Security in the Software Defined Networking Infrastructure

A thesis submitted by

Peter Edge

for the award of

Master of Science (Research)

July 2019

Abstract

Software Defined Networks (SDN) is a paradigm in which control and data planes of traditional networking devices are decoupled to form a distrubuted model. Communication between the separate planes requires a protocol such as OpenFlow to leverage programmable routing and forwarding decisions on the network. In this model, Application Programmable Interfaces (APIs) make it possible to inject policy and forwarding rules via the control plane or controller. The most prominent challenges resulting from the separation is link security between the separated elements through which private network data is now traversing.

One main area of concern is the method of transmission with which the majority of Open-Source controllers currently communicate. The preferred practice is for a Transport Layer Security (TLS) channel initiation by an OpenFlow switch wishing to communicate with a controller. Many developers have replaced the TLS method of communication with straight Transport Control Protocol (TCP) due to handshake sequence issues caused by certificate exchange during the TLS connection phase.

This thesis and the subsequent research will ask questions on security around the controller to device links that pass flow tables , network abstractions and multi-layer information to multiple controlled network elements.

The main objective of this research is to develop testing procedures that allow for accurate and repeatable experiments. Therefore, in researching security vulnerabilities between controllers and forwarding devices, benchmarking performed on secure links tests the capability of authentication mechanisms to function properly under load.

The outcomes of this research include a series of quality industry standard tests to benchmark typical SDN controllers and forwarding devices. A critical analysis of typical devices at low, medium and high loads. An SDN security taxonomy is presented to help with future categorising of device testing in context of SDN architecture.

Declaration

This Thesis is entirely the work of **Peter Edge** except where otherwise acknowledged. The work is original and has not previously been submitted for any other award, except where acknowledged.

Principal Supervisor: **Dr Zhongwei Zhang**

Associate Supervisor: **Dr David Lai**

Student and supervisors signatures of endorsement are held at the University

Peter Edge
July 2019

Acknowledgements

In the process of preparing and submitting this thesis, I would like to acknowledge the following:

- The Australian government. Participation in the Research Training Program (RTP) made it possible for me to study at a highly regarded university such as the University of Southern Queensland.
- Ara Institute of Canterbury. The testing and analytical research infrastructure required to complete this study was provided by the Ara Cisco Networking Academy. It would not have been possible without the equipment.
- Merran Laver of Eyline Editing for the professional service, expert proofreading and editing.

I would like to make a special acknowledgement to my supervisory team. My Principal Supervisor Dr Zhongwei Zhang for support and guidance from the beginning of my studies. I am very grateful for his patience and understanding. My Associate Supervisor Dr David Lai who was seconded to the role midway through my degree. David has been very supportive over this period.

Associated publications

Davar, Z., Edge, P. and Zhang, Z.[2018] ’SDN Managed Hybrid IoT as a Service’ CloudCom 2018 (IEEE International Conference on Cloud Computing Technology and Science 2018) (Peer Reviewed)

IARIA International Conference on Networking and Services 2019 (Published)

International Journal on Advances in Networks and Services 2019 vol 12 n 1&2.

Invitation to submit and publish extended version of the paper to the International Journal On Advances in Internet Technology, v13 n 1&2 2020.

Table of contents

List of figures ix

List of tables xi

List of Abbreviations xii

1 Introduction 1

1.1 Background and Motivation . . . 1
1.2 Problem statement . . . 3
1.3 Aims and Objectives . . . 4
1.4 Structure of the thesis . . . 6

2 SDN Literature - History, Threats and Security 7

2.1 History . . . 7
2.1.1 Control Data Separation . . . 11
2.1.2 OpenFlow and Network Operating Systems . . . 11
2.1.3 Network Virtualisation . . . 12
2.2 Cloud computing . . . 14
2.3 Data centres . . . 15
2.4 Connectivity through providers . . . 16
2.5 Software Defined Networking . . . 19
2.6 Threat modelling . . . 20
2.7 Threats . . . 22
2.7.1 Table of threats . . . 22
2.7.2 Spoofing . . . 22
2.7.3 Tampering . . . 23
2.7.4 Repudiation . . . 23
2.7.5 Information disclosure . . . 23

2.7.6 Denial of service 23
2.7.7 Elevation of privileges 24
2.8 Severity and location of attacks 24
2.9 Security and SDN 24
2.9.1 Security for SDN 24

3 OpenFlow Protocols and Virtual Environments 26
3.1 The Open Flow protocol 26
3.2 OpenFlow operation 27
3.3 Switch components 29

4 Testing Methods and Equipment Used 41
4.1 Guidelines for openFlow device behaviour in the test environment 41
4.2 OpenFlow controller capacity 43
4.2.1 Scenario 1 Load balancing 43
4.2.2 Scenario 2 Forced TABLE_MISS 43
4.2.3 Scenario 3 Controller failure 43
4.3 Cisco OpenFlow beta Testing Trial 44
4.4 Equipment Required 45
4.4.1 OpenVswitch 45
4.4.2 VMware ESXi 45
4.4.3 Mininet 45
4.4.4 OpenFlow controllers 46

5 Testing and Performance Evaluations 48
5.1 Attacking the network 48
5.1.1 Reconnaissance 48
5.1.2 Attack 51
5.1.3 Attacks and prevention 52
5.2 Testing 53
5.2.1 Performance evaluation 54
5.2.2 SDN Utilisation and Adoption 56

6 Conclusion and Future Directions 59
6.1 Recommendation for Future Works 59

References 63

Appendix A Attack Scripts **66**
A.1 Reconnaissance phase 66
A.1.1 arpmon.py 66
A.1.2 sdn-detect.py 68
A.1.3 controller-detect.py 71
A.2 Attack Phase 76
A.2.1 lldp-replay.py 76

Appendix B Data Sheets and Specifications **78**
B.1 Cisco 9300 Switch 78
B.1.1 Data Sheet 78
B.1.2 The Fondation of Software-Defined access 79
B.1.3 Built for security, IoT, mobility, and cloud 79
B.2 OpenFlow Port Specifications 80
B.2.1 Required Ports 80
B.2.2 Physical Ports 80
B.2.3 Logical Ports 80
B.2.4 Reserved Ports 81

List of figures

1.1 Control and data plane in one device 5

2.1 Innovation timeline for programmability in Networking 10
2.2 GENI 12
2.3 PlanetLab 12
2.4 FIRE 12
2.5 NTT SD WAN Architecture (ONF TR -506 2014) 17
2.6 Google B4 network traffic optimisation (ONF TR - 506 2014) 18
2.7 An overview of SDN operation flow and control (ONF - 521 2016) 19
2.8 SDN De-coupled view of control and data planes (ONF - 521 2016) 20
2.9 SDN Controller Logic (Edge.P 2018) 20
2.10 Traditional and SDN threat vectors (Edge.P 2018) 21

3.1 The OpenFlow Protocol Components (ONF TS - 215 2016) 27
3.2 Data flow through the OpenFlow Switch (ONF TS - 215 2016) 28
3.3 Pipeline processing 30
3.4 OpenDayLight Hello 32
3.5 Network byte order (Edge.P 2018) 33
3.6 OpenDayLight Hello reply packet 35
3.7 OpenDaylight features request packet 36
3.8 OpenDaylight Hello features reply packet 36
3.9 Multiple connections TCP/TLS (Flowgrammable.com 2017) 38

4.1 Mininet editor topology (Edge.P 2018) 46

5.1 GNS3 Topology for attacking OpenFlow controllers 49
5.2 OpenDaylight Cbench test plot 55
5.3 Ryu Cbench test plot 55

5.4 SDN security taxonomy 58

List of tables

2.1 Table of Threats 22

3.1 Main components of a flow entry in a flow table 29
3.2 OpenFlow message Types 34
3.3 Version capabilities for the OpenFlow protocol 35

4.1 Controller Configuration Parameters 42
4.2 Switch configuration parameters 44

List of Abbreviations

Acronyms / Abbreviations

ASICS Application Specific Integrated Circuits

ATAWAD Anytime Anywhere Any Device

ATM Asynchronous Transfer Mode

BGP Border Gateway Protocol

CLI Command Line Interface

CA Certificate Authority

CPU Central Processing Unit

CORD Central Office Re-Defined as a Data Centre

DHCP Dynamic Host Configuration Protocol

DARPA Defence Advanced Research Projects Agency

DOS Denial of Service

DDOS Distributed Denial of Service

DTLS Datagram Transport Layer Security

EU European Union

FIRE Future Internet Research Experimentation

GUI Graphical User Interface

HA High Availability

IaaS Infrastructure as a Service

ICT Information & Communications Technologies

IGP Interior Gateway Protocol

IETF Internet Engineering Task Force

IEEE Institute of Electrical and Electronics Engineers

ISP Internet Service Provider

LAN Local Area Network

OSPF Open Shortest Path First

NETCONF Network Configuration Protocol

SNMP Simple Network Configuration Protocol

MIB Message Information Base

RCP Routing Control Platform

TED Traffic Engineering Database

MPLS Multi Protocol Label Switching

RFC Request for Comment

PCE Path Computation Execution

NOS Network Operating System

ONF Open Networking Foundation

LSP Label Switching Path

NSF National Science Foundation

RIB Routing Information Base

GENI Global Environment for Network Information

ISP Internet Service Provider

PaaS Platform as a Service

SaaS Software as a Service

IP Internet Protocol

TCP Transport Control Protocol

API Application Programmable Interface

ODL OpenDaylight

DSCP Differentiated Services Code Point

TLS Transport Layer Security

SSL Secure Sockets Layer

HTTP Hypertext Transfer Protocol

UDP User Datagram Protocol

TCAM Tenary Content-Addressable Memory

SSH Secure Shell

NTT Nippon Telegraph and Telephone

VLAN Virtual Local Area Network

MSTP Multiple Spanning Tree Protocol

LACP Link Aggregation Control Protocol

STP Spanning Tree Protocol

OFCP OpenFlow Configuration Point

NE Network Element

PKI Public Key Infrastructure

OVS OpenVSwitch

JVM Java Virtual Machine

NSX VMware Network Virtualisation

LLDP Link Layer Discovery Protocol

SD-WAN Software Defined Wide Area Network

OXM OpenFlow Extensible Match

HPE Hewlett Packard Enterprise

ONOS Open Network Operating System

SDN Software Defined Networking

WAN Wide Area Network

Chapter 1

Introduction

1.1 Background and Motivation

In this section, a brief overview of traditional network devices will be given. It has become increasingly obvious that the traditional paradigm of networking, based on proprietary networking devices, is not meeting the demands of modern networking. Traditional proprietary network routing devices house multiple planes which define the boundaries of architecture. The control plane provides logical intelligence controlling forwarding behaviour through routing protocols. The data or forwarding plane forwards traffic based on logic of the control plane. Configuration and monitoring take place via the management plane. The operating system software implements complex protocols that manage interaction between the planes. Interpretation of control software varies between vendors and configuration syntax can vary between models from the same vendor. Software packages contain multiple technologies often beyond the consumer's requirements and technical staff's sophistication to implement them. Furthermore, the cost of constantly upgrading and patching proprietary network operating system software and replacing hardware inflates the company's ICT budget.

In a relatively short history of the Internet and World Wide Web, there are several developments capable of triggering the extent of digital disruption that the industry is currently witnessing.

Improvement of Wide Area Network (WAN) connection protocol increased efficiency between Internet Service Providers (ISP). Corporate networks benefited from the availability of leased lines and hence, remote offices and branches had access to HQ.

Interior Gateway Protocols (IGP) steadily evolved and improved to meet the requirements necessary to provide a reliable network infrastructure capable of handling increased personal

and business data. LAN routing improved ensuring the corporate network kept pace for this increase.

Virtualisation of network elements provide data centre network engineers opportunities to overlay routing, switching and provisioning solutions in multi-tenanted installations. Operating system software platforms for server and desktop computer leverage cloud services network features are made possible through data centre storage.

Data has migrated to the cloud front with a decline in on premise hardware installations. Innovation occurred, primarily governed by afore mentioned gradual changes to network speed, delivery and sheer data increase.

This is innovation by necessity and less as avant-garde. From around 2014, change appears to have driven process and innovation for corporate and carrier networks.

Despite 20 years of research and experimentation from academics and through vendor labs, nothing had really changed until this point. From 2014 to the present, change has been rapid and occurring in a quantum leap for areas such as cloud services.

Advancements in network virtualisation created opportunities for isolated experimentation and research which could be replicated to live networks. The rate of change followed successful implementation of distributed forwarding devices.

Addressing security with this change continues to fall behind. Evolution of security measures to address emerging attack vectors within communication methods used in the distributed model are not progressing at the same pace as adoption for the technology.

Protocols utilised in achieving control and data plane distribution fail simple testing and attack simulation. Communication methods between devices used in lab experiments have migrated into commercial installations and continue to be the common practice for SDN controller development.

Using insecure connection methods increase the attack surface for the distributed model. The scale of vulnerability for attacks in some cases is directly related to the role of controller and forwarding devices in the software defined topology. Dynamic roles such as load-balancing and fail-over exacerbate connection issues for authentication and creation of secure links.

Adopting and using programmable networks introduces agility and flexibility for the network engineer. Moreover, the attraction for less expensive devices and non-proprietary system operating software is a major factor in the adoption.

Nonetheless, security issues remain embedded regardless of sophisticated solutions adopted by industry to utilise SDN in LAN and WAN topologies.

This thesis looks at the ability to exploit weaknesses in the standard communication method used for channel operations in the majority of controller instantiations.

1.2 Problem statement

In this section arguments will be presented for and against the adoption of SDN in the corporate network, campus network and data centre.

Uptake and adoption of SDN is mostly prominent in the data centre. Multi-tenanted networks can reap a big benefit leveraging what an OpenFlow enabled network has to offer in flexibility and rapid change across the infrastructure. A flow-based paradigm is well suited to protect traffic on each network slice for each virtual tenant.

OpenFlow switches can determine the logical network for every flow and tunnel the traffic to the end of the logical flow. When compared with a traditional method of associating a VLAN to a physical interface. Furthermore, the typical network architecture of Core, Distribution and Access layers binds the organisation to equipment procurement cycles of approximately three years. Innovation in this model is difficult to achieve due to inflexibility in the layered model.

In a technology dependent environment like university campus, corporate design companies and engineering firms with high-end processing and compute requirements, change is often required spontaneously to meet the needs of staff or contractors.

Introduction of new services means passing a company policy and acceptance tests. This could mean weeks of testing just to ensure the software can be deployed in the next image build.

The current network architecture and looking forward for the next two decades is heavily reliant and will evolve into further virtualisation of network functions and the over bearing architecture will be virtualised.

The virtualisation model is the only possible method of sustaining demands of the digital society we have become. Commerce in many developed countries is experiencing a cyclic two/three-year disruption throughout large companies. The digitalisation of the world is driving this disruption. Ideas and innovation occur more rapidly than ever before. Music, movies, medical records, urban data Internet of Things (IoT) available online are a small example of how businesses are disrupted.

Digital disruption in the network means changes and advancements on how the disruption is managed from an infrastructure perspective to support the pace at which data is stored, classified, shared, secured, and parsed for scientific analysis.

Information and Communications Security is a business requirement for continuity. The introduction of technology based on fundamentally insecure process simply exacerbate the problem. For digitalisation and disruption to proceed at the projected pace, communications infrastructure requires solid foundations and scalable, secure solutions.

Numerous threat analysis have been completed on OpenFlow controllers in the last decade [27] [13] [32] [29] [16]. There is no change in the protocol beyond extra features released in updates which are based on an attempt to stay on course with the rapid advancement in virtualisation of networks and the abstraction and blurring of network device roles and functions at the data centre and ISP level. The requirement for a security policy that checks, and forces TLS between controller and switch has not changed in the time this research had been undertaken.

1.3 Aims and Objectives

Designing and managing network innovation becomes difficult in a closed proprietary model. To implement change through the current infrastructure model, skilled personnel log into each device and make the changes locally. Potentially, this method of administration is exposed to the risk of erroneous configuration commands that can be difficult to troubleshoot. Applying change in a corporate network containing routers, switches, firewalls and load balancers takes careful project planning and management to avoid downtime and loss of production. The current state of provider and corporate networks comprise three decades of development through which very little has changed operationally. The task of maintaining a modern network continues to be a model of vendor lock-in with devices reaching end of life or being too old to run the latest version of system software. Meanwhile, the demand for flexible mobile services has grown exponentially over the last decade. Flexibility is lost in the static service production procedures. A representation of multiple plane architecture housed in one device is shown in Fig 1.1.

Internet popularity and adoption grew rapidly in the mid-1990s. The modest demands of file transfer and email for academics were outpaced by the interest from the public and commerce.

By the late 1990s it became obvious the current infrastructure couldn't cope with the adoption and interest the internet had generated. Researchers developed and tested new protocols with the idea of improving network services and delivery speeds. However, standardising and passing new ideas through validation via the Internet Engineering Task Force (IETF) was a slow frustrating process. Nevertheless, the IETF has continued to be a

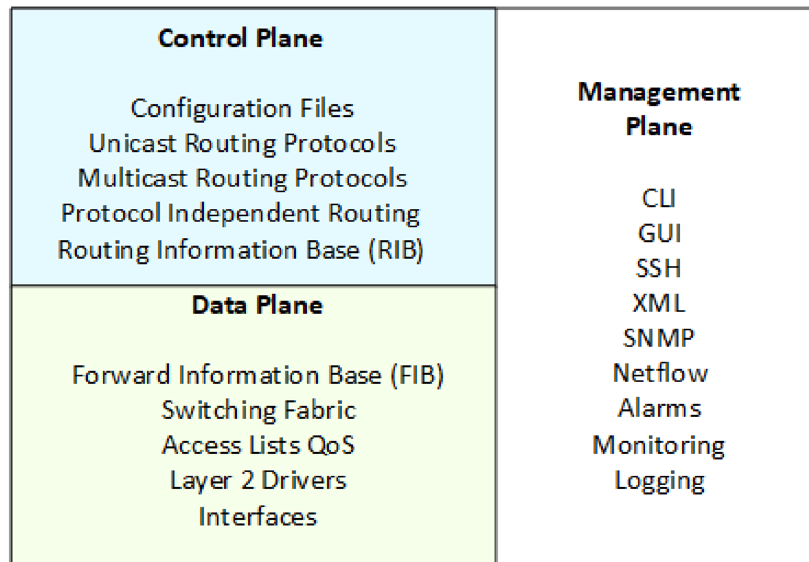


Fig. 1.1 Control and data plane in one device

voice of reason and order in an environment where a rich source of ideas, developments and experiments grow constantly from academic endeavour. Without guidance, the Information and Communications Technologies industry would struggle to reach today's level of technical sophistication.

1.4 Structure of the thesis

- **Chapter 1** presents a background to traditional networking. A background is also given for Software Defined Networking (SDN) with a comparison to traditional methods. This chapter also offers an argument on one hand for the benefits of implementing SDN networking and on the other hand for valid security implications of doing so.
- **Chapter 2** covers a history of SDN with a section discussing threat vectors and security of the SDN technology.
- **Chapter 3** Looks at the OpenFlow protocol suite and the associated virtualisation methods employed in testing and implementing SDN.
- **Chapter 4** presents data and results from throughput, latency and attack experiments.
- **Chapter 5** concludes the thesis and looks into future work and possible research opportunities.

Chapter 2

SDN Literature - History, Threats and Security

2.1 History

This section will give a brief history of attempts to introduce programmability into network operation. Significant changes to industry standards, operation guidelines and validation of protocols in the last ten years make it easier to introduce innovation to the network. Some argue that this is a step backwards. However, innovation and flexibility of network infrastructure are now more achievable.

