2017-12-15

Quantum-safe Switch-Controller Communication in Software-Defined Network

Dilruba, Raushan Ara

UNIVERSITY OF CALGARY

Quantum-safe Switch-Controller Communication in Software-Defined Network

by

Raushan Ara Dilruba

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE

DEGREE OF MASTER OF SCIENCE

GRADUATE PROGRAM IN COMPUTER SCIENCE

CALGARY, ALBERTA

December, 2017

Abstract

Software Defined Networking (SDN) is one of the key networking technologies with broad applications in data centers, cloud computing, and networking. SDN uses isolation of data planes and control planes to offer more efficient control of data communication. Security and reliability of communication between controllers that manage network control information and network switches that route traffic in the data plane, is essential in SDN architecture. The communication between these two layers uses OpenFlow protocol. We propose a quantum-safe solution for providing security and reliability for OpenFlow communication that exploits multiple paths between a switch and a controller. We analyze our solution regarding security and reliability and show its efficiency is comparable to an alternative solution that does not provide quantum-safe security.

Dedication

To

Dr. Rei Safavi Naini

- for her endless effort to make this world more secure

and

Zayra Afsheen Oriel

- my dearest daughter

- I wish your mom's work will inspire you to make this world a better place by your contributions.

Acknowledgements

I would like to express my gratitude to my husband Shah Arif for pushing me to go for higher studies and always supporting throughout this time from beginning to the end. When I started, my daughter was four years old, and now she is almost eight years old. Despite being so young, she was always understanding and cooperative during my long hours of absent and not listening to her stories. My friends here in Calgary - Shimu, Munna, Farheen and family, Atashi Di and family, Tatiana and family, Monzu and family, Rito apu and family, Rownok apu and family, Rizu bhai and family - it would not have been possible without your constant help that I asked for any day anytime. Thank you so much for being so caring. My mother, father, and mother-in-law prayed every day from back home Bangladesh for my good health. I missed them.

My endless thanks to Dr. Rei - my supervisor, teacher, mentor, advisor. Your enthusiasm and dedication, your criticism, your professionalism, your extra detail-oriented work made my life literally hell and there were times I thought to quit, yet helped to produce this work (I am grateful for that). You never want to compromise the quality of the work. You have exposed me to the research world by guiding me and trusting me. You always listened to me and my ideas regarding security awareness for the younger children. Thank you for your financial support from research assistantships and the cloud project.

I would also like to thank my committee members Dr. Mea Wang and Dr. Renate Scheidler for providing their constructive feedback. The feedback from the anonymous reviewers of CCS 2016 and Asia CCS 2017 have also helped to shape the work. Thanks to my lab mates Mamun, Morshed, Xi, Tam, Setareh, Masoumeh, Ahmad, Sepideh for sharing your ideas.

Finally, I would like to mention that, this degree is not just a degree to me. I went through emotional and financial ups and downs. This degree taught me to ask myself, *If I don't have to think about money, what is the thing that I would do that will make me happy?* The degree is the first step to the real world where rejuvenated myself wishes to make more significant impacts.

Table of Contents

List of Figures and Illustrations

Symbolslist

D	A secret message. 1, 32
D_k	kth secret message. 1, 65, 67, 68
OFm	A variable OpenFlow message. 1, 77
$Path$	Path between OpenFlow switch and OpenFlow controller. 1, 62
$\mathscr{P}$	Prime number. 1, 27, 66
$\mathscr{S}$	Total number of OpenFlow switches. vii, 1, 60, 73, 78, 101, 113
$\mathscr{T}$	Total number of IP addresses for one interface. 1, 72
a_i	ith random positive integer. 1
b	Total number of blocks of message D. 1, 67
$block$	Message D is broken into p-bit blocks. 1
$block_i$	ith block of message D. 1, 67
c	OpenFlow controller. 1, 60–62
$data$	Reconstructed message. 1, 77
g	Primitive root of p. 1, 27
int_i	OpenFlow switch's ith interface. 1, 61
int_k	OpenFlow controller's kth interface. 1, 62
ip	IP address. 1, 60, 61
$l_j(x)$	Lagrange basis polynomials. 1, 32
len	Length of OpenFlow message in bits. 1, 77
m	Total number of available paths between OpenFlow switch and controller in `mtdQSafeOF` setting. 1, 14, 15, 71–73, 77, 82, 111, 117

mID_k	Message ID. 1, 67
$metadata_j$	A secret message. 1, 67, 79, 80
n	Total number of shares of a secret created using (t,n) Shamir Secret Sharing Scheme. ix, 1, 12–15, 17, 32–34, 52, 59–63, 65, 67–72, 77–82, 89, 93, 95, 111, 117
p	Length of prime number in bits. 1, 66, 67, 77
pr	A random prime number. 1, 77
r	Control nodes. 1, 61, 62
s	OpenFlow switch. 1, 47, 60–62
$secret\_a$	A secret integer value of Alice. 1, 27
$secret\_b$	A secret integer value of Bob. 1, 27
sh	Variable shares of the OpenFlow message. 1, 77
$share_j$	$j$the share. 1, 32, 67
t	Minimum number of shares to reconstruct the secret using (t,n) Shamir Secret Sharing Scheme. vii, ix, 1, 13, 32–34, 59, 63, 68, 70, 73, 78, 79, 82, 89–94, 96, 101
t_ip	Time interval for randomizing IP in `mtdQSafeOF` setting. 1, 14, 15, 71, 72, 77, 108, 109, 117
t_pa	Time interval for randomizing path in `mtdQSafeOF` setting. 1, 14, 15, 71, 77, 111, 117
$thresholdConf$	Maximum number of paths the adversary can eavesdrop, but the message confidentiality can be preserved. 1, 15, 65, 69, 70, 73

thresholdRel	Maximum number of paths the adversary can block, but the message reliability can be preserved. 1, 15, 65, 69, 70, 73
u	An integer. 1
v	Total number of paths attacker gets access to for eavesdropping. 1, 73
w	Total number of paths attacker gets access to for blocking. 1, 74
x	A random integer value. 1, 32
x_j	jth random integer value. 1, 32, 67

Chapter 1

Introduction

We are living in an era of digital society. Search engines (Google, Yahoo), social networks (Facebook, Twitter), online stores (Amazon), e-commerce sites (online banking sites) have impacted and improved human life. Communication is now so much easier than ever. People can access these services from anywhere using their computers, mobile phones, and many other gadgets. In the near future, people will be able to control and access their home appliances, for example, air conditioner, refrigerator, cars, and humidifiers, that is known as Internet of Things (IoT).

1.1 Computer Networks

The underlying building block of the digital services is computer networks. The Internet connects almost everything and is accessible from anywhere. These services are hosted in data centers across the world and interconnected by wide area networks (WAN). One central problem for computer networking research is to provide fast, reliable and secure networking.

Network devices that generate, route and transmit data are called *network nodes*. Network nodes can be of two types; (i) *hosts*: that generate data and communicate with other hosts or ask for services from servers, for example, personal computers, phones and servers, (ii) *networking nodes*: that process data based on predefined rules and perform actions (e.g., drop or forward), examples are switches and routers. The data (a list of bits bytes) is formatted into *packets* that are carried by a packet-switched network. Packets contain two kinds of data; (i) *control information* that includes the source and destination address, error codes and sequencing information and is included in *Packet Headers*, and (ii) *payload* that includes the actual data.

The packet processing in networking devices can be modeled as *match:action* processing where the process matches certain patterns of packet headers and then performs the action. The packets

are processed and forwarded, and this is referred a *network policy*. The management of this policy is crucial because of the complex network hardware, applications, connectivity, services and physical locations. A network policy can change when there is an architecture change, device failure, or attacks, and then needs re-configuration or re-enforcement.

Important network management tasks:

Switching: uses the physical address or MAC (Media Access Control) address of a network node to perform forwarding decisions. It operates at Layer 2 (data-link) of OSI model. Switches are plug and play devices that have three functionalities, namely, (i) address learning, (ii) forwarding, and (iii) loop preventing. Below is an example of *match:action* for switching where if a MAC address matches, the packet is forwarded to the specific port.

match: destinationMAC=01:00:00:30:00:02 *action*: forward to port 2

Routing: uses the destination IP address to forward packets to their destinations. One rule could be to forward packets to port 4 that matches IP address 1.0.0.0/24 as written below.

match: destinationIP=1.0.0.0/24 *action*: forward to port 4

Monitoring: It is important to know the network status, bottlenecks, failures, attacks. For this purpose, network nodes, switches, and routers need to collect statistics. One way to gather the statistics is to match the desired protocol number from the packet header and count. For example, we can configure to know TCP (protocol number 6) on port 80.

match: protocol=6, sourcePort=80 *action*: count

Firewall: A firewall (FW) is a hardware or software or a combination of hardware and software that monitors the transmission of packets that attempt to pass through the perimeter of a network. FWs perform security functions like packet filtering. It determines whether to allow or deny the incoming or outgoing packets, based on security policy rules that have been established. Packet filters use packet headers to decide whether to block the packet or allow it to pass through a FW. Let us say; the security policy is to allow all web traffic (TCP protocol with port 80) and block all SSH (Secure SHell, TCP protocol with port 22) traffic. The rule can be written as follows:

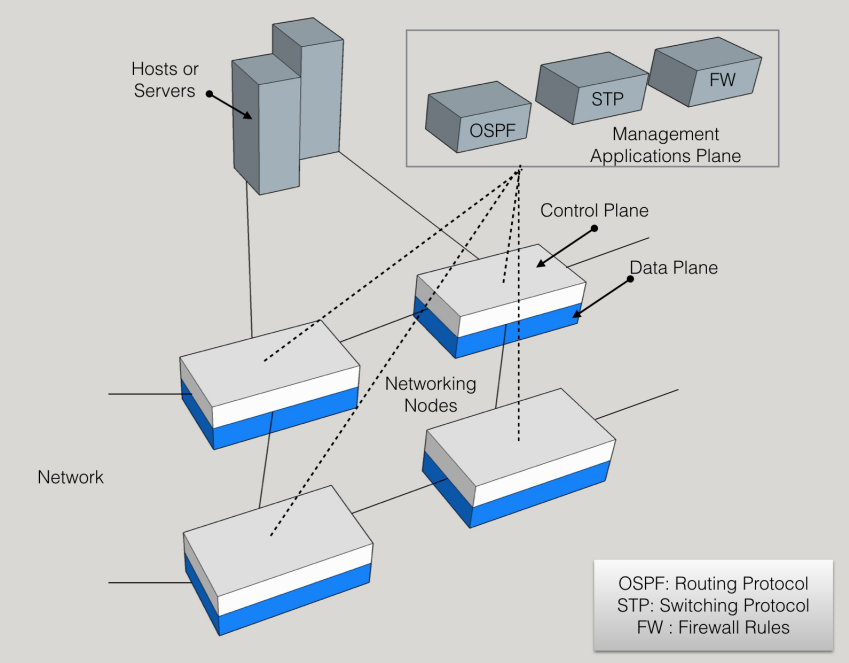


Figure 1.1: The traditional network and three functionality layers

match: protocol=6, sourcePort=80 *action*: permit *match:* protocol=6, destinationPort=80 *action*: permit

match: protocol=6, sourcePort=22 *action*: drop *match:* protocol=6, destinationPort=22 *action*: drop

Computer Network Management Issues Computer networks have three functionality planes: (i) the *data plane* that is composed of different nodes that carry and forward data based on network policy, (ii) the *control plane* that is responsible for representing the protocols to generate the policies and install in the forwarding tables of the data plane nodes, and (iii) the *management plane* that has the services and tools to remotely monitor and configure the control plane [67].

In a traditional network system (illustrated in Figure 1.1), the control and data plane is tightly integrated into the same hardware. The control plane is responsible for network discovery and mapping. The control plane computes the path based on the device configuration and the neighbor discovery protocols. Once the path computation has been done, the routing table is pushed to the data plane's programmable chip where the routing table is called forwarding table. It is a static architecture where the control plane functions are limited and very proprietary to the hardware

vendor. Some of the issues of the traditional network are: functionalities are implemented in the dedicated appliances, appliances have proprietary APIs, it needs specific configurations, the configurations are time-consuming and sometimes error-prone, the tasks are automated by home-grown scripts and the service innovation is slow and costly. These problems make the task of network management complicated. For example, one needs to configure switches for Layer 2 routing and routers for Layer 3. These switches and routers are vendor specific, and thus the protocols STP (Spanning Tree Protocol) for Layer 2 and OSPF (Open Shortest Path First Protocol) for Layer 3 must be configured using proprietary instructions. It is not possible to customize for new routing protocols. FWs and Monitoring devices' configurations are also device specific.

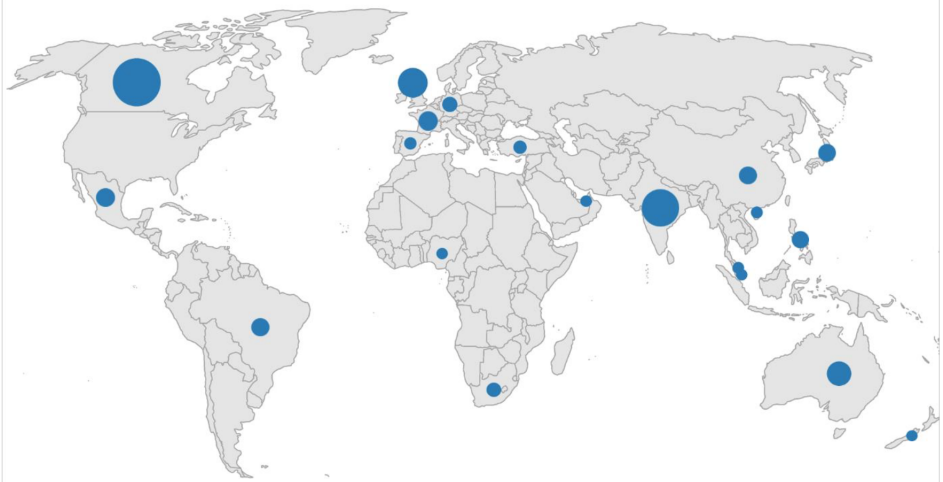
There have been many innovations in the user space technologies, for example, computer systems, operating systems, mobile communication and application development. But the change in the underlying network infrastructure has been slow-paced due to its closed architecture and being vendor proprietary.

1.2 Network Security

Network security is about protecting the network against security attacks. Network security attack can be broadly categorized into confidentiality attacks (when the attacker can get network data), integrity attacks (when attacker alters the data) and availability attack (when attacker blocks the data communication that makes it unavailable). The recent increase in cyber attacks is alarming as it affects government, private companies, health sectors, education sectors, even individuals. The *WannaCry Ransomware Attack* was a world-wide cyber attack in May 2017, targeting Microsoft Windows Operating Systems by encrypting data and demanding ransom. Ransomware attacks cost 1.3 dollars in 2016 according to FBI [1] where Canada was second after the USA in reporting a maximum number of ransomware attacks (3,772 out of 298728) (Figure 1.2).

Top 20 Foreign Countries by Victim

Excluding the United States[12]



1. Canada	3,772	6. Brazil	533	11. Germany	350	16.United Arab Emirates	202
2. India	2,188	7. Mexico	521	12. South Africa	337	17. Malaysia	193
3. United Kingdom	1,509	8. China	473	13. Turkey	286	18. Singapore	192
4. Australia	936	9. Japan	447	14. Spain	229	19. Nigeria	188
5. France	568	10. Philippines	439	15. Hong Kong	223	20. New Zealand	187

Figure 1.2: Ransomware attack statistics (from FBI's report 2016)

1.3 The Emerging Software-defined Networking (SDN)

Software-defined networking (SDN) is an approach to networking that simplifies network management tasks by providing an open interface between networking nodes (*data plane*) and the controlling software (*control plane*). The development of SDN is promoted by the Open Networking Foundation (ONF) through open standards. In their release of 'SDN architecture' issue 1 [14], the purpose of SDN is outlined as *"to enable innovation in proprietary vendor's network through open Application Program Interfaces (APIs)"*. The expectations from the emerged new architecture are to have a global view of the network and the resources, easier control of various switches, adequate provisioning/releasing resources, for example, faster convergence, higher utilization, and more fault-tolerance. SDN differs from the traditional network in the following features:

Decoupled control plane and data plane: SDN decouples the control plane from the data plane. The control plane becomes the centralized logical controller that gathers information and provides a global view. Management applications run on top of the control plane.

Open, standard interface: The interface between data plane and control plane is standard and open. One of the protocols that are being standardized is OpenFlow [49]. Now the vendor will sell the hardware data plane devices made with ASICs (Application Specific Integrated Circuits) or FPGA (Field Programmable Gate Arrays), or software switches running on servers or clouds. Controllers have different applications and functionalities developed by programmers or software engineers. This model brings benefit to network administrators because they can use any vendor data plane and then can configure and control by using the controllers that fulfill their requirements.

General packet processing model and unified configuration: In SDN, packet processing is unified regardless of the device functionality (switching, routing, and filtering). The model uses a match-action table that contains a list of rules. The table is referred to the *Flow Table* that contains the matching rule and the corresponding instructions. There are also other fields like counter, priority, timeouts, and cookies (Figure 1.3) . The rule match is against the header information for MAC, IP, TCP source and destination address/port.

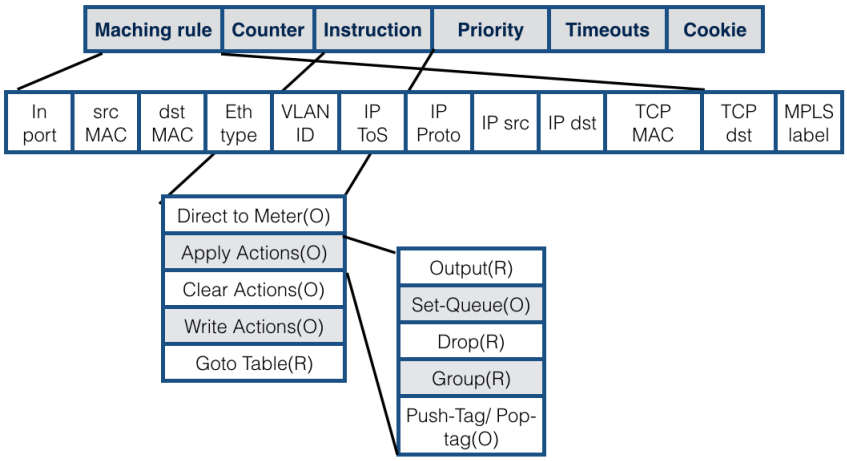


Figure 1.3: OpenFlow table rule structure

These features simplify the design and the deployment of the task of network management . In SDN, network administrators only need to install the required applications on the controller. Through the open interface, the unified rules will be installed in the data planes flow table. Network administrators do not need to configure per device as they did in the traditional network settings

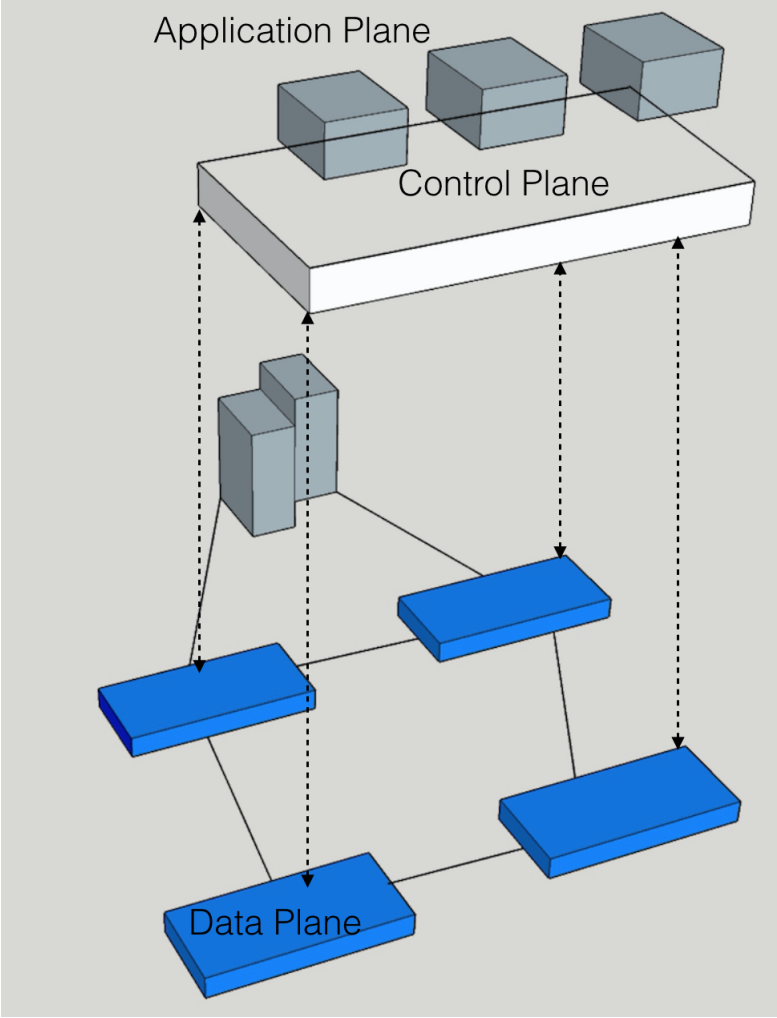


Figure 1.4: Software-defined networking and three functionality layers

where they needed to configure nodes, for example, switch, router and FW individually. Figure 1.4 abstracts an SDN.

1.3.1 OpenFlow Protocol Security

An important component of SDN architecture is the communication channel between the data plane and the control plane. Currently, OpenFlow protocol has been proposed particularly for this channel, and that introduces new threats. OpenFlow has a well-established open standard that is maintained and updated by Open Networking Foundation (ONF) [49]. Providing secrecy, reliability, and authenticity for any data passing through this channel is very important.

Attacks on the OpenFlow channel can be broadly grouped into *Passive Attacks* and *Active Attacks* (we only consider attacks on OpenFlow channel and not other components of SDN). A passive attacker's motivation is to eavesdrop on data rather than altering it. If OpenFlow channel is plain-text, the attacker can obtain switch ID, controller ID, OpenFlow Packet-in message, table ID, statistics, and have a detailed map of the network. Passive eavesdropping [20], [16] attack on OpenFlow channel allows learning the network related information and can be used in multistage attacks. In these attacks after a passive learning stage, the attacker will use the information to launch active attacks. In multistage attacks, [39] the attackers use advanced techniques to gain access to the network and persistently retain control over the system for an extended period without being noticed [91].

Active attacks can affect availability, integrity, or confidentiality of the OpenFlow channel. OpenFlow channel availability attacks can cause interruption of the TCP connection. One such attack is the TCP SYN Attack [52] that keeps a controller waiting for acknowledgment reply, resulting in the switch and the controller not being able to establish an OpenFlow channel. The attacker who has already gained access to the internal network can also pose an APT (Advanced Persistent Threat) to the network.

In [66] seven threat vectors on SDN have been identified, the third of which is plain-text communication on the control plane. To secure this channel, TLS (Transport Layer Security) protocol is suggested in OpenFlow switch specification [49], [66], [27].

1.3.2 Long Term Security

Today's network security is based on cryptographic algorithms that use hard mathematical problems (Discrete Logarithm or Integer Factorization problems) and the assumption that these problems cannot be solved using currently available computers. It is well known that many important cryptographic algorithms will become insecure if a quantum computer exists. Shor invented a quantum algorithm [89] that demonstrates that the integer factorization problem can be efficiently solved on a quantum computer which brakes the widely used RSA scheme. The scheme assumes

that factoring large numbers is computationally expensive. But the assumption is correct for classical computers. So far no classical algorithm is known that can factor in polynomial time. *"Shor's algorithm shows that factoring is efficient on an ideal quantum computer, so it may be feasible to defeat RSA by constructing a large quantum computer. It was also a powerful motivator for the design and construction of quantum computers and the study of new quantum computer algorithms.*" [9].

1.3.3 Quantum-safe Security

The recent announcement by NSA [15] and the recommendation to move to quantum-safe crypto algorithms, followed by the NIST report [30] that advocates a plan for moving towards quantum-safe cryptography has generated much interest in designing systems that stay secure even if a quantum computer exists.

We consider a quantum-safe approach that uses physical assumptions instead. That is, we assume there are multiple network paths between a controller and a switch and not all of them can be captured by the adversary at the same time. This assumption is reasonable as in many cases controllers and switches are located at different geographic locations and connected through a network. For reliability in all cases, one needs to assume multiple paths. So our proposed solution can be seen as employing multiple paths for providing both security and reliability.

1.4 Contributions

1.4.1 Using network multiple paths to provide security and reliability

Our objective is to show how to use *multiple paths* between a switch (controller) and controller (switch) to provide security and reliability for OpenFlow protocol communication in an SDN architecture. We assume paths are generated by a path manager (PM) that knows the network architecture and can specify network paths between two nodes, for data packets. A switch (controller) can request n paths to a controller (switch) to the PM and can receive the paths details.

The nodes can ask these paths be used for delivery of their data packets, and this is enforced by the system. Paths are opaque to the adversary, and the packets sent on them are inaccessible to the adversary. The adversary, however, can select up to a threshold number of paths to eavesdrop or block. Packets that are sent on a path that is eavesdropped on are accessible to the adversary. The uncorrupted paths serve as a shared secret carrying resource for the sender and receiver and replace the traditionally used shared secret key for providing security.

1.4.2 Quantum safe security

The main cryptographic primitive that we use is the Shamir (t,n)-threshold secret sharing scheme (SSS) that divides a secret into n shares, such that any t shares can reconstruct the secret, and any $t-1$ or fewer shares do not leak any information about the secret. Shamir SSS provides information theoretic security, and its security guarantee will hold even if a quantum computer exists. In our schemes, the adversary has access to $t-1$ shares and that is why we say that our scheme is quantum safe.

1.4.3 `bQSafeOF` (Basic Quantum-safe OpenFlow)

The `bQSafeOF` (Basic Quantum-safe OpenFlow) provides confidentiality and reliability by directly using Shamir SSS: each message is broken into n shares, each sent along one path between the two. The receiver runs the reconstruction algorithm of the SSS to recover the secret. The system provides (i) perfect secrecy if up to $t-1$ paths are eavesdropped upon, and (ii) reliable communication if up to $n-t$ paths are jammed (blocked). This scheme provides perfect secrecy and reliability as long as the adversary is within the specified bound. These bounds, however, are sharp, and going slightly (one more path) above them results in complete loss of security (data compromise or unavailability).

1.4.4 `mtdQSafeOF` (Moving Target Defense Quantum-safe OpenFlow)

To alleviate the total loss of security and reliability of `bQSafeOF` when the bounds are exceeded, we use randomization of the adversary's view to prevent them from targeting their attack. Randomization has been widely used as a Moving Target Defense (MTD) strategy [96]. We use this approach combined with `bQSafeOF` and refer to the system as `mtdQSafeOF` (MTD Quantum-safe OpenFlow). We provide randomization in two settings. Firstly, we consider randomization when a controller is connected to multiple switches, and the goal is to prevent the attacker to target a particular switch. Here randomization of the adversary's view is by randomizing paths' identifiers at consecutive time intervals. We assume each switch sends at most one message in a time interval. Once a path is compromised, the adversary is not only able to find the message share that is sent on that path but also use the accompanying metadata to determine the switch that uses the path. It allows the attacker to gradually determine all the paths that are connected to a switch and targeting that switch. We consider a setting where the starting and the ending points of a path are specified by some interface nodes of the network, and PM calculates paths to connect these nodes. By randomizing paths at every t_ip time, the adversary cannot target a specific switch, and so their success chance of decreases (depending on the total number of paths) for subsequent messages.

Secondly, we consider a setting where there are $m > n$ paths available between the switch and the controller and PM can choose n paths randomly in every t_pa time unit. The success chance of targeting a specific switch-controller communication even reduces more when IP randomization strategy is used. However, randomization has cost. The m paths may have different delays and data rates, and PM may choose slower routes.

1.4.5 Security Evaluation

Our scheme's security level is dependent on the prime length. If the prime is 128 bit, it is equivalent to the security level of AES 128. We considered a prime length that covers most of the OpenFlow messages and reduces creating OpenFlow message blocks. We have analyzed OpenFlow message

lengths from the OpenFlow switch specification [49], and our analysis shows that the prime length can be 160 bits as it covers around 70% of total OpenFlow message types. This length is larger than the NIST 112-bits-of-security requirement [25]. For message that are larger than the prime length, one needs to break the message.

We evaluate the security when the attacker accesses *threshold* paths. The *threshold* is defined as the maximum number of paths that attacker can capture, but the system still provides confidentiality ($thresholdConf$) and reliability ($thresholdRel$). The $thresholdConf$ is $t-1$ and $thresholdRel$ is $(n-t)$. We first compare `bQSafeOF` with a system design where the same security and reliability level is achieved using TLS protocol [40] and a sufficient number of disjoint paths (to provide same reliability) between the controller and the switch. We call the alternate solution `notQSafeOF`. The probability of successfully eavesdrop ping on block *specific* switch-controller communication within threshold number of path access is always zero for both `bQSafeOF` and `notQSafeOF`(if we assume that the TLS key has been compromised using a quantum computer). The reliability of the `notQSafeOF` is better because it requires all n paths to be blocked whereas `bQSafeOF` reliability is less than n. When the attacker has access to more than threshold number of paths, the probability of successful attack becomes one.

We also note that the attacker may use the static IP addresses and path assignments to build *switch-IP* information. To attack a *specific* switch-controller communication, the attacker needs that switch-IP information. The probability of attacking a specific switch-controller communication is one.

For `mtdQSafeOF` i.e. when `bQSafeOF` is combined with IP change at every t_ip time unit and path assignment change (choosing n paths out of m paths) at every t_pa time, if the attacker has access to threshold or more than threshold number of paths, the probability of attacking successfully any specific controller-switch communication becomes less than one. Because of constantly IP changing, the attacker is not able to build the switch-IP mapping table, and the attacker cannot target a *specific* switch-controller connection.

1.4.6 Performance

We used OpenSSL library to implement a (t,n)-SSS and `bQSafeOF`. The selection of parameters takes into account the OpenFlow protocol standard. To evaluate the efficiency of our system, we compare `bQSafeOF` with a system design `notQSafeOF` where the same security and reliability level is achieved using TLS protocol [40] and a sufficient number of disjoint paths (to provide same reliability) between the controller and the switch. We also wrote code to change IP addresses and paths randomly for `mtdQSafeOF`. The `notQSafeOF` design is not post-quantum secure and only serves the purpose of demonstrating the practicality of our solution.

Our experiment results show that the computation cost (share construction and reconstruction for `bQSafeOF` and message encryption and decryption for `notQSafeOF`) is comparable although our code is not optimized as opposed to the OpenSSL code. The communication cost -one way and round trip of `bQSafeOF` performs better than the `notQSafeOF`. The reason is, `bQSafeOF` prepares shares before sending them, and `notQSafeOF` creates a secure channel, encrypts and then sends data over the channel. The `mtdQSafeOF` has operational cost for IP changing and route changing at some time unit interval.

1.5 Difference From the Existing Work

Our work is different from existing work in three ways:

1.5.1 Application of (t,n)-Shamir Secret Sharing Scheme

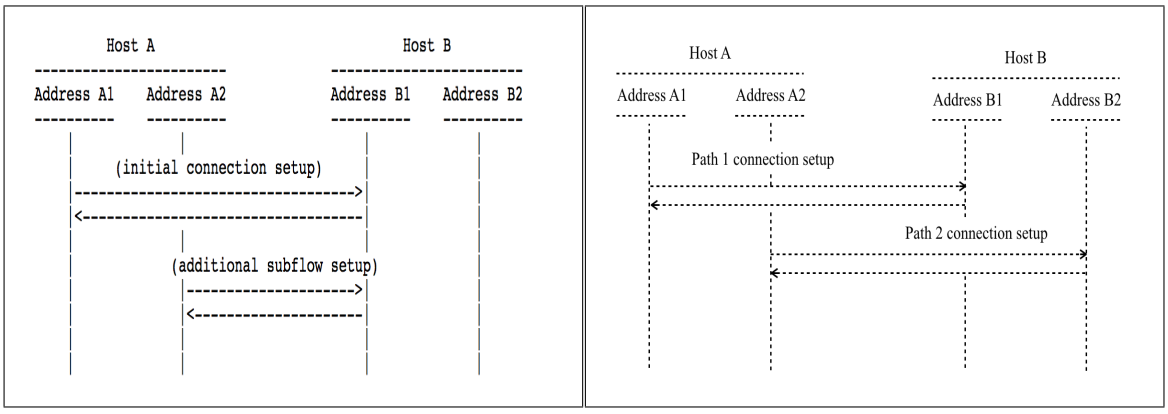
Threshold cryptography [38] is a well-studied problem for almost 40 years now. Shamir Secret Sharing Scheme is one of the threshold cryptosystem. However, the applications are mostly in cloud distributed storage servers [70], [44], [90], authentication systems [56], ad-hoc networks for key management [99], to share the key for RSA and ECC in MANET [46], threshold digital signature systems [51], [81], threshold signature schemes [37], [55], electronic voting [34] and security against malicious administrators in SDN [74]. Multi-path routing and secret sharing were

used to secure VoIP communication [78] where secret sharing was applied to the output of different speech compression coding and in a MANET [73]. In [72], data confidentiality has been provided to MANET (Mobile Adhoc Networks) using (t,n) Shamir Secret Sharing. The message is broken into n shares, encrypted and then transmitted through n node-disjoint paths. But the application of threshold cryptography in practical network communication was limited due to the inflexibility of network structure. The recent advancement in network virtualization, cloud, and software-defined networking opens a possibility of applying those security models. It is now relatively easier to manage and design the network, for example, providing disjoint paths. In [66], it is suggested to apply threshold cryptography in providing security for OpenFlow channel.

