Handling Elephant Flows in a Multi-tenant Data Center Network

By

Sadia Bashir

2011-NUST-MS-CCS-030

Supervisor

Dr. Nadeem Ahmed

NUST-SEECS

A thesis submitted in partial fulfillment of the requirements for the degree of MS-CCS

In

School of Electrical Engineering and Computer Science,

National University of Sciences and Technology (NUST),

Islamabad, Pakistan.

(July 2015)

Approval

It is certified that the contents and form of the thesis entitled "**Handling Elephant Flows in a Multi-tenant Data Center Network**" submitted by **Sadia Bashir** have been found satisfactory for the requirement of the degree.

Advisor: **Dr. Nadeem Ahmed**

Signature: ____________

Date: ____________

Committee Member 1: **Dr. Junaid Qadir**

Signature: ____________

Date: ____________

Committee Member 2: **Dr. Syed Ali Khayam**

Signature: ____________

Date: ____________

Committee Member 3: **Dr. Syed Ali Haider**

Signature: ____________

Date: ____________

Abstract

A virtualized data center is highly multifarious environment, shared among hundreds of co-located tenants hosting heterogeneous applications. High degree of virtual machine consolidation leads to diverse traffic dynamics with uneven traffic demands. Tenants' virtual machines generate a subset of elephants or mice flows that traverse the underlay fabric in aggregate, i.e., encapsulated in tunneling protocols such as VXLAN, NVGRE and STT. Elephant flows are long-lived and bandwidth intensive whereas mice flows are short-lived, latency-sensitive and highly bursty in nature. Both types of flows require different treatment from underlay fabric but encapsulation obfuscates the overlay traffic characteristics and demands. Existing approaches employed in data centers such as ECMP are either agnostic to elephant and mice flows or have no visibility into virtual traffic which may be used to precisely detect, isolate and treat elephant flows differently than mice flows. If elephant flows are not identified and addressed in aggregated virtual traffic, they may affect mice flows generated from co-located applications, hence degrading application performance of co-located tenants. Therefore, there is a need to manage virtual traffic along with physical traffic in data centers. This work identifies and treats elephant flows belonging to different tenants on virtual network components such as virtual switches residing in the hypervisor. Virtual switch is at the optimal location to detect and handle elephant flows before they saturate the underlay network fabric and may be used to ameliorate the network performance with minimal overhead both on host and in-network. The proposed scheme has been tested against state of the art approaches such as Hedera and ECMP and has shown improved bisection bandwidth on a 4-array fat-tree interconnect.

Certificate of Originality

I hereby declare that this submission is my own work and to the best of my knowledge it contains no materials previously published or written by another person, nor material which to a substantial extent has been accepted for the award of any degree or diploma at National University of Sciences & Technology (NUST) School of Electrical Engineering & Computer Science (SEECS) or at any other educational institute, except where due acknowledgement has been made in the thesis. Any contribution made to the research by others, with whom I have worked at NUST SEECS or elsewhere, is explicitly acknowledged in the thesis.

I also declare that the intellectual content of this thesis is the product of my own work, except for the assistance from others in the project's design and conception or in style, presentation and linguistics which has been acknowledged.

Author Name: **Sadia Bashir**

Signature: ________________

Acknowledgment

Prima facie, I am extremely grateful to **ALLAH**, the **ALMIGHTY, WHO** gave me good health and wellbeing that were necessary to complete this dissertation.

I place on record, my sincere gratefulness to my beloved **parents** and siblings especially my brother **Muhammad Bilal**, for I would have never been able to complete this research work without their unceasing encouragement, support and prayers. I wish to express my sincere thanks to my research advisor **Dr. Nadeem Ahmed** for his support and guidance in all of my research endeavors. I am also grateful to my thesis Committee members **Dr. Syed Ali Haider, Dr. Syed Ali Khayam** and **Dr. Junaid Qadir** for helping me and sparing time for me. I am extremely thankful and indebted to **Dr. Junaid Qadir** for his sincere and valuable guidance and encouragement extended to me.

I owe my deepest gratitude to **Dr. Amir Shafi** for providing me with the necessary infrastructure for implementation and for his continuous encouragement.

I take this opportunity to express gratitude to **Dr. Peter Charles Bloodsworh** and my very first mentor **Mr. Shabbir Khan** for their valuable guideline.

I also place on record, my sense of gratitude to one and all, who directly or indirectly, have lent their hand in this venture.

Sadia Bashir

Contents

List of Abbreviations

Abbreviations	Descriptions
AAPC	All-to-All Personalized Communication
CAPEX	CAPital EXpenditure
ECMP	Equal Cost Multi-Path
EoR	End of Row
NVGRE	Network Virtualization with Generic Routing Encapsulation
OPEX	OPerational EXpenditure
OVS	Open Virtual Switch
OVSDB	OVS Database
PCI	Peripheral Component Interconnect
PIF	Physical Interface
PMS	Pod Management Switch
SDDC	Software Defined Data Centers
SDN	Software Defined Network
SRIOV	Single Root I/O Virtualization
STT	Stateless Transport Tunneling
ToR	Top of Rack
VIF	Virtual Interface
VM	Virtual Machine
VNI	Virtual Network Identifier
VTEP	Virtual Tunnel End Point
VXLAN	Virtual Extensible Local Area Network

List of Figures

List of Tables

Chapter 1

Introduction

This chapter is organized in four sections. After general introduction, subdivision 1.1, states motivation for carrying this research work. In section 1.2, problem statement is stated. In Section 1.3, thesis contributions are listed. In Section 1.4, outline for the rest of the thesis is concluded.

Modern data centers have widely adopted virtualization technology which has greatly transformed the deployment and management of traditional networks. In virtualized data centers, physical infrastructure is multiplexed in the form of virtual machines to deploy a large number of customers (a.k.a., *tenants*) running different services/applications. In addition, data center environments have become denser and more heterogeneous with high level of Virtual Machine (VM) consolidation. Application architectures are also decomposing because of the advent of REST and JSON technologies. This decomposition is leading to more easterly or westerly deployment of a tenants' applications across different physical servers within the data center.

Changing application architectures have changed data center traffic dynamics as well. According to [23] and [22], more than 80% of data center traffic is between virtual machines (east-west) instead of between user and applications (north-south). Moreover, high degree of VM consolidation is causing more data traffic per-physical host [8], contributing more into virtual traffic volume. This virtual traffic is encapsulated in VXLAN [28], NVGRE [19] and STT [15] tunnels by hypervisor for scalability, performance , and broadcast/multicast isolation purposes. Therefore, east-west traffic that traverses data center interconnect is typically aggregated.

Virtual data center traffic is a mix of elephants and mice flows with varying traffic characteristics and demands. Elephant flows are large TCP flows having large size (from few MBs to GBs or even more) which may persist for a duration generally from a few seconds to hours. They are not latency-sensitive and require high throughput. Mice flows are short flows which last usually less than ten seconds and are highly latency-sensitive and bursty in nature. The ratio of mice to elephant flows in a data center is 80% to 20% with 20% of elephant flows carrying 80% of the bytes and 80% of mice flows carrying 20% of rest of the bytes in the data center [6] and [26]. Applications such as Hadoop, MapReduce, VM Migration, cloning and backup creation for consistency purposes generate elephant flows while gaming, voice, video and web traffic generate mice flows. Therefore, a large volume of traffic in data centers is flow-oriented or TCP-based.

1.1 Motivation

Bandwidth achieved by east-west traffic between a tenants' virtual machines depends on certain factors not in tenant's control such as random placement of tenants' virtual machines because of shared tenancy model used by various cloud providers [1], highly dynamic network workloads, oversubscription ratios and routing mechanisms used by underlying fabric. The applications in multi-tenant data centers rely on the underlying shared network to transfer gigabytes to petabytes of data at high rates but underlying Internet Protocol (IP) does not provide guaranteed services because it is the best effort delivery protocol and applications are serviced in best effort manner by default. Nonetheless, this default behavior is not desirable all the time because every network application has different demand in terms of latency and throughput. According to a latest study on virtual data centers [8], data centers have "elephant" virtual machines capable of generating many times more bandwidth demands as compared to average virtual machines whereas high heterogeneity may lead to isolation and/or fairness issues because of uneven demands.

Most data centers usually employ traditional techniques such as Equal Cost Multi Path (ECMP) for routing flows because of its simplicity, fault-tolerance and support in enterprise switches, but first, ECMP has no visibility inside aggregated virtual traffic and secondly, it is unable to identify and treat mice and elephant flows differently. Thus in the same flow carrying different mice and elephant flows from different tenants, elephant flows may saturate the network buffers consuming high throughput disproportionately for a long period of time leaving mice flows end up waiting in long queues behind large flows. This may add non-trivial queuing delay to mice flows traversing the same path thus deteriorating the performance of other co-located tenant applications. Approaches proposed previously either provide no visibility into virtual traffic or require too much network resources to derive virtual traffic information from physical traffic. Therefore, it has now become crucial to identify and correlate traffic between underlays and overlays because virtual traffic volume generated from virtual machines to the underlay physical network determines application performance in cloud data centers.

1.2 Problem Statement

The problem statement of the thesis is as below:

"*To design and implement elephant flow detection and handling mechanism for a virtualized data center network environment to ameliorate the multiple tenant's co-located application performance*"

1.3 Thesis Contribution

Contribution of this thesis is development of mechanism which identifies and handles elephant flows belonging to a tenant on openvswitch. Openvswitch resides in the hypervisor in close proximity of virtual machines where it can derive information and other traffic statistics not easily derivable by inspection network traffic alone. Elephant flow identification and handling on openvswitch ensures that different network traffic is treated according to its requirements and co-located tenant applications or services are not affected.

1.4 Thesis Organization

The rest of the thesis is structured as follows:

Chapter 2 discusses the state of the art related to the current research, background, and reviews the relevant literature aimed at finding elephant flow detection and handling techniques in data center networks. Chapter 3 discusses problem formulation and design goals in detail. In Chapter 4, proposed elephant detection and handling methodology is discussed. In Chapter 5, implementation and results are given along with detailed discussions. In Chapter 6, the conclusion and future work is stated.

Chapter 2

Background and Literature Review

This chapter reviews literature that has been used to formulate problem definition. Section 2.1 gives an overview of traditional and virtualized data centers and their benefits. In section 2.2, latest trends in virtualized data centers have been discussed comprehensively. Section 2.3 gives a general classification of elephants and mice flows. In section 2.4, different elephant flow detection schemes from literature have been analyzed. Section 2.5 discusses general techniques and methodologies used to manage elephant and mice flows in general, in SDN-controlled data centers and in virtualized data centers. In the last section analysis of various elephant flow detection and management techniques with respect to virtualized data center is discussed briefly.

2.1 Overview of Data Centers

2.1.1 Virtualized Data Centers

In traditional data centers, dedicated infrastructure is used to host independent applications. This model is expensive in terms of CAPEX and OPEX because of resource under-utilization, maintenance cost and vertical scaling to handle dynamic workloads. Virtualization technology has been deployed in recent data centers to efficiently utilize infrastructure and to overcome other problems in traditional data centers. Server/machine virtualization, storage virtualization and network virtualization are used to virtualize underlying compute, storage and network resources respectively. With server virtualization, a single physical server can be emulated as multiple servers possibly running multiple operating systems on the same hardware.

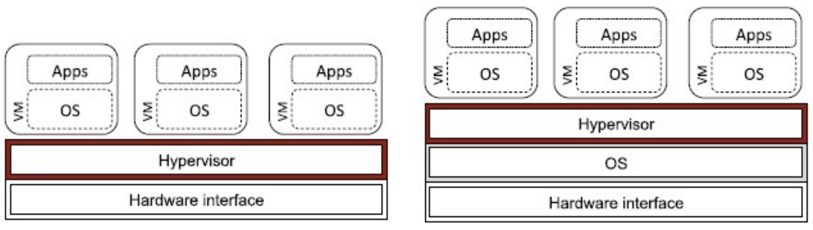


Figure 2.1: Type 1 vs Type 2 hypervisor

Server virtualization (also called *processor virtualization*) is achieved by running hypervisor either directly on top of hardware (*type 1* hypervisor) or atop an operating system (*type 2* hypervisor) [18] which is responsible to provide hardware abstraction as shown in Fig. 2.1. Network virtualization is tightly coupled with machine virtualization which is implemented by running virtual switches or routers in the hypervisor [37]. Network virtualization is used to dynamically create virtual overlay networks to manage communication between instances of a single tenant.

Leveraging large RAM capacities and high performance storage and network architectures result in high degree of VM consolidation as shown in Fig. 2.2 which has enabled cloud providers to service large number of multiple tenants simultaneously by dynamically scaling resources. High level of reliability and availability is provided in cloud data centers by efficiently managing and re-optimizing physical resources in completely automated or semi-automated way according to dynamically changing requirements.

