2017

Load-Balancing in Local and Metro-Area networks with MPTCP and OpenFlow

Austin Jerome
University of Central Florida

LOAD-BALANCING IN LOCAL AND METRO-AREA NETWORKS WITH MPTCP AND OPENFLOW

by

AUSTIN JEROME
B. Tech, Anna University, 2012

A thesis submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Engineering
in the Department of Electrical and Computer Engineering
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Fall Term
2017

Major Professor: Mostafa Bassiouni, Murat Yuksel

ABSTRACT

In this thesis, a novel load-balancing technique for local or metro-area traffic is proposed in mesh-style topologies. The technique uses Software Defined Networking (SDN) architecture with virtual local area network (VLAN) setups typically seen in a campus or small-to-medium enterprise environment. This was done to provide a possible solution or at least a platform to expand on for the load-balancing dilemma that network administrators face today. The transport layer protocol Multi-Path TCP (MPTCP) coupled with IP aliasing is also used. The trait of MPTCP of forming multiple subflows from sender to receiver depending on the availability of IP addresses at either the sender or receiver helps to divert traffic in the subflows across all available paths. The combination of MPTCP subflows with IP aliasing enables spreading out of the traffic load across greater number of links in the network, and thereby achieving load balancing and better network utilization. The traffic formed of each subflow would be forwarded across the network based on Hamiltonian 'paths' which are created in association with each switch in the topology which are directly connected to hosts. The amount of 'paths' in the topology would also depend on the number of VLANs setup for the hosts in the topology. This segregation would allow for network administrators to monitor network utilization across VLANs and give the ability to balance load across VLANs. We have devised several experiments in Mininet, and the experimentation showed promising results with significantly better throughput and network utilization compared to cases where normal TCP was used to send traffic from source to destination. Our study clearly shows the advantages of using MPTCP for load balancing purposes in SDN type architectures and provides a platform for future research on using VLANs, SDN, and MPTCP for network traffic management.

To my dearest Mom, you are the cornerstone of my life.

To my dearest Dad, you are and will always be my first role model.

ACKNOWLEDGMENTS

I would like to greatly thank my advisor, Dr. Mostafa Bassiouni and especially Dr. Murat Yuksel without whose guidance and support this thesis would not have been completed. I would like to thank my committee members Dr. Cliff Zou and Dr. Yier Jin for their valued comments, suggestions and support.

I also owe my deepest gratitude to Dr. R. Paul Wiegand for allowing me to use resources at the laboratory in the Institute of Simulation and Training building for conducting simulations for the thesis.

I would also like to thank Dr. Syed Hassan Ahmed for his inputs during my defense and for all his suggestions to better improve my thesis. I would like to thank Joshua Bowren for being there to support me during my defense.

To my family back in India; my Mom, my Dad and my brother Aldin, I would not have made it without your constant encouragement and love. To all my extended family and friends here in the United States, thank you for your encouragement and support. Finally, to my late grandmother who passed away, although you might not be here with us physically, your presence and love will remain with me forever.

TABLE OF CONTENTS

LIST OF FIGURES vii

LIST OF TABLES viii

LIST OF ACRONYMS ix

CHAPTER 1: INTRODUCTION 1

1.1 Brief Overview 1

1.2 Literature Background and SDN Overview 2

1.3 Multi-Path TCP in a Nutshell 4

1.4 Problem Statement 6

CHAPTER 2: MPTCP-LIA, OLIA, BALIA AND IP ALIASING 8

2.1 MPTCP-LIA, OLIA, BALIA 8

2.2 IP Aliasing 12

CHAPTER 3: VLAN PATHS WITH IP ALIASING 13

3.1 Conceptual Overview 13

3.2 Topology Setup 13

3.3 Description and Procedure Details 15

3.3.1 Hosts and IP Aliases 15

3.3.2 Ingress Fan-Out 16

3.3.3 Subflow Identifiers 17

3.3.4 Associating Subflows to Hamiltonian Paths 18

3.3.5 VLAN-Specific Hamiltonian Paths 20

3.4 Heuristic for Load-Balancing Path Creation 21

CHAPTER 4: PERFORMANCE COMPARISON AND SIMULATION RESULTS ..29

4.1 Topology 1: Enterprise-Level Local Area Network 31

4.2 Topology 2: Mid-Size Metro-Area Network 34

4.3 Topology 3: Large Metro-Area or Datacenter Network 36

CHAPTER 5: CONCLUSION AND FUTURE WORK 42

5.1 Conclusion 42

5.2 Future Work 43

LIST OF REFERENCES 45

LIST OF FIGURES

Figure 3.1: Sample topology 14

Figure 3.2: Ingress Fan-Out 17

Figure 3.3: Hamiltonian Paths 18

Figure 3.4: Pseudo-Code for Path Selection 19

Figure 3.5: VLAN Specific Hamiltonian Paths 20

Figure 3.6: Pseudo-code for VLAN-Specific Path selection 21

Figure 3.7: Pseudo-code for randomly finding paths 24

Figure 3.8: Sample topology with link numbers 26

Figure 4.1: Topology 1 31

Figure 4.2: Results for Topology 1 32

Figure 4.3: Topology 2 34

Figure 4.4: Results for Topology 2 35

Figure 4.5: Topology 3 37

Figure 4.6: Results for Topology 3 38

Figure 4.7: Aggregate for LOW Load Case 39

Figure 4.8: Aggregate for MEDIUM Load Case 40

Figure 4.9: Aggregate for HIGH Load Cases 40

Figure 4.10: Overall Aggregate for all Cases 41

LIST OF TABLES

Table 3.1: Number of flows per link..... 26

Table 3.2: Number of flows per link after alteration 27

Table 3.3: Number of flows per link after Standard Deviation Enforcement..... 28

LIST OF ACRONYMS

BALIA	Balanced Adaptation Linked Increase Algorithm
IP	Internet Protocol
LIA	Linked Increase Algorithm
MPTCP	Multi-Path Transmission Control Protocol
OLIA	Opportunistic Linked Increase Algorithm
SDN	Software Defined Networks/Networking
VLAN	Virtual Local Area Network
WAN	Wide Area Network

CHAPTER 1: INTRODUCTION

This chapter introduces Software Defined Networking (SDN) and OpenFlow. Moreover, the concepts of IP Aliasing and MPTCP (Multi-Path TCP) are also presented followed by the state of the art recent literature background.

1.1 Brief Overview

Over the years, computer networking has progressed from being just a medium to connect one machine to another to being the backbone of the modern Internet. Today, networking involves critical components like security, quality of service, high availability, confidentiality through concepts such as tunneling, collision detection and avoidance, traffic load balancing, shortest path detection, thus making computer networking to the level of sophistication like never. Having said that, still there exist areas which require improvements to further enhance the networking infrastructure that would benefit the world of technology both productively and economically, load balancing in the network is one such area. Load balancing is the ability to balance traffic across two or more network links in a Wide Area Network (WAN) which are connecting different network entities.

This thesis sheds light to a problem in networking in the area of load balancing in Virtual LANs (VLANs), which are basically broadcast domains that are partitioned and isolated in a network at the data link layer. It gives a possible solution to this problem by using an emerging form of networking called software defined networking (SDN) and Multi-Path TCP (MPTCP). By combining these two concepts, it makes use of additional paths from source to destination other than just the shortest or the one which is already being used.

1.2 Literature Background and SDN Overview

In RFC 7149 [1], M. Boucadair and C. Jacquent explains that software defined networking (SDN) is a set of techniques used to facilitate the design, delivery and operation of network services in a deterministic, dynamic and scalable manner. Further it assumes the introduction of a high level of automation in the overall service delivery and operation procedures. Likewise, RFC 7426 [2] gives an architectural perspective of SDN and offers an understanding of various relevant terminologies. It explains that SDNs are basically a physical separation of the control plane and the forwarding plane. The control plane is the set of networking functions, which bring intelligence to the network device like a switch by telling it where exactly to forward packets. Other than this, it may inform the network device which data or what type of data is to be blocked. The forwarding plane, a.k.a. data plane, is responsible for sending or forwarding traffic to another network device. In other words, the control plane, via software interfaces, defines how to treat the data packets in the network while the data plane executes those functions in the actual networking hardware. The beauty of SDN is that it allows engineers to switch the network preferences and other properties of the routers without physically presence.

Similarly, the benefits of the SDN technology enable networks to be directly programmable due to the decoupling from their forwarding function. It also makes networks more agile as abstracting control from the forwarding plane lets administrators dynamically adjust network-wide traffic flow to meet the changing needs of the network. More importantly, it allows networks to be centrally managed where network intelligence is centralized in SDN controllers, that maintain a global view of the network which appears to applications and policy engines as a single logical switch. This centralized logical view allows networking devices to be programmatically configured where SDNs let network managers configure, manage, secure and optimize network

resources very quickly via dynamic, and automated SDN programs. Another advantage with SDNs is that network devices do not need proprietary software to run which would mean devices being more cost effective. Further, this allows for better economics for service and network providers.

However, SDN still is quite some strides away from maturing in load-balancing. For example, Zhou et al. [3] discuss problems that could exists in the link between the controller and the switch in case of overwhelming traffic. The authors point out that even if we deploy switches and their controllers very carefully, it's difficult for controllers to adapt to the changing traffic load. This could affect resource utilization. More or less, it is essential to balance a load across different controllers in any networking cluster instead of a static network configuration. They propose a dynamic adaptive algorithm for SDN controllers, which is running as a module of SDN controllers and the controllers in distributed environment can cooperate with each other to keep load balancing. The algorithm checks for load on switches in the network and expertly diverts the load to under-loaded controllers thereby achieving load balancing.

The OpenFlow protocol [20] is used as a southbound protocol which is basically from the controller to the switches and vice versa, to inform switches on where packets must be forwarded. Although not of all the SDN designs use it, OpenFlow is a Layer 2 protocol that enhances the very definition of SDN. OpenFlow is responsible for sending and inserting flows inside switches and giving forwarding instructions to switches. Load balancing at the controllers' end is a common obstacle which SDN administrators face. Supporting the argument, Yannan Hu et al. [4], while making use of the OpenFlow protocol for communicating to switches, report on issues with load balancing between SDN controller and switches. They argue that instead of balancing load between controllers by static network planning, controller load balancing should be an indispensable primitive of the distributed control plane. For this purpose, they proposed an

architecture called BalanceFlow for wide-area OpenFlow networks which can partition control traffic load among different controller instances in a more flexible way. The BalanceFlow works at the granularity of flows while distributing them. They introduce a concept of having one 'super controller' in the network whose purpose is to balance flows across other controllers. Both earlier mentioned papers on balancing load between controllers and switches give approaches and solutions to solve the problem with that perspective in mind. However, the problem of balancing load from switch to switch, especially when there are multiple links/paths available to move from source to destination is still an open issue. Therefore, this thesis explores such notion of SDN-based traffic load balancing across various switches simultaneously.

