Optimizing Flow Rule Installations on SDN based Switches

Naman Grover

A Thesis Submitted to

Indian Institute of Technology Hyderabad

In Partial Fulfillment of the Requirements for

The Degree of Master of Technology

भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad

Department of Computer Science and Engineering

June 2015

Declaration

I declare that this written submission represents my ideas in my own words, and where ideas or words of others have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be a cause for disciplinary action by the Institute and can also evoke penal action from the sources that have thus not been properly cited, or from whom proper permission has not been taken when needed.

[illegible]

(Signature)

(Naman Grover)

CS13M1014

(Roll No.)

Approval Sheet

This Thesis entitled Optimizing Flow Rule Installations on SDN based Switches by Naman Grover is approved for the degree of Master of Technology from IIT Hyderabad

[illegible]

(Dr. Kiran Kuchi) Examiner
Dept. of Electrical Engineering
IITH

[illegible]

(Dr. Subrahmanyam Kalyanasundaram) Examiner
Dept. of Computer Science and Engineering
IITH

[illegible]

(Dr. Kotaro Kataoka) Adviser
Dept. of Computer Science and Engineering
IITH

[illegible]

(Dr. Bheemarjuna Reddy Tamma) Chairman
Dept. of Computer Science and Engineering
IITH

Acknowledgements

First and foremost, I would like to express my sincere gratitude to my adviser Dr. Kotaro Kataoka for his valuable guidance, constant encouragement, motivation, enthusiasm, and immense knowledge. Furthermore I would like to thank Pranet peer group for healthy discussions on various potential research topics and constant willingness to help me in my endeavors. Many individuals contributed in many different ways to the completion of this thesis. I am deeply grateful for their support, and thankful for the unique chances they offered me. Finally, I thank my family for supporting me throughout all my studies at the institute. I would like to make a special mention of the excellent facility provided by my institute, IIT Hyderabad.

Abstract

Traditional network monitoring involving packet capturing or flow sampling has many challenges such as scalability, accuracy and availability of processing resource when networks become large-scale, high-speed and heterogeneous. SDN is a promising approach to address these challenges, in which highly granular flow rule installations can provide us with fine-grained flow based statistic. But each SDN switch has its own capacity limitation, such as its cache memory called TCAM, which can get exhaused with a large number of highly granular flow rule installations. Thus, network nodes need coordination of resources with other network nodes to monitor the network in a scalable manner. This thesis introduces an intelligent framework, called liteFlow, which divides flow rule installations into two parts, monitoring and forwarding flow rules. The proposed system distributes the load of monitoring flows among SDN switches, and makes the scalability and accuracy of network monitoring manageable. Also, we introduce a forwarding mechanism which uses a more abundant L2 cache in SDN switches based on MAC labels.

Contents

Chapter 1

Introduction

Network monitoring is fundamental to examine the state of enterprise networks. It is used for daily network management operations like traffic engineering, troubleshooting, anomaly detection, QoS support and accounting etc. Today's networks are large and complex. Distributed environment and resource constraints make network management rather difficult. In addition, the network operations mentioned above require fine grained application-level flow details, which may cause an additional overhead to capture.

Flow-based measurement techniques such as NetFlow[1] and sFlow[2] provide generic support for some measurement tasks. However, their network resource consumption is very high [3]. Also, sampling makes these techniques unusable for other monitoring operations, though we can make small changes in these techniques to support a particular monitoring operation, such as [4] [5]. This essentially means, for different network operations, we need different parameters, that limit the scalability of these techniques. Hence, enabling a fine-grained and robust monitoring framework, which can cater to large variety of monitoring operations is interesting.

OpenFlow [6] is widely adopted realization of SDN. It has enabled switches to perform flow-based control of packets. Each switch maintains flow tables on them, which consists of flow rules, dictating the actions to be performed on incoming packets. Incoming packets are indexed in the flow table by extracting packet match fields. Based on the matched flow rule, corresponding actions, such as forwarding, dropping, broadcasting etc., are taken. In a flow rule, there is also a field called *counter*, which provides a few statistics about the matched packets on the corresponding flow. OpenFlow provides a variety of fields, on which packets can be matched. It also provides the flexibility of choosing match fields, and wildcarding others. With an intelligent mechanism of installing flow rules on switches, coupled with the central view of the network elements in SDN, this thesis aims to design and implement a platform which can provide fine-grained, unsampled, application-level statistics that are useful for a plethora of network monitoring applications.

In OpenFlow, if an unknown packet arrives at a switch, it sends the packet as a *packet_in* message to the SDN controller. Because of this *packet_in* message, the controller has to make a decision on what flow rule should be installed on the switch. Installing a 5-tuple flow rule $<srcIP, dstIP, srcPort, dstPort, protocol>$ or other high level flow rules, and wildcarding the rest of the match fields, will suffice to the need of knowing application-level detail of a flow.

Table 1.1 [21] shows the number of L2 and TCAM(ternary content-addressable memory) flow rules entries supported by four SDN switches. The IP based flow rules are matched in the TCAM, while MAC based forwarding rules are stored in L2 MAC tables. As shown, TCAM rule space has minimal capacity limits when compared to L2 MAC table rule space. If we were to install a 5-tuple IP based flow rule on all path switches for a flow, this will result in TCAM rule space exhaustion, and unfilled L2 tables rule space. In this research work, we try to leverage L2 and TCAM rule space to get fine-grained flow based statistics.

Table 1.1: Switch Table Sizes

Table	**Broadcom Trident**	**HP ProVision**	**Intel FM6000**	**Mellanox SwitchX**
TCAM	2K+2K	1500	24K	0?
L2/Eth	100K	64K	64K	48K
ECMP	1K	unknown	0	unknown

Using the current OpenFlow protocol, we introduce *liteFlow*, a lightweight, distributed flow-based monitoring platform for SDN. *liteFlow* consists of three separate modules, $PseudoMAC$ Forwarding and $FlowPartitioner$. $pseudoMACForwarding$ provides a forwarding backbone utilizing l2 entries of the switches. $FlowPartitioner$ divides the flow rule space into $forwarding$ and *monitoring* rules. It also attempts to distribute the IP *monitoring* responsibility among switches while consuming TCAM rule space optimally.

1.1 Overview of our work

This work includes the study of SDN and OpenFlow for engineering the enterprise network. *liteFlow* is proposed to address efficient hardware resource utilization on SDN switches with an application to network monitoring. OpenFlow is used for implementing liteFlow. liteFlow load balances and reduces the load of flow rule installations on TCAM of switches. It has capability to efficiently migrate flow rules from one switch to other without noticable packet loss.

1.2 Thesis Outline

The thesis is structured as follows. Section 2 describes about the SDN and related concepts essential for this work. We define this work in Section 3. Literature survey and related studies are presented in Section 4. *liteFlow* approach to flow rules installation and optimization is presented in Section 5. We evaluate *liteFlow* in Section 6. In Section 7, we describes our approach towards Network Function Virtualisation.

Chapter 2

Software Defined Networking

Computer networks are large, complex and difficult to manage. Traditional networks are ossified considering their non-programmable, vertically integrated, closed and vendor specific architecture. It is difficult to control and manage the network as there is no centralized way to do so. Networking devices run complex and distributed control software that is typically closed and proprietary, and each device needs to be configured individually.

Software Defined Networking (SDN) paradigm promises to simplify the control and management of the network. SDN aims to make networks more simple, dynamic, open and programmable. OpenFlow [6] was the first open standard interface for implementing the SDN.

2.1 Software Defined Networking

A traditional networking device consists of data plane and control plane as shown in Figure 2.1. Data plane is used to forward a packet and control plane is used to determine where to forward the packet. For instance, in a learning switch, data plane is responsible for packet forwarding and control plane keeps a MAC table to determine the output port for the incoming packet. SDN architecture separates out the control plane and data plane of a networking device. In SDN architecture, control plane is programmable and logically centralized (known as the controller) which allows network administrators to control all the data-plane elements by writing a single control program. Network intelligence is centralized in the SDN controller which maintains a global view of the network. Switches communicate with a centralized controller through an open standard (such as OpenFlow). SDN facilitates the deployment of new services and protocols in the network, due to its vendor independence architecture and network virtualization. It also reduces the capital and operational costs for deploying and managing the network. Common SDN applications are network virtualization [7], network monitoring [8][9], load balancing [10], user authentication [11] and cloud or data center network [12] etc.

Figure 2.2 shows a logical view of SDN architecture. With a global view of the network at the controller, applications and policy-engines which are built on top of the controller, view networking devices as a single logical switch. Controller communicates with all the devices through an open-standard. Networking devices are simple and implement only basic packet forwarding mechanism.

SDN is not a new idea but has gained traction in recent times [13][14]. Many vendors (such as

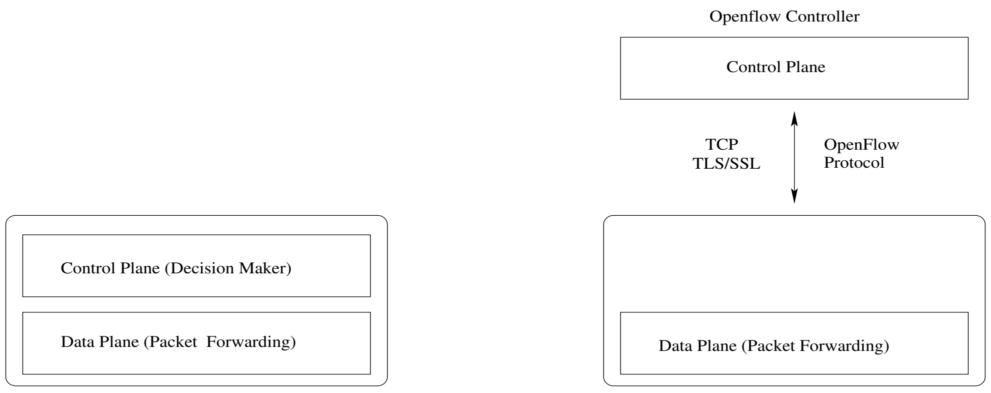

Openflow Controller
Control Plane
TCP
TLS/SSL
OpenFlow
Protocol
Control Plane (Decision Maker)
Data Plane (Packet Forwarding)
Data Plane (Packet Forwarding)

Figure 2.1: Traditional vs OpenFlow Switch

Cisco) have their proprietary implementations of the concept of SDN. OpenFlow is a widely accepted implementation of SDN across the industry and academic research communities. The OpenFlow protocol is open source and aims at making network programmable, innovative and vendor agnostic. One of the advantages of OpenFlow and its vendor independence is the rise of the concept of virtual switches. These are software level switches which are implemented usually as user-space or kernel-space software. One such example is Open vSwitch [15] which implements the OpenFlow protocol. This enables any regular computer to be used as networking hardware and reduces the need to purchase expensive hardware from proprietary vendors.

2.2 OpenFlow Protocol

OpenFlow is a protocol designed by the Open Networking Foundation(ONF) which promotes and adopts SDN through open standards development. OpenFlow was the first SDN standard to realize the concept of Software Defined Networking. The OpenFlow protocol is spoken between OpenFlow enabled switch (SDN switch) and OpenFlow Controller as shown in Figure 2.1. OpenFlow allows to control the network on per-flow basis in a fine-grained manner.

