: 8 Gbps

Figure 3.11: Plots from `iperf3` logs for high-RTT path experiments (case 4)

Second, the background traffic rate should be measured, and the cheetah-flow rate threshold should be correspondingly adjusted to leave sufficient headroom in link usage. We use these metrics to compute the rate threshold in the next chapter.

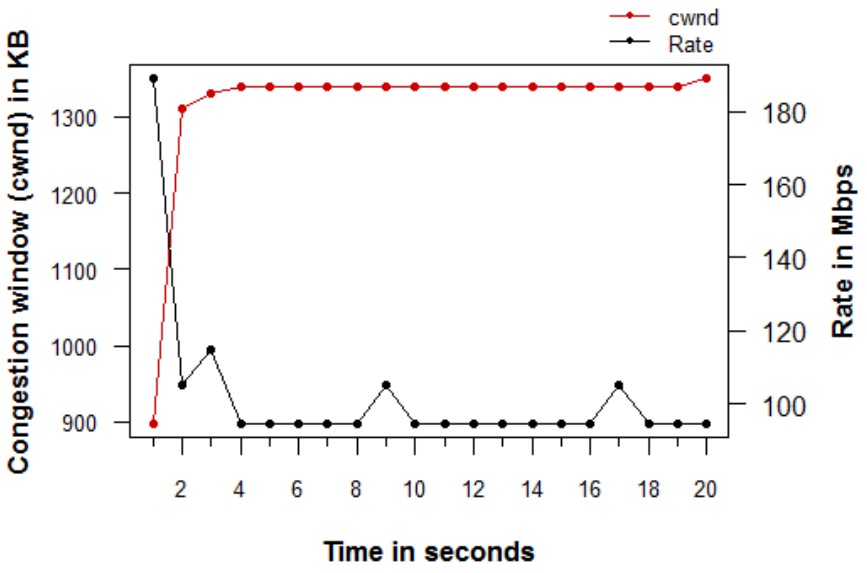


Figure 3.12: Case 5; InstaGENI; EF rate: 100 Mbps

3.6 Related work

Lan and Heidemann [48] offered the names elephants for large-sized flows, and cheetahs for high-rate flows. The percentage of cheetahs that were also elephants in one of the two analyzed traces was 72%. High-rate flows are likely to be caused by users moving large datasets, and hence such a high correlation can be expected. Therefore, we compare our work to papers that focused on elephant flow identification.

As stated in Section 3.1, most of the prior algorithms proposed for elephant flow identification [37–42] find the highest-rate flows, not flows whose rate exceed a set threshold since packet timestamps are not stored in the flow cache. Furthermore, many of these solutions use pre-lookup packet sampling to improve processing-power efficiency.

NetFlow/IPFIX [30] implementations in traditional routers do not offer flexibility and scalability. Typically, NetFlow/IPFIX records are exported from a router only once every 5 mins, which is too long to be useful for cheetah flow redirection. Reducing this interval will place a heavy computational load on the route processor.

NFV offers an alternative solution of implementing flow-record creation in commodity hardware as we have done. A 2015 NFV survey paper [49] identifies many middlebox functions, such as Network Address Translation (NAT) and Deep Packet Inspection (DPI)

for intrusion detection, along with basic packet forwarding, as potentially suitable for commodity hardware implementation. Our work shows that CFINF falls in this category.

Finally, papers [50–54] focus on the engineering challenge of keeping up with high packet rates. Streaming algorithms, such as bloom filters, are used to improve efficiency and execution speed. However, these algorithms are summary based and do not provide the rates of individual flows.

3.7 Conclusions

Our work showed the feasibility of implementing a high-speed Cheetah Flow Identification Network Function (CFINF) on a high-performance multi-core commodity server. The CFINF accurately determines T-duration flight rates of flows from mirrored packets, and compares these flight rates to a set rate threshold R. For an identified cheetah flow, its flowID is sent to a Cheetah Flow Traffic Engineering System (CFTES). CFTES communicates with the SDN controller to set a filter rule in the router to isolate the cheetah flow to a separate queue in order to shield other flows from its adverse effects. Using real traffic traces collected by CAIDA on a 10-Gbps link, the performance of CFINF was evaluated. With 10 cores, CFINF could handle the 10 Gbps packet traces. A performance optimization of CFINF, which consisted of dropping small-sized packets, was tested, and found to result in only a 10% drop in the accuracy of reported cheetah flows, while offering significant computational savings (e.g., an analysis of 366 1-min traces showed that in the worst case, only 26.6% of flows would remain if length-based packet filtering was executed).

As link speeds have increased from 1 Gbps to 10 Gbps so has the capabilities of computers to use such high-speed links. Organizations will continue to deploy high-performance hosts with high-speed disks, memory, and NICs to leverage link capacity. Such data movements among hardware rich hosts will manifest as cheetah flows. Therefore, we can expect cheetah flows to have higher rates when link capacities are increased to 40 Gbps and 100 Gbps. The scalability of our CFINF design to accommodate for high-speed traffic is linear and is achieved by adding more CPU cores and memory. These aspects are not only meant for current networking needs but can be generalized to future NFV design.

Finally, the question of determining a rate threshold for cheetah flows was raised, and we describe our solution in the next chapter.

Chapter 4

A Pragmatic Approach of Determining Heavy-Hitter Traffic Thresholds

4.1 Introduction

In this chapter, we address the *problem of determining a rate threshold for cheetah-flow identification*. This problem is challenging because: (i) if the rate threshold is high, an un-redirected CF can cause packet drops or increased packet latencies for other flows, and (ii) if the rate threshold is low, then a significant number of flows will be classified as CFs, and a redirected CF suffers from lower throughput as its packets are sent to a lower-priority queue. The diurnal variation of link utilization requires the rate threshold to be dynamic (i.e., varying with time) instead of a static value. Several portions of this chapter is selected from our published work *A pragmatic approach of determining heavy-hitter traffic thresholds* [16]

Our *solution to this problem* is to have the CFTES measure background traffic characteristics, and base the rate threshold on average background traffic rate and a burstiness measure. We define the burstiness measure as the coefficient of variation (standard deviation divided by mean) of the flight-duration (T_1) number of bytes over a time interval T_2. The

flight duration T_1 is also used by CFINF to measure flow flight rates for CF identification. An example of T_1 is 100 ms and of T_2 is 1 hour. The time interval T_2 is set to a large value to avoid having to change the R threshold provided by CFTES to CFINF often since most of the CFINF processing power is required to keep with the high rate of packet arrivals.

A simulation model was used to determine the rate threshold as a function of various background traffic parameters and CF parameters. Our *key findings* from the simulation results are as follows: (i) Packet drop rate increases with increased burstiness, even when the average rate was same, with higher drop rate for higher flow RTT. (ii) The ratio of packet drop rate for a high-RTT CF to a low-RTT CF increases with increasing CF rates, indicating that high-RTT high-rate CF's are more severe (iii) Redirection of CFs using a low-priority scavenger queue results in reduced throughput, which is used to maximize the rate threshold for a certain level of acceptable packet loss rate.

The CFTES architecture and method for computing the rate threshold are described in Section 4.2. Section 4.3 presents our simulation study. Section 4.4 reviews related work. The chapter is concluded in Section 4.5.

4.2 Cheetah Flow Traffic Engineering System Architecture

Fig. 4.1 shows a deployment scenario for CFINF and CFTES. CFINF receives mirrored packets from a router and identifies high-rate (cheetah) flows from the live traffic. When identifying a cheetah, the CFINF sends the cheetah flow identifier to CFTES, which then communicates with a SDN controller to configure firewall filter rules in the router in order to isolate cheetah-flow packets to a separate queue and/or a different traffic-engineered path from the default IP-routed path. The expected use case for CFTES is enterprise access links that are typically under-provisioned due to cost reasons. User applications have no incentive to signal the network prior to sending a cheetah flow if the flow is carried on the same network as general-purpose traffic. It is for this reason that an in-network solution such as CFTES is a pragmatic approach for a provider to handle cheetahs in a manner that their adverse effects on other flows are limited.

In our prior work [55], we also proposed a complementary two-queue solution to CFTES, which allows enterprises to fill the provider-link headroom with cheetah flows without adversely affecting the provider's ability to meet its best-effort service-level agreements. By redirecting CFs to a lower-priority scavenger queue, the solution solves the problem of CFs having adverse effects on other flows, without incurring additional cost to maintain a separate IP/Optical path. Further, it prevents CF packets from be delivered out-of-sequence at the receiver, which in TCP causes the overall flow throughput to drop.

The CFTES consists of three main functions (see Fig. 4.1): (i) working with an SDN controller to perform redirection actions on identified Cheetah Flows using the Traffic Engineering Module (TEM), (ii) analyzing the port mirrored network traffic to create a database for predicting the rate threshold over a time period T_2, which is realized by the Traffic Analysis Module (TAM), and (iii) implementing bidirectional SNMP query and response to retrieve packet discard count through a Router Query Agent (RQA).

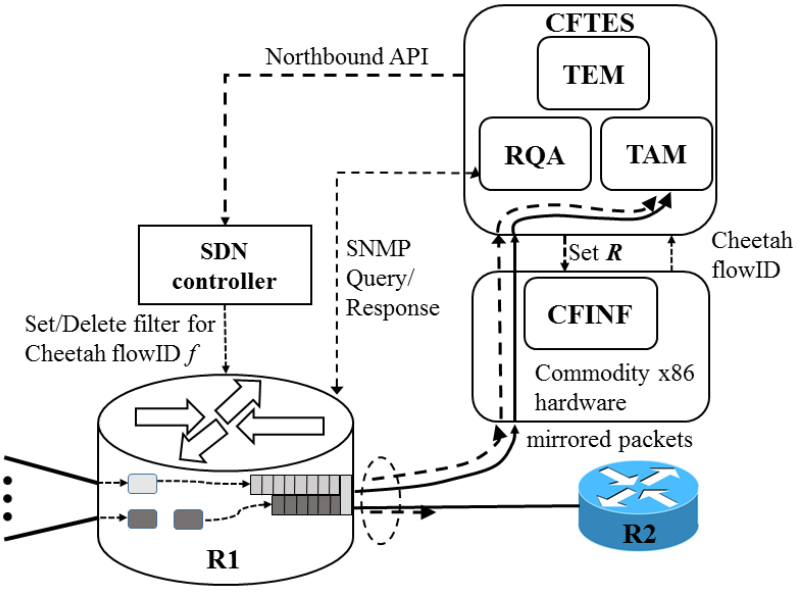


Figure 4.1: CFINF and CFTES deployment scenario (© 2018 IEEE)

The CFINF identifies CFs using the CF rate threshold, R, set by the CFTES. The value of the rate threshold depends on the characteristics of the background traffic, such as average rate and variability. To quantify the variability of background traffic, we define a metric named burstiness, b, whose definition was given in Sec. 4.1.

