Lancaster University

Doctoral Thesis

Scalable and Responsive SDN Monitoring and Remediation for the Cloud-to-Fog Continuum

---

Lyndon John Fawcett

A thesis submitted in fulfilment of the requirements

for the degree of Doctor of Philosophy

in the

School of Computing and Communications

September 26, 2019

Declaration of Authorship

I, Lyndon John Fawcett, declare that this thesis titled, "Scalable and Responsive SDN Monitoring and Remediation for the Cloud-to-Fog Continuum" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

_____

Date:

_____

Abstract

Since the inception of the digital era the sharing of information has been revolutionary to the way we live, inspiring the continuous evolution of computer networks. Year by year, humankind becomes increasingly dependent on the use of connected services as new technologies evolve and become more widely accessible. As the widespread deployment of the Internet of Things, 5G, and connected cars rapidly approaches, with tens of billions of new devices connecting to the Internet, there will be a plethora of new faults and attacks that will require the need to be tracked and managed. This enormous increase on Internet reliance which is stretching the limits of current solutions to network monitoring introduces security concerns, as well as challenges of scale in operation and management. Todays conventional network monitoring and management lacks the flexibility, visibility, and intelligence required to effectively operate the next generation of the Internet. The advent of network softwarisation provides new methods for network management and operation, opening new solutions to network monitoring and remediation. In parallel, the increase in maturity of Edge computing lends itself to new solutions for scaling network softwarisation, by deploying services throughout the network.

In this thesis, two proof-of-concept systems are presented which together harness the use of Software Defined Networking, Network Functions Virtualisation, and Cloud-to-Fog computing to address challenges of scale and network security: SIREN is an open platform which manages the resources within the Internet, bridging network and infrastructure management and orchestration.

TENNISON is a network monitoring and remediation framework which tackles monitoring scalability through adapting to network context and providing a suitable architecture to the network topology, including the use of centralised, distributed, and hierarchical deployments.

Acknowledgements

With these next few sentences, I want to express my gratitude to the many people who have made my time at Lancaster University a fruitful and enjoyable experience.

Firstly, I would like to thank my supervisor, Nick Race, for his support and guidance in the PhD as well as the countless opportunities to develop myself professionally.

I would also like to thank all of the industry partners that I have engaged with during the PhD which ultimately helped in steering and validating this thesis.

During the last few years, I have been extremely lucky to work and co-author with so many brilliant people at Lancaster University. Therefore, I would like to thank my peers for their encouragement and for making my experience over the last few years both lively and productive. In no particular order, I would like to thank: Noor Shirazi, Ben Lewis, Antonios Gouglidis, Arsham Farshad, Antony Chung, Mu Mu, Colin McLaughlin, Steven Simpson, Nic Hart, Sarah Hill, Jamie Jellicoe, Jamie Bird, David Hutchison, Shiyam Alalmaei, Chris Edwards, Utz Roedig, Andreas Mauthe, Matthew Broadbent, Oliver Bates, Ric Derbyshire, Rick Withnell, Yehia El Khatib, Karen Coupe, and Ellie Davies.

I am grateful to my PhD examiners Prof. Ning Wang (University of Surrey) and Dr. Andrew Scott (Lancaster University) for taking the time to provide feedback on this thesis.

Additionally, I would also like to acknowledge and thank my family: Bryan, Judith, and Daniel for their love and support during these years.

Lastly, I would like to thank my amazing fianceé, Tansy Wickham-Pusey, who kept me sane and reassured me whenever I doubted myself. She has been a pillar of support for me and I owe a lot of my success to her.

Contents

List of Figures

List of Tables

List of Abbreviations

AI	**A**rtifical **I**ntelligence
AN	**A**ctive **N**etworks
AT	**A**ctive **T**echnologies
BSS	**B**usiness **S**ystem **S**upport
CAPEX	**C**apital **E**xpendature
CDN	**C**ontent **D**istribution **N**etwork
CM	**C**loud **m**anager
CORD	**C**entral **O**ffice **R**earchitected as a **D**atacenter
EMS	**E**lement **M**angement **S**ystem
DPI	**D**eep **P**acket **I**nspection
DOS	**D**enial **O**f **S**ervice
DDOS	**D**istributed **D**enial **O**f **S**ervice
ETSI	**E**uropean **T**elecommunications **S**tandards **I**nstitute
IDS	**I**ntrusion **D**etection **S**ystem
IPFIX	**I**P **F**low **I**nformation **E**xport
IPS	**I**ntrusion **P**revention **S**ystem
LoC	**L**ines **o**f **C**ode
LCM	**L**ife **C**ycle **M**anager
MANO	**M**anagment **AN**nd **O**rchestration
ML	**M**achine **L**earning
NFV	**N**etwork **F**unctions **V**irtualisation
NFVI	**N**etwork **F**unctions **V**irtualisation **I**nfrastucture

NFVO	**N**etwork **F**unctions **V**irtualisation **O**rchestrator
NSH	**N**etwork **S**ervice **H**eader
NBI	**N**orth **B**ound **I**nterface
OSM	**O**pen **S**ource **M**ANO
ONOS	**O**pen **N**etwork **O**perating **S**ystem
OPEX	**O**erational **E**xpendature
ODL	**O**pen **D**ay **L**ight
OvS	**O**pen **v**irtual **S**witch
OF	**O**pen **F**low
OSS	**O**perational **S**ystem **S**upport
ONAP	**O**pen **N**etwork **A**utomation **P**latform
PaaS	**P**latform **a**s **a** **S**ervice
PNF	**P**hysical **N**etwork **F**unction
PCA	**P**rinciple **C**omponent **A**nalysis
WAN	**W**ide **A**rea **N**etwork
WG	**W**orking **G**roup
VNF	**V**irtual **N**etwork **F**unction
VIM	**V**irtualised **I**frastructure **M**anager
VNFM	**V**irtual **N**etwork **F**unction **M**anager
VNFFG	**V**irtual **N**etwork **F**unction **F**orwarding **G**raph
FP	**F**alse **P**ositive
FN	**F**alse **N**egative
REST	**R**e **E**presentational **S**tate **T**ransfer
SDN	**S**oftware **D**efined **N**etworks
SFC	**S**ervice **F**orwarding **C**hain
SR	**S**ource **R**outing
SR	**S**egment **R**outing
SBI	**S**outh **B**ound **I**nterface

List of Publications

The research presented in this thesis has appeared in various journals and conferences as listed below:

1. Rotsos, Charalampos, Daniel King, Arsham Farshad, Jamie Bird, **Lyndon Fawcett**, Nektarios Georgalas, Matthias Gunkel et al. "Network service orchestration standardization: A technology survey." Computer Standards and Interfaces 54 (2017): 203-215.

2. **Fawcett, Lyndon**, and Nicholas Race. "Siren: a platform for deployment of VNFs in distributed infrastructures." In Proceedings of the Symposium on SDN Research, pp. 201-202. ACM, 2017.

3. Mu, Mu, **Lyndon Fawcett**, Jamie Bird, Jamie Jellicoe, Steven Simpson, Hans Stokking, and Nicholas Race. "Closing the gap: human factors in cross-device media synchronization." IEEE Journal of Selected Topics in Signal Processing 11, no. 1 (2017): 180-195.

4. Rotsos, Charalampos, Arsham Farshad, Nicholas Hart, Alejandro Aguado, Sarvesh Bidkar, Kyriakos Sideris, Daniel King, **Lyndon Fawcett** et al. "Baguette: Towards end-to-end service orchestration in heterogeneous networks." In Ubiquitous Computing and Communications and 2016 International Symposium on Cyberspace and Security (IUCC-CSS), International Conference on, pp. 196-203. IEEE, 2016.

5. **Fawcett, Lyndon**, Mu Mu, Matthew Broadbent, Nicholas Hart, and Nicholas Race. "SDQ: Enabling rapid QoE experimentation using Software Defined Networking." In Integrated Network and Service Management (IM), 2017 IFIP/IEEE Symposium on, pp. 656-659. IEEE, 2017.

6. **Fawcett, Lyndon**, Matthew Broadbent, and Nicholas Race. "Combinatorial Auction-Based Resource Allocation in the Fog." In Software-Defined Networks (EWSDN), 2016 Fifth European Workshop on, pp. 62-67. IEEE, 2016.

7. Lewis, Benjamin, **Lyndon Fawcett**, Matthew Broadbent, and Nicholas Race. "Using P4 to Enable Scalable Intents in Software Defined Networks." In 2018 IEEE 26th International Conference on Network Protocols (ICNP), pp. 442-443. IEEE, 2018.

8. **Fawcett, Lyndon**, Sandra Scott-Hayward, Matthew Broadbent, Andrew Wright, and Nicholas Race. "TENNISON: A Distributed SDN Framework for Scalable Network Security." IEEE Journal on Selected Areas in Communications (2018).

9. **Fawcett, Lyndon**, Matthew Broadbent, and Nicholas Race. "Siren: A platform for deploying virtual network services in the cloud to Fog continuum." In Wireless Communications and Networking Conference Workshops (WCNCW), 2018 IEEE, pp. 202-207. IEEE, 2018.

10. **Fawcett, Lyndon**, Mu Mu, Bruno Hareng, and Nicholas John Paul Race. "REF: Enabling Rapid Experimentation of Contextual Network Traffic Management using Software Defined Networking." IEEE Communications Magazine 55, no. 7 (2017): 144-150.

Chapter 1

Introduction

Historically, computer networks were purpose-built services exclusive to a few closed communities [113]. Since the commercialisation of the Internet in the 90s, computer networks have grown to become a critical infrastructure that underpins many of the amenities and fundamental services that humankind globally have grown accustomed to in modern life. Today, the Internet has a wide range of uses including: media consumption, social interaction, information retrieval, and communication. The Internet is a tool that since its birth has been continually pushing the limits of its design further [47, 9]. As a result of an increase in demand [1] and reliance on computer networks and the Internet, network operators are faced with increasing challenges of security, management scalability, resource allocation, and configuration complexity [21, 103, 102]. Additionally, due to the inflexibility of the current Internet implementation, advancements on its design are challenging [9].

In an era where the Internet is moving beyond simple data sharing to operating autonomous cars and cities, security is of uppermost importance, and the implications of a lack of security are severe [224, 201]. Despite the relative maturity of the Internet, the network and services on it are often still vulnerable to attacks, with the threat of total outages as a result. Importantly, attacks from inside the network are often more threatening than remote ones, resulting in orders of magnitude greater cost [119]. However, the edge of the network is

often neglected due to the difficulty and cost of deploying to this. Nonetheless, edge networks still need to be considered from a security point of view to effectively protect businesses and homes [192]. The importance of this has further increased with the emergence of 5G, where services are deployed throughout the network [10].

The intersection of these challenges motivates a combined approach to network security, which moves away from the current trend of protecting networks at their gateways. This thesis suggests that networks should be secured at multiple points, providing dynamic security that is proportionate to network context.

1.1 Contemporary Network Monitoring and Remediation

Network monitoring and remediation in today's computer networks is typically separated into various modular components. This includes a method of monitoring, data management, and remediation. With these, network operators face challenges when managing network anomalies and attacks, primarily due to limited data and the inability to correlate network events together [191]. Frameworks such as S-FlowRT [138], OpenNMS [16], and Zabbix [223] are used to assist network operators with network monitoring, consolidating monitoring data. On their own, these tools are limited in scalability, support for adaptation, and general flexibility [206].

In terms of network monitoring solutions, integrated offerings such as middles boxes provided in Intrusion Protection Systems add additional delay to the network and are prone to vendor lock-in, again offering limited flexibility [186]. Due to the operational and capital cost, time, and network impact of these tools,

they are only placed at key locations within a network, limiting the visibility and control of monitoring within the network.

In summary, current network monitoring and remediation solutions offer limited visibility, flexibility, control, and extensibility all whilst requiring high operational and capital costs. Given the ever growing use of the Internet, and the increase in devices on the Internet, this is an aspect networking that would greatly benefit from a new approach.

1.2 Prospects for Next Generation Networking

Telecommunications, Cloud, and edge technologies coming to the next generation of the Internet have the potential for great impact within the networking world, incrementally assisting in breaking away from the previously frigid design.

Software Defined Networking (SDN) has emerged as a concept for the dynamic control of configuration of computer networks. Arguably, its primary reason for success is the capability to implement conventional networks whilst extending them, allowing for partial adoption of the technology. Fundamentally, SDN separates the control and data planes within the network. This control is then ceded to a software-based controller, that defines the behaviour and operation of the network. The characteristics of SDN include a holistic view and ability to dynamically program the network. In addition, the protocol for communication for the control plane, OpenFlow [147], is then used to manipulate the data plane, defining counters for each flow entry/rule in the switch flow table. The flow rule definition also supports a large number of packet header fields, of which support the collection of network statistics which can be used for traffic monitoring.

These characteristics enable a powerful feedback loop as follows: network attacks can be detected by capturing traffic flow information and analysing the flow statistics with respect to known signatures/patterns or through the application of machine learning techniques. Having detected an attack by a volume, type or pattern of traffic, an appropriate action can be performed. In this situation, the benefit of SDN is that it can be used to program the flow rules to block or filter traffic, or apply another remediation mechanism. However, a performance/accuracy trade-off arises when deploying a traffic monitoring service at scale. The volume of information to be collected can lead to overall performance degradation, whilst introducing a longer collection interval can lead to inaccuracy or delayed remediation. The impact of the volume of monitoring data is particularly significant in a centralised SDN security system which carries the potential to overwhelm the controller processing functions. It is possible for the monitoring of a DoS attack to generate sufficient monitoring traffic towards the controller that the controller itself becomes subject to DoS [172]. A solution is therefore required to offer a flexible and proportionate monitoring capability with distributed functionality to disperse the control and monitoring load for scalability and resilience.

As the prospect of implementing 5G and Network Functions Virtualisation (NFV) becomes closer to reality, there is a need for an architectural change to the way that virtual network services are deployed. Emerging computing architectures such as Edge and Fog Computing are shown as key technology enablers for scalability and responsiveness for both NFV and 5G [117]. By tying SDN, NFV, Fog computing, and network security together, Figure 1.1 shows an architectural overview for a next generation network monitoring system.

When presented with new technologies, many unique and seemingly useful systems are created, however, these are often come with caveats. In particular, solutions in the research area of network orchestration and security either offer

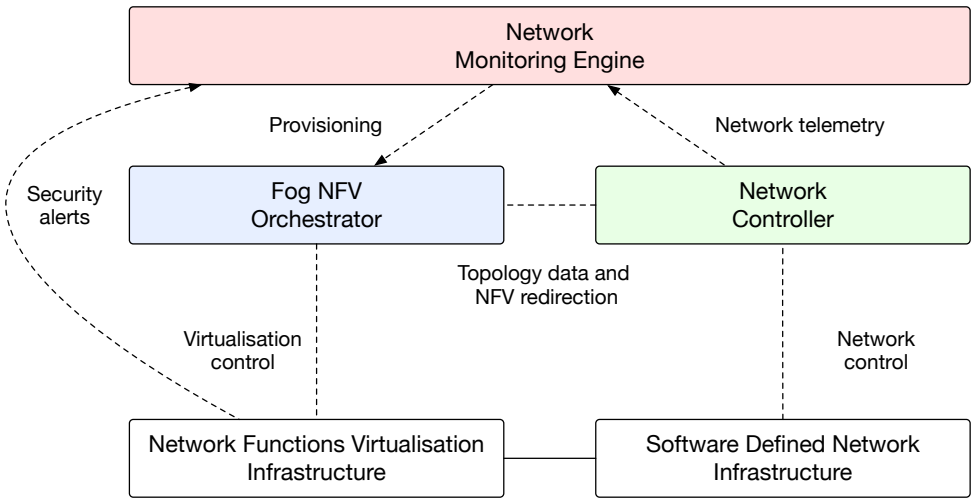


FIGURE 1.1: Next generation network monitoring architecture

little improvement or are designed to work with a single environment, often lacking features such as scalability and general applicability. This thesis targets the aforementioned gap in networking research, presenting two complementary prototype systems each with their own design, which both target scalability through different means, one focusing on Cloud-to-Fog infrastructure orchestration and the other on efficient SDN monitoring and remediation. Together, these designs create a solution to scalable and responsive SDN monitoring and remediation for the Cloud-to-Fog continuum.

1.3 Thesis Statement

This thesis testifies that by using a multi-faceted approach to SDN monitoring, a scalable solution to effective and detailed monitoring is possible. Such a solution includes utilising distributed service placement, separating networks into islands of monitoring, and an efficient design that is capable of proportionally adapting to threat.

1.4 Thesis Aims and Contributions

The contributions and aims of this thesis are as follows:

1. **Design for a scalable SDN monitoring framework (Tennison):** This thesis aims to design a modernised software solution to network monitoring. This is achieved through analysis of existing monitoring solutions as well as integration of a variety of emerging technologies. In particular, this thesis makes use of Network Functions Virtualisation, Fog Computing, and Software Defined Networking to achieve scalable and responsive monitoring.

2. **Design for a Cloud-to-Fog management and orchestration NFV platform (Siren):** To achieve scalable monitoring, this thesis details the design of a Cloud-to-Fog system capable of orchestrating and connecting monitoring components enabling Network Functions Virtualisation for Tennison. This is achieved by reviewing and adapting existing Cloud NFV specifications and technologies to work in variety of heterogeneous environments.

3. **A realisation of the scalable SDN monitoring design through a proof-of-concept:** Based on the design of Tennison, a prototype implementation is created to understand and iterative improve upon the Tennison design through observation the implications of SDN monitoring in reality. This proof-of-concept is the primary aspect of this thesis, which is evaluated for verification of thesis goals, such that the design performs as expected and that scalable and responsive SDN monitoring is feasible.

4. **A realisation of the Cloud-to-Fog management and orchestration through proof-of-concept:** Based on the design of Siren, a prototype

implementation for a Cloud-to-Fog based NFV management and orchestration platform is created to demonstrate the benefit of distributed NFV. Critical to the primary goal of this thesis, the SIREN proof-of-concept is also used to enable Cloud-to-Fog NFV for TENNISON.

5. **An evaluation of both TENNISON's and SIREN's impact and performance at scale:** The evaluation of both TENNISON and SIREN deployed and tested at scale shows the capabilities of the design as well as potential avenues for improvements made possible by forthcoming technologies. Furthermore, evaluating these systems in real-world examples, including detecting and remediating against live attacks at scale, both exercises and satisfies this thesis's claims as detailed in Section 1.3.

1.5 Thesis Structure

This thesis is structured into seven chapters details as follows:

- **Chapter 2 (Background and Related Work)** initially highlights the need for, as well as ongoing movement towards, software-based network solutions. Following this, it describes the history and evolution of programmable networks, going into detail on various methods of implementing Software Defined Networks and Network Functions Virtualisation. It then goes on to review the related work in SDN, monitoring, security, and scalability. The chapter concludes by highlighting the gaps in research that need to be filled in order to satisfy the aforementioned aims.

- **Chapter 3 (Design)** starts off by describing the Cloud-to-Fog continuum, showing its growth over the past 20 years, concluding with its applicability as an enabler for NFV. It then goes on to demonstrate through

a content caching use case the benefits of deploying network services intelligently to the edge of networks. Finally, it explores and brings to light the limitations of current solutions in this area, highlighting a gap for a new NFV orchestration system. This then moves on to discus the various design considerations for monitoring, deployment, and orchestration. Finally, the core of the chapter is revealed, detailing the design of the two primary architectural contributions in this thesis.

- **Chapter 4 (Implementation)** describes in technical detail the innovations that were required for the aims of this thesis. In particular, the implementation of the Cloud-to-Fog NFV orchestration system as well as a scalable SDN monitoring framework, which are named SIREN and TENNISON respectively are highlighted. On top of this, various smaller technical contributions are highlighted, such as NFV Overlays and an automated experimentation framework.

- **Chapter 5 (Evaluation)** demonstrates the capabilities of the implemented systems, showing details on scalability, responsiveness. This chapter also highlights the trade-offs and performance of TENNISON in three distinct deployment approaches: single, distributed, and tiered. Looking forward, this section also motivates a piece of future work through an analysis of operating with P4 over OpenFlow. Finally, the Chapter presents a functional and non-function comparison to illustrate how TENNISON compares against similar systems.

- **Chapter 6 (Conclusion)** concludes this thesis by firstly revisiting the thesis statement and then providing an overview of the contributions. The core of this chapter highlights how maturity of supporting ecosystem in NFV and SDN benefit the proof-of-concept presented in this thesis. Finally, Chapter 6 presents various avenues of future work around network

monitoring and scalability in SDN using upcoming artificial intelligence and data plane technologies.

Chapter 2

Background and Related Work

To date, network operators face management scalability, resource allocation, and configuration complexity challenges [103, 102]. These are typically resolved by over-provisioning networks, resulting in significant capital and operational expenditure [93]. On top of this, challenges in the network are likely to worsen in years to come as the Internet grows with the adoption of the Internet of Things (IoT) [125], increasing traffic demands by an order of magnitude [98]. Collectively, this is placing pressure on Internet Service Providers (ISPs) to reflect on their approach to building both network and system infrastructures and ultimately look towards more cost-effective solutions.

New networking paradigms, such as SDN and NFV offer the potential for reduced costs, better resiliency, and quicker time to market [118]. Historically, implementations of these have been limited to Cloud based environments, more recently, various efforts have looked into moving these somewhat closer to the edge [5]. With emerging computing architectures such as those introduced by Fog computing, these benefits can be expanded beyond the Cloud, making way for new services that were previously not feasible.

2.1 Programmable networks

The concept of a flexible network that can be easily configured, managed, and adapted has been desired since the early stages of the Internet and as such there are various works on the topics. In the mid 1990's a significant amount of research was conducted into active and programmable networks [23]; this was envisaged as way for new functionality and services to be realised within network infrastructures. In the last 20 years, this notion of network programmability has matured, resulting in concepts such as Network Functions Virtualisation (NFV) and Software Defined Networks (SDN). Figure 2.1 shows a high level overview of a modern SDN architecture.

This section highlights the key works that contributed to the concept of programmable networks.

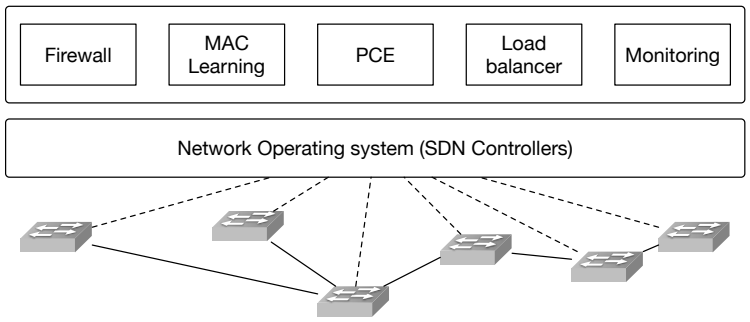


FIGURE 2.1: Simplified view of an SDN architecture

The birth of programmable networks started with Active Networking, this proposed two distinct approaches to providing flexibility within the network: programmable switches and capsules. The first approach does not require editing of the packet format. It assumes that switching devices support the dynamic loading of programs, which are used to process packets. The latter, on the other hand, suggests that packets should be replaced by micro programs, which are encapsulated in transmission frames and executed at each node along their path. In [198] an Active Network is described as a network that can be programmed remotely and can perform computation and modify packets within the data plane.

Active Networking is form of Active Technologies [198], a previous innovation in the computer systems field. In particular, this extends Active Messaging [211] in order to provide programmability to the network, with the aim of alleviating previous challenges in conventional networks, in this case known as "passive" networks.

It is thought that one of the first inspirations for programmable networks was through an idea that was conceived in the late 80s [105] called NCP [46] which relied on the separation the control plane from the data plane. Since then, there have been various notable efforts towards Active Networks including Lara++ [169], RCP [67], NCP [46], Tempest [164], GSMP [65], and PCE, [33], Tempest [164], Programming In Networks [110, 109], and Smart Packets [170].

The primary piece of work that solidified this concept was Tennenhouse's paper [197] which was the first to detail the potential of programmable networks and was a corner stone for Active Networking. Many of the paper's claims are still relevant in contemporary programmable networks. Another stepping stone into the evolution of Active Networking was Tempest [164], which was the first paper on the design of an Active Network, describing how one might implement the concept. Tempest aimed at creating an attractive and pragmatic solution for network operators that was agnostic of precise methodology and could easily run alongside existing solutions.

The next piece of high impact work was Smart Packets. Smart Packets [170] is an Active Networks implementation that relies on the use of serialised Java objects sent in single packets that represent state for the forwarding device. Smart packets and general Active Networks did not manage to achieve wide scale adoption by industry. This was partially due to the requirement of a clean slate approach and the use of new specialised hardware that no mainstream vendor was producing.

2.1.1 Software Defined Networking

Key to the advancement of programmable networks has been the concept of Software Defined Networking which was coined in the early 2000s [78]. The primary development over previous research in was the separation of the control and data planes, creating a controller based architecture. In this there have been a lot of efforts towards defining the separation of the control and data planes, notably, 4D [87] which envisions four planes. Moving on from this is Ethane and its predecessor SANE [36] which consists of a centralised architecture with two planes.

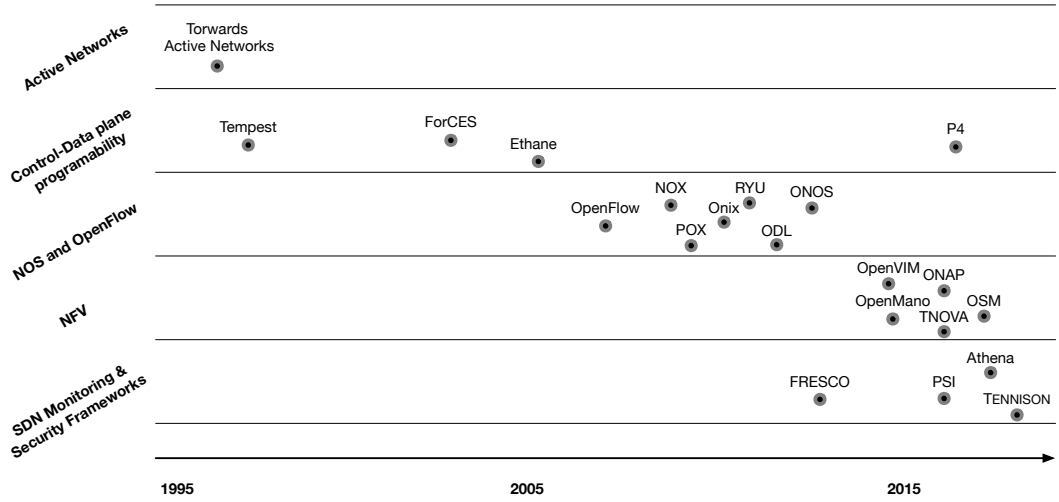


FIGURE 2.2: Selected contributions to programmable networks and SDN monitoring

ForCES [64], OpenFlow [121], and POF [187] represent the state of the art of recent approaches for designing and deploying programmable data plane devices. In a manner different from active networks, these new concepts are based on modifying forwarding devices to support flow tables, which can be dynamically configured by remote entities through simple operations such as adding, removing or updating flow rules, i.e., entries within flow tables.

Since 2015, the development of IETF's SDN architecture ABNO [3] which relies standards such as PCE [73], and ALTO [174] claims to offer a pragmatic

approach to SDN that is closer than previous efforts to being implemented by ISPs [3].

Outside of academia, Google's B4 [99] is one of the most well known successes of SDN and arguably brought the rest of industry's interest on to it. B4 is an SD-WAN deployment which was created to meet large bandwidth requirements, rate-limiting, and recently with Google Espresso [217], demand monitoring with dynamic routing at the edge. Now SDN is used by a variety of large organisations. Adoption has increased significantly with hardware vendors such as HP [95], CISCO [44], PICA [158], and Corsa [53] providing commercialised SDN solutions. On top of this, various ISPs have grown interest in SDN for its benefits in reduced CAPEX and OPEX [59, 15, 29, 42].

2.1.1.1 OpenFlow

OpenFlow is one of the first widely accepted implementations of SDN. Arguably, the primary reason why OpenFlow [121] gained traction in both academia and industry was its cleaner integration into existing networks than previous programmable network efforts [105]. On top of this, vendor support with implementations meant that OpenFlow could be deployed to the network and operate at line-rate, making it suitable for a production grade system. Figure 2.4 shows the protocol's control relationship with the layered Internet architecture. OpenFlow's first full release, OF1.0 [144] included a relatively primitive set of functionality with a 12 field match actions within a single table. This was suitable for simple routing tasks. The next popular release was OF1.3 [145]. By now the protocol had expanded to include multiple tables, metering, and groups. These are useful for a wider variety of applications including monitoring and quality of service. Figure 2.3 details the OpenFlow pipeline, which includes a series of match action tables in the data plane and an OpenFlow channel in the control plane.

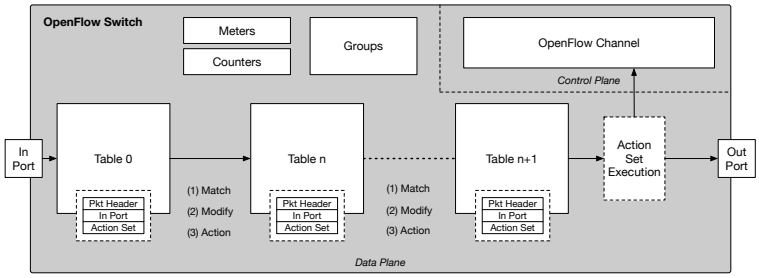


FIGURE 2.3: OpenFlow Pipeline

At the time of writing, industry and hardware vendors have settled on this version of OpenFlow, making this and previous versions compatible with the majority of OpenFlow hardware and software. Newer versions of OpenFlow up to 1.5 [146] offer improvements on the atomicity of actions between the controller and switches with the use of bundles and synchronised tables. OpenFlow's future is unclear; OpenFlow version 1.6 was drafted in 2016 but has since been kept private to ONF members [148]. With the move to new protocols such as P4, and increasing use of proprietary SDN protocols, OF1.3 could be the final widely supported version of OpenFlow. This said, prior announcements [58] from ONF for OpenFlow suggests that the protocol is going to evolve to offer a programmable data plane, which would put it in direct competition with P4.

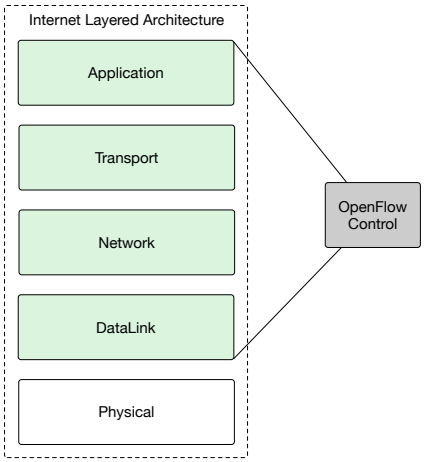


FIGURE 2.4: OpenFlow's Relationship with OSI model

2.1.1.2 P4

Similar to the prior Active Networks concept, P4 (Programming Protocol-Independent Packet Processors) provides the ability to reprogram the packet processing core of networking hardware dynamically, the benefits of this are demonstrated in PISCES [177], which uses P4 for routing traffic using custom protocols. This offers more flexibility over protocols such as OpenFlow which are currently restricted to inflexible pipelines. As described in the first P4 publication [26], the authors hope that P4 will act as a guide for OpenFlow 2.0.

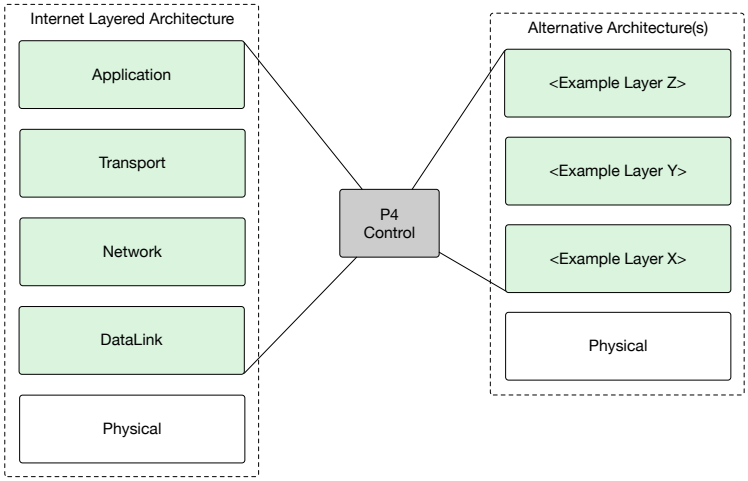


FIGURE 2.5: P4's Relationship with OSI model

Due to its infancy, unlike OpenFlow, currently there are no public industry deployments of P4. However, this is likely to change; as with OpenFlow, P4 has benefited from vendor adoption with line-rate capable hardware from BareFoot [18], and more recently Netronome [131]. Recently P4 has moved to version 16 from 14. This new version of the language generalises the language in order to improve portability of P4 applications between hardware. This move will assist in future P4 deployments.

Despite OpenFlow's success and continued progression with multiple releases, SDN networks based on this protocol have various short comings such as, restrictive programablity, controller DoS, and control plane poisoning [133].

As shown in Figure 2.5, P4 offers capability around this, allowing the network to be specialised for its purpose, as opposed to OpenFlow which apart from a limited pipeline configuration follows a one size fits all policy. Note that the layers on the right hand-side of Figure 2.5 are intentionally blank illustrating that an alternative architecture could be created with P4.

2.1.2 Scalable Programmable Networks

Since the early days of SDN and more recently with NFV, due to their centralised approach, scalability has been a concern [32]. ESTI with OSM [72] and their MANO specification [71] have envisaged various possibilities to tackle scale, these include the use of tiered orchestration as well as distributed virtualisation infrastructures. There are a number of proposed solutions to challenges around scalability within programmable networking. These include, pushing logic down to the switch, or distribution of the SDN control plane and network partitioning.

2.1.2.1 In Network Intelligence

In Network Intelligence for SDN is a method of controlling the network. Specifically this pushes control logic down to networking devices in order to improve resilience, scale, and performance [128, 56]. The topic area of In Network Intelligence currently has limited research as there are challenges around realising solutions in a pragmatic way that is usable by industry.

DIFANE [220] and DevoFlow [56] are not involved with the setup of every new flow. Instead the logic is pushed down to the switch or controller communication is used sparingly where appropriate. Whilst these solutions appear to offer an idyllic solution to programmable networking, they are not currently supported within production grade hardware.

2.1.2.2 Distributed SDN Controller Performance

As highlighted in Chapter 1, the ability to scale up controller processing in response to network traffic variations is critical to the network security system. Both hierarchical and distributed control mechanisms have been proposed, as surveyed in [172]. A few of these works specifically consider load-balancing. In Kandoo [92], local decision-making is separated from network-wide decision-making. Certain applications can be supported by event processing at local controllers reducing the load on the root controller. ElastiCon [60] proposes an elastic distributed controller architecture to dynamically adjust the controller pool in response to changing traffic conditions.

Hydra [40] presents a solution to support latency-sensitive applications by partitioning the control function based on functional slicing rather than topological slicing (as in [92, 60]). Functional slicing splits control plane functions (and, therefore, applications) across servers. The performance results presented in [40] show an improvement in controller throughput and response time under increasing load for latency-sensitive applications. However, the dependency of the solution on communication-awareness, and the anticipated variation in communication between applications in a large-scale network would require the placement algorithm to be run very frequently in larger networks.

Designed by ON.Lab, Open Network Operating System (ONOS) was launched in 2014 as a SDN network operating system for service provider networks with a focus on high availability, scalability and performance [22]. ONOS implements distributed control with multiple controller instances forming a cluster. The clustering of controllers is a process through which one or more controllers are connected and data about the state of the network is shared between them. The intention of clustering is twofold: 1) to ensure that in the event of one controller failing the other remaining controllers in the cluster will ensure the network remains functional, and 2) To provide scale-out to the system; making

it possible to manage networks with 100s of networking devices and 1000s of hosts. The ONOS cluster instances synchronise to provide the global network view graph using the RAFT [94] consensus algorithm. The StorageService interface ensures a consistent state between the databases across all the instances of an ONOS cluster. Each network element is assigned a master ONOS instance and the remaining instances will be on standby for that network element. If the master instance fails, an election takes place between the remaining instances to elect a new master. It is possible to balance the masters to provide an even distribution of network elements to each member of the cluster.

2.1.3 Software Defined Network Monitoring and Security

The first protection architecture for SDN was proposed in [37], prior to the development of the OpenFlow protocol. Since the introduction of OpenFlow, many researchers have focused on taking advantage of the characteristics of SDN for intrusion detection, monitoring and remediation services. However, there are several challenges to security in programmable network, as first highlighted by [7]. The remainder of this section details research and cutting edge industry solutions for SDN monitoring with scalability and SDN security frameworks

2.1.3.1 SDN Monitoring Solutions and Scalability

The combination of the global network view and the granularity of the network statistics captured at the data plane has generated significant interest in network monitoring advances with SDN. Combinations of traditional monitoring protocols such as NetFlow/IPFIX and sFlow with the SDN protocol, OpenFlow, have been explored [85, 222].

Prior work has aimed to tackle the challenges of monitoring at scale. For example, FlowSense [218] uses a push-based approach to receive flow statistics from switches. Adaptive rate monitoring has also been introduced; OpenNetMon [207] and OpenTM [204] poll selected switches at an adaptive rate to reduce network and switch CPU overhead. PayLess [43] uses an adaptive sampling algorithm to vary polling frequency based on measured throughput. Similarly, FlowCover [188] reduces the monitoring communication cost by optimizing the polling function. OpenMeasure [115] uses online learning to adapt flow measurement. This enables scalable measurement with monitoring of the most informative flows and optimal placement of monitoring rules across multiple switches. Proxy-based Monitoring [195] introduces a monitoring table in the proxy to specify the measurement interval for traffic monitoring and associated flow rules are pushed to the OpenFlow switches. Flow-stats-requests/replies are then only exchanged for those specified monitored flows rather than all flows. This reduces the volume of statistics communication in a similar fashion to OpenTAM [150], which is an ONOS-specific adaptive monitoring tool. However, there are several identified limitations to the work; packet capture performance is limited to 60Mbps, the system is limited to 600 condition entries (i.e. rules for capture/monitoring) and it is based on OpenFlow 1.0.

In FlowRadar [114], the authors address the challenge of monitoring in data centers where the existing NetFlow implementation options are unsuitable either due to the prohibitive cost of high-end routers (hardware-based) or excessive switch CPU resource requirements (software-based). The FlowRadar solution is to balance the workload by encoding per-flow counters with low memory requirement and constant insertion time at switches. The decoding and analysis of the flow counters is then performed at the remote collector where there is available computation resource. FlowRadar provides a scalable solution for network-wide monitoring across the data centre independent of SDN or OpenFlow.

In contrast, with [189, 190], the authors propose to use only OpenFlow messages and capabilities within SDN to emulate NetFlow in traditional networks. This works by randomly sampling the traffic and maintaining per-flow statistics in separated records that are then reported to the collector. Three different flow-sampling based methods are proposed; ip-suffix based, port-based and hash-based (5-tuple). The evaluation shows that the hash-based method achieves the best results in terms of matching the theoretical maximum flows sampled while reducing controller communication and controlling the number of entries installed at the switch.

SDN Mon [157] seeks to improve on monitoring application granularity with an SDN monitoring framework that separates the monitoring logic from the forwarding logic. SDN-Mon achieves monitoring in a similar way to that is proposed in this thesis, by using multiple tables to separate monitoring and forwarding. However, these tables are not OpenFlow tables, but instead are situated within an application that sits on a customised version of the Lagopus software switch. As such, SDN-Mon only works with this switch. UMON [213] also addresses the separation of forwarding and monitoring logic. This is achieved by introducing an additional monitoring table at the end of the forwarding pipeline. New monitoring actions are also introduced to support statistics collection on non-routing fields such as, SYN, ACK etc. This enables, for example, port scan detection based on fine-grained monitoring. However, the implementation is specific to OpenvSwitch.

Most recently, Tsai et al. [205] present an overview of SDN monitoring solutions identifying the challenges and open issues. The research developments are classified according to the monitoring phase, i.e. collection, preprocessing, transmission, analysis, and presentation, with the majority of research focused in the preprocessing phase. This includes solutions such as OpenSketch [219] and OpenTM [204] detailed above. With respect to integrating monitoring

in hybrid environments with legacy network devices, the benefit of solutions leveraging sFlow is highlighted. A number of items are identified as open issues. OpenMeasure [115] is an example of an adaptive measurement approach but without consideration of resource usage or the use of the measurement data for security functions. Leveraging monitoring and data collection to detect security threats is also identified as an area for further work with the importance of multi-domain collaborative network monitoring also highlighted.