Table 2.1: A flow entry

Match Fields	Counters	Actions

Table 2.2: Match fields used to match packets against flow entries

Ingress Port	Ether src	Ether dst	Ether type	VLAN Id	VLAN Priority	IP src	IP dst	IP ToS bit	TCP/UDP src port	TCP/UDP dst port

In OpenFlow protocol, switches only consist of a forwarding plane that is equipped with flow tables. A switch can have multiple flow tables. Each flow table contains several flow rules. Flow rules are similar to forwarding or routing rules in traditional switches and routers. Each packet is

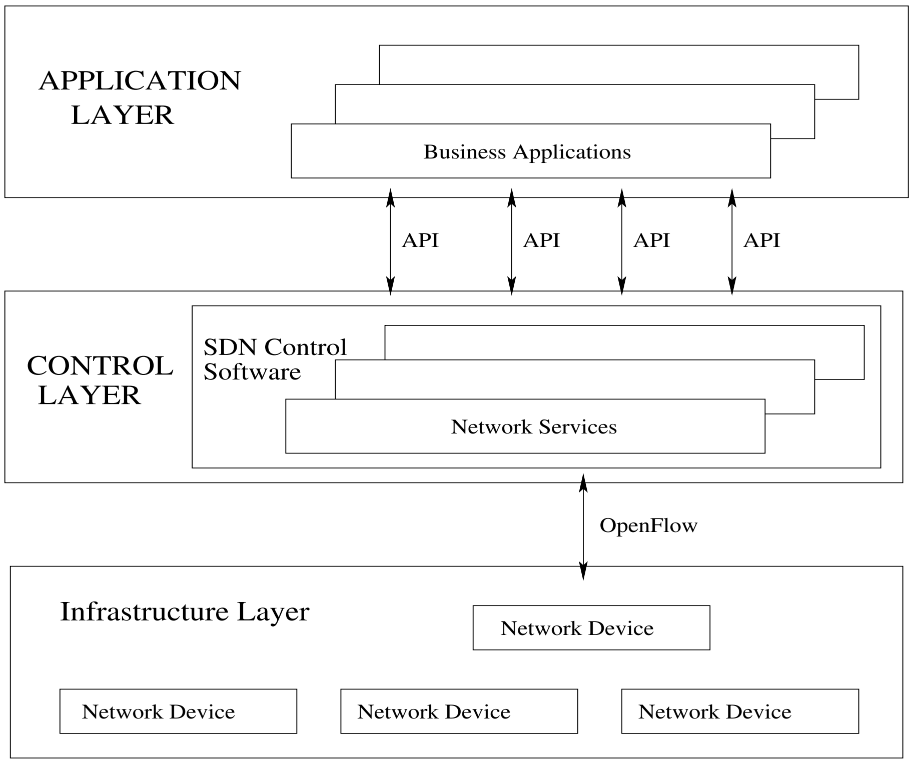


Figure 2.2: Software-Defined Network Architecture (Source: [])

matched with flows rules in the flow tables. A flow rule includes a match, actions and counters as shown in Table 2.1. The OpenFlow protocol defines the fields which are included in the flow rules for matching. It currently supports matching up to the transport layer as shown in Table 2.2.

When the OpenFlow switch receives a packet and it has no matching flow rule for the packet, it forwards the packet to the controller through the packet in message. The logic implemented in the controller then determines the actions for such packets. Depending on the logic, an OpenFlow switch can work as a router, switch, firewall, or network address translator etc. Controller either installs a flow rule on the switch by sending a f low mod message or sends a packet out message. If a flow rule is installed on a switch, then the packet in message will not be sent for packets which match to that flow rule unless it is mentioned in the action explicitly. Once a flow rule is matched to a packet then counters corresponding to that flow are updated and corresponding actions are executed on that packet of the flow. The flow rules also have two timeout values: Idle timeout and Hard timeout, which control when the flow should be removed from the flow table of the switch automatically. Flows can also be removed by the controller explicitly. The OpenFlow protocol works on top of TCP and has support for TLS/SSL encryption.

Currently a few hardware vendors like Big Switch Networks, HP, and Pronto support OpenFlow in their hardware switches. Some of the available OpenFlow controllers are Floodlight [16], Ryu [17], Trema [18], NOX/POX [19] etc.

2.3 Pipeline Processing

The OpenFlow pipeline of every OpenFlow switch contains multiple flow tables, each flow table containing multiple flow entries. The OpenFlow pipeline processing defines how packets interact with those flow tables (see Figure 2.3). An OpenFlow switch with only a single flow table is valid, in this case pipeline processing is greatly simplifed. The flow tables of an OpenFlow switch are

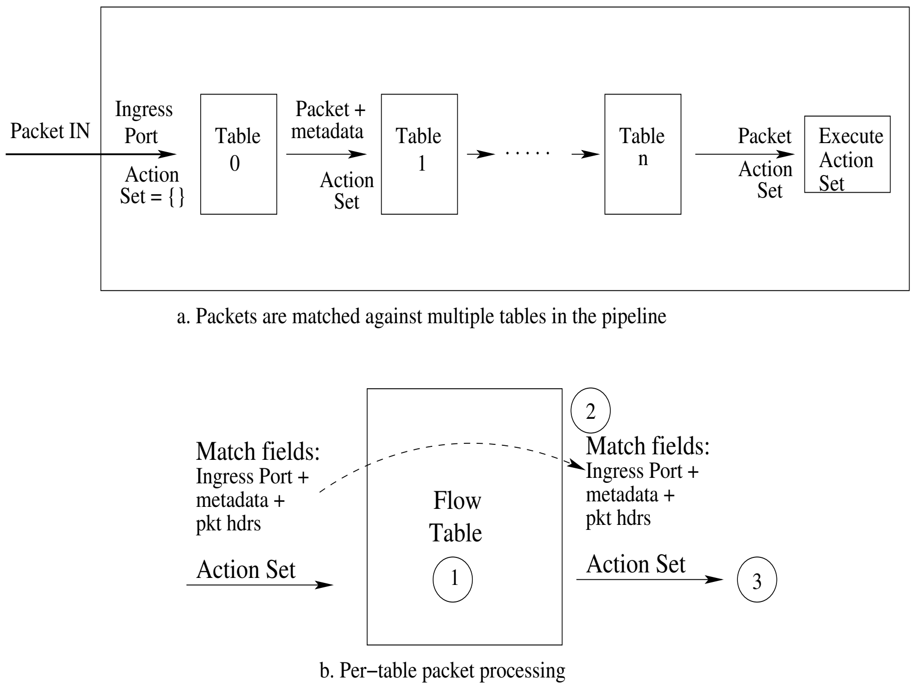


Figure 2.3: Packet Flow through the processing pipeline (Source: [20])

sequentially numbered, starting at 0. Pipeline processing always starts at the first flow table: the packet is first matched against entries of flow table 0. Other flow tables may be used depending on the outcome of the match in the first table.

If the packet matches a flow entry in a ow table, the corresponding instruction set is executed. The instructions in the flow entry may explicitly direct the packet to another flow table, where the same process is repeated again. A flow entry can only direct a packet to a flow table number which is greater than its own flow table number, in other words pipeline processing can only go forward and not backward. Obviously, the flow entries of the last table of the pipeline can not include the Goto instruction. If the matching flow entry does not direct packets to another flow table, pipeline processing stops at this table. When pipeline processing stops, the packet is processed with its associated action set and usually forwarded.

If the packet matches a flow entry in a flow table, the corresponding instruction set is executed. The instructions in the flow entry may explicitly direct the packet to another flow table, where the same process is repeated again. A flow entry can only direct a packet to a flow table number which is greater than its own flow table number, in other words pipeline processing can only go forward and not backward. Obviously, the flow entries of the last table of the pipeline can not include the Goto instruction. If the matching flow entry does not direct packets to another flow table, pipeline

processing stops at this table. When pipeline processing stops, the packet is processed with its associated action set and usually forwarded.

Chapter 3

Problem Statement

Due to SDN being deployed mostly in campus networks, or proof of concept implementations, there is not much emphasis given to optimization at the SDN switch level. As discussed in the next Section, there has been a lot of work done on flow monitoring in SDN, but none talk about the switch hardware load incurred due to it. Flow monitoring require heavy duty flow rule installations on switches. This hefty task may lead to exhaustion of cache memory of switches, called TCAM. And hence, new flow rules will be installed in software of the switches, lead to very slow packet matching and switching. Our aim is to provide a framework which allows us to load balance and reduce the load incurred on the TCAM of switches due to flow monitoring.

Chapter 4

Related Work

There has been a considerable amount of work done in the field of flow-based network monitoring for traditional networks. Sekar et al. [22] present a minimalist approach for network flow monitoring. They use flow sampling and sample-and-hold as sampling primitives and configure these primitives on routers using cSamp [23] in a coordinated fashion across the network. NetFlow [1] and sFlow [2] are commonly used technologies for implementing network flow monitoring. As they rely on sampling techniques, they can miss the several small flows and are not well suited for some applications such as [24] which require some specific packets involving connection setup phase of a TCP flow.

Network monitoring using OpenFlow has also been explored in recent years. OpenSAFE [9] routes the traffic for network analysis and requires separate monitoring appliances. OpenNetMon [25] uses adaptive polling for determining throughput, latency and packet loss. OpenSketch [26] is an SDN based measurement architecture similar to OpenFlow. A three stage pipeline (hashing, filtering, and counting) is implemented in the commodity switches. It provides a measurement library to use these sketches. Upgradation or replacement of SDN switches is required to support this. FlowSense [27] uses push based approach to determine the link utilization in the network. It uses only *packet_in* and $flow\_mod$ messages to gather the required information. Our approach for $FlowMon$ is similar to FlowSense.

Some research has gone into effective usage of TCAM resources and reducing controller load. Devoflow [28] aims to reduce the controller-switch interaction and the number of TCAM entries in the switch. This is done through an effective mechanism of devolving controller's flow setup responsibility back to the switches. Controller maintains the visibility over only large elephant flows, while switches take local routing actions to forward the rest of flows without invoking controller. There system requires a new design for OpenFlow. DIFANE [29] also propose a mechanism for reduction of controller load by keeping the traffic in the data plane. In their approach, controller runs a partition algorithm to partition flow rules into high level flow rules and low level flow rules. High level flow rules are assigned to designated switches, called Authority Switch. Instead of the controller, Authority switch is invoked for flow setup. There approach has fixed Authority switches.

For $forwarding$ flows, we use a similar concept of label switching used in [30] and MPLS []. In particular, in order to use large L2 MAC tables makes switching using MAC addresses a favourable option. As in [30], we also change destination MAC address to a MAC label in the edge switches, and forward packets based on these labels in the core switches. But [30] has one or more MAC labels for each host depending on the traffic which is flowing through. While we use MAC labels for aggregating flows from different hosts traversing the same path.

Chapter 5

liteFlow: Load Balancing Platform for Lightweight and Distributed Flow Monitoring in SDN

Network monitoring is fundamental to examine the state of enterprise networks. It is used for daily network management operations like traffic engineering, troubleshooting, anomaly detection, QoS support and accounting etc. Today's networks are large and complex. Distributed environment and resource constraints make network management rather difficult. In addition, the network operations mentioned above require fine grained application-level flow details, which may cause an additional overhead to capture.

Flow-based measurement techniques such as NetFlow[1] and sFlow[2] provide generic support for some measurement tasks. However, their network resource consumption is very high [3]. Also, sampling makes these techniques unusable for other monitoring operations, though we can make small changes in these techniques to support a particular monitoring operation, such as [4] [5]. This essentially means, for different network operations, we need different parameters, that limit the scalability of these techniques. Hence, enabling a fine-grained and robust monitoring framework, which can cater to large variety of monitoring operations is interesting.

OpenFlow [6] is widely adopted realization of SDN. It has enabled switches to perform flow-based control of packets. Each switch maintains flow tables on them, which consists of flow rules, dictating the actions to be performed on incoming packets. Incoming packets are indexed in the flow table by extracting packet match fields. Based on the matched flow rule, corresponding actions, such as forwarding, dropping, broadcasting etc., are taken. In a flow rule, there is also a field called *counter*, which provides a few statistics about the matched packets on the corresponding flow. OpenFlow provides a variety of fields, on which packets can be matched. It also provides the flexibility of choosing match fields, and wildcarding others. With an intelligent mechanism of installing flow rules on switches, coupled with the central view of the network elements in SDN, this thesis aims to design and implement a platform which can provide fine-grained, unsampled, application-level statistics that are useful for a plethora of network monitoring applications.

In OpenFlow, if an unknown packet arrives at a switch, it sends the packet as a *packet_in* message to the SDN controller. Because of this *packet_in* message, the controller has to make a decision on what flow rule should be installed on the switch. Installing a 5-tuple flow rule $<srcIP, dstIP, srcPort, dstPort, protocol>$ or other high level flow rules, and wildcarding the rest of the match fields, will suffice to the need of knowing application-level detail of a flow.