The CFTES-TAM along with CFTES-RQA is our *solution* in computing a dynamic rate threshold R. The CFTES-TAM receives a copy of the port mirrored packets from the router port and uses the traffic capture to perform two functions. *First*, in the Setup stage it uses a traffic capture of the i^{th} hour of the j^{th} day (represented as $\mathbf{P}_j^i$ captured during the hour D_j^i) to compute estimated packet drop rate values for different CF rates. The traffic $\mathbf{P}_j^i$ is replayed with different CF rates through a switch port with limited buffer capacity in a simulation environment, to estimate the packet loss rate l_{acpt} for some CF rate R, and is described in detail in Section 4.3.2. Fig. 4.2 shows the time period for which the traffic is captured, which is then used to compute the rate threshold R. A database or table is created where each entry forms a tuple (i.e., (l_{acpt}, R)) of computed packet drop rate l_{acpt} values for a specific CF rate R. $\mathbf{P}$ can be a simple vector of packet length and packet timestamp of the monitored traffic, using which the average background traffic rate $\overline{r}_{bg}$ and burstiness b is computed, to create a tag ($\overline{r}_{bg}$, b) for this database.

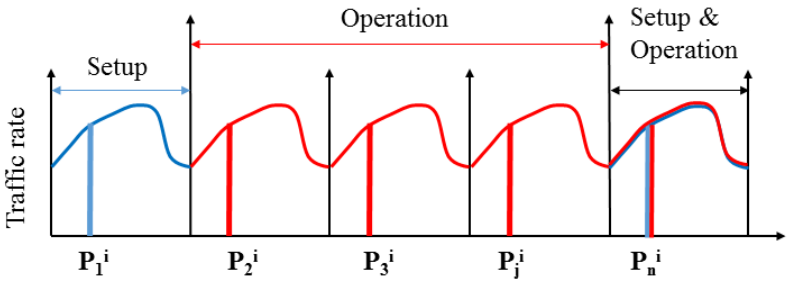


Figure 4.2: CFTES-TAM operation (© 2018 IEEE)

The rate threshold R is chosen, based on the acceptable packet loss rate of background traffic, l_{acpt}. Intuitively, the more CFs are redirected (i.e., a smaller rate threshold), the less packet losses occur in the background traffic. Given a value of l_{acpt}, there could be a range of values of R that result in background packet loss rates lower than l_{acpt}. However, setting R to a small value has some negatives. Due to the lower priority of the scavenger queue, a redirected CF will experience lower throughput compared to the case where it stays in the primary queue, which is shown in Fig. 4.7 in Section 4.3.2. Therefore, the optimal rate threshold should be the one that keeps the background-traffic loss rate lower than l_{acpt}, while redirecting as few CFs as possible. Since, background-traffic loss rate increases

monotonically with rate threshold, which is studied in Section 4.3.2, the optimal value of rate threshold is the maximum value of R (chosen from the database) that ensures the computed background-traffic loss rate no larger than the acceptable background-traffic loss rate.

In the *second* phase (or the Operation phase), the rate threshold for the hour in the next day D^i_{j+1} is predicted from the i^{th} hour of the past days. For example, in Fig 4.2, the rate threshold R that just results in an acceptable value of packet loss rate, l_{acpt}, which was computed from $\mathbf{P}^i_1$, is used to predict R to be used in D^i_2. The burstiness and average traffic rate is also computed for $\mathbf{P}^i_2$, which is used for improving the accuracy of the prediction of rate threshold for the next day.

For the next setting of R for hour D^i_3 the burstiness (i.e., b) and average rate (i.e., $\overline{r}_{bg}$), which was computed from $\mathbf{P}^i_2$ is searched in the database. If a combination of average traffic rate and burstiness (i.e., ($\overline{r}^k_{bg}$, b^k) tuple as k^{th} tag) exists in the database such that the distance between the last computed $\overline{r}_{bg}$ and $\overline{r}^k_{bg}$ is less than some small positive number ϵ_r, and the distance between the last computed b and b^k is less than another small positive number ϵ_b, then the database entries corresponding to the tuple ($\overline{r}^k_{bg}$, b^k) is searched to find R, such that the packet loss rate is just less than l_{acpt}. This value of R along with the previous R values for the i^{th} hour can be used in predicting the rate threshold for hour D^i_3. However, if such a tuple is not found in the database where the distance is less than ϵ_r and ϵ_b, it triggers the creation of a new Setup phase, to create new database entries of packet loss rate and rate threshold for a new traffic pattern characterized by the last computed burstiness b and average rate $\overline{r}_{bg}$ as shown in Fig. 4.2.

The *final* module of the CFTES is the Router Query Agent (RQA). It requests the packet discard counter and the received packets counter of the Router port that is being monitored, by using SNMP and computes packet discard rate. SNMP query and response latency is in the order of seconds (e.g., 10 sec). The average packet discard rate can be computed over a window of the last N queries, where N is an administrator configurable parameter. If this measured packet discard rate l_{msr} is higher than the acceptable packet loss rate l_{acpt}, then it could mean that a high-rate flow is using, or used, the same queue as the background traffic, which means that the rate threshold R estimated and chosen, was

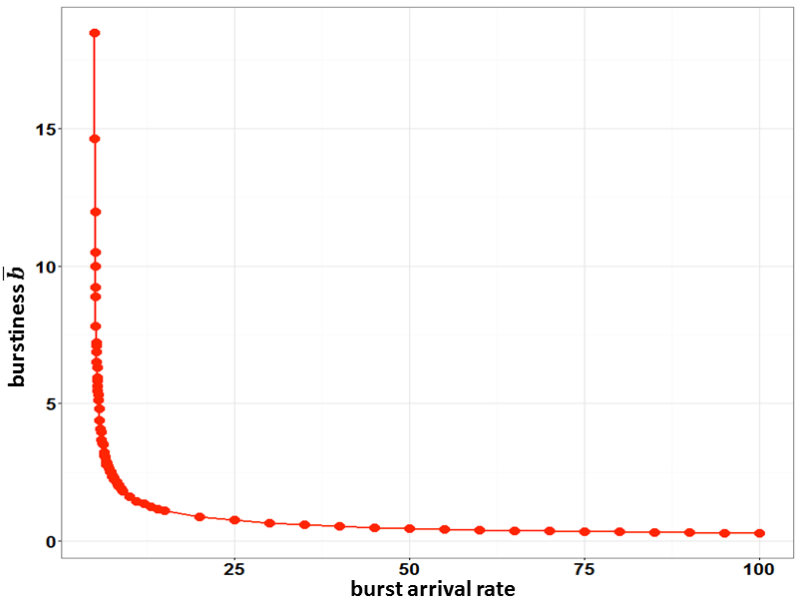


Figure 4.3: Mean burstiness against burst arrival rate, $\overline{r}_{bg} = 200$ Mbps, $d = 100$ ms (© 2018 IEEE)

high. A control action is therefore taken to reduce the rate threshold R by a small amount ΔR, so that packet losses are reduced.

4.3 Simulation Study

4.3.1 Simulation Setup

In this section we present the simulation setup. First we describe the model for background traffic, and then show the effect of model parameters on burstiness. Last we describe how the cheetah-flow redirection is implemented in our simulator.

Background traffic model To study the impact of Cheetah Flow on the general Internet traffic, we use a Poisson Pareto Burst Process (PPBP) model to generate background traffic, which provides a simple yet accurate approach to model network traffic [56].

The PPBP background traffic is characterized by four parameters: (i) burst arrival rate λ (bursts/sec), (ii) Hurst parameter h, (iii) mean burst duration d (sec), and (iv) a constant bit rate per burst r (bits/(bursts×sec)). The PPBP consists of an aggregation of bursts, where a burst is defined by a collection of packets with a constant inter-arrival time. The duration

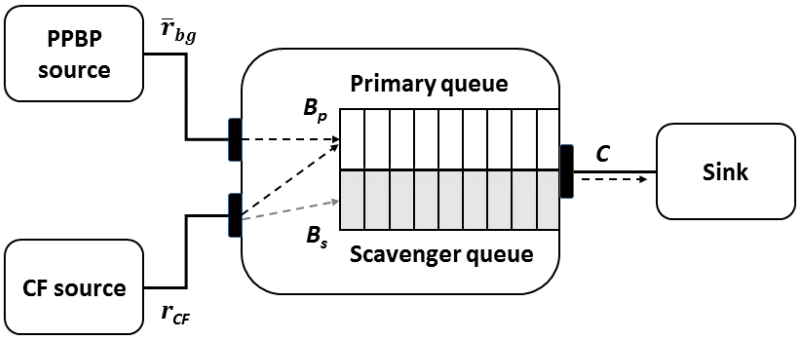


Figure 4.4: Simulation model (© 2018 IEEE)

of bursts is determined by a random variable of Pareto distribution of a shape parameter, α, and a scale parameter β. We assume that each burst consists of a positive integer number of MTU-sized (in bytes) packets. The number of packets within a burst is equal to $\lceil \frac{rt}{8MTU} \rceil$, where t is a sample value of the Pareto distributed random variable, and r is the average bit rate within a burst. Hurst parameter, h, defines if there is a long range dependency in a process, and equal to 0.5 for zero dependency, which is used in the simulation. In the PPBP model, the shape α and scale β are a function of the Hurst parameter, h and mean burst duration d, and the values used are presented in Table 4.1. The inter-arrival time of the first packets in bursts is exponentially distributed with an arrival rate of λ, i.e., the burst arrival process is Poisson.

The average rate of the PPBP background traffic, $\overline{r}_{bg}$, can be computed as [56]

$$\overline{r}_{bg} = dr\lambda \tag{4.1}$$

Burstiness The definition used for burstiness makes it a random variable. For a given value of average rate, burstiness is mainly determined by the burst arrival rate, λ, *or* the bit rate per burst r (since $r\lambda$ is a constant), and not impacted by the Hurst parameter. For example, consider an average background-traffic rate of 200 Mbps and mean burst duration of 100 ms. Fig. 4.3 shows how average burstiness $\overline{b}$ varies with burst arrival rate, where $\overline{b}$ is defined as the average values of burstiness obtained from 60 runs. The average burstiness decreases sharply for small values of λ, and approaches zero for large values of λ.

Cheetah-Flow redirection

This section describes the simulation model, shown in Fig. 4.4. The PPBP source generates the background traffic with an average rate of $\overline{r}_{bg}$, and the CF source generates a CF, rate limited to r_{CF}, with HTCP as a transport layer protocol. Packets from the PPBP traffic and CF are enqueued in this router buffer if there is empty space. The buffer is dequeued at the output link at rate C.