In this new era of communications networks, leadership is changing. The Institute of Electrical and Electronics Engineers (IEEE) and Internet Engineering Task Force (IETF) had the responsibility of fostering good practice, relationships and promoting internet standards that make up the Internet Protocol Suite (TCP/IP). Whilst some view the IETF process as cumbersome and time consuming, the original design philosophy of the Defence Advanced Research Projects Agency (DARPA) team of engineers was to have the internet support multiple types of communications services. In addition, the resources used in the internet architecture must be accountable.

The IETF is an open standards organisation with no formal membership. Hence, membership comprises an international community of network designers, operators, vendors, and researchers concerned with the evolution of the internet architecture.

The IEEE is the biggest technical professional society in the world and is a rich offering of publications, standards, and certifications.

The components of the internet are networks, which are interconnected to provide some larger service. To this end, the need for a steering and validation organisation is important

to ensure development of interconnecting services and programs following strict protocol guidelines for future stability.

The innovation we see today is not a product of overnight success in communications technology. Clearly, it is an accumulation of contributions over a period of 20 years. The importance of the innovation from the early period has been punctuated by milestones culminating in what we are witnessing in research innovation currently.

However, it could be argued that lengthy processes and stringent procedures in place during the first two significant decades of communications protocol development slowed innovation and disrupted progress. Examples of this disruption are evident in the original design of Wide Area Network (WAN) transport protocols unable to cope with demand within a decade of implementation.

This is not necessarily poor design, rather a lack of foresight to the exponential growth and popularity of the interconnected networks. As a result, in some cases, the fix has often been a scrambled attempt to patch the technology as opposed to a long-term overhaul of a problematic area of networking.

Multi-Protocol Label Switching (MPLS) is an excellent example of the need to find a better method of routing due to exponential growth of internet applications. MPLS is a technique sitting somewhere between Internet Protocol-over-Asynchronous Transfer Mode (IP-over-ATM) and multilayer switching. MPLS bridged these original technologies as a technique to become one of the most successful developments in the last 15 years. However, the need for MPLS highlights rapid change in the use and operation demands of the internet.

The IP-over-ATM standard was first released in 1994 under RFC 1577 [19]. The IETF established the MPLS working group in early 1997. Whilst initiatives into label switching techniques from Toshiba, Cisco and IBM appeared between 1994 to 1996, the first Request for Comment 3469 [30] for MPLS was released in 2000. The longevity and resilience of the MPLS technique is an example of IETF contribution to the industry.

Today, most people consider the internet, as one of the basic requirements for existence and others believe that the internet should be a human right. From a technical perspective, the internet continues to be a collection of devices, often poorly configured, passing billions of packets per day between boundaries of countries with differing laws and security polices pertaining to the ownership of data.

Examples in separating architecture design boundaries date back to the mid-1990s. Network administrators and engineers resisted the unpopular notion [9] of isolating the control and forwarding elements in routers. Somewhat paradoxically, in-box advancements assisted researchers in enabling distributed control outside traditional networking devices. Fig 2.1 represents significant events in the history of network programmability research and progress.

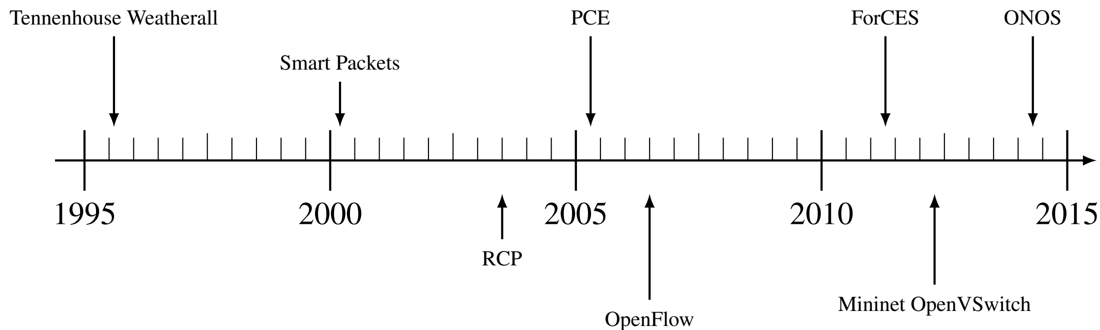


Fig. 2.1 Innovation timeline for programmability in Networking

Routing protocols including Open Shortest Path First (OSPF) using area-based routing and Border Gateway Protocol (BGP) introducing route reflectors, isolate and share the planes along with logical knowledge of the network to increase efficiency.

Meanwhile, Simple Network Management Protocol (SNMP) and Network Configuration Protocol (NETCONF) [22] provided methods of interrogating network Management Information Base (MIB) to perform network management. SNMP V1 was first proposed in the late 1980s.

Individual vendor's improvements per device benefit the throughput and line speed data flows with the introduction of in-box virtualisation in addition to research and development on Application Specific Integrated Circuits (ASICS).

Active networks identify an era from the mid to late 1990s through which a (DARPA) sponsored research program sought to introduce programmable functions into the network with application programming interfaces (APIs). Smart Packets [26][31] investigated the possibility of injecting code into network nodes with a vision for a network architecture,

Activenet design for programmable nodes. A call to the greater research community failed to create any momentum from the ideas.

Netscript [4] describes a network layer utility using passive data streams to communicate between network elements, namely routers. This effort pursued development of a method for extending existing protocols. Beyond the Active Networks initiative, significant historical events in network design and research endeavours can be viewed in three separate phases.

2.1.1 Control Data Separation

Routing Control Platform (RCP) addresses the issues in fully distributed path-selection computation that (BGP) routers must perform [2] [8]. The method discusses and introduces a helping method for the routers in selecting best paths to forward packets on behalf of the router.

A study into the effects of openness and flexibility of SDN networking [22] argues that separation of forwarding and control elements has many researchers concentrating on designing and implementing an SDN enabled network. However, issues compounded by separation of the architecture exist and are analysed in this study including downgraded efficiency relative to controller channel connections to forwarding switches. As speed and congestion increases table lookup and matching degrades throughput in the infrastructure.

While ForCES [5] is a separate research design project to SDN, both methods leverage the architectural boundaries of control and forwarding within the design of resources. ForCES forwarding nodes (FF-N) use Logical Function Blocks (LFB) to interact with the forwarding nodes and ForCES protocol. SDN forwarding protocols such as OpenFlow employ flow tables for increased flexibility in building match criteria for flows.

Path Computation Element (PCE) addresses the compute of network paths [7]. An element in this case is defined as a network node capable of computing a network path or route based on information stored on a Traffic Engineering Database (TED). The benefit of this proposal to calculate Label Switched Path (LSP) values for the management of Multi-Protocol Label Switching (MPLS) paths and health of the link through the MPLS domain.

2.1.2 OpenFlow and Network Operating Systems

The number of SDN controllers has grown since the release of the first network operating system NOX [12]. Academic endeavours from Stanford and Berkeley universities with input from Nicira Networks successfully created a programmatic interface to the network. The

motivation for the work came partially from frustration in the lack of freedom on academic networks within the universities. Released as an experimental protocol by Stanford university in 2008, OpenFlow [21] was initially installed across two buildings at the campus to promote the adoption by Stanford and other universities.

Openflow is based on Ethernet switches and runs an internal flow table that can communicate with standard interfaces. Version 1.1 of the protocol was released in 2001 and was defined by the Open Networking Foundation (ONF) as the first standard communications interface defined between the control and forwarding layers of SDN architecture.

2.1.3 Network Virtualisation

PlanetLab [3] is presented as an overlay testbed. The ambition was for over 1,000 nodes as a globally-distributed collection of diverse connections. To achieve the diversity and geographical distribution, applicants to the program offer a piece of on-premise hardware exposed to the internet. A Linux operating system is maintained on the hardware by the PlanetLab technicians, creating an alternative Internet of sorts. Contributing institutions could request a slice of the distributed network to experiment, monitor, analyse and observe behaviour of distributed storage, network mapping or new technologies.

Fig. 2.2 GENI

Fig. 2.3 PlanetLab

Fig. 2.4 FIRE

PlanetLab currently consists of 1,353 nodes at 717 sites worldwide. This research became closely based around a National Science Foundation (NSF) project Global Environment for Network Innovations (GENI) [6] and provided a platform for researchers to experiment and develop trials for new Internet architectures.

A group of researchers met with government officials in Zurich in 2007. By 2008, Future Internet Research and Experimentation (FIRE) [11] was formed with a budget of 40 million Euro. A recent strategy listed by FIRE is an idea for a Digital Single Market to enhance the European Commission's (EU) position as a world leader in the digital economy. A common purpose for the project platforms was the vision of a purpose built globally connected slice of

test internet. These clean-slate architectures broadened the vision of control and data plane separation.

Researchers at Stanford University were responsible for developing Mininet [18]. The Mininet prototyping environment is complimentary to the 'Clean Slate' initiatives of PlantLab and GENI. Using standard Linux operating systems, Mininet hosts, links, switches and controllers connect via a virtual Ethernet pair to simulate a wired connection between two virtual interfaces. The idea of a large network running on a laptop within a virtual machine was a revolution in testing and attacking the network and advanced research on SDN by providing a safe environment to experiment on early distributed control and forwarding elements. Topologies can be built quickly with controller, switches and hosts configurable in either command line (CLI) or graphical user interface (GUI) mode. The inclusion of OpenVswitch [24] to the repository introduced real world functionality with support for distribution across physical servers once testing was validated in a virtual environment.

Packet switching techniques have improved vastly, utilising Forwarding Information Base (FIB) and Routing Information Base (RIB) feature development. Moreover, in-box isolation of control and data planes are standard virtualisation feature in many vendor platforms. Path Computation Execution (PCE) is aided by introducing micro processing separate from standard routing table calculations and interface caching algorithms. Furthermore, PCE has become a fundamental building block for label switching techniques including (MPLS).

Virtualisation techniques helped blur the lines between the traditional layered approach to networking with each layer working in isolation relying on very little interaction between the layers above and below. However, the control plane still bore a heavy load of Central Processing Unit (CPU) compute, providing the data plane with information necessary to maintain Non-Stop Forwarding and High Availability (HA) to device interfaces in the process of creating and updating RIB, FIB and adjacency tables.

Notwithstanding the advancements in single device hardware and software capabilities, the ability to react to network behaviour and demand of services is hindered by proprietary systems. Consumers expect a convergence model that equates to Any Time, Any Where, Any Device (ATAWAD) [1]. As the demand grows, so do challenges for network administrators and Internet Service Providers (ISPs). The expectation to have all devices able to connect to the internet and seamlessly access the user's assets tests platform flexibility due to the lack of a defined network perimeter. Users and devices are moving constantly connecting often with a mixture of hardware platforms for the same model of device. Short intervals of streaming video or application requirements online requires agility of the network and flexibility in administration of services to deliver.

2.2 Cloud computing

The cloud computing paradigm enables ubiquitous access to shared pools utilized for Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS). The multi-layers of Network virtualisation developments have served data centre requirements linking and meshing network protocols and functions. Network overlay fabrics leverage the virtualisation concept to form the cloud offerings including proprietary and open source products.

SaaS reduces the cost of software ownership removing the technical requirement to install, manage and license the software.

At a lower level of the cloud services, PaaS abstracts the environment dealing with operating system and server system software along with the underlying hardware and network.

Further down the stack, the fundamental building blocks of cloud services is defined by IaaS. Brokerage services for IaaS are common with the competition to offer complete virtual data centres growing among providers. Adopting an IaaS service gives the client a high level of control over their infrastructure without the investment and inflexibility of on-premise servers.

Traditional network protocols including IP and Border Gateway Protocol (BGP) were designed for the internet of the 1980s without the foresight of what the information technology would become in less than two decades.

Whilst IP and BGP were both engineered to be reliable and robust, the original (IETF) development held no vision of the changes driven by shear volume of data that we now generate as a society. To overcome rigid design of internet protocols, Data Centre network engineers leveraged the flexibility of overlay networking solutions. In the infrastructure design of network overlay, software is used to abstract and virtualise other connections. In a physical network, a link will exist between two nodes. The network overlay utilises the physical network to provide a path for routing or switching in which tagging and encapsulation methods allow traffic to move through the software defined overlay without knowledge of the physical network. Multi-tenant provider networks utilise this environment for rapid provisioning and deployment of PaaS and IaaS. High speed network equipment onsite at data centres allow ISPs and cloud services providers a link to Metro Ethernet. One of the main benefits in performance for users with access to the high-speed metro network is by-passing the internet. Cloud vendors focus their business to move the cloud closer to densely populated areas of users to mitigate the issues in internet connections.

2.3 Data centres

The overwhelming adoption of SDN is occurring in the data centre. Within the last decade, the complexity of interconnected networks has grown with the introduction of data centre storage, cloud-based applications and multiple devices for which data should be always available, across all types of services.

Due to virtualised infrastructure, many major data centres can utilise space housing multiple servers and services on one piece of hardware. Much of the complexity exists in the networking infrastructure for the data centre. An ISP must now provide a means of access to all storage facilities. Adoption of SDN and NFV provides an overlay fabric applied to the traditional network physical infrastructure.

To achieve the concept of ATAWAD and produce seamless network connectivity to so-called cloud applications, the use of network overlays allows mapping of physical nodes to virtual connections, circuits, routing and forwarding. Network programmability is allowing data centre architects and engineers to mitigate the drawback of traditional layer 2 protocols like Spanning Tree Protocol (STP). The greatest bane to data center engineers was, and to a large extent still is (STP, 802.1d) [20]. Created to prevent loops in densely connected Layer 2 topologies, STP wastes resources by blocking 50 percent or more of the links in a redundantly-connected network, it has terrible failover times, and is prone to misconfiguration. Over the years there have been many enhancements to STP to reduce its adverse effects in networks:

Spanning Tree Solutions

- Rapid Spanning Tree Protocol (RSTP, 802.1w) reduces failover from 30 to 50 seconds to 6 seconds.[20]
- Multiple Spanning Tree Protocol (MSTP, 802.1s) improves utilisation of VLAN groups to mitigate blocking.
- Link Aggregation Control Protocol (LACP, 802.1ax) bundles multiple parallel links so STP treats them as a single link.
- Shortest Path Bridging (SPB, 802.1aq) and Transparent Interconnection of Lots of Links (TRILL) to utilise shortest path first algorithms normally seen at Layer 3 to prevent loops and shorten failover times.

Are these solutions considered patches on a problematic protocol? In the case of SPB and TRILL, replacing STP altogether for a different loop prevention scheme. If there were no

trees in the topology, there would be no need for Spanning Tree. The development by various vendors of switching fabrics in recent years has a goal of replacing STP with virtualisation. The concept is to create a flat virtualised data plane that gives the appearance of a single switch and is a compelling case for the deployment of SDN base networks.

2.4 Connectivity through providers

With benefits and cost savings achieved within the data centre, tier 1 and 2 ISPs now seek a Software Defined Wide area Network (SD-WAN) solution.

Areas of the Internet may experience some re-synchronising of paths and link outages. However, it is rare that a region is unavailable and does not allow transmission traffic between two nodes. Redundancy will provide a path between the transit ISPs within the region. Major outages are further down the chain of hierarchy falling to the responsibility of a tier 3 ISP or local service that has become unavailable due to failure or human error.

For a tier 3 ISP (last mile provider), the very nature of business is to procure new customers, realise their requirements, provision the network and bring them online with their purchased transit agreement. The tier 3 customer is generally not involved in agreements that the tier 3 provider has with upstream transit. Hence, the majority of change occurring at tier 3 has potential to disrupt business and affect customers financially. Successful ISPs at tier 3 have adopted automation and best practices for testing and recording change management.

Service Level Agreements (SLAs) create a mechanism for ensuring transit purchased is available as close to the five nines as possible (99.999 percent uptime). The ISP will provide a quality and type of service reflecting agreements in the SLA.

As the demand for seamless, fast access to private and third-party services grows within business and private sector, routers and switches have evolved to serve the demand. Traditional methods of packet matching at Layer 3 proved a bottleneck to speed increases in data transmission on the LAN and out to the WAN. Vendors addressed this problem individually however the results were very similar. From a discussion based on vendor specific, security hardened network devices installed on a corporate LAN, it is difficult to extract information from the devices without physical access to the router or switch.

The security discussion around PCEP and private routing domains changes somewhat, the corporate data is passing through a tier 3 provider on its way to a remote interconnected network via the internet. It is not feasible to have a route capable device for every customer purchasing transit from an ISP. Network virtualisation provides segregation at layer 2 and layer 3 within the provider core meaning multiple customers are able to share a single network

element. With the utilisation of MPLS and Virtual Routing Functions (VRF), this is a sound method of segregation and security for private routing domains. SDN has played a part at this level already. In the last five years projects have provided the successful segregation of routing domains and protocols at the network edge allowing rapid recovery from traditionally cumbersome protocols.

Nippon Telegraph and Telephone

Nippon Telegraph and Telephone (NTT) integrated a solution recognised as a BGP route controller into the edge devices of their own corporate network to prevent the need for BGP within these devices. In this case, the routing information is passed to the OpenFlow controller which builds a flow table based on the BGP controller's knowledge of the customers network. Once the OpenFlow controller establishes a VLAN for the corresponding IP address, the information is pushed to the OpenFlow agent and onto the edge forwarding devices.

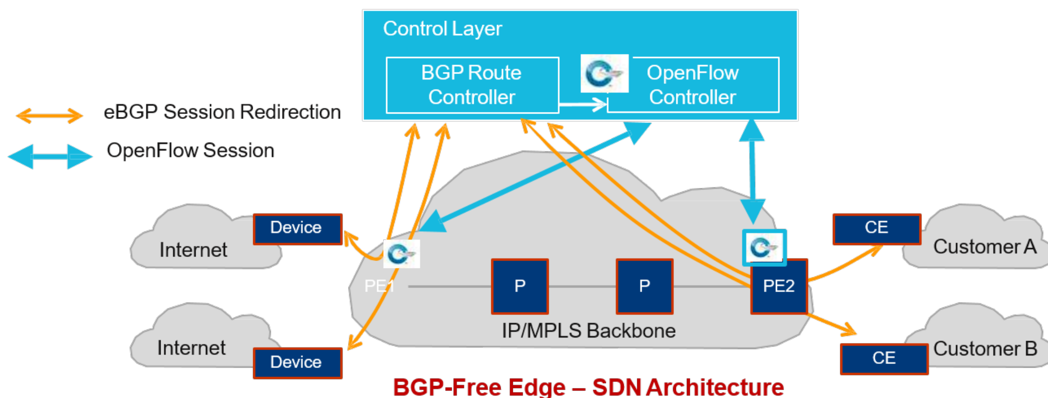


Fig. 2.5 NTT SD WAN Architecture (ONF TR -506 2014)

The concept shown in figure 2.5 is an example of the necessity to isolate private routing domains and information. The peering required between client and provider is a common model of internet connectivity through a large transit network.

Delays in re-convergence experienced by BGP after a link failure remains a well known problem with the protocol. Traditionally, internet routes are summarised into a customer routing table to speed up table lookup. To minimise lookup time from the customer perspective, in the event of a path failure, routes comprising internet paths are copied down to the client devices and inversely, some of the private routing information is configured with the ISP. Additionally, common practice is to configure VRF on the Provider Edge (PE) router to isolate traffic at layer 3.

The secure channels between OpenFlow agents in the BGP route controller and PE routers must perform encrypted transmission across transport and auxiliary links. A failure to ensure this is maintained, would expose customers to each other within their respective routing domains at the MPLS edge.

Google B4 Traffic Engineering

Another successful example of the legacy to hybrid migration can be viewed in Google's optimisation of application data between data centres with the B4 project [15].

The motivation was to enable OpenFlow on internet facing user traffic and optimise that traffic between two Google global data centres. The link between the targeted data centres are capable of carrying ten percent of Internet traffic.