Table 5.1 [21] shows the number of L2 and TCAM flow rules entries supported by four SDN switches. The IP based flow rules are matched in the TCAM, while MAC based forwarding rules are stored in L2 MAC tables. As shown, TCAM rule space has minimal capacity limits when compared to L2 MAC table rule space. If we were to install a 5-tuple IP based flow rule on all path switches for a flow, this will result in TCAM rule space exhaustion, and unfilled L2 tables rule space. In this research work, we try to leverage L2 and TCAM rule space to get fine-grained flow based statistics.

Table 5.1: Switch Table Sizes

Table	**Broadcom Trident**	**HP ProVision**	**Intel FM6000**	**Mellanox SwitchX**
TCAM	2K+2K	1500	24K	0?
L2/Eth	100K	64K	64K	48K
ECMP	1K	unknown	0	unknown

Using the current OpenFlow protocol, we introduce *liteFlow*, a lightweight, distributed flow-based monitoring platform for SDN. *liteFlow* consists of three separate modules, $PseudoMAC$ Forwarding and $FlowPartitioner$. $pseudoMACForwarding$ provides a forwarding backbone utilizing l2 entries of the switches. $FlowPartitioner$ divides the flow rule space into $forwarding$ and *monitoring* rules. It also attempts to distribute the IP *monitoring* responsibility among switches while consuming TCAM rule space optimally.

5.1 PeudoMac Forwarding

liteFlow uses a label-switching forwarding technique, similar to [30], to optimize the number of flow rules installed by $FlowPartitioner$. In this approach, we assign a label to each path in the network. The label is essentially a 48 bit MAC address chosen distinctly and randomly. Each path in the core network is assigned this unique MAC label, using which forwarding decisions are taken. The path labels are used to aggregate flows on the path so that L2 rule space can be conserved in switches. In OpenFlow-compatible switches, packet header rewriting can be done at line rates [30]. By leveraging this fact, this forwarding approach works by changing the destination MAC address of packets at the ingress switch to a suitable path label, called $PseudoMAC$ address, and forwarding packets through core non-edge switches by matching on destination MAC based flow rules. At the egress, we use destination IP based flow rules to change destination address MAC back from the pseudo to the one of the host.

Figure 5.1 illustrates how the packet forwarding with $PseudoMAC$ address work. Hosts $H1 \rightarrow H3$ and $H2 \rightarrow H3$ are two *hostpair* which are communicating on the path $S1 \rightarrow S4$. $P1$ is the assigned $PseudoMAC$ label for this path. For both the $hostpairs$, we change the destination MAC address to $PseudoMAC$ address "$P1$" and forward the packet through port 2 of $S1$. On the core switches $S2$ and $S3$, they match the packets on the destination MAC address $P1$ and forward

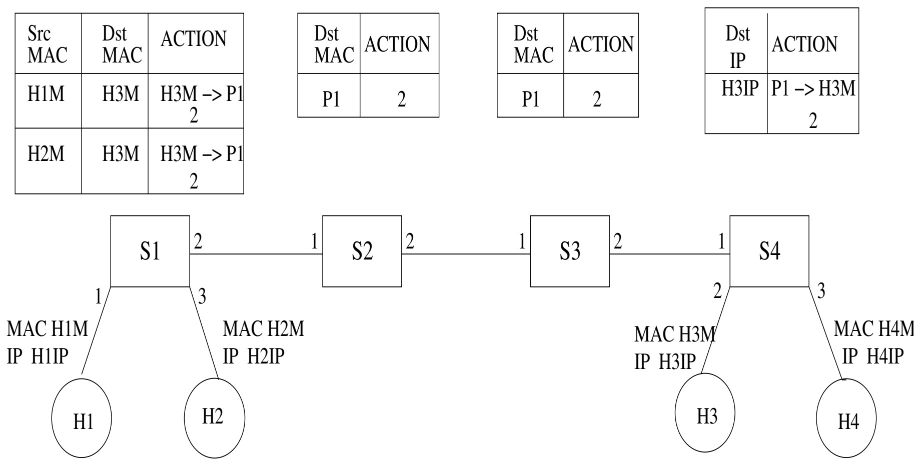


Figure 5.1: PseudoMAC Forwarding Approach

the packet through port 2. At the egress switch $S4$, using the destination IP based flow rules, we revert the destination MAC address $P1$ to $H3M$ of the host $H3$. In the process, we achieve packet forwarding through flow rules installed in L2 MAC table. Note that at the egress switch, we have flow rule in TCAM rule space, but the number of flow rules would depend just on the number of flow rules installed at that switch.

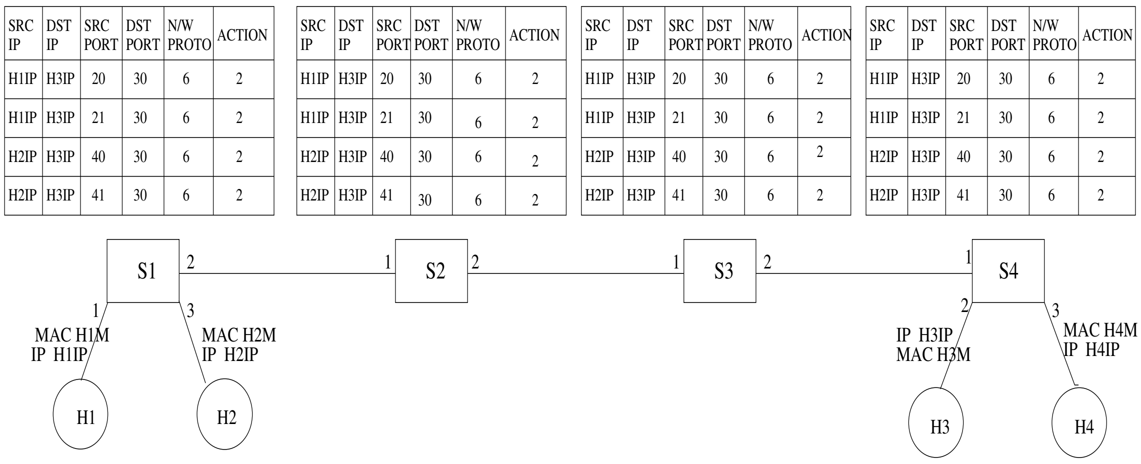
5.2 Flow Partitioner

Highly granular flow rule installations on switches will provide us with the data needed for application-level monitoring. As seen in Table 5.1, TCAM is scarce. By installing more and more highly granular IP flow rules will lead to TCAM exhaustion, leading to packet matching in the software of the switch. This in turn will lead to switch delays. We propose a few approaches in which these highly granular flow rules can be installed in the switches, while still maintaining a bound on TCAM exhaustion limits of each switch. The way we achieve it is by partitioning the flow rules to be installed into $forwarding$ flow rules, which will only forward the packets through interfaces, and *monitoring* flow rules, which will record monitoring statistics. This partitioning of flow rules is named as flow partitioning. By this partitioning, we intend to have lower flow rule count and load balanced flow installation in the switches as explained later in this section.

We use a simple topology with 4 switches, $S1$, $S2$, $S3$ and $S4$ connected in a linear manner shown in Figures 5.2, 5.3, 5.4 and 5.5. Two hosts $H1$ and $H2$ are connected to $S1$ and two hosts $H3$ and $H4$ are connected to $S4$. The blocks shown in Figures 5.2, 5.3, 5.4 and 5.5 are the flow table of each switch. There are 2 TCP connections from $H1$, with source transport ports 20 and 21, to $H3$ with port 30. Also, there are similar TCP connections from $H2$, with source transport port 40 and 41, to $H3$ with port 30. These port numbers are randomly chosen for explanation purposes. The path for these TCP connections is $S1 \rightarrow S4$. Note that $*$ in these figures is used to represent wildcard values. Also note that we will use 5-Tuple $<srcIP, dstIP, srcPort, dstPort, protocol>$ as *monitoring* flow rule from here on.

5.2.1 Naive Approach

This is a basic approach in which we install 5-tuple flow rules on all the path switches between hosts. So basically, we use 5-tuple flow rules both for *forwarding*, as well as *monitoring* purposes. Figure 5.2 explains the flow rules installations incurred as a result of using this approach. Since we are installing 5-tuple flow rules on all path switches, there will be a *packet_in* message for each new TCP connection between a *HostPair*. Hence, between *HostPair* $H1 \rightarrow H3$, two TCP makes way for two 5-tuple flow rules on all path switches. Same goes for *HostPair* $H2 \rightarrow H3$. We can use any flow rule as *monitoring* flow rule. Hence, this approach gives us robustness, but the flow rules installed are redundant in nature.



SRC IP	DST IP	SRC PORT	DST PORT	N/W PROTO	ACTION
H1IP	H3IP	20	30	6	2
H1IP	H3IP	21	30	6	2
H2IP	H3IP	40	30	6	2
H2IP	H3IP	41	30	6	2

SRC IP	DST IP	SRC PORT	DST PORT	N/W PROTO	ACTION
H1IP	H3IP	20	30	6	2
H1IP	H3IP	21	30	6	2
H2IP	H3IP	40	30	6	2
H2IP	H3IP	41	30	6	2

SRC IP	DST IP	SRC PORT	DST PORT	N/W PROTO	ACTION
H1IP	H3IP	20	30	6	2
H1IP	H3IP	21	30	6	2
H2IP	H3IP	40	30	6	2
H2IP	H3IP	41	30	6	2

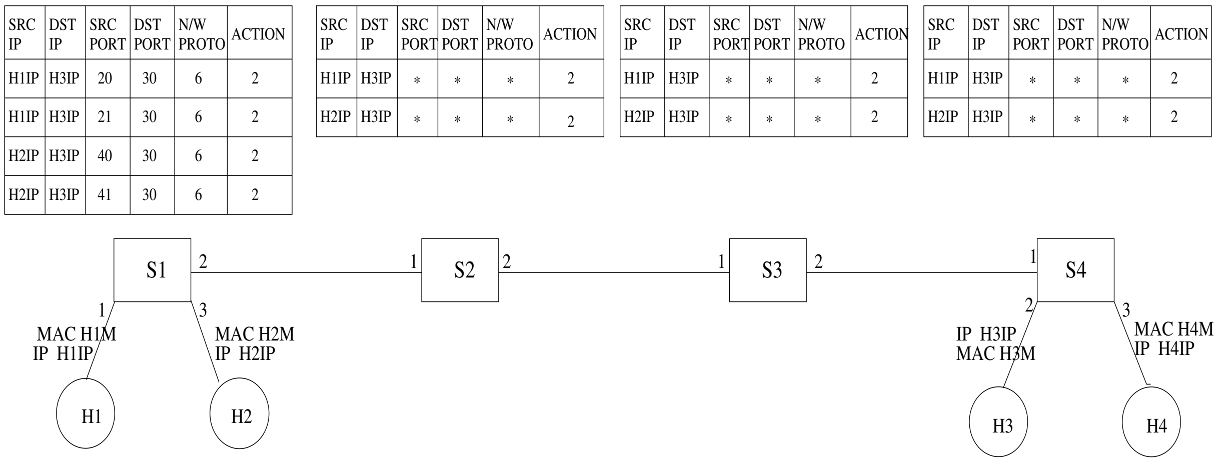
SRC IP	DST IP	SRC PORT	DST PORT	N/W PROTO	ACTION
H1IP	H3IP	20	30	6	2
H1IP	H3IP	21	30	6	2
H2IP	H3IP	40	30	6	2
H2IP	H3IP	41	30	6	2

Figure 5.2: Naive Approach

5.2.2 Ingress Switch Approach

Instead of using *monitoring* flows as *forwarding* flows, we keep them separate in this approach. On the ingress switch, we install 5-Tuple *monitoring* flow rules, while on all the other switches we install *forwarding* flow rules. The switch on which *monitoring* flow rules are installed for a *HostPair*, is known as *AuthoritySwitch*(*AS*) for that *HostPair*. Hence, ingress switch always act as the *AS* for a *HostPair* in this approach.