The new thing in our scheme is, it uses multiple paths between the controller and the switch in an SDN settings to achieve confidentiality and reliability against quantum attacks.

1.5.2 More than one path for the same connection between the same pair of endpoints

Current communication system is based on single destination address for the same connection while transferring data. In real life, the source and the destination endpoints may have multiple paths in between them. When the source wants to send data to the destination, it chooses one path, establish connection and sends data. If this connection fails for any hardware or software fault or attack, the system chooses another path, establishes connection and sends the rest of the data.



(a) MPTCP scenario [48] (b) `bQSafeOF` scenario

Figure 1.5: MPTCP vs. our scheme scenario

The multipath TCP (MPTCP) concept establishes more than one connections between the same pair of endpoints using multiple paths. First it establishes the main connection and starts sending data (flow). If multiple paths are available, it establishes the secondary connection and sends additional data (subflows). Data transfer happens using more than one paths. Figure 1.5a illustrates the concept.

Our scheme also uses the multiple paths between the same pair of endpoints but `bQSafeOF` creates shares of the data and sends each share through each path. At the destination side, it collects shares and re-constructs the data. Figure 1.5b illustrates the concept. The difference is although MPTCP is using multiple paths but the destination address is single. In our case, there are multiple destination address for single end host and our scheme uses disjoint multiple paths to send the data.

1.5.3 Going over the t bound of (t,n)-SSS

For (t,n)-SSS, if the attacker has access to t shares, they can reconstruct the share and thus the probability of successfully eavesdropping or blocking one specific switch controller communication is one. We use MTD strategy that changes attacker's view and the attacker's success probability of attacking one specific switch controller communication becomes less than one.

1.5.4 Multiple path MTD technique

In MTD [79], network randomization techniques have been applied in [29], [33], [18] and [60]. Our `mtdQSafeOF` is different because it applies randomization techniques in multiple destination for a single data transfer whereas the existing works are for single destination end host.

1.6 Thesis Organization

The rest of the thesis is structured as follows: Chapter 2 gives the background of network communication in single and multiple paths, secure and reliable communication techniques, Shamir Secret Sharing and its applications, Moving Target Defense and its techniques. Chapter 3 provides preliminaries on traditional vs. software-defined networking (SDN), SDN to improve traditional network's security, Security issues of SDN itself, related work on secure and reliable OpenFlow communication. Chapter 4 details the models, assumptions and the proposed solution with a security analysis. Chapter 5 is about implementation, experiment details and comparison of performance results, and we put concluding remarks in Chapter 6.

Chapter 2

Reliable and Secure Network Communication

2.1 Introduction

This chapter provides background on (i) basics of network communication, (ii) reliable network communication protocols, (iii) single versus multiple path communication, (iv) secure network communication techniques, (v) Shamir secret sharing and its application, and (vi) moving target defense. The aim is to have a foundation on the available reliable and secure network protocols, understand Shamir secret sharing and its application in network communication and background on moving target defense.

2.2 Network Communication Between Two Entities

Data communication between two entities follows the OSI (Open Systems Interconnection) model. The OSI model enables data communication interoperability without knowing the underlying technology or structure. The model abstracts seven layers bottom up as: (i) physical, (ii) data-link (iii) network, (iv) transport, (v) session, (vi) presentation and (v) application. Each layer serves the above layer and is served by the layer below. Each layer has its own communication protocols. The protocols are based on the assumption that the communication uses a single destination address.

2.3 Reliable Communication

Reliable communication protocols provides *reliability* in terms of the data delivery to the intended recipient(s). Network communication reliability can be affected by a path failure between two nodes. A general network is a collection of routers, switches, and links. There can be multiple

path options available between two entities. Protocols at the network layer (routers) can calculate and build a topology map of the network. The open shortest path first (OSPF) protocol collects link state information from the neighboring router and uses Dijkstra's shortest path algorithm [41] to compute paths for each route based on destination IP. The cost factors are the distance of the router, link availability, link data throughput and these are expressed as a unitless number. Protocols at the transport layer help to provide acknowledgment of the packet receipt and sequencing the packet in the correct order that provides reliable, ordered and error-checked data delivery between two hosts. Transmission Control Protocol (TCP) is one of such that works with IP.

Dijkstra's algorithm: considers points (nodes) where some or all pairs are connected, and the length of the connection is given. At least one path exists between any two nodes. At this settings, two problems were considered. Problem 1 is to construct the tree of minimum total length among the nodes. A tree is a graph with one and only one path between every two nodes. Problem 2 finds the path of minimum total length between two given nodes.

2.3.1 Transmission Control Protocol

The functional specification and the standard of TCP are maintained by the Internet Standard RFC: 793 (version 1981). The motivation behind this protocol is to address *the robustness in the presence of communication unreliability and availability in the presence of congestion*, specifically when computer communication is playing increasingly important role in the military, government and the civilian environments. TCP is described as *connection-oriented, end-to-end reliable protocol*. The functional specification defines the header format. The header carries information, for example, source and destination port numbers, sequence number, acknowledgment number, window and checksum. The header format is represented in the Figure 2.1.

There are some operations expected of TCP. They are.

(i) Basic data transfer: The first functionality of TCP is to be able to send a continuous stream of data (in the form of octets) in both directions (sender to receiver or receiver to sender). It can create segments of data and use a numbering system to identify and reorder each segment at the receiver

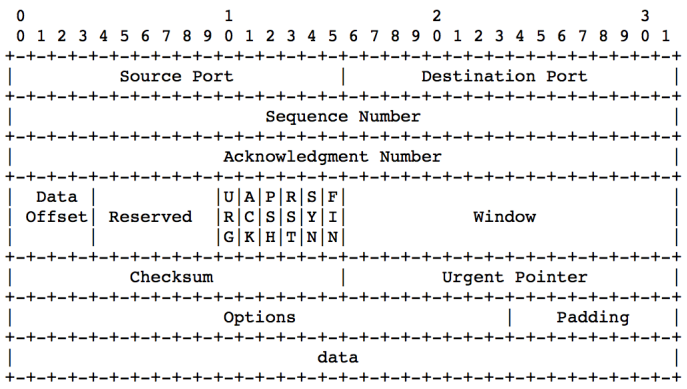


Figure 2.1: The format of TCP header

side.

(ii) Reliability: Reliability is defined by the process that recovers damaged, lost, duplicated or delivered out-of-order TCP transmitted data. This process is ensured and implemented by a positive *Acknowledgment* from the receiver side. The data is re-transmitted if no acknowledgment is received within a timeout interval. The segments are assigned a sequence number that helps to reorder. There are also processes to handle data damage.

(iii) Flow control: TCP should be able to provide a way for the receiver to regulate the amount of data sent by the sender. It is known as flow control. This feature is implemented by a *window* concept where every ACK indicates a range of acceptable sequence numbers beyond the last segment successfully received.

(iv) Multiplexing: A host may require handling many processes simultaneously. TCP achieves this requirement by creating a *socket*. A socket is generated by binding the IP address and a port number. The socket number can be uniquely identified for each connection.

The connection establishment of TCP is known as a 3-way handshake. It is illustrated in Figure 2.2. The Figure shows the client and server states after each TCP communication. The host or client sends SYN to the listening server. The server sends an acknowledgment SYN-ACK in reply. Again the client sends an ACK to confirm, and TCP connection establishes.

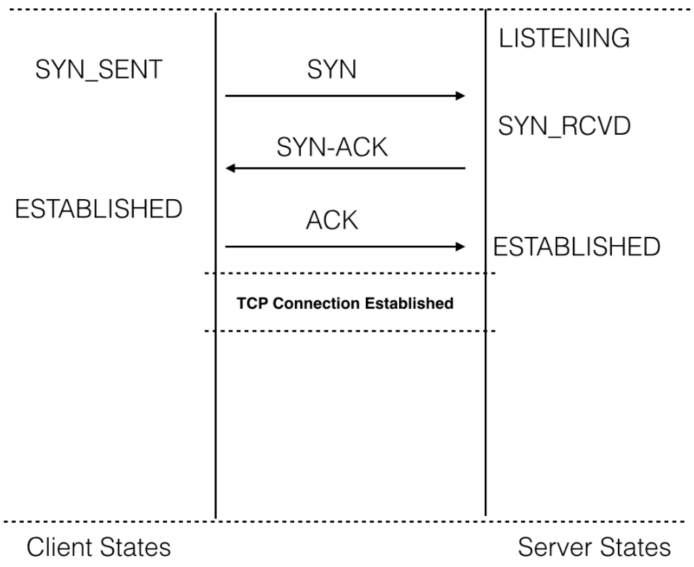


Figure 2.2: TCP 3-way handshake and OpenFlow data transmission

2.3.2 Example of Reliable Network Communication

Figure 2.3 illustrates a reliable communication between two hosts. Many paths may be used for a single communication as individual packets are routed to the destination. No fixed path is established. Packets are routed according to the best path available at the time computed by the router. Prior to transmission, each communication is broken into packets which are addressed and numbered using IP and TCP protocols. All the destination packets may be reassembled into order according to their sequence number using TCP.

2.4 Single Path vs. Multiple Path Communication

The underlying structure of the Internet is a collection of multiple paths. But unfortunately, communication protocols are designed on the assumption that it is single path communication and thus not utilizing the multiple path resources. The survey [84] describes why the future of the Internet is multipath. Here we summarize some of the reasons:

Resource Pooling: Packet switching, Content Delivery Networks (CDNs), peer to peer networks and cloud computing are some of the technologies that are based on resource pooling tech-

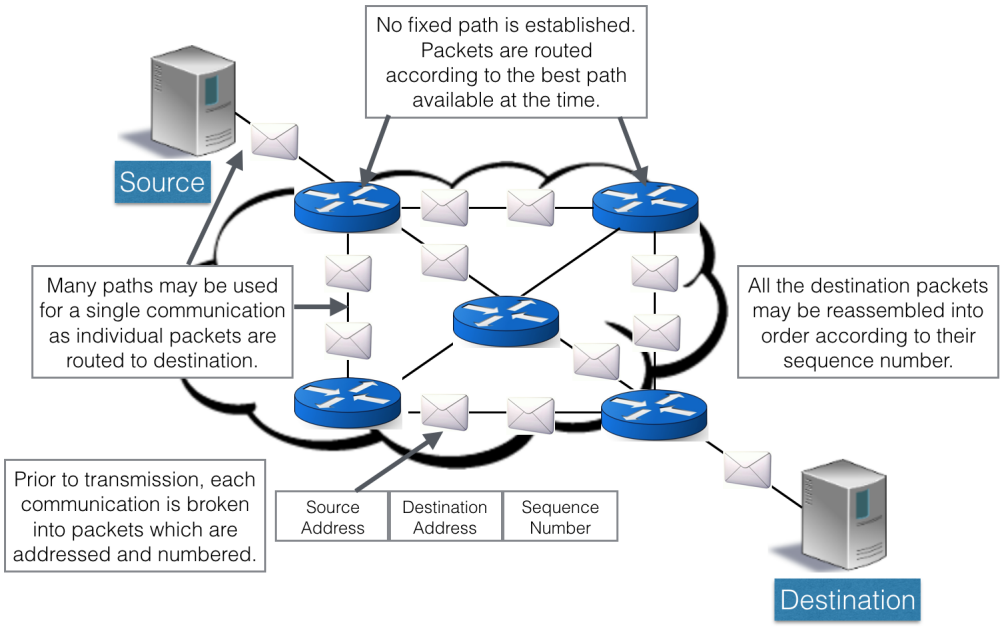


Figure 2.3: Reliable network communication

niques. The main idea is to present a collection of resources as a single pooled resource focusing on improving resource efficiency. In summary, multipath routing can pool together a number of distinct links.

Load Balancing: Load balancing allows controlling congestion by distributing the load amongst the elements of the resource pool. The load balancing technique can be over resources (multiple access links or paths) or temporal (balanced between peak or nonpeak hours).

Reliability and Fault Tolerance: Reliability and fault tolerance have become extremely important. By utilizing multipathing, network communication can enhance reliability at various layers [22]. At the network layer, it can be multihomed servers with different ISP connections or backup paths as an alternate to the primary path failure. At the physical layer, antenna diversity can be used to control fading effects in wireless communication.

Security: In [97], multiple radio interfaces have been proposed for achieving greater data confidentiality in a wireless network by encrypting data and splitting it to multiple parts that are then transmitted using different physical channels using Wyners wiretap channel theory [94].

High Throughput: At the transport layer, multipath TCP (MPTCP) allows the pooling of mul-

tiple links into support higher connection speed. At the physical layer, MIMO technology is revolutionary in wireless channels by providing enhanced communication rates.

Role in the Future Internet Architecture: Multipath will play an important role in future Internet architectures and next-generation wireless and mobile networks. Typical mobile phones can multi-home heterogeneous connectivity, for example, Wi-Fi, 3G. 5G standard is expected to be standardized in the near future. Multipathing is also useful in emerging networking configurations, such as data centers, clouds, P2P, CDN and software-defined networking (SDN).

Anonymization for Privacy: In [62], dynamic traffic distribution mechanisms were applied to the Tor anonymity software that uses multipath to gain improved throughput and decreased latency in a Tor network.

2.5 Secure Network Communication

Secure Communication refers to the scenario when two entities communicate, and they do not want some third party to listen to them. The basic properties of security are: (i) confidentiality: that is a property of a system that ensures message content is only readable by the intended recipients., (ii) integrity: message integrity ensures that modifications to messages by third parties can be detected, and (iii) availability: for any information system to serve its purpose, the information must be available when it is needed.

2.5.1 Cryptographic Premitives

Cryptography is the most practiced technique for providing secure network communication. It is a way to hide the original message in another form of the message while passing through the network so that no third party is able to find the original message (under some assumptions). A key is shared between the two entities (let us say Alice and Bob) prior communication. Alice encrypts the message and sends the ciphertext to Bob. Bob uses the same key to decrypt the ciphertext. As the third party (let us say Eve) does not have the key, Eve only sees the ciphertext and cannot

get the original message. This encryption technique is known as *Symmetric Cryptography*. In an Internet system, hundreds of thousands of entities want to achieve secure communication. In this infrastructure, the problem is to share the key between two entities. The problem is solved using *Public Key Cryptography*, a technique that uses two keys (i) *public key*: is publicly available and (ii) *private key*: only to the key holder. Now, Alice uses Bob's public key to encrypt the message and sends it to Bob. Only Bob can use his private key to decrypt it. Eve cannot decrypt the message as he does not know Bob's private key. This technique is also known as *Asymmetric Cryptography*. When many entities communicate with each other, one crucial issue is to authenticate the entity to establish whether the entity is the real entity. In the network, client-server communication system, *certificate* is the medium that verifies the authenticity of the user and the certificate authority (CA) issues a certificate. Other users can ensure the identity of the user by checking the certificate information. The whole infrastructure is known as *Public Key Infrastructure*. There are several practical issues, for example, (i) translating the theoretical concept to the encryption-decryption programs by writing the correct code, (ii) installing the cryptographic application correctly, and (iii) configuring the system properly so that it can run the application successfully. The National Institute of Standards and Technology (NIST) regulates the security standard while the research community helps to verify the implementation of the cryptographic protocols. There are several cryptographic protocols available that work in different layers of the OSI model. Secure shell protocol (SSH), transport layer security (TLS) and internet protocol security (IPSec) work in OSI application layer, transport layer and IP layer respectively.

Asymmetric cryptography: The underlying concept of asymmetric cryptography is that of hard mathematical problems. The assumption is that the computation power of currently available computers cannot solve those hard problems. The RSA cryptosystem can be used to securely exchange a key from a client Alice to a server Bob. Alice randomly generates a key, encrypts it with the public key in Bob's certificate, sends this over the untrusted communications infrastructure to Bob, who decrypts the message with his private key. Diffie-Hellman requires Alice and Bob to agree to a

large public prime $\mathscr{P}$ and a public value g that is a primitive root of $\mathscr{P}$. Alice chooses a secret value $secret\_a$ and sends $Y\_A=g^{secret\_a}$ mod $\mathscr{P}$ to Bob; Bob chooses a secret value $secret\_b$ and sends $Y\_B=g^{secret\_b}$ mod $\mathscr{P}$ to Alice. Alice computes $(Y\_B^{secret\_a}$ mod $\mathscr{P}$ and Bob computes $(Y\_A^{secret\_b}$ mod $\mathscr{P}$ that is the shared key. This works because $(g^{secret\_a})^{secret\_b}$ mod $p = (g^{secret\_b})^{secret\_a}$ mod p.

Symmetric cryptography: Symmetric ciphers can be of two types. *Stream ciphers* are the simpler of the two and allow the encryption of arbitrary-length plaintexts. Stream ciphers work by generating a pseudo-random stream of data from a key, which is known as a key stream and is combined with the plaintext to yield the ciphertext. *Block ciphers* operate on fixed-size blocks of data and consequently plain text must be padded to fill a multiple of the block size.

2.5.2 Transport Layer Security (TLS)

In this section, we describe the TLS protocol. TLS consists of two protocols arranged in two separate layers. The TLS Record Protocol is placed directly on top of the transport protocol, namely TCP. The TLS Handshake Protocol and the application data are placed on top of the TLS Record Protocol. Figure 2.4 shows the layered protocols.

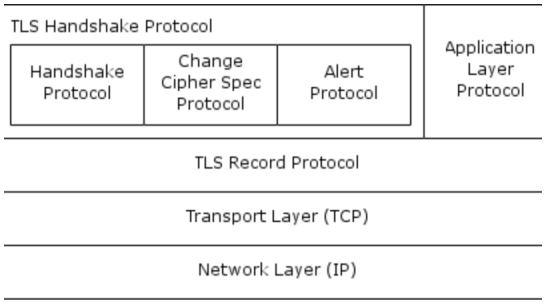


Figure 2.4: TLS protocols- hand shaking protocol and record protocol

Transport layer security (TLS) and its predecessor secure sockets layer (SSL) protocols secure the transport layer communication in OSI model. TLS aims to provide three main properties, authenticity, confidentiality and integrity and does so using both symmetric and public key cryptography. In RFC 5246[11], the goals of TLS are listed as below:

(i) *Cryptographic security:* TLS should be used to establish a secure connection between two parties.

(ii) *Interoperability:* Programmers should develop TLS applications in such a way that the cryptographic parameters can be utilized without knowing each other's code.

(iii) *Extensibility:* TLS framework should be extensible so that it can incorporate new public key and bulk encryption methods as required. The objective is to prevent introducing new protocol that can be vulnerable to threats and to avoid implementing an entire new security library.

(iv) *Relative efficiency:* Public key operations and other cryptographic operations are highly CPU intensive. TLS protocol has a feature to cache a session optionally that helps to decrease the number of connections that need to be established from scratch. This way it achieves efficiency.

Properties of TLS

(i) Origin authentication

Authentication is the process that verifies the clients and servers' identity. In TLS, authentication takes place outside of the scope of the protocol and relies on external public key infrastructure (PKI). Before the configuration of a TLS server, an X.509 certificate containing identity information and public key(s) must be generated. This certificate must then be digitally signed by a certificate authority (CA) who is a third party trusted to validate the details of certificates. The resulting signed certificate can then be sent to clients as part of the protocol handshake, where the client independently verifies that the certificate has been signed by a CA they trust.

(ii) Message confidentiality

Confidentiality is a property of a system that ensures message content is only readable by the intended recipients. In TLS, this is implemented using symmetric cryptography, for example, using the Rijndael cipher that is the basis of the Advanced Encryption Standard (AES), in conjunction with a key (or secret) known only by the client and server. In the majority of applications a client and server do not possess a pre-shared key (PSK), so a key exchange algorithm must be used to share a key across an untrusted communications channel. Currently, the key exchange algorithms

supported by TLS use public-key cryptography based upon the integer factorization problem (e.g., Rivest-Shamir-Adleman (RSA)) and the discrete logarithm problem (e.g., Diffie-Hellman).

In TLS RFC 5246, The representation of all data items is explicitly specified. The basic data block size is one byte (i.e., 8 bits). Multiple byte data items are concatenations of bytes, from left to right, from top to bottom. It also gives the vector size and other relevant data structures.

(iii) Message integrity

In recent versions of TLS, strong hash functions, i.e., those where collisions are computationally difficult to find are used in conjunction with a keyed-hash message authentication code (HMAC), to simultaneously provide message integrity and authentication. As TLS aims to be a future-proof protocol, the sets of algorithms that implement the cryptographic primitives mentioned above (known as cipher suites) are fully configurable and are standardized by the Internet Assigned Numbers Authority (IANA).

TLS Protocol Details

(i) Handshaking protocol

Figure 2.5 shows the anatomy of the handshake for the TLS v1.2 protocol and the corresponding RFC code standard, where optional messages are shown with dashed arrows. The handshake protocol works as follows:

ClientHello - this message initiates a connection from the client and contains the highest SSL/TLS version supported, a session ID, a list of cipher suites and compression methods. As of TLS v1.2, the contents of this message may include a list of TLS feature set extensions supported by the client.

ServerHello - the server responds with a similar message, specifying the cipher suite chosen for communication, and may list the subset of the TLS extensions it supports.

SupplementalData (optional) - if the client supports an appropriate TLS extension, the server may send arbitrary application-specific data.

Certificate (optional) - the server may send an X.509 identity certificate to the client.

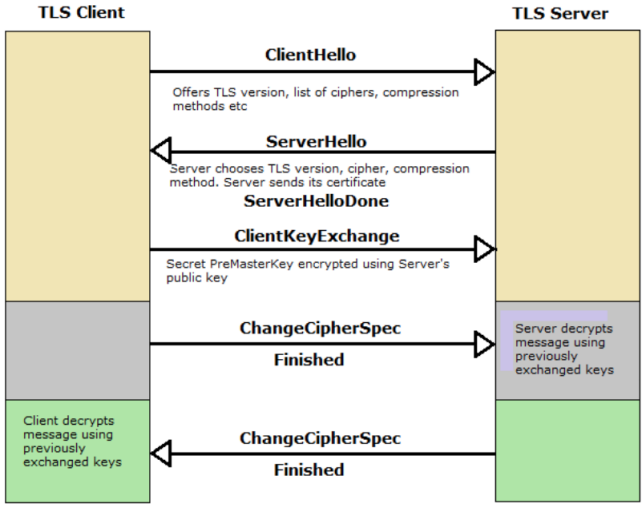


Figure 2.5: TLS handshake protocol

ServerKeyExchange (optional) - if the cipher suite specifies a key exchange algorithm that requires server interaction, such as Diffie-Hellman, the server sends the data in this message.

CertificateRequest (optional) - if the server requires mutual authentication, this message is sent to request an X.509 certificate from the client.

ServerHelloDone - this message signals the end of the server's half of the handshake.

SupplementalData (optional) - the client may then send arbitrary data to the server for application-specific purposes.

Certificate (optional) - if a CertificateRequest was received, the client sends an X.509 identity certificate to the server.

ClientKeyExchange - the client either initiates (in the case of RSA) or completes (in the case of Diffie-Hellman) a key exchange with this message.

CertificateVerify (optional) - the client verifies that the server possesses the private key associated with the public key from its X.509 certificate.

ChangeCipherSpec - this message signals that all further messages from the client to the server are encrypted under the shared secret.

Finished - this message contains a hash of all previous messages received, to ensure that the handshake was not manipulated by a third party.

Subsequent ChangeCipherSpec and Finished messages are then sent from the server to the client. If the plaintext of the Finished message matches the locally-generated hashes, the handshake is complete and encrypted application data can be exchanged.

(ii) Record Layer Protocol

The TLS Record Protocol provides connection security to higher layer protocols, with two basic properties, confidentiality and data integrity. The protocol takes messages that are to be transmitted, fragments the data into manageable blocks (16384 bytes or less), optionally compresses the data, applies a MAC, encrypts the block and finally sends it to the receiver (Figure 2.6). A received message is decrypted, verified, decompressed and reassembled before being delivered to the higher layer protocol.

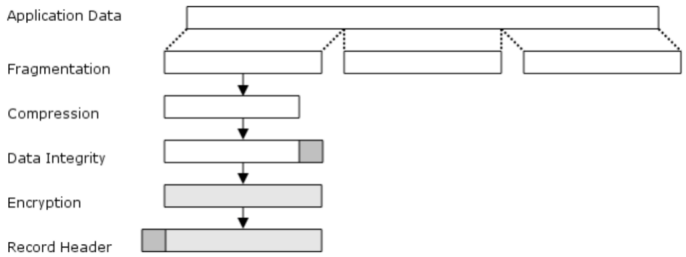


Figure 2.6: TLS record layer protocol

2.5.3 TLS code vulnerabilities

OpenSSL TLS implementation has around 70000 lines of code [19]. It is a complex software with many options and capabilities that make it vulnerable to misconfiguration and a wide range of attacks. Vulnerabilities of SSLv2 have been used [21] to design an attack that can decrypt TLS 1.2 handshakes using 2048-bit RSA in under 8 hours and at the cost of $440, on Amazon EC2. SSLv3 is also broken [19]. The recent TLS implementation by Amazon was vulnerable to a timing attack in the case of CBC-mode cipher suites, which could be extended to complete plain text recovery in some settings.

2.6 Shamir Secret Sharing Scheme.

Shamir's Secret Sharing (SSS) scheme [87] splits a secret D into n shares such that using at least t ($t \leq n$) shares can reconstruct the secret, and any $t-1$ or fewer shares do not reveal any information about the secret. The scheme uses polynomial interpolation over a finite field F_p. Let the secret $D \in \mathscr{D}$ be an element of F_p, and $t \leq n \leq p$.

Share Construction: To construct the n shares of D for a (t,n)-SSS, the following steps are taken.

1. Choose $t-1$ random positive values a_1, a_2, ... $a_{t-1} \in F_p$, and use them to construct the share distribution polynomial,

$$f(x) = D + \sum_{i=1}^{t-1} a_i \times x^i$$

2. The j^{th} share $share_j$ is obtained for random values of x. We denote the values of x as x_j where $j = 1,2,..,n$.

$$share_j = f(x_j) = D + \sum_{i=1}^{t-1} a_i \times (x_j)^i \mod p$$

Secret Reconstruction: Given t shares, $(x_j, share_j)$ where $j = 1,2,...,t$.

1. Lagrange interpolation is used to obtain $f(x)$. That is,

where $l_j(x)$ are Lagrange basis polynomials.

2. The secret is $f(0)$.

Let $X = \{(share_j), j = 1 \cdots (t-1)\}$ denotes the set of $t-1$ shares. Then it can be proved that,

$$H(D|X) = H(D)$$

and the secret remains perfectly secure.

Here $H(X) = -\sum p_i \log p_i$ is the entropy function.

$$f(x) = \sum_{j=1}^{t} share_j l_j(x)$$

2.6.1 Application of SSS

Threshold cryptography [38] is a well-studied problem for almost 40 years now. Shamir Secret Sharing Scheme is one of the threshold cryptosystem. However, the applications are mostly in cloud distributed storage servers [70], [44], [90], authentication systems [56], ad-hoc networks for key management [99], to share the key for RSA and ECC in MANET [46], threshold digital signature systems [51], [81], threshold signature schemes [37], [55], electronic voting [34] and security against malicious administrators in SDN [74]. *These problems are different from what is considered here.*

Multi-path routing and secret sharing were used to secure VoIP communication [78] where secret sharing was applied to the output of different speech compression coding and in a MANET [73]. In [72], data confidentiality has been provided in MANET (Mobile Adhoc Networks) using (t,n) Shamir Secret Sharing. The message is broken into n shares, encrypted and then transmitted through n node-disjoint paths. But the application of threshold cryptography in practical network communication was limited due to the inflexibility of network structure. The recent advancement in network virtualization, cloud, and software-defined networking opens a possibility of applying those security models. It is now relatively easier to manage and design the network, for example, providing disjoint paths. In [66], it is suggested to apply threshold cryptography in providing security for OpenFlow channel. Motivated by these combinations, our effort is to propose a practical solution using the well-known Shamir Secret Sharing in a controlled network within SDN to achieve confidentiality and reliability against quantum attacks.

2.6.2 Challenges

There are several challenges to design a protocol for network communication using Shamir Secret Sharing. The first is that the scheme assumes that the t shares are in possession of t different parties. In communication, it can be achieved by using disjoint paths between sender and the receiver. Now, the current protocols are designed using one source address and one destination address that can

travel using multiple paths. But, to design a protocol that uses SSS correctly, the multiple paths should be disjoint. Path-disjointness is a well-studied problem in network fault, quality of service and reliable routing. Paths between a given pair of source and destination nodes in a network are called *link disjoint* if they have no common (i.e., overlapping) links, *node disjoint* if, besides the source and destination nodes, they have no common nodes and *end-host disjoint* when source host and destination host have multiple interfaces that are connected through multiple paths (IPv6 multiple interface concept). The motivation behind path disjointness is to provide reliability in the communication network for several reasons, mostly, node failures or adversarial attacks. This problem and the algorithm was first proposed in 1967 by Frisch [50] where the algorithm finds the smallest disconnecting set between two given nodes. Figure 2.7 illustrates a case for traditional communication and SSS communication. In the traditional communication, there are multiple paths available between the source and the destination. A routing algorithm chooses a path and sends messages through it. If one path fails, the backup path can be used. For SSS, it creates shares and sends them along each path.

The second challenge is designing a protocol that uses multiple disjoint paths but can send and receive from a single source or destination. In this case, the hosts may have multiple addresses for the sender and the receiver. The sender or the receiver should be able to identify t shares of the same message.

The third challenge is to motivate a threat model where the adversary has a partial view of the network. Because, if the adversary has access to t paths out of n paths, they can reconstruct the message. Also if they have access to $(n-t)$ paths, they would be able to block the communication thus making it unreliable.

2.7 Moving Target Defense Technique

Moving target (MT) techniques frequently change the system parameters to lessen the probability of a successful *targeted attack* and limit the *span of an attack*. The technique creates *additional*

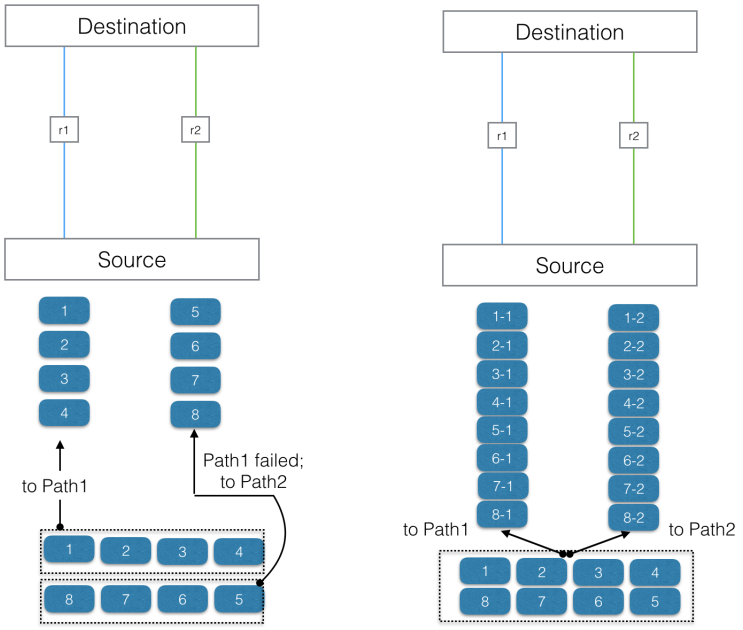


Figure 2.7: Traditional vs. SSS communication

uncertainty for attackers by dynamically changing system parameters that make systems less static and less deterministic to increase attackers workload. MTD techniques fall into five major categories [80] according to the software stack model illustrated in Figure 2.8: (i) dynamic networks, (ii) dynamic platforms, (iii) dynamic runtime environments, (iv) dynamic software, and (v) dynamic data.

We are interested in learning more about *Dynamic Networks* as our focus is to design a protocol for network communication. Techniques in the dynamic networks domain continuously modify network properties to increase the workload required by the attacker and reduce the probability of success for network-borne targeted attacks. Dynamic network modifications often focus on frequent changes to addresses and ports but might also include rotating protocols and changing the logical network topology. Dynamic network techniques are primarily intended to hinder attacker reconnaissance but might also play a role in preventing the launch of an attack. Such techniques can prevent attackers from discovering an exploitable condition or invalidate the results of a previous scan before an attack can be developed. If attackers are aware of an exploitable condition, the attack might be undermined if the target is constantly shifting to new locations.

