Improving Network and Middlebox Resilience with Virtualisation

Lyn Hill, BSc (Hons)
School of Computing and Communications
Lancaster University

A thesis submitted for the degree of
Doctor of Philosophy

February, 2025

Declaration

I declare that the work presented in this thesis is, to the best of my knowledge and belief, original and my own work. The material has not been submitted, either in whole or in part, for a degree at this, or any other university. This thesis does not exceed the maximum permitted word length of 80,000 words including appendices and footnotes, but excluding the bibliography. A rough estimate of the word count is: 50061

Lyn Hill

Improving Network and Middlebox Resilience with Virtualisation

Lyn Hill, BSc (Hons).
School of Computing and Communications, Lancaster University

A thesis submitted for the degree of *Doctor of Philosophy*. February, 2025

Abstract

Modern networks strive to balance performance and resilience in their designs and operations, the former for maintaining a competitive edge and the latter for ensuring continued service during periods of disruption. These goals are not diametrically opposed but are difficult to cater to simultaneously, a problem made more difficult by the use of high-performance hardware solutions known as "middleboxes". These middleboxes limit the applicability and effectiveness of established resilient design practices for the networks they are utilised in, especially in regards to the preservation of state. State is the contents of memory retained by hardware to aid in its operations, and its loss is the cause of observable disruption to end-users. Middleboxes are popular with network operators for their high-performance and ease of use for enacting network policy, but their blackbox design and widespread use have created a distinct vulnerability to disruption. Prior research in this domain has proposed their replacement with network virtualisation/softwarisation, both to enable greater network elasticity and allow for more complex resilience techniques. These proposals have seen limited adoption due to industry prioritising performance scalability over resilience in the name of competitiveness and guaranteeing SLAs, with hardware middleboxes orders of magnitude faster than current virtualisation solutions and unlikely to be replaced in the near future. The popularity of SDN and NFV will continue to rise in industry, but certain network applications will require hardware solutions to fulfil and cannot be replaced through virtualisation. This thesis takes the position that SDN and NFV can instead find use in enhancing the resilience of this existing infrastructure rather than replace it, so that the flexibility of software can be exploited without sacrificing the performance of hardware. These blackbox middleboxes represent a key issue for research: if internal state cannot be observed or extracted, it must be captured or recreated externally through novel means that are sufficiently quick, accurate and reliable for real-world use.

To address this problem, this thesis presents Remediate (REsilient MiddlEbox Defence Infrastructure ARchiTEcture), a state recovery framework that explores multiple approaches to preserving state for middlebox devices with differing degrees of accessibility. This proof-of-concept implementation is divided into two major publications, "Middlebox Minions" and "Katoptron", that each explore different techniques for recreating or transferring state. The first contribution, "Katoptron", targets blackbox hardware by recreating state using traffic filtering and packet

sampling. The second contribution, "MiMi", targets white and greybox software using inserted drivers and logging interpretation respectively. Remediate incorporates these two contributions as its mechanisms for enabling stateful failover in multiple kinds of middleboxes, distributing state in a platform-agnostic and scalable approach using message streaming and datastores. Overall, this framework allows for state recreation and retention across failovers for both hardware and software in any combination or direction. This is especially demonstrated in its viability across multiple popular stateful mechanisms for networking and security, as well as the reduction in traffic necessary to ensure accurate failover by 95% and provide continuation of service without visible disruption.

Publications

The following publications have been generated while developing this thesis, and to an extent has guided the thesis into what it has become:

Lyn Hill, Charalampos Rotsos, Will Fantom, Chris Edwards and David Hutchison (2022). "Improving network resilience with Middlebox Minions". In: *NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium.* doi: 10.1109/NOMS54207.2022.9789819

Lyn Hill, Charalampos Rotsos, Chris Edwards, and David Hutchison (2024). "Katoptron: Efficient State Mirroring for Middlebox Resilience". In: *NOMS 2024-2024 IEEE Network Operations and Management Symposium.* doi: 10.1109/NOMS59830.2024.10575815

Acknowledgments

I would like to acknowledge a number of people whose presence and support was fundamental to me completing this degree. First off, I want to thank my boyfriend Chris for supporting me all these years both emotionally and even assisting with the work, proofreading papers and helping me practice presentations. I spent a long time doing this, longer than I should have, but he was there for me. My friends in Lancaster and especially in my office, Revika, Will, Paul, Taylor, Ben and Ellie, who made staying here worth it after I had finished my bachelors. If it wasn't for Ellie, I wouldn't have even considered returning, or gotten an RA job in the department in the first place. My friends further afield, David, Joseph and Janet, Pete, Murray and Jesper. My family, with my brothers Timothy and Trevor and my mum Rosemary. I can also add my dog Scruffy to this list, for living as long as he has and trucking along at 19 years old.

On the academic level, I have to thank Charalampos (Haris), my supervisor, for having both the patience and willingness to take me on as a PhD and even take a chance on me as an unproven bachelors student for my first research job. I was sitting in an obscure corner of the infolab basement and came to find me just to tell me he was willing to give me a job based on nothing at all. David, my senior supervisor, providing guidance on direction and potential routes to explore both in research and writing. Chris Edwards, my secondary supervisor, for tracking progress. Research wise, a large portion of my initial exploration was inspired by the works of Justine Sherry, if one is reading these acknowledgments to understand where and how my research came about.

Contents

List of Figures

List of Tables

Acronyms

APLOMB Appliance for Outsourcing Middleboxes. 46

ARPANET Advanced Research Projects Agency Network. 7, 29

ASIC Application-Specific Integrated Circuit. 6, 51

CapEx Capital Expenditure. 27

COE Chain Output Equivalence. 45

CoMb Consolidating Middleboxes. 46

CREW Concurrent-Read, Exclusive-Write. 42

CSNET Computer Science Network. 29

EBPF Extended Berkeley Packet Filters. 26

ETSI European Telecommunication Standards Institute. 26

FIB Forwarding Information Base. 52

FTMB Fault Tolerant MiddleBox. 43

HSRP Hot Standby Router Protocol. 44

IETF Internet Engineering Task Force. 18

ISG Industry Specification Group. 27

KPI Key Performance Indicators. 125

LOBUS LOad-Balancing over UnStructured networks. 23

MANO Management and Orchestration. 35

MTBF Mean Time Between Failures. 35

NEP Network Equipment Providers. 69

NFV Network Function Virtualisation. 5, 14, 26

NSFNET National Science Foundation Network. 30

NSS Network Service Support. 46

OF OpenFlow. 23–25

ONF Open Networking Foundation. 5, 23

OpEx Operational Expenditure. 27

PAL Packet Access Logs. 43

PNF Physical Network Function. 35

PoF Point of Failover. 78

QUIC Quick UDP Connections. 139

RIB Routing Information Base. 52

SCADA Supervisory Control And Data Acquisition. 34

SDN Software Defined Networking. 5, 14, 22

SFC Service Function Chains. 26

SLA Service Level Agreement. 5

TCAM Ternary content-addressable memory. 52

VNF Virtualised Network Function. 26

VRRP Virtual Router Redundancy Protocol. 44

Chapter 1

Introduction

Computer networks and the Internet today form the lynchpin of modern communication systems, intersecting with all areas of modern society across the majorities of countries on the globe (Leiner, Cerf, D. D. Clark, Kahn, Kleinrock, Lynch, Jon Postel, Larry G. Roberts, et al., 2009). From social media and content distribution, such as video sharing and messaging, to critical infrastructure such as power grid monitoring and control (Mather, 2018). These networks have become so ubiquitous and intertwined that they are now a fundamental aspect of modern society, with nearly two-thirds of the global population utilising it (Cisco, 2018), a critical infrastructure unto itself that underpins all other systems. Since its inception in the 90s (Leiner, Cerf, D. D. Clark, Kahn, Kleinrock, Lynch, Jon Postel, Lawrence G. Roberts, et al., 1997), the Internet is now relied upon to facilitate a wide range of purposes, both critical and non-critical, and its infrastructure and design has evolved in response to this growing demand (J. M. McQuillan et al., 1977). The challenges that affect it have changed over time as its use cases have evolved (Ingham et al., 2002), with the early Internet focusing on the growth of its scalability and availability of service (L. Roberts, 1978), reflected both in policy and growing infrastructure. Now in its modern incarnation, today's challenges concern performance and reliability in parts of the world where computer networks are now a firmly established critical infrastructure. With so many systems both commercial and non-commercial dependent on the Internet to facilitate their operations, resilience in the face of disruption has become a primary challenge of its design (J. P. G. Sterbenz et al., 2010).

Current design practices focus on developing infrastructures at the network edge in a bid to push functionality outwards from the transit core (Panda et al., 2016). This is done for a number of reasons, including improving perceived performance from customers by minimising hops and latency, as well as reducing the overall volume of traffic that requires transit. This is achieved using technology known as "Middleboxes"; purpose-built hardware built to perform a singular function, used to

deploy new network policies through packet processing (Brim et al., 2002). As a result of this, faults and failures at the edge are now increasingly visible to end-users, leaving it vulnerable to disruption (Sherry and Ratnasamy, 2012). A load balancer misconfiguration in December 2012 caused outages across multiple Google services, with up to 40% of traffic during this period being affected (Brodkin, 2012) despite failsafes designed to minimise the extent of disruption a misconfigured box might cause. In a 2011 survey of 1000 organisations (F5, 2011), 42% of the respondents indicated observable failures of a firewall at the network layer due to DDoS attacks caused disruption to their services. Despite this, middlebox designers and their users have typically prioritised performance over resilience due to the difficulty and costs involved. Like much of the existing network infrastructure, their design has ossified, with expansion in existing hardware favoured over continued innovation in their operations. In spite of this, middlebox resilience and, by extension, the network edge must be assessed in terms of their scalability and resistance to failure scenarios. The proliferation of these devices and their potential limitations further highlight this need. This thesis proposes that an overreliance on existing middlebox solutions may impede the network's overall resilience and pose a risk of exposing failure to end-users, with modern network research such as virtualisation offering a means to overcome this potential gap in network resilience.

1.1 Middleboxes and their evolution

In the early 90s as NSFnet opened up to private and commercial use, businesses turned to the recently emerged Internet service as a new frontier of operations (Leiner, Cerf, D. D. Clark, Kahn, Kleinrock, Lynch, Jon Postel, Lawrence G. Roberts, et al., 1997). This demand inspired innovation with the development of new technologies to handle growing traffic, encouraging further demand. The explosive growth (Sekar, Ratnasamy, et al., 2011) far outpaced any standardisation body for these early building blocks of the network fabric, giving rise to the design of devices we now call the middlebox. First coined in 1999 by Lixia Zhang, the IETF defines middleboxes as "any intermediary box performing functions apart from normal, standard functions of an IP router on a data path between a source host and a destination host" (Brim et al., 2002). These include a wide range of popular technologies now employed almost universally in networks such as load balancers, firewalls, web proxies, IPS and WAN optimisers. Initially formed as software solutions for the deployment of these new functions, the need for performance and security in a rapidly growing network infrastructure encouraged the transition to hardware solutions (Ingham et al., 2002). These fulfil individual roles within the infrastructure cheaply and efficiently, further increasing demand for commercialised drop-in implementations. Today, middleboxes are a fundamental component of many networks to enable and expand network

functionality (S. Huang et al., 2017), with their high-performance relied upon by businesses to meet ever-increasing Service Level Agreements (SLA) and retain a competitive edge. These SLA serve as a powerful financial motivator for companies by converting QoS guarantees into monetary agreements, with any failure to meet these agreements costly both in fees and reputation. This has created a problem. Middleboxes are not without issues for both usage and design, many of which arose from the nature by which they were first created. A lack of standardisation both for design and access leaves most middleboxes as "blackbox" devices: remotely inaccessible, statically built, and typically not repairable, intended instead as a disposable unit (Sekar, Ratnasamy, et al., 2011). Individually, these characteristics are not notable compared to the benefits these hardware devices offer. Deployed in the thousands, however, across hundreds of networks (S. Huang et al., 2017), these issues are rapidly amplified into significant management issues with no single solution, difficult to replace, for they are now necessary for businesses to guarantee the services they now provide.

Concerns over potential protocol ossification existed as early as the 90s, with a number of alternative solutions proposed by research initiatives (Gavras et al., 2007, Elliott, 2008) such as active networks (Tennenhouse et al., 1997) (Gavras et al., 2007) to the ForCES group in the early 2000s. Each struggled to find adoption outside of research (Handigol et al., 2009) over concerns regarding their feasibility, especially with easier solutions rapidly gaining popularity (middleboxes). By the early 2010s however, the concepts first established by these proposals, such as a separated data and control plane in packet processing devices (L. Yang et al., 2004a) and virtualisation of network functions, saw widespread adoption by the establishment of the Open Networking Foundation (ONF) and OpenFlow (McKeown et al., 2008b).

These concepts evolved into Software Defined Networking (SDN) and Network Function Virtualisation (NFV), with the former focused upon the aforementioned separation of the control and data plane to enable a unified global control over operations, and the latter the softwarisation of the network to be hosted on general-purpose processing hardware. SDN and NFV have been growing progressively in both academia and industry in recent years, best demonstrated by the rise of SDN-supporting routing fabric that allows for product-specific fine-grain control. The flexibility of software has also allowed NFV to be deployed in areas that would otherwise require bespoke hardware to accomplish, such as control boxes in mobile towers. By softwarising packet processing, evolvability can be rapidly iterated upon far more effectively than middleboxes with none of the issues of planning and acquiring potentially thousands of new devices to replace the existing infrastructure. Areas where performance is more critical, however, have seen far less penetration by these technologies. Typical enterprise and mobile networks deploy significant quantities of middleboxes to enable faster processing and minimise the volume of

traffic in transit for the core. These are expensive and difficult to operate and maintain but provide a critical advantage that software cannot; whilst NFV is effectively a softwarised equivalent of these hardware devices and is especially suited for the purpose of multi-functional packet processing, it cannot compete with bespoke hardware at a fundamental performance level or guarantee the same degree of stability as middleboxes can.

1.2 Middleboxes and state

Middleboxes are most commonly built using Application-Specific Integrated Circuits (ASIC), purpose-built circuits offering a fixed set of network manipulation instructions. This takes the form of a pipeline of processing steps using a match-action design. Packets are received, their headers are parsed and their contents are matched to actions based on the contents of this header. The limited reconfigurability of these actions is implemented using simple registers and performed by the coprocessor that facilitates the operations of the ASIC. Operations in the pipeline are assisted by Ternary Content-Addressable Memory (TCAM), a high-speed and highly parallel memory used for lookup and forwarding operations that require some form of memory. The contents of these tables can be defined as its "state", derived from the results of processing decisions executed on incoming packets. This memory is limited and the ASIC lacks other forms of generic memory such as a stack or heap, motivating efficient use of the TCAM.

The loss of these table contents requires the reprocessing of incoming flows to regenerate their lookup information, which will incur delay in the network. The limitations of these ASICs are expanded upon in Section 3.1.2. Prior research in this domain has attempted to tackle the problem of the loss of state with middleboxes, but often advocates for their replacement with software due to the difficulty of its recreation without awareness of the internal values at the point of failure. They can be broadly categorised by their approach into state capture or recreation, then further subdivided in technique such as VM replication, live replay or packet capture. Each of these approaches have their positive and negative traits with varying areas of focus, expanded upon in more detail in the related work in Section 2.5. The key metrics are accuracy of recovery to minimise disruption or rejection at switchover and the speed of recovery for the same reasons. The majority of prior work in this domain has explored this issue with no concise resolution. Earlier work such as Remus (Cully et al., 2008) encapsulates the target in a virtual machine to allow for checkpointing of the entire environment's state.

This is significantly resource intensive and necessitates modifying the target architecture beyond the tolerable limits set forth by industry requirements (Sherry, Gao, et al., 2015). Later work focuses instead on minimising this incurred delay

through more complex means of state capture such as Fault-Tolerant MiddleBox (FTMB) (Sherry, Gao, et al., 2015). It uses fine-grained checkpointing of the internal state of a middlebox such as the order of thread execution via modification to ensure the greatest possible accuracy of recreated state with minimal delay to per-packet processing. Despite these iterative improvements to the technique, modification, if not replacement, is still required by these proposed solutions, with little work conducted on the greater challenge of capturing state without this. This technical challenge has given rise to a gap in research and industry for state capture or recreation without modification that this thesis targets. As discussed prior, state is typically generated through decisions made on a per-flow basis and, in many circumstances, is generated from TCP handshakes during the first few packets of a flow. Furthermore, the level of accuracy for the recreation of state in this domain far exceeds that of the rest of the network fabric. With networking, there is an implicit understanding that packets will be lost and connections will fail. To recap the challenges that this thesis attempts to tackle: Enterprise and commercial networks use devices known as middleboxes at the network edge to improve performance and minimise latency, but they introduce vulnerabilities with their limited design that may leave these networks at risk of visible disruption. These devices have limitations on their openness, minimising the effectiveness of most state preservation approaches. To enhance their resilience, new techniques can be explored using NFV and SDN that may allow for this gap in existing resilient design to be filled without necessitating their replacement.

1.3 Networks and resilience

Since the early design days of Internet protocols under ARPANET (J. McQuillan et al., 1980), packet-switching network architectures were designed to ensure recovery from a wide range of failure scenarios versus the circuit-switching approach of telecommunications, including packet loss and congestion to link and hardware failure. Redundancy has been the most common and basic approach to network resilience both in hardware and software, from multiple network paths allowing for traversal redundancy to clustered containers distributing processing between nodes. However, the effectiveness of redundancy in the areas of networks that employ middleboxes is limited. These devices apply transparent cross-layer protocol processing and retain network state to extend protocol operations. As a result, they increase system interdependence in the network and violate both the end-to-end and survivability principles (Detal et al., 2013). This internal information (or "state"), retained for each connection currently active or recently observed by the middlebox, is necessary for the device to perform its operations, and its loss would force the middlebox to reprocess and recreate this state, causing noticeable disruption. Upon failure, the loss of this state is all but guaranteed, and any connections transferred to a

redundant box will inevitably need to reconstruct this lost state. Resilience techniques for middleboxes are limited in their application, rarely incorporating any form of internal mechanisms for fault recovery per box, let alone Failover mechanisms to redundant middleboxes. Building collaborative high-availability recovery mechanisms incurs a noticeable performance degradation for most middlebox devices due to the complexity and speed of modern ASICs, while complete state reconstruction is not always guaranteed. Academic middlebox surveys highlight that even minor processing middlebox latencies are operationally intolerable, with a reported limit of 1ms per packet as the upper ceiling (Sherry, Gao, et al., 2015). The scale of deployment for these network boxes leaves simple redundancy as the only option, both a costly and imperfect solution.

Despite their low cost per device, the high volume necessary for both the primary and redundant hardware, as well as the difficulty in their configuration and lack of standardised design makes it very difficult for networks to replace these boxes where required, despite their intention as easily replaceable hardware solutions. Middleboxes are intended to be replaceable drop-in solutions to enable network policy and functionality, replaced every few years as part of network development and expansion. The combination of high volume and heavy reliance discourages their frequent replacement however, as to replace any one type of middlebox may consist of thousands of devices across the network, motivating network operators to minimise turnover. This high-volume technique only ensures eventual service recovery, lacking the ability to recover lost state between middlebox instances and prevent long-lasting service degradation. The volume of hardware necessary to achieve this is far in excess of what is required for their packet processing limits, with the primary goal instead being to achieve a 'five nines' degree of availability.

High availability is typically measured in this nomenclature, with 'five nines' referring to a service being available for 99.999% of a year, roughly translating into 5 minutes of downtime per year at most. This expected degree of availability is legally guaranteed in the form of SLAs which define both availability and expected performance and are a key area of competition between businesses. This only further restricts the options of these companies; competition has given rise to SLAs that software could not achieve in performance, but the inaccessibility of hardware forces networks to deploy middleboxes in quantities exceeding their requirements to mask failure through simple redundancy when it occurs, further exaggerating the operational costs of their use. This conflict forms the basis of the problem area that this thesis targets; is it possible to exploit the flexibility of software to enhance the resilience of these mechanisms without the need to replace them and lose these performance benefits?

1.4 Aims

With the scope of the problem established, a key research question is formed: "Given the operational requirements of middleboxes and their use, can the resilience of middleboxes be enhanced externally through the preservation of state without interference, modification or replacement?". With this question in mind, several aims or goals of this thesis can be made.

- What are the limitations of existing approaches to enabling greater resilience for middleboxes, especially in regards to hardware and blackbox devices, and why are they not adopted outside of research?
- Can state be created externally without observation of the interior operations of a middlebox that is both sufficiently timely and accurate as to provide effective failover for a middlebox, regardless of the level of observability or technology it is implemented in?
- How can a proposed solution to the problem of state preservation be implemented into existing infrastructure without the requirement of replacing existing infrastructure or disrupting the behaviour of normal operations outside of network failure in a quick and reliable fashion?

The aim of this thesis is to create a generic redundant approach to establishing persistence of state through failover between network functions for both software and hardware. There are a number of obstacles to this concept that must be tackled, such as the potential for non-determinism and the lack of openness in blackboxes. These problems are discussed in more detail in Sections 2.3.3 and 2.4.5 and form some of the major challenges of this work.

1.5 Contributions

The contribution of this thesis is Remediate(REsilient MiddlEbox Defence Infrastructure ArchiTEcture), a novel resilience framework that provides stateful failover for software and hardware middleboxes in a range of levels of accessibilities. This includes a wide range of stateful middleboxes, such as IDS, firewalls and load balancers. This is achieved through a range of mechanisms designed to support state retention across white, grey and blackboxes without modifying or replacing existing infrastructure, easily reconfigured with a focus on flexibility. Two publications focus on prototype implementations of major components of this framework: Middlebox Minions (MiMi) and Katoptron. The specific contributions of this thesis are summarised below:

- **High-level resilience framework** A non-modifying resilience framework for middleboxes, realised using virtualisation technologies. Attached to a pre-existing network externally to the current infrastructure, it interfaces with existing orchestration and cloud management layers to establish persistent state across replicas through the deployment of state preserving mechanisms.

- **State-preserving mechanisms** A selection of external state preservation mechanisms, able to extract or recreate the state of white, grey and blackbox middleboxes and distribute it to other replicas. These mechanisms can be implemented in a number of technologies and are able to target both software and hardware middleboxes. This state preservation can ensure the persistence of state across failovers, regardless of source or technology, and can be distributed to any number of replicas of identical or differing technologies.

- **Generic state-recovery system** This system is fast, non-modifying, high-level and easily deployed and incorporated into existing technologies. It does not require the replacement of any underlying infrastructure and can be scaled to deploy and operate in a purely redundancy-based role or be fully incorporated into VNF operations. It offers minimal overhead, supports hardware and software redundancies and is easily reconfigured to be fit for purpose. Finally, it supports a wide range of configurations and network layouts to maximise its usability.

These contributions form the novel resilience framework. Its strength lies in its high-level design, able to support different middlebox architectures for both software and hardware. This allows it to adapt to all possible network configurations and user setups.

1.6 Thesis structure

This thesis is structured into six chapters, with the following five detailed below:

- **Background and related work**
  The background provides a discussion on the trend of how network functions evolved, with their origins and progression, accompanied by the technologies that supported them. Following this, it describes SDN and NFV and their roles within this area. Next, the background goes into the topic of resilience, defining it and providing examples of how each technology discussed so far handles failures. It then goes on to review the related work on resilience approaches. The chapter concludes by highlighting the gaps in existing research and the work necessary to fill this hole.

- **Design**
  The design first establishes the motivations of this thesis, establishing the arguments for the viability of combining software and hardware networks together and how their respective traits may be exploited to better serve existing infrastructure. Furthermore, the limitations of ASIC hardware commonly used in middleboxes are clearly established, as is how software may be used to provide stateful failover where it would otherwise be infeasible. From this argument, a series of high-level design requirements and considerations based on these requirements are established, serving as the motivating factors for how this solution must be built. The design overview of both contributions of this thesis is then discussed in detail before finally concluding the chapter to lead into their implementations.

- **Implementation**
  The implementation chapter describes in technical detail the explorations of this thesis, created to achieve the stated goals of the design. The first section discusses Middlebox Minions, or MiMi, the software-targeted side of this state replication approach to middleboxes, as well as its scalable distribution and generically applicable design. The second project, Katoptron, exploits NFV to replicate state in blackbox devices using targeted filters and non-modifying techniques to mirror it across physical replicas. Finally, it concludes with the testbed design for the experimentation and evaluation built to determine the effectiveness of these two contributions.

- **Evaluation**
  The evaluation discusses the contributions of this thesis and its testing. Firstly, it details the shared environment and elements of the different experiments, including the tools and workloads. This is then followed up by the specifics of the testing environments for each major project's implementation, followed by a breakdown of the testing performed. These details include the metrics being evaluated, how they are being evaluated, and why. Finally, each section will showcase the advantages of these approaches and demonstrate their suitability as effective and generic solutions to ensuring stateful failover across unmodified replicas. The chapter then concludes with any remaining explorations beyond the major projects, including minor experimentation and secondary work.

- **Conclusion**
  This chapter concludes this thesis, first reiterating the problem, followed by a presentation of the contributions of this body of work. This will examine the goals of the design and showcase how NFV is well placed to serve in auxiliary roles in areas of networking where it might otherwise have struggled to see

adoption. Middleboxes will continue to remain in place in enterprise networks for the foreseeable future, but the growing presence of SDN and NFV can allow for effective hybrid infrastructures without advocating for their replacement. Finally, this chapter will conclude the thesis by presenting future directions this work may take in terms of how it may expand the concept to both network testing, SFCs, and other areas of networking.

Chapter 2

Background and related work

In the previous chapter, a summary of the problem area was presented, touching on a large number of topics that require further expansion. The background of this area covers a number of disparate topics, including middleboxes, the early history of network development and the principles of resilience. This chapter will go into much greater detail on these topics to lay the groundwork for understanding the later arguments and their overall position. It is organised into six sections. The first, section 2.1, provides definitions for terminology, technology and concepts used throughout this thesis, also expanding on some defined in the glossary. Section 2.2 then opens the discussion on the topics at hand, setting up the following sections. Section 2.3.2 delves into network functions, middleboxes, the end-to-end principle, SDN, NFV and how these technologies evolved over time and their relationships with one another. Section 2.4 then discusses the concept of resilience, its definition, history and the types of resilience important to this thesis for VNFs and middleboxes. Finally, the related work in Section 2.5 discusses relevant research within this specific domain, ranging from historical to up-to-date approaches and how they shape the argument in the subsequent chapter. The background then closes with a summary.

2.1 Definitions

To prelude the background discussion, a number of definitions must first be established for the terminology used throughout this section and later areas of this thesis. This section will be broken down into subsections, grouped by associated terminology.

2.1.1 Networking terminology

There are a number of technologies that this thesis will discuss, both directly and indirectly, throughout the text. Firstly, a Network Function is defined by ETSI as

"a functional building block within a network infrastructure, which has well-defined external interfaces and a well-defined functional behaviour" (N. ETSI, 2013). Their history is detailed more thoroughly in Section 2.3.2. Software Defined Networking (SDN) (Haleplidis et al., 2015) is an approach to network management that supports the separation of the control and forwarding planes of network infrastructure via standardised interfaces. It emerged in 2011 with the ForCES group's OpenFlow and was built as a way of modelling the operations of network functions in software. This concept evolved into its own independent concept known as Network Function Virtualisation (NFV) (White Paper, 2012). It is a paradigm that promotes the implementation of network functions in software to be run on general-purpose server hardware that can be instantiated wherever in the network as required without the need for the installation of proprietary hardware.

2.1.2 Middlebox terminology

A middlebox is a physical or digital realisation of a network function, typically implemented within router fabric in modern design, but for the majority of their lifespan, they have taken the form of bespoke network hardware. For this thesis, the term middlebox predominantly refers to these hardware solutions rather than software, unless stated otherwise such as in other published work. In this form, the middlebox is inexpensive, bespoke hardware used to enforce new network policies through drop-in devices in the network. In brief, middleboxes are effective solutions to the problem of establishing new network functionality, with simplistic architectures and high packet processing speeds that employ TCAM and internal registers to allow for stateful operations. They are discussed in more detail in sections 2.3.3 and 2.4.5.

2.1.3 Resilience terminology

Resilience in networking has many definitions, but for this thesis, we use the definition "The ability of the network to provide and maintain an acceptable level of service in the face of various faults and challenges to normal operation" (J. P. G. Sterbenz et al., 2010). This term is used interchangeably and covers a wide range of topics, which shall be explored in more detail in Section 2.4. This section will define the terms used to refer to approaches to resilience. Redundancy is the inclusion of extra components that are not strictly necessary to functioning in case of failure in other components. This can take many forms, from additional links to spare middleboxes and routing fabric. Failover is a procedure by which a system, upon detection of a fault or failure in the area it is monitoring, transfers operations and/or traffic to a secondary area. For example, upon detection of a failure in a middlebox, all traffic is rerouted to be sent to a second redundant middlebox to bypass this failure. Failover relies on redundancy

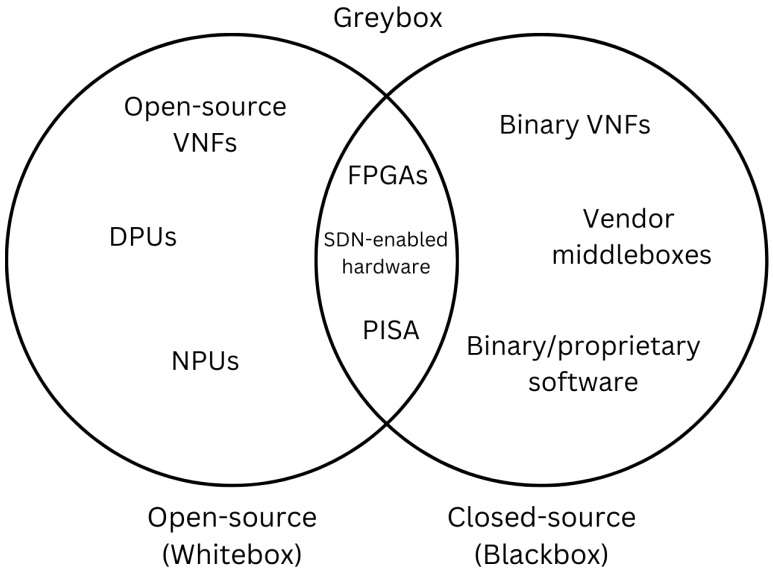

Figure 2.1: A Venn diagram of the degrees of openness present in both platforms and network function implementations and where examples of these sit within this framework.

to function, with redundancy being the practice of having redundant components and failover being the practice of utilising it to ensure the continuation of operations. Stateful failover is an extension of this practice to allow for the retention of state across instances rather than its loss at the original point of failure. Resilience against all forms of disruption or failure is key to operating networks, especially those that provide a service or facilitate business with customers. It is considered such a vital aspect to ensuring continued failure-free operations that guaranteeing an expected level of service, both in performance and prevention of disruption, is a negotiable arrangement. A service level agreement, or SLA, is a contractual agreement between a customer and a service provider that outlines what services will be provided and defines the acceptable range of performance and availability of those services, with any breaches made to this expected guarantee incurring heavy fines or penalties for the company and the customer.

2.1.4 White, Grey and Blackbox network devices

The broad definition of a middlebox effectively encompasses all possible network functionality beyond routing, including both its physical and virtual implementations.