1.3 Multi-Path TCP in a Nutshell

Using commonly known load-balancing protocols to achieve network-efficiency is something that has been persistently looked upon by the research community in the recent years. For instance, Hong et al. [5] used the Equal Cost Multi-Path (ECMP) protocol by developing a system called SWAN which enables inter-datacenter wide area networks to carry significantly more traffic to achieve higher efficiency and utility. However, the problem with ECMP is that it uses hashing techniques to balance load across different links. Here, all links need to get the same percentage of hash values before data can be sent, which means that all paths need to have the same capacity for the protocol to come into effect. Such technique still does not utilize the full potential of having multiple paths from sender to receiver as there could be instances where subflows are not being created as secondary links and the subflows are still being sent on the same path. Further, Ronald van der Pol et al.'s work on multi-pathing with MPTCP and OpenFlow [6] explains this matter. The authors here proposed another method of utilizing multiple paths from

the source to the destination, which involves the use of Multi-Path TCP. It explains that MPTCP does load balancing in the end nodes as part of the TCP process.

On the other hand, RFC-6824 [7] gives details about MPTCP by explaining that it is a protocol which allows for traffic to travel from source to destination through multiple paths by generating separate TCP traffic sub-streams in each link at the source. The simultaneous use of multiple paths for a TCP/IP session could improve resource usage within the network, and thus, improve user experience through higher throughput and improve resilience to the failure. The MPTCP operates at the transport layer and aims to be transparent at both the higher and lower layers. Ronald van der Pol et al.'s paper demonstrated an experiment which consisted of three paths being present from a network in CalTech to Caltech Cern and showed the working of MPTCP over these three paths. Some of these links on the paths had a bandwidth of 10 Gb/sec and there were a few 1 Gb/sec links. At the end of the demonstration, they found that the application could map eight MPTCP subflows on three paths, and concluded that MPTCP should be able to fill these paths with up to 12 Gb/sec of traffic from end-to-end; thus, significantly improving throughput whilst also balancing load across the three paths.

Meanwhile, the Complex systems have also been created by the research community to test how MPTCP could improve efficiency in OpenFlow networks. Sandri et al. [8] created a system called as MultiFlow which uses MPTCP in OpenFlow networks. Their proposal is to improve throughput in shared bottlenecks by forwarding subflows from the same MPTCP connection through multiple paths. They validated the approach in a testbed where shared bottlenecks occur on the links at the endpoints. Their experiment had three distinct paths from source to destination composed of OpenFlow switches. The machines would transfer files of 1 Mb, 10 Mb and 100 Mb. The idea was to compare MPTCP subflows using all separate routes versus

MPTCP subflows making use of all existing paths. The result showed similar throughput for the smaller size at 1Mb but showed significant increase with the files of 10 Mb and 100 Mb. They also conducted an experiment where they compared TCP Reno with MPTCP/MultiFlow with the same files. Here, they found a higher throughput for the instance with MultiFlow and better resilience to failures, where if one link fails, the load is transferred to the other links in the ongoing connection. This shows the high availability property of an MPTCP connection. Although both papers speak about the use of MPTCP in SDNs, where one talks about load balancing and throughput while the other gives more emphasis on throughput, there is no experimentation done on the use of MPTCP on a more local level which would involve VLANs. Apart from this, not much information on load balancing statistics is given comparing different links which contains traffic running on a normal TCP with that running on an MPTCP connection.

1.4 Problem Statement

As discussed above, the use of MPTCP brings an interesting solution for the network load-balancing issue. Distinctions could be made by VLAN administrators into how much traffic is moving across VLANs and how much load links in a topology could take based on subflows forming from sender to receiver through the different paths between the two and accordingly, MPTCP would balance out the load in the paths between each of its subflows. Keeping this in mind, the objectives of this thesis are as follows:

- To look at how MPTCP could be beneficial for network administrators who are responsible for establishing VLANs which make use of SDNs and OpenFlow.
- To give statistics that would give a fair distinction on how much improvements MPTCP brings to load balancing on mesh style networks.

- To develop heuristics which would help direct MPTCP flows across the network in an efficient manner that would further help in improving load balancing and network utilization across the network.

CHAPTER 2: MPTCP – LIA, OLIA, BALIA AND IP ALIASING

In this chapter, we will discuss MPTCP and its workings. We will discuss the congestion control algorithms used within MPTCP which are Linked Increase Algorithm (LIA), Opportunistic Linked Increase Algorithm (OLIA), and Balanced Adaptation Linked Increase Algorithm (BALIA). Then, we will explain IP Aliasing and why it could prove to be useful when used in parallel with MPTCP.

2.1 MPTCP – LIA, OLIA, BALIA

As we have already discussed in Chapter 1, MPTCP gives the ability for multiple paths to form between peers or between senders and receivers in the network. Like TCP, MPTCP is also as reliable when it comes to sending and receiving data. The simultaneous usage of the multiple paths in the network allows for the improvement in resource usage within the network and it also provides for better network throughput and overall improved resilience to network failure.

It is important to note that MPTCP will behave like normal TCP to a non-MPTCP application. Any MPTCP connection would begin just like any regular TCP connection. If there are more than one paths available between the source to the destination, then MPTCP creates additional TCP sessions on these paths. On the sender or destination machines, it would appear as though there is still a single connection between the sender and the receiver when in fact there are multiple TCP streams active on various paths between the two. These additional streams are termed “subflows”.

One relevant question is how does MPTCP identify multiple paths. These paths are identified by the presence of multiple addresses at the hosts and the combinations of these multiple addresses equate to additional paths. For example, if a sender host contains IP addresses A1 and

A2 and the receiver host contains IP addresses B1 and B2. Then, after the first path between addresses A1 and B1 is established as a normal TCP connection, MPTCP comes into effect and generates additional subflows through the additional paths and can have a connection between A1 and B2 in one path or A2 and B1 in the other path. The discovery and formation of additional subflows will be achieved through a path management method where a host can initiate new subflows by using its own additional addresses or by updating its available addresses to the other host. It is important to note that the paths generated by the MPTCP need not be unique and that more than one of them can follow the same route from one point to another, with only the IP address pair of sender-receiver being different in the paths. As stated earlier, the number of paths are dependent on the number of IP address pairs that can be generated from sender's IP addresses and receiver's IP addresses. In this paper, we control the direction of these paths using SDN terminology which is explained in the following chapter. MPTCP also adds connection-level sequence numbers to allow for the reassembly of segments arriving from the multiple subflows with differing network delays. Finally, subflows are terminated as regular TCP connections with a four-way FIN handshake and the MPTCP connection is terminated by a connection level FIN.

With this, new questions arise: How does MPTCP utilize the additional paths existing between sender and receiver? How much data does it send in each of these paths? The solution to these questions lies in congestion control algorithms like Linked Increase Algorithms (LIA), Opportunistic Linked Increase Algorithm (OLIA), and Balanced Adaptation Linked Increase Algorithm (BALIA) which are used during traffic flow across the paths in an MPTCP connection. As per RFC 6356 [17], these new congestion control algorithms are necessary for multipath transport protocols such as Multipath TCP and traditional single path congestion control algorithms (e.g., Additive Increase Multiplicative Decrease) have problems in a multipath context.

For example, one of the biggest problems is that running existing algorithms such as standard TCP independently on each path could give more than its fair share at links traversed by more than one of its subflows. Apart from this, it is important that a source with multiple paths available should transfer more traffic using the least congested of the paths which is a property called “resource pooling” where a bundle of links would effectively behave like one shared link with bigger capacity. It is this feature that would increase the overall efficiency of the network and also its robustness to failure.

LIA couples the additive increase function of the subflows and uses the unmodified TCP behavior in case of a drop. It aims to set the multipath flow’s aggregate bandwidth to be the same as that of regular TCP flow would get on the best path available to the multipath flow. For estimating the bandwidth of regular TCP flows, the multipath flow estimates loss rates and round-trip times, and then, it computes the target rates. It adjusts the overall aggressiveness to achieve the desired rate accordingly. The algorithm ensures bottleneck or link fairness and fairness in the overall network sense. This algorithm achieves performance where a multipath flow would perform at least as well as a single path flow would on the best of the paths available to it. It also manages to achieve to move as much traffic as possible off its congested paths but yet could do better compared to the other following congestion control algorithms. The design of LIA forces a tradeoff between optimal congestion balancing and responsiveness to network dynamics [15]. Hence, to provide good responsiveness, LIA’s implementation must depart from optimal congestion load balancing that leads to the traffic not being optimally sent to the least congested path among the paths available between sender and receiver.

Contrary to LIA, Opportunistic Linked Increase Algorithm’s (OLIA) design is not based on a tradeoff between responsiveness and optimal congestion balancing as it tries to provide both

simultaneously. Just like LIA, OLIA also couples additive increase and uses unmodified TCP behavior in case of a loss, however the difference is in the additive increase term where the term used in OLIA is an adaptation of the increase term in the Kelly and Voice's algorithm [18] which is essential to provide optimal resource pooling. OLIA guarantees responsiveness and non-flappiness by measuring the number of transmitted bytes since the last loss, it reacts to events within the current window and adapts to changes faster. Also, by adapting the window increases as a function of round-trip times, OLIA compensates for different round-trip times. This ensures that OLIA provides better TCP fairness and optimal congestion balancing. However, the problem with OLIA is that it can be unresponsive to changes in the network conditions in some scenarios like when the paths used by a user all have similar round-trip times.

Balanced Adaptation Linked Increase algorithm (BALIA) is another congestion control algorithm which further improves on OLIA by balancing the tradeoff between the properties of TCP friendliness and TCP responsiveness [16]. TCP friendliness is basically how much more throughput an MPTCP flow would get when it shares the network with other TCP flows while TCP responsiveness characterizes how fast the MPTCP algorithm reacts to any network changes. TCP window oscillation is another factor which is considered in BALIA. Window oscillation property characterizes how severely the window size fluctuates around the equilibrium point. It has been proved mathematically that there is always a tradeoff between TCP friendliness and responsiveness, and between responsiveness and window oscillations. Hence, it is not possible to maximize performance of all three parameters. BALIA's design is to allow window oscillation up to an acceptable level in order to improve both friendliness and responsiveness which is achieved by parameterizing these properties and systematically choosing the parameters. On experimentation with the three algorithms [16], it was found that BALIA struck a good balance

between TCP friendliness and responsiveness. In this thesis, we use MPTCP with the BALIA algorithm for congestion control in order to give the fairest possible transmissions across all MPTCP channels.

2.2 IP Aliasing

IP Aliasing is nothing but associating more than one IP address to an interface [10]. Therefore, this allows for one node on a network to have multiple connections where each can serve a different purpose. IP aliasing can be used to provide multiple network addresses on a single interface thereby, this opens the possibility where you can have the computer on two different logical network subnets whilst using a single physical interface.

IP Aliasing allows for MPTCP to create a number of subflows depending on the amount of IP addresses that have been associated with the sender and the receiver's interfaces. These subflows can be directed or forwarded across different paths available from the sender to the receiver using SDN designs where flows in the switch would decide where or which link a packet from a certain IP address would take. This concept is exactly what is used in this thesis to achieve the objective of load balancing the traffic across switches.

CHAPTER 3: VLAN PATHS WITH IP ALIASING

In this chapter, we discuss the conceptual and procedural part of the proposed framework. For clarity, an example topology is considered with explanations such as the number of hosts, VLAN information and the use of IP Aliasing on each of those hosts. Also, 'path' creation is explained and a pseudo-code is included that shows that how packets coming to a switch select a path to be forwarded across the network.