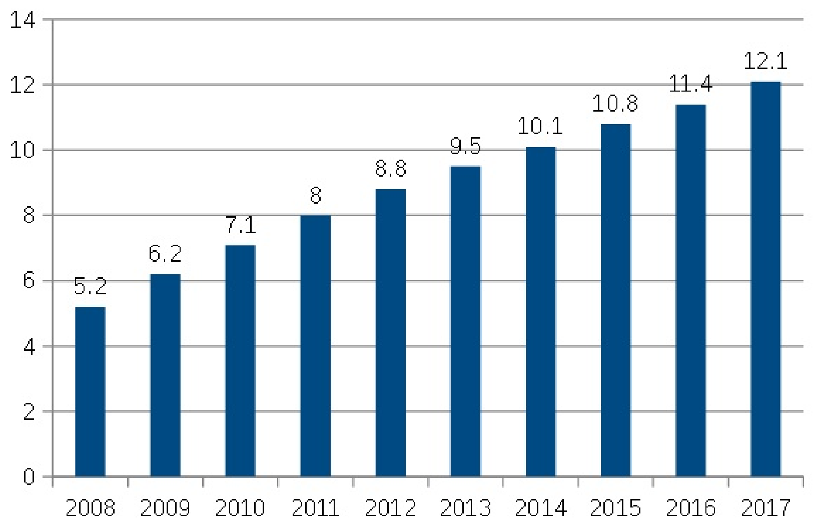


Figure 2.2: Mean VM density/server

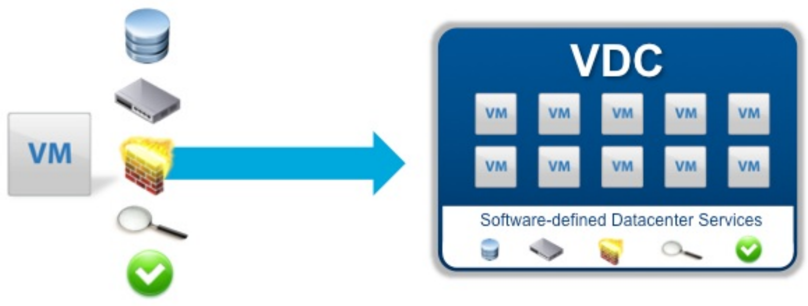


Figure 2.3: Software defined data center architecture

2.1.2 Software Defined Data Centers (SDDC)

The most recent paradigm shift in the world of virtualization is the emergence of Software Defined Data Centers (SDDC). A software defined data center extends virtualization concepts such as abstraction, pooling, and automation to the entire data center infrastructure by intelligently configuring and maintaining hardware through automated software systems. This is different from traditional data centers where hardware and devices define the infrastructure. Fig. 2.3 explains the concept of software defined data centers.

2.1.3 Software Defined Networks (SDN)

Software Defined Networking is the recently emerged architecture purporting to provide cost-effectiveness, manageability, dynamism and adaptability in networking solutions. SDN architecture decouples network control from forwarding functions enabling it to be programmable and abstracted from underlying network infrastructure. The structural design of SDN comprises of application, control and data forwarding layers. SDN applications are contained in application layer, the centralized or distributed control layer makes decisions and policies which regulate the network elements while data forwarding layer employs programmable switches.

A set of Application Programming Interfaces (APIs) such as north-bound APIs and south-bound APIs are used to support communication between applications and control plane and between control plane and forwarding or data plane respectively as shown in the Fig. 2.4. North-bound APIs facilitate communication between applications and other services such as routing, traffic engineering, quality of service (QoS), security, bandwidth management, access control, and many other forms of

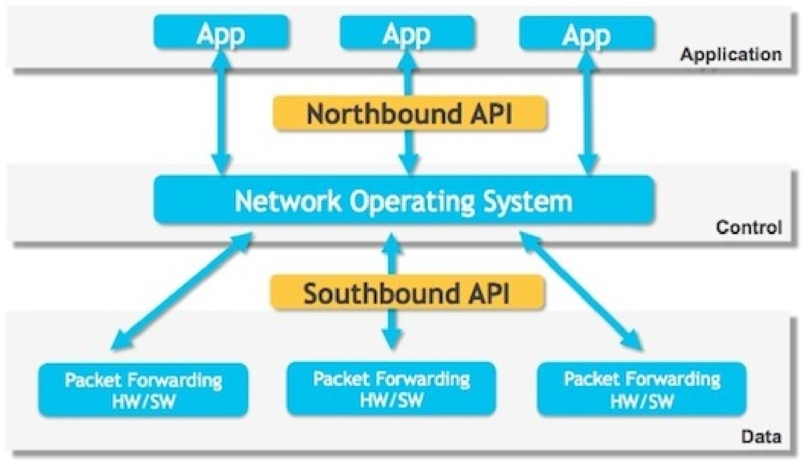


Figure 2.4: SDN reference model

the network management while south-bound APIs such as OpenFlow (OF) protocol [32] enable communication between control layer and data forwarding layer.

The forwarding plane of a network switch is accessed through OpenFlow protocol enabling software programs to be executed on switches and perform functionalities like packet lookups and forwarding among network elements. The SDN/OF controller globally regulates the network state by formulating network policies because of unrestricted access it has to programmable switches and resources, in this way all network elements can be updated timely to react to the current flow activities through a centralized set of decisions and policies. In a simple operation for managing flows in SDNs, when a new flow arrives at switch its first packet is sent to the controller which computes the forwarding path for new flow according to its global view of network state. The forwarding path entries are then installed in the switches' flow tables and subsequent flow packets are switched according to the newly installed entries. Two modes can be used to populate flow table entries in a flow table: proactive and reactive. In proactive mode, controller logic pre-populates flow table entries while in reactive mode, a flow table entry is installed after its first packet misses the table and is forwarded to the controller to compute forwarding action. This simplified process of flow management gives a glimpse on the interaction among three layers of SDN paradigm which fulfills the promise of simplicity and innovation in traditional networks.

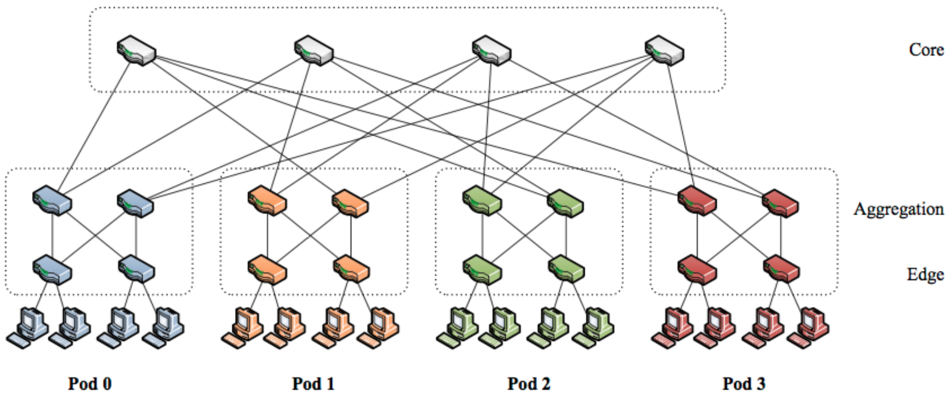


Figure 2.5: k-array fat-tree: three-tier topology

2.2 Current Trends in Data Center Networks

The proliferation of new technologies, such as cloud computing and virtualization is deriving several new trends in data centers. We explore the following trends in context of this study.

2.2.1 Topology

Data center networks typically employ *clos* or *multi-rooted* tree topologies consisting either of two- or three-tiers of edge/access, aggregate and core switches. Machines are arranged in rows of racks which are interconnected with Top-of-Rack or End-of-Row (EoR) switches. Edge switches are connected to an aggregation layer in the middle which is further connected to the core layer of the interconnect. Edge layer switches (ToR and EoR) are connected to storage and compute nodes directly. The aggregation tier interconnects edge and core switches and provides services like load-balancing on the intermediate links while core tier is the high-speed backplane. A three-tiered fat-tree architecture is shown in Fig. 2.5.

A fat-tree network has high bisection bandwidth, redundancy and fault-tolerance which can be built using identical commodity switches having same number of ports in low cost. It is a switch-centric architecture and its size is the function of number of k switch ports [41]. It can accommodate $k^3/4$ total number of servers in $k^2/4$ pods with k/2 edge-to-host and edge-to-aggregate links and $(k/2)^2$ number of core switches where each core switch has its one port connected to each k pod. There are k.k/4 total paths available between a pair of source and destination hosts.

2.2.2 Path Multiplicity

Multi-rooted tree topologies have multiple equal-cost paths between all sets of source and destination pairs. Path multiplicity may be exploited either concurrently or in backup configuration to deliver high bisection bandwidth between arbitrary hosts in large clusters. In addition, multiple disjoint paths are statistically independent and diverse in terms of transmission errors. For short flows, exploiting path multiplicity is of little advantage while for large flows, it can be used to achieve better performance and high throughput. However, only a subset of multiple paths can provide good results as compared to utilizing all of them because it involves management cost associated with high number of paths [38].

2.2.3 Forwarding

Data center network physical fabrics are managed by different forwarding techniques. The most commonly deployed techniques are based on layer 2 or layer 3 forwarding. In a L3 network, IP addresses are assigned to hosts based on their physical location in the hierarchy while a L2 approach works by assigning MAC addresses to hosts and performs forwarding based on flat MAC addresses. Each of these forwarding techniques have limitations associated as described in [34]. Virtualization presents challenges to both L2 and L3 networks in terms of virtual machine management, scalability, domain and performance isolation so this is where an overlay network comes in. Virtual instances in an overlay network communicate with one another using tunneling protocols such as VXLAN which is layer 2 network instantiated and managed by network virtualization layer. The L2 overlay network manages all unicast and multicast communication among a tenant's virtual instances on underlay layer 3 network.

2.2.4 Oversubscription

Oversubscription ratio is the worst-case achievable bandwidth among end-hosts over total available bisection bandwidth of the network topology. It is used as a means to reduce total cost of the data center network topology. Data center designs are typically oversubscribed by a factor of 2.5:1 (400 Mbps) to 8:1 (125 Mbps). A 1:1

oversubscription ratio indicates that full bandwidth is available for communication among arbitrary end hosts while an oversubscription ratio of 5:1 indicates that only 20% of host bandwidth is available for communication [3]. Although, it makes a good technical and financial sense but it has its downsides as well because multiple devices are connected to a single switch port which may struggle for that port's bandwidth resulting in poor response time.

2.2.5 Bisection Bandwidth

Bisection bandwidth is a worst-case performance metric of a data center network. It is defined as the smallest bandwidth available between two equal segments of a network. A "*full bisection bandwidth*" can support all servers communicating simultaneously at full link speed with no over-subscription. Full bisection bandwidth between all pairs of hosts is calculated as follows:

Aggregate bandwidth = total # of hosts * host NIC capacity

Although, a fat-tree offers full bisection bandwidth but with all-to-all communication patterns, hotspots are still possible. Personalized All-to-All communication (AAPC) patterns are the most challenging communication patterns in HPC applications in terms of scalability and performance and fat-tree interconnection networks are generally deployed to leverage high bisection bandwidth among communicating hosts for this type of applications.

2.2.6 Data Center Traffic Trends and Patterns

Today's virtualized data centers are highly multifarious environments hosting a range of applications and services from various tenants. The set of applications include internet messaging, gaming, web-services, computational intensive applications such as web-content indexing, data analysis, large-scale machine leaning and scientific computing. In addition, high performance computing applications such as Hadoop and MapReduce are also hosted which are notorious in generating large amounts of data requiring full bisection bandwidth.

Latest studies have shown that a significant amount of traffic in data centers is virtual which flows in east-west direction while a relatively small amount of traffic flows in north-south direction. Basically, change in application architectures and the way they are put together in virtual data centers have dramatically changed traffic patterns. For example, MapReduce works with several clients and a server machine which are possibly scattered across different physical hosts. It works in two phases: Map and Reduce, in map phase, a master maps tasks to different clients in the form of set of commands which generate little traffic while in reduce phase, a large network-wide bandwidth is required to transfer MBs or GBs of data, generating a large amount of east-west traffic. Similarly, a web-application has a three-tier architecture. Its first tier is called *front-tier* or *presentation-layer* responsible for receiving requests from users. Second layer is the layer of *business logic* servers upon which client requests are load-balanced. Business logic servers maintain persistent connections to the *backend data servers* which make up the third tier of web-applications. The web-applications are engineered in such a way that more traffic is generated in east-west direction instead of in north-south direction.

Data center applications exhibit different types of traffic patterns: 1) many-to-one 2) one-to-many 3) one-to-one [41]. Many-to-one patterns appear in data center when a front-end of web application breaks an HTTP request into different small requests which are fetched to several workers generating one-to-many traffic patterns. Workers or business logic servers process these requests and generate response back to the front-end web servers thus in many-to-one traffic pattern.

MapReduce applications generate one-to-many traffic pattern in map phase while many-to-one pattern in reduce phase, in this way generating all-to-all or many-to-many traffic patterns. Virtual machine migration and email servers upon receiving authentication request from users generate one-to-one traffic pattern. In addition to these traffic patterns, data centers also generate control traffic. In short, many-to-one and one-to-many are most prevalent and bandwidth intensive traffic patterns exhibit in data centers.

Moreover, due to high degree of VM consolidation on servers, data center's traffic is highly heterogeneous and it travels in aggregate. Since different services with different demands and different peak-loads are co-located, it smoothens overall burstiness of aggregate traffic leaving it to compete for underlying network resources. This fact stresses the need to manage virtual traffic along with physical traffic because it determines the performance of data center applications.