For this thesis, middlebox will typically refer to these physical implementations, although this term is not fixed among other bodies of work such as those discussed in the related work. This is due to the term middlebox predominantly being a definition found in standards and literature and not typically used in real-world contexts where the network function names themselves are used instead; for example, a firewall or gateway. The terms white, grey and blackbox used throughout the rest of this thesis refer to the level of openness that a middlebox supports. A whitebox is an open-source or programmable middlebox implementation, typically taking the form of virtual instances or VNFs. A blackbox is the opposite of a whitebox: closed-hardware or software implementations that abstract all internal operations and state, typically sold and distributed as products. These generally take the form of hardware, intended as unmodifiable drop-in solutions to enable a network function, but also appear in the form of binary pre-compiled software packages. Finally, greyboxes are configurable hardware or software implementations of a middlebox that are neither open-source nor closed solutions, generally offering either output, programmability or some measure of access and control. Examples that fall under these loose terms, detailed later in the thesis in Sections 2.3.3 and 3.1.2, are presented in Figure 2.1. Blackboxes are the focus of this thesis, being the hardest of the possible types of network function implementation to determine their internal operations and thus replicate their behaviour and internal state.

2.2 Opening

As stated in the introduction, the goal of this thesis is the establishment of accurate and efficient resilience mechanisms for blackbox middleboxes. Their blackbox nature and ubiquity in enterprise and commercial networks is especially problematic to their resilience against loss of state, with state transfer and preservation rarely possible and internal recovery systems ad-hoc and uncommon (Sherry, Gao, et al., 2015). The in-transient problems of middlebox usage, like their lack of standardisation, have no singular root cause but occurred naturally over time as they evolved to accommodate the needs of the networking industry at the time (Brim et al., 2002). As networks continue to grow and become more complex beyond what current hardware designs are able to achieve, industry and academia have turned to the growing adoption of the paradigm of NFV for the potential of its flexibility and elasticity. This brings with it its own share of issues regarding performance and reliability, limiting its adoption in real-world networks until the last few years, and even then only in areas where performance is not critical, with middleboxes remaining entrenched. An area of research has developed that has attempted to tackle this issue, mitigating the downsides of VNF use, such as its lesser performance or reliability, or creating new resilience techniques or mechanisms in an effort to motivate its use over blackbox hardware.

This operational reality has given rise to hybrid networks, utilising NFV and physical networks together in conjunction to minimise replacing existing infrastructure. By incorporating software into secondary roles for resilience purposes, greater network resilience could be achieved without sacrificing performance and potentially prevent middlebox ossification in the future. The scope of this work relies on three networking paradigms: middleboxes, network resilience and NFV. To familiarise the reader with the challenge of this goal, we must establish these fields and how this work fits under each umbrella. Firstly, we shall discuss the history of the Internet and its initial design principles and how they failed, giving rise to the concept of network functions and their two implementations: VNFs and middleboxes. We shall also briefly discuss SDN in this section, the precursor technology to NFV that defined clear APIs, and the means to realise network functions in software. With these concepts established, we shall discuss resilience and the relevant methods in the domain of middleboxes and VNFs, ending finally with related work that presents related efforts and research in this domain and how it compares to the work of this thesis.

2.3 Network Functions and middleboxes

Network functions first arose in the widening adoption of the Internet as its use began to propagate into the public sphere in the early 90s (Leiner, Cerf, D. D. Clark, Kahn, Kleinrock, Lynch, Jon Postel, Larry G. Roberts, et al., 2009). The growth of traffic as well as the number and types of users outside of research quickly made evident the need for certain features that could not be supported by existing protocol design and specifications. These came to be known as network functions, the very first of which being packet filters, the precursors to firewalls (Ingham et al., 2002). This section will discuss a number of relevant topics, starting with the history of Internet design principles and the end-to-end principle in section 2.3.1 and a description of network functions in section 2.3.2. This is then followed by middleboxes and their issues, along with their early history and how they first came about in Section 2.3.3. This section will explore the history of network functions and how they came about, as well as their later hardware manifestations in the form of middleboxes and the issues that plague them. This is followed by summaries of both Network Function Virtualisation in Section 2.3.6 and Software Defined Networking in Section 2.3.5, as well as their own attempts to fulfil the roles currently occupied by bespoke hardware and the problems that stem from their own use.

2.3.1 Internet-design principles

In 1981, Saltzer, Reed and Clark established the first definition of the end-to-end principle of network design (Saltzer et al., 1984), motivated by the need for a formal

specification of the manner in which two end-points must act to communicate across a networked space. The definition is as follows: "The function in question can completely and correctly be implemented only with the knowledge and the help of the application standing at the endpoints of the communication system. Therefore, providing that questioned function as a feature of the communication system itself is not possible." This is known as the "end-to-end" argument and specifies that the communication system alone cannot be the sole means by which reliability guarantees are enacted for endpoint communication. Features can be implemented in the network at intermediary nodes, but they cannot guarantee end-to-end correctness alone. This argument was formed in counter to the general perception that reliability measures within a communication system are perceived as an engineering trade-off regarding performance rather than a requirement for correctness. It has been re-interpreted many times since its inception, with many definitions of the end-to-end argument now in existence, but for this thesis, its role on implementing resilience is the most important.

Reliability is necessary for maintaining performance, but it quickly becomes difficult to determine where to establish resilience mechanisms, with trade-offs both at the lower level (the end points) and higher (communication system). The origins of this argument are found in the earlier works of academics when first establishing both security and reliability across expanding networks, such as the "wait" message of the MIT Compatible Time Sharing System upon sending any command as a questionably useful delivery acknowledgment (Corbató, 1963). This network was notably unreliable and offered little to no guarantees beyond this simple message. Later arguments began with the need for encryption and were significantly expanded on by later work (Diffie et al., 1976, Kent, 1976, Needham et al., 1978). Despite this well established principle, the pursuit of performance slowly led to the rise of new functionalities implemented within the network itself. These came to be known as network functions, and by extension, their later incarnation middleboxes.

2.3.2 Network Functions

A network function is defined by the IETF and ETSI as "a functional building block within a network infrastructure, which has well-defined external interfaces and a well-defined functional behavior" (N. ETSI, 2013). Network functions encompass a wide array of functionality within modern networking, spanning essentially all activity beyond routing and can be implemented in both software and hardware. Their predecessors originated in the mid-80s as software incorporated into user systems built to serve singular roles within the growing Internet infrastructure. One of the earliest examples of this are firewalls, first established in 1987 (Ingham et al., 2002) as simple packet filters on end-user systems. These mechanisms were necessitated

by a growing need for security in the previously open approach to networks, where a level of trust was considered sufficient within the previously smaller community. As these networks grew, the potential for malicious traffic arose and necessitated the need for security, establishing the first packet filtering techniques at the router level. This need for security shifted its implementation from a user level on a per-system basis, where each system may have inconsistent implementations of policies, to network gateways at the router level to maximise its coverage of incoming traffic and ensure consistency. It suffered from its inability to keep awareness of the established sessions, minimising their effectiveness at stopping long-term or distributed attacks, establishing the first need for persistent state awareness and storage in Julkunen et al., 1998. This motivated the separation of this new functionality into its own hardware from the routing infrastructure with the term "middlebox" defined by Lixia Zhang in 1999 (Brim et al., 2002).

2.3.3 Middleboxes

Middleboxes are the physical implementations of network functions and are widely used by networks to enable new functionalities, taking the form of devices such as WAN optimisers, firewalls, IDS and proxy caches (Z. Wang et al., 2011b, Sherry and Ratnasamy, 2012). As a broad summary, they are formed of a configurable pipeline of processing stages, divided into the individual steps of parsing, lookup and functions, demonstrated in Figure 2.2. The internal architectures are discussed in more detail in Section 3.1.2. Beginning in the late 90s to the early 2000s, companies such as Cisco began to sell the first commercially available network functions in hardware as products (Leiner, Cerf, D. D. Clark, Kahn, Kleinrock, Lynch, Jon Postel, Lawrence G. Roberts, et al., 1997, Feamster et al., 2014). These devices became increasingly popular within enterprise and cellular networks (Z. Wang et al., 2011a) for a number of reasons, the chief of which was their high performance to low cost ratio. As networks grew and became more complex, newer functionalities were necessary to ensure their continued expansion, such as load balancing and traffic inspection. Middleboxes became the primary means by which these new features were implemented, purchased as complete units that were easily inserted into existing infrastructure and configured individually. Their simplistic architecture and ease of deployment (bump in the wire) are, in part, intended to make these devices replaceable. Treated as a product first and foremost, companies will replace the boxes they use as their needs change. As hardware-accelerated devices, their packet processing speed is an order of magnitude faster than the software equivalent. Their many benefits shaped the design of these network infrastructures, but they are not without their flaws.

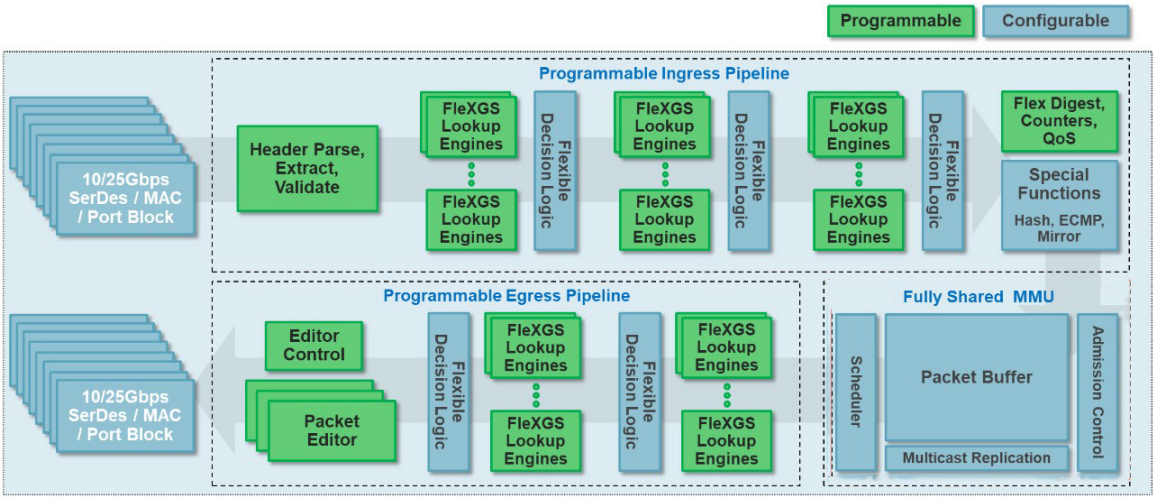

Figure 2.2: Broadcom Trident 3 Internal Architecture Diagram (Arcilla et al., 2019)

2.3.3.1 Middlebox issues

There are a number of issues associated with their use, both for the end-user and the wider network infrastructure. To begin with, there is no singular universal model for their design, operations or API agreed upon by vendors and academics. There have been numerous attempts to define a standard in the past, such as RFC 3234 (Brim et al., 2002 and its high-level definition or the work of organisations such as the ForCES group (L. Yang et al., 2004b). More recent efforts like ETSI's NFV and PNF architectural frameworks (ETSI, 2012) and IETF's OpenConfig YANG models (Bierman, 2011, Bierman, 2018) have seen greater levels of adoption, but for the immediate future there is no universal deployment or configuration method shared between companies and their products, creating a significant combined issue of maintenance and operation as demonstrated in a survey of middlebox deployments (Sherry and Ratnasamy, 2012). The array of potential middleboxes and their differences require companies to maintain dedicated management; misconfigurations make up for 33% of faults that occur with middleboxes, with other issues including bad operation policies borne from differing syntax from box to box and poor interoperability (Potharaju et al., 2013) showcasing this need. For other networks and traffic, these devices break the end-to-end principle. The nature of the interference is dependant on the middlebox, ranging from modification of headers or the dropping of packets in accordance with security policies to modifying header and packet contents for traffic shaping, DNS policies and caching preferences (Detal et al., 2013).

This interference can cause issues with other areas of networking and has heavily contributed to a level of ossification in hardware, not only because companies find

them difficult to remove for performance and maintenance purposes but also due to their simplistic design, which minimises the expansion of existing Internet protocols by baking assumptions about protocol semantics. Work by Honda et al., 2011 evaluates the range at which middleboxes interfere and even drop TCP extensions as mangled packets or replace fields that do not conform to its expected traffic. The extent of their use is substantial, despite the range of issues they incur. Their numbers reportedly rival those of routers within enterprise networks, with a survey (Sherry and Ratnasamy, 2012) across 57 networks varying in size from under 1K hosts to more than 100K, indicating an average of 1946 middleboxes to 2850 L3 routers in major networks. Another study that probed for the presence of middlebox deployments observed hundreds across more than 1000 ASes (S. Huang et al., 2017). Middleboxes are low-cost per unit, but the sheer volume of middleboxes necessary to deploy a feature across a network typically scales in the thousands (S. Huang et al., 2017), magnifying the problem drastically.

Furthermore, this has created a unique problem: despite their intention as replaceable hardware, easily swapped in and out where necessary, networks instead have become so reliant upon these devices that they are fundamental to their business models, ossified both on a per-box basis and an industry-wide problem. The variance of products dissuades replacement of a line of middleboxes, as it necessitates replacing an entire line at once, incurring significant costs for both hardware and new operation/maintenance training for engineers. As a result, middleboxes are simultaneously replaced too often and not often enough. Their direct interference with packets that pass through them and widespread use has further contributed to the ossification of protocols through the baked-in assumptions they enforce, breaching the end-to-end principle fundamentally. This was in large part motivated by the pursuit of performance, shifting packet processing and functionally onto the communication medium rather than the end points and creating this wide variety of issues. In short, middleboxes are expensive and difficult to operate, maintain and replace, but they are utterly necessary to guarantee expected levels of performance and retain a competitive edge.

2.3.4 Early history of programmability attempts

In the early 90s as the Internet began to grow, concerns over its expandability and protocol ossification had already begun to grow. Several radical alternatives were proposed, the key among them being the concept of programmable networks. Programmability would enable a level of flexibility not present in network design at the time, with the earliest work focusing on the switching fabric itself. The Devolved Control of ATM Networks (DCAN) (Merwe et al., 1997, Rooney, 1997) was one of the first that explored the separation of the control and management of

the switches from the hardware itself via an external device. A number of initiatives established themselves with this new concept of networking in mind, including "Global Environment for Network Innovations" (GENI) (Elliott, 2008), "Future Internet Research and Experiment Initiative" (EU FIRE) (Gavras et al., 2007, Serrano et al., 2022). The IETF established an early approach to label/tag switching with the General Switch Management Protocol (GSMP) (Worster et al., 2002) that separated the physical switch into multiple partitioned virtual switches with a unified controller, able to program labels and alter the directing of traffic. This can be considered a precursor to protocols such as MPLS, now widely used today.

One particular project relevant to the history of SDN was the concept of "active networks". Active networks proposed that networking devices should be a collection of resources accessible via an API, their functionality remotely programmable (Feamster et al., 2014). Switches and routers could be programmed remotely through bespoke network traffic that could reprogram the operations of the switching fabric during operation. The motivations for active networks mirror those of SDN and NFV, such as the replacement of middleboxes with virtualised and controllable NFs (Tennenhouse et al., 1997) that can be operated and configured from a centralised location. Active networks as a concept never saw any significant adoption beyond these initiatives, primarily over a lack of motivating problems it could directly solve that did not already possess alternatives with significantly lower barriers for entry (Feamster et al., 2014). The concepts established by active networking, such as the separation of the control and dataplane (L. Yang et al., 2004a), the centralised control of every network element (Caesar et al., 2005) or virtualisation of ad-hoc hardware solutions (middleboxes) remained. Their potential benefits on the Internet and networking in general motivated smaller and more focused initiatives to form in pursuit of these goals.

2.3.5 Software Defined Networking

Software Defined Networking (SDN) is a precursor technology to NFV, first established as a means by which network functions could be realised technically, enabling the remote operation, teardown and startup of VNFs alongside the control of the rest of the network infrastructure. It is defined by the IETF as "A programmable networks approach that supports the separation of control and forwarding planes via standardized interfaces" (Haleplidis et al., 2015), and the ONF as "A network in which the control plane is physically separate from the forwarding plane, and a single control plane controls several forwarding devices" (Larry Peterson, Carmelo Cascone, Brian O'Connor, Thomas Vachuska, and Bruce Davie, 2016). It was created as an attempt to model VNF functionality, allowing for their creation and utilisation across a wide array of generic hardware and SDN-enabled switches. From there, the concept evolved

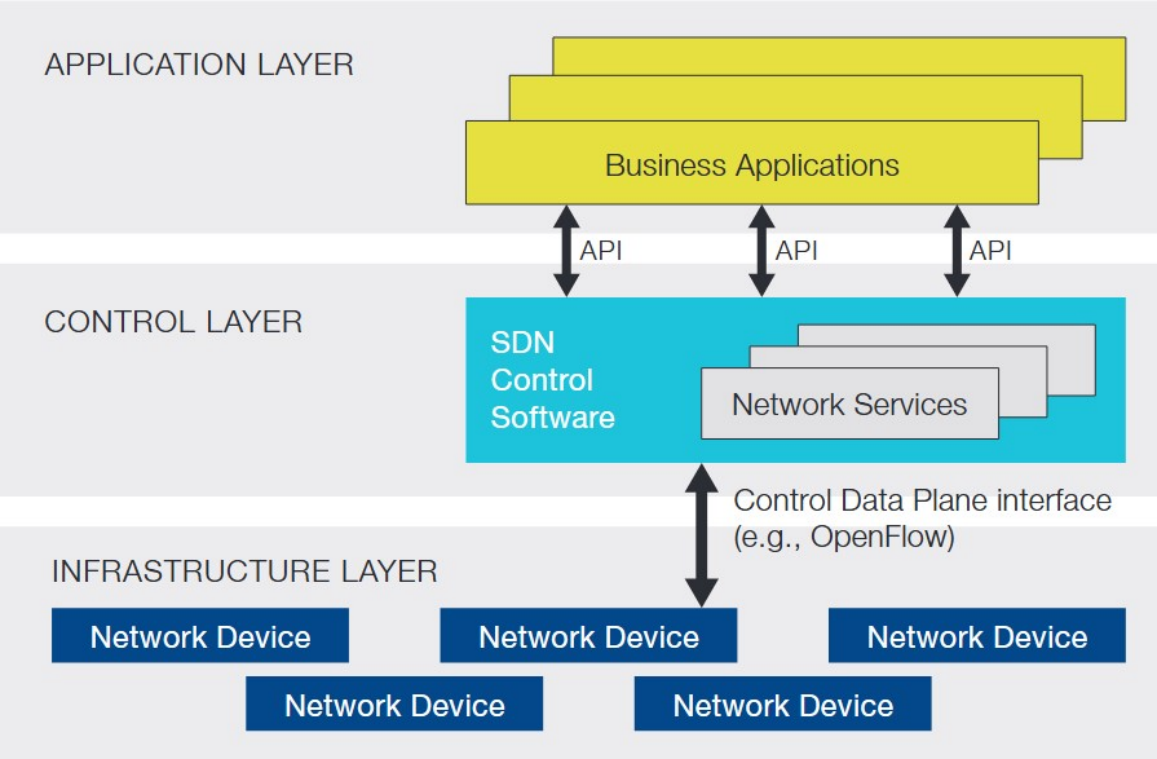

Figure 2.3: Layers of the Software-Defined Network Architecture (Tank et al., 2017)

to define an architectural model that enabled control of the network and its operations by dividing the logic of the network infrastructure into a management/control plane (exposed via an API to be accessible to network operators) and the dataplane (the forwarding infrastructure), presented in Figure 2.3. Emerging in the 2000s, the IETF ForCES group established the concept of separating the control plane from the forwarding plane but saw little uptake outside of small-scale projects in academia like "LOad-Balancing over UnStructured networks (LOBUS)" (Handigol et al., 2009) and Ethane (Casado et al., 2007). This changed in 2011 with the establishment of the Open Networking Foundation (ONF); a consortium established to promote SDN through OpenFlow (McKeown et al., 2008a), one of the first communication protocols to standardise an interface between the two planes with support from vendors, establishing OpenFlow-enabled switches whose forwarding logic could be operated remotely through a well-defined API (McKeown et al., 2008b). OpenFlow presented a realistic means by which network functions could be programmatically redefined within a limited scope of flexibility. OpenFlow became one of the most popular industry-supported SDN protocols with a wide range of supporting hardware and bespoke research roles, able to be utilised to realise VNFs in its language. Major

cloud operations, like Google, have adopted it as the basis of several internal network infrastructures including B4 (Jain et al., 2013) and Jupiter (Poutievski et al., 2022). SDN does not play a significant role within the body of work that forms this thesis under its modern definition, but it is important to cover as it represents a stepping stone in the evolution of the concept of programmable networks and how they have interacted with middleboxes over the last 30 years of networking. This timeline can be loosely defined as beginning in the early 90s with projects such as active networks, followed progressively by ForCES, the emergence of OpenFlow as the dominant SDN technology and finally network function virtualisation.

2.3.5.1 SDN architecture

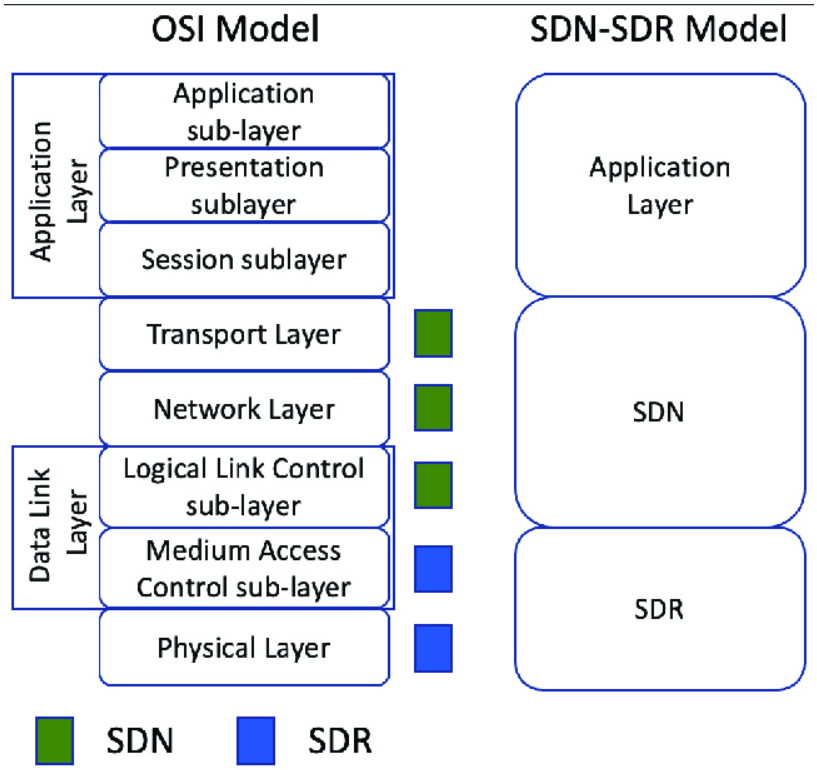

Figure 2.4: Mapping SDN and SDR to the OSI model (Niknami et al., 2023)

To better understand the role of SDN and how it shaped modern thinking regarding programmable networking, the structure of the SDN architecture and how it works shall be discussed. In principle, the SDN paradigm divides the operations of the network into discrete planes via abstractions, forming three separate layers: the data, control and management planes. The data plane is comprised of the physical

hardware itself, built in such a way that its operations can be clearly defined and operated through a southbound interface via the control plane. Its functions include header parsing, extraction and packet operations. The control plane dictates the logic of the device itself and how it will employ these functions on traffic received for the purposes of actions like routing and security. Some define a third layer, in-between what has been defined as the control and dataplane so far. This third layer takes the place of the control plane, with the one defined above renamed the management layer, and provides the abstraction interfaces for enabling control of the hardware (Zilberman, Watts, et al., 2015). This third layer is more commonly referred to as the Networking Operating System (NOS) and essentially fulfils the role of a translation layer for non-standardised hardware interfaces and SDN. An early example of a NOS is Ethane (Casado et al., 2007), with later works mostly adopting OpenFlow as the de-facto standard protocol (McKeown et al., 2008b) for controllers such as NOX (Gude et al., 2008), as well as hardware-specific implementations such as Broadcom's OF-DPA (*OpenFlow data-plane abstraction (OF-DPA): Abstract switch specification* 2014) and Openvswitch (Pfaff et al., 2015). SDN's planes do not directly correspond with layer models such as OSI, but SDN's scope can be broadly isolated to the link, network and transport layers, demonstrated in Figure 2.4. The scope that it may cover however can vary wildly between technologies, however.

Most SDN implementations utilise flow-centric logic, associating sequences of packets with matching headers and fields such as origin and/or destination address. What identifies a flow and how it is acted upon by the network device can be changed and is dictated by the control plane and enacted upon by the data plane respectively. These flows are matched against entries in a table known as the flow table, which pairs packets to match against the respective action to perform. For example, a packet received by the device that matches an existing flow table entry, to drop all packets from that source MAC, will be discarded. A similar packet from the same source, but matching a more specific rule that further states those going to a particular TCP destination port are to be forwarded would not be discarded. Rules are matched in order of specificity, with wildcard rules able to be established for those that fall within a specified range. SDN further enables traffic received that does not match against any existing flow table entries to be used to create new entries.

2.3.5.2 SDN in production networks

Much like NFV, SDN has been growing in popularity both in academia and industry for its potential to enable fine-grain control over traffic within the network. This is best demonstrated with deployment methodologies such as Panopticon (Levin et al., 2014) wherein only a subset of key switches in the network are replaced with SDN-enabled hardware, allowing for a significant degree of control and observation

over flows with minimal infrastructure replacement. This is especially relevant to VNFs when discussing Service Function Chains (SFC); logically-defined pipelines of network functions, typically beginning at the ingress of a network. These NFs are not physically placed in a chain, but instead traffic is directed to the next link. With the potential to deploy VNFs anywhere within the network, traffic steering protocols are required to properly utilise this flexible approach (Hajar et al., 2018) or similar routing policies such as Flowtags (Fayazbakhsh et al., 2013) where traffic tagging can allow for both flow tracking and network-wide policy enforcement.

OpenFlow is not the only SDN protocol currently supported in industry; forwarding plane programming languages such as P4 (Bosshart et al., 2014) and kernel modules such as Extended Berkeley Packet Filters (EBPF) expand beyond the rigid definitions of current networking design to allow for both target and protocol independence. Other companies prefer to maintain their proprietary approaches outside of the ONF and IETF. These proprietary approaches tend to be specific to each company's product use cases, such as Cisco, which extends the concept of SDN into an IoT platform with their Internet of Everything (IoE) through which all of their devices are integrated. This includes the Typhoon, QFP and their development environment onePK (Cisco, 2013). The degree to which this environment is supported by the ONF and other hardware outside of their respective companies varies from product to product.

2.3.6 Network Function Virtualisation

Motivated by the multitude of issues associated with these blackboxes and the potential flexibility and elasticity of software, both vendors and industry pursued significant research in the field of softwarisation of network functions, culminating in the proposal of Virtualised Network Functions (VNF). VNFs were first proposed by the ForCES group in 2004 (L. Yang et al., 2004b) as part of their proposal to define a framework and set of protocols for the separation of the control and forwarding planes into discrete elements that could communicate with one another and be rendered as totally separated physical or virtual implementations. ETSI defines Network Function Virtualisation (NFV) as "the implementation of network functions in software that can run on a range of industry standard server hardware and that can be moved to, or instantiated in, various locations in the network as required, without the need for installation of new equipment." (White Paper, 2012). A virtualised network function is a softwarised NF hosted on generic processing hardware that serves a singular role akin to its hardware equivalent. As software, the flexibility of VNFs is substantially greater, allowing them to be modified, created and destroyed to meet the needs of the user. This early proposal establishes many of the later concepts of virtualisation for scaling and redundancy purposes as well as orchestration and control of the forwarding

elements of a network from a logically centralised point.

There was little adoption in 2004 at the time of this proposal over concerns of the unreliability of softwarised forwarding elements versus the growing market for bespoke hardware. The desire for softwarisation remained, however, for the potential flexibility and cloud elasticity offered, especially as the flaws of middlebox usage became more readily apparent, limiting networks from protocol expansion and new technologies. In 2012, a consortium of telecom companies published a white paper (White Paper, 2012) introducing NFV, motivated by the same reasons for removing this increasingly ossified technology to allow for innovation in the industry. With this white paper, they announced the establishment of a Industry Specification Group (ISG) under ETSI to establish a formal definition of VNFs beholden to a standards body for industry use and development. Since then, a number of standards and definitions have been created regarding NFV and its usage, including its management (ETSI, 2014a), architecture (ETSI, 2012) and reliability requirements (ETSI, 2016).

2.3.6.1 NFV Benefits and Drawbacks

The advantages of NFV described in this formal introduction include reducing OpEx/CapEx costs, faster innovation and greater flexibility in infrastructure design, especially in regards to redeploying in response to faults and traffic requirements. With this recognition of its potential advantages, the call to action also established the challenges faced by NFV that needed to be resolved if it is intended to replace the existing infrastructure of middleboxes. These challenges included the recognised performance trade-off shifting from proprietary hardware to general-purpose processors, as well as the reduction in security, resilience and network stability by stepping to open software from closed devices. The white paper proposed the use of appropriate hypervisor techniques and any possible software innovations that followed to minimise this increased latency. The purpose of this industry-led standards body was to standardise deployment and accessibility mechanisms for VNFs and prevent the issues that plagued the market-driven middlebox approach, and ensure compatibility across all future platforms. With organisations such as BT, Orange, China Mobile, AT&T and more pushing for their use, VNFs have rapidly gained in popularity over the last decade, with the industry increasingly integrating the technology into their product lines (Nokia, 2022). There are limitations to the performance of virtualisation outside of simple processing speeds. Parallelisation is often touted as a means by which this gap can be closed, but potential conflict over shared resources, such as CPU thrashing limits, scaling to the number of logical cores of the host before performance loss exponentially rises (C. Wang, Spatscheck, Gopalakrishnan, and Applegate, 2016). Placement and capacity limits exacerbate this issue, preventing naive scale-up and necessitating careful evaluation of both the resource use and operations of VNFs and

their hosting platforms to maximise performance (Cao et al., 2015).

The distribution of operations is non-trivial, requiring the use of additional resources such as load balancing to ensure consistent distribution and minimise disruption. While virtual and physical performance have typically remained orders of magnitude apart, modern processing hardware has allowed some deployments in industry for virtualised instances, such as at the base of mobile towers for monitoring and control roles—roles where their performance is non-critical but operational costs can be cut where hardware is not strictly necessary. Outside of these non-critical roles, other companies have pursued offloading the software onto programmable hardware such as FPGAs, as is the case with Nvidia's Cumulus (IDC, 2021) or user-space data plane programming like Intel's DPDK (L. Foundation, 2015) or Click (Martins et al., 2014), in an effort to close this gap. Despite this, production networks hesitate to replace performance-critical hardware with software amidst known concerns about performance and stability issues, especially given the scale and volume at which middleboxes are currently deployed.

2.4 Resilience

To discuss the gap in research that this thesis targets, we must first address the field of resilience itself and why it is important to networking, as well as detail why this subject area is difficult and worth pursuing. Resilience is an extremely wide field in regards to computer systems and even networking, and it can be difficult to define succinctly. It is an umbrella term that holds many independent concepts, such as design principles, technologies and properties of systems, many of which are not directly relevant to this thesis. For this body of work, the definition of resilience used and what it means to networks and systems is derived from the work of Resilinets (J. P. G. Sterbenz et al., 2010) and detailed in Section 2.1.3, and forms the basis of this thesis' approach to resilience as a whole.