3.1 Conceptual Overview

The main idea is to maximize the potential of MPTCP and improve network utilization by coupling it with IP Aliasing. It is due to IP Aliasing's feature of allowing more than one IP address associated with a machine's network interface that more MPTCP substreams can form based on the addresses available. Then by using the SDN format of forwarding, we can accordingly divert where the subflows are to be forwarded in the network. To achieve this, we create Hamiltonian paths which are associated to each switch connected directly to the hosts in our topology. The paths are created using standard deviation to a find a minimum and maximum range for flows in each link. We explain this concept in more details in the coming subsections.

3.2 Topology Setup

The topology is created using Mininet which is a popular network simulating tool. Mininet can be used for simulating a network with realistic configuration and extracting results from it to understand how performance could be. The SDN controller would be running on a separate Ubuntu Linux virtual machine and would listen for OpenFlow messages from switches that are trying the connect to it. In our case, we used OpenFlow 1.3 and the switches in Mininet, once up, connect to

the SDN controller via the OpenFlow protocol and then be visible in the controller's Graphical User Interface. Further, we arranged a Mesh topology, where each switch has multiple links connecting to one another thus ensuring that multiple paths could exist between any sender and receiver. This is essential in order to understand the potential of MPTCP where different substreams can be placed on different paths between the sender and the receiver thus balancing connection loads across multiple links. The diagram of the first topology is given in Figure 3.1.

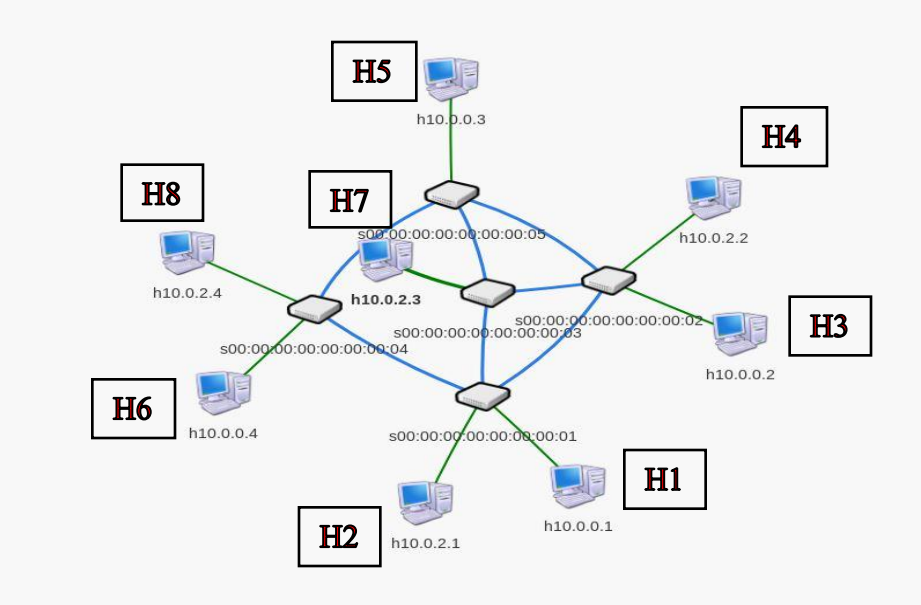


Figure 3.1: Sample topology

In the above topology, H1, H3, H5 and H6 are in VLAN 100 while H4, H2, H7 and H8 are in another VLAN 105. There are five switches in this topology. For terminological purposes, we call the first switch which any host's packet encounters after leaving the host an *ingress* (I) *switch*; and, after leaving the Ingress Switch, all other switches, the packet encounters would be called as *core* (C) *switches*. For example, let us assume that we have a connection from H5 to H1, where a packet traverses through switches with datapath IDs s00:00:00:00:00:00:00:05,

s00:00:00:00:00:00:00:02, s00:00:00:00:00:00:00:03 and s00:00:00:00:00:00:00:01 to reach its destination. Here, s00:00:00:00:00:00:00:05 would be the I switch while s00:00:00:00:00:00:00:02, s00:00:00:00:00:00:00:03 and s00:00:00:00:00:00:00:01 would be C switches. Each link between the C switches has been set to 10 Mb/s maximum bandwidth while each link which connects a host to an I switch has been set to 1000 Mb/s maximum bandwidth. For monitoring the traffic flow, we considered a sFlow network monitoring tool [13]. The Static Entry Pusher REST application [19] is used to push flows to switches which would forward packets from their source to destination. The Static Entry Pusher application as part of the Floodlight Controller [11] and is already activated upon setting up the Floodlight Controller.

3.3 Description and Procedure Details

3.3.1 Hosts and IP Aliases

MPTCP allows for multiple TCP streams called as MPTCP subflows to form between sender to receiver depending on IP availability in both machines. This means that if there is more than one IP address associated with a host, MPTCP starts up another TCP session using the additional IP with the destination machine's IP if the destination machine is also MPTCP capable. Thereby, to make complete use of the traits of MPTCP, IP aliasing is used on each host to associate multiple IP addresses with each host. The amount of IP addresses associated with a host would be dependent on the number of links passing out of the host's I switch to C switches. For example, take H1 from the above diagram, where H1 is connected to s00:00:00:00:00:00:00:01 which is H1's I switch. s00:00:00:00:00:00:00:01 has three links passing on to the C switches, s00:00:00:00:00:00:00:02, s00:00:00:00:00:00:00:03 and s00:00:00:00:00:00:00:04. Thereby, H1 would have two IP alias addresses associated to it and one original IP address giving it a total of

three IP addresses which would be used during the creation of subflows when MPTCP is active in any connection.

3.3.2 Ingress Fan-Out

The subflows created would be forwarded out of the I switch based on flows pushed through the Static Entry pusher. Also, each subflow associated with an IP of the host would be forwarded out through different ports going to the next core switch. For example, as discussed above that H1 would have three IPs, subflows associated with each of the three IPs would be forwarded out through three different ports. This means that if the IP addresses are 10.0.0.1, 10.0.0.11 and 10.0.0.21, then subflows associated with 10.0.0.1 as source address would be forwarded out of the I switch from the port which leads to the C switch s00:00:00:00:00:00:00:02. The subflows associated with 10.0.0.11 as source address would be forwarded out of the I switch from the port which leads to the C switch s00:00:00:00:00:00:00:03 and the subflows associated with 10.0.0.21 as source address would be forwarded out of the I switch from the port which leads to the C switch s00:00:00:00:00:00:00:04. This phase of the procedure is called the 'Fan-Out' phase which is forcing the MPTCP subflows to utilize different links enroute to the destination. Every connection from a host would have a fan-out phase once the packets reach the I switch which would be advantageous in balancing traffic during subflow creation in the MPTCP process. Below in Figure 3.2 is an example diagram showing the Ingress fan-out phase from Host H8 which would be associated with two IP addresses due to two links exiting out of its I switch towards the C switches.

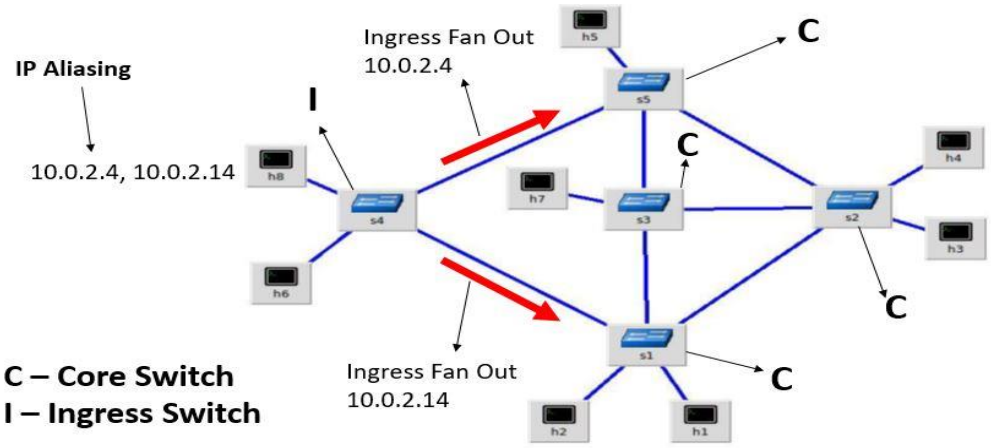


Figure 3.2: Ingress Fan-Out

3.3.3 Subflow Identifiers

Once packets from a host leave the I switch and move towards its destination through C switches they would follow Hamiltonian paths which would be preset for the host and its TCP streams once they reach the C switches. A Hamiltonian path [22] is a path in an undirected or directed graph that visits each vertex exactly once. It is important to note that a graph that contains a Hamiltonian path is named as a traceable graph. Another important property to Hamiltonian graphs is that all Hamiltonian graphs are biconnected, however every biconnected graph need not be Hamiltonian. All topologies that are going to be used in this thesis would satisfy the Hamiltonian graph property. Each host in the topology would have a source address and depending on the number of links going outwards to the C switches from the I switch, the host would have more source addresses associated with it based on IP aliasing. Moreover, due to the fan-out mechanism implemented for each connection going out of the ingress switch, each IP address from hosts connected to the I switch would have a common egress port that takes traffic to the C switches. Using such idea, we can conclude that whenever traffic from an I switch leaves towards

the C switch, the traffic can move out from any one of its outgoing ports which is directly connected to one of the core switches and thereby the identifier element for all host's traffic moving outwards from its I switch is the pair of Datapath ID of the I switch and the outgoing port number associated with one of its source address. Thereby using these flow identifiers, we can create a HashMap associated to all hosts in the topology where the HashMap would consist of the Datapath ID of the switch and an outgoing port which would be the egress port that takes traffic from the host to one of the C switches in the topology.

3.3.4 Associating Subflows to Hamiltonian Paths

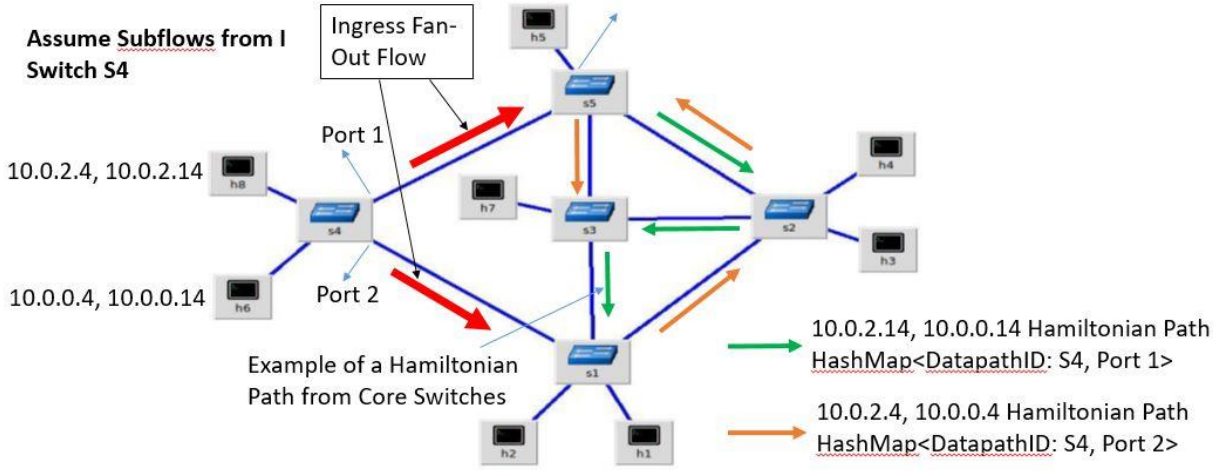


Figure 3.3: Hamiltonian Paths

To systematize the path finding process for each subflow, we associate a Hamiltonian path traversing the C switches to each HashMap associated to hosts connected to an I switch. This means that once a packet belonging to a HashMap associated with a host leaves its I switch towards a C switch, there would be flow identifier entries in that C switch which can be associated to a path for this HashMap. Using this flow identifier entry, the C switch takes the flow's packets

towards other C switches and finally to its destination. The above diagram in Figure 3.3 shows example Hamiltonian paths of host H8 and host H6 connected to I switch S4 and their subflow identifiers: HashMap<Datapath ID: S4, Port 1>, HashMap<Datapath ID: S4, Port 2>.