Research from Tangari et al. [193, 194], develops on the algorithm behind adapting OpenFlow flow statistics reports in order to achieve efficient monitoring under intensive network load. On top of this, they also observe the benefit of decentralising and tiering control of SDN for improved monitoring scalability.

2.1.3.2 SDN Security Frameworks

In cutting edge large scale enterprise network deployments, typically a systems such as [96, 31, 138] are deployed to monitor, traffic on the network and enforce network policy. The following details academic and industrial SDN security frameworks that fill a similar gap.

The first protection architecture for SDN was proposed in [37], prior to the development of the OpenFlow protocol. Since the introduction of OpenFlow, various research from both academia and industry [181, 41, 89, 90, 104] has focused on taking advantage of the characteristics of SDN for intrusion detection and prevention services.

FRESCO [181] is a framework that focuses on providing a platform for rapid design and development of security specific modules as OpenFlow applications; it claims that popular security functions can be created with 90% fewer lines of code. At the core of its design it hosts a self-named "security kernel", which supports multiple security modules that can run alongside each other without conflict.

CIPA [41] applies an artificial neural networks across OF-SDN switches for the detection of distributed, coordinated intrusion attacks such as scanning, worm outbreak and DDoS. The false positive/detection rate and communication overhead are all shown as improvements over Gamer's [84] anomaly detection solution.

With [89] and [90], Ha et al. consider intrusion detection in SDNs. In [89], a flow grouping scheme is proposed to determine which flows to forward to which Intrusion Detection Systems (IDSs) to achieve the best intrusion detection performance. Principal Component Analysis (PCA) [100] is used for grouping the suspicious flows and gravity-based clustering is used to assign these groups to IDSs. In the example results, each of the network taps feed into an aggregation switch from which the assignment to IDSs is made.

In [90], the authors present results for optimising the sampling rate for each switch to improve inspection performance of malicious traffic in large networks. The sampling rate adjustment is designed to fully utilise the inspection capability of the malicious traffic while keeping the total volume of sampled traffic below the maximum processing capacity of the IDS. However, the malicious traffic rate must be estimated to begin with and can then converge to the actual value based on the adjusted sampling frequency. The selection of optimal rate would strongly influence the convergence time. Simulation results showed that the algorithm converged in about 100 s for the smaller network, which is somewhat impractical.

SDN4S [104] is proposed as a system and solution to minimise the time between incident detection and resolution by using automated countermeasures based on SDN. The system creates incident-specific response work-flows that automatically implement actions and network countermeasures. The work is motivated by the challenge of managing an increasing volume of network threats and hence security alerts, with limited resources to analyse and respond to these

alerts. The solution is based on the concept of playbooks, which match/tailor the security incident response to a high level policy. SDN4S has holds similarities with the work in this thesis; for example, there is a similar component architecture, the ability to receive alerts from external security systems, and the OpenFlow-based network protection mechanism. However, although the motivation of SDN4S is to minimise response time, there is no evaluation of the response time or of the effectiveness of the detection/protection mechanisms.

PSI [221] is proposed as a new enterprise network security architecture to address the challenge of existing enterprise security approaches lacking precise defences in *isolation, context,* and *agility*. The authors describe these as follows: for *isolation*, the defence system must ensure that security policies do not interfere with each other; for *context*, the defence system must be able to enforce customised policies for individual network devices, and for *agility*, the defence system must be able to change policy at fine-grained time-scales. PSI and this thesis address a similar set of problems related to usable network security; that of appropriate, efficient network security. Both systems achieve this by leveraging SDN and NFV. PSI emphasises the use of NFV with the tunneling of all network traffic through a cluster of virtualised appliances within which the relevant services are applied to the traffic. In contrast, this thesis emphasises the use of SDN. Rather than tunneling all traffic through a cluster (albeit virtual and hence flexibly deployed), this thesis leverages the SDN switch design to effectively apply security policy in the data plane through the selection of traffic for monitoring at different granularities. This results in flows being conservatively mirrored (rather than redirected), reducing overall network load and latency for benign traffic. Additionally, unlike PSI, the use of IPFIX and sFlow provides visibility in legacy networks.

Finally, Athena [112] is an SDN anomaly detection framework. Athena addresses the issue of scalability across large, distributed SDN deployments.

The framework supports the development of machine-learning based security applications with two scenarios illustrated; DDoS detection and Link Flooding Attack mitigation. However, Athena does not support interoperability. Furthermore, Athena does not offer adaptive measurement for resource optimisation.

Public information on industrial efforts towards creating an SDN security framework is limited. In late 2018 Corsa released a product called Red Armor [52], this commercial solution to network security uses OpenFlow to secure the network. Due to the product's infancy, there are no results on showing its capability, performance, or usefulness. This said, Corsa's recent hardware is unique as it offers a virtualisation resource on the switch. This means that the switch is capable of more taxing security and monitoring use cases, including encryption and Deep Packet Inspection.

2.1.4 Network Functions Virtualisation

NFV moves away from the traditional networking paradigm, decoupling software from hardware, making it possible to run packet processing logic elsewhere as a Virtual Network Function (VNF) [118]. This is made possible with the flexibility provided by Software Defined Networking (SDN), which separates the control and forwarding planes and enables programmability of the control plane, allowing networking hardware to be controlled remotely. These new concepts yield many benefits for ISPs which were originally only available to Cloud providers; network functions can be run in a variety of locations, scaled automatically, managed remotely, and chained together. This '*softwarisation*' of networks can benefit both customers and ISPs, with reductions in the lead-time for new services, lower capital and operational costs and savings in power consumption [6].

Realising NFV is challenging and requires automation to simplify the process. To aid this, ETSI created a specification called Management And Orchestration (MANO), which is a specification for an architecture that details a view of how NFV could be realised [70]. This architecture provides control of NFV-Infrastructures (NFVIs) to operators through multiple layers of abstraction. More recently, Verizon (and others) have advanced the MANO architecture with an SDN-NFV specification [210], this is significant because it connects the interfaces between the MANO system and the network infrastructure, which in turn creates a complete solution for deploying NFV in the Cloud. Currently the amalgamation of designs are primarily made for Cloud architectures which typically include homogeneous highly connected servers. Generally, these are extensions to Cloud Management (CM) tools [122] such as OpenStack [175].

2.2 Emerging Computing Architectures

In the ever-changing landscape of the Internet, following the success of Cloud computing the concept of Fog Computing, first coined by Flavio Bonomi at Cisco [25, 208] has recently emerged. It describes an environment where there is distributed compute and storage located between the cloud and the end point, bringing the functionality of the Cloud closer to the target in a variety compute locations known as *Cloudlets* [168], *Micro-Clouds* [68], or *Nano-Datacenters* [108]. These Fog locations can include a range of devices from low-powered Customer Premises Equipment (CPEs) at the edge of the network, servers within telephone exchanges, through to network devices within the ISP core. Comparable to the history of programmable networks, principles behind the concept of Fog computing have existed before under different names; the Fog is similar to the previously defined and less popular term The Mist [55] and complementary to the recently defined Dew computing [184]. Since the concept of Fog emerged

in 2012, two significant developments have taken place:

1) *Cisco IOx* is an advancement of Cisco's work with IoT analytics [129, 57], and is now a propriety Fog platform for interfacing with the Internet of Things (IoT). In its current state, the platform offers a method for deploying abstract policies to IOx compatible Cisco networking devices to create low cost alerts for IoT devices.

2) *OpenFog reference architecture* created by the OpenFog Consortium is the first reference architecture for the Fog [50]. The primary contribution being their description of multiple pillars required to make the Fog work. OpenFog also highlights that the Fog is multi-purpose and can be used as a service to devices on the network such as IoT, or as a service to controlling the Network.

With the increased general compute capability at each point in the Internet, the Fog can be used to provide a low latency highly scalable infrastructure for NFV [208]. However, when using Network Functions Virtualisation to provide networking services, the current solution is to deploy services on resource-rich, homogeneous, centralised, and well maintained servers within cloud data centres [123]. This contrasts with the Fog, where compute resides within a variety of locations with little to no maintenance and heterogeneous resources.

Various research efforts from ONLabs [155, 156] and multiple Universities [30, 167, 185] have already been conducted into NFV-MANO solutions hosted outside of the cloud datacentre in locations such as the operator's edge with the purpose to virtualise the customer network functions. Collectively, these papers motivate the need to move away from Cloud computing by highlighting a major challenge of NFV-MANO, which is creating an infrastructure that can scale to run millions of VNFs [123]. However, the above solutions solve challenges of NFV orchestration by creating a new infrastructure and not considering what other Fog future infrastructures may contain. NFV is not intended to require a clean slate installation, it aims to be a gradual change

between existing infrastructures. This deployment will include installation into many heterogeneous environments [122].

Further impacting NFV deployment to the Fog is that many current attempts at making NFV systems [38, 34] run each network function in its own virtual machine, this is done to simplify orchestration and to make sure that each VNF is isolated. However, this is wasteful; depending on the VNF in use, a significant amount of the compute resource is consumed by the host operating system rather than the VNF. Recent maturity of containerisation can reduce the performance impact and overall cost of running VNFs verses additional VMs [11]. Multi-tenancy can be used to further conserve compute resource by running multiple VNFs on one machine, or even in a single process. However, this complicates management and orchestration, and means that VNFs can affect each other, potentially sharing fate if one goes down. As well as this, Unikernels have been offering a different solution to conservative virtualisation, creating an operating system with only the dependencies required for its function [116]. More recently, a proof of concept has been made using Unikernels for NFV [214]. However, the performance benefits versus the complexity of creation is still unclear [28].

With these different visions on how the future internet will look with mixed compute capabilities at different NFVI-PoPs, there is no single place where all the compute resource resides. It is clear that there needs to be an orchestrator that is agnostic of what the future internet contains but also takes into consideration the capabilities of each NFVI. Highlighted by the findings in this section, there are a number of challenges that need to be addressed before the concepts of Fog computing and NFV can be brought together, including the unsuitability of current management platforms, the unique heterogeneity of resources and a volatility in the availability of such.

2.3 Management and Orchestration

One of the primary benefits of using programmable networks is the reduction of Operational Expenditure (OPEX), this is in part achieved through automation and orchestration [71, 93].

To realise NFV, innovative network service design and deployments are required. Progress has been made towards this goal in both academia and industry. Most efforts in this area have centred around designing a solution for managing and orchestrating NFV in the Cloud [72, 49], arguably the most influential work is ETSI's NFV Management and Orchestration (MANO) specification [71]. However, various other research efforts including, [69], have highlighted the benefits of running NFV services closer to the target (anything directly benefiting from the service) and from the requirements of applications like these, the concept of Fog computing has emerged.

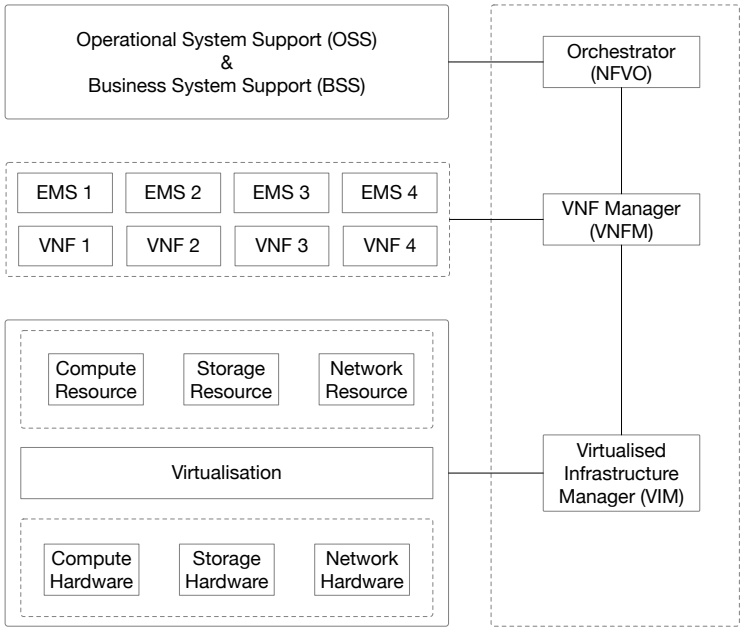


FIGURE 2.6: ETSI's NFV Framework Design Architecture

2.3.1 Management and Orchestration Standardisation

As MANOs mature, it is imperative that standardisation is used so that framework components are compatible with one another. In the area of MANO standards, there are have been many competing standards. ETSI is the first standards institution to explore the applicability of the NFV paradigm in operator infrastructures and to develop Proof of Concept implementations. Furthermore, as pictured in Figure 2.6 ETSI leads the design of the well known NFV MANO architecture [118]. NFV standardisation is not limited to ETSI, and other standardisation bodies, like the IETF NFVRG charter [132], the Open Platform for NFV (OPNFV) industrial forum [160] and the TM Forum's ZOOM, develop MANO reference implementations and propose extensions to the MANO architecture.

The MANO specifications abstract the control of virtualised infrastructures and VNF instances to external entities, like the OSS/BSS and the service orchestrator of an operator. ETSI's MANO is currently the most popular NFV management framework, with numerous open-source and commercial implementations. Operators explore the adoption of MANO compatible managements systems for various compounding reasons. Firstly, NFV MANO is a flexible component-based architecture which re-uses existing infrastructure management frameworks, like SDN Network Operating Systems and the OpenStack framework. Therefore, existing components can be extended by vendors, simplifying the development of NFV platforms. Secondly, the maturity and relatively detailed specification of the MANO components enable seamless interoperability between implementations from different vendors. Thirdly, the architecture provides by-design multiple carrier-grade features, like scalable hierarchical control, billing, and flexible service and function lifecycle specification.

Integration between the different functional components of the ETSI architecture is achieved through reference points, a distributed information plane

that models state updates and control operations. The root element of the information plane is the Network Service (NS), which represents the service chain of a service. A NS consists of one or more Virtual Network Functions (VNF), like firewalls or load balancers, connected using Virtual Links (VL), while a VNF Forwarding Graph (VNFFG) defines VNF ordering. Furthermore, a NS may include Physical Network Functions (PNF), available in the underlying network infrastructure. Finally, the MANO information model defines data repositories of NS templates, VNF catalogues, and NFVI resources, which simplify the specification and deployment of a NS.

Alongside ETSI's MANO architecture, the MEF has been steering the standardisation efforts for the MEF Lifecycle Service Orchestration (LSO) [120], which is an architecture that focuses on improving automation in network service management. MEF extends the MANO architecture and introduces support for end-to-end network infrastructure management, capitalising on the flexible control of CE technologies. LSO targets challenges of delivering Network as a Service (NaaS) functionalities in the operator infrastructure, such as on-demand, agility, and heterogeneity of virtual and physical NFs. LSO refines the service lifecycle model of the MANO standards and introduce new lifecycle capabilities, including mechanisms to automate network service request fulfilment, control of service resource and scaling, enhanced performance monitor and guarantees and assurances for service survivability. Based on [120] LSO aims to improve the time to establish and modify services for their future Internet vision. The development of the LSO standards is still in early stages and it currently focuses on service requirement specification in order to drive the architecture design. The following paragraphs detail ESTI's MANO architecture as well as various NFV forwarding technologies.

Virtualised Infrastructure Manager (VIM) The VIM provides direct control and monitoring capabilities for a single NFV Infrastructure (NFVI) domain to the upper layers of the MANO architecture. VIM responsibilities include the management of the compute, network, and storage resources of a datacenter and it exposes interfaces for resource control and VNF image management. Current implementations re-use existing Cloud Management Systems (CMS), such as OpenStack, to realize the VIM layer. Nonetheless, the design goals of existing CMSs cannot accommodate some VIM requirements, like carrier-grade support, high-performance I/O, and fine-grain and timely resource control. Currently, OPNFV [160], in collaboration with ETSI, designs and develops new open-source VIM and infrastructure virtualisation platforms, that bridge this requirement gap.

Virtual Network Function Manager (VNFM) The VNFM sits between the NFVO and the VIM systems and is responsible for the lifecycle management of individual VNF instances, including VNF configuration, monitoring, termination, and scaling. VNF management is typically realized using an Element Manager (EMS), which monitors and reports the state of each VNF to the VNFM and is capable to modify the configuration of the VNF. The deployment of an NFVM is not mandatory according to the MANO specifications and the functionality of this layer can be implemented by the NFV orchestrator. Current MANO frameworks either lack an NFVM or develop a very thin adaptation layer between the NFV orchestrator and the VIM, responsible to propagate VNF image deployment requests. Nonetheless, a VNFM can enable seamless interoperability between VNF implementations from different vendors and across cloud infrastructures.

Network Functions Virtualisation Orchestrator (NFVO) The NFVO is responsible for the deployment and dynamic re-optimisation of network services. Effectively, the NFVO receives NS requests from external entities, like the OSS and the service orchestrator, and coordinates the deployment and configuration of VNF instances across the NFVI domains. In parallel, the NFVO monitor the service performance and dynamically re-optimises the deployment of VNF instance to meet the NS requirements. When creating a new NS, the NFVO optimises placement of VNFs whilst ensuring sufficient resources and connectivity are available. Current NFVO implementations provide a thin layer capable of launching and destroying VNF chains across the NFVI domains of the operator and provide limited support for dynamic reoptimisation of the service deployment.

On top of standards around the general architecture of NFV, there are standardisation efforts around traffic routing within an NFV architecture. As this thesis makes use of NFV, understanding the surrounding NFV forwarding technologies and solutions is important. The following looks into three forwarding technologies, Service Function Chaining, Segment Routing, and a Network Service header as solutions for NFV routing and redirection.

Service Function Chaining (SFC) SFC [91] is an IETF working group as well as an architecture which both aim to define the architectural principles and protocols for the deployment and management of NF forwarding graphs. An SFC deployment operates as a network overlay, logically separating the control plane of the service from the control of the underlying network. The overlay functionality is implemented by specialised forwarding elements, using a new network header. At the time of writing, multiple open-source platforms introduce SFC support. The Open vSwitch soft-switch has introduced SFC support both in the data and the control (OpenFlow extensions) plane. The

OpenStack cloud management platform exploits the Open vSwitch SFC support and implements a high-level SFC control interface [176].

Segment Routing (SR) Segment Routing [80] is an architecture for the instantiation of service graphs over a network infrastructure using source routing mechanisms, specified by the IETF Source Packet Routing in Networking (SPRING) WG [159]. SR is a data plane technology and uses existing protocols to store instructions (segments) for the packet path in its header. SR segments can have local or global semantics, and the architecture defines three segments types: a node segment forwards a packet over the shortest path towards a network node, an adjacency segment forwards the packet through a specific router port and a service segment introduces service differentiation on a service path.

Next Generation Service Overlay Networks (NGSON) NGSON [111] is an IEEE standardisation effort [97] that allows for the establishment of dynamic services across different service providers. It boasts support for context-aware services chains that enhance the network Quality of Experience.

Network Service Header (NSH) NSH [161] is an encapsulation technique for forwarding traffic to NFV clusters and through service chains. The Network Service Header contains information that defines the position of a packet in the service path, using a service path and path index identifiers, and carries metadata between service functions regarding policy and post-service delivery. At the time of writing, NSH is not widely used due to it having no support or implementation in networking hardware.

2.3.2 NFV Management and Orchestration Implementations

The following section details the state of art NFV MANO implementations. As MANO and NFV are relatively new concepts, the following implementations are relatively immature and in most cases incomplete and not production ready.

Cloudify [48] was initially built as a generic Cloud orchestrator, but has more recently shifted its focus towards NFV, adopting specifications and standards from the ETSI MANO, YANG and TOSCA groups. Cloudify acts as an overlay framework over various technologies. It offers a multi-VIM hybrid approach for managing the Cloud, assuming that system operators may have more than one type of infrastructure that they want to control. Furthermore, it supports deep integration with various existing tools, including those responsible for automation, containerisation, and orchestration. Cloudify also has optional agentless support, broadening support further. However, this mode somewhat limits Cloudifys lifecycle management.

Open Source MANO (OSM) [72] works alongside the ETSI MANO specification and is based on an earlier project, OpenMANO. By default it uses OpenVIM, but this is changeable for other VIMs, including OpenStack. OSM orchestrates instances of OpenMANO, doing so through the RIFT.io project. This manages distributed MANOs to provide NFV capabilities at scale.

OpenBaton [35] is a product of the OpenSDNCore project and is built up to be an ETSI-compliant MANO. It has support for all of the components in the ETSI specification. The software supports OpenStack as an infrastructure. The Virtual Infrastructure Manager (VIM) within OpenBaton is separate component to the core so that it can be swapped out for alternative VIMs. Agents

are used within VMs as part of the service package to execute and monitor the services.

ONAP [135] is a merge of Open-O and ECOMP [143] and is under the Linux foundation. It is an NFV-MANO framework uses ODL and OpenStack to offer orchestration and lifecycle monitoring. ONAP's main benefit over other orchestrators is collaboration with other partners and its data centric approach to orchestration [136].

ZOOM [203] is part of the TM Forum and focuses on research and standardisation of Zero-touch orchestration and management. The project has a particular focus around support for hybrid networks and design of future network operations systems.

SONATA [66] is an NFV framework that provides a suite of tools in a SDK to offer a DevOps-enabled service platform and orchestration system.

T-NOVA [216] is a MANO for provisioning of Network Functions as a service (NFaaS). Its primary focus is around NFaaS where the orchestrator uses a Network Function marketplace where network functions can be bought and sold between vendors and operators.

OpenContrail [182, 45] is an NFV solution based on OpenStack which focuses on achieving forwarding data plane traffic to NFV clusters using network tunnels and BGP to route traffic.

CloudNFV [49] is an orchestration platform with SDN functionality. It's primary selling point is that it uses a unified management with orchestration in a single data model. This relies on OpenStack as the VIM.

OpenVIM [151] focuses on the VIM layer of the ESTI MANO architecture, providing features not available in current cloud management systems that are relevant to NFV. On top of this, it also offers a lighter weight solution to virtual machine management than OpenStack [173].

CORD [154] is a MANO based on the use of compute resources within the datacenter so that NFV can be placed somewhat closer to the edge of the network. Furthermore CORD offers a full stack solution that makes use of the SDN controller ONOS.

Open Platform for NFV (OPNFV) [160] is a project focusing on providing carrier-grade NFV, and moving standards forward. The lower infrastructure management layers have received the most attention, as to build a strong foundation. OPNFV currently has no orchestration layer and therefore cannot be used on its own, but offers a basis to start or be used in conjunction with another MANO. The current VIM supports OpenStack, but support for ARM architectures is likely forthcoming as part of the ARM-Band project.

2.3.3 Container Management and Orchestration Implementations

Unlike systems based on ETSI's MANO architecture, container orchestration tools are not typically designed with NFV as the primary use case. Rather, they are intended to help Cloud operators manage their infrastructure and to help developers deploy and scale services more easily. Since these are also important features required in a Fog scenario, they warrant consideration in this context too. Furthermore, because they have existed for longer and have a broader purpose, these projects are generally more mature offering more features and greater stability.

Kubernetes [107] is a container orchestration tool originally designed by Google to automate operation, scale, and deployment of containers across a cluster of machines. As well as deploying services, Kubernetes has a major focus on maintaining the services that it deploys, with policies for automatic scaling and failover. In order for a master to control the workers a client must be installed on each worker node. These software components are compatible with both ARM and x86 architectures. As of 2019, a sub-project under Kubernetes named KubeEdge has been announced [106] that specifically aims to address the challenges of applying Kubernetes to edge networks.

Swarm [63] is a native clustering tool for Docker which now comes packaged with the Docker engine. It uses the Docker remote API to manage and run containers on multiple hosts. It also uses an agent on each host which is managed by a single Swarm manager. Swarms orchestration consists of clustering; it makes a group of Docker hosts appear as a single machine. Services are run from the manager machine and are served to the Swarm workers.

Fleet [83] is a cluster management tool from CoreOS that acts as a foundation layer for other higher layer container orchestration solutions. Fleet is built upon the Linux init system, specifically systemd, which is responsible for initialising and managing services. Fleet extends this functionality across a cluster of machines. In order to use Fleet, agents (the CoreOS operating system) must be installed on all hosts.

MESOS [12] is a cluster management tool that provides a distributed systems kernel abstraction: it clusters a group of servers and makes them appear as one. MESOS is designed to work at large scale with examples including thousands of hosts. Similar to Fleet, MESOS offers management of the lower layers, but also supports the higher layer orchestration through Platform-as-a-Service

(PaaS) solutions such as Mesosphere Marathon. To use MESOS there must be a master node, and agents must be installed on the workers.

Rancher [162] is a container management platform from Rancher Labs that offers orchestration through a framework called Cattle. It is a fork from Swarm and thus offers similar orchestration capabilities. As well as this, Rancher supports MESOS, Swarm, and Kubernetes.

Mirantis [127] Recently Mirantis have created MCP Edge [126] which specifically aims at deploying from the core to the edge of the network. This aims to offer production grade cloud control over the edge of the network, providing the usual benefits such as system monitoring and support for CI/CD.

Some of these tools offer more complete solutions to NFV and orchestration than others. When considering which is appropriate, it is important to note that when using a complete solution, extending or adapting the design can be challenge. On the other hand, when using general or half-stack solutions, there is greater flexibility for the design of a new system and architecture. As the relative youth of NFV, most of these solutions are missing features required for deployment. One of theses being VNF forwarding graphs. Some have this in concept but production ready standardised implementations do not exist. The OpenStack consortium has a work in progress solution of this [149] so they may be waiting for that to be finished. On top of this, these orchestrators are typically designed with the notion that the resource is going to be physically homogeneous and networked similarly and thus are not well suited for Fog based environments.

2.4 Summary

This chapter has detailed background and related work around scalable network monitoring, NFV, and computing towards the edge. Several pieces of work have been identified on their relevance towards to the aim of this thesis. With the objective to design and verify a scalable and responsive monitoring system in mind, the current technologies of SDN NFV and Orchestration currently lack the features to fully achieve this. As a result further research in edge orchestration, scalablity, and monitoring is required before this thesis can be realised. This said, best practices and failures of past research can help in directing this work.

The technologies and research gaps have been highlighted that are required to realise scalable and monitoring responsive monitoring for the Cloud-to-Fog continuum. It is clear that work is required on advancing NFV service overlays, edge VNF orchestration, SDN monitoring methodology, and connections between these technologies. Specifically, background and related work have detailed:

- The potential benefits of SDN, NFV, and Fog Computing for network monitoring and remediation
- The lack of SDN monitoring frameworks that focus on scalability and adaptability whilst still retaining monitoring capability
- The lack of standardised and usable solutions for traffic routing between virtualisation and network infrastructures
- The limited research on the potential for scalablity and function of a framework that considers monitoring across the network

Finally, the findings from this chapter including existing tools, standards, and observed successful practicalities from related research are considered in

the design and implementation of both the SIREN and TENNISON frameworks described in this thesis.

Chapter 3

Designing Responsive and Scalable Network Monitoring

This chapter describes a design for a responsive and scalable network monitoring systems. Initially, Section 3.1 goes into detail on why a new design is required, highlighting gaps in current systems, as well as advantages of emerging technologies. Based on the motivations discussed, Sections 3.2 and 3.3 go into detail on the design requirements as well as design considerations, covering topics of SDN scalability, monitoring, and orchestration methodology.

In summary, this chapter details the motivations, requirements, design considerations, and architecture for components of a novel SDN network monitoring system: Tennison, as well as a Cloud-to-Fog orchestration system: Siren.

3.1 Motivation

There is a growing trend towards highly configurable networks and services. Achieved through the movement of packet processing functionality into software [118], this progress enables network providers to dynamically adapt their provision in response to varying demand. Thus far, focus has been primarily on data centre and cloud environments. More recently, interest has shifted towards access and last-mile networks, encapsulating telephone exchanges, homes and

business environments in the process [154, 217, 25]. This complete spectrum of networks and devices is called the Fog [25]. When combined with Network Functions Virtualisation (NFV), the Fog presents a number of interesting opportunities for network operators: services can now be pushed even closer to the edge of networks, and in some cases, only a single hop away from the target devices. In conjunction with existing capabilities, this new enhancement allows for greater performance and efficiency, regardless of the type of service to be deployed. The potential benefits are best described through use of an example. Content Delivery Networks (CDNs) store and host replica copies of content, primarily multimedia. These are then used to serve consumer requests, rather than retrieving the content from the origin server. In current networks, these content replicas are located in centralised strategic locations, such as Internet Exchange Points (IXPs). Yet there are still a number of network hops to reach the final destination, which may ultimately be located in the users home or place of work. CDNs exist primarily to improve the experience given to the user. Despite current deployment strategies, these final hops can nonetheless have an impact on the service delivered. In a Fog scenario, it would be possible to locate and serve content within the last-mile network, or even within the users local network. This reduces the chance of network impairment impacting user experience.

3.1.1 The Cloud-to-Fog Continuum

The Cloud-to-Fog continuum as shown in Figure 3.1 describes an Internet compute architecture where compute resource for NFV is available all the way from the datacenter to the residence and enterprise locations. Current NFV systems [72, 135, 48] as well as the ETSI NFV specification [70] rely on the use of Cloud for the infrastructure. Whilst there are advantages to running NFV in the Cloud such as scale and ease of management, the network edge also provides

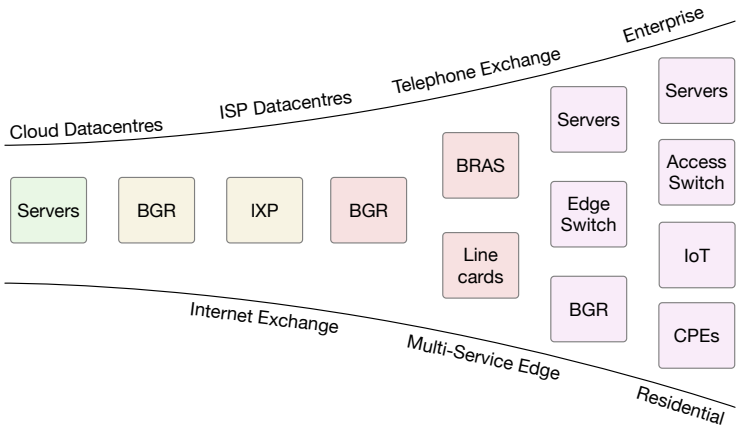


FIGURE 3.1: Cloud to Fog Continuum

advantages due to proximity as detailed in Section 2.2. As highlighted by the Fog specification [50] from the Open Fog Consortium [88] the Fog is a fertile ground for NFV deployment.

This thesis claims that by considering the spectrum of environments capable of hosting compute resources, benefits can be reaped from each such that deployment and placement of virtualised network services can be done efficiently [75, 74].

3.1.1.1 Analysis of SDN/NFV Performance in Edge Networks

Key to enabling deployment across the Cloud-to-Fog continuum is low power, cheap, small form factor, and reliable resources that can be distributed across the network. The following section analyses how such devices have improved over the last 25 years.

Along with the increased interest in deploying to the edge, is the increase in resources at reduced cost. Figures 3.2 and 3.3 show an analysis of 3,870 CPEs sold between 1998–2019 demonstrates the upward trend in device capabilities, such that they can now reasonably be considered as compute hosts as a Fog-NFVI, the devices clearly differ vastly in their capabilities relative to those

found in Cloud datacenters. Data on the above is available on GitHub[1].

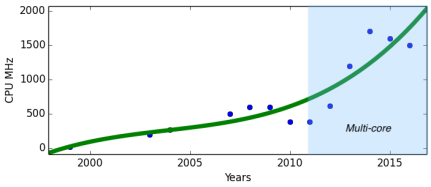


FIGURE 3.2: CPU resources

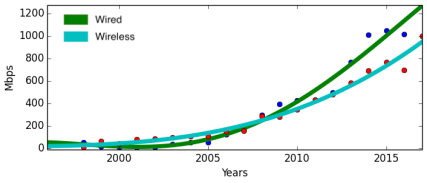


FIGURE 3.3: Network resources

Alongside the growing compute capability, ARM have been actively targeting NFV and SDN with their low power processor architecture [13]. They are approaching NFV with a micro-service based architecture, knowing that network services are split up into multiple separate components, they are producing high processor count chipsets, which can cleanly delegate resources to each service component [13].

3.1.1.2 Experimentation Environment

Demonstrating the use of considering the context in the Fog, Figure 3.4 shows the simple experiment topology that is representing two home networks that share an aggregate switch to the Internet. This case study compares the cost to the network in three scenarios: 1) a vCache VNF being deployed using a first fit clustering policy 2) a vCache VNF being deployed using an context-optimised policy and 3) using no CDN and requesting video directly from content provider source. The vCache is deployed to the NFVI (Raspberry Pi) as a Docker container. The video in use for the case study is a 1080p version of the Big Buck Bunny standard testing video and is 276.1MB in size. During each experiment iteration, three clients are watching the video and traffic is being monitored and recorded on the aggregate switch to evaluate the difference between deployment

[1] https://github.com/lyndon160/cpe-scraper

techniques. In the in second test, where optimised placement is used, the content provider requests information about latency between the service customers and the available NFVIs to determine the closest one to deploy to.

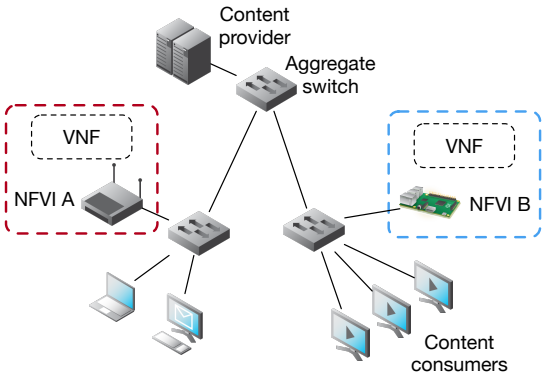


FIGURE 3.4: Experimentation Topology

3.1.1.3 Analysis of Fog Placement

Figure 3.5 shows the results of three VNF placement techniques. The stacked bar chart shows live data, which is data that was pulled over the aggregate switch whilst clients were watching the video, and pre-pushed data, which is the cost of pushing the vCache VNF.

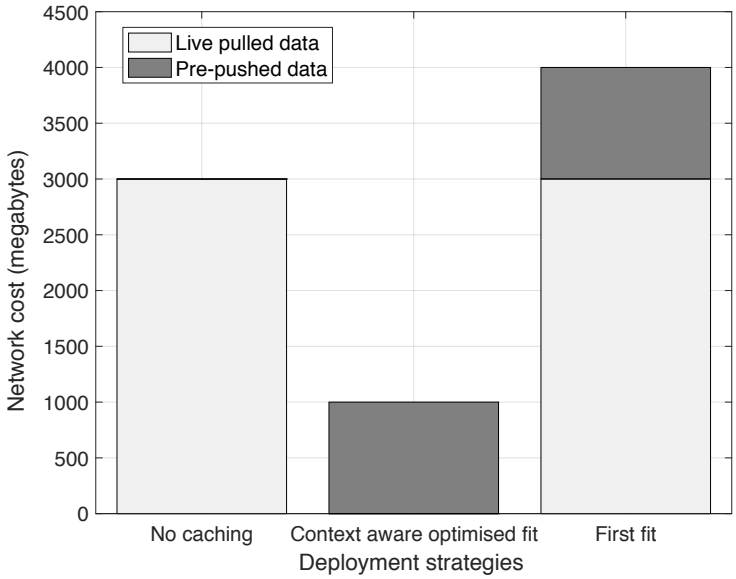


FIGURE 3.5: Fog Placement Cost Reduction

In this instance, the contextual information has made it so that the VNF can be placed much closer to the client on *NFVI B*, thus reducing observed traffic on the aggregate switch by two thirds when compared to no caching. More importantly, a suboptimal placement policy such as first fit which in this instance places the VNF on *NFVI A* can cost more to the network when compared with no caching, this is due to the VNF being pushed whilst the content being requested over the aggregate switch.

This concurs with similar work in the area of edge optimisation; In a mobile messaging use case, it has been shown that pushing services to the edge can reduce traffic up to 50% [69].

3.1.1.4 Limitations of Scalability and Distribution within Contemporary Solutions and Technologies

In order to realise an SDN monitoring framework that can be deployed from the Cloud-to-Fog, aspects of scalablity and distribution need to be addressed within SDN and NFV. The following reviews different aspects required for such a framework and their components that do not match up to the requirements.

SDN Controller Scale: ONOS has limitations around scale [76, 137]. As shown in ONOS's own performance tests [137], over version releases, the controller is improving but it's clear that its scalability is still limited. To add to this, SDN monitoring requires additional switch to controller messaging, which further impacts scalablity.

VNF Forwarding: In order to redirect traffic from the network to and NFV infrastructure a forwarding methodology outside of standard routing is required. In current cloud based NFV solution this is solved with a network choke-point that redirects traffic in and out of the NFV cluster. The complexity of this increases when deploying across to distributed NFVIs where there is no single

point in the network where all traffic traverses. At the time of writing, there is no widely integrated VNF forwarding graph technology. The closest to this is the Network Service Header (NSH), which serves this purpose but is currently does not have a production grade implementation.

NFV Deployment to Heterogeneous Environments: NFV deployments thus far are not concerned with specific deployment details. In order to deploy from the Cloud-to-Fog continuum an orchestrator needs to be aware of the network topology and the infrastructure topology. Cloud management platforms are also not concerned with specific placement of VMs/Containers. The closest is Kubernetes which allows each machine to have a label which can be used to determine its location. Solutions like this go against the ethos of this tool, where resources are homogeneous and containers can be dropped on any machine regardless of its context.

3.1.2 Summary

This section has motivated the need to use edge computing to increase the scale of NFV deployments, specifically when considering network monitoring. Furthermore, this section has highlighted multiple research challenges that need to be addressed before a scalable monitoring system operating across the Cloud-to-Fog continuum can be realised.

The following summarises the challenges that need to be addressed in order to fulfil the goals of this thesis as detailed in Section 1.4. Improving scale of SDN controllers with controller distribution and tiering. Designing and prototyping a forwarding technology capable of dynamically routing traffic to distributed NFVIs. Finally, this section has motivated the need to extend the Cloud to the network edge, thus considering heterogeneous environments in NFV orchestration is required.

In summary, there needs to be an architecture that includes SDN and NFV that is capable of integrating network monitoring across the network rather than a single point in the Cloud.

3.2 High Level Design Requirements

Elicited from background research and current research challenges in network monitoring as highlighted in Chapters 1 and 2, the following details the high level design requirements for a responsive and scalable network monitoring system.

- A wide a range of inputs should be supported such that the system can best understand the state of the network.
- Place monitoring based network functions towards the edge of the network to optimise use of the network.
- Monitoring that is conscious of network capacity and is capable of proportionally adapting to the context.
- Extenisbility so that additions such as an operator interface and new detection algorithms can be integrated with ease.
- Monitoring should be able to adapt to network capacity such that the network controller, data plane, and monitoring system are not overloaded.

The design requirements above provide an overview of the directions of research required to successfully design and implement a scalable network monitoring system.

3.3 Design Considerations

This section details various design considerations for monitoring and orchestration in the Cloud-to-Fog continuum. Specifically, this section compares the available technologies and solutions to monitoring agility and control, monitoring, deployment flexibility, network service orchestration, virtualisation, and technology agnostic architecture. These considerations are based on best practices in modern software design as well as empirical evidence from an edge compute analysis in Section 3.1.

3.3.1 Monitoring Agility and Control

As motivated in Section 3.1, an SDN controller is required for both monitoring and forwarding traffic to distributed NFV Infrastructures. The following highlights the options for SDN controllers as well as methods for NFV connectivity.

3.3.1.1 SDN Controllers

In terms of SDN controllers, there are a variety of options, each with varying support for industry grade requirements. As for scalablility and resiliency, ONOS, the industry grade controller offers controller distribution. At the time of design and implementation, ONOS was the only industry grade SDN controller that supported distribution. Since then there is similar capability from ODL [134]. Alternatively there is Ryu, which does not offer distribution or failover but instead has simple design that it enables it to focus on stability and supporting the latest versions of the OpenFlow protocol [166].

3.3.1.2 NFV Connectivity

As discussed in Chapter 2, NFV can have various benefits to monitoring, especially towards the edge of the network. When applying NFV towards the edge

of the network, additional considerations need to be made as to where and how traffic is redirected. Various other SDN monitoring frameworks [186, 180, 221] achieve NFV by the use of middle-boxes or simple port mirroring. Neither of these solutions are agile and require operator intervention to install and adapt.