Disruptions and faults come in many forms, from individual hardware failures and user error misconfigurations to malicious attacks and even regional disasters such as earthquakes. It is an important and often vital part of system design, but difficult to effectively and efficiently achieve with the size and complexity of modern networks, especially with the level by which they are now depended upon. Computer networks and their growth from individual research projects into large interconnected systems created a design principle born of necessity: the primary assumption of all network design is that the system is fallible and failure is guaranteed with time. The modern Internet protocol stack assumes that these failures will occur and has been designed in such a way as to enhance the resilience of service at every level, from live pathing identification at the link-layer level in case of broken or changing connections to the transport layer TCP protocol for guaranteeing payload delivery. This mindset has

led to the creation of a significant body of techniques and research into effective resilience strategies, including but not limited to: redundancy (Lyons et al., 1962), service migration (C. Clark et al., 2005), survivability planning (Ellison, Fisher, Linger, Lipson, T. Longstaff, et al., 1997), segmentation, multi-routing (Hopps, 2000), graceful restarts, fast-reroutes (Bernard Fortz et al., 2000, Li et al., 1998) and so on. These concepts will be touched upon, with those relevant to this thesis expanded in more detail.

2.4.1 History of resilience

The beginnings of this resilient design mindset can be first observed with the earliest research into computer networks with ARPANET. Originally, very early computer communications were modelled after the existing telephone networks using circuit-switching, which established a link between two end points that remained dedicated for the duration of the connection. This approach was considered highly vulnerable to disruption, especially in the face of large-scale disasters. The Advanced Research Projects Agency Network (ARPANET), first established in 1969 as a network bridge between four geographically separated terminals at research centres and universities, was one of the first realisations of a packet-switched network. Motivated more by the desire to maximise bandwidth use and minimise latency (L. Roberts, 1978), its enhanced resilience against disruption became evident as the network began to grow in size (J. M. McQuillan et al., 1977). Pathless communication allowed for two approaches to emerge: connectionless and connection-oriented communications, or virtual circuits and datagrams, respectively. Virtual circuits, used in ARPANET, required a connection to first be established between two end points, while datagrams transported packets independently. From there, the development of networking protocols grew rapidly, resulting in a "protocol war" between competing concepts, with ARPANET just one of a multitude of projects emerging during the 70s and 80s, conflicting most notably with X.25 (Rybczynski, 2009). In the 1980s, TCP/IP emerged as two conjoined protocols governing connectionless and connection-oriented networking, respectively, with UDP following on top of IP for connectionless routing. Its association with ARPANET heavily contributed to its becoming the standard across all interconnected networks in 1983 (J. Postel, 1981), subsuming X.25 and other competing protocols. As awareness of computer networks grew, Computer Science Network (CSNET) was established in 1981 to support all systems that could not directly interact with ARPANET, which was by this stage a governmental project. This network rapidly outpaced ARPANET (which was shut down in 1985), ballooning from three initial sites to eighty-four in three years. The rapid growth of CSNET and projects like it made it clear that simple fault tolerance at the message passing and hardware level was insufficient to handle the potential number of components within

a network as well as the newly discovered risk of malicious activity (Ingham et al., 2002) as previously discussed in Section 2.3.2. By 1989, National Science Foundation Network (NSFNET), the successor to CSNET, gave rise to the first ISP, and in 1991, the removal of access restrictions from purely public-funded organisations and research into commercial use established what would become the Internet.

The rapid rise in the size and complexity of these network projects leading to the early Internet in turn gave rise to new concepts of resilience as their necessity became apparent to ensure the continued operations of the network in the face of a wide myriad of disruptions. These concepts include, but are not limited to, security, fault tolerance and survivability planning. Each of these can then be broken down further into the means by which they are achieved, such as redundancy or high availability. Survivability planning (Ellison, Fisher, Linger, Lipson, T. A. Longstaff, et al., 1999) alone has a significant volume of literature that ranges from the principles of survivability (J. P. Sterbenz et al., 2002) to regional failure modeling (H. Yu, Qiao, Anand, et al., 2010, H. Yu, Anand, et al., 2011) to protecting critical infrastructure from attacks (Buldyrev et al., 2010, McDaniel et al., 2009, D. Xu et al., 2004). Another example can be demonstrated in the evolution of traffic pattern handling. Beginning with ARPANET, early routing protocols adapted to traffic fluctuations but caused unexpected poor performance (J. McQuillan et al., 1980), motivating the concept to evolve instead towards controlling the distribution of the traffic instead. This concept, known as "traffic engineering" (Bernard Fortz et al., 2000), has propagated many strategies in the last fifteen years, such as weighted links (B. Fortz et al., 2002), routing strategies (**??**), MPLS-aware protocols (Elwalid et al., 2001) and burst traffic adaptation (H. Wang et al., 2006). The growth of networks and their increased reliance on all factors of society, especially critical infrastructure, have radically increased the level of research and development on ensuring networked systems are resilient in the face of disruption. This section will break down the details relevant to this body of work, including the resilience framework used (Figure 2.5), the types of failure that must be contended with, and the bodies of research concerning both VNF and middlebox resilience.

2.4.2 $D^2R^2 + DR$ Framework

There are many surveys (Cholda et al., 2007) and academic frameworks for defining resilient design (Vlacheas et al., 2011), the challenges it faces (Cetinkaya et al., 2013) and even what resilience itself can mean (J. P. G. Sterbenz et al., 2010). Each attempts to establish the concepts that fall under the term resilience and the interactions between them. These definitions of what resilience might mean differ primarily due to what they are targeting. For this thesis and its work, its approach to resilience is motivated by the D^2R^2 framework. First established by ResiliNets in

2005 (David Hutchison, 2015), it stands for "Defend, Detect, Remediate, Recover" and "Diagnose, Refine". Created to fill a perceived gap in existing research, the framework serves to provide a systematic view of the resilience of the network as a whole. It is derived from four axioms:

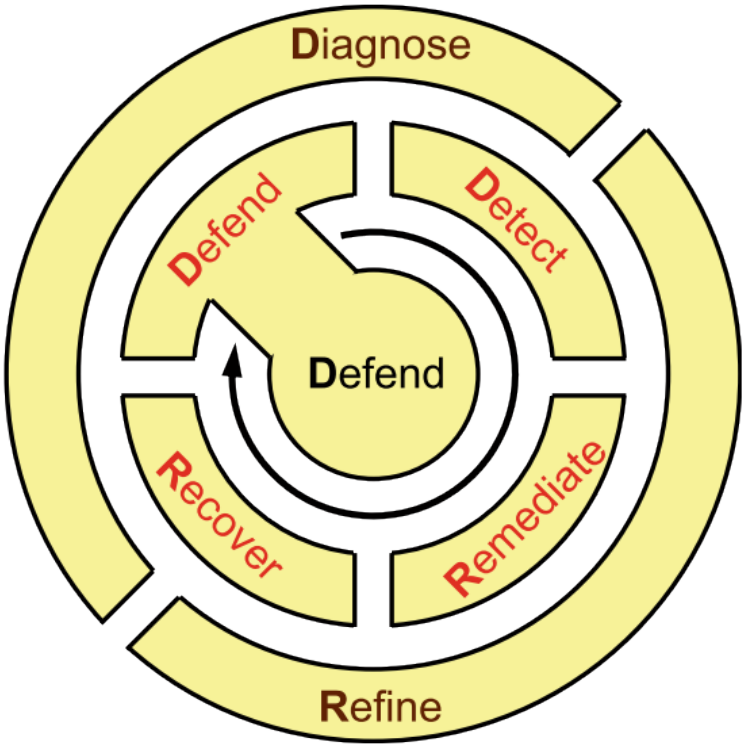

Figure 2.5: The ResiliNets Strategy framework, formed of two cycles $D^2R^2 + DR$, represented as a cycle of its steps

1. **Faults are inevitable**
   Failures will occur at all levels of the network and cannot be prevented, only mitigated and challenged. Systems are not infallible and networking is an especially vulnerable field due to its size and complexity.

2. **Understanding normal operation is necessary**
   The remediation and recovery mechanisms, as well as those for the prevention of faults and failures, must be designed with the understanding of how the system is intended to operate during failure-free scenarios.

3. **Expectation and preparation for adverse events and conditions is necessary**

To plan defence, the potential challenges that the system will face must be understood. These can be predicted from an awareness of similar systems and past events, as well as observed potential vulnerabilities.

4. **Response to adverse events and conditions is required for resilience** The immediate response to adverse events is known as remediation within this framework; the immediate actions taken to mitigate the impact of the event, but not necessarily resolve it. A stop-gap solution to prevent total failure or system degradation, allowing engineers time to resolve the adverse event and the system to recover fully.

At a high level, the framework defines a resilience strategy in the form of a loop, as depicted in Figure 2.5. This loop consists of two separate loops, one encapsulating the other. The first loop, or "Defend, Detect, Remediate, Recover", is the strategy for resolving failures when they occur. The core loop represents the passive day-to-day operations of the network, defending it from faults and failures using strategies such as redundancy and fault-tolerant design. In the event of a failure however, the control loop then transitions to the centre ring. These represent the active def1ences and follow the steps by which to resolve whatever problem has occurred. The final outer ring is the state of the system once the fault is over and the steps are followed. The framework works as a cycle between these four quadrants, with the system defending from faults and failures by default until an error is detected, upon which remediation steps are taken to mitigate the impact of this fault before being fully recovered from the incident. This cycle is followed by the outer loop, Diagnose and Refine, referring to the determination of the cause of the fault and the refinement of both the system and the recovery strategy.

It was developed with the intent of ensuring the resilience of a network was clearly established and planned from the beginning, rather than a patchwork of mechanisms that are not coordinated or intended to operate in tandem. Resilience research can typically be labelled as a stage of the two loops, *e.g.* refine (H. Yu, Qiao, J. Wang, et al., 2014), detect (Sampaio et al., 2018) and defend (Belyaev et al., 2014). The work of this thesis falls within the remediation cycle of this framework and provides several reasons for our perception of both faults and related work, as well as the reasoning behind our approach with NFV as a viable alternative and addition to physical hardware. Expanding on the stage of the cycle that this thesis falls into, this section will detail what remediation is and how it applies. Remediation is the step following the detection of an adverse event or condition in the system, where it attempts to minimise or remediate the impact of this event. This differs from the recovery stage, which follows remediation, where the problem is resolved. Instead, the problem is active and identified, with initial steps taken to effectively dampen its negative impact. An example of this could be described by the networking

principle of graceful degradation. Graceful degradation is the ability of a network or system to provide reduced or limited service during periods of significant service degradation. For example, if the network is under excessive strain and is unable to service all traffic received, a graceful degradation would be the masking of this over-saturation to end-users through delayed responses to requests, static responses that can inform the end user of the issue currently being experienced, and minimising the drop rate of connections that are externally visible. A non-graceful degradation in this circumstance would be to drop all incoming traffic and offer no service or notification under the network strain.

Existing research into resilience is an expansive topic and can be categorised into the quadrants of the cycle. A significant body of work exists in planning and design, or the "Defend" and "Refine" stages, that propose how to place resources (Lira et al., 2013) or VNFs (Qu et al., 2016) when accounting for survivability during disruption. "Detect" varies between identifying the presence of a failure or the potential cause of a failure, such as heavy-hitter detection for potential DOS (Sivaraman et al., 2017). "Remediation" attempts to minimise the immediate impact and extend survival time (Belyaev et al., 2014). This body of work falls under the remediation stage as it seeks to provide another route to graceful degradation and fault management; if the system executes correctly, the entire loss of hardware blackboxes will be completely masked from end users. Section 2.5 will discuss research in this domain.

2.4.3 Types of failure

When discussing resilience, it is important to consider the types of faults or failures that the system operator is attempting to mitigate. A fault can be defined as "a flaw in the system that may cause an error such as a bug" (J. P. G. Sterbenz et al., 2010). Faults may be transient and unobserved in most circumstances, while the severity of failures can range from disruptive to causing service degradation. The severity of these disruptions is highly dependent on the resilience policy of the system, both for design and action plans when faults occur. A failure is the potential result of a fault; it is a deviation from intended behaviour, typically caused by an underlying fault or exterior circumstances. Failures that are observable to the end-user would be considered significantly disruptive and are a large part of the mitigation strategies of action plans to mask both faults and failures. Faults, and even to some extent, failures, are an inevitable result of software development and unavoidable in large-scale systems such as computer networks. The goal of resilience is to ensure the system is not only capable of operating in an expected fashion even when faults occur but also obscures unintended behaviour to outside observers. These observable failures, known as service disruption, can originate from other sources beyond system flaws. Colman-Meixner et al., 2016 defines four types of service disruption: human error

(user mistakes, configuration issues), software failure (bugs, age, security breaches), physical failure (component or cable failure) and disaster scenarios (earthquakes, fire). The degree to which a system pursues resilience is dependent on its use. The use case of the network will shape its policy; a small-scale system with few users might employ simple redundancy for recovery purposes, whereas a larger-scale system serving many users may focus on disruption and traffic tolerances. Critical infrastructure that relies on network communication, such as the power grid (Mather, 2018) or industrial SCADA (Stouffer et al., 2006) will have a much greater focus on survivability and security to maximise resilience against the failure scenarios relevant to its use. The body of resilience research is extensive, and for this thesis, we shall focus specifically on middleboxes, VNFs and localised failures.

2.4.4 VNF Resilience

The rise of software in networks has been firmly established as the current trend of evolution for both infrastructure and expansion. With this rise in use, virtualisation brings a new range of possibilities and risks to ensuring network resilience. As software, instances can be created, migrated and torn down far more easily and quickly than physical infrastructure. Additionally, its nature as software can be modified and accessed so that its state can be potentially captured and migrated. These potential advantages bring an additional set of factors that must be understood and accounted for to properly utilise VNFs and mitigate their risks and downsides. NFV is unlikely to exist in a vacuum without SDN serving as a means of deployment and control, so a discussion on their risks and overall reliability will touch on its employment through SDN in certain areas.

2.4.4.1 New risks

Software introduces a great deal of complexity over hardware by virtue of its design. In the scope of typical networks, hardware can refer to either a generic COTS device such as a router or switching fabric, or more specialised devices such as middleboxes, build to perform a singular function or role. While these can be subject to design faults, mechanical failure, or external circumstances such as power loss, software is subject to these same conditions for both itself and the hosting platform, as well as a myriad of new risks. These can include deprecated support, software faults, resource use and hosting issues outside of its control. For example, VNFs are hosted on physical hardware configured to act as hypervisors and will share the resources of this host system with other VNFs, bringing with it the need to manage the sharing of resources where needed as well as enforcing the degree of isolation necessary depending on their use case. The resources of the network at large and each hosting platform must be taken into consideration when devising traffic

flows (Y. Chen et al., 2018) for efficiency, as well as survivability planning during regional failure scenarios (H. Yu, Qiao, J. Wang, et al., 2014, Lira et al., 2013, Abhishek et al., 2020) or localised redundancy (J. Xu et al., 2012) much like non-virtualised networks. Additionally, the logic of the control plane would necessitate significant monitoring and control capabilities, both desired mechanisms but difficult to employ, warranting further research. Vulnerabilities are not isolated purely to the forwarding plane, with the separated control plane posing a new area of risk for failure. Separation of the control plane through fault would equally disrupt operations of the forwarding plane, be it through paralysis of forwarding decisions or a slow deviation from intended operation (A. Wang et al., 2014). Despite these new risks, the flexibility of software brings with it new approaches not possible with hardware. Software can be instantiated, restarted or migrated far more rapidly than hardware, as well as modified and its internal values accessed. This allows for far more complex resilience mechanisms to be created versus the limitations of hardware, as well as the use of existing concepts such as consensus.

Consensus is an older concept of distributed computing wherein multiple processes agree to a single outcome of an operation to ensure a consistent data view across all processes. Examples of this in protocols include Paxos (Lamport, 2001) and Raft (Ongaro et al., 2014). These enforce synchronicity of state between multiple instances of a service to prevent conflicting state results from received input. Non-deterministic execution is a well-established problem for VNF resilience, not just in arbitrary replicas with enforced synchronicity but also across redundancies, which will be explored in Section 2.5. Consensus protocols predate NFV but have found significant use within a domain where replication is far easier to deploy. Beyond reused concepts, new techniques have emerged from the ease of instantiation and teardown. The potential to migrate or create new services where needed on the fly is unique to software and virtualisation, allowing for VNFs to be placed across the network between server platforms in accordance with the system's needs. This flexible network topology brings with it many possibilities for scalability and traffic rerouting (Hantouti et al., 2019) alongside the risks of software and increased complexity.

2.4.4.2 Reliability of VNFs

The reliability of an NFV deployment is complex to estimate due to the number of components involved, including the hardware resources, the Management and Orchestration (MANO) structure and the VNF itself. Modelling this requires the estimation of the Mean Time Between Failures (MTBF) of these components, described as a non-trivial task due to the number of components in operation at once in the environment, as even minor deviations can change these values. Physical Network Functions (PNF) are typically evaluated for this in an isolated offline environment,

but due to this interactivity, this is infeasible with VNFs. The reliability of a VNF is dependent on a wide number of factors, with an upper limit effectively established by the off-the-shelf hardware it is typically hosted upon, which is evaluated as only three-nines (ETSI, 2016) in the "nines" evaluation nomenclature (Dell, 2015). This is most typically expressed (in this domain) as the 'five nines' expectation, wherein a service is available for 99.999% of the time within the span of a year, or 5.26 minutes of downtime within this period. ETSI standards propose a number of management approaches to maintaining VNFs and when to replace them in their lifecycle (ETSI, 2014a) and handle faults when they occur (ETSI, 2014b).

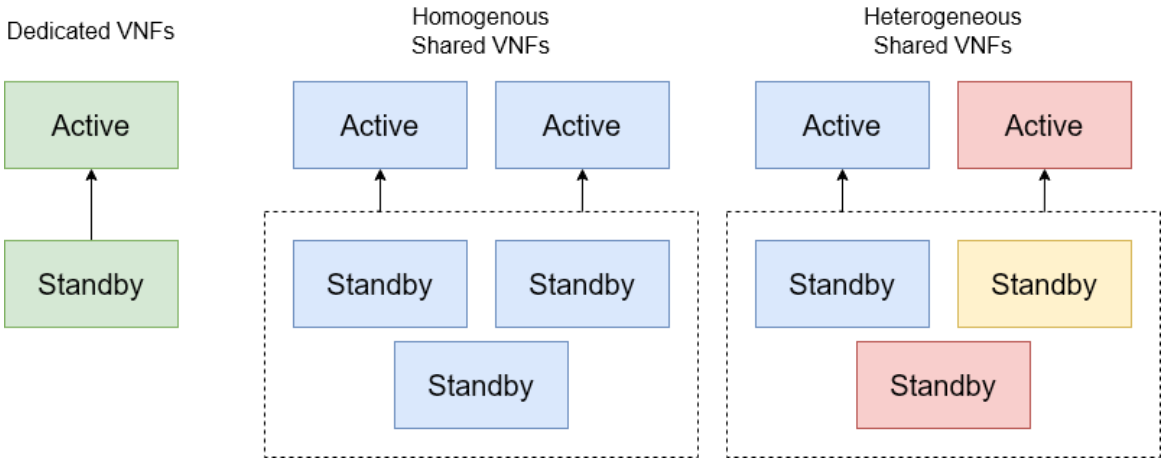

Figure 2.6: VNF protection schemes as proposed by Casazza et al., 2019 . Colours represent a VNF base image *e.g.* blue VNFs are identical to each other while red VNFs differ from blue

To accommodate for their potential lesser reliability and ensure continued availability, ETSI standards (ETSI, 2016) the concept of "primary/secondary" VNFs, or active and standby replicas. Protection schemes defined under this approach (Casazza et al., 2019) establish two approaches to redundancies depicted in Figure 2.6. The first approach, depicted on the left, uses dedicated redundant VNFs or standby that sit idle until the active fails, upon which they take over. This is a direct 1:1 redundancy, and is a very common method. The second approach, depicted in the middle and right examples, uses a pool of available standbys to be consumed by any of the linked actives upon their failure. These pools may contain homogeneous VNFs, or identical VNFs to the ones in active use, or diverse in their options. These approaches are akin to pre-existing protection schemes focused around network pathing, with the simple dedicated redundancy in an active/standby state being a well-established approach to recovery but incurring delay from potential startup. The shared pool of VNFs are presumably active to be selected at the point

of failure and to be migrated when needed. These pools may be broken down further based on active pool homogeneity or whether some measure of context switching is necessary for secondaries to the chosen active VNF.

These technical definitions set forth approaches to utilising VNFs and exploiting their ease of replication to overcome any perceived diminishment in their dependability and overall reliability during heavy load versus traditional hardware approaches. Distributed NFV systems introduce a higher probability of fault in their design, configuration and operations which legacy systems do not. These are not so much disadvantages to resilience in NFV and SDN as new areas that must be explored to determine the best practices and new techniques beyond what blackbox hardware can achieve. We shall explore more specific examples below in the related work.

2.4.5 Middlebox resilience

There is no singular resilience technique that fits all circumstances, especially in regards to middlebox usage. These approaches can be roughly divided into two groups: resilience in applications and resilience in infrastructure. Modern networks, such as those of cloud infrastructures, are complex, geographically disparate structures that carefully tune efficiency and resilience trade-offs in their layout and operations. For example, a significant degree of vertical scaling for a service would allow for greater resilience against unexpectedly high volumes of traffic, but in doing so would be inefficient; compute resources are left underutilised, raising OpEx where less resources could be used instead (e.g., 50% CPU utilisation vs 95% CPU utilisation) but render the system more inflexible to changing traffic conditions. X-as-a-service platforms and datacenters are another example of this effort to balance, allowing for greater efficiency through a centralised location for resource usage as well as greater potential redundancy through horizontal scaling possibilities, at the cost of increased risk against disaster scenarios due to their centralised point of failure.

Redundancy is the oldest and most simplistic form of resilient design (J. P. G. Sterbenz et al., 2010), utilising multiple copies of a point of failure to minimise potential inaccessibility and downtime by switching to the redundant (or alternative) entity when fault or failure disrupts operations of the original entity. For example, a WAN may considerably overprovision link capacity to handle link failures and traffic bursts, with more than double the necessary resources to function at normal levels of traffic (Jain et al., 2013). It is the most popular form of resilience for its simplicity and ease of deployment, both in hardware and software, offering a layer of protection against faults with minimal setup. More complex techniques that retain state, such as service migration or graceful failover, are more effective at minimising service disruption but necessitate careful planning and design or require significant overhead in link use, processing or costs. Cost is the defining factor when designing

networks, balancing between efficient design and resilience. Maintaining redundancies or complex systems increases the OpEx and CapEx of the system and is only necessary during periods of disruption, but the loss of network operations from disruption will in turn cost the business that operates the network. The more important resilience is for a network, the greater the cost of its absence, motivating this balance as well as the means by which it is pursued, with advantages and disadvantages to be found with each approach.

Middleboxes face their own issues with resilience, especially in relation to state retention. State in the context of middlebox hardware typically refers to the contents of its TCAM and lookup tables that are created by the middlebox to facilitate decision-making on incoming flows. As hardware, they are far less susceptible to fault than software and are typically perceived as high performance and reliable. Their closed or blackbox design masks their specific operations and the contents of memory to the network operators, intended purely to be dropped into the network and configured to requirements. This blackbox nature heavily contributes to resilience and operation issues with middleboxes, far more than transient network or hardware failure. A large-scale study (Potharaju et al., 2013) of middlebox failures identified that the majority of faults (40 to 80% depending on middlebox function) that occur with middlebox use are network rather than hardware-related, with transient connection issues accounting for 42% of failures whilst overloading and misconfiguration issues are far less common than thought at only 13%. They categorise failures for middleboxes as: connectivity errors (ARP conflicts, port errors), hardware failures (defective components), misconfiguration issues (bad rules, VLAN misallocation), software faults (firmware bugs, OS errors) and excess utilisation (high load exceeding capacity).

The type and severity of these problems are dependent on the kind of middlebox, with VPNs subject to the greatest degree of connectivity issues while firewalls are the most commonly overloaded. More importantly, this paper identified that primary:standby paired middlebox redundancy, the most common and primary form of middlebox resilience (Sherry, Gao, et al., 2015), is ineffective in 33% of cases for load balancers and firewalls, caused mainly due to misconfiguration and faulty mechanisms; these represent the most common cause of faults in middlebox hardware (Sherry and Ratnasamy, 2012). Many of these issues cannot be resolved through research and are still predominantly user error, with misconfiguration issues and connectivity problems occurring due to the sheer variety of devices and the complexity of their management and operation. The loss of state is thus uncommon, but the majority of middleboxes in use by networks are stateful, and the loss of this state is difficult to mask. Even transient faults can cause disruption (Allman, 2003), with long-term failures reportedly very costly for both the industry and middlebox vendors (Harris, 2011). The blackbox nature of this hardware greatly hinders the range of options for resilience and state propagation as a whole.

Few middlebox devices possess failover mechanisms and rely on complex internal state recovery, although it is difficult to ascertain their effectiveness without direct vendor information (Sherry and Ratnasamy, 2012). These are limited by the desire of both vendors and the industry to minimise added latency during failure-free operations; delay beyond even 1 ms is reportedly highly problematic (Sherry, Gao, et al., 2015). In short, middlebox users rely on good hardware design to minimise rates of failure and simple 1:1 redundancy to mitigate these failures when they do occur but are plagued by configuration and network issues. This is an expensive and inefficient approach to network resilience, with reported rates of hardware use equaling those of even routing backbones (Sekar, Egi, et al., 2012a) but current middlebox design renders improvement difficult to establish. NFV proposes their replacement but brings with it a new wave of complexity and potential issues. As a result, a new body of research has developed within the last ten years that seeks to integrate the two technologies and enhance their overall resilience in an area that is currently lacking. This is a difficult topic that this research attempts to mitigate, and it is also a major component of the focus of this thesis.

2.5 Related work

Blackbox middleboxes present a difficult problem for resilience. Middlebox failures are uncommon but significant problems, with the loss of state being the largest concern for hardware vendors. Any mechanism that attempts to mitigate its impact or likelihood is bound by strict requirements regarding latency, with a reported limit of 1 ms total overhead (Sherry, Gao, et al., 2015) for end-to-end communications that would be tolerated by both vendors and their users. This must also ensure the correctness of the restored state. This is complicated further by the closed nature of the hardware, which prevents all remote access or awareness of key metrics for state capture or recreation. The focus of this thesis in regards to resilience is ensuring this continuation of state across blackbox redundancies in an effort to improve upon the overall resilience of the network. Related research to this issue primarily focuses upon VNF and software-related middleboxes and can be roughly categorised by their own approach to this topic, each with its advantages and disadvantages, which we shall summarise in Table 2.1.

There are many approaches to state capture, retention and recreation. A key operational characteristic is whether data is replayed, and these approaches can be roughly divided into two groups: replaying and non-replaying. *Replaying refers to the capture of packets, traffic or the whole system state (Scales et al., 2010) to be replayed through a backup to re-establish the state through unmodified operation.* The correctness of this new state to the lost is highly dependent on the target device and the approach utilised for the checkpointing technique. *The second group, non-*

	Benefits	**Drawbacks**
Replaying		
VM Capture	Perfect state capture	Expensive, slow
Packet capture	No missing flows	Delay, Non-deterministic
Coarse logging	Deterministic execution	Delay to regular traffic
Fine logging	Low delay, deterministic	Requires modification
Non-replaying		
Live replay	Fast failover	Non-deterministic
1:1 redundancy	lightweight, fast	Loss of state issues
Controlled packet duplication	Low cost, no delay	Non-deterministic
Log interpreter	No delay	Requires modification
VM migration	Full control	complex, slow

Table 2.1: Comparison of state preservation methods from both past research and current practices, organised into replaying and non-replaying techniques

replaying, targets some form of output or live copy that does not rely on captured traffic (Dunlap, King, et al., 2002). These can vary in complexity but tend to be more targeted to the primary device, especially if it is interpreting output. Each of these approaches has both advantages and drawbacks, which shall be covered below in their own sections, with a final section on cloud strategies for this problem. With this in mind, we shall discuss past projects and their efforts in this domain.

2.5.1 VM capture

VM capture uses the ability of modern hypervisors to checkpoint a VM's state at runtime, specifically its active memory. This encapsulation has awareness of the whole system's memory and state, but in doing so, it introduces a significant level of abstraction that may impair performance. Remus (Cully et al., 2008) is an early work in virtualisation resilience that focused predominantly on a generic solution that requires no significant retooling towards a specific process. This is achieved through the exploitation of Xen's (Barham et al., 2003b) in-built checkpointing mechanism; intended as a snapshotting mechanism for periodic backups, Remus has optimised the process to convert it into a live checkpointing service. The target software is encapsulated within a VM and paired with a redundant live VM through an active-passive arrangement. To mitigate the potential delays caused by repeated snapshotting on the primary VM, as is typical with services that await confirmation from the auxiliary it has received it (Bressoud et al., 1995), Remus disregards enforcing

deterministic execution to minimise this delay. This whole system checkpointing guarantees that all visible state, that is, state that would be observed without direct observation of the internal mechanisms of the software, is captured within the Remus checkpointing, which occurs every 25 ms. While this may not be relevant to middlebox work due to the VM targeting and excess delay, its design predates middlebox concerns and represents a significant effort in attempting to derive state from outside sources that is both generic and applicable to any and all software that can be encapsulated.

2.5.2 Packet capture

Packet capture is the use of packet traces to regenerate state through the replaying of traffic, repeating the actions of the middlebox to simulate the original events. This approach is less impactful on overall performance than VM capture. Pico (Rajagopalan, Williams, and Jamjoom, 2013) is a later evolution of the work established by Remus and targets middlebox VMs specifically, such as Suricata (O. I. S. Foundation, 2022) and Snort. Rather than targeting the entire system state, the paper identifies traffic flows as the primary observable state to be preserved. From this, state is further divided into two groups: per-flow state retained by the middlebox and static configuration information. This per-flow data represents the key targeted state. By observing the majority of state is established at flow establishment, such as stateful NAT, these flows can be identified and replicated from the target VM to a buffer before distribution to multiple replicas. The frequency and classification of this is defined by the group they are organised into, dictated by Pico's configuration. This work builds upon their earlier work Split/Merge (Rajagopalan, Williams, Jamjoom, and Warfield, 2013) that identifies the per-flow state for VMs, evaluated against the Bro IDS (Paxson, 1999). Upon the failure of the primary packet processor, the SDN controller for the network will redirect the buffered standby flows to the other replicas. This approach is both highly available and significantly generic, whilst also radically reducing the delay caused by checkpointing caused by Remus down to 8.5 ms and represents a significant improvement in refining the necessary state from VM-level to flow-level. However, it still suffers from a significant level of delay from the replaying approach to high availability from the buffered flows to the new replicas, as well as primarily targeting software over hardware.

2.5.3 Coarse-grain log-based checkpointing

Log-based checkpointing generates a log that tracks a specified set of values in accordance with the research's methods. These logs are then used to replay the traffic or actions taken. Coarse-grain logging consists of values that refine the selection process to a moderate degree. For example, maintaining a log of a subset of packets

relevant to replaying state rather than retaining all packets. SMP-ReVirt (Dunlap, Lucchetti, et al., 2008) is one of the earliest papers to pursue execution replay at a finer grain for application-level checkpointing. With awareness that multiprocessor systems were increasingly the norm within the 2000s, execution replay as a means for recovery was becoming more complex due to the potential for thread racing and shared variables. This approach utilises a coarse logging system combined with replaying, with specific events recorded for replaying while others are ignored to minimise data storage needs and further delay. ReVirt categorises the types of events to be recorded as deterministic or non-deterministic. Deterministic actions such as branch instructions of memory reads are unrecorded in the logging system, as they will execute as they did before regardless of when traffic is replayed. Non-deterministic actions, or asynchronous events, are far more important and will not necessarily occur as before when replaying. These consist of events such as system clock reads, which cannot be recreated through replaying, or virtual interrupts; the state change caused by the interrupt is deterministic, but the point where it occurs in the timeline of events is non-deterministic.