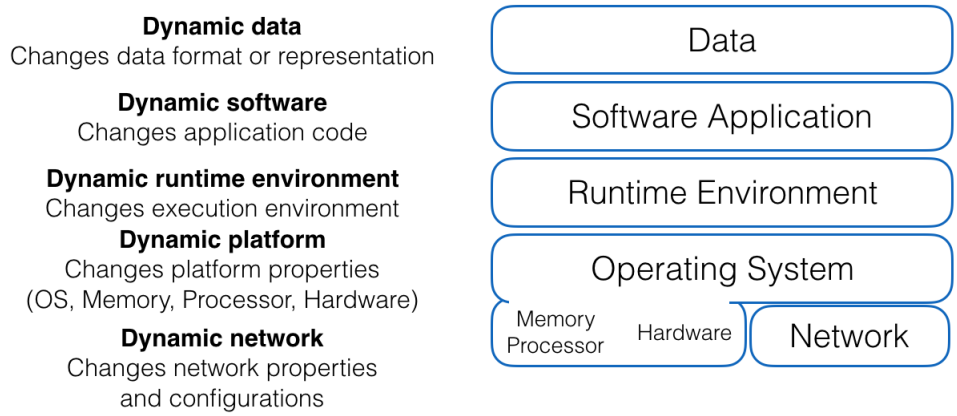


Figure 2.8: MTD techniques domains

Moving Target Defense techniques can be used to defend against many security threats. All the current techniques on MTD are summarized in [79] which also discusses its strengths and weaknesses. Random Host Mutation is a technique which is described in [17] and the same technique is applied to OpenFlow network [60]. MTD techniques can (1) alter the pathways to computers; (2) confuse attacks that rely on identifying the type of host, OS, or services; or (3) change the type of computer system; either continuously or after an attack is detected [96].

2.7.1 MTD - a potential approach for our design

In our design, the basic system provides confidentiality and reliability up to a sharp threshold number. If the attacker can exploit more than the threshold number of paths, the system loses its confidentiality and reliability completely. Our hypothesis is, using MTD techniques, the system can create a more randomized view to the attacker thus going beyond the sharp threshold that would restrict attacker to identify the target.

2.7.2 Challenges

There are several challenges in designing MTD using network domain.

Firstly, in the case, when services must remain reachable (for example, webserver), it can be challenging to apply MTD techniques that constantly changes the service's IP address or other

parameters. For the other services, MTD has a wider range of options.

Secondly, appropriate threat models are crucial for designing an MTD to understand its benefits and effectiveness.

Thirdly, it is also challenging to create the amount of uncertainty for attackers using randomization of network parameters. The uncertainty in a random value is measured by the number of all possible values and their probabilities. In dynamic network techniques, limited entropy is due to fixed infrastructure values, for example, IP address range or port numbers.

Fourthly, it is also challenging to find the effectiveness of the particular MTD techniques. One case that MTD could be useful is incremental attacks [47] where attacker requires the information from previous phases to launch successfully.

2.7.3 Related work on MTD Network Dynamic

Techniques for the dynamic randomization of network attributes [29]

Here, three MTD techniques are proposed in dynamic network context; (i) network randomization for TCP/UDP ports: in a distributed system model, it synchronizes time and at a specific time interval TCP ports are configured; (2) network randomization for IP addresses - the IP randomization module changes the IP of the networking device (for example, SDN switch) not the hosts thus making it transparent to the end hosts; (3) network randomization for network paths - the assumption behind this technique is the underlying network is a mesh topology that creates an overlay network and different paths are available between two communicating nodes. The system selects different paths/routes at some interval that restricts adversary from eavesdropping or Denial of Service attacks.

The threat model is defined as the attacker is inside the network and can successfully analyze traffic. The goal of the adversary can be launching D/DoS, reconnaissance, targeting a *specific* service, or targeting a *specific* host on the network. The MTD system's goal is to restrict an attacker to learn the real IP address and port numbers of the services to limit the scope of *targeted* reconnaissance attacks.

The performance costs for TCP port randomization and IP randomization are bandwidth counts captured by the network performance software. For route randomization, the costs are Round Trip Time, bandwidth and data transfer times captured by IPerf tool. Randomization interval for port and IP are 1 sec and 30 sec respectively. The results show that port randomization has least performance impact, IP randomization has the greater impact while path randomization causes an additional delay for additional random hops.

Effectiveness of IP address randomization in decoy-based moving target defense [33]

The decoy based MTD puts a large number of fake virtual nodes to prevent an attacker to locating or targeting real nodes. The approach uses randomizing IP addresses and assign them to real and decoy nodes. Refreshing IP addresses disrupts the TCP/IP that relies on IP. In this work, the authors proposed a system that balances the attacker's probability of detection vs. the minimization of the number of disrupted connections. They modeled the interaction between the node and the adversary (attack model) in which the adversary's goal is to detect a real node in minimum possible time. Then they formulated the problem of when to randomize IP based on the tradeoff between reducing the probability of detecting a real node vs. the minimum disruption.

The experiments are on (i) time for the adversary to scan a node, (ii) ability of the adversary to detect a real node, (iii) scanning capability of the adversary, (iv) number of connections lost due to randomization.

Random host mutation for moving target defense [18]

IP randomization techniques can be beneficial to deceive attacker, but the tradeoff is with the service reachability in IP network. The proposed technique balances between the impact of changing IP addresses and restricting an attacker by providing a transparent IP mutation at the end host. The mutation of the IP address is unpredictable but has less performance and manageability issues.

The hosts are assigned to a temporary routable virtual IP (vIP) that changes randomly and synchronously over time. The protocol design has several constraints, (i) assigning address from an unused address space, (ii) using the used address in a time interval, (iii) shared addresses for

a different network range. One system keeps the mapping of virtual IP and real IP. The protocol allows two ways to communicate the MT hosts; hostname or host rIP. The experiment's overhead metrics are sddress space overhead and (ii) routing and firewall updates overhead.

OpenFlow random host mutation [60]

The technique is similar to [18], but the application is in a software-defined network setup. The implementation of this technique requires two major components: (1) subnet gateways to perform rIP-vIP translation, and (2) a central management authority which co-ordinates mutation across the network. SDN controller: 1) coordinates mutation across OpenFlow switches based on host mutation requirements and available unused address space, (2) determines optimal set of new vIPs for hosts using SMT , (3) manages active connections by installing flows in OpenFlow switches along with required address translations actions, and (4) handles DNS updates. Each OpenFlow switch performs the vIP-rIP translations as specified by the controller.

2.8 Summary

We summarize that the underlying network has multiple paths available, but due to the current protocols assumption on single path communication and inflexibility of path management, the additional paths are unused. Recent technology trends, for example, virtualization, software-defined networking opens a possibility to manage paths effectively. Shamir secret sharing was widely proposed for distributed storage management, authentication and mobile ad-hoc networks (MANET), but not in a communication system. MTD techniques are useful for incremental attacks where an attacker uses previous phase's information. The network dynamic techniques changes system parameters, for example, IP, port, routes and creates a randomized view to the attacker that lessen the probability of successfully targeting a specific communication.

Chapter 3

Software-defined Networking

3.1 Introduction

This chapter provides background on Software-defined Networking and discusses the difference between the traditional network, its components, real deployments, security and reliability issues.

3.2 Software-defined networking

This section gives an overview of the SDN components, network architecture, and the real deployments. Our main objective is to understand the switch and the controller communication structure.

3.2.1 SDN vs. the Traditional Network

The development of Software-defined networking is promoted by Open Networking Foundation (ONF) through open standards development. Four pillars of SDN are, (i) the control and data planes are decoupled, (ii) forwarding decisions are flow-based, instead of destination-based, (iii) control logic is moved to an external entity i.e. SDN controller, and (iv) the network is programmable through software applications running on top of the controller that interacts with the underlying data plane devices. In their release of 'SDN architecture' [14], SDN is defined as *"The aim of SDN is to provide open interfaces that enable the development of software that can control the connectivity given by a set of network resources and the flow of network traffic through them, along with possible inspection and modification of traffic that may be performed in the network. These primitive functions may be abstracted into arbitrary network services, some of which may not be presently apparent"*. SDN has several components, (i) *Data Plane* is responsible for traffic forwarding, and corresponds to the switching and routing part of the traditional network. (ii)

Control Plane is a collection of one or more controllers. The controller decides the network flow and installs forwarding rules of the data-plane. (iii) *Application Plane* comprises of one or more networking applications. Such applications may include switching, firewall, intrusion detection applications and provide freedom to control the network. (iv) *Interfaces*: SDN requires two interfaces; one is the *northbound interface* between the application plane and the control plane, the other one is the *southbound interface* between the control plane and the data plane. (v) *Control Network* is the network between the control plane and the data plane that consists of switches or routers to propagate events [59]. Control networks for SDNs may take any form, including a star (a single controller), a hierarchy (a set of controllers connected in a full mesh, which connects to forwarding nodes below), or even a dynamic ring (a set of controllers in a distributed hash table) [57]. Figure 3.1 depicts a typical SDN architecture. We discuss in details as bottom-up:

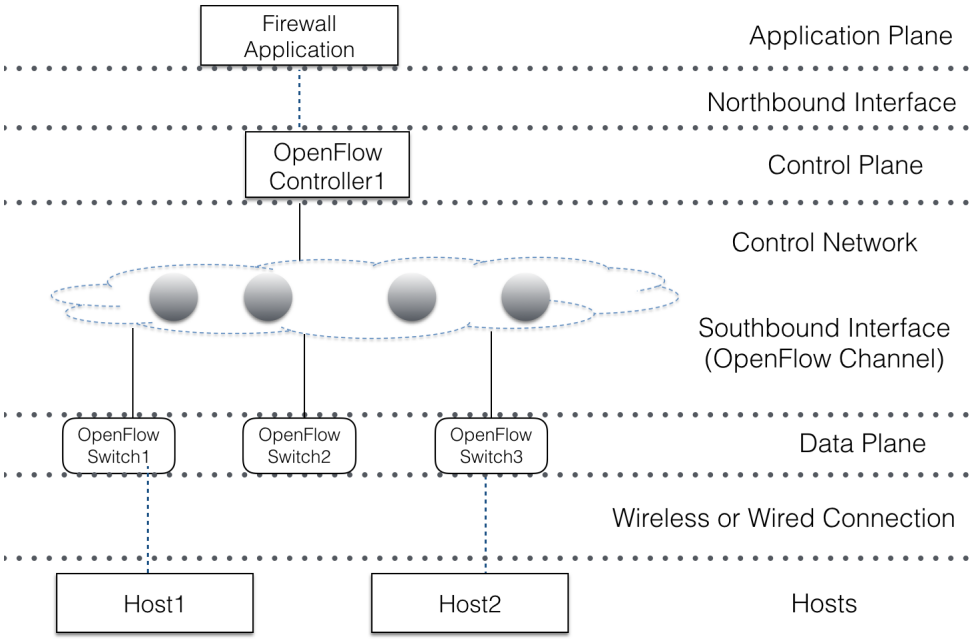


Figure 3.1: An SDN architecture

Data Plane

The data plane consists of a set of one or more network elements (switches, routers), that contains a set of traffic forwarding or traffic processing resources. The main difference resides in the fact that those traditional switches or routers are now only forwarding devices without embedded

control functions. The most important feature is that these devices are now built on using open and standard interfaces (e.g., OpenFlow). The standard interface is crucial to ensuring configuration and interoperability among different data and control plane devices. In the traditional network, the absence of the open interface made it difficult to operate heterogeneous devices. The data plane device can be hardware or software. These devices are called *OpenFlow switch*, and the standardization is being maintained by the Open Networking Foundation (ONF). Logically the device has a *Data path* (flow table, group Table), a communication channel and ports as depicted in the Figure 3.2. The tables perform lookups and forwarding, and the communication channel connects to the external controller. OpenFlow ports are the interfaces for sending OpenFlow packets between two OpenFlow switches. The packets go out using egress port of one switch and get in using ingress port of another switch. An OpenFlow port must support three types of OpenFlow ports namely *physical ports*, *logical ports* and *reserved ports*. OpenFlow physical ports correspond to the hardware interface of the switch, for example, Ethernet port of an Ethernet switch. The OpenFlow logical ports do not associate directly with a hardware interface. These are higher level abstractions that may be defined using non-OpenFlow methods (e.g., tunnels, loopback interfaces). The OpenFlow reserved ports are responsible for generic forwarding actions such as sending to the controller, flooding, or forwarding using non-OpenFlow methods (normal switch processing). some are *required* and some are *optional* where an OpenFlow switch must support the required ones.

OpenFlow enabled devices are available on the market as commercial or open source products. These are available as hardware from Hewlett-Packard, Huawei, Juniper, NetFPGA, IBM, NEC. Software switches are emerging for data centers and virtualized networks. Open vSwitch [6] and Pica8 [8] are some of the software-based implementations.

Southbound Interfaces

Southbound Interfaces are between the OpenFlow switch and the OpenFlow controller. It is also known as *control channel* or *OpenFlow channel*. The controller configures and manages the switch through this interface. The switch sends packets and the controller collects event statistics

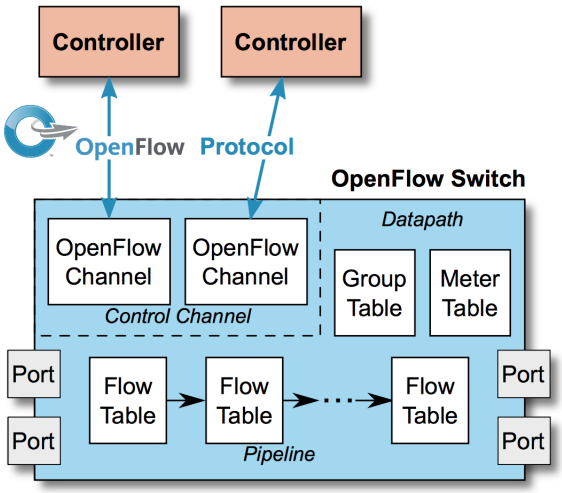


Figure 3.2: Components of OpenFlow switch

also. OpenFlow [75] protocol is the most widely accepted and deployed standard for SDN. There are other protocols that are developing, for example, ForCES [43] andOpen vSwitch Database (OVSDB) [82]. The OpenFlow switch establishes an OpenFlow channel with the controller using TCP/IP connection. OpenFlow switch can support single connection to a single controller or multiple connections to multiple controllers. It is suggested to use TLS to provide secure communication. The OpenFlow channel transmits OpenFlow messages between the OpenFlow controller and the OpenFlow switch that maintains the format specified in OpenFlow switch specification [49]. There are three types of OpenFlow messages passing through OpenFlow channel;

(1) *Symmetric* messages are initiated by either the switch or the controller and sent without solicitation (Hello, Echo Request, Error, and Experiment)

(2) *Controller-to-switch* messages are initiated by the controller and used to manage or inspect the state of the switch directly (Handshake, switch config, Flow table config, Modify state, Packet-out, Barrier, Role Request, Multiplier, Bundle, Set Asynchronous config) and

(3) *Asynchronous* messages are initiated by the switch and used to update the controller of network events and changes to the switch state.

OpenFlow messages carry sensitive information, for example, data plane ID (MAC address,

VLAN ID), controller ID, maximum number of packets that switch can buffer, number of tables supported by the switch, the type of connection from switch to the controller, flow statistics, table statistics, port statistics, group statistics, queue statistics, commands to add, delete, modify flow entries, current configuration, and statistics. The structure of OpenFlow message is defined in the OpenFlow switch specification document. We analyze one message. Figure 3.3 shows the structure of the OpenFlow packet-in message.

```
/* Packet received on port (datapath -> controller). */
struct ofp_packet_in {
    struct ofp_header header;
    uint32_t buffer_id;     /* ID assigned by datapath. */
    uint16_t total_len;     /* Full length of frame. */
    uint8_t reason;         /* Reason packet is being sent (one of OFPR_*) */
    uint8_t table_id;       /* ID of the table that was looked up */
    uint64_t cookie;        /* Cookie of the flow entry that was looked up. */
    struct ofp_match match; /* Packet metadata. Variable size. */
    /* The variable size and padded match is always followed by:
     *   - Exactly 2 all-zero padding bytes, then
     *   - An Ethernet frame whose length is inferred from header.length.
     * The padding bytes preceding the Ethernet frame ensure that the IP
     * header (if any) following the Ethernet header is 32-bit aligned.
     */
    //uint8_t pad[2];       /* Align to 64 bit + 16 bit */
    //uint8_t data[0];      /* Ethernet frame */
};
OFP_ASSERT(sizeof(struct ofp_packet_in) == 32);
```

Figure 3.3: Structure and size of packet-in message

Packet-in messages are received by the switch and sent to the controller. Data type *uint8_t*, *unit16_t*, *unit32_t*, and *unit64_t* are 8, 16, 32 and 64 bits respectively. The total size of this message is *sizeof (struct ofp_packet_in)* that equals to 32 bytes. The header is of size 8 bytes and included in the OpenFlow message. The header contains 8 bits version, 8 bits type, 16 bits length and 32 bits transaction ID. Transaction ID is associated with this packet and replies use the same ID as in the request to facilitate pairing. Figure 3.4 shows the structure of the header.

Table 3.1, 3.2 and 3.3 shows the *controller to switch*, *Asynchrounous* and *Symmetric* types of OpenFlow messages and their purposes in details respectively.

OpenFlow controller

The controller plane comprises a set of SDN controllers, each of which has exclusive control over a set of resources exposed by one or more network elements in the data plane (its span of

```
/* Header on all OpenFlow packets. */
struct ofp_header {
    uint8_t version;    /* OFP_VERSION. */
    uint8_t type;       /* One of the OFPT_ constants. */
    uint16_t length;    /* Length including this ofp_header. */
    uint32_t xid;       /* Transaction id associated with this packet.
                           Replies use the same id as was in the request
                           to facilitate pairing. */
};
OFP_ASSERT(sizeof(struct ofp_header) == 8);
```

Figure 3.4: Structure and size of the header of OpenFlow message

Table 3.1: Information that controller-to-switch OpenFlow message carries

Direction	Purpose	Details
C $\rightarrow$ S; S $\rightarrow$ C	Features	Request the identity and basic capabilities of a switch
C $\rightarrow$ S; S $\rightarrow$ C	Configuration	Set and query configuration parameters
C $\rightarrow$ S	Modify-state	To add, delete and modify flow/group entries in and to set switch port properties.
C $\rightarrow$ S; S $\rightarrow$ C	Read-state	Collects current config, statistics, and capabilities of switch
C $\rightarrow$ S	Packet-out	Packet-out messages contains a full packet or a buffer ID referencing a packet stored in the switch
C $\rightarrow$ S; S $\rightarrow$ C	Barrier	To ensure message dependencies have been met or to receive notifications for completed operations.
C $\rightarrow$ S; S $\rightarrow$ C	Role-Request	To set the role of its OpenFlow channel, controller ID or query.
C $\rightarrow$ S; S $\rightarrow$ C	Asynchronous-Configuration	To set an additional filter on the asynchronous messages that it wants to receive on its OpenFlow channel or to query that filter.

control). Traditional operating systems provide and manage the concurrent access to lower level devices and resources (CPU, hard drive, network adapters and memory) and also enable security protection. These functionalities provide abstractions, standard application programming interfaces (APIs) to developers and lead to evolving developer's programming languages and different applications. In contrast, the network operating system is a set of device-specific, lower level instruction sets, and closed in nature.

SDN's promise is to enable abstractions and APIs in Network Operating Systems (NOS). A developer does not need to worry about low-level details in data plane to define network policies. The NOS is also called a controller. The NOS is responsible for network configurations based

Table 3.2: Information that asynchrounous OpenFlow message carries

Direction	Purpose	Details
S $\rightarrow$ C	Packet-in	Transfers the control of a packet to the controller. It can also be configured to buffer packets.
S $\rightarrow$ C	Flow-removed	Informs controller about removal of a flow entry from a flow table.
S $\rightarrow$ C	Port-status	Informs controller of a change of its role. When a controller elects itself as master, the switch sends role-status message to former.
S $\rightarrow$ C	Controller-status	When an OpenFlow channel status changes, the switch sends messages to all controllers.
S $\rightarrow$ C	Flow-monitor	Informs controller of a change in flow table

Table 3.3: Information that symmetric OpenFlow message carries

Direction	Purpose	Details
C $\rightarrow$ S; S $\rightarrow$ C	Hello	Exchanged between switch and controller upon connection
C $\rightarrow$ S; S $\rightarrow$ C	Echo	To check the liveliness of a controller-switch
C $\rightarrow$ S; S $\rightarrow$ C	Error	To notify problems to the other side of the connection
	Experimenter	For future OpenFlow revisions

on policies and abstractions for connecting and interacting with forwarding devices. Some of the controllers are ONIX [65], OpenDayLight [7], HP SDN controller [11], Juniper openContrail [4], Beacon [45], POX [13], NOX [53] and Floodlight [3]. Figure 3.5 shows the OpenDaylight SDN controller architecture.

3.2.2 The Traversal of a Packet in an SDN

The objective is to understand OpenFlow channel communication structure that helps us in our protocol design.

The SDN applications run on top of the controller. Hub, L2_learning, L3_learning, firewall are some examples of SDN applications. OpenFlow switch flow table can be configured by these applications through the controller. Let us take the example of a firewall. The firewall application installs OpenFlow rules that ALLOW or DROP flow. In SDN architecture, the packet filtering is

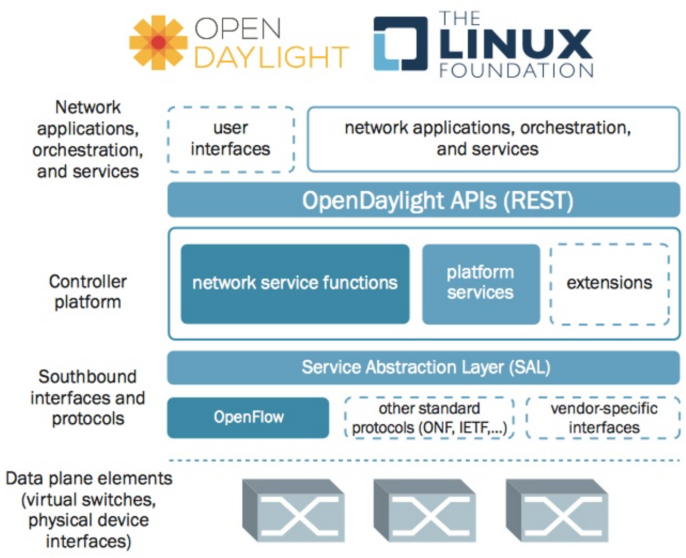


Figure 3.5: OpenDaylight SDN controller architecture [10]

an application running on top of the controller whereas the traditional firewall is hardware based or software based, running inside the network elements. The benefit of this FW application is that it can install rules to many switch data planes through the controller. Here, we describe different OpenFlow message traversal for FW example.

In Figure 3.6 Alice is connected to the OpenFlow switch $s1$ and Bob is connected to the OpenFlow switch $s2$. Both switches are controlled by the controller. Functionalities like Layer2 learning and Firewall are available through applications that run on top of the controller. The process starts when Alice sends packets to Bob. We describe the packet traversal using the Figure 3.6: (1) when a packet comes in the port of OpenFlow switch $s1$ from Alice, it passes through the *flow table* of that switch, (2)The *flow table* contains the matching rule and the corresponding instructions with some other fields like counter, priority, timeouts, and cookie (Figure 1.3) . The rule match is against the header information for MAC, IP and TCP source and destination address/port. If no rule matches, an OpenFlow message (Packet-in) is forwarded to OpenFlow controller. (3) Now, the controller decides on the path and installs the rule in the Flow table by Packet-out message. (4) then, the packet and the subsequent packets follow the instruction and communication between Alice and Bob establishes. In this example, Packet-in and Packet-out are the OpenFlow message type.

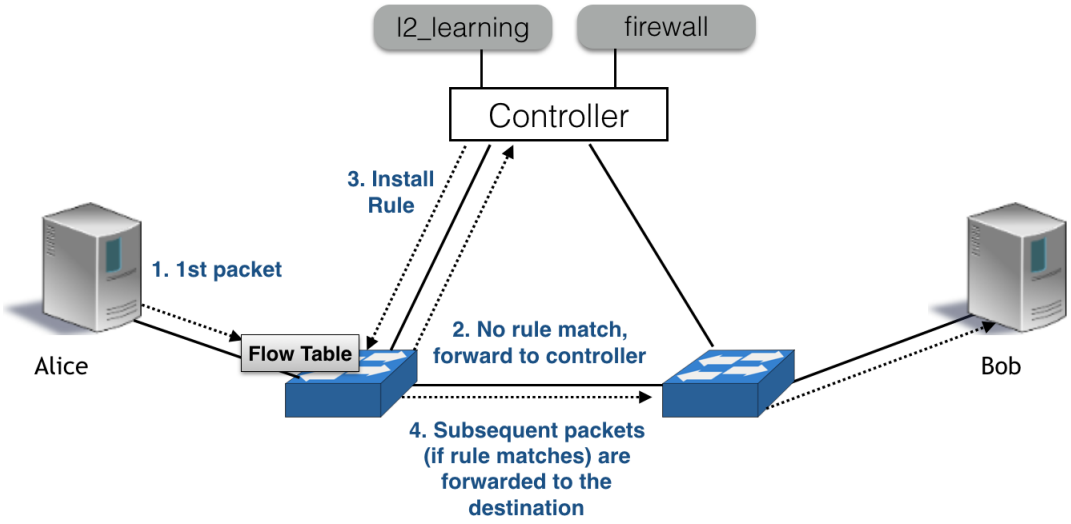


Figure 3.6: OpenFlow message traversal between controller and switch in SDN

3.2.3 The Network Topology between the Switch and the Controller

Our objective is to use *multiple paths* to provide secrecy and reliability to OpenFlow channel. We need to know the architecture of the controller and the switch placement and the network topology between the controller and the switch.

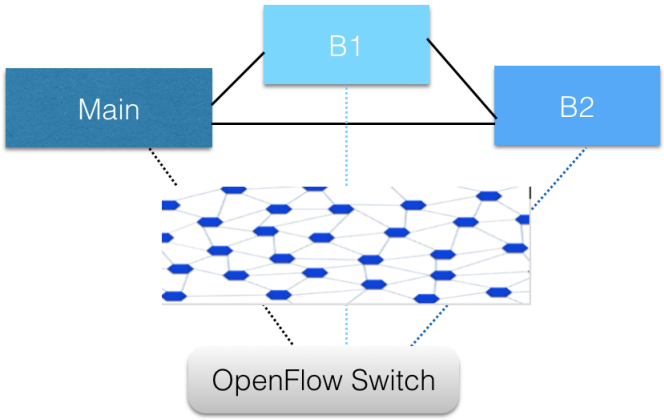


Figure 3.7: In WAN architecture controllers are located geographically distant location

In Wide Area Network (WAN) settings, controllers can be placed in geographically distant locations. In this case, the controller is attached to another network but has the capability of

controlling the remote switches. Figure 3.7 illustrates three controllers are remotely connected to an OpenFlow switch. In between the switch and the controller, there can be many networking nodes. This type of architecture was discussed in [57] and the authors introduced *The controller placement problem*. The problem deals with two questions; given a topology, how many controllers are needed? And where should those controllers go? The key points from the paper are, (i) the controller-switch architecture is closer to server-client architecture. (ii) control networks for SDNs may take any form, including a star (a single controller), a hierarchy (a set of controllers connected in a full mesh, which connects to forwarding nodes below), or even a dynamic ring (a set of controllers in a distributed hash table). (iii) controllers were used to referring to geographically distinct controller locations, as opposed to individual servers.

3.2.4 The Example of Real SDN Deployments

A real SDN architecture deployment case is Google Inter-Datacenter WAN [12], [61]. Google provides services to global users, such as Google Web Search, Google+, Gmail, Youtube, Google Maps. It requires a significant amount of data to be moved from one region to another, making these applications and services very WAN-intensive. Figure 3.8 shows the global view of Google's Datacenters.

Figure 3.8: Global view of Google WAN [12]

Google observed that the current technology is complex in management and configuration that is also an obligation to scaling network. Thus, they found SDN could be a prominent solution. They adopted SDN solution to achieve several objectives: (i) the deployment of any novel routing,

scheduling, monitoring, and management functionality and protocols are expected to be simpler, and the result would be a more efficient network, (ii) it would support standard routing protocols and centralized Traffic Engineering (TE), (iii) it is expected to have a dedicated, software-based control plane running on commodity servers which have simplified coordination and orchestration for both planned and unplanned network changes, (iv) low-cost routers built from merchant silicon was another objective, as the network should scale up with growth, and (v) another goal for the network is Centralized traffic engineering. It uses multipath forwarding to balance application demands across available capacity in response to failures and changing application requirements.

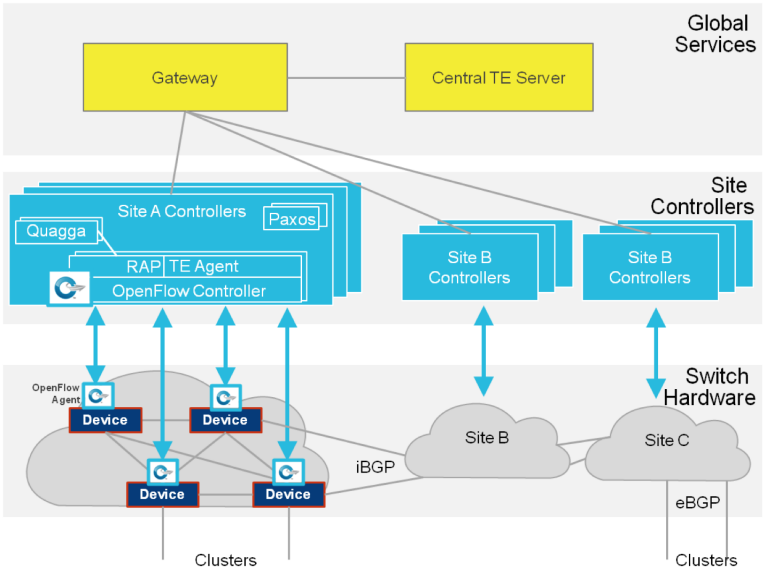


Figure 3.9: Google SDN architecture [12]

Figure 3.9 shows a closer look at the SDN architecture of Google. If we simplify the network diagram, we can see there are three layers. (i) OpenFlow enabled data-plane consisting of switching devices (ii) OpenFlow-enabled controller (iii) Application layer consisting of switching, filtering applications.

3.2.5 Summary

We summarize that the Control plane and the southbound interface is the new addition in SDN compared to the traditional network. The widely accepted southbound protocol is OpenFlow protocol.

The protocol transfers OpenFlow message, and we categorize them as follow :(i) No response required: One that is initiated by the controller or the switch and sent to the switch or controller without waiting for any reply, and (ii) Response required: One that is initiated by the controller or the switch and sent to the switch or controller and also waiting for any reply. The OpenFlow switch establishes an OpenFlow channel with the controller using TCP/IP connection, and vice-versa. The controllers can be placed in a geographically distant location or one data center. There can be a network between the switch and the controller that is referred as *Control Network*. Google has deployed SDN in their WAN architecture to gain faster performance and easy operability.

3.3 Using SDN to Improve Security of Traditional Networks

SDN adds flexibility in network management and unifies flow control. The overall structure itself improves the security of the traditional networks. We categorize the security applications in following categories:

3.3.1 Prevention

A traditional network employs many security tools and techniques in the perimeter to prevent its inside network from malicious attacks. One technique is to filter out the malicious traffic. The filtering can be done using firewalls. As an SDN controller can generate and view a full topology view, a firewall can be deployed as an application on top of the controller instead of a hardware box from the traditional network. The tradeoff is with the efficiency, but researchers are providing many solutions. One challenge in the traditional network is that middleboxes, for example, FW (firewalls), gateways, IDS (intrusion detection systems), proxies and IPS (intrusion prevention systems), play a critical role but are highly complex in the configuration such as correct rule creation using network view. In [85] a proposal was made to use SDN to simplify the middleboxes. A network-wide virtual firewall is proposed using SDN [23] that enforces security policies for internal network communication and outgoing-incoming communications. Intrusion Prevention

Systems monitor the system, identify intrusions, attempt to block and report it. SnortFlow [95] is an Open-Flow based IPS that provides IPS for a cloud system.

3.3.2 Detection

IDS (intrusion detection systems) monitor network, detect suspicious activities and report it. They cannot take further actions to prevent them. The detection can be exact rule-based, pattern-matching based or statistical anomaly based. Four prominent anomaly detection algorithms were implemented in an SDN context that could identify malicious activities more accurately [76]. DoS flooding attack has been detected using SDN and selective packet inspection method on Global Environment for Network Innovations (GENI) [32].

3.3.3 Anonymity Privacy

Providing privacy is a crucial aspect in a cloud environment especially hybrid clouds that are composed of a private and a public cloud. SDN is used to create multiple paths between a private and a public cloud and provides information theoretic security by creating n shares of a message using a (t,n) Shamir secret sharing scheme, thus providing private anonymous interconnections [42].