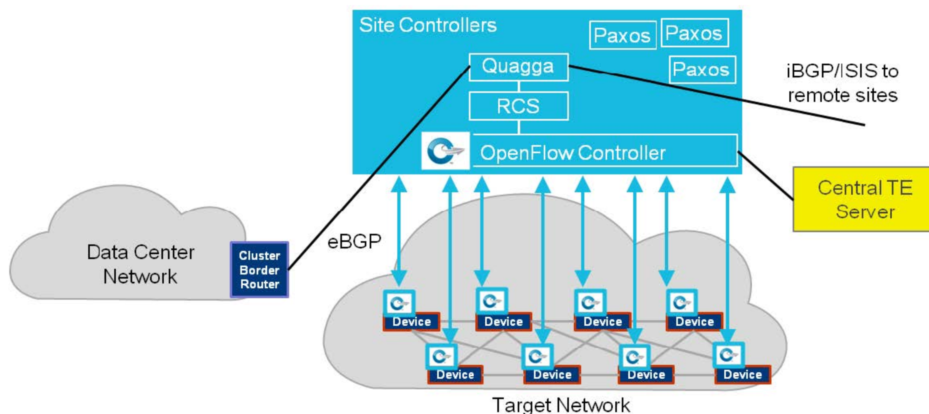


Fig. 2.6 Google B4 network traffic optimisation (ONF TR - 506 2014)

Without Traffic Engineering (TE), the links between the centres carry a variety of user based services traffic including Google+, Gmail, YouTube, Google Maps and others.

A pre-migration assessment of the links exposed inefficient prioritising of application data. The project was implemented in three stages to mitigate disruption due to the monolithic size of the undertaking. Architecture design for the B4 project is shown in fig 2.6.

The result produced a WAN fabric enhanced by the optimisation of links at 95 percent constant service achieved by segregating like traffic types per link.

Whilst the B4 network is an excellent example of improving scalability, flexibility and enhanced management in a WAN based, OpenFlow enabled environment, it is impossible to realise the impact on security or failure within Google's operational domain. Ownership across all tiers of infrastructure is a distinct advantage for this engineering project.

2.5 Software Defined Networking

Software Defined Networking (SDN) is an emerging paradigm utilising the intelligent control plane by physically separating it from the data plane and distributing control from a single controller to multiple forwarding components. To declare SDN an emerging paradigm doesn't disregard the history of attempts to introduce programmability in networking. Conversely, no other attempt has disrupted the industry or driven innovation across all sectors.

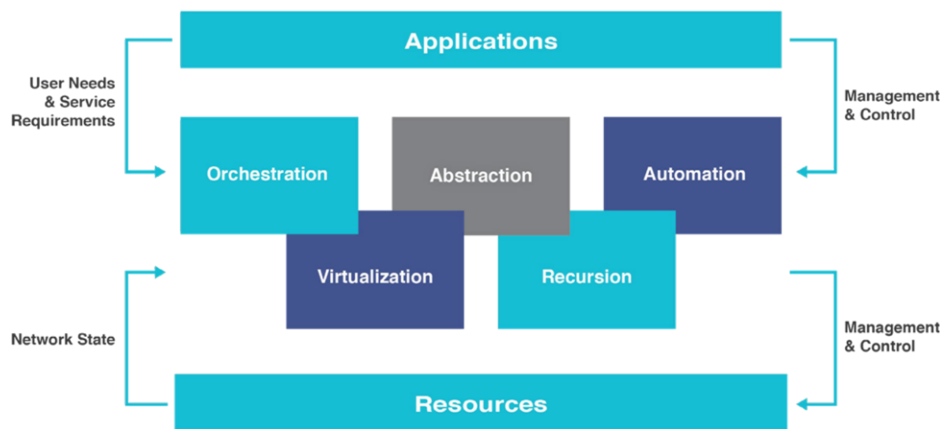


Fig. 2.7 An overview of SDN operation flow and control (ONF - 521 2016)

SDN in its current instantiation has gained traction in the industry. Hardware vendors supporting the OpenFlow protocol grew from 2015. Adoption of the OpenFlow APIs allowing programmability within the network gained popularity from 2007 to 2010. The notion of allowing applications direct access into private, isolated routing and storage domains, is one some consider as the most disruptive and advanced development networking has witnessed. The de-coupling of control and data plane for network elements solves many problems for network designers and engineers cost primarily, as the controller holds all the intelligence for the network, the data plane/switch devices require only simple bare-metal, from any vendor and do not require expensive feature-rich operating systems. Network changes take place in one location the controller. The other devices simply pass, and forward packets based on the flow table. Network monitoring with SDN allows the network to respond to its own changes and congestion. SDN gives us the ability to turn our entire network or data centre into a cloud and slice flow space via network overlay technologies and virtualisation.

Fig 2.8 shows a centralised controller with distributed data forwarding.

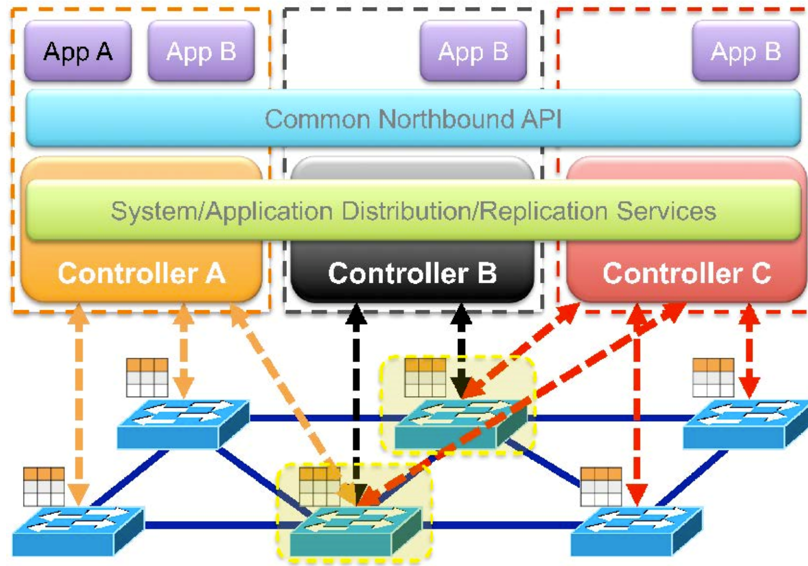


Fig. 2.8 SDN De-coupled view of control and data planes (ONF - 521 2016)

Architecture

An SDN network is defined as a programmatic abstracted interaction with a network. The concept of an application plane is introduced allowing southbound applications to influence the management of network elements. SDN applies the flexibility of software to the entirety of the networking space. A logical view of the SDN controller is shown in fig 2.9

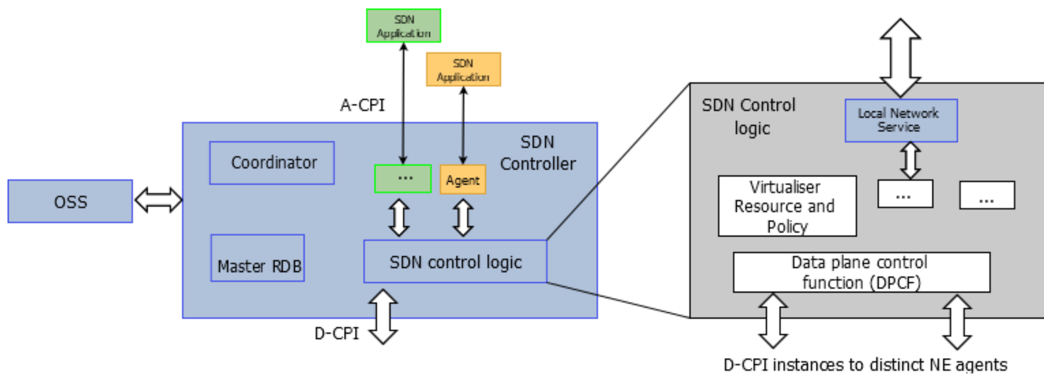


Fig. 2.9 SDN Controller Logic (Edge.P 2018)

2.6 Threat modelling

The adoption of SDN controllers based on OpenFlow has expanded the attack surface for traditional attack vectors. In some cases, it introduces new and more potent versions

of existing attack methods. Examples of these attacks are TCP SYN-flood and Denial of Service vulnerabilities. Fig 2.10 shows a simplified but typical programmatically capable network installation with the following assets. Control Plane, Data Plane and Application Plane.

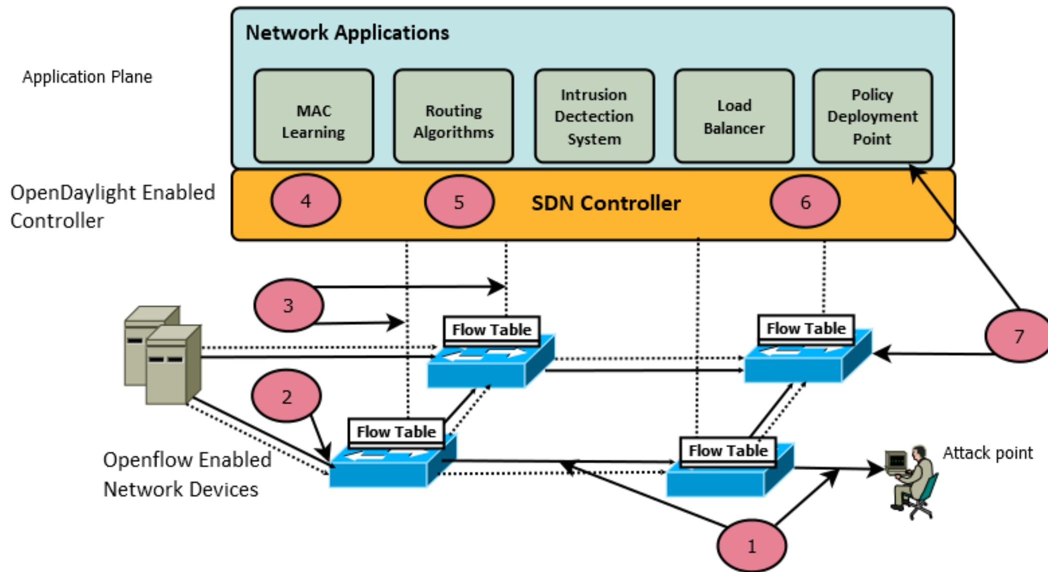


Fig. 2.10 Traditional and SDN threat vectors (Edge.P 2018)

The diagram also shows several attack vectors that have been consistently identified within traditional networks and the SDN infrastructure. There are three distinct attack vectors that have been highlighted as being unique to SDN.

Vectors 3, 4 and 5 are produced as a direct result of separation or decoupling of the control and data/forwarding planes.

This thesis and most of the asociated analysis will be concentrated predominately on vector 1 and 3. Vector 1 is an attack vector present in both distributed and closed network topologies.

In identifying the specific threats owned by the threat vectors specified in the above Fig 2.10, it is important to quantify the model of SDN network to which this reference is applied. Firstly, it can be assumed that the typical OpenFlow switch and controllers mentioned in the previous chapter are the basis for all threat analysis, testing and attacking recorded in the research.

A typical OpenFlow controller and switch configuration is the starting point for many complex scenarios that could potentially include multiple controllers and switches, load

balancing connections between multiple switches and auxiliary connections from the same controllers.

Potential architecture possibilities aside, it is critical to identify threats and compose counter-measures for the OpenFlow system that is the basis of a programmable network. Therefore, in identifying threats, the threat will be applied to the simple controller/switch initially with a small amount of time being dedicated to an expanded threat scenario.

2.7 Threats

Threats are based on data flows and interaction of SDN components and interaction with external actors. This is done in accordance with the Spoofing, Tampering, Repudiation, Information disclosure, Denial of Service, Elevation of Privileges (STRIDE) methodology according to the Microsoft Corporation [25]. Table 2.1 shows the relationship of attacks and the specific security property affected.

2.7.1 Table of threats

Table 2.1 Table of Threats

Attack Type	**Security Property**
Spoofing	Athentication
Tampering	Integrity
Repudiation	Non-Repudiation
Information Disclosure	Confidentiality
Denial of Service (DOS)	Availability
Elevation	Authorisation

2.7.2 Spoofing

The impersonation of the OpenFlow Configuration Point (OFCP) is a potential attack from the control plane and is an attempt to disrupt the network elements (NE) attached to the controller. An attack on the configuration point where OpenFlow messages are being processed by the controller could mean an adversary has control over policy injection, port state, flow tables and routing domain information that would be normally private to networking elements. If the infrastructure is compromised at this level, very little of the

network elements would remain secure with most private information available for alteration, information harvesting or theft.

2.7.3 Tampering

Because of the ability to impersonate the OFCP, all messages sent between the NE and the controller have the potential for tampering of some degree. At this point nothing is secure on the switch/controller channel. As well as intercepting and altering genuine messages between the devices, false policy and flow information can be passed to the switch and an attempt made to re-route information and network traffic away from the original path or to a device that is under resourced to handle the extra traffic.

2.7.4 Repudiation

A repudiation threat involves carrying out a transaction in such a way that there is no proof after the fact of the principals involved in the transaction. This is a threat that might lead to financial losses, legal implications, and lawsuits if not solved by appropriate and legitimate proof. Hence, repudiation has to be dealt with utmost care in all online/ offline transactions.

In the context of software defined networks, non-repudiation may involve the ability for a system to counter the repudiation threats. Strong encryption, logging features and the use of digital signatures will prevent information disclosure (covered in the next section).

2.7.5 Information disclosure

The resultant ability to gain access to OpenFlow communication channel due to non-mandatory security policy and lack of encrypted channels between controllers and network elements allows a threat actor to begin the planning and harvesting of information that will allow a future DOS attack on the installation. By collecting statistical data on flow tables and their capacity to hold a specific amount of information the actor has the knowledge required to overwhelm ports and flow databases. This is done by simply re-directing flows towards the ports with weak forwarding ability.

2.7.6 Denial of service

Once flow table entries can be modified, the next stage of the attack is to manipulate flows from multiple ingress ports to one of few egress ports and overwhelm the egress ports.

Fake switches can also be utilised to increase the burden on the processing and forwarding of the network elements. Interrogation of port capability and flow table capacity statistics from previous stages can be used to design traffic flows and hinder the ability to process flows.

A recommended mitigation technique at controller setup is to configure secured auxiliary connections to improve the performance of the switch – controller connection.

2.7.7 Elevation of privileges

After successfully impersonating an OFCP, the actor may have the opportunity to elevate privileges by altering or generating policy.

2.8 Severity and location of attacks

In the analysis of attacks and where they might occur in relation to SDN and traditional attack vectors, a simple single topology was assumed as the starting point for analysis. The very attraction for an SDN implementation to convert and liberate the virtualised resources in networks makes installation insecure. The model for data storage and availability is one of shared resources in the data centre and the ISP level of operation.

2.9 Security and SDN

There has, over the last three years, emerged a classic split in the research community with regard to SDN controllers, switches and the methods with which programmability is introduced into the network. One of the aims of this paper is to outline, document and comment on the inadequacies and major issues surrounding OpenFlow based SDN controllers.

2.9.1 Security for SDN

The contrasting developments in the networking industry are taking this technology, which is in some regard still experimental, and creating security solutions on a fundamentally insecure platform. These solutions include; Network Forensics, Security Policy Alteration and Security Service Insertion [27] [23] [17] [28]

The thing to keep in mind however, is that some of the documented solutions are extremely functional, elegant, and practical and will help in solving many network issues. Security with

SDN.This situation drives the need to ensure the underlying SDN infrastructure is secure, can be secured easily and can be monitored to check integrity is maintained. It simply is not good enough that security solutions are being designed on protocols with flawed security.

Chapter 3

OpenFlow Protocols and Virtual Environments

3.1 The Open Flow protocol

Software Defined Networking has roots at Stanford University where Martin Casado completed a PhD under supervisors Nick McKeown, Scott Shenker and Dan Boneh in 2008 [12]. Development of OpenFlow began around this time and the Stanford computer science department promoted the term SDN.

Control of OpenFlow was transferred to the research community placed under the auspices of the not-for-profit Open Network Foundation (ONF) in 2011. The ONF was co-founded by Nick McKewon and Scott Shenker with support from over 150 networking-equipment, semiconductor, computer, software, telecom and data-centre companies. The philosophy behind the foundation primarily is to advance innovation in the networking industry. A theme promoted by McKewon et al during the early development at Stanford University.

Fig 3.1 shows the relationship of switch and controller utilising the OpenFlow protocol for communication between the control plane, forwarding element and pipeline processing of the switch.

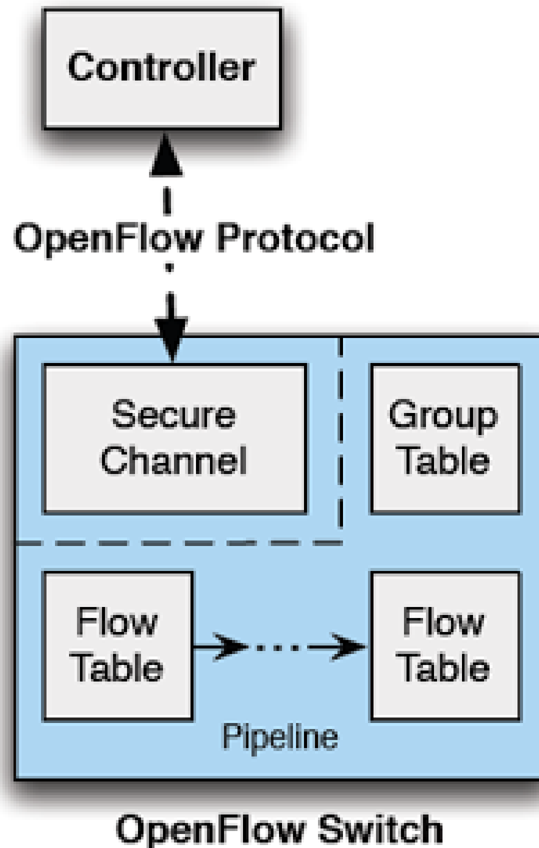


Fig. 3.1 The OpenFlow Protocol Components (ONF TS - 215 2016)

3.2 OpenFlow operation

As a control protocol, OpenFlow brings standardisation to connections between controller and switch. This is the primary function of OpenFlow which can be defined as:

- flow instructions that comprise the forwarding behaviour of the switch
- messages the switch sends to the controller to inform the controller of changes effecting forwarding.
- a packet format containing the instructions and messages.
- a protocol for sending and receiving messages between the controller and switch.

This representation is the basis of all OpenFlow enabled switches functioning as a pure OpenFlow element. A hybrid version of this configuration employs traditional Ethernet switching with OpenFlow virtualised over some of the physical ports on the device. For

testing and analysis, the experiments carried out in this thesis use a mixture of software and hardware switches.

The OpenFlow switch can be described in several components as shown in Fig 3.2. The switch agent conveys the OpenFlow protocol to one or more controllers. The switch components are discussed in the following sections.

The controller also conveys to the data plane using the internal protocol. The switch agent translates commands from the controller to send to the data plane. The data plane performs all packet forwarding and manipulation according to flow rules. However, in some cases, it will send packets to the switch agent for further handling in the absence of a matching flow rule.