Thus, ReVirt records only the order of execution between co-processors, avoiding per-instruction logging as well as data stores to further minimise necessary storage. This is done via the "Concurrent-Read, Exclusive-Write (CREW)" protocol (LeBlanc et al., 1987). Implemented in Xen through observation of its shared page tables for memory access awareness at the hypervisor-level, performance was mainly evaluated at the rate at which these logging techniques would consume storage, with the observation that the greater the number of coprocessors, the greater the volume of logging necessary to allow for accurate recovery during replay. The overhead is suggested to be negligible in most cases, with a reported 12% during kernel-based events and 5% for database events. This overhead rapidly rises depending on the number of shared variables and thread events, observed less in packet delay and primarily in the size of its storage requirements. Logging itself incurs a time delay from overhead of up to 8% per packet. Finally, it concludes with awareness of its limitation solely to software but proposes a more hybrid approach with the employment of a different approach to logging that is more generic and accessible to blackboxes. The benefits of this approach are clear, but it requires a significantly invasive technique to achieve these results, as well as significant tuning to the exact operations of the target to specify where deterministic and non-deterministic actions occur.

2.5.4 Fine-grain log-based checkpointing

Fine-grain logging is a refinement of the same principles as coarse-grain log-based checkpointing but records and controls the replayed events in far greater detail, such as thread accesses and the specific path that each packet takes between context switches.

This is done to minimise non-determinism completely, but is difficult to do both quickly and efficiently. Fault Tolerant MiddleBox (FTMB) (Sherry, Gao, et al., 2015) argues that non-deterministic execution of packet processing in software middleboxes impairs the effectiveness of prior work in this domain, including both live replica systems where deviation is not controlled for and checkpointing systems where delay is introduced by excessive attempts to control it. Prior work in this domain has struggled with this issue, with heavy-handed approaches to multiple replicas such as Colo (Dong et al., 2013) shifting the delay caused by approaches such as Remus to the replicas only. Non-determinism in this instance is defined as thread racing and hardware values that cannot be repeated, such as clock times. Instead, FTMB establishes Packet Access Logs (PAL), lightweight recorded execution logs of packets and variables as they are acted upon by the multi-threaded CPU.

These PAL are maintained as ordered lists of every thread interaction with a variable as it occurs in the system, which in turn is tied to each packet that it acts upon. This is done to control for non-deterministic execution by replaying all thread actions and packets as they occur upon failure, while also minimising the extent of delay necessary for the checkpointing necessary on a per-packet basis. As a replaying approach, delay is incurred by this start-up of the recovery VM, measured at an average of 275 ms. Checkpointing during failure-free operations averages far lower at only 30 microseconds. This approach is intended to be a fast and generic solution to recovery and deterministic state preservation across multiple VMs rather than more typical active:passive/1:1 pairings, with PALs and recorded packets retained in a separated middleground datastore. Overall, this approach is significantly faster than Pico, Colo and Remus while attempting to allow for the significant potential of scaling recovery of VMs. However, it does necessitate the modification of the target middlebox software to allow for these observations to be recorded, as well as offering little in the way towards middlebox hardware.

2.5.5 VM Migration

VM migration is the practice of transitioning an active VM from one hosting node to another in an effort to preserve its continuity against disruptions to the hosting platform. VM migration is generally limited in scope to company-specific implementations, working within their own ecosystem with no interoperability. This is due to the complexity of the task involved, both in the need for control over the VM space for instantiation and translation and also in the network itself. More generic solutions typically concern hosting-specific mechanisms, such as VMotion in VMware. As a resilience mechanism for middlebox platforms, older examples such as the work by C. Clark et al., 2005 serve to demonstrate the difficulties associated with live migrations, including but not limited to the necessary constraints employed by

active memory use and disk accessibility. As discussed in this paper, live VM-level migration requires the migration of memory and active connections, with storage clusters utilising uniformly available NAS to sidestep this issue. Complications in the page copying are primarily focused on "hot" pages, where memory is active and changing at a rate too frequent to allow for significant delay. By defining an active window of time on a per-VM basis, page pre-copy prior to shutdown can greatly diminish the overhead and the rate of new copies necessary by changing pages. Depending on the application, downtime is measured in the tens of miliseconds when suspension occurs for final transfers of the most at-risk memory. Live connections are transferred via ARP adverts indicating shifting IPs. This approach is limited due to its need to establish shutdown windows, inability to handle VNF failures and poor scalability. Modern networking environments no longer employ the use of VM or container migration, instead instantiating elsewhere for load balancing or disaster recovery and are rarely used against live services due to the constraints involved.

2.5.6 Live replay and simple redundancy

Live replay is the practice of duplicating a stream of traffic directed to a service and directing it to a copy of this service, maintaining a 'hot replica' (Juan-Marin et al., 2007). It is a costly technique resource-wise, effectively doubling hardware, processing and bandwidth usage in the low probability failures occur. Redundancy is a step beneath this, simply consisting of redundant links or hardware to utilise in the circumstance of failures should they occur with no other mechanisms in place. Both live replay and redundancy are standardised techniques for improving resilience, as discussed in the prior background. This is the most simplistic approach, with the least investment required in complexity, but leaves all recovery exposed to non-deterministic execution and lost state. Many platforms have built-in configuration tools for paired redundancy deployments, both at the router and middlebox level. This includes generic networking protocols such as Virtual Router Redundancy Protocol (VRRP) and proprietary mechanisms like Cisco's Hot Standby Router Protocol (HSRP) (Li et al., 1998). For more failover at the middlebox level, this is primarily isolated to proprietary platforms, much like VM migration, due to the necessary internal control to allow for a smooth transition. Many of the larger hosting platforms for containers and VMs have integrated systems for failover and high availability, or at the very least redundancy. Examples include AWS elastic scaling (Amazon, 2019) which allows distribution across multiple geographic locations to enable high availability and automatic scaling; DigitalOcean's Reserved IP (Ocean, 2022) (a form of virtual IP) that implements active:passive failover and VMWare's in-built vPAC active-active redundancy support (VMware, 2023).

2.5.7 Service Function Chain techniques

Service Function Chains add an additional layer of complexity to the resilience techniques discussed thus far. As chains of virtual instances, they are hosted on physical servers and logically connected via traffic forwarding and flow tagging. The state of packets in these chains is dependant on every link in said chain operating correctly, with deviation from expected behaviour or breakage affecting every other link. Furthermore, while multiple links in a chain may be hosted on the same platform, it is not uncommon for instances to be spread across multiple physical hosts. This greatly limits the feasibility of many of the techniques discussed thus far. There are several bodies of work within this area that focus on moving beyond 1:1 redundancy, expanding the awareness of state from each individual entity to provide failover to an entire chain. CHC (Khalid et al., 2019) establishes the concept of Chain Output Equivalence (COE); given a stream of packets as input, a chain with many instances per NF must produce the equivalent output to a hypothetical equivalent chain with individual NFs of infinite capacity. This is complicated by the statefulness of VNFs potentially influencing the correctness of processing across multiple varying instances.

Individual approaches do not consider shared state, impairing their effectiveness even when working on every NF within a SFC. The CHC, or "correct, high-performance chains" framework defines logical chains of NFs using the DAG API that are utilised by a manager entity, directing traffic between clones for each NF and collecting statistics from each instance. The kinds of state for each node are classified by order of the least to most amount of information necessary to ensure correctness of recovery and organised by flow to be directed to NFs in order to minimise overlap across instances that modify on the same criteria. This extracted pathing and state information is retained in a separate data store. This approach highlights the meta-awareness of the framework necessary to operate and maintain correctness of recovery across an entire chain, taking into account the differing state requirements, the potential for multiple replicas of each NF and their interactions. The latency caused by the overhead of these mechanisms is varied, with hardware clock logging measured on a CPU-based testbed at 3.5 microseconds and packet logging at 34 microseconds.Full recovery from faults can observe latency spikes measured within miliseconds, depending on the current network load.

Another project, Reinforce (Kulkarni et al., 2020), differs in its approach by utilising a similar understanding of deterministic and non-deterministic actions to SMP-Revirt, relying on programmer annotation to indicate state operations that are non-deterministic by a NF. Logging is maintained for both application state and packet positioning within the SFC. When a labelled action is performed by a member of the chain, it is linked to the packet that triggered it, with this information buffered until sufficient information is retained to restore a checkpoint for the entire chain. Overhead analysis suggests non-deterministic checkpointing in this two-stage

approach incurs a per-packet latency of 8 microseconds on failure-free operations within the local node, with far greater impact on remote nodes. Non-deterministic checkpointing is far more costly and scales poorly with both frequency of rate and the length of the chain, but its lazy replication approach is a useful demonstration of how state is initialised by the beginnings of flows and how performance can be achieved without direct modification of the target platform or per-thread logging.

2.5.8 Cloud-based approaches

Other work in this domain has also attempted to tackle the issue of middlebox resilience and the difficulties surrounding it. While the primary comparisons we make typically favour their replacement with virtualisation or focus purely on software instances of middlebox usage, other work has attempted to do away with middleboxes entirely. One example of this is Appliance for Outsourcing Middleboxes (APLOMB) (Sherry, Hasan, et al., 2012) that proposes the outsourcing of middlebox usage to cloud computing, motivated primarily by an OpEx and CapEx argument for in-house management and operation costs. This proposal attempts to tackle the issues of redirecting traffic that would normally be handled at the local entry points or chokes within the traffic flow, as well as the vetting and management of this additional necessary complexity for maintaining key gateways. Furthermore, it also attempts to design the platform through which middlebox services would be rented; more specifically, the business logic of policy chains and how these purchased services would be facilitated, alongside the necessary scaling of middlebox availability.

A similar concept is demonstrated in an earlier piece of work by Sekar, Ratnasamy, et al., 2011 that argues the innovations in virtualisation in 2011 could be radically expanded upon in support to allow for innovation within this domain and prevent the ossification of protocol and function that hardware middleboxes incurred. They argued this could be achieved not with individualised entities like a WAN optimiser or IDS virtual instance, but instead with discrete blocks of processing logic that could be utilised across multiple "middleboxes" in a fashion akin to unikernels and their division of the kernel into smaller discrete units. Another similar project to this is Network Service Support (NSS) (Panda et al., 2016), itself heavily motivated by the push for processing closer to the network edge by carriers and CDNs to minimise transit and consolidate processing within centralised locations, as well as Akamai and Amazon's EC2. As a final example, Consolidating Middleboxes (CoMb) (Sekar, Egi, et al., 2012b) seeks to address the management issues of multiple discrete boxes by consolidating all possible individual middlebox functionality into a singular local node, akin to the abstract concept of NFV.

2.6 Summary

The increasing rate of SDN adoption in modern networks has motivated a large body of new research dedicated to the resilience of software-based VNFs, with the progressive replacement of bespoke hardware and their approaches to countering non-determinism. The presence of hardware middleboxes, or blackboxes, has become an increasingly obvious problem however. The discussion of this background has made clear the evolution of middleboxes and how they came to dominate a large section of networking, and with it introduce a distinct weakness of resilience in their closed nature and non-standardised designs. Replacement is one solution, although this brings with it its own problems of performance loss and system alteration. Operators and enterprise infrastructure design decisions are motivated primarily by cost. Capital and operating costs (CAPEX and OPEX, respectively) typically dictate the design of a network, which shapes the focus of its development. For example, high performance is heavily favoured as it is a part of competitive business practices and SLA, so it takes priority over redundancy and resilient design. This is best depicted by the widespread use of blackbox hardware, despite the complications to resilience it may introduce. This is not to suggest that resilience is not important, as recovery functionalities for middleboxes can be bought at extra cost, but that the cost of enabling greater resilience may outweigh the cost of handling disruptions when they occur. The majority of research discussed so far concerns itself with mitigating the performance impact of new resilience techniques, but their focus is almost exclusively on open software. Enterprise more commonly use purchased solutions to enable network functionality, the products of which are blackboxes such as precompiled software with obsfuciated internal code or fixed hardware solutions. These blackboxes are employed in significant numbers and are unlikely to be replaced entirely within the near future.

There are a wide myriad of strategies to approach the topic, each with its own benefits and drawbacks, each highlighting the difficulty of this task. Many of these are not possible without direct access to the hardware, including both the hosting platform for VM or application capture and the fine-grain logging of FTMB for system clock and thread events. The ideas evaluated present a clear picture of the scale of the problem and the potential work that still needs to be done around blackbox middleboxes that are unlikely to be replaced in the near future. These cloud-based solutions each argue for the issues of middleboxes to be resolved through their replacement with software and, in doing so, attempt to address many of the issues associated both with their use and their replacement with VNFs. This is a common trend in middlebox research: hardware is limited in the scope of what it can achieve, with many network functions deployed in software when its operations exceed that which is possible with physical devices. Those that are still realised with hardware, however, such as the case with firewalls or load balancers, will remain in place as the core issue of performance loss is

not addressed by any of the differing arguments that advocate for the same concept as shown by these papers. CoMB argues for consolidation with virtualisation for management purposes, NSS for scaling, and APLOMB for operating costs. While there are many benefits that could be realised with a rentable processing service akin to other web services, there is still a clear gap displayed in research: how do we enhance the resilience of hardware middleboxes without arguing for their total replacement?

Chapter 3

Hybrid networks and resilient design

With the background of the problem now made clear from the previous chapter, this chapter will go into detail on the motivations of this work, its technical arguments and a design for a generically applicable middlebox resilience framework. Firstly, section 3.1 will discuss the motivations created by the understanding of the work covered in the background and related work, discussing the CAP theorem, the limitations of ASIC hardware and the current use of hybrid networking. From these motivating arguments, section 3.2 establishes a set of high-level design requirements that any proposed design must adhere to, followed by section 3.3 which discusses potential approaches in an attempt to meet these requirements. From here, the argument ends and the design overview begins in Section 3.4, establishing the work of this thesis: a generically applicable middlebox resilience framework, able to support both software and hardware-based blackbox services for the purposes of retaining state across failover. This is achieved through the use of a range of resilience mechanisms, including the use of different types of drivers for different targets. Finally, this chapter concludes with a summary of the arguments and designs presented in Section 3.5.

3.1 Argument

The previous chapter and its summary in Section 2.6 has established a clear definition of the scope of the problem. With existing platforms unlikely to change and incapable of being modified, the remaining approach is to enhance the resilience of platforms without requiring modification or replacement through the incorporation of software alongside existing hardware. This serves as the motivation for the work of this thesis, attempting to answer this stated problem while adhering to the strict limitations that dictate the feasibility of any real solution. Building on this motivation, this

section will establish an argument for the technical considerations of how this might be achieved, beginning with a description of the combined use of virtual networking techniques alongside legacy networks, known as hybrid networks.

3.1.1 Hybrid Networking

A hybrid network is the combination of programmable infrastructure with legacy networking hardware. This typically includes the presence of compute resources for virtulisation purposes and SDN-enabled hardware alongside standard network devices such as switches and routers, depicted in Figure 3.1. The degree of softwarisation varies between networks, ranging from routers equipped with some degree of virtualisation cabability to enable new network functions to employing the use of SDN-enabled routers to control the flow of traffic with legacy switches remaining in-between. Hybrid networks are not so much the goal but rather a transitional step. They act as an intermediate architecture between legacy network technologies and entirely programmable and virtualised devices that will progressively replace them. These changes do not happen all at once but instead occur slowly as networks grow and older hardware breaks to be replaced with newer components that incorporate programmable networking, forming these hybrid networks. An example of a deployment strategy for hybrid networks is Panopticon (Levin et al., 2014): a hybrid network that deploys SDN-enabled switches at key locations in the network, allowing for traffic flows to be controlled on a network-wide basis despite only controlling 10% of the hardware present. Despite this growing rise in SDN adoption in enterprise networks and the increasing hybridisation of networking, this thesis takes the position that blackboxes will continue to be used and even grow in usage due to their high performance and inability to be easily replaced, like routing infrastructure. Given this, this thesis further claims that by exploiting network programmability, these blackbox devices need not be replaced but instead aided by softwarisation techniques to enhance their resilience. This in turn allows for the continued operation of existing hardware without necessitating its replacement and encourages the continued growth of SDN adoption in enterprise deployments.

Programmable elements allow for a degree of freedom not available to proprietary software or hardware appliances, enabling functionality such as maintaining and recreating state to be built as separate mechanisms in and around this middleware. By maintaining the existing infrastructure in its place, this also minimises disruption and maintains expected levels of performance during failure-free operations. The state must be quantified and replicated across multiple instances, with SDN serving as both the means by which this is achieved. NFV can also be utilised to provide redundancy in the absence of existing infrastructure using VNFs modelled to replicate behaviour, serving as a remediation layer approach. The lower performance of

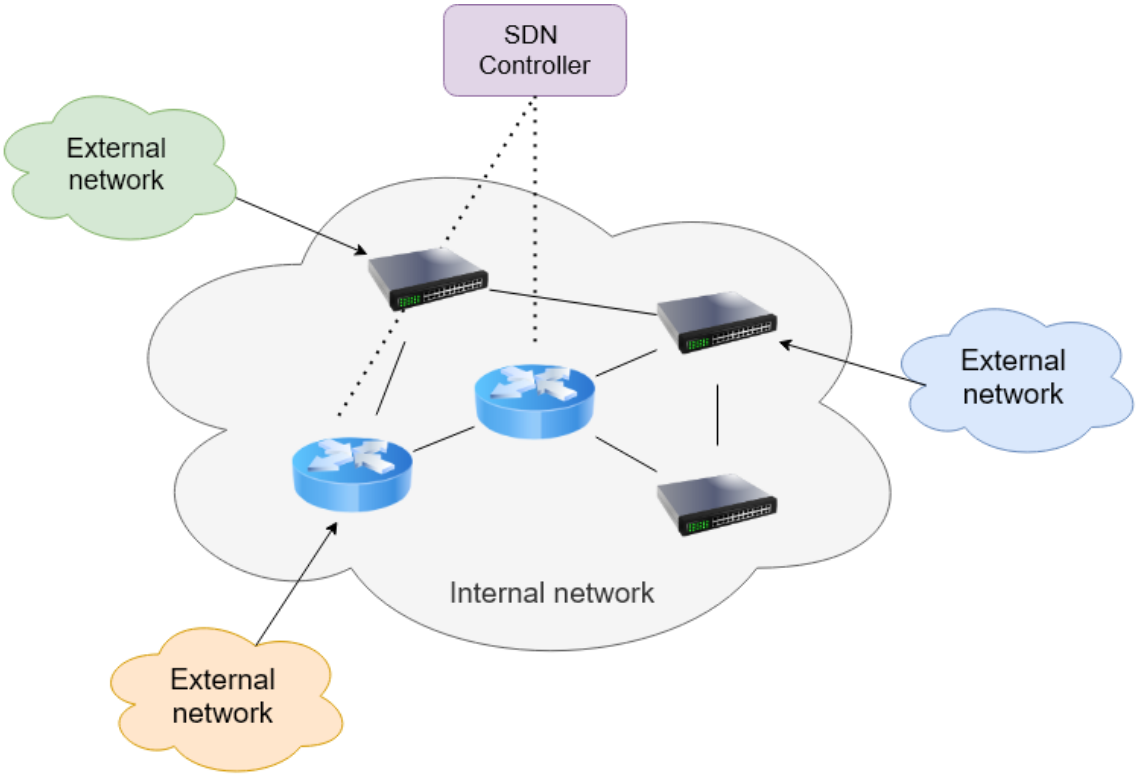

Figure 3.1: Example of hybrid network architecture. VNFs are typically hosted on SDN-enabled routers to provide network functionality.

software-based approaches is less important during this stage of the recovery cycle, acting as a stop-gap solution while engineers handle the fault. To accomplish this non-interfering state replication and enhance their resilience without modification, a thorough understanding of the technologies used in these blackboxes is required.

3.1.2 ASIC Hardware

Application-Specific Integrated Circuit (ASIC) are the most common hardware realisation of these hardware middleboxes still in use today. These are Integrated Circuit (IC) designs for traffic processing, offering fast implementations of a fixed set of network manipulation instructions (fixed-programmability). Unlike general-purpose CPUs, they are optimised to process traffic and have a tight integration only with ports, able to support traffic rates at speeds upwards of Tbps. These rates are an order of magnitude beyond other solutions such as FPGAs (Zilberman, Audzevich, et al., 2015) and hardware-offloaded virtualisation (Sarrar et al., 2012) such as DPUs (Nvidia, 2022). ASIC design is driven by a single pipeline of processing stages

performed sequentially, with each stage consisting of multiple parallel processing steps. With most network devices, these steps are a match-action design; the header of incoming packets is parsed for its contents, with actions enacted on the packet decided based on the contents of said header. The complexity of these steps, especially that of header parsing, varies from device to device and has an effect on its degree of reconfiguration.

The more simplistic a device, the fewer options are required, encouraging manufacturers to design products to fit their intended use so as to keep both manufacturing and energy costs low. More simplistic ASICs have fixed configuration, providing a closed feature set that can be selected to act in the pipeline, as showcased in Figure 3.2. The trident series uses a tile-based architecture which consist of a lookup table and policy, as well as the memory it acts upon depending on the product in question. *i.e.*a TCAM tile in the egress pipeline would be a predefined policy and lookup to the TCAM memory at that stage of the packet processing pipeline. These are reconfigured with simple registers but offer nothing beyond enabling or disabling a step. More commonly, configuration is far broader through the use of tables, similar to SDN; table entries define the pairing of match to action. The size of these tables greatly affects the design of the device, as they are implemented in memory. This reconfiguration is generally performed through the coprocessor, facilitating monitoring and operating configuration agents. As networks have developed, manufacturers have moved away from proprietary systems that maximise performance (Marvell, 2024) towards well-defined RISC architectures, standardising their platforms and streamlining the hardware whilst still allowing for some measure of programmability.

Operations in the pipeline are assisted by the Ternary content-addressable memory (TCAM), a high-speed memory that can search every entry within a single clock cycle. TCAM is highly parallel memory, often used for lookup tables, forwarding and similar operations that mandate some form of basic memory. This memory module is expensive due to the increasing cost of both silicon and energy consumption incurred by expanding the volume of parallel circuitry, encouraging an efficient approach to TCAM usage. Rule compression and pre-classifiers are common features to reduce the consumption of memory, especially by rules expressing a range that must be expressed as each possible header. Wildcard matching is performed by the TCAM, able to perform lookup and classification against every possible entry within a single clock cycle. Non-wildcard matches such as exact matching do not need TCAM to operate, instead using hashing mechanisms. The second source of state within these hardware platforms is dependent on the functionality of the device. Forwarding or routing will depend upon the Forwarding Information Base (FIB) and Routing Information Base (RIB) respectively, with FIB retained in the coprocessor and RIB in TCAM. Calculation and installation may be handled by the coprocessor however, with some of this address information retained in more traditional memory. Alternatively,

PISA architectures such as Intel Flexpipe and Tofino may retain different memory approaches for matching (*e.g.* exact matching using hash tables in SRAM, wildcard in TCAM).

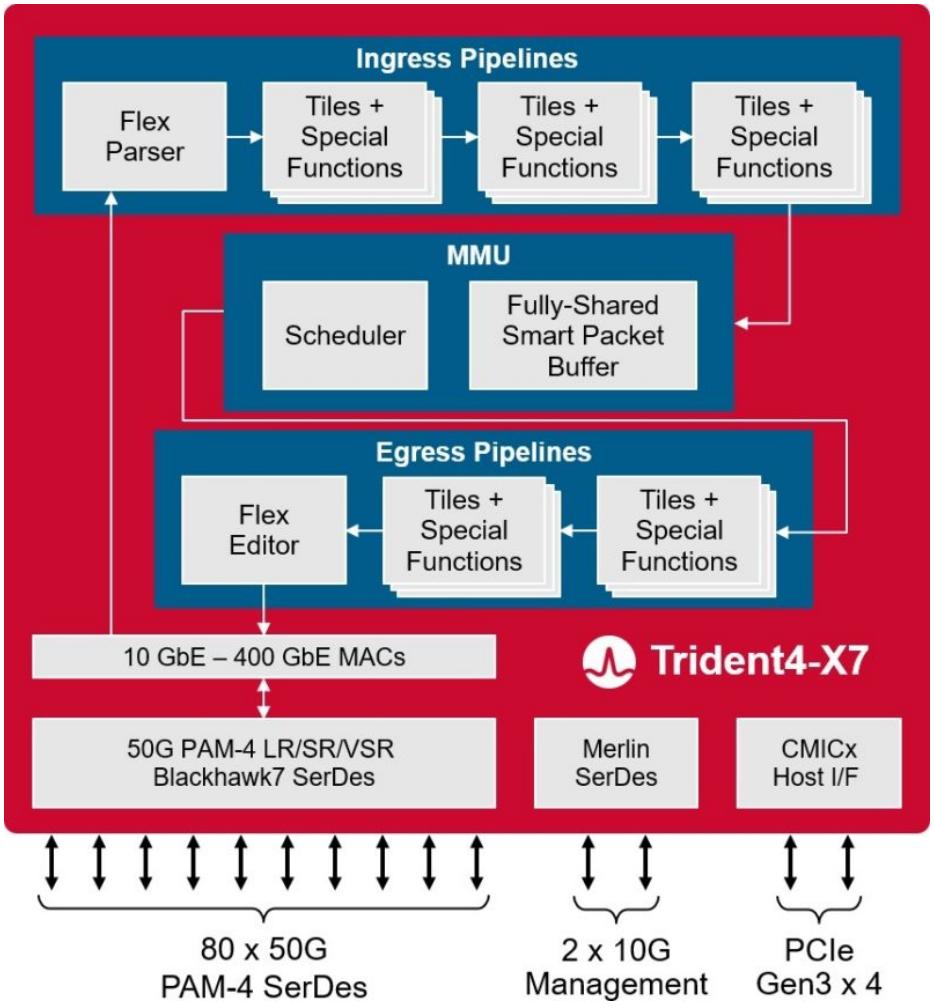

Figure 3.2: Broadcom Trident architecture - Examples of pipelines of interchangeable built-in functions (Broadcom, 2024)

3.1.3 Replicating state in ASICs

ASIC design is fundamentally decided by business more so than technical limitations; as purpose-built products, they have long development cycles to fit them to their intended purpose or market. Resources are limited on the silicon, including the number of logical circuits, the space on the die and the number of pins; increasing one

resource will consume space utilised by another. Other complications include total power consumption, cooling and overall production. Smaller semiconductor sizes are produced by fewer facilities and thus reserved for the largest businesses with the greatest capital to purchase them, as well as the cost difference between standard designs and more bespoke commissioned work. These motivations, alongside their more technical limitations, render blackbox middleboxes expressed as ASICs high-performance but simplistic in their operations; state is expressed only in the form of lookup and forwarding table entries, with little flexibility to allow for more advanced packet processing that requires a coprocessor. Emulating the functionality of an ASIC, if not the performance, is thus easily achieved in software. This is demonstrated by the increasingly common software alternatives to popular network functions currently provided by dedicated hardware, often sold in conjunction (Cisco, 2020). Due to their blackbox nature and their re-use on several products, it is difficult to determine the exact operations made on a per-device basis, but certain assumptions can be made. As specialised hardware with minimal internal memory, lacking both a heap and a stack, there are both technical and economical reasons to minimise the inefficiency of its internal memory usage. TCAM is expensive, while DRAM and SRAM are scalable but costly, both in energy use and orders of magnitude slower versus TCAM. Pseudo-random generation requires multiple clock cycles to generate, incurring latency, which runs counter to the key focus of ASIC design of speed and high-performance.

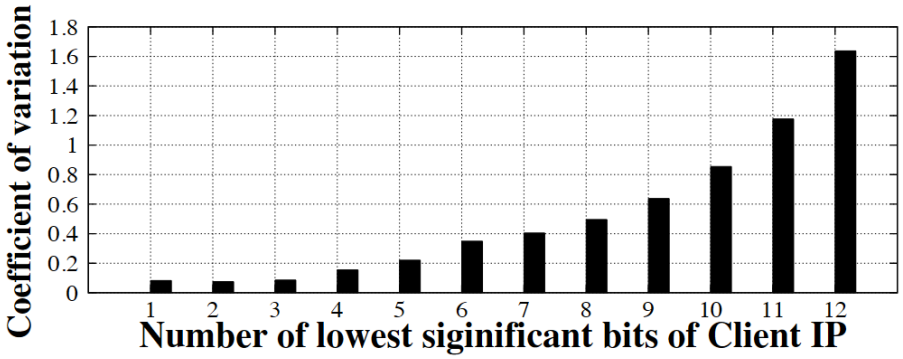

Figure 3.3: Coefficient of variation of fraction of requests for different bits of the client IP address (Kang et al., 2015a)

With this understanding in mind, the state and loss thereof of these blackboxes can be broadly predicted for the format it will take and especially its origin; with a lack of randomisation provided by hardware or memory to waste, it is instead entirely derived from the packet contents themselves (*e.g.* traditional five-tuple) allowing for state to be easily replicated with this same traffic. Any and all of the five-tuple can be used, although different fields from the five-tuple offer different degrees of entropy.

i.e. Niagara (Kang et al., 2015a) split its traffic proportionally by utilising the low-order bits of the source IP as they provide the greatest entropy for its load balancing, depicted in Figure 3.3. Furthermore, non-deterministic execution that would typically affect threaded operations, such as the order of thread accesses for a shared resource, is not an issue for single pipeline devices. The operations of each stage may be performed in parallel to hasten operations, but each stage follows consecutively to the next (*i.e.*header parsing may independently create metadata regarding decisions for a packet at different stages of the parsing, but the parsing still operates sequentially).

This, combined with their reliance upon the contents of packets to provide sufficient randomness, affords us the opportunity to pursue a form of "lazy correctness"; if state is both deterministic and derived from capturable traffic, a best-effort approach to regenerating state with no guarantees on enforcing correctness is significantly more viable. This means the concern for the loss of state is its effects on latency and perceived failure by end-users. The state lost by the failure of the ASIC, including the contents of these lookup tables, can be regenerated rapidly by the hardware itself, even if handled by the coprocessor. However, any time spent recreating state, especially numerous active flows all at once, will still create unnecessary latency. Instead of allowing this to occur, this determinism can be used to establish a live replica and minimise or even remove this delay. As a final note, its loss is still dependent on both the traffic protocol and the operations of the rest of the network (*e.g.* Timeouts and mass restarts causing surges of traffic).

3.1.4 CAP theorem

The argument on the nature of hardware internal execution, hybrid networks and the consistency of its state can be framed using an existing theorem on distributed systems. CAP theorem (Gilbert et al., 2002), also known as Brewer's theorem after its creator Eric Brewer, states that any distributed data store can only provide two of the following three guarantees: Consistency, Availability and Partition tolerance. These guarantees form the core systemic requirements of a distributed system that work in tandem, and shall be briefly summarised:

- **Consistency** - The operations of the system are predictable and consistent with no deviation from expected behaviour or results. In a distributed environment, this consistency is the assurance that each of the partitions is observing the most up-to-date and identical state/message as the other partitions.
- **Availability** - The operations of the system are functional and working as expected at any arbitrary time when called upon. In a distributed environment, this availability is the assurance that the system will always service the request given to it when prompted.

- **Partition tolerance** - The operations of each partition continue to function independently of other partitions regardless of fault or miscommunication. In a distributed environment, this partition tolerance is the assurance that regardless of the number of dropped messages, disruptions or partitions present, the system will continue to be able to operate. Unlike the other two guarantees, partition tolerance is exclusive to a distributed environment and is the primary cause of complexity in assuring the other two guarantees.