3.3.4 Moving Target Defense

MTD introduces constantly adapting a system that prevents or delays network attacks. We have discussed MTD in detail in Section 2.7. SDN is a possible solution that can provide dynamic network MTD. In [60], the authors use OpenFlow to develop an MTD architecture that transparently mutates IP addresses with high unpredictability and rate, while maintaining configuration integrity and minimizing operation overhead. The presented technique is called OpenFlow Random Host Mutation (OF-RHM) in which the OpenFlow controller frequently assigns to each host a random virtual IP that is translated to/from the real IP of the host. The real IP remains untouched, so IP mutation is completely transparent to end-hosts.

3.3.5 Administration

Shamir secret sharing scheme has been used to provide security against malicious administrators [74].

3.4 Current Security Issues of SDN

In this emerging architecture, the security issues of traditional networks has appear in SDN too. But, it has some extra security issues as SDN has a new working protocol OpenFlow protocol [75]. As shown in Figure 3.10, there are at least seven identified threat vectors in an SDN architecture [66]. Our focus is to find and categorize the threats and the attacks that affect confidentiality and the reliability of the OpenFlow Channel. We describe some example attacks for each threat vector.

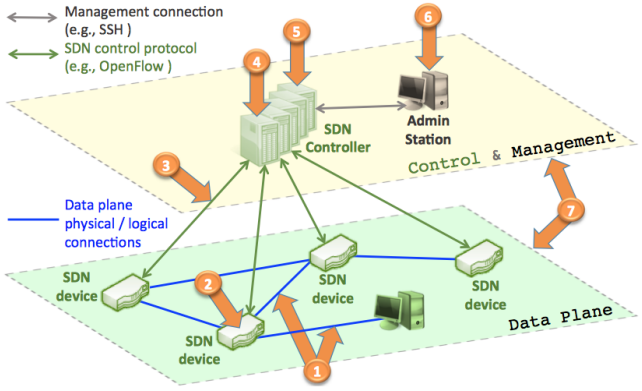


Figure 3.10: Threat vectors in SDN [66]

Threat Vector 1: The first threat vector is in the data plane traffic. The OpenFlow switches are connected to another OpenFlow switch, to the controller, or to the end host. This vector only considers the connectivity among the switches or the end host. The attackers can compromise an end host or an OpenFlow switch. The end host sends a packet with spoofed IP address, every time OpenFlow switch receives a packet that is not installed in the flow table it sends a packet-in message to the controller. The controller needs to reply with a packet-out message and installs the rule in the switch's flow table. If there are requests more than the switch or controller's processing

capability, it could create a bottleneck, and the service may drop. This type of attack may affect OpenFlow channel's availability directly if the controller or switch is exhausted.

Threat Vector 2: The second threat vector is in the data plane device's vulnerability, and the attacker is able to compromise data plane devices (OpenFlow switch). When the attackers have access to the OpenFlow switch, they are able to know the OpenFlow rules (Passive Confidentiality Attack) or modify and insert fake traffic flows (Active Integrity Attack). They can also exploit the vulnerability to block communication between the switch and the controller (Active Availability Attack). The compromised switch can be used to eavesdrop the OpenFlow channel (Passive Confidentiality Attack).

Threat Vector 3: This threat vector affects the OpenFlow Channel. The channel can be a wireless or a wired connection between the controller and the switch. An eavesdropper can target this channel to learn what message is passing through (Passive Confidentiality Attack), or change the content of the message, for example, Man-in-the-Middle attack (Active Integrity Attack) or the connection can be jammed or blocked to drop the communication (Active Reliability Attack).

Threat Vector 4: The fourth threat vector is in the control plane, and the attacker compromises one or more OpenFlow controllers. Since the controller computes all the network path computations and network views, the attacker can compromise the full or a partial network under that controller. A fake or forged packet-out message can be sent as a reply through the OpenFlow Channel (Active Integrity Attack). The attacker can directly launch an eavesdropping attack in the OpenFlow Channel (Passive Confidentiality Attack). It can also launch a DDoS attack to block the OpenFlow channel communication (Active Reliability Attack).

Threat Vector 5: The threat vector 5 is similar to the threat vector 3 except it is between the controller and the management stations. The network administrator uses this channel to connect and configure the controllers, for example, install network applications or configure the controller operating system. The attacker can launch an eavesdropping attack to know the configuration parameters (Passive Confidentiality Attack), or enforce forged data in the control plane (Active

Integrity Attack) or jam or block the communication (Active Reliability Attack).

Threat Vector 6: This threat is on the management plane when the attack is on the administrative station. The attacker can use this station to install malicious applications on the controller that can affect the whole or partial network. A malicious firewall rule might allow redirecting traffic to the attacker's domain (Active Integrity Attack). It can configure the controller such that the controller can send unnecessary messages to the switch making the OpenFlow channel blocked (Active Reliability Attack).

Threat Vector 7: The threat vector 7 deals with the lack of trusted resources for forensis and remediation. If there is no mechanism to protect the compromised data, device and activity logs, it would be difficult to investigate any incident and quickly restore the state. There should be a proper authentication and trust system as well as secure the activity log.

For instance, information disclosure can be achieved through side-channel attacks targeting the flow rule setup process. When reactive flow setup is in place, obtaining information about network operation is relatively easy. An attacker that measures the delay experienced by the first packet of a flow and the subsequent can easily infer that the target network is a reactive SDN, and proceed with a specialized attack. This attack - known as fingerprinting [88] - may be the first step to launching a DoS attack intended to exhaust the resources of the network, for example. If the SDN is proactive, guessing its forwarding rule policies is harder, but still feasible [63]. Interestingly, all reported threats and attacks affect all versions (1.0 to 1.3.1) of the OpenFlow.

3.5 Secure SDN Communication

In an SDN environment, the entities can be end hosts (computers, laptops, mobile phones, servers), networking devices (OpenFlow switches), controllers, control nodes (networking devices in control network between OpenFlow switch and OpenFlow controller) and the communication can be for any two of them. The Figure 3.11 shows the different interactions between entities in SDN. The attacker can plan passive attack first. In this attack model, attacker's objective is to collect

information by listening to the OpenFlow communication channel. If this channel is plain-text, the attacker can obtain switch id, controller id, OpenFlow packet-in message, table ID, and statistics. In the second phase, the attacker is able to plan an active attack consisting sabotage of the network. The attacker who already has gained access to the internal network is also vulnerable to APT (Advanced Persistent Threat). In this type of attack, attackers use *advanced* techniques to gain access to the network and *persistently* retain control over the system for a long time unnoticed [91].

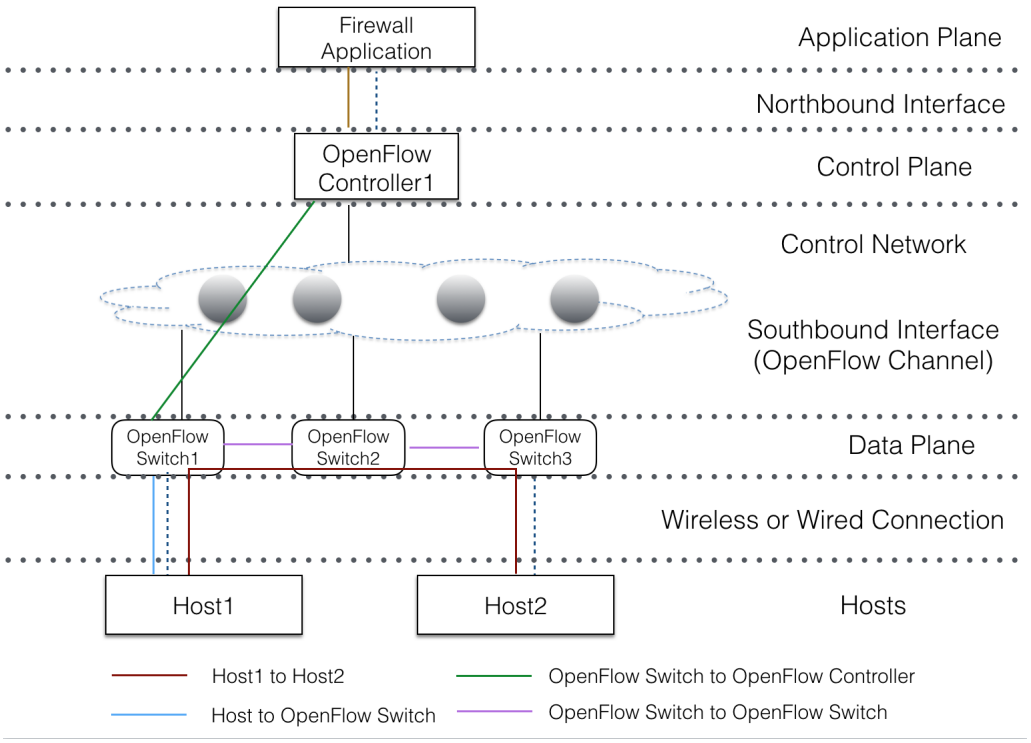


Figure 3.11: An example of SDN enterprize network

3.5.1 OpenFlow Secure Protocols

Security threats and attacks on SDN and OpenFlow protocols can be grouped into two, (i) those that are similar to traditional networks [86], [27], and (ii) those that are unique to SDN. In [66] seven threat vectors on SDN have been identified, the third of which is plain-text communication on the control plane. To secure this channel, TLS protocol is suggested in OpenFlow switch specification [49], [66], [27]. A solution for securing Software-defined Mobile Network uses Host

Identity Protocol along with Public Key Infrastructure and IPSec [71]. The protocol offers protection against TCP SYN Flood Attack, TCP Reset Attack, and eavesdropping. The solution requires an additional node *Security Gateway* between controller and switch that adds resource cost. Security of the northbound interface between application and controller, using TLS v1.2 has been considered in [24]. Identity-based cryptography has been used to provide security for southbound interface [68]. The solution uses the controller as a Trusted Authority (TA) to generate private keys for switches whose identity can be their MAC addresses. This protocol reduces system configuration, key establishment delay, and bandwidth consumption. *All above solutions use computational cryptography (public-key presumably) and will be insecure in the presence of a quantum computer.*

3.5.2 Reliability of SDN

In SDN the control plane is separated from the data plane which introduces a new problem. While many papers have acknowledged the issue of control link reliability [98], few of them suggested some solution in different settings. The authors of [59] described three parts of a control plane - functionality wise; (i) control platform - responsible for state distribution handling; (ii) control applications - an interface provided by the control platform for various programs; and (iii) control network (or connectivity infrastructure), for propagating events to switches or between multiple controllers. Here the importance of control link reliability has been discussed on the occasion of link or switch failure. The path is defined as the set of routers between controller and switch or among controllers. If each control path is viewed as a logical link, communication packets are transmitted over an overlay network, named control network, above the physical network. The work proceeds with assuming a dependent failure model where two overlaid control paths may share a common physical link or switch. Here, the network switches are grouped as multiple domains where multiple controllers are responsible for managing those switches. The topology is taken from [26] and the authors first formulated the resiliency problem between the controller and the switch in SDN architecture.

The proposals are on the occasion of link or switch failure, whereas our problem setting is for

the adversarial attack.

3.6 Summary

We summarize that software-defined networking provides flexibility in providing network management. SDN can be used to improve the traditional network's security although the separation of control and data plane introduces security threats for SDN itself. We have observed that the crucial addition is the southbound interface that transmits network configuration data between the controller and the switch. Currently, the only security suggestion is to apply TLS. Our study finds that there are multiple paths available between the controller and the switch and we use those to provide confidentiality and reliability of the OpenFlow communication using Shamir Secret Sharing and network dynamic MTD technique.

Chapter 4

Secure and Reliable OpenFlow Protocol

4.1 Introduction

In this chapter, we discuss the problem of secure and reliable OpenFlow protocol. We considered an SDN architecture that has a control network between the control plane and the switch plane. We use *multipath* and send shares of an OpenFlow message through n disjoint paths. The receiver can reconstruct the message from at least t shares. Shamir Secret Sharing scheme constructs and reconstructs OpenFlow messages and provides reliability and quantum-safe security. We modeled the adversary and their powers. We designed the first protocol `bQSafeOF` that takes into account OpenFlow protocol's parameters. The security analysis shows that the system's security is completely lost if the adversary captures t paths and reliability is lost if the adversary blocks $n-t+1$ paths. Our second system `mtdQSafeOF` uses a randomization technique along with `bQSafeOF`; we refer to the randomization as Moving Target Defense Technique. The security analysis shows that the probability of eavesdropping on a specific controller-switch communication is less than one when the adversary eavesdrops on t or more than t paths. The reliability also increases. We also provide an alternate solution to compare the performance with ours. The other solution provides confidentiality using TLS protocol and achieves the same level of reliability by adding n redundant paths. This solution is not quantum-safe. We call the system `notQSafeOF` (Not Quantum Safe OpenFlow).

4.2 Models and Assumptions

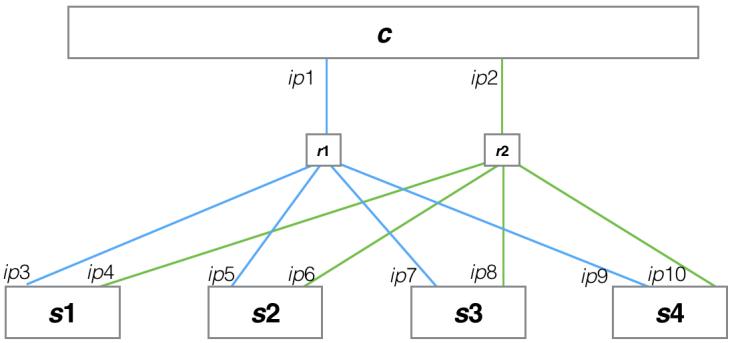
4.2.1 Goals

Our goal is to provide confidentiality and reliability of OpenFlow protocol. OpenFlow [75] protocol is one of the protocols that has been proposed particularly for the southbound interface. The protocol is widely used by the controller to manage switches, receive events from switches and send packets to switches. The OpenFlow switch establishes an OpenFlow channel with the controller using TCP/IP connection. We use multiple paths between the controller and the switch to provide confidentiality and reliability instead of using the primitive cryptographic key for ensuring security.

In practice, there is a control network available between the controller and the switch. It consists of switches or routers that propagate events [59]. Control networks for SDN may take many forms, including a star (a single controller), a hierarchy (a set of controllers connected in a full mesh, which connects to forwarding nodes below), or even a dynamic ring (a set of controllers in a distributed hash table) [57]. The control network can be wired or wireless data center [35], [54].

4.2.2 Network Assumption

We consider an SDN of one OpenFlow controller in the control-plane and $\mathscr{S}$ OpenFlow switches in the switch-plane. In a realistic SDN architecture, one OpenFlow controller might manage more than one OpenFlow switch. We assume that there are n disjoint paths available between the switch and the controller. A *path* is a connection between a switch and a controller that goes through one or more control nodes. Multiple paths between a switch and a controller have different IP addresses. Every path is unique, but two or more paths can share control nodes. We call two paths disjoint when they do not share any control nodes. The disjointness guarantee can be physical or virtual. Figure 4.1 illustrates an end host disjoint SDN where one controller c is connected to the four switches $s1$, $s2$, $s3$ and $s4$, and each controller-switch has two paths. The paths are identified by the controller and switch's IP addresses. The controller has two IPs, $ip1$ and $ip2$. Switches



Disjoint paths between *c* and *s*1	*Path*1: *ip*1-*ip*3 *Path*2: *ip*2-*ip*4
Disjoint paths between *c* and *s*2	*Path*1: *ip*1-*ip*5 *Path*2: *ip*2-*ip*6
Disjoint paths between *c* and *s*3	*Path*1: *ip*1-*ip*7 *Path*2: *ip*2-*ip*8
Disjoint paths between *c* and *s*4	*Path*1: *ip*1-*ip*9 *Path*2: *ip*2-*ip*10
Paths that share control node *r*1	*ip*1-*ip*3, *ip*1-*ip*5, *ip*1-*ip*7 and *ip*1-*ip*9
Paths that share control node *r*2	*ip*2-*ip*4, *ip*2-*ip*6, *ip*2-*ip*8 and *ip*2-*ip*10

Figure 4.1: End host disjoint network example

have IPs $ip3$, $ip4$, $ip5$, $ip6$, $ip7$, $ip8$, $ip9$, and $ip10$. The table shows the paths for c-$s1$, and they are $s1$-$s3$ and $s2$-$s4$. These paths are disjoint as they go through different control nodes $r1$ and $r2$. Each switch-controller has disjoint paths, but paths of different switches can share control nodes. For example, one path of $s1$ ($ip1$- $ip3$), $s2$ ($ip1$-$ip5$), $s3$ ($ip1$-$ip7$) and $s4$ ($ip1$-$ip9$) shares the same control node $r1$.

Path Creation

In our system, each OpenFlow switch and OpenFlow controller has multiple interfaces, and the minimum requirement of interfaces is n. IP addresses are assigned to the interfaces. A path manager *PM* knows the control network and is able to assign paths to a given pair of switch and a controller.

Figure 4.2 shows a model with one controller and two switches. We represent OpenFlow controller as c, j^{th} OpenFlow switch as sj, such as $s1$ and $s2$, i^{th} interface of j^{th} switch as $sj.int_i$, k^{th} interface

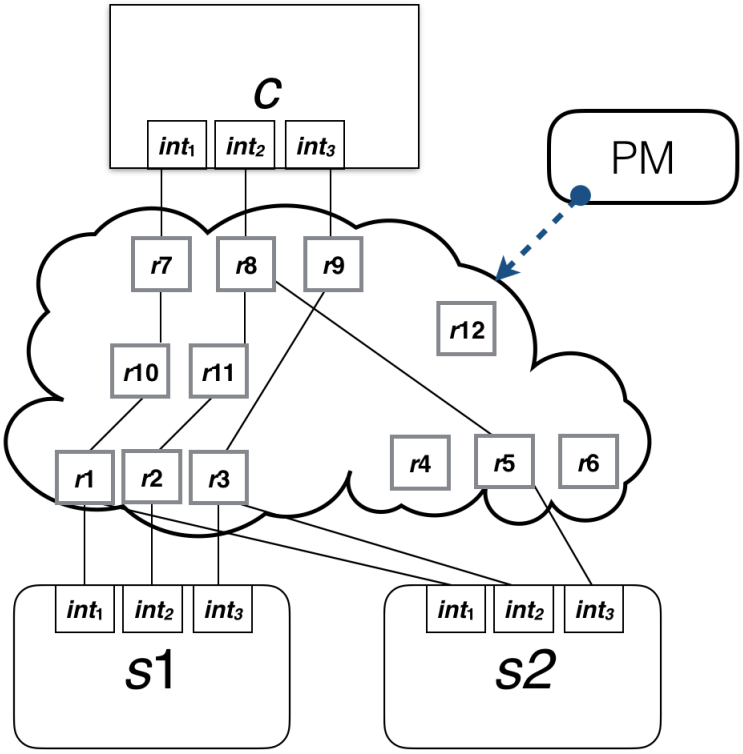


Figure 4.2: Path creation example

of controller as $c.int_k$ and l^{th} control nodes as rl. PM creates paths between sj and c. In this example, there are ten control nodes in the control network. In a case where the system requires n=3 disjoint paths, each sj and c need 3 interfaces. PM creates 3 disjoint paths between s1 and c, and s2 and c. Let us say, the paths between s1 and c are:

$Path1$: $s1.int_1$, $r1$, $r10$, $r7$ and $c.int_1$.

$Path2$: $s1.int_2$, $r2$, $r11$, $r8$ and $c.int_2$.

$Path3$: $s1.int_3$, $r3$, $r9$ and $c.int_3$.

The paths between s2 and c are:

$Path1$: $s2.int_1$, $r1$, $r10$, $r7$ and $c.int_1$.

$Path2$: $s2.int_2$, $r3$, $r9$ and $c.int_3$.

$Path3$: $s2.int_3$, $r5$, $r8$ and $c.int_2$.

Note that, s1 and s2's paths can share same rs, but all paths of sj's are disjoint.

4.2.3 System Assumption

The switch or the controller can create n shares of one OpenFlow message and send them through n disjoint paths. The receiver can reconstruct the message out of t shares where $2 \leq t \leq n$. The smallest possible system could use two shares to reconstruct a message. We call the system (t,n)-SSS that creates the shares and reconstructs shares using (t,n) Shamir Secret Sharing Scheme.

4.2.4 Security Assumption

The security properties that we consider are *confidentiality* and *reliability* in an adversarial case. The system provides *confidentiality* when the attacker eavesdrops on $t-1$ or fewer paths of one switch and one controller and cannot learn anything about the message. If the attacker has access to t or more paths at a time, the attacker learns the message, and the message loses its confidentiality. The system provides reliability when attacker blocks or jams atmost $n-t$ paths. In the case when $n-t+1$ or more paths are blocked, the system loses reliability. The difference to the traditional system is, in the traditional system, the attacker needs to eavesdrop on *one* path to learn the communication (either the communication is in plain text, or the quantum computer has compromised the key that was exchanged at the TLS handshake protocol), but here, the attacker needs to access at least $t > 1$ paths to learning the message. In the case of providing reliability, in the traditional system, if n paths are available, an attacker can block upto $n-1$ paths, but the system still provides reliability. If all n paths are blocked, there is no reliability. Figure 4.3 demonstrates the assumption.

4.2.5 Adversary Assumption

The adversary's goal is to affect security properties *confidentiality* or *reliability* e.i., learning the OpenFlow messages or blocking the communication. The attack techniques for compromising confidentiality is *eavesdropping or monitoring or listening*, and for compromising reliability is *jamming or blocking*. The eavesdropping attack (passive) and the Denial-of-Service (DoS) attack

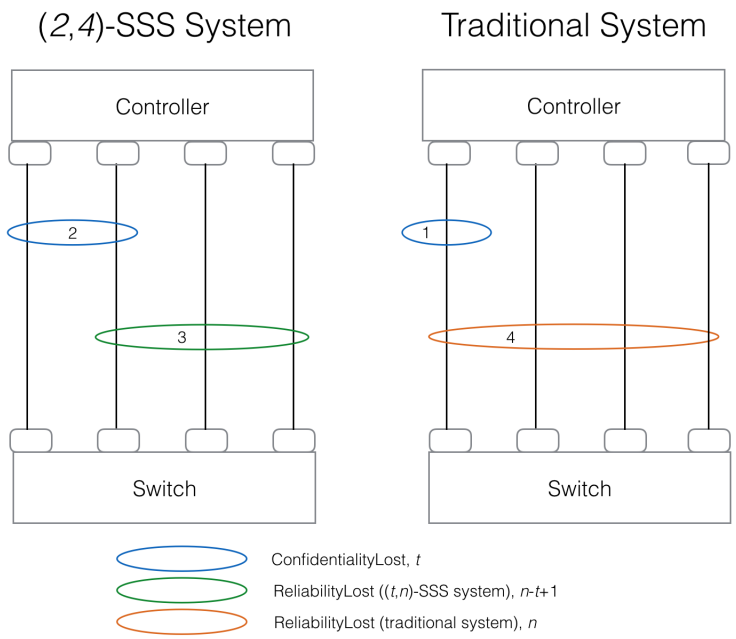


Figure 4.3: (2,4)-SSS vs. traditional system security assumptions

(active) are the two typical examples of cyber attacks considered in IT security [92].

Target	*specific* controller-switch communication			
Security property affected	confidentiality		reliability	
Attack technique	eavesdrop		jam/block	
Number of paths accessed	*thresholdConf*	more than *thresholdConf*	*thresholdRel*	more than *thresholdRel*

Figure 4.4: Adversary's goal, techniques and powers

For now, we do not consider adversaries that can tamper with messages that are sent over the channel, although our approach can be extended to allow for such adversaries (using extensions of Shamir Secret Sharing). The control network path assignments are opaque to the adversary; therefore the attacker chooses paths randomly. But at the consecutive period, the attacker may know the path assignments if the paths or the IP addresses are static. The attacker can access a threshold number of paths or more than threshold number of paths for each attack technique

types. The threshold numbers, $thresholdConf$ and $thresholdRel$, are determined such that they ensure the system's confidentiality or reliability. When we say, the attacker can access a the path; we mean that the attacker has access to any control nodes that the path is going through. If any other paths are going through the same control nodes, all of the paths may get affected. The target is a *specific* switch-controller communication. To successfully target a specific switch-controller communication, the attacker needs to know the exact switch-controller address. The attacker needs the static IP addresses and the static path assignments to build that information over a continuous period. The adversary is considered having a quantum computer. Figure 4.4 shows the adversary's goal, technique, and powers.

4.2.6 Trust Assumption

We assume the OpenFlow controllers, the OpenFlow switches and the Path Manager that assigns multiple disjoint paths, are trusted, and the software running on them functions correctly; stays un-compromisable during the working of the system. We also assume the controllers and switches do not fail and stay up and working.

4.3 `bQSafeOF` Design

The Basic Quantum Safe OpenFlow (`bQSafeOF`) protocol uses a (t,n) Shamir Secret Sharing Scheme and sends the n shares of an OpenFlow message through n disjoint paths between the controller and the switch, and so provide reliability and security when $n-t$ of the paths are blocked, or $t-1$ are eavesdropped upon. Figure 4.5 illustrates a (3,4)-SSS.

We carefully choose parameters based on the OpenFlow switch specification and NIST recommendation.

The `bQSafeOF` system has the following modules:

OpenFlow Message Receiver. The system receives OpenFlow message generated by OpenFlow switch or OpenFlow controller and kth message is denoted as D_k. Our analysis of all specified 112

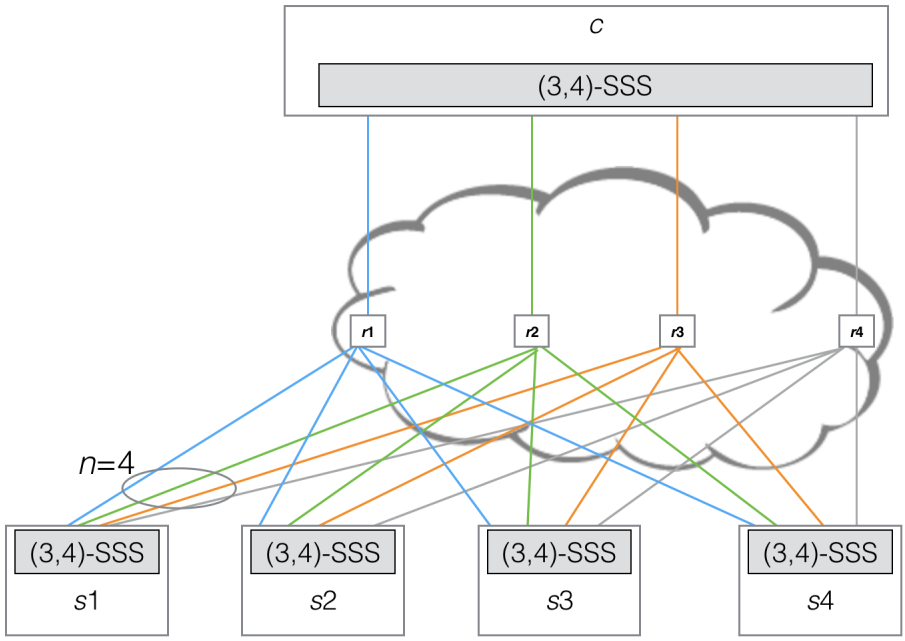


Figure 4.5: (3,4)-SSS design

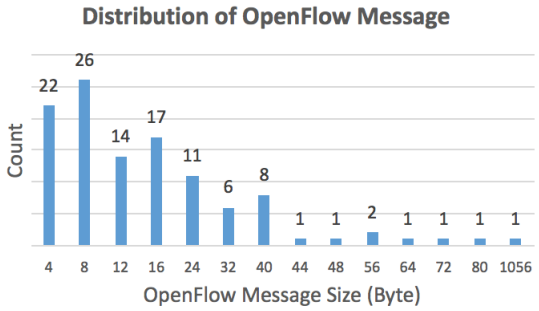


Figure 4.6: Distribution of OpenFlow message based on their sizes [49]

types of OpenFlow messages in OpenFlow switch specification finds that the minimum message length is 4 bytes, and the maximum is 1056 bytes that can go up to 64 kilo bytes in the future. The distribution is shown in Figure 4.6. It is important to know the lower bound, upper bound and the distribution of the OpenFlow message for choosing the prime size for the function (t,n)-SSS.

Initialization. Let $\mathscr{P}$ denotes a p-bit prime used in the (t,n)-SSS. In practice for OpenFlow messages, one needs to choose the prime size carefully to offer the required security level, while ensuring that message sizes are efficiently mapped into the prime $\mathscr{P}$. We considered a length that covers most of the OpenFlow messages and reduces creating OpenFlow message blocks. From

our analysis, the length can be 160 bits as it covers around 70% of total OpenFlow message types. This length is larger than the NIST [25] 112-bits-of-security requirement. We generate p-bit prime and keep it on a public server from where the OpenFlow switch and the OpenFlow controller can download it.

Share Constructor. The Share Constructor creates n shares of OpenFlow message D_k using (t,n)-SSS. An OpenFlow message, D_k is broken into p-bit blocks if $D_k > p$-bit, the last one possibly padded with zero, and (t,n)-SSS share construction is applied to each $block_i$ independently. For a message that is of length $|D_k|$, the maximum number of blocks is, $b = \frac{|D_k|}{p} + 1$ and the ith block of kth message is represented as $D_k.block_i$. Here the j^{th} share of the i^{th} block is represented as $block_i.share_j$ where $i = 1,2,..,b$ and $j = 1,2,..,n$. We generate n shares of message D_k by concatenating block shares and represent the jth share of the kth message as,

$$D_k.share_j = block_1.share_j \parallel block_2.share_j \parallel \cdots \parallel block_b.share_j$$

Here $\parallel$ denotes concatenation of strings. For a message D_k, n shares require at most $(|D_k| + p) \times n$ bits of space.

Packet Constructor. Two pieces of information need to be attached as header to the shares, (i) x_j of corresponding $share_j$: It is required for (t,n)-SSS reconstruction, and (ii) mID_k : kth message ID helps to identify that these shares are for the same message D_k. We call both pieces of information together as *metadata* and jth metadata is represented as $metadata_j = (mID_k, x_j)$. For n shares of message D_k, the final header and share looks like below:

$$metadata_j \parallel D_k.share_j$$

where $j = 1,2,..n$, k is any number and $\parallel$ denotes concatenation of strings.

It creates TCP/IP packets by attaching header to the shares. The final n packets have below construction:

$$TCPHeader \parallel IPHeader \parallel metadata_j \parallel D_k.share_j$$

Connection Setup and Send. OpenFlow switch specification [49] suggests TCP connection between the switch and the controller. Once packets are generated, the sender (OpenFlow controller

or the OpenFlow switch) establishes the n required TCP connections associated with the n paths and send each packet on one path. When it sends a packet, it acts as a client.

Packet Receiver. It receives packets through n disjoint interfaces and re-orders packets using TCP protocol. When it receives a packet, it acts as a server.

Packet Matching. It runs a function to match packets of the same OpenFlow message. The function identifies mID_k and keeps in a group for the same mID_ks until it finds t same IDs.

Message Re-construction. Once t packets of message D_k are found, it uses re-construction to find the message. The re-construction takes p-bit $block_i.share_j$ for all $j = 1,2,..t$ as input and reconstructs $D_k.block_i$. The process continues until b blocks of D_k are reconstructed. All the concatenation of reconstructed blocks are the message.

$$D_k = D_k.block_1 \parallel D_k.block_2 \parallel .... \parallel D_k.block_b$$

4.3.1 `notQSafeOF` Design

We first provide an alternate solution that works in the traditional system and provides the same level of *confidentiality* and *reliability* as (t,n)-SSS. The purpose is to compare the security of both systems. We call the alternate solution Not Quantum Safe OpenFlow Protocol (`notQSafeOF`). One can use an encryption system for message confidentiality, together with additional channels to provide reliability. In this system design, n disjoint OpenFlow channels are established between the controller and the switch and each OpenFlow message is sent n times, through the n disjoint paths. We can use Advanced Encryption Standard (AES) with 2048-bit RSA for key establishment.

The parameter sizes correspond to 128-bit security. The system achieves *reliable message delivery* if, $n-1$ paths between the controller and the switch fails or are jammed and achieves *confidentiality* if all n OpenFlow channels are eavesdropped on under the circumstances that the key is not compromised. When AES uses RSA to exchange keys, it would be possible to compromise the key using a quantum computer. The design is illustrated in Figure 4.7.

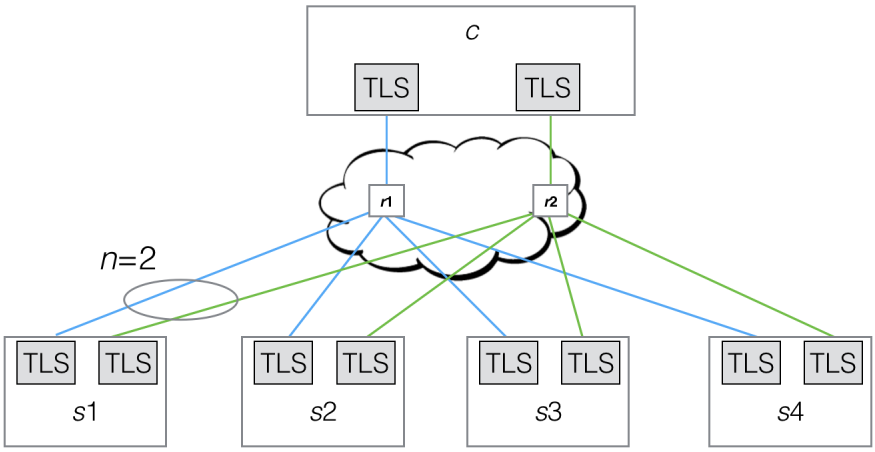


Figure 4.7: notQSafeOF design

4.3.2 Security Analysis of `notQSafeOF`

As the adversary is assumed to have a quantum computer, the adversary is able to compromise the encryption key that was exchanged during the TLS key handshaking protocol. We allow an attacker to access threshold number of paths so that we can compare the security analysis of our proposed solution `bQSafeOF` with `notQSafeOF`.