Network Service Header (NSH) The Network Service Header [161] contains information which defines the position of a packet in the service path, using a service path and path index identifiers, and carry metadata between service functions regarding policy and post-service delivery. The NSH Request For Comments (RFC) is specifically made to solve the challenge of NFV connectivity, however, as of 2019, NSH is only supported in software by a modified version of OvS.

Source Routing (SR) with IPv6 The use SR for providing NFV connectivity is arguably the most elegant solution currently available, however it requires IPv6; many of the SDN controllers, OpenFlow and P4 implementations have varying levels of IPv6 support [140, 39].

Vendor locked tunneling This is achieved by using SNMP, REST, or propriety interfaces to create GRE tunnels. This solution quickly and easily rectifies the issue but is not future proof and also locks the framework into a single vendor.

VLAN based tunneling VLANs with SDN can be used to create multiple overlay networks dynamically. However, this solution is imperfect as it requires the use of double tagged VLANs if VLANs are already which are not supported by a the majority of enterprise or customer networks. On top of this, if the traffic has to traverse a non-SDN network, the traffic will not be routed correctly,

whereas with SR or encapsulation, traffic will be routed correctly over non-SDN networks.

Middle-boxes and encapsulation Middle-boxes are placed between networking devices to provide additional functionality within the network. For the use of NFV connectivity, middle-boxes can be used to create tunnels from the network to an NFV Infrastructure. The added advantage of tunneling traffic is that the data-plane traffic is encapsulated, keeping all of its original headers, and encrypting traffic if required.

3.3.2 Monitoring Methodology

Conventional network monitoring uses a variety of methods to monitor traffic. This includes sampled header monitoring, middleboxes, and mirroring. With a deployed SDN network, monitoring is similar to conventional methods but can be done dynamically and without additional hardware.

The requirement for monitoring granularity depends on the resources available and the types of attack that need to be remediated. Some attacks require deep packet inspection whereas others can be detected via flow header monitoring, or even aggregated summaries of traffic flows.

The following lists the different methods of monitoring, specifically detailing their trade-offs.

Redirection: This change moves the packet to another destination, which adds network latency and packet processing latency. This is useful when analysis is required to be done remotely and that monitoring needs to be done in real time. A solution like this would only be applied to a small subset of the network traffic.

Mirroring: The duplication of traffic, forwarding the packet normally and then cloning the packet off to another location on the network. This style of monitoring analyses traffic after an event has occurred, and thus impacts the latency of attack detection. If duplicated traffic is managed correctly, there is no latency or bandwidth impact to the source traffic.

Middlebox: Offers immensely detailed monitoring by performing full packet inspection, typically to all network traffic that traverses the box. The primary downside to the solution is the impact of additional latency to all traffic, difficulty of processing large volumes of traffic, limited network visibility, and expense to deploy and manage.

Full OpenFlow Packet-In: This solution offers the most detailed monitoring conveniently in one location across the entire SDN network but also offers the worst performance and is not scalable to network size or traffic load [101]. This method also delays the packet significantly, increasing the packet's latency by over 20ms and much more when under load [139].

Header monitoring: A lighter weight monitoring solution, that when managed correctly adds little impact to the network, but offers limited monitoring detail. At the cost of monitoring granularity, sampling of header monitoring can be utilised for an even lighter weight solution.

3.3.3 Deployment Flexibility

Deployment flexibility is important ensuring that the system can be used in multiple environments under varied conditions. Architecturally, there are multiple options to deploying a monitoring system, including: tiered, distributed, and centralised deployment. In this case, a completely decentralised solution, such as peer to peer or super peer are not used because of concerns around

performance, network visibility, incompatibility with existing SDN controller architectures, and security.

Centralised Deploying with a centralised architecture offers simplicity, reduced costs, and with a master slave configuration, resiliency. However, with there being only one instance in operation at anyone time, both scalability and availability are compromised. When compared to other SDN network monitoring solutions such as [221, 112, 181, 213, 114], they favour a centralised approach, in some this is due to a lack of mature distributed controllers available during their development phase, and in others because of the simplicity offered by a centralised approach making prototyping much easier than alternative methods

Distributed deployments offer increased resiliency, availability, and depending on the level of shared state, they can also improve the system's scalability. Whilst distribution increases system complexity, if the distributed substrate is abstracted and robust automation is used with deployment, complexity can be reduced for developers working on top of the system.

Tiered deployments offer a range of benefits at the cost of deployment, system, and management complexity. Where scalability, resiliency, and a separation of concerns is a primary requirement, a tiered approach is best suited. In the interest of future compatibility, tiered architectures are considered under NFV standardisation, with ETSI's NFV architecture [72].

3.3.4 Network Service Orchestration Methodology

At the time of writing, the open source implementations of MANOs, as highlighted in section 2.3.2 have limited support for orchestration, and typically rely on a variant of a first fit policy. The reason behind the use of this single policy

is around what the Virtualisation Infrastructure Manager (VIM) defaults to, for example, in typical Cloud OpenStack deployments, first fit deployment is well suited. On top of this, as explained in Chapter 2, they only support Cloud based infrastructures where placement resources are relatively simple and well understood challenges when compared to the Fog. With Network Functions Virtualisation and Fog computing, network service orchestration is more complex than conventional NFV, which deploys to the Cloud. The number of locations in the Cloud-to-Fog give room for further optimisation in orchestration. Orchestration is required to place network services in the best location within the network depending on an optimisation factor.

For this thesis, three orchestration methods are considered for different contexts. The primary orchestration for network monitoring is a distance and context aware based orchestration. In scenarios where the network is shared and a range of network services need to be considered, an auction based orchestrator can be used.

Cost-based Orchestration Important to motivating the use of edge networks, this method of orchestration considers the economic cost of deploying network services at different locations depending on the cost of network transfer. This motivates the decision to place services that require high data rates to be placed closer to the network target or client where the service is served.

Service Agnostic Auction-based Orchestration This method of orchestration provides a platform for multiple customers to rent out infrastructure for period of time, thus naturally supporting multi-tenancy. This has the benefit of offloading the orchestration logic to one or more third parties.

Monitoring Orchestration Facilitating network monitoring, this method of orchestration is a distance and context aware technique that considers the distance between the network target and the service and also considers the available capacity on each link to place a network service. This method of orchestration also has the ability to automatically scale services and move services depending on demand.

3.3.5 Virtualisation Technology

Virtualisation is used to execute multiple services on a singe machine, typically offering a form of resource management and isolation to the service. The deployment of network services to the Cloud-to-Fog continuum raises many challenges, including the use of an appropriate virtualisation technology. As stated in Chapter 2, the Cloud-to-Fog is a highly heterogeneous environment, including multiple computing architectures as well as varied amounts of compute, memory, and disk.

Virtual Machines (VMs) Offer full operating systems were code does not require to be recompiled. From an ease of use point of view, these are VMs simple to adopt and have been used for over a decade. Previously VMs were the primary choice for any virtualised software.

Containers are analogous with processes with increased isolation. They share the host's kernel but have their own libraries and network interfaces. The primary benefits of containers include: same compilation process, significantly reduced size and memory usage, short initialisation (<1s) [79], and isolation. On the other hand, containers provide a weaker level of isolation than other solutions, and due to the additional network interfaces, can have reduced network performance. Since the wide adoption of containers, they are a popular choice

for micro-service architectures and are now supported by a variety of tools for service orchestration [107, 162, 12, 8, 86].

UniKernels These are small kernels that are compiled in to single purpose applications. Only the libraries required for the application are on the UniKernel, making the resulting OS compact and efficient. At the time of writing, this technology is relatively immature and producing efficient VNFs or applications is poses significant challenges [209], especially when deploying to non-x86 architectures, such as ARM due to the lack of support at this time. In summary, the pros of UniKernels is that they offer strong isolation, reduced overheads, and small base image size.

3.3.6 Technology Agnostic Architecture

To improve the system's robustness and longevity, the fundamental design of the system should be able to resist obsolescence as well take on new best practices. Robustness and longevity can be ensured by using a componentised and loosely-coupled design where parts of the system can be easily segmented and replaced if needed. The trade off here is potentially a performance hit and also increased time to develop. However, as this space is moving quickly, future proofing and being technology agnostic is important to the longevity of the design. The following details the various areas that are prone to change in the near future.

Southbound Protocol: The southbound protocol in an SDN refers to the protocol between the SDN controller and SDN compatible switch [121]. OpenFlow is the current forerunner of the southbound protocol and data plane implementation for SDN technology. This said, P4, the data plane implementation has recently gained traction offering various benefits over OpenFlow. The data

plane design should be designed to be technology agnostic such that it can reap the benefits of new protocols.

SDN Controller: Over the years there have been many SDN controllers, each with their own unique aspects. The most stable and mature controllers at the moment are: ONOS, ODL, and RYU. Whilst there is uncertainty about which controller will continue to have support, the design of the system should be agnostic of the controller, such that it can be changed with ease in the future.

Network Hardware: With the emergence in P4, new hardware might be implemented with new benefits. This is also true for OpenFlow, for example, in 2018 Corsa released an OpenFlow switch which was combined with large internal compute resource and additions to the protocol

The system should be broken up into subsystems and sub-subsystems so that components like message queues, interfaces, and databases can be replaced if needed.

3.4 Design Overview

With the design considerations noted previously, Figure 3.6 shows a high level overview of the different components of the system. At the overview level of the design, the system is split up into five isolated pieces: the monitoring and security framework, the network controller, the orchestrator, the NFVIs, and the forwarding devices. This general design follows a loosely coupled architecture so that each component can be updated individually and can be used on their own. Other implementations of similar systems such as [181, 112, 221] have merged these components together into one monolithic block.

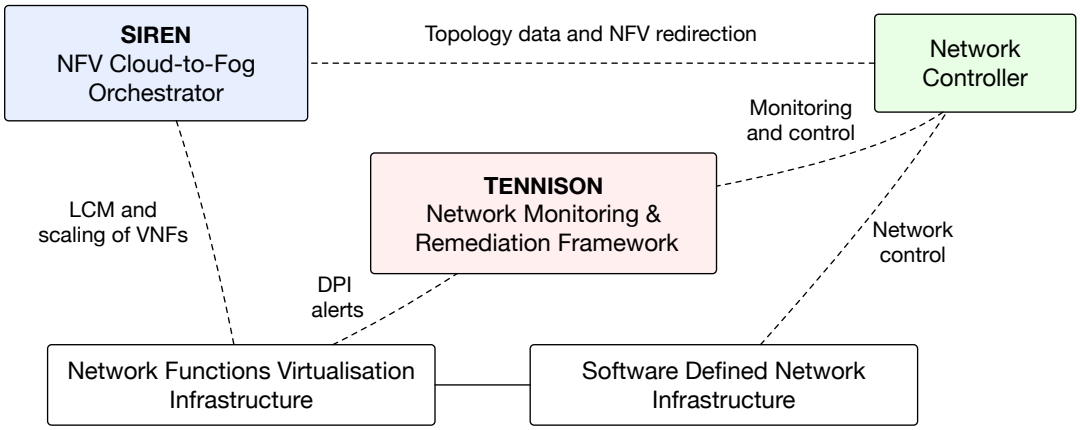


FIGURE 3.6: Grand Architecture Overview

The overall design as illustrated in Figure 3.6 shows SIREN influencing the network controller and managing the Network Functions Virtualisation Infrastructure. Moreover, the network infrastructure is managed by the network controller, which reports monitoring information and exposes control to TENNISON. TENNISON also receives monitoring alerts from deployed network functions from the NFV Infrastructure. In summary, the interaction between these three loosely coupled components and two infrastructures provide a scalable and responsive network monitoring system.

3.5 TENNISON: Monitoring and Remediation Framework

TENNISON is a framework that focuses on providing scalable monitoring and remediation. This section describes the TENNISON system architecture highlighting the individual components and features of the design.

The overall system architecture for TENNISON, in Figure 3.7, is formed of three distinct architectural layers. The lower layers (Bro and Snort DPIs, ONOS Controllers, and sFlowRT) are the appliances and instances deployed within the network, cumulatively referred to as the *Collection* layer. These instances are

fundamental to the operation of TENNISON, in that they provide both control and monitoring functionality to the higher layers.

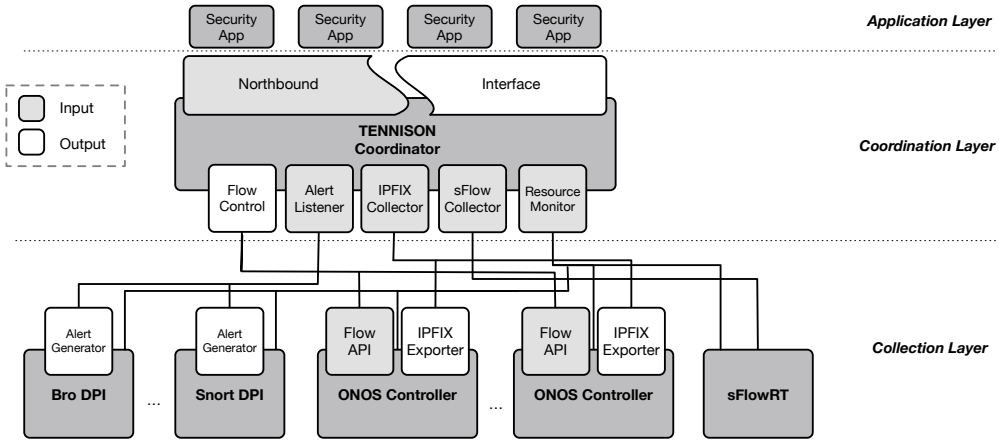


FIGURE 3.7: TENNISON System architecture

In some cases, minor extensions have been made to the Collection interfaces to enable them to report and communicate with the TENNISON *Coordinator*, which forms the *Coordination* layer of the architecture, and is responsible for storing and aggregating all of the information generated by these interfaces. The interfaces between these interfaces and the Coordinator represent a flow of information, with the interfaces providing input or output (see legend in Figure 3.7). From this illustration, it is evident that both the packet inspection tools and the flow monitoring applications produce input while the ONOS controller alone offers an output. It is this output that enables the coordinator to modify the forwarding plane of the network, via the ONOS controller, and provide the functionality and programmability required for TENNISON to operate effectively.

The *Coordinator* is responsible for coordinating the overall operation of the architecture, and acts as an intermediary between the upper and lower layers; Application and Collection layers, respectively. The *Coordinator* is intentionally built independent of any other component, and is completely technology

agnostic. This allows interoperability with alternative technologies and flexibility in terms of design and placement in the network. Furthermore, every aspect is componentised and networked and only holds state if required. As a result, the system is easily distributable (i.e. multiple TENNISON instances) as only the database is the primary aspect which holds state, and as such, TENNISON is able to scale to the size of network.

The central role of the coordinator grants an authoritative view of the network topology and its current state with the ability to simultaneously modify both the network *and* the monitoring services running within it.

The final, and uppermost layer, is the *Application* layer, which hosts the security applications. These applications interact with the coordinator leveraging its functionality to realise custom and dynamic security behaviours. This enables the creation of applications tailored to specific threats, or integration with existing tools already deployed in a network.

The remainder of this section contains a detailed breakdown of each of the components, including contextualising them in the overall system architecture.

3.5.1 TENNISON Coordinator

The TENNISON Coordinator is central to the TENNISON architecture, and is responsible for the overall operation of the system. In this section, the subsystems within the Coordinator that together provide the rich functionality to security applications in the Application layer (see Figure 3.8) are detailed.

We start by describing a series of Southbound Interface (SBI) modules that communicate with appliances within the collection layer. These modules provide a rich set of information to the coordinator, allowing it to make a wide range of powerful, yet informed, decisions with respect to network monitoring and attack remediation.

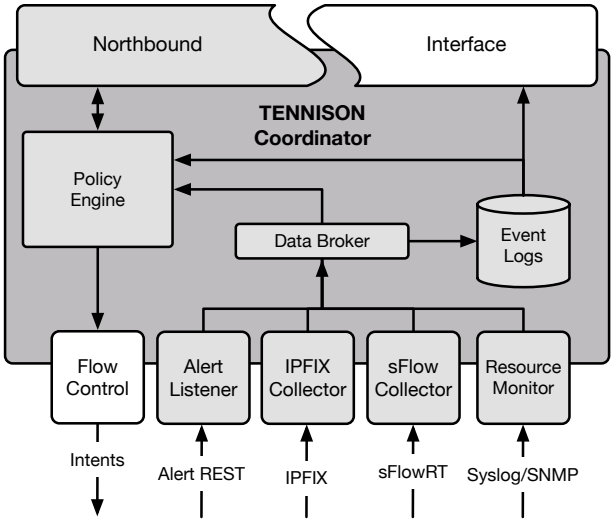


FIGURE 3.8: TENNISON Coordinator subsystems including southbound interface

3.5.1.1 Southbound Interface (SBI) Modules

The first is the *Flow Control* interface, which is an *output* from the Coordinator. The Coordinator uses this to control the flow of traffic through a network, directing and shaping this towards the various appliances under its control. Importantly, the Coordinator does this in an intent driven way; it describes the changes at a high level without specifying precisely how this should be achieved. For instance, the Coordinator does not define the exact switch on which a flow should be modified. It is the responsibility of the network controller to decide the optimal placement of this given its awareness of the topology. More details of the interaction with the network controller are provided in Section 4.1.2.

The remaining four SBI modules all provide *inputs* to the Coordinator. The *Alert Listener* is a REST interface used to collect messages generated from the various packet inspection appliances located in the network. An alert message is generated by a DPI tool when a suspicious flow or packet flows through the appliance. This alert is then passed from the appliance to the coordinator via this interface module. A similar process is followed for the *IPFIX Collector*, *sFlow Collector* and *Resource Monitor* interfaces. The difference between them

being the type and content of the message received on each interface. In the case of the *IPFIX Collector*, aggregate flow records are received from the network controller. Similarly, the *sFlow Collector* provides sampled flow record statistics when high volume monitoring is used in the network. Finally, the *Resource Monitor* collects resource usage information (e.g. cpu/memory/flow table occupancy) from the switches and controllers in the network.

This resource information is used for dynamic decision-making in the TENNISON monitoring and security system. For example, placement of the monitoring to avoid potential switch flow table overflow and avoid controller processing overload. Together, these four interfaces provide a holistic view of the entire network, including traffic levels, threat analysis and resource utilisation.

3.5.1.2 Data Broker

Regardless of the message type, once a message is received at the Coordinator, its content is passed to the *Data Broker*. This acts as an intermediary to determine the destination of the message within the system. There are two possible destinations; the *Policy Engine* and/or the *Event Logger*. Importantly, the *Data Broker* enables extensibility of the southbound interface. It does this by providing a generic interface to which new collectors can connect. Furthermore, the *Data Broker* queues messages, acting as a virtual buffer between incoming messages and the policy engine.

3.5.1.3 Event Logger

The *Event Logger* is a long-term storage medium, realised with a key-value store, that keeps each message in its entirety. This database can then be searched by other components to retrieve historical information (through the Northbound API or Policy Engine, as illustrated in Figure 3.8). It provides persistent storage for the coordinator, enabling rapid recovery in cases of failure or migration.

The event logger can also be configured in such a way that stored messages are automatically expunged after a fixed period. This allows the storage footprint to remain consistent without the need for explicit maintenance.

3.5.1.4 Policy Engine

The *Policy Engine* is an integral part of the coordinator and contains the logic through which new messages are processed (originating from one of the input interfaces), historical trends are analysed (through the Event Logger) and resulting actions are taken (through the Flow Control). The policy engine has a permanent storage state to record security policies and monitoring decisions. It is also fully configurable by security applications, to add, remove or modify logic, as required. Examples of the behaviour of this policy engine under various attack scenarios, are described in Section 4.1.3.

3.5.1.5 Northbound Interface

The *Northbound Interface* is key to enabling the programmability and extensibility inherent in Tennison. For clarity, the upper components of the coordinator are illustrated separately in Figure 3.9. The northbound interfaces enable the security applications to both read the current network state and to modify it according to custom internal logic. These interfaces are clearly defined such that applications from different sources (and with different objectives) can be used in parallel.

The security applications connecting to the northbound interface first register with the controller by providing a unique ID and a set of credentials. This enables the coordinator to identify the application, and to authorise it with the appropriate permissions. This includes permission to read the network topology, and permission to interact with neighbouring applications, as required. Tennison hosts two application types; *managed* and *unmanaged*. A

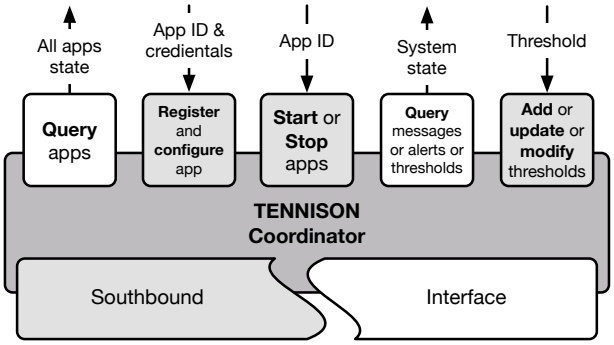


FIGURE 3.9: TENNISON Coordinator Northbound Interface

managed application can be controlled by another security application, whereas an unmanaged application cannot. This can be used to create a hierarchy of applications, to share state and behaviour between applications, and to remove potential conflicts in terms of network behaviour. To support *managed* applications, the Coordinator provides the necessary northbound API calls to query existing applications, including their availability, uptime and current state. Using this information, a neighbour application (if authorised to do so) can also start, stop or configure other managed applications. Any application, regardless of its management status, can query the coordinator to search for messages (e.g. IPFIX, sFlow, DPI alert, etc.) received by the Coordinator (including current and historical information). Similarly, using this same interface, the application can query the current logic of the Policy Engine. This enables the requesting application to understand the current behaviour of the architecture, aiding in avoiding potential conflicts. Finally, the northbound interface enables the security applications (regardless of their management status) to add, remove and/or modify the rules within the Policy Engine. From a system security perspective, this capability is closely controlled by the application authorisation, with relatively fine-grained permission supported in TENNISON.

3.5.2 TENNISON Multi-level Monitoring

TENNISON operates a tiered system of monitoring to provide scalable network security. The multiple levels of monitoring are illustrated in Figure 3.10, with light-weight monitoring for a high volume of flows at Level 1 (L1) and Level 2 (L2) leading to detailed monitoring for a reduced flow count at Level (L3).

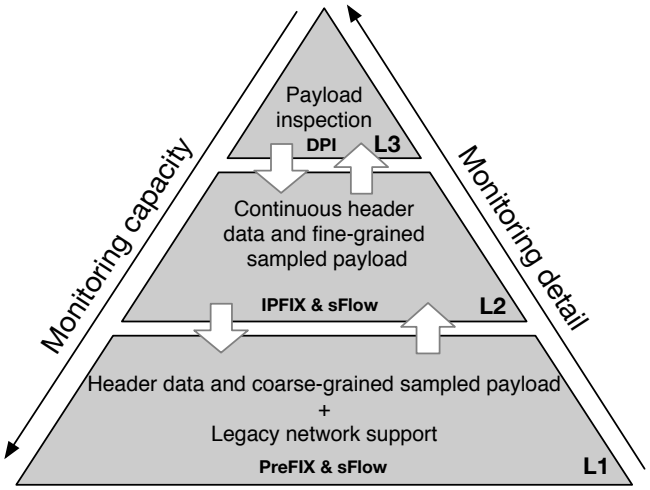


FIGURE 3.10: TENNISON multi-level monitoring triangle

"PreFIX" identified in Figure 3.10 is part of the first level of monitoring (L1) and, by default, provides network layer four header information for the *first* packet in the flow for *every* flow. At both L1 and L2, sFlow provides latent, always-on sampled monitoring. Furthermore, as sFlow can be configured on non-SDN switches, it provides TENNISON with visibility of legacy networks. sFlow captures flow information based on sampling. The sFlow agent in the network element is configured to export sFlow records to sFlow-RT, which then reports alerts to the TENNISON Coordinator for remediation and to support multi-stage attack detection. The sampling rate, polling rate, and packet header length are configurable and can be dynamically updated based on the network state and immediate monitoring requirements.

In addition, at L2, IPFIX data input to TENNISON based on defined OpenFlow monitoring intents provides a more fine-grained and continuous monitoring capability suitable for detection of attacks that can evade a sampling-based

monitoring approach.

Finally, L3 represents TENNISON's capability to forward suspicious traffic towards DPIs for classification. This leverages TENNISON tunneling, which provides the means to forward and mirror specific traffic from any host on the network to any destination without modifying the packet. This is a further example of the scalability of the system. As the network increases in size and DPI processing throughput reduces, additional DPI instances and tunnels can be instantiated on-the-fly ensuring optimal network protection provision. As identified in Section 3.3.2, there are various monitoring methodologies available for use, each with their own trade-offs. These methods include: Redirection, mirroring, middleboxes, OpenFlow Packet-Ins, and header monitoring.

Adaptive behaviour allows for various methods of monitoring to be applied depending on traffic classification and network context. Traffic classification can be used identify the severity and profile of the attack and adapt network monitoring behaviour accordingly. Network context can benefit adaptation in understanding the amount of available resources.

3.5.3 SDN Controller Distribution

Controller distribution adds availability to the network whilst improving resiliency by taking away the single point of failure. However, with this there is additional complexity and difficulty of adapting the controller to new paradigms. On top of this, in cases where state is continuously shared by data rich controller applications, performance can become degraded due to the increased overhead from state sharing. For the goal of scalability as a part of this thesis, the implementation of controller distribution and correct configuration and usage is paramount to reaping its benefits effectively.

In terms of controllers, the TENNISON and SIREN designs are agnostic of

the network controller. In the case of SDN controller distribution with TENNISON, in the design, all unnecessary SDN applications are stripped from the chosen SDN controller. On top of this, the SDN controller TENNISON applications themselves require no state transfer, instead relying on the controller's distributed function for operations that operate over multiple controller domains.

3.5.4 Tiered Network Monitoring

As a part of TENNISON's flexible deployment options, this section details an alternative architecture that focuses on providing greater scale and a separation of concerns.

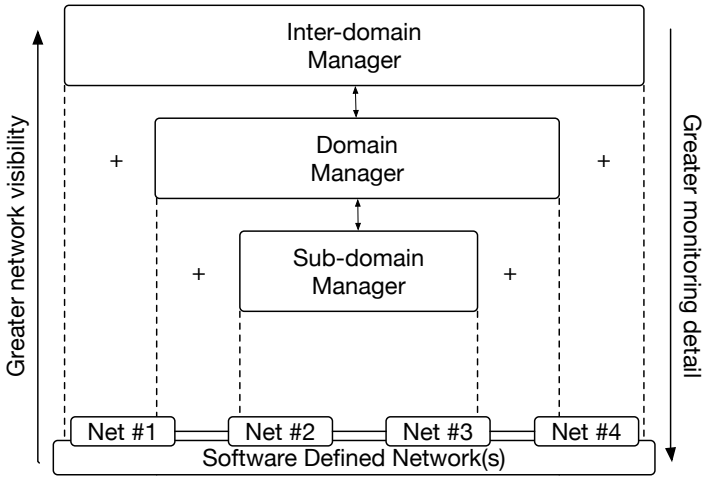


FIGURE 3.11: Tired architecture design

As described in Section 3.3 and motivated in Chapter 3.1, multiple approaches to scalability should be included within the design.

Background research detailed in Chapter 2 found that other SDN monitoring frameworks including [14, 112, 221, 181] do not address scalability and instead focus on other challenges within SDN monitoring. Based on performance from published by ONOS [137], a full state sharing distributed architecture has its limitations for scalability. To incorporate increased scale as well as a separation

of concerns, this section describes a tiered approach. As shown in Figure 3.11, each tier in this architecture has a wider reach but has less information about the network.

The primary reasons for this architecture over others are:

- The tiered architecture provides a centralised control and a wider view of the network than other architectures.
- Results on scaling SDN [137], as well as various academic publications [184, 220, 101, 72, 193] and ETSI's specifications [72], suggest that both a tiered or east-west architecture is a direction that will be supported by SDN and NFV frameworks as the technologies mature.
- Distributed ONOS can still be used with a tiered system, providing resilience and increased scale at multiple levels.

In this architecture, it is envisaged that the network is split up into multiple subdomains, each of which is controlled by an ONOS cluster, and in turn a monitoring framework.

Subdomain Manager The Subdomain Manager is largely the same to the standard implementation of the monitoring framework, but with another interface which is connected to the domain manager, sharing detailed information on alerts and events. The bottom level of the monitoring framework in Figure 3.11 also has a fail over node in the design for redundancy and will also be able to operate autonomously if required, such that if the link between the subdomain and the domain manager was to fail, security and monitoring would continue. This will simply work by fully replicating the underlying database that holds both the events and the policy engine. The addition, to the subdomain controller includes a new northbound application, the Tier Manager. This

application shares data with the meta monitoring framework, in this case that is the Intra-domain coordinator.

Domain Manager The Domain Manger is responsible managing fine-grained alerts and events between the framework's interfaces. It is also the component that configures Subdomain Managers, pushing down local domain polices. In an enterprise scenario, the domain manager would be under the control of the organisation that was using it, allowing them to disseminate their own network monitoring policy across multiple subdomains.

Inter Domain Manager The Inter-Domain Manager is responsible for managing coarse-grain information between domains. The Inter Domain Manager will also be the insertion point for updates on new security vulnerabilities and detection methods, such that new identified attacks could be mitigated. Where Tennison is provided as a service, the inter-domain manager would be maintained and operated by the providers of the service, allowing for remote security network patches and visibility of multiple networks at once.

3.6 Siren: Infrastructure Management and Orchestration Platform

In this section, the design for Siren, the Infrastructure Management and Orchestration Platform is detailed. Siren is an infrastructure provisioning, management, and orchestration framework. Siren addresses a gap in current MANOs where heterogeneous environments are not considered.

The remainder of this section goes into detail on the components and orchestration methods within the Siren architecture, illustrated in Figure 3.12.

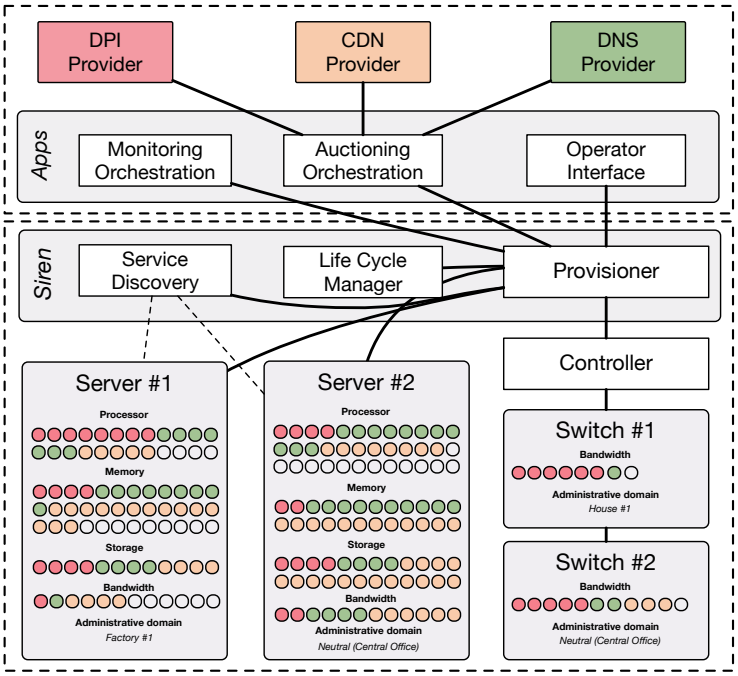


FIGURE 3.12: Cloud-to-Fog Infrastructure Management and Orchestration Platform

3.6.1 Service Discovery

The Service Discovery module is a centralised location where data about SIREN compatible NFVIs are stored. Each NFVI is reported by their respective *Agent* on boot of the infrastructure. The agent passes information about the available resources, IP address, firewall policy, and global connectivity status via an anchor. This data is then correlated with topology information from the network controller to automatically ascertain the NFVI's network location.

3.6.2 Service Provisioner

The Service Provisioner is aware of the available network resources. Its key function is to operate within the heterogeneous Fog environment, translating requests from the orchestrator to deployed VNFs on the NFVIs. SIREN supports deployment to both ARM and x86 architectures in Docker format.

3.6.3 Agents

Agents are small programs that are run on each node within the NFV Infrastructures. In the ESTI MANO architecture, an agent can be part of many elements, including the element manager and the virtualisation infrastructure manager. Agents are useful for reporting the status of their host, passing data about available resources and the state of running services. In SIREN's design, the agent is responsible for launching services, and reporting status information to both the life cycle manager and service discovery modules.

3.6.4 Life Cycle Manager (LMC)

The Life Cycle Manager keeps track of all of the running network services and checks and reports errors as well as terminating a service when its time has expired. The LMC is critical component in the design of SIREN, ensuring that the infrastructure is reset after a service has expired.

3.6.5 Orchestration Methodologies

Orchestration is used to decide which network services are placed where. SIREN supports multiple methods of orchestration. Depending on the context, a different orchestration method is used. In the context of experiments with TENNISON, a simple cost-based metric is used for placement of network services. Additionally, SIREN supports multi-tenancy through an auctioning system for choosing placement of network services.

3.6.5.1 Auctioning

For the NFV placement problem [19], to support multi-tenancy, a novel combinitoral auctioning [54] approach is considered when deploying to heterogeneous environments such as the Fog. This allows bidders to define bids containing

combinations of discrete sets of resources. In our example scenario, each of the providers is able to bid on exactly what they require for their service. Furthermore, as resources in the Fog could be located in different administrative domains, depending on the sensitivity of the service, the auction may be filtered by a resources administrative domain. For example, a bid could contain a *package* consisting of 128MB of RAM, 2 CPUs and 10GB of storage across 400 devices located specifically in local home networks. Importantly, the purpose of these bids is to reserve and allocate those resources for a fixed period of time; resources are auctioned again for other adjacent time slots. This system enables multiple providers to use the same set of resources at different times during the same day. The bidding operates in multiple phases, each bidder creates their packages in each phase. If two bidder's packages overlap, one bidder has to buy out the overlapping package. Once bidding is over, the *Auctioneer* will alert each *Service Provider* as to which of these reservations has been successful. The *Service Providers* will then supply details of the service they wish to run on their reserved infrastructure, using a supported templating language (such as Docker's *Dockerfiles*).

The next step begins the realisation of the aforementioned reservation. The *Auctioneer* informs the *Provisioner* of each successful reservation, which is then actioned on the constituent devices. This includes creating the relevant containers and services, and ensuring that they are kept within their particular resource constraints. The service is now deployed, and will remain until the reservation window expires. When this occurs, the *Provisioner* will clear the existing reservations, and make way for new services to be deployed.

3.6.5.2 Cost-based

As suggested by [27], the cost of placement of a network function is orthogonal to the cost that creates on the network. Therefore, network provider cost must

be taken into account.

Equation 5.2 in Chapter 5 shows an example of a path cost calculation for lease of a resource for a single hour. C represents the cost set by the provider for an amount of network transfer m. A network provider may also wish to cost their network on a per hop basis, where some hops might be more expensive than others. Thus the equation supports a standard fee plus a fee for the path that the service will traverse. These values are entirely configurable within the system. For the purposes of illustration, a cost has been added to the reservation of bandwidth, however, these are not necessarily representative of a true cost, as this would likely be different from one network provider to another. C_p *and* C_n represents the value in cents per mbps. The equation for the cost-based placement algorithm is detailed in Section 5.2.

3.6.5.3 Network Awareness-based

With a network controller as an abstraction to the network, contextual network information is readily available. Information about network traffic, deployed network services, and the topological distance between the two, optimisation can be applied to best deploy services to the network such that they reduce the number of hops on average per flow. This deployment can then be updated periodically to address the change in network traffic profiles. The implementation of this is detailed in Section 4.2.4.

3.7 Data Plane Pipeline Design

The data plane design is important for ensuring efficient and correct processing of packets. For monitoring, the data plane needs to collect network statistics and report them back to a central data point. On top of this, a form of mirroring and network overlay is required for sending traffic to NFVIs.

In this space, there are two main technologies, OpenFlow, which has existed since 2008, and P4 which was initially conceived in 2014.

3.7.1 OpenFlow

Network monitoring was not initially a primary objective of OpenFlow. However, since OpenFlow 1.3, various features have been added that can aid network monitoring and remediation. These include: table statistics, multiple tables, VLAN modification, and metering.

The design details for an OpenFlow monitoring and Cloud-to-Fog supported NFV data plane are illustrated in Figure 3.13. The OpenFlow monitoring pipeline separates monitoring, NFV, and forwarding functions out into four OpenFlow tables. Initially, packets enter the pipeline where their headers are monitored, packets from new flows are sent to the controller for initialisation. Whilst this has an obvious performance impact, several measures are taken to mitigate this. Firstly, known flows can be proactively initialised for header monitoring on initialisation of the controller, secondly, Packet-Ins from the pipeline will only occur once in a pipeline, meaning that forwarding applications that require reactive behaviour can share the same packet-in. Next, the packet traverses to the Dynamic ACL table, this is where packets are dropped, metered or forwarded. Subsequently, the packet enters the NFV overlay table, at this point, a new header is appended to the packet for routing on an overlay network, at the same time this packet is mirrored and finally forwarded to the forwarding table where routing takes place. In addition, asynchronously, the switch is periodically reporting information about header statistics.

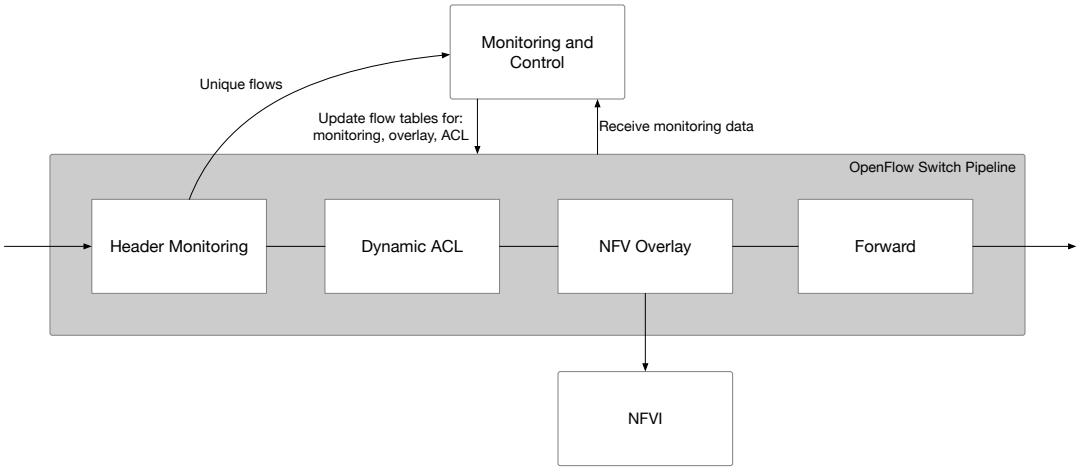


FIGURE 3.13: OpenFlow monitoring pipeline design

3.7.2 P4

P4 (Programming Protocol-Independent Packet Processors) is a data plane protocol that allows one to configure how packets are processed on a packet forwarding device. Since 2014, P4 has matured, and is approaching a point where network monitoring is feasible. This section describes what a P4 pipeline for monitoring will look like. Later on are some results on the calculated performance improvements over an OpenFlow implementation for monitoring. As TENNISON uses ONOS, and ONOS is aiming to generically implement P4 [141], when integrated, the TENNISON application will be able to seamlessly benefit from P4.

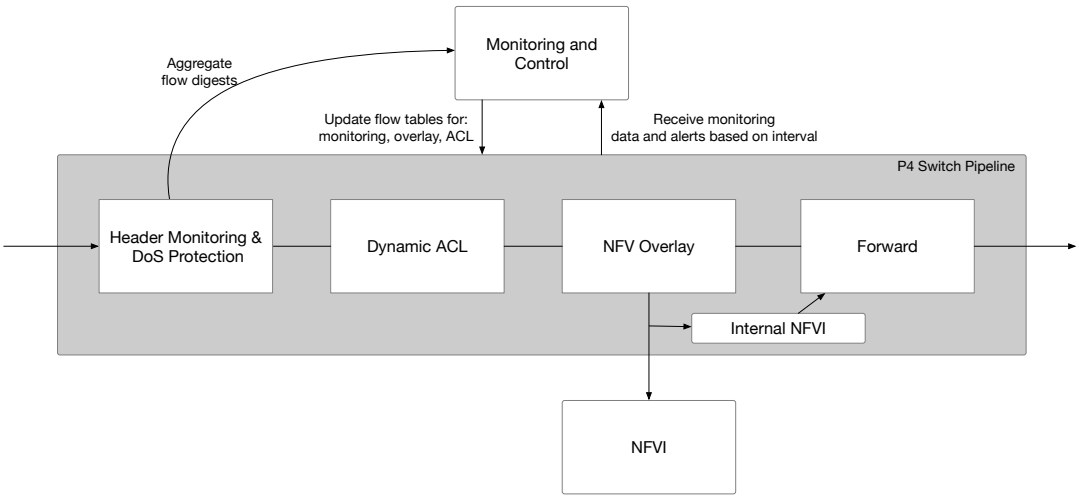


FIGURE 3.14: P4 monitoring pipeline design

The design for a P4 monitoring pipeline is illustrated in Figure 3.14. There are three key differences to note between the presented design for the P4 and OpenFlow pipelines. These differences consist of: message digests, DoS prevention, and internal NFVI redirection.