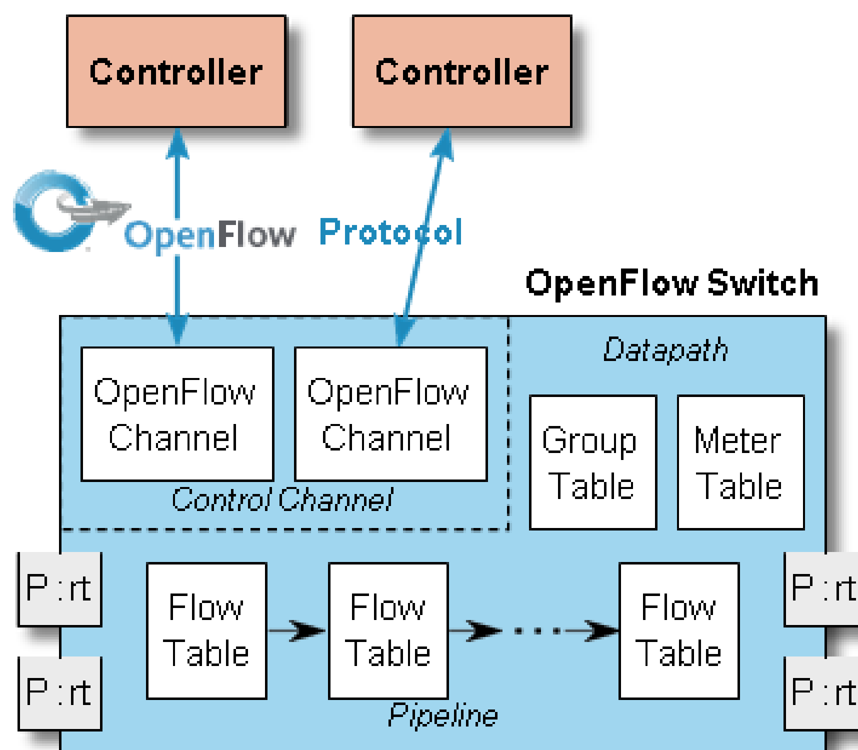


Fig. 3.2 Data flow through the OpenFlow Switch (ONF TS - 215 2016)

The data-path and pipeline manage the flow of packets between flow tables. Every packet is compared to a least a single flow table. Flow entries at the table level will consist of match fields used to compare extracted header information from ingress packet flows.

A match in the table uses the flow rule instruction set to extract actions taken on matched packets. This could include dropping the packet, punting the packet to another flow table, or pushing it to an egress port.

3.3 Switch components

An OpenFlow Logical switch consists of one or more flow tables and a group table to perform lookups and forwarding. One or more channels connect to external controllers. The switch communicates with the controllers and the controllers manages the switch using the OpenFlow switch protocol.

Fig 3.3 shows the main switch components of an OpenFlow switch. Packets are processed at the first flow table and may continue to be processed at different tables in the pipeline. Flow entries match packets in priority order, with the first matching entry in each table being used. If a matching entry is found, the instructions associated with the specific flow entry are executed.

Table 3.1 Main components of a flow entry in a flow table

Match Fields	Priority	Counters	Instructions	Timeouts	Cookie	Flags

Each flow entry contains the fields shown in table 3.1.

1. **Match Fields:** Match against packets. Includes ingress ports and packet headers. The field may also contain metadata from a previous table.
2. **Priority:** Precedence can be set for the flow entry using Differentiated Services Code Point (DSCP).
3. **Counters:** Counters are updated when packets are matched.
4. **Instructions:** Setting instructions can be executed by the controller and is the basis of policy and matching actions. Once the instructions are set, they are executed on a match.
5. **Timeouts:** This is the maximum amount of time a flow can sit idle in the switch before the flow is expired.
6. **Cookies:** A data value that operates outside the packet matching domain and can be used for flow statistics, modification or deletion of flows.
7. **Flags:** these alter the way flow entries are managed.

OpenFlow pipeline

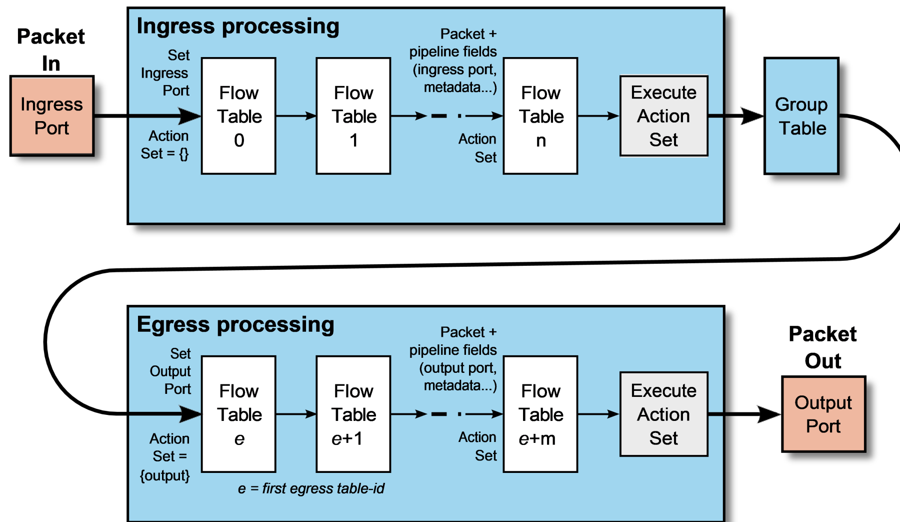


Fig. 3.3 Pipeline processing

OpenFlow-compliant switches come in two types: OpenFlow-only, and OpenFlow-hybrid. OpenFlowonly switches support only OpenFlow operation, in those switches all packets are processed by the OpenFlow pipeline, as shown in fig 3.3, and can not be processed otherwise. OpenFlow-hybrid switches support both OpenFlow operation and normal Ethernet switching operation, i.e. traditional L2 Ethernet switching, VLAN isolation, L3 routing (IPv4 routing, IPv6 routing), ACL and QoS processing. Those switches should provide a classification mechanism outside of OpenFlow that routes traffic to either the OpenFlow pipeline or the normal pipeline. For example, a switch may use the VLAN tag or input port of the packet to decide whether to process the packet using one pipeline or the other, or it may direct all packets to the OpenFlow pipeline. This classification mechanism is outside the scope of this specification. An OpenFlow-hybrid switch may also allow a packet to go from the OpenFlow pipeline to the normal pipeline through the NORMAL and FLOOD reserved ports.

OpenFlow Ports

OpenFlow ports behave in the same way they would on any other switch. Ports are designated ingress or egress depending on the traffic flow. As with normal switching, ports are dedicated and for changes to occur i.e. adding, removing, or changing ports, this must be managed carefully.

For OpenFlow switches, the controller is responsible for the changing and altering flow tables in the event of changes to ports. OpenFlow switches support the concept of OpenFlow only or a Hybrid switch which is capable of supporting physical and logical ports for ingress and egress. OpenFlow defines three types of standard ports and an OpenFlow switch must support all three.

- **Physical** ports map directly to the ports on a bare-metal switch and could be shared by a number of logical ports through virtualisation.
- **Logical** ports are non-physical and do not respond to any physical interface. These interfaces are often virtualised into loopback, null and link aggregation interfaces to handle the virtual switching functions available.
- **Reserved** ports are designated for internal processing including flooding and hand-off for flows. (Reserved ports can be either required of optional. See appendix B for a full list of required ports)

Channel connections

The logical OpenFlow switch as a de-coupled data-forwarding element connects to the OpenFlow controller via a channel used for passing network information, flow rules and policy affecting how the switch handles phases of the flow management and messages relating to how unknown flows are processed.

Channel connections operate over at least TCP and therefore do not require message reliability mechanisms from the OpenFlow protocol.

Ideally, channel setup will be via TLS. OpenFlow 1.3 and greater also ships with SSL which is not as secure as TLS. Furthermore, SSL mechanisms are prohibited by the IETF for inclusion in protocol development and maintenance meaning TLS is the current best practice secure connection option.

Secure connection start-up is not mandatory in any of the current instantiations of OpenFlow controllers deemed production ready on the market today.

On the original OpenFlow specification (and all versions since) the default method of communication is TCP or TLS implemented between the control layer device and the forwarding elements. It is possible for the switch or controller to initiate this authentication method. Intitial pairing and communication occurs on a user-specific port or the default port of 6653.

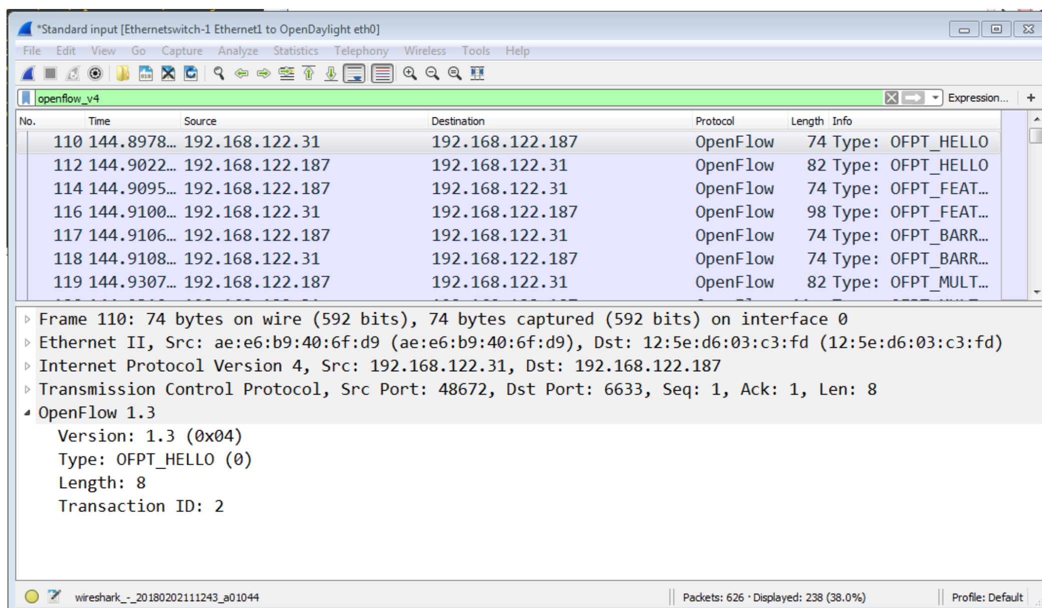


Fig. 3.4 OpenDayLight Hello

The field OFPHET_VERSIONBITMAP asks for the highest version of the OpenFlow protocol available on the controller.

```
/* 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. */
};
```

The construction of the OpenFlow header displayed in fig 3.4. This request for a pairing is the initial attempt at forming a neighborship with the device on the other end of the link. The pairing process shown in fig 1.11 shows a TCP pairing for devices.

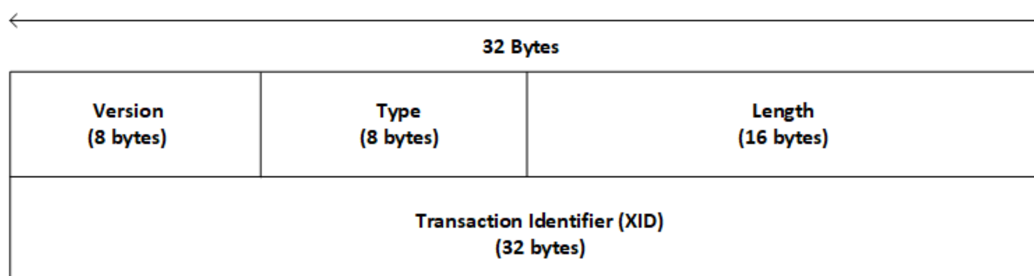


Fig. 3.5 Network byte order (Edge.P 2018)

OpenFlow controllers manage switches over one or more networks. A reliability mechanism is required to manage the connection in terms of session start up, error checking, flow control and session tear down. The channel should provide TCP/IP connectivity. TLS or TCP handle all reliable channel maintenance for the connection once it is established. Whilst the preferred method is to utilise TLS, this rarely happens.

This negotiation of versions is an important phase in the process of channel establishment because of features that may not be available for certain versions of OpenFlow. Tests have shown that under severe load, the controller and switch can disregard the OFPHET_VERSIONBITMAP field and actually accept a connection from a lower protocol version. Depending on features, it is a vulnerability if the connection pre-requisites for the older version degrade the connection standard due to outdated security protocols.

OpenFlow messages

The switch sends a HELLO message to the controller (it is also possible for the connection to be initiated from the controller.) Fig 3.3 shows the field for matching OpenFlow versions as devices begin the pairing process.

Table 3.2 OpenFlow message Types

Message	Type Number	Message Category
HELLO	0	Symmetric Message
ERROR	1	Async Message
ECHO REQUEST	2	Symmetric Message
ECHO REPLY	3	Symmetric Message
EXPERIMENTER	4	Symmetric Message
Switch Configuration Messages		
FEATURES REQUEST	6	Controller/Switch Message
FEATURES REPLY	6	Controller/Switch Message
GET CONFIG REQUEST	7	Controller/Switch Message
GET CONFIG REPLY	8	Controller/Switch Message
SET CONFIG	9	Controller/Switch Message
Asynchronous Messages		
PACKET IN	10	Async Message
FLOW REMOVED	11	Async Message
PORT STATUS	12	Async Message
Controller Command Messages		
PACKET OUT	13	Controller/Switch Message
FLOW MOD	14	Controller/Switch Message
GROUP MOD	15	Controller/Switch Message
PORT MOD	16	Controller/Switch Message
TABLE MOD	17	Controller/Switch Message

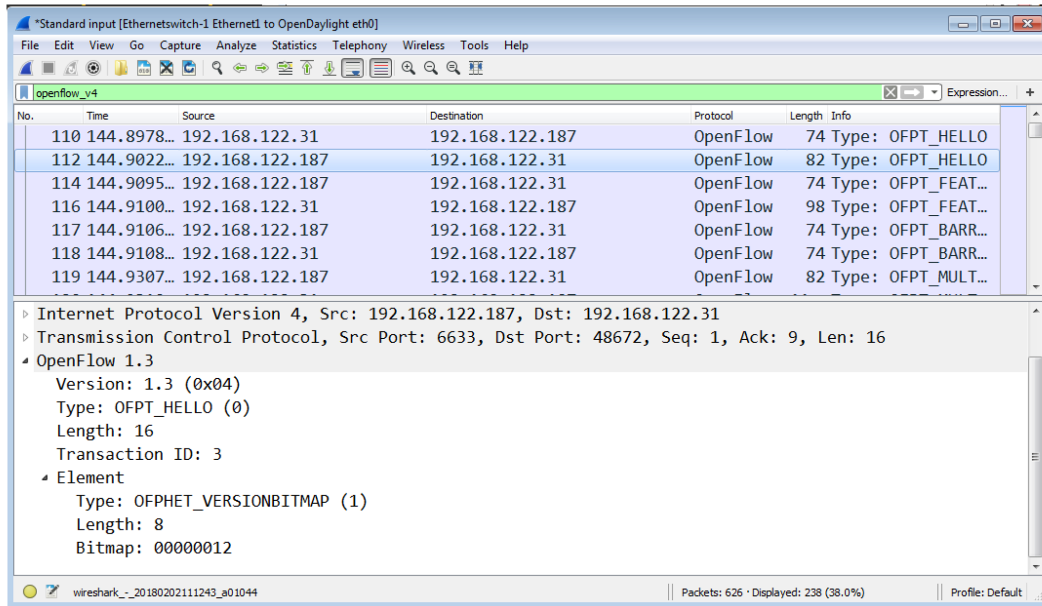


Fig. 3.6 OpenDayLight Hello reply packet

OpenFlow messages belong to a category of message depending on their type and are represented as type number in the packet header and are categorised as shown in Table 3.2. HELLO, ERROR, ECHO REQUEST, ECHO REPLY and EXPERIMENTER are all immutable messages and are required to discover and setup the secure or non-secure channel to the other network devices. This creates a security attack point for any network element as flooding of HELLO packets is the beginning of a DOS attack.

Table 3.3 Version capabilities for the OpenFlow protocol

Version	**Match Fields**	**Statistics**
1.0	Ingress Port	Per table statistics
	Ethernet: src, dst, type, VLAN	Per flow statistics
	IPv4: src, dst, proto, ToS	Per port statistics
	TCP/UDP: src port, dst port	Per queue statistics
1.1	Metadata, SCTP, VLAN tagging	Group statistics
	MPLS: label, traffic class	Action bucket statistics
1.2	OpenFlow Extensible Match (OXM)	
	IPv6: src, dst, flow label, ICMPv6	
1.3	PBB, IPv6 Extension Headers	Per-flow meter
		Per-flow meter band

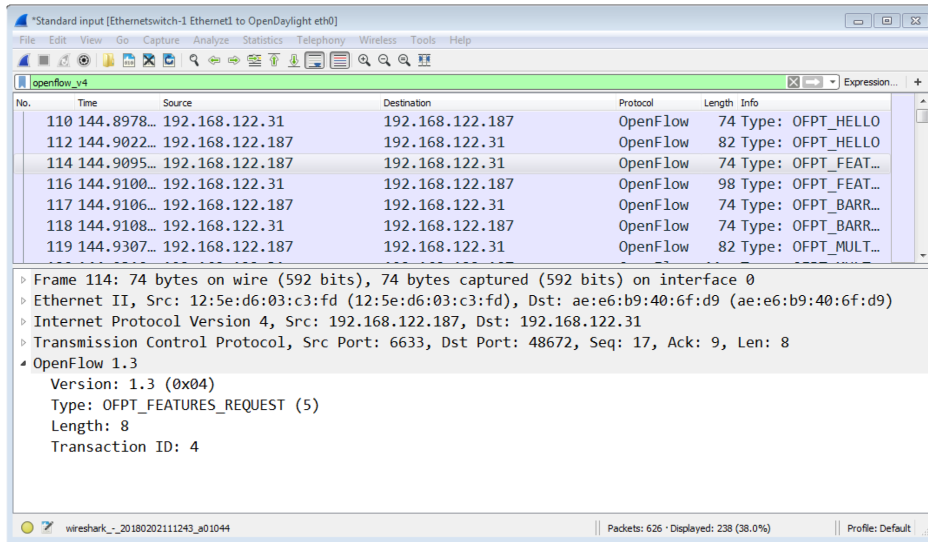


Fig. 3.7 OpenDaylight features request packet

Shown in figs 3.7 and 3.8 is the process by the controller to interrogate the switch for a list of features available for the version of software on the OpenFlow switch. The switch replies with the features list. Table 3.3 lists features available with major version releases.

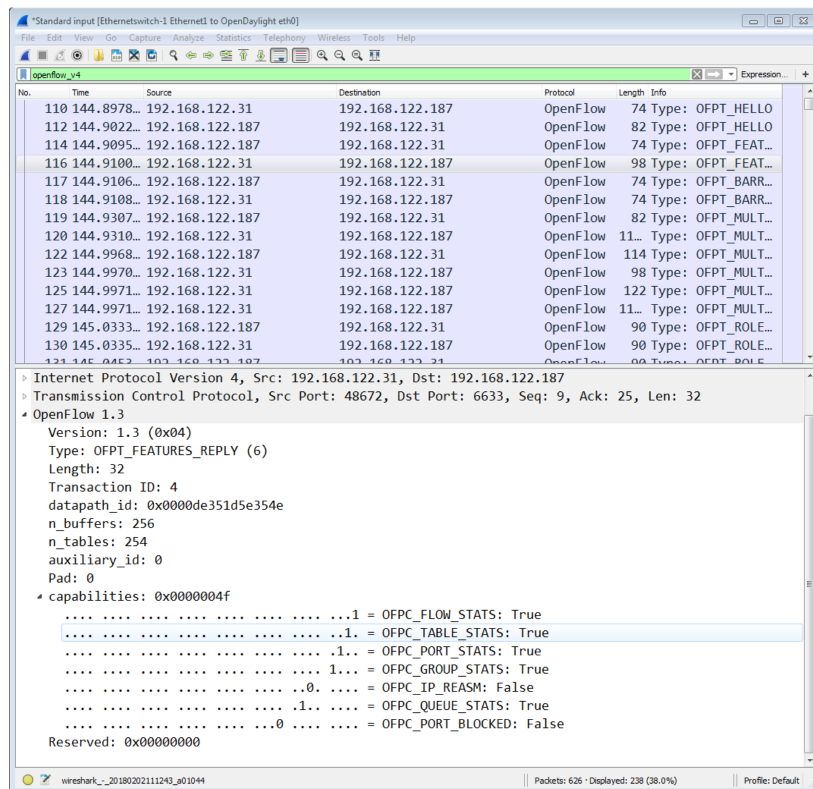


Fig. 3.8 OpenDaylight Hello features reply packet

Connection Setup

The OpenFlow switch communicates with the OpenDaylight or other OpenFlow based controllers like RYU, NOX or ONOS. The connection to the controller initiates from the switch via a Uniform Resource Identifier (URI). The configuration of the URI represents the transport protocol, the network, and the port number to connect to devices. An example is shown below:

URI Options

```
tcp:192.168.10.20:6633
tcp:192.168.10.20
tls:192.168.10.20:6633
tls:[3ffe:2a00:100:7031::1]:6633
```

Following a successful connection setup, standard OpenFlow messages exchange between the controller and switch over the channel. An OFPT_FEATURES_REQUEST, sent by the controller activates discovery of the basic capabilities of the switch. Capability of buffers, number of tables and connection type are examples of discovery.