In this model, the router buffer is divided into one Primary queue of size B_p, and one Scavenger queue of size B_s. The Scavenger queue has a lower priority, and packets from this queue are scheduled to the output link only when there is no packet in the Primary queue. The background traffic load exhibits burstiness in short time periods where packets are resident in the Primary queue. Most of the time the Primary queue is empty as the average background traffic rate is less than the link capacity. Therefore, the Scavenger queue is always served, which prevents starvation in terms of cheetah flow packets scheduling. We developed a packet-level simulator, written in `Python`[1], to implement all the functions described above. Table 4.1 shows values used for the parameters in the simulation.

Table 4.1: Values for input parameters (© 2018 IEEE)

Parameter	Value
Background traffic rate, $\overline{r}_{bg}$ (Mbps)	200
Hurst parameter, h	0.5
Shape and scale in Pareto distribution, α, β	2, 0.05
Mean burst duration, d (ms)	100
Burst arrival rate, λ (bursts/sec)	{10, . . . , 600}
Bit rate per burst, r (Mbits/(bursts×sec))	{200, . . . , 0.003}
Router output link capacity, C (Gbps)	1
Primary queue buffer size, B_p (MB)	5
Scavenger queue buffer size, B_s (MB)	5
CF rate, r_{CF} (Mbps)	{50, . . . , 900}
Maximum RTT of CFs, RTT_{max} (ms)	{10, 100}

4.3.2 Numerical results

First, the impact of different CF max-rates on background-traffic packet drop rate was quantified when background-traffic characteristics were fixed. Next, we varied the background-traffic burstiness, while fixing the CF rate limit at 700 Mbps and average background traffic

[1]The simulation software is available in `GitHub` [57]

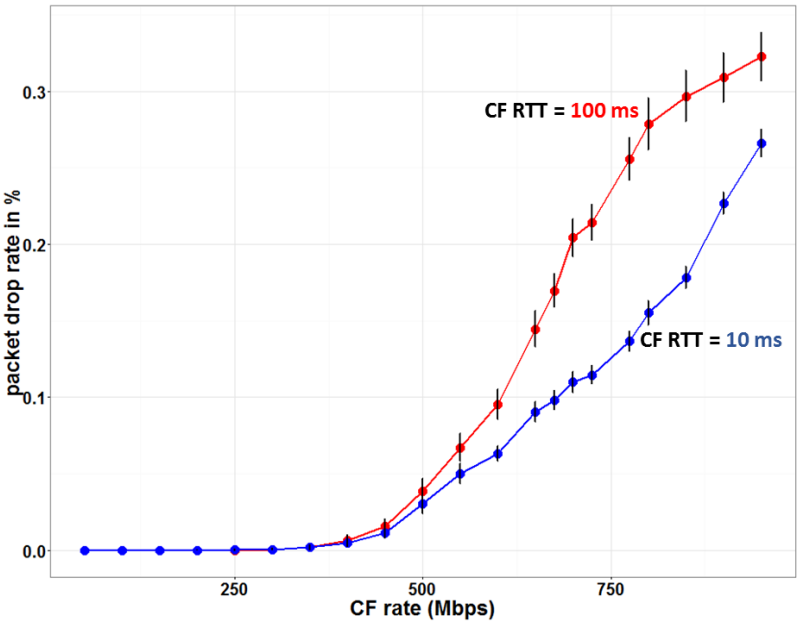


Figure 4.5: Background-traffic packet-drop rate against CF rate (© 2018 IEEE)

rate at 200 Mbps. Third, since CFs whose rate exceed threshold R are redirected to a scavenger queue, the adverse impact of such redirection on CF throughput is quantified.

The packet drop rate can be modeled as a function of (i) dynamic parameters such as background traffic rate, burstiness, CF-RTT, and CF rate, (ii) static parameters such as link capacity, and Primary and Scavenger queue buffer size. Simulation results over a wide parameter range can be used to form a model constructed with machine learning techniques. We consider this as a future work, since such a model would provide more insight about the CFTES sub-system.

Impact of CF rate on background-traffic packet-drop rate The generated background traffic and a single CF were both sent to the primary queue. The average rate of background traffic, $\overline{r}_{bg}$, was 200 Mbps, and the burstiness was fixed at 1.23. We varied the rate of CF, r_{CF}, from 50 Mbps to 900 Mbps, and considered two RTT values for the CF, i.e., 10 ms and 100 ms. For each value of the CF rate and CF RTT, 60 runs were executed, and the average and 95% confidence intervals of the background-traffic packet drop rate was computed.

Fig. 4.5 shows how the background-traffic packet drop rate varies with the CF rate. For both CF RTT values, the packet drop rate increases monotonically with the rate of CF with higher RTT CFs causing more packet drops. This is because most high-speed

data-transfer applications use large TCP sender and receiver buffers on high Bandwidth-Delay Product (BDP) (i.e., high RTT) paths to avoid sender-side waits for acknowledgments. Correspondingly, since the sender can send out a large congestion-window sized burst, the probability of the burst causing switch-buffer overflows increases for high-RTT CFs. For example, a 700-Mbps rate-limited CF on a 100-ms RTT path can create bursts as as large as 8.75 MB, and top-of-rack datacenter and enterprise switches often have small buffer sizes on the order of 5 MB as assumed in Table 4.1 [33].

Besides, other two interesting observations are made from Fig. 4.5. First, the packet drop rate caused by both the CF classes rises sharply for CF rate around 800 Mbps. This coincides with a headroom of zero as the average background traffic rate is 200 Mbps and the link rate is 1 Gbps. Second, even with a large headroom, non-zero packet drops occurred with a relatively high background-traffic burstiness of 1.23. For example, when the CF rate was 300 Mbps and the headroom was 500 Mbps, packet drop rates were 0.0005% and 0.0006% for the low-RTT and high-RTT flows.

Fig. 4.5 can be used as an illustrative example of how CFTES decides the rate threshold R to keep the background-traffic packet loss rate no larger than an acceptable value, l_{acpt}, given the burstiness and average rate values of background traffic. For example, in Fig. 4.5, if a packet drop rate of 0.0006% is acceptable, then R can be set to any values no larger than 300 Mbps when assuming RTT of 100 ms. However, among all the feasible values in terms of background-traffic packet loss rate, the optimal value that should be assigned as R is the maximum one, i.e., 300 Mbps in the example. This is because there is a disadvantage to CFs when the CFINF rate threshold R is lowered. A later section illustrates this disadvantage.

Another rule used by CFTES to decide the rate threshold is that the value of R is based on the results of high-RTT CFs. In Fig. 4.5, the ratio of the packet drop rate between the high-RTT CF to the low-RTT CF is not constant. It increases from 0.54 to 1.9 when the CF rate changes from 200 Mbps to 800 Mbps, while decreasing from 1.9 to 1.2 for CF rates above 800 Mbps. It is challenging for an online system such as a CFINF to compute the RTT of TCP flows, and therefore, it is prudent to set the CFINF rate threshold R to a low-enough value based on the results of high-RTT CFs so that even a high-RTT CF does not cause an unacceptably high packet drop rate.

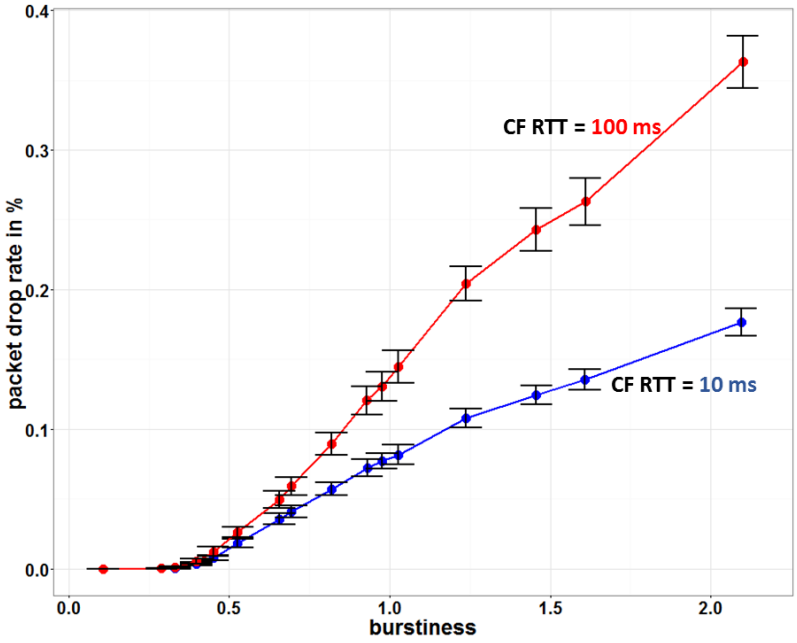


Figure 4.6: Background-traffic packet drop against burstiness (© 2018 IEEE)

Impact of burstiness on background-traffic packet-drop rate In this experiment, there was still a single CF sent to the primary queue. The CF was rate limited to 700 Mbps, i.e., $r_{CF} = 700$ Mbps. Since the average background-traffic rate was 200 Mbps (20% link utilization), the average headroom on the 1 Gbps link was only 100 Mbps (see Fig. 4.4). In general, with link utilization known, these experiments can be conducted for different link speeds (i.e., 1 Gbps, 10 Gbps, and 100 Gbps). The burst arrival rate λ was varied from 10 to 600 bursts/sec to generate burstiness values in the range of 0 to 2.09, while the rest of the parameters were left unchanged from the values indicated in Table 4.1.

Fig. 4.6 shows how CFs of different RTTs impact background-traffic packet drop rate for different values of background-traffic burstiness. Each point in the plot (e.g., a red point for CF RTT 100 ms) is the average value obtained from 60 runs, and the error bars show the 95% confidence intervals. We observe that even with a headroom of 100 Mbps, background-traffic burstiness can cause significant packet drops.

With the same level of burstiness, a higher-RTT CF causes more packet drops than a lower-RTT CF. As described above, the TCP congestion window is typically larger for high-RTT flows, which in turn, increases the probability of switch buffer overflows and packet drops.