At this scenario, the *thresholdConf* number of paths that the attacker can access and the system still provides confidentiality, would be zero. It is because, if the attacker has access to one path then using a quantum computer, the attacker can compromise the cryptographic key. The *thresholdRel* number of paths the attacker can access, and the system still provides reliability would be $(n-1)$. The reason is that only one path is necessary to send the message and so the attacker can block the rest of the paths. The attacker cannot eavesdrop or block the communication of any or specific switch-controller communication while accessing *thresholdConf* or *thresholdRel* number of paths and therefore, the success probability of the adversary in breaking confidentiality or reliability of a specific switch-controller communication is zero.

Now, we describe a scenario when the attacker has access to more than *thresholdConf* of (at least any one path) or *thresholdRel* (at least n paths) number of paths. System's IP addresses and the path assignment by the PM are static. Therefore the attacker is able to know the information of

the specific path of switch-controller communication. In this case, the probability of successfully breaking confidentiality and reliability of a specific switch-controller is 1.

4.3.3 Security Analysis of `bQSafeOF`

When the system creates shares of a message and sends them through multiple paths, it increases the attacker's effort. For this system, the attacker needs to know at least ($t > 1$) number of IP addresses of one switch for eavesdropping. However for `notQSafeOF` one IP address is sufficient to identify and perform eavesdropping attack. So, the *thresholdConf* for `bQSafeOF` is $(t-1)$, and that is why it is necessary for the attackers to build a switch-IP mapping table to launch an attack on a specific switch-controller communication. When the system IP address and the path assignments are static, at some point, it is assumed that the attacker is able to build that switch-IP mapping table and can use this information to target a specific switch-controller communication. In the case of a blocking attack, `bQSafeOF` allows at best $(n-t)$ paths to be blocked that is less than `notQSafeOF`'s *thresholdRel*= n. So, *thresholdRel* of `bQSafeOF` is $(n-t)$.

The attacker cannot eavesdrop or block the communication of a specific switch-controller communication while accessing *thresholdConf* or *thresholdRel* number of paths and therefore, the probability of successfully breaking confidentiality and reliability of a specific switch-controller is zero.

When the attacker has access to more than *thresholdConf* (at least t paths) or *thresholdRel* (at least $n-t+1$ paths) number of paths, the probability of successfully breaking confidentiality and reliability of a specific switch-controller is 1.

4.4 `mtdQSafeOF` Design

Static parameters in an architecture help the attacker keeps the attack running for a long time until there is a change in any parameter and increases the likelihood of a successful *targeted attack*. In the `bQSafeOF` system design, there are fixed parameters, such as the source IP addresses,

destination IP addresses and path assignments. We use two randomization techniques along with `bQSafeOF`; (i) *IP randomization:* that generates dynamic random IP addresses for *clients*. Here, the controller and the switch act as *clients* when they sends message and act as *servers* when they receive messages. The dynamic assignment changes at every t_ip time, and (ii) *Path assignment randomization:* When there are $m > n$ paths, PM can choose n paths randomly out of m paths at every t_pa time. Figure 4.8 illustrates an `mtdQSafeOF`with (3,4)-SSS.

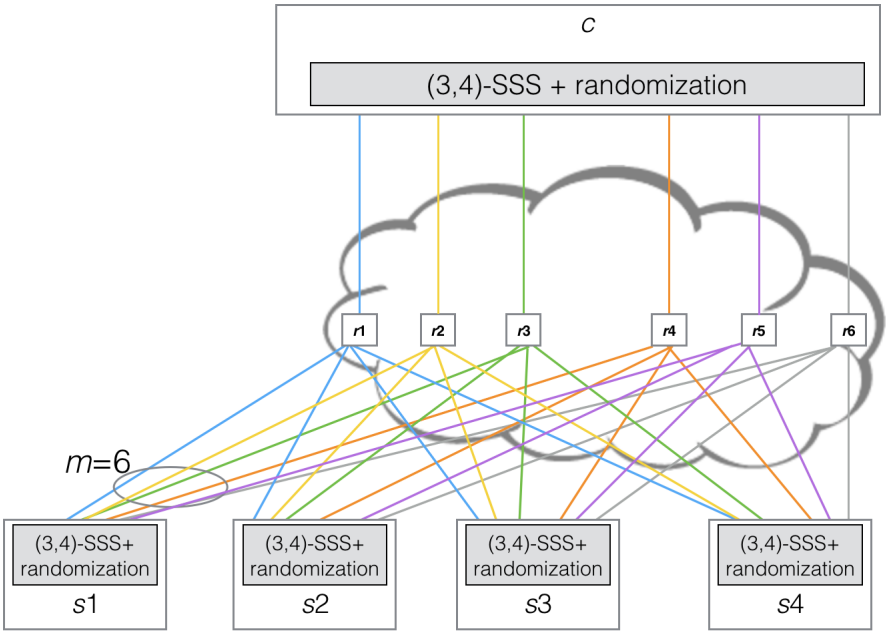


Figure 4.8: mtdQSafeOF design with (3,4)-SSS

The randomization techniques restrict the attacker to have the switch-IP mapping table and launch an attack on the specific switch-controller communication. It also increases the attacker's effort and moves the target; hence it is a Moving Target Defense (MTD) technique [96]. MTD techniques are effective for the incremental attacks [47] where the attackers first gather knowledge of the target.

The Randomization Technique

IPv6 supports assigning dynamic randomly generated IP addresses to clients, and that is known as SLAAC (StateLess Address Auto Configuration) [93]. The next hop router advertises the 64-bit (leftmost) subnet, and the client can generate its 64-bit (rightmost) interface address. In our design,

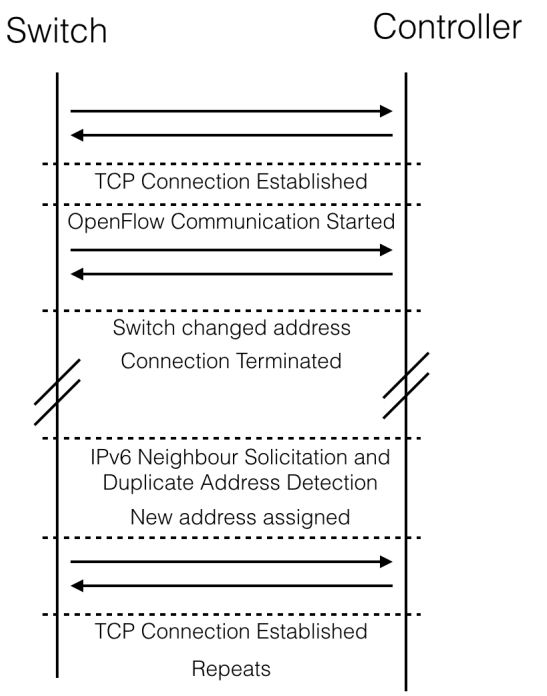


Figure 4.9: The overhead of changing IP address

on the client side, it changes all the interface addresses at the same time and uses a pseudo-random number generator to generate the 64-bit number.

Path manager (PM) on the other hand selects n paths randomly out of m paths.

Address Management

In our design, we keep the server's address fixed, or change it less frequently. We change the client's IP address. IPv6 has a feature to assign multiple addresses to one interface [58]. Let us say, the controller has n interfaces, and each interface can have $\mathscr{T}$ IP addresses. In our design, one address is used as the server address that remains fixed and the other one as client address that changes at every t_ip time.

Connection Management

Frequent changing of IP address and path assignment can result in connection tear down. In the OpenFlow protocol, there are two types of message exchanges. (i) *No reply required:* when the sender sends a message and does not expect any reply. In this case, an IP address change has

no impact. (ii) *Reply required:* The other one is when the client requests a response. In this case, the new IP address assignment goes through some protocols, such as IPv6 Neighbor Solicitation and Duplicate Address Detection that adds some time delay. Once the address is assigned, the TCP connection establishes. TCP coonection establishment adds another round of time delay. We illustrate the connection management in Figure 4.9.

4.4.1 Security Analysis of `mtdQSafeOF`

For the threshold number of path access for any specific switch-controller communication, `mtdQSafeOF`'s security is same as `bQSafeOF`'s security. For a (t,n)-SSS `mtdQSafeOF`, when the attacker has access to a $thresholdConf$ $(t-1)$ number of paths, the system provides confidentiality of any specific switch-controller communication. When the attacker has access to a $thresholdRel$ $(n-t)$ number of paths, the system provides reliability of any specific switch-controller communication. Therefore, the adversary's probability of successfully eavesdropping or blocking any specific switch-controller communication is zero.

IP is changing, so the attacker is not able to build the switch-IP information and cannot target a specific switch-controller communication when the attacker has access to more than $thresholdConf$ (at least t) paths. Therefore, the attacker's probability of successfully eavesdropping any specific switch-controller communication is,

$$= \frac{\mathscr{S}\binom{m}{t}\binom{\mathscr{S}\times m-m}{v-t}}{\binom{m\times\mathscr{S}}{v}}$$

where m is the total number of available paths between any switch and controller where $m > n$, (t,n)-SSS is the share construction and reconstruction module of `bQSafeOF`, $\mathscr{S}$ is the number of switches, and v is the number of paths that the attacker gets access to, where $t \leq v$.

When the attacker has access to more than $thresholdRel$ (at least $n-t+1$) paths, the attacker's probability of successfully blocking any switch-controller communication is,

$$= \frac{\mathscr{S}\binom{m}{n-t+1}\binom{\mathscr{S}\times m-m}{w-(n-t+1)}}{\binom{m\times\mathscr{S}}{w}}$$

where w is the number of paths that the attacker gets access to, where $(n-t+1) \leq w$.

4.5 Conclusion

In this chapter we described the models and assumptions. We proposed two systems `bQSafeOF` and `mtdQSafeOF` and analyzed their security. We also provide an alternate design `notQSafeOF` using TLS and analyzed its security. Figure 4.10 shows the probability of successful attacks for an attacker's different goals power and techniques when `notQSafeOF`, `bQSafeOF` and `mtdQSafeOF` are applied. We have shown that we can increase the security using a moving target defense technique along with the (t,n)-SSS whereas the (t,n)-SSS is secure only for a threshold number.

Target	specific switch-controller communication			
Security property affected	confidentiality		reliability	
Attack technique	eavesdrop		jam/block	
Number of paths accessed	*thresholdConf*	more than *thresholdConf*	*thresholdRel*	more than *thresholdRel*
Probability of successful attack:				
notQSafeOF (with compromised TLS key)	0	1	0	1
bQSafeOF	0	1	0	1
mtdQSafeOF	0	$\frac{\mathscr{S}\binom{m}{t}\binom{\mathscr{S}\times m-m}{v-t}}{\binom{m\times\mathscr{S}}{v}}$	0	$\frac{\mathscr{S}\binom{m}{n-t+1}\binom{\mathscr{S}\times m-m}{w-(n-t+1)}}{\binom{m\times\mathscr{S}}{w}}$

Figure 4.10: Probability of successful attacks for different cases

Our schemes can be applicable for protecting collusion attacks. In collusion attack, participants collude with each other and attempts to discover unauthorized secret data that would have been protected otherwise [31]. For example, participants $P1$, $P2$, and $P3$ can collude, read and discover a message m_1, m_2, m_3 even though they did not know the message individually. Our scheme

`bQSafeOF` and `mtdQSafeOF` protects against collusion of $t-1$ attackers. It is because, the system creates n shares and sends along n disjoint paths. If attackers collude with $t-1$ participants, they are not able to reconstruct the message when `bQSafeOF` is in use. When `mtdQSafeOF` is in use, the probability of successfully recovering the message is less than one because of the randomization of attacker's view.

Chapter 5

Implementation and Experiments

This chapter details the implementation of (i) `bQSafeOF`, (ii) `mtdQSafeOF`, and (iii) `notQSafeOF`, goals of the experiments, choice of tools, parameter settings, experiment setup, and results. It concludes with a summary of our findings.

5.1 Implementation

5.1.1 `bQSafeOF` Implementation

The `bQSafeOF`'s main module is a (t,n)-SSS that needs to be implemented with a 160-bit prime. It uses modular arithmetic operations per share generation and secret reconstruction. We found several code implementations available for (t,n)-SSS but those have some limitation according to our requirements. The Ubuntu GFShare Library *libgfshare* [5] provides Shamir Secret Sharing with the prime length 8-bit whereas we require a 160-bit length of the prime. Shamir's Secret Sharing Scheme *ssss* [83] is from point-at-infinity organization. The code is written in C and links against the GNU libgmp multi-precision library and requires the /dev/random entropy source. It is for UNIX/Linux machines. The prime size is from 8-bit to 1024-bit. The only limitation is the maximum message size can only be 128 characters. In our case, the OpenFlow message can go up to 1024-bit, and for the experiment purpose, we may need to use message sizes higher than 128 characters. *SHAREMIND* [28] is a virtual machine for privacy-preserving data processing that relies on share computing techniques with three computing participants. It is implemented in C++ and works on prime lengths of 32-bit. It is available commercially. Shamir's secret sharing scheme algorithm [36] is available under Crypto++ Library where it uses the prime length of 32-bit. The Library is a free C++ class library of cryptographic schemes. Both SHAREMIND and Crypto++ provide a prime length of 32-bit, but our requirement is 160-bit. Therefore, we

wrote our code using OpenSSL Cryptography and SSL/TLS Toolkit. The Big Number library OpenSSL/bn.h can be used for arithmetic operations on arbitrary size integers. OpenSSL library has functions to generate prime numbers and perform arithmetic operation modulo a prime. To generate cryptographically secure random numbers, *time* is used as the seed.

time(&tim);

RAND_add(&tim, sizeof(tim), 0.0);

We implemented our system using C language. We wrote four functions.

(1) *The initialization function* takes a p-bit prime pr and len-bit length of OpenFlow. It uses the OpenSSL random function *BN_generate_prime_ex()* function to generete pr. The outputs of the program are: p-bit pr, $(t-1)$ p-bit coefficients $a_1, a_2 \cdots a_{t-1}$, len-bit OpenFlow message OFm, n labels $x_1, x_2 \cdots x_n$.

(2) *Packet construction program* reads pr, a_1, $a_2 \cdots a_{t-1}$, OFm, x_1, $x_2 \cdots x_n$. The output of this program is n number of shares sh, sh_1, sh_2, $\cdots$ sh_n.

(3) *Packet reconstruction program* reads pr, x_1, $x_2 \cdots x_{t-1}$, sh_1, $sh_2 \cdots sh_{t-1}$. It reconstructs OFm and saves as $data$.

(4) *Connection establishment and transmission function* is a TCP server-client application. The server listens on a port, accepts the connection, and receives a packet on the connection. The client reads server IP and port from a file, connects, and sends shares.

5.1.2 `mtdQSafeOF` implementation

There are two randomization techniques: (i) IP address randomization: `mtdQSafeOF` uses `bQSafeOF` and IPv6 random address generator as its IP address randomization module. We use the module every t_ip seconds that changes the n number of IP addresses of n interfaces. (ii) Path randomization: the second technique uses path randomization module with `mtdQSafeOF` and chooses n paths of m paths at every t_pa time unit.

5.1.3 `notQSafeOF` implementation

For the `notQSafeOF`, we used *AES_cbc_encrypt()* function under OpenSSL/aes.h library to encrypt and decrypt OpenFlow messages. To connect and send encrypted messages, we wrote a code that establishes an SSL connection with the provided server's IP address and port number.

5.2 Goals of the Experiments

The experiments have three goals; (i) Goal 1: Comparing the basic module `bQSafeOF` with the other solution `notQSafeOF`, (ii) Goal 2: Simulating `bQSafeOF` in a large network with one controller and $\mathscr{S}$ switches, and (iii) Goal 3: Finding operational cost of IP and path randomization of `mtdQSafeOF`.

Goal 1: The first goal of the experiments is to compare `bQSafeOF` with the other solution `notQSafeOF` to find the computation cost and communication cost for one controller and one switch. The computation cost includes packet construction and packet reconstruction. It can be calculated for different values of t and n that are the number of paths captured by the adversary and the total number of paths respectively. The communication cost is to measure the delay when sending and receiving an OpenFlow packet. `bQSafeOF` uses n paths to send and receive packets and `notQSafeOF` uses n paths also. We experimented with one path between the switch and the controller. We also did a round trip delay calculation for both systems.

Goal 2: Our second goal is to simulate one controller and $\mathscr{S}$ switches and to find computation and communication cost. We compare the results with the one switch one controller results calculated in goal 1.

Goal 3: The goals of MTD experiments are to find the operational cost of IP change and path assignment randomization. The randomization of IP and path assignment result in connection tear down between the switch and the controller that introduces the extra overhead.

5.2.1 Experiments

We did two sets of experiments. (1) Varying the number t that is the adversary's listening power, but keeping the number $n-t$ reliability the same: The purpose of this experiment is to find the performance when t increases, but reliability is kept the same. (2) Varying reliability but keeping t the same: The purpose is to find performance when we increase the number t.

5.2.2 Performance Measures

We describe the performance measures in detail.

(i) *Computation cost:* For `bQSafeOF`, it is the cost of creating blocks of messages and constructing shares and reconstructing OpenFlow messages of those blocks. For the `notQSafeOF`, it is the cost of encryption and decryption of the same message. The cost is calculated by the *time* to do the computation for a fixed amount of data.

(ii) *Communication delay:* Communication delay is the time calculated to set up the connection and transmit all packets from the sender to the receiver. In our scheme, the sender connects to the receiver using TCP protocol and sends all the shares over the established connection. We construct the shares before the connection setup, and so once the connection is established, there is no extra computation cost. In the `notQSafeOF` however because of using SSL, computation cannot be separated from the communication, and time consists of the sender connecting to the receiver using SSL, encrypting data and sending the ciphertext.

(iii) *Communication overhead:* It is the extra bits that pass through n paths. We calculate the overhead. No simulation.

In our design, n shares are created, and n headers are attached to the packets. Tha total size is the size of all n packets with headers. We subtract the actual OpenFlow message size from the total size and find the overhead.

In our scheme, for one path, total output bits = (Number of blocks $\times$ p) + $|metadata_j|$ where p is the bit length of the prime. Here,

If $(D_k \bmod p) > 0$

then, number of blocks = $[\frac{|D_k|}{p}] + 1$

else number of blocks = $\frac{|D_k|}{p}$

where $|D_k|$= OpenFlow message size in bits, p = prime size in bits, $|metadata_j| = |mID_k| + |x_j|$, $|mID_k|$ is the size of message ID in bits and $|x_j|$ is the size of the label. Total output bits for n paths is $n \times$ Total output bits for one path where n = number of disjoint paths. We calculate the communication overhead = total output bits for n paths $-$ input OpenFlow message size.

Let us say, $|D_k|$=192 bits, p=160 bits, $|metadata_j|$=16 bits, then total output bits for one path is $(2 \times 160) + 16 = 336$ bits. For $n = 3$ paths, total output bits for 3 paths = $336 \times 3 = 1008$ bits and the communication overhead = $1008 - 192 = 816$ bits.

In `notQSafeOF`, the encryption algorithm is AES-128 where AES output length is 128 bits. So, for one path, total output bits = Number of blocks $\times$ 128. Here,

If $D_k \bmod 128 > 0$

then number of blocks = $[\frac{|D_k|}{128}] + 1$

else, number of blocks = $\frac{|D_k|}{128}$.

For n paths, total output bits = $n \times$ total output bits for one path.

We calculate total output bits for 3 paths when $|D_k|$=192 bits is $2 \times 128 \times 3 = 768$ bits, therefore communication overhead = $768 - 192 = 576$ bits.

Communication overhead to send 192 bits of OpenFlow message in our scheme is 816 bits and in `notQSafeOF` is 576 bits. Figure 5.1 displays the result for `bQSafeOF` with p=160 bits, $|metadata_j|$=16 bits and (t, n)-SSS=(2,3)-SSS, and `notQSafeOF` with 2 paths (same reliability as (2,3)-SSS) and AES output size 128 bits. It shows that the overhead of (2,3)-SSS is higher than the AES one.

(iv) *Round trip delay:* In an SDN architecture, round trip delay is the time it takes to start communication between two hosts. To calculate the round trip delay, we simulate an environment where one host pings another host while the switch generates the OpenFlow message and sends

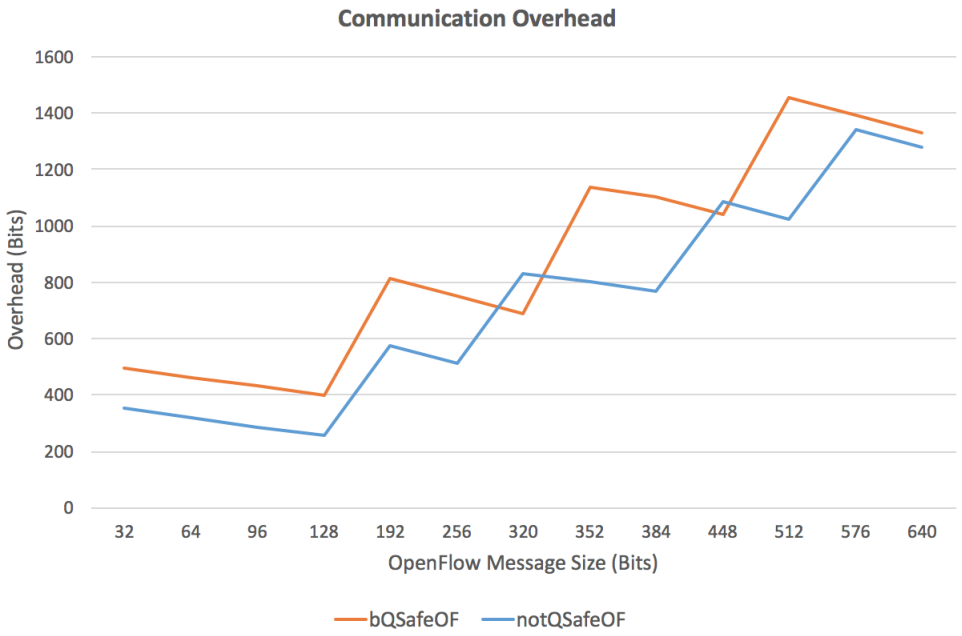


Figure 5.1: Communication overhead

to the controller. The controller replies with another OpenFlow message and installs a rule in the switch; then the two hosts start communicating. We measure the *time*. `notQSafeOF` uses TLS for n paths.

5.3 Tools and Dataset

5.3.1 Our Requirements

We need to run `bQSafeOF`, `mtdQSafeOF` and `notQSafeOF` on an SDN platform to measure the performance. We need one controller, several switches, several control nodes between controller and switch to create multipath. The switch and controller should have the speed to have multiple interfaces and change IP frequently.

`bQSafeOF` requirements: Figure 5.2 describes the requirements for `bQSafeOF` experiments.

Requirement A: The OpenFlow switch creates an OpenFlow message. Our `bQSafeOF` creates n shares.

Requirement B: The OpenFlow switch has n interfaces to send n shares.

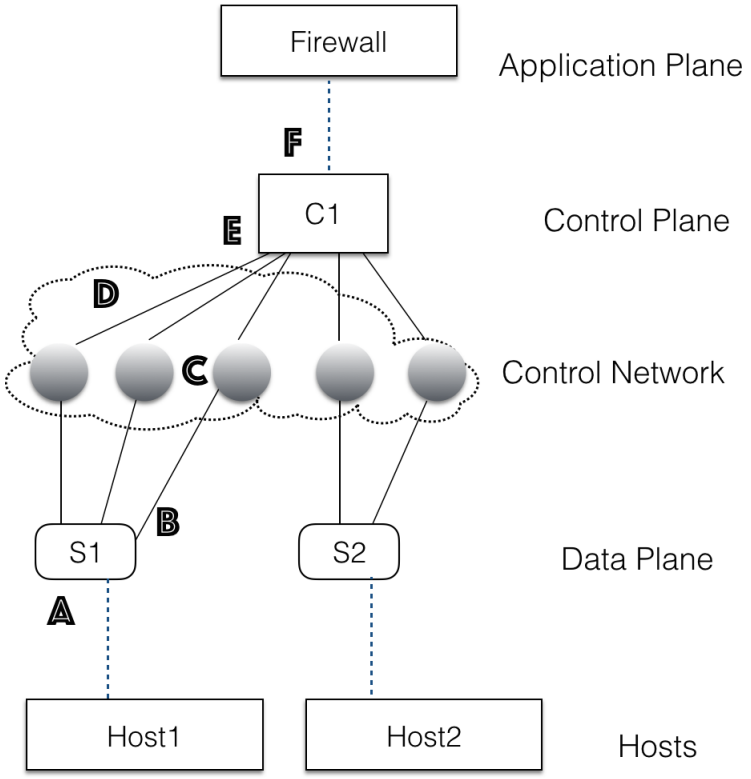


Figure 5.2: The requirements for our experiments

Requirement C: There are several control nodes to facilitate *n* disjoint paths.

Requirement D: There are *n* paths between each switch and controller available.

Requirement E: The controller has *n* interfaces and is able to receive *n* shares.

Requirement F: The controller receives at least *t* shares and reconstructs the OpenFlow message.

`mtdQSafeOF` requirements: `mtdQSafeOF` requires that IP addresses of each switch's interfaces change at some time interval. It also needs to create *m* multipaths and select *n* of them at some time interval for the path randomization case.

`notQSafeOF` requirements: The `notQSafeOF` requires to run TLS that exchanges keys between switch and controller, encrypts a message, sends the *n* copies of the ciphertext through *n* paths, the receiver recieves *n* ciphertext but it decrypts one ciphertext.

Our background study shows that the existing SDN simulators do not fulfill our requirement. In

this section, we discuss the available SDN simulators and their limitations to run our experiments.

5.3.2 SDN simulator

Mininet [69] is known as *the instant virtual network in your laptop*. It provides a development environment for SDN and is a widely used tool in SDN research community. Switches in Mininet are software-based switches like Open vSwitch or the OpenFlow reference switch. OpenFlow controllers, for example, python based POX, C based NOX, java based OpenDayLight can be connected to the OpenFlow switches in Mininet. Mininet can create a network of any number of OpenFlow switches and any number of hosts connecting to these OpenFlow switches. The connection between OpenFlow switch and OpenFlow controller is TCP based. We wanted to simulate a topology using Mininet and POX [13]. Firstly we installed Mininet and POX, secondly we designed an example case, and lastly, we ran the experiment.

Installing Mininet and POX

Mininet can be downloaded as a pre-packaged ubuntu virtual machine (VM). We setup a test environment in virtual operating systems. The host operating system is MAC OSX 10.9.4 desktop 64 bit, CPU 2 GHz Intel Core i7, RAM 8 GB 1600 MHz and HDD 250GB. We installed VMBox hypervisor and configured the host and guest OS connectivity using following steps:

Step 1: Installing VMBox and two guest operating systems

VMBOX and two guest operating systems (Ubuntu 64) bit were installed. Here, the host operating system is *dilruba* and the name of guest operating systems are *Alice* and *Bob*. Figure 5.3 below shows the illustration of the host and guest operating system configuration.

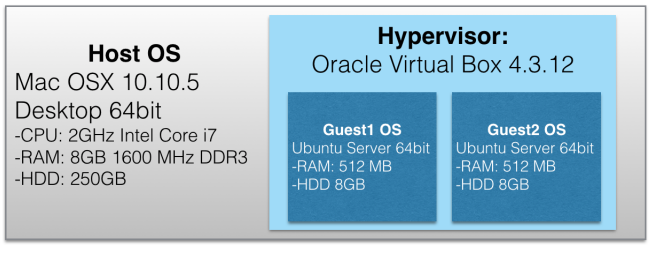


Figure 5.3: Host and two guest operating systems in a virtual setup

Step 2: Configuring VMBox and guests connectivity

While configuring host and guest network, the following points were taken care of: (i) a separate network is necessary for host OS to access guest VMs, (ii) to do that, each guest VM should have two network adapters, (iii) one is for guest VM to access outbound network, namely NAT adapter, and (iv) another is for host OS to access each guest VM, namely host-only adapter. Below Figure 5.4 shows the illustration of the network IP configuration of the host and guest OSs.

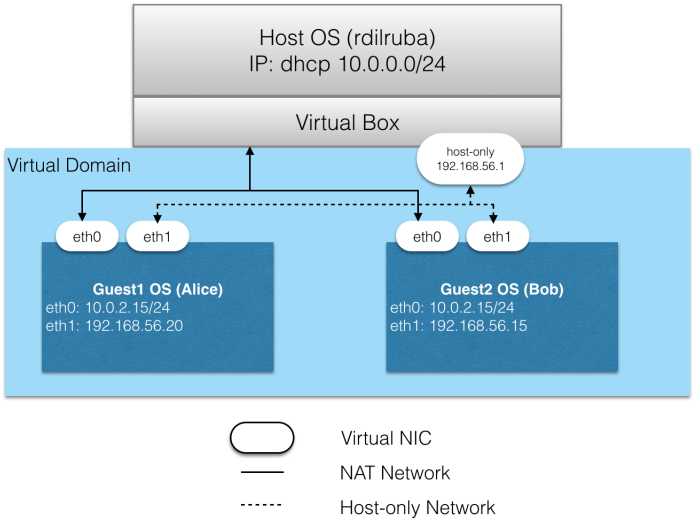


Figure 5.4: IP network configuration for host (*rdilruba*) and 2 guests (*Alice*, *Bob*)

We then configure VMBox using the following steps: (i) setup default networking for guest VM with NAT, (ii) create host-only networking in VirtualBox, (iii) setup secondary adapter for each guest VM, (iv) configure guest VM, and (v) test PING and SSH connectivity between host-guest, guest-guest. Then we do VM configuration as below:

VM1 = SVN-VM_1: IP 192.168.56.101

VM2= SVN-VM: IP 192.168.56.201

The Example Case

Our Mininet topology consists of one switch and two hosts. We set up two types of controller arrangement. In one case, controller, switch, and hosts are in the same virtual machine. In the second case, the controller is configured in a different virtual machine, and the switch and hosts

are in another virtual machine where the switch is connected to controller remotely. Figure 5.5 illustrates the two cases. The firewall application can add a rule or delete a rule based on MAC address. A CSV file with MAC source and destination has been provided as firewall policy.

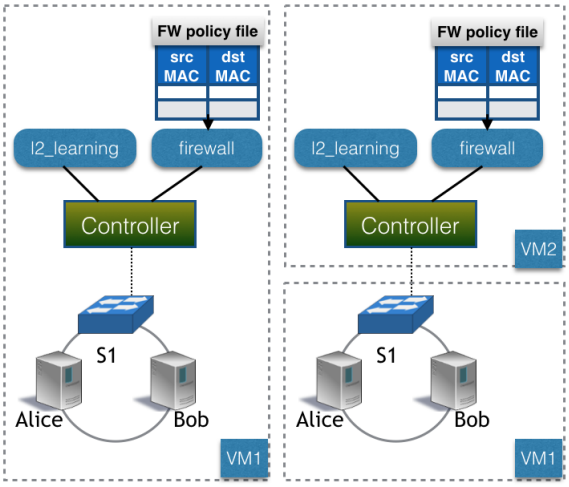


Figure 5.5: Our Mininet topology in two arrangements. In the left one, all components are in same virtual machine. In the right one, controller and applications are in separate virtual machine connected to the switch remotely.

Running Mininet and POX

Now we create the Mininet topology and attach switches to POX by using following commands.

killall controller (We do not want to use default controller, POX is used instead.)

pox.py log.level –DEBUG forwarding.l2_learning

Case 1: Controller in the same VM

mn –controller remote

This creates OVSController c0 (127.0.0.1:6633), OVSSwitch s1 (lo: 127.0.0.1), Host h1 (h1-eth0: 10.0.0.1) and Host h2 (h2-eth0: 10.0.0.2)

Case 2: Controller and switch in different VM

Here, controller IP: 192.168.56.101 and switch IP: 192.168.56.201

mn –controller remote,ip=192.168.56.101

This creates OVSController c0 (192.168.56.101:6633), OVSSwitch s1 (lo: 127.0.0.1), Host h1 (h1-eth0: 10.0.0.1) and host h2 (h2-eth0: 10.0.0.2)

Firewall application takes a CSV file as input policy and installs a rule to the switch OpenFlow table [2]. It reads the FW policy file, adds the rule to FW table using *addRule* and sends the rule to switch's OpenFlow table using *sendRule*. We run the application using the commands below. It is running in parallell with *forwarding.l2_learning* module. The application code is kept in /pox/pox/misc/firewall.py

pox.py log.level –DEBUG forwarding.l2_learning misc.firewall.py

We test if the firewall rules are working by pinging h2 from h1 and it works.