When considering state across multiple middleboxes, these same principles apply. State is a useful tool for packet processing, but its use has trade offs. The work discussed thus far has prioritised consistency over availability when handling network partitions, which in this context refers to parallel nodes and redundancies. Ensuring the consistency of recovery is important, but it has become the prevailing concern of research in this field, whilst industry has prioritised performance and availability above all. External systems, such as a remote database that all replicas draw their state from, can help to guarantee consistency and partition tolerance for these replicas, but introduces overheads and creates a new vulnerability in the database itself - if it fails, then the partition tolerance is lost. CAP theorem presents a challenge for software middleboxes in regards to state management, with both consistency and partition tolerance requiring systems such as Paxos or leader-election processes to enforce a single view of the system state on all partitions present (Kang et al., 2015a, Kang et al., 2015b). In enterprise deployments, either consistency or availability are traditionally sacrificed (Gilbert et al., 2012). Physical middleboxes sacrifice consistency to minimise the introduction of overheads incurred by enforcement of consistency. Alternative approaches have formed that seek to drop both of these in favour of partition tolerance. Chubby is a coarse-grain lock service, built by Google, that acts as a distributed database with consistency enforced using Paxos and replicated state machines (Burrows, 2006). Chubby can be partitioned into cells, each of which would encompass a datacenter, expanding the scope of its communication to sacrifice partition tolerance at a local scale to guarantee availability and consistency. This theorem holds great significance to the growing use of SDN and NFV as it introduces the potential problems of non-determinism at a significant scale from what it seeks to replace. Problems such as these, combined with the inability to match packet processing rates of dedicated hardware may be a contributing reason as to why hardware middleboxes are still widely used in production.

3.1.5 Summary

To restate the problem, NFV and SDN have seen slow but steady adoption, but have been limited in certain kinds of networks such as enterprise and business deployments where hardware solutions remain steadfast. These blackboxes, or “middleboxes”, are

widely used in modern networking for their high-performance and general reliability, but are limited in scope when failures inevitably occur due to their closed nature. The flexibility of software in regards to innovating design has seen NFV gain in popularity as a means of deploying network functionality, but hardware middleboxes will continue to remain dominant despite their limitations as they can provide competitive levels of high-performance for the businesses that operate customer-facing or enterprise networks. Research in recent years has proposed a number of new solutions to enabling greater middlebox resilience, but seen little to no adoption.

These approaches propose modification or replacement of existing infrastructure due to the difficulty of the task when dealing with blackboxes, and inevitably fall short of the arguments of cost. Replacing existing infrastructure, as well as sacrificing orders of magnitude performance in the name of resilience is an untenable position in the eyes of enterprise. Given our understanding of the limitations of ASICs in regards to state for blackbox middleboxes, as well as a lack of significant enforcement for consistency in software blackboxes in industry, a new argument and approach to resilience may be viable. This thesis surmises that a 'best-effort' approach to resilient design is feasible using SDN and NFV in conjunction with unmodified middleboxes, encouraging its further adoption in the areas of both research and real world network adoption where it has otherwise languished. This section summarises the argument and position of this thesis in its intentions, as well as the existing hole in research and industry for this approach to resilience.

3.2 High-level Requirements

With the scope of the problem area established, this section will highlight several key limitations to enabling greater resilience to this area of networking. These blackboxes are too widespread and ingrained to simply be replaced at the sacrifice of performance in the name of recovery, but their closed nature preventing modification and awareness of internal logic greatly inhibits research efforts to enhance their resilience. With this in mind, any recovery or remediation approach proposed would have to abide by a strict set of criteria:

- **Able to target both software and hardware**
  Middleboxes exist in both software and hardware in modern networks, each with its own place and use. For this work, these take the form of blackbox hardware, which is completely inaccessible, and greybox software, which offers some mild openness. Creating a redundancy platform able to provide state-aware redundancy should target both forms of middlebox to maximise its viability in modern networks. These goals should be approached separately as they have different requirements, forming two halves of the design. The scope

of this support is dependent on the network function itself and, if software, the complexity of its operations, with blackboxes being the focus of this work over greyboxes.

- **Support for unmodified blackbox middleboxes**
  Grey and blackboxes obscure their internal workings and structure from the end user in a bid to protect their design, as they are first and foremost products. For hardware, this is done via a lack of documentation on design and operations beyond configuration and operation, as well as secured casing in more extreme circumstances. For software, precompiled code, DRM and code obfuscation are techniques designed to protect their internal operations from being seen. This makes the task of copying, extracting or replicating state from these middleboxes difficult, but it is vital that no presumption is made that this is feasible, both on an individual or universal scale. With the clear understanding that modification is infeasible and their replacement too costly and disruptive, all efforts to re-establish state cannot modify the target middlebox.

- **Minimise incurred overhead**
  Latency and overhead incurred on performance and traffic during failure-free operations should be kept to an absolute minimum wherever possible. Reducing incurred latency is a key goal of research conducted in this area, as discussed in Chapter 2.5, with a reported limitation of 1ms per-packet latency incurred as the maximum tolerable limit. As a remediation step over recovery, some latency is inevitable. With this in mind, any approach pursued must strive to avoid anything that would incur delay on traffic on the main branch as much as possible - delays incurred on copied traffic is not important.

- **Guarantee sufficient correctness of recovery**
  The loss of state is problematic, but incorrect state recovery can cause greater disruption on top of the pre-existing failure. This is dependent on the type of state, which is further dictated by the target. Blackbox hardware and greybox software are not intrinsically stateful, but many stateful devices like load balancers, NAT gateaways, IDS and so on are still realised in this format for production and carrier-grade use. For blackbox hardware, this state will take the form of internal monitoring and forwarding flow entries, while greybox software may also include the use of hardware clocks, thread accesses and other internal decisions. The extent that our best-effort approach needs to recreate these decisions is informed by our understanding of the limitations of hardware solutions and the work of Reinforce (Kulkarni et al., 2020) showcasing how the majority of state is established only at the start of most traffic flows for our targets. The design must provide sufficient correctness to be a viable solution

to state-aware redundancy.

- **Technology-agnostic**
  The design must accommodate a wide variety of potential target configurations and minimise the extent to which it must be modified to accommodate their variations. This general applicability must take into account whether it is a grey or blackbox, the kind of functionality it provides, and the individual products' variations therein. Some fitting may be necessary to accommodate, but this should be kept to a minimum and affect the design exclusively.

The design requirements above are derived from observations made in this area of research as a whole and inform the direction that this thesis takes to successfully establish a resilience system able to provide state-aware redundancy to both grey and black middleboxes.

3.3 Design considerations

This section discusses the technologies and limitations that must be tackled to meet the requirements discussed above. It compares the existing approaches and their solutions, broken down into each consideration.

3.3.1 Extracting or recreating state

The means by which state is transferred to the redundant unit is one of the most important considerations in the design of this framework. Firstly, the approach pursued will differ for the two sides of this thesis: software and hardware. This is due to differing approaches to how these are deployed in real-world systems. Software VNFs are often deployed in parallel as clusters, and the approach to distributing state will differ to that of a more singular approach that hardware will employ. None of the methods for extracting relevant variables discussed in prior work will be utilised due to concerns over the presumption of openness impeding their real-world viability. The modification of software entities that we create can be considered an acceptable alternative in the circumstance of whitebox VNF redundancies to blackbox hardware or greybox software. Possible approaches to be considered include:

- **Encapsulation** Software encapsulation with the use of VMs or containers is a theoretically non-interfering approach by means of no direct modification to the entities in question, but necessitates significant reconfiguration of the hosting platform approach. This is employed by Remus (Cully et al., 2008) and would allow for a significant level of control over the state captured for checkpointing,

but incurs a significant performance overhead from both the encapsulation and checkpointing.

- **Traffic cloning** Directly cloning traffic is a simple approach to replicating state and is used for a number of purposes such as security and monitoring. Older approaches such as port mirroring (cisco, 2023), or "Switched Port Analyser (SPAN)", are used for analysing duplicate packets of the traffic but suffer from performance issues, packet loss and bottlenecks. For the purposes of populating state, this would be a costly approach in terms of bandwidth, with a logarithmic rise from the initial load before linearly scaling with each new copy necessary

- **Traffic filtering** Similar to cloning, filtering traffic to reduce the overall volume of packets necessary to establishing state is far more cost-effective and efficient. Based on our understanding of ASIC limitations as well as the 'lazy correctness' approach pioneered by Reinforce (Kulkarni et al., 2020) this approach offers a range of benefits including ease of deployment, efficiency and sufficiently correct results.

- **Log interpretation** Output logs generated by middleboxes for monitoring allow one to interpret the state from the decisions that it has made when observation of the internal state is not possible, such as with blackboxes. These decisions can then be interpreted and recreated in software redundancies. This approach requires some measure of fitting to the blackbox to function, including both the log interpretation technique and the target middlebox. However, it offers a viable means of acquiring state through a common output seen with middlebox usage without interference of live traffic or incurring delay on live traffic.

- **Direct state extraction** Whitebox deployments are becoming increasingly common as both industry and organisations, such as the ONF, continue to invest in and promote the use of SDN, respectively. By their nature as open-source or at least accessible middleboxes, their internal values can be observed or accessed, enabling them to be extracted and replicated to another middlebox. The means by which this can be achieved varies between software, from standardised platforms such as OpenFlow and P4runtime that utilise match-action table blocks accessible via API, to software popular in production like Traefik and Seesaw requiring inserted drivers. Examples of this technique in research can be seen in Split/Merge (Rajagopalan, Williams, Jamjoom, and Warfield, 2013), a hypervisor for virtual middleboxes that defines state in its middleboxes as policy and flow state, the latter of which is maintained by all replicas of that middlebox and retained in an external (to each VM) datastore. Any state generated is enforced across all replicas to maintain a consistent view of this

state, with the source of this state selected from each VM's output via a modulo rotation. Regardless of the approach, direct interpretation or extraction creates a highly efficient approach to distributing state from the very source. This is not the focus of our work but may become more relevant in the future as NFV continues to grow.

3.3.2 Failover approach

The means by which stateful failover occurs is important to its overall performance and design. Several approaches have been employed by prior work, divided into two groups: "replaying" and "non-replaying." Some of these approaches have been presented in the work discussed so far in the background. Given the criteria specified by the design requirements, only a subset of these are suitable for our purposes:

- **Packet buffering** A form of checkpointing state, as shown in Pico (Rajagopalan, Williams, and Jamjoom, 2013), buffers captured packets from the primary stream of traffic to be replayed through the redundancy at the point of detected failure. This is the least costly approach in terms of system resources, as it maintains the backup on standby to be utilised only at the point of failure. However, this incurs a slight delay from its startup and the replayed traffic, demonstrated with Pico at approximately 5ms, alongside the potential risk of packet loss from link saturation.

- **Live replay** Maintaining a live replica is costly on resources, depending on the number of virtual or physical instances used as redundancy, but it removes any delay from their startup state instantiation. This approach is more technically simplistic to implement and ensures an up-to-date level of state.

3.3.3 Deployment architecture

The architecture of the framework is an important consideration, as it affects the design, functionality and flexibility of its use. This would define whether the components of the framework, such as the traffic control, redirection, failover, and state extraction mechanisms, are centralised within a single deployed application, distributed between components with a centralised logical control or divided into independent microservices. The nature of the deployment can also affect the scalability of its resources, which is another factor to consider.

- **Monolithic architecture** The architecture may be monolithic, with all of its components within a centralised deployment, or it may sit external to the target middlebox. Every component would then be created rather than using

third party tools or mechanisms, as well as hosted and deployed as a whole system. Prior work in this domain, such as Pico (Rajagopalan, Williams, and Jamjoom, 2013), has generally favoured centralised solutions that encapsulate or replace the existing infrastructure, typically to facilitate the greater level of control required.

- **Decentralised architecture** A decentralised architecture is similar to the monolithic, but rather than being deployed as a single framework or system, its components operate independently of one another with a centralised logical control or management. This approach would adhere better to a networking environment for scalability purposes, as well as offer some measure of internal redundancy in the event of component failure.

- **Microservices** A microservice architecture would further divide the distributed architecture into smaller, loosely coupled services. Microservices are beneficial for flexibility, deployability and technology-agnostic design and are intended to be used with third-party applications that can be replaced without significant modification to the rest of the architecture. For this work, it would involve dividing the liveness, failover, traffic redirection, and state distribution into separate independent mechanisms and utilising any pre-existing tools to facilitate its work.

3.3.4 Technology-agnostic architecture

To ensure the viability and longevity of this proposed system, its design must be sufficiently generic so that it may be created in multiple technologies, with no aspects of its design reliant on a technology-specific mechanism or tool. This ensures the widest possible compatibility and appeal for its use. This can be achieved in one of two ways:

- **SDN technology** SDN will continue to grow and expand in use as commodity processors gain in clock speeds and bridge the performance gap. Designing the system to take advantage of a separated control plane and current shared SDN practices can allow for both compatibility with all current SDN technologies as well as future expansion. SDN allows for far greater flexibility in its mechanisms, replicating both network primitives and providing the means to explore beyond.

- **Hardware primitives** Existing network layers and protocols are heavily ossified and unlikely to change. Designing the platform to function with existing network primitives realised in hardware would ensure its compatibility with almost all current networks, but limit the scope of what it could achieve to fairly basic mechanisms.

3.3.5 Retaining and distributing state

The storage and distribution of state is a similar consideration to that of the deployment architecture. Specifically for non-packet-based state distribution, state that is created must be distributed to all redundant middleboxes. As a question of consistency of state view across multiple redundancies, this problem may be considered purely in the CAP theorem example, discussed earlier in Section 3.1.4. There are a number of ways this can be achieved:

- **Centralised datastore** A centralised datastore, separate from the primary and redundant middleboxes, can receive and distribute state across all middleboxes. This approach has a number of advantages, including ensuring a singular view of the state is maintained as well as enforcing a separation between the primary and redundant middleboxes. This latter point is beneficial for maintaining the goal of limiting awareness of the existence of the framework by the primary middleboxes. An example of this approach would involve the primary middlebox state being extracted or recreated and distributed to all redundant middleboxes through a publish/subscribe channel system.
- **Direct distribution** Rather than a single centralised datastore that retains all state, state may be distributed directly to the redundant middleboxes from the primary. This follows a number of existing practices, such as zoning and Paxos, with an enforced leader already elected as the primary middlebox. This approach removes the potential weakness of a datastore that could serve as a single point of failure and replaces it with a direct message channel, but it might impair the scalability and separation between the primary and redundant middleboxes.
- **Peer-to-peer** The state distribution may follow the practices of Paxos and leader election protocols more directly than direct distribution by allowing for a new leader to be elected after failover has occurred. The prior approach treats failover as a one-off, with the service to be restored to the primary middlebox. This approach would differ by allowing the establishment of a new consensus leader, introducing some greater resilience alongside further complexity.

3.4 Design overview

With these design considerations taken into account, this thesis presents REMEDIATE (REsilient MiddlEbox Defence Infrastructure ArchiTEcture) a generic state recovery system for middleboxes that meets all of the proposed design requirements of Section 3.2. It is a non-modifying architecture for middleboxes realised using

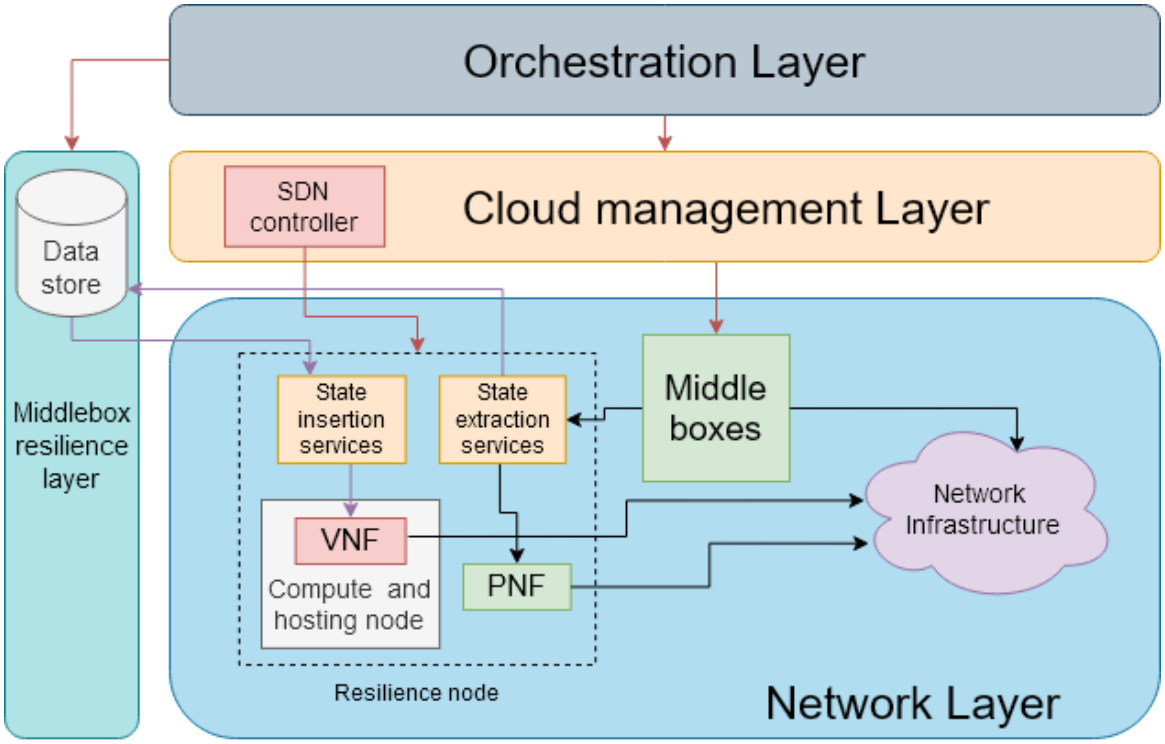

Figure 3.4: High-level architecture overview. The middlebox resilience layer serves as the centralised logic of the framework, utilising the existing network orchestration and cloud management to operate the state extraction mechanisms and VNFs within the network. Red arrows indicate control over an element, black indicates the flow of traffic, purple indicates state.

virtualisation and VNF technologies, able to target both greybox software and blackbox hardware in a variety of configurations/layouts with minimal overhead. Figure 3.4 presents a high-level overview of the system as a whole and its components. This consists of the topology of a network and its interaction with the middlebox resilience layer, incorporated as an external system. This external system interfaces with existing orchestration and cloud management layers, encapsulating existing middlebox implementations with state-preserving mechanisms to establish persistent state across replicas regardless of the source or technology of their implementation, able to derive state from white, grey or blackbox middleboxes. These mechanisms, discussed in more detail in their respective sections in the implementation, form the two major techniques of the architecture and publications, Katoptron and Middlebox Minions. This section will break down the various components and their interactions, with more specific details to be presented in the implementation.

3.4.1 Levels of middlebox access

The synchronisation of state across devices is a complicated task in its own right, with the work of this thesis focusing upon blackboxes as a target, complicating this further. Several approaches are discussed in Section 3.3.1, presenting a selection of methods used in previous research. To provide the widest possible coverage of potential targets, the work of this thesis and Remediate must target the three levels of interaction, each with their own approach.

- **Full state visibility - complete programmability**
  The first scenario consists of direct awareness of the internal state, its operations and metrics and may be feasible with certain open-source/open VNFs utilised in modern networks. Not all VNFs popularised in businesses are pre-compiled paid-for software, but instead may be utilising popular open-source tools and technology. One example includes the OpenConfig initiative (initiative, 2024), a consortium of industry figures to establish a set of standardised APIs and open tools for managing network devices. Direct access allows for modification and insertion of drivers capable of retrieving table entries and similar variables. An example of this would be the capture and dissemination of OpenFlow-specific table entry instantiation commands being received from the control plane/SDN controller via the channel. The state extracted would be dependent on the tool itself and whether it can be easily replicated and inserted into other VNFs without interpretation. Tools such as P4-16 stateful elements would necessitate a different approach, with their contents read and replicated individually to their respective stateful element in the replica.

- **Partial state visibility - moderate programmability**
  The second scenario consists of blackbox hardware or greybox software with observable output, such as logging or diagnostics that provide information on the decision-making and actions taken by the blackbox. This information can be interpreted through an external driver separate from the device and used to establish similar but not identical state using an operationally equivalent VNF backend. This approach requires some measure of fitting to the circumstance, both in the log interpretation and the state acquisition by the redundant replica. An example of this would be interpreting the logging output of a blackbox firewall using an external driver and serialising this state into OpenFlow table instantiation commands akin to the new rules it has created, recreating the functionality as a separate mechanism. Another example would utilise the external read and write handlers of a Click (Martins et al., 2014) element in a pipeline or direct modification of the element's state through exploitation of the hotconfig mechanism that allows for state to be transferred between routers

during liveness. The equivalence of functionality between the primary PNF and the redundant VNF is an important factor in its selection; they might both be firewalls, but the redundant VNF must allow for the means to replicate the decisions of the primary middlebox both before and after failover. This may require the creation of an appropriate VNF in the absence of a pre-existing tool. Despite this fitting requirement, as a scenario, it is far more likely than the first scenario due to external logging mechanisms being quite common for this form of network device for monitoring and diagnostic reasons.

- **No state visibility - zero programmability**
  The third and final scenario to be targeted consists primarily of blackbox middleboxes that offer no such external information to the operations and decisions it is making. To replicate state across instances, the limited potential for non-determinism in hardware (as detailed in section 3.1.2) must be exploited using traffic cloning and replication to maintain a live replica. On a technical level, this approach requires far less fitting than the prior scenarios. However, traffic cloning is an expensive procedure and infeasible in most network setups (doubling the volume of traffic just to provide more accurate failover results). To minimise the impact of this approach, traffic filtering and a reduction in the overall volume of information needed is a key factor in this approach, as pursued by this thesis' contributions. The majority of the state generated for flow tables is formed from flow initiation within the first few packets of each flow. This allows for the use of a fraction of each flow to pre-populate the state within a redundant middlebox. It is here that the complexity of the design occurs, as the type of traffic and the volume necessary to ensure the continuation of state varies from middlebox to middlebox, although all within the range of a handful of packets. This is explored in more detail in the evaluation.

3.4.2 Resilience nodes

The resilience node is a logical abstraction of the encompassing or pairing of existing NF implementations with either virtual or physical redundancies, aided by state-preserving mechanisms as proposed above. The state of the primary NF is preserved through a variety of mechanisms that target specific kinds of middlebox, to be copied to these replicas and establishing a multi-direction stateful failover system without replacing the existing/underlying infrastructure and hardware. These resilience nodes form one of the major contributions of this body of work. At a high level, the resilience node is the unified resilience approach that targets the three levels of accessibility of middlebox implementations, able to facilitate both software and hardware NF implementations and enable state persistence in both multi-directional and multi-targeted ways. For example, the role of the primary middlebox may be fulfilled by

existing hardware, with its resilience node extracting state through a traffic filter mechanism to a scalable cluster of VNFs serving as remediation redundancy in the absence of redundant hardware units to provide failover, leveraging the use of the compute resources of the network. The resilience node is comprised of at most 3 functionalities: the state extraction service, the state insertion service and the redundancy. This section will break down these high-level components, with further details expanded upon in the implementation and evaluation.

3.4.2.1 State extraction and insertion services

State preservation is the means by which the objective of this thesis (improving the resilience of middleboxes and networks using virtualisation technology) is accomplished. To target the widest possible range of middlebox implementations, the three kinds of middlebox must be targeted for these preservation mechanisms: whiteboxes, greyboxes and blackboxes as defined in Section 2.1.4. To accomplish this, a multitude of mechanisms are needed, as discussed in Section 3.4.1, that operate in tandem with both physical and virtualised replicas to the primary middleboxes. These mechanisms form the core of this thesis: MiMi and Katoptron, the former targeting white and grey, and the latter black middleboxes. This resilience approach utilises multiple possible state extraction techniques to offer a significant level of flexibility to the network engineer for the potential number of network layouts and tools utilised. The individual components of the resilience node are depicted in Figure 3.4, with compute resources used to provide VNF hosting and managed by the cloud management layer. The VNFs, used for both filtering and providing virtualised redundancies, are managed by the network-wide SDN controller, with PNFs separately operated by their respective network engineering teams. The state extraction service depicted in the diagram is a combination of several extraction mechanisms. State is pulled from these extractors and held within the datastore to be distributed among the redundant middleboxes via the insertion services.

These extractors are made up of drivers, specifically targeting greyboxes, sit external to the middlebox and reconstruct state from the datastore. The resilience node enforces separation between its targets and the redundancies so as to minimise the level of "awareness" that the system must have to the resilience node. These mechanisms are built and operated externally from the platform, with the enforced separation being part of the goal of minimising interaction, interference, modification or replacement of the underlying infrastructure. This is achieved through three systems: the external middlebox resilience layer that operates and manages both VNFs and PNF redundancies but not the original middleboxes; the external state repository to minimise direct interaction from the primary to the redundancy/redundancies and finally, the use of external state establishing mechanisms in the resilience node, such

as log interpretation or traffic filtering. Outside of whitebox scenarios, the state extraction mechanisms of the resilience node are all external and operate either up or downstream from the target middleboxes, with best-case scenarios of physical/virtual network function(s) encapsulated within a resilience node with no awareness of its presence in their configuration or operations.

3.4.2.2 Facilitating both hardware and software

Remediate is able to accommodate both hardware and software in both directions; more specifically, state can be extracted externally from both physical and virtual middleboxes that require state for their operations, as well as utilise either software or hardware to serve in the redundant role and receive this extracted state. The aim of this is to accommodate the widest possible number of network configurations and maximise the viability of the work proposed in this thesis. This is achieved through the use of external state extractors for each of the white, grey and blackbox implementations. The arrangement/configuration is dependent on the preference of the network operator, although there are technical considerations for the approach chosen. Whiteboxes serving as the primary packet processors are best matched with equivalent software replicas, modified with inserted drivers. More likely scenarios would consist of either closed-source software or hardware with observable decision-making such as logging (greyboxes) or the same with no external view of inner decisions (blackboxes). For the former, the external interpreter and listener re-establish state for a targeted VNF, matching performance and operations as accurately as possible. For the latter, the performance difference and limitations on non-determinism encourage the use of hardware redundancies for primary hardware with state provided by traffic filter mechanisms. The scalability of this approach is limited, however, although there is potential to employ VNF redundancies in clusters with state distributed amongst the replicas through the external datastore. This body of work does not explore this technique however. There are limits to the degree to which parallelism can match the performance gap between software and hardware (C. Wang, Spatscheck, Gopalakrishnan, and Applegate, 2016). Nevertheless, the flexibility of Remediate ensures the possibility to provide hardware/software to hardware/software is available to the end user for whatever their requirements.

3.4.3 External State Repository

Bandwidth in networks is a scalable commodity and the basis of the operators' business operations. As a business decision, they limit the bandwidth they offer for operations that are not conducive to this goal, as any bandwidth used for purposes such as resilience is bandwidth not being used to service customers. As networks grow in size, so too will the size of state that must be retained. In principle, this

bandwidth capacity can continue to be scaled as the volume of traffic grows as well as the state, but in practice, there are physical and economic limits that must be considered. Therefore, minimising the increase in bandwidth use and latency incurred by state updates from Remediate is a high-level requirement. To allow for scalability and reduce the volume of potential traffic between replicas, the extracted state is sent to and distributed by an external repository.

A datastore is a common network tool often used for a variety of purposes, including caching and message brokering. They differ from databases, where they maintain not only data but serve as a global repository for files held only in memory. An external datastore serves as a centralised state repository, minimising communications between these two sides to a single entity like a BGP reflector, further enforcing the separation of the two layers and minimising the complexity of the communications. The state extraction mechanism used by the resilience node propagates the state to the state repository. From the repository, message brokering distributes state to all available listeners that form the second half of the state extraction mechanisms. Through this middleground, the number of potential VNFs or PNFs utilised as redundancies can be altered at any time by the user, with no awareness on the part of the original middlebox. State transfer may vary in its scope and direction, with the number of entities on either side of the primary:redundancy balance highly flexible.

3.4.4 SDN management

VNFs are utilised by Remediate in a redundancy capacity, including the use of purpose-built or modified VNF copies of the primary VNF for whitebox scenarios as well as scaling for performance purposes. These VNFs may be defined and configured from the external resilience layer, but deployment and operations are handled by the network operators and their connectivity by the existing network-wide SDN controller already in place. Operations for these redundant VNFs would be controlled by the existing managers using the orchestration and management layers. This is done to minimise the degree of new technology necessary to utilise Remediate and exploit pre-existing assets in a network infrastructure. Whilst SDN is not universally adopted in industry, it is growing at a rapid pace (Feamster et al., 2014), and any deployment utilising VNFs for redundancy purposes is likely to possess some degree of supporting infrastructure, with virtualisation rarely used without some measure of SDN technology for monitoring and control purposes. Examples of real-world systems include JunOS Space (Juniper, 2009) for managing Juniper Network-specific fabric or relying on NFV MANO approaches pursued by more traditional Network Equipment Providers (NEP) such as Cisco and Ericsson that use cloud technologies such as VMware.

3.4.5 Infrastructure

The network infrastructure represents the existing network architecture and its elements. The goal of this thesis is to cater to a wide variety of possible approaches to middlebox usage, with a focus on those that are vulnerable to state loss still implemented within dedicated hardware. The middlebox resilience mechanisms sit externally and should not warrant the replacement of any of the underlying infrastructure, presuming no complications to interfacing with the network orchestration. Even in the absence of network overviews or controllers, Remediate is able to be deployed in an entirely independent format, encapsulating any targeted middleboxes and feeding state to available redundancies through static deployments using protocols such as Cisco's HSRP (Li et al., 1998) or PRP/HSR (IEC, 2016). Examples of this are best represented by the prototypes developed and examined in the evaluation section, with their use cases requiring only a minimum of a switching and state extraction mechanism realised in open-source user-space software.

3.4.6 Cloud management

The cloud management layer represents the network resource and infrastructure management, essentially encompassing all available entities that may only be utilised in the short to medium term rather than permanent fixtures. This includes both the SDN controller, which is orchestrated by the layer above the cloud management, and the resilience node state extraction mechanisms. These mechanisms are realised in software but do not strictly fall under the scope of VNFs that might be instantiated by a typical controller. The middlebox resilience layer provides the patterns and mechanisms for their use by the management layer, where resources can be appropriately scaled in the case of any flexible VNF deployment. Other responsibilities include the switching functionality, which covers both failover and service restoration; these can be achieved in a number of ways, but more commonly through flow tagging or SDN-enabled switches inserted at key sections within the network. The means by which this is achieved is primarily dependent on the network in question and falls outside of the scope of this research, as it is a common operation in overlay networks that would seek to provide resilience against disruption through redundancy.

3.4.7 Orchestration

The network orchestration layer, presented at the top of Figure 3.4, represents the total network control available to the network operators, which itself is performed through the separate MANO platforms via engineers. The external resilience layer works in coordination with the orchestration to operate the redundant systems within the network infrastructure. There are four primary functions performed by

the orchestration, using the existing VNFI (compute and hosting nodes) and SDN controller as the interface: redundant VNF initialisation, teardown, traffic control and redirection. This separated approach is used so that the system may remain flexible to the layout of the network it is externally fitted to, utilising any existing SDN or NFV systems present in the network rather than mandating their presence. Once established, redundant VNFs are deployed using the existing VNFI and hosted on the hardware supporting the redundancy node elements. These should remain active during normal operations and are not intended to be used outside of their almost static redundancy purposes.