OpenFlow controllers manage switches over one or more networks. A reliability mechanism is required to manage the connection in terms of session start up, error checking, flow control and session tear down. The channel should provide TCP/IP connectivity. TLS or TCP handle all reliable channel maintenance for the connection once it is established. Whilst the preferred method is to utilise TLS, this rarely happens. Once the switch and controller have detected each other on the network, the controller issues a reply to the switch.

Auxiliary Connections

Whilst auxiliary unreliable transport protocols UDP and DTLS are optional and allowed for channel connections, these protocols are only able to use a small subset of OpenFlow protocol messages and fields.

OpenFlow rule creation

An OpenFlow enabled switch will react to ingress packets in one of two ways. This is dependent on whether a flow rule, or match, is present for installed flow rules.

- **Proactive**: With this method, rules are inserted into switches before they are needed. (This is possible because the packet matches an existing rule)
- **Reactive**: Rules are inserted into switches in response to packets observed by the controller via TABLE_MISS messages. This response is normally to a packet that does not match a rule. The switch encapsulates the packet and sends it to the controller for a decision. The controller can either acknowledge the message and ignore the packet or respond with an action to apply an optional rule to install in the switch flow table to match future packets.

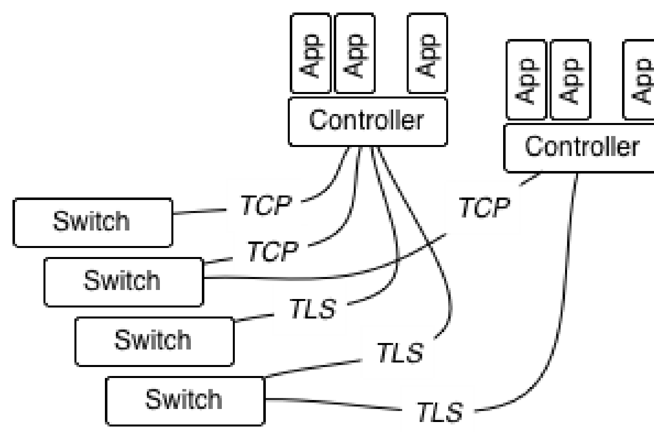


Fig. 3.9 Multiple connections TCP/TLS (Flowgrammable.com 2017)

If the switch/controller platform is secured according to specification, the flow rules for a switch can only be installed by an OpenFlow controller over a TCP connection initiated by the switch. Unfortunately, the flexibility of allowing the platform to run in a basic listener mode is attractive to network operators for reducing complexity. In this mode, the controller can allow TCP connections from any network source that introduces major vulnerabilities due to the lack of authentication and access control.

In this mode it is possible that a rouge or fake switch can bind to the controller, with the feature in later versions for the protocol to run concurrent controller connections to switches in this situation. Fig 3.9 shows multiple SDN controllers with multiple switches connected. As the number of connected switches grows, the complexity of ensuring a robust certificate exchange mechanism is implemented. The controllers shown in fig 3.9 are handling secure and insecure switch connections.

When a switch is handling a reactive rule creation there are two ways of dealing with the new flow. Either the complete packet or a portion of the packet header is transmitted to the

controller to resolve the query. Sending the complete packet, especially in a high data flow connection, would be taxing on bandwidth. If only the header were sent, it would be stored in Ternary Content-addressable Memory (TCAM) or node memory whilst waiting for a flow entry to be returned.

Considering that the default method for secure channels to initiate pairing is plain TCP, the controller is now vulnerable. If the controller and the switch are not configured for TLS, SSH or any other type of key exchange for access control, there is no limit to the number of fake/rouge switches that could be introduced to the topology. Secondly. With numerous new flows introduced, the node memory would be quickly bottlenecked, and the switch would be overloaded. This is the beginning of a Denial of Service (DoS) attack.

As the OpenFlow protocol has matured and developed, it is important to note that the mechanism for initial communication and subsequent rule handling has changed very little.

Version 1.0.0 of the OpenFlow protocol required TLS. However, for the release of version 1.4.0, clear TCP was allowed as an option.

The changes have predominately occurred in the addition of message types and the ability to have concurrent controller connections. These changes were added to version 1.3.3. This addition has included negotiation of master and equal status roles of controllers in multiple connection scenarios previously mentioned in earlier in th chapter.

Load balance roles

- **Equal** is a default role. The switch exchanges the same OpenFlow messages to each controller and does not distinguish or load balance between them.
- **Master** is similar to the equal role but there can only be one master. If a master is utilised, all other switches will become slaves.
- **Slave** is the role of all switches except the master once a master is set.

Below are the contents of an OpenFlow packet negotiating load balance roles:

```
1  handle_role_request_message (struct ofp_role_request role_request) {
       struct ofp_role_request role_reply;
2  I* pseudo-code to validate the generation_id. Here the
3  * generation_is_defined and cached_generation_id are global
4  * variables *I
5  if (generation_is_defined && (int64_t) (role_request.generation_id -
       cached_generation_id) < 0)
6  {
7  send_error_message(OFPET_ROLE_REQUEST_FAILED, OFPRRFC_STALE);
8  }
9  else
10 {
11 cached_generation_id = role_request.generation_id; generation_is_defined
       = true;
12 I* Here connection is the connection data structure which is
13 * maintained by the switch *I connection.role = role_request.role;
       role_reply.role = role_request.role;
14 role_reply.generation_id = cached_generation_id; send_openflow_message (
       connection, role_reply);
15 }
16 }
```

Chapter 4

Testing Methods and Equipment Used

In this section methods used in evaluation and analysis will be presented. To assess security for switch to controller secure channels, a variety of hardware and software were used. Evaluating connection quality and robustness should include tests to challenge and stress the connections. Combinations of switch controller connection were used in different topologies to simulate the data centre or corporate environment. A data centre environment network has the potential for load balancing between controllers. meaning connections could be established and dropped continuously over a short period.

The controller to switch connection relationship is important in the testing of secure connections. In a load balancing scenario, controllers are required to start on slave and master roles to satisfy the relationship between the controller clusters. In failing to return to the optimum role at the end of a load balancing event, does the controller fail safe?

4.1 Guidelines for openFlow device behaviour in the test environment

Testing in this section will follow the Open Networking Foundation guidelines as set out in OpenFlow Controller Benchmarking Methodologies [10].

In a secure environment, the following secure channel behaviour steps must be recognised.

a) In any switch to controller OpenFlow connection, the preference is for a TLS negotiation for the secure channel between devices. Private Key Infrastructure (PKI) must be in force on the controller with a trusted source for the Certificate Authority (CA). The controller will pass the public certificate to the switch prior to forming a secure encrypted connection at Secure Sockets Layer (SSL).

b) When switch to controller secure channel connections fail, the failure should fall back to a secure connection on restart. This is viewed as a connection failure as opposed to device failure. Connection failure could be caused by:

- Overloading of channel capacity.
- Corruption on one or both ends of the link.
- Loss of communication caused by congestion.

c) Secure channel connections created on the fly (load balancing events) should come up as TLS secure connections when slave controllers are bought online.

Table 4.1 shows configuration parameters for the controller.

Table 4.1 Controller Configuration Parameters

Parameter	**Comments**	**Iteration 1**	**Iteration 2**
Channel Type	TCP or TLS	TCP	TLS
Role Request	Must be able to send Role Request message once an OF channel is established in a switch	Enabled	Enabled
Topology Discovery	LLDP or any other protocol to discover topology	Enabled	Enabled
Echo Request/Reply	Optional parameter. Depending on the frequency of transmission, it might affect the test outcome.	Disabled	Enabled with frequency n1/sec
Support multiple controller interaction	Must support interaction with other controller and elect new master when current master goes down	Enabled	Enabled

In proactive mode, an OpenFlow enabled switch contains a flow table matching flow information and flows arriving on the ingress ports. In reactive mode the switch will not have a matching flow table for flows and will either drop the packet, or generate a TABLE_MISS result and send the packet to the controller in a PACKET_IN message.

The controller reacts to the PACKET_IN message by setting and appropriate flow for the packets. This behaviour is an opportunity to exploit the controller and test the resilience of the

connections. Controller performance is heavily influenced by the processing of PACKET_IN messages to achieve good network convergence and recovery. By writing custom packets with a fake or devised flow match and flooding the OpenFlow switches, a situation is created where the switch will continually write TABLE_MISS events and hence send all packets to the controller for a flow table entry.

4.2 OpenFlow controller capacity

Channel resilience and robustness can be measured initially through determining the capacity of the channel. Testing the capacity of the channel requires connecting multiple OpenFlow forwarding devices to the controller in an ever-increasing number and measuring throughput and latency for each added switch set.

4.2.1 Scenario 1 Load balancing

The objective is to test resilience of secure channels in the event of load balancing setting for the network. In a load balancing topology, the secondary controllers will start on demand to take the extra load experienced in the network. Based on the secure environment steps earlier mentioned in section, secondary, slave controllers are required to start in secure TLS mode to satisfy behaviour.

4.2.2 Scenario 2 Forced TABLE_MISS

Switch to controller secure connections are required to cope with multiple or continuous TABLE_MISS events at the OpenFlow enabled switches caused by flooding of custom, corrupt packets into the switches.

4.2.3 Scenario 3 Controller failure

While planning for an SDN based network deployment, it is required to know the OpenFlow controller's capacity of channel handling. Therefore a thorough benchmarking of the channel is needed. It is important to know how many simultaneous channels a controller can support, and at which rate the controller is able to establish connection with switches. Controller failure testing is designed to look at switch behaviour and general topology workload. Once a baseline is established, the controller must be connected to multiple OpenFlow switches through secured connections.

Recovery time is also an important factor in negative failure of the secure connections between devices. Recovery is measured in the ability of the controller to re-discover the network topology and re-connect to forwarding elements. Secure links can only be formed once the flooding of Link Layer Discover Protocol (LLDP). The LLDP messages are received through PACKET_IN messages. Rapid end-to-end path convergence is reliant on the discovery process for the topology.

Table 4.2 is showing configuration parameters for switches.

Table 4.2 Switch configuration parameters

Parameter	**Comments**	**Iteration 1**	**Iteration 2**
Channel Type	TCP or TLS	TCP	TLS
Multipart Reply	Should be able to reply to Multipart Request from controller	Disabled	Enabled

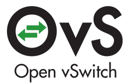
4.3 Cisco OpenFlow beta Testing Trial

Ara Institute of Canterbury is currently taking part in a trial as part of a testing lab for benchmarking of the ‘Faucet’ SDN OpenFlow controller against the Cisco 9300 Catalyst switch. The trial team was in weekly contact with the developers from Cisco to trial new updates and capabilities coded into the Cisco version of OpenFlow, reporting back on findings, bugs, successes and failures directly to Cisco.

The Faucet controller, a derivative of the RYU controller, was developed by Research and Education Network New Zealand (REANZ) in conjunction with Waikato University and is now installed in multiple corporate networks in Australia and New Zealand. Faucet is supported by multiple large vendors including Allied Telesis, HPE-Aruba, NoviFlow and Cisco. Including the Cisco 9300 in the test lab equipment list for this thesis is a bonus to the project even at the late stages of the testing.

4.4 Equipment Required

4.4.1 OpenVswitch

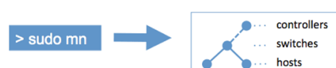


This is an open source, production ready multi-layer soft switch with native support for well-known management interfaces and protocols. The Open vSwitch community maintains and develops the capabilities of libraries, documentation and uses cases from all over the networking community. For researchers and developers, this technology provides and environment to quickly spin up a topology capable of emulation for security, monitoring, Quality of Service and automated control in minutes. Full support is for all OpenFlow versions up to v 1.5.

4.4.2 VMware ESXi

For a larger hypervisor working surface, ESXi has been idea for basing larger testing environments. There is a good selection of server hardware available for use as an SDN test bed. Utilising ESXi for this type of project minimises setup time for complex experiments allowing researchers to quickly revert between results and compare throughput of devices. There has been excellent adoption in the Open Source community making many devices available via virtual appliance format (OVA). Using Open Virtualisation Format machines and Vagrant development files, almost any configuration is available for use in this project. The ability to snapshot configurations and test conditions allow experiments to be reverted quickly back to a known state and re-run the test.

4.4.3 Mininet



This was developed by researchers at Stanford University to mitigate restrictive use of the universities network for testing and developing. Mininet is available as a pre-built virtual

machine, source code or package installation including an upgrade capability. Using process-based virtualisation, the software is capable of producing complex network components for simulation testing with a lightweight virtual footprint. Another feature is the ability for a Linux server to house the OpenFlow controller, the switch, and benchmarking utilities on the same device. For testing purposes, the time saving is significant.

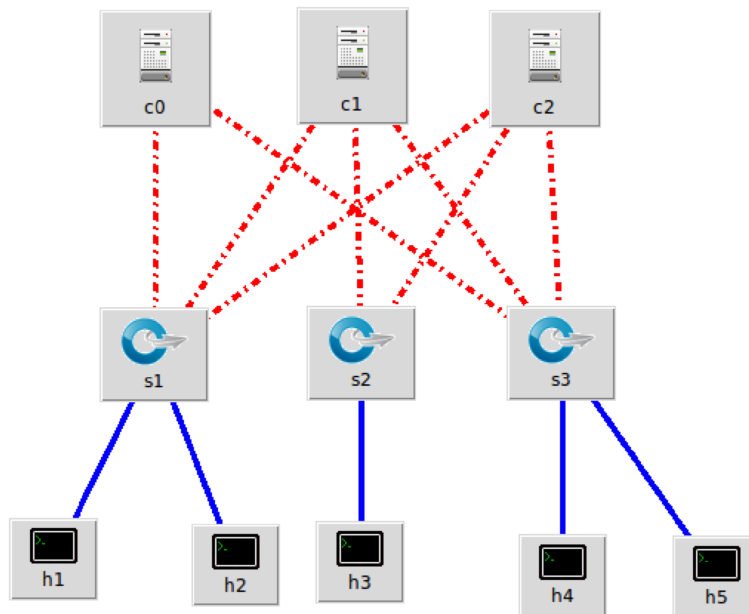


Fig. 4.1 Mininet editor topology (Edge.P 2018)

4.4.4 OpenFlow controllers

OpenDaylight

This is the main product of the ONF and labelled production ready. The OpenDaylight Controller is implemented solely in software and is kept within its own Java Virtual Machine (JVM). This means it can be deployed on hardware and operating system platforms that support Java. For best results, it is suggested that the OpenDaylight controller uses a recent Linux distribution and at least Java Virtual Machine 1.7. At the time of the Carbon release in May 2017, the project estimated that over 1 billion subscribers were accessing OpenDaylight-based networks, in addition to use within large enterprises

RYU

The RYU Controller is an open source SDN Controller which is designed to increase the speed of a network by making it easy to manipulate and change how traffic is forwarded. RYU provides software components with well-defined API that makes it easy for developers to create new network management and control applications. RYU supports various protocols for managing network devices, such as OpenFlow, Netconf, OF-config, etc.

Faucet

The Faucet controller, a derivative of the RYU controller, was developed by REANZ in conjunction with Waikato University and is now installed in multiple corporate networks in Australia and New Zealand. Faucet is supported by multiple large vendors including Allied Telesis, HPE-Aruba, NoviFlow and Cisco. Including the Cisco 9300 in the test lab equipment list for this thesis is a bonus to the project even at the late stages of the testing.

GNS3 network simulator

The GNS3 simulator is a powerful network simulation tool with numerous options for testing and benchmarking almost any device from all major vendors of switch, route, compute, storage or monitoring solutions available today. The most recent inclusions are Docker and OpenVSwitch (OVS).

Chapter 5

Testing and Performance Evaluations

5.1 Attacking the network

The following section documents tests carried out on OpenFlow controllers from an OpenFlow switch perspective. The testing assumes access to the communication channel is possible due to the absence of secure transport protocol communications between the control and data planes of the network. Without the secure link established, controllers are easy targets for port and packet mapping attacks leading to complete control of the connections.

Attacking the network is carried out with the use of modified Python scripts. The scripts are created to allow two phases of attack - a reconnaissance phase and an attack phase. The use of LLDP as a discovery protocol provides an advantage for the installation of switch and controller with each device easily discovering the other at the end of the communication channel.

Using the network topology shown in fig 4.2, it was possible to flood and overwhelm the OpenDaylight controller. If an adversary is able to talk directly to a OFCP from a rouge or fake switch, private network information and flow tables from other switches give the threat actor a complete picture of the network.

5.1.1 Reconnaissance

An OpenFlow controller that is using TCP and LLDP protocols is completely discoverable by simple fingerprinting techniques. Using LLDP to discover and enumerate the controllers southbound interface, the information harvested is used against the controller to mount more tailored attacks including DDos, Man in the middle, Replay or elevation of privileges.

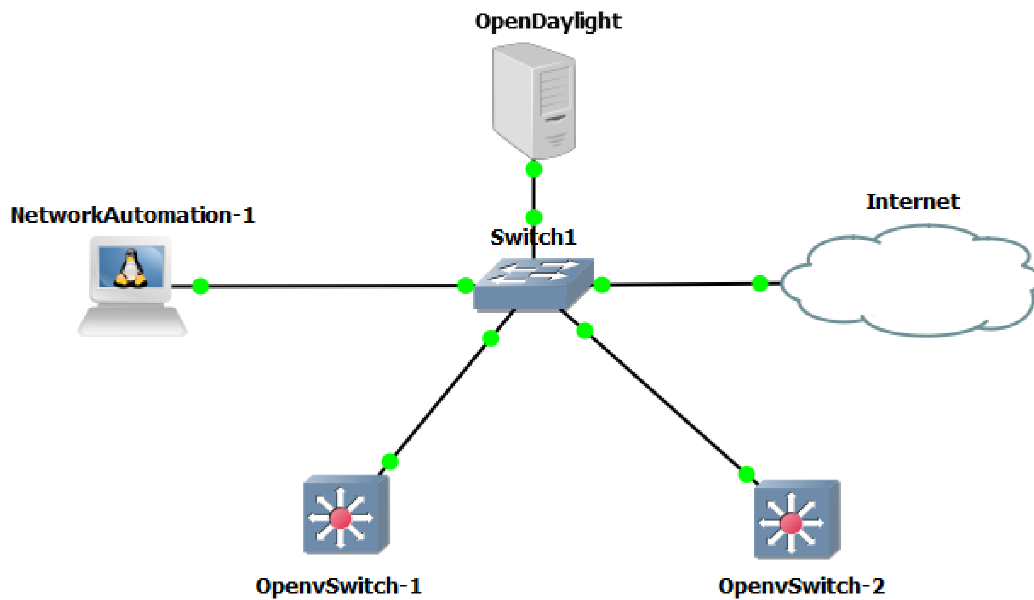


Fig. 5.1 GNS3 Topology for attacking OpenFlow controllers

Knowing the vulnerabilities of a target controller or its components ensures actors the ability to use known or craft new attacks.