Figure 5.3 shows the flow rule installations which took place in the same scenario used above. 2-tuple $<srcIP, dstIP>$ *forwarding* flow rules in the path switch provide an aggregate path for all *monitoring* flows of *HostPairs*. As seen, the benefits of using a 2-tuple *forwarding* flow rule is that packets from all TCP connections between a *HostPair* will match the same flow rule in the path switches. As more and more TCP flows arrive, the *monitoring* flows at *AS* will increase. However, *forwarding* rules at on the path switch will remain the same. A potential pitfall of this approach is that the capacity limits of the ingress switch will get exhausted more quickly as more TCP flows come in for a *HostPair*. Hence, TCAM load on an ingress switch gets higher than that of path switches. This can be a significant problem if Wi-Fi access points are connected to a switch where a large number of hosts may connect. Note that a *monitoring* and *forwarding* flow rule can not be on the same switch. If it was not the case, new *monitoring* flows will match *forwarding* flows on *AS* which generating a *Packet_In* event at the controller.



SRC IP	DST IP	SRC PORT	DST PORT	N/W PROTO	ACTION
H1IP	H3IP	20	30	6	2
H1IP	H3IP	21	30	6	2
H2IP	H3IP	40	30	6	2
H2IP	H3IP	41	30	6	2

SRC IP	DST IP	SRC PORT	DST PORT	N/W PROTO	ACTION
H1IP	H3IP	*	*	*	2
H2IP	H3IP	*	*	*	2

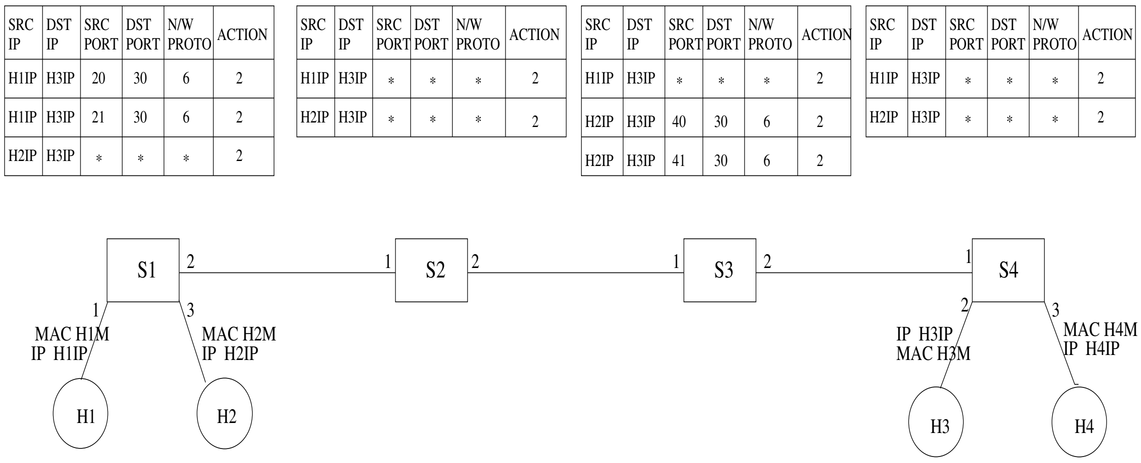
SRC IP	DST IP	SRC PORT	DST PORT	N/W PROTO	ACTION
H1IP	H3IP	*	*	*	2
H2IP	H3IP	*	*	*	2

SRC IP	DST IP	SRC PORT	DST PORT	N/W PROTO	ACTION
H1IP	H3IP	*	*	*	2
H2IP	H3IP	*	*	*	2

Figure 5.3: Ingress Switch Approach

5.2.3 Load Balanced Approach

In this approach, the *ingressSwitch* problem faced in the above subsection is solved by a randomization algorithm. Instead of simply using the ingress switch as *AS*, we use Uniform Hashing Algorithm [31] to calculate the *AS* for a *HostPair*.



SRC IP	DST IP	SRC PORT	DST PORT	N/W PROTO	ACTION
H1IP	H3IP	20	30	6	2
H1IP	H3IP	21	30	6	2
H2IP	H3IP	*	*	*	2

SRC IP	DST IP	SRC PORT	DST PORT	N/W PROTO	ACTION
H1IP	H3IP	*	*	*	2
H2IP	H3IP	*	*	*	2

SRC IP	DST IP	SRC PORT	DST PORT	N/W PROTO	ACTION
H1IP	H3IP	*	*	*	2
H2IP	H3IP	40	30	6	2
H2IP	H3IP	41	30	6	2

SRC IP	DST IP	SRC PORT	DST PORT	N/W PROTO	ACTION
H1IP	H3IP	*	*	*	2
H2IP	H3IP	*	*	*	2

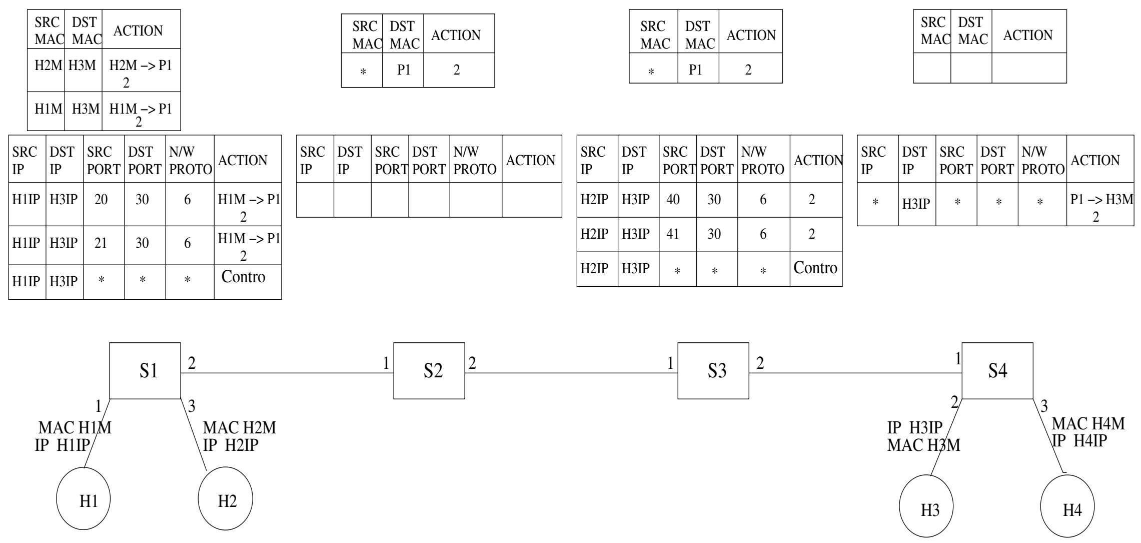
Figure 5.4: Load Balanced Approach

Figure 5.4 shows the idea of *LoadBalancing* Approach. $S1$ acts as *AS* for *HostPair* $H1 \rightarrow H3$, while $S3$ acts as *AS* for *HostPair* $H2 \rightarrow H3$. Hence the *monitoring* load is balanced between switch $S1$ and $S3$. Again, note that *forwarding* and *monitoring* flow rules for the same *HostPair* can't be on the same switch. If that was allowed, then packets would match the *forwarding* rule on *AS* and we won't have any *monitoring* rules. This approach is optimal as we won't have peaks as in the case of *ingressSwitch* Approach. But we are spending TCAM for Ip-based *forwarding* flow rules. For more number of *HostPair*, TCAM rule space at the switches will get exhausted. In addition to that, if a *HostPair* involves a large number of *monitoring* flow rules, then the *AS* for *HostPair* can get exhausted.

5.2.4 LiteFlow Approach

In this approach, we address the drawbacks mentioned in *LoadBalanced* Approach of TCAM rule space exhaustion due to 2-tuple IP based $forwarding$ rules and large number of *monitoring* flows at the AS for a $HostPair$. We use $PseudoMAC$ $forwarding$ explained in section 3.2, instead of 2-tuple $forwarding$. Using *pseudoMAC* approach to $forwarding$ reaps a lot of benefits over *LoadBalanced* approach.

Here, in addition to $forwarding$ and *monitoring* flow rules, we have a third type of 2-tuple flow rule called *controlleraction* flow rule. On a packet match, this flow rule pushes the packet to the controller. Also, we use the concept of *priority* of flow rules in OpenFlow to implement this system. Here *monitoring* > *controlleraction* > $forwarding$ is the priorities assigned to flow rules. So, packet matching will be done first for *monitoring* flow rules, then for *controlleraction* and lastly for $forwarding$ flow rules. $forwarding$ flow rules in this approach provides a backbone path for packets. We are going to leverage this fact for seamless $flowmigration$.



SRC MAC	DST MAC	ACTION
H2M	H3M	H2M –> P1 2
H1M	H3M	H1M –> P1 2

SRC IP	DST IP	SRC PORT	DST PORT	N/W PROTO	ACTION
H1IP	H3IP	20	30	6	H1M –> P1 2
H1IP	H3IP	21	30	6	H1M –> P1 2
H1IP	H3IP	*	*	*	Contro

SRC MAC	DST MAC	ACTION
*	P1	2

SRC IP	DST IP	SRC PORT	DST PORT	N/W PROTO	ACTION

SRC MAC	DST MAC	ACTION
*	P1	2

SRC IP	DST IP	SRC PORT	DST PORT	N/W PROTO	ACTION
H2IP	H3IP	40	30	6	2
H2IP	H3IP	41	30	6	2
H2IP	H3IP	*	*	*	Contro

SRC MAC	DST MAC	ACTION

SRC IP	DST IP	SRC PORT	DST PORT	N/W PROTO	ACTION
*	H3IP	*	*	*	P1 –> H3M 2

Figure 5.5: liteFlow Approach

Figure 5.5 shows the flow rule installation in TCAM and L2 MAC table of the switches. The TCP connections are identical to the approaches mentioned in the previous sections. We use same *pseudoMac* flow rules illustrated in Section 5.2. Similar to *LoadBalanced* approach, we have the concept of AS here too, with AS for $HostPair$ $H1 \rightarrow H3$ being $S1$ and for $H2 \rightarrow H3$ being $S3$. If the AS is ingress switch of the path, note that the action required a change in destination Mac to $P1$ for *monitoring* flow rule.

controlleraction flow rule is required when a new 5-tuple flow arrives at the AS. If there was no *controlleraction* flow rule, the packet would match *pseudoMac* flow rules in L2 MAC tables, and controller would get no event for new 5-tuple connection. Hence, it won't install *monitoring* flow rule in the corresponding switch.

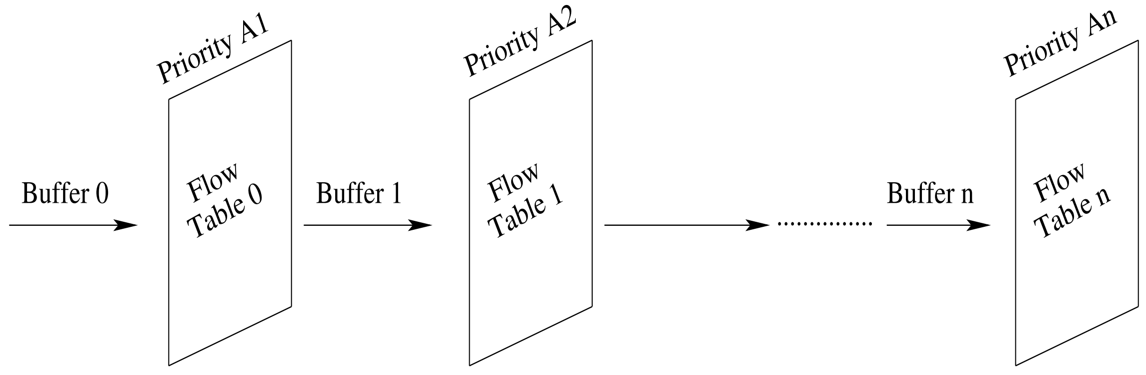


Figure 5.6: Logical Flow Table Architecture

5.3 Benefits of liteFlow Approach

5.3.1 Flow Migration for HostPair

Taking topology from Figure 5.5 as reference, suppose *HostPair* $H1 \rightarrow H3$ has 100 monitoring flows with $S1$ as its AS, *HostPair* $H2 \rightarrow H3$ has 50 monitoring flow rules with $S3$ as its AS, and *HostPair* $H1 \rightarrow H4$ has 20 monitoring flow rules with $S1$ as its AS. The resulting TCAM state of the switches is not load balanced due to the growth of a large number of monitoring flows for some *HostPair*s. To hinder this growth of *monitoring* flows on AS, we introduce a $FlowMigration$ approach which migrates *monitoring* flows from more loaded AS of a *HostPair* to a less loaded switch, which will then act as new AS of *HostPair*. This process can then again recover load balanced flow rule installations property of $liteFlow$.