3.8 Summary

This Chapter has detailed an architecture to achieve the aforementioned aims of this thesis, and to address the challenges highlighted in Section 3.1. The decisions made here are based on the state of the art technologies as well as fundamental concepts in distributed systems, Software Defined Networks, and Network Functions virtualisation.

Fundamentally, the design decisions in this chapter have all been focused on achieving, scalability, responsiveness, extensibility, and efficiency, all whilst providing a pragmatic solution to implementation and deployment. Key to scalability, the recursive design of Tennison allows for tiering to be done without significant additional implementation.

The design of Siren and Tennison are intentionally agnostic of the southbound and northbound technologies to improve the longevity and validity of the designs. This is particularly prudent with upcoming data plane technologies such as P4, which show promise for greatly improving upon current southbound solutions.

This leads us onto the next Chapter where the designs of both Siren and Tennison are realised, detailing their implementations.

Chapter 4

Implementation

In the previous chapter, the design of two systems was described which would enable deployment a scalable monitoring service to the Cloud-to-Fog Continuum. This chapter describes the implementation of both the orchestration system: Siren, and the SDN monitoring system: Tennison, as well as their relationship.

4.1 Implementing Tennison

This section details the implementation of Tennison based on the design shown in Section 3.5.4. In addition to the core features of the system, supporting programs including the GUI and experimentation frameworks are detailed in Sections 4.1.3 and 4.1.9. On top of this, the operation of Tennison's example northbound applications are detailed to show the event flow of the system.

Previously, in section 3.5, the design of Tennison was detailed, Figure 4.1 shows the prototype implementation of this design, consisting of 4 layers.

The bottom most layer in the implementation shows the physical infrastructure. This consists of the physical network, the network controller, ONOS Core, and the deployed virtualised network services, which are delivered by Siren. The ONOS core connects directly with the next layer, passing control

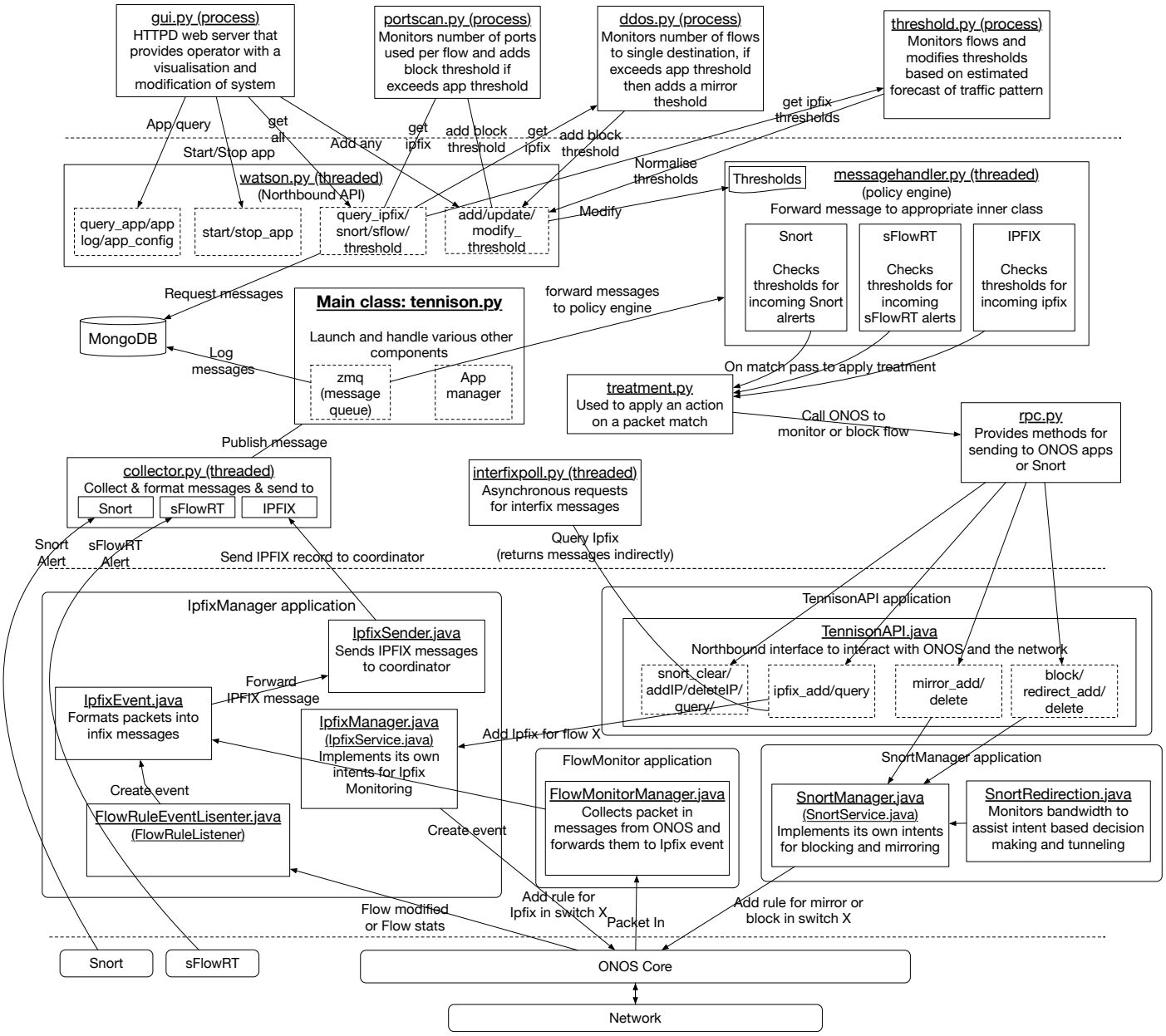


FIGURE 4.1: TENNISON Implementation Overview

and information about the network to the TENNISON and SIREN ONOS applications. This layer offers an abstraction to the core TENNISON application, providing light weight monitoring data to TENNISON, and making complex network changes as intents. On top of this, alerts from Snort and sFlowRT are passed up to the 3rd layer which informs TENNISON about network anomalies through deep packet inspection and provide light weight monitoring for legacy components of hybrid networks.

The third layer shows the core of the TENNISON implementation. The system is centered around a message queue that collects information from the network and is then accessed by the policy engine. These messages are then processed by the policy engine which as shown in Section 5.1.5.3, can process hundreds of thousands of network events per second before queuing events.

Finally, the uppermost layer shows TENNISON's northbound applications which interface to the third layer over a REST API. Applications attached to TENNISON have the ability to view all network events, manipulate the policy engine table, and directly interface with the TENNISON ONOS apps. On top of this, each application that registers with TENNISON is given a clone of the message queue, this ensure that it does not affect the operation of other TENNISON applications operation. The application that exercises most of TENNISON North Bound Interface is the GUI, which visualises TENNISON's policy engine, the network topology, connected DPIs, and network statistics about hosts on the network. The other applications shown in Figure 4.1 offer attack detection for DDOS, DOS, and port scans. Each one of these detection applications are under 200 Lines of Code and are available for review on TENNISON's GitHub page [199].

The remainder of this section goes into detail on the implementation of unique aspects of TENNISON not implemented in similar systems.

4.1.1 TENNISON Security Pipeline

A key benefit of TENNISON is the ability to provide network monitoring and remediation without interfering with forwarding functionality and additional services. This transparency is achieved with the TENNISON security pipeline, as shown in Figure 4.2. The pipeline manifests itself as an ONOS driver, which positions TENNISON's security tables in front of other network application tables.

A number of designs were considered for the security pipeline during the development of TENNISON, alongside an analysis of the level of conformance of current market SDN switching equipment with OpenFlow 1.3 functionality (such as, multi-table support offering the flexibility of match-action entries per table, and group chaining functionality). In order to provide a practical solution capable of deployment with available SDN devices, and because of limitations identified with current SDN hardware switches relating to the number of tables available, the match fields per table, and the actions available per table, the TENNISON security pipeline assumes only the base non-extended OpenFlow 1.3 requirements.

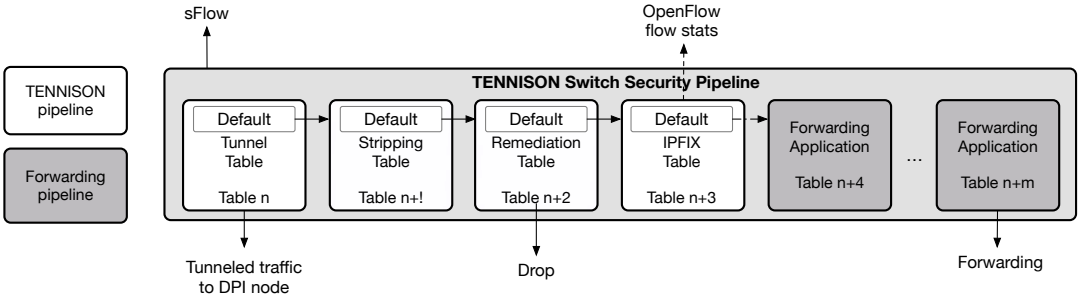


FIGURE 4.2: TENNISON Security Pipeline

As shown in Figure 4.2, tables controlled by TENNISON precede the regular forwarding functionality. This enables transparent monitoring and actions to be implemented without the requirement to modify the forwarding functionality.

This is important to generalise the process of monitoring networks; it is no longer tightly coupled to forwarding.

TENNISON employs a security pipeline with a total of four flow tables:

- The **Tunnel table** is the first table in the overall pipeline and acts as an overlay forwarding application that uses VLANs to separate tunneled traffic. The primary purpose of the table is to forward traffic to the nearest DPI service on the network.
- The **Stripping table** is used to strip tunneling forwarding logic (pop VLAN headers) from mirrored flows. Although not strictly required, this table is maintained to support compatibility across all OpenFlow 1.3 compliant hardware and software switches.
- The **Remediation table** contains the remediation intents. Following the required tunneling management, this table appears next in the pipeline in order to optimise network performance by blocking/dropping traffic identified as malicious before it absorbs further processing resources.
- The **IPFIX table** contains the monitoring intents. This table is last in the security pipeline and may pass monitored traffic either to the DPI table for further analysis or directly on to the forwarding pipeline.

4.1.2 Network Controller

TENNISON relies upon the presence of a network controller to operate effectively. Using SDN technology, this controller should be capable of viewing and modifying the underlying physical or virtual network paths, and support traffic steering or manipulation. As long as this functionality is present, TENNISON does not require a specific SDN technology. For the purpose of this work, ONOS is used as the Network Controller [22].

4.1.2.1 Controller distribution

Designed by ON.Lab, Open Network Operating System (ONOS) was launched in 2014 as a SDN network operating system for service provider networks with a focus on high availability, scalability and performance. ONOS implements distributed control with multiple controller instances forming a cluster. The clustering of controllers is a process through which one or more controllers are connected and data about the state of the network is shared between them. The intention of clustering is twofold: 1) to ensure that in the event of one controller failing the other remaining controllers in the cluster will ensure the network remains functional, and 2) to provide scale-out of the system; making it possible to manage networks with hundreds of networking devices and thousands of hosts.

The ONOS cluster instances synchronise to provide a global network view graph using the RAFT consensus algorithm. The *StorageService* interface ensures a consistent state of the databases across all the instances of an ONOS cluster. Each network element is assigned a master ONOS instance and the remaining instances will be secondaries for that network element. If the master instance fails, an election takes place between the remaining instances to elect a new master. It is possible to balance between the masters to provide an even distribution of network elements to each member of the cluster.

For Tennison, distributed ONOS provides a scalable and fault-tolerant substrate. Tennison also leverages the ONOS default distributed forwarding and routing applications.

4.1.2.2 Security Intents

As previously described, the Coordinator can change the behaviour of the network by interacting with the network controller. To do this, the coordinator pushes intents down to the controller, which then actions these to effect changes

in the network. These intents are topology-agnostic, with the controller handling their optimal placement. Several modifications have been made to ONOS to support integration with TENNISON, creating a more generic intent API. These modifications are present as an ONOS application and are represented as the Flow API in Figure 3.7. The ONOS application is registered with the highest priority in the ONOS event processing pipeline to support the flow illustrated in Figure 4.2. The provided intents are as follows:

Monitoring intent A monitoring intent instructs the controller to insert a monitoring rule in the network for a specific flow. Based on this rule, the ONOS controller will insert flows into switches that the flow traverses. It will then use this flow to receive detailed OpenFlow statistics about the flow. These statistics are then aggregated and converted into IPFIX data which is then passed to the TENNISON Coordinator via the *IPFIX Collector*.

Redirection intent A redirection intent instructs the controller to insert a rule in the network to redirect a specific flow (defined by a tuple) towards a specific type of appliance. For example, it may instruct the controller to redirect all TCP traffic destined for port 80 towards the nearest Snort DPI instance. This redirect modifies the complete flow, and stops any packets of the flow from traversing the normal forwarding path within the network. Based on the security pipeline logic, the redirection intent is written to the first table of the TENNISON pipeline. The redirect prevents the flow from continuing along the forwarding pipeline, and the traffic is tunneled to a new destination (e.g. DPI appliance).

Mirror intent The mirror intent has similar functionality to that of redirection. However, there is a significant difference in that the original flow remains in the network. As such, the traffic is forked, with the duplicate flow tunneled

towards a new destination (e.g. DPI appliance), whilst the original packets are forwarded as normal.

Remediation intent A remediation intent is the final step of a security application used by TENNISON, and enables the coordinator to make a definitive action with regard to an identified and detected threat. This includes completely blocking a flow in the network, or rate limiting a flow to control its behaviour. As previously noted, this intent is written to the third table in the TENNISON pipeline.

4.1.2.3 ONOS Application Pipeline

The ONOS application pipeline dictates the order in which applications receive events from the network. Figure 4.3 shows the TENNISON's implementation of the ONOS application pipeline, where its applications are prepended. This is of particular importance to TENNISON's functionality, enabling it to get events before any other applications, ensuring that it is receiving all available data and that it can drop monitoring events as to not unnecessarily overload the application ahead in the pipeline.

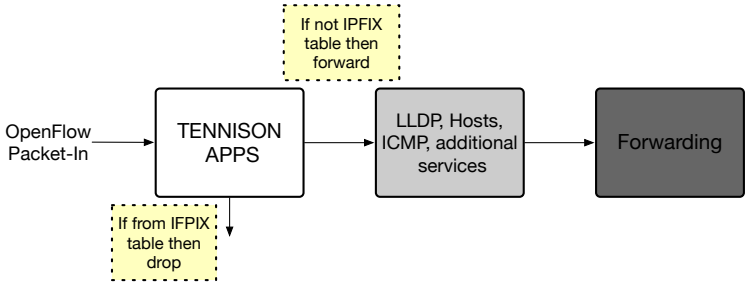


FIGURE 4.3: ONOS TENNISON Application Pipeline

4.1.2.4 Implementing Multi-level Monitoring

There are two key factors in the implementation of multi-level monitoring in TENNISON. The first is the TENNISON policy engine, which enables the network

operator to define and specify monitoring and security detection and protection mechanisms in accordance with the network deployment environment. Figure 4.4 provides a visualisation of the policy engine in the form of a match action table. This table is hit when a new *event* enters the coordinator; an event could be generated from a variety of sources including DPI, sFlow-RT, IPFIX, PreFIX, or a custom event from a northbound application. The three columns in Figure 4.4 represent the following: *Matches*, which consist of packet headers or alert types; *Conditions*, which verify if an event violates a threshold; and *Treatments*, which manipulate or upgrade the level of monitoring for specified traffic. For example, policy #2 can be read as follows: given a DPI alert on a specified MAC src address, that flow will be blocked. Similarly, for policy #4, given an sFlow_RT alert for an identified IP src address, and with the threshold specified by policy #1 exceeded, that flow will be rate limited.

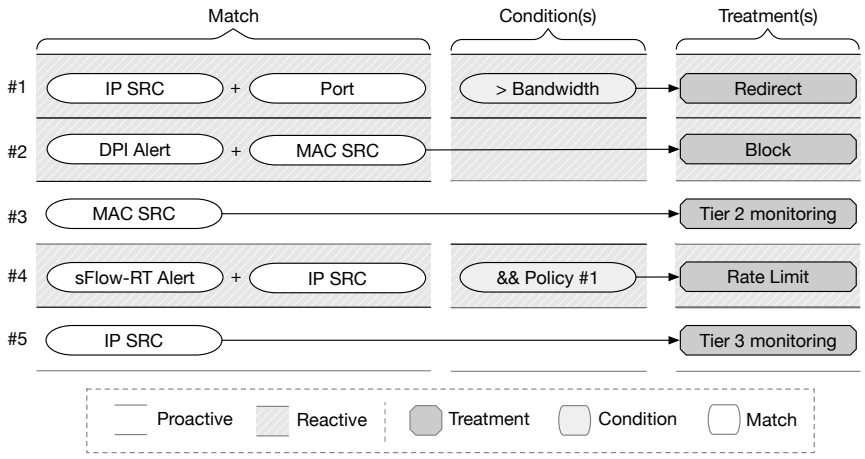


FIGURE 4.4: TENNISON Policy Engine Illustration

The second key element in TENNISON multi-level monitoring is the resource monitor, which provides three specific functions based on analysis of the resource usage information; (1) The optimal placement of the monitoring rule (i.e. on which switch(es) along the traffic path to place the rule) is determined with the objective of maximising network protection while maintaining network performance (i.e. avoiding potential switch flow table overflow and controller

processing overload). (2) In the case that a switch/controller pair is approaching a resource limit, the management of the switch will be transferred to another controller. This enables more efficient network operation and accelerates detection/protection times in the case of attacks due to reduced latency on the data-control communication path. (3) In a high-traffic volume scenario, the resource usage information is used to dynamically adjust the monitoring level (simultaneously reducing the IPFIX monitoring and increasing the sFlow sample-based monitoring to actively manage the network resources). While introducing a marginal increase in attack detection time, this multi-level monitoring design enables continued network operation under high load.

4.1.3 TENNISON Security Functions in Operation

TENNISON is designed to support the detection and remediation of a variety of attack types. In order to demonstrate TENNISON's multi-level monitoring capability, this section describes four attack scenarios (summarised in Table 4.1) that are used to exercise different components of the system. TENNISON's detection capability goes beyond these four attack scenarios, but they serve to illustrate the operation of TENNISON in the context of a series of specific attacks. Some steps have been simplified for ease of understanding. The accuracy and timeliness of TENNISON in detecting and remediating these attacks is detailed in Section 5.

4.1.4 Single Host Volumetric Denial of Service Attack

Attack: A single host is flooded with a high volume of traffic. The attacking traffic may appear to originate from anywhere on the network.

The system components required in the DoS Detection/Protection method are illustrated in Figure 4.5. (1) The operator defines the response to sFlow event alerts. In this example, the source is blocked. (2) The defined policy is

TABLE 4.1: Summary of Attack Detection/Protection Mechanisms

Attack Type	**Attack Identification**	**Fields of Interest**	**Detection Method**	**Protection Method**
DoS	Volume of traffic flows targeting a single host exceeds a defined threshold	IP Source and Destination	Level 1 (sFlow)	Block/Drop
DDoS	A volume of traffic flows from multiple source IPs targeting a single host exceeds a defined threshold	IP Source and Destination	Level 2 (IPFIX) & Level 3 (DPI)	Block/Drop
Scanning	Increase in Attacker, Host A, ratio to target addresses	IP Source and Destination & Port	Level 2 (IPFIX)	Block/Drop
Intrusion	Attacker tries to log in to an FTP server with a username containing the predefined control sequence	Port & Username Field	Level 3 (DPI)	Block attacker and FTP Server

installed in the policy engine. (3) sFlow datagrams received by sFlow probes are sent to sFlowRT. (4) A DoS event is detected based on the pre-configured sFlow application and the alert triggered in TENNISON. (5) The security policy is generated (i.e. block rule) by TENNISON and the event is logged. (6) the security policy is received by the ONOS flow rule subsystem (Flow API), and (7) OpenFlow flow rules are sent by ONOS to the network elements (not shown in Figure 4.5). (8) Finally, an alert is posted to the GUI to inform the operator that a DoS has been detected and blocked.

4.1.5 Distributed Volumetric Denial of Service Attack

Attack: A single host is targeted by a volume of traffic originating from a significant number of sources. The attacking traffic may appear to originate from anywhere on the network.

The system components required in the DDoS detection/protection method

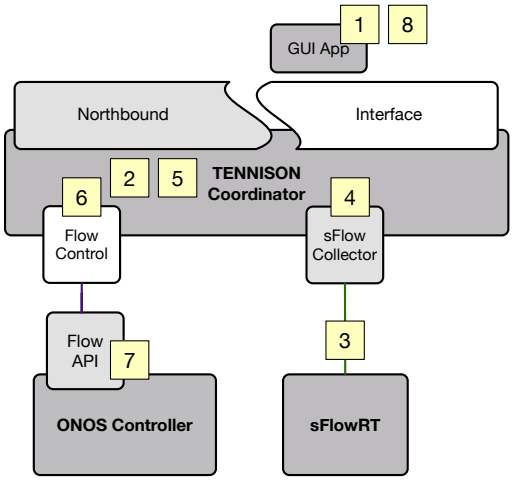


FIGURE 4.5: TENNISON sFlow DoS Detection/Protection

are illustrated in Figure 4.6. (1) ONOS exports IPFIX messages to TENNISON. (2) The DDoS security application identifies suspicious traffic and marks it for further monitoring. (3) A threshold is installed in the policy engine to use lightweight monitoring for any subsequent traffic matching that flow. (4) The coordinator instructs ONOS to install rules to use lightweight traffic monitoring (using TENNISON's ONOS intents). (5, 6) IPFIX monitoring on the ongoing traffic triggers the suspected DDoS threshold. (7) The threshold action is modified to redirect the suspicious flows to Snort. (8) The coordinator instructs ONOS to install rules to redirect traffic. (9) Snort identifies a DDoS attack and sends an alert to the coordinator. (10) The alert matches the alert threshold in TENNISON policy engine and triggers the block flow action. (11) The coordinator uses the ONOS TENNISON intents to block the attacking flow(s). (12) An alert via the GUI informs the operator of the attack remediation. (13) This process is repeated until all of the flows taking part in the DDoS are blocked.

4.1.6 Scanning Attack

Attack Definition: A single host (Attacker, Host A) scans another host on the network. The Attacker scans the top one hundred TCP ports as defined in the

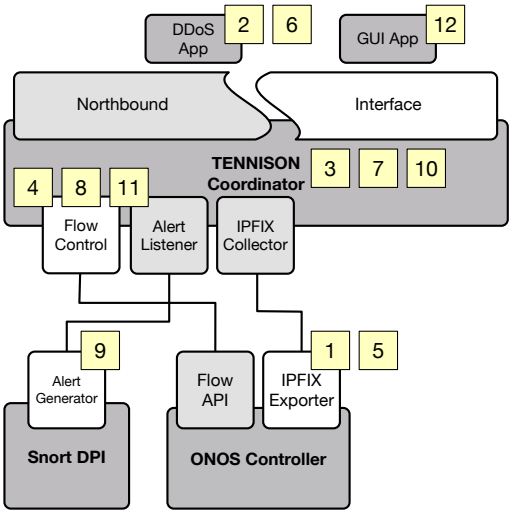


FIGURE 4.6: TENNISON IPFIX DDoS Detection/Protection

nmap-services file. The ports are scanned in random order. The attack may occur over an extended period of time (minutes to hours).

The system components required in the Scanning detection/protection method are illustrated in Figure 4.7.

(1) ONOS exports IPFIX messages to TENNISON detailing ongoing traffic. This data is read by the port scan security function. (2, 3) The security function identifies a port scan and posts an alert to the policy engine to block any subsequent traffic matching the malicious flow. (4, 5) The incoming message to the coordinator triggers the threshold in the TENNISON policy engine then applies the block flow action. (6, 7) The coordinator uses the ONOS TENNISON intents to block the attacking flows. (8) An alert is posted to the GUI informing the operator of the port scan detection performed to remediate against the malicious flow.

4.1.7 Intrusion Attack

Attack Definition: Detection of a scan for a vulnerable version of VSFTPD (Very Secure FTP daemon). VSFTPD version 2.3.4 was compromised with a backdoor in June 2011 [212]. In Table 4.1, the detection method is identified

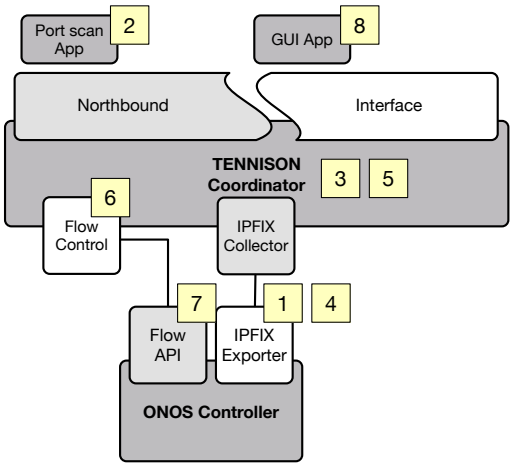


FIGURE 4.7: TENNISON Scanning Detection/Protection

as DPI. This can be further detailed as the detection of Ascii characters 58 41 10 (a smiley face) in the ftp username field with implicit recognition of the ftp control port, 21.

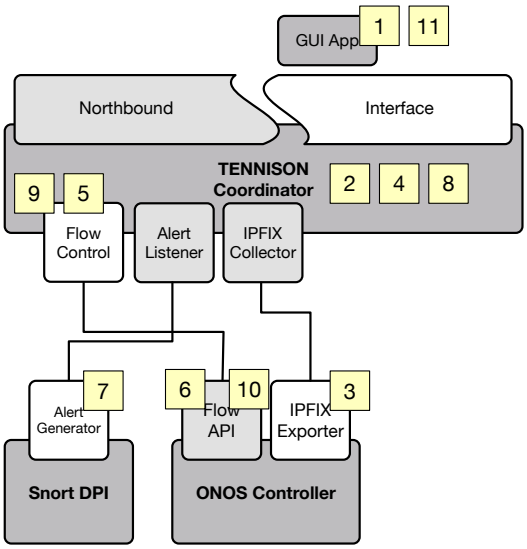


FIGURE 4.8: TENNISON Intrusion Detection/Protection

The system components required in the Intrusion detection/protection method are illustrated in Figure 4.8. (1-2) A new security policy is added to the TENNISON policy engine such that a flow rule is generated and pushed to the network elements to detect ftp traffic, (3-5) ftp traffic is then redirected to the DPI for inspection. (7) In the example provided, if the username contains a smiley face,

an alert is generated by the DPI and an event raised at the TENNISON coordinator. (8-10) A block rule is then created and pushed to the relevant network elements to isolate the malicious host. (11) The operator is alerted via the GUI that an attack has been blocked.

4.1.8 TENNISON Web Console

The web console provides the operator with network management and debugging information and also provides control of network monitoring for manual operation. The GUI is connected to TENNISON's northbound API, as described in Section 3.5.1.5. The GUI is served by a dockerised NGINX instance which is proxied to a python flask server, making use of a web backend with web sockets, accounting and shared state. Ultimately, the GUI can handle reporting live information to multiple users at once, with a minimal overhead per active user.

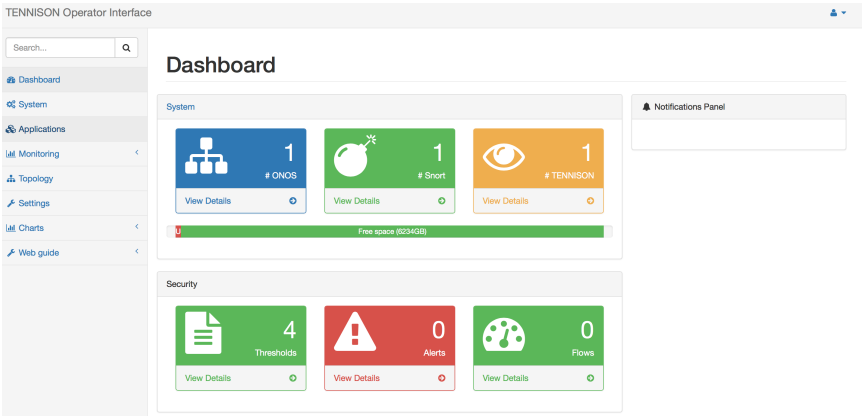


FIGURE 4.9: TENNISON GUI Dashboard

Figure 4.9, is a screen shot of the GUI dashboard. Information presented on the dashboard is a summary of what is already present on the other web-tabs. The dashboard shows information about the systems status and configuration, detailing the number of ONOS, TENNISON, and Snort instances connected, as well as load information about the load of the policy engine, number of alerts, and overall number of flows observed. The dashboard is particularly useful for observing and validating the system's behaviour.

FIGURE 4.10: TENNISON GUI Monitoring

Figure 4.10 shows an extract of the TENNISON's monitoring page. Providing an overview of traffic on the network, the monitoring page shows header information of the monitored flows in the network.

The flow data shown on the page is live and is made using WebSockets [215]. WebSockets are used between a client side JavaScript application and a server side python process with the flask library [82], resulting in live data sent between the coordinator to the GUI. The benefit of this is that the GUI provides a more tactile feel with the data, assisting with debugging and demonstrating the system's capability.

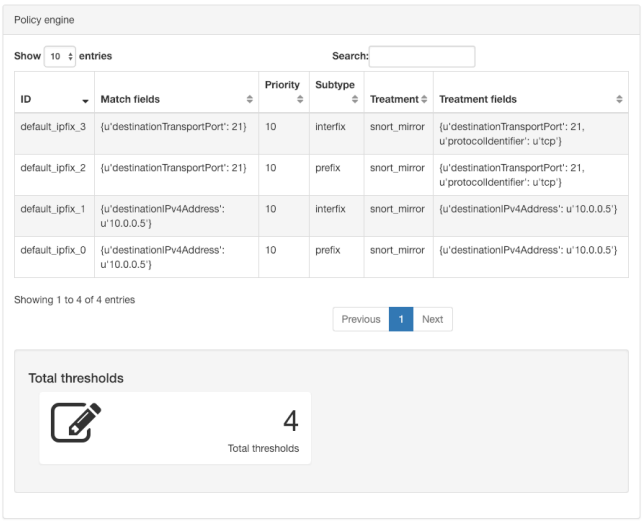


FIGURE 4.11: TENNISON Policy Engine

Figure 4.11 shows the state of the policy engine within the TENNISON coordinator. For manual operation, policy entries can be modified, deleted or created here; this is especially useful for testing or for temporarily patching security holes on the network until the underlying cause is remedied.

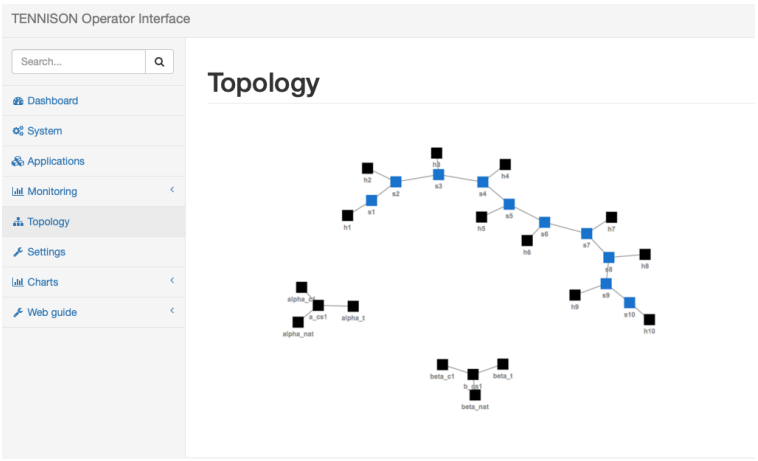


FIGURE 4.12: TENNISON GUI Topology

Figure 4.12 shows the network topology, including TENNISON nodes. The example shows TENNISON running in tiered mode with two sub domain managers and two instances of ONOS. When interacting with the topology shown in Section 4.2.5, similar to the SIREN topology, a web modal pops up with device information detailing the resources available and connection information.

As well as the pages above, the GUI has various other pages which for brevity are not illustrated here. These include a log-in page, tiered GUI page, tiered topology page, and an application page. The applications page allows for TENNISON northbound applications to inject their own management page which is automatically generated from their configuration file. An example of an application management page is available in Section 4.17.

4.1.9 Experimentation Framework

In order to conduct rigourous and repeatable experiments with tiered Tennison, distributed ONOS, and Siren, a considerable amount of automation was required as manual initialisation was not feasible, taking several hours for an experiment run. When testing at the largest scale as shown in Section 5.1.7, the environment required six instances of Tennison, twelve instances of ONOS, twenty-four instances of Snort, one-thousand instances of OpenvSwitch, and one-thousand emulated hosts. Whilst Mininet is designed to spawn the switches and hosts at this scale, significant work was done in automatically spawning and connecting ONOS, Tennison, and Snort to the network.

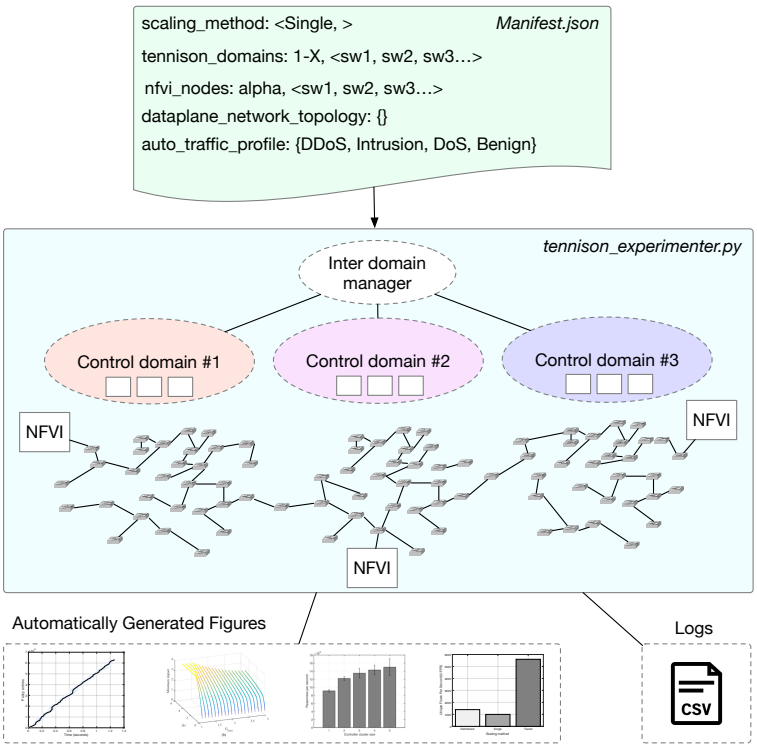


Figure 4.13: Tennison Experimenter Design

Figure 4.13 shows a representation of the experimentation framework, showing a manifest file that describes the experiment parameters which is then fed into the experimenter application. The experimenter then launches all the components along with a profile that specified timings of replayed network events,

this is ran for a predetermined time (default 10 minutes) the results are gathered and then the experiment is terminated or rebuilt for another iteration.

4.1.10 Tiered Implementation

As detailed in Section 3.5.4, tiered TENNISON is required to maximise the scalability of SDN network monitoring and remediation. Tiered TENNISON is based on a recursive design, reusing most of the core system at each tier of operation. TENNISON's recursive implementation means that a single code base can be used and maintained for different systems, instead of having three separate code bases for each tier. The following section details the specific additions to the core of TENNISON required to support tiering. These include additional north and south bound interfaces to TENNISON's coordinator.

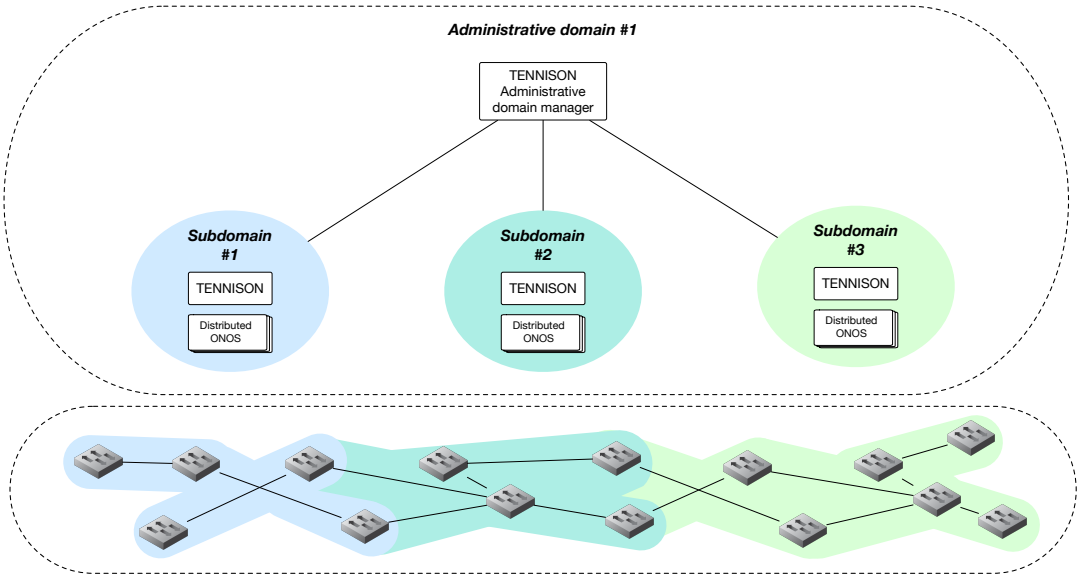


FIGURE 4.14: Tiered TENNISON

Figure 4.14 shows an overview of TENNISON's tiered implementation, showing a domain manager that has control over multiple subdomains, which in turn connects to distributed ONOS clusters. In order to experiment with a prototype of tiered TENNISON, only the subdomain and domain tiers from Section 3.5.4 has been implemented. In the tiered TENNISON prototype implementation, each

subdomain operates independently with full control over the network it is managing. Changes sent down from the domain manager are subject to acceptance via the configuration in the tiered manager at each subdomain.

The remainder of this section describes the required additions to TENNISON, including collectors, GUI, and methodology.

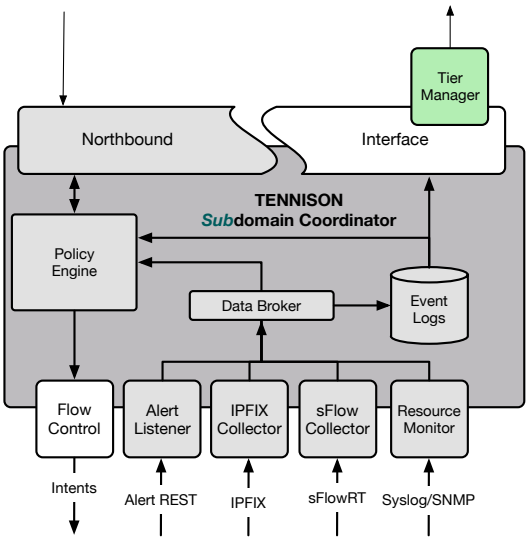


FIGURE 4.15: Sub Domain Manager

The element in Figure 4.15 highlighted in green shows the additions for the sub domain managers. The Tier Manager Application sits above TENNISON's northbound API and relays the state of the policy engine and the event logs to TENNISON's domain manager. As shown in 4.17, the Tier Manager comes with a web interface application page providing functionality to configure the type and detail of data passed up to the domain manager.