Challenges with Mininet

The limitations of this simulation are:

(1) We cannot create nodes between the switch and the controller for creating multiple paths.

(2) The switches have only one interface where we need multiple interfaces.

(3) The controllers have only one interface where we need multiple interfaces.

5.3.3 Hypervisor: Oracle Virtual Box

We observed from the previous experiment that the model is similar to TCP client-server model. For one switch one controller simulation, we use two VMs.

5.3.4 NS-3

We used NS-3 (Network Simulator). NS-3 offers a simulation environment where we can create nodes, connect them, assign protocol stack and addresses, install the application and run the simulation.

5.3.5 Summary

We choose our computer to find computation cost, hypervisor to find TLS and TCP communication cost and NS-3 to find all other communication costs for `bQSafeOF` and `mtdQSafeOF`.

5.4 Justification of Number Choices

For experiment, we have chosen dataset one 10^1 to 10^6 when we did comparison between the other solution `notQSafeOF` and `bQSafeOF`. It is because, `notQSafeOF` is optimized and to do the comparison it was necessary to increase the data in 10^6 level.

(2,3)-SSS is the smallest scheme possible and that is why we choose it to compare with `notQSafeOF`. We also choose (3,7), (5,9) and (7,11) because the cost of constructing and re-constructing shares is not efficient. That is why we wanted to keep the number of available paths between two nodes realistic and small.

For the experiments of Goal 2, our scaled system is consisted of four switches and each switch controller has four paths in between them. This is also to keep the numbers practical.

5.5 Goal 1: Comparing `bQSafeOF` with `notQSafeOF`

We run the experiment on the operating system MAC OSX 10.10.5, processor 2GHz Intel Core i7, memory 8GB 1600 MHz. We run `bQSafeOF` using *gettimeofday()* function excluding any file read and write time. The function *gettimeofday* returns time in microsecond precision. For `notQSafeOF`, we run OpenSSL *AES_cbc_encrypt()* function using *gettimeofday()* and recorded time excluding any file read-write. We run the programs twenty times to get the data for each point and take the average. Our code is not optimal whereas the OpenSSL AES is optimized.

Data Set 1:

Data set 1 consists of OpenFlow message length starting from 10^1 bits up to 10^5 where the prime is 160-bit. Table 5.1 shows the number of blocks for the corresponding data size. SSS block size is 160 bits, and AES block size is 128 bits.

Data Set 2:

Data set 2 is comprised of OpenFlow message sizes starting from 32 bits up to 640 bits increased by 32 bits. We use this data set for the communication delay experiments.

Table 5.1: Block sizes for 160 bit SSS and 128 bit AES

Data Size	SSS-160 No Of Blocks	AES-128 No Of Blocks
10^1	1	1
10^2	1	1
10^3	7	8
10^4	63	79
10^5	625	782
10^6	6250	7813

5.5.1 Packet Construction vs Packet Encryption

The experiment is to find the computation cost comparison between AES-128-CBC encryption and (2,3)-SSS, the smallest possible SSS. We run `bQSafeOF` program and OpenSSL *AES_cbc_encrypt()*. Figure 5.6 shows that the output at the first two points remained constant since these are computing only one block for both SSS and AES. For these two points, AES is around 5.4 times faster than (2,3)-SSS since ours is a non-optimal code. When data increases and block number increases (from 10^3 to 10^5), AES-128-CBC Encryption is on an average 10 times faster than (2,3)-SSS share construction.

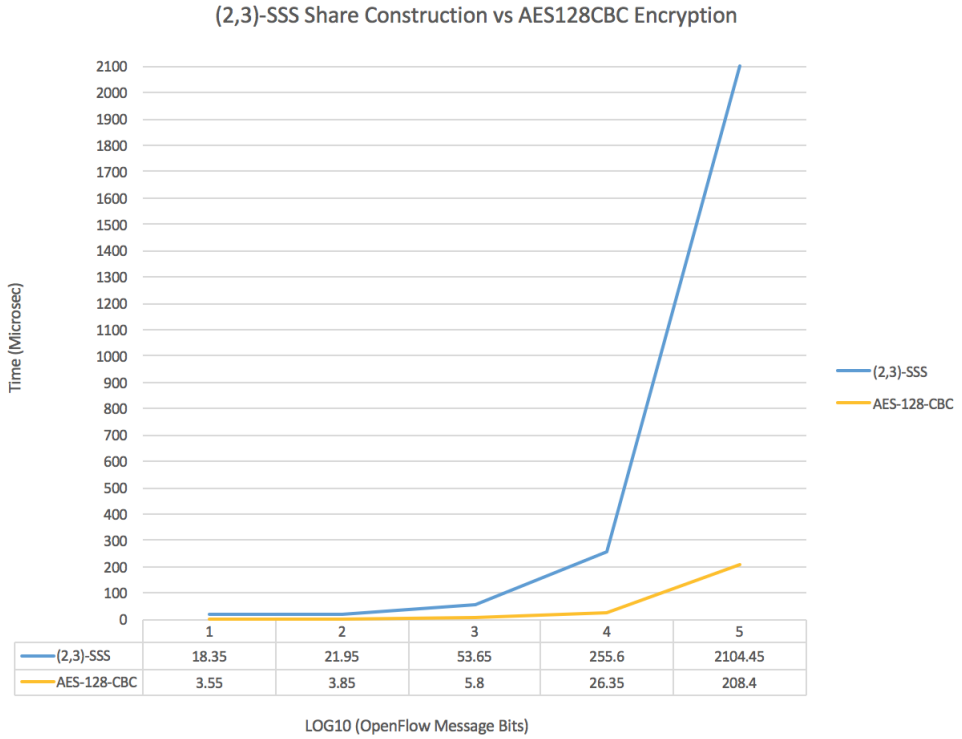
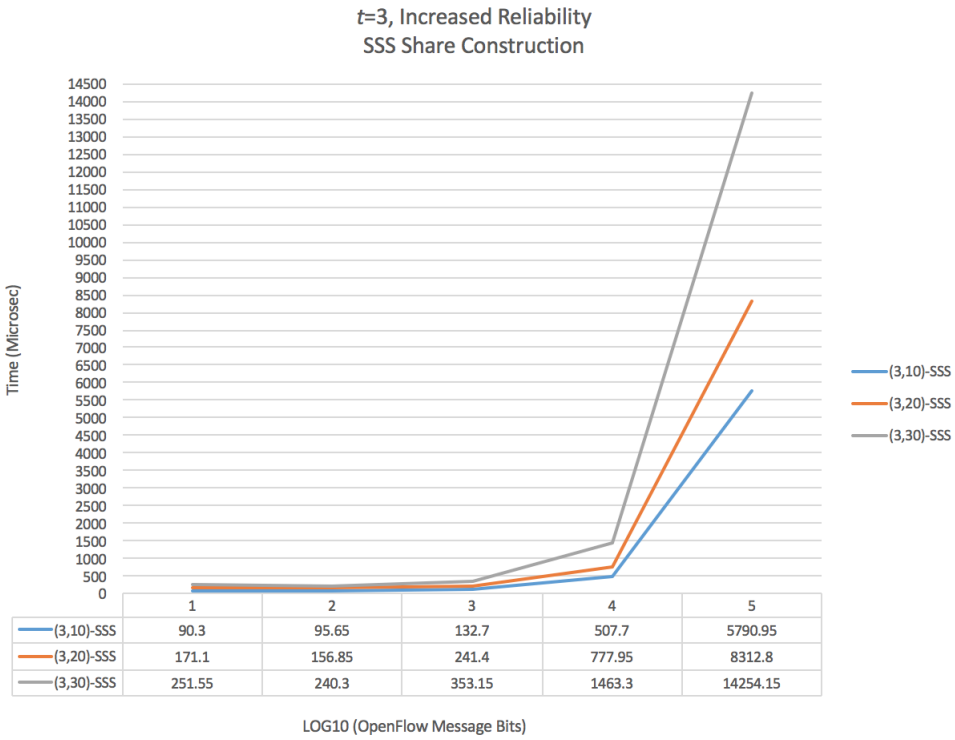


Figure 5.6: (2,3)-SSS construction vs AES-128-CBC encryption

5.5.2 Packet construction- keeping t the same and increasing reliability $(n-t)$

(2,3)-SSS is the smallest SSS, but in practice, we may use larger t and n. This experiment provides insight into the computation comparison when we keep the t same but increase n which increases the reliability as well. We calculate reliability by $(n-t)$ that means $(n-t)$ paths can be blocked

or jammed by the attacker but still the receiver is able to receive and reconstruct data. We run the `bQSafeOF` program for (3,10)-SSS, (3,20)-SSS and (3,30)-SSS share construction where t=3 and reliabilities are 7, 17 and 27, respectively, for data set 1. The x-axis is logarithmic (base 10) of the data set 1 and the y-axis represents the corresponding output in microseconds.



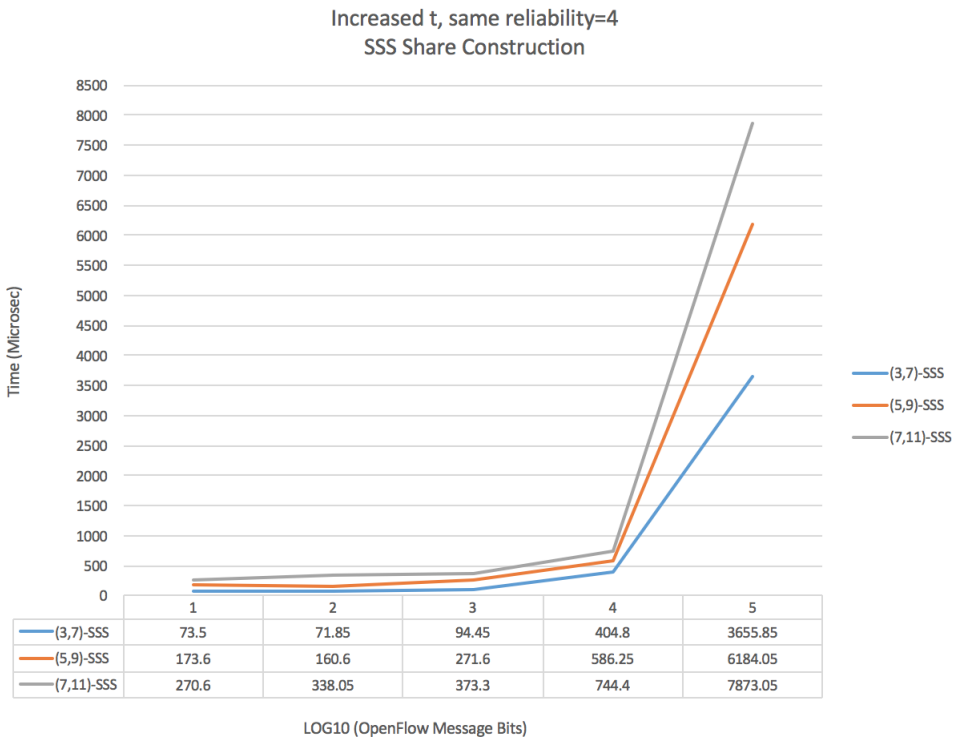
	1	2	3	4	5
(3,10)-SSS	90.3	95.65	132.7	507.7	5790.95
(3,20)-SSS	171.1	156.85	241.4	777.95	8312.8
(3,30)-SSS	251.55	240.3	353.15	1463.3	14254.15

Figure 5.7: Construction: same t and increased reliability

In Figure 5.7, the output for the first two points are the same as they are processing only 1 block. The 3rd one is an average 1.4 times (for 7 blocks) greater than the 2nd one; the 4th one is an average 3.7 times greater than the 3rd one (for 63 blocks which is 9 times greater than the previous point). Lastly, the 5th point is an average 10.6 times greater than the 4th point (for 625 blocks which is 9.9 times larger). For each point (3,20)-SSS is 1.6 times greater than (3,10). Again (3,30) is 1.6 times greater than (3,20) for each point.

5.5.3 Packet construction- increasing t and keeping the same reliability $(n-t)$

This experiment is to find the computation cost if we increase t, and keep the same reliability $(n-t)$. We run the packet construction program (3,7)-SSS, (5,9)-SSS and (7,11)-SSS. Figure 5.8 shows that the cost remained same for the first two points as these are only one block. It increases from 10^3 bits (7 blocks) by 1.4 times from point 1 and 2, 10^4 bits (63 blocks) by 3.22 times from point 3 and 10^5 bits (625 blocks) by 9.78 times from point 4.



	1	2	3	4	5
(3,7)-SSS	73.5	71.85	94.45	404.8	3655.85
(5,9)-SSS	173.6	160.6	271.6	586.25	6184.05
(7,11)-SSS	270.6	338.05	373.3	744.4	7873.05

Figure 5.8: Construction: increased t and the same reliability

5.5.4 Packet re-construction vs packet decryption

The experiment is to find the comparison of computation between AES-128-CBC decryption and (2,3)-SSS, the smallest possible SSS. We run `bQSafeOF` program and OpenSSL *AES_cbc_encrypt()* for data set 1. The x-axis is logarithmic (base 10) of the data set 1 and the y-axis represents the corresponding output in microseconds.

Figure 5.9 shows that the output at the first two points remained the same since these are

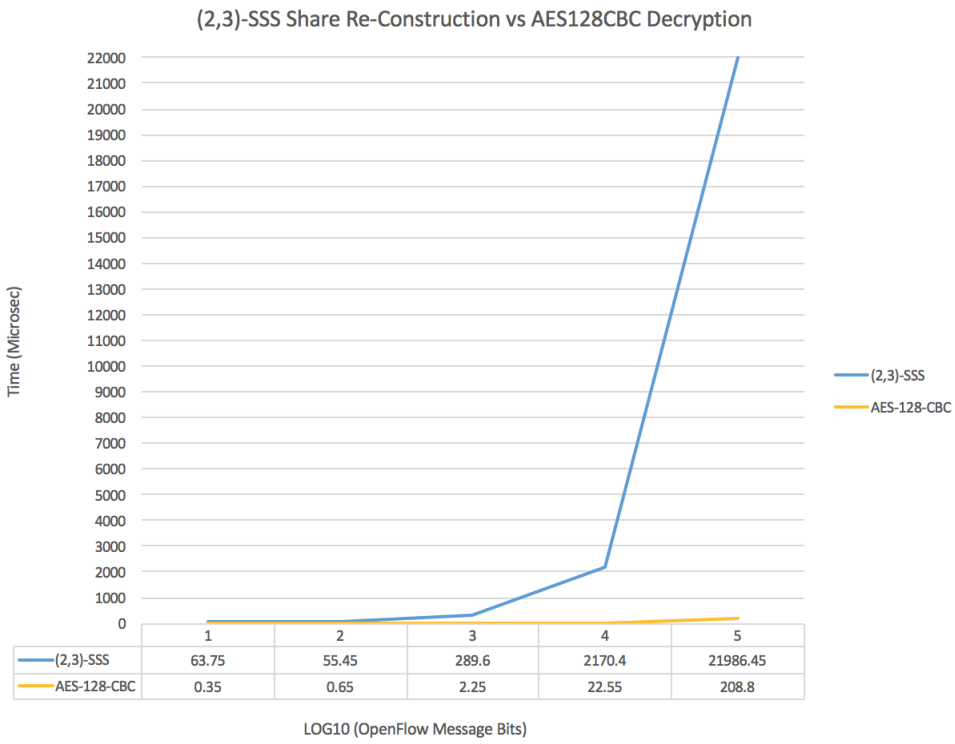
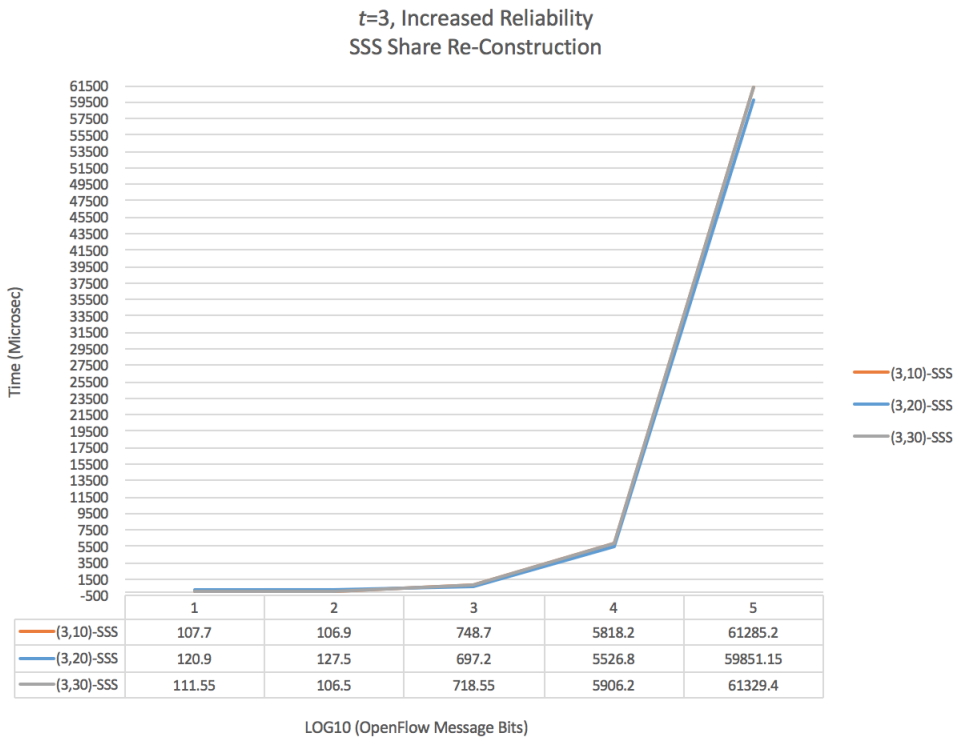


Figure 5.9: (2,3)-SSS re-ronstruction vs. AES-128-CBC decryption

computing only one block for both SSS and AES. For these two points, AES Decryption is on average 133 times faster than (2,3)-SSS share re-construction since ours is a non-optimal code. When data increases and block number increases (for 10^3 to 10^5), AES-128-CBC Decryption is on average 110 times faster than (2,3)-SSS share re-construction.

5.5.5 Packet re-construction- keeping the same t and increasing reliability $(n-t)$

This experiment compares the SSS reconstruction cost for t=3 and increasing reliability. Figure 5.10 shows share reconstruction for (3,10)-SSS, (3,20)-SSS and (3, 30)-SSS are same as it is re-constructing using three shares. Increasing reliability has no effect on computation.



	1	2	3	4	5
(3,10)-SSS	107.7	106.9	748.7	5818.2	61285.2
(3,20)-SSS	120.9	127.5	697.2	5526.8	59851.15
(3,30)-SSS	111.55	106.5	718.55	5906.2	61329.4

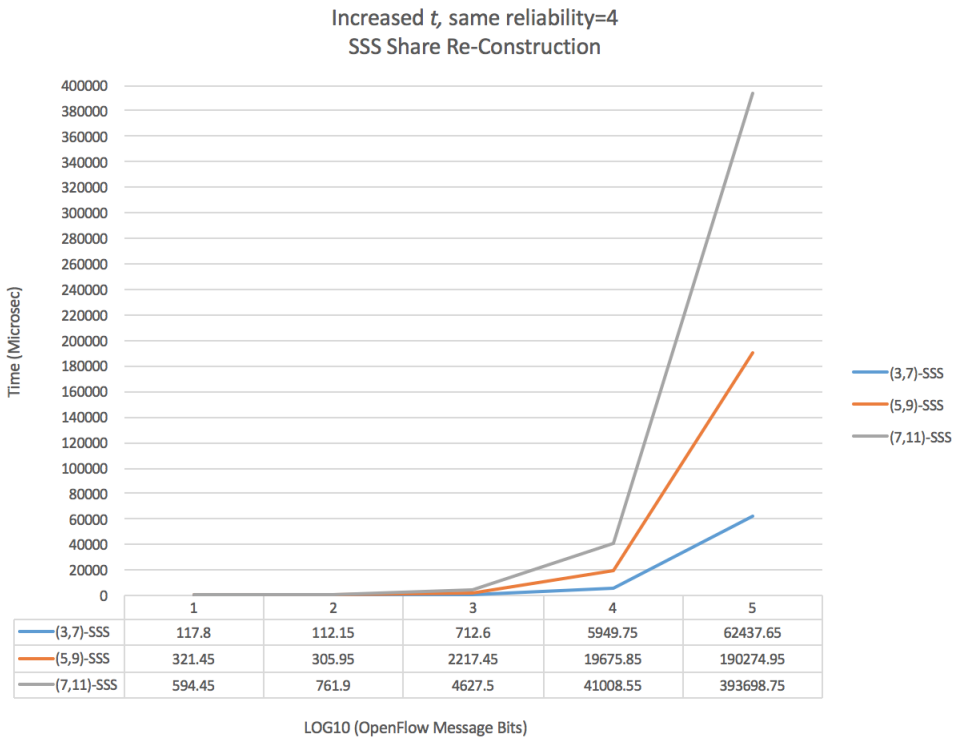
Figure 5.10: Re-construction: the same t and increased n reliability

5.5.6 Packet re-construction- increasing t and keeping the same reliability $(n-t)$

The experiment compares SSS reconstruction when t increases. We run the packet re-construction program (3,7)-SSS, (5,9)-SSS and (7,11)-SSS. Here the reliability is the same for all, and it is 4. (3,7)-SSS takes three shares, (5,9)-SSS takes five shares (adding two more shares from the previous one) and (7,11)-SSS takes seven shares and reconstructs data. Figure 5.11 shows that first two points remained the same for all. The third point is an average 6.63 times greater than the 1st and 2nd point, the 4th point is an average 8.6 times greater than the 3rd point, and the 5th point is an average 10.08 times greater than the 5th point.

5.5.7 Communication delay - TCP vs TLS in single path

Communication delay is the time calculated to set up the connection and transmit all packets from the sender to the receiver. In our scheme, the sender connects to the receiver using TCP protocol and sends all the shares over the established connection. We construct the shares before the con-



	1	2	3	4	5
(3,7)-SSS	117.8	112.15	712.6	5949.75	62437.65
(5,9)-SSS	321.45	305.95	2217.45	19675.85	190274.95
(7,11)-SSS	594.45	761.9	4627.5	41008.55	393698.75

Figure 5.11: Re-construction: increased t and the same reliability

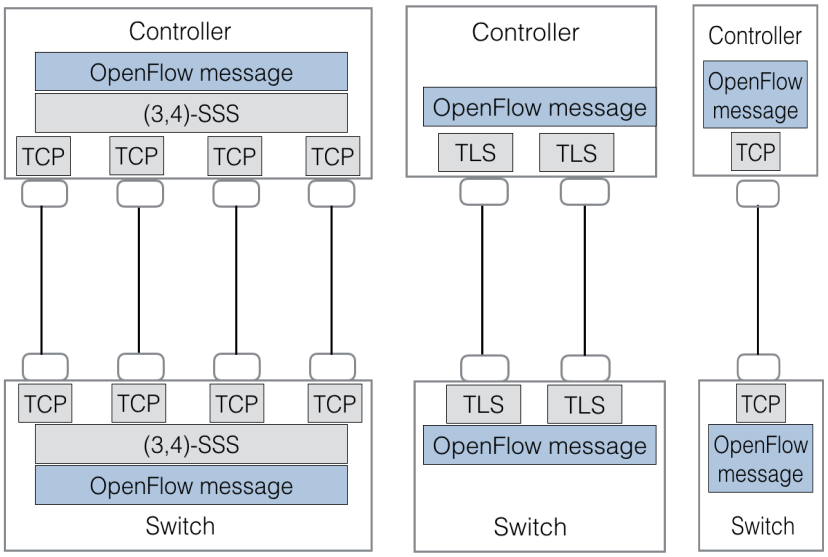
nection setup, and so once the connection is established, there is no extra computation cost. In `notQSafeOF` however because of using SSL, computation cannot be separated from the communication, and time consists of the sender connecting to the receiver using SSL, encrypting data and sending the ciphertext.

TLS protocol has connection setup overhead as it goes through different phases. The authors of [77] did an experiment in SDN to measure the connection establishment delay of TCP and TLS. According to their results, the delay with SSL certificate option is almost three times that of the plaintext TCP.

5.5.8 Experiment Setup

In SDN the controller and the switch send OpenFlow message over a single TCP connection using OpenFlow protocol. Our system `bQSafeOF` uses (t,n)-SSS to construct and reconstruct an OpenFlow message, and sends it over n paths. The alternative solution `notQSafeOF` uses TLS to connect

and encrypt-decrypt an OpenFlow message, and sends it over $(n-t+1)$ paths that provide same level of reliability as `bQSafeOF`. We illustrated an example for (3,4)-SSS `bQSafeOF` (Figure 5.12a), and `notQSafeOF` (Figure 5.12b) with the same reliability (4-3+1=2 paths), and single path TCP (Figure 5.12c) in Figure 5.12. We send OpenFlow message through these networks and perform experiments.



(a) Multipath (3,4)-SSS `bQSafeOF`

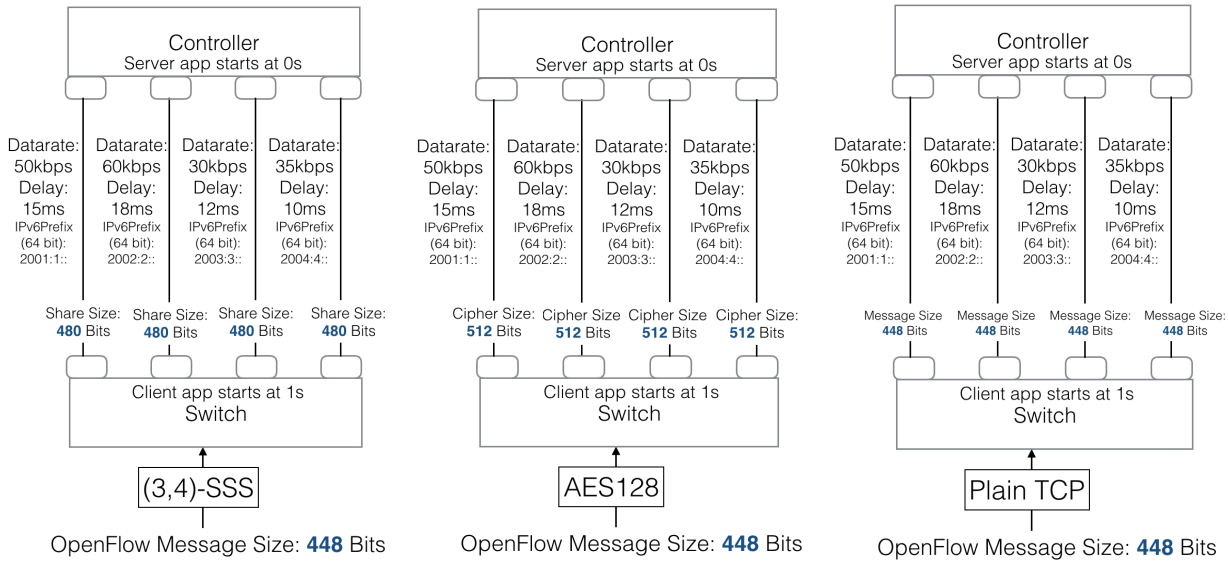
(b) Multipath `notQSafeOF`

(c) Singlepath OpenFlow

Figure 5.12: Experiment setup

5.5.9 Communication delay -Singlepath OpenFlow Protocol vs Multipath `bQSafeOF` vs Multipath `notQSafeOF`

The experiment is to find the communication delay in a multipath setting for `bQSafeOF` and `notQSafeOF`, and compare the results with a single path TCP OpenFlow protocol. We simulated the network using the NS-3 simulator. In our architecture, there are four paths between a controller and a switch as shown in Figure 5.13. Each path has a different data rate (*datRate*) and delay (*Del*), for example, Path1 has *datRate*: 50kbps and *Del*: 15ms, Path2's *datRate* is 60kbps and *Del*:18ms, Path3 has 30kbps *datRate*, and 12 ms *Del* and Path4's *datRate* is 35kbps and *Del* 10ms.



(a) (3,4)-SSS `bQSafeOF` (b) Multipath `notQSafeOF` (c) Singlepath OpenFlow Protocol

Figure 5.13: Communication delay experiment setup

To measure the time for multipath `bQSafeOF`, we run (3,4)-SSS share constructor and feed it with data set 2. We calculate the total running time as follow,

$$T_{mb} = t_1 + t_2$$

.

Here, t_1 is the TCP connection establishment time and t_2 is the time to send the t shares through t paths. We find t_1 for each path from the Wireshark packet capture. TCP connection establishment times for Path1, Path2, Path3 and Path4 are 0.05, 0.05, 0.06 and 0.05 seconds respectively. The SSS share constructor takes the OpenFlow message and creates four shares. We take the size of the share and send it through each path. In an ideal case (where there is no attack, or all links are up and running), the controller needs to wait for receiving at least three shares. There could be four options, Path1-Path2-Path3 or Path1-Path2-Path4 or Path1-Path3-Path4 or Path2-Path3-Path4. We calculate the lowest t_2 time and the highest t_2 time to send three shares among these options.

To measure the time for multipath `notQSafeOF`, we calculate the total time as follow,

$$T_{mn} = t_3 + t_4$$

.

Here, t_3 is the TLS connection establishment time and t_4 is the time to send encrypted packets through (1+$n-t$) paths. NS-3 does not have TLS module, so we could not run TLS key generation process. From the experiment [77], we learned that TLS connection with OpenFlow establishment took 65 ms (avg) whereas TCP connection with OpenFlow establishment took 20 ms (avg). Therefore we took 3.25 times of TCP connection establishment time as TLS connection establishment time and added that delay with each path sets. Therefore t_3 for Path1, Path2, Path3 and Path4 are 0.17, 0.18, 0.21 and 0.18 seconds respectively.

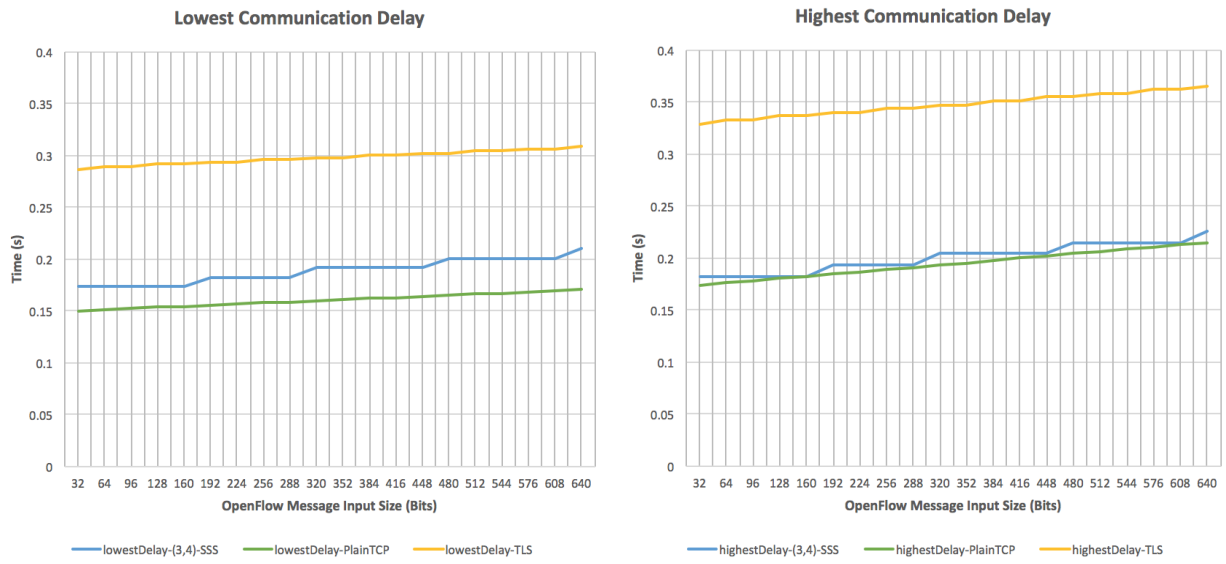
We run OpenSSL AES128 Encryption by feeding different OpenFlow messages as above. It creates the ciphertext and we take the size of the ciphertext and send it through 1+($n-t$) paths that provide the same level of reliability as (3,4)-SSS. Here, it is 1+(4-3)=2 paths. So we take different path options, for example, Path1-Path2, Path1-Path3, Path1-Path4, Path2-Path3, Path2-Path4 or Path3-Path4. The controller only needs one ciphertext to start the decryption process. Therefore, we take the minimum time t_4 of two paths to reach the controller. We choose the lowest t_4 and highest delay t_4 among all six options.

To measure the time for single path TCP, we calculate the total time as follow,

$$T_s = t_1 + t_6$$

.

Here, t_1 is the TCP connection establishment time and t_6 is the time to send OpenFlow packet through a single path. We run TCP client-server application by feeding different sizes of OpenFlow messages. The result helps to compare with our (t,n)-SSS 5.13c. We take the four path options individually and send the OpenFlow message size and measure the time t_6. We add the TCP connection establishment time t_1 and take the lowest and highest delay among four path options.