2.3 The Elephant and Mice Phenomenon

As described in chapter 1, elephant flows are large flows carrying 80% of the bytes in data centers while mice flows are short flows carrying 20% of the bytes. A long-lived TCP flow primarily operates in TCP congestion avoidance phase and continuously sends bursts of packets equal to maximum window size. Short flows operate in slow start phase for their entire life and their burst size depends on initial window size. UDP flows also fall in the category of mice flows. Mice flows are quite high in number and very bursty in nature and they tend to last for a few seconds or a fraction of a second. Therefore, it is not possible to apply adaptive routing schemes to mice flows. Various schemes have been proposed in literature to identify and treat elephant flows which are discussed in proceeding sections.

2.4 Elephant Detection and Related Work

Two main approaches exist in the literature to identify elephant flows:

2.4.1 In-Network Detection Mechanisms

In-network elephant flow detection mechanisms collect flow stats from network ASICs and a centralized controller uses this information to classify elephant flows for further treatment.

2.4.1.1 Sampling

In this approach, sampling is used to collect flow stats from all interfaces of a network device and collected stats are pushed in a pre-defined format to a centralized controller running in the network. sFlow and NetFlow are industry standards which use sampling technique.

sFlow [31] is an industry standard used to monitor flows in the network. it works with two components: 1) sFlow agent and 2) remote sFlow collector as shown in the

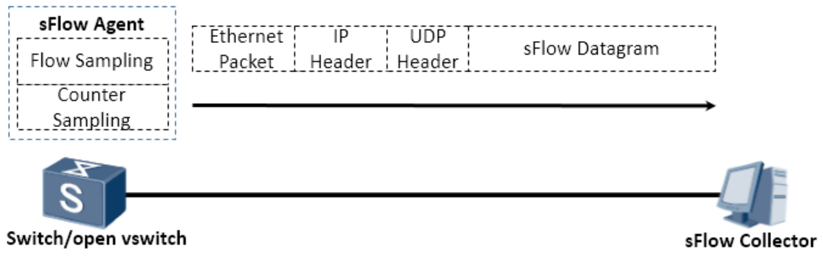


Figure 2.6: sFlow architecture

Fig. 2.6.

An sFlow agent is embedded in each network device which collects traffic statistics from each sFlow-enabled device interface using sFlow sampling based on a pre-configured sampling ratio, encapsulates collected stats into sFlow packet and if sFlow packet buffer overflows or an sFlow packet expires, pushes sFlow packets to the collector. sFlow collector is a server responsible to receive packets sent from agent and displays traffic statistics in a report after analysis. Two sampling modes are used by sFlow agents to collect statistics from network device: 1) Flow sampling and 2) Counter Sampling. An agent running in flow sampling mode analyzes packet content and forwarding rule of a flow and encapsulates this information to sFlow packet. If an agent is configured to run in counter sampling mode, it collects traffic statistics from an interface and CPU and memory usage on a device. It defines large flow as a flow consuming 10% of link bandwidth and a population containing more than 10^4 packets as a threshold.

Most of in-network detection schemes monitor each flow that hit the data center fabric. sFlow is an industry standard and its purpose is to monitor traffic for troubleshooting and anomaly detection. It can give visibility into encapsulated traffic at the cost of high overhead because agents running on each network device send records/data in UDP encapsulated packets from each interface and they utilize significant network and device resources. It may be error prone as it relies on network administrator's configuration. Moreover, it monitors each flow in the network and detects heavy flows after they saturate network devices. According to [6], millions of new flows hit edge switches every second. Therefore, monitoring each flow to detect elephant flows is expensive and incur high monitoring overhead.

2.4.1.2 Maintain Per-flow Statistics

In this approach, each incoming flow is monitored at the ToR switch and flow statistics are pulled from these switches at a regular interval. Hedera, Helios and Network Tomography based approaches use this scheme to detect elephant flows in the network.

Hedera [4] leverages a centralized controller which pulls flow byte counts at a 5 second interval from all ToR switches and categorizes elephant flows as flows consuming more than 10% link bandwidth.

In Helios [17], a Pod Monitoring Switch (PMS) is used to pull statistics from switches at regular intervals and a threshold of 15 Mb/s is used to classify flows as elephant flows.

In DevoFlow [14], OpenFlow based switches locally classify elephant flow as a flow that has transferred 1-10MB.

Hedera [4] and Helios [17] are relatively simple approaches as compared to sFlow but they provide no visibility inside virtual traffic. Moreover, they cannot proactively detect elephant flows and they have to monitor each new incoming flow irrespective of the fact whether it is a small or a large flow. Hundreds of millions of new flows are generated every second and monitoring this much traffic at short timescales from a lot of switching fabric is not feasible because in software defined networks, switch to controller channel has limited bandwidth and in case of millions of new flows, collecting stats may saturate these channels.

2.4.1.3 Network Tomography

Network tomography is the process of studying internal characteristics of network by deriving information from end-points.

In [25], network tomography along with SDN counters are collectively used to detect elephant flows in the network. It selectively monitors possible elephant flows occupying more than 10% of link capacity. It detects elephant flows by using SDN counters and SNMP link counters in two steps: 1) It finds talky ToR pairs and 2) Locates server-to-server elephant flows in ToR pairs using network tomography.

This work adapts relatively simple approach as compared to sampling and maintaining per flow statistics approaches. It relies on SDN and SNMP link counters and narrows down the search space by focusing on possible elephant flows but as it operates in network so does not provide visibility into virtual traffic and secondly it uses on SDN flow counters which are not reliable to detect elephant flows because single flows may be hiding behind wildcarded flow entries [45].

2.4.2 Host/server-based Detection Mechanisms

Host-based detection mechanisms rely on a host's view of traffic to detect elephant flows. Following are the main schemes in this area of study:

2.4.2.1 Detection by Monitoring Host TCP Socket-buffer

In (Mahout)[13], a shim layer is used on each host which is responsible to monitor flows originating from the host and as soon as a TCP-socket buffer crosses a threshold of 100KB, the shim layer sets DSCP value of the flow by setting it to 000011 to signal the controller.

2.4.2.2 Detection at Server

In (MicroTE) [7], traffic predictability is exploited by applying a kernel patch on each server. This kernel patch is responsible to collect flow's information every 0.1 seconds and sends this information to a server designated for whole rack called "designated server" every 1 second. The designated server after collecting data from all servers in the rack, aggregates server-to-server data into rack-to-rack data and determines predictable ToR pairs. In order to further reduce network overhead, this information is compressed into ToR-to-ToR traffic and then send to the centralized controller.

2.4.2.3 Detection at Server using Openvswitch

In (EMC2) [29], a web-service is implemented for cloud data centers which leverages openvswitch based sFlow and NetFlow functionality. It initiates two threads one for sFlow and other for NetFlow at web-server which continues to receive and analyze packets received from sFlow and NetFlow running at each hypervisor. It stores flow records along with its stats and path information.

Host-based elephant detection approaches mitigate the effect of in-network monitoring overhead and can signal network before elephant flows hit the network fabric and saturate switches but a shim layer or a kernel patch has to be installed in each monitoring host which is not in tenant's control. Mahout [13] and MicroTE [7] do not provide visibility into virtual traffic but as EMC2 [29] leverages openvswitch at the edge and relies on its sFlow and NetFlow component, it may give visibility in virtual traffic but at the expense of network overhead.

A summary of comparison of elephant flow detection schemes is given in Table 2.1:

Table 2.1: Comparison of Elephant Flow Detection Schemes

	Elephant Flow Detection	**Pull-based/Push-based**	**Require a kernel patch**	**Visibility into Virtual Traffic**	**Monitoring overhead**	**Threshold (Bytes)**
sFlow	In-network All switches	Push-based	No	Yes	All Flows	10^4 in population
Hedera	In-network ToR-switches	Pull-based	No	No	All Flows	10%
Helios	In-network ToR-switch	N/A	No	No	All Flows	15Mb/s
Network Tomography	In-network ToR-switches	Pull-based	No	No	Selective Flows	10%
Mahout	On Host	Push-based	Yes	No	Selective Flows	100KB
MicroTE	On Host	Push-based	Yes	No	All Flows	N/A
EMC2	On Host - OVS	Push-based	No	Yes	All Flows	N/A
DevoFlow	In-network All switches	Push-based	No	No	Selective Flows	any flow transferring between 1-10MB

2.5 Elephant and Mice Flow Management and Related Work

As we know that data center interconnects usually employ multi-rooted tree topologies which have redundant paths between a pair of source and destination to provide fault tolerance in case any node failure occurs and to load balance traffic across multiple physical paths. Main flow scheduling schemes either work to preserve TCP flow's packet order by assigning each flow a specified dedicated path between a given source and destination pairs or to split it up across multiple paths to exploit path diversity. Any of these schemes can be employed in data centers to harness its advantages for the particular type of traffic class. Different approaches have been proposed in literature to manage flows to handle congestion or to provide high throughput.

Proceeding sections first discuss general flow scheduling schemes in data centers and then techniques specifically used to manage elephant and mice flows. In general, three main flow scheduling approaches are used to manage flows in a data center network and existing elephant and mice flow management schemes fall in any one of the three categories:

2.5.1 Flow-based Scheduling Techniques

Flow-based traffic splitting schemes rely on hash-based techniques [10] which apply a direct hash function to the 5-tuple of L3 and L4 headers (source_IP, destination_IP, source_Port, destination_Port and protocol) of a flow and use the resulting hash value to select the outgoing link. These schemes are very simple to implement at a network device because they are stateless and it is fairly easy to compute hash. They can avoid TCP flow disruption and evenly distribute load across multiple outgoing paths. However, in case of tree topologies, it is not desirable all the time to distribute load evenly because of asymmetric topology architecture which make it almost impossible to tune the load distribution resulting in poor network utilization.

ECMP [21] is a hash-based flow-splitting technique widely adopted for routing in data centers with multi-rooted topologies because of enterprise support. In its simplest implementation, next hop is chosen by determining CRC32 (or other) hash modulo n number of available next hops. Performance of ECMP depends on the performance of hash function selected. ECMP performs static load-splitting among flows as it does not account for flow demand or path dynamics in making allocation decisions.

Hedera [4] and Mahout [13] use flow-based scheduling technique to handle elephant flows. A centralized scheduler is used to reroute elephant flow after demand estimation to a suitable path. All subsequent packets of the flow traverse the new path to increase throughput.

In MicroTE [7], the centralized controller has a routing component which routes predictable and unpredictable traffic. It takes the traffic matrix composed of ToR-to-ToR entries and makes decisions according to available historical measurements. It routes predictable traffic optimally while unpredictable traffic with weighted ECMP. In weighted ECMP, each path is assigned a weight which denotes the bandwidth consumed by predictable traffic. Weighted ECMP is used to mitigate the adverse effects of existing ECMP based approaches.

2.5.2 Packet-based Scheduling Techniques

In packet-based splitting techniques, packets belonging to a single flow are forwarded along different paths that are available between a given source and destination pair. Packets belonging to a flow can be sprayed across different paths either in a random fashion which require no flow state to be maintained by switches or in round-robin fashion in which switches have to maintain per-flow state. These mechanisms ensure fair queues and load-balanced links at the cost of flow disruption. Therefore, additional mechanisms to handle DUPACKs generated by spurious fast retransmissions because of out-of-order packet arrival at the receiver. In [16], Random Early Detection (RED) queue management scheme along with a hypervisor-based solution to handle DUPACKs has been used. Moreover, state-keeping at network device can guarantee in-order arrival of a flows' packets at destination but it increases the overhead drastically.

2.5.3 Flowlet-based Scheduling Techniques

Kandula *et al.* [42] proposed this scheme. In flowlet-based splitting techniques, a single flow is subdivided into multiple short flows based on flow's intrinsic character-

```
                          +-------------------------------+
                          |           Application         |
+---------------+         +-------------------------------+
|  Application  |         |             MPTCP             |
+---------------+         + - - - - - - - + - - - - - - - +
|      TCP      |         | Subflow (TCP) | Subflow (TCP) |
+---------------+         +-------------------------------+
|      IP       |         |       IP      |      IP       |
+---------------+         +-------------------------------+
```

Figure 2.7: Comparison of standard TCP and MPTCP [2]

istics such as its rate and burstiness. It works to achieve low-overhead and accuracy of packet-based splitting and packet ordering of flow-based splitting techniques. It divides a TCP flow into a sequence of multiple short subflows of different sizes and packet bursts by exploiting intra-flow packet spacing. It resides on a switch/router and split up a flow according to the varying split vector and consecutive flows can be routed independently. Intra-flow packet spacing is the time which is larger than maximum Round Trip Time (RTT) of the set of multiple parallel paths. It is low-overhead and robust against packet reordering.

Multipath TCP (MPTCP) [40] is an extension to TCP which is used by data center networks to load balance traffic among all multiple paths by utilizing available bandwidth. It works on the assumption that end hosts are multi-addressed and splits up a single TCP flow into multiple parallel TCP subflows either by using different ports or different IP addresses between a pair of end-hosts. Each subflow is transmitted on a different path. Fig. 2.7 shows a comparison between standard TCP and MPTCP. However, MPTCP has been observed to cause TCP incast for various traffic patterns [11].

CONGA [5] breaks down a flow into flowlets based on congestion in the data center by modifying hardware switches. Edge switches detect flowlets and the feedback received from destination edge switch is used to estimate congestion on the path. Based on this information, a flowlet is assigned to least congested path. CONGA aims to reduce the Flow Completion Time (FCT) of all flows which increases throughput as well. CONGA can also load balance overlay traffic in the data center.