An important property of all paths in the topology is that each path should be allowed to visit a node, which is a switch in this case, only once. This is done to avoid loops in the topology which is also the main property of Hamiltonian paths. Going by this approach, the number of paths that all hosts connected to any I switch will be the number of flow identifier entries each switch must maintain. This space complexity can be abbreviated into *O(n k)* where '*n*' is the number of I switches in the topology and '*k*' is the number of outgoing ports in an I switch to the C switches in the topology which corresponds to each HashMap having a path associated to it. The pseudo-code for how the path is to be followed by a connection of a host within a core switch in the topology is given below.

```
if packet ∈ (Map< Ingress Switch (I), Outgoing Port (P)>) {
        packet → Path(T);
} else if!(∃ (T  → packet) {
        packet → Controller;
}
```

Figure 3.4: Pseudo-code for Path Selection

According to this pseudo-code, in any switch, there could exist multiple paths for each HashMap of outgoing port and datapath ID of the I switch. The question here is how does the packet reach its destination host once it reaches the switch which contains the connection straight to the host. Here the packet must come out of the path to be directed towards its destination. Priorities for flows within the switch in its flow table is made use off to overcome this. For all switches that contain direct connections to hosts, there are flows set up in switches which match

to the destination host's IP address and these flows have a higher priority compared to the other flows in the flows table. Thus, when the packet first reaches any switch, it checks if there is direct connection to the host from this switch through the destination flow, and if not, it continues the direction that the path flows forward it too.

3.3.5 VLAN-Specific Hamiltonian Paths

The above algorithm can be further improved for better load balancing in the network when the VLAN setups are considered. This is done by creating separate paths for every host belonging to different VLANs which are connected to a I switch. This ensures further segregation of paths which means paths can be spread out in more number of ways. This allows for better load balancing. Apart from this, it also allows for monitoring subflows through links based on VLANs present in the topology which would give network administrators more control in the network. The below diagram shows VLAN specific Hamiltonian paths from I switch S4.

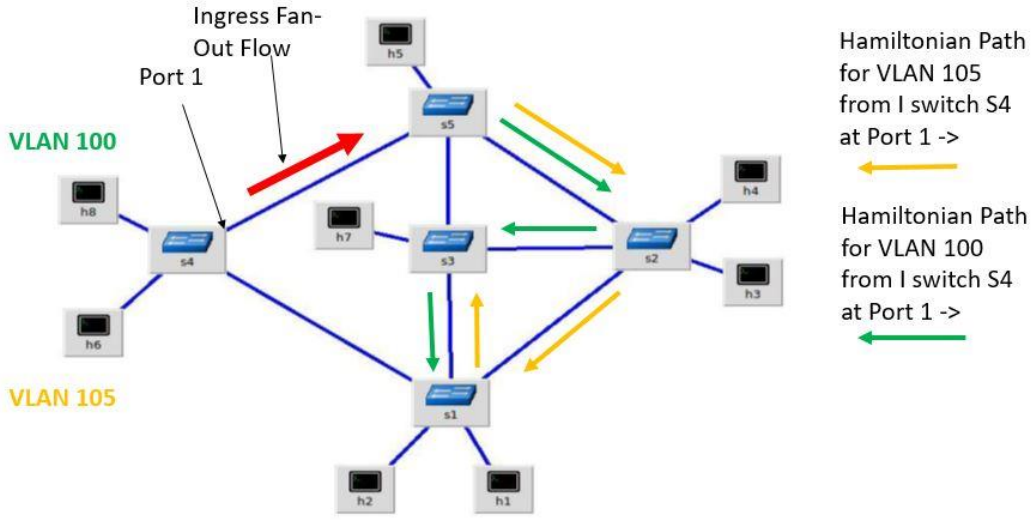


Figure 3.5: VLAN Specific Hamiltonian Paths

Again, we can also compute the number of paths that would be associated to hosts connected to any I switch. Here, the average space complexity of the flow tables in the switches

would be O($V\ n\ k$) where V is the number of VLANs present among all the hosts connected to the I switches, n is the number of I switches and k is the average number of outgoing ports that takes traffic from an I switch to the C switches. The pseudo-code for how the path is to be followed by a connection of a host belonging to a certain VLAN within a C switch in the topology is given below.

```
if packet ∈ VLAN ID (X) {
        if packet ∈ (Map< Ingress Switch (I), Outgoing Port (P)>) {
                packet → Path(T);
        } else if!(∃ (T  → packet) {
                packet → Controller;
        }
}
```

Figure 3.6: Pseudo-code for VLAN-Specific Path selection

If in the condition where there are no paths matching the packets, then the packet must be forwarded to the controller as PACKET-INs where then the controller can make its decision on where to forward the packets. Packets coming back to the switches as PACKET-OUTs from the controller should not follow the paths or HashMap placed on the switches but follow the forwarding rules as set by the controller's forwarding algorithm.

3.4 Heuristic for Load-Balancing Path Creation

With the number of paths that would need to be created for the topology based on the above discussion known, now the question is how the paths are to be created. The most important factor in the creation of paths is to balance out load across each link of the topology that all paths would use. This means that on creating paths, there could be a likelihood that high number of paths utilize one particular link in the topology which could overload that link and create an imbalance. To

avoid this, we create a range (upper bound and lower bound) of flows that a link can have by inspecting the standard deviation of the count of flows traversing a link.

We first create paths randomly for each existing HashMap in the topology using the principle that a switch in the topology can be visited only once and each time a packet leaves its host towards the destination there must be an Ingress Fan-Out phase from the Ingress switch. The below diagram shows the Pseudo-Code for the creation of paths and randomly picking one for each hashmap of I switch and the outgoing port.

```
int[][] topology; //2-D Matrix representing the complete topology
ArrayList<List<Integer>> listOfPaths = new ArrayList<>(); //Stores the list of all paths
List list;
int[] path;
int len; //number of nodes on topology
int row = len;
int col = len;

// Method to loop through neighbours of the Ingress Switch
findPathFromIngress(int[][] graph, int startNode) {
        int tempGraph[][] = topology[row-1][col-1];
        list.add(startNode); //Adding Ingress Node to list
        for(all i in rows) {
                if(i == startNode) {
                        continue;
                }
                for(all j in col) {
                        if(j == startNode) {
                                continue;
                        }
                        tempGraph[i][j] = graph[i][j];
                }
        }
        for(All neighbor in startNode) {
                hamiltonPath(tempGraph, neighbor);
        }
}
```

```
//Method to setup Hamiltonian Path finding
void hamiltonPath(int[][] graph, int startNode) {
	int graphLength = graph.length;
	path = new int[graphLength];
	int colNo = startNode – 1;
	path[0] = startNode;
	findHamiltonPath(graph, colNo, 0);
}

//Recursive Method that uses backtracking approach to find Hamiltonian path
void findHamiltonPath(int[][] graph, int colNo, int pathPos) {
	int graphLength = graph.length;
	for(int i=0; i < graphLength; i++) {
		if(graph[i][colNo] != 0) { //Checks for connected adjacent nodes

			//Check if Node is already present in path
			if(checkDuplicate(path, i + 1)) {
				continue;
			}
			pathPos++; //Increase path by one by adding the next switch
			path[pathPos++] = i + 1;

			//If path length has reached maximum number in topology
			if (pathPos = graphLength – 1) {
				list = Arrays.asList(path);
				listOfPaths.add(list); //Add path to the Final List
				pathPos--;
				continue;
			}

			//Remove the path obtained and search for other paths
			graph[i][colNo] = graph[colNo][i] = 0;
			findHamiltonPath(graph, i, pathLen);

			//Keep backtracking a node from path to find another path
			pathLen--;

			//Transform back to original graph
			graph[i][colNo] = graph[colNo][i] = 1;

		}
	}
	path[pathLen + 1] = 0; //Backtrack the path Array
}
```

```
//Method that detects duplicate Nodes
boolean checkDuplicate(int[] path, int node) {
        for(All i in path) {
                if (i == node) {
                        return true;
                }
        }
        return false;
}

//Method that randomly picks paths from list of paths
List<Integer> returnPath(ArrayList<List<Integer>> listOfPaths) {
        Random ranGen = new Random(); //initialize randomizer
        int index = ranGen.nextInt(listOfPaths.size());
        return listOfPaths.get(index);
}
```

Figure 3.7: Pseudo-code for randomly finding paths

In the pseudo-code above we have five methods, the first method *findPathFromIngress* is mainly iterating through the neighbor switches from the Ingress switch in a loop. Here, from the whole topology which is denoted by a 2-dimensional matrix, we take out the Ingress switch node and find the Hamiltonian path from the neighbors by calling the method *hamiltonPath.* In the *hamiltonPath* method, we set up the parameters necessary to traverse and find the Hamiltonian paths. Then we use the *findHamiltonPath* method which is the core method that is used to find the Hamilton paths. This is a recursive method that uses recursion and backtracking to keep obtaining unique paths. Here the idea is to keep adding new Nodes to the path array until the length is that of the total number of nodes in the topology minus the I switch. Backtracking is then done to keep checking for new paths other than the paths that already has been reported. Paths are checked for uniqueness with the function *checkDuplicate.* This method checks if the path already exists in the path array while going through the path array. If it does, it returns false so that duplicate paths are

not reported again. The paths obtained are added to a ArrayList and then the *returnPath* method is called which is used to pick one of the paths in the ArrayList randomly. The paths which are randomly selected, are used in the topology and per the path, flows are set up in each of the switches.

So, let's take an example using Hamiltonian paths for host H1 which has three paths associated with it which are created based on the ingress fan-out and the point where every node or switch can be visited just once. They are given below.