(a) Lowest delay

(b) Highest delay

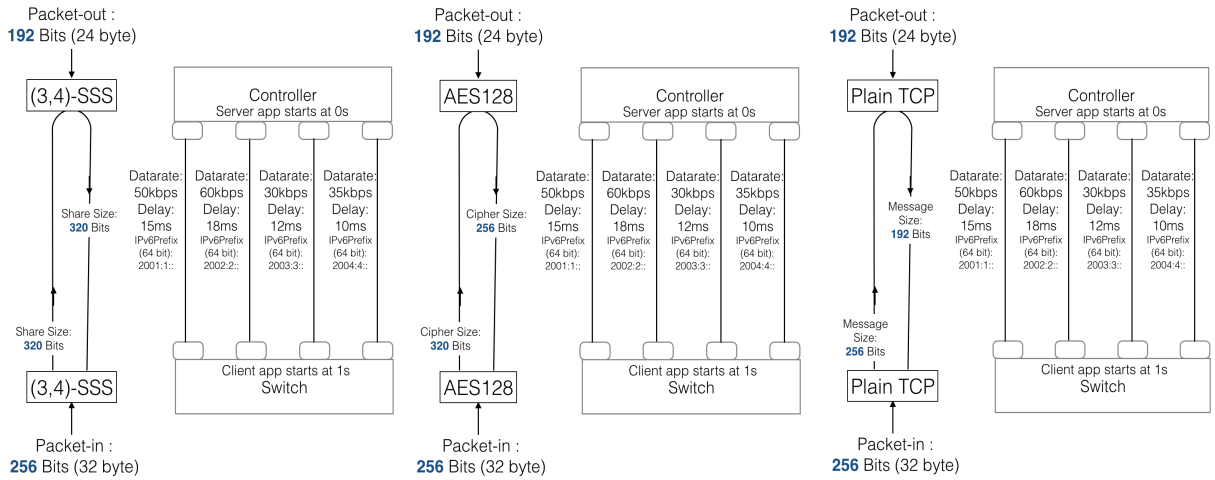
Figure 5.14: Communication delay in the 4 multipath setup

Figure 5.14 shows the comparison among the three experiments. (3,4)-SSS performs better than `notQSafeOF`, and it is comparable with the singlepath TCP OpenFlow protocol.

Multipath cmmunication can be affected by *Propagation Delay* and *Data Rate*.

5.5.10 Round Trip delay

In an SDN architecture, an OpenFlow message is generated when two hosts are attached to the same switch, and one host starts communicating with another host. The packet comes to the OpenFlow switch, if there is no matching rule, the OpenFlow switch sends a *Packet-in* message to the controller. The controller receives the message and creates a *Packet-out* message and replies to the switch. When the rule is installed in the OpenFlow message, communication between two hosts starts. The size of the packet-in message is 32 bytes (256 bits), and the size of the packet-out message is 24 bytes (192 bits). Share construction and reconstructions are done in the machine and the machine time is captured. Figure 5.15 shows an example setup for the three experiments. There are four paths between a switch and a controller. The paths have different delays and data rates.



(a) (3,4)-SSS `bQSafeOF` (b) Multipath `notQSafeOF` (c) Singlepath OpenFlow protocol

Figure 5.15: Round trip delay experiment setup

To measure the round trip time for multipath `bQSafeOF` we calculate the round trip time as follows:

$$T_{rmb} = tr_1 + tr_2 + tr_3 + tr_4 + tr_5 + tr_6 + tr_7 + tr_8$$

tr_1: the time taken by the switch to create shares of message, tr_2: time to establish TCP connection, tr_3: time to send the share to controller, tr_4: time to reconstruct, tr_5: time to create shares of reply message, tr_6: time to establish TCP connection, tr_7: time to send to the switch and tr_8: time to reconstruct the message at switch side. There are 4 sets of paths, Path1-Path2-Path3 or Path1-Path2-Path4 or Path1-Path3-Path4 or Path2-Path3-Path4. We calculate the lowest T_{rmb} time and the highest T_{rmb} time to send three shares among these options.

To measure the time for multipath `notQSafeOF`, we calculate the total time as follows,

$$T_{rmn} = tr_9 + tr_{10} + tr_{11} + tr_{12} + tr_{13} + tr_{14} + tr_{15} + tr_{16}$$

.

In the experiment for `notQSafeOF`, we calculate, tr_9: time to encrypt message, tr_{10}: time to establish TLS connection, tr_{11}: time to send to the controller, tr_{12}: time to decrypt, tr_{13}: time to

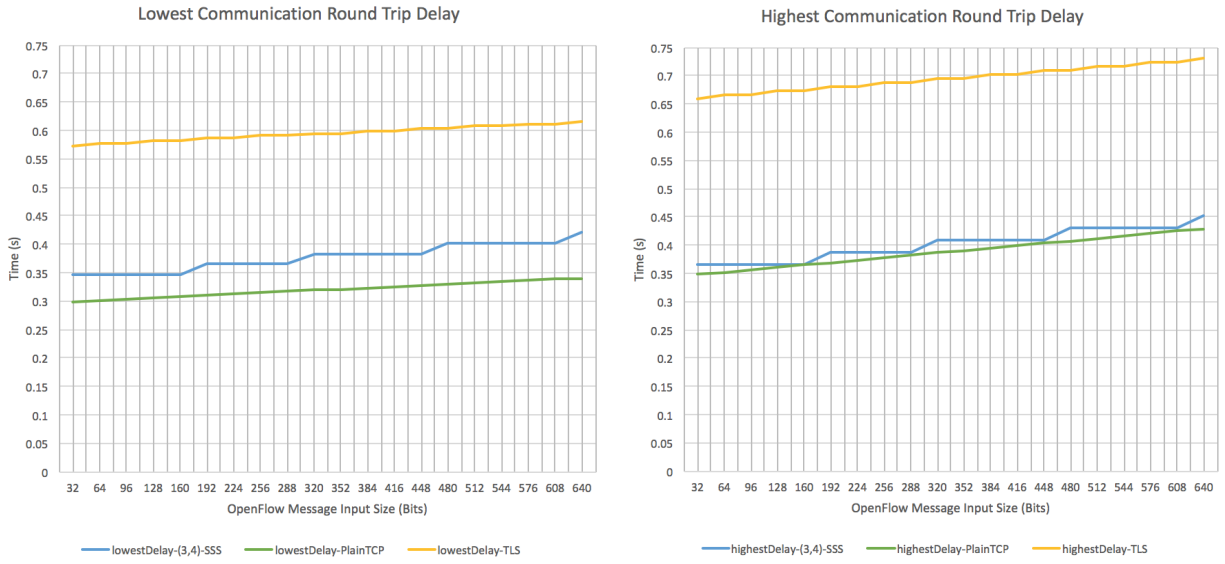
encrypt reply message, tr_{14}: time to establish TLS connection, tr_{15}: time to send to switch and tr_{16}: time to decrypt the message at switch side. We take different path pairs, for example, Path1-Path2, Path1-Path3, Path1-Path4, Path2-Path3, Path2-Path4 or Path3-Path4.The controller only needs 1 ciphertext to start the decryption process. Therefore, we take the minimum delay of two paths to reach the controller. We choose the lowest and highest delay T_{rmn} among all six options.

To measure the time for single path TCP, we calculate the total time as follows,

$$T_{rs} = tr_2 + tr_{17} + tr_6 + tr_{18}$$

.

For single path TCP, we calculate tr_2: time to establish TCP connection, tr_{17}: time to send the message to controller, tr_6: time to establish TCP connection from the controller to the switch and tr_{18}: time to send the message from the controller to the switch. We take the four path options individually and send the OpenFlow message size and measure the time. We add the TCP connection establishment time and take the lowest and highest delay T_{rs} among four path options.



(a) Lowest delay

(b) Highest delay

Figure 5.16: Communication round trip delay in the 4 multipath setup

We feed different sizes OpenFlow Messages from 32 bits to 640 bits. We keep the size of the

reply OpenFlow message the same as the input size. Figure 5.16 shows the graph, and we notice that (3,4)-SSS is performing better than `notQSafeOF`.

5.6 Goal 2: `bQSafeOF` in one controller and $\mathscr{S}$ switches settings

The experiment is designed to measure the costs in a scaled system. In a practical scenario, one controller manages several switches. Here, the scaled system refers to one controller and $\mathscr{S}$ switches. In our experiment, we use NS-3 to create the network and multipath between the controller and $\mathscr{S}$ switches.

5.6.1 Computation cost at a scaled system

When one controller is managing $\mathscr{S}$ switches and the switch is only connected to a single controller, packet construction or reconstruction at switch side is not affected by the rest of the switches and thus has no extra delay.

When controller creates packets for $\mathscr{S}$ switches, extra delays may add up. But the delay is not only of our system design, but the delay is also for any controller that manages $\mathscr{S}$ switches. Therefore we ignore the packet construction delay at the controller side. $\mathscr{S}$ switches create an extra delay at the controller side since it is receiving different shares of different OpenFlow messages from several switches.

Packet reconstruction has two types of delays: (i) wait time to receive t shares, and (ii) pattern matching time to find if the shares are from the same OpenFlow message. Figure 5.17 shows the flowchart for packet reconstruction and the two types of delays. The fastest pattern matching algorithm's time complexity at searching phase is $\mathscr{O}(a+b)$ for a-bit $string_1$ and b-bit $string_2$ [64].

The experiment calculates the time for packet matching. In this particular experiment, there are four paths between each switch and controller. We increase the number of switches from 1 to 4. For one switch four shares need to be matched. When the number of swiches increases, the number of shares participating in pattern matching is:

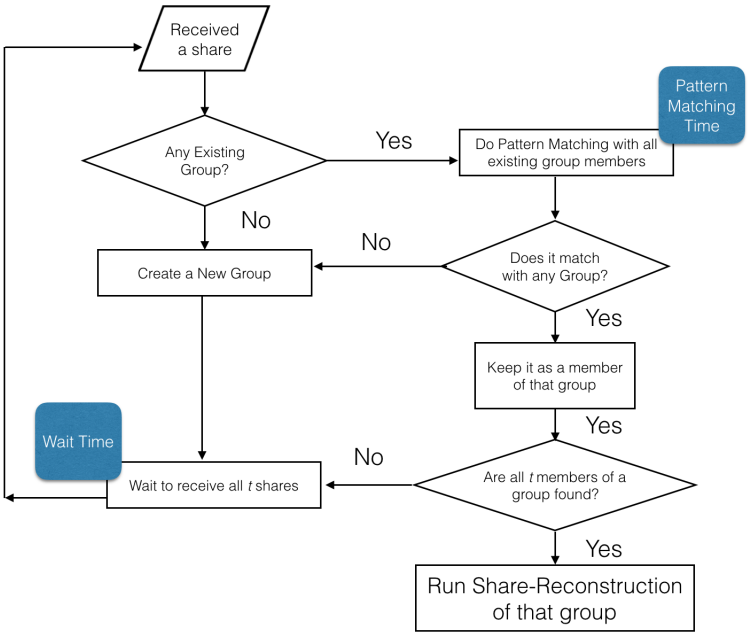


Figure 5.17: Flow chart for packet reconstruction

$$NumberOfPatternMatching = 4 \times NumberOfSwitch$$

For example, for three switches 12 shares require matching. We use KNUTH-MORRIS-PRATT ALGORITHM [64]. We give a 32 character input and run the algorithm *NumberOfPatternMatching* times. We record the output with microsecond precision. Figure 5.18 shows the results when the number of switches is 1 to 4 and the corresponding outputs are 6.6, 8.15, 10.2 and 12 microseconds.

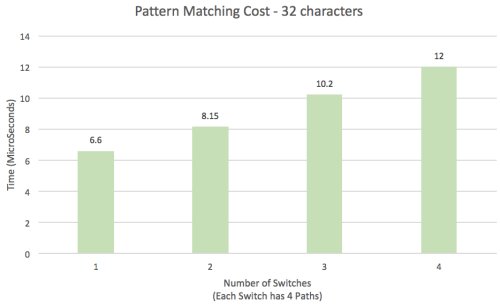
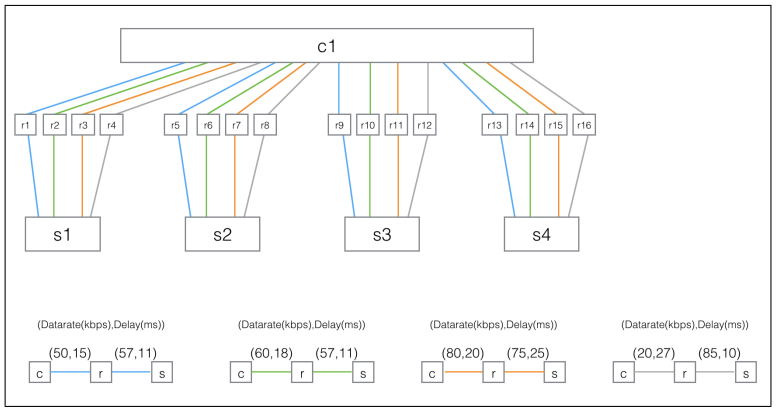


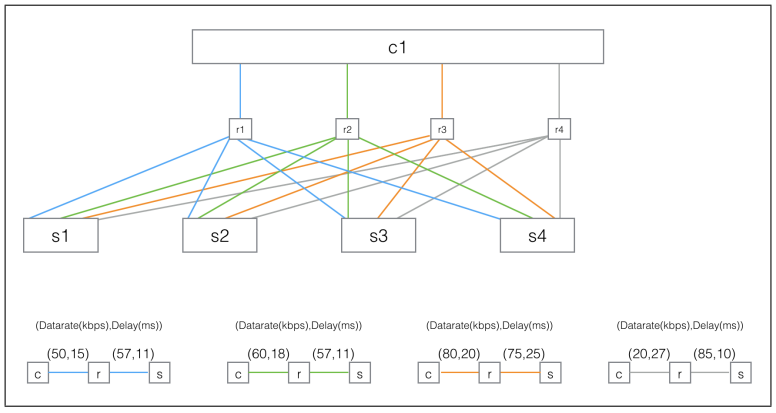
Figure 5.18: Computation cost for pattern matching at the controller side

5.6.2 Communication cost of a scaled system

When a control node is shared among different switches, there is an extra delay. In this experiment, we measure the extra delay for an example network. Our example scaled system consists of one controller, four switches and four control nodes between the controller and each switch comprising sixteen control nodes. There are four paths between each switch and controller. Each link from a switch to a control node and a control node to the controller has a different data rate and delay. In the first scenario, a control node is used by only one path (Figure 5.19a). In the second scenario, one control node is shared among four paths (Figure 5.19b). We send 320 bits through each path and measure the time for both scenarios. Figure 5.20 shows the results that when four switches share one control node, switch1 is least affected and switch4 is most affected, in this particular example network.

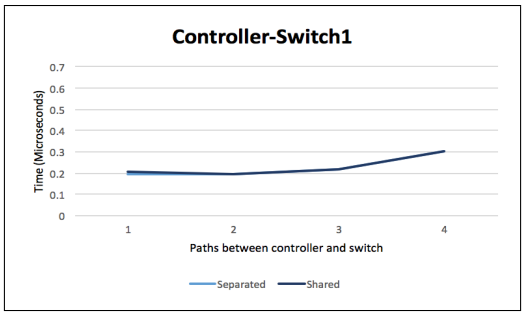


(a) Scenario 1 : control nodes are not shared

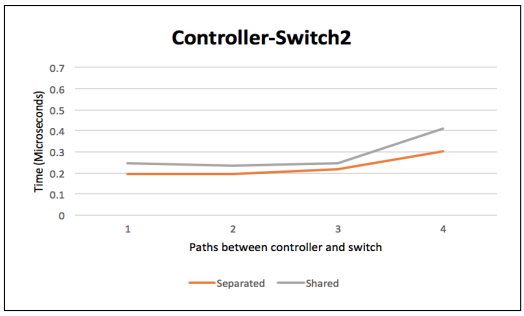


(b) Scenario 2 : control nodes are shared

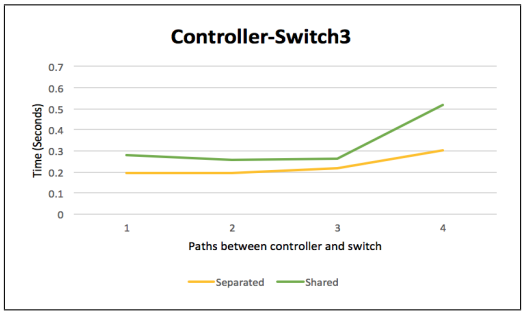
Figure 5.19: Experiment setup for communication cost at a scaled system



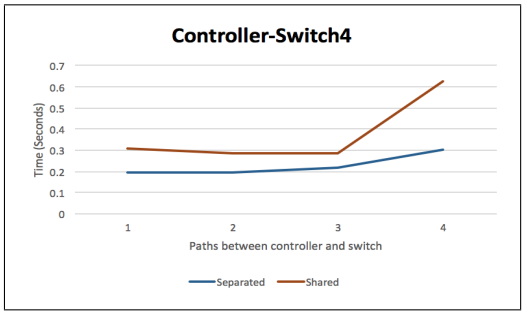
(a) Controller-switch1



(b) Controller-switch2



(c) Controller-switch3



(d) Controller-switch4

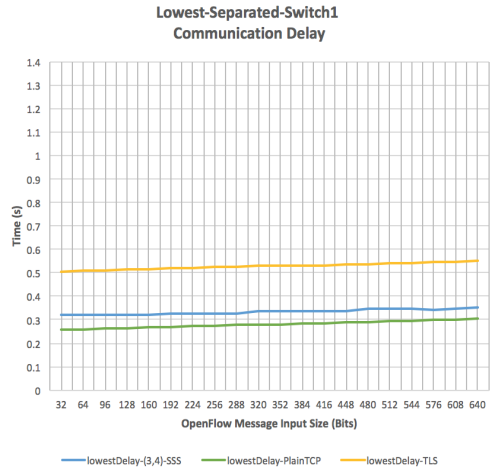
Figure 5.20: Delay distribution

To understand the costs, we categorize them into four cases: Lowest/Highest delay for separated/shared paths. We have taken the cases for switch1 (least affected) and switch4 (most affected). We list all eight cases below: (i) LowestDelay-Separated-Switch1, (ii) HighestDelay-Separated-Switch1, (iii) LowestDelay-Shared-Switch1, (iv) HighestDelay-Shared-Switch1, (v) LowestDelay-Separated-Switch4, (vi) HighestDelay-Separated-Switch4, (vi) LowestDelay-Shared-Switch4, and (vii) HighestDelay-Shared-Switch4.

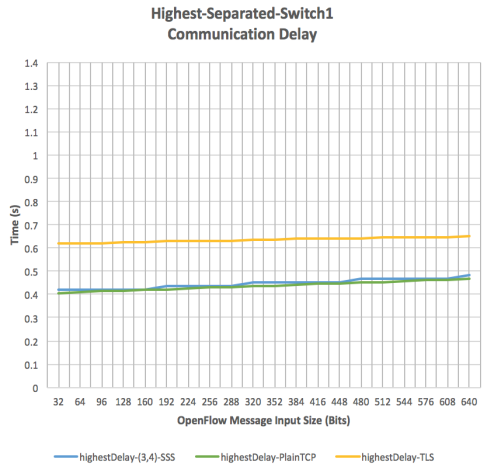
We ran the same experiment as in Section 5.5.9 for all scenarios by running (3,4)-SSS `bQSafeOF`, `notQSafeOF`, and single path OpenFlow, and recorded the communication delays. For all communication cost delay experiments we used data ranging from 32 to 32^{20} bits. The X-axis of the figures represents the data size in bits, and the Y-axis represents the output time in seconds.

The analysis in Figure 5.21 and Figure 5.22 shows that `bQSafeOF` is performing better than `notQSafeOF` and is comparable with plaintext TCP. In both scenarios Switch1's delay remains the same because of the delay distribution of the system (please refer to Figure 5.20a). How-

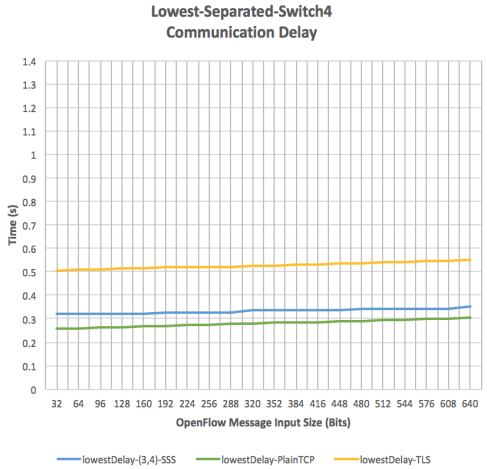
ever Switch4's communication cost for scenario 2 (Figure 5.22c and 5.22d) is higher than scenario 1 (Figure 5.21c and 5.21d) because of the delay distribution of the system (Figure 5.20d). Figure 5.22d shows the case for Switch4 for the highest delay. Here the performance of all three is relatively the same. This is because when there are four paths of increased delay where $Path1 < Path2 < Path3 < Path4$, the highest delay for `notQSafeOF` is $Path3$, but for `bQSafeOF` and single path TCP, it is $Path4$. Therefore, in the worst case, the performance of `notQSafeOF` can be comparable to the other two.



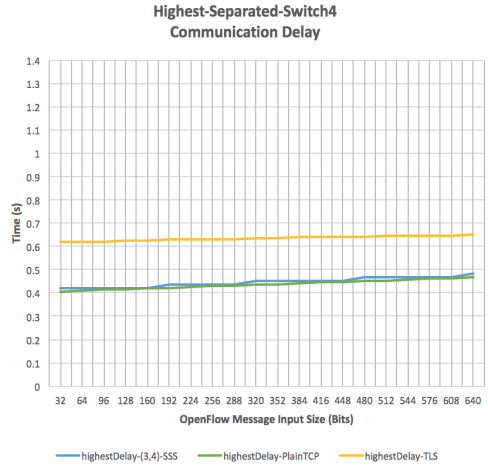
(a) Lowest-Separated-S1



(b) Highest-Separated-S1

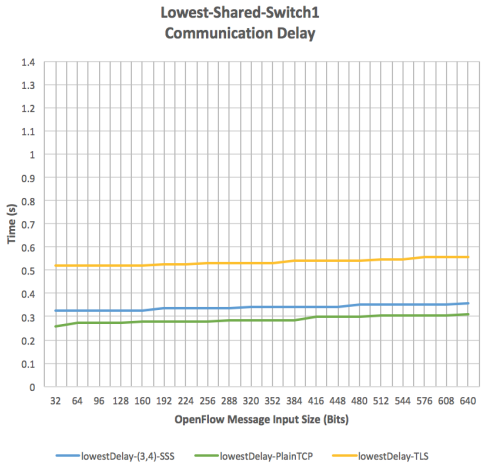


(c) Lowest-Separated-S4

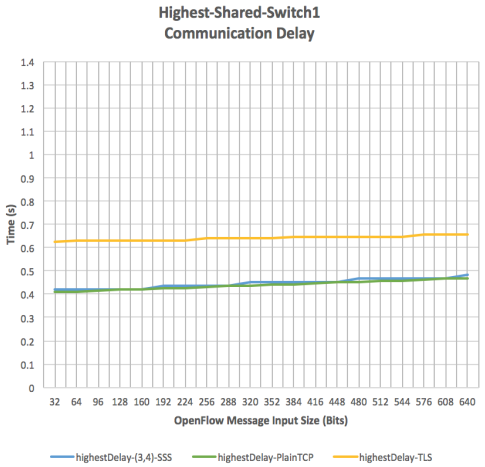


(d) Highest-Separated-S4

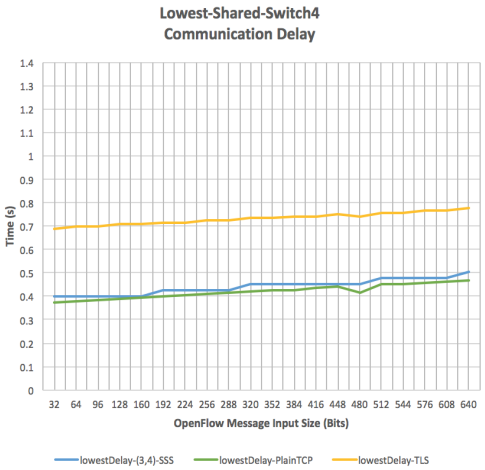
Figure 5.21: Communication cost in a scaled system for scenario 1 (not sharing control node)



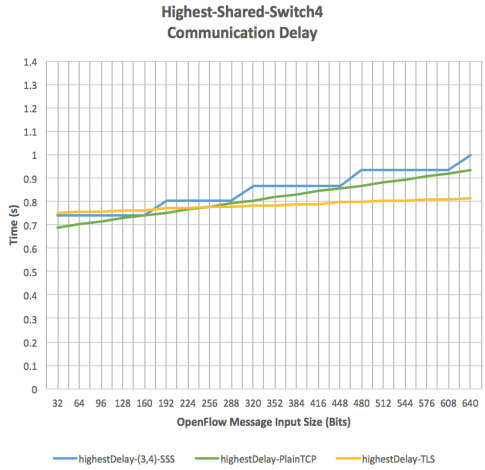
(a) Lowest-Shared-S1



(b) Highest-Shared-S1



(c) Lowest-Shared-S4



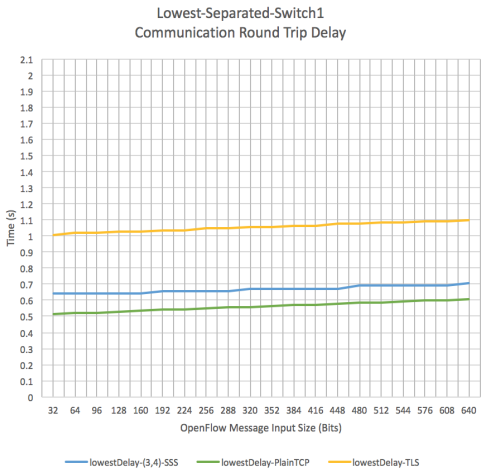
(d) Highest-Shared-S4

Figure 5.22: Communication cost in a scaled system for scenario 2 (sharing control node)

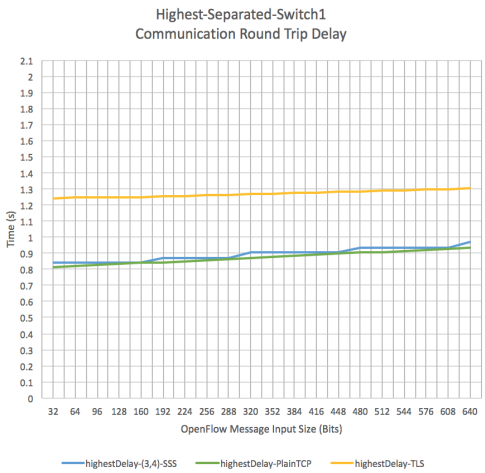
5.6.3 Round trip delay in a Scaled System

In this experiment we took the example network for both scenarios explained in section 5.6.2. We used the procedures as in section 5.5.10 plus the pattern matching cost at the reconstructor side of the controller by running (3,4)-SSS `bQSafeOF`, `notQSafeOF`, and single path OpenFlow, and recorded the communication delays. For all communication cost delay experiments we used data ranging from 32 to 32^{20} bits. The X-axis of the figures represents the data size in bits, and the Y-axis represents the output time in seconds.

In all cases, `bQSafeOF` performs better than `notQSafeOF`. `bQSafeOF`'s performance is compa-



(a) Lowest-Separated-S1



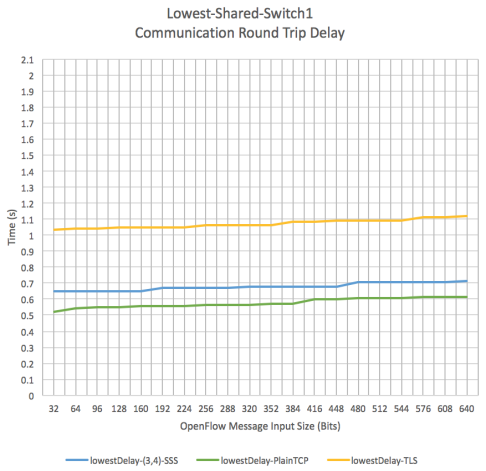
(b) Highest-Separated-S1

(c) Lowest-Separated-S4

(d) Highest-Separated-S4

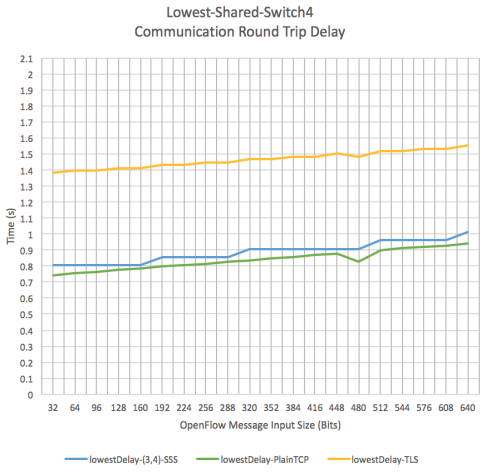
Figure 5.23: Round trip cost in a scaled system for scenario 1 (not sharing control node)

rable to single path TCP OpenFlow communication. The only case where `bQSafeOF` and `notQSafeOF` performs similar is at the Highest-Shared-S4 case (Figure 5.24d). This is because when there are four paths of increased delay where $Path1 < Path2 < Path3 < Path4$, the highest delay for `notQSafeOF` is $Path3$, but for `bQSafeOF` and single path TCP, it is $Path4$. Therefore, in the worst case, the performance of `notQSafeOF` can be comparable to the other two.

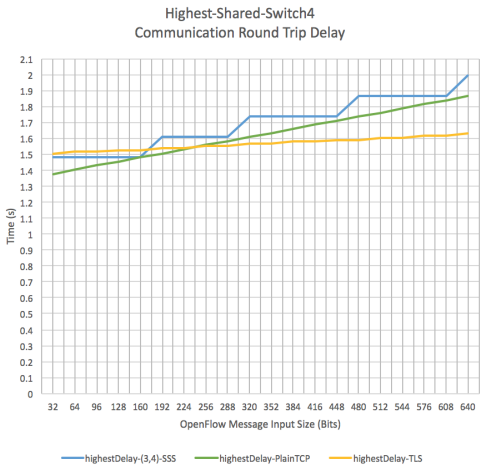


(a) Lowest-Shared-S1

(b) Highest-Shared-S1



(c) Lowest-Shared-S4



(d) Highest-Shared-S4

Figure 5.24: Round trip cost in a scaled system for scenario 2 (sharing control node)

5.7 Goal 3: Cost of MTD Technique

The goal of the MTD experiments is to find any operational cost due to IP change and path randomization.

5.7.1 IP Randomization Cost

In `mtdQSafeOF` the system IP changes randomly every t_ip seconds. *Operational Delay* is the delay measured after IP changes. The delay occurs because a new IPv6 address assignment goes

through a number of steps, such as Neighbor Solicitation or Duplicate Address Detection to avoid address duplication.

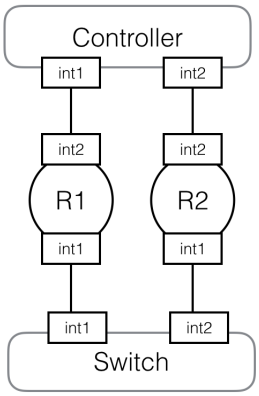


Figure 5.25: Experimental topology created in NS-3

We used NS-3 (Network Simulator). NS-3 offers a simulation environment where we can create nodes, connect them, assign protocol stack and addresses, install the application and run the simulation. The topology is illustrated in Figure 5.25 where a controller and a switch are connected through 2 paths. The controller and the switch each have two interfaces and have IPv6 address configured. Interfaces are connected to distinct routers where IPv6 routing is enabled. Here, the switch is acting as the client, and the controller is acting as the server. We have written code in NS-3 to simulate the topology. We run the program, and the switch connects to the TCP server to send bytes of data. The program outputs pcap tracing for every interface. We can analyze the pcap file using Wireshark. Our objective is to find the time delay after an IP change.

We record the delay for two cases; (i) the first time when the network is ready to communicate, and (ii) when there is an IP change after t_ip seconds. Figure 5.26 shows the Wireshark output for the case (i) where it took 0.019 seconds to assign an IP address and Figure 5.27 shows the delay for case (ii) where it took 1.003 seconds to change and assign a new IP address.