The *Domain Manager* is at the highest level of control in TENNISON's tiered architecture, having the widest visibility of network monitoring. Figure 4.16 shows the additions for the domain manager, including a generic southbound input and a tiered GUI application. The four center southbound interfaces on the *Domain Manager* are not used as messages from the *Sub Domain Managers* are already processed and serialised, instead all messages are sent through a generic input southbound interface. The Flow Control interface operates similarly to

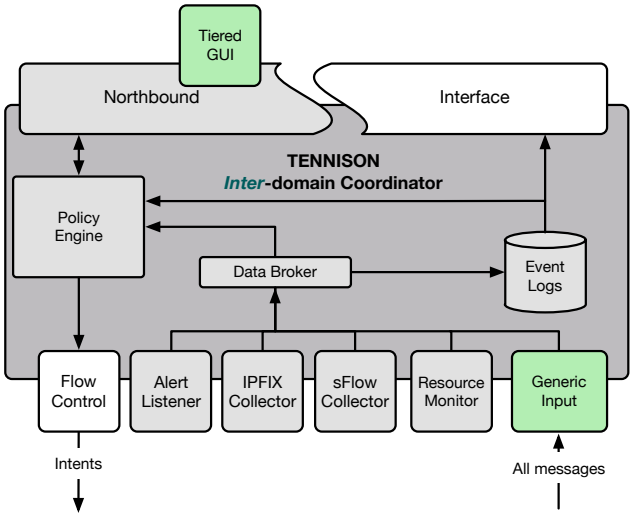


FIGURE 4.16: Domain Manager

the *Sub Domain Manager* but instead of pointing to the network controller the intent is passed down to the sub domain manager which then forwards it on to its own network controller.

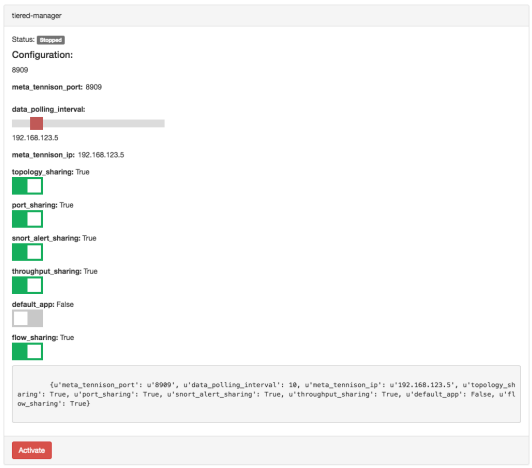


FIGURE 4.17: Tiered Manager GUI Configuration Component

Figure 4.17 shows the tiered manager application GUI for TENNISON's sub domain. The TENNISON GUI has been updated to allow the modification of northbound applications at runtime. For the tiered manager, this allows the user to dynamically set the level of data that is sent from one tier to another.

This includes the polling rate, and a series of checkboxes to select the security primitives to be shared, such as packet headers and snort alerts.

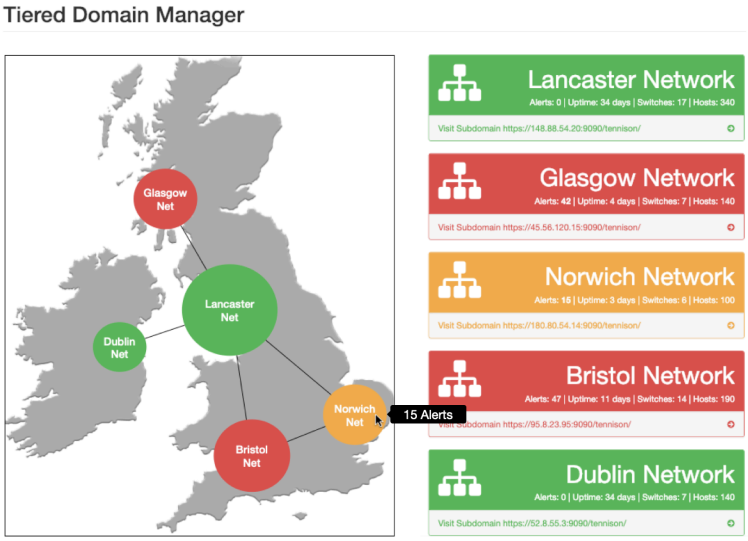


FIGURE 4.18: Tiered Domain Manager GUI

Managing multiple instances of TENNISON requires a central access point. Figure 4.18 shows the tiered GUI manager for the tiered TENNISON domain manager. On the GUI, each site is shown along with a summary of the site's status, on top of this is an access URL that redirects the operator to that sub domain's instance of TENNISON. The left side of the figure shows the geographical network topology, detailing the location, size, and number of uncleared alerts at each site through a traffic light system. The Topology diagram was created in JavaScript with D3[1] and automatically updates in size and colour from web-sockets that are periodically sent data from the TENNISON's Tiered Domain Manager.

This section has detailed the prototype implementation of tiered TENNISON, showing the additions to the standard architecture and graphical interface. Results on the impact of this design are available in Section 5.1.7.

[1]https://www.d3.org

4.1.11 Summary of TENNISON Functionality

To conclude this section, the attributes in Section 3.1 are revisited, detailing each challenge is overcome from a functional standpoint.

Chapter 2 highlighted security frameworks and applications that build on the SDN characteristics of programmability and logically centralised control. A number of these works identify the deficiencies of relying on centralised control when deploying network security solutions. Specifically, this relates to the performance limitation introduced by the control communication channel. The response to a network attack should be as efficient as possible, and not constrained by communication channel resource, or indeed, controller processing capacity. A further limitation of recently proposed security frameworks/applications is the reliance on modifications to OpenFlow, SDN devices, or interference with fundamental network forwarding behaviours. These modifications often restrict the proposed solutions to deployment in a specific network type, with specific network equipment, and for a single or limited set of security functions.

The TENNISON framework is motivated by the desire to present an adaptive and extensible security platform that is technology-independent and capable of supporting a wide range of security functions. TENNISON does not remove the requirement for controller interaction, as demonstrated in Avant-Guard and OFX [180, 186]. However, the level of controller interaction is rendered flexible and proportionate to the threat detection requirements. This is further helped by the use of other monitoring and inspection tools that are deployed to the network, relieving pressure from the SDN control channel. For example, with one level of monitoring, sFlow provides a separate monitoring path is used while maintaining the programmability of remediation mechanisms via the SDN controller. This level of defence can be configured by the network operator or automatically adjusted in response to resource monitoring in order to optimise network protection. Through efficient monitoring, tiering, and distribution,

TENNISON has multiple solutions for scaling network monitoring.

- **Efficiency and Proportionality:** TENNISON provides an efficient monitoring and remediation framework where resource consumption is commensurate to current threat levels. This includes tailoring the type and scale of monitoring according to the anticipated attack types within a network along with actual resource usage such that the monitoring and security functions are appropriately distributed. TENNISON supports standard network monitoring protocols such as sFlow and IPFIX, incorporating them in a multi-level monitoring function. This means that separate monitoring paths are available and the optimal monitoring method can be selected per security function. The remediation mechanisms are similarly efficient; enabling an operator to prevent, scale or limit an attack, depending on certainty or severity.

  As demonstrated, a volumetric attack can be efficiently detected using TENNISON Level 1 monitoring with sFlow while more fine-grained attacks engage Level 2 IPFIX monitoring and/or Level 3 DPI capabilities.

- **Scalability and Visibility:** The network view provided by the distributed ONOS control function and leveraged by the TENNISON coordinator enables placement of the monitoring and remediation rules on the appropriate network devices for optimal security protection.

  TENNISON has an overview of the entire network with scalability enabled by the lightweight and latent monitoring solution. Visibility across the network includes legacy equipment, through compatibility with ubiquitous protocols such as sFlow. Furthermore, TENNISON is designed with a coordinator separate to the distributed SDN controller such that control and management can be separated, as required, supporting parallel processing across multiple devices. This both helps with scaling out the

system, and maintaining openness in the selection of TENNISON's controller. On top of this, TENNISON's support for a tiered architecture maximises both scalabiltiy and network visibility, providing monitoring and control over networks with thousands of devices.

- **Programmability and Extensibility:** The pre-defined security functions described in this section are activated or deactivated via the GUI. To modify a threshold in an existing function, for example, DDoS attack trigger threshold, the new threshold value is simply added via the GUI. This will automatically be mapped to the relevant TENNISON components. To introduce a new security function, the template sFlow/IPFIX/DPI application can be modified and activated via the GUI.

  The TENNISON framework provides a rich API that allows operators to define the behaviour of the network and its resources in response to new and existing threats. This includes offering multiple monitoring and remediation mechanisms, and a GUI, each of which can be flexibly used by the operator. Rather than simply providing a finite set of security functions, it is possible to build new security functions within the TENNISON framework.

- **Transparency:** As per the TENNISON security pipeline, benign traffic is processed through the network as normal while suspicious traffic is monitored and dropped or forwarded, as determined by the activated security functions. On top of this, the ONOS pipeline was modified to ensure that TENNISON's apps processed packets first.

- **Availability and Resiliency:** Due to the distributed control platform, the monitoring and security capabilities of TENNISON are maintained even in the case of a failure of a controller instance.

One of the key requirements of a SDN security platform is the resilience of the control plane to support high availability and provide redundancy in the case of controller failure [171]. For this reason, TENNISON has been designed to integrate with a production-grade distributed controller, ONOS [22], which supports high availability and fault tolerance.

TENNISON is transparent to other network functions within the network. This is realised using a custom *security pipeline*, which means that it can run seamlessly alongside other network functions without modifications. This is built using the OpenFlow multi-table feature to provide the network monitoring and remediation without interfering with the basic forwarding functionality and additional services of the network.

- **Interoperability:** The integration of Snort DPI with TENNISON is demonstrated with the intrusion attack example. Available DPI instances are automatically detected by TENNISON. The operator can configure an alert to be displayed on the TENNISON GUI to highlight a particular attack detection.

- **Accessibility:** TENNISON security policies can be added or modified both via an API and through a purpose-built graphical user interface (GUI). The GUI presents the extensive functionality of the API in a user-friendly and accessible style. Consequently, it is easy for researchers and operators alike to extend the TENNISON system for further research or to adapt to a new scenario.

Finally, TENNISON works in conjunction with de-facto industry-standard security tools. The DPI connectivity has been tested and operated with Snort, however, the implementation is agnostic in that it supports a range of other DPIs including Zeek [202] and Peafowl [153].

4.2 Implementing SIREN

This section details the implementation of the Infrastructure Management Framework, SIREN based on the design shown in Section 3.6. Initially, this section illustrates SIREN's implementation through a class diagram, which shows all primary components and how they are connected.

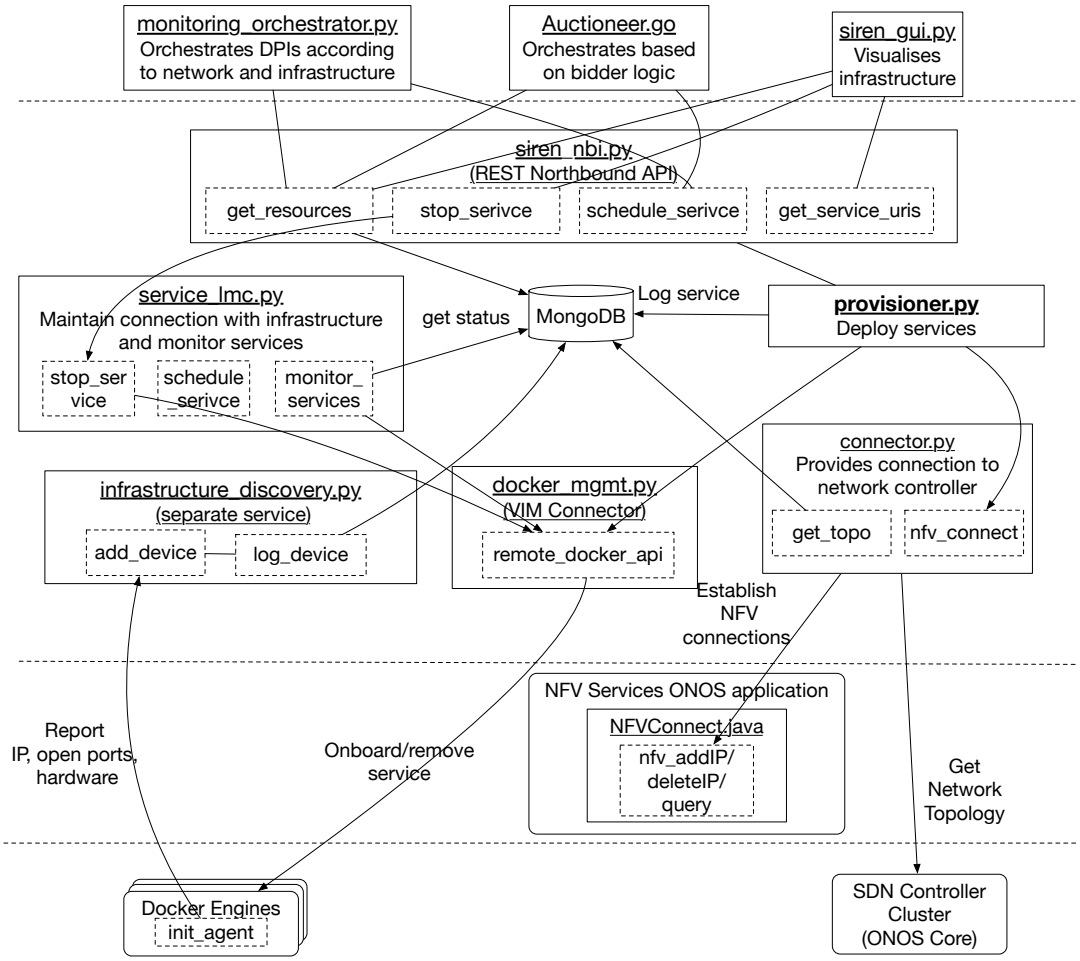


FIGURE 4.19: SIREN Implementation Overview

Figure 4.19 shows the relationship and code layout of the SIREN prototype implementation. Within the diagram, SIREN is constructed of four layers, following the design shown in Section 3.6. The bottom most layer shows the network and virtualisation infrastructures, one abstracted by ONOS, and the other by docker. These components are then connected to the core of SIREN, to get container state, launch new containers, and retrieve the network topology.

In SIREN the Docker Engine is used as an agent for each NFVI. The remote Docker API [62] is used to poll information about the host including the running services, health of each service, and amount of resources remaining. To retrieve information about the network, ONOS's northbound REST API [142] is used, in effect, making SIREN a northbound application to ONOS.

The upper most layer shows the northbound applications, including the GUI and orchestration additions. These application interface with SIREN's REST NBI [183] to indirectly monitor and manipulate the network and virtualisation infrastructures. The monitoring orchestrator application takes the network topology, virtualisation topology, and deploys monitoring services to the network. The Auctioning application [4] advertises virtualisation resources to third parties, and then proceeds to auction them off. SIREN's life cycle manager automatically removes services after the lease period has expired.

4.2.1 Test Virtual Network Functions

A set of example, over-the-top or application layer network functions were created for demonstration purposes. These represent the functions that each service provider wants to deploy. In reality, one service provider may offer a multitude of different services.

4.2.1.1 DPI

Snort [163] is used to perform DPI on packets so that they can then be forwarded to a central control which then informs the network operator. Alternatively this alerting process could be automated, effectively making an IPS. These two alternatives have a trade-off between network overhead and remediation time. For the purposes of the results below, the DPI was in a store, check, and forward mode. SIREN also supports using this in a mirroring configuration, whereby the user does not see any increased latency.

4.2.1.2 DNS

PiHole [61] is a popular locally executable DNS service that black-holes unwanted DNS requests. This offers the ability to block certain domain names whilst providing lower latency than public DNS servers. This would be a service that could be used by either business or home users. In this example, clients are configured to use PiHole for all of their DNS requests.

4.2.1.3 CDN

A CDN based service is generally useful to home and business customers. For the CDN, a NGINX server hosts a 'big bunny' video file. Through the operator interface detailed below, the service IP address is presented which can be used for a new clients to connect to it. This is configured through the client's video player, usually via a manifest file.

4.2.2 Network Controller

Originally built with Ryu, SIREN moved to using ONOS for compatibility with TENNISON. For NFV orchestration, the network controller provides NFV connectivity and updates on the network topology.

SIREN's network controller connectivity is essential for effective orchestration of network services. In particular for vDPIs, the controller passes information about network traffic and topology so that SIREN can be deployed to areas that require increased levels of monitoring.

4.2.3 Dynamic Redirection and Mirroring to Distributed VNFs

The dynamic redirection and mirroring network function takes in the network topology and SIREN's infrastructure state to map locations on the network to

create an overlay network for traffic that needs to be redirected to a VNF. In the network monitoring use case, the NFV overlay is used for mirroring traffic to a DPI. In the content caching use case, this entails redirecting traffic to the nearest cache.

Currently, open source industry-grade SDN controllers do not have a generic, vendor agnostic tunneling solution. Where tunneling is required in an SDN environment, it is typically manually configured on the SDN switch using an existing tunneling protocol, such as Generic Routing Encapsulation (GRE). This approach is unsuitable for scenarios where there is a need for the dynamic scaling up/down of functionality, for example, allowing network functions to be added or removed within the network automatically. For this reason, Tennison introduces a new OpenFlow-based tunneling solution.

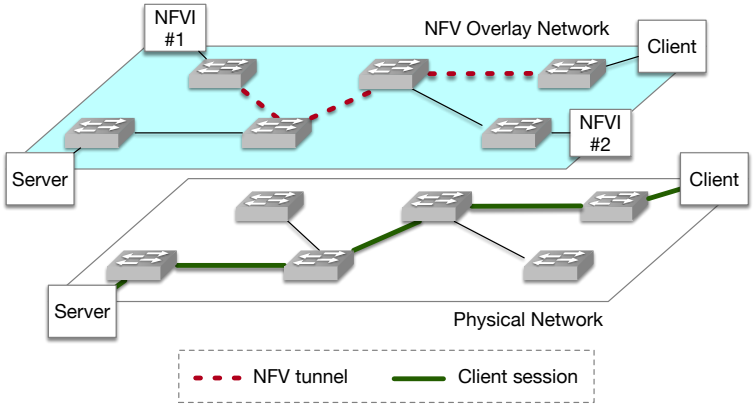


Figure 4.20: Overlay Network Tunnels

This solution is offered as an ONOS application that can be called by other applications to create point-to-point tunnels; importantly using only OpenFlow to achieve this. In the case of Tennison, tunnels are created between the switches and DPI nodes on the network. Suspicious traffic is then tagged with a VLAN ID for mirroring and forwarded via the tunnels to the nearest DPI on the network. When new DPIs are added to the network, tunnel paths are automatically recomputed, ensuring that flow tagged by the system as suspicious are routed to the nearest DPI.

The integration of tunneling in the TENNISON security pipeline is illustrated in Figure 4.2 and described in Section 4.1.1.

4.2.4 Monitoring Orchestrator

The monitoring orchestrator is a northbound application to SIREN, and is the key component in use for TENNISON. This is important as it automatically and intelligently places DPI services into the network and connects them to the network using ONOS.

Algorithm 1 shows how the core operation of the monitoring orchestrator works. Firstly, resources from both ONOS and SIREN are queried, returning information on the underlying infrastructure and network. The algorithm runs at controller startup and then via events which show that the policy is not satisfied. This information is then used to satisfy the policy as detailed in Section 4.2.4.1.

The *policySatisfied* method takes in the operator written network policy. This method checks to see if the policy is satisfied, getting resources deployed to the infrastructure, and confirming that they are in accordance with the network policy. One example of this is to see if one of the resources has reached its policy limit, in which case the policy is not satisfied and the loop continues.

The core element of the algorithm is the *findOptimumResource* method that takes in virtualisation and network resources as well as a map links them together. With this information the method initially returns the policy's minimum amount of vDPI as randomised locations over the network such that there is mixed distribution of them. After the initial deployment, this method will get available resources when services become exceeded. Policy permitting, the method will then proceed to return optimum locations based on one closest to the overloaded service. After this cycle has executed, vDPIs that are not used

(assuming at least a minimum deployment) will be freed so that resources are available for overloaded locations.

Algorithm 1: Monitoring Orchestration Algorithm

```
Result: DPIs automatically deployed and scaled within the network
policy = getPolicy();
netRes = getNetworkResources();
virtRes = getVirtualisationResources();
replicas = 0;
map = getVirtToNetMap();
while !policy.satisfied() do
    resource = findOptimumResource(netRes, virtRes, map, policy);
    Provisioner.deploy(resource, images.dpi);
    replicas ++;
    ONOS.addNFVI(resource.address);
end
Function findOptimumResource(netRes, virtRes, map, policy):
    if replicas < policy.min_replicas() then
        return randomResource();
    end
    r = getExaustedResource(policy.req_per_second)
    if r! == null then
        return closestResource(r, netRes, virtRes, map);
    end
    return;
```

4.2.4.1 Orchestration Policy Manifest

Figure 4.21 details the manifest file for Siren's monitoring orchestrator. This is used to configure the orchestration policy, allowing the operator define parameters that influence how vDPI services are deployed to the network. These parameters include the minimum resources (CPU, Memory, Disk, Network) required to deploy a service; this helps the orchestrator find appropriate NFVIs. On top of this, a number of requests per second is defined which is how the orchestrator identifies whether a vDPI VNF is overloaded. Finally the scaling policy is defined, which sets the minimum and maximum number of replicas that the orchestrator is permitted to deploy.

```
# Siren Monitoring Orchestrator Policy

service_requirements:
  cpu: 1
  memory_gb: 0.5
  disk_size_gb: 10
  req_per_second: 5000
  app_name: "siren_snort_dpi"
  remote_connection: "siren"

scaling_policy:
  min_replicas: 2
  max_replicas: 10
```

FIGURE 4.21: Monitoring Orchestrator Yaml policy file

The policy shown in Figure 4.21 is similar to policies used within Cloud based orchestration solutions. For future work, this policy could be extended to include cool down periods for scaling, adaptive thresholds, and configurable polling times.

4.2.5 SIREN: Web Console

This section details the web console which assists the operator in managing SIREN and its infrastructure. The web console is illustrated in Figure 4.22.

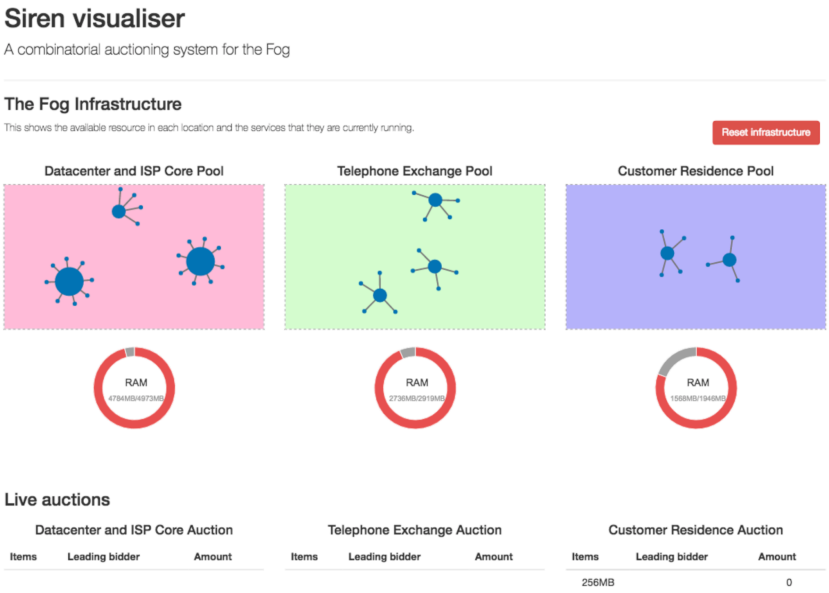


FIGURE 4.22: SIREN GUI

The web console shows the operator available resources in the NFV Infrastructure and what services are currently running. The primary purpose of this is to demonstrate as well as verify SIREN's operation. From left to right, the GUI shows pools of resources within the Cloud-to-Fog continuum. At the bottom of the web page, information about live auctions is also presented, so one can see which providers are competing for each resource. When an NFVI is clicked on the interface, a list of URIs is provided which the operator can use to verify the services that is running on each device.

4.2.6 SIREN in Operation

This section describes a high level step walk through of the operation of SIREN via the use of an example of one of its orchestration methods.

The following time series goes through the steps illustrated on Figure 4.23, showing SIREN in operation using its monitoring orchestration application:

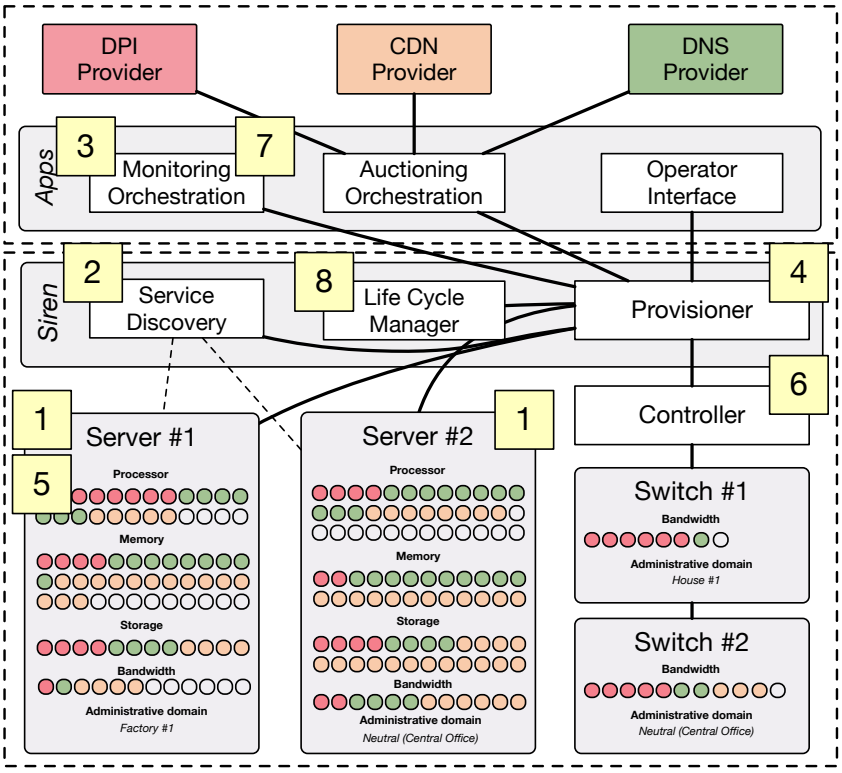


FIGURE 4.23: SIREN Operation

1) Firstly the server boots and registers itself with the discovery service, reporting IP address, firewall, CPU, Memory, Storage, and Network interfaces. 2) This information is then crossed with topology data to tag the server with a location. On top of this, the provisioner is notified of a new resource. 3) The monitoring orchestrator request for an initial deployment, spreading DPI services across the network according to the orchestration policy. 4) The provisioner interfaces with the infrastructure over remote docker API to start the new service. On top of this, the new service is added to the LMC. Additionally, the controller is contacted to create a new network overlay. 5) The docker engine pulls in and executes the new network service. 6) The network controller creates a new network overlay in preparation for the new network service. 7) After a policy predefined time, the orchestrator evaluates the network traffic and current deployment and then continues to deploy more services in optimal locations, repeating steps 4-7. 8) After the policy defined lifetime, the Life Cycle Manager removes the service from the NFVIs.

4.2.7 SIREN Summary

This section has presented a prototype implementation named SIREN and its operational behaviour for managing and orchestrating network services in a Fog environment. Key to this thesis, SIREN enables TENNISON to be effectively deployed over heterogeneous environments.

The SIREN prototype implementation has focused on the support required to operate TENNISON effectively in the Cloud-to-Fog continuum. This has included supporting heterogeneous environments, scaling out virtual network functions to network traffic and context, and creating solutions to unsolved challenges in NFV, including VNF placement and NFV service chain forwarding.

This Chapter has presented two proof-of-concept implementations that together are capable of scalable network monitoring and remediation in the Cloud-to-Fog continuum. In the next Chapter, these systems are evaluated, validating their ability to monitor networks at scale in a responsive manor.

Chapter 5

Evaluation

In this chapter, the designs for the Siren and Tennison prototypes implementations are evaluated. Firstly, in Section 5.1, a comprehensive evaluation of Tennison is shown, evaluating its scalability, responsiveness, performance increase over upcoming technologies, and finally a direct comparison versus similar frameworks. Next, in Section 5.2, an evaluation of Siren is detailed via an economically motivated example, demonstrating the benefit of operating across the Cloud-to-Fog continuum.

5.1 Tennison Evaluation

In this Section, a comparison of Tennison against similar frameworks is provided. This is followed by an analysis of Tennison's application performance evaluated in the context of the attack types described in Section 4.1.3.

5.1.1 Framework Comparison

A further comparison of Tennison against similar work is available from [**hohlfeldguest**].

In Table 5.1, the features of prior scalable, distributed monitoring and security systems are detailed and compared the Tennison framework.

TABLE 5.1: Scalability comparison of SDN security systems

System Name	Controller	Multi-level monitoring	OF response methodology	Attack detection (Extensibility)	Distribution	SIEM-Like Web Interface
TENNISON [199, 76]	ONOS	Yes	Hybrid	4+ (Yes)	Yes (Control & System)	Yes
FRESCO [181]	NOX	No	Reactive	3+ (Yes)	No	No
CIPA [41]	POX	No	Reactive	3+ (Yes)	No	No
SDN4S [104]	HPE VAN	No	Reactive	1+ (Yes)	No	Yes
PSI [221]	ODL	No	Proactive	1+ (via NFV)	Yes (System)	No
Athena [112]	ONOS	No	Reactive	2+ (Yes)	Yes (Control & System)	No
CHAOS [179]	Floodlight	Yes	Reactive	1	No	No
OFX [186]	Ryu	No	Hybrid	2+ (Yes)	Yes (Switch)	No
Kandoo [92]	Kandoo	No	Hybrid	No (No)	Yes	No
ElastiCon [60]	Floodlight	No	Reactive	No (No)	Yes	No
Hydra [40]	Floodlight	No	Reactive	1	Yes	No

In summary, the key differences between TENNISON and the other similar frameworks listed in Table 5.1 are the dependence on the the OpenFlow Packet-In response technique, extensibility, and scaling methods. As detailed in the design consideration's for this thesis in Section 3.3, the methods used in TENNISON are to optimise scalability, responsiveness, and extensibility.

TENNISON has various aspects that show its ease of use. Table 5.2 shows the comparison of lines of code (LoC) for each attack detection application between TENNISON and Athena [112]. As shown in Appendix A.1 TENNISON has a published API, an element not found in the other listed frameworks. It is not possible to provide the comparison with other similar security frameworks [221, 181, 40, 41] as either their source code is not openly available or they do not support user applications.

TABLE 5.2: User Application LoC Comparison

System	DDoS	DoS	Intrusion	Scan
TENNISON	107	304	0	135
Athena	1946 [14]	-	-	-

As shown in similar work [77, 181, 112], LoC can be loosely attributed to the development learning curve of a system. The comparison in Table 5.2

shows that application creation and integration with TENNISON is relatively easy. This is due to TENNISON's rich RESTful northbound API, which enables users to easily and succinctly create applications without directly adding to the complexity of the larger system.

5.1.2 Evaluation Environment

To evaluate TENNISON, a topology incorporating 350 nodes connected to 19 partially connected switches, representative of a large sized business network [2] with access, distribution and core networking layers. This topology was realised using the network namespace deployment tool: Mininet [124]. Mininet is an emulation tool that enables evaluation, validation and measurement of SDN applications. In order to create the network topology, Mininet instantiates Linux Namespaces [130] to act as hosts and software switches, (such as Open vSwitch (OvS). One of the significant benefits of Mininet, beyond using a simulator to create such a topology, is that each of these namepsaces have their own virtual network interfaces with a fully fledged and isolated networking stack. This allows testing of systems at significant scale whilst consuming minimal resources. The testbed runs on a general-purpose server with 256GB RAM, and two Intel Xeon E5-2697v4s totalling 32 cores. The server runs Ubuntu 16.04.3 and resources were shared between Mininet (Hosts, OvS), the SDN Controllers (ONOS v.1.11) and TENNISON. In order to test performance at scale, the controller benchmarking tool cbench [178] is used, which is capable of emulating the control plane of thousands of SDN switches at once.

5.1.3 Distributed SDN Controller Performance

Prior to testing the performance of TENNISON, an evaluation of ONOS was carried out using cbench to understand the scalability properties and potential

overheads associated with running a distributed SDN controller. Two specific experiments were conducted.

Firstly, the maximum throughput of the controller with respect to packet-in processing was measured. This was achieved by sending packet-ins originating from 16 emulated switches, and then counting the number of response packet-outs from the controller. Following ONOS's testing practice [137], this experiment was repeated one hundred times to ensure the average results was valid. The results in Figure 5.1 show the number of responses per second for an increasing number of controller instances. From this, it can be determined that the performance of ONOS increases negatively in a logarithmic way according to the cluster size. Furthering this, cluster sizes larger than 5 are unstable, failing to distributed messages in a timely manner ultimately causing syncronisation issues between nodes.

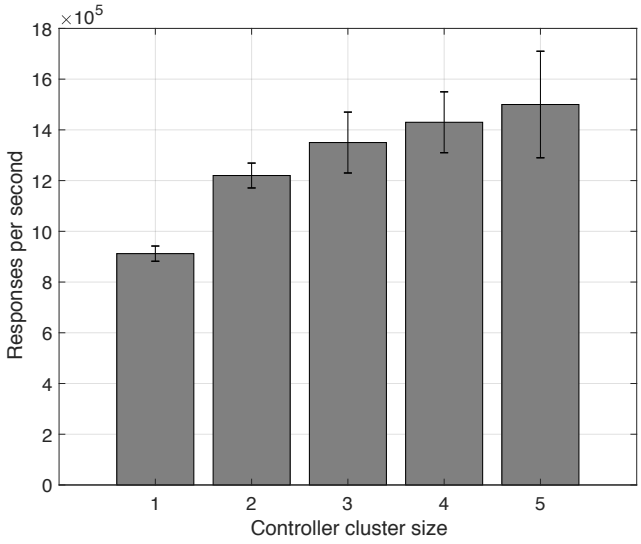


FIGURE 5.1: ONOS Controller Performance (Responses/s) - multiple controller instances

Secondly, the impact that an increasing number of switches has on a variety of ONOS cluster sizes was measured. For this one thousand packet-ins per second are generated per switch, and the delay in responses is measured and averaged.

The results in Figure 5.2 illustrate the latency with both an increasing number of switches and an increasing number of controller instances. These results highlight the trade-off between cluster size and the number of switches controlled by the cluster. For a network of up to 32 switches, a single controller instance provides the optimal response time. The reduction in latency is only achieved with additional controller instances for larger network sizes. Another factor that affects the optimum cluster size is the level of fault tolerance desired. Notice that 64 switches on a 5 node cluster has a greater latency than on a 4 nodes cluster, this is due to the addition communication overhead between nodes. ONOS's unique design includes multiple instances of RAFT within a single controller instance, this means that, by default, partition tolerance is available from clusters that include more than one controller instance.

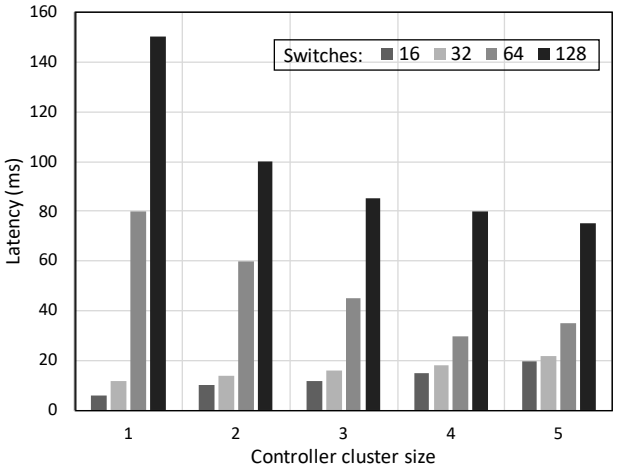


FIGURE 5.2: Controller Performance (Latency) - varying controller-switch ratios

The results presented in Figures 5.1 and 5.2 provide a benchmark for the performance of the distributed controller and the relationship between network size and distributed control. Adding to this, in Section 5.1.5 the impact of multiple controller instances on detection and remediation time in an enterprise environment is explored.

5.1.4 Attack Detection/Protection Latency

The performance of TENNISON against the aforementioned attack scenarios is measured by the length of time it takes to detect and subsequently protect against each attack. The measurement is subdivided to capture the time required for each step in the detection and protection process. This is repeated ten times and averaged for a final result. By accurately timing each stage, it is possible to analyse each stage of the detection/protection process and to identify any potential bottlenecks in the system. The attacks (DoS, DDoS, scanning and intrusion) stress different parts of the system. The start point for each measurement was taken from the beginning of the attack.

Figure 5.3 shows the breakdown of the attack remediation. The following sections describe the evaluation of each attack scenario and discuss the breakdown of attack detection/protection time. In the Figures, 'Monitor rule' refers to the time to install the appropriate monitoring, 'App detection' refers to the TENNISON northbound application attack detection time, 'Mirror rule' refers to the time to install the mirroring rule, 'Alert' refers to the time for Snort/sFlowRT to detect the attack and generate an alert, and 'Block rule' refers to the time to install the block rule in the relevant network elements. Note that the time to detect an attack can include the attack execution time for example x packets within y seconds includes $\leq y$ in the measurement. This means that time for an attack to execute is included in the final measurement, this will be discussed per scenario, as appropriate.

5.1.4.1 DDoS

For this attack, a single host is flooded with TCP SYN requests from multiple source IP addresses. The attack is executed using Hping3 with the following configuration: small packet size, SYN flag, random source, and fast sending rate.

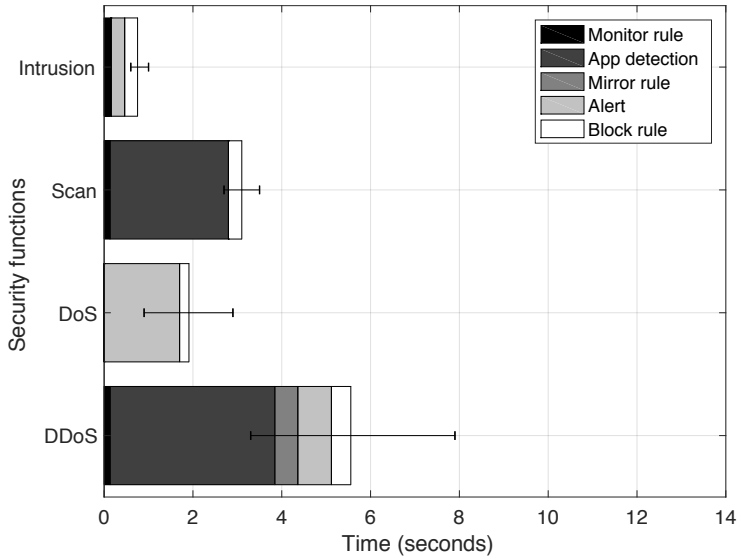


FIGURE 5.3: Attack Remediation Latency - Single Controller

The method of detection for a DDoS attack is reactive as the number of host connections has to be tracked and the traffic then has to be forwarded to a DPI for confirmation. The attack is first detected when the number of unique flows targeting a single host exceeds a configurable threshold within TENNISON's IPFIX DDoS application. For this experiment, the threshold was set to 70 connections from multiple sources to a single destination within 10 seconds. It is important to note that using the environment describe in section 5.1.2, it takes the attacker at least 2.5 s to send sufficient packets to exceed the set threshold defined in both Snort and TENNISON. The traffic is mirrored to Snort where the attack is confirmed via the Snort rule (see Figure 5.4). Once an alert is sent to the coordinator, ONOS is instructed to block traffic for that destination in the network.

```
alert tcp any any -> any 80
(flags: S; msg:"TCP DDoS";
flow: stateless; threshold: type both,
track by_dst, count 70, seconds 10;
sid:10001;rev:1;)
```

FIGURE 5.4: Snort DDoS classification rule

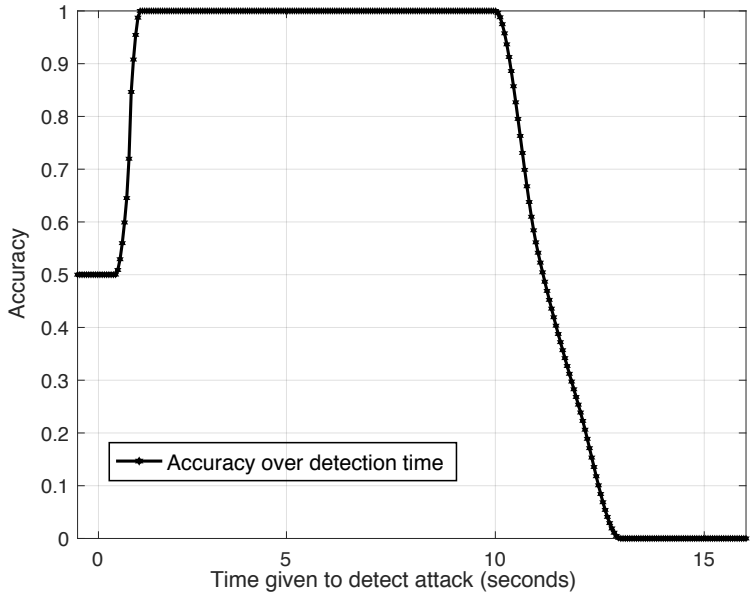


FIGURE 5.5: Accuracy of DDoS detection and remediation

As shown in Figure 5.3, the DDoS attack detection time is the longest of all those measured. This is a consequence of the application logic and the time to complete the attack. However, this combined detection approach ensures a high detection accuracy (low false positive rate), as illustrated in Figure 5.5.