Traffic redirection can be done either actively using techniques such as flow tagging or dedicated flows, as discussed prior, or passively using redundant routing algorithms. Beyond flexibility in how the external system can adapt to the pre-existing infrastructure, the other goal of its use is to enable a centralised point of orchestration for this redundancy system; replicas are deployed by the external platform using the network's orchestration layer through its interface. For hardware middleboxes and networks that lack any significant SDN/NFV presence to utilise, the approach to their management differs. Blackboxes already lack outside coordination and require dedicated teams of engineers for their operations, who are required to configure and monitor specific sets of boxes, replacing them where necessary. This is not a problem under Remediate however, as PNFs used for redundancy purposes can exploit this pre-existing management instead, with these devices simply falling under this umbrella as additional managed hardware. The external state resilience mechanisms that service these PNF redundancies fall outside of this umbrella, with traffic control, failover and service restoration performed by the inserted VNF mechanisms. These are operated instead by either the SDN controller or the cloud management itself, depending on the network arrangement. For example, failover and state capture which are both performed by the resilience mechanism of the node would be managed by the SDN controller. If utilising inserted SDN-enabled switches or flow tagging for controlling the direction of traffic, however, this would instead fall under the jurisdiction of the cloud management layer as network infrastructure management over deployment.

3.4.8 Remediate - Middlebox Resilience layer

The logic and core operations of Remediate are represented by the external middlebox resilience layer presented in Figure 3.4. As discussed in Section 3.4.7, it sits externally to the rest of the network as an additional element, providing redundancy using existing control mechanisms. This includes the patterns/templates for targeted VNFs, resilience mechanisms for capturing state and communications to the external state repository. PNF and VNF elements are operated directly via their respective

control mechanisms and engineers as discussed in their specific sections, with external orchestration performed through configuration files or manual control by the network engineers at the resilience layer using the overarching network configuration layer. The state repository, represented as the external datastore incorporated into this resilience layer, operates independently of direct control but its setup is dictated by this layer. Once Remediate is incorporated into a network, the number of additional elements required to enable state-aware resilience to whatever middlebox configuration is in place should be minor, minimising the overall footprint of the work proposed by this thesis.

3.5 Summary

This chapter has detailed the high-level design to achieve the aforementioned aims of this thesis and to address the challenges highlighted in Section 3.2. It has been built on the awareness of the existing state of middlebox placement and design, as well as the steady rise of SDN and NFV in enterprise deployments. The decisions have focused on achieving the design challenges extracted by analysing related literature and work, such as accuracy, scalability, efficiency and an effective and easily deployed design. It is key that the proposed system is both non-modifying and cross-compatible with most use cases, including blackbox hardware and greybox software, as well as offering mechanisms for most possible network configurations. The design and its two realisations MiMi and Katoptron, detailed in the next chapter, are intended to be agnostic of specific technologies so as to allow for the requirements of its users and the changing landscape of technology as virtualised networking continues to grow. In the next section, the two branches of this design that were built and evaluated will be detailed, including the specific mechanics made to achieve these aims and the linkage between systems.

Chapter 4

Implementation

In the previous chapter, a high-level design for Remediate was presented, displaying how the proposed system would be integrated into an existing network and operate through pre-existing orchestration and managers. From this design, two major branches of implementation have been developed as the primary research contribution of this thesis, depicted in Figure 4.2. This chapter details the implementation of both the software-targeted state extraction system "Middlebox Minions (MiMi)" and the hardware-targeted state recreation filtering system "Katoptron". It will be split into three sections; shared elements between the implementations, the high-level designs of MiMi and Katoptron, and finally the prototypes built for the evaluation. The shared elements, split across the design and prototype, consist of any mechanism or component, such as the testbed and Fast Failover (FF) implementations similar for both prototypes. After this, sections 4.2 and 4.3 will detail the design and mechanisms of each system. Finally, the remaining part, consisting of sections 4.4 and 4.5, will detail the tested implementations used for their evaluations in their respective publications.

4.1 Resilience Framework

This section is split into two halves: the shared elements across the two architectural designs, and the shared elements in the prototypes themselves. The two projects form major contributions of the overall design and represent the body of work of this thesis and Remediate. As a result, they share certain aspects of their design to be realised as separate publications which shall be discussed here to avoid repetition.

4.1.1 Point of Failover Architecture

The points of failover (PoF) are logical structures within the network that serve as the monitoring, switchover and failure junctions between the primary packet processing route (the existing middlebox) and the redundancy (the PNF or VNF(s)). This point of failover can be a singular entity such as a programmable network device (*e.g.* OpenFlow on merchant silicon (McKeown et al., 2008b) or a P4 switch (Bosshart et al., 2014) able to support fast-failover path recovery) or set of protocols in the switching fabric depending on the complexity and forwarding setup of the network. They are formed of three functions: liveness monitoring, failover mechanisms and, in the case of Katoptron, service restoration mechanisms. Firstly, liveness protocols operate upstream from the target middlebox, observing for network disruption or middlebox failure. To achieve this functionality, the PoF and network can use an array of different techniques, ranging from voltage checks, L2 updates, ethernet link sensing or Fast ReRoute (FRR) paths with RSVP-TE signaling (Chiesa et al., 2019b). These approaches offer different trade-offs with respect to hardware requirements and failure detection guarantees. More advanced mechanisms, such as heartbeat monitors (IBM, 2024), are dependent on the product as they must be supported or inbuilt but are otherwise infeasible for blackboxes due to the modification requirement if not. Simple link failure monitors require little bandwidth to operate at a local level and tend to be more common than vendor-specific link aggregations like PAgP (Systems, 1998) or LACP (Huawei, 2022).

The second function, failover, is the means by which traffic is redirected from one device to another across the network when needed. During detected failures, the PoF will failover to the redundant path, masking the change from the primary to the secondary middlebox from the traffic in flight and bypassing the failure. Both MiMi and Katoptron are agnostic of the mechanism used to maximise the scope of applicable network configurations for their use. Finally, the point of failover is utilised to facilitate service restoration to the original middlebox when the issue has been resolved. Operating independently of a manager, a single point of failover and the filter are sufficient to reconstruct state on live flows and maintain a hot replica to a target middlebox. These points, much like the filter, should sit as 'bumps on the wire', or unobserved by the traffic itself as it passes through to the blackbox during failure-free operations. Two switchover points are recommended, however, for service restoration purposes, which shall be discussed in more detail in Section 4.5.2.

4.1.2 Points of Failover proof of concept

The two mechanisms both utilise very similar failover, built using OpenFlow's inbuilt group tables, allowing a set of static links to be swapped in the progressive chain in case of link breakage or failure. The "FAST-FAILOVER" group establishes a

list of actions known as 'buckets'. Each bucket contains a list of parameters and actions. For the FF group, these buckets specify a watch port to observe for liveness. Liveness is evaluated through link sensing and observing for loss of Ethernet preamble. If the interface goes down, that bucket and its actions are no longer in use, falling to the next bucket. Using this ordered list, a set of links and their redundancies can be specified, falling over to the next available link in the chain. These in-built mechanisms implement typical real-world systems used for monitoring link liveness and redundant link protocols. For both Katoptron and MiMi, these failover mechanisms are implemented into specific static switches in their testbeds on either side of the target middleboxes with a set of static flows to handle active and redundant links. For the evaluation, MiMi's points of failover are present before and after a simulated middlebox, with link failures triggered by external scripts during experimentation. For Katoptron, the switch is linked to the filter on the path to the primary middlebox and the redundant middlebox directly.

```
ovs-ofctl -OOpenFlow13 add-group cs1 'group_id=1,type=ff,
    bucket=watch_port:1,output:1,bucket=watch_port:2,output:2'
ovs-ofctl -OOpenFlow13 add-flow cs1 'dl_type=0x806,
    nw_dst=10.0.0.20,priority=2,actions=group:1'
ovs-ofctl -OOpenFlow13 add-flow cs1 'dl_dst=00:00:00:00:00:FF,
    priority=1,actions=group:1'
```

Figure 4.1: OF flow table rules for a fast-failover bucket and watch ports

This code snippet, presented in Figure 4.1, is an example of the group table structure. It consists of two buckets, each containing a port to observe for liveness and an output port. This is accompanied by two matching flows to direct traffic to the group table rule. Katoptron builds on the failover mechanism, expanding the point of failover with restoration mechanisms that are further elaborated on in Section 4.5.2.

4.1.3 Management layer

The management layer coordinates the operations of the resilience frameworks and the mechanisms that make up their distributed architectures. The frameworks are distributed architectures formed from a number of separate mechanisms, managed by a centralised layer. The layers for both MiMi and Katoptron are sufficiently similar in their roles and structures that they may be combined here as a shared architecture. Key differences between the two architectures will be highlighted. These shared mechanisms include the failover, VNF controllers, liveness monitoring and VNF instantiation and teardown. The operations of these mechanisms are primarily independent, operating automatically once configured and deployed, such as link

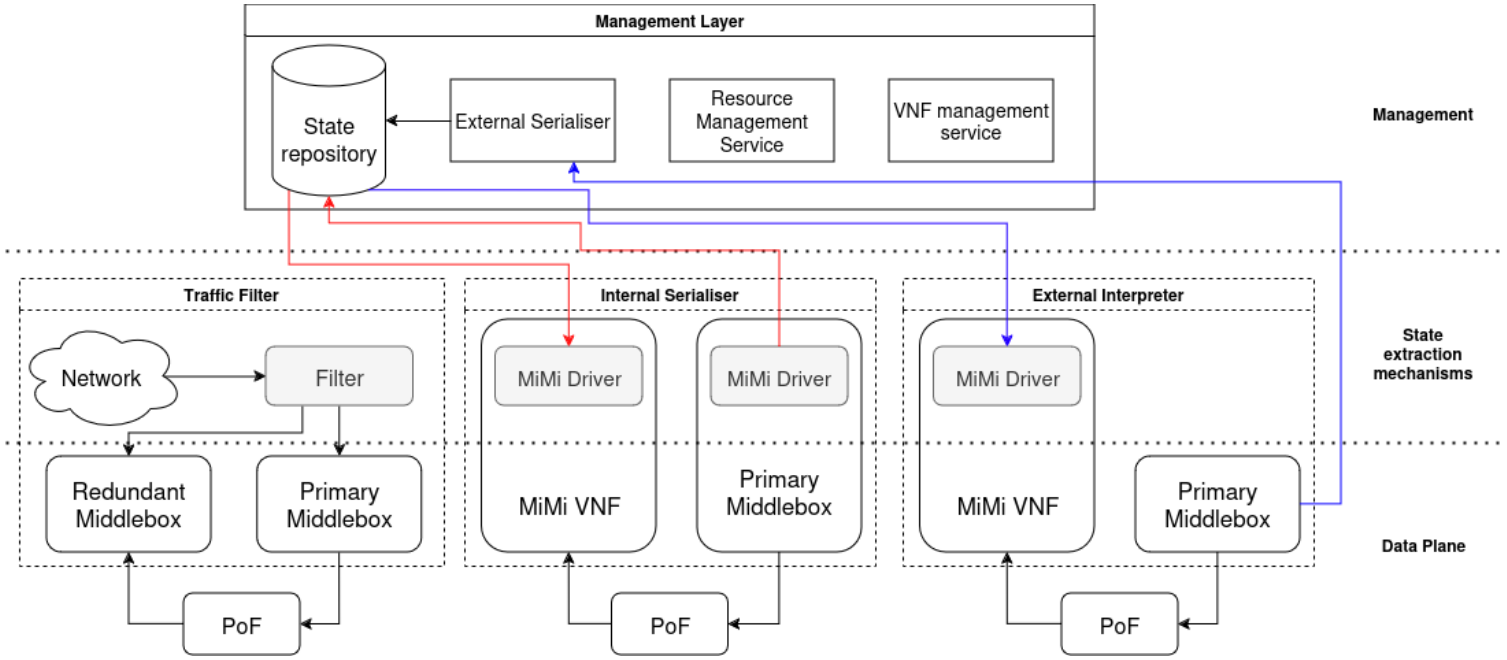

Figure 4.2: The overall system architecture, presenting a logical view of the management layer and its interactions with the state capture mechanisms. On the left, the Katoptron traffic filter approach using blackbox VNFs to pre-populate state tables through targeted filtering. In the centre, an inserted driver directly serialises state to be distributed by the repository. On the right, log output from a greybox is interpreted and converted into a serialised format, to be distributed by the state repository to whitebox VNFs. The blue arrows indicate the flow of state for MiMi using an external interpreter, the red using an internal directly to the repository.

monitoring and failover. However, some level of control is necessary for the purposes of monitoring and operations, with examples including the VNFI and controllers for instantiation and teardown of any replicas utilised for redundancy (in the case of MiMi) or the filter mechanisms (in the case of Katoptron). The individual mechanisms not shared between the two architectures shall be discussed separately. For MiMi, this includes the state repository, publish and subscribe channels, interpreters and serialisers. These are a part of the state repository mechanism, which is expanded upon below in its own section. For Katoptron, this includes the traffic filters and service restoration mechanisms.

- **VNF management** The VNF management is a major component of the management layer and is responsible for the instantiation and teardown of all virtual elements of MiMi and Katoptron. This includes the whitebox redundant VNFs, state serialisers, traffic filters and, in certain circumstances, the PoF

mechanisms. These mechanisms operate independently of a control plane in most circumstances, with the VNF management responsible only for their deployment and ensuring continued operations. For MiMi, the redundant VNFs operate as hot replicas, receiving state during normal failure-free operations, and are live at all times rather than migrated or activated at the point of failure. The memory and bandwidth costs for this are a consideration however, and the extent of scalability should be taken into account. It is worth highlighting that the architecture has a negligible impact on the resources of the infrastructure during idle mode, as the backup VNF functionality will be limited to collecting state updates and signal liveness to the management layer. Furthermore, modern memory ballooning techniques in VMs (Barham et al., 2003a) can also minimise the need for internal memory fragmentation on the virtualised servers. Katoptron is intended to target blackboxes via its filter mechanism to recreate state in its target redundancies, which are best suited as similar physical hardware for ease of deployment within existing networks. However, it is capable of targeting precompiled blackboxes such as software VNFs. Any VNFs and the filters deployed in software fall under the VNF management. The performance difference between the physical blackbox and a VNF serving as redundancy is partially mitigated by its role as a failure remediation tool; this intermediary stage between failure and normal operations provides a diminished service rather than disruption.

- **State Repository** The state repository is a middleground that distributes serialised state to the redundant VNFs. It takes the form of a communications channel or datastore that implements a publish/subscribe channel model, configured to receive traffic from the serialisers or interpreters in a 1:N arrangement. The state repository does not hold state for longer than the current live state view window, which is maintained only until it is received by the subscribers. While the operations of the state repository fall under the scope of the management layer, its actual function operates otherwise independently of the rest of the management layer.

- **Interpreters and Serialisers** The interpreter and serialiser drivers provide MiMi's state recreation and extraction techniques (or grey and whitebox) respectively. For the former, the interpreters sit external to the target middlebox and are configured by the management layer for its communications with the state repository and logging system. The serialisers consist of inserted or modified code to existing VNFs and are configured by the management for communications with the publish/subscribe channels.

- **Liveness monitoring** There are a number of liveness protocols used in

production networks, but Katoptron limits the scope of solutions that are suitable. Heartbeat monitors and keepalive protocols are functional with software such as white and some greyboxes, but are infeasible with blackboxes. This limits the liveness monitoring to link sensing methods such as L2 updates, port sensing and voltage checks. Larger networks outside of this testbed may use other approaches incorporated into MPLS and existing FRR techniques, but both Katoptron and MiMi are agnostic of the approach as long as it provides adequate failure detection and failover.

- **Service restoration** Failover is the action of changing the direction of traffic from the primary to the redundancy and is handled by the PoF upon detection of liveness isuses. Service restoration is the opposite direction, for when the primary has been restored. This can be done in a number of ways: blindly, with no regard for the traffic in flight, or through bleedover techniques to slowly syphon traffic back to the primary. This is dependent on the network itself, with arguments to be made for each. Katoptron proposes a bleeding mechanism using an SDN-enabled switch, expanded upon further in Section 4.5.2.

- **Traffic filters** The traffic filter sits upstream from the target middlebox, filtering state-generating packets from the primary stream to the redundancy. Traffic flows bidirectionally, but the filtering is only applied to incoming streams to a network as that is the primary source of state is derived. Traffic direction from this box is dependent on the network design and layout, either sitting directly on the wire before the middlebox in a more simplistic architecture or relying on flow tagging techniques for forwarding and directing purposes, *e.g.* linked directly to the primary and redundant middlebox, or placed elsewhere in the physical layout with traffic tagged and forwarded across the network in accordance with these labels. This is one of the key motivations behind reducing the overall volume of traffic necessary to forward to the redundancies so as to minimise impact on the system itself.

4.2 White and greybox resilience

Middlebox Minions (MiMi) is the VNF resilience half of the framework that provides methods for white and greybox stateful failover, as well as the scalable state distribution for the overall Remediate framework. This section describes the components and features of its design. Figure 4.2 presents the combined architecture, with MiMi contained within. As a broad summary, MiMi is a resilience framework that reconstructs state and maintains partial packet processing and forwarding correctness during middlebox failures. This is done using pairings of key middleboxes with VNFs,

where a set of modules maintain state synchronisation between the primary middlebox and the minion VNF. It is built on two basic assumptions: the network infrastructure offers compute resources to support VNFs (like 5G RAN, cloud in ORAN, NFV-MANO and Core architectures) and a fast-failover path is in place when failures occur. MPLS (Atlas et al., 2005) and group table entry-type SDN technologies provide built-in support for fast rerouting actions, while recent research efforts have developed efficient fast-reroute hardware designs (Chiesa et al., 2019a). The architecture is divided into three layers: management, drivers and middleboxes/VNFs. Firstly, the management layer orchestrates all the operations of the VNFs, including liveness and failover. It manages all VNFs and their shared resources. In parallel, it uses the northbound API of the network control plane to establish a backup path to the VNF if the middlebox fails. The management layer uses off-the-shelf database software (*e.g.* Redis) to store the forwarding state of each middlebox. The state (extraction/recreation layer) or driver layer offers two approaches to targeting VNF grey and whiteboxes using serialisers or interpreters respectively. Blackbox middleboxes for both software and hardware are the focus of this thesis, but a whitebox (*i.e.*open-source software) approach has been both established and evaluated as an important early consideration in this body of work.

These serialisers or interpreters, referred to from this point as drivers, are built to target the primary software in use. The state acquired or created by the drivers is sent directly to the datastore, an independent storage shown in line with the management layer. This state repository will distribute to the redundant replicas, allowing for both scalability and anonymity of the 1:N connection for the target middlebox. Finally, the VNF redundancies serve as the backup devices for the target box. They are created, controlled and torn down by the management layer, which also handles their resource utilisation, network connectivity and orchestration. It utilises targeted open-source/modifiable VNFs that are fit to purpose for the primary middlebox and the type of driver utilised. The number of replicas utilised per box is equally defined by the user, dependent on the platform. The remainder of this section contains a detailed breakdown of each of the components, their use within the system and their function in the overall architecture.

4.2.1 State drivers

The state drivers are the means by which state is extracted or generated for the redundant VNFs and is key to the state-aware redundancy strategy. These drivers are purpose-built for their target middlebox and require creation or modification to fit this role. For example, the logging output of a greybox is defined by the network operators and the box itself, so any log-interpreting driver created would be built to fit in accordance with the expected format of these specific logs. In this section, the

two approaches and their use are detailed. The drivers developed for MiMi consist of two distinct mechanisms, targeting two of the three kinds of middlebox accessibility: white and grey boxes. Both solutions are built with the awareness that the redundant VNFs or drivers are modified to fit their targets and intended use, offering a more specific solution over Katoptron's generic hardware-based solution.

4.2.1.1 Log interpretation

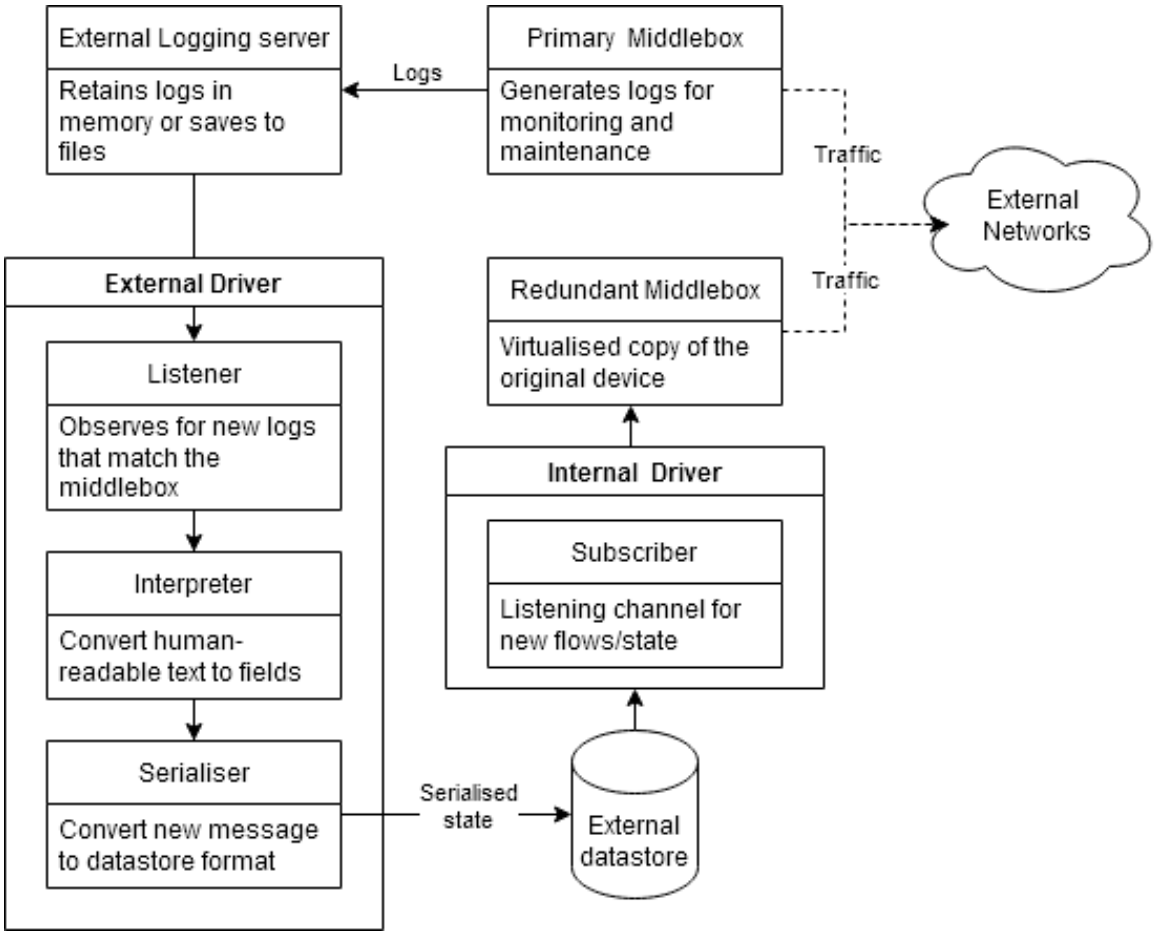

Figure 4.3: High-level diagram of a log interpreter interacting with a primary and redundant middlebox using a datastore distribution and internal subscriber. The flow of activity is represented with these arrows, forming an almost complete circuit from the primary to the redundant middlebox for transferring state.

This section forms the non-modifying approach to recreating state for greybox software, using SDN as the resilience strategy. While access to the internal operations

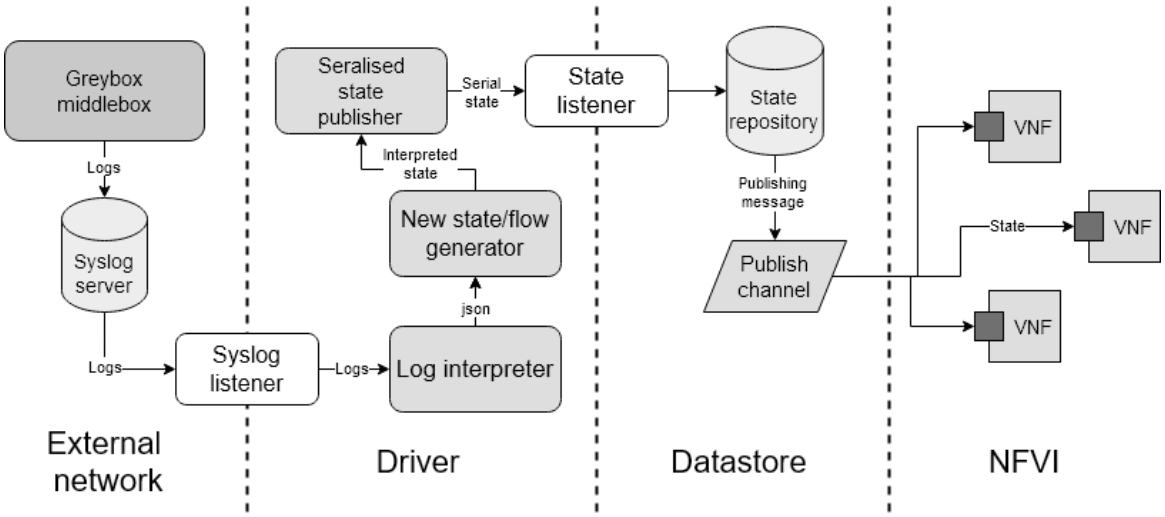

Figure 4.4: Another view of the distribution of state from the primary middlebox to the redundant VNFs. The arrows indicate the flow of actions through the components that make up the distribution of state.

of pre-compiled software may not be possible, greybox software is not as isolated from the network compared to physical middleboxes. They must still be configured and managed by the network, offering a significant level of control via its API, with one aspect of this consisting of its logging mechanisms. Logging is an important and necessary part of the operations of system-level tools such as this, used for monitoring and error correction. The contents of this logging is defined by its configuration, such as printing the headers of all incoming packets, but in typical cases, it will report actions taken. Using this information, interpretation of this logging output produces an approximation of the internal state decisions made by the software greybox. Access to this logging information can be approached either via its API or by external mechanisms such as rsyslog (Gerhards, 2009), a common approach to network logging for potentially container-based or remote elements.

This interpreter must be built for purpose, both to interpret contents and convert them into an appropriate "state". This state in our work takes the form of lookup and forwarding table entries, constructed by the driver and distributed to all VNF replicas via the datastore layer. Figures 4.3 and 4.4 provide visual demonstrations of these log interpreter systems, with the former presenting a logical view of the actions taken and the latter showing how it might look within an architecture divided into its individual components. Finally, Figure 4.5 provides a sequence diagram of the steps taken for log interpretation state distribution. This mechanism is fairly simplistic and sufficiently high-level that most current SDN technologies, including OpenFlow, P4 and similar flow-based approaches, are able to implement it. To give an example, for

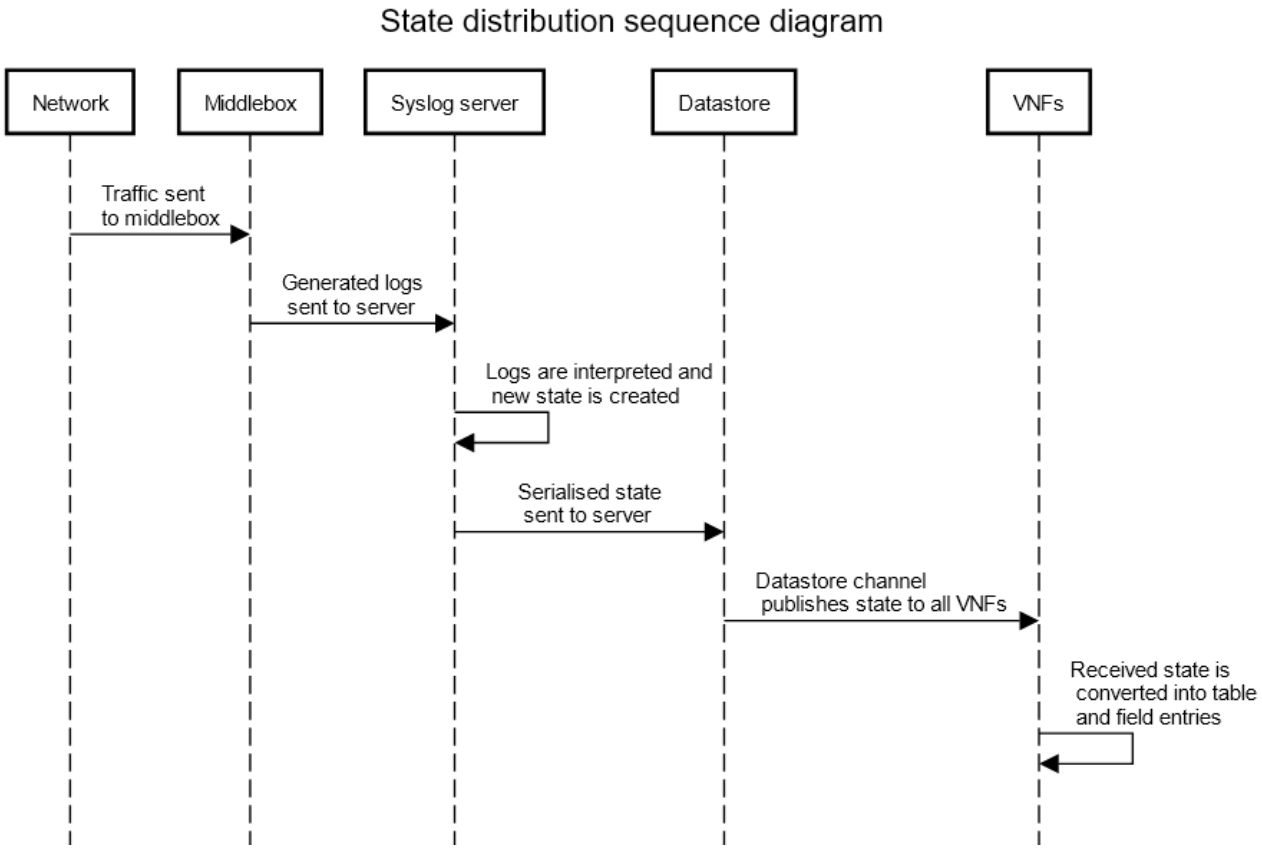

Figure 4.5: Sequence diagram of the distribution of state for a log interpreter

our implementation and evaluation, we focus on OpenFlow and NetFilter (Org, 2004) state driver variants.

Packets are processed by the netfilter greybox, with logs generated detailing the traffic received and the chosen backend server. The driver will receive log entries and use string splitting and tokenising to divide its contents into fields. A new flow table entry for the redundant VNFs (consisting of OF-based load balancers) can then be constructed in much the same fashion, inserting the anticipated fields from this log entry into a generically applicable entry. This is then sent to the datastore where it shall be published, received and written into their respective flow or forwarding tables. This ensures that in the event that traffic is redirected to that particular redundant unit, it will continue to be served to the target destination regardless of other lost metrics and prevent delay or rejection.

4.2.1.2 Direct extraction

This section forms the modifying approach to recreate state for whitebox software, using SDN as both the primary and resilience strategy. In certain circumstances, SDN software may be used in place of more commercial solutions, *i.e.* open-source software is the most practical and viable approach for that particular use case. While log interpretation approaches still apply in these circumstances, direct interaction is more efficient. The approach to this direct access varies with the technology chosen. Many SDN technologies share a similar interaction structure, consisting of defined elements within a pipeline model and lookup tables corresponding to certain elements, externally accessible and modifiable via open APIs. Both Click (Kohler et al., 2000) and P4 (Bosshart et al., 2014) follow this design format, as it is based on how physical hardware is typically built in networking. Regardless of the technology, the goal of direct extraction is to pull the contents of these tables from the primary middlebox and insert them directly into the redundant replicas. An example using OpenFlow can utilise the separation of the control and dataplane to both observe and replicate flow instantiation instructions for its flow tables, which can then be serialised and distributed. Redundant VNFs then receive these instantiation instructions via inserted listeners that communicate with the datastore, which then execute these instructions, creating duplicate flows to those of the primary VNF. This example is only one of a variety of approaches that can be pursued with open and modifiable software for enabling state replication.