In our approach, we are very considerate about the buffer state of flow tables in the switches. $FlowMigration$ in *LoadBalanced* approach could have been done simply by deleting *monitoring* flow rules on the AS and adding them on the new AS for a *HostPair*. But this approach would have a considerable delay incurred due to deleting a number of flow rules from AS and adding the same to a different AS. Consider Figure 5.6, in which n flow table are shown with priorities $A1$ to An with $A1 > A2 > ...An$. If we delete a number of *monitoring* flow rules from Flow Table 0, then this will lead to longer buffers at $Buffer0$ due to time it will take to remove them, called the flow removal time. Also, similar time will be taken to add the same number of flows at new AS, called Flow Installation Time.

Considering $liteFlow$ approach, where we have different priorities for *monitoring* and $forwarding$ flow rules, deleting flow rules from old AS and migrating them to new AS will have same transition time as compared to *LoadBalanced* approach. But there is also a problem of correctness of *monitoring* statistics in this approach. Suppose we migrate a *monitoring* flow rule from $FlowTable0$ in the old AS to the new AS. During this interval, there could be traffic which may pass through $forwarding$ flow rule installed in *FlowTable n*. The *monitoring* flow will not be able to record this statistic in any of the AS. Hence, we will lose statistics for this flow. If *monitoring* flow rule was first installed on new AS and then removed from old AS, then there could be overlapping of statistics. This process of installation and removal or vice-versa needs to be done atomically in order to guarantee statistics correctness.

We propose a different migration mechanism in which the old AS will record the statistics of current *monitoring* flow rules, while new AS will record statistics for future *monitoring* flow rules. This migration is called $FlowMigration$.

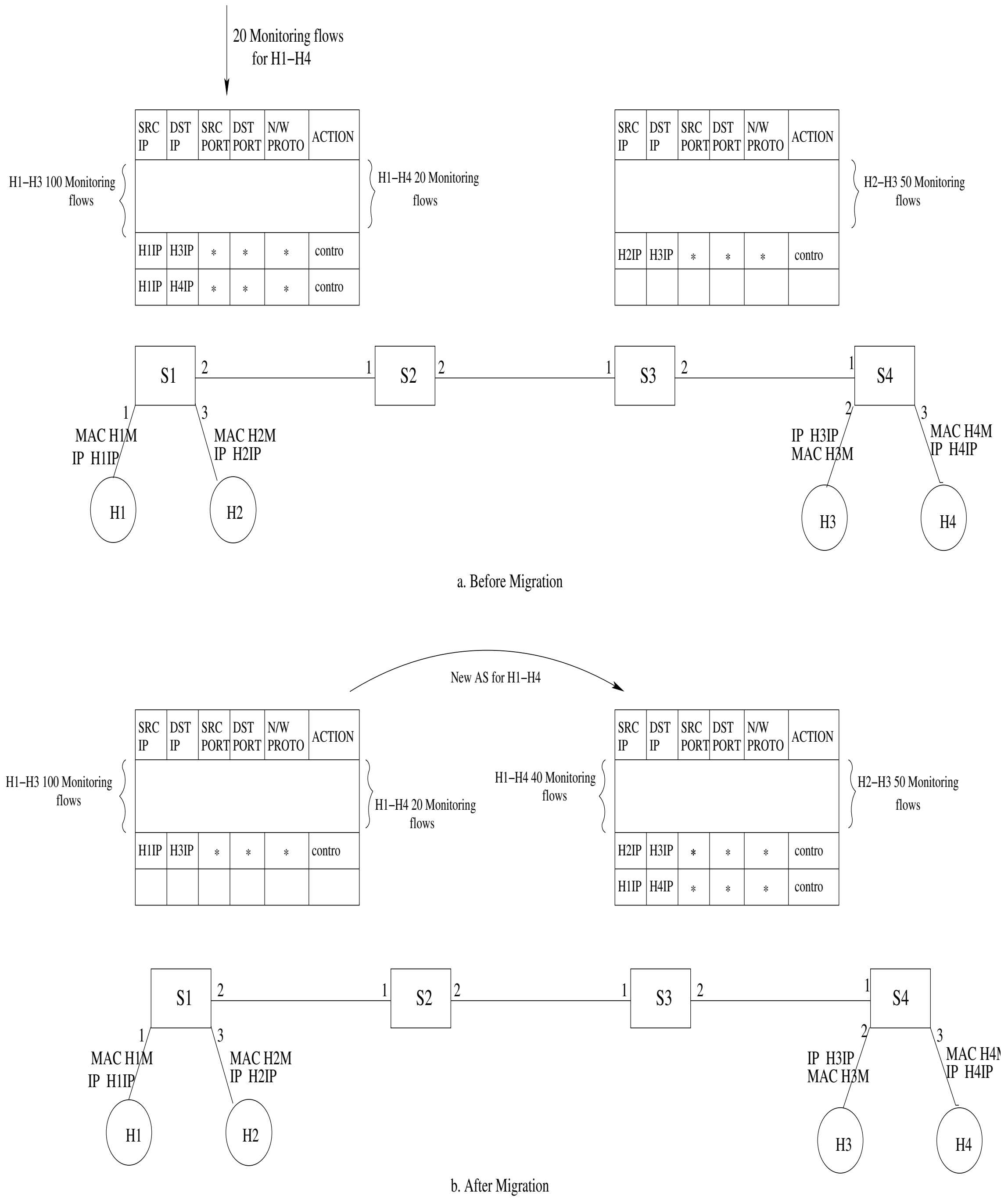


Figure 5.7: Flow Migration Scenario

Figure 5.7 shows the scenario described earlier in this Section. $HostPair\ H1 \rightarrow H3$ has 100 monitoring flows with $S1$ as its AS, $HostPair\ H2 \rightarrow H3$ has 50 monitoring flow rules with $S3$ as its

AS, and *HostPair H*1→*H*4 has 20 monitoring flow rules with *S*1 as its *AS*. At this moment, assume 20 more *monitoring* flows arrive for *HostPair H*1→*H*4 shown in Figure 5.7(a). If migration event is initiated, suppose the new *AS* for *H*1→*H*4 is *S*3. Instead of moving *monitoring* flow rules for *S*1 to *S*3, we just move *controlleraction* flow rule. This will not install any new *monitoring* flow rules at old *AS* as there is no *controlleraction* flow rule installed on it for the concerned *HostPair* and packets will match lower priority *forwarding* rules installed in the switch. While, new *AS* will now have the new *monitoring* flows shown in Figure 5.7(b). By doing this, we won't have the problem of delay incurred by flow deletions as described above. Also statistics loss won't occur as we are not moving current *monitoring* flow rules. In the new *AS*, the old *monitoring* flows will be used for *forwarding* as there is a *controlleraction* flow rule for the concerned *HostPair* now. Hence, figure 5.7(b) shows *S*3 40 *monitoring* flows for *H*1 *H*4 instead of just 20 new flows. In System Design section, we discuss approaches to limit these effects. The installation of *monitoring* flow rules for *forwarding* is the downfall of this approach. This loss can be significant for a *HostPair* with large number of *monitoring* flows as all these will be used for *forwarding* at new *AS* because of migration of *controlleraction* flow rule. A possible solution of this obstacle is by using hard_timeout value in the *monitoring* flow rules. After a hard_timeout, *monitoring* flows will be removed from old *AS*, and will then be installed at new *AS*.

5.3.2 Forwarding in L2 MAC Table

Since we are using L2 MAC table for *forwarding* flows as seen in Figure 5.5, we are saving a considerable amount of TCAM. As Table 5.1 shows, the TCAM capacities are much lower in the switches as compared to L2 MAC tables, using L2 MAC for *forwarding* packets reduces the *TCAM* utilization. Now TCAM is only used for *monitoring* purposes. In the evaluation section, we compare *LoadBalanced* and *liteFlow* on the number of flow rules installed in the TCAM of the switches.

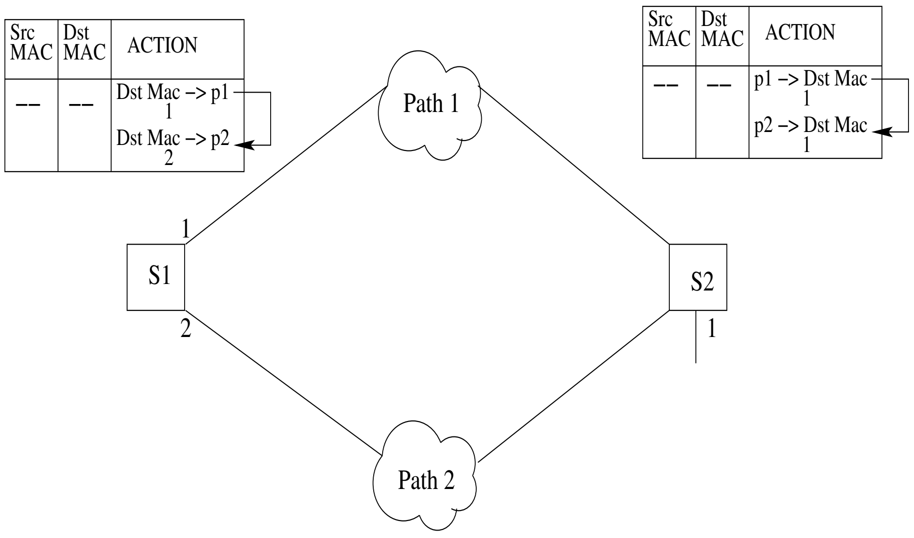


Figure 5.8: Path Change Scenario

5.3.3 Changing Paths due to TCAM Load

Considering topologies, with more than one end to end path, we offer support for changing path between end to end hosts by simply changing the pseudoMac label at the ingress. Figure 5.8 shows the simple change which needs to be done for changing path keeping the rest similar to the earlier scenarios. We do not discuss a path computation problem in this thesis.

5.4 System Design and Implementation

5.4.1 Design

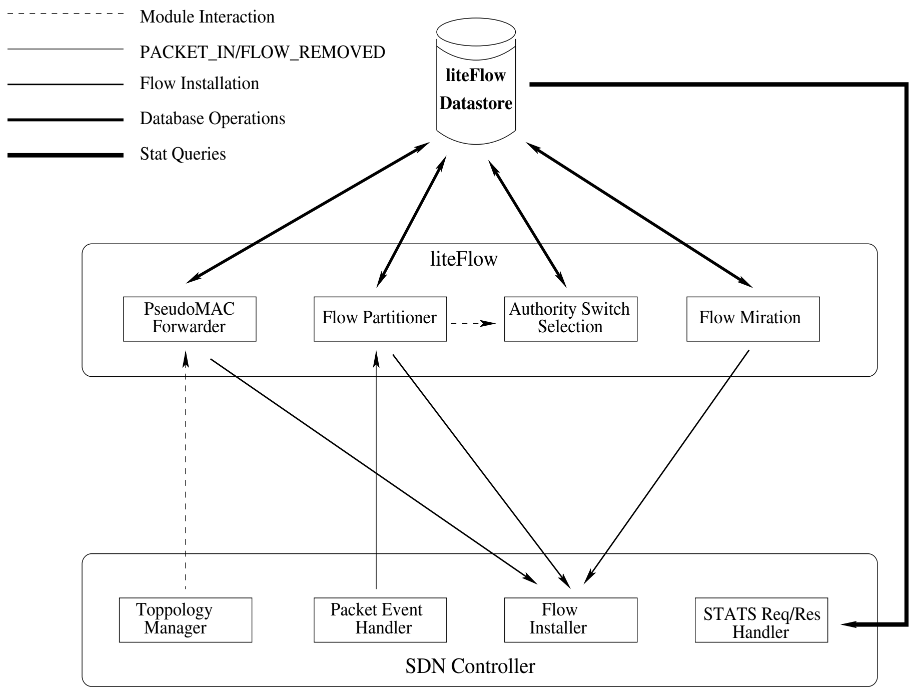


Figure 5.9: System Design of liteFlow

Figure 5.9 shows the system design of *liteFlow*. All Modules can perform its task without interacting with other modules because fo the shared *liteFlow* database.