The accuracy of the IPFIX DDoS application was measured with respect to the configured Snort detection threshold. The threshold configured within Snort is designed to detect DDoS SYN Flood attacks. It takes a number of parameters including packet count, which affect the time to detect an attack and the False Positive Rate (FPR). The accuracy measurement is based on Equation 5.1 which describes the relationship between True Positives (TP), False Positives (FP), True Negatives (NP), and False Negatives (FN).

$$Accuracy = \frac{TP + FP}{TP + FP + TN + FN} \tag{5.1}$$

The results in Figure 5.5 show that for a detection time between 0 and 1.7 s, the accuracy is low due to a high False Positive Rate (FPR). However, increasing the Snort detection time, increases accuracy. Of course, this also increases

the overall attack remediation time. Once the detection time is increased beyond 10 s, the accuracy drops sharply. This is due to a high True Negative Rate (TNR), which occurs because the threshold is now too high to detect the DDoS SYN Flood attack. Due to this trade-off, we conclude that a Snort threshold detection time of 2 s will provide an optimum remediation time and high accuracy. Analysis of the error bars in Figure 5.3 highlights a significant variance in latency for the DDoS attack when compared to the other attacks. This fluctuation in results between experiment iterations is due to the host-intensive nature of the attack, which causes the attacker to fluctuate the number of packets per second being sent, thus changing the time until thresholds are met within the system.

5.1.4.2 Scanning

For this attack, a single host scans 200 ports on another host on the network. Nmap is used to perform this network scan. The method of detection for a port scan is reactive as the number of ports has to be tracked. The Tennison northbound port scan application tracks the number of ports accessed across all hosts on the network. Once the number of ports between two hosts exceeds the configured threshold within the defined period, the source of the attack is blocked. Similar to the Tennison IPFIX DDoS application, the majority of detection time is attributed to the gradually increasing threshold within the application logic to determine whether or not the traffic is malicious. The results for the port scan are also in Figure 5.3.

5.1.4.3 Intrusion

For this exploit, a backdoor version (June 2011, 2.3.4) of VSFTPD was used. The detection method for this exploit is proactive as the required thresholds

```
alert tcp any any -> any 21
(msg:"VSFTPD␣Backdoor"; flow:established,
to_server; content:"USER␣"; depth:5;
content:"|3a␣29|"; sid:2013188; rev:1;)
```

FIGURE 5.6: Snort VSFTPD backdoor classification rule

to detect the attack are pre-installed within TENNISON; a threshold to mirror FTP traffic and a threshold to define the response to the Snort alert. The FTP server is exploited by connecting over a network and using ':)' as the username upon login. This opens a network interface on port 6500 that provides direct shell access to the server's host. For detection of the exploit, a threshold must be inserted within TENNISON, notifying it that there is an exploitable server on the network. Then as the attacker logs in, traffic is redirected to Snort which reads the payload, scanning for ':)' as the username. If detected, an alert is sent to TENNISON, which in turn blocks the attacker.

Figure 5.3 shows that this attack is the quickest to detect. This fast detection time is due to the proactive nature of the detection method with pre-installed thresholds. Furthermore, only one packet is required to detect this attack, the login packet, whereas the other attacks are detected over time following observation of multiple packets. The results from this attack are indicative of the general performance of the TENNISON framework as the attack exercises the complete security pipeline (i.e. monitoring, redirecting, and blocking) but without the variance included by the DDoS/Port Scan applications, which are dependent on the specific threshold configuration.

5.1.4.4 High-volume DoS

For this attack, a single host is flooded with TCP-SYN requests. The attack is executed using Hping3. An sFlowRT DoS application is configured with

sFlowRT's default threshold to detect network traffic towards any host exceeding 20,000 packets/s. The sFlow sample rate is set to 1:500. Once the threshold is exceeded, an alert is sent to the coordinator. The coordinator then sends a block intent to ONOS for the flow that exceeded the threshold.

In this case, detection time is primarily dependent on the sFlow sampling rate configured in the network elements and the processing speed of sFlowRT.

5.1.5 System Scalability

This section describes the efforts TENNISON makes towards ensuring scalability of monitoring. In order to show that TENNISON will operate in a variety of network sizes and topologies, this section also analyses multiple components of the system and compares the results with statistics from real network traces.

5.1.5.1 Multi-Level Monitoring

The limitations of reactive based SDN applications under extreme traffic volumes, i.e. heavy communication and processing workload on the controller, are highlighted in [24, 56]. To combat this, TENNISON implements multi-level monitoring.

A specific optimisation is also applied to protect the network controller against the effects of traffic flooding scenarios, such as those caused by DDoS attacks. The solution makes use of the TENNISON resource monitor and introduces a thresholding mechanism to scale back the volume of monitoring traffic, when appropriate, to prevent the controller and control plane from becoming overwhelmed by traffic.

5.1.5.2 Distributed Control Cost

As previously highlighted, in order to scale to larger networks, an increased number of SDN controllers will be necessary to manage the additional networking devices and requests from the network. As network state information is shared between all of the ONOS controller instances, this may ultimately lead to an increase in the TENNISON detection and remediation times. To determine the potential impact of multiple controller instances, additional experiments were carried out to measure the time to remediate a DDoS attack (as per the test set-up described in Section 5.1.4.1) in distributed cluster configurations of varying sizes.

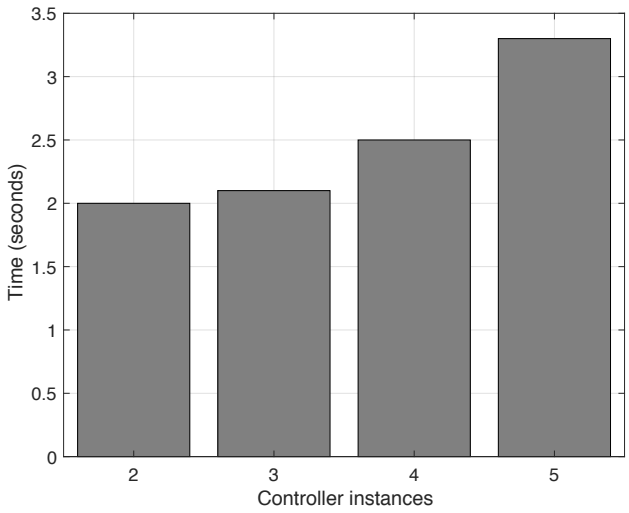


FIGURE 5.7: DDoS Attack Remediation Latency - Distributed Control Cost

Figure 5.7 shows the impact (additional delay) of adding ONOS controller instances to the cluster. The results show that after the second instance is added, the time to detect and remediate gradually increases with each new instance. Note that this increase in the remediation time is not attributed to the design of TENNISON but is a result of the distributed ONOS implementation, which requires state transfer on network events. Research in this area is currently exploring the optimum design for distributed controllers [152].

5.1.5.3 Monitoring Performance Analysis

In this section, the scalability of TENNISON is analysed from the perspective of the cost of per flow monitoring. Policy engine performance, flow monitoring setup time, and RAM usage per monitored flow are explored.

As described in Section 3.5.2, the policy engine in TENNISON stores the thresholds against which the security applications detect an event/attack. The size of this threshold table and the matching algorithm for event detection is a potential source of delay in the system. The policy engine matching delay is therefore analysed. Figure 5.8 shows the time that it takes to process an incoming flow against the threshold table within the policy engine for an increasing volume of policies (thresholds). The results indicate that the incurred delay is minor e.g. 500 ms to test a threshold when the policy engine contains 100K thresholds, with the delay increasing approximately linearly.

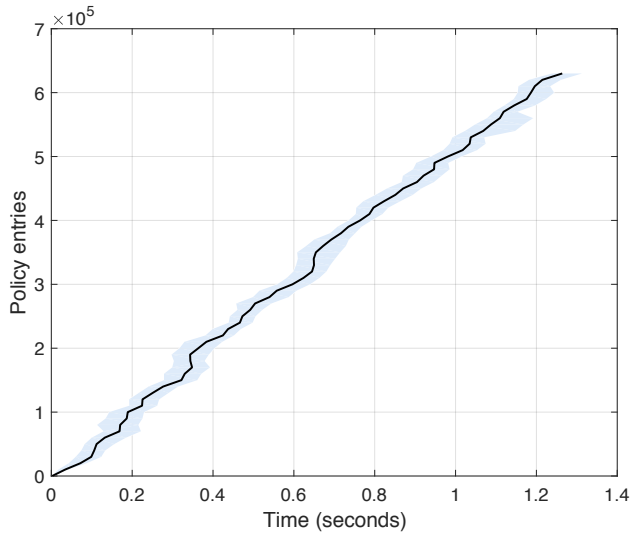


FIGURE 5.8: Policy Engine performance

Figure 5.9 identifies the time it takes to install a set of monitoring rules. This measurement indirectly shows the maximum capacity of newly emerged flows that can be monitored per second. Importantly, this does not describe the ability of the system to manage throughput, but merely shows that this is

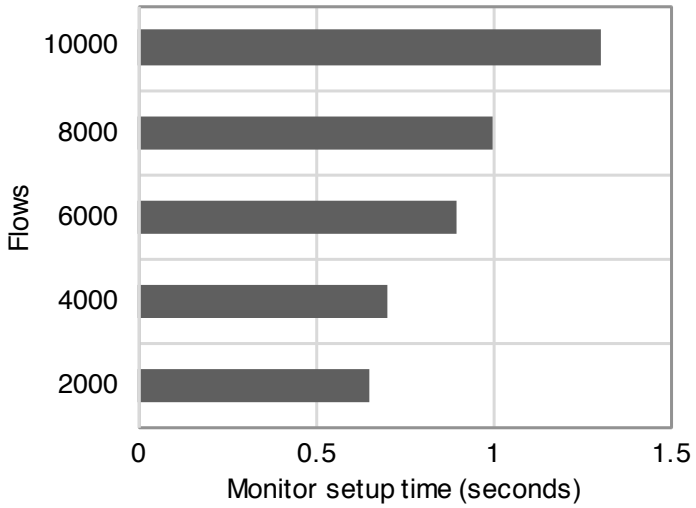


FIGURE 5.9: Monitor setup time

the maximum number of new flows that could be monitored per second. The results show that TENNISON can handle up to 10000 newly introduced flows in bursts and 8000 continuously. For flows that have already been observed and monitoring has already been setup, TENNISON will process packets at line rate as per the capabilities of the networking hardware on which TENNISON is deployed.

Figure 5.10 shows the RAM usage per flow based on actual system readings during the DDoS attack. The memory usage per flow was calculated from the overall system's baseline memory usage and the memory usage measured under different traffic loads. The results show that, on average, each flow monitored consumes around 64 KB of RAM. This means that with 6 GB of RAM, the system can keep track of 100,000 flows.

Based on the results in Figures 5.8, 5.9, and 5.10, TENNISON can scale to a range of network environments. For example, one of Facebook's datacenters manages 500 unique flows per second [165], and in [20], the authors discuss 10 different datacenters that individually support anywhere between 20 to 5000 active flows. In these examples, even in the worst case of all active flows starting

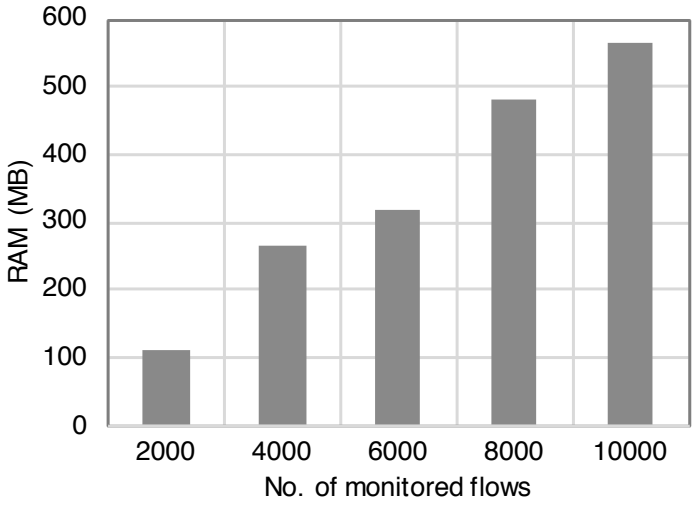


FIGURE 5.10: RAM usage

at the same time, TENNISON would easily monitor each flow, using less then 300 MB of RAM to do so.

The results presented in this section highlight TENNISON's ability to provide timely detection and remediation against four different attack scenarios. In addition, an assessment of realistic security system performance within a distributed SDN cluster has been presented. The use of a distributed SDN cluster is critical to achieve scalability. However, as highlighted in our evaluation, the benefit of increased controller processing speed must be balanced with the speed of attack detection and protection. To the best of our knowledge, this work represents the first analysis of an SDN-based security framework within such an environment.

5.1.6 Impact of Monitoring with TENNISON

The impact of TENNISON on the network depends on a variety of conditions. When traffic is redirected to the NFV overlay, bandwidth on that path is consumed. The latency of traffic on the network is not impacted due to the adoption of mirroring and header monitoring over other methods. However, if the unique

flow count is high enough, it can start to overload the SDN controller, as previously discussed in Section 3.7, this can be resolved with the upcoming P4 data plane technology.

5.1.7 Tiered TENNISON Evaluation

This section evaluates TENNISON operating in tiered mode. It covers three aspects of scale with the system, analysing the improved raw responses per second, the response latency over a number of switches and finally the additional delay added to remediating across domains. The results in this section were conducted on an 18 core server (32 virtual cores) and 200GB of RAM using the TENNISON experimenter environment to automate the process. This proved useful in running multiple instances of ONOS on the same machine as multiple cores could be dedicated to each node.

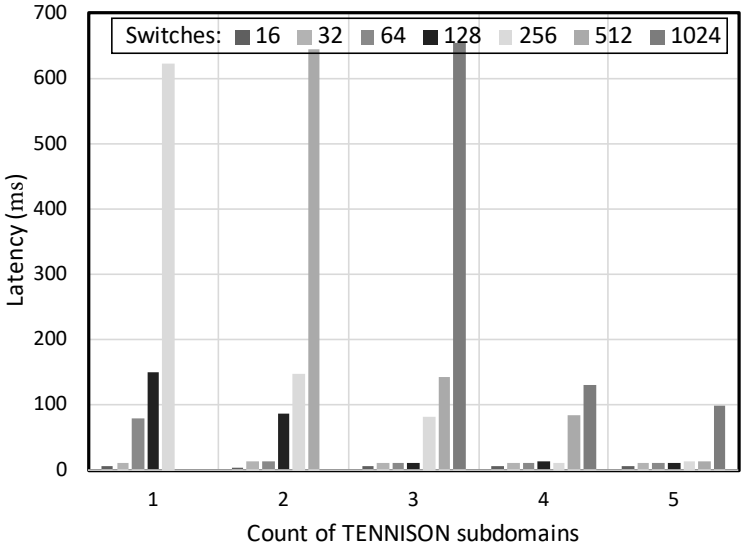


FIGURE 5.11: Tiered TENNISON latency

This provides a baseline for the performance of ONOS operating in a tiered fashion. Similar to the results shown in Figure 5.2, Figure 5.11 shows the round-trip latency between each switch and the controller. These results are

significantly better than with distributed ONOS. However, notice that the forecast (with r^2 value of 95) tapers of after 6 million flows, this can be attributed to the limitations of scale running on a single server. The first set of results under 1 and 2 subdomains do not support the larger network sizes. For future work and to further this scale, the same test could be deployed over multiple linked servers at once.

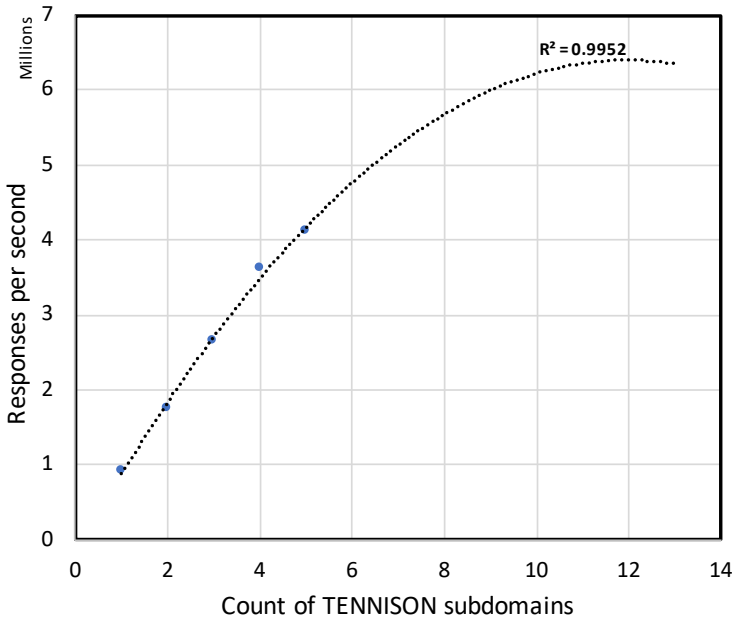


FIGURE 5.12: Tiered TENNISON responses

Figure 5.12 shows further improvement on the previous results shown in Figure 5.2. It is now clear that with TENNISON's method of monitoring, it is possible to scale up to 1000s of networking devices, which was an initial objective of scalability for TENNISON and the thesis goals as detailed in Section 1.4.

As with the other results in scalability, Figure 5.11 shows that there are diminishing returns for scalability when increasing the number of subdomains. This is clear in the difference subdomain count between 4 and 5 where latency is only slightly improved upon. This can be attibuted to the limits of the available hardware on and not a limitation of the scalability of the architecture.

To test the ultimate purpose of tiered TENNISON (sharing of network monitoring information across domains), one of the attack scenarios was conducted across two domains. This included one domain which did not have the port scan application enabled, and another that did. A port scan was performed across the two domains, a trigger was caused on the domain operating the port scan detection application that sent an alert across to the inter-domain manager. Compared to the results of testing the port scan in a single domain, there was an additional 1.2 second delay. However, this is attributed to the additional holding delay, which is configured in the tiered manager GUI. Furthermore, these results are consistent with the matching delay within the policy engine, adding these together plus network delay brings us to a similar result. This can therefore confirm that at the prototypes current state, this is the delay that can be expected for cross domain remediation.

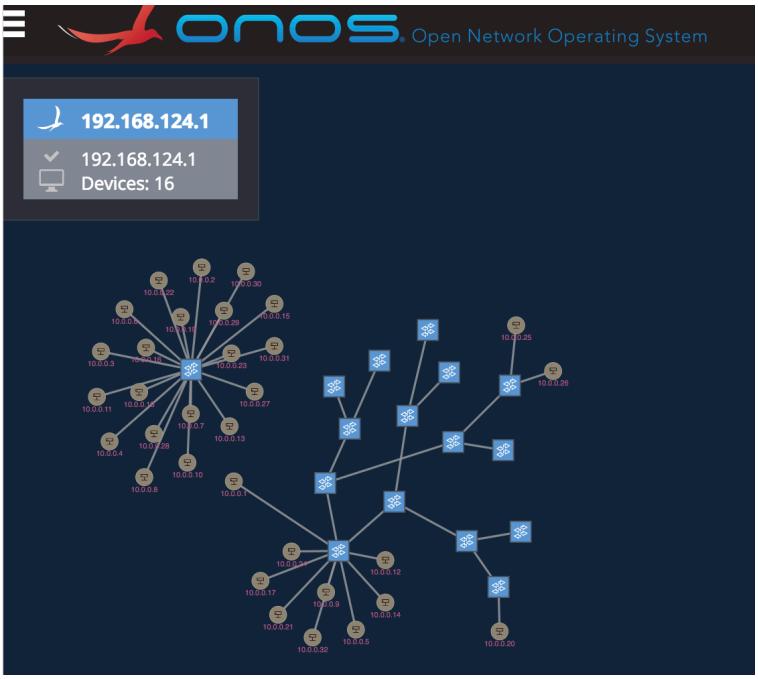


FIGURE 5.13: ONOS GUI with Tiered Operation

Figure 5.13 shows the ONOS GUI in operation with Tiered TENNISON, two instances of ONOS each controling 16 switches of a simple 32 switch flat network. As the network state is not shared at the controller level, the other

side of the network appears as a single switch and 16 hosts connected to the rest of the network.

5.1.7.1 Tiered TENNISON Summary

Driven by the previous results on distributed and centralised TENNISON deployments, the design was adapted to support a tiered architecture. The results in this document describe a clear benefit in separating network management out into multiple SDN islands, showing that SDN and network monitoring can scale-out to thousands of networking devices.

5.1.8 Comparative Design Evaluation

This section compares the difference between Single, Distributed and Tiered operation with TENNISON shows the difference in number of servers verses performance. These results are generated using the same technique that ONOS use to benchmark their controller [139], using cbench [178] to exercise each controller node.

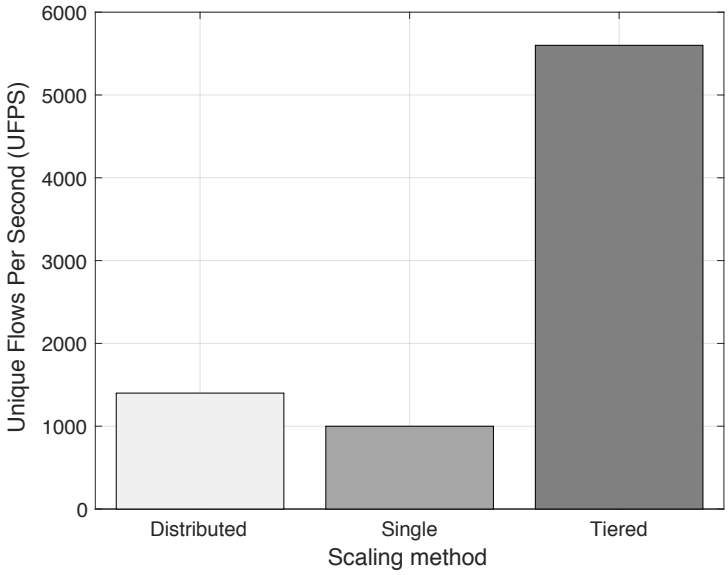


FIGURE 5.14: Tiered vs Distributed vs Single TENNISON

Technology	UFPS	Setup overhead	Setup latency
P4	Line-rate [51]	Digest size	<1ms [51]
OpenFlow	8000 [Appendix A]	Packet size	>10ms [139]

TABLE 5.3: Performance Difference Between OpenFlow and P4

Figure 5.14 shows the difference in scalability between single, distributed, and tiered deployments. From this, it can be seen that from left to right, each method has greater scalability, however, it is important to note that this comes at a cost of increased deployment and management complexity. In a smaller environment it may still make sense to either a single or distributed deployment depending on the needs to the operator.

5.1.9 P4-Enabled TENNISON

This section details the performance benefits of a P4 design within TENNISON. As mentioned in Section 3.7 on P4 design with TENNISON, P4 offers the potential for significant improvements in performance. Based on P4 and specifically Barefoot's Tofino switch, TENNISON's data plane would have significant performance gains from a P4 versus its current OpenFlow pipeline. Adding to this, at the time of writing, the P4-Enabled Barefoot Tofino switch is one of the world's fastest packet switching hardware, giving a single device the capability to push packets at 6.5tbps all through a P4 pipeline [51].

Summarising the difference in performance between P4 and OpenFlow for TENNISON's security pipeline, the values shown in Table 5.3 show significant improvements in unique flows handled per second, traffic overhead, and flow initialisation latency. This is primarily due to P4 being able to create digests of observed flows and send them to the controller at a periodic interval, instead of sending a message for each unrecognised flow.

5.2 SIREN Evaluation

In this section, the SIREN prototype implementation, and in turn its design is evaluated. Firstly, this section goes through an example of SIREN operating on network provider cost, considering where services should be placed based on economic cost of data transfer. It then analyses how network placement affects the latency various types of virtualised network services, highlighting the importance of placement consideration.

5.2.1 Network Provider Cost

As suggested by [27], the cost of placement of a network function is orthogonal to the cost that creates on the network. Therefore, network provider cost must be taken into account.

Equation 5.2 shows an example of a path cost calculation for rent for a single hour. C represents the cost set by the provider for an amount of network transfer m. A network provider may also wish to cost their network on a per hop basis, where some hops might be more expensive than others. Thus the equation supports a standard fee plus a fee for the path that the service will traverse. These values are entirely configurable within the system. For the purposes of illustration, a cost has been added to the reservation of bandwidth, however these are not necessarily representative of a true cost, as this would likely be different from one network provider to another. C_p *and* C_n represents cost per mbps. For the purposes of illustration for Figure 5.17, the C_p *and* C_n have been assigned a value of 10 cents per mbps.

$$C_l = m * C_p + m * \sum_{n=1}^{path} C_n \tag{5.2}$$

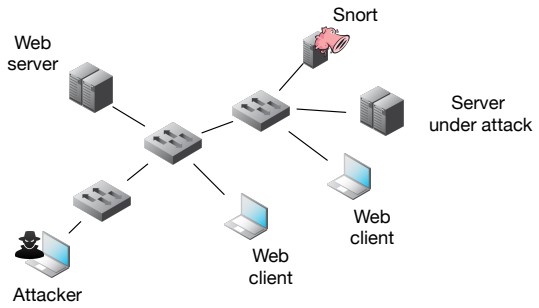


FIGURE 5.15: Experimentation Environment

5.2.2 Experimentation Environment and Scenarios

To demonstrate the purpose of selective placement and to show SIREN in operation, a simplified topology which is representative of a typical ISP topology with access, distribution and core networking layers is used. The network topology, shown in Figure 5.15 consists of 7 switches in the distribution and access layer and another 4 (not shown) in the core which are connected in a straight line, this is were the core NFVIs are located. Each link in this topology is given a 5ms delay to simulate an increased latency the further away on the network the end point is. This topology was realised using the virtual container orchestrator Mininet [196]. Mininet is an emulation tool that enables evaluation, validation and measurement of SDN applications. In order to create the network topology, Mininet instantiates LXCs (Linux containers) to act as hosts and software switches (such as Open vSwitch (OvS)). One of the significant benefits of Mininet (beyond using a simulator to create such a topology) is that each of these containers has a fully fledged and isolated networking stack. This is particularly useful when simulating separate Fog devices where it is desirable to run various network functions, which themselves are real applications.

The testbed runs on a general-purpose server with 256GB RAM, and two Intel Xeon E5-2697v4s totalling 32 cores. The server runs Ubuntu 16.04.3 and

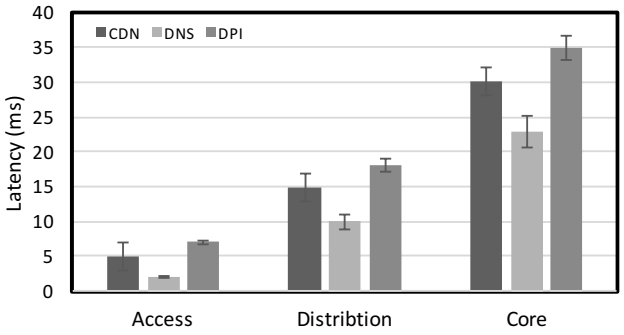


FIGURE 5.16: Latency Results From Mininet Experiment

resources were shared between Mininet (Hosts, OvS) DPI nodes, the SDN Controller (ONOS v.1.10) and SIREN.

In order to emulate traffic on the network, similar to that seen in an ISP network, a simple web poller was used from 24 hosts. These hosts (connected to the access layer) were continuously pulling a 243MB "big bunny" video file from the core network at rates between 10-50mbps. In stressing the network, queues can be seen in action, which can affect the latency of a connection, especially when large amounts of data are being transferred as part of that connection. The impact of this increases as traffic reaches the core were traffic is aggregated.

5.2.3 Analysis

For the following results two aspects were taken into consideration: Latency, which is used to determine the benefit a service will provide at each location; Cost to the network, which is used to determine if, at least from network provider fees, a service is economically viable to run at a certain location. The results on latency were executed 10 times each and then averaged.

Demonstrating the differences in requirements between services, the results in Figure 5.16 show the latency of each service at increasing distances from the client endpoint. This shows that the most bandwidth intensive service, the DPI, increased the latency greater than less demanding services. In terms of service

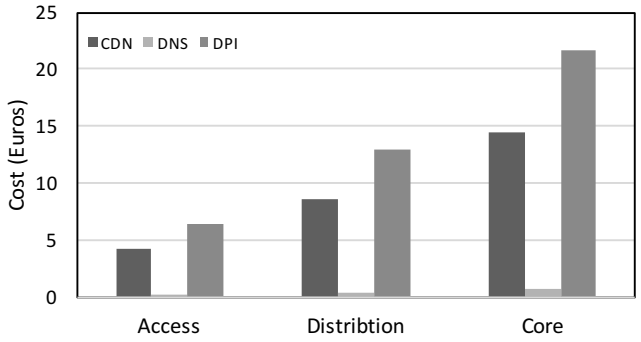


FIGURE 5.17: Example Network Provider Fees Per Month

placement, the results from Figure 5.16 are a motivating factor that the DPI service provider would use to ensure that their service was deployed as close to the customer as possible, in order to reduce costs.

The results in Figure 5.17 shows an example calculated cost of running each network service at different locations within the network. This demonstrates that services that are demanding in terms of bandwidth, are clearly more expensive to deploy network function further away from the customer. Whereas with a service such as DNS, where bandwidth requirements are small, assuming the latency was at an acceptable rate, depending on the wider network policy, it may be economical for a DNS provider to deploy to the distribution or core layers. The primary takeaway of these results from Figures 5.17 and 5.16 is that there are different classes of network services which impact providers and clients differently depending on their location within the Internet.

In summary of the SIREN evaluation, and based on the results in section 3.1, SIREN can orchestrate VNFs across distributed resources, and link them to the data plane, all whilst achieving an improved efficiency both from an economic and quality of service aspects. These elements are key to enabling TENNISON in orchestrating network monitoring across the Cloud-to-Fog Continuum.

5.3 Summary

In this chapter, various aspects of Siren and Tennison were evaluated. Each one of these evaluations was designed to demonstrate the overall systems scalability, responsiveness, and effectiveness at network monitoring and remediation.

Section 5.1.1 has highlighted the differences between Tennison and similar frameworks. The importance of a well designed northbound interface is evaluated here, showing that Tennison requires significantly less LoC for user applications. On top of this, a functional comparison is made between Tennison's capabilities, showing that is has a comparable feature set to similar frameworks.

Furthermore, Section 5.1.7 has shown how different architectures can be used to increase system scalability, demonstrating a significant increase in scale when running Tennison in a tiered architecture. In terms of the future of SDN technology in the monitoring space, this chapter has evaluated potential performance increases with the upcoming P4 data plane technology, again showing that Tennison's scalability and responsiveness can be increased further.

In summary, these evaluations demonstrate the feasibility of effectively monitoring networks using software defined networking at varying levels of scale, satisfying this aims of this thesis. The evaluation here is important for understanding the future of network automation, monitoring, security, and NFV deployments.

Chapter 6

Conclusion and Future Work

Today's network infrastructure is impacted by insufficient monitoring capability, as well as limited ability to react against attacks. Adding to this is the lack of data visibility across the network and continuity in data between monitoring systems. With the integration of 5G, smart cities, connected cars, and IoT, the increase in attack surface and nodes on the network requires a new approach to network monitoring

This thesis has tackled these challenges and has presented a solution which consists of a multi-level distributed monitoring and remediation framework for Software Defined Networks, which is enabled by the Cloud-to-Fog continuum. Together, the two systems in this thesis gather data from multiple sources to build a holistic view of the network, providing dynamic network monitoring and remediation at multiple points within the network. With a unique security pipeline, Tennison offers lightweight visibility across a large number of flows. Supported by Siren, monitoring is automatically distributed throughout the network, utilising a bespoke tunneling solution to efficiently mirror suspicious traffic. The evaluation of Tennison validates its detection capability and illustrates its performance for low latency protection, as well as scaling to large networks.

In summary, TENNISON with SIREN advances the state-of-the art in SDN-based network monitoring, attack detection and protection. TENNISON has been shown to perform effectively against a range of network attacks and provides a flexible framework that can be built upon to develop novel attack detection mechanisms in response to new threats. As technologies mature, future work will focus on further advancing TENNISON's scalability, as well as ability to detect attacks with the use of AI.

6.1 Thesis Contributions

This thesis targets a specific set of challenges within the network monitoring space, motivating technologies and emerging architectures as a solution to a next-generation monitoring framework. The these objectives are highlighted by challenges in current networks, as well as potential benefits of upcoming technologies.

The result of this has been a design and implementation for both an orchestration platform as well as a monitoring framework. Together, these create a solution for scalable and responsive network monitoring in the Cloud-to-Fog.

The following lists the primary contributions present in this thesis:

- Documented experiences and evaluations for different architectural approaches for SDN Monitoring.
- Addressing NFV provisioning, management, and orchestration in the Fog-to-Cloud continuum.
- Though a novel approach, detailing how SDN can be capable of performing network monitoring at scale.

- Using existing specifications such as OpenFlow in a new to create a pipeline that is capable of scalable network monitoring on off the shelf hardware.
- Design for a P4 security pipeline, which provides a new field of work in SDN network monitoring.
- An open proof-of-concept for Cloud-to-Fog NFV management and orchestration with multiple orchestration options.
- An open and feature rich scalable SDN network monitoring and remediation proof-of-concept.

The contributions listed above are a vital step towards managing, monitoring, and securing the next generation of the Internet. The research in this thesis has provided a solution to challenges around monitoring in future networks, where scaling especially towards the edge of the network is of paramount importance.

6.1.1 Thesis Impact

This thesis has presented two proof-of-concept systems that together create a scalable and responsive network monitoring solution, that can be deployed over the Cloud-to-Fog continuum. Both of these frameworks are open source and are available on GitHub [199, 200] and at the time of writing have received considerable traction with over a thousand unique downloads. Within these two frameworks there are various smaller innovations such as the OpenFlow and P4 pipelines, benefiting industry and research in network monitoring. On top of this, this thesis has highlighted the gaps and NFV forwarding technologies. The contributions from this thesis could be used by a wide range of beneficiaries, including ISPs, enterprise organisations, and content providers.

The SIREN proof-of-concept has demonstrated the ability to orchestrate VNFs across the Cloud-to-Fog continuum. Lessons from SIREN contribute to current MANO solutions to realise NFV deployment outside of the Cloud. On top of this, SIREN offers a new business model in leasing resources similar to current Cloud providers, but instead outside of the cloud. As demonstrated in [74], resources can be auctioned off to the highest bidder, overcoming the challenge of deciding the best location to place network services, whilst generating revenue for infrastructure providers.

The TENNISON framework would also be compatible with a new form of business model, and could be deployed as a package by an ISP to various clients. The information from tiered TENNISON could then come back to a SoC where the network can be analysed and managed centrally. This could also be used to monitor the ISP's own network. Alternatively, a company may decide to deploy their own private TENNISON configuration bespoke to their setup.

The thesis also contributes a testing platform, which can be used by researchers to automate the testing of network monitoring and Cloud-to-Fog orchestration systems. Rather than having to develop and build the underlying policy or provisioning engine, this work provides a common solution in which a researcher can build and modify the behaviour of the network monitoring system through the NBI without the burden of extensive development. This significantly reduces the barrier to entry for research and development in network security and Cloud-to-Fog orchestration.

6.2 Future Work

Recently, there has been clear interest from industry in an SDN monitoring solution that can be deployed throughout the network; since the start of TENNISON and SIREN multiple commercial monitoring systems with a focus on edge

deployment have emerged [52, 17, 31, 106]. As these tools mature, a software defined network monitoring solution will become easier to deploy and use in everyday networks.

The following highlights the primary avenues of research for future work that are related to the main design and development contributions in this thesis: TENNISON and SIREN.

6.2.1 Monitoring with Data Plane Programabiltiy

Programmable data planes are reaching an increased level of maturity, with a number of hardware-based solutions emerging on the market. Driven by the move to programmable hardware, these enable developers to make decisions on how to handle packets in a very flexible and powerful manner on the switch itself. Complementing the programmable control plane offered by technologies such as OpenFlow, it is envisaged that these advances will help to address some of the inevitable performance and latency drawbacks of locating a software-based control plane somewhere other than the switch itself. Fundamentally, investigating whether, by pushing some of this logic into the data plane, it is possible to make decisions on how and where to forward packets at very high data rates. This would be without the need to involve a global controller. This may mitigate some of the outstanding scale and performance concerns of existing SDN technologies by reducing the amount of controller-bound traffic. Programming the behaviour of a data plane does carry implicit drawbacks in that decisions are made with far less visibility of the overall network state and conditions.

As shown in Sections 3.7, and 5.1.8, P4 is an ideal candidate to operate as a replacement for OpenFlow, offering lower overheads and greater capability. As

well as this, P4 hardware implementations are showing support for x86 compute in the packet processing pipeline, opening up the possibility of performing sampled deep packet inspection as well as encryption on the switch within the data plane.

6.2.2 Advancing with the Evolution of Edge Computing

As the vision for edge computing evolves and 5G is deployed, similar to CORSA and Barefoot, more network vendors may consider collocating x86 compute resource in network switches. This will further motivate deploying network functions throughout the network.

This thesis has described multiple approaches to VNF placement. New switches coming to market such as [52, 17] have general compute on the same device with a loop-back channel that links them to the data plane. This technology would allow for greater scale, reduced network monitoring traffic, monitoring encryption, and lower latency.

Since the initial design of SIREN, various edge computing solutions have emerged [106, 52]. These are tools that in the future will undoubtedly assist in deploying network services and network monitoring solutions throughout the network.

6.2.3 Applying Artificial Intelligence

Since the initial design of TENNISON, the world of micro-services and software development has started to move towards a more data centric model. Since the launch of TENNISON new technologies have matured to assist with data enrichment, visualisation, and data analysis. With this, core components of TENNISON could be replaced by industrial grade tools such as Prometheus for collection, Kafka for the data bus, and TensorFlow for AI.

The TENNISON coordinator already has a rich API, which allows statistics and state to be ascertained by other sources. Furthermore, the behaviour and logic in TENNISON can be defined through this interface in real-time. In coupling this together, there is the opportunity for implementing new and novel functionality to further enhance the effectiveness of TENNISON in operation. One method of achieving this would be the application of machine learning techniques. This includes the analysis of traffic within the network to identify malicious behaviour, the identification of new features and emerging attack patterns, and the prediction of threats that impact the longevity of the network. Given the aforementioned API, it would be possible to integrate this functionality without the need to directly modify the internals of TENNISON; in other words, the changes could be decoupled, enabling a logical separation of concerns. Furthermore, due to the architectural design of TENNISON, existing algorithms and implementations of machine learning techniques can be used without the need to fundamentally re-implement them for this specific context. This allows for a concentration on evaluation and the opportunity to understand how and which these approaches can be applied correctly and effectively in the context of network monitoring and remediation.

6.2.4 Extending Network Monitoring Visibility

The integration of host-based sensing and monitoring within the TENNISON platform would greatly increase visibility of the underlying infrastructure. Participating hosts will be able to provide system status information to TENNISON, acting as additional evidence for decisions made within the coordinator. For example, sensing on the host can be integrated into the threshold logic that is at the core of the coordinator behaviour, improving in-network monitoring granularity whilst performing further actions such as heavyweight deep-packet

inspection.

Additionally, this integration provides room for new functionality, as behaviour and observations made on the host can then drive decisions made within the network. This coupling provides close integration between monitoring at different layers, each of which has their own benefits and drawbacks. By combining this with the functionality already present within TENNISON, visibility can be pushed even closer into the network edge, all the way to the end-host itself. Further synergies are anticipated when remediation is considered; where it may not be possible to mitigate or stop an ongoing attack or malicious behaviour on the device itself, the network can instead provide this close to the source. This is only possible with a wider and more converged view possible with an extended TENNISON.