TinyFlow [48] is another scheme which breaks elephant flows into multiple flowlets. The key concept behind TinyFlow is that a large elephant flow is composed of many short flows. In TinyFlow, each edge switch keeps track of elephant flows and breaks it to a sequence of mice flows as soon as a flow completes the byte count of 10KB.

The new short flow is transmitted to a different egress port. Edge switches make this decision locally and require no global state to be maintained in the data center. This random assignment of subflows to different paths may cause several subflows to face more congestion or higher RTTs.

RepFlow [47] is another flow management scheme for short flows. It works by replicating short flows based on a flow-size threshold. if flow size is smaller than 100KB, a replica TCP connection is established between a pair of sender and receiver using different ports and identical packets are sent by both TCP connections. Because of using different ports, ECMP hashing is likely to select two different paths for two identical flows. This is how it reduces FCT of short flows and exploits their ephemeral nature. The replicated flows generate negligible overhead and it does not require modification in TCP protocol at end-hosts or in switching fabric.

A comparison of the elephant flow scheduling techniques is given in the Table 2.2:

Table 2.2: Comparison of Elephant Flow Scheduling Schemes [41]

	Objective	**Working Mechanism**	**Load Balancing Method**	**Control Plane**	**Reaction to Congestion**	**Targeted Traffic**	**Operating Layer**
Hedera	High throughput	Exploiting multipath	Link utilization based flow re-routing, ECMP	Centralized	Relocation of flows	Large flows	L4
Mahout	High throughput	Exploiting multipath	Flow re-routing, ECMP	Centralized	Relocation of flows	Large flows	L4
MicroTE	Minimize congestion	Adaptive load-balancing	Optimization, weighted ECMP	Centralized	Proactive to congestion	All flows	L4
MPTCP	Fairness, high throughput	Load-balancing, multipath, cross-layer design	ECMP, Multipath	Distributed	Relocation of subflows	All flows	L3, L4
CONGA	Reducing mean FCT for short flows	Load-balancing, multipath, hardware modification	congestion-ware flowlet forwarding	Distributed	egress port congestion metric	All flows	L3, L4
TinyFlow	Reducing Mean/Tail FCT for short flows	Load-balancing, multipath, cross-layer	ECMP, Splitting large flows	Distributed	Break down large flows	Large flows	L3,L4
RepFlow	Reducing Mean/Tail FCT for short flows	Exploiting multipath	ECMP, Replicating short flows	Distributed	N/A	Short flows	L5

2.5.4 SDN-OpenFlow based Elephant Flow Management Techniques

Different schemes to manage elephant flows exists in SDN-controlled networks, this section briefly discusses some proposed solutions from SDN domain which identify or mange elephant flows by modifying control or data forwarding layer.

DevoFlow [14] is a slight modification of OpenFlow model which strives to minimize control-to-switch interactions by aggressively using wildcarded rules and approximate counters. It improves two frontiers of OpenFlow model, first on global control and second on statistics gathering. It reduces switch-to-control communication by transferring control to local switch which plays with wildcarded rules intelligently. It extends the action part of a wildcarded entry to a *clone bit*, and clones existing wildcard entry if this bit is set to true otherwise ignores it. The cloned wildcarded rule inherits all fields of a microflow for an exact match entry. Additionally, it augments local switch with a possible set of local flow actions such as multipath and re-routing support. It enhances statistics gathering by adding triggers, reports and by approximate counters. Switches send a report to controller when a trigger is met for a rule while streaming algorithms are implemented in switch ASICs which are used to classify flows transferring more than a threshold bytes thus statistics are selectively pushed to controller.

In [20], authors propose an event-based technique to notify controller about a significant local event such as *elephant* events. It devolves the control plane hierarchically into local controllers and a logically centralized root controller. Local controllers run in close vicinity of switches. In order to detect an elephant flow, an application runs on local controllers to detect elephant flows and notify root controller about an elephant event. The root controller in turn notifies local controllers to install flow entries to re-route elephant flows using global network state.

HybridTE [46] works in an SDN-controlled environment and handles both elephant and mice flows. It relies on some already existing elephant flow detection schemes to identify elephant flows. It uses static routing schemes to route mice flows proactively and periodically re-routes elephant flows at the granularity of five seconds on the shortest paths between a pair of ToR switches. Moreover, it takes false positive mice and false negative elephant flows into account as well.

Furthermore, OpenSketch [50], PayLess [12], OpenTM [44], OpenSample [43], and FlowSense [49] are flow monitoring and management approaches specifically for SDN-controlled data centers. [50], [12], and [44] are query-based monitoring approaches that make use of wildcarded rules, variable frequency-based adaptive polling and periodic polling techniques to monitor flows on switches. [49] uses a passive-push based technique to push packetIn and flowRemoved messages for monitoring link utilization while [43] uses sFlow and TCP sequence numbers to monitor flows and link-level statistics to quickly detect elephant flows and to achieve low latency.

2.5.5 Elephant Flow Management Techniques for Virtualized/ Overlay Environments

This section discusses a few proposed approaches to manage virtual traffic volume in a virtualized data center.

DomainFlow [33] is a multipath flow management prototype which manages overlay (VXLAN) traffic in SDN-controlled data center. It modifies physical OpenFlow-enabled switches to work with rule-based wildcarded and exact-match entries which use VXLAN outer header's UDP port to distribute overlay network traffic across multiple paths using ECMP in the data center.

In [9], Khalil *et al.* has recently proposes a virtual fabric-based approach to manage virtual traffic volume in a virtualized data center. It decentralizes the main network fabric controller to adjust resource allocation according to the virtual data center's respective processing needs, performance and QoS. In order to meet these objectives, a hierarchy of controllers is developed consisting of network fabric controllers, virtual fabric controllers and node controllers. Each physical node in the network is associated with a set of node controllers that periodically pulls node resource utilization and performance measures information and notify global network fabric controller about this information. Each virtual fabric has its own controller associated with it which forwards estimated resources to global and corresponding node controllers in order to meet demands. By leveraging a hierarchy of controllers, vFabric achieves fairness, congestion avoidance and performance isolation among the traffic of different virtualized data centers.

2.6 Summary

This chapter presents background and literature survey of the thesis. It first gives a broad view of virtualized data centers and their trends. Then it discusses elephant flow detection methodologies previously proposed and deployed in data centers. Proceeding section comprehensively analyzes flow management techniques in general following a detailed list of proposed mechanisms used to manage elephant and mice flows specifically. Last two sections of the chapter give an overview of the approaches proposed to manage virtual traffic in SDN-controlled and virtualized data centers.

Chapter 3

Problem Formulation and Design Goals

This chapter presents the problem definition and design objectives in detail. Different scenarios have been discussed to formulate problem.

3.1 Problem Definition

As already discussed, tenants have no control on how their application instances are arranged in the data center, therefore, different instances of a tenant application are scattered across different physical machines in the data center generating a high proportion of east-west virtual traffic. The virtual traffic is encapsulated using tunneling mechanisms, which make it to travel in aggregate, so there is possibility that the aggregated flows between a given pair of physical hosts are a mix of elephants and mice flows encapsulated in the same outer L2 and L3 headers as shown in the Fig. 3.1.

The biggest problem in virtualized environments nowadays is that east-west traffic has to traverse shared underlay fabric and the performance of tenant's applications and services highly depend on the performance of underlying shared network. But first of all, underlay fabric has no visibility into virtual traffic, secondly, static routing or switching mechanisms such as ECMP are usually employed in the data centers. Due to the lack of visibility, data center fabric elements can not isolate large flows from small flows from encapsulated virtual traffic while static routing mechanisms such as ECMP are not capable of handing elephant and mice flows differently. Consequently, an aggregated flow is always hashed to the same path while its constituent flows may require different paths in order to fulfill their requirements. As we know that elephant flows are large flows with high bandwidth requirements, therefore, they may saturate network buffers consuming high bandwidth disproportionately for a long period of time leaving mice flows end up waiting in long queues behind large flows. This may add non-trivial queuing delays to mice flows traversing the same path along with elephant flows within the same aggregate of virtual flow. Consequently, bandwidth intensive applications getting the larger share of network resources, may affectively degrade the performance of co-located tenant's applications which generate latency sensitive mice flows.

Let us suppose two physical servers hosting two different services from two different tenants as shown in the Fig. 3.2. Virtual machines having Virtual Network Identifier (VNI) 0x16 belongs to a tenant generating elephant flows and it transfers huge files from server 1 to server 2 whereas VNI 0x22 belongs to the tenant generating short mice flows consisting of UDP traffic. Virtual Tunnel End Point VTEP1 at physical server 1 encapsulates each flow destined to physical server 2 with its L2 and L3 headers while VTEP2 at physical server 2 decapsulates packets and sends them to the corresponding virtual machines. Switches on the path switch packets belonging to both services entirely based on their outer headers while the flow in real have packets from both services comprising of elephants and mice flows. In this way, UDP flows can get stuck behind large TCP flows adding more latency to UDP traffic, thus effecting performance of co-located tenants badly.

Existing schemes work to identifying elephant flows either on physical host or

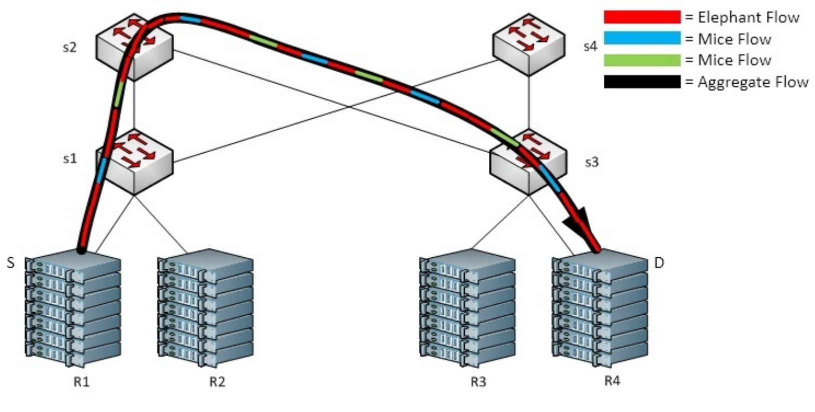


Figure 3.1: An encapsulated aggregate flow

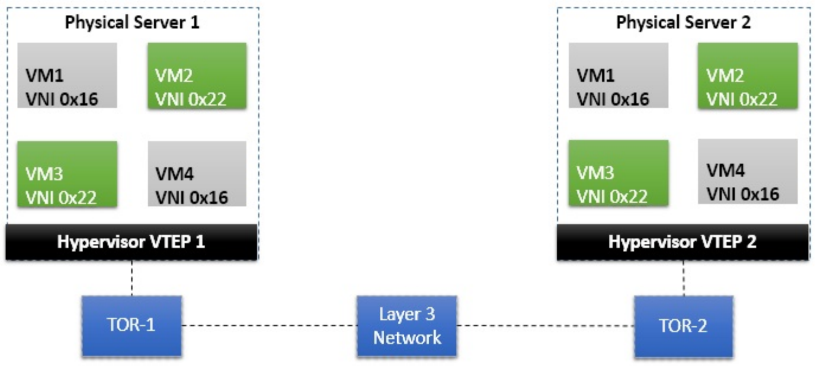


Figure 3.2: Tunnel between two servers

in network and handle them efficiently but none of them identifies elephant flows in a virtual overlay network and map them to different paths when needed. Not isolating elephants from mice flows in an aggregate of overlay traffic may degrade the performance of co-located applications. Hence a mechanism is required which may identify elephant flows belonging to different tenants and map them to different paths in order to minimize bad impact they have on mice flows of other tenants.

3.2 Design Objectives

Our objective is to "*improve performance of co-located tenant applications by identifying elephant flows in virtual network elements and mapping them to different paths where required*"

Previous section explains the problem definition with the help of example. It concluded that aggregation of short and large flows from virtual overlay networks traversing the underlay fabric may affect the performance of co-located applications. If large flows are not identified and mapped to different paths in the physical fabric, they may affect short flows causing long delays i.e., elephants stepping on the mice. Two types of totally different flows with different requirements and demands should be handled as per their requirements. Therefore, a mechanism is required which may identify elephant flows in the virtual overlay network i.e., in the vicinity of virtual machines generating virtual traffic and notify underlay network elements to isolate them from mice flows. Providing isolation at the level of traffic may ultimately improve the unpredictable or bad performance of co-located applications.

3.3 Summary

This chapter introduces the problem of handling virtual traffic in current virtual networks and discusses it with the help of examples. Different scenarios are discussed which highlight the problem in virtual networks. At the end, design objective is presented to provide better isolation among tenant applications when they communicate which in-turn may improve performance of co-located applications. The design objective states that application performance can only be improved if and only if elephants and mice flows from virtual traffic are identified and isolated in the underlay fabric.

Chapter 4

Methodology

This chapter discusses the proposed methodology to identify elephant flows in virtual overlay network and to handle them on physical underlay.

4.1 Elephant Flow Detection

We propose openvswitch-based elephant flow detection scheme for virtualized environments where different applications or services are consolidated on a single physical server. Opevswitch is used to monitor and detect elephant flows because it is at an optimal location at the edge and is in close vicinity of all the virtual machines hosted on a physical server. The proposed detection mechanism efficiently identifies elephant flows belonging to a tunnel when they reach a specified threshold on number of bytes in a given time. This mechanism provides visibility into virtual traffic without introducing any in-network monitoring overhead. Details about design methodology and working mechanism have been discussed in proceeding sections.