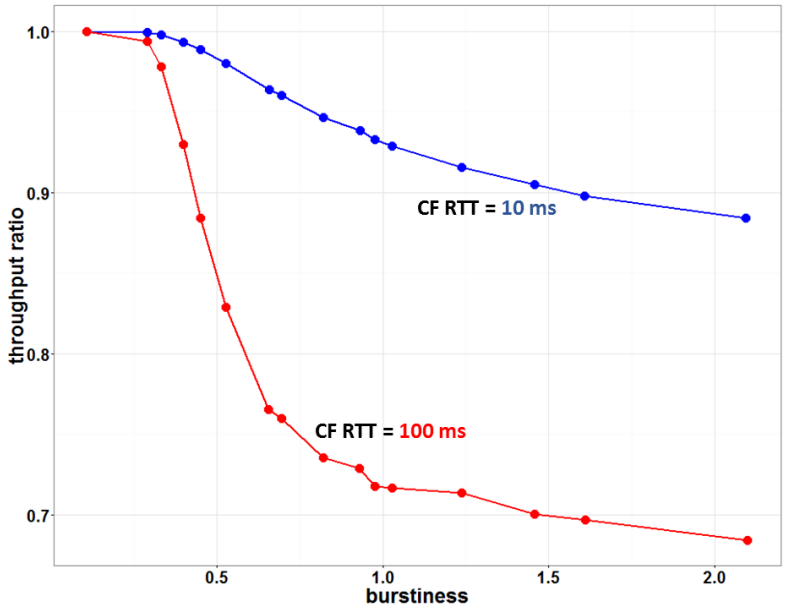


Figure 4.7: Ratio of CF throughput (© 2018 IEEE)

At a background-traffic burstiness level of 0.33 (which corresponds to a burst-arrival rate λ of 600 bursts/sec), there were no packet losses for both the high-RTT and low-RTT CFs of rate 700 Mbps. This implies that with an average headroom of 100 Mbps, if background-traffic burstiness is less than 0.33, a CF does not cause traffic loss in the background traffic. Even for a λ of 80 bursts/sec, the high-RTT CF causes a background-traffic packet-drop rate of only 0.0008%, while the low-RTT CF causes a 0.0005% packet drop rate, both of which are likely to be acceptable levels of packet loss. However, at burstiness levels of 1.2, packet loss rates can be significant, e.g., 0.2%. This loss rate can be understood in the context of an ESnet-reported HTCP measured throughput drop from 10 Gbps to less than 1 Gbps on a 50-ms RTT path when the packet drop rate was 0.0046% [17].

Based on these findings, the CFTES management module is designed to (i) periodically measure the background-traffic burstiness and average rate, (ii) estimate the expected background-traffic burstiness and average rate for the next period, and (iii) based on the maximum tolerable limit for background-traffic packet drop rate, set the rate threshold R used by the CFINF.

A final observation from Fig. 4.6 is that at low levels of burstiness, the high-RTT CF caused a background-traffic packet drop rate that was 1.5 times more than the low-RTT CF, but at the highest value of burstiness considered (2.09), this factor almost doubles. This also

justifies our choice of setting the CFINF rate threshold R to a lower-enough value based on high-RTT CFs as in the previous section. However, there is a disadvantage to CFs when the CFINF rate threshold R is lowered. The next section illustrates this disadvantage.

Impact of CF redirection on CF throughput When CFs are redirected to the scavenger queue, their packets receive a lower priority than packets from general-purpose flows. Therefore, the throughput of redirected CFs will be smaller than non-redirected CFs. The lower the rate threshold R, the larger the number of CFs that will be adversely impacted. Specifically, this section presents the impact of background-traffic burstiness and CF RTT on the ratio by which CF throughput decreases as a result of CF redirection.

The simulation setup was the same as that described in Section 4.3.2. CF throughput was determined under two scenarios: (i) Scenario 1: CF packets were sent to the primary queue as before; (ii) Scenario 2: CF packets were sent to the scavenger queue as shown in Fig. 4.4. The 5-MB buffer was divided equally between the primary and scavenger queues in Scenario 2. Fig. 4.7 shows the ratio of the CF throughput in Scenario 2 to the CF throughput in Scenario 1 for different values of background-traffic burstiness and two values of CF RTT. As in the results presented in Section 4.3.2, average values obtained from 60 runs are presented for all points.

The trade-off associated with redirection in order to prevent background traffic from experiencing packet loss can be clearly seen in a reduction of CF throughput. This reduction is especially high for high-RTT flows, e.g., for our parameter settings, when background-traffic burstiness is even just 1, CF throughput drops by 30%.

In Section 4.3.2, we noted that the maximum tolerable limit for background-traffic packet drop rate should be considered by the CFTES when computing the rate threshold R to be used by the CFINF. The results presented in this section show that the rate threshold R should be set as high as possible so that fewer CFs are subject to redirection, since redirection could lead to a drop in CF throughput.

4.4 Related work

According to Lan and Heidemann [58], heavy-hitter flows are associated with elephants, which are large sized flows, and cheetahs, which are high-rate flows. High-rate or Cheetah Flows that were elephants shows a correlation of 72% in one of the traces. Therefore, we compare our work with existing literature that defines an elephant flow, and a high-rate flow.

An elephant flow is a flow that exceeds some value of a size threshold [59, 60], whereas if the rate of a flow exceeds some specified rate threshold [61, 62] then it is classified as a high-rate flow. The definition of these flows does not include factors like link utilization and/or packet loss rate, resulting in a static definition of a heavy-hitter threshold.

Traffic feedback parameters are critical for defining a heavy-hitter as it can be changed dynamically based on a closed loop control system to maintain system stability, and only a handful of prior work investigates this problem. The distribution of flow sizes that are found in a datacenter presents a bimodal distribution due to the presence of mice an elephant flows. The intersection point of the two probability distribution curves for flow sizes provides the optimal elephant flow size threshold [63]. This is an initial step towards closing the loop for elephant flow detection.

The authors in *FuzzyDetec* [64] use fuzzy logic to continually compute the threshold for elephant flows based on current network utilization and the load on the controller for computation and provision of routes for redirecting an elephant once it is detected. Our method uses the background traffic characteristics to compute a heavy-hitter threshold.

4.5 Conclusions

In this chapter, we demonstrate a pragmatic approach to determine a Cheetah Flow redirection threshold. We found that the burstiness of the background traffic, network utilization, and high-rate flow RTT are key factors in computing this threshold. Increase in burstiness of the background traffic for the same network utilization (i.e., average rate) increases the background traffic packet loss rate.

We provide a solution for computing the rate threshold using control feedback techniques. The solution maximizes the rate threshold to avoid classifying too many flows as cheetah flows, thereby avoiding redirection and reduction of throughput. At the same time, the solution aims to keep packet loss rate within acceptable limits. We conclude that the choice of the rate threshold should not be lower than is necessary to tolerate an acceptable degree of packet loss rate.

Chapter 5

A network service for diagnosing throughput problems

5.1 Introduction

This chapter describes a network service that identifies hosts that transmit large datasets at low throughput and identifies the root cause of such issues. Large data transfers generate high volume network traffic in an end-to-end connection, which are known as cheetah flows. The throughput of a large data transfer is an indicator of the performance of the hosts involved in the transfer as well as the network connecting the hosts. Network parameter values in a transfer are difficult to obtain as a transfer may cross multiple domains and network paths change based on the traffic load and routing policies. Even for a single domain, network parameters are not shared. Network tomography methods [65] may be applied in such cases to determine network parameters. The parameters can also be modeled as latent variables. Due to the limited visibility of network parameters, we focus our work on identifying resource limitations in the end host. We divide the work into two parts: (i) The *first* component describes a network service that will monitor large data transfer throughput using a combination of transfer logs and host resource utilization logs. From these logs, the system creates a Performance Root Cause Analysis Report, which users of any transfer can view. The notion of a peer transfer pair is established and is used to shed light on the

impact of resource and parameter settings for poor performance. A peer transfer pair is two or more host pairs that transfer data through the same network. (ii) The *second* component of our work demonstrates, through a set of real-world experiments, how comparison of peers can reveal and help isolate the root cause of low throughput. The throughput of a data transfer is compared between peers that share network paths. This can reduce the variation of the throughput caused by different network paths without much influence from network parameters. Such an approach is simple and practical to implement, as transfer host pairs are compared against each other, and users of these hosts are notified that a different pair of hosts are efficiently using network bandwidth and enjoying better performance. Such a notification would make an user cognizant that their transfers are slower than others. With this information the users can consult the users of those peer pairs that have better performance to gain insight into the transfer bottleneck. We propose a network service that collects data transfer logs and host performance metrics from end hosts to identify and diagnose performance problems.

Section 5.2 describes the Cheetah Flow Throughput Monitoring System (CFTMS) architecture and deployment. Section 5.3 shows the method of identifying low throughput transfers by comparing transfers between different peers on the network. We provide a set of controlled experiments to show how the CFTMS can identify hosts with resource limitations by comparing transfer pairs in Section 5.4. Related work is reviewed in Section 5.5. Finally, key conclusions are provided in Section 5.6.

5.2 Cheetah Flow Throughput Monitoring System

Fig. 5.1 illustrates the deployment of an Cheetah Flow Throughput Monitoring System (CFTMS) in a provider network. The throughput of a data transfer is the ratio of the amount of data transferred to the duration of the transfer. The CFTMS receives file transfer logs and resource utilization logs from data-transfer end hosts, and network data from IPFIX/NetFlow collectors and SNMP managers. While a user/administator can monitor the transfer logs from their own end hosts, a provider-deployed CFTMS can aggregate information from many data-transfer end hosts, and potentially integrate network information from multiple

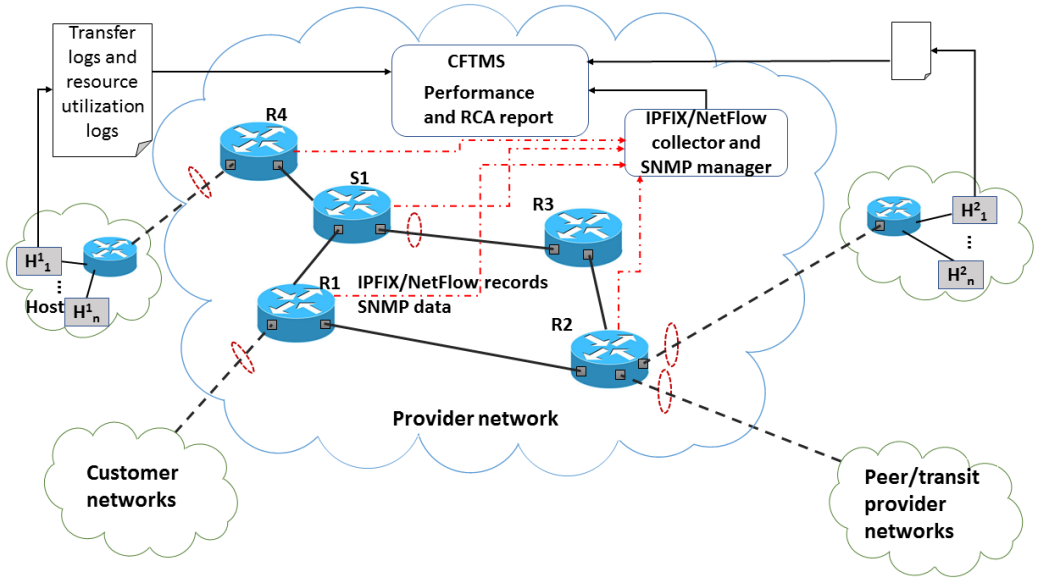


Figure 5.1: Cheetah Flow Throughput Monitoring System Deployment

provider networks to perform a root cause analysis. Since the bottleneck of a data transfer could be one of the two end hosts or any network link on the end-to-end path, the CFTMS can view all information and provide better diagnostics.