First we present the various algorithmic explainations of the concepts defined above and various scenarios related to it.

The following are the datastructures used in the implemention of *liteFlow*

- ***Monitoring_Flow :*** 5-Tuple $<srcIP, dstIP, srcPort, dstPort, protocol>$ object
- ***Flow_db :*** is a HashMap where $Monitoring\_Flow$ is used as key and $AuthoritySwitch$ as value
- ***HostPair :*** is an object of $<SrcIP,\ dstIP>$
- ***HostPair_Info :*** is an object of $<HostPair, AS, List < Monitoring\_Flow >>$

- ***HostPair_List :*** is a list all reported *List <HostPair>*
- ***AS_HostPair_List :*** is a HashMap where *AS* is used as key and *HostPair_List* as value
- ***NodePort :*** is an object of *<Switch, Port>*
- ***Path :*** is a list of *List <NodePort>*
- ***Path_MAC_db :*** is a HashMap where *Path* is used as key and *pseudoMac* as value
- ***MAC_Path_db :*** is a HashMap where *pseudoMac* is used as key and *Path* as value
- ***Current_TCAM_Load :*** is a HashMap where *AS* is used as key and *count* of flows installed as value
- ***Capacity_TCAM_Load :*** is a HashMap where *AS* is used as key and *MaxCount* of flows installed as value

5.4.2 PseudoMac Forwarding

Algorithm 1 PseudoMac Forwarding

```
1: procedure PSEUDOMAC_GENERATOR()
2:     PseudoMac ← Select_Random(x2-xx-xx-xx-xx-xx)    ▷ Choose a MAC from a Locally Administered Range
3:     return PseudoMac
4: end procedure
5:
6: procedure PATH_SELECTOR(srcHost, dstHost)
7:     srcSwitch ← get_AttachmentPoint(srcHost)
8:     dstSwitch ← get_AttachmentPoint(dstHost)
9:     Path ← get_FloodLight_Route(srcSwitch,dstSwitch)    ▷ Can be some other routing module
10:    return Path
11: end procedure
12:
13: procedure PSEUDOMAC_FORWARDER(srcHost, dstHost)
14:    Path ← Path_Selector(srcHost, dstHost)
15:    if Path_MAC_db.contains(path) then
16:        return Path_MAC_DB.get(Path)
17:    else
18:        PseudoMac ← PseudoMac_Generator()
19:        Path_MAC_db.put(Path, PseudoMac)
20:        MAC_Path_db.put(PseudoMac, Path)
21:        Install_PseudoMac_Rules(PseudoMac, Path)    ▷ Install Dst Mac based rules
22:        return PseudoMac
23:    end if
24: end procedure
```

Label Generation

Since we are using MAC labels in this approach, it is important that label space is mutually exclusive to MAC addresses used by the hosts. Using one of the set of Locally Administered Address Ranges, such as x2-xx-xx-xx-xx-xx, we have generated mutually exclusive random pseudoMac label.

Algorithm 2 Authority Switch Selection

```
1: procedure SELECT_AS(PATH, SRCHOST, DSTHOST)
2:     $AS \leftarrow Uniform\_Hashing\_Algorithm(Path, srcHost, dstHost)$
3:     if $Current\_TCAM\_Load.get(AS) > Capacity\_TCAM\_Load.get(AS)$ then
4:         return Switch on $Path$ with largest remaining TCAM
5:     else
6:         return $AS$
7:     end if
8: end procedure
```

Path Selection

An administrator can use any algorithm for path discovery in this module. We have used default floodlight controller routing module for path computation at the expense of L2 Mac tables. But we have them in abundance. For example, as future scope we plan to integrate TCAM utilization as a factor in path costs.

PseudoMac Forwarding

As described in Section 3.2, we assign a different pseudoMac label to each path discovered in the topology. And we forward packet based on destination MAC forwarding rules installed by this module.

5.4.3 Authority Switch Selection

Choosing a switch as *AS* randomly is the essence of load balancing monitoring responsibilities. But since this is random in nature, we resort to a second alternative of choosing the max remaining TCAM capacity switch as *AS* when the load on randomly chosen switch is beyond a threshold. *Algoritm* 2 states a simple procedure for *AS* selection.

5.4.4 Flow Migration

Algoritm 3 states a simple procedure for Flow Migration.

Migration Event

We propose an event-based migration algorithm to migrate HostPair to a new *AS*. When we have reached the TCAM capacity limits of the switch, migration event is fired.

HostPair for Migration

In case of migration event, we also need a HostPair who should be migrated to another *AS*. Migrating the HostPair with the least number of *monitoring* flows looks most logical, because there will be *monitoring* flow rule duplicacy on the new *AS* as discussed in Section 3.4.1. Though monitoring stats for old *monitoring* flows will be taken from the old *AS*, but there will be some redundancy on the new *AS* where forwarding old *monitoring* flows will be done by higher granular flows. This redundancy can be solved by using a hard_timeout value in *monitoring* flow rules.

Algorithm 3 Flow Migration

```
procedure IS_MIGRATION_REQUIRED($AS$)
    if then$Current\_TCAM\_Load.get(AS) > Capacity\_TCAM\_Load.get(AS)$
        return $true$
    else
        return $false$
    end if
end procedure

procedure SELECT_HOSTPAIR_FOR_MIGRATION($AS$)
    $Minimum\_HostPair \leftarrow$ null
    $HostPair\_Count\_list \leftarrow AS\_HostPair\_Count\_db.get(AS)$
    while $HostPair\_Count\_list.hasNext()$ do
        $hostPair\_Count \leftarrow HostPair\_Count\_list.getNext()$
        if $HostPair\_Count.get\_Count()$ is Minimum then
            $Minimum\_HostPair \leftarrow HostPair\_Count.get\_HostPair()$
        end if
    end while
    return $Minimum\_HostPair$
end procedure

procedure MIGRATE_FLOWS(AS)
    $Migrate\_HostPair \leftarrow Select\_HostPair\_For\_Migration$(AS)
    $srcHost \leftarrow Migrate\_HostPair.getSrc()$
    $dstHost \leftarrow Migrate\_HostPair.getDst()$
    $New\_AS \leftarrow Select\_AS(srcHost, dstHost)$
    $Uninstall\_Conrtoller\_Action\_Flow(AS, srcHost, dstHost)$
    $Install\_Controller\_Action\_Flow(new\_AS, srcHost, dstHost)$
    return New_AS
end procedure
```

Migrate Flows

Only *controlleraction* flow rule is migrated to new *AS*. Gradully, new *monitoring* flows will be installed on new *AS*. While, old *monitoring* flows stats will be collected from the old *AS*. By migrating *controlleraction* flow rule, the controller can know about new flows of the HostPair, and hence install *monitoring* flows on the new *AS*.

5.4.5 Flow Partitioner

This module waits for PACKET_IN and FLOW_REMOVED events to install/remove flows from the switches or maintain datastructures of *liteFlow* system. *Algorithm* 4 describes the pseudocode used for its implementation.

PACKET_IN Event

On a PACKET_IN event, we install the *monitoring* flow for HostPair just arrived. For sake of simplicity, migration of *monitoring* flows to new *AS* is also fired by this mod ule. Migration event can also be fired independent of *FlowPartitioner*. Datastructure *Flow_db* gives the *monitoring* statistics. Monitoring applications can be implemented using this datastructure.

FLOW_REMOVED Event

FLOW_REMOVED event is used for garbage collection.

Interaction between Topology Manager and PseudoMAC Forwarder

Using SDN controller's Topology manager, whenever there is an event of link addition or deletion, $pseudoMAC$ is initiated. On this initiation of a new path, $PseudoMAC$ forwarder promptly installs destination MAC based flow rules in the network core using $FlowInstaller$ in SDN controller. The $PseudoMAC$ and concerned path are stored in the datastore.

Chapter 6

Evaluation

6.1 Mininet Evaluation of *FlowPartitioner*

Using Mininet [32], we tested *FlowPartitioner* on the topology shown in Figure 6.1. For traffic generation, we have set up Iperf [33] servers on $H21$, $H22$, $H23$ and $H24$ attached to switch $S6$. All the hosts connected to switch $S1$ act as Iperf clients to the four Iperf servers connected to $S6$.

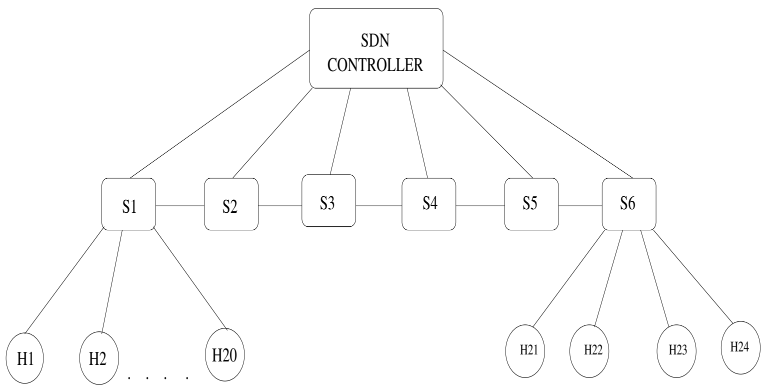


Figure 6.1: Mininet Topology For Evaluation

Setup: We start *iperf* connections from all hosts on $S1$ to all servers on $S6$, which makes up 80 end-to-end *HostPair*s. At a point of time, we start three *iperf* connections from all the hosts on $S1$ towards all *iperf* servers, which makes up 240 total 5-tuple *monitoring* flows from $S1$ to $S6$. Considering the reverse flow as well, we get 80 more end-to-end *HostPair*s, making up 240 more 5-tuple *monitoring* flows from $S6$ to $S1$. This makes up a total of 160 *HostPair*s acknowledging 480 *monitoring* flows in both the directions.
We run the same traffic generation script when all the flows are removed from the switches due to *idle_timeout* set to 10 secs.

- *IngressSwitch* **Approach** : According to theorical calculations, all the path switches, i.e. $S2$ to $S5$, should have a maximum 2-tuple *forwarding* flow rules installations equal to the number of *HostPair*s, equal to 160. $S1$ should act as *AuthoritySwitch* for flows from $S1$ to

$S6$, and also install 2-tuple $forwarding$ flows for the reverse path. So, the maximum flow rules on $S1$ and $S6$ should not exceed 320.

- *LoadBalaced* **Approach** : Theoritical calculations show that on the upper scale, each switch on an average should act as authority switch for 27 hostpairs considering 6 switches and 160 hostpairs. This makes 81 5-tuple *monitoring* flow rules on each switch. Also, each switch will have 2-tuple $forwarding$ rules for rest of the hostpairs, making it 133 in number. Total maximum comes out to be 214 flow rules on each switch. Since it involves randomization approach, we may not accurately get this number, but on a number of experiments, we should get this number of flow rules on each switch.

- *lite$Flow$* **Approach** : Similar to *LoadBalaced* Approach, each switch should act as an $AuthoritySwitch$ to 27 hostpairs, making up 81 5-tuple *monitoring* flow rules on a switch. In addition, each authority switch will have 1 2-tuple *controlleraction* flow rule for each $HostPair$ for which it is acting as $AuthoritySwitch$, hence accounting for 27 2-tuple *controlleraction* flow rules on each switch. Apart from this, the path switches $S2$ to $S5$ should have 2 pseudo mac rules each, switches $S1$ and $S6$ should both have 80 source destination mac based to be stored in L2 MAC table and 20 and 4 destination IP based flow rules respectively to be stored in TCAM. This calculations provides an estimate of maximum 108 flow rules on switches $S2$ to $S5$, while ingress switch $S1$ have 128 and switch $S6$ have 112 maximum flow rules approximately in the TCAM of respective switches.

Our evaluation will be divided into three parts :

6.1.1 Load Balancing Property

We first compare $Ingress$, *LoadBalaced* and *lite$Flow$* Approaches of *FlowPartitioner* on their Load Balancing properties.

- $IngressSwitch$ **Approach** : Figure 6.2 shows the percentage of flow rules installed in $IngressSwitch$ Approach. As seen, the load on switch $S1$ and $S6$ is tremendous as compared to path switches. The black curve is protruding in some cases. These cases are all of the time when flows are being deleted from the switches.