4.1.1 Openvswitch Design Overview

The basic openvswitch architecture is shown in Fig. 4.1.

Openvswitch consists of a *userspace* and *kernel space* components. Userspace is a slow path component which runs within the management domain of virtual hosts while kernel space is fast path component which runs within the kernel. Openvswitch imports configuration, forwarding path and connectivity management interfaces through which it manages configuration, manipulates the forwarding state and

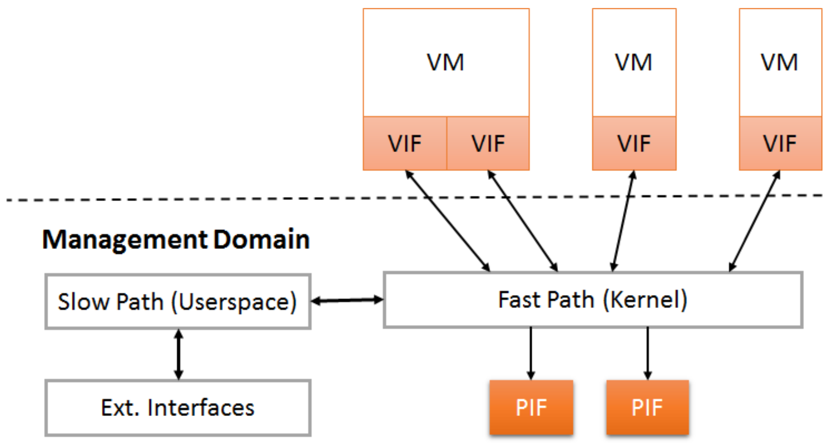


Figure 4.1: Openvswitch architecture

manages local connectivity with hypervisor respectively [37].

The *configuration interfaces* provide connectivity to local or remote processes which manipulate configuration of openvswitch database server. Openvswitch database server is JSON-based configuration database server which is used for persistent configuration of openvswitch daemon. A local or remote process can read/write OVSDB configuration state as key-value pairs and can set up triggers on asynchronous events using OVSDB protocol [36]. Furthermore, it is also responsible to provide binding between large virtual environments and network ports.

The *forwarding path* manages openvswitch flow tables in kernel datapath. It implements the main logic of packet lookup based on L2, L3 or L4 headers which further decides to forward or drop the packet or to en/decapsulate the packet. Through forwarding path interface a remote process can manipulate the forwarding table directly by using OpenFlow protocol.

The *connectivity management* interfaces are used to support integration to local virtualization environments and to manipulate virtual network topological configuration.

4.1.2 Why Openvswitch

Openvswitch is virtual networking component within the hypervisor which provides connectivity between Virtual Interfaces (VIFs) and Physical Interfaces (PIFs) as shown in Fig. 4.2. It is tightly integrated with the hypervisor where it can infer information through virtual machine introspection which is not easily derivable by

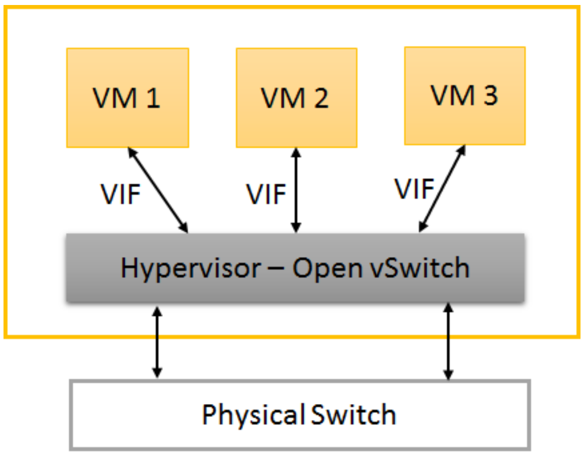


Figure 4.2: Structure of a virtual network [35]

inspecting network traffic alone. It has the flow-level view of all the traffic between VIFs and PIFs and between all VIFs on the physical server. Furthermore, it has full control and visibility into virtual traffic which makes it the best candidate to modify traffic behavior and/or L2/L3 packet bits either in the outer or inner headers. Atop all, it is very close to the source of traffic which makes it the best measurement point.

4.1.3 Openvswitch-based Elephant Flow Detection: Architecture

Elephant detection process works in a control loop of four basic steps as is given in Fig. 4.3. Different steps of this loop are discussed below.

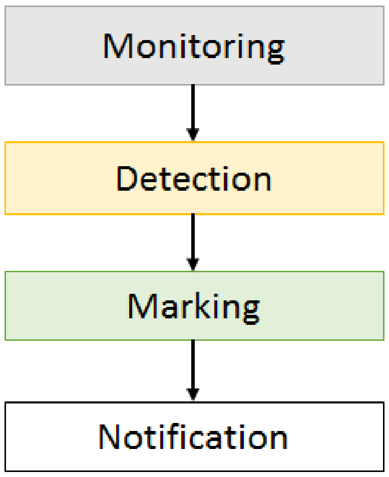


Figure 4.3: Elephant flow detection control loop

4.1.3.1 Monitoring and Detection

Elephant flow detection takes place at openvswitch forwarding path which runs within the linux kernel space. This design basically extends the Mahout [13] elephant flow detection mechanism to virtualization environments except that Mahout monitors TCP socket buffers whereas the proposed solution monitors flow statistics at openvswitch datapath. Rate and time based methodology is adopted to detect elephant flows on openvswitch datapath. Flow statistics associated with each flow entry in the flow table are monitored and elephant flow is identified based on byte count and time threshold. Monitoring for elephant flow detection starts as soon as the new flow entry is installed in the flow table. Outgoing TCP flows associated with tunnels are monitored only.

4.1.3.2 Marking

Marking of elephant flows takes place at forwarding path when a flow crosses a specified threshold of bytes within a given time period, both outer and inner IPV4 header DSCP values are set to a specified value. Marking basically servers two purposes: 1) outer IPV4 DSCP marking signals the controller and 2) inner DSCP value is used to match virtual flow with its tunnel at the controller because a single tenant may have multiple virtual machines generating elephant flows.

4.1.3.3 Notifying Userspace Openvswitch Process

After an elephant flow has been detected and marked, the openvswitch userspace process needs to be notified. Relevant flow and tunnel information is added to the corresponding table in OVSDB so that this information can be exported to a remote process such as an OpenFlow controller. Userspace process is notified only once for each detected elephant flow. Remote controller may use this information to manage large flows efficiently and to have a better global view of large flows in the data center. Additionally, virtual flow's L3 or L4 header information can be used by ECMP for hash calculation to provide better *entropy* in the system.

Algorithm 1 gives a detailed description of elephant detection.

Algorithm 1 Pseudocode for openvswitch-based Elephant Flow Detection

```
1: When a new flow entry
2: If TCP and Tunnel
3:   if packet count == 1
4:     set previous time = current time of the flow
5:     previous byte count = initial byte count of the flow
6:     set flow marking time = 0
7:   end if
8:   if current time - previous time >threshold
9:     if current byte count - previous byte count >threshold
10:      if elephant flow marking time == 0
11:        set IP header DSCP value
12:        notify userspace
13:        flow marking time = current time
14:      end if
15:      else
16:        if current time - flow marking time >threshold
17:          flow marking time = current time
18:          set IP header DSCP value
19:        end if
20:    end if
21:  end if
22: end if
```

4.2 Elephant Flow Handling

Elephant flows are handled in two steps: first at openvswitch and later at OpenFlow controller.

4.2.1 Elephant Flow Handling at Openvswitch

Elephant flows are partially handled at openvswitch i.e., openvswitch is used to break a large elephant flow into multiple subflows or flowlets as shown in Fig. 4.4. As already discussed, openvswitch is at the optimal location on the edge because it has full visibility and control into traffic that passes through it and it can easily modify the behavior of traffic before it can hit and overwhelm the network. Large flows greatly differ in their rate, size, duration and burstiness [39], [51], so deriving this information in the network to handle large flows is a hectic task for both openFlow and non-openFlow based networks. At openvswitch, a flow is closer to its source, so it is fairly easy to introspect its varying rate and to treat it according to some pre-defined criteria. In order to divide a large flow into small flows at openvswitch, a byte count threshold is specified and as soon as an elephant flow's unstable rate crosses the threshold, it is marked with a different DSCP value in outer IPV4 header. The bursts of packets with different DSCP values are identified as subflows at the ToR switch. In this way, a large flow is subdivided into many subflows sequentially with each subflow being routed to a different path. Fig. 4.5 gives a hierarchical view of flow splitting schemes.

4.2.2 Elephant Flow Handling at OpenFlow Controller

ToR switch redirects all detected elephant flows and subflows to controller because a different DSCP value results in a table miss. The controller maintains a global view of elephant flows and congestion state of the network and puts each subflow to lightly loaded path. In this way, large flows occupy path resources for only short time without hurting mice flows.

4.3 Working Mechanism

The proposed mechanism works by taking byte count and time threshold as an input pre-configured by the administrator. When a new flow entry is installed at the datapath, elephant flow detection control loop starts as well. If the flow achieves the maximum rate through which it crosses the threshold specified in a given time, it is classified as elephant flow and is further processed for marking and handling on openvswitch and as well as on controller. Otherwise, it just leaves the control

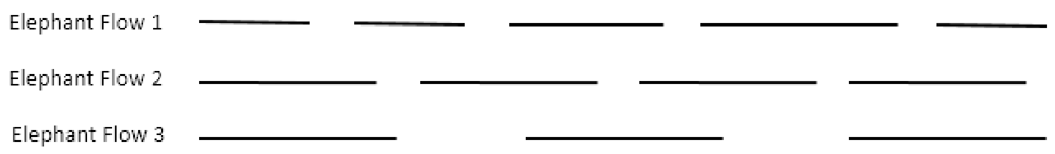


Figure 4.4: Dividing an elephant flow into subflows at openvswitch

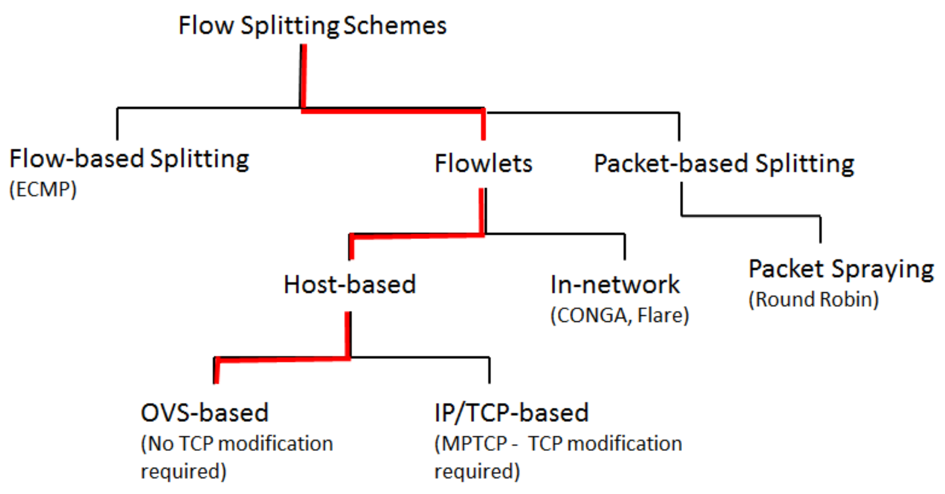


Figure 4.5: Host-based flow splitting

loop and is processed by openvswitch on normal basis. A detected elephant flow is handled partially at openvswitch and partially at the controller which simplifies the task of managing large flows in the network. As detection is performed at openvswitch, a large flow is broken down into many short flows based upon its intrinsic characteristics such as rate and burstiness. As soon as an elephant flow crosses the specified threshold of byte count, it is split up and is marked with different DSCP values. The different DSCP value allows ToR switches to redirect it to the controller which further schedules subflows based on the current network situation. Furthermore, information related to all detected elephant flows on virtual networks is added to openvswitch database so that it can be exported to any remote process. A remote process such as openFlow controller can set up triggers on the relevant table and is updated as soon as a new elephant flow event is triggered. The openvswitch database flow entry contains virtual flow and its corresponding tunnel information. This information is used by controller to efficiently handle virtual traffic and to load balance the underlay fabric. An elephant flow entry is removed from openvswitch database table immediately after it expires.

4.4 Summary

In this chapter, the methodology and working mechanism to handle heavy hitters from virtual overlay networks is discussed. It first discusses openvswitch architecture and why it is suitable for elephant flow detection and handling in current virtualized environments and then describes a detailed elephant flow detection control loop at openvswitch along with algorithm used. Later it discusses how detected elephant flows are handled partially at openvswitch and partially at the controller.

Chapter 5

Implementation and Results

Previous chapter discusses methodology from an abstract view, this chapter presents implementation details. Section 5.1 gives implementation details of elephant flow detection and handling at openvswitch and controller respectively. Section 5.2 describes all hardware and software details of the testbed setup. Sections 5.3 and 5.4 discuss the traffic simulation and performance criteria. Details about benchmarking and result discussions are discussed in section 5.5.

5.1 Elephant Flow Detection and Handling Implementation Details

Internal details of elephant flow detection control loop are represented in Fig. 5.1. In order to implement detection scheme, openvswitch source code has been modified.