A Throughput Monitoring System-client (TMS-client) is executed on end hosts to collect and push data-transfer logs, which list data-transfer size and duration, to the CFTMS. The TMS-client also includes low-overhead resource-utilization monitoring scripts to measure CPU, disk, memory, and network utilization. The TMS-client collects data-transfer parameters, such as the maximum TCP buffer size, number of TCP streams, and the application type. Prior work on instrumentation of data transfers includes NetLogger [66], a low-overhead event logger, and SNAP [67], a scalable network-application profiler that logs TCP socket-level information. A TMS-client implementation could use these frameworks for log collection.

Network monitoring data may include SNMP measurements, such as bits/sec and packet-discard rates, and IPFIX/NetFlow records for each link of a data-transfer path. This network data is collected by an SNMP manager and IPFIX/NetFlow collector, as illustrated in Fig. 5.1. Using this information, the CFTMS can determine which link on the data-transfer path, if any, is congested and hence a source of packet loss.

The CFTMS identifies hosts that are often engaged in low-throughput data transfers,

and generates Performance and Root Cause Analysis (PRCA) reports. Users or data-transfer server administrators can download the PRCA reports for their hosts, or the CFTMS could use a publish-subscribe model to send out PRCA reports and users/administrators could subscribe to these reports. A method for identifying poorly performing data-transfer hosts is described in the next two sections.

Data-transfer applications, such as GridFTP, log data-transfer size and duration, from which transfer throughput can be computed. However, for the CFTMS to use end-host transfer logs, users need to subscribe to the CFTMS service, and install/execute the TMS-client on their data-transfer servers. To avoid this overhead, we considered whether throughput could be determined from IPFIX/NetFlow records. Through a set of experiments, we found that this is not feasible for all kinds of transfers. Large flow transfers are characterized by a continuous stream of packets with gaps that are only caused by TCP congestion control. However, multiple interactive data transfers could occur in a persistent HTTP client-server session, i.e., one that uses the same TCP connection. User think times between the multiple data transfers could cause long silence periods on such persistent TCP connections. NetFlow active timeout intervals are typically set to 1 minute. There could be multiple think-time idle periods within 1 minute. Furthermore, longer-duration TCP connections will have multiple NetFlow records. Sizes reported by multiple NetFlow records for a given TCP connection cannot simply be added to determine transfer sizes because of the same think-time idle period problem. Hence per-transfer throughput cannot be accurately estimated from NetFlow records (even if the records were created without packet sampling). The same reasoning explains why transfer throughput cannot be estimated even if all packets of a flow (TCP connection) are captured for analysis inside a provider's network. Effectively, user/administrator participation in the CFTMS service is essential, as transfer logs are required from the end hosts to compute throughput.

Another issue is that there are often multiple providers on the end-to-end path of a WAN data transfer. If any one of these providers deploys an CFTMS, ideally the CFTMS should receive IPFIX/NetFlow records and/or SNMP link-level packet-discard measurements from the remaining providers. However, for privacy reasons and competitive considerations, providers may be unwilling to share their collected data with others. But without this

information, the CFTMS may not be able to provide an accurate root-cause analysis.

5.3 Peer transfer pairs and throughput comparison

This section defines a peer-transfer pair. In Fig. 5.1 Customer Network 1 connects n hosts i.e., H_1^1 through H_n^1 and Customer Network 2 connects m hosts that are H_1^1 through H_m^1. A peer transfer pair $\mathbb{P}$ is defined as any combination of transmit and receive host pairs, such that the data transfer traverses the same network paths.

$$\begin{aligned} \mathbb{P} =& \{(H_i^x, H_j^y), (H_k^x, H_l^y), (H_p^x, H_q^y), ...\} \\ & s.t. \forall\ i,k,p\ \in\ \{1\ .\ .\ .,\ n\};\ j,l,q\ \in\ \{1\ .\ .\ .,\ m\} \\ & where\ |\mathbb{P}| \leq m \times n \end{aligned} \tag{5.1}$$

where H_i^x is the i^{th} host in Customer Network x and H_j^y is the j^{th} host in Customer Network y. A data transfer pair (H_i^x, H_j^y) represents a data transfer from H_i^x to H_j^y and the rate of the data transfer is given by $r_{i\to j}^{xy}$. The set $R_{i\to j}^{xy}$ is a collection of all the data transfer throughputs $r_{i\to j}^{xy}$ experienced by the hosts H_i^x and H_j^y.

A comparison of the throughput sets for different host pair combinations (where the two hosts are in networks x and y) $\{R_{i\to j}^{xy}, R_{k\to l}^{xy}, R_{p\to q}^{xy}, \ldots\}$ provides information regarding the distribution of the highest throughput values that a host pair achieves.

Since a particular host pair can have users who may choose different transfer applications (or different settings of protocol parameters), the throughput values that these users achieve would be different. The throughput values may be widely distributed or may appear in groups or clusters. Clusters form due to different static parameter settings. Transfer application type, number of streams, maximum sender disk read rate, sender NIC rate, sender TCP window size, maximum receiver disk write rate, receiver TCP window size, and receiver NIC rate can be classified as static parameters. Dynamic parameters include host CPU utilization, host disk utilization, and host network utilization of the sender and receiver by the data transfer process. Since other concomitant processes can utilize CPU, disk, and network resources during the transfer, these are variable and hence dynamic in nature. If the

effect of dynamic parameters are more dominant than the effect of static parameters then the throughput values will have a greater variance without forming any groups or clusters. In such a scenario, the data transfer throughput model will be used. On the other hand, a narrow variance in transfer rates indicates the dominant effect of static parameters and a cluster-based technique can be used for throughput comparison.

Assuming that the dominant effect is from static parameters, the number of clusters can be determined by simply keeping a count of the different combination of static parameters used. For example, consider a case where the same sender and receiver are used by two different users, and the first user uses four TCP streams, whereas the second user uses one TCP stream. The other static parameters in the end hosts remain the same, meaning there are two combinations of static parameter setting used, and hence the possibility of throughput values forming two clusters. The clusters should also have zero or minimal overlap of throughput values, to strengthen the definition of a cluster. The cluster that has the highest throughput values in $R_{i \to j}^{xy}$ represent the achieved throughput performance of the host pair. Let the centroid or median of the cluster with highest throughput be given by $\Re_C^{i \to j}$, where C is the cluster with high values of transfer throughput. Therefore, the cluster centroid C provides the maximum achieved throughput to, which the users in other clusters would aspire.

For a single host pair (H_i^x, H_j^y), the difference in the cluster means could indicate that users transfer data using different applications (i.e., `scp` versus `iperf`), varying number of TCP streams, or that protocol parameters are set improperly (i.e., small TCP window size). An experiment using a transfer pair with two different TCP stream settings for the same application shows this behavior. Comparing host static parameter settings (extracted from end host transfer logs) from the cluster with low rates to the cluster with high rates can identify the static parameter setting that resulted in low throughput and construct an PRCA report. The transfer log would also keep track of the user performing the transfer. Finally, the PRCA is published to the user of the host informing them of the slow transfer. For a peer transfer pair (i.e., $\{(H_i^x, H_j^y), (H_k^x, H_l^y)\}$) the solution is non-trivial. Assuming that transfers between these pairs take place through the same provider network path, such that network parameters do not result in the difference of transfer throughput between the

```
traceroute to 144.92.42.116 (144.92.42.116), 30 hops max, 60 byte packets
 1  cr01-udc-ae10-15003.net.virginia.edu (128.143.230.2)  0.202 ms  0.195 ms  0.186 ms
 2  br01-udc-xe-1-2-1.net.virginia.edu (128.143.236.5)  0.282 ms  0.286 ms  0.276 ms
 3  equinix-peering-vt.net.virginia.edu (192.35.48.30)  2.972 ms  2.996 ms  3.037 ms
 4  10gigabitethernet2-2.core1.ash1.he.net (206.126.236.37)  2.494 ms  12.295 ms  2.489 ms
 5  100ge12-2.core2.chi1.he.net (184.105.64.241)  29.117 ms  28.946 ms  29.143 ms
 6  wiscnet.v960.core1.chi1.he.net (216.66.3.22)  73.843 ms  72.932 ms  72.913 ms
 7  r-uwmilwaukee-hub-et-1-1-0-2254.uwsys.net (143.235.42.6)  38.235 ms  38.232 ms  38.223 ms
 8  r-uwmadison-hub-et-2-1-0-3700.uwsys.net (143.235.33.22)  38.253 ms  38.300 ms  38.292 ms
 9  uwmadison-sitemanaged.uwsys.net (143.235.40.1)  32.462 ms  33.646 ms  33.642 ms
10  fdt-wisc.net.wisc.edu (144.92.42.116)  31.909 ms !X  31.887 ms !X  31.903 ms !X
```

Figure 5.2: Traceroute output from FDT Uva to FDT Wisc.

peer transfer pairs, a discrepancy could arise from the difference in host capacity. In our experiment (experiment number 1 and 2) in Section 5.4.2, we show that in a peer transfer pair ($\{(H_i^x, H_j^y), (H_i^x, H_l^y)\}$), where the sender was the same, the difference of the highest mean cluster throughput was due to a receiver that had slow disks. This information can then be provided to the other host pairs between which transfers were observed, to notify these users of low performance.

5.4 Experimental study

5.4.1 Experimental setup and controlled experiments

Table 5.1: Experimental setups and Transfer-throughput statistics in Mbps

No.	App.	Str.	Sender	Receiver	Min	25%	Med.	Mean	Max	cov
1	iperf	1	FDT(W)	FDT(V)	364.6	589.1	629.7	614.6	743	0.121
2	iperf	1	FDT(W)	IDC(V)	137.5	200	214.4	210.4	239.6	0.096
3	iperf	4	FDT(W)	FDT(V)	577.9	935.2	940.9	929.5	953.7	0.061
4	iperf	4	FDT(W)	IDC(V)	113.2	226	258	245.6	282.5	0.137
5	scp	1	FDT(V)^{cpu}	FDT(W)	111.4	120.7	126.6	127.2	151	0.071
6	gridFTP	4	FDT(W)	FDT(V)^{cpu}	461.3	534.5	558.3	565	736	0.099

This section describes the different experiments that were conducted to demonstrate CFTMS utility as a network service. Two hosts, named FDT and IDC, at the University of Virginia (UVa or V, for short) and one host, named FDT, at the University of Wisconsin (UWisc or W, for short) were used in these experiments. The path between the two universities traversed the commercial Internet, as illustrated in Fig. 5.2.