(10.0.0.1) H1 → S1 → S3 → S2 → S5 → S4

(10.0.0.11) H1 → S1 → S2 → S3 → S5 → S4

(10.0.0.21) H1 → S1 → S4 → S5 → S2 → S3

As we see in these three paths that the I Switch of host H1 is switch S1 which has Datapath ID s00:00:00:00:00:00:00:01, and the IP address 10.0.0.1 is the original IP address of H1 while 10.0.0.11 and 10.0.0.21 are aliased IP addresses created using IP aliasing for the purpose of MPTCP and path creation. Apart from this, we also notice that each node (switch) is visited just once. Now, per these three created paths, we see that the link between S4 and S5 is present as a route taken for all three Hamiltonian paths, thereby, this link has 3 subflows passing through it just based on these paths. The link between S1 and S2 is just being used once which is on the second path, thereby this link has 1 subflow passing through it. In this manner, we can calculate the number of subflows that could potentially be passing through all the links in this topology in case all hosts are active and sending traffic. We create a table showing the number of subflows passing through each link in this topology. Before that, we show the same topology this time with their link numbers in Figure 3.4 and then show the table in Table 1.

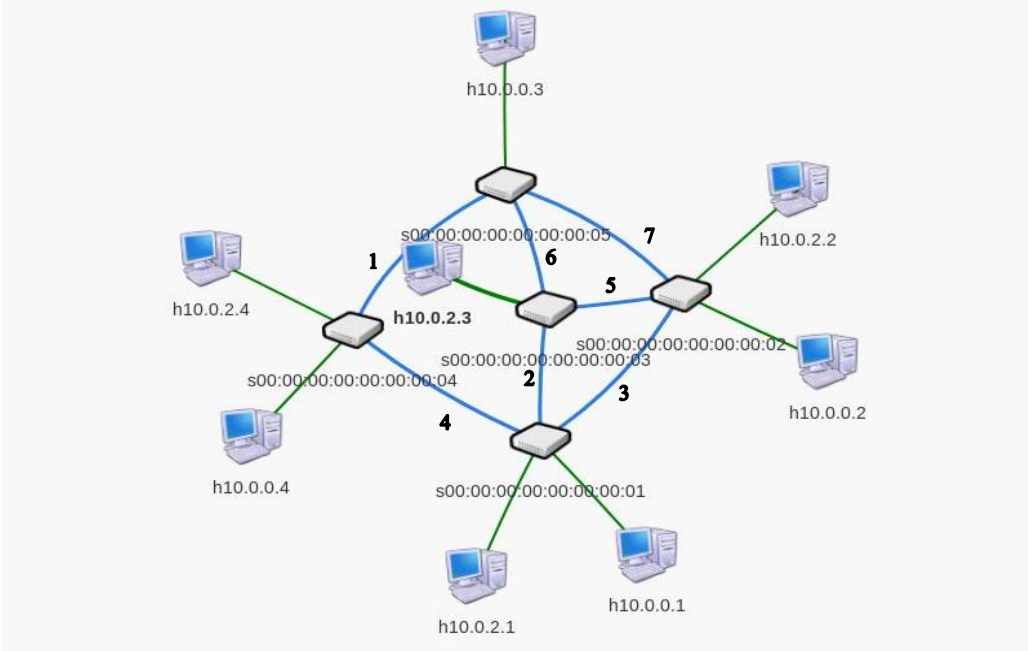


Figure 3.8: Sample topology with link numbers

Table 3.1: Number of flows per link

Link	Number of Flows
1	15
2	10
3	9
4	17
5	10
6	12
7	15

Once this table is created, we observe the number of flows going through each link. If we observe that in any one of the links there is a clear maximum as shown in the above table where the link 4 has a clear maximum number of flows, we try to recreate paths which are going through that link so that it could avoid that link thus reducing the load on that link. While recreating paths, we avoid paths which would be forced to use that link due to the fan-out phase of the process and modify the routes of other paths. We keep selecting the link with the maximum count of flows and

recreating those paths traversing that link until we get to a point where we have at least two links having the same maximum number of flows as shown in Table 2 below for the same topology.

Table 3.2: Number of flows per link after alteration

Link	Number of Flows
1	15
2	10
3	9
4	16
5	10
6	12
7	16

Here in Table 2 above, we notice that links 4 and 7 have the same amount of maximum flows, which is 16, passing through them. It is at this point we calculate the standard deviation to get the upper bound and the lower bound for flows that can pass through each link.

Once the bounds are obtained, we keep recreating paths for each HashMap iteratively until we have met the requirements of each link in the topology having the number of flows corresponding to the lower bound and the upper bound. The range obtained for the above topology after calculation of the standard deviation was 10-15 where 10 is the minimum number of flows that can be pass through a link and 15 is the maximum number of flows that can pass through a link. After recreating paths within this requirement, finally, we come up with the below table which meets the lower and upper bound requirements to balance out the flows across each link in the topology.

Table 3.3: Number of flows per link after Standard Deviation Enforcement

Link	Number of Flows
1	15
2	12
3	10
4	15
5	14
6	10
7	12

In Table 3 above, all the links have their count of flows per the bounds calculated. Once this is calculated and now that the paths are set, we can push flows to the switches in Mininet using Floodlight's Static Flow Pusher. After this is done, the hosts would then communicate with MTPCP using the path flows which are set up in each of the switches.

CHAPTER 4: PERFORMANCE COMPARISON AND SIMULATION RESULTS

In this chapter, we will discuss the simulation results of the proposed idea and we will compare them against regular TCP. We will use the two techniques mentioned above which are VLAN specific paths and paths which do not give consideration for the number of VLANs in the switch. Apart from this, we will also measure performance using standard MPTCP without the use of paths. Thereby, in a nut shell we would be comparing four techniques:

1. MPTCP with VLAN-Specific Hamiltonian paths in place in the switches [VS]
2. MPTCP with paths without considering the number of unique VLAN hosts which are directly connected to the switch [NVS]
3. MPTCP without any paths incorporated [MPTCP]
4. Regular TCP

Regular TCP will be the benchmark for our experiments and we will accordingly compare the other three techniques to see the amount of percentage increase or decrease in performance with respect to the standard TCP.

Tests will be done in three different mesh like topologies involving client-server communication between each host in the topology using the *iperf* tool [21] in Linux which is a performance tool that is used for measuring the maximum achievable bandwidth on IP networks. Standard MPTCP and TCP cases will use the Floodlight's forwarding module to be directed from source to destination. The forwarding module uses a reactive SDN approach where when a switch sees an unknown packet for the first time, it forwards it to the SDN controller as a PACKET-IN message. Then, the controller calculates the shortest path to the destination and sends that

information back to the switch as a PACKET-OUT message. The switch, finally, receives this forwarding information and forwards the packet towards the destination accordingly.

Further, the test cases will be further divided into three categories (i.e., LOW, MEDIUM, and HIGH) based on the amount of traffic load in the topology. A LOW loaded case means that there are few client-server communications taking place in the topology which means, for example in a topology consisting of eight hosts, there would be between two to four client-server communications in a LOW load case and apart from this, there would not be any parallel sessions on going where one server can serve multiple clients. So, in a LOW loaded case, each server will be allowed to serve just one client, and one client is only allowed to talk to one server. A MEDIUM loaded case means that servers can serve multiple clients simultaneously and clients are allowed to connect multiple different servers which increases the number of TCP sessions going across the topology, thereby increasing the load on the topology. However, depending on the number of hosts in the topology, a limited number of machines will be communicating with each other. A HIGH loaded case is when all hosts in a topology are communicating with each other, either as a client or as a server, thus maximizing the traffic load in the topology. Further information will be given about the loads using the topology examples. Each *iperf* session in any case between client and server will be run for two minutes to get a fair estimation of the overall bandwidth. For each case, there will be multiple separate simulation runs done and the average of will be taken as the performance measure for that case. Confidence Intervals will also be calculated for each case where the confidence percentage would be 90%.

4.1 Topology 1: Enterprise-Level Local Area Network

The diagram in Figure 4.1 below is the first topology we are going to use for our experiments, which is comparable to an Enterprise-Level Local Area Network. It is similar to the one in Fig 3.1.

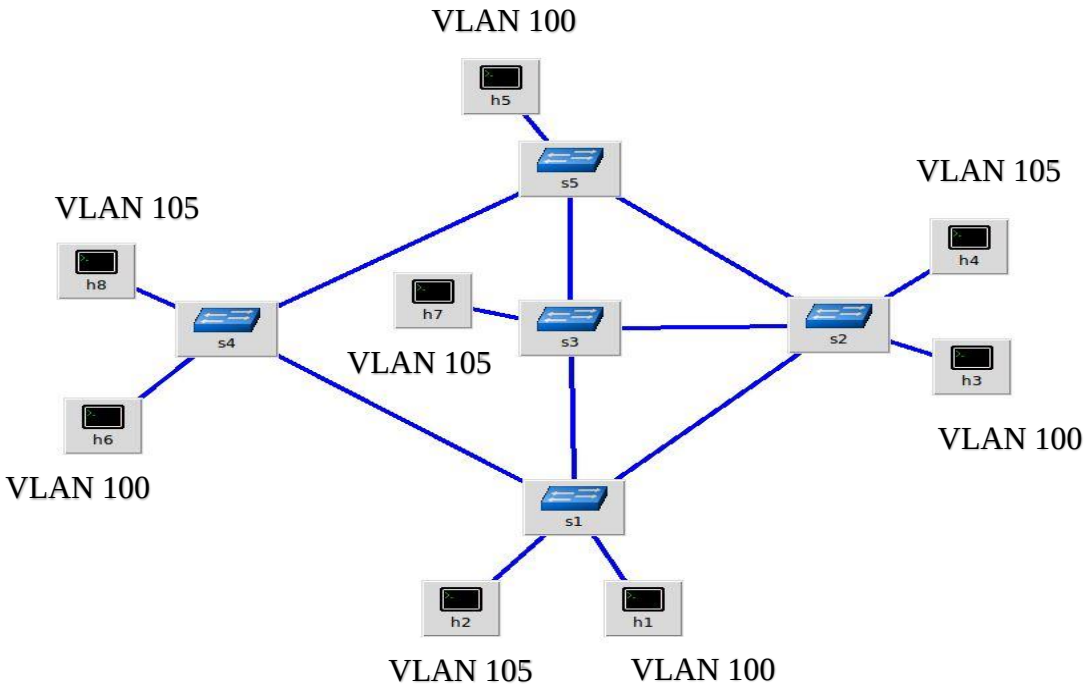
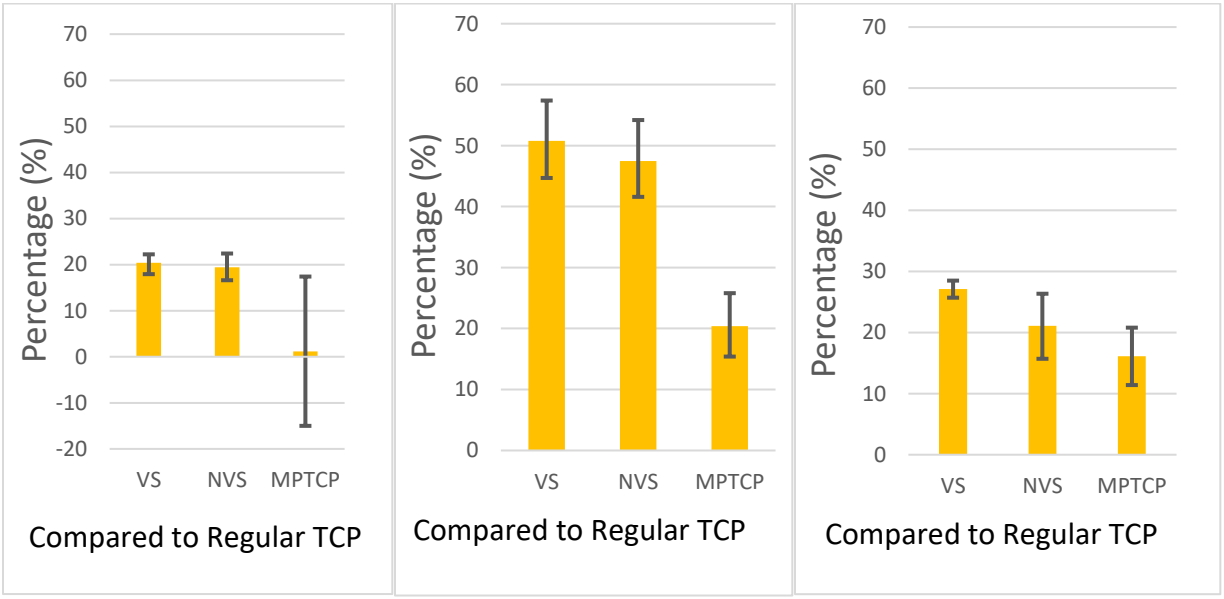