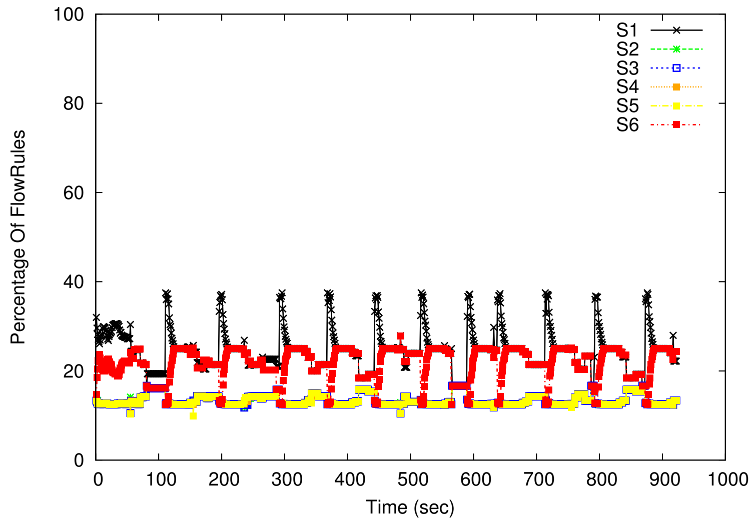


Figure 6.2: Percentage of Flow Rules Installed in Ingress Switch Approach

- *LoadBalanced* **Approach** : Figure 6.3 shows the percentage of flow rules installed in *LoadBalanced* Approach. As seen, all the curves are overlapping providing a well balanced reslt on all the switches. Spikes on blue and yellow curve are all recorded when the flows are removed from the switches. But overall, the load balancing property is achieved in this approach.

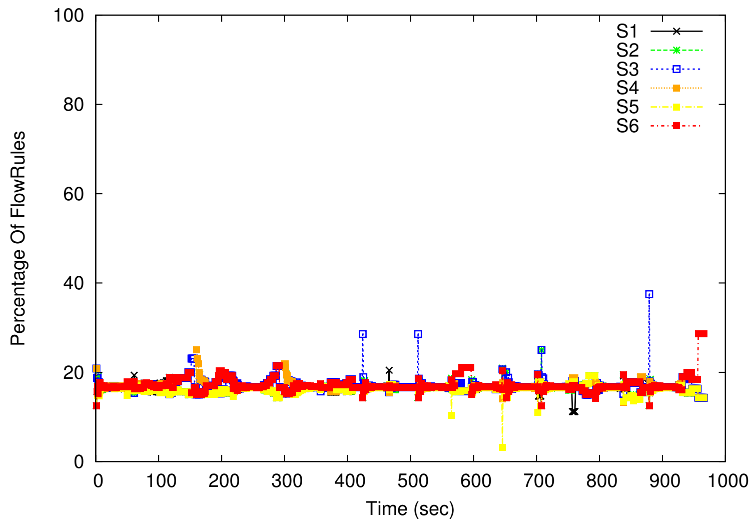


Figure 6.3: Percentage of Flow Rules Installed in LoadBalaced Approach

- *liteFlow* **Approach** : Figure 6.4 shows the percentage of flow rules installed in *liteFlow* Approach. This approach follows the same principle of Load Balancing as the *LoadBalaced* approach. But we see red and black curves dominate the percentage graph because of *PseudoMAC* forwarding TCAM based flow rules in the ingress switch. Though, the number of such flow rules installed is equal to the number of hosts connected to that particular switch. This number will be very insignificant to the actual number of *monitoring* flows each host carry.

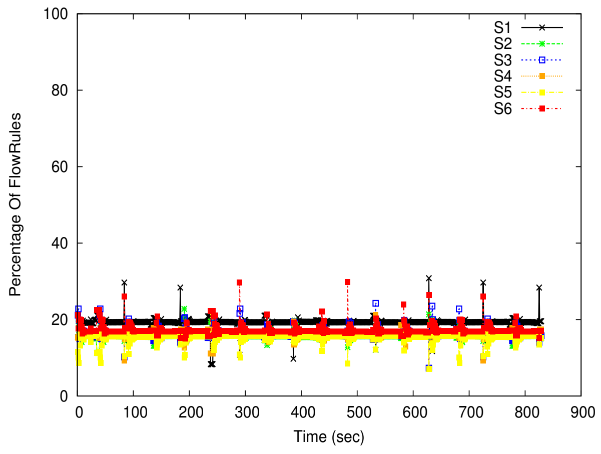


Figure 6.4: Percentage of Flow Rules Installed in liteFlow Approach

6.1.2 Flow Rule Count

In this section, we compare *LoadBalanced* and *liteFlow* approach with respect to their flow rule installations.

- *LoadBalanced* **Approach** : Figure 6.5 shows the number of flow rules installed in *LoadBalanced* Approach. Almost equal flow rule installation is witnessed on each switch.

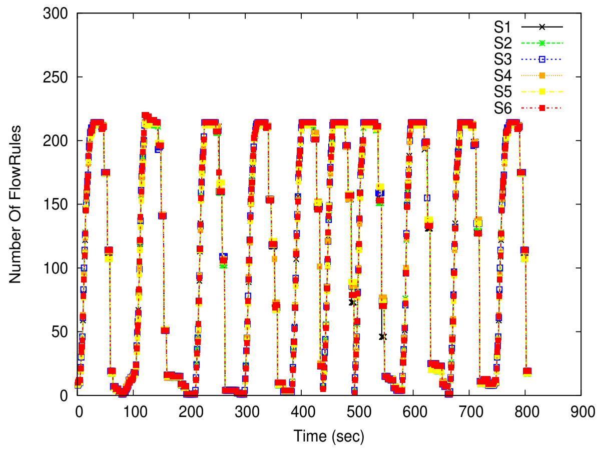


Figure 6.5: Flow Rules Installed in Load Balanced Approach

- *liteFlow* **Approach** : Figure 6.6 shows the number of flow rules installed in *LoadBalanced* Approach. Almost equal flow rule installation is witnessed on each switch except for $S1$ and $S6$. $S1$ is the attached switch to 20 hosts, and there will be 20 more flow rules installed due to *pseudoMAC* forwarding. Similarly on $S6$, 4 more flow rules will be installed.

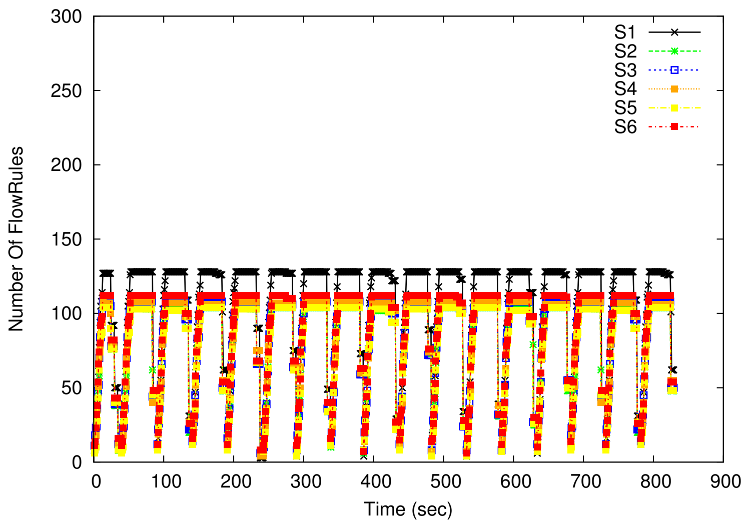


Figure 6.6: Flow Rules Installed in liteFlow Approach

Table 6.1 shows the maximum flow rules installations in the above two approaches, providing an comparison on the number of flow rules installed on each switch. Hence, we conclude that *liteFlow* approach is not just load balanced, but also maintains almost 50% less flow rules as compared to *LoadBalaced* approach.

Table 6.1: Comparing Maximum Flow Rule Installations

SwitchID	**LoadBalanced Approach**	**LiteFlow Approach**
S1	214	128
S2	214	104
S3	214	108
S4	214	108
S5	214	104
S6	214	112

6.1.3 Flow Migration in liteFlow

Flow Migation is useful when the capacity of a switch has exhausted and more *monitoring* flows for a *HostPair* are initiating. Using the same evaluation setup, we highlight the benefit of *FlowMigration* to *liteFlow*.

Table 6.2: TCAM Capacity of Switches

Switch ID	**TCAM Capacity**
S1	120
S2	120
S3	120
S4	120
S5	120
S6	120

Table 6.2 shows the TCAM capacities of each switch. This has been chosen so that *FlowMigration* can take place at switch *S*1, as it has maximum installation limit of 128 discussed previously.

Figures 6.7 and 6.8 are the plots of *liteFlow* without and with migration. As shown, the gap between the black curve and red curve has narrowed down due to migration happening at switch *S*1. We migrate when the capacity limit of a switch has been reached. In actual deployment scenarios, we can relax this condition to an earlier point. Also, we migrate the flows from *HostPair* which has the least *monitoring* flow rules. There can be a discussion on which *HostPair* to choose.

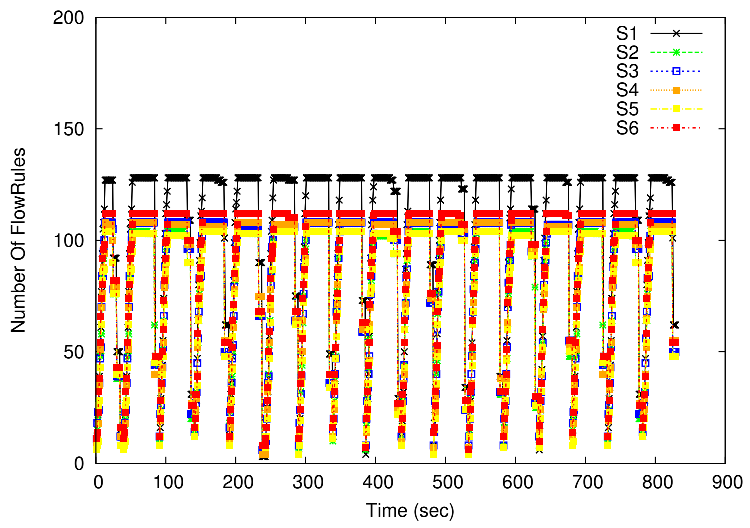


Figure 6.7: Result Plot LiteFlow Approach Without Migration

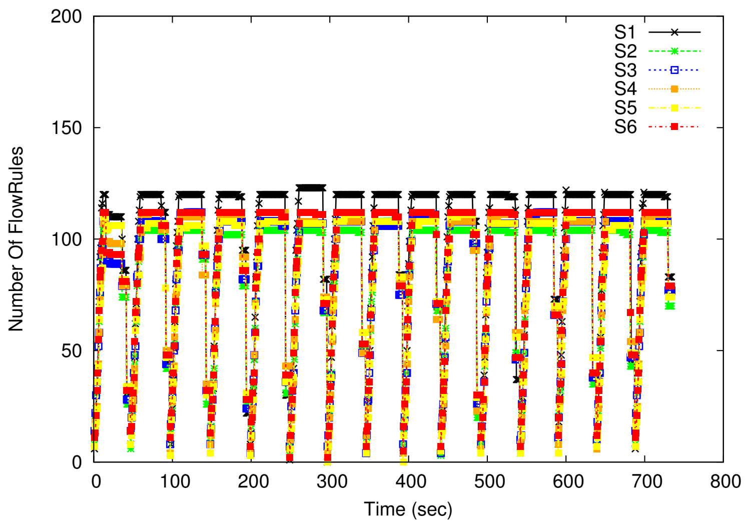


Figure 6.8: Result Plot LiteFlow Approach With Migration

Table 6.3 shows Maximum flow rule installations with/without $FlowMigration$. As seen, the extra load of $monitoring$ flow rules has been migrated to other switches due to $FlowMigration$ on $S1$.

Table 6.3: Comparing Maximum Flow Rule Installations of LiteFlow with/without Migration

SwitchID	**LiteFlow without Migration**	**LiteFlow with Migration**
S1	128	121
S2	104	108
S3	108	112
S4	108	112
S5	104	112
S6	112	112

6.2 Trials in IIT Hyderabad Campus Network

$liteFlow$ is deployed on a trial setup in Indian Institute of Technology Hyderabad. 20 desktop computers and Wi-Fi access point were connected to a production access switch $HP-3800$ supporting OpenFlow as shown in Figure 6.9.