A large file (specifically, a file larger than 10 GB) was transferred in each experimental setting. There were six different experimental settings, as listed in Table 5.1. Variables in

the experiment include: (i) sender/receiver hosts, (ii) data-transfer application (`iperf`, `scp`, `GridFTP`), (iii) number of streams (parallel TCP connections), and (iv) whether or not the CPU was loaded (represented as FDT(V)^{cpu}).

Multiple runs were executed under each experimental setting over a 24-hour period with an idle interval of 30 minutes between each run. The experiments were executed over a 6-day period. Running experiments over a 24-hour period allowed us to control for the diurnal effects of Internet traffic. Data transfer logs were recorded and used to compute throughput.

The RTT between either host at UVa and the host at UWisc was 33.5 ms on average. In all the experimental settings, the TCP buffer sizes at the sender and at the receiver were four times the bandwidth delay product (BDP), where the BDP was computed with a bottleneck link rate of 1 Gbps. This setting is based on our previous findings on the TCP sender-side and receiver-side buffer sizes in Linux, which halves these buffer sizes so that the congestion window of a flow can increase to 2 $\times$ BDP. A minimum of 2 $\times$ BDP for the congestion window is necessary to ensure that hosts do not suffer from low throughput performance due to poor choice of protocol parameters. The congestion-control algorithm used was HTCP.

Fig 5.2 shows the `traceroute` output from FDT(V) to FDT(W). The end-to-end path traversed networks operated by four different administrative organizations (domains). We did not have access to SNMP data and/or NetFlow records from these providers, which limited our analysis to end-host resource limitations and TCP- and application-layer parameter settings.

5.4.2 Analysis of results

Table. 5.1 shows the statistics of throughput achieved between the sender and receiver hosts for different data transfer applications used (i.e., `iperf`, `scp`, `gridFTP`). It also shows how the different parameter settings used for the same application can cause differences in transfer throughput (e.g., the median throughput for four `iperf` streams from FDT(W) to FDT(U) is 940.9 Mbps, whereas the same application using one stream enjoys a median throughput of 629.7 Mbps) described as Case 1. In a peer transfer pair, Experiment 3 and Experiment 4 reveal resource limitations of the receive host, which is explained as Case 2. The effect of burdening the send and receive host with a CPU intensive operation can be

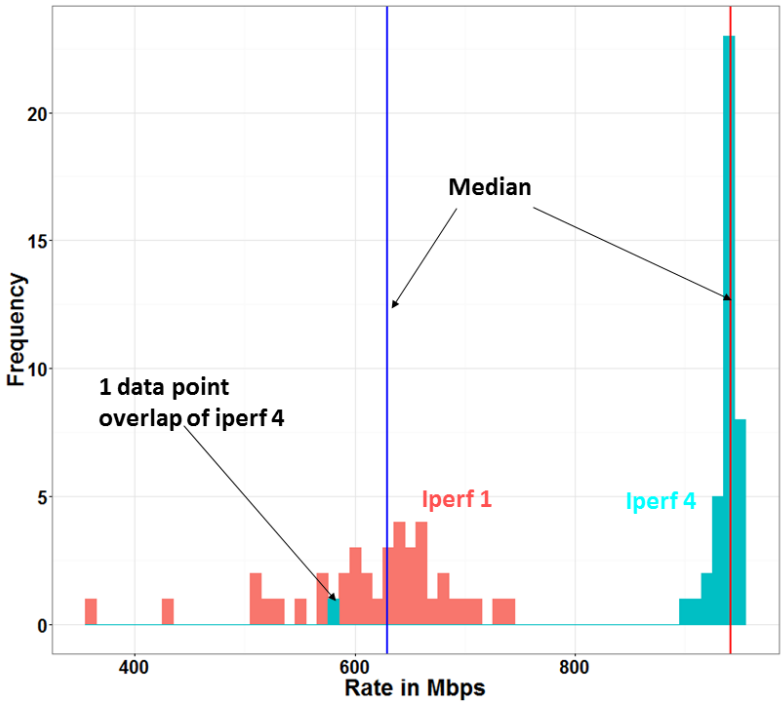


Figure 5.3: Throughput histogram from experiment 1 and 2

seen in Experiment 5 and Experiment 6, respectively, referred to as Case 3. These three different cases are analyzed below. Case 1 and Case 2 assume that static parameters are dominant in the transfer, but Case 3 illustrates the impact of dominant dynamic parameters.

Case 1: Impact of improper static parameter settings: Experiment 1 and Experiment 3 show the effect of using a different number of TCP streams. The end hosts are the same in these experiments. Such a scenario can be compared to two users transferring data between the same end hosts. User 1 does not use any concurrent connections for his/her data transfer (as in Experiment 1). On the other hand User 2 creates four parallel streams as in Experiment 2 for his/her data transfer. Comparing the parameter values from sets of transfers generated by these two users will show only two combinations of static parameter settings being used. Hence, throughput values would likely form two clusters.

Fig 5.3 shows the histogram of throughput values for the user using 1 `iperf` stream and the user using 4 `iperf` streams. There is almost no overlap between these throughput clusters with only 1 (out of 40 data points) data point from the higher cluster falling within the range of throughput values for the lower cluster. The minimum overlap implies that there is a difference in values of static parameters for these two experiments. Comparing the

static parameters will reveal the cause of low throughput. The transfer logs also contain the identity of the user, hence a notification can be sent to User 1 with a suggestion to increase the number of streams to achieve better throughput.

Comparing these two experiments not only reveals that using more streams results in a high median throughput, but the spread of the throughput or (throughput variance) is lowered, with a single stream having a coefficient of variation (`cov`) of 0.121 indicating a much larger spread (i.e., almost two times) than a transfer with four streams with a `cov` of 0.061. This can also be seen from the histogram.

Case 2: Impact of a resource limited host: Experiment 1 and Experiment 2 resemble data transfers using the same application settings that result in two transfer peers (i.e., (FDT(W)→FDT(V), FDT(W)→IDC(V)), with a common receive host. The median throughput for FDT(W)→FDT(V) is 629.7 Mbps, which is much larger than the median throughput of FDT(W)→IDC(V) (214.4 Mbps). The host pair FDT(W)→FDT(V) indicates that there is a performance issue with pair FDT(W)→IDC(V), which was indeed the case. Upon diagnosis of the host IDC(V) we found that it incurred almost 100% disk utilization during the transfer suggesting that the IDC(V) host was writing at maximum disk speed. Despite increasing the number of streams from one to four as in Experiment 4 the median throughput of FDT(W)→IDC(V) transfers remained low at 258 Mbps, implying that application setting did not contribute to reduced throughput. The dynamic parameter for transfers between FDT(W)→IDC(V) would show a 100% disk utilization by the receiver caused by the transfer process.

Similar to the previous case, upon comparing the dynamic parameters (dynamic parameter is required for finding the resource that is almost completely utilized by the transfer process) for these two transfers (i.e., Exp. 1 and Exp. 2), it is clear that the maximum disk write speed of IDC(V) is lower than the maximum disk write speed of FDT(V). Users who transfer data between FDT(W)→IDC(V) would be unaware that, in practice, high throughput values for transfers are obtainable. Information from a peer transfer would therefore be valuable to determine the low performing host pairs and also for constructing a root cause for the performance issue. This is a real-world scenario where transfers between two different host pairs are compared against each other to find which one is performing better and,

subsequently, analyzing the cause of this performance gap.

Case 3: Impact of dynamic parameters: Experiment 5 and Experiment 6 show the impact of computationally burdening the sending and receiving hosts, respectively. Experiment 5 uses an `scp` application and Experiment 6 uses a `gridFTP` application to demonstrate the effect of CPU burdening on transfer throughput. Since these experiments were performed on hosts using a real network, the impact of CPU taxing is unclear due to the inherent noise that results from other dynamic parameters such as changing network utilization. These two experiments show that there is an impact on throughput due to CPU loading when Experiment 6 is compared with Experiment 3. The receiver host CPU in Experiment 3 was not loaded so the transfer throughput rates are high. Although Experiment 3 and Experiment 6 use different applications, `iperf` and `gridFTP` perform similarly from a functional perspective. Due to changes in the file transfer library in FDT(W) over our experimentation period `iperf` had failed to work and hence the change to `gridFTP`.

5.5 Related work

To identify throughput performance problems, various solutions have been proposed and implemented. These solutions either use data from active measurement, passive measurement data, or a combination of both. Our work uses data from passive measurement. Active measurement involves running bandwidth probing experiments using test nodes, which are invasive and alter the nature of normal traffic. The perfSONAR (pS) [68, 69] infrastructure is a widely-deployed set of perfSONAR nodes used by science networks for active traffic measurements. These perfSONAR nodes run a toolkit called Bandwidth Test Controller (BWCTL) [70], which measures throughput, among other network metrics. Thousands of perfSONAR buoys are deployed in large RENs like Internet2 and ESnet, which enables the network administrator to run throughput tests between two geographically distant perfSONAR hosts. In contrast, our CFTMS relies on passive measurements by collecting end host resource utilization, data transfer logs, and network measurement data (i.e., SNMP statistics, NetFlow/IPFIX records). The perfSONAR measurements do not account for the performance of hosts that are involved in a large data transfer. Therefore, a transfer that is

limited by hardware capabilities results in degradation of throughput [71] where the network is not the bottleneck. In other words, the BWCTL throughput observed between two perfSONAR nodes is not an accurate measure of the data transfer throughput. Our method properly diagnoses that the limitations of end host are the cause of poor performance.

GridFTP [28] is a popular data transfer application for moving large datasets whose application logs can be analyzed to monitor throughput performance degradation as a class of passive data measurement. GridFTP transfer logs were used to find throughput variance [72] and performance anomalies [73, 74]. However, transfer logs have limited information and cannot provide a Root Cause Analysis (RCA) of a low transfer throughput. Our solution addresses this through resource utilization logs and network data.

The authors in another work [75], integrate perfSONAR measurements with GridFTP logs (using NetLogger) to detect bottleneck conditions of a data transfer. The authors present a tool called Periscope [76], which can localize the bottleneck of the data transfer. Using active network measurements from perfSONAR deployments the authors design a system to identify the RCA and localize performance problems in ISP networks [77]. Other work [78–80] also address the problem of identifying and localizing network issues and rely on data from active perfSONAR measurements. Our work is fundamentally different because it uses passive measurements and real performance data from actual transfer hosts. We solved the problem of obtaining network topology and link statistics (e.g. loss rate, bottleneck link) from multiple domains owned by different network providers by showing a peer throughput comparison model that is both novel and pragmatic.

5.6 Conclusions

This work proposed a network service for monitoring and diagnosing data transfer throughput performance. The network service uses a Cheetah Flow Throughput Monitoring System (CFTMS) to collect transfer logs, host utilization logs, and network data. Using this data, a low throughput transfer can be identified and a PRCA for reduced performance can be created to help users diagnose the problem. We described the effect of static and dynamic parameters on transfer throughput and presented a method to compare peer transfer throughput.