Applying the reconnaissance scripts and running them on a Linux machine in the network, fingerprinting the machine displays local network information about the controller. Scripts used in the initial attack included: (Script samples available in Appendix 1)

lldp-replay

Used to discover devices listening on the network. The script replays LLDP traffic observed at a given interface back out the same interface.

- iface - Interface to use
- count - Times to replay
- capture - Capture LLDP frame to file
- replay - Replay captured LLDP frame from file

arpmon

Arpmon monitors ARP requests and responses recieved at a particular interface. In watch mode, it simply prints details of ARP traffic seen. In map mode the script will create a table of IP address to MAC bindings on sniffed ARP traffic.

– iface - Interface to use

– mode - Set mode (watch or map)

controller-detect

This module attemps to fingerprint the network controller. Using LLDP traffic, it will dump the contents of the LLDP message and start a port scan for the purposes of detecting northbound and southbound interfaces.

– iface - Interface to use

– lldp - Determine controller based on LLDP traffic

– target - Determine controller based northbound interface

– ports - Set ports to scan when –target is specified

5.1.2 Attack

As previously discussed in section 1.13, SDN networks are vulnerable to attacks that exist in conventional networks. Additionally, SDN topologies introduce new attack vectors.

ARP cache poisoning problems in SDN networks are exacerbated when controllers deliver ARP messages to the data plane and forwarding elements which in turn will pass the message onto a host connected to the port of a data plane forwarding device.

Furthermore, in and SDN network, it is possible to poison the ARP cache of a data plane element without the controller observing the attack.

dp-arp-poison

This poisons the target ARP cache without the controller observing the attack. Relies on the flows being installed for ARP traffic and this traffic not being sent to the controller by the flow. In other words, avoiding a TABLE_MISS event.

– iface - Interface to use

– victim - IP address of victim

– victim-mac - MAC address of victim

– loop - Continue poisoning until stopped

```
NXST_FLOW reply (xid=0x4):
cookie=0x20000000000000, duration=1.357s, table=0, n_packets=1,
n_bytes=42, idle_timeout=5, idle_age=0, priority=1,arp,in_port=2,
dl_src=9e:79:f1:97:f1:68,dl_dst=66:5b:51:bb:ae:eb actions=output:1

cookie=0x20000000000000, duration=0.359s, table=0, n_packets=0,
n_bytes=0, idle_timeout=5, idle_age=0, priority=1,arp,in_port=1,
dl_src=66:5b:51:bb:ae:eb,dl_dst=9e:79:f1:97:f1:68 actions=output:2
```

Above is a sample rule to allow an ARP request to traverse the network.

dp-mitm

This module is used to perform a Man in the Middle attack (MITM) without poisoning the controllers view of the network.

– ip - Interface to use

– target1 - IP address for first target

– target1- MAC - MAC address for first target

5.1.3 Attacks and prevention

The preceding sections and numerous papers written on successful attacks makes it clear that the OpenFlow protocol as a communications transport is completely vulnerable unless some form of mutual authentication is provided and enforced at each end of the connection. With so many documented cases, a recent trend is to speed up attacks and reduce time in enumerating and fingerprinting interfaces and decrease the time it takes to discover and attack an SDN network using OF as a transport. Future work needs to look closely at how we can prevent easily mounted attacks on both known and new, more sophisticated versions of switch (OVS).

5.2 Testing

Testing of the OpenFlow protocol is possible in both physical and virtual environments. Physical switches are available through numerous vendors but remained expensive at the time testing was carried out. After many unsuccessful attempts to procure or loan demonstration devices from vendors, a switch was finally loaned from Cisco Sytems. The device is a Cisco Catalyst 9300. The loan is combined with a testing program for a Beta version of the OpenFlow protocol software modified to support the 9300 switch.

As part of the research in this project, we look at methods used for discoverability of an OpenFlow enabled switch, initiating of connection to the controller and the subsequent ongoing transmission between the devices. Whilst the research is concerned mainly with security on secure channels within the topology, additional attack vectors because of TLS establishment failure raise further questions.

Traditional threat vectors that are present in a traditional network perpetuate in an SDN enabled installation. If fact, there are new threats identified with the introduction of a distributed model.

Before testing SDN controllers for the ability to deal with concurrent OpenFlow connections from multiple switches in a production environment, it is necessary to benchmark the controllers for latency capabilities in processing PACKET_IN messages to realise changes in behaviour for the production environment.

```
controller='10.0.42.5'
#controller='172.16.4.158'
port='6633'
loops=20
msPerTest='10000'
#msPerTest='3000'
macPerSwitch=1000
startupDelay=100
warmup=1
#switch amount array
#swSet=( 10 14 16 18 20)
swSet=( 16 )
#swSet=( 10 15 20 30 )
```

The following tests were processed on OpenDaylight, Faucet and RYU controllers. Testing was carried out using the Cbench benchmarking tool. When running in latency mode for a given number of unique MAC addresses per switch, cbench sends a PACKET_IN request only after the PACKET_OUT reply is received for the previous request. This gives the

number of packets a controller can process if sent serially. Taking the inverse of the cbench output for latency mode will give the average time required by the controller to process a PACKET_IN request.

As shown in output above, the tool targets a designated controller by IP address on the specified port to fake increasing numbers of switches connected to the controller. It should be noted, the small increment of switch numbers, **swSet** = (10 14 16 18 20), could be taken as emulating increased traffic from an array of network devices connected to the controller connecting/disconnecting like load balancing or disconnection in relation to path decisions at the controller dictating path control for various packet types, protocols or applications.

Logging these outputs gives a snapshot of lengthy controller operation which can be variably set via **msPerTest**='10000'.

While the increase and frequency of messaging is stabilised and controlled, the OF switches are able to deal with the steady flood of PACKET_IN messages. The control is in the warmup and delay between each increment within the switch set. The fexibility of the Cbench tool allows the tester to design monitoring schedules fine tuned to suit the device and environment.

5.2.1 Performance evaluation

To maintain consistency across all performance testing and evaluation, it is necessary to normalise the operational capabilities of controllers. Baselining tests are designed to evaluate devices and data paths for throughput thresholds and provide consistency for future testing. Once a baseline is established, security testing methods can be carried out with the confidence that the capacity of the device at the time of testing is accurately known.

Consistency is key to security testing. If the secure link fails to establish or re-establish a connection due to concurrent and multiple controller connections, a window of vulnerability appears for the network and routing domain.

It is interesting to note in Fig 4.3 below that at the point of between five and seven switches in the latency testing, the controller actually handled more flows per second and stabilised to as-steady deviation for the remainder of the test.

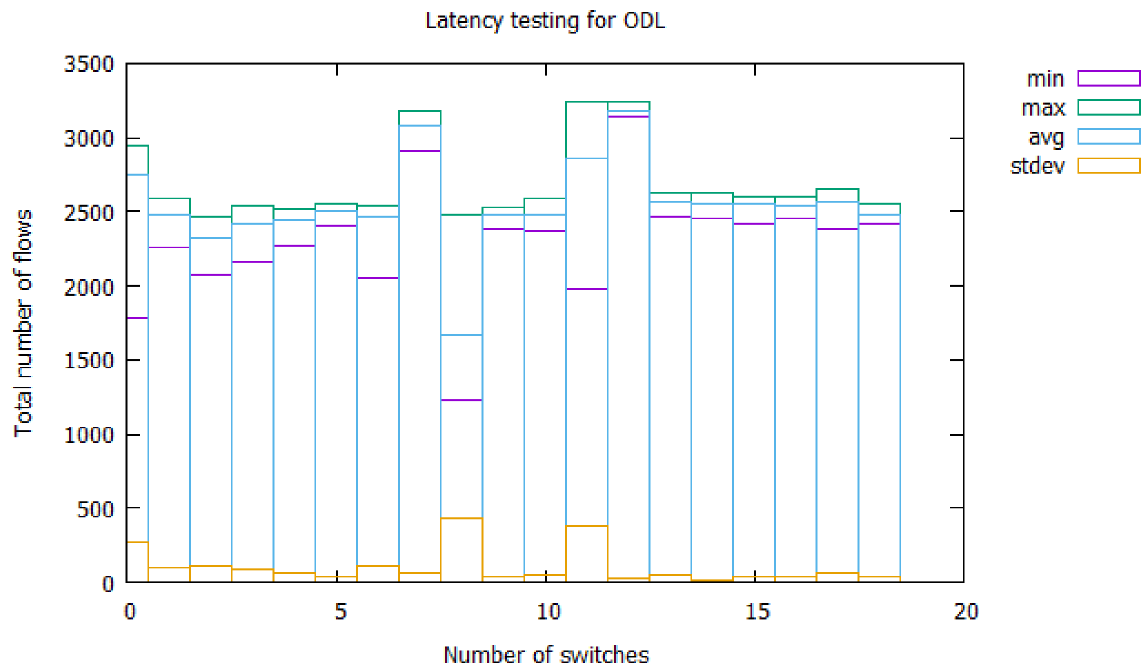


Fig. 5.2 OpenDaylight Cbench test plot

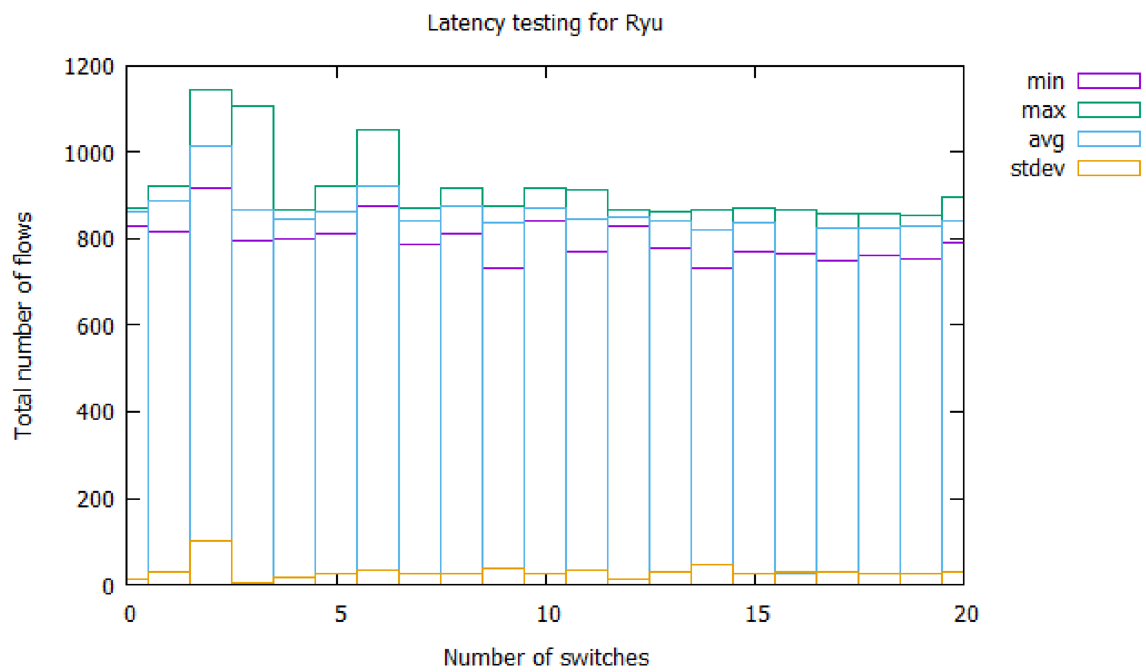


Fig. 5.3 Ryu Cbench test plot

In the most recent versions of OpenFlow including the latest version1.3.3, an added feature is the ability to have concurrent connections to multiple controllers. This feature adds a failover/redundancy to the topology and is attractive to designers for cases where loss of controller connectivity in mission critical scenarios is not an option. This also creates a complexity to the setup and maintenance of the controller to switch channel. In the early adoption of the SDN/OpenFlow, the most common method of secure channel communication was in fact no authentication.

Another real issue for concurrent connections is the bottlenecks that occur once the flow rate is increased. Some OpenFlow controllers behave unpredictably once the flow rate increases above a certain threshold. Malformed packets can cause shut down of the controller (an attack in itself). If the controller does not shutdown, the increased flow rate can cause the handshake malfunctions between switch and controller to the point where the HELLO packet that contains a negotiation of OpenFlow version number (match is required) is disregarded and the connection is possibly made between devices containing miss-matched version numbers.

5.2.2 SDN Utilisation and Adoption

The use of software and virtualisation to improve network fabric processes continues to drive the ICT industry in meeting the challenges created by today's connected, online lifestyles and business methods. This thesis presented an analytical view into security within a technology which has seen widespread adoption in the last 5 years. The adoption of new network technologies in this era can often equate to the adoption of a composite of protocols embedded into the technology without the knowledge of what is 'under the hood'. Whilst one would currently purchase a network switch with the knowledge that it is either OpenFlow capable or not, many vendor, complete solution, packages house the insecure protocols discussed in this paper.

In the case of private customer routing domains within the data centre, networks are nearing the point of full automation from customer on-boarding to actual customer data present on the provider network. Considering the implications of insecure protocols discussed in this thesis, is security guaranteed? SDN techniques are required to accommodate vendor-specific components. In the process of adoption, network, policy and operational personnel teams are becoming more diverse in membership and skill set. Project teams and Network Operation Centre (SOC) personnel are required to deal with other candidate protocols other than OpenFlow which is simply one of the many candidates mentioned throughout this thesis.

SDN techniques are an instantiation of the policy-based management framework. Within this context, SDN techniques can be used to activate capabilities on demand, to dynamically invoke network and storage resources, and to operate dynamically adaptive networks according to events (e.g., alteration of the network topology), triggers (e.g., dynamic notification of a link failure), etc. Because the complexity of activating SDN capabilities is largely hidden from the end user and is software handled, a clear understanding of the overall ecosystem is needed to figure out how to manage this complexity and to what extent this hidden complexity does not have side effects on network operation.

The concept of introducing programmability into communication networks is not new. With numerous examples introduced over the last 20 years, including telephone system programmability projects, nothing has the potential to disrupt compared with the introduction of SDN. Initial disruption came to the data centre in the form of virtualisation and the blurring of roles between Application Developers and Operations Engineers emerge as a consolidation to the role of DevOps.

The disruption is not confined to changes in communication networks and standards. It will affect skill sets required to administer the SDN aware network. Current network engineers will be required to have some software and scripting skills as automation becomes the standard method of deploying and maintaining a network.

The manufacturers of network devices are currently feeling this disruption through the growing deployment of network white boxes. (A device constructed of non-proprietary components. Silicon, hardware etc).The ideal personnel to administer an SDN based network will be programmers with core networking knowledge. How well does this profile fit the skills market today? Can we realistically allow programmers to implement applications and policy network wide without fully understanding the underlying infrastructure? Who owns the resultant outage in the case of an errant code at layer 2 or 3 or a security breach across multiple tenants in the case of ISP/Data Centre installations?

In an undocumented case within VMware Australia, developers proposed testing applications via the VMware Network Virtualisation and Security Platform (NSX) for local testing on the branch production installation. The response from the networking team inquired if the developers had knowledge of which areas of the network would be affected by the testing, whether the network would experience outages and who would be responsible for finding and eradicating errors. This level of knowledge for a programming team is beyond the scope of most software engineers and subsequently, new policy was developed to allow developers access to forwarding elements of the network if it was clearly documented they understood how the testing would affect the network.

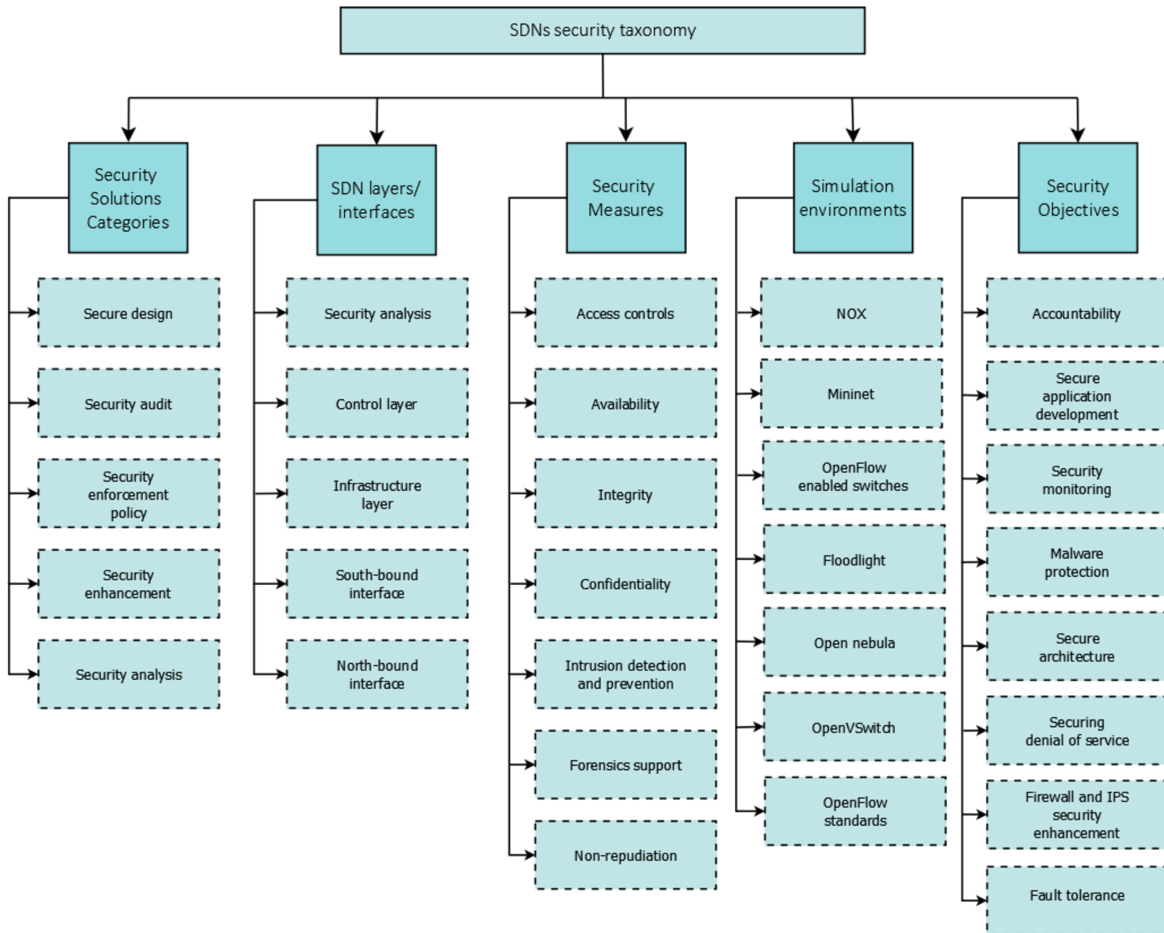


Fig. 5.4 SDN security taxonomy

The interest and subsequent discussion and development in SDN is partly based on the vision that networks can gain agility, robust monitoring, and security from the introduction of policy injection and programmability offered by these developments. Educational institutions are discovering the need to address a new type of student emerging. Somewhere between and Electro-Technologist, Network Engineer and Software Engineer exists a space where the demands of Internet of Things (IoT) with wireless sensors capable of measuring numerous elements and activities, complex virtualised network functions to collect and quantify data and software engineering to optimise and secure the network fabric privately and in the data, centre cannot be addressed by a single discipline.