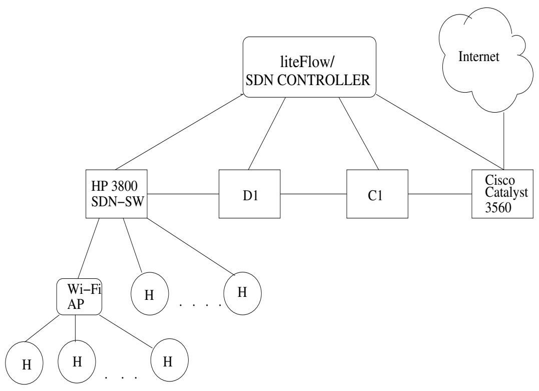


Figure 6.9: SDN Topology

We have set up a distribution layer switch $D1$, and a core SDN switch $C1$, both of which are Open vSwitch [15] switches. Though OVS doesn't have a TCAM associated with it, but for experimental purposes we take varying TCAM sizes of OVS switches into consideration, as in Table 6.4. It shows the TCAM capacities in the number of flow rules that can be installed in the TCAM of the corresponding switch.

Table 6.4: TCAM Capacity of Switches

Switch ID	**TCAM Capacity**
HP 3800	200
D1	300
C1	500

Detailed specifications of the controller and switch used are given in Tables 6.5 and 6.6.

Table 6.5: SDN controller specification for deployment

Model	HP Pavilion g6 Notebook
Operating System	Linux Ubuntu 14.04
CPU	AMD A10-4600M APU
RAM	4GiB
NIC	1 Gbps Ethernet
SDN Controller	floodlight 0.90[16]
OpenFlow Version	1.0

Table 6.6: SDN switch specification for deployment

Switch Model	HP J9573A 3800-24G-Poe+-2SFP+
Firmware Version	KA.15.13.0005

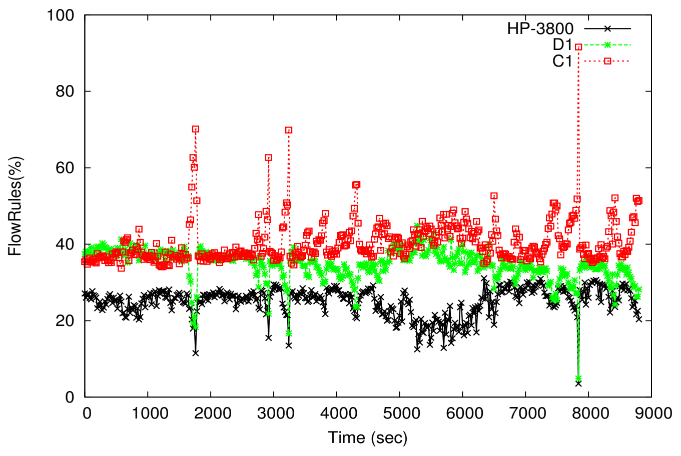


Figure 6.10: Result Plot of *LoadBalanced* on SDN Setup

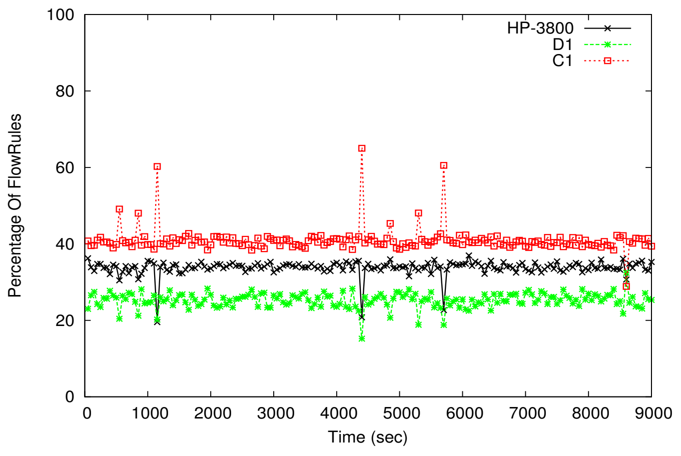


Figure 6.11: Result Plot of *liteFlow* on SDN Setup

LoadBalaced Trial

Figure 6.10 shows the plot of percentage of flow rules installed on SDN switches w.r.t. time duration for which this data was captured. Using TCAM capacities as in Table 6.4, peak flow rule count is shown in Table 6.7. The data presented in Table 6.7 is of the moment when the peak occurred for $HP-3800$. Notice that *LoadBalaced* approach is able to keep the peak flow rule values well under the actual capacities.

liteFlow Trial

Figure 6.11 shows the result plot of *liteFlow* trial on SDN setup. As seen, the load on switch $C1$ is the most due to destination IP based flow rules in *PseudoMAC* forwarding. But the absolute values are very less as compared to *LoadBalaced* approach, as seen in table 6.7. We oberve that TCAM utilization due to 2-tuple forwarding rules, as in the case of *LoadBalaced* approach is very high, which essentially means that a *HostPair* did not have a large number of *monitoring* flows

Table 6.7: Max. Flow Rule Installation for Deployment

Switch ID	**Max. in** *LoadBalaced*	**Max. in** *liteFlow*
HP 3800	187	167
D1	264	138
C1	467	188

between them during the period of observation.

Chapter 7

Conclusion and Future Work

This thesis proposed *liteFlow*, a platform for flow-based monitoring in SDN. Monitoring applications require as much finer detail as possible. For achieving this, the need for fine grained flow rules on the switches arises, which can exhaust the TCAM resources of the switches. *liteFlow* address this issue by carefully installing flow rules in a load-balanced, distributed and non-redundant manner without compromising the accuracy and granularity needed for network monitoring platform. On the basis of current TCAM load, we propose a michanism for shifting load on other switches which are less loaded.

This thesis demonstrated a proof of concept implementation of *liteFlow* in a small test-bed in IIT Hyderabad exhibiting that it significantly reduces the peak number of flow rules installed at each SDN switch.

The following scenarios are interesting to work as future work to extend *liteFlow*.

- **Analysis of switch TCAM capacities on Network Delay:** We develop *liteFlow* on the premise that TCAM exhaustion will lead to network delay. We can develop a prototpe which can benchmark switch on effects of TCAM exhaustion on network loads.

- **Considering Multi-Path Routes:** When there are more than one path between end-hosts, a path of switches which has more remaining TCAM capacity is optimal. This can be achieved by providing TCAM capacities of switches to the routing algorithm in PD module as a metric.

- **Choosing *HostPair* for migration:** In this work, we migrate the flows for thr *HostPair* which had the least number of *monitoring* flows on *AS*. There could be other scenarios to consider for choosing the *HostPair*, for example the *HostPair* with maximum *monitoring* flows, or the *HostPair* with most recent *monitoring* flow. Effects of choosing different policies for choosing *HostPair* for migration can have different consequences on TCAM of the new *AS*.

References

[1] B. Claise, "Cisco systems netflow services export version 9," 2004.

[2] P. Phaal and M. Lavine, "sflow version 5," *URL: http://www. sflow. org/sflow_version_5. txt, July*, 2004.

[3] C. S. Inc., "NETFLOW PERFORMANCE ANALYSIS," tech. rep., Cisco Systems Inc., 2005.

[4] C. Estan, K. Keys, D. Moore, and G. Varghese, "Building a better netflow: technical report," in *ACM SIGCOMM*, vol. 4, 2004.

[5] T. Zseby, "Sampling and filtering techniques for ip packet selection," 2009.

[6] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[7] S. Dabkiewicz, R. van der Pol, and G. van Malenstein, "Openflow network virtualization with flowvisor," 2012.

[8] L. Jose, M. Yu, and J. Rexford, "Online measurement of large traffic aggregates on commodity switches," in *Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services–Hot-ICE. USENIX*, 2011.

[9] J. R. Ballard, I. Rae, and A. Akella, "Extensible and scalable network monitoring using opensafe," *Proc. INM/WREN*, 2010.

[10] R. Wang, D. Butnariu, J. Rexford, *et al.*, "Openflow-based server load balancing gone wild," 2011.

[11] A. K. Nayak, A. Reimers, N. Feamster, and R. Clark, "Resonance: dynamic access control for enterprise networks," in *Proceedings of the 1st ACM workshop on Research on enterprise networking*, pp. 11–18, ACM, 2009.

[12] M. Banikazemi, D. Olshefski, A. Shaikh, J. Tracey, and G. Wang, "Meridian: an sdn platform for cloud network services," *Communications Magazine, IEEE*, vol. 51, no. 2, pp. 120–127, 2013.

[13] B. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, T. Turletti, *et al.*, "A survey of software-defined networking: Past, present, and future of programmable networks," *Communications Surveys & Tutorials, IEEE*, vol. 16, no. 3, pp. 1617–1634, 2014.

[14] N. Feamster, J. Rexford, and E. Zegura, "The road to sdn," *Queue*, vol. 11, no. 12, p. 20, 2013.

[15] "Open vswitch." `http://openvswitch.org/`.

[16] "Floodlight." `https://github.com/floodlight/floodlight`.

[17] "Ryu." `http://osrg.github.io/ryu`.

[18] "Trema." `http://trema.github.io/trema`.

[19] "Nox." `http://www.noxrepo.org/`.

[20] "Flowtables." `https://github.com/floodlight/floodlight`.

[21] B. Stephens, A. Cox, W. Felter, C. Dixon, and J. Carter, "Past: Scalable ethernet for data centers," in *Proceedings of the 8th international conference on Emerging networking experiments and technologies*, pp. 49–60, ACM, 2012.

[22] V. Sekar, M. K. Reiter, and H. Zhang, "Revisiting the case for a minimalist approach for network flow monitoring," in *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pp. 328–341, ACM, 2010.

[23] V. Sekar, M. K. Reiter, W. Willinger, H. Zhang, R. R. Kompella, and D. G. Andersen, "csamp: A system for network-wide flow monitoring.," in *NSDI*, vol. 8, pp. 233–246, 2008.

[24] S. Guha, J. Chandrashekar, N. Taft, and K. Papagiannaki, "How healthy are today's enterprise networks?," in *Proceedings of the 8th ACM SIGCOMM conference on Internet measurement*, pp. 145–150, ACM, 2008.

[25] N. L. van Adrichem, C. Doerr, and F. A. Kuipers, "Opennetmon: Network monitoring in openflow software-defined networks," *IEEE NOMS*, 2014.

[26] M. Yu, L. Jose, and R. Miao, "Software defined traffic measurement with opensketch.," in *NSDI*, vol. 13, pp. 29–42, 2013.

[27] C. Yu, C. Lumezanu, Y. Zhang, V. Singh, G. Jiang, and H. V. Madhyastha, "Flowsense: monitoring network utilization with zero measurement cost," in *Passive and Active Measurement*, pp. 31–41, Springer, 2013.

[28] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "Devoflow: scaling flow management for high-performance networks," in *ACM SIGCOMM Computer Communication Review*, vol. 41, pp. 254–265, ACM, 2011.

[29] M. Yu, J. Rexford, M. J. Freedman, and J. Wang, "Scalable flow-based networking with difane," *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 4, pp. 351–362, 2010.

[30] K. Agarwal, C. Dixon, E. Rozner, and J. Carter, "Shadow macs: Scalable label-switching for commodity ethernet," in *Proceedings of the third workshop on Hot topics in software defined networking*, pp. 157–162, ACM, 2014.

[31] A. Ostlin and R. Pagh, "Uniform hashing in constant time and linear space," in *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pp. 622–628, ACM, 2003.

[32] B. Lantz, B. Heller, and N. 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*, p. 19, ACM, 2010.

[33] C.-H. Hsu and U. Kremer, "Iperf: A framework for automatic construction of performance prediction models," in *Workshop on Profile and Feedback-Directed Compilation (PFDC), Paris, France*, Citeseer, 1998.

[34] Y. Gao, Y. Zhao, R. Schweller, S. Venkataraman, Y. Chen, D. Song, and M.-Y. Kao, "Detecting stealthy spreaders using online outdegree histograms," in *Quality of Service, 2007 Fifteenth IEEE International Workshop on*, pp. 145–153, IEEE, 2007.