To address the limitations of collecting network data, we proposed a pragmatic method that compares peer transfer pairs through a set of controlled experiments in a real campus traffic environment. We demonstrated how low throughput transfers can be identified and a simple PRCAs can be constructed with limited information.

Chapter 6

Conclusions and Future Work

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

6.1 Summary and conclusions

In this dissertation, we presented our work on advancing research in traffic management of heavy-hitters and high-performance networking. Our main contributions are as follows: (i) We characterized the size, rate, and duration of large dataset transfers; (ii) we developed, implemented, and evaluated the Cheetah Flow Identification Network Function, which is the first Network Virtualization Function detecting high-rate flows in real-time for the purpose of traffic engineering; (iii) we developed a methodology to identify heavy-hitter thresholds that optimizes for packet drops against the number of cheetah flows that are redirected; and (iv) we present a network service to diagnose end host throughput performance.

Chapter 2 presented a method to characterize the size, rate, and duration of large dataset transfers. We identified the significance of reconstructing parallel flowsets from NetFlow records collected at IP routers. High-performance file transfer applications use parallel TCP connections for large dataset transfers. Therefore, to provide an accurate picture of large data movement, reconstructing parallel flows was necessary. We applied our flowset reconstruction method to NetFlow records collected from June-Sept. 2014 at all 66 ESnet routers. Our findings revealed that todays scientists move 100 GB to TB sized datasets at

rates of 1 to 2.5 Gbps, and seldom use the network for transfers lasting more than 10 hours. We found that the median rate of flowsets increases and rate variance decreases with the number of per-FS component flows. Therefore, we *concluded* that flowset reconstruction study can impact network planning, traffic engineering, and help improve user experience, since large dataset transfers are among the most demanding of network applications.

Chapter 3 described the design, implementation, and evaluation of a high-speed Cheetah Flow Identification Network Function (CFINF) on a high-performance multi-core commodity server. The primary goal for CFINF was to detect a cheetah flow in real-time from among millions of flows arriving every minute on a high-speed link using software that is executable on an x86 general purpose multi-core server. Consequently, the algorithms of CFINF had to be optimized to process packets in constant time on an average. To address the scaling of the CFINF to support high link speeds, the algorithms were designed to be executed independently on multiple cores. High-performance NIC features such as packet coalescence, receive-side scaling, kernel bypass, and polling-mode operation enable the design to scale to high link speeds. The CFINF is central to the development of a Cheetah Flow Traffic Engineering System, which isolates cheetah flow packets once such a flow is detected. We found that with 10 cores, CFINF could handle the 10 Gbps packet traces. We tested a performance optimization of CFINF, which consisted of dropping small-sized packets and found it resulted in a 10% drop in the accuracy of reported cheetah flows, while offering significant computational savings. Our *conclusion* from the CFINF evaluation is that with current advances in high-performance flexible networking hardware and algorithms optimizing for speed, cheetah flow detection can be performed on an x86 multi-core server.

Chapter 4 presented a pragmatic approach to determine a cheetah flow redirection threshold. We provide a solution for computing the rate threshold using control feedback techniques. This work complements our design of the CFTES and provides an automated mechanism to compute a cheetah flow rate threshold. We present a scavenger queue to isolate detected cheetah flows, which solves the problem of packet reordering and subsequent loss in throughput. We define a parameter called burstiness of the traffic, which is used for computing the rate threshold. We found that the burstiness of the background traffic, network utilization, and high-rate flow RTT are key factors in computing this threshold.

Increase in burstiness of the background traffic for the same network utilization; i.e., average rate, increases the background traffic packet loss rate. Our solution maximizes the rate threshold to avoid classifying too many flows circumventing the trade-offs associated with cheetah flow classification. For our scavenger queue cheetah flow redirection it results in a reduction of flow throughput. Our *conclusion* finds the choice of the rate threshold should not be lower than is necessary to achieve an acceptable rate of packet loss.

Chapter 5 presented our network service for diagnosing throughput performance. We described the design of a Cheetah Flow Throughput Monitoring System (CFTMS), which identifies transfers with low throughput. A Performance Root Cause Analysis is then published to the user associated with the data transfer. Our findings indicate the challenge in measuring transfer throughput from passive network data such as NetFlow. Therefore, we use a combination of end host transfer logs and resource utilization logs is necessary to diagnose the cause of low-throughput in a transfer. We identified that collecting network data across multiple organizations may not be feasible due to privacy and confidentiality issues. Therefore, we limit the scope of our research to diagnosing end-host performance issues. Through a set of real-world experiments we provide a method to diagnose performance problems. Our *conclusion* is that it is feasible to identify the cause for low-throughput performance even with limited information from the host logs and data transfer logs.

6.2 Future Work

This work can be extended in the following directions:

1. The current CFTES implementation, could include the potential of investigating the control plane action of redirecting the cheetah flows. Since it takes some time for the SDN controller to set filter rules for redirection, the real-time constraints on the detection and redirection has to be explored. The integrated CFTES software including the Traffic Analysis Module can be tested over an organization access link, e.g. the University of Virginia Internet access link, to quantify the effectiveness of CFTES.

2. For CFTMS, our proposed solutions to identify low throughput performance can be prototyped and evaluated on a experimental network setup. Network data such

as SNMP and NetFlow data from the experimental testbed can be generated. The combination of host logs, transfer logs, and network data can be analyzed to provide more detailed Performance Root Cause Analysis.

Bibliography

[1] National Science Foundation Network. https://en.wikipedia.org/wiki/National_Science_Foundation_Network.

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

[3] ESnet. https://www.es.net/.

[4] Global internet hosts in the domain name system 1993-2017. https://www.statista.com/statistics/264473/number-of-internet-hosts-in-the-domain-name-system/.

[5] Artur Barczyk. World-wide Networking for LHC Data Processing. In *National Fiber Optic Engineers Conference*, page NTu1E.1. Optical Society of America, 2012. http://www.osapublishing.org/abstract.cfm?URI=NFOEC-2012-NTu1E.1.

[6] Randal Bryant, Randy H Katz, and Edward D Lazowska. Big-data computing: creating revolutionary breakthroughs in commerce, science and society, 2008.

[7] Large Synoptic Survey Telescope. https://www.lsst.org/about/dm.

[8] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, Jon Zolla, Urs Hölzle, Stephen Stuart, and Amin Vahdat. B4: Experience with a Globally-deployed Software Defined WAN. *SIGCOMM Comput. Commun. Rev.*, 43(4):3–14, August 2013.

[9] H. Qian, Xin Huang, and C. Chen. SWAN: End-to-end orchestration for cloud network and WAN. In *2013 IEEE 2nd International Conference on Cloud Networking (CloudNet)*, pages 236–242, Nov 2013.

[10] Liang Guo and I. Matta. The war between mice and elephants. In *Proceedings Ninth International Conference on Network Protocols. ICNP 2001*, pages 180–188, Nov 2001.

[11] Douglas Leith and Robert Shorten. H-TCP: TCP for high-speed and long-distance networks.

[12] T. Jin, C. Tracy, and M. Veeraraghavan. Characterization of high-rate large-sized flows. In *2014 IEEE International Black Sea Conference on Communications and Networking (BlackSeaCom)*, pages 73–76, May 2014.

[13] R. Addanki, S. Maji, M. Veeraraghavan, and C. Tracy. A measurement-based study of big-data movement. In *2015 European Conference on Networks and Communications (EuCNC)*, pages 445–449, June 2015.

[14] S. Maji, M. Veeraraghavan, M. Buchanan, F. Alali, J. Ros-Giralt, and A. Commike. A high-speed cheetah flow identification network function (CFINF). In *2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pages 1–7, Nov 2017.

[15] J. Ros-Giralt, A. Commike, R. Lethin, S. Maji, and M. Veeraraghavan. High-performance algorithms and data structures to catch elephant flows. In *2016 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–7, Sept 2016.

[16] Sourav Maji, Xiaoyu Wang, Malathi Veeraraghavan, Jordi Ros-Giralt, and Alan Commike. A pragmatic approach of determining heavy-hitter traffic thresholds. In *2018 IEEE European Conference on Networks and Communications (EuCNC)*, 2018.

[17] Eli Dart, Lauren Rotman, Brian Tierney, Mary Hester, and Jason Zurawski. The science DMZ: A network design pattern for data-intensive science. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '13, pages 85:1–85:10, New York, NY, USA, 2013. ACM.

[18] Bryce Allen, John Bresnahan, Lisa Childers, Ian Foster, Gopi Kandaswamy, Raj Kettimuthu, Jack Kordas, Mike Link, Stuart Martin, Karl Pickett, and Steven Tuecke. Software as a service for data scientists. *Communications of the ACM*, 55(2), February 2012.

[19] Tian Jin, C. Tracy, and M. Veeraraghavan. Characterization of high-rate large-sized flows. In *Communications and Networking (BlackSeaCom), 2014 IEEE International Black Sea Conference on*, pages 73–76, May 2014.

[20] Tiago Fioreze, Ro Zambenedetti Granville, Aiko Pras, Anna Sperotto, and Ramin Sadre. Self-Management of Hybrid Networks: Can We Trust NetFlow Data. In *In: 11th IFIP/IEEE International Symposium on Integrated Network Management (IM 2009)*, pages 1–5, 2009.

[21] Péter Megyesi and Sándor Molnár. Analysis of elephant users in broadband network traffic. In *Advances in Communication Networking - 19th EUNICE/IFIP WG 6.6 International Workshop, Chemnitz, Germany, August 28-30, 2013. Proceedings*, pages 37–45, 2013.

[22] On-Demand Secure Circuits and Advance Reservation System (OSCARS). http://www.es.net/engineering-services/oscars.

[23] T. Fioreze and A. Pras. Self-management of hybrid optical and packet switching networks. In *Integrated Network Management (IM), 2011 IFIP/IEEE International Symposium on*, pages 946–951, 2011.

[24] Arif Merchant, Mustafa Uysal, Pradeep Padala, Xiaoyun Zhu, Sharad Singhal, and Kang Shin. Maestro: Quality-of-service in Large Disk Arrays. In *Proceedings of the 8th ACM International Conference on Autonomic Computing*, ICAC '11, pages 245–254, New York, NY, USA, 2011. ACM.

[25] Hyunchul Kim, KC Claffy, Marina Fomenkov, Dhiman Barman, Michalis Faloutsos, and KiYoung Lee. Internet traffic classification demystified: myths, caveats, and the best practices. In *Proceedings of the 2008 ACM CoNEXT Conference*, CoNEXT '08, pages 11:1–11:12, New York, NY, USA, 2008. ACM.

[26] Yeonhee Lee and Youngseok Lee. Toward Scalable Internet Traffic Measurement and Analysis with Hadoop. *SIGCOMM Comput. Commun. Rev.*, 43(1):5–13, January 2012.