The SDN security taxonomy illustrated in fig 5.1 presents a scheme for categorising areas of SDN architecture, tools and objectives for selected items. Policy Development aligned with the categories would ensure consistency across technologies with attention on standards and environments aligned with objectives.

Chapter 6

Conclusion and Future Directions

6.1 Recommendation for Future Works

Experimenter policy

The OpenFlow protocol is on a steady version development path with later iterations attempting to satisfy a market that demands more abstraction for the multi-tenant data centre installations. SD-WAN is becoming an option for service providers to build flexibility into the edge network.

The EXPERIMENTER field available in all versions since its invention, is the logical place to consider policy to enforce connection monitoring and regulation. The field is an immutable message written into the protocol for future development and research and if used is passed as a method for writing actions on a match within the OpenFlow flow tables. This paper is one of very few attempts at actual policy.

Policy development

Researching a protocol such as OpenFlow is a huge task and consumes so much time analysing all aspects of the technology. The direction of this research paper has changed somewhat from the idea of 'Security in the Software Defined Network' three years ago. In that time, many more papers have been written about security and many on attacking the protocol given the lack of security. The intention is to carry this work forward in the direction of policy and prevention. Policy development: For policy development, the idea is based on the communication method used between the controller and switch or switches. Approaches could include:

- As a priority, flow tables must support TABLE_MISS flow entries to process misses. Outcomes are sent to the controller, DHCP or to a subsequent table. Writing a reactive rule based on public and private addressing is one idea for managing control channel security and whether it should be forced.

- In response to a reactive flow, if a packet has no flow match and is sent to the controller as part of the table-miss, policy should check the connection and look for the presence of an encrypted connection, the ability to create a secure connection or send a network notification that says packets will be dropped as part of the denial process until a secure connection is estalished.

The experimenter field

As opposed to other extensions, experimenter messages are not associated with a specific OpenFlow object, and therefore can be used to create entirely new APIs and manage entirely new objects.

```
/* Experimenter extension message. */
struct ofp_experimenter_msg {
struct ofp_header header; /* Type OFPT_EXPERIMENTER. */
uint32_t experimenter; /* Experimenter ID:
* - MSB 0: low-order bytes are IEEE OUI.
* - MSB != 0: defined by ONF. */
uint32_t exp_type; /* Experimenter defined. */
/* Experimenter-defined arbitrary additional data. */
uint8_t experimenter_data[0];
};
```

The use of the EXPERIMENTER field requires absolute knowledge of the protocol and how best to utilise the extensible nature of OpenFlow Extensible Message (OXM). It is feasible that fingerprinting and exploit methods are the best developed ways of quickly establishing connection status and building the secure connection. To build a secure TLS connection, is not a trivial matter and another part of policy could look at the readiness of the interface to become secure.

Since this research began three years ago the industry has moved so far and so fast in terms of adoption and development of an advancement considered the most disruptive technology in network communications since TCP/IP. How far have we really progressed with regard to the initial questions this paper was originally asking?

How can we ensure integrity of the secure communication channel between an OpenFlow switch and controller? Are we witnessing another blind push for a technology that has not matured before being implemented in major installations and the foundation technology of numerous multi-tenanted data centre and cloud solutions?

What actually constitutes a software-programmed network? SDN is often confused with NFV and Industry is tiring of the hype that is to be SDN. Furthermore, some suggest the hype of the programmatical network has come and gone. The hype curve has been and gone.

Whatever one thinks of the past, present or future of this technology, it is making a difference to everyone who picks up a smart device, logs into a corporate network using cloud services. In fact, any use of the internet will mean contact with some form of SDN.

If SDN has indeed matured over the past 20 years, the deployment to WAN has still to make an impact on large-scale networks. As previously mentioned, NTT's innovative use of edge routing improvements and the Google B4 network are among the largest independent projects deployed as in house developments.

Numerous technology companies including Cisco, Big Switch and HPE offer turnkey solutions with a large price tag and a closed source architecture often with OpenFlow as the underlying protocol. It is not obvious if these offerings employ the security methods outlined in this paper. In the case of HPE, OpenDaylight is the underlying controller for the data centre installation.

Solutions offered by the large players are cost-prohibitive. There are alternatives that require knowledge across many of the available Open Source technologies. Central Office Redesigned as a Data Centre (CORD), developed through the ONF is one such alternative. However, adopting a solution such as CORD requires network operators to have expertise in a number of projects including SDN, NFV and multiple cloud technologies. One may consider this as a 'Back to the Future' scenario.

For two decades, Linux threatened to take over the corporate world of server and desktop deployment operating systems. Why did it never happen? All the parts were there, you just had to put it together to make it viable for your business. Consequently, Linux hardly dented the Microsoft share of this market.

Looking at the last decade, Linux is stealthily making an impact in areas of IoT, SDN and embedded devices. CORD potentially gathers a cluster of excellent, developing technologies

and if implemented successfully, turns your central office into a datacentre and natively provides programmable cloud services to the branch offices. This includes data, voice, conferencing, and security all built on free, open source software. You just need to make it work. OpenStack, Docker and Onos make up an impressive suite of programs to spin up CORD. The philosophy behind CORD is to utilise the role of the CO. More commonly referred to as the telephone exchange in Australia and New Zealand.

The aim is to eliminate the hundreds of closed proprietary devices that reside in that space and replace them with white boxes and open source network overlay. In other words, build a cloud in the central office and populate it with a datacentre that serves all the remote sites that comprise an enterprise regardless of size.

The role of SDN is without doubt, still not completely grounded the in ICT industry at this point. One thing is for certain however, it will continue to be developed as the preferred method of automating networks to provide agility and speed to changes in large enterprise environments, within LANs and multi-tenanted data centres. Some questions remain:

- What is the future of SDN in the WAN?
- What part will SDN play in the IoT?
- Will OpenFlow remain as the industry preferred protocol to develop SDN?
- How much of future security direction will rely on SDN for solutions?

The answers to the above are bound to be answered if not in a timely manner, at least in hind sight of future issues emanating from the sheer pace with which we are developing, adopting, and deploying technologies not proven as reliable. The problem of security in protocols including OpenFlow, require additional research.

References

[1] Boucadair, M. and Jacquenet, C. [2014], Software-defined networking: A perspective from within a service provider environment, Technical report.

[2] Caesar, M., Caldwell, D., Feamster, N., Rexford, J., Shaikh, A. and van der Merwe, J. [2005], Design and implementation of a routing control platform, *in* 'Proceedings of the 2Nd Conference on Symposium on Networked Systems Design & Implementation-Volume 2', USENIX Association, pp. 15–28.

[3] Chun, B., Culler, D., Roscoe, T., Bavier, A., Peterson, L., Wawrzoniak, M. and Bowman, M. [2003], 'Planetlab: an overlay testbed for broad-coverage services', *ACM SIGCOMM Computer Communication Review* **33**(3), 3–12.

[4] da Silva, S., Yemini, Y. and Florissi, D. [2001], 'The netscript active network system', *IEEE Journal on Selected Areas in Communications* **19**(3), 538–551.

[5] Doria, A., Salim, J. H., Haas, R., Khosravi, H., Wang, W., Dong, L., Gopal, R. and Halpern, J. [2010], 'Rfc 5810: Forwarding and control element separation (forces) protocol specification', *Internet Engineering Task Force* .

[6] Elliott, C. [2008], Geni-global environment for network innovations, *in* 'Local Computer Networks, 2008. LCN 2008. 33rd IEEE Conference on', IEEE, pp. 8–8.

[7] Farrel, A., Vasseur, J. and Ash, J. [2006], 'Rfc 4655: A path computation element (pce)-based architecture', *IETF, August* .

[8] Feamster, N., Balakrishnan, H., Rexford, J., Shaikh, A. and Van Der Merwe, J. [2004], The case for separating routing from routers, *in* 'Proceedings of the ACM SIGCOMM workshop on Future directions in network architecture', ACM, pp. 5–12.

[9] Feamster, N., Rexford, J. and Zegura, E. [2014], 'The road to sdn: an intellectual history of programmable networks', *ACM SIGCOMM Computer Communication Review* **44**(2), 87–98.

[10] Foundation, O. N. [n.d.], *OpenFlow Controller Benchmarking Methodologies*.

[11] Gavras, A., Karila, A., Fdida, S., May, M. and Potts, M. [2007], 'Future internet research and experimentation: the fire initiative', *ACM SIGCOMM Computer Communication Review* **37**(3), 89–92.

[12] Gude, N., Koponen, T., Pettit, J., Pfaff, B., Casado, M., McKeown, N. and Shenker, S. [2008], 'Nox: towards an operating system for networks', *ACM SIGCOMM Computer Communication Review* **38**(3), 105–110.

[13] Hu, Z., Wang, M., Yan, X., Yin, Y. and Luo, Z. [2015], A comprehensive security architecture for sdn, *in* 'Intelligence in Next Generation Networks (ICIN), 2015 18th International Conference on', IEEE, pp. 30–37.

[14] *IETF and the Internet Society* [n.d.].
URL: *https://www.internetsociety.org/internet/history-of-the-internet/ietf-internet-society/*

[15] Jain, S., Kumar, A., Mandal, S., Ong, J., Poutievski, L., Singh, A., Venkata, S., Wanderer, J., Zhou, J., Zhu, M. et al. [2013], B4: Experience with a globally-deployed software defined wan, *in* 'ACM SIGCOMM Computer Communication Review', Vol. 43, ACM, pp. 3–14.

[16] Kloti, R., Kotronis, V. and Smith, P. [2013], Openflow: A security analysis, *in* 'Network Protocols (ICNP), 2013 21st IEEE International Conference on', IEEE, pp. 1–6.

[17] Koorevaar, T., Pourzandi, M. and Zhang, Y. [2016], 'Elastic enforcement layer for cloud security using sdn'. US Patent 9,304,801.

[18] Lantz, B., Heller, B. and McKeown, N. [2010], A network in a laptop: rapid prototyping for software-defined networks, *in* 'Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks', ACM, p. 19.

[19] Laubach, M. [1994], Ip over atm working group's recommendations for the atm forum's multiprotocol bof version 1, Technical report.

[20] Marschke, D., Doyle, J. and Moyer, P. [2015], *Software Defined Networking (SDN): Anatomy of OpenFlow Volume I*, Vol. 1, Lulu. com.

[21] McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S. and Turner, J. [2008], 'Openflow: enabling innovation in campus networks', *ACM SIGCOMM Computer Communication Review* **38**(2), 69–74.

[22] Nunes, B. A. A., Mendonca, M., Nguyen, X.-N., Obraczka, K. and Turletti, T. [2014], 'A survey of software-defined networking: Past, present, and future of programmable networks', *IEEE Communications Surveys & Tutorials* **16**(3), 1617–1634.

[23] Paladi, N. [2015], Towards secure sdn policy management, *in* 'Utility and Cloud Computing (UCC), 2015 IEEE/ACM 8th International Conference on', IEEE, pp. 607–611.

[24] Pfaff, B., Pettit, J., Amidon, K., Casado, M., Koponen, T. and Shenker, S. [2009], Extending networking into the virtualization layer., *in* 'Hotnets'.

[25] Scandariato, R., Wuyts, K. and Joosen, W. [2015], 'A descriptive study of microsoft's threat modeling technique', *Requirements Engineering* **20**(2), 163–180.

[26] Schwartz, B., Jackson, A. W., Strayer, W. T., Zhou, W., Rockwell, R. D. and Partridge, C. [1999], Smart packets for active networks, *in* 'Open Architectures and Network Programming Proceedings, 1999. OPENARCH'99. 1999 IEEE Second Conference on', IEEE, pp. 90–97.

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

[28] Sezer, S., Scott-Hayward, S., Chouhan, P. K., Fraser, B., Lake, D., Finnegan, J., Viljoen, N., Miller, M. and Rao, N. [2013], 'Are we ready for sdn? implementation challenges for software-defined networks', *IEEE Communications Magazine* **51**(7), 36–43.

[29] Shalimov, A., Zuikov, D., Zimarina, D., Pashkov, V. and Smeliansky, R. [2013], Advanced study of sdn/openflow controllers, *in* 'Proceedings of the 9th central & eastern european software engineering conference in russia', ACM, p. 1.

[30] Sharma, V. et al. [2003], 'Ietf rfc 3469,"', *Framework for Multi-Protocol Label Switching (MPLS)-based Recovery* .

[31] Tennenhouse, D. L. and Wetherall, D. J. [1996], 'Towards an active network architecture', *ACM SIGCOMM Computer Communication Review* **26**(2), 5–17.

[32] Yan, Q., Yu, F. R., Gong, Q. and Li, J. [2016], 'Software-defined networking (sdn) and distributed denial of service (ddos) attacks in cloud computing environments: A survey, some research issues, and challenges', *IEEE Communications Surveys & Tutorials* **18**(1), 602–622.

Appendix A

Attack Scripts

A.1 Reconnaissance phase

A.1.1 arpmon.py

Adapted from Dylan Smyth

```
import modules.sdnpwn_common as sdnpwn

from scapy.all import *
import sys
import subprocess
import errno
import signal

class packetHandler:

    mode = None #Modes = watch, map
    hostMacMap = {}
    hostList = None
    currentPacket="";

    def __init__(self):
        currentPacket="";

    def packetIn(self, pkt):
        currentPacket=pkt;

```

```python
        if ARP in pkt:
            self.arpIn(pkt);

    def arpIn(self, pkt):
        arpTypes=['', 'who-is', 'is-at'];
        try:
            arpType=arpTypes[pkt.op];
        except:
            arpType="Unknown";

        srcIp=pkt.psrc
        srcMac=pkt.hwsrc
        dstIp=pkt.pdst

        if(self.mode == "watch"):
          #print("\n" );
          if(arpType == "who-is"):
            print("From " + str(srcIp) + "(" + str(srcMac) + ") " + arpType + " to " + dstIp);
          elif(arpType == "is-at"):
            print(str(srcIp) + " " + arpType + " " + str(srcMac) + " to " + dstIp);

        elif(self.mode == "map"):
          if(str(dstIp) not in self.hostMacMap):
            self.hostMacMap[dstIp] = "?"
          if(str(srcIp) not in self.hostMacMap):
            self.hostMacMap[srcIp] = srcMac
            printHostMacMap(self.hostMacMap);
          else:
            if(self.hostMacMap[srcIp] != srcMac):
              self.hostMacMap[srcIp] = srcMac
              printHostMacMap(self.hostMacMap);

def printHostMacMap(hostMacMap):
    subprocess.call("clear")
    print("IP\t\t\t\tMac");
    for h in sorted(hostMacMap):
        print(h + "\t\t\t" + hostMacMap[h]);

def signal_handler(signal, frame):
  print("")
```

```
62   sdnpwn.message("Stopping...", sdnpwn.NORMAL)
63   exit(0)
64
65 def info():
66   return "Monitors ARP requests and responses received at a particular
     interface. Watch mode simply print details of ARP traffic seen. Map
     mode will create a table of IPs mapped to MAC addresses based on
     sniffed ARP traffic."
67
68 def usage():
69
70   sdnpwn.addUsage("-i | --iface", "Interface to use", True)
71   sdnpwn.addUsage("-m | --mode", "Set mode ( watch or map )", True)
72
73   return sdnpwn.getUsage()
74
75 def run(params):
76
77    intf = sdnpwn.getArg(["--iface", "-i"], params)
78    mode = sdnpwn.getArg(["--mode", "-m"], params)
79
80    if((mode != None) and (intf != None)):
81        pktHandler = packetHandler();
82        pktHandler.mode = mode
83        sdnpwn.message("Starting sniffer on interface " + intf + "\n",
     sdnpwn.NORMAL);
84        signal.signal(signal.SIGINT, signal_handler)
85        sniff(iface=intf, prn=pktHandler.packetIn)
86    else:
87        print(info())
88        print(usage())
```

A.1.2 sdn-detect.py

Adapted from Dylan Smyth

```
1 import modules.sdnpwn_common as sdnpwn
2 from scapy.all import *
3 import netifaces
4 import time
5 from scipy import stats
6
```

```
def info():
  return "Determines if a network is likely to be an SDN by observing Round-Trip Times (RTT) for traffic."

def usage():
  sdnpwn.addUsage("-m", "Protocol to use (ICMP | ARP) (Default ARP)")
  sdnpwn.addUsage("-t", "IP of local host to send traffic to (Defaults to default gateway)")
  sdnpwn.addUsage("-i", "Interval at which packets are sent (Default 1)")
  sdnpwn.addUsage("-c", "Number of packets to send. More packets means better detection accuracy.(Default 10)")
  sdnpwn.addUsage("-v", "Enable verbose output")

  return sdnpwn.getUsage()

def run(params):
  global verbose

  verbose = False
  testMethod = "arp"
  dstIP = ""
  count = 10
  interval = 1

  if("-m" in params):
    testMethod = (params[params.index("-m")+1]).lower()
  if("-t" in params):
    dstIP = params[params.index("-t")+1]
  if("-i" in params):
    interval = float(params[params.index("-i")+1])
  if("-c" in params):
    count = int(params[params.index("-c")+1])
  if("-v" in params):
    verbose = True

  if(dstIP == ""):
    sdnpwn.message("No target given, using default gateway", sdnpwn.NORMAL)
    try:
      dstIP = netifaces.gateways()['default'][netifaces.AF_INET][0]
    except:
```

```
44       sdnpwn.message("Could not determine gateway address. Please
     specify a target using the -t option.", sdnpwn.ERROR)
45       return
46     sdnpwn.message("Default gateway detected as " + dstIP, sdnpwn.NORMAL
     )
47 
48   try:
49     if(testForSDN(testMethod, dstIP, count, interval)):
50       sdnpwn.message("SDN detected!", sdnpwn.SUCCESS)
51     else:
52       sdnpwn.message("SDN not detected", sdnpwn.WARNING)
53   except PermissionError as e:
54     sdnpwn.message("Needs root!", sdnpwn.ERROR)
55 
56 def testForSDN(testMethod, dstIP, count, interval):
57   global verbose
58   rtt = []
59   sentMS = 0
60 
61   if(testMethod == "icmp"):
62     sdnpwn.message("Testing with ICMP", sdnpwn.NORMAL)
63     icmp = (IP(dst=dstIP)/ICMP())
64     for i in range(0,count):
65       sentMS = int(round(time.time() * 1000))
66       resp = sr1(icmp)
67       rtt.append((int(round(time.time() * 1000))) - sentMS)
68       time.sleep(interval)
69 
70   elif(testMethod == "arp"):
71     sdnpwn.message("Testing with ARP", sdnpwn.NORMAL)
72     for i in range(0,count):
73       sentMS = int(round(time.time() * 1000))
74       resp = arping(dstIP)
75       rtt.append((int(round(time.time() * 1000))) - sentMS)
76       time.sleep(interval)
77 
78   initValue = rtt[0]
79   rtt.pop(0)
80   #Perform T-Test to check if first latency value is significantly
     different from others in our sample
81   res = stats.ttest_1samp(rtt, initValue)
82   if(verbose == True):
```

```
83      sdnpwn.message("Initial RTT: " + str(initValue), sdnpwn.VERBOSE)
84      sdnpwn.message("RTTs for other traffic: " + str(rtt), sdnpwn.VERBOSE
        )
85      sdnpwn.message("Calculated p-value for inital RTT is " + str(res[1])
        , sdnpwn.VERBOSE)
86    if(res[1] < .05 and all(i < initValue for i in rtt)): #If the p-value
        is less that 5% we can say that initValue is significant
87      return True
88    else:
89      return False
```