Figure 4.1: Topology 1

Here, there are a total of eight hosts which are placed under two VLANs, i.e., VLAN 100 and VLAN 105. There is a total of five switches and seven links which connects them to each other. These links are all 10 Mb/s links. The links that connects the hosts to the switches are 1000 Mb/s links. For this topology, a LOW loaded case is when there is a single host, which acts like a client, is connected to another host, which acts like a server, and both these hosts will not communicate with other machines or take connections from other machines as long as there is traffic flow between the two machines. An example LOW loaded case would be when H3-H6, H4-H8, H1-H5 and H2-H7 have each established a client-server connection where H3, H4, H1 and H2 are

clients and H6, H8, H5 and H7 are their respective servers and are running simultaneously. In the MEDIUM loaded case, we can have a server serving multiple clients simultaneously, thereby, taking the topology above if H1 is the server for client H3. It can simultaneously be the server for client H5 and H6 as well. H3, H5 or H6 can also act as servers to other machines. Apart from this, since there are 8 hosts with 4 in each VLAN for this topology and the possibility of having 16 different combination here of client-server communications, we consider a medium load case to have about 6 to 8 client server communications going on simultaneously for this case. Finally, let's take the high load case where here, again we can have a server serving multiple clients and a client connecting to multiple servers. Apart from this, for a high loaded case we can have 12-16 client server connections running simultaneously for this topology. After running the tests for all the cases, we see the following results.



(a) LOW load (b) MEDIUM load (c) HIGH load

Figure 4.2: Results for Topology 1

The above diagram shows the results where we see that MPTCP with VLAN specific Hamiltonian paths (VS) has the best performance with over 20% increase on average in throughput compared to regular TCP for the LOW load case shown in Figure 4.2(a). MPTCP without VLAN paths(NVS) has the second best with just under 20% increase on average compared to regular TCP and the standard MPTCP gives about over 1 percent increase on average compared to regular TCP. In terms of the confidence interval, which is denoted by the dark bolded line with the caps on the ends in the diagram, for a 90% confidence percentage we get a result of 18.5% to 22.8% increase for VS and for NVS we get about 16.6% to 22.4% increase, while for standard MPTCP we get 15% decrease from regular TCP to 17.4% increase from regular TCP.

For the MEDIUM load case shown in Figure 4.2(b), we observe VS with the best throughput with over 50% increase on average in throughput compared to regular TCP with NVS having over 47% percent increase on average and the standard MPTCP having about 20% average increase. In terms of confidence interval, we get 44.7% to 57.4% increase for VS, NVS gives 41.6% to 54.2% increase and for standard MPTCP we get 15% to 25.4% increase from regular TCP.

And finally, for the HIGH load case shown in Figure 4.2(c), we see over 27% average increase in VS, over 21% average increase in NVS and over 16% increase in average in standard MPTCP compared to regular TCP. While in terms of confidence interval, we get 25.7% to 28.5% increase for VS, NVS gives 15.7% to 26.35% increase and for standard MPTCP we get 11.4% to 20.8% increase from regular TCP. Thereby, overall, we clearly see that Multipath TCP with VLAN specific paths show a significant improvement from regular TCP. This is also evident with Multipath TCP without VLAN specific paths and standard MPTCP although not as much.

4.2 Topology 2: Mid-Size Metro-Area Network

Now let's perform the same tests on a bigger topology which is comparable to a Mid-Size Metro-Area Network. The below diagram in Figure 4.3 is the second topology which has eight switches and ten hosts. It has twelve links which connect the switches to each other which are all at 10 Mb/s. Again, like the earlier topology, the links connecting the hosts to the switches are running at speeds of 1000 Mb/s. In this topology, there are hosts which contain three different VLAN IDs which are 100, 105 and 110. Here again we test for LOW, MEDIUM and HIGH load cases.

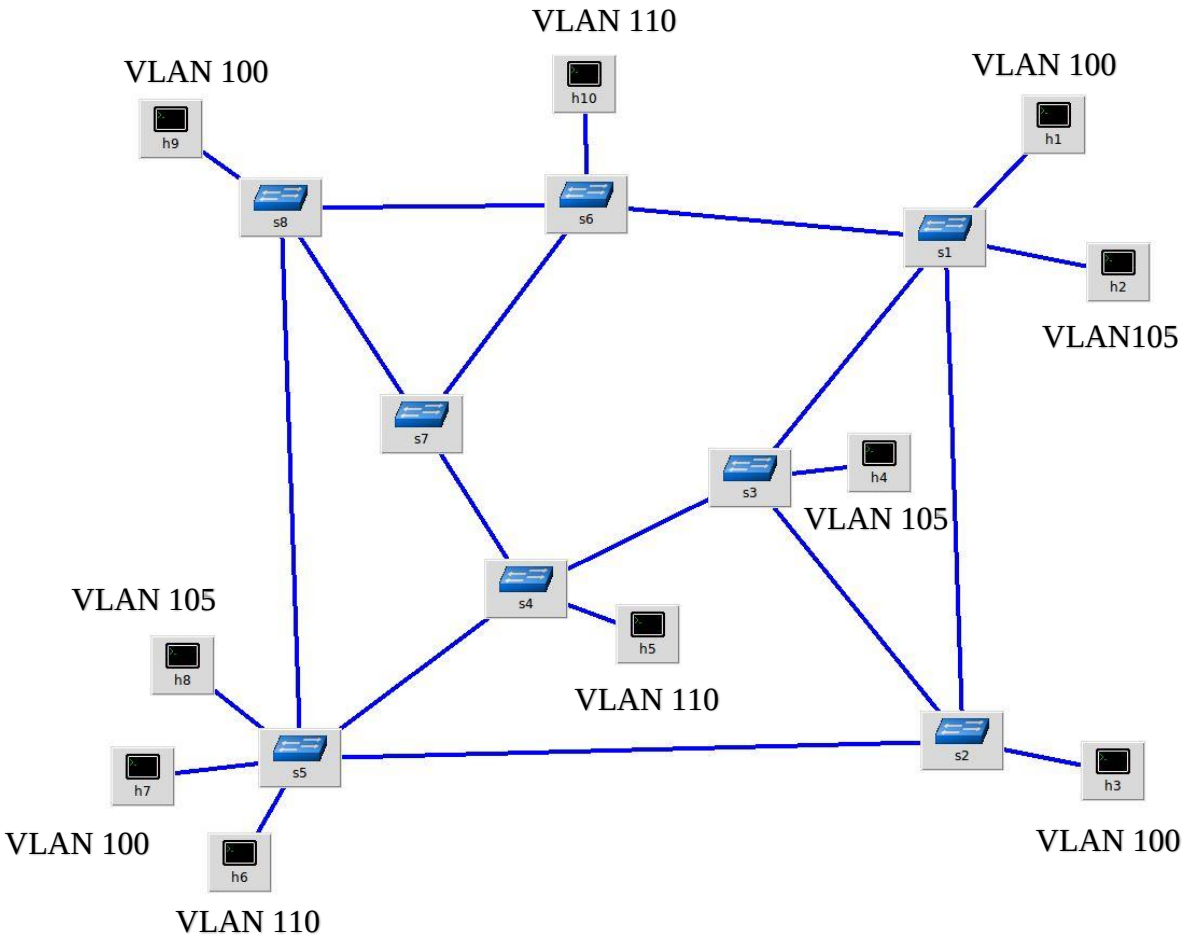
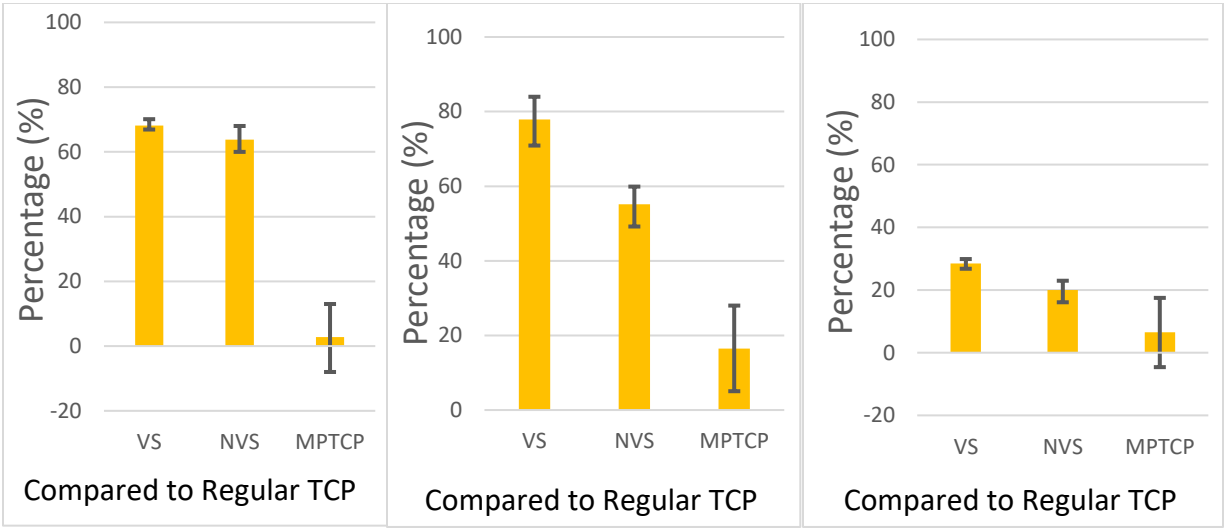


Figure 4.3: Topology 2

To reiterate, LOW load cases will not have simultaneous connections on servers from clients and clients will not connect to multiple servers. MEDIUM load cases on the contrary will have such client-server connections. In this topology of 10 hosts, 6 to 10 client-server connections are considered a MEDIUM load case while, 12 and above connections is considered a HIGH load case. The below diagram shows the results of the tests performed.