Openvswitch kernel module is modified to implement detection scheme. Each flow entry is monitored for a threshold of 12500000 bytes (i.e., 100Megabit or 12.5 MegaByte) on a 1000 Mbps NIC for a period of 2 seconds, i.e., 10% of NIC utilization. After an elephant flow is detected, both VXLAN and GRE modules are signaled to mark outer IPV4 header's DSCP bits with a specified value. The first packet of each elephant flow is marked with DSCP value 1. This DSCP marking is performed both on external and internal IPV4 headers with the same DSCP value. After openvswitch VXLAN and GRE modules have been informed to mark packets of detected elephant flows, an elephant flow push notification is sent to the userspace process. The notification upcall sends a kernel to userspace netlink message containing modified packet,

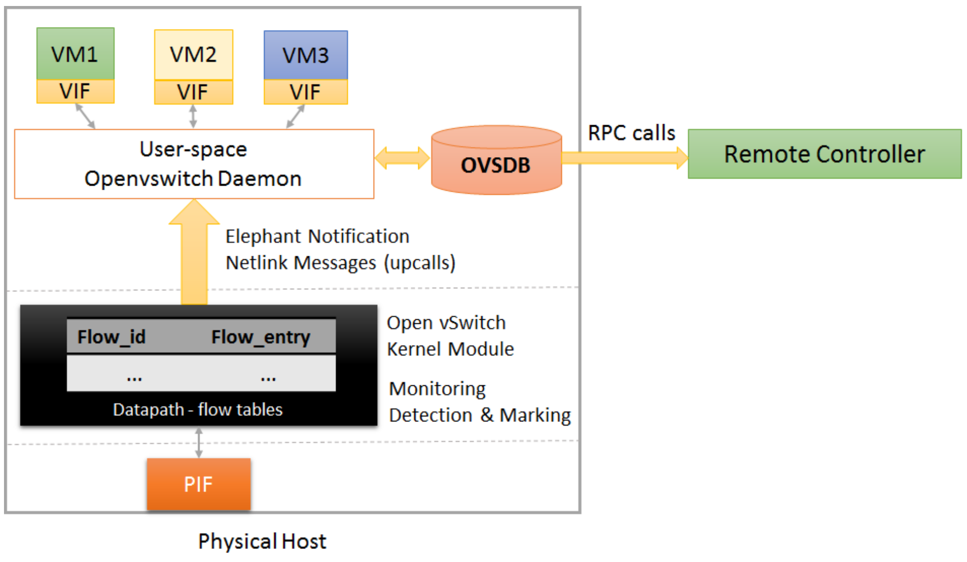

VM1
VIF
VM2
VIF
VM3
VIF
User-space Openvswitch Daemon
OVSDB
RPC calls
Remote Controller
Elephant Notification Netlink Messages (upcalls)
Flow_id
Flow_entry
Datapath - flow tables
Open vSwitch Kernel Module
Monitoring Detection & Marking
PIF
Physical Host

Figure 5.1: Detection control-loop detailed architecture

datapath ID, flow key and upcall command ID. In userspace process, an upcall handler thread listens to all incoming upcalls received from kernel module and processes these upcalls based upon the specific command ID. After an elephant upcall is processed, it informs OVSDB sever to add an elephant flow entry to the elephant flow table. OVSDB schema has also been modified to add a new *"elephant flow table"* which contains a list of columns including br-name, eth_dst, eth_src, ip_dst, ip_src, ttl, tos, tp_src, tp_dst, tcp_flags, tun_type, tun_ip_src, tun_ip_dst, tun_tos, tun_ttl and tun_id etc as shown in Fig. 5.2. For handling elephant flows, an elephant flow is divided into flowlets by marking flow's tunnel header with different DSCP values after the flow crosses a specified number of bytes at openvswitch. For example, for every detected elephant flow, first DSCP value is set to 1 which increments every time the flow crosses a specified threshold of bytes, 12.5MB in this case. This information is used at ToR switch by proactively installing wildcard rules to redirect packets with different DSCP values to the controller.

Elephant flow scheduling mechanism has been implemented using PoX controller [30] and is tested against *4-array* fat-tree topology which is implemented in Mininet [27] as shown in Fig. 2.5. The controller application detects elephant flows and route them on different paths in two ways: first using *Round Robin*, a stateless method and secondly by routing flows on the *least-loaded* path which is a statefull method of routing flows. The stateless scheduling mechanism routes flows by not taking load on candidate paths into consideration while a stateless scheduling scheme selects a least-loaded path to route a flow in the data center. Furthermore, elephant flows on controller are sub-divided into short flows based on a time threshold. Each identified elephant flow is divided into a flowlet as soon as it completes a duration of 2.0 seconds or greater. Each flowlet is routed to a different path from a set of parallel multiple equal cost paths.

5.2 Testbed Setup

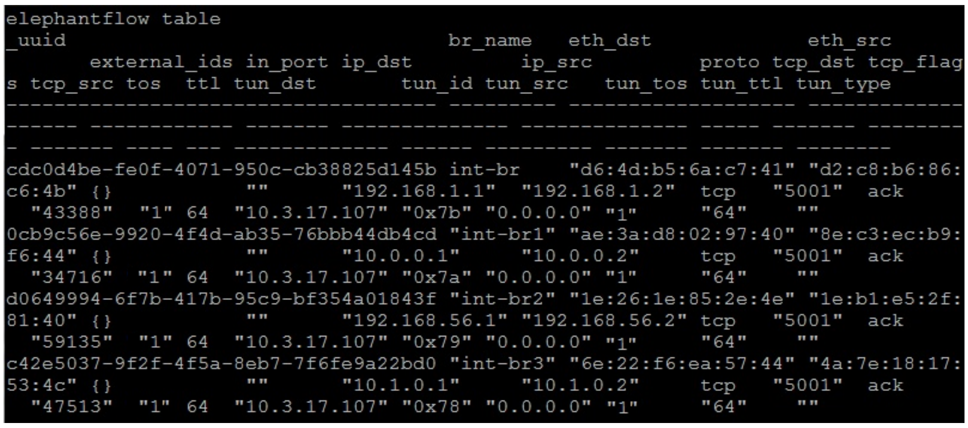
Following components have been used to setup testbed.

Hardware: Two systems with x86-64 bit Intel(R) Core(TM)2 Duo CPU processor with 3.0GHz capacity were used for setting up VXLAN and GRE tunnels. Furthermore, a c4.xlarge with high frequency Intel Xeon E5-2666 v3 (Haswell) with optimized processors amazon EC2 instance was used for benchmarking of results.

Virtualization Hypervisor: Kernel based Virtual Machine (KVM) [24] module was installed on Ubuntu 14.04 Linux Operating System with 3.13.0-24-generic kernel.

Switch: Openvswitch version 2.3.90 with 7.11.2 DB Schema version is used to implement elephant flow detection, marking and handling scheme.

OpenFlow Controller: On the two hardware machines, OVS-controller is used while OpenFlow 1.0 controller was used for scheduling elephant flows in PoX-Mininet based data center.



```
elephantflow table
_uuid                                br_name   eth_dst             eth_src
       external_ids in_port ip_dst         ip_src         proto tcp_dst tcp_flag
s tcp_src tos  ttl tun_dst       tun_id tun_src   tun_tos tun_ttl tun_type
------------------------------------ --------- ------------------- -------------
------ ------------ ------- -------------- -------------- ----- ------- --------
- ------- ---- --- ------------- ------ --------- ------- ------- --------
cdc0d4be-fe0f-4071-950c-cb38825d145b int-br    "d6:4d:b5:6a:c7:41" "d2:c8:b6:86:
c6:4b" {}           ""      "192.168.1.1"  "192.168.1.2"  tcp   "5001"  ack
  "43388"  "1" 64  "10.3.17.107" "0x7b" "0.0.0.0" "1"     "64"    ""
0cb9c56e-9920-4f4d-ab35-76bbb44db4cd "int-br1" "ae:3a:d8:02:97:40" "8e:c3:ec:b9:
f6:44" {}           ""      "10.0.0.1"     "10.0.0.2"     tcp   "5001"  ack
  "34716"  "1" 64  "10.3.17.107" "0x7a" "0.0.0.0" "1"     "64"    ""
d0649994-6f7b-417b-95c9-bf354a01843f "int-br2" "1e:26:1e:85:2e:4e" "1e:b1:e5:2f:
81:40" {}           ""      "192.168.56.1" "192.168.56.2" tcp   "5001"  ack
  "59135"  "1" 64  "10.3.17.107" "0x79" "0.0.0.0" "1"     "64"    ""
c42e5037-9f2f-4f5a-8eb7-7f6fe9a22bd0 "int-br3" "6e:22:f6:ea:57:44" "4a:7e:18:17:
53:4c" {}           ""      "10.1.0.1"     "10.1.0.2"     tcp   "5001"  ack
  "47513"  "1" 64  "10.3.17.107" "0x78" "0.0.0.0" "1"     "64"    ""
```

Figure 5.2: Elephant flow entries in OVSDB "elephantflow" table

5.3 Traffic Simulation

Iperf tool has been used to generate TCP and UDP based elephant and mice flows in the data center. Iperf provides different options to generate different flows for example, varying time duration, varying size in KB, MB, or GB, adjustable TCP-window size and two or three parallel flows to test performance metrics of a data center.

TCP traffic is generated as front-end traffic following different patterns such as stag_prob, strides, random and all_to_all [3]. The *stag_prob* traffic pattern is inter-pod or intra-pod traffic pattern where a host sends a flow to another host in the same subnet with a certain probability called "*SubnetP*" and to another host in another pod with probability "*PodP*" while to the rest of the hosts with probability *1-SubnetP-PodP*. Our testbed generates flows with 0.2, 0.3 and 0.5, 0.3 probabilities. In *stride* traffic pattern, a host with index x sends the traffic to another host with *index(x+i)modTotal_hosts*. A stride traffic pattern simulates MapReduce and Hadoop traffic patterns. In stride1, each host at index x sends traffic to the host with index next to it such that (x+1)modTotal_hosts. A 4-array fat-tree topology can accommodate $4^3/4$=16 hosts so (x+1)mod16. In stride2, each host with index x sends traffic to the host at index (x+2)mod16. In stride4, host at index x sends traffic to host with index x+4 and in stride8, each host at index x sends traffic to host at index x+8mod16. In *random* traffic pattern, traffic is send to any other host with equal probability and it simulates RPC and HTTP traffic patterns. In random traffic pattern, a host sends traffic to any other host in the network with uniform probability i.e., 1 and there are destination hosts which may receive 2 or 3 flows from senders, and in *randombij* traffic patterns, bijection exists between senders and receivers so that each destination host is the receiver of no more than exactly one flow. In *hotspot* traffic pattern, all hosts try to send one or two randomly selected hosts in the data center which may increase load on certain paths causing a 'hotspot'.

In our test, background UDP traffic has also been generated in a separate thread using a random traffic pattern.

5.4 Performance Criteria

Detection: The performance criteria for elephant flow detection mechanism is that it identifies elephant flows belonging to different tunnels accurately without any false positives and false negatives. A false positive is defined as a short flow identified mistakenly as an elephant flow while a false negative is an elephant flow which has not been identified as elephant flow.

Data Center Performance: Data center bisection bandwidth has been used as a performance metric to evaluate data center performance because it is the worst-case bandwidth to be measured in data centers.

5.5 Benchmarking and Result Discussion

Both elephant flow detection and scheduling schemes have been tested and evaluated extensively by generating a range of TCP flows using iperf utility. However, detection mechanism has been tested and evaluated separately than scheduling mechanism. Following sections give a detailed description of testing and evaluation of the proposed methodology.

5.5.1 Elephant Flow Detection: Testing and Evaluation

In order to test detection schemes, two physical machines are used with Ubuntu 14.04, 3.13.0-24-generic kernel, KVM and openvswitch 2.3.90 installed on them. Four virtual machines are hosted on both systems. All virtual machines are configured with different IP addresses from different virtual overlay networks. The virtual overlay network is created and managed by openvswitch by establishing VXLAN and GRE tunnels between them. Three GRE tunnels with 120, 121 and 122 key values and one VXLAN tunnel with 123 VNI is established between four virtual machines. This simulation shows a virtual network environment with four different tenants communicating over four tunnels. This test case is represented in Fig. 5.3.

Iperf utility in server mode is run on physical server 2 and in client mode on physical server 1. Large and short flows of different sizes and rates are generated between pair of virtual machines over same overlay. Results have been summarized in Table 5.1.

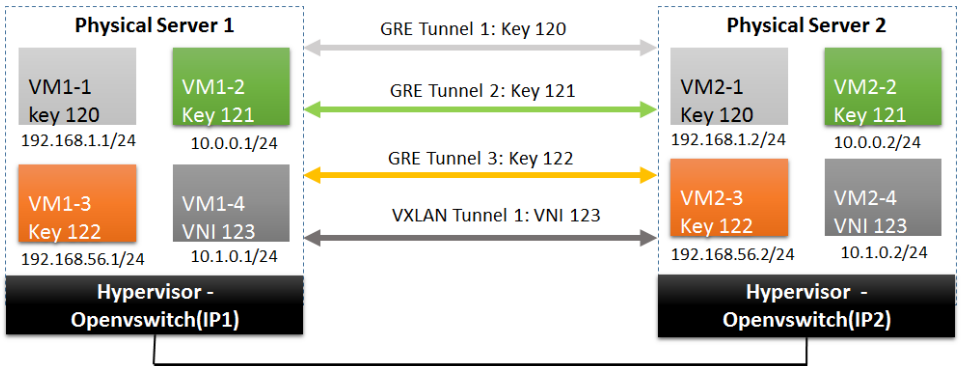


Figure 5.3: Testbed configuration for detection scheme

Results show that elephant flows belonging to different tunnels have been identified efficiently. First of all, it differentiates among flows belonging to different tunnels, secondly it differentiates between elephants and mice flows. Flows are identified as elephant flows at different duration of their life time because each flow grows at a different rate than other flows and they live for a different time duration. It is likely that a flow acquire a sufficiently high rate within a few seconds after it starts and it may take more time to acquire a rate to be fallen into the category of elephant flows. All short TCP flows are not identified as elephant flows by the proposed schemes because they were not able to acquire a sufficiently high rate to cross the threshold specified.