6.2.5 Integration with Maturing NFV Technologies

The area of Software Defined Networking and Network Functions Virtualisation is continually evolving, accruing numerous standards, of which few are widely used.

The NFV forwarding technology used in this thesis made use of VLANs. Whilst this solution meant that the system could operate on physical hardware without changes, it is not standardised or elegant, as a result, VLANs could not be used alongside network monitoring. As mentioned in Section 3.3, there are various efforts towards creating a standardised solution to NFV forwarding, including NSH [161] and SRv6 [81]. When these specifications are implemented in hardware, the systems in this thesis can make use of them.

At the time of designing and implementing TENNISON and SIREN, NFV orchestrators such as OSM or ONAP are not mature enough to implement a full monitoring solution for the Cloud-to-Fog continuum. For example, they still

do not have a solution to VNF forwarding, or distributed and heterogeneous NFVIs. Moving forward, as these NFV MANOs mature and support aspects such as VNF placement, network context, and distirbuted NFVIs, then they could be used to benefit the work in this thesis and could be merged with Siren. This would provide Tennison with the maturity of a production grade orchestrator, potentially offering multiple methods of orchestration that could operate over a variety of infrastructures.

Appendix A

Supplementary results

A.1 ONOS Scaling Analysis

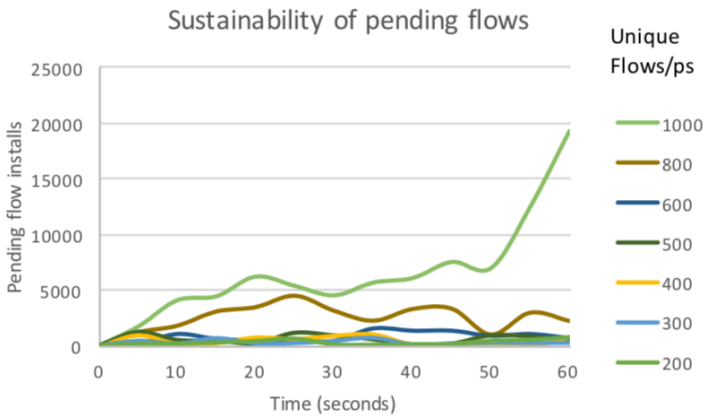


FIGURE A.1: ONOS Pending Flows

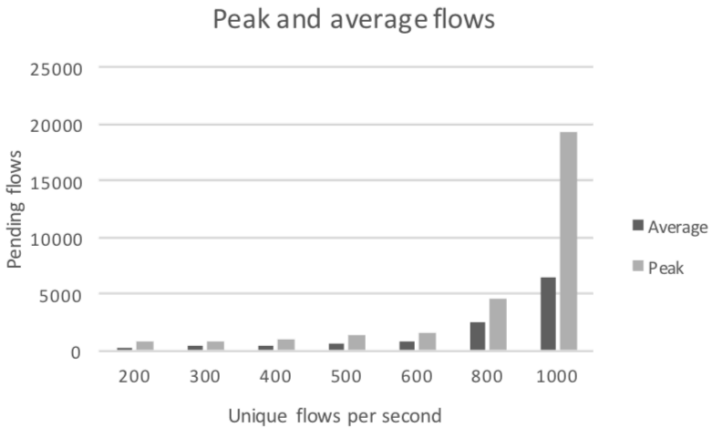


FIGURE A.2: ONOS Peak Flows

Appendix B

Tennison Developer's Guide

Developer's Guide

Lyndon Fawcett
l.fawcett1@lancaster.ac.uk
or
lyndonfawcett@gmail.com

November 9, 2018

Executive Summary

This document is aimed at developers for Tennison, reducing the learning curve of execute, testing, using, and adding to the system. Enclosed is a comprehensive developer's guide on how to use and build on top of the Tennison system. It firstly explains a technical overview of the system, describing relationships between system components and their associated code. Supporting this, snippets from the Git wiki are included in the appendix. After this, it covers a "getting started" guide, explaining how to get the system up and running, including how to build the latest version of ONOS with the Tennison applications. Finally, after understanding Tennison, the guide shows one how to exercise the system through a series of attacks.

Please note that this document is a continuing work in progress and is not necessarily up to date with the codebase.

Contents

1 Technical system overview

This section covers the technical aspects of the system, firstly describing the system as a whole, then moving into detail on specific components and APIs.

The general relationship between the various TENNISON components and their code are explained in Appendix A. These have been split into 4 sections, from the bottom up, this includes: the network, the network controller (ONOS), the coordinator, and northbound applications. In the example, all northbound applications are written in python, though as they are interfaced over rest, new applications could be written in any language[1]. ONOS's northbound applications are written in Java using the Karaf framework Maven build files to perform linking between interfaces and primary implementations of interfaces or abstract classes. The coordinator and the truffle code around snort (not shown on diagram) are both written in python 3.

Appendix B shows the class diagram of the coordinator. A python application that holds a set of thresholds that affect the southbound connection to the network via ONOS and are modified by the northbound interface through remote applications.

Appendix C describes the specific details around the ONOS Flow API. These are otherwise known as onos-tennison-apps in the git project. The REST API here allows one to modify the network, redirecting and blocking traffic. Later in this appendix specifics around implementations of intents are discussed.

API calls offered by TENNISON's northbound interface (known as watson.py in code) are detailed in Appendix D. This also instructs one on how to install an application such that it can be started, stopped, and altered by the coordinator (and the GUI).

For modification of TENNISON's external components one should refer to the collectors. These are shown in Appendix E and describe the communication generated from snort, sFlowRT, and ONOS-IPFIX to the coordinator. A new one of these collectors would have to be created to support any additional monitoring engines or DPIs.

Appendix F describes the REST calls available for modifying rules within snort. The live manipulation of snort rules is managed in the tennison.py class (not shown in the relationship diagram).

[1]Note: if an auto-ran then command would have to be modified to match language, the default is python.

2 TENNISON usage guide

2.1 Getting started (The quick way)

The following guide is assuming you are using vagrant box 11 with the correct vagrant file. It is recommended that this is ran with at least 2GBs of RAM, and more if running at scale. Currently there are various steps required to do to ensure that the system operates as expected.

To launch ONOS locally[2]:

```
$ onos-buck run onos-local clean
in /home/vagrant/onos/
```

Using the keyword "clean" in the command makes sure that the the local instance of ONOS will be refreshed. Running as a service will run an older version and will not work.

To install the ONOS apps:

```
$ ./install_apps_remote
in /home/vagrant/secapp/onos-tennison-apps
```

These apps must be installed each time the ONOS instance is reset.

To launch Mininet with TENNISON and snort:

```
$ sudo ./tennison_dev.py -m -a -p
in /home/vagrant/secapp/topology/tennison_dev/
```

To access the ONOS GUI go to:

```
http://127.0.0.1:8181/onos/ui/index.html
```

To access the TENNISON GUI go to:

```
http://127.0.0.1:8080/
```

2.1.1 TENNISON experimenter

This is the latest tool for running TENNISON and clustered TENNISON. Check this tool out in the tools/dev/ directory.

[2]If ONOS is restarted then Mininet should also be restarted, otherwise ONOS may not detect the Snort instances.

2.2 Getting started (From scratch)

Before reading this section, see install_tennison.sh in the coordinator git repository

2.3 Developer guide

This is a quick guide on how to develop with ONOS inside our vagrant box. ONOS provides various guides on how to do this, however they can be misleading as the documentation is fragmented and based on different versions of the code. https://wiki.onosproject.org/display/ONOS/Developer+Guide

To build ONOS (assuming version 1.10 or higher is used) do:

```
$ onos-buck build onos
in /home/vagrant/onos/
This should take around 5-10 minutes to run, less time if rebuilding.
```

To build the ONOS apps:

```
$ onos-buck-publish-local
```

```
$ mcis
in /home/vagrant/secapp/onos-tennison-apps
```

To install ONOS apps run (ONOS must be running):

```
$ ./install_apps_remote
in /home/vagrant/secapp/onos-tennison-apps
To make sure the apps are launched in the right order, restart ONOS
```

Instructions on how to install used an IDE with ONOS can be found at: https://wiki.onosproject.org/display/ONOS/Importing+ONOS+projects+into+IntelliJ+IDEA

2.3.1 Upgrading the ONOS version

First, copy the SecurityPipeline.java file, the onos-drivers.xml file and the local cell file from the current ONOS installation:

```
$ cp ~/onos/drivers/default/src/main/java/org/onosproject/driver/
pipeline/SecurityPipeline.java ~
$ cp ~/onos/drivers/default/src/main/resources/onos-drivers.xml ~
$ cp ~/onos/tools/test/cells/local ~
```

onos-drivers.xml is used to link together pipelines and devices, it must be modified if testing with something other than OvS. The local cell file tells ONOS the IP address to use when running locally.

Next, remove the old version of ONOS and use git to clone the desired version of the repository from https://github.com/opennetworkinglab/onos :

```
$ sudo rm -r ~/onos
$ git clone https://github.com/opennetworkinglab/onos
in /home/vagrant/
```

Finally, replace the files in the new installation with the files we copied from the old installation, source the bash profile and build ONOS:

```
$ mv ~/SecurityPipeline.java ~/onos/drivers/default/src/main/java/org/onospro
pipeline/
$ rm ~/onos/drivers/default/src/main/resources/onos-drivers.xml
$ mv ~/onos-drivers.xml ~/onos/drivers/default/src/main/resources/
$ rm ~/onos/tools/test/cells/local
$ mv ~/local ~/onos/tools/test/cells/
$ source ~/onos/tools/dev/bash_profile
$ onos-buck build onos
```

This may take some time to complete as ONOS we have to rebuild from scratch.

To update the ONOS apps, look in the /home/vagrant/onos/onos.defs file for "ONOS VERSION", then find and replace the version (assuming ONOS 1.10.4 was the old version) and rebuild the apps using the commands:

```
$ grep -rl '1.10.4-SNAPSHOT' ./ | xargs sed -i
's/1.10.4-SNAPSHOT/{ONOS_VERSION}/g'
$ mcis
in /home/vagrant/secapp/onos-tennison-apps/
```

If the ONOS version is relatively new, the maven repositories may not have been updated yet so the additional command below must be run before building the apps:

```
$ onos-buck-publish-local
in /home/vagrant/onos
```

Manually update onos-drivers.xml. Copy over the line for the security driver to the latest onos-drivers.xml

3 Attack demonstration guide

This section includes attack instructions and explanations which were used for the WP4 demonstration.

3.1 Attack Demo - Port scan

Firstly, enable the ipfix portscan app in the TENNISON GUI.

Exploit Portscan attack on single host (h12)

```
mininet> h8 nmap -n -p- -sA 10.0.0.12
```

Note: -sA (SYN scan) instead of -sT (connect() scan). A connect() scan causes the source port to also change and an actual connection is attempted, this results in a port scan being detected in both directions and therefore both being blocked. -n to prevent nmap resolving addresses (delay at the start)

Detection The ipfix-portscan-app monitors the variance of ports being contacted by a single host. On the detection of an ongoing port scan from a specific source the following ipfix threshold is installed:

```
"subtype": "prefix",
"treatment": "block",
"fields":{"sourceIPv4Address": X.X.X.X},
"treatment_fields":{"sourceIPv4Address": X.X.X.X},
"priority": 10
```

3.2 Attack Demo - DDoS

Firstly, enable the ipfix DDoS app in the TENNISON GUI.

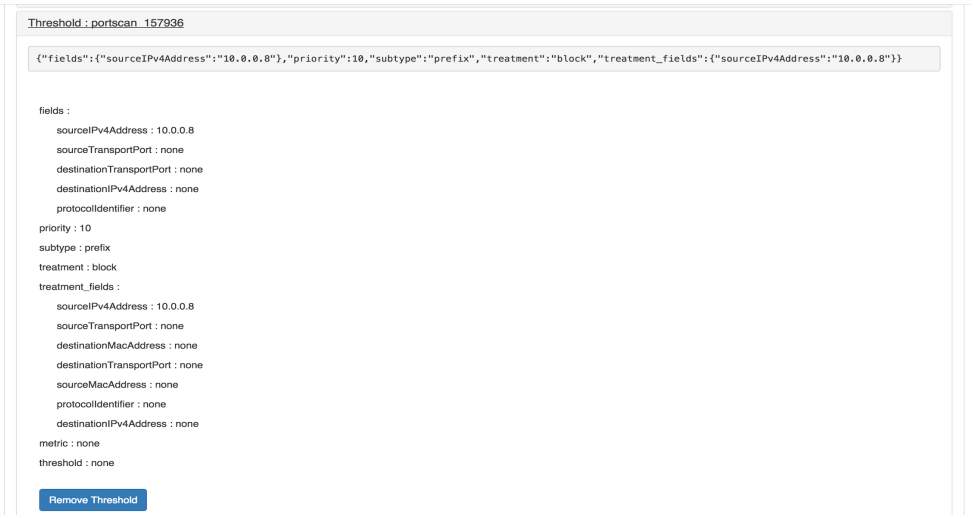


Figure 1: Port scan threshold details

Exploit Running SYN flood DDoS attack on a web server (h15). The distributed aspect is simulated by spoofing random source IP addresses (–rand-source).

```
mininet> h10 hping3 -c 500 -d 120 -S -w 64 -p 80 --fast --rand-source h15
```

For sending at different rates use '-i' and specify the delay between packets.

Detection The distributed nature is first picked up by ipfix-ddos-app which discovers nodes with a high variance of querying nodes. This is then considered a server undergoing a potential DDoS attack and so all traffic querying the server is mirrored by ipfix-ddos-app installing the following (example) threshold:

```
'treatment_fields': {'destinationIPv4Address': '<server>'},
'treatment': 'snort_mirror',
'fields': {'destinationIPv4Address': '<server>'},
'subtype': 'prefix',
'priority': 10
```

The idea what then to use snort to make the distinction between malicious traffic and legitimate traffic, which may not be possible. The extent of snort rules for detecting SYN flood attack only goes as far as:

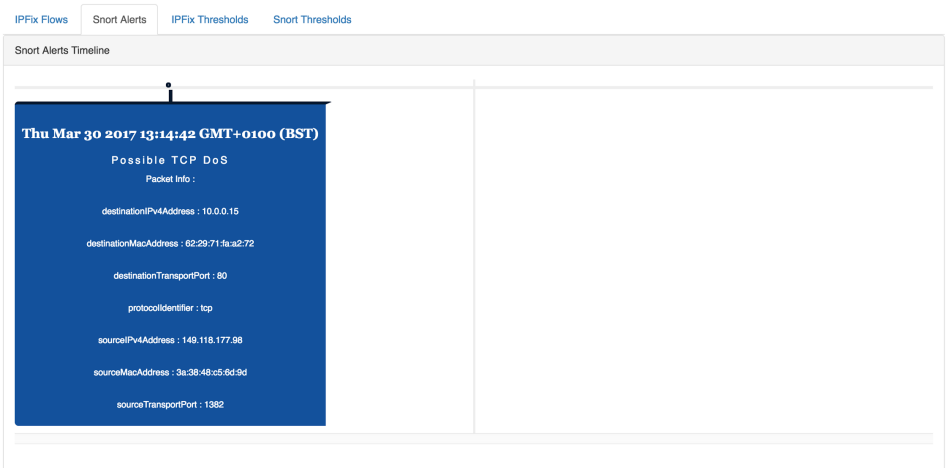


Figure 2: Screen capture of GUI displaying snort DDoS alert

Other Solutions

1. It may be possible to differentiate by some other payload value (insert something unique to the attack traffic).
2. Different DDoS attacks may be possible for snort to separate attack traffic from legitimate triaffic .e.g jolt attack, teardrop, IGMP, dos auth - these all have corresponding rules in dos.rules - after looking into there they all seem specific to windows NT/95 etc.
3. Remediate on the snort alert, would cause all traffic to be blocked and therefore enforcing the denial of service. The following snort threshold would be used to match the snort alert and then block the destination IP address.

3.3 Attack Demo - Intrusion detection

No app is required for detection of this exploit (though one could be made to enhance detection).

Exploit To demonstrate intrusion detection, we attempt to open the backdoor in the exploitable vsftpd server.

Start ftp server:

```
mininet> h5 vsftpd &
```

Attempt the exploit:

```
mininet> h20 ftp h5
Name (10.0.0.5:vagrant): x:)
Password: any
$ nc 10.0.0.5 6200
```

Once the ftp client hangs the backdoor will be open on port 6200, access with:

Detection We assume that we know there is an FTP server on the network so we mirror all port 21 traffic. Alternatively, to be more precise mirroring can be done on the server itself. For information on what these look like, go to the IPFIX thresholds tab on the GUI or look in examples/threshold.yaml.

Attempting the exploit will the trigger the following snort rule:

```
alert tcp any any -> any 21 (msg:"VSFTPD␣Backdoor";
flow:established,to_server; content:"USER␣"; depth:5;
content:"|3a␣29|"; distance:0; sid:2013188; rev:1;)
```

Note: the ftp client seems to be miss-calculating the checksum on USER and PASS packets (the packets we want to detect). By default snort does not consider packets with invalid checksums, so the following config is needed (within rules file):

The snort alert is then matched with the following snort threshold which blocks all communication from the host attempting the exploit, therefore blocking the backdoor. The $ symbol takes the sourceIPv4Address from the alert message (taken from the packet that triggered the alert).

```
alertmsg: 'VSFTPD Backdoor',
priority: 10,
treatment: block,
treatment_fields: {sourceIPv4Address: $}
```

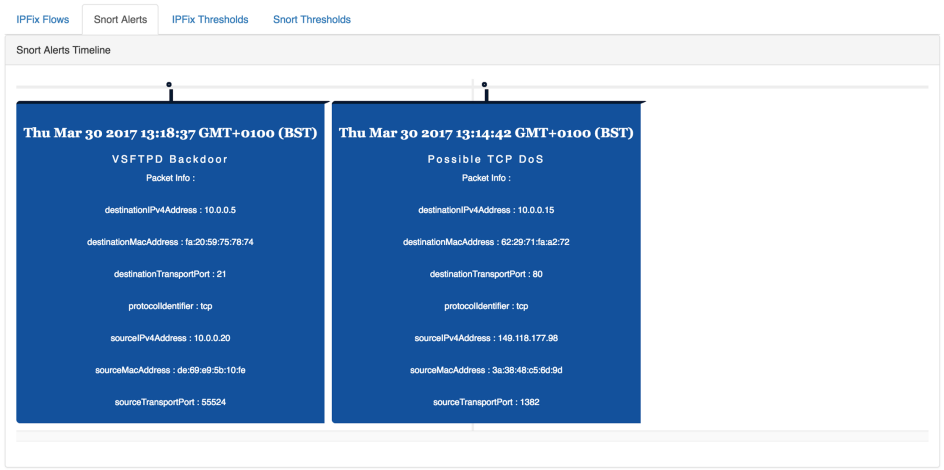
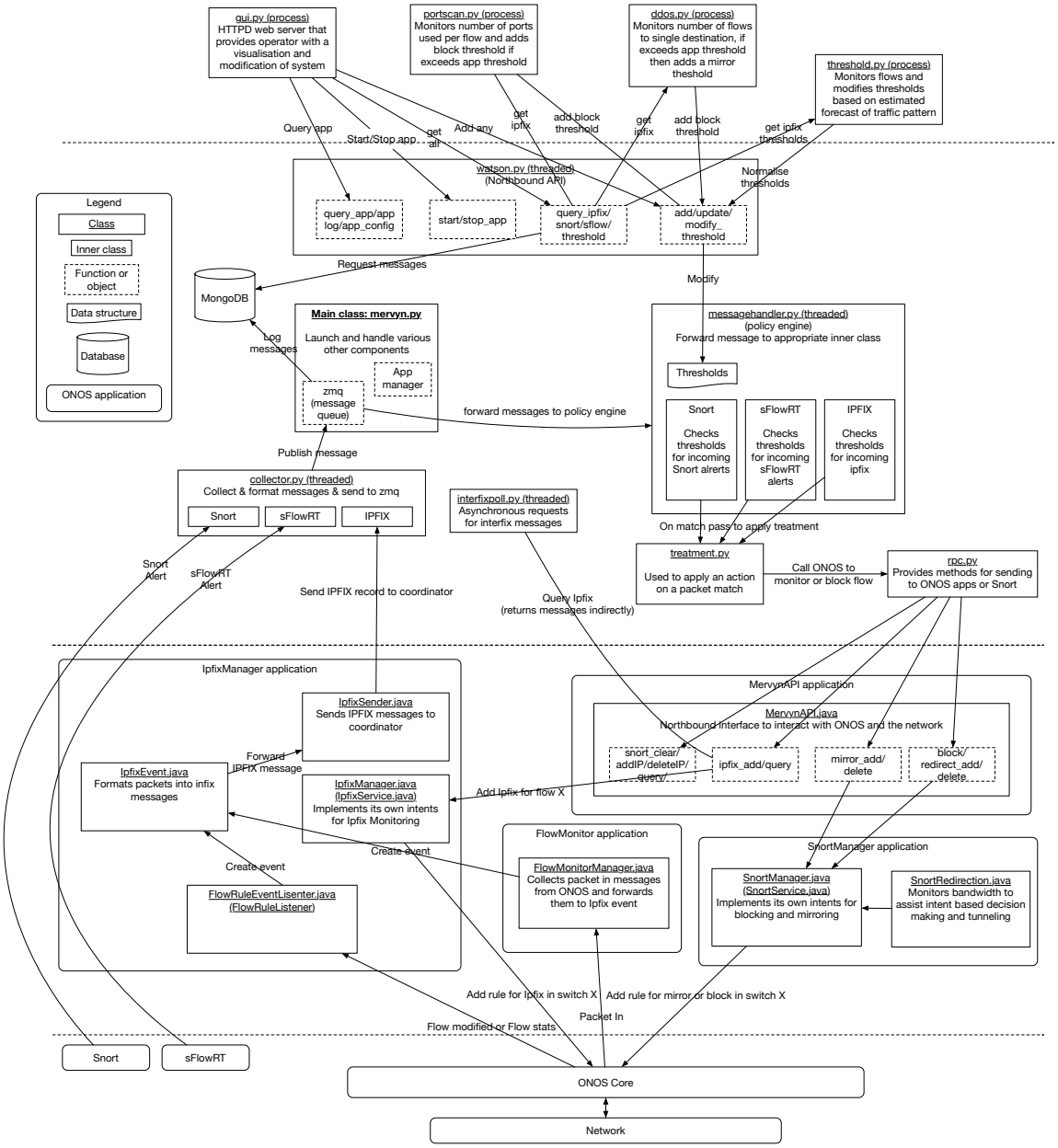
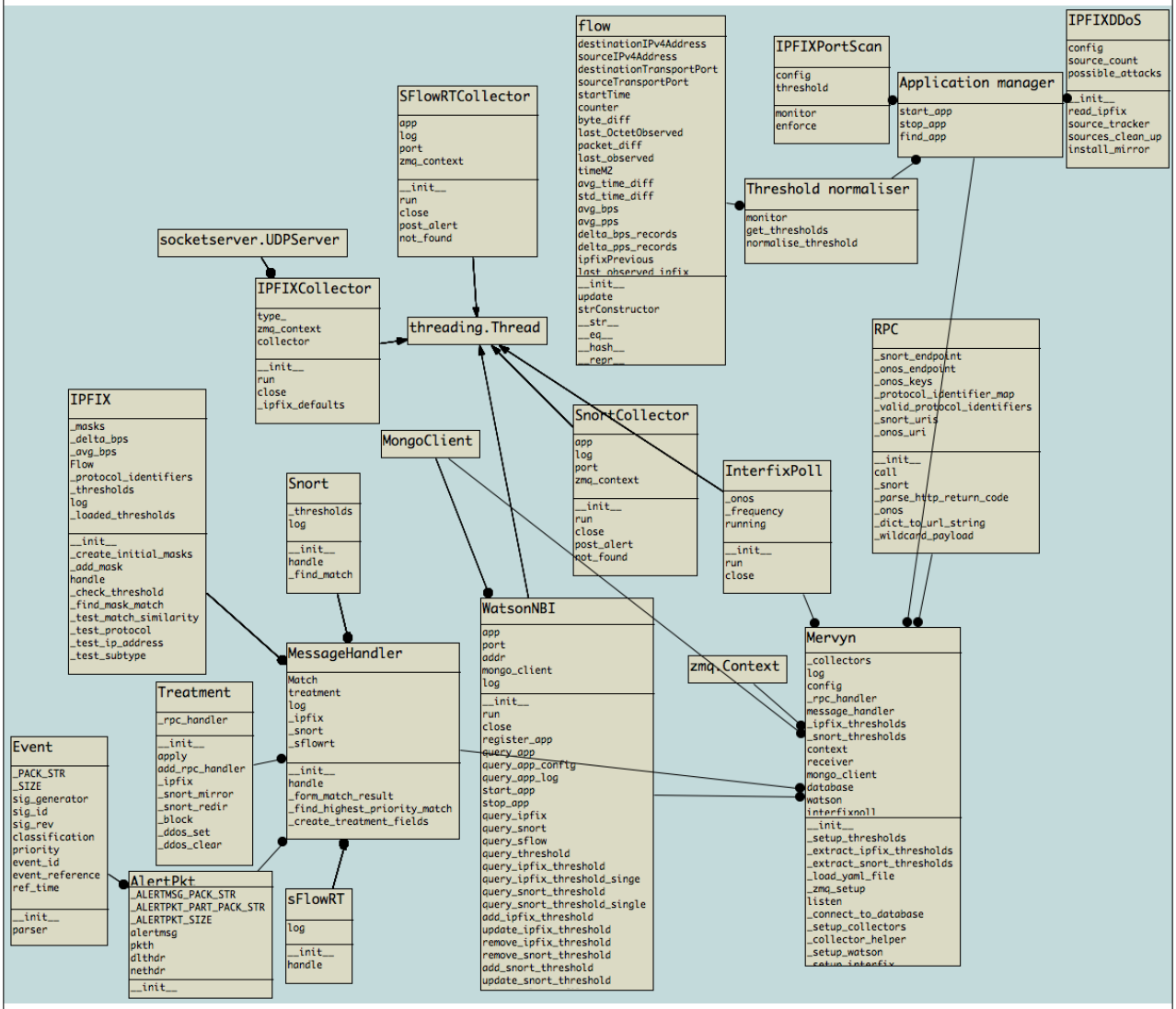


Figure 3: Screen capture of GUI displaying snort VSFTP alert

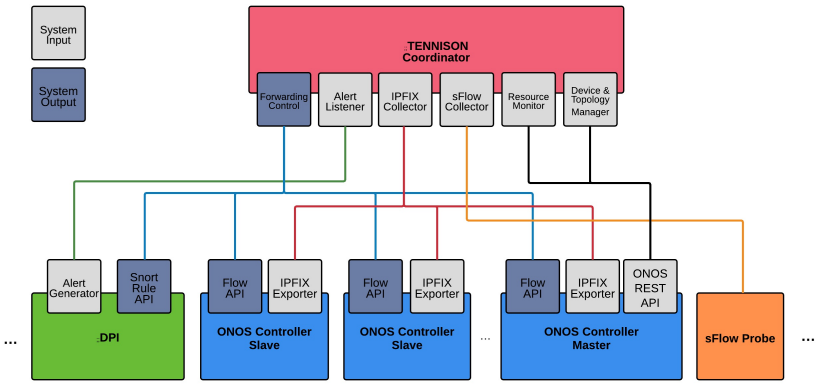
A TENNISON technical overview



B Class diagram of coordinator



C TENNISON flow API



TENNISON Overall Architecture (Modified with additional DPI input API)

ONOS Flow API

The coordinators access into ONOS for monitoring, remediation and snort management. Building on top of the Mervyn ONOS application (mervynapi)

Endpoint: http://onos:8181/mervyn

Monitoring

Path	Type	Description	Parameters
/ipfix/add/ `<saddr>` / `<sport>` / `<daddr>` / `<dport>` / `<protocol>`	GET	Adds an IPFIX monitoring intent	

			`<saddr>` - IP source address. e.g. 192.168.1.1. `<sport>` - Transport Source Port. e.g. 47393. `<daddr>` - IP destination address. e.g. 192.168.1.2. `<dport>` - Transport Source Port. e.g. 80. `<protocol>` - Transport Protocol. e.g. {TCP, UDP, ICMP, ICMP6} **Wildcards not permitted**
/ipfix/query	GET	Triggers interfix messaes to be sent that match the specified flow.	None
/ipfix/query/flow/{saddr}/{sport}/{daddr}/{dport}/{protocol}	GET	Query a specific flow and send the necessary interfix message.	`<saddr>` - IP source address. e.g. 192.168.1.1. `<sport>` - Transport Source Port. e.g. 47393. `<daddr>` - IP destination address. e.g. 192.168.1.2. `<dport>` - Transport Source Port. e.g. 80. `<protocol>` - Transport Protocol. e.g. {TCP, UDP, ICMP, ICMP6} **Wildcards permitted**
/redirect/add/{saddr}/{sport}/{daddr}/{dport}/{protocol}	GET	Adds redirection monitoring intent	5-tuple flow
/redirect/delete/{saddr}/{sport}/{daddr}/{dport}/{protocol}	GET		

		Removes all redirection rules that match the given flow from every switch.	`<saddr>` - IP source address. e.g. 192.168.1.1. `<sport>` - Transport Source Port. e.g. 47393. `<daddr>` - IP destination address. e.g. 192.168.1.2. `<dport>` - Transport Source Port. e.g. 80. `<protocol>` - Transport Protocol. e.g. {TCP, UDP, ICMP, ICMP6} **Wildcards permitted**
/mirror/add/{saddr}/{sport}/{daddr}/{dport}/{protocol}	GET	Adds mirror monitoring intent	`<saddr>` - IP source address. e.g. 192.168.1.1. `<sport>` - Transport Source Port. e.g. 47393. `<daddr>` - IP destination address. e.g. 192.168.1.2. `<dport>` - Transport Source Port. e.g. 80. `<protocol>` - Transport Protocol. e.g. {TCP, UDP, ICMP, ICMP6} **Wildcards permitted**
/mirror/delete/{saddr}/{sport}/{daddr}/{dport}/{protocol}	GET	Removes mirror rules that match the given flow from every switch.	`<saddr>` - IP source address. e.g. 192.168.1.1. `<sport>` - Transport Source Port. e.g. 47393. `<daddr>` - IP destination address. e.g. 192.168.1.2. `<dport>` - Transport Source Port. e.g. 80. `<protocol>` - Transport Protocol. e.g. {TCP, UDP, ICMP, ICMP6} **Wildcards permitted**

Remediation

Path	Type	Description	Parameters
/block/add/{saddr}/{sport}/{daddr}/{dport}/{protocol}	GET	Adds block remediation intent.	`<saddr>` - IP source address. e.g. 192.168.1.1. `<sport>` - Transport Source Port. e.g. 47393. `<daddr>` - IP destination address. e.g. 192.168.1.2. `<dport>` - Transport Source Port. e.g. 80. `<protocol>` - Transport Protocol. e.g. {TCP, UDP, ICMP, ICMP6} **Wildcards permitted**
/block/delete/{saddr}/{sport}/{daddr}/{dport}/{protocol}	GET	Removes block rules that match the given flow from every switch.	`<saddr>` - IP source address. e.g. 192.168.1.1. `<sport>` - Transport Source Port. e.g. 47393. `<daddr>` - IP destination address. e.g. 192.168.1.2. `<dport>` - Transport Source Port. e.g. 80. `<protocol>` - Transport Protocol. e.g. {TCP, UDP, ICMP, ICMP6} **Wildcards permitted**

Snort Management/Discovery

Path	Type	Description	Parameters
snort/add/ `<ip>`	GET	Adds a node to be considered as a Snort instance. This node will then be forwarded traffic using addMirrorRule. Added in phase 2.	`<ip>` - IP Address of snort instance
snort/query	GET	Returns snort instances that have been added and discovered along with their connecting switch and port number	None
snort/del/ `<ip>`	GET	Removes the snort instance from above. Added in Phase 2.	`<ip>` - IP Address of snort instance
snort/clear	GET	Removes all snort instances from above	None

Examples

- `$curl -u karaf -X GET --header 'Accept: application/json' 'http://127.0.0.1:8181/mervyn/ipfix/query'`
- `$curl -u karaf -X GET --header 'Accept: application/json' 'http://127.0.0.1:8181/mervyn/block/add/*/*/*/*/*'`
- `$curl -u karaf -X GET --header 'Accept: application/json' 'http://127.0.0.1:8181/mervyn/block/add/8.8.8.8/*/*/*/*'`
- `$curl -u karaf -X GET --header 'Accept: application/json' 'http://127.0.0.1:8181/mervyn/block/delete/8.8.8.8/*/*/*/*'`
- `$curl -u karaf -X GET --header 'Accept: application/json' 'http://127.0.0.1:8181/mervyn/block/delete/*/*/*/*/*'`
- `$curl -u karaf -X GET --header 'Accept: application/json' 'http://127.0.0.1:8181/mervyn/snort/add/10.0.0.1'`
- `$curl -u karaf -X GET --header 'Accept: application/json' 'http://127.0.0.1:8181/mervyn/mirror/add/*/*/*/*/*'`
- `$curl -u karaf -X GET --header 'Accept: application/json' 'http://127.0.0.1:8181/mervyn/mirror/remove/*/*/*/*/*'`
- `$curl -u karaf -X GET --header 'Accept: application/json' 'http://127.0.0.1:8181/mervyn/mirror/delete/*/*/*/*/*'`

Monitoring and Remediation Intents

The flow API allows monitoring to be inserted into the network in an intent based format. That is, when it is necessary to monitor a flow *somewhere* in the network it is requested that the flow is monitored by only providing the flow information. The placement of the monitoring mechanisms within the network is then choose by ONOS [application] though a series of data driven decisions. Following the'*what* not *how* (or *where* in our case)' paradigm, used by the forwarding intent based forwarding already provided by ONOS.

This approach aims to choose the optimal placement for monitoring whilst also reducing the total number of flow rules in the network when compared to the brute-force method of monitoring everything everywhere.

The same approach is also used for remediation. A description of how the placement is chosen for individual intents is outlined below.

IPFIX Monitoring intent

If using the L2 reactive forwarding application then an immediate suitable placement decision for an IPFIX monitoring rule cannot be made, due to the zero correlation between a L2 forwarding rule (source and destination MAC address) and IPFIX monitoring rule (source and destination, IP address and Transport port). Therefore an approach is used that results in an *eventual* single placement.

Initially the monitoring flow rule is installed on every device. Redundant rules are the removed over time in a garbage-collection-like process, until the flow is being monitored at a single device. The decision to remove rules is based on comparisons of their current activity (number of bytes and packets that have matched a flow); if a monitoring rule has less activity that the equivalent flow at a different device, it is considered redundant and therefore removed.

Redirect & Mirror intent

The number eligible devices for a redirection of mirroring rule are immediately reduced because of the requirement of having a Snort instance directly connected to the device. This set is then further reduced to devices that contain a flow rule that matches* that of the requested flow to be mirrored/redirect.

If this final set of eligible devices is not empty the redirect/mirror rule is install on a single device.

If no eligible devices were found for redirecting/mirroring, the flow is installed on all devices that have a matching* flow, in the hope that a snort instance will be later discovered at one of the devices.

As a last resort, if no devices contain a matching* flow the flow is install everywhere. This scenario should only really occur as a result of an error or manual request with no triggering traffic. With no matching traffic, the monitoring rules will eventually idle timeout.

Note: There could be some work here that uses topology and resource usage information to produce a cost value as an additional input to the decision making process.

Remediation intent

A block rule is installed onto **all** devices that currently contain a flow that matches* the requested flow to be blocked. Using this more inclusive method rather than remediating at a single device prevents a flow avoid the remediation itself by [re]routing.

* [Installed OpenFlow flow rule criteria is a superset of the requested flow]

D TENNISON northbound interface

TENNISON Northbound Interface

Endpoint: http:// `<tennison>` :2401/

Messages and Alerts

Path	Type	Description	Parameters	Returns
/tennison/ipfix/query/ `<app-id>`	GET	Retrieve IPFIX messages that have been received since the last query or registration.	`<app-id>` - Unique application identifier	JSON list of IPFIX message
/tennison/snort/query/ `<app-id>`	GET	Retrieve Snort messages that have been received since the last query or registration.	`<app-id>` - Unique application identifier	JSON list of snort message (excluding packet data)

Thresholds

Path	Type	Description	Parameters	Returns
/tennison/thresholds/query	GET	Retrieve both IPFIX and snort thresholds	None	JSON list of thresholds
/tennison/thresholds/ipfix/query	GET	Retrieve IPFIX thresholds	None	JSON list of thresholds
/tennison/thresholds/snort/query	GET	Retrieve snort thresholds	None	JSON list of thresholds
/tennison/thresholds/ipfix/add/ `<threshold-id>`	POST	Adds an IPFIX threshold	`<threshold-id>` - Unique threshold application identifier. Threshold - Content-Type: application/json. Required fields: subtype, treatment, fields, priority.	200 success 400 No content/Missing required fields

/tennison/thresholds/ipfix/update/ `<threshold-id>`	POST	Updates fields of an IPFIX threshold	`<threshold-id>` - Unique threshold identifier. Threshold - Content-Type: application/json	200 success 400 No content 404 Threshold not found
/tennison/thresholds/ipfix/remove/ `<threshold-id>`	POST	Remove an IPFIX threshold	`<threshold-id>` - Unique threshold identifier.	200 success 404 Threshold not found
/tennison/thresholds/snort/add/ `<threshold-id>`	POST	Adds a snort threshold	`<threshold-id>` - Unique threshold identifier. Threshold - Content-Type: application/json. Required fields: rule, treatment, fields, priority.	200 success 400 No content/Missing required fields
/tennison/thresholds/snort/update/ `<threshold-id>`	POST	Updates a snort threshold	`<threshold-id>` - Unique threshold identifier. Threshold - Content-Type: application/json	200 success 404 Threshold not found
/tennison/thresholds/snort/remove/ `<threshold-id>`	POST	Removes a snort threshold	`<threshold-id>` - Unique threshold identifier.	200 success 404 Threshold not found

Application Management

Path	Type	Description	Parameters	Returns
/tennison/app/register/ `<app-id>`	POST	Register application with tennison to be eligible for message from this point.	`<app-id>` - Unique application identifier	None
/tennison/app/query	GET	Get registered applications.	None	JSON list of application and the last query times
	GET	Get all configurations for application.		

/tennison/app/query/ `<app_id>` /config			`<app_id>` - Application ID	JSON list of application configurations.
/tennison/app/query/ `<app_id>` /log	GET	Get latest log for application.	`<app_id>` - Application ID	JSON containing log. e.g. {"log": "lorem ipsum"}
/tennison/app/start/ `<app_id>`	GET	Start application.	`<app_id>` - Application ID	200 success 400 Application already running 404 Application not installed
/tennison/app/stop/ `<app_id>`	GET	Stop application.	`<app_id>` - Application ID	200 success 400 Application already running 404 Application not installed

Examples

Register application

```
$ curl -X POST http://127.0.0.1:2401/tennison/app/register/ipfix-anom
```

Get registered applications

```
$ curl -X GET http://127.0.0.1:2401/tennison/app/query
```

Get IPFIX messages

```
$ curl -X GET http://127.0.0.1:2401/tennison/ipfix/query/ipfix-anom
```

Get all thresholds

```
$ curl -X GET http://127.0.0.1:2401/tennison/thresholds/query
```

Add new snort threshold

```
$ curl -X POST -H "Content-Type: application/json" -d '{"fields":{"sourceIPv4Address": "10.0.0.200"}, "rule":"alert tcp any any -> any any",
"priority":10, "treatment":"block"}' http://127.0.0.1:2401/tennison/thresholds/snort/add/new-snort-threshold
```

Add new IPFIX threshold

```
$ curl -X POST -H "Content-Type: application/json" -d '{"subtype": "ipfix", "treatment": "snort_mirror", "fields":{"sourceIPv4Address":
"10.0.0.255"}, "priority": 10}' http://127.0.0.1:2401/tennison/thresholds/ipfix/add/new-ipfix-threshold
```

Installing Northbound Applications

Although any application can register itself with the coordinator due to the openness of the REST interface, for an application to be considered as 'installed' and therefore managed by the coordinator there are a few requirements that need to be met.