4.2.2 State repository

The state repository is a vital component for the separation of the existing infrastructure from the redundant VNFs and serves as the means by which scalable state distribution is possible. The state repository implementation is an off-the-shelf datastore, held in memory, that retains serialised state provided by the state drivers. This datastore can be realised using a multitude of common technologies, including Redis, Kafka and MongoDB. The datastore is not a database but instead a FIFO message queue or event store; serialised state is received as messages to be distributed and held within memory till they are pulled from the queue by all subscribers. To minimise interactions between the primary greybox and redundant VNFs, communication between the two is masked through an intermediary, the datastore. This is done for three reasons:

- To prevent awareness of the redundant VNFs by the primary middlebox and limiting any need to modify or interfere in its operations
- To allow for a scalable and centralised distribution point for the redundant replicas

- To serve as an adaptation layer, simplifying fitting for the targeted VNFs to receive state

To expand on these stated goals, the use of a datastore removes the need for direct 1:1 communication between the driver interpreting state from the primary middlebox to the redundant VNF. This is especially prudent if multiple replicas are being utilised, greatly simplifying the configuration. State can be serialised and received by a remote datastore, wherein it can then be distributed to the 1:N replicas in the user's chosen approach, such as a publication/subscription model. MiMi, in its implementation and evaluation, utilises a pub/sub model, shifting the burden of modification from the target middlebox to the redundant VNFs.

4.2.3 VNF Infrastructure

The final component is the VNFI and the VNF replicas, modified with inserted drivers to receive state. MiMi is agnostic of the NFV/SDN technology utilised, as long as it supports either modification or interaction with an inserted driver. There are two approaches available for populating the redundant VNFs with state. With the serialised state retained in the repository/datastore, it must be distributed to all VNF replicas. This is done via the publish/subscribe model, where drivers inserted or operating externally from the target VNFs subscribe to this channel and receive the serialised state. For internal or inserted drivers on OF-based targets, this received state is directly translated into new flow table entries. For external drivers, it is converted into new flow instructions and inserted into the control plane - data plane communication channel, to be received by all replicas and become functional state. The extent of this modification is dependent on the technology in which it is realised; with OpenFlow as evaluated in this thesis' implementation of MiMi, this approach is both scalable and realisable in existing mechanisms, necessitating very little modification. This consisted of the insertion of a separated polling thread into the redundant load balancer's control application, running a function that, when messages are received from the pub/sub channel, a new flow-mod is created from its separate fields.

4.3 Blackbox resilience

Katoptron (κάτοπτρον, or "mirror") is a platform-agnostic failover system that focuses on the PNF and non-modifying state collection half of this architecture. This section describes the components and features of its design. As a broad summary, Katoptron is a high-availability service that propagates state between unmodified hardware middleboxes and backup/redundant PNF or VNF appliances. The service aims to

achieve two key functionalities: to maintain a hot replica of the state of the primary middlebox with no output from the blackbox itself, and to facilitate failover and service restoration without replacing any pre-existing infrastructure. The rest of its goals are shared with MiMi and the overarching project itself, which facilitates failover for network functions. The platform utilises targeted packet mirroring to allow network devices to construct equivalent state and thus facilitate an easy transition between hardware and software. This is achieved via traffic cloning and packet filtering to replicate specific packets necessary for establishing state. Our approach exploits the fact that blackbox hardware consists of specialised ASICs with minimal internal memory, such as a heap or stack, which dissuades the use of circuit space on pseudo-random generation or system clocks, and instead derives state from received input. This greatly minimises potential non-determinism, allowing Katoptron to exploit a "lazy correctness" approach highlighted by Reinforce (Kulkarni et al., 2020) with regard to most state being created at flow start. Throughout this section, filtering specifically refers to the process of acquiring a subset of the original middleboxes traffic using approaches including but not limited to the example given. This best-effort approach differs from past work, focusing on enforcing correctness and introducing novel approaches to improving resilience in areas of networking where its enforcement is unnecessary. Overall, this allows for a significant reduction in the complexity of state recreation, exploiting the existing concepts of traffic cloning and hot replicas and shifting the focus to minimising the cost of such techniques on network bandwidth.

The architecture, presented in Figure 4.2, is represented on the left-hand portion of the diagram. The packet filter replicates a subset of the traffic necessary for the establishment of state, forwarding it to the redundant middlebox. It sits in line with the target middlebox, serving as the only interruption to failure-free operations. Writing operations are more costly than reads, and replicating all incoming traffic would incur delay on normal traffic. To minimise this as much as possible, packet classifications to identify the beginnings of flows are the primary operations, with cloning minimised to only the necessary packets. This filter is platform-agnostic and highly adaptable to the user's requirements, allowing it to be easily modified to whatever traffic is required to recreate state for that specific network function. The management layer handles liveness protocols much like MiMi, traffic redirection, failover and service restoration. The degree of its complexity is dependent on the redundancy utilised. When targeting only a single PNF serving as the hot replica, the replicated traffic needs only to be directed to its target. When servicing multiple replicas, load balancing is required to manage the scaling redundancies. This can also be expanded to operate and manage the VNF and VNFI in a similar fashion to MiMi. Finally, the points of failover serve as key junctions to facilitate redirecting traffic when directed to by the manager or autonomous protocols, as well as cloned traffic for state

population and service restoration. By focusing on the blackbox-targeted side of this architecture, the mechanics of the state replication differ significantly in approach from those demonstrated in MiMi, addressing a key gap in existing literature. The remainder of this section will go into detailed presentations of the mechanics of these layers and how they achieve their aims.

4.3.1 Packet Filter

The packet filter is a platform-agnostic state extractor, replicating key packets in the flow of traffic and redirecting them to the redundancy to pre-populate their lookup tables. In order to maintain a failover path with up-to-date forwarding state, Katoptron uses a packet filter to duplicate and forward specific packets key to establishing state, such as the initial packets from each flow, via the failover path to prime the redundant NF (either another blackbox or VNF). With awareness that the limited programmability of ASIC platforms keeps the overall complexity of the packet processing logic simple, it can be easily replicated using off-the-shelf VNF appliances to match the functionality. We present a set of middlebox scenarios with this in mind, focusing on those still executed in hardware and able to be replicated in software such as load balancing. The packet filter can be constructed from a wide array of SDN technologies, including but not limited to Click, P4, eBPF and other similar programmable packet processing pipelines.

The use of filters is two-fold: with the assumptions stated earlier, traffic can be recreated from a subset of packets from each flow, removing the need to replicate all traffic in service. Secondly, unlike VNFs, which are often hosted on the same or linked nodes, hardware middleboxes may be geographically separated and directed around the network. Cloning all traffic is a simple approach to high availability, but costly due to the bandwidth consumption incurred. By reducing the potential volume by 95-98% (packets replicated to overall volume of same flows), this radically reduces the impact and overall cost of its utilisation. Traffic filters are placed in line with the ingress of the target box or chain. These filters are platform-agnostic and adaptable to the expected traffic requirements of the target. The majority of stateful traffic on network devices concerns TCP flows, with most middlebox state focused on tracking newly established connections. Typically in blackboxes, initial packets of flows provide the five-tuple elements for the key to its hash table entry, with all subsequent packets hashed and evaluated against existing entries. This renders most traffic irrelevant for state purposes, leaving the vast majority of it to be dropped. A block diagram of an example filter is depicted in Figure 4.6.

The specific steps in the filter pipeline are user-defined but can be summarised in a high-level example, as depicted in Figure 4.6. The filter clones incoming traffic and forwards one stream out towards the primary middlebox to limit the number of

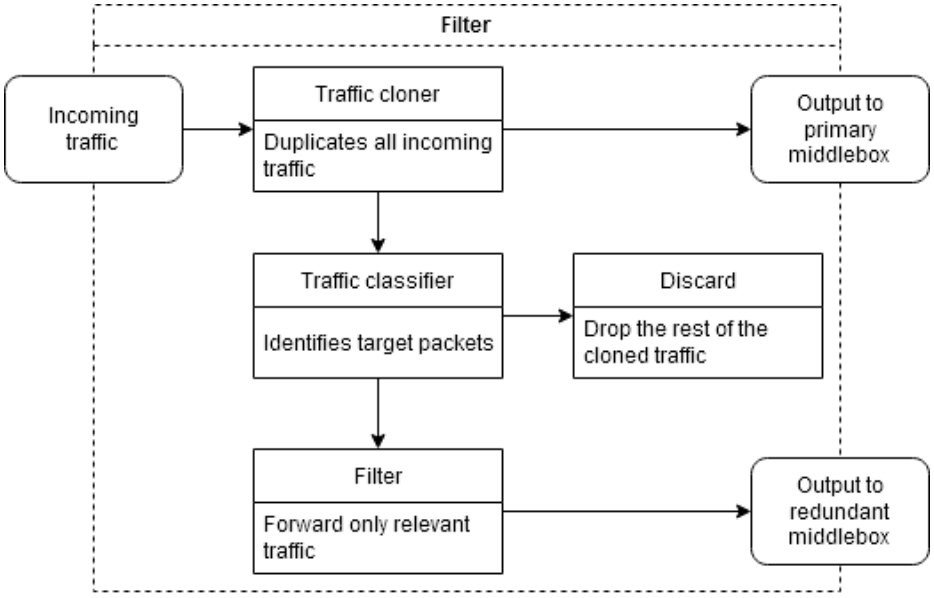

Figure 4.6: High-level diagram of a generic filter structure, with the arrows indicating the flow of operations for the packets that pass through the stages or operations defined.

processing steps applied to failure-free operations. Classification is then performed on this duplicated traffic, with the vast majority, or 95 to 98%, dropped with the filtered subset, then forwarded to establish state in the redundant middlebox. This example is costly, with all traffic requiring a write to memory to clone its contents, and the far cheaper operation of traffic classification (reading memory) performed only on the duplicated traffic. Another pipeline example can consist of all incoming traffic classified first, with only state-relevant packets then cloned and directed to their respective middleboxes. This might incur traffic re-ordering or delay on the most important packets to these devices however. Traffic is typically identified by a standard 5-tuple hash (source port and IP, destination port and IP and protocol), with most classification operations consisting of stateless checks for the presence of flags, although this can once again be extended to maintain awareness of flows that it is monitoring, rendering it a stateful device. The implementation specifics are up to the user and their system requirements, with examples of filter logic presented in Section 4.5.1 including pipelines of both examples given here.

4.3.2 Service restoration

So far, the discussion of the filter has focused on one direction, duplicating state-generating packets to pre-populate tables to improve redundancy. This is the most

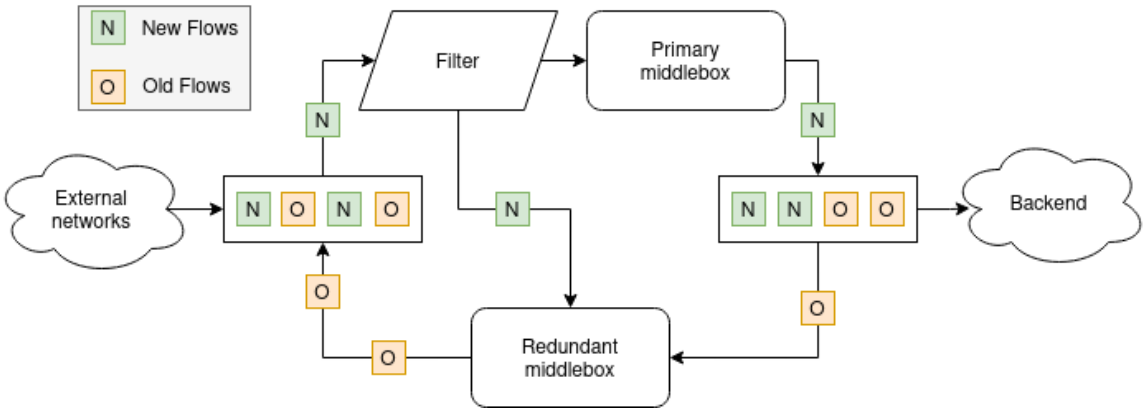

Figure 4.7: Switchover points bleeding flows slowly till all established flows in the redundant path finish or expire

simplistic implementation of Katoptron's operations, with returning traffic simply forwarded through the filter as a bump in the wire. Upon restoration of the primary middlebox, switchover will be triggered by engineers from the redundant unit back to the original. However, doing so would trigger a loss of state on any traffic still traversing the redundant middlebox, resulting in short-term service degradation as flows reset. The return path can be expanded to facilitate 'service restoration', with Katoptron able to support two lossless restoration techniques:

- An additional filter on the egress of the middleboxes to acquire state in a similar fashion to the ingress
- Expansion of the point of failover to bleed flows slowly from the redundant path by redirecting newer flows to the primary

The first approach utilises an additional filter operating in a similar fashion to the ingress, replicating key packets to form an internal state. As these packets are emerging from the network itself to the external network, the targeted packets concerning state are primarily those of the server responses to these flow establishing packets. This approach is the more naive of the two, with restoration to the original middlebox consisting of a forced switchover regardless of traffic in flight that, even with the filter, will cause some degree of packet loss for those currently being processed. The second approach alters the switchover points where failover occurs, expanding functionality to maintain per-flow state. When service restoration is triggered, both middlebox paths remain active. Established flows continue to be directed via the redundant path so as not to disrupt packets in-flight, while new flows are then directed via the restored primary middlebox, as demonstrated in Figure 4.7. This

allows Katoptron to slowly drain flows from the redundant path until all flows have completed, upon which its path can be deactivated manually by network engineers. It requires some measure of statefulness to implement, but greatly minimises further state loss during switchover and ensures all traffic in-flight is not lost by service restoration.

4.4 MiMi prototype

This section details the implementation of the prototype MiMi design shown in section 4.2. To evaluate its feasibility, a software prototype was created consisting of a container-based distributed architecture formed of 3 mechanisms joined in a simulated network testbed:

- A set of load balancer VNFs for the purposes of evaluation
- A set of drivers for the extraction or recreation of state
- A state distribution middleware service

The evaluation attempts to determine MiMi's compatibility with existing middlebox technologies, in this case a load balancer NF. This type of application maintains a mapping between information derived from incoming traffic flows and internal logic and a selected backend address. For middleboxes, especially those implemented in hardware, this information is typically limited to the standard five-tuple of source IP, source port, destination IP, destination port and protocol (generally TCP). Due to limited access to real-world blackbox appliances, software equivalents have been used to best emulate two scenarios for testing using an OpenFlow-based load balancer built as a Ryu application and a NetFilter instance using a source NAT policy. This section will break down the services and their components.

4.4.1 Middlebox Scenarios

Two middlebox implementations were created to evaluate the two approaches for extracting state; drivers inserted directly into a whitebox (open and modifiable) VNF and interpreting the logging output of a closed greybox VNF. The two chosen technologies consist of OpenFlow and IPTables (Team, 1998). They are configured to replicate the functionality of load balancers in a near-identical fashion, consisting of a load balancer that utilises the hash of the incoming packet to select a backend for each flow. Both technologies and their implementations are expanded below.

4.4.1.1 OpenFlow-based Load Balancer

Ryu is a component-based SDN framework built with Python and implements support for the OpenFlow protocol. It is used throughout both MiMi and Katoptron's proof-of-concept implementations for its ease of development and integration with other networking tools. TCP packets are the only packets targeted, with an index generated from a hash function of the source IP and TCP port and modulated against by the total number of backend servers. This selects the specific destination server from the list, with the selection not completely consistent with each hash to introduce some variation and state to be lost. For MiMi, the load balancer integrates a large number of additional features that are used at different stages of its evaluation, enabled by a configuration file external to the load balancer. These features include threaded polling and publishing mechanisms, which message passing system is used, publishing methods and the load balancing itself. These features are shared across the main and backup instance and are separately enabled or disabled through flags. Packets received by the load balancer are split into their respective fields, with ARP requests responded to and TCP traffic distributed to specific backend servers before writing the flow to the datapath.

4.4.1.2 Kernel-based Load Balancer

IPtables is a user-space utility program, native to the Linux kernel, that allows the user to configure IP packet filter rules. These rules correspond to specific sections of the packet processing chain: prerouting, input, forward, output and postrouting. These stages within the chain happen sequentially, with stacks of rules in each stage allowing for targeted modification of how packets are processed or forwarded. For example, rules inserted into the INPUT table will affect all packets that are to be locally delivered, whereas rules in the POSTROUTING table can be used to alter anything as they leave the overall chain after routing decisions have been made. For testing, the IPtables is configured within the testbed initialisation with a series of PREROUTING and POSTROUTING rules that establish the load balancer mechanism. These consist of a set of PREROUTING NAT table rules that match client IP addresses and modify the destination address to one of the five backend servers using a randomised probability. The POSTROUTING rules alter the origin address for returning messages to mask the backend server IP addresses using the front-facing virtual IP of the load balancer and a logging rule to enable pushing its logs to the rsyslog server, further discussed in section 4.4.4.2. As a kernel-space program, IPtables is far more performance-focused than the OpenFlow variant, used to evaluate MiMi and the external log-interpretation approach under heavy load.

4.4.2 State repository and distribution

State is retained on a short-term basis, with an external datastore retaining current and near-current state to be distributed to the redundancies. The MiMi prototype utilises two state distribution services in its evaluation: Redis and Kafka. Redis is an open-source in-memory datastore that retains information in key:value pairings, built in C. Apache Kafka is a distributed stream processing platform that publishes data streams in real-time, built in Scala. Each is used in its own evaluation, with Redis utilised for the majority of the evaluation, alongside several approaches to publishing. Both services are hosted externally from the testbed on the host system and are configured to realise a Publish/Subscribe distribution mechanism. This is done to allow for scalability and greater separation of the VNFs from the datastore. Each VNF subscribes to this channel at initialisation, with publishing configured to distribute state received as soon as it is received. Redis and Kafka implement these distribution systems differently, but they are used interchangeably in experimentation, with evaluations focusing on differing techniques to publishing methods as discussed below.

4.4.3 Publishing methods

There are two key factors to publishing state that must be taken into account: the speed at which state is received, which affects how accurate it is (the more up-to-date, the more accurate its use), and the volume of traffic consumed in doing so. One of the stated goals of this thesis' work is to ensure that the overall impact on the rest of the system is kept to a minimum so as not to affect failure-free operations. However, any propagation latency from batching techniques used to reduce the overall volume of traffic being broadcast may also affect its accuracy and, therefore, its usefulness. In the evaluation, four approaches to publishing flows from the primary load balancer are evaluated in different scenarios: a direct push to the Redis queue, the same mechanism in Kafka, a batched publish and the selected fields publish function. These first two are simple to implement, using in-built mechanisms to serialise flows directly to the queue with no other logic in place. The batch publish function expands upon this with a Python-Redis pipeline. Pipelines are a simple subclass of the Redis Python client that allow for multiple commands to be buffered into a single request. They are typically used to reduce the overall volume of traffic by minimising the number of sent and received TCP packets between the client and the server. Due to technical limitations of the virtualised testbed, 100 miliseconds is used as the base level of delay to batch instructions, as the smallest value the system can reasonably achieve during testing while also being sufficiently long as to batch enough instructions at once. Flows are batched within a 0.1-second time period into this pipe to be executed as a batch communication to Redis. Finally, the "part publish" method consists of

reducing the data sent to Redis to a subset of the fields.

4.4.4 State Drivers

For the evaluation, both a white and greybox application are targeted, formed of two load balancers with different implementations. Both of these scenarios perform packet-level load balancing and offer some form of state control, either in the form of internal observable state or exterior configured messages. Early in this exploration, we also attempted to integrate work with the HAProxy project, but the user-space version of HAProxy relies heavily on the socket layer of the underlying OS and it was technically impossible to extract this state.

4.4.4.1 Direct extraction driver

The OpenFlow application implements a reactive load balancing control using a 5-tuple hash to randomise backend server selection and exact match rules to rewrite the MAC and IP addresses, forwarding the packet to the correct output port. The MiMi driver is implemented as part of the application and uses a Redis Python library to synchronise with the remote state repository. Several of the publishing mechanisms discussed above in Section 4.4.3 were directly inserted into the primary Ryu load balancer. These mechanisms are utilised during the flow instantiation from packets received at the end of the decision-making process, with an identical copy of the flow being pushed to the dataplane distributed to the middleground instance. A listener sits on the redundant Ryu load balancer on a separate thread, pushing this flow to its own dataplane to maintain a direct 1:1 copy of the current flow table. The CPython implementation of the Python language is single-threaded by default through an enforced global lock to ensure safe concurrency. However, there are a number of libraries and operations that release the global lock and allow for multi-threading. This approach is highly accurate, as the flows match exactly across the primary middlebox and any redundancies it is replicated to. Whitebox scenarios, while uncommon, present the best scenarios for motivating direct modification, as they overcome many of the issues faced by blackboxes and even greybox designs.

4.4.4.2 Log interpreter driver

For the log interpretation evaluation, the Ryu-based load balancer is replaced with an instance of IPTables. A set of rules is inserted into the NAT table for the post-routing chain that selects a random backend server and forwards the traffic to it. In this implementation, because the NetFilter rules are executed at the kernel level, we opted to design an external MiMi driver instead. As a summary, this driver uses the logging capabilities of NetFilter to output its actions via a syslog server, which can

extract the relevant fields from this logging and interpret its contents. To be more specific, it is configured to generate logs with an identifiable prefix from the INPUT chain using an in-built action LOG. This default action will send all generated logs to a Python-based syslog server listening on a UDP port, polling every 0.5 seconds for the tagged logs to pull new flows in batches. Logs pulled are decoded into byte strings and separated into fields before being serialised into JSON and published to the Redis instance. For these experiments, the redundant unit is perceived as an open and modifiable VNF for the purposes of enabling this approach while allowing the primary packet processor (the middlebox/blackbox) to remain undisturbed. The consumer (the backup load balancer) will then pull these JSON messages on its own separate thread from the primary load balancing operations and use the five-tuple to create a flow and push it to the datapath.

4.5 Katoptron prototype

This section details the implementation of the prototype Katoptron design and testbed for the purpose of evaluating its effectiveness and performance. Depicted in Figure 4.8, this takes the form of a replicated network ecosystem and utilises several tools to explore a variety of middlebox targets and use cases. The figure represents how the primary components of the system interact, with automated clients generated through separated configuration files utilising different mechanisms to evaluate the prototype. As discussed in Section 4.3, this section will be broken down into the specific components of the testbed and prototype parts of this implementation.

4.5.1 Traffic Filter

The traffic filter is the most important aspect of the Katoptron design and it is hinged on several technical assumptions made in prior sections for allowing continued state through replication of state-forming packets without loss of generality. It is implemented using the Click programmable router (Kohler et al., 2000), a modular and extensible packet processing switch formed of individually defined elements chained in their operations. Click is a user- and kernel-space programmable router that is both highly efficient and open source. Individual functions are defined as elements and chained together in a pipeline structure using the Click configuration language. These elements are implemented in C++. Two kinds of filters were implemented for Katoptron's evaluation: a flow initialisation filter and first N filter. The first N filter is demonstrated in figure 4.9. Divided into components, it is formed of default elements that are predefined to Click to minimise the extent of functionality that needs to be built bespoke for its implementation and evaluation. In a short

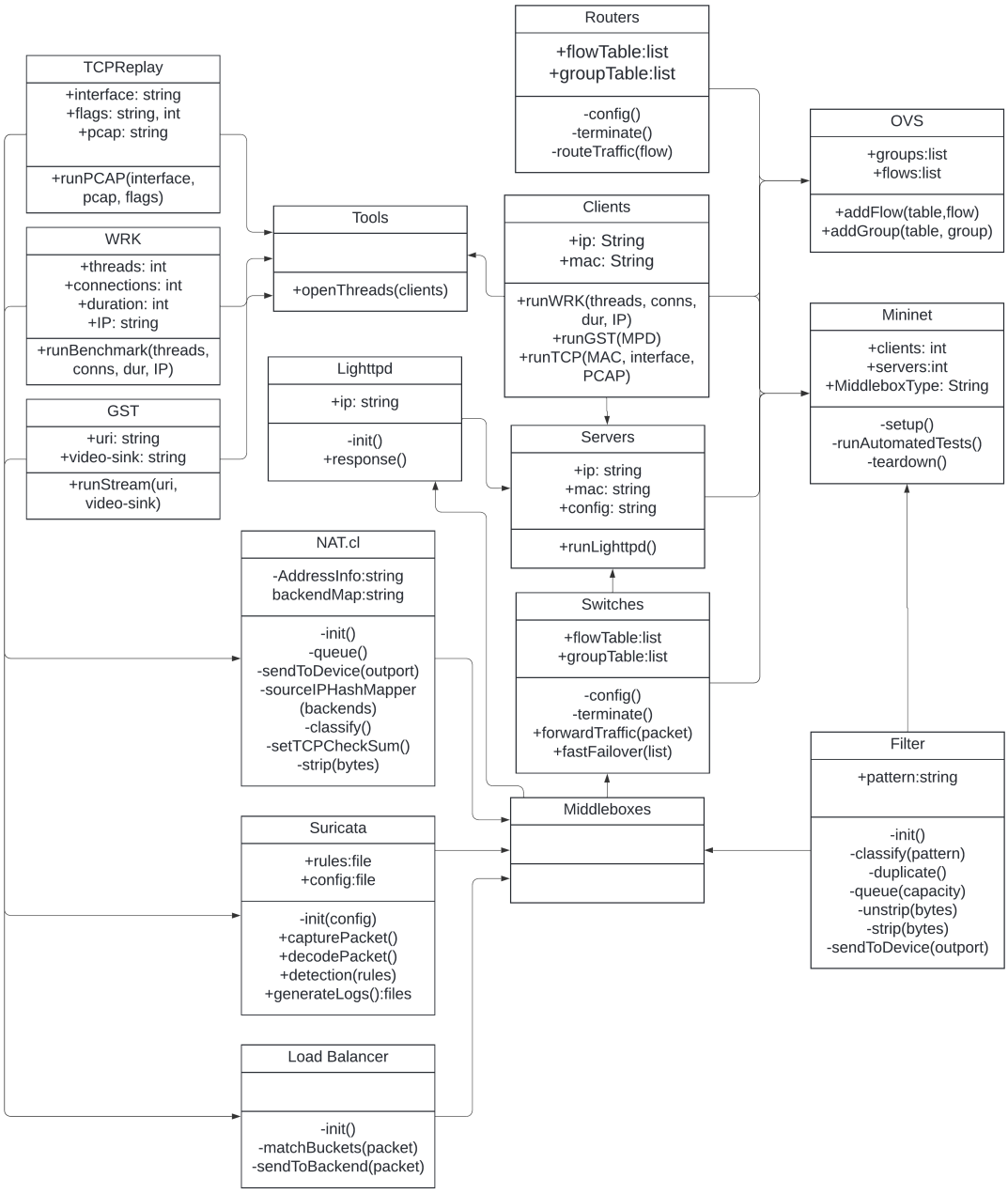

Figure 4.8: Katoptron prototype class diagram overview, depicting the components constructed for the tested implementation. Not depicted are the management layers for instantiation and teardown built using open-source tools, scripting and available MANO.

summary, the filter receives traffic and clones five initial packets from each flow. What follows is a technical breakdown of the steps taken and the elements involved.

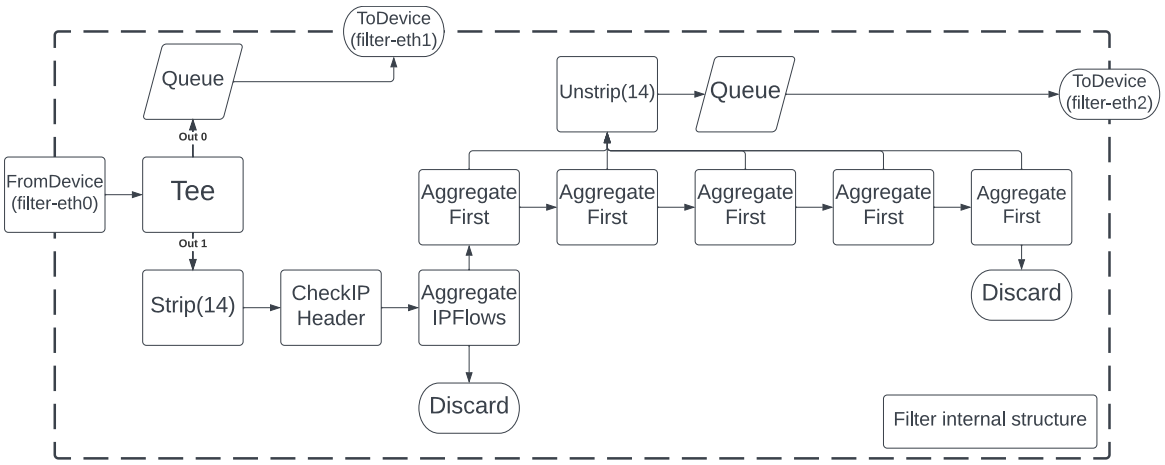

Figure 4.9: Five packet aggregator filter in Click, with arrows indicating the flow of traffic through the elements from left to right, starting at FromDevice. For example, at the tee, two copies are made of the packet and either sent out of the interface or sent to the next step in the chain.