We did an experiment for the topology (Figure 5.25) when control nodes are not shared. First, we assigned IPv6 addresses to the switch and the controller. At this stage, we send 54 bytes from the first and second interfaces of the switch and measure the time just before it started to send the

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	2010:10::200:ff:fe00…	2001:1::200:ff:fe00:1	TCP	78	49153→9 [SYN] Seq=0 Win=32768 Len=0 WS=4 TSval=0 TSecr=0
2	0.002000	::	ff02::1	ICMPv6	74	Neighbor Solicitation for 2010:10::200:ff:fe00:3 from 00:00:00:00:00:03
3	0.003183	::	ff02::1	ICMPv6	74	Neighbor Solicitation for fe80::200:ff:fe00:3 from 00:00:00:00:00:03
4	0.017183	::	ff02::1	ICMPv6	74	Neighbor Solicitation for fe80::200:ff:fe00:4 from 00:00:00:00:00:04
5	0.019183	::	ff02::1	ICMPv6	74	Neighbor Solicitation for 2010:10::200:ff:fe00:4 from 00:00:00:00:00:04
6	0.064991	2001:1::200:ff:fe00:1	2010:10::200:ff:fe00:3	TCP	78	9→49153 [SYN, ACK] Seq=0 Ack=1 Win=32768 Len=0 WS=4 TSval=32 TSecr=0
7	0.064991	2010:10::200:ff:fe00…	2001:1::200:ff:fe00:1	TCP	74	49153→9 [ACK] Seq=1 Ack=1 Win=131072 Len=0 TSval=64 TSecr=32
8	0.066175	2010:10::200:ff:fe00…	2001:1::200:ff:fe00:1	TCP	128	49153→9 [FIN, ACK] Seq=1 Ack=1 Win=131072 Len=54 TSval=64 TSecr=32
9	0.132639	2001:1::200:ff:fe00:1	2010:10::200:ff:fe00:3	TCP	74	9→49153 [ACK] Seq=1 Ack=56 Win=131072 Len=0 TSval=100 TSecr=64
10	0.133823	2001:1::200:ff:fe00:1	2010:10::200:ff:fe00:3	TCP	74	9→49153 [FIN, ACK] Seq=1 Ack=56 Win=131072 Len=0 TSval=100 TSecr=64
11	0.133823	2010:10::200:ff:fe00…	2001:1::200:ff:fe00:1	TCP	74	49153→9 [ACK] Seq=56 Ack=2 Win=131072 Len=0 TSval=133 TSecr=100

Figure 5.26: Operational delay measurement- case (i)

Apply a display filter … <⌘/>

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	::	ff02::1	ICMPv6	74	Neighbor Solicitation for 2012:12::200:ff:fe00:5 from 00:00:00:00:00:05
2	0.006000	::	ff02::1	ICMPv6	74	Neighbor Solicitation for fe80::200:ff:fe00:5 from 00:00:00:00:00:05
3	0.020183	::	ff02::1	ICMPv6	74	Neighbor Solicitation for 2012:12::200:ff:fe00:6 from 00:00:00:00:00:06
4	0.022183	::	ff02::1	ICMPv6	74	Neighbor Solicitation for fe80::200:ff:fe00:6 from 00:00:00:00:00:06
5	1.003000	fe80::200:ff:fe00:5	ff02::2	ICMPv6	58	Router Solicitation from 00:00:00:00:00:05
6	29.9970…	2012:12::200:ff:fe00…	2001:1::200:ff:fe00:1	TCP	78	49155→9 [SYN] Seq=0 Win=32768 Len=0 WS=4 TSval=65000 TSecr=0
7	30.0619…	2001:1::200:ff:fe00:1	2012:12::200:ff:fe00:5	TCP	78	9→49155 [SYN, ACK] Seq=0 Ack=1 Win=32768 Len=0 WS=4 TSval=65032 TSecr=65000
8	30.0619…	2012:12::200:ff:fe00…	2001:1::200:ff:fe00:1	TCP	74	49155→9 [ACK] Seq=1 Ack=1 Win=131072 Len=0 TSval=65064 TSecr=65032
9	30.0631…	2012:12::200:ff:fe00…	2001:1::200:ff:fe00:1	TCP	128	49155→9 [FIN, ACK] Seq=1 Ack=1 Win=131072 Len=54 TSval=65064 TSecr=65032
10	30.1296…	2001:1::200:ff:fe00:1	2012:12::200:ff:fe00:5	TCP	74	9→49155 [ACK] Seq=1 Ack=56 Win=131072 Len=0 TSval=65100 TSecr=65064
11	30.1308…	2001:1::200:ff:fe00:1	2012:12::200:ff:fe00:5	TCP	74	9→49155 [FIN, ACK] Seq=1 Ack=56 Win=131072 Len=0 TSval=65100 TSecr=65064
12	30.1308…	2012:12::200:ff:fe00…	2001:1::200:ff:fe00:1	TCP	74	49155→9 [ACK] Seq=56 Ack=2 Win=131072 Len=0 TSval=65133 TSecr=65100

Figure 5.27: Operational delay measurement- case (ii)

TCP packet. Then we changed each interface's IP address five times and recorded the delay. Figure 5.28 shows the operational delay for IP changes. The initial delay for both interfaces is on average 0.02 seconds. For every IP change the delay is an average 1 second and it is almost constant for both interfaces. We conclude that IP change has a constant delay, but that is very negligible.

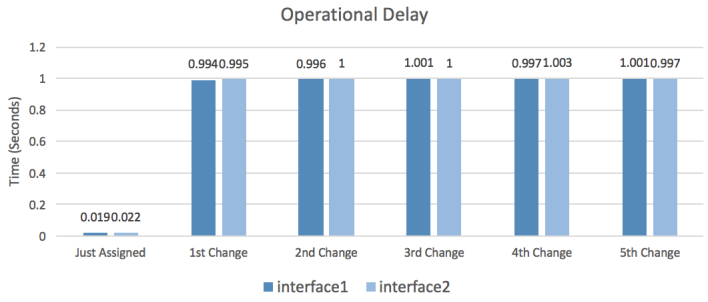


Figure 5.28: Operational delay - after IP change

5.7.2 IP Change Cost in a Scaled Shared Path System

We experimented with a scaled system (Figure 5.19b) with four switches and four control nodes where the control nodes are shared by the paths. First, we send 400 bits through all four paths to

find the path's speed. Our experiment result shows that path2 is the fastest and path3 is the slowest. Figure 5.29 shows the speeds.

Fastest to slowest path	Time (second)
2	1.1796
1	1.18937
4	1.22265
3	1.25547

Figure 5.29: Path speeds based on their data rate and delays

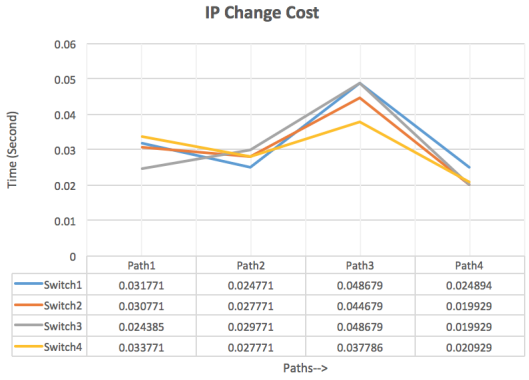


Figure 5.30: Operational delay - after IP change

For the IP randomization experiment, we changed IP addresses and recorded the delay. Figure 5.30 shows the result of the IP change delay. The IP change delay is also affected by the path speeds. From the figure, we note that for the paths passing through control node 3, the delay is the highest. This is because of the path 3's data rate and delay.

5.7.3 Path Randomization Cost

When more than n paths are available, the `mtdQSafeOF` path manager chooses any n paths randomly every t_pa seconds. In this experiment, m=6 paths with different data rates and delays are available between one controller and one switch. Figure 5.31 shows the fastest path to the slowest path while sending 400 bits. (2,3)-SSS runs at the switch, and the controller side and the change interval is 5 seconds.

(i) *Without path randomization*: First, we send 50752 bytes without path randomization. The

Fastest to slowest path	Time (second)
5	1.1182
2	1.1796
1	1.18937
6	1.1959
4	1.22265
3	1.25547

Figure 5.31: Fastest to slowest paths

system chooses three paths and sends 50752 bytes through each path. We capture the time when the controller receives the data.

	Path1		Path2		Path3	
With Path Randomization	r6-r3-r1	14.57679 s	r2-r4-r4	14.0612 s	r5-r6-r5	11.327 s
Without Path Randomization	r6	11.4623 s	r2	9.0506 s	r5	8.2827 s

Figure 5.32: Path randomization cost

(ii) *With path randomization*: In the second experiment, we send the same data and randomize paths uniformly every 5 seconds. Every 5 seconds the system chooses different sets of paths and sends the remaining data. We record the time when the controller receives the full amount of data.

Figure 5.32 shows the results with and without path randomization. With randomization, the cost is higher than without randomization since it selects three routes for each path and every path's delay is different.

5.8 Summary

The section details the implementation of `bQSafeOF`, `mtdQSafeOF` and `notQSafeOF`. We also described the experiment goals, and design of the experiments.

There were three experiment goals and we summarize our results.

(i) To compare our basic module `bQSafeOF` with the other solution `notQSafeOF`.

In our experiment (section 5.5.1), the *computation cost* of (2,3)-SSS packet construction is 10 times slower than the AES-128-CBC encryption since ours is a non-optimal code. In the experiment (section 5.5.4), (2,3)-SSS packet re-construction is 110 times slower than the AES-128-CBC

decryption. This is because, AES-128-CBC of openSSL is an optimized program.

The *communication delay* and *round trip delay* were measured for a fixed setup (section 5.5.8) with four disjoint paths between the switch and the controller where each path had different data rates and delays. The result shows `bQSafeOF`'s communication cost and round trip cost are average 1.6 times faster than the `notQSafeOF`. It is because `bQSafeOF`'s creates share before the communication starts but in case of `notQSafeOF` the encryption or decryption cannot be separated from the communication. For the communication cost and the round trip cost, we observed that it is dependent on the path's data rate and delay.

(ii) To measure `bQSafeOF`'s performance for one controller and $\mathscr{S}$ number of switches.

The system settings include four switches connected to one controller. Each switch has four disjoint paths with different data rates and delay. Four switches share one control node for each path. The experiment setup is illustrated in Figure 5.19. In our experiments, for communication delay (section 5.22) and round trip delay (section 5.24), `bQSafeOF` is average 1.6 times faster than `notQSafeOF`. We observe that the performance can be greatly affected by how paths are sharing control nodes and the path's data rate and delay.

(iii) To measure the operational cost for randomizing IP and path in `mtdQSafeOF`.

In the first experiment (section 5.7.1), there were disjoint paths between one switch and one controller. Everytime the system changes an IP, it took sometime to assign a new IP. We observed that the delay for assigning IP for a fixed path with fixed data rate and delay was almost constant and very negligible (Figure 5.28). Then our second experiment (section 5.7.2) took place in a network where there were four switches connected to one controller. Each switch had four paths with different delay and data rate and paths of different switches were sharing control nodes. Our observation from the results conclude that IP randomization is affected by the path's data rate and delay. It is also affected by how paths are connected to a control node (Figure 5.30). Our third experiment (section 5.7.3) compares the cost when paths are randomized and when not randomized. For a fixed setup, we observed that the path randomization is adding extra delay. In

our experiment, path randomization delay was 1.2 times than without path randomization (Figure 5.32).

Chapter 6

Remarks

6.1 Conclusion

The recent announcement by NSA [15] and the recommendation to move to quantum-safe crypto algorithms followed by the NIST report [30] that advocates the plan for moving towards quantum-safe cryptography has generated much interest in designing systems that stay secure even if a quantum computer exists. We consider a quantum-safe approach that uses physical assumptions instead. That is, we assume there are multiple network paths between a controller and a switch and not all of them can be captured by the adversary at the same time. For reliability in all cases, one needs to assume multiple paths. So our proposed solution can be seen as employing multiple paths for providing both security and reliability.

From our background study we summarize that the underlying network has multiple paths available, but due to the current protocols assumption on single path communication and inflexibility of path management, the additional paths are unused. Recent technology trends, for example, virtualization, software-defined networking open a possibility to manage paths effectively.

Software-defined networking provides flexibility in providing network management. SDN can be used to improve the traditional network's security. Google has deployed SDN in their WAN architecture to gain faster performance and easy operability. However, the separation of control and data plane introduces security threats for SDN itself. We have observed that the crucial addition is the southbound interface that transmits network configuration data between the controller and the switch. The widely accepted southbound protocol is OpenFlow protocol. The OpenFlow switch establishes an OpenFlow channel with the controller using TCP/IP connection, and it is vice-versa. The controllers can be placed in a geographically distant location or one data center. There can be a network between the switch and the controller that is referred as a control network. Currently,

the only security suggestion is to apply TLS. Our study finds that there are multiple paths available between the controller and the switch and we will use those to provide confidentiality and reliability of the OpenFlow communication using Shamir Secret Sharing. Shamir secret sharing was widely proposed for distributed storage management, authentication and mobile ad-hoc networks (MANET), but not in a communication system.

Our first solution `bQSafeOF` uses (t,n)-SSS to create n shares of an OpenFlow message and sends each share along one disjoint path between the OpenFlow switch and the OpenFlow controller. A path manager assigns paths to the switch and the controller. The path assignment is opaque to the adversary. We assume that the adversary can only eavesdrop or block (DDoS attack) the paths but cannot tamper with the message. The scheme provides confidentiality when the adversary gets access to $(t-1)$ paths and reliability when $(n-t)$ paths are blocked or jammed by the adversary. The difference to the traditional network is that, in a traditional network, the attacker needs to get only one IP to target a specific switch-controller communication. In the case of `bQSafeOF`, the attacker needs to know at least t IP addresses or path identifiers to target a specific switch-controller communication. This way our system increases the attacker's effort. However, for a system that has static IP address or port numbers, we assume that at some time, the adversary can build the IP mapping table and therefore the probability of successfully targetting a specific switch-controller communication becomes one. In our design, the basic system provides confidentiality and reliability up to a sharp threshold number. If the attacker can exploit more than the threshold number of paths, the system loses its confidentiality and reliability completely.

Moving target (MT) techniques frequently change the system parameters to lessen the probability of a successful *targeted attack* and limit the *span of an attack*. The technique creates *additional uncertainty* for attackers by dynamically changing system parameters that make systems less static and less deterministic to increase attackers workload. Dynamic network modifications often focus on frequent changes to addresses and ports but might also include rotating protocols and changing the logical network topology. Our hypothesis is that using MTD techniques, the system can cre-

ate a more randomized view to the attacker, thus going beyond the sharp threshold would restrict attacker to identify the target.

Our second system `mtdQSafeOF` uses two randomization techniques along with `bQSafeOF`; (i) *IP randomization:* that generates dynamic random IP addresses for *clients*. Here, the controller and the switch act as *clients* when they send a message and act as *servers* when they receive messages. The dynamic assignment changes at every t_ip time, and (ii) *Path assignment randomization:* When there are $m > n$ paths, PM can choose n paths randomly out of m paths at every t_pa time. The randomization techniques restrict the attacker to have the switch-IP mapping table and launch an attack on the specific switch-controller communication. The security analysis shows that the system security is not completely lost using the MTD technique.

We also provide an alternate design `notQSafeOF` using TLS and a sufficient number of extra paths to compare the operational performance of our proposed solutions. We note that `notQSafeOF` is not post-quantum safe while our solutions are. The experimental results are promising by showing that our scheme `bQSafeOF` performs better than the alternate solution `notQSafeOF`. `mtdQSafeOF` has minimal overhead for the IP randomization and path randomization techniques.

6.2 Future Work

An immidiate future step is to implement `bQSafeOF` and `mtdQSafeOF` on vendor specified switch and controllers and evaluate its effectiveness in a working network. The `bQSafeOF` enables a direction to design networking protocol for multipath setup.

Bibliography

[1] 2016 Internet crime report. https://pdf.ic3.gov/2016_IC3Report.pdf. Accessed: 2017-09-05.

[2] Coursera SDN module 4 OpenFlow Firewall, https://gist.github.com/wh5a/6015943. https://gist.github.com/wh5a/6015943.

[3] Floodlight OpenFlow controller - project Floodlight. http://www.projectfloodlight.org/floodlight/.

[4] Juniper networks, opencontrail. http://www.opencontrail.org. Accessed: July 03, 2017.

[5] Libgfshare package in Ubuntu, https://launchpad.net/ubuntu/+source/libgfshare. https://launchpad.net/ubuntu/+source/libgfshare. Accessed: 201705-10.

[6] Open vSwitch. http://openvswitch.org/. Accessed: 2017-09-20.

[7] The opendaylight platform — opendaylight. https://www.opendaylight.org/. Accessed: July 03, 2017.

[8] pica8. http://www.pica8.com/. Accessed: 2017-09-20.

[9] Shor's algorithm - wikipedia. https://en.wikipedia.org/wiki/Shor%27s_algorithm. Accessed: June 29, 2017.

[10] What is the Brocade SDN Controller (bsc)? https://www.sdxcentral.com/sdn/definitions/sdn-controllers/opendaylight-controller/brocade-vyatta-controller/. Accessed: 2017-07-03.

[11] HP SDN Controller architecture, technical solution guide. Technical report, 2013.

[12] Migration use cases and methods. Technical report, Migration Working Group, Open Networking Foundation, 2014.

[13] POX. `http://www.noxrepo.org/pox/about-pox/`, Dec 2014.

[14] SDN architecture, Open Networking Foundation. Technical report, June 2014.

[15] N. S. Agency. CNSA suite and quantum computing FAQ, January 2016.

[16] A. Akhunzada, A. Gani, N. B. Anuar, A. Abdelaziz, M. K. Khan, A. Hayat, and S. U. Khan. Secure and dependable software defined networks. *Journal of Network and Computer Applications*, 61:199–221, 2016.

[17] E. Al-Shaer, Q. Duan, and J. Jafarian. Random host mutation for moving target defense. In A. Keromytis and R. Di Pietro, editors, *Security and Privacy in Communication Networks*, volume 106 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 310–327. Springer Berlin Heidelberg, 2013.

[18] E. Al-Shaer, Q. Duan, and J. H. Jafarian. Random host mutation for moving target defense. In *SecureComm*, pages 310–327. Springer, 2012.

[19] M. R. Albrecht and K. G. Paterson. Lucky microseconds: A timing attack on amazon's s2n implementation of tls. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 622–643. Springer, 2016.

[20] M. Antikainen, T. Aura, and M. Särelä. Spook in your network: Attacking an SDN with a compromised OpenFlow switch. In *Nordic Conference on Secure IT Systems*, pages 229–244. Springer, 2014.

[21] N. Aviram, S. Schinzel, J. Somorovsky, N. Heninger, M. Dankel, J. Steube, L. Valenta, D. Adrian, J. A. Halderman, V. Dukhovni, E. Käsper, S. Cohney, S. Engels, C. Paar, and Y. Shavitt. Drown: Breaking TLS using SSLv2. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 689–706, Austin, TX, 2016. USENIX Association.

[22] P. Bailis and K. Kingsbury. The network is reliable. *Queue*, 12(7):20:20–20:32, July 2014.

[23] J. N. Bakker, I. Welch, and W. K. Seah. Network-wide virtual firewall using SDN/OpenFlow. In *Network Function Virtualization and Software Defined Networks (NFV-SDN), IEEE Conference on*, pages 62–68. IEEE, 2016.

[24] C. Banse and S. Rangarajan. A secure northbound interface for SDN applications. In *Trustcom/BigDataSE/ISPA, 2015 IEEE*, volume 1, pages 834–839. IEEE, 2015.

[25] E. B. Barker and A. L. Roginsky. (draft) sp 800-131a, transitions: Recommendation for transitioning the use of cryptographic algorithms and key lengths. Technical report, Gaithersburg, MD, United States, July 2015.

[26] N. Beheshti and Y. Zhang. Fast failover for control traffic in software-defined networks. In *Global Communications Conference (GLOBECOM), 2012 IEEE*, pages 2665–2670. IEEE, 2012.

[27] K. Benton, L. J. Camp, and C. Small. OpenFlow vulnerability assessment. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, pages 151–152, New York, NY, USA, 2013. ACM.

[28] D. Bogdanov, S. Laur, and J. Willemson. Sharemind: A framework for fast privacy-preserving computations. In *Proceedings of the 13th European Symposium on Research in Computer Security: Computer Security*, ESORICS '08, pages 192–206, Berlin, Heidelberg, 2008. Springer-Verlag.

[29] A. R. Chavez, W. M. S. Stout, and S. Peisert. Techniques for the dynamic randomization of network attributes. In *2015 International Carnahan Conference on Security Technology (ICCST)*, pages 1–6, Sept 2015.

[30] L. Chen, S. Jordan, Y.-K. Liu, D. Moody, R. Peralta, R. Perlner, and D. Smith-Tone. Report on post-quantum cryptography. *National Institute of Standards and Technology Internal*

Report, 8105, 2016.

[31] Q. Chen, Y.-P. P. Chen, S. Zhang, and C. Zhang. *Detecting Collusion Attacks in Security Protocols*, pages 297–306. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[32] T. Chin, X. Mountrouidou, X. Li, and K. Xiong. Selective packet inspection to detect DoS flooding using software defined networking (SDN). In *2015 IEEE 35th International Conference on Distributed Computing Systems Workshops*, pages 95–99, June 2015.

[33] A. Clark, K. Sun, and R. Poovendran. Effectiveness of IP address randomization in decoy-based moving target defense. In *Decision and Control (CDC), 2013 IEEE 52nd Annual Conference on*, pages 678–685. IEEE, 2013.

[34] R. Cramer, R. Gennaro, and B. Schoenmakers. A secure and optimally efficient multi-authority election scheme. *Transactions on Emerging Telecommunications Technologies*, 8(5):481–490, 1997.

[35] Y. Cui, H. Wang, X. Cheng, and B. Chen. Wireless data center networking. *IEEE Wireless Communications*, 18(6), 2011.

[36] W. Dai. Crypto++ 5.6.5 free C++ class library of cryptographic schemes. `https://www.cryptopp.com/docs/ref/class_secret_sharing.html`. Accessed: 201705-10.

[37] Y. Desmedt and Y. Frankel. *Shared generation of authenticators and signatures*, pages 457–469. Springer Berlin Heidelberg, Berlin, Heidelberg, 1992.

[38] Y. G. Desmedt. Threshold cryptography. *Transactions on Emerging Telecommunications Technologies*, 5(4):449–458, 1994.

[39] M. Dhawan, R. Poddar, K. Mahajan, and V. Mann. Sphinx: Detecting security attacks in software-defined networks. In *NDSS*. The Internet Society, 2015.

[40] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246, RFC Editor, August 2008.

[41] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numer. Math.*, 1(1):269–271, Dec. 1959.

[42] S. Dolev and S. T. David. SDN-based private interconnection. In *2014 IEEE 13th International Symposium on Network Computing and Applications*, pages 129–136, Aug 2014.

[43] A. Doria, J. H. Salim, R. Haas, H. Khosravi, W. Wang, L. Dong, R. Gopal, and J. Halpern. Forwarding and Control Element Separation (ForCES) Protocol Specification. RFC 5810, RFC Editor, March 2010.

[44] Q. Duan, Y. Wang, F. Mohsen, and E. Al-Shaer. Private and anonymous data storage and distribution in cloud. In *2013 IEEE International Conference on Services Computing*, pages 264–271, June 2013.

[45] D. Erickson. The Beacon OpenFlow controller. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, pages 13–18, New York, NY, USA, 2013. ACM.

[46] L. Ertaul and N. Chavan. Security of ad hoc networks and threshold cryptography. In *Wireless Networks, Communications and Mobile Computing, 2005 International Conference on*, volume 1, pages 69–74. IEEE, 2005.

[47] D. Evans, A. Nguyen-Tuong, and J. Knight. Effectiveness of moving target defenses. In S. Jajodia, A. K. Ghosh, V. Swarup, C. Wang, and X. S. Wang, editors, *Moving Target Defense*, volume 54 of *Advances in Information Security*, pages 29–48. Springer New York, 2011.

[48] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure. The Open vSwitch Database Management Protocol. RFC 6824, RFC Editor, January 2013.

[49] O. N. Foundation. OpenFlow switch specification version 1.5.1 (protocol version 0x06). https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.1.pdf, Year = 2015, March 26.

[50] B. I. T. Frisch. An algorithm for vertex-pair connectivity. *International Journal of Control*, 6(6):579–593, 1967.

[51] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Robust threshold DSS signatures. *Inf. Comput.*, 164(1):54–84, Jan. 2001.

[52] M. C. Gorla, V. M. Kamaraju, and E. K. Çetinkaya. Network attack experimentation using OpenFlow-enabled GENI testbed. 2015.

[53] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker. NOX: Towards an operating system for networks. *SIGCOMM Comput. Commun. Rev.*, 38(3):105–110, July 2008.

[54] N. Hamedazimi, Z. Qazi, H. Gupta, V. Sekar, S. R. Das, J. P. Longtin, H. Shah, and A. Tanwer. Firefly: A reconfigurable wireless data center fabric using free-space optics. *ACM SIGCOMM Computer Communication Review*, 44(4):319–330, 2015.

[55] L. Harn. Group-oriented (t, n) threshold digital signature scheme and digital multisignature. *IEE Proceedings-Computers and Digital Techniques*, 141(5):307–313, 1994.

[56] L. Harn. Group authentication. *IEEE Transactions on Computers*, 62(9):1893–1898, Sept 2013.

[57] B. Heller, R. Sherwood, and N. McKeown. The controller placement problem. In *Proceedings of the first workshop on Hot topics in software defined networks*, pages 7–12. ACM, 2012.

[58] R. Hinden and S. Deering. IP Version 6 Addressing Architecture. RFC 4291, RFC Editor, February 2006.

[59] Y. Hu, W. Wendong, X. Gong, X. Que, and C. Shiduan. Reliability-aware controller placement for software-defined networks. In *2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*, pages 672–675. IEEE, 2013.

[60] J. H. Jafarian, E. Al-Shaer, and Q. Duan. OpenFlow random host mutation: Transparent moving target defense using software defined networking. In *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, HotSDN '12, pages 127–132, New York, NY, USA, 2012. ACM.

[61] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat. B4: Experience with a globally-deployed software defined WAN. *SIGCOMM Comput. Commun. Rev.*, 43(4):3–14, Aug. 2013.

[62] H. T. Karaoglu, M. B. Akgun, M. H. Gunes, and M. Yuksel. Multi path considerations for anonymized routing: Challenges and opportunities. In *New Technologies, Mobility and Security (NTMS), 2012 5th International Conference on*, pages 1–5. IEEE, 2012.

[63] R. Kloti, V. Kotronis, and P. Smith. OpenFlow: A security analysis. In *Network Protocols (ICNP), 2013 21st IEEE International Conference on*, pages 1–6, Oct 2013.

[64] D. E. Knuth, J. James H. Morris, and V. R. Pratt. Fast pattern matching in strings. *SIAM Journal on Computing*, 6(2):323–350, 1977.

[65] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Z. Google, R. Ramanathan, Y. I. NEC, H. I. NEC, T. H. NEC, and S. Shenker. Onix: a distributed control platform for large-scale production networks. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, pages 351–364, Berkeley, CA, USA, 2010.

[66] D. Kreutz, F. M. Ramos, and P. Verissimo. Towards secure and dependable software-defined networks. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, pages 55–60, New York, NY, USA, 2013. ACM.

[67] D. Kreutz, F. M. V. Ramos, P. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig. Software-defined networking: A comprehensive survey. *CoRR*, abs/1406.0440, 2014.

[68] J.-H. Lam, S.-G. Lee, H.-J. Lee, and Y. E. Oktian. Securing distributed SDN with IBC. In *2015 Seventh International Conference on Ubiquitous and Future Networks*, pages 921–925. IEEE, 2015.

[69] 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*, Hotnets-IX, pages 19:1–19:6, New York, NY, USA, 2010. ACM.

[70] H. Y. Lin and W. G. Tzeng. A secure erasure code-based cloud storage system with secure data forwarding. *IEEE Transactions on Parallel and Distributed Systems*, 23(6):995–1003, June 2012.

[71] M. Liyanage, M. Ylianttila, and A. Gurtov. Securing the control channel of software-defined mobile networks. In *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2014 IEEE 15th International Symposium on a*, pages 1–6. IEEE, 2014.

[72] W. Lou, W. Liu, and Y. Fang. Spread: Enhancing data confidentiality in mobile ad hoc networks. In *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, volume 4, pages 2404–2413. IEEE, 2004.

[73] W. Lou, W. Liu, Y. Zhang, and Y. Fang. Spread: Improving network security by multipath routing in mobile ad hoc networks. *Wireless Networks*, 15(3):279–294, 2009.

[74] S. Matsumoto, S. Hitz, and A. Perrig. Fleet: defending SDNs from malicious administrators. In *Proceedings of the third workshop on Hot topics in software defined networking*, pages

103–108. ACM, 2014.

[75] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. OpenFlow: Enabling innovation in campus networks. In *ACM SIGCOMM Computer Communications Review*, April, 2008.

[76] S. A. Mehdi, J. Khalid, and S. A. Khayam. Revisiting traffic anomaly detection using software defined networking. In *International Workshop on Recent Advances in Intrusion Detection*, pages 161–180. Springer, 2011.

[77] S. Namal, I. Ahmad, A. Gurtov, and M. Ylianttila. Enabling secure mobility with OpenFlow. In *Future Networks and Services (SDN4FNS), 2013 IEEE SDN for*, pages 1–5. IEEE, 2013.

[78] R. Nishimura, S.-i. Abe, N. Fujita, and Y. Suzuki. Reinforcement of VoIP security with multipath routing and secret sharing scheme. *Journal of Information Hiding and Multimedia Signal Processing*, 1(3):204–219, 2010.

[79] H. Okhravi. Survey of cyber moving targets. Technical Report 1166, MIT Lincoln Laboratory, Sep. 2013.

[80] H. Okhravi, T. Hobson, D. Bigelow, and W. Streilein. Finding focus in the blur of moving-target techniques. *Security and Privacy, IEEE*, 12(2):16–26, Mar 2014.

[81] M. Omar, Y. Challal, and A. Bouabdallah. Reliable and fully distributed trust model for mobile ad hoc networks. *Computers and Security*, 28(3):199 – 214, 2009.

[82] B. Pfaff and B. Davie. The Open vSwitch Database Management Protocol. RFC 7047, RFC Editor, December 2013.

[83] B. Poettering. Shamir's secret sharing scheme, http://point-at-infinity.org/ssss/. `http://point-at-infinity.org/ssss/`. Accessed: 2017-05-10.

[84] J. Qadir, A. Ali, K.-L. A. Yau, A. Sathiaseelan, and J. Crowcroft. Exploiting the power of multiplicity: a holistic survey of network-layer multipath. *IEEE Communications Surveys & Tutorials*, 17(4):2176–2213, 2015.

[85] Z. A. Qazi, C.-C. Tu, L. Chiang, R. Miao, V. Sekar, and M. Yu. Simple-fying middlebox policy enforcement using SDN. *ACM SIGCOMM computer communication review*, 43(4):27–38, 2013.

[86] S. Scott-Hayward, G. O'Callaghan, and S. Sezer. SDN security: A survey. In *Future Networks and Services (SDN4FNS), 2013 IEEE SDN For*, pages 1–7. IEEE, 2013.

[87] A. Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, Nov. 1979.

[88] S. Shin and G. Gu. Attacking software-defined networks: A first feasibility study. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, pages 165–166, New York, NY, USA, 2013. ACM.

[89] P. W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *Foundations of Computer Science, 1994 Proceedings., 35th Annual Symposium on*, pages 124–134. IEEE, 1994.

[90] S. Takahashi and K. Iwamura. Secret sharing scheme suitable for cloud computing. In *Proceedings of the 2013 IEEE 27th International Conference on Advanced Information Networking and Applications*, AINA '13, pages 530–537, Washington, DC, USA, 2013. IEEE Computer Society.

[91] C. Tankard. Advanced persistent threats and how to monitor and deter them. *Network Security*, 2011(8):16 – 19, 2011.

[92] A. Teixeira, D. Pérez, H. Sandberg, and K. H. Johansson. Attack models and scenarios for networked control systems. In *Proceedings of the 1st international conference on High Confidence Networked Systems*, pages 55–64. ACM, 2012.

[93] S. Thomson, T. Narten, and T. Jinmei. IPv6 Stateless Address Autoconfiguration. RFC 4862, RFC Editor, September 2007.

[94] A. D. Wyner. The wire-tap channel. *Bell Labs Technical Journal*, 54(8):1355–1387, 1975.

[95] T. Xing, D. Huang, L. Xu, C. J. Chung, and P. Khatkar. Snortflow: An OpenFlow-based intrusion prevention system in cloud environment. In *2013 Second GENI Research and Educational Experiment Workshop*, pages 89–92, March 2013.

[96] J. Yackoski, J. Li, S. A. DeLoach, and X. Ou. Mission-oriented moving target defense based on cryptographically strong network dynamics. In *Proceedings of the Eighth Annual Cyber Security and Information Intelligence Research Workshop*, CSIIRW '13, pages 57:1–57:4, New York, NY, USA, 2013. ACM.

[97] T. Ye, D. Veitch, and J. Bolot. Improving wireless security through network diversity. *SIGCOMM Comput. Commun. Rev.*, 39(1):34–44, Dec. 2008.

[98] S. Yeganeh, A. Tootoonchian, and Y. Ganjali. On scalability of software-defined networking. *IEEE Communications Magazine*, 2(51):136–141, 2013.

[99] L. Zhou and Z. J. Haas. Securing ad hoc networks. *IEEE Network*, 13(6):24–30, 1999.