Table 5.1: Elephant flow detection scheme evaluation

	Flow Size (M/KByte)	**Flow Duration (Seconds)**	**Bandwidth (M/Kbits/s)**	**TCP Window size (KB)**	**Detected jiffies (Seconds)**	**No. of Bytes at detection**
1	50.0 M	33.3	12.4 (M)	85.0	6	12500116
2	50.0 M	425.6	985 (K)	85.0	43	12501048
3	40.0 M	335.8	1.05 (M)	85.0	11	12500198
4	10.0 M	82.7	1.01 (M)	85.0	34	12502900
5	512 K	13.1	320 (K)	85.0	-	-

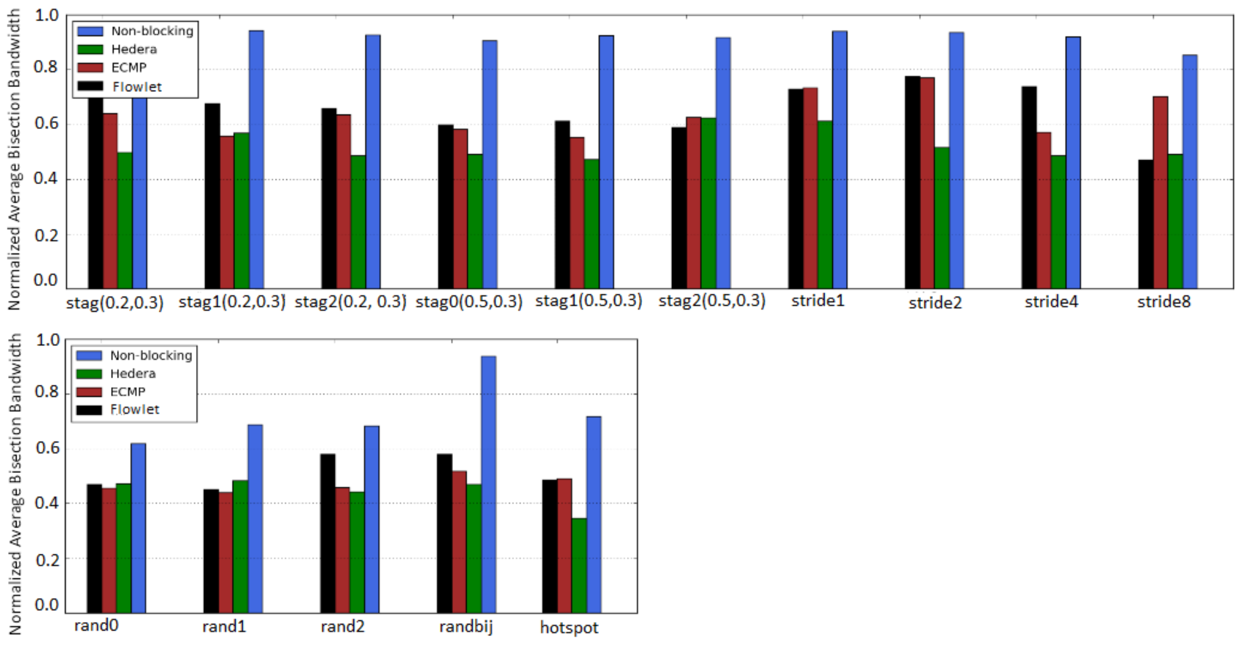


Figure 5.4: Simulation results, duration 60s, flowlet interval 2.0 seconds - stateless

5.5.2 Elephant Flow Scheduling: Testing and Evaluation

The performance of controller is benchmarked against ECMP [29] controller, Hedera [4] controller, and a non-blocking topology. All controllers were run with 4-array fat-tree topology implemented using Mininet except non-blocking because it consists of a single non-blocking switch. Different TCP traffic patterns are generated by running **iperf** commands between different pairs of Mininet hosts.

Iperf commands were run for the duration of 30, 40, 50, 60, 70 , 80, 90 and 100 seconds and results were gathered by using linux bandwidth monitoring utility "**bwm-ng**". Virtual ethernet NICs of all mininet hosts were monitored for their number of bytes sent i.e., tx_bytes are sampled over a specified duration for all traffic patterns using tcpbrobe and bwm-ng utilities on all controllers described above. Using this sampled data, average bisection bandwidth for all hosts is calculated and results are plotted. The normalized average bisection bandwidth of data center against each controller is benchmarked on c4.xlarge amazon EC2 instance as shown in Fig. 5.4, Fig. 5.5, Fig. 5.6, and Fig. 5.7.

Elephant flows were subdivided into flowlets using different intervals from few milliseconds to few seconds i.e., from 0.05 milliseconds to 2.5 seconds and with both stateless and statefull scheduling techniques but long intervals such as 2.0 and 1.5

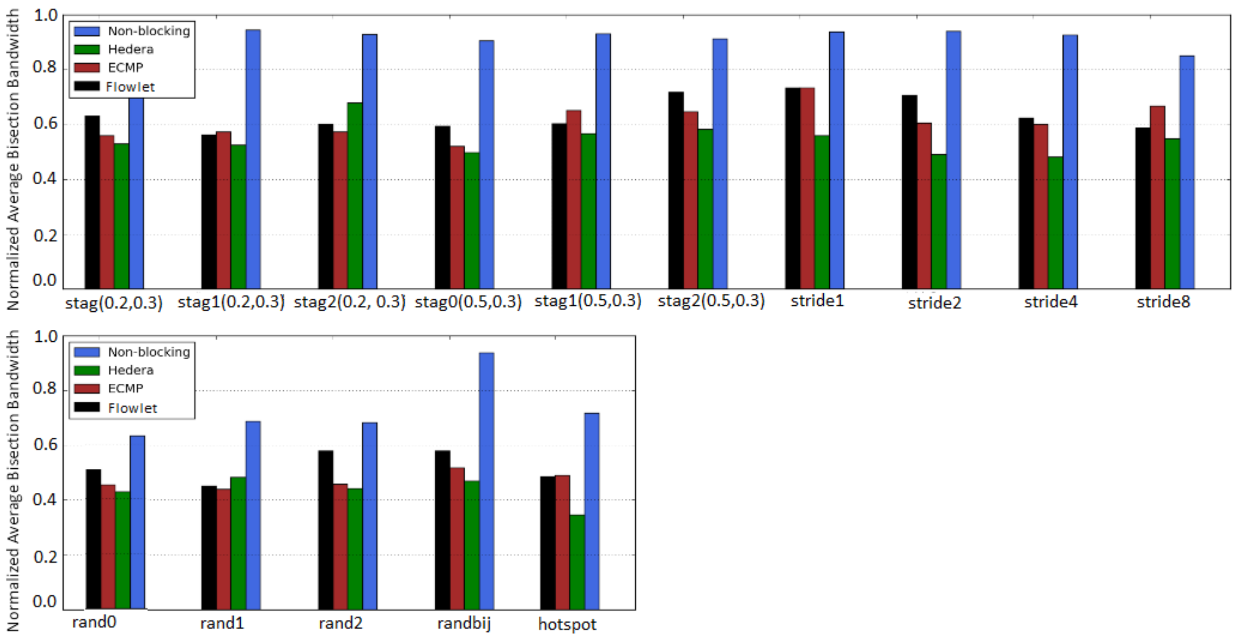


Figure 5.5: Simulation results, duration 70s, flowlet interval 2.0 seconds - stateless

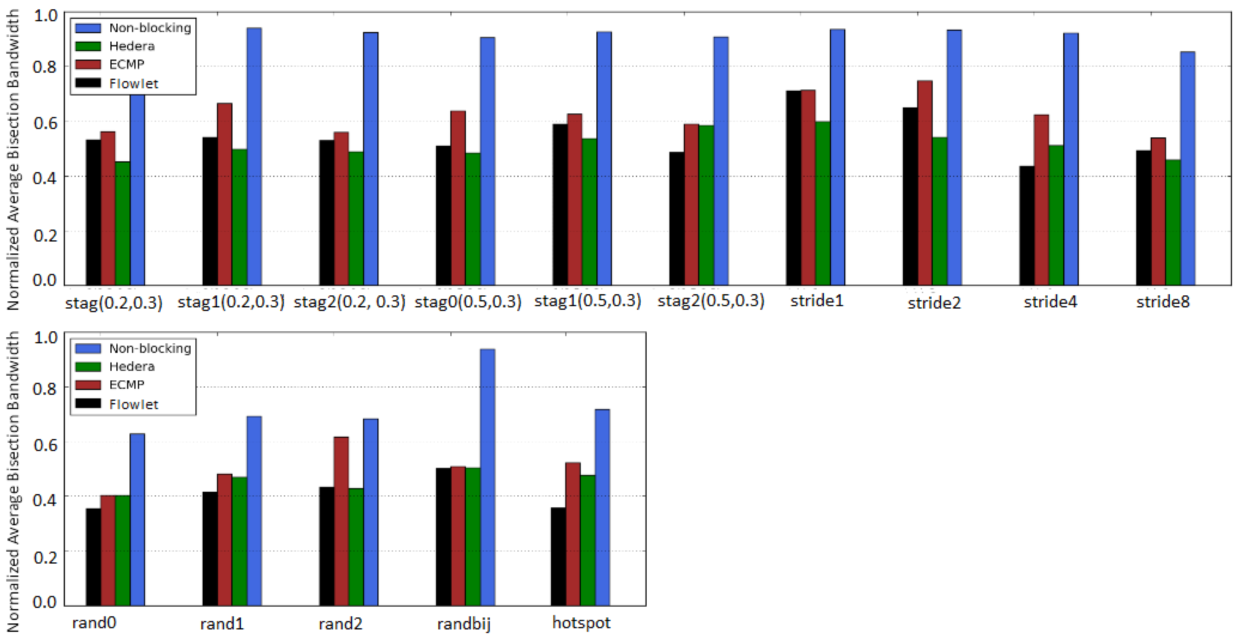


Figure 5.6: Simulation results, duration 50s, flowlet interval 0.05 seconds - statefull

with stateless round robin scheduling always perform good. The benchmarking results indicate that elephant flows with 2.0 interval flowlets using Round Robin scheduling outperforms Hedera and ECMP for all traffic patterns.

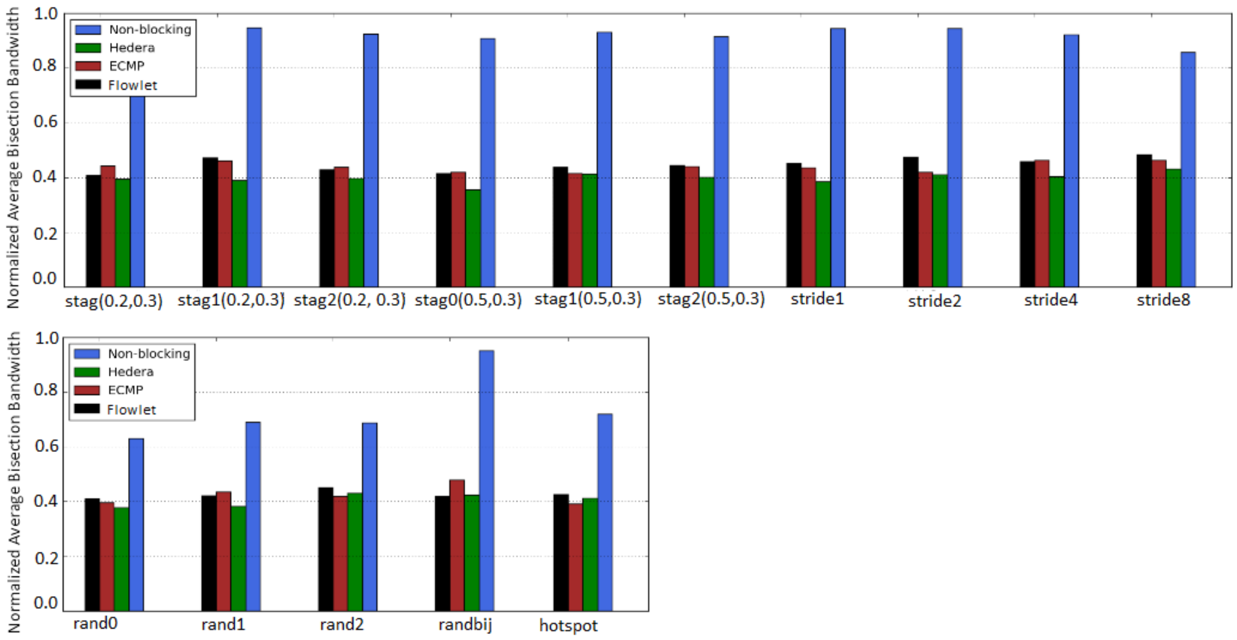


Figure 5.7: Simulation results, duration 60s, flowlet interval 2.0 seconds with background mice flows - stateless

5.6 Summary

In this chapter, implementation details along with testbed setup are discussed in detail. Openvswitch, PoX and Mininet is used along with KVM based virtualization. OVS-controller is used to manage and setup flows on physical hosts while a PoX controller is used with Mininet 4-array fat-tree topology. Traffic is generated with iperf and bechmarking tests are performed on fat-tree topology against ECMP, Hedera and proposed scheduling schemes. At the end, bisection bandwidth is calculated using tcpprobe and bwm-ng utilities and results show that proposed technique outperforms Hedera and ECMP for all traffic patterns with stateless scheduling and a small number of flowlets.