(a) LOW load (b) MEDIUM load (c) HIGH load

Figure 4.4: Results for Topology 2

After running the tests for the LOW loaded case, we see the results in Figure 4.4(a). Here we see close to 70% average increase in VS while NVS isn't far off with about 64% average increase compared to regular TCP and standard MPTCP we see close to 3% average increase. In terms of the confidence interval, which is again denoted by the dark bolded line with the caps on the ends in the diagram, again with 90% confidence, we get 66.9% to 70.1% increase in VS, for NVS we get 60% to 68% increase compared to regular TCP and standard MPTCP gives an 8% decrease to a 13% increase compared to regular TCP. The results are similar in the MEDIUM load case as shown in Figure 4.4(b). However, we see an even greater increase in the VS. VS obtained

close to 78% average improvement compared to regular TCP while NVS showed just 55% average improvement, and finally standard MPTCP obtained over 16% increase in performance. In terms of confidence interval, we get 70.9% increase to 83.98% increase in VS compared to regular TCP, we get 49.2% to 59.9% increase in NVS compared to regular TCP and about 5.05% to 28% increase in standard MPTCP compared to regular TCP. Finally taking the HIGH load case shown in Figure 4.4(c), we again see similar trends. Here, VS obtains over 28% average increase, NVS obtains about 20% average increase and finally standard MPTCP obtains about 6.5% average increase compared to regular TCP. In terms of confidence interval, we get 26.8% increase to 29.9% increase in VS, we get 16.1% to 22.98% increase in NVS compared to regular TCP and about 4.6% decrease to 17.5% increase in standard MPTCP compared to regular TCP.

4.3 Topology 3: Large Metro-Area or Datacenter Network

Let's now take an even bigger topology which is comparable to a Large Metro-Area or a Datacenter Network. In this topology given below in Figure 4.5, there are 15 switches and 8 hosts. There are 28 links connecting the switches together and each link has a maximum bandwidth of 10 Mb/s. Apart from this again, all links connecting hosts to switches are 1000 Mb/s links. There are two unique VLANs among the hosts in this topology which are VLAN 100 and VLAN 105. Here too, tests would be done for LOW load, HIGH load and MEDIUM load where again LOW load cases will not have simultaneous connection on servers from clients and clients will not connect to multiple servers.

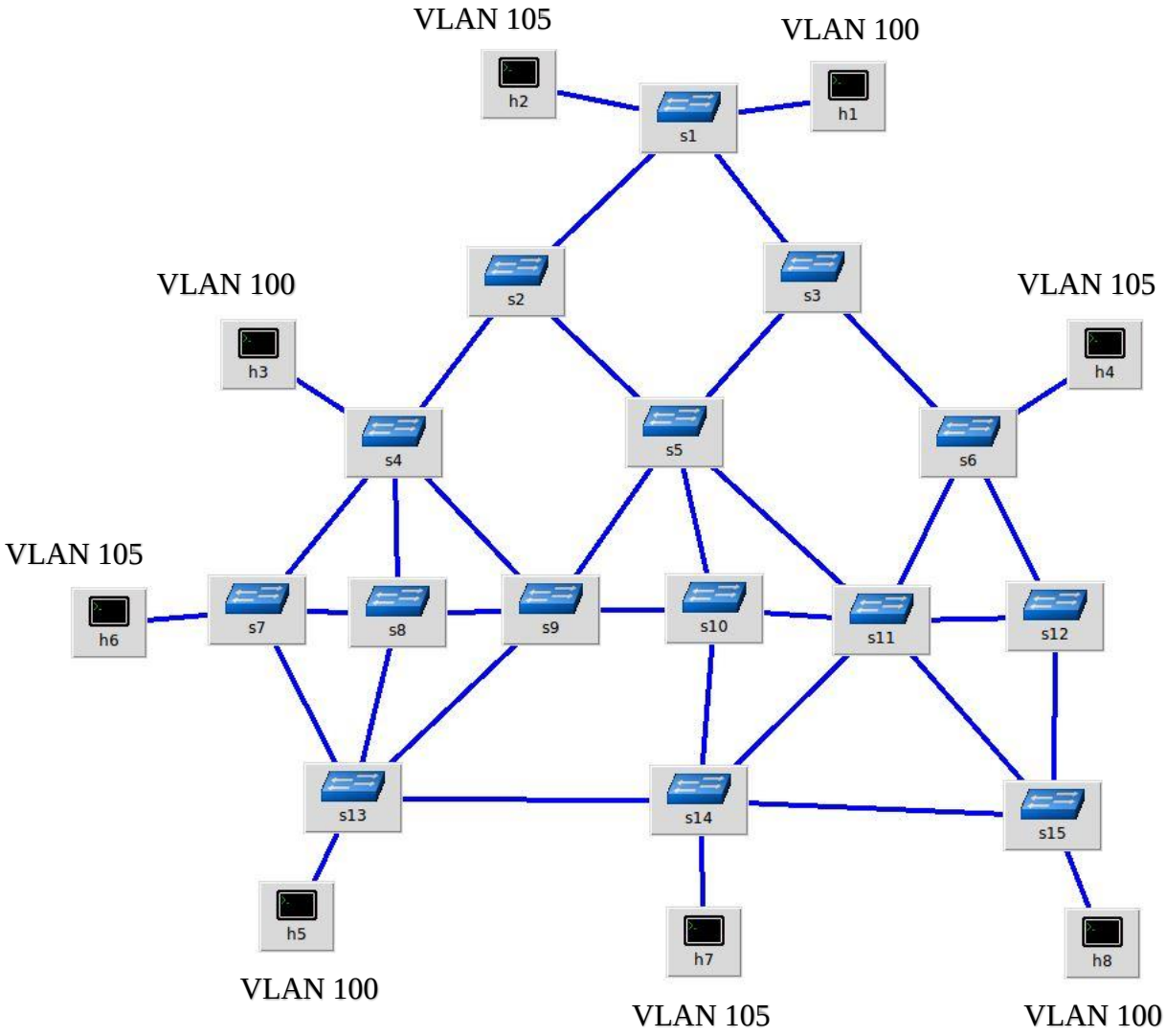
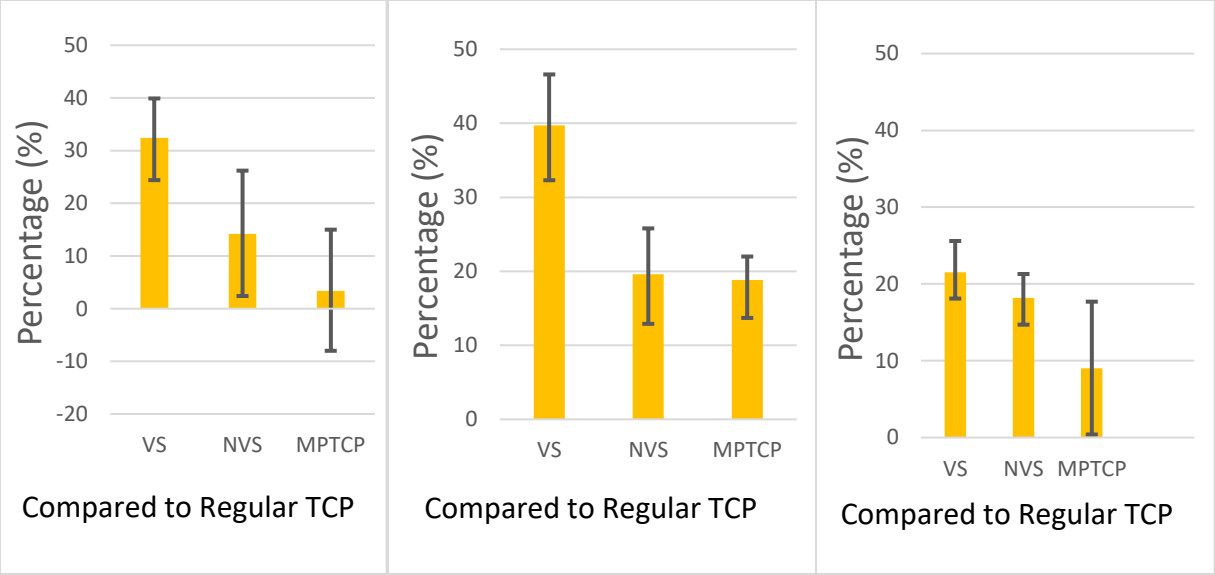


Figure 4.5: Topology 3

MEDIUM load cases will have simultaneous client-server connections and in the topology of 8 hosts, 6 to 8 client-server connections are considered a MEDIUM load case while, 10 and above connections is considered a HIGH load case which it will also have simultaneous client-server connections. After running the tests, we see the following result shown in Figure 4.6.



(a) LOW load (b) MEDIUM load (c) HIGH load

Figure 4.6: Results for Topology 3

In the above results, for the LOW load case shown in Figure 4.6(a), we see a 32.4% average increase in VS compared to regular TCP. NVS has over 14% average increase and standard MPTCP has just over 3% increase on average. In terms of confidence interval, we get 24.4% increase to 39.9% increase in VS compared to regular TCP, we get 2.4% to 26.2% increase in NVS compared to regular TCP and about 8% decrease to 15% increase in standard MPTCP compared to regular TCP. In the MEDIUM load case shown in Figure 4.6(b), we have an almost 40% average increase in VS and we see over 19% average increase in the NVS case and last, we see an over 18% average increase in standard MPTCP compared to regular TCP. In terms of confidence interval, we get 32.3% to 46.6% increase in VS, we get 12.9% to 25.8% increase in NVS compared to regular TCP and about 13.7% increase to 22% increase in standard MPTCP compared to regular TCP. Finally, in the HIGH load case shown in Figure 4.6(c), we see over 21% average increase for VS, over 18% average increase for NVS and standard MPTCP achieves over 9% average

improvement over regular TCP. In terms of confidence interval, we get 18.1% increase to 25.6% increase in VS, we get 14.7% to 21.3% increase in NVS compared to regular TCP and about 0.4% to 17.7% increase in standard MPTCP compared to regular TCP.

Now let's aggregate each case to get an average percentage increase for all topologies for the LOW load cases only. The results are given below.

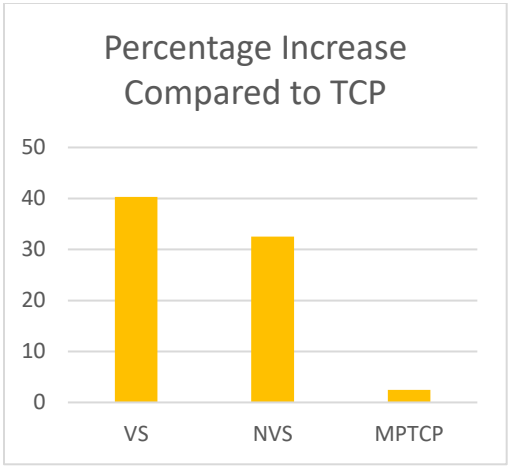


Figure 4.7: Aggregate for LOW Load Cases

The results show over 40% increase on aggregate for VS compared to regular TCP for the LOW load cases. For NVS we have over 32% increase compared to regular TCP and finally for standard MPTCP we have over 2% increase for LOW load cases. Now let's do the same for MEDIUM load cases and we see the results below. Here the results show an aggregate of over 55% increase in VS compared to regular TCP, over 40% increase in NVS and over 18% increase in standard MPTCP compared to regular TCP for MEDIUM load cases.