- Location - each application has its own directory were all relevant files (executables, configs etc.) are located. Application directories should be in mervyn/apps with the directory name being the name of the application. E.g. mervyn/apps/ipfix-ddos-app/
- Main file - applications are launched via a python script `main.py` located in the application directory.
- Registering - when an application registers for monitoring messages (/tennison/app/register/ `<app-id>`) it should use the same ID as the application directory name.
- Config - application configurations should be stored in config.json
- Log file - any textual output from the application should go to output.log

Note: applications will not be launched with the working directory within the application directory, so avoid using relative file paths. An example of an applications directory structure is shown below.

```
mervyn
|    apps
|    |    ipfix-ddos-app
|    |    |    main.py
|    |    |    config.json
|    |    |    output.log
|    |    |    ipfixddos.py
|    |    another-application
```

E TENNISON collector API

IPFIX Collector

Uses sockets for the standard IPFIX transport method. UDP port 4739

Example pickled IPFIX data


```
{
         "vlanId" : 0,
         "ingressInterface" : 3,
         "subtype" : "ipfix",
         "flowStartMilliseconds":"2017-03-09T10:01:14.373000",
         "sourceTransportPort" : 0,
         "time":"2017-03-09T10:01:59.376000",
        "destinationMacAddress":"a6:49:e0:cc:4d:74",
         "flowEndMilliseconds":"2017-03-09T10:01:59.373000",
         "ipClassOfService" : 0,
         "packetDeltaCount" : 0,
         "egressInterface" : 0,
         "protocolIdentifier" : 1,
         "type" : "ipfix",
         "destinationTransportPort" : 0,
         "sourceMacAddress":"4e:ff:3c:1f:7e:91",
         "sourceIPv4Address" : "10.0.0.2",
         "exporterIPv4Address" : "127.0.0.1",
         "octetDeltaCount" : 0,
         "ethernetType" : 2048,
         exporterIPv6Address":"of:00:00:00:00:00:00:0b",
         "destinationIPv4Address" : "10.0.0.1"
}
```


Snort Alert Collector

REST API Port 8081

e.g. http://coord:8081

Path	Type	Description	Parameters

/snort/alert	POST	Adds snort alert onto the message queue	Content-Type: application/json

Example pickled snort alert


```
{
        "transhdr" : 0,
        "alertmsg" : "{'fields': {'sourceIPv4Address': '10.0.0.3'}, 'rule': 'alert icmp any any -> any any', 'treatment':
'snort_mirror', 'priority': 10}",
        "val" : 0,
        "event" : {
                "event_reference" : 131072,
                "ref_time" : {
                        "tv_sec" : 1747356,
                        "tv_usec" : 1477890048
                },
                "sig_rev" : 65536,
                "event_id" : 131072,
                "sig_generator" : 65536,
                "sig_id" : 10651137,
                "priority" : 0,
                "classification" : 0
        },
        "pkt" : {
        "sourceIPv4Address" : "10.0.0.2",
        "destinationIPv4Address" : "10.0.0.1",
        "protocolIdentifier" : "icmp",
        "sourceTransportPort" : 0,
        "destinationTransportPort" : 15
        },
        "pkth" : {
                "caplen" : 382730240,
                "len" : 0,
                "ts" : {
                        "tv_sec" : 447323224,
                        "tv_usec" : 0
                }
        },
        "data" : 973078528,
        "nethdr" : 0,
        "dlthdr" : 0,
        "type" : "snort",
        "time" : ISODate("2017-02-09T17:38:34.170Z")
```

```
}
```

sFlow Collector

REST API Port 8082

e.g. http://coord:8082

Path	Type	Description	Parameters
/sFlowRT/alert	POST	Adds sFlow-RT alert onto the message queue	Content-Type: application/json

Example sFlow-RT alert

```
{
     "agent":"10.80.80.188",
    "dataSource":"49",
    "metric":"ddos_blackhole_target",
    "threshold":20000,
    "value":20065.11977859513,
    "timestamp":1485861345308,
    "thresholdID":"ddos_blackhole_attack",
    "eventID":4,
    "flowKey":"10.80.80.51,external",
    "action":"ddos_set"
}
```

```
{
    "metric":"ddos_blackhole_target",
    "timestamp":1485863215066,
    "flowKey":"10.80.80.51,external",
    "action":"ddos_clear"
}
```

F Truffle API

Snort Rule API (Truffle)

REST API Port 8082

e.g. http://snort:8082

e.g. http://92.168.100.2:8082

Path	Type	Description	Parameters
/snort/rule/clear	POST	Clear all rules from snort	None
/snort/rule/delete	POST	Delete specific rule from snort	Content-Type: application/json { "rules" : ["rule 1", "rule 2] }
/snort/rule/add	POST	Add rule to snort	Content-Type: application/json { "rule" : ["rule 1", "rule 2] }

Bibliography

[1] Cisco Networking Academy. *Connecting Networks Companion Guide*. Pearson Education, 2014.

[2] Cisco Networking Academy. "Connecting Networks Companion Guide". In: Pearson Education, 2014. Chap. 1.1.1 Enterprise Network Campus Design.

[3] Alejandro Aguado et al. "ABNO: A feasible SDN approach for multivendor IP and optical networks". In: *Journal of Optical Communications and Networking* 7.2 (2015), A356–A362.

[4] *Airship: Auction based orchestrator for Siren*. URL: https://github.com/broadbent/airship.

[5] Ali Al-Shabibi and L Peterson. "Cord: Central office re-architected as a datacenter". In: *OpenStack Summit* (2015), pp. 1–38.

[6] Alcatel-Lucent. *The declining profitability trend of mobile data: what can be done?* 2016. URL: http://www3.alcatel-lucent.com/belllabs/advisoryservices/documents/Declining_Profitability_Trend_of_Mobile_Data_EN_Market_Analysis.pdf.

[7] D Scott Alexander et al. "Security in active networks". In: *Secure Internet Programming*. Springer, 1999, pp. 433–451.

[8] *Amazon Web Serices (AWS)*. URL: https://aws.amazon.com.

[9] Mostafa Ammar. "ex uno pluria: The Service-Infrastructure Cycle, Ossification, and the Fragmentation of the Internet". In: *ACM SIGCOMM Computer Communication Review* 48.1 (2018), pp. 56–63.

[10] Jeffrey G Andrews et al. "What will 5G be?" In: *IEEE Journal on selected areas in communications* 32.6 (2014), pp. 1065–1082.

[11] Prof Ann and Mary Joy. "Performance Comparison Between Linux Containers and Virtual Machines". In: Lxc (2015).

[12] *Apache Mesos*. URL: http://mesos.apache.org.

[13] ARM. *Future of ARM edge computing with SDN and NFV*. 2018. URL: https://www.anandtech.com/show/13475/arm-announces-neoverse-infrastructure-ip-branding-future-roadmap.

[14] *Athena DDoS user application*. URL: https://github.com/shlee89/athena/blob/master/athena-tester/src/main/java/athena/user/application/Main.java.

[15] *AT&T) listings on NFV and SDN*. 2017. URL: http://about.att.com/innovation/sdn.

[16] Tarus Balog et al. "OpenNMS". In: *SNMP walker and network manager* (2004).

[17] *Barefoot Deep Insight*. URL: https://www.barefootnetworks.com/products/brief-deep-insight/.

[18] *Barefoot's Tofino Product line*. 2017. URL: https://www.barefootnetworks.com/products/brief-tofino/.

[19] Arsany Basta et al. "Applying NFV and SDN to LTE mobile core gateways, the functions placement problem". In: *Proceedings of the 4th workshop on All things cellular: operations, applications, & challenges*. ACM. 2014, pp. 33–38.

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

[21] Theophilus Benson, Aditya Akella, and David A Maltz. "Unraveling the Complexity of Network Management." In: *NSDI*. 2009, pp. 335–348.

[22] Pankaj Berde et al. "ONOS: towards an open, distributed SDN OS". In: *Proceedings of the third workshop on Hot topics in software defined networking*. ACM. 2014, pp. 1–6.

[23] Samrat Bhattacharjee, Kenneth L Calvert, and Ellen W Zegura. "An architecture for active networking". In: *High Performance Networking VII*. Springer, 1997.

[24] Andrea Bianco et al. "Scalability of ONOS reactive forwarding applications in ISP networks". In: *Computer Communications* 102 (2017), pp. 130–138.

[25] Flavio Bonomi et al. "Fog computing and its role in the internet of things". In: *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*. ACM. 2012, pp. 13–16.

[26] Pat Bosshart et al. "P4: Programming protocol-independent packet processors". In: *ACM SIGCOMM Computer Communication Review* 44.3 (2014), pp. 87–95.

[27] Mathieu Bouet, Jeremie Leguay, and Vania Conan. "Cost-based placement of virtualized deep packet inspection functions in SDN". In: *Military Communications Conference, MILCOM 2013-2013 IEEE*. IEEE. 2013, pp. 992–997.

[28] Ian Briggs et al. *A Performance Evaluation of Unikernels*. Tech. rep. tech. rep, 2014.

[29] *British Telecom (BT) Accelorates Journey to SD-WAN*. 2018. URL: https://www.globalservices.bt.com/en/aboutus/news-press/bt-new-automation-platform-accelerates-journey-to-sd-wan.

[30] Zvika Bronstein. "NFV Virtualisation of the Home Environment". In: *Consumer Communications and Networking Conf. (CCNC)* Ccnc (2014).

[31] BT. *BT Saturn: SATURN (Self-organising Adaptive Technology underlying Resilient Networks)*. 2012. URL: https://ipacso.eu/discover-european-pacs-innovators/cyber-security-innovations-a-selection/reference-case-2.html.

[32] Zheng Cai, Alan L Cox, and TS Ng. *Maestro: A system for scalable openflow control*. Tech. rep. 2010.

[33] Kenneth L Calvert et al. "Directions in active networks". In: *IEEE Communications Magazine* 36.10 (1998), pp. 72–78.

[34] Rafael Cantó Palancar et al. "Virtualization of residential customer premise equipment. Lessons learned in Brazil vCPE trial". In: *it - Information Technology* 57.5 (2015). ISSN: 1611-2776. URL: http://www.degruyter.com/view/j/itit.2015.57.issue-5/itit-2015-0028/itit-2015-0028.xml.

[35] Giuseppe Antonio Carella and Thomas Magedanz. "OpenBaton: A framework for virtual network function management and orchestration for emerging software-based 5g networks". In: *Newsletter* 2016 (2015), p. 190.

[36] Martin Casado et al. "Ethane: Taking control of the enterprise". In: *ACM SIGCOMM Computer Communication Review*. Vol. 37. 4. ACM. 2007, pp. 1–12.

[37] Martin Casado et al. "SANE: A Protection Architecture for Enterprise Networks." In: *USENIX Security Symposium*. Vol. 49. 2006, p. 50.

[38] Ivano Cerrato et al. "Toward dynamic virtualized network services in telecom operator networks". In: *Computer Networks* 000 (2015). ISSN: 13891286. URL: http://linkinghub.elsevier.com/retrieve/pii/S1389128615003485.

[39] Fabricio E Rodriguez Cesen et al. "Design, Implementation and Evaluation of IPv4/IPv6 Longest Prefix Match support in P4 Dataplanes". In: *17º Workshop em Desempenho de Sistemas Computacionais e de Comunicação (WPerformance 2018)*. Vol. 17. 1/2018. SBC. 2018.

[40] Yiyang Chang et al. "Hydra: Leveraging Functional Slicing for Efficient Distributed SDN Controllers". In: *arXiv preprint arXiv:1609.07192* (2016).

[41] Xiao-Fan Chen and Shun-Zheng Yu. "CIPA: A collaborative intrusion prevention architecture for programmable network and SDN". In: *Computers & Security* 58 (2016), pp. 1–19.

[42] *China Telecom (CT) trends with NFV and SDN*. 2018. URL: https://www.ctamericas.com/nfv-sdn-trends-china-need-know/.

[43] Shihabur Rahman Chowdhury et al. "Payless: A low cost network monitoring framework for software defined networks". In: *Network Operations and Management Symposium (NOMS), 2014 IEEE*. IEEE. 2014, pp. 1–9.

[44] CISCO. *Automate and program your network faster*. 2019. URL: https://www.cisco.com/c/en_uk/solutions/software-defined-networking/overview.html.

[45] CISCO. *OpenContrail: an open-source network virtualization platform for the cloud*. 2019. URL: http://www.opencontrail.org.

[46] Seyhan Civanlar and Vikram R Saksena. *Internet NCP over ATM*. US Patent 5,828,844. 1998.

[47] David Clark. "The design philosophy of the DARPA Internet protocols". In: *ACM SIGCOMM Computer Communication Review* 18.4 (1988), pp. 106–114.

[48] Cloudify. *Cloudify, a Cloud and NFV Orchestrator*. 2016. URL: https://cloudify.co.

[49] CLOUDNFV. *Cloud NFV White Paper*. 2015. URL: http://cloudnfv.com/WhitePaper.pdf.

[50] Openfog Consortium and Architecture Working. "OpenFog Reference Architecture for Fog Computing". In: February (2017). URL: https://www.openfogconsortium.com/specification1.0pdf.

[51] CORSA. *Barefoot Tofino 2 performance*. 2018. URL: https://globenewswire.com/news-release/2018/12/04/1661637/0/en/Barefoot-Networks-Unveils-Tofino-2-the-Next-Generation-of-the-World-s-First-Fully-P4-Programmable-Network-Switch-ASICs.html.

[52] CORSA. *Red armor security*. 2018. URL: https://www.corsa.com/red-armor-security/nse7000/.

[53] Corsa. *Software-defined network security (SDNS) lets you economically inspect 100% of your traffic without impacting network throughput performance. All the time.* 2019. URL: https://www.corsa.com/network-security-solutions/.

[54] Peter Cramton et al. "Combinatorial auctions". In: (2006).

[55] Jon Crowcroft et al. "Unclouded vision". In: *International Conference on Distributed Computing and Networking*. Springer. 2011.

[56] Andrew R Curtis et al. "DevoFlow: scaling flow management for high-performance networks". In: *ACM SIGCOMM Computer Communication Review*. Vol. 41. 4. ACM. 2011, pp. 254–265.

[57] Amir Vahid Dastjerdi et al. "Fog computing: Principles, architectures, and applications". In: *arXiv preprint arXiv:1601.02752* (2016).

[58] *Data Plane Programmability the next step in SDN and OpenFlow*. 2017. URL: http://sites.ieee.org/netsoft/files/2017/07/Netsoft2017_Keynote_Bianchi.pdf.

[59] *Deutsche Telekom (DT) listings on NFV and SDN*. 2019. URL: https://www.sdxcentral.com/listings/deutsche-telekom/.

[60] Advait Dixit et al. "Towards an elastic distributed SDN controller". In: *ACM SIGCOMM Computer Communication Review*. Vol. 43. 4. ACM. 2013, pp. 7–12.

[61] *DNS Black hole repositories*. 2017. URL: https://pi-hole.net/.

[62] *Docker API*. URL: https://docs.docker.com/engine/api/v1.24/.

[63] *Docker Swarm mode overview*. URL: https://docs.docker.com/engine/swarm/.

[64] Avri Doria et al. *Forwarding and control element separation (ForCES) protocol specification*. Tech. rep. 2010.

[65] Avri Doria et al. *General switch management protocol (GSMP) V3*. Tech. rep. 2002.

[66] Sevil Dr axler et al. "SONATA: Service programming and orchestration for virtualized software networks". In: *Communications Workshops (ICC Workshops), 2017 IEEE International Conference on*. IEEE. 2017, pp. 973–978.

[67] Nandita Dukkipati. *Rate Control Protocol (RCP): Congestion control to make flows complete quickly*. Citeseer, 2008.

[68] Yehia Elkhatib, Barry Porter, and et al. "On using micro-clouds to deliver the fog". In: (2016).

[69] Yehia Elkhatib et al. "On using micro-clouds to deliver the fog". In: *IEEE Internet Computing* 21.2 (2017), pp. 8–15.

[70] Etsi and Jürgen Quittek. "GS NFV-MAN 001 - V1.1.1 - Network Functions Virtualisation (NFV); Management and Orchestration". In: 1 (2014).

[71] Network Functions Virtualisation ETSI. "Introductory white paper". In: Technical Report, SDN and OpenFlow World Congress. 2012.

[72] OSM ETSI. *Open Source MANO*. 2016. URL: https://osm.etsi.org.

[73] Adrian Farrel, J-P Vasseur, and Jerry Ash. *A path computation element (PCE)-based architecture*. Tech. rep. 2006.

[74] Lyndon Fawcett, Matthew Harold Broadbent, and Nicholas John Paul Race. "Combinatorial Auction-Based Resource Allocation in the Fog". In: (2016).

[75] Lyndon Fawcett and Nicholas Race. "Siren: a platform for deployment of VNFs in distributed infrastructures". In: *Proceedings of the Symposium on SDN Research*. ACM. 2017, pp. 201–202.

[76] Lyndon Fawcett et al. "Tennison: a distributed SDN framework for scalable network security". In: *IEEE Journal on Selected Areas in Communications* 36.12 (2018), pp. 2805–2818.

[77] Seyed Kaveh Fayazbakhsh et al. "Enforcing network-wide policies in the presence of dynamic middlebox actions using flowtags". In: *11th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 14)*. 2014, pp. 543–546.

[78] Nick Feamster, Jennifer Rexford, and Ellen Zegura. "The road to SDN: an intellectual history of programmable networks". In: *ACM SIGCOMM Computer Communication Review* 44.2 (2014).

[79] Wes Felter et al. "An updated performance comparison of virtual machines and linux containers". In: *2015 IEEE international symposium on performance analysis of systems and software (ISPASS)*. IEEE. 2015, pp. 171–172.

[80] Clarence Filsfils et al. *Segment routing architecture*. Tech. rep. 2018.

[81] Clarence Filsfils et al. "SRv6 network programming". In: *Internet-Draft* (2017).

[82] *Flask: Python Microframework for web services*. URL: http://flask.pocoo.org/.

[83] *Fleet: A low-level cluster engine that feels like a distributed init system*. URL: https://coreos.com/fleet/docs/latest/.

[84] Thomas Gamer. "Collaborative anomaly-based detection of large-scale internet attacks". In: *Computer Networks* 56.1 (2012), pp. 169–185.

[85] Kostas Giotis et al. "Combining OpenFlow and sFlow for an effective and scalable anomaly detection and mitigation mechanism on SDN environments". In: *Computer Networks* 62 (2014), pp. 122–136.

[86] *Google Cloud Compute (GCC)*. URL: https://cloud.google.com/compute.

[87] Albert Greenberg et al. "A clean slate 4D approach to network control and management". In: *ACM SIGCOMM Computer Communication Review* 35.5 (2005), pp. 41–54.

[88] OpenFog Consortium Architecture Working Group et al. "OpenFog Reference Architecture for Fog Computing". In: *OPFRA001* 20817 (2017), p. 162.

[89] Taejin Ha et al. "Suspicious Flow Forwarding for Multiple Intrusion Detection Systems on Software-Defined Networks". In: *IEEE Network* 30.6 (2016), pp. 22–27.

[90] Taejin Ha et al. "Suspicious traffic sampling for intrusion detection in software-defined networks". In: *Computer Networks* 109 (2016), pp. 172–182.

[91] Joel Halpern and Carlos Pignataro. *Service function chaining (sfc) architecture*. Tech. rep. 2015.

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

[93] Enrique Hernandez-Valencia, Steven Izzo, and Beth Polonsky. "How will NFV/SDN transform service provider opex?" In: *IEEE Network* 29.3 (2015), pp. 60–67.

[94] Heidi Howard et al. "Raft refloated: Do we have consensus?" In: (2015).

[95] HPE. *Why HPE for SDN*. 2019. URL: https://www.hpe.com/uk/en/what-is/sdn.html.

[96] IBM. *SIEM QRadar: Detect and stop advanced persistent security threats*. URL: https://www.ibm.com/uk-en/marketplace/ibm-qradar-siem.

[97] IEEE. *IEEE 1903.2-2017 - IEEE Standard for Service Composition Protocols of Next Generation Service Overlay Network*. 2017. URL: https://standards.ieee.org/standard/1903_2-2017.html.

[98] Cisco Visual Networking Index. "Forecast and methodology, 2014-2019 white paper". In: *Retrieved 23rd September* (2015).

[99] Sushant Jain et al. "B4: Experience with a globally-deployed software defined WAN". In: *ACM SIGCOMM Computer Communication Review*. Vol. 43. 4. ACM. 2013, pp. 3–14.

[100] Ian Jolliffe. *Principal component analysis*. Springer, 2011.

[101] Murat Karakus and Arjan Durresi. "A survey: Control plane scalability issues and approaches in software-defined networking (SDN)". In: *Computer Networks* 112 (2017), pp. 279–293.

[102] Hyojoon Kim and Nick Feamster. "Improving network management with software defined networking". In: *IEEE Communications Magazine* 51.2 (2013), pp. 114–119.

[103] Hyojoon Kim et al. "The evolution of network configuration: a tale of two campuses". In: *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*. ACM. 2011, pp. 499–514.

[104] Theofrastos Koulouris, Marco Casassa Mont, and Simon Arnell. *SDN4S: Software Defined Networking for Security*. Hewlett Packard Labs, 2017.

[105] Diego Kreutz et al. "Software-defined networking: A comprehensive survey". In: *Proceedings of the IEEE* 103.1 (2015), pp. 14–76.

[106] *KubeEdge*. URL: https://kubeedge.io/en/.

[107] *Kubernetes: Production-grade container orchestration*. 2016. URL: https://kubernetes.io.

[108] Nikolaos Laoutaris, Pablo Rodriguez, and Laurent Massoulie. "ECHOS: edge capacity hosting overlays of nano data centers". In: *ACM SIGCOMM Computer Communication Review* 38.1 (2008).

[109] Aurel A Lazar. "Programming telecommunication networks". In: *Building QoS into Distributed Systems*. Springer, 1997, pp. 3–22.

[110] Aurel A. Lazar, Koon-Seng Lim, and Franco Marconcini. "Realizing a foundation for programmability of atm networks with the binding architecture". In: *IEEE Journal on Selected Areas in Communications* 14.7 (1996), pp. 1214–1227.

[111] Seung-Ik Lee and Shin-Gak Kang. "NGSON: Features, state of the art, and realization". In: *IEEE Communications Magazine* 50.1 (2012).

[112] Seunghyeon Lee et al. "Athena: A framework for scalable anomaly detection in software-defined networks". In: *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE. 2017, pp. 249–260.

[113] Barry M Leiner et al. "A brief history of the Internet". In: *ACM SIGCOMM Computer Communication Review* 39.5 (2009), pp. 22–31.

[114] Yuliang Li et al. "FlowRadar: A Better NetFlow for Data Centers". In: *NSDI*. 2016, pp. 311–324.

[115] Chang Liu, AMehdi Malboubi, and Chen-Nee Chuah. "OpenMeasure: Adaptive flow measurement & inference with online learning in SDN". In: *Computer Communications Workshops (INFOCOM WKSHPS), 2016 IEEE Conference on*. IEEE. 2016, pp. 47–52.

[116] Anil Madhavapeddy and David J Scott. "Unikernels: the rise of the virtual library operating system". In: *Communications of the ACM* 57.1 (2014), pp. 61–69.

[117] Redowan Mahmud, Ramamohanarao Kotagiri, and Rajkumar Buyya. "Fog computing: A taxonomy, survey and future directions". In: *Internet of everything*. Springer, 2018, pp. 103–130.

[118] Margaret et al. "NFV Introductory White Paper". In: *Citeseer* 1 (2012).

[119] Tom Markham and Charlie Payne. "Security at the network edge: A distributed firewall architecture". In: *DARPA Information Survivability Conference & Exposition II, 2001. DISCEX'01. Proceedings*. Vol. 1. IEEE. 2001, pp. 279–286.

[120] A Mayer and S Mansfield. "The third network: Lifecycle service orchestration vision". In: *Technical report, MEF* (2015).

[121] Nick McKeown et al. "OpenFlow: enabling innovation in campus networks". In: *ACM SIGCOMM Computer Communication Review* 38.2 (2008), pp. 69–74.

[122] Rashid Mijumbi et al. "Management and orchestration challenges in network functions virtualization". In: *IEEE Communications Magazine* 54.1 (2016), pp. 98–105.

[123] Rashid Mijumbi et al. "Network function virtualization: State-of-the-art and research challenges". In: *IEEE Communications Surveys & Tutorials* 18.1 (2015), pp. 236–262.

[124] *Mininet: An Instant Virtual Network on your Laptop (or other PC)*. URL: http://mininet.org.

[125] Daniele Miorandi et al. "Internet of things: Vision, applications and research challenges". In: *Ad Hoc Networks* 10.7 (2012), pp. 1497–1516.

[126] *Mirantis MCP*. 2018. URL: https://www.mirantis.com/software/mcp-edge/.

[127] *Mirantis: Run Docker on-premises*. URL: https://www.mirantis.com/.

[128] Jeffrey C Mogul et al. "Devoflow: Cost-effective flow management for high performance enterprise networks". In: *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*. ACM. 2010, p. 1.

[129] Roberto Mora. *Cisco iox: An application enablement framework for the internet of things*. 2016. URL: http://www.cisco.com/c/en/us/products/cloud-systems-management/iox/index.html.

[130] *namespaces - overview of Linux namespaces*. URL: http://man7.org/linux/man-pages/man7/namespaces.7.html.

[131] Netronome. *P4 Programability for the Netronome Agilio SmartNic*. 2018. URL: https://www.netronome.com/blog/p4-programmability-for-the-netronome-agilio-smartnic/.

[132] *NFVRG Charter*. URL: https://datatracker.ietf.org/rg/nfvrg/about/.

[133] Jiseong Noh et al. "Vulnerabilities of network OS and mitigation with state-based permission system". In: *Security and Communication Networks* 9.13 (2016), pp. 1971–1982.

[134] ODL. *OpenDayLight clustering documentation*. 2018. URL: https://docs.opendaylight.org/en/stable-fluorine/getting-started-guide/clustering.html.

[135] ONAP. *Open Network Automation Platform (ONAP) Architecture*. 2017. URL: https://www.onap.org/wp-content/uploads/sites/20/2018/11/ONAP_CaseSolution_Architecture_112918FNL.pdf.

[136] *ONAP Wikipedia*. 2018. URL: https://en.wikipedia.org/wiki/ONAP.

[137] ONOS. *Performance and Scaleout of ONOS 2.0*. 2019. URL: https://wiki.onosproject.org/display/ONOS/2.0-Performance+and+Scale-out.

[138] ONOS. *sflow-rt*. 2019. URL: https://sflow-rt.com.

[139] *ONOS automated benchmarks*. URL: https://wiki.onosproject.org/display/ONOS/System+Test+Plans+and+Results.

[140] *ONOS IPv6 support (Experimental)*. 2017. URL: https://wiki.onosproject.org/display/ONOS/IPv6.

[141] *ONOS P4 Brigade*. URL: https://wiki.onosproject.org/display/ONOS/P4+brigade.

[142] *ONOS REST API*. URL: https://wiki.onosproject.org/display/ONOS/Appendix+B\%3A+REST+API.

[143] *Open-O and ECOMP combine to create ONAP*. 2018. URL: https://www.lightreading.com/nfv/nfv-mano/open-o-ecomp-combine-to-create-onap/d/d-id/730522.

[144] *OpenFlow 1.0 Specification*. 2009. URL: http://flowgrammable.org/sdn/openflow/actions/.

[145] *OpenFlow 1.3 Specification*. 2011. URL: http://flowgrammable.org/sdn/openflow/actions/#tab_ofp_1_3.

[146] *OpenFlow 1.5 Specification*. 2016. URL: https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf.

[147] *OpenFlow Switch Specification Version 1.5.1*. Open Networking Foundation. URL: https://www.opennetworking.org/sdn-resources/technical-library.

[148] *OpenFlow Wikipedia Development*. 2018. URL: https://en.wikipedia.org/wiki/OpenFlow#Development.

[149] OpenStack. *OpenStack VNF Forwarding Graph*. 2019. URL: https://docs.openstack.org/tacker/latest/user/vnffg_usage_guide.html.

[150] *OpenTAM Traffic Analysis and Monitoring*. URL: https://wiki.onosproject.org/display/ONOS/OPEN-TAM\%3A+Traffic+Analysis+and+Monitoring.

[151] OSM. *How to use OpenVIM with OSM*. 2019. URL: https://osm.etsi.org/wikipub/index.php/OpenVIM_installation_(Release_TWO).

[152] Aurojit Panda et al. "SCL: Simplifying Distributed SDN Control Planes." In: *NSDI*. 2017, pp. 329–345.

[153] *Peafowl: High performance Deep Packet Inspection (DPI) framework to identify L7 protocols and extract and process data and metadata from network traffic*. URL: https://github.com/DanieleDeSensi/peafowl.

[154] Larry Peterson. "Cord: Central office re-architected as a datacenter". In: *Open Networking Lab white paper* (2015).

[155] Larry Peterson et al. "Central office re-architected as a data center". In: *IEEE Communications Magazine* 54.10 (2016).

[156] Larry Peterson et al. "XOS : An Extensible Cloud Operating System". In: *2nd International Workshop on Software-Defined Ecosystems* (2015), pp. 23–30.

[157] Xuan Thien Phan and Kensuke Fukuda. "SDN-Mon: Fine-Grained Traffic Monitoring Framework in Software-Defined Networks". In: *Journal of Information Processing* 25 (2017), pp. 182–190.

[158] PICA8. *PICOSő The First Two-in-One Open Network Operating System (NOS) Coupling Full Enterprise Support with Classic SDN*. 2019. URL: https://www.pica8.com/product/#sdn-edition.

[159] Stefano Previdi et al. *Source Packet Routing in Networking (SPRING) Problem Statement and Requirements*. Tech. rep. 2016.

[160] Christofer Price, Sandra Rivera, et al. "Opnfv: An open platform to accelerate nfv". In: *White Paper. A Linux Foundation Collaborative Project* (2012).

[161] Paul Quinn, Uri Elzur, and Carlos Pignataro. *Network service header (nsh)*. Tech. rep. 2018.

[162] *Rancher: Container Orchestration*. URL: https://rancher.com.

[163] Martin Roesch et al. "Snort: Lightweight intrusion detection for networks." In: *Lisa*. Vol. 99. 1. 1999, pp. 229–238.

[164] Sean Rooney et al. "The Tempest: a framework for safe, resource assured, programmable networks". In: *IEEE Communications Magazine* 36.10 (1998), pp. 42–53.

[165] Arjun Roy et al. "Inside the social network's (datacenter) network". In: *ACM SIGCOMM Computer Communication Review*. Vol. 45. 4. ACM. 2015, pp. 123–137.

[166] *RYU Docs. Supports OF versions 1.0 1.1 1.2 1.3 1.4 and 1.5*. 2016. URL: https://media.readthedocs.org/pdf/ryu/latest/ryu.pdf.

[167] Fernando Sánchez and David Brazewell. "Tethered Linux CPE for IP service delivery". In: *Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft)*. IEEE. 2015, pp. 1–9.

[168] Mahadev Satyanarayanan et al. "The case for vm-based cloudlets in mobile computing". In: *IEEE pervasive Computing* 8.4 (2009).

[169] Stefan Schmid. "LARA++ design specification". In: *Lancaster University DMRG Internal Report, MPG-00-03* (2000).

[170] Beverly Schwartz et al. "Smart packets for active networks". In: *Open Architectures and Network Programming Proceedings, 1999. OPENARCH'99. 1999 IEEE Second Conference on*. IEEE. 1999, pp. 90–97.

[171] Sandra Scott-Hayward. "Design and deployment of secure, robust, and resilient SDN Controllers". In: *Network Softwarization (NetSoft), 2015 1st IEEE Conference on*. IEEE. 2015, pp. 1–5.

[172] Sandra Scott-Hayward, Sriram Natarajan, and Sakir Sezer. "A survey of security in software defined networks". In: *IEEE Communications Surveys & Tutorials* 18.1 (2016), pp. 623–654.

[173] Teodora Sechkova, Michele Paolino, and Daniel Raho. "Virtualized Infrastructure Managers for Edge Computing: OpenVIM and OpenStack Comparison". In: *2018 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB)*. IEEE. 2018, pp. 1–6.

[174] Jan Seedorf and Eric Burger. *Application-layer traffic optimization (ALTO) problem statement*. Tech. rep. 2009.

[175] Omar Sefraoui, Mohammed Aissaoui, and Mohsine Eleuldj. "OpenStack: toward an open-source solution for cloud computing". In: *International Journal of Computer Applications* 55.3 (2012).

[176] *SFC with NSH and OVS*. URL: http://www.openvswitch.org/support/ovscon2015/16/1040-elzur.pdf.

[177] Muhammad Shahbaz et al. "Pisces: A programmable, protocol-independent software switch". In: *Proceedings of the 2016 ACM SIGCOMM Conference*. ACM. 2016, pp. 525–538.

[178] Rob Sherwood and Yap Kok-Kiong. *Cbench: an OpenFlow controller benchmark tool*. 2010. URL: https://github.com/mininet/oflops/tree/master/cbench.

[179] Yuan Shi et al. "CHAOS: An SDN-Based Moving Target Defense System". In: *Security and Communication Networks* 2017 (2017).

[180] Seungwon Shin et al. "Avant-guard: Scalable and vigilant switch flow management in software-defined networks". In: *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. ACM. 2013, pp. 413–424.

[181] Seungwon Shin et al. "FRESCO: Modular Composable Security Services for Software-Defined Networks." In: *NDSS*. 2013.

[182] Ankur Singla and Bruno Rijsman. *Day One: Understanding OpenContrail Architecture*. 2013.

[183] *Siren REST API Docs*. URL: https://github.com/Siren-Project/Siren-Provisioner/blob/master/README.md.

[184] Karolj Skala et al. "Scalable distributed computing hierarchy: Cloud, fog and dew computing". In: *Open Journal of Cloud Computing (OJCC)* 2.1 (2015), pp. 16–24.

[185] Joao Soares et al. "Cloud4NFV: A platform for Virtual Network Functions". In: *2014 IEEE 3rd International Conference on Cloud Networking (CloudNet)* (2014). URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6969010.

[186] John Sonchack et al. "Enabling Practical Software-defined Networking Security Applications with OFX." In: *NDSS*. Vol. 16. 2016, pp. 1–15.

[187] Haoyu Song. "Protocol-oblivious forwarding: Unleash the power of SDN through a future-proof forwarding plane". In: *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*. ACM. 2013, pp. 127–132.

[188] Zhiyang Su et al. “FlowCover: Low-cost flow monitoring scheme in software defined networks”. In: *Global Communications Conference (GLOBECOM), 2014 IEEE*. IEEE. 2014, pp. 1956–1961.

[189] José Suárez-Varela and Pere Barlet-Ros. “Reinventing NetFlow for OpenFlow Software-Defined Networks”. In: *arXiv preprint arXiv:1702.06803* (2017).

[190] Jose Suarez-Varela and Pere Barlet-Ros. “Towards a NetFlow implementation for OpenFlow Software-Defined Networks”. In: *Teletraffic Congress (ITC 29), 2017 29th International*. Vol. 1. IEEE. 2017, pp. 187–195.

[191] Subashini Subashini and Veeraruna Kavitha. “A survey on security issues in service delivery models of cloud computing”. In: *Journal of network and computer applications* 34.1 (2011), pp. 1–11.

[192] Norton by Symantec. *Norton Cybercrime Report*. 2013.

[193] Gioacchino Tangari et al. “Decentralized monitoring for large-scale software-defined networks”. In: *Integrated Network and Service Management (IM), 2017 IFIP/IEEE Symposium on*. IEEE. 2017, pp. 289–297.

[194] Gioacchino Tangari et al. “Self-adaptive decentralized monitoring in software-defined networks”. In: *IEEE Transactions on Network and Service Management* 15.4 (2018), pp. 1277–1291.

[195] Yoshiaki Taniguchi et al. “Design and Evaluation of a Proxy-Based Monitoring System for OpenFlow Networks”. In: *The Scientific World Journal* 2016 (2016).

[196] Mininet Team. *Mininet: An instant virtual network on your laptop (or other PC)*. 2012.

[197] David L Tennenhouse and David J Wetherall. "Towards an active network architecture". In: *ACM SIGCOMM Computer Communication Review* 26.2 (1996), pp. 5–17.

[198] David L Tennenhouse et al. "A survey of active network research". In: *IEEE communications Magazine* 35.1 (1997), pp. 80–86.

[199] *TENNISON: SDN Monitoring and Remediation Framework*. 2018. URL: https://github.com/SDN-Security/TENNISON/.

[200] *Web scraped CPE motivation results and generation*. 2017. URL: https://github.com/lyndon160/CPE-Scraper.git.

[201] *The importance of network monitoring in 5G*. 2018. URL: https://blog.mobile-network-testing.com/total-cost-of-ownership/monitoring-network-performance/.

[202] *The Zeek Network Security Monitor*. URL: https://www.zeek.org.

[203] *TM Forum. ZOOM (Zero-Touch Orchestration Operations and Management)*. URL: https://www.tmforum.org/collaboration/zoom-project/.

[204] Amin Tootoonchian, Monia Ghobadi, and Yashar Ganjali. "OpenTM: traffic matrix estimator for OpenFlow networks". In: *International Conference on Passive and Active Network Measurement*. Springer. 2010, pp. 201–210.

[205] Pang-Wei Tsai et al. "Network Monitoring in Software-Defined Networking: A Review". In: *IEEE Systems Journal* (2018).

[206] Muhammad Usman, Aris Cahyadi Risdianto, and JongWon Kim. "Resource monitoring and visualization for OpenFlow SDN-enabled multi-site cloud". In: *Information Networking (ICOIN), 2016 International Conference on*. IEEE. 2016, pp. 427–429.

[207] Niels LM Van Adrichem, Christian Doerr, and Fernando A Kuipers. "Opennetmon: Network monitoring in openflow software-defined networks". In: *Network Operations and Management Symposium (NOMS), 2014 IEEE*. IEEE. 2014, pp. 1–8.

[208] Luis M Vaquero and Luis Rodero-Merino. "Finding your way in the fog: Towards a comprehensive definition of fog computing". In: *ACM SIGCOMM Computer Communication Review* 44.5 (2014), pp. 27–32.

[209] Pier Luigi Ventre et al. "Performance evaluation and tuning of virtual infrastructure managers for (micro) virtual network functions". In: *2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*. IEEE. 2016, pp. 141–147.

[210] Verizon. *SDN-NFV Reference Architecture*. 2016. URL: http://innovation.verizon.com/content/dam/vic/PDF/Verizon_SDN-NFV_Reference_Architecture.pdf.

[211] Thorsten Von Eicken et al. "Active messages: a mechanism for integrated communication and computation". In: *ACM SIGARCH Computer Architecture News*. Vol. 20. 2. ACM. 1992, pp. 256–266.

[212] *VSFTPD Backdoor Command Execution (Metasploit)*. 2011. URL: https://www.exploit-db.com/exploits/17491/.

[213] An Wang et al. "Umon: Flexible and fine grained traffic monitoring in open vswitch". In: *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies*. ACM. 2015, p. 15.

[214] Haddad Wassim, Mahkonen Heikki, and Manghirmalani Ravi. *NFV Platforms with MirageOS Unikernels*. Accessed: 2017-02-20. 2016.

[215] *WebSockets definition*. URL: https://en.wikipedia.org/wiki/WebSocket.

[216] Georgios Xilouris et al. “T-NOVA: A marketplace for virtualized network functions”. In: *2014 European Conference on Networks and Communications (EuCNC)*. IEEE. 2014, pp. 1–5.

[217] Kok-Kiong Yap et al. “Taking the edge off with espresso: Scale, reliability and programmability for global internet peering”. In: *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. ACM. 2017, pp. 432–445.

[218] Curtis Yu et al. “Flowsense: Monitoring network utilization with zero measurement cost”. In: *International Conference on Passive and Active Network Measurement*. Springer. 2013, pp. 31–41.

[219] Minlan Yu, Lavanya Jose, and Rui Miao. “Software Defined Traffic Measurement with OpenSketch.” In: *NSDI*. Vol. 13. 2013, pp. 29–42.

[220] Minlan Yu et al. “Scalable flow-based networking with DIFANE”. In: *ACM SIGCOMM Computer Communication Review* 41.4 (2011), pp. 351–362.

[221] Tianlong Yu et al. “PSI: Precise Security Instrumentation for Enterprise Networks”. In: *Proc. NDSS*. 2017.

[222] Adel Zaalouk et al. “Orchsec: An orchestrator-based architecture for enhancing network-security using network monitoring and sdn control functions”. In: *Network Operations and Management Symposium (NOMS), 2014 IEEE*. IEEE. 2014, pp. 1–9.

[223] SIA Zabbix. *Zabbix. The Enterprise-class Monitoring Solution for Everyone*. 2014.

[224] Zhi-Kai Zhang et al. “IoT security: ongoing challenges and research opportunities”. In: *2014 IEEE 7th international conference on service-oriented computing and applications*. IEEE. 2014, pp. 230–234.