A.1.3 controller-detect.py

Adapted from Dylan Smyth

```
1  import signal
2  import time
3  from scipy import stats
4  import http.client as httpc
5
6  import modules.sdnpwn_common as sdnpwn
7
8  def signal_handler(signal, frame):
9    #Handle Ctrl+C here
10   print("")
11   sdnpwn.message("Stopping...", sdnpwn.NORMAL)
12   exit(0)
13
14 def info():
15   return "Attempts to fingerprint the network controller."
16
17 def usage():
18
19   sdnpwn.addUsage("-i | --iface", "Interface to use")
20   sdnpwn.addUsage("-l | --lldp", "Determine controller based off LLDP
       traffic")
21   sdnpwn.addUsage("-d | --dump-lldp", "Dump the contents of the LLDP
       message")
22   sdnpwn.addUsage("-n | --ignore-content", "Do not detect controller
       based on LLDP content")
23   sdnpwn.addUsage("-t | --target", "Determine controller based
       northbound interface")
```

```
24   sdnpwn.addUsage("-p | --ports", "Set ports to scan when --target is
       specified.")
25   sdnpwn.addUsage("-x | --proxy", "Define a proxy server to use when --
       target is specified.")
26   sdnpwn.addUsage("-v | --verbose", "Show verbose output")
27 
28   return sdnpwn.getUsage()
29 
30 def lldpListen(interface, dumpLLDP, ignoreLLDPContent):
31     sniff(iface=interface, prn=lldpListenerCallback(interface, dumpLLDP,
       ignoreLLDPContent), store=0, stop_filter=lldpStopFilter)
32 
33 def lldpListenerCallback(interface, dumpLLDP, ignoreLLDPContent):
34   def packetHandler(pkt):
35     global lldpTimeTrack
36     lldpContents = {"ONOS": "ONOS Discovery"}
37     #LLDP: 0x88cc, BDDP: 0x8942
38     if(pkt.type == 0x88cc):
39       lldpTime = int(round(time.time()))
40       if(len(lldpTimeTrack) > 0):
41         if(lldpTime == lldpTimeTrack[-1]):
42           return #This is a simple way to try to detect duplicate LLDP
       messages being picked up by the sniffer.
43       lldpTimeTrack.append(lldpTime)
44       if(ignoreLLDPContent == False):
45         for c in lldpContents:
46           if(lldpContents[c] in str(pkt)):
47             sdnpwn.printSuccess("LLDP contents matches " + c)
48             exit(0)
49       if(dumpLLDP == True):
50         print(pkt)
51   return packetHandler
52 
53 def lldpStopFilter(pkt):
54   global lldpTimeTrack
55   if(len(lldpTimeTrack) >= 6):
56     return True
57   else:
58     return False
59 
60 def run(params):
61   global lldpTimeTrack
```

```
62
63 lldpTimeTrack = []
64
65 defaultGuiPorts = {"Floodlight & OpenDayLight": 8080, "OpenDayLight (
   DLUX Standalone)": 9000, "OpenDayLight (DLUX w/t Karaf) & ONOS":
   8181}
66 defaultGuiURLs = {"Floodlight": "/ui/index.html", "OpenDayLight (DLUX)
   ": "/dlux/index.html", "OpenDayLight (Hydrogen)": "/index.html", "
   ONOS": "/onos/ui/login.html"}
67 guiIdentifiers = {}
68 ofdpIntervals = {"Floodlight": 15, "OpenDayLight (Lithium & Helium)":
   5, "OpenDayLight (Hydrogen)": 300, "Pox?": 5, "Ryu?": 1, "Beacon":
   15, "ONOS": 3}
69
70
71 iface = None
72 verbose = False
73 dumpLLDP = False
74
75 signal.signal(signal.SIGINT, signal_handler) #Assign the signal
   handler
76
77 dumpLLDP = sdnpwn.checkArg(["--dump-lldp", "-d"], params)
78 ignoreLLDPContent = sdnpwn.checkArg(["--ignore-content", "-n"], params
   )
79 verbose = sdnpwn.checkArg(["--verbose", "-v"], params)
80
81 if(sdnpwn.checkArg(["--lldp", "-l"], params)):
82   #Test by observing LLDP traffic on an interface
83   iface = sdnpwn.getArg(["--iface", "-i"], params)
84   if(iface is None):
85     sdnpwn.message("Please specify an interface with --iface option",
   sdnpwn.ERROR)
86     return
87   sdnpwn.message("Collecting 6 LLDP frames. This may take a few
   minutes...", sdnpwn.NORMAL)
88   lldpListen(iface, dumpLLDP, ignoreLLDPContent)
89   sdnpwn.message("Got all LLDP frames. Getting mean time between
   frames...", sdnpwn.NORMAL)
90   timeBetweenMessages = []
91   timeBetweenMessages.append((lldpTimeTrack[1] - lldpTimeTrack[0]))
92   timeBetweenMessages.append((lldpTimeTrack[3] - lldpTimeTrack[2]))
```

```
93      timeBetweenMessages.append((lldpTimeTrack[5] - lldpTimeTrack[4]))
94
95      meanTimeBetweenMessages = 0
96      for i in timeBetweenMessages:
97        meanTimeBetweenMessages += i
98      meanTimeBetweenMessages = round((meanTimeBetweenMessages/len(
      timeBetweenMessages)))
99
100
101
102     sdnpwn.message("Mean time between frames is: " + str(
      meanTimeBetweenMessages), sdnpwn.NORMAL)
103
104     matches = 0
105     for k in ofdpIntervals:
106       if((meanTimeBetweenMessages < (ofdpIntervals[k] + (ofdpIntervals[k
      ]/100*5))) and (meanTimeBetweenMessages > (ofdpIntervals[k] - (
      ofdpIntervals[k]/100*5)))):
107         sdnpwn.message("Mean time matches " + k, sdnpwn.NORMAL)
108         matches+=1
109     if(matches == 0):
110       sdnpwn.message("Could not determine controller from LLDP times.",
      sdnpwn.NORMAL)
111
112   elif(sdnpwn.checkArg(["--target", "-t"], params)):
113     #Test using a URL
114     target = sdnpwn.getArg(["--target", "-t"], params)
115     sdnpwn.message("Testing visibility of northbound interface on host "
        + str(target), sdnpwn.NORMAL)
116     ports = sdnpwn.getArg(["--ports", "-p"], params)
117     if(ports is None):
118       ports = []
119       for p in defaultGuiPorts:
120         ports.append(defaultGuiPorts[p])
121     else:
122       ports = ports.split(",")
123
124     sdnpwn.message("Enumerating ports...", sdnpwn.NORMAL)
125     for p in ports:
126       try:
127         conn = httpc.HTTPConnection(target, int(p))
128         if(sdnpwn.checkArg(["--proxy", "-x"], params)):
```

```
129           conn.setTunnel((sdnpwn.getArg(["--proxy", "-x"], params)))
130         req = conn.request("GET", "/")
131         sdnpwn.message("Made HTTP connection to " + str(target) + " on port " + str(p), sdnpwn.SUCCESS)
132         for c in defaultGuiPorts:
133           if(defaultGuiPorts[c] == p):
134             sdnpwn.message("Port used by " + str(c) + " for GUI interface", sdnpwn.VERBOSE)
135         sdnpwn.message("Testing GUI URLs for port " + str(p), sdnpwn.NORMAL)
136         for u in defaultGuiURLs:
137           try:
138             conn = httpc.HTTPConnection(target, int(p))
139             conn.request("GET", defaultGuiURLs[u])
140             res = conn.getresponse()
141             reqStatus = res.status
142             if(reqStatus >= 200 and reqStatus < 400):
143               sdnpwn.message("Got " + str(reqStatus) + " for " + defaultGuiURLs[u], sdnpwn.SUCCESS)
144               sdnpwn.message("URL associated with " + u + " GUI interface", sdnpwn.VERBOSE)
145             else:
146               if(verbose == True):
147                 sdnpwn.message("Got " + str(reqStatus) + " for URL " + str(u), sdnpwn.VERBOSE)
148           except Exception as e:
149             if(verbose == True):
150               sdnpwn.message("Error testing URL: " + str(e), sdnpwn.VERBOSE)
151         print("")
152       except Exception as e:
153         if(verbose == True):
154           sdnpwn.message("No connection to " + str(target) + " on port " + str(p), sdnpwn.VERBOSE)
155           sdnpwn.message(str(e), sdnpwn.VERBOSE)
156 else:
157   sdnpwn.message("No detection method given. Exiting.", sdnpwn.WARNING)
158   print(info())
159   print(usage())
160   return
```

A.2 Attack Phase

A.2.1 lldp-replay.py

```
import signal
from scapy.all import *

import modules.sdnpwn_common as sdnpwn

class FrameHandler:
  iface=None
  outFile=None

  def __init__(self, iface, outFile):
    self.iface = iface
    self.outFile = outFile

  def handler(self, pkt):
    if(pkt.type == 0x88cc): #frame is LLDP
      sdnpwn.message("Got LLDP frame...", sdnpwn.NORMAL)
      wrpcap(self.outFile, pkt)


def signal_handler(signal, frame):
  #Handle Ctrl+C here
  print("")
  sdnpwn.message("Stopping...", sdnpwn.NORMAL)
  exit(0)

def info():
  #Description of the what the module is and what it does. This function
    should return a string.
  return "Replays LLDP traffic observed at a given interface back out
    the same interface."

def usage():
  '''
  How to use the module. This function should return a string.
  sdnpwn_common contains functions to print the module usage in a table.
  These functions are "addUsage", "getUsage", and "printUsage". "
    addUsage" and "getUsage" are shown below.
```

```
35   The parameters for addUsage are option, option description, and
       required (True or False)
36   '''
37   sdnpwn.addUsage("-i | --iface", "Interface to use", True)
38   sdnpwn.addUsage("-c | --count", "Times to replay (Default 1)", False)
39   sdnpwn.addUsage("-w | --capture", "Capture LLDP frame to file", False)
40   sdnpwn.addUsage("-r | --replay", "Replay captured LLDP frame from file
       ", False)
41
42   return sdnpwn.getUsage()
43
44 def run(params):
45
46   signal.signal(signal.SIGINT, signal_handler) #Assign the signal
       handler
47
48   iface = sdnpwn.getArg(["--iface", "-i"], params)
49   count = sdnpwn.getArg(["--count", "-c"], params, 1)
50
51   if(sdnpwn.checkArg(["--capture", "-w"], params)):
52     outFile = sdnpwn.getArg(["--capture", "-w"], params)
53     frameHandler = FrameHandler(iface, outFile)
54     sdnpwn.message("Starting listener on interface " + iface, sdnpwn.
       NORMAL)
55     sniff(iface=iface, store=0, prn=frameHandler.handler, count=1,
       filter="ether proto 0x88cc")
56     sdnpwn.message("LLDP frame saved to " + outFile, sdnpwn.SUCCESS)
57   elif(sdnpwn.checkArg(["--replay", "-r"], params)):
58     inFile = sdnpwn.getArg(["--replay", "-r"], params)
59     pkt = rdpcap(inFile)
60     for c in range(int(count)):
61       sendp(pkt, iface=iface)
62     sdnpwn.message("Replayed " + inFile + " " + str(count) + " times",
       sdnpwn.SUCCESS)
```

Appendix B

Data Sheets and Specifications

B.1 Cisco 9300 Switch

B.1.1 Data Sheet

The Cisco® Catalyst® 9300 Series switches are Cisco's lead stackable enterprise switching platform built for security, IoT, mobility, and cloud. They are the next generation of the industry's most widely deployed switching platform. Catalyst 9300 Series switches form the foundational building block for Software-Defined Access (SD-Access), Cisco's lead enterprise architecture. At up to 480 Gbps, they are the industry's highest-density stacking bandwidth solution with the most flexible uplink architecture. The Catalyst 9300 Series is the first optimized platform for high-density Wi-Fi 6 and 802.11ac Wave2. It sets new maximums for network scale. These switches are also ready for the future, with an x86 CPU architecture and more memory, enabling them to host containers and run third-party applications and scripts natively within the switch.

The Catalyst 9300 Series is designed for Cisco StackWise® technology, providing flexible deployment with support for nonstop forwarding with Stateful Switchover (NSF/SSO), for the most resilient architecture in a stackable (sub-50-ms) solution. The highly resilient and efficient power architecture features Cisco StackPower®, which delivers high-density Cisco Universal Power over Ethernet (Cisco UPOE®) and Power over Ethernet Plus (PoE+) ports. The switches are based on the Cisco Unified Access™ Data Plane 2.0 (UADP) 2.0 architecture which not only protects your investment but also allows a larger scale and higher throughput. A modern operating system, Cisco IOS® XE with programmability offers advanced security capabilities and Internet of Things (IoT) convergence.

B.1.2 The Fondation of Software-Defined access

Advanced persistent security threats. The exponential growth of Internet of Things (IoT) devices. Mobility everywhere. Cloud adoption. All of these require a network fabric that integrates advanced hardware and software innovations to automate, secure, and simplify customer networks. The goal of this network fabric is to enable customer revenue growth by accelerating the rollout of business services.

The Cisco Digital Network Architecture (Cisco DNA) with Software-Defined Access (SD-Access) is the network fabric that powers business. It is an open and extensible, software-driven architecture that accelerates and simplifies your enterprise network operations. The programmable architecture frees your IT staff from time-consuming, repetitive network configuration tasks so they can focus instead on innovation that positively transforms your business. SD-Access enables policy-based automation from edge to cloud with foundational capabilities.

B.1.3 Built for security, IoT, mobility, and cloud

The Cisco® Catalyst® 9300 Series switches are Cisco's lead stackable enterprise switching platform built for security, IoT, mobility, and cloud. They are the next generation of the industry's most widely deployed switching platform. Catalyst 9300 Series switches form the foundational building block for Software-Defined Access (SD-Access), Cisco's lead enterprise architecture. At up to 480 Gbps, they are the industry's highest-density stacking bandwidth solution with the most flexible uplink architecture. The Catalyst 9300 Series is the first optimized platform for high-density Wi-Fi 6 and 802.11ac Wave2. It sets new maximums for network scale. These switches are also ready for the future, with an x86 CPU architecture and more memory, enabling them to host containers and run third-party applications and scripts natively within the switch.

The Catalyst 9300 Series is designed for Cisco StackWise® technology, providing flexible deployment with support for nonstop forwarding with Stateful Switchover (NSF/SSO), for the most resilient architecture in a stackable (sub-50-ms) solution. The highly resilient and efficient power architecture features Cisco StackPower®, which delivers high-density Cisco Universal Power over Ethernet (Cisco UPOE®) and Power over Ethernet Plus (PoE+) ports. The switches are based on the Cisco Unified Access™ Data Plane 2.0 (UADP) 2.0 architecture which not only protects your investment but also allows a larger scale and higher throughput. A modern operating system, Cisco IOS® XE with programmability offers advanced security capabilities and Internet of Things (IoT) convergence.

B.2 OpenFlow Port Specifications

B.2.1 Required Ports

OpenFlow ports are the network interfaces for passing packets between OpenFlow processing and the rest of the network. OpenFlow switches connect logically to each other via their OpenFlow ports, a packet can be forwarded from one OpenFlow switch to another OpenFlow switch only via an output OpenFlow port on the first switch and an ingress OpenFlow port on the second switch.

An OpenFlow switch makes a number of OpenFlow ports available for OpenFlow processing. The set of OpenFlow ports may not be identical to the set of network interfaces provided by the switch hardware, some network interfaces may be disabled for OpenFlow, and the OpenFlow switch may define additional OpenFlow ports.

OpenFlow packets are received on an ingress port and processed by the OpenFlow pipeline which may forward them to an output port. The packet ingress port is a property of the packet throughout the OpenFlow pipeline and represents the OpenFlow port on which the packet was received into the OpenFlow switch. The ingress port can be used when matching packets. The Open- Flow pipeline can decide to send the packet on an output port using the output action, which defines how the packet goes back to the network.

An OpenFlow switch must support three types of OpenFlow ports: physical ports, logical ports and reserved ports.

B.2.2 Physical Ports

The OpenFlow **physical ports** are switch defined ports that correspond to a hardware interface of the switch. For example, on an Ethernet switch, physical ports map one-to-one to the Ethernet interfaces.

In some deployments, the OpenFlow switch may be virtualised over the switch hardware. In those cases, an OpenFlow physical port may represent a virtual slice of the corresponding hardware interface of the switch.

B.2.3 Logical Ports

The OpenFlow **logical ports** are switch defined ports that don't correspond directly to a hardware interface of the switch. Logical ports are higher level abstractions that may be

defined in the switch using non-OpenFlow methods (e.g. link aggregation groups, tunnels, loopback interfaces).

Logical ports may include packet encapsulation and may map to various physical ports. The processing done by the logical port is implementation dependent and must be transparent to OpenFlow processing, and those ports must interact with OpenFlow processing like OpenFlow physical ports.

The only differences between physical ports and logical ports is that a packet associated with a logical port may have an extra pipeline field called Tunnel-ID associated with it and when a packet received on a logical port is sent to the controller, both its logical port and its underlying physical port are reported to the controller.

B.2.4 Reserved Ports

The OpenFlow **reserved ports** are defined by this specification. They specify generic forwarding actions such as sending to the controller, flooding, or forwarding using non-OpenFlow methods, such as "normal" switch processing.

A switch is not required to support all reserved ports, just those marked *"Required"* below.

- *Required*: **ALL**: Represents all ports the switch can use for forwarding a specific packet. Can be used only as an output port. In that case a copy of the packet starts egress processing on all standard ports, excluding the packet ingress port and ports that are configured OFPPC_NO_FWD.

- *Required*: **CONTROLLER**: Represents the control channel with the OpenFlow controllers. Can be used as an ingress port or as an output port. When used as an output port, encapsulates the packet in a packet-in message and sends it using the OpenFlow switch protocol. When used as an ingress port, this identifies a packet originating from the controller.

- *Required*: **TABLE**: Represents the start of the OpenFlow pipeline. This port is only valid in an output action in the list of actions of a packet-out message, and submits the packet to the first flow table so that the packet can be processed through the regular OpenFlow pipeline.

- *Required*: **IN PORT**: Represents the packet ingress port. Can be used only as an output port, sends the packet out through its ingress port.

- *Required*: **ANY**: Special value used in some OpenFlow requests when no port is specified (i.e. port is wildcarded). Some OpenFlow requests contain a reference to a specific port that the request only applies to. Using ANY as the port number in these requests allows that request instance to apply to any and all ports. Can neither be used as an ingress port nor as an output port.

- *Required*: **UNSET**: Special value to specify that the output port has not been set in the Action-Set. Only used when trying to match the output port in the action set using the OXM_OF_ACTSET_OUTPUT match field. Can neither be used as an ingress port nor as an output port.

- *Optional*: **LOCAL**: Represents the switch's local networking stack and its management stack. Can be used as an ingress port or as an output port. The local port enables remote entities to interact with the switch and its network services via the OpenFlow network, rather than via a separate control network. With an appropriate set of flow entries, it can be used to implement an in-band controller connection (this is outside the scope of this specification).

- *Optional*: **NORMAL**: Represents forwarding using the traditional non-OpenFlow pipeline of the switch. Can be used only as an output port and processes the packet using the normal pipeline. In general will bridge or route the packet, however the actual result is implementation dependent. If the switch cannot forward packets from the OpenFlow pipeline to the normal pipeline, it must indicate that it does not support this action.

- *Optional*: **FLOOD**: Represents flooding using the traditional non-OpenFlow pipeline of the switch. Can be used only as an output port, actual result is implementation dependent. In general will send the packet out all standard ports, but not to the ingress port, nor ports that are in OFPPS_BLOCKED state. The switch may also use the packet VLAN ID or other criteria to select which ports to use for flooding.