Chapter 6

Conclusion & Future Work

In this chapter, the conclusion with a summary of the research findings along with future directions is presented.

6.1 Conclusion

We presented how openvswitch can be leveraged to handle important aspects related to traffic engineering in data centers simply by using the information it maintains in the datapath. At the edge, elephant flows generated by a tenant's virtual machines are detected efficiently and accurately without introducing any overhead either on the edge or in network. Handling large flows at openvsiwtch by breaking them into multiple short flows using DSCP values further simplifies the problem. Moreover, it provides visibility to detected large flows by exporting their L2, L3 and L4 information to openvswitch database. With this solution in place, a network controller is freed from the burden of monitoring each incoming flow and then classifying it as elephant flow after it meets a certain criteria, and can optimally schedule large flows so that they have less impact on mice flows.

We have evaluated our detection and handling schemes by performing different tests using different traffic patterns. Analysis revealed that elephant flows related to different tenants are identified accurately with minimal overhead and it reports no false positives and false negatives. By applying elephant flow detection and handling schemes, bisection bandwidth is improved and network overhead has been reduced greatly.

This modified version of openvswitch can be used in virtualized data centers with high degree of VM consolidation where bandwidth of the system is major issue. By augmenting proposed scheme with existing routing mechanisms, it is expected to reduce network overhead, and improve network performance greatly.

6.2 Future Work

Our proposed solution is currently tested for a small data center on Mininet, but it can be extended to a cloud-based virtualized data center where virtual machines are consolidated with high level and multiple physical servers are involved each with openvswitch installed. In our proposed work, byte count and time threshold is manually configured, this work can be further extended to automate the elephant flow detection configuration process on openvswitch. Additionally, as openvswitch is very close to the source of traffic generated by virtual machines, it can learn about what type of applications are responsible to generate elephant or mice flows.

Moreover, openvswitch allows its datapath component to be offloaded to hardware for efficiency purposes, there is also need to know that what is the behavior of elephant detection process in hardware after datapath has been offloaded to it. Finally, in High Performance Computing environments, there is possibility that physical machines use passthrough PCI mechanisms such as SRIOVs, using these mechanisms, openvswitch is completely bypassed so it will not be possible to detect elephant flows at openvswitch. Hence, a mechanism is required which could be used on host machine to cater this type of situations.

References

[1] Amazon web services: Overview of security processes.

[2] Ietf rfc- 6824 tcp extensions for multipath operation with multiple addresses.

[3] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. A scalable, commodity data center network architecture. *ACM SIGCOMM Computer Communication Review*, 38(4):63–74, 2008.

[4] Mohammad Al-Fares, Sivasankar Radhakrishnan, Barath Raghavan, Nelson Huang, and Amin Vahdat. Hedera: Dynamic flow scheduling for data center networks. In *NSDI*, volume 10, pages 19–19, 2010.

[5] Mohammad Alizadeh, Tom Edsall, Sarang Dharmapurikar, Ramanan Vaidyanathan, Kevin Chu, Andy Fingerhut, Francis Matus, Rong Pan, Navindra Yadav, George Varghese, et al. Conga: Distributed congestion-aware load balancing for datacenters. In *Proceedings of the 2014 ACM conference on SIGCOMM*, pages 503–514. ACM, 2014.

[6] Theophilus Benson, Aditya Akella, and David A Maltz. Network traffic characteristics of data centers in the wild. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pages 267–280. ACM, 2010.

[7] Theophilus Benson, Ashok Anand, Aditya Akella, and Ming Zhang. Microte: Fine grained traffic engineering for data centers. In *Proceedings of the Seventh COnference on emerging Networking EXperiments and Technologies*, page 8. ACM, 2011.

[8] Robert Birke, Mathias Björkqvist, Cyriel Minkenberg, Martin Schmatz, and Lydia Y Chen. When virtual meets physical at the edge: A field study on

datacenters' virtual traffic. In *Proceedings of the 2015 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pages 403–415. ACM, 2015.

[9] Khalil Blaiech and Omar Cherkaoui. Virtual fabric-based approach for virtual data center network. In *Proceedings of the 2nd International Workshop on Software-Defined Ecosystems*, pages 31–38. ACM, 2015.

[10] Zhiruo Cao, Zheng Wang, and Ellen Zegura. Performance of hashing-based schemes for internet load balancing. In *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 1, pages 332–341. IEEE, 2000.

[11] Yanpei Chen, Rean Griffith, Junda Liu, Randy H Katz, and Anthony D Joseph. Understanding tcp incast throughput collapse in datacenter networks. In *Proceedings of the 1st ACM workshop on Research on enterprise networking*, pages 73–82. ACM, 2009.

[12] Shubhajit Roy Chowdhury, M Faizul Bari, Rizwan Ahmed, and Raouf Boutaba. Payless: A low cost network monitoring framework for software defined networks. In *Network Operations and Management Symposium (NOMS), 2014 IEEE*, pages 1–9. IEEE, 2014.

[13] Andrew R Curtis, Wonho Kim, and Praveen Yalagandula. Mahout: Low-overhead datacenter traffic management using end-host-based elephant detection. In *INFOCOM, 2011 Proceedings IEEE*, pages 1629–1637. IEEE, 2011.

[14] Andrew R Curtis, Jeffrey C Mogul, Jean Tourrilhes, Praveen Yalagandula, Puneet Sharma, and Sujata Banerjee. Devoflow: Scaling flow management for high-performance networks. In *ACM SIGCOMM Computer Communication Review*, volume 41, pages 254–265. ACM, 2011.

[15] B Davie and J Gross. A stateless transport tunneling protocol for network virtualization (stt). *draft-davie-stt-04*, 2013.

[16] Advait Dixit, Pawan Prakash, and Ramana Rao Kompella. On the efficacy of fine-grained traffic splitting protocolsin data center networks. In *ACM SIG-*

COMM Computer Communication Review, volume 41, pages 430–431. ACM, 2011.

[17] 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*, 41(4):339–350, 2011.

[18] Marisol García-Valls, Tommaso Cucinotta, and Chenyang Lu. Challenges in real-time virtualization and predictable cloud computing. *Journal of Systems Architecture*, 60(9):726–740, 2014.

[19] Pankaj Garg and Yu-Shun Wang. Nvgre: Network virtualization using generic routing encapsulation. 2014.

[20] Soheil Hassas Yeganeh and Yashar Ganjali. Kandoo: a framework for efficient and scalable offloading of control applications. In *Proceedings of the first workshop on Hot topics in software defined networks*, pages 19–24. ACM, 2012.

[21] Christian E Hopps. Analysis of an equal-cost multi-path algorithm. 2000.

[22] [Online]. Available: http://philipcao.com/2014/07/28/2014-gartner-magic-quadrant-for-data-center networking. 2014 gartner magic quadrant for data center networking.

[23] [Online]. Available: http://www.cisco.com. Cisco global cloud index: Forecast and methodology 2013-2018 white paper.

[24] [Online]. Available: http://www.linux kvm.org. Kvm. 2015.

[25] Zhiming Hu and Jun Luo. Cracking network monitoring in dcns with sdn. In *Proc. IEEE INFOCOM*, 2015.

[26] Srikanth Kandula, Sudipta Sengupta, Albert Greenberg, Parveen Patel, and Ronnie Chaiken. The nature of data center traffic: measurements & analysis. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, pages 202–208. ACM, 2009.

[27] Bob Lantz, Brandon Heller, and Nick McKeown. A network in a laptop: rapid prototyping for software-defined networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, page 19. ACM, 2010.

[28] Mallik Mahalingam, D Dutt, Kenneth Duda, Puneet Agarwal, Lawrence Kreeger, T Sridhar, Mike Bursell, and Chris Wright. Virtual extensible local area network (vxlan): A framework for overlaying virtualized layer 2 networks over layer 3 networks. Technical report, 2014.

[29] Vijay Mann, Akshay Vishnoi, and Sarvesh Bidkar. Living on the edge: Monitoring network flows at the edge in cloud data centers. In *Communication Systems and Networks (COMSNETS), 2013 Fifth International Conference on*, pages 1–9. IEEE, 2013.

[30] J Mccauley. Pox: A python-based openflow controller, 2014.

[31] N McKee. Inmon corporation's sflow: A method for monitoring traffic in switched and routed networks. Technical report, 2001.

[32] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, 2008.

[33] Yukihiro Nakagawa, Kazuki Hyoudou, Chunghan Lee, Shinji Kobayashi, Osamu Shiraki, and Takeshi Shimizu. Domainflow: Practical flow management method using multiple flow tables in commodity switches. In *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies*, pages 399–404. ACM, 2013.

[34] Radhika Niranjan Mysore, Andreas Pamboris, Nathan Farrington, Nelson Huang, Pardis Miri, Sivasankar Radhakrishnan, Vikram Subramanya, and Amin Vahdat. Portland: a scalable fault-tolerant layer 2 data center network fabric. In *ACM SIGCOMM Computer Communication Review*, volume 39, pages 39–50. ACM, 2009.

[35] Justin Pettit, Jesse Gross, Ben Pfaff, Martin Casado, and Simon Crosby. Virtual switching in an era of advanced edges. In *2nd Workshop on Data Center–Converged and Virtual Ethernet Switching (DC-CAVES)*, 2010.

[36] Ben Pfaff and Bruce Davie. The open vswitch database management protocol. 2013.

[37] Ben Pfaff, Justin Pettit, Keith Amidon, Martin Casado, Teemu Koponen, and Scott Shenker. Extending networking into the virtualization layer. In *Hotnets*, 2009.

[38] Junaid Qadir, Anwaar Ali, Kok-Lim Alvin Yau, Arjuna Sathiaseelan, and Jon Crowcroft. Exploiting the power of multiplicity: a holistic survey of network-layer multipath. *arXiv preprint arXiv:1502.02111*, 2015.

[39] Lin Quan and John Heidemann. On the characteristics and reasons of long-lived internet flows. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pages 444–450. ACM, 2010.

[40] Costin Raiciu, Sebastien Barre, Christopher Pluntke, Adam Greenhalgh, Damon Wischik, and Mark Handley. Improving datacenter performance and robustness with multipath tcp. *ACM SIGCOMM Computer Communication Review*, 41(4):266–277, 2011.

[41] Roberto Rojas-Cessa, Yagiz Kaymak, and Ziqian Dong. Schemes for fast transmission of flows in data center networks.

[42] Shan Sinha, Srikanth Kandula, and Dina Katabi. Harnessing tcps burstiness with flowlet switching. In *Proc. ACM SIGCOMM Workshop Hot Topics in Networks (HotNets)*. Citeseer, 2004.

[43] Junho Suh, TT Kwon, C Dixon, W Felter, and J Carter. Opensample: A low-latency, sampling-based measurement platform for sdn. ICDCS, 2014.

[44] Amin Tootoonchian, Monia Ghobadi, and Yashar Ganjali. Opentm: traffic matrix estimator for openflow networks. In *Passive and active measurement*, pages 201–210. Springer, 2010.

[45] Philip Wette and Holger Karl. Which flows are hiding behind my wildcard rule?: adding packet sampling to openflow. In *ACM SIGCOMM Computer Communication Review*, volume 43, pages 541–542. ACM, 2013.

[46] Philip Wette and Holger Karl. Hybridte: Traffic engineering for very low-cost software-defined data-center networks. *arXiv preprint arXiv:1503.04317*, 2015.

[47] Hong Xu and Baochun Li. Repflow: Minimizing flow completion times with replicated flows in data centers. In *INFOCOM, 2014 Proceedings IEEE*, pages 1581–1589. IEEE, 2014.

[48] Hong Xu and Baochun Li. Tinyflow: Breaking elephants down into mice in data center networks. In *Local & Metropolitan Area Networks (LANMAN), 2014 IEEE 20th International Workshop on*, pages 1–6. IEEE, 2014.

[49] Curtis Yu, Cristian Lumezanu, Yueping Zhang, Vishal Singh, Guofei Jiang, and Harsha V Madhyastha. Flowsense: Monitoring network utilization with zero measurement cost. In *Passive and Active Measurement*, pages 31–41. Springer, 2013.

[50] Minlan Yu, Lavanya Jose, and Rui Miao. Software defined traffic measurement with opensketch. In *NSDI*, volume 13, pages 29–42, 2013.

[51] Yin Zhang, Lee Breslau, Vern Paxson, and Scott Shenker. On the characteristics and origins of internet flow rates. In *ACM SIGCOMM Computer Communication Review*, volume 32, pages 309–322. ACM, 2002.