[27] Center for Applied Internet Data Analysis (CAIDA). The CAIDA Anonymized Internet Traces 2014 Dataset. http://www.caida.org/data/passive.

[28] GridFTP. http://globus.org/toolkit/docs/3.2/gridftp/.

[29] bbFTP Usage Examples. http://www.nren.nasa.gov/bbftp.html.

[30] B. Claise, B. Trammell, and P. Aitken. Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information. IETF RFC 7011 (INTERNET STANDARD).

[31] Open Networking Foundation. https://www.opennetworking.org/.

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

[33] Andreas Bechtolsheim, Lincoln Dale, Hugh Holbrook, And Ang Li. Why Big Data Needs Big Buffer Switches. https://www.arista.com/assets/data/pdf/Whitepapers/BigDataBigBuffers-WP.pdf, 2016. Online; accessed 12 Jan 2017.

[34] Yin Zhang, Lee Breslau, Vern Paxson, and Scott Shenker. On the Characteristics and Origins of Internet Flow Rates.

[35] Srinivas Shakkottai, Nevil Brownlee, and Claffy K.C. A study of burstiness in TCP flows. In *International Workshop on Passive and Active Network Measurement*, pages 13–26. Springer, 2005.

[36] R. Mijumbi, J. Serrat, J. Gorricho, N. Bouten, F. De Turck, and R. Boutaba. Network Function Virtualization: State-of-the-Art and Research Challenges. *Communications Surveys Tutorials, IEEE*, 18(1):236–262, Firstquarter 2016.

[37] Tatsuya Mori, Masato Uchida, Ryoichi Kawahara, Jianping Pan, and Shigeki Goto. Identifying elephant flows through periodically sampled packets. In *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pages 115–120. ACM, 2004.

[38] Fang Hao, M. Kodialam, T. V. Lakshman, and Hui Zhang. Fast, memory-efficient traffic estimation by coincidence counting. In *IEEE Infocom*, 2005.

[39] N. Kamiyama and T. Mori. Simple and Accurate Identification of High-Rate Flows by Packet Sampling. In *Proceedings IEEE INFOCOM 2006. 25TH IEEE International Conference on Computer Communications*, pages 1–13, April 2006.

[40] Y. Lu, 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, Aug 2007.

[41] 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*, volume 1, pages V1–370–V1–374, April 2010.

[42] Y. Li, H. Liu, W. Yang, D. Hu, X. Wang, and W. Xu. Predicting Inter-Data-Center Network Traffic Using Elephant Flow and Sublink Information. *IEEE Transactions on Network and Service Management*, 13(4):782–792, Dec 2016.

[43] Jordi Ros-Giralt, Alan Commike, Dan Honey, and Richard Lethin. High-performance Many-core Networking: Design and Implementation. In *ACM/IEEE INDIS*, pages 1:1–1:7, New York, NY, USA, 2015. ACM.

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

[45] Zhenzhen Yan, Malathi Veeraraghavan, Chris Tracy, and Chin 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.

[46] Hash functions. http://www.cse.yorku.ca/ oz/hash.html.

[47] Chameleon cloud. https://www.chameleoncloud.org/.

[48] Kun-chan Lan and John Heidemann. A Measurement Study of Correlations of Internet Flow Characteristics. *Comput. Netw.*, 50(1):46–62, January 2006.

[49] Y. Li and M. Chen. Software-Defined Network Function Virtualization: A Survey. *IEEE Access*, 3:2542–2553, 2015.

[50] A. Molina, S. Tartarelli, F. Raspall, and S. Niccolini. Implementation of an IPFIX compliant flow traffic meter: challenges and performance assessment. In *Proc. of the 3rd IEEE IPOM 2003*, Oct 2003.

[51] Francesco Fusco and Luca Deri. High Speed Network Traffic Analysis with Commodity Multi-core Systems. In *10th ACM IMC '10*, 2010.

[52] Zhen Zhang, Binqiang Wang, and Julong Lan. Identifying elephant flows in internet backbone traffic with bloom filters and LRU. *Computer Communications*, 61:70 – 78, 2015.

[53] Hao Wu, Hsu-Chun Hsiao, and Yih-Chun Hu. Efficient Large Flow Detection over Arbitrary Windows: An Algorithm Exact Outside an Ambiguity Region. In *Proceedings of the 2014 Conference on Internet Measurement Conference*, IMC '14, pages 209–222, New York, NY, USA, 2014. ACM.

[54] Da Tong and Viktor Prasanna. High Throughput Sketch Based Online Heavy Hitter Detection on FPGA. *SIGARCH Comput. Archit. News*, 43(4):70–75, April 2016.

[55] Malathi Veeraraghavan Naoaki Yamanaka Weiqiang Sun Fatma Alali, Xiao Lin. SDN-enabled headroom services for high-speed data transfers. In *The 23rd IEEE Asia Pacific Conference on Communications (APCC)*, 2017.

[56] M. Zukerman, T. D. Neame, and R. G. Addie. Internet Traffic Modeling and Future Technology Implications. In *IEEE INFOCOM 2003*, volume 1, pages 587–596 vol.1, March 2003.

[57] GitHub link for simulation software. https://github.com/UVA-High-Speed-Networks/simPYTrafficSimulationPPBPandHTCP.

[58] Kun chan Lan and John Heidemann. A measurement study of correlations of Internet flow characteristics. *Computer Networks*, 2006.

[59] A. R. Curtis, W. Kim, and P. Yalagandula. Mahout: Low-overhead datacenter traffic management using end-host-based elephant detection. In *2011 Proceedings IEEE INFOCOM*, pages 1629–1637, April 2011.

[60] Andrew R. Curtis, Jeffrey C. Mogul, Jean Tourrilhes, Praveen Yalagandula, Puneet Sharma, and Sujata Banerjee. DevoFlow: Scaling Flow Management for High-performance Networks. *SIGCOMM Comput. Commun. Rev.*, 41(4):254–265, August 2011.

[61] Mohammad Al-Fares, Sivasankar Radhakrishnan, Barath Raghavan, Nelson Huang, and Amin Vahdat. Hedera: Dynamic Flow Scheduling for Data Center Networks. In *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*, NSDI'10, pages 19–19, Berkeley, CA, USA, 2010. USENIX Association.

[62] Nathan Farrington, George Porter, Sivasankar Radhakrishnan, Hamid Hajabdolali Bazzaz, Vikram Subramanya, Yeshaiahu Fainman, George Papen, and Amin Vahdat. Helios: A Hybrid Electrical/Optical Switch Architecture for Modular Data Centers. *ACM SIGCOMM Computer Communication Review*.

[63] C. Bi, X. Luo, T. Ye, and Y. Jin. On precision and scalability of elephant flow detection in data center with SDN. In *2013 IEEE Globecom Workshops (GC Wkshps)*, pages 1227–1232, Dec 2013.

[64] MT Pham, KT Seow, and CH Foh. Towards intelligent datacenter traffic management: Using automated fuzzy inferencing for elephant flow detection. *International Journal of Innovative Computing, Information and Control*, 10(5):1669 – 1685, January 2014.

[65] Tian Bu, Nick Duffield, Francesco Lo Presti, and Don Towsley. Network Tomography on General Topologies. *SIGMETRICS Perform. Eval. Rev.*, 30(1):21–30, June 2002.

[66] D. Gunter, B. Tierney, B. Crowley, M. Holding, and J. Lee. Netlogger: a toolkit for distributed system performance analysis. In *Proceedings 8th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (Cat. No.PR00728)*, pages 267–273, 2000.

[67] Minlan Yu, Albert G Greenberg, David A Maltz, Jennifer Rexford, Lihua Yuan, Srikanth Kandula, and Changhoon Kim. Profiling Network Performance for Multi-tier Data Center Applications. In *NSDI*, volume 11, pages 5–5, 2011.

[68] Andreas Hanemann, Jeff W Boote, Eric L Boyd, Jérôme Durand, Loukik Kudarimoti, Roman Łapacz, D Martin Swany, Szymon Trocha, and Jason Zurawski. Perfsonar: A service oriented architecture for multi-domain network monitoring. In *International Conference on Service-Oriented Computing*, pages 241–254. Springer, 2005.

[69] perfSONAR-PS. http://psps.perfsonar.net/.

[70] Bwctl. http://software.internet2.edu/bwctl/.

[71] Esma Yildirim and Tevfik Kosar. Network-aware End-to-end Data Throughput Optimization. In *Proceedings of the First International Workshop on Network-aware Data Management*, NDM '11, pages 21–30, New York, NY, USA, 2011. ACM.

[72] 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 *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '12, pages 81:1–81:11, Los Alamitos, CA, USA, 2012. IEEE Computer Society Press.

[73] Dan Gunter, Brian L. Tierney, Aaron Brown, Martin Swany, John Bresnahan, and Jennifer M. Schopf. Log Summarization and Anomaly Detection for Troubleshooting Distributed Systems. In *Proceedings of the 8th IEEE/ACM International Conference on Grid Computing*, GRID '07, pages 226–234, Washington, DC, USA, 2007. IEEE Computer Society.

[74] D. Gunter, B. Tierney, K. Jackson, J. Lee, and M. Stoufer. Dynamic monitoring of high-performance distributed applications. In *Proceedings 11th IEEE International Symposium on High Performance Distributed Computing*, pages 163–170, 2002.

[75] E. Kissel, A. El-Hassany, G. Fernandes, M. Swany, D. Gunter, T. Samak, and J. M. Schopf. Scalable integrated performance analysis of multi-gigabit networks. In *2012 IEEE Network Operations and Management Symposium*, pages 1227–1233, April 2012.

[76] Ezra Kissel, Dan Gunter, Taghrid Samak, Ahmed El-Hassany, Guilherme Fernandes, and Martin Swany. An Instrumentation and Measurement Framework for End-to-End Performance Analyis. 2011.

[77] P. Kanuparthy, D. H. Lee, W. Matthews, C. Dovrolis, and S. Zarifzadeh. Pythia: detection, localization, and diagnosis of performance problems. *IEEE Communications Magazine*, 51(11):55–62, November 2013.

[78] Y. Zhang, P. Calyam, S. Debroy, and M. Sridharan. Pca-based network-wide correlated anomaly event detection and diagnosis. In *2015 11th International Conference on the Design of Reliable Communication Networks (DRCN)*, pages 149–156, March 2015.

[79] Y. Zhang, S. Debroy, and P. Calyam. Network-Wide Anomaly Event Detection and Diagnosis With perfSONAR. *IEEE Transactions on Network and Service Management*, 13(3):666–680, Sept 2016.

[80] Jorge Batista, Constantine Dovrolis, Danny Lee, and Shawn McKee. Identifying and localizing network problems using the PuNDIT project. *Journal of Physics: Conference Series*, 664(5):052027, 2015.