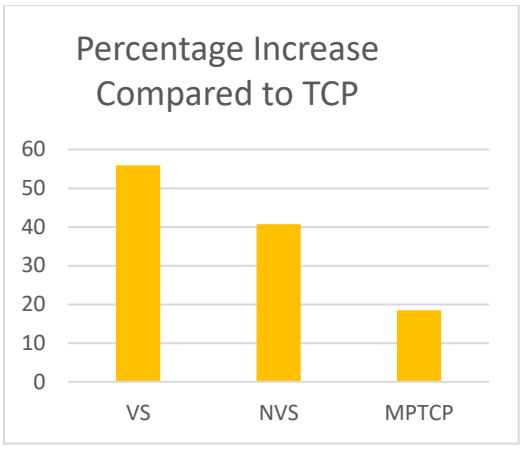


Figure 4.8: Aggregate for MEDIUM Load Cases

Lastly, we get the aggregate for all HIGH load cases in the three topologies. The results are given below.

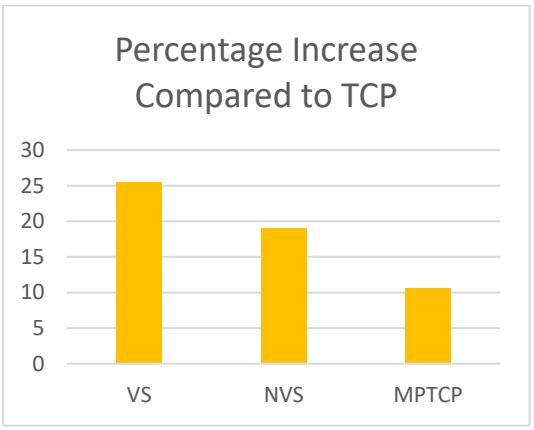


Figure 4.9: Aggregate for HIGH Load Cases

Again, here we see over 25% increase for VS, over 19% increase in NVS and about 10% increase in standard MPTCP compared to regular TCP for HIGH load cases. Now finally, on aggregating all the cases to together which is LOW, MEDIUM and HIGH, we see the below results.

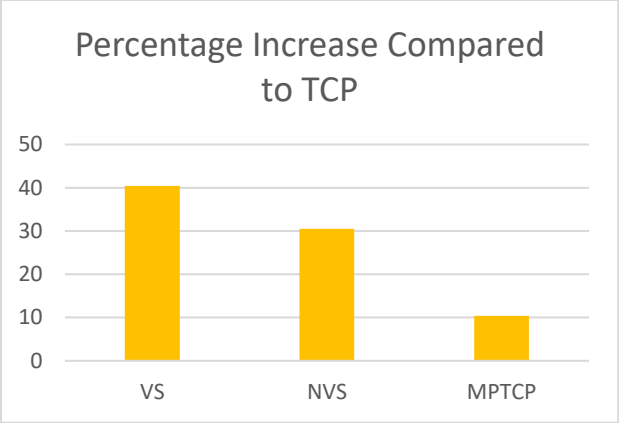


Figure 4.10: Overall Aggregate for all Cases

The final aggregate shows that there is an overall 40% increase in throughput in VS compared to regular TCP. NVS has over 30% increase in throughput and finally standard MPTCP has an overall 10% increase compared to regular TCP.

Thereby to conclude the results obtained, we clearly see the best throughput in VS which achieves a 40% increase on the regular TCP which we use on the internet today. Apart from this, we also observe that in MEDIUM load cases we get the best improvement in performance for MPTCP in general compared to regular TCP which could be because in LOW load cases, there can exists scenarios where most shortest path from sender to receiver might not have interference from other connections using the same path where the HIGH load case, all links are being used but MPTCP shines here over regular TCP due to its congestion control mechanism and its ability to spread out traffic across all the links. Finally, the results clearly show the usefulness of this technique compared to regular TCP where it not only improves throughput in the network but also brings in load balancing and network efficiency.

CHAPTER 5: CONCLUSION AND FUTURE WORK

In this chapter, we give the conclusion for the thesis and provide grounds for future work in this area.

5.1 Conclusion

As seen in the results from the simulation that the use of MPTCP can greatly improve load balancing and network utilization in a network topology. Whereas directing MPTCP traffic with the use of VLAN specific paths and even without VLAN specific paths produces even better results in the overall throughput of the network. The concept of IP Aliasing could assist with associating multiple IP addresses with any host which would help with the generation of additional MPTCP subflows. MPTCP congestion control algorithms like LIA, OLIA and BALIA are used to handle flow control and the amount of data that would be sent through a subflow. In this thesis, we use the BALIA algorithm due to its effects on TCP neutrality and responsiveness.

We used SDN techniques to direct subflows through the network at switch level as needed and we coupled the SDN and MPTCP concepts and created Hamiltonian paths for the MPTCP subflows to better utilize the network to improve load balancing. The paths are initially randomly created and the flows through each link is calculated. Then using standard deviation, we get a range for flows that can be allowed to pass a link at minimum and maximum. Once that is obtained, we can accordingly modify the paths to get optimal results and maximum network utilization. The number of paths is dependent on the egress ports on the switch connecting to adjacent switches from the ingress switch which is basically the switch that is directly connected to the hosts. The amount of unique VLANs which the hosts directly connected to the ingress switch belong to, is considered for the creation of paths in one technique while in the other technique this parameter is

not considered. Upon seeing the results, we notice that MPTCP with VLAN specific paths achieved the best throughput ahead of MPTCP without VLAN specific paths, standard MPTCP and regular TCP which is currently being used in the Internet. For LOW load cases, we see over 40% increase compared to regular TCP in MPTCP with VLAN specific paths, whereas, we see over 32% increase compared to regular TCP in MPTCP without VLAN specific paths and finally we over 2% increase in standard MPTCP compared to regular TCP. The story is similar in MEDIUM load cases but even better performance was observed here where we see around 55% increase compared to regular TCP in MPTCP with VLAN specific paths, whereas, we see over 40% increase compared to regular TCP in MPTCP without VLAN specific paths and we see over 18% increase in standard MPTCP compared to regular TCP. Finally, for HIGH Load cases, we see around 25% increase compared to regular TCP in MPTCP with VLAN specific paths, whereas, we see over 19% increase compared to regular TCP in MPTCP without VLAN specific paths and we see around 10% increase in standard MPTCP compared to regular TCP. The results obtained in the simulations show clear increase in throughput for the MPTCP process with the addition of paths compared to the regular TCP.

5.2 Future Work

Future work would include the automation of the path calculation process which could be fed into the switches which would be extremely beneficial and cost effective as it would save manual labor for the network administrator for the calculation of paths. Apart from this, path creation can also by dynamic rather than static where, path for subflows can keep changing based on the state of the network and the load taken by each link in the network. A hybrid of the Ingress fan-out phase with shortest path to destination from the first visited core switch is another

technique that can be considered, which again would be much more efficient compared to regular TCP. Ultimately, we can see with this thesis the benefits of using MPTCP in our networks especially coupled with the SDN style architecture where the direction of traffic can be controlled at the switch level.

LIST OF REFERENCES

[1] RFC 7149: Software-Defined Networking: A Perspective from within a Service Provider Environment, M. Boucadair, C. Jacquenet, March 2014.

[2] RFC 7426: Software-Defined Networking (SDN): Layers and Architecture Terminology, E. Haleplidis, K. Pentikousis, S. Denazis, J. Hadi Salim, D. Meyer, O. Koufopavlou, January 2015.

[3] A Load Balancing Strategy of SDN Controller Based on Distributed Decision, Yuanhao Zhou, Mingfa Zhu, Limin Xiao, Li Ruan, Wenbo Duan, Deguo Li, Rui Liu, Mingming Zhu, September 2014, Trust, Security and Privacy in Computing and Communications (TrustCom), 2014 IEEE 13th International Conference.

[4] BalanceFlow: Controller load balancing for OpenFlow networks, Yannan Hu, Wendong Wang, Xiangyang Gong, Xirong Que, Shiduan Cheng, October 2012, Cloud Computing and Intelligent Systems (CCIS), 2012 IEEE 2nd International Conference.

[5] Achieving high utilization with software-driven WAN, Chi-Yao Hong, Srikanth Kandula, Ratul Mahajan, Ming Zhang, Vijay Gill, Mohan Nanduri, Roger Wattenhofer, August 2013, SIGCOMM '13 Proceedings of the ACM SIGCOMM 2013 conference.

[6] Multipathing with MPTCP and OpenFlow, Ronald van der Pol, Sander Boele, Freek Dijkstra, Artur Barczyk, Gerben van Malenstein, Jim Hao Chen, Joe Mambretti, November 2012, High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion.

[7] RFC 6824: TCP Extensions for Multipath Operation with Multiple Addresses, A. Ford, C. Raiciu, M. Handley, O. Bonaventure, January 2013.

[8] On the Benefits of Using Multipath TCP and Openflow in Shared Bottlenecks, Marcus Sandri, Alan Silva, Lucio A. Rocha, Fabio L. Verdi, March 2015, Advanced Information Networking and Applications (AINA), 2015 IEEE 29th International Conference.

[9] Iperf - https://en.wikipedia.org/wiki/Iperf.

[10] IP Aliasing - https://en.wikipedia.org/wiki/IP_aliasing.

[11] FloodLight Controller – https://floodlight.atlassian.net/wiki/display/floodlightcontroller/Floodlight+VM.

[12] Mininet Virtual Machine – http://mininet.org/.

[13] sFlow - http://www.sflow.org/.

[14] WireShark - https://www.wireshark.org/.

[15] Opportunistic Linked-Increases Congestion Control Algorithm for MPTCP, Ramin Khalili, Nicolas Gast, Miroslav Popovic, Jean-Yves Le Boudec, February 2013.

[16] Balanced Linked Adaptation Congestion Control Algorithm for MPTCP, A. Walid, Q. Peng, J. Hwang, S. Low, July 2014.

[17] RFC 6356: Coupled Congestion Control for Multipath Transport Protocols, C. Raiciu, M. Handly, D. Wischik, October 2011.

[18] Stability of end-to-end algorithms for joint routing and rate control, Frank Kelly, Thomas Voice, April 2005, ACM SIGCOMM Computer Communication Review.

[19] Static Flow Pusher - https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/pages/1343518/Static+Entry+Pusher+API.

[20] OpenFlow Protocol - https://3vf60mmveq1g8vzn48q2o71a-wpengine.netdna-ssl.com/wp-content/uploads/2014/10/openflow-spec-v1.3.0.pdf.

[21] Iperf Documentation - https://iperf.fr/.

[22] Hamiltonian Path - https://en.wikipedia.org/wiki/Hamiltonian_path

[23] Hamiltonian Path Example Code - https://stackoverflow.com/questions/5766160/enumerate-all-hamiltonian-paths

[24] Hamiltonian Path Problem - https://en.wikipedia.org/wiki/Hamiltonian_path_problem