4.5.1.1 Pipeline breakdown

For the first-N filter depicted in Figure 4.9, packets received are replicated into two sets: the primary route, where they are forwarded to the middlebox, and the traffic to be modified. After header verification, this implementation targets the first five initial packets of each stream and is built using pre-existing primitives of Click "Aggregate First"; using a stateful check, incoming flows are hashed, and only the first packet of each hash is emitted through output zero to the redundant device, while the rest is directed to the next aggregateFirst. Chained in a line, the overall effect is to take the first five packets of each flow using existing elements. These elements are stateful, retaining information on a per-flow basis in the form of counters to identify whether this is the initial packet in a flow. This information is minimal however, using direct hashing of the five-tuple against a direct lookup, and is easily recovered if lost as a deterministic action, posing no additional risk of increasing the impact of lost state by introducing more. The second implementation, depicted in Figure 4.10, uses a different order for classifying SYN packets. Incoming packets are received and classified as to whether they are ARP requests, responses or other traffic. Requests are discarded out of output 0 of eth-classifier1, with ARP responses (responses from the client to the server for destination's location) forwarded towards the server and all

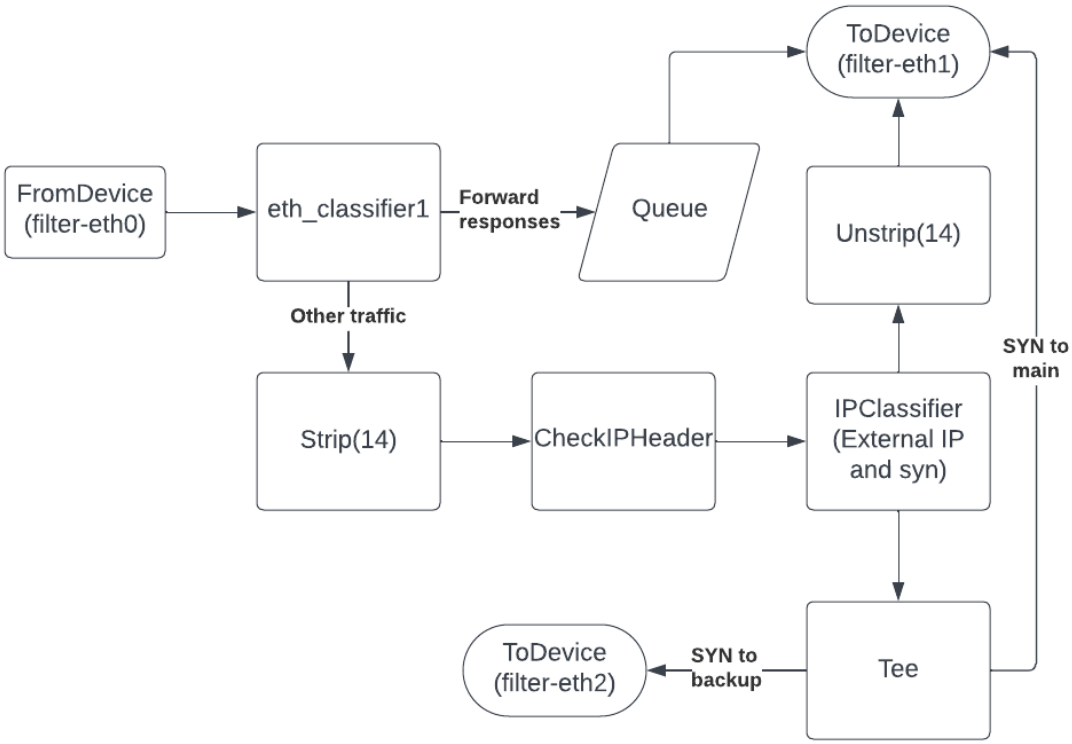

Figure 4.10: SYN filter in Click

other traffic continuing along the pipeline. Ethernet headers are stripped, with an IP header check to confirm this is IP traffic. The second classifier identifies if it's intended for the backend IP range and a SYN packet. All non-SYN traffic is then forwarded towards the backend servers, with SYN packets duplicated and distributed to the two middleboxes. This approach uses a stateless check and retains no information, minimising the volume of traffic cloned to reduce the processing costs of doing so but potentially incurring additional delay through multiple packet classifiers on all traffic flows for incoming packets. The two pipeline approaches are similar in structure but built to the minimum requirements for state for each scenario: the first packet of each flow for web traffic and the first several packets for security fingerprinting. The order of operations also differs as part of the evaluation; cloning packets received is the heavier of the two actions and incurs greater delay as it must write to memory, whereas classification only reads. This is done to examine the effect of both delay incurred by the filter as well as potential packet reshuffling from the singularly delayed SYN packets in the SYN filter (the rest of the traffic continues whilst they are isolated for cloning before being sent onwards). Furthermore, the first-N filter is mildly stateful,

maintaining awareness of flows on multiple elements unlike the SYN filter which is an otherwise stateless pipeline.

4.5.1.2 Filter implementation technologies

The two filters utilised within the evaluation are implemented in the Click programmable router. These filters are platform-agnostic, with the primitives supported by Click easily replicated in other similar technologies, such as P4, but may not be as well placed for their use. During the evaluation, a data plane filter version was implemented using the P4 language (Bosshart et al., 2014), compatible with the P4 Behavioural Model v2 (BMv2) switch and based on the five packet sampling variant. This P4 filter implementation uses a hash function and an array of registers to support a stateful flow tracker. Unfortunately, due to hash collision, the P4 program must oversample sample traffic to guarantee that every set of initial packets from a new flow is sent to the redundant path. Due to the performance limitations of the BMv2 switch, the evaluation uses the Click variant.

4.5.2 Service restoration mechanics

Service restoration is a manually triggered action in a real-world context, to be initiated when failures in the primary middlebox have been resolved. To prevent state loss and facilitate this in the OVS switches on either side of the two middleboxes, this implementation operates an automated LEARN action in the OVS platform. This rule, held in a lower priority to failover groupings, utilises the "LEARN" action to generate return flows on the secondary point of failover (the backend of the bypass) upon packet matches. Matches are made only on the returning path, with generated rules being short-lived reversals of the packet's fields. They ensure that during restoration, any established flows maintain service via the established state on the backup middlebox. This helps to minimise delay and any loss of state from the switchover from the redundant path back to the primary. Figure 4.11 is an example snippet of the restoration mechanic, as well as an example of the flows it generates in Figure 4.12:

```
ovs-ofctl add-flow bs1 'in_port=2,ip,tcp,priority=10,actions=learn( \
priority=11,idle_timeout=60,dl_type=0x800,nw_proto=6,dl_dst=dl_src, \
dl_src=dl_dst,nw_dst=nw_src,nw_src=nw_dst,tp_dst=tp_src,tp_src=tp_dst,\
output=NXM_OF_IN_PORT[]),goto_table=1'
```

Figure 4.11: OF flow table rule used to create flows for returning path traffic to slowly bleed off from the redundant path

```
 cookie=0x0, duration=70.657s, table=0, n_packets=25, n_bytes=1850, priority=10,tcp,in_port="bs1-eth2" actions=learn(table=1,idle_timeout=60,priority=11,eth_type=
0x800,nw_proto=6,NXM_OF_ETH_DST[]=NXM_OF_ETH_SRC[],NXM_OF_ETH_SRC[]=NXM_OF_ETH_DST[],NXM_OF_IP_DST[]=NXM_OF_IP_SRC[],NXM_OF_IP_SRC[]=NXM_OF_IP_DST[],NXM_OF_TCP_DS
T[]=NXM_OF_TCP_SRC[],NXM_OF_TCP_SRC[]=NXM_OF_TCP_DST[],output:NXM_OF_IN_PORT[]),resubmit(,1)
 cookie=0x0, duration=70.665s, table=0, n_packets=0, n_bytes=0, priority=2,arp,arp_tpa=10.0.0.20 actions=group:1
 cookie=0x0, duration=70.662s, table=0, n_packets=12646, n_bytes=250972945, priority=1,dl_dst=00:00:00:00:01:ff actions=group:1
 cookie=0x0, duration=70.660s, table=0, n_packets=85020, n_bytes=5675380, priority=0 actions=resubmit(,1)
 cookie=0x0, duration=58.779s, table=1, n_packets=0, n_bytes=0, idle_timeout=60, priority=11,tcp,dl_src=00:00:00:00:01:03,dl_dst=00:00:00:00:01:ff,nw_src=10.0.1.3
,nw_dst=10.0.0.1,tp_src=80,tp_dst=49030 actions=output:"bs1-eth2"
 cookie=0x0, duration=58.779s, table=1, n_packets=0, n_bytes=0, idle_timeout=60, priority=11,tcp,dl_src=00:00:00:00:01:01,dl_dst=00:00:00:00:01:ff,nw_src=10.0.1.1
,nw_dst=10.0.0.2,tp_src=80,tp_dst=48000 actions=output:"bs1-eth2"
 cookie=0x0, duration=58.218s, table=1, n_packets=0, n_bytes=0, idle_timeout=60, priority=11,tcp,dl_src=00:00:00:00:01:02,dl_dst=00:00:00:00:01:ff,nw_src=10.0.1.2
,nw_dst=10.0.0.3,tp_src=80,tp_dst=52882 actions=output:"bs1-eth2"
```

Figure 4.12: Example of generated restoration rules in OpenFlow table

4.5.3 Middlebox Scenarios

For prototyping and evaluation, a number of network functions were required. These are implemented in VNFs for ease of development rather than physical hardware, but have been approached as if they were blackboxes; no modification or interference in their operations, treated as an isolated entity. The functions chosen were prioritised for their stateful mechanisms and common deployment in hardware over software. The details of their use are expanded upon below in the evaluation in Section 5.3.

4.5.3.1 NAT

A reverse Network Address Translator (NAT) serving as the entry point to set of backend servers is a common use case for controlling outside network awareness of server information. It acts as the front-facing IP address for incoming requests, distributing traffic between the five servers present in the CDN using fixed-source IP hashmapping. This reverse NAT has been implemented in Click and handles standard operations including bidirectional traffic, NAT functions and ARP. The Click-based NAT is statically configured and modelled after an existing implementation to Click "mazu-nat", commonly used in other evaluations such as FTMB (Sherry, Gao, et al., 2015).

4.5.3.2 IDS

Suricata (O. I. S. Foundation, 2022) is an open source high-performance IDS and IPS that is widely used in software deployments for its ease of configuration and extensibility. It utilises signature-based detection and can be configured to operate passively (for the IDS) and actively (for the IPS) on traffic received. For evaluation, we used the Emerging Threats open ruleset (Inc, 2021), another popular and widely used open-source ruleset utilised by other software IDS such as Snort (Roesch, 1999). Suricata is configured to serve in an IDS role, generating alerts based on incoming traffic to the server and monitoring connections and flows, but not preventing or affecting the overall traffic. Each instance of Suricata during the evaluation utilises

32 packet processing threads and 4 management threads, with a total of 23,681 rules from the ET rulesets with coverage for phishing, malware, FTP, exploits and so on.

4.5.3.3 Load balancer

Load balancers are often deployed in hardware and are one of the most common forms of middlebox utilised in networks. For Katoptron's evaluation, two implementations of a load balancer were utilised. The first implementation is based in IPtables using a similar design to the MiMi evaluation variant, using a hash randomisation load balancing algorithm. The second implementation utilises OVS, or Open vSwitch, an open-source user-space programmable switch implemented in C and widely utilised by both network virtualisation technologies and protocols such as OpenFlow, with a module now integrated into the Linux kernel. This static OVS implementation is similar in structure to the flow instantiation of OpenFlow, configuring a static switch on the path. This consists of a set of weighted bucket rules, dividing the traffic between the five backend servers in the testbed. This establishes a level of traffic awareness in the load distribution, increasing the degree of state retained by the network system beyond a static hash approach.

4.5.4 CDN

The CDN used to evaluate the various middlebox functions is formed of five backend servers, each hosting the same content. These servers consist of instances of "lighttpd" (Bosshart et al., 2014), an open-source high-speed webserver compiled in C and the in-built Python HTTP server libraries. The files hosted consist of a set of HTTP files of varying sizes to provide material for the clients to request for experiments, as well as an MPEG-DASH video with several bitrate levels. This video, 'big buck bunny' (Benjamin Rainer et al., 2012), is a popular testing mechanism for DASH player streaming. This CDN backend is not a service or network function fulfilled by middleboxes, but a common network scenario and useful for the evaluation of middleboxes often used in conjunction. This includes network functions such as load balancers, firewalls and NAT middleboxes that may be used by a server for controlling the flow of traffic and its interactions with the backend servers.

4.6 Summary

This section has presented the design and prototype implementations of both Katoptron and MiMi, including how they may be realised in existing software and the specific technologies used in their evaluations. The MiMi design implementation serves as an initial exploration into the topic of state preservation through external

methods, with the majority of its work concerning the distribution of state via the datastore and the logging interpreter system. Its prototype implementation has focused on evaluating the effectiveness of this external driver approach as well as possible latency, accuracy of copied state and overall feasibility. The Katoptron design implementation has focused on the means by which this filtering approach can be realised in a multitude of technologies to avoid limiting its flexibility and potential for adoption. Its prototype implementation has focused on ensuring the evaluation of the filter is at the forefront to best determine its effectiveness across multiple technologies, levels of traffic and targeted middleboxes. This chapter has presented both prototype/proof-of-concept implementations that form the two halves of Remediate. The following chapter will detail the evaluations of each prototype, including the testbed design, approach, workloads, tools and their overall effectiveness in each of the aforementioned key evaluation goals.

Chapter 5

Evaluation

In this chapter, the designs of the resilience framework and the implementation of the two mechanisms prototype implementations are evaluated. Firstly, in Section 5.1 the topologies and workload used to evaluated the resilience of each separate project are described, with further details in their respective sections. Secondly, in Section 5.2 a breakdown of the testing approach of MiMi is discussed, followed by an extensive evaluation of differing traffic scenarios and sampling rates and their impact on the effectiveness of this approach to redundant VNFs. Finally, in Section 5.3, the filter approach of Katoptron is evaluated in multiple implementation technologies and multiple target middlebox functions.

5.1 Experimentation Platform

MiMi and Katoptron share aspects of their prototype designs and evaluation processes, which will be discussed in this section. Details specific to either MiMi or Katoptron will be discussed in their respective sections.

5.1.1 Testbed environment

The testbed environment, presented in a high-level diagram in Figure 5.1, is similar for the two projects for the purposes of testing, with some key differences. To begin with, the experiments performed in this evaluation are made from a series of independent programs tied together in a unifying Python environment. The strawman implementation is implemented using an emulation tool called Mininet. Mininet (Lantz et al., 2009) is a network emulation tool that allows for rapid testing and development of SDN technologies, built with support for technologies such as OpenFlow (McKeown et al., 2008b). It uses Linux namespaces (Kerrisk, 2013) to emulate network isolation with veth pairs and SDN technologies such as OpenvSwitch

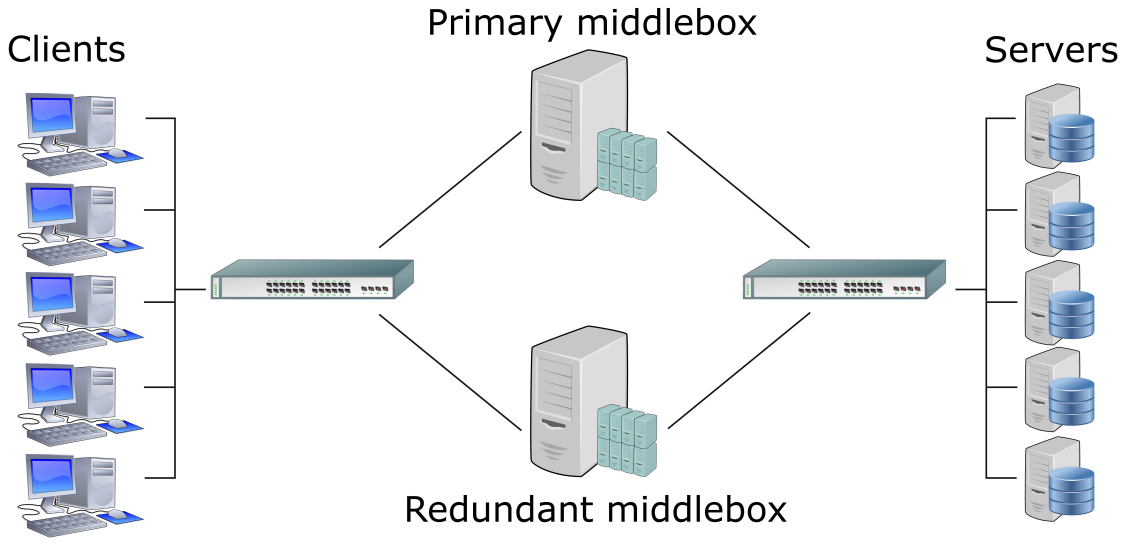

Figure 5.1: Simplified high-level diagram of the testbed used across multiple experiments - a client/server model with traffic served through middleboxes providing network functions typical at network gateways. Each experiments details differ and are expanded in more detail in their respective sections

allowing networked communication. The environment is automatically configured using config files at initialisation. Upon instantiation, a number of elements are created including the topology, mininet nodes, servers and experiment processes. This also includes any project-specific elements, such as the middleware for MiMi, which may alter the topology or flow of traffic in accordance with the experiment. For example, the Suricata experiments will establish hosts running instances of Suricata and use forwarding via the IP address route with ARP enabled, while other projects use a NAT function to provide routing to the backend. Clients are statically configured to operate in accordance with their automated tests, with more details on these configurations discussed in the evaluation in sections 5.2.1 and 5.3.1.

These environments also configure the services used by the two projects. Client processes used to generate traffic loads such as WRK (Glozer, 2023), scoot-player (Broadbent, 2015) and GST (Taymans, 1999) (detailed below in Section 5.1.2) are statically configured to run on a per-client basis according to the specifics of the experiment, utilising Python subprocesses to manage discrepancies in completion time. The topologies used for the testbeds vary between both projects and experiments but all follow the broad structure of Figure 5.1. The testbed environment for MiMi uses a singular topology with a client/server network split into separate

subnets, joined by a load balancer. The testbed for Katoptron is similar. Built off the structure of MiMi's evaluation, Katoptron's testbed is divided into three independent topologies, one for each experiment variant: a reverse NAT, a weighted load balancer and an IDS/IPS. These topologies each consist of the same Mininet layout with modifications specific to their use-case, establishing the clients, backend servers, fast failover rules and initialisation of the respective technology (e.g., the testbed network is configured to mirror the network of the malicious traffic captures). To emulate failure scenarios, link failures are used to force the OpenFlow switches to forward traffic via the redundant paths. Link failure is achieved using the link sensing capability of the Ethernet layer. Without loss of generality and for ease of testing, link failures on either or both ends of the service are simulated by manually shutting down the respective interface. The number and rate of these link failures back and forth vary between experiment scenarios. The implementation of this mechanism differs slightly between MiMi and Katoptron and will be detailed in their respective sections, but it is ultimately done to achieve the same effect.

5.1.2 Tools

Tool	Traffic	Details
Scootplayer	MPEG-DASH	Experimental logging player
WRK	HTTP requests	HTTP benchmarking tool
Gstreamer	video formats	Multimedia framework
TCPReplay	PCAPs	Packet capture and replay tools

Table 5.1: A brief summary of the tools used for the evaluation and their use

The prototype implementations for the two publications both use traffic generator tools to create realistic workloads. These tools (summarised in Table 5.1) will be discussed in more detail here with the workloads summarised and further elaborated below in their own section. The first workload (short-term *WEB* flows) consists of requesting small web/HTTP objects from the server to the client. This workload is generated using the WRK (v.4.1.0) HTTP traffic generator running on two threads for each client, requesting a small web object (two sizes available: 5.7 and 617Kb). The number of connections running in parallel varies between workload per test. The second workload (long-term CDN streams) consists of MPEG-DASH streams of the "Big Buck Bunny" test video common to DASH testing (Benjamin Rainer et al., 2012). This is served from the backend to the client via one of two video streamers. For Mimi, Scootplayer (Broadbent, 2015), an experimental DASH streamer with logging support. For Katoptron, the clients use a dummy gstreamer plugin that emulates the behaviour of an MPEG DASH client but does not perform any video decoding.

The players differ between the two publications due to replacing Scootplayer with gstreamer for its limitations in handling intermittent connection failures. This stream defaults to the highest quality version (8000kbit) segmented in 1 second chunks, with chunk sizes varying between 100kb to 1.4Mb and quality representations split six ways from 2500kbit to 8000kbit. These settings are used as they provide a wide range of chunk bitrates for the player to utilise when handling disrupted connections and make observation of its error handling easier. In both scenarios, the servers use the lighttpd (Kneschke, 2003) HTTP server (v.1.4.45) to emulate the server. The final workload is not generated but instead uses live captured traffic from an existing dataset using TCPReplay. TCPReplay is an open-source suite of network utilities for replaying captured traffic. It is a popular tool for testing purposes, with a wide range of functionality beyond just broadcasting, including modification of playback as well as the contents themselves.

5.1.3 Workloads

Workload	**Tools**	**parameters**	**Metrics**
Short flows	WRK	threads connections object size	timeouts read errors write errors connection errors
Streaming/long flows	Scootplayer Gstreamer	clients duration	Buffer events Resolution changes Failed connections
Attack/traffic traces	TCPReplay CICID2017 dataset	Tracefile	Signatures Alerts

Table 5.2: The workloads, the tools used to generate them and their parameters

The evaluations utilise three workload models, summarised in Table 5.2, to emulate typical Internet traffic workloads. These consist of short and long term flows, as well as live captured traffic for Katoptron. The *WEB* workload generates HTTP traffic between the client end-hosts using the WRK software for static content and the serve end-hosts, with server static pages using Lighttpd services. This workload emulates short-lived HTTP traffic, typically generated by an Internet web server. To control the duration of HTTP flows, we use two objects of different size: a 5KB HTML page and a 627KB binary object. On each client, the WRK instances use two threads and our workload will run up to 500 concurrent TCP flows. To evaluate the performance of the workload, there are four statistics produced by WRK that are considered: socket connection errors, read errors, write errors and timeouts. Connection errors are any

refusal of a TCP connection that would be reported as a socket connection error. Read and write errors are reported TCP errors from a failure to read and write to a connection respectively. Finally, timeouts are any connections that fail to respond to a request in the default two second timeout window. For these experiments, the connection and timeout errors reported are the focus, evaluated against the RTT of the TCP connection, which sits within a range of 35-50ms for HTTP traffic; a transfer window of 50ms thus being far slower than a timeout, which defaults to 2 seconds in WRK. The *streaming* workload emulates a varying number of MPEG-DASH streams between the client and the server endpoints. The DASH client will actively switch between the 20 sets of encoded chunks at varying bit rates and resolutions in response to changing network conditions.

The workload runs a total of 375 parallel connections, split across five clients for a duration of 120 seconds. During an experimental run, we count the total number of buffer events (client does not have enough data to play the next frame), resolution changes (client selects a lower bit rate video format due to detected poor network condition) and the number of failed connections. Together, the two workload scenarios offer a typical level of normal web traffic, emulating both short and long flows with differing characteristics and responses to loss. For example, shorter flows, such as simple HTTP requests, will suffer less from intermittent failures as they are less likely to be impacted due to their short-lived flow duration, allowing for rapid replacement with a greater number of active flows overall through the system. Longer flows, such as those of the DASH stream, are far more likely to be observed by the end user, both from the greater potential for loss of flow from their longer duration and from the loss of chunks that may not be mitigated through buffering and the adaptive streaming technique being triggered.

Finally, the *attack* workload, used by Katoptron, replays traffic traces (PCAP files) from an open-source IDS evaluation dataset using TCPReplay. During replay, we modify the Ethernet header fields to match the host MAC addresses in the emulated topology. The trace files come from the CICID2017 dataset (Sharafaldin et al., 2018), which contains labelled network traces from real networks and the data represent traffic that spans four different days. A breakdown of the attacks for each day is depicted below:

- Tuesday - Brute force and scans for FTP and SSH
- Wednesday - DoS/DDoS and heartbleed
- Thursday - Brute force, XSS, SQL injections, dropbox exploits and portscans
- Friday - Botnet, portscan and DDoS

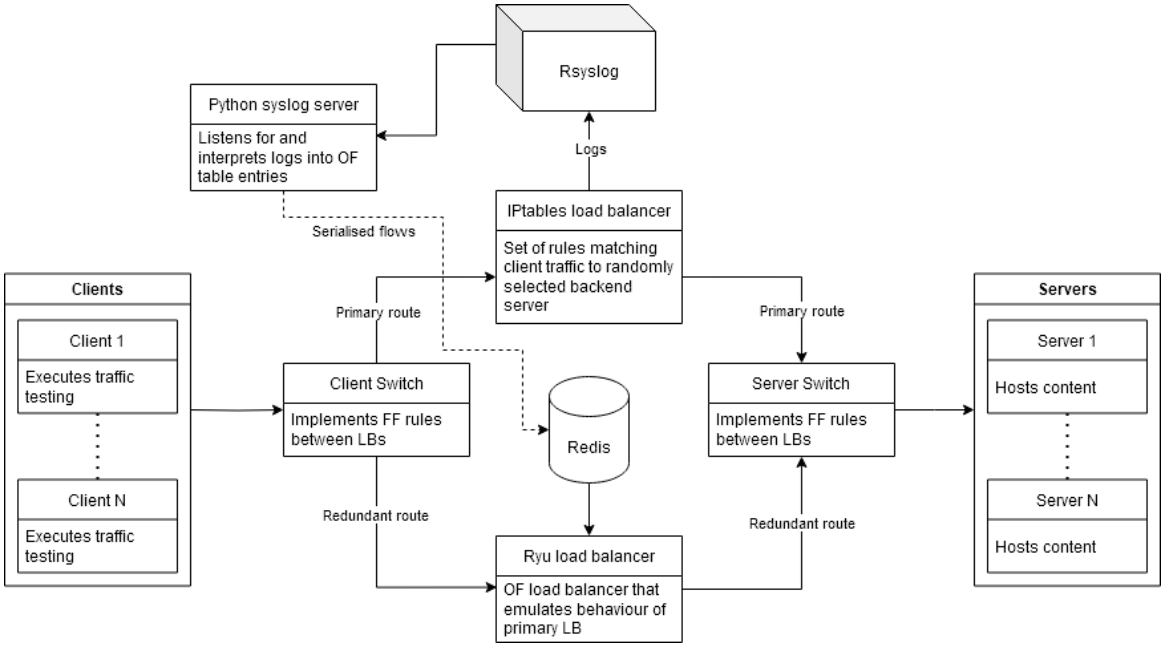

Figure 5.2: Experiment topology consisting of a client-server model as shown in Figure 5.1, showcasing the syslog variant of the evaluation, emulating a caching service and load balancer with full details of the testbed implementation

The workload is used exclusively to test the IDS middlebox, and during each experimental run, we record the number of signatures and the number of alerts per signature reported by the IDS instance. The first metric is the most important and reflects the number of unique attacks detected by the IDS, while the later metric reflects the number of unique instances of an attack detected. In order for an IDS to operate correctly, the first metric is essential, while the larger metric is less important.

5.2 MiMi performance evaluation

This section evaluates the performance of the MiMi resilience mechanism. The analysis uses network emulation to run a small-scale network topology equipped with the MiMi service and state repository. Both state extraction approaches are evaluated in their effectiveness at acquiring sufficiently accurate state for different types and loads of traffic, as well as sampling rates and the impact of sampling delay and batching.

5.2.1 Experimental Setup

The experimental topology is depicted in Figure 5.2, consisting of a set of clients and servers arranged to emulate a CDN/caching service. It consists of multiple clients accessing a load-balanced HTTP service supported by a set of servers, separated on subnets. This topology is realised using namespaces via Mininet (Lantz et al., 2009), utilising Linux's in-built namespaces to simulate the isolation between nodes, each with their own virtual interfaces with a fully fledged network stack. The switches are statically configured, with the topology divided into subnets to emulate different networks. The *primary* hardware load balancer exposes a HTTP service via a virtual IP address, with each request randomly redirected to a backend server. A *minion* VNF acts as a hot replica, available to remediate service delivery upon failure detection. The load balancers are implemented as Ryu applications using a round-robin distribution of traffic flows. The client and servers connect with the load balancers using OpenFlow-enabled switches configured with static L2 forwarding rules and Fast Failover group table entries, implementing a network failover mechanism. When the link to the primary middlebox is disabled, all traffic is rerouted to the minion VNF. This failure model serves as an effective analogue to how such a system would typically be implemented, observing for simple link breakages and switching to the alternate route if so. All of the components within this testbed are either software or virtualised equivalents of hardware (such as switches and routers) using Mininet, as it is an effective testing and simulation tool for its ease of deployment and accuracy compared to other simulation methods.

These experiments were executed on a Dell server (dual Socket Xeon 4114, 20 cores, 32gb RAM, Ubuntu 18.04) using the Mininet platform. For each experiment, four scenarios were conducted.

- Clean - No failures and replication disabled
- Copied - No failures and replication enabled
- Fail - Failures and replication disabled
- Failover - Failures and replication enabled

Firstly, the control or base scenario is labelled as clean in the results, with the system running without any failures to measure the performance when MiMi's flow replication mechanism is disabled. The second scenario is the same, but with the mechanism enabled to observe for impact, labelled as copied. Thirdly, failures are triggered at fixed intervals while flow replication is enabled, labelled as failover. Finally, the same scenario of failures, but with flow replication disabled, is labelled fail. To evaluate the effectiveness of the state recreation in

the redundant middlebox/minion VNF, the number of failed connections and overall traffic throughput are compared, as well as traffic-specific metrics. Each experiment is performed multiple times (from 5 to 10 iterations depending on experiment results) with an average result calculated to ensure consistency. For the WEB workload, the experiment runs for five minutes with a link failure triggering every 30 seconds, totaling five state transfers. For the DASH workload, 15 failures are triggered at the same evenly spaced 30-second intervals. It is worth noting that state is only mirrored in one direction, from the primary middlebox to the minion VNF. This is due to the design intending to be served as a failover mechanism to solve immediate loss of hardware, with restoration to the primary path an action taken by the network engineers in charge as they resolve the problem fully. This means that state is lost when the traffic is restored to the primary and magnifies the failure scenario beyond what would be encountered in reality, as a form of stress test. The number of connections per experiment is dictated by a relative 'sweet spot' of traffic load, wherein a sufficient volume of traffic is consuming the majority of available bandwidth with what the system can handle to better replicate typical operating conditions in an otherwise limited virtualised environment.

5.2.2 State mechanism designs

MiMi evaluates two approaches to extracting/recreating state from the primary middlebox. Following the vein of the three possible formats specified in Section 3.4.1, Middlebox Minions explores the first two scenarios: an open or modifiable VNF and a VNF with observable configurable output. The latter is more common and the main approach of MiMi, but the former is explored as a matter of course. The first approach utilises a driver inserted into the primary middlebox, serialising flow state and distributing it to the datastore. The driver experimentation intercepts and replicates OpenFlow instantiation commands, serialising it into JSON and sending it to the Redis middleware (Sanfilippo, 2009). This approach is an initial exploration of the concept and is not feasible in most real-world deployments. The second approach interprets the log output of the middlebox, which is commonly used for monitoring and troubleshooting purposes. Logs are generated from actions performed and distributed to the syslog host, giving access to a degree of decision-making of the middlebox. An external driver intercepts the syslog output and interprets its contents, extracting relevant fields to recreate the OpenFlow instantiation command before sending it to the Redis datastore. This second approach is agnostic of the technology of the primary middlebox, relying on fairly standardised logging mechanisms and practices. This driver must be adapted to fit the log output of its target middlebox, creating a small but not unreasonable barrier to entry.

5.2.3 Direct extraction evaluation

The first approach extracts OpenFlow flow table instantiation commands from the control plane channel, serialising and distributing it via the Redis datastore with the auxiliary middlebox receiving it from the pub/sub channel.

5.2.3.1 WEB workload

The first workload consists of the HTTP traffic using WRK, generating large numbers of short-term flows where the loss of state in the middlebox will reset active connections. This workload represents a significant portion of normal web traffic, requiring little in the way of persistent state, only needed for the very short duration of the flow to fulfil its request. These flows are short and able to quickly restart upon loss with WRK reporting on latency, request rate, total requests, read errors and timeouts. These breakdown in WRK to socket read and write errors, and TCP timeouts waiting for a response, with a default window of two seconds. The results are depicted in Figure 5.3, split between two scenarios: 1000 and 2000 parallel connections requesting a small 5.7kb file, divided across 10 threads and 5 clients equally. Failovers were triggered every 30 seconds, with state copied only in one direction (from primary to redundant middlebox with no state copied back on the return path), for a duration of 5 minutes totaling 5 triggered failovers.

Does MiMi reduce short flow timeouts? Overall, the web workload is robust against disruption as anticipated, with two trends emerging: TCP timeouts for connections are reduced by a significant margin, with a 41% reduced rate between stateless and stateful failover per 1000 parallel connections. This continues in the 2000 set, with a reduction of timeouts by 40%. Each connection is too short-lived to be hit by more than one switchover every 30 seconds, with retransmissions occurring only within their two second window.

Effect on traffic rates Timeouts incur retransmitted packets, increasing the overall traffic load from an average of 98K requests to 133K. Using the drivers, this increase is only 17% per 1000 connections (114K) and similarly halved for the 2000 set. There is an observable improvement in overall connection retention with no clear increase in latency due to the invasiveness of this technique, although the evaluation of this is impaired by the nature of the traffic and its short-lived statefulness.

5.2.3.2 DASH workload

The second workload consists of MPEG-DASH video streams using Scootplayer (Broadbent, 2015), generating a small number of long-term flows where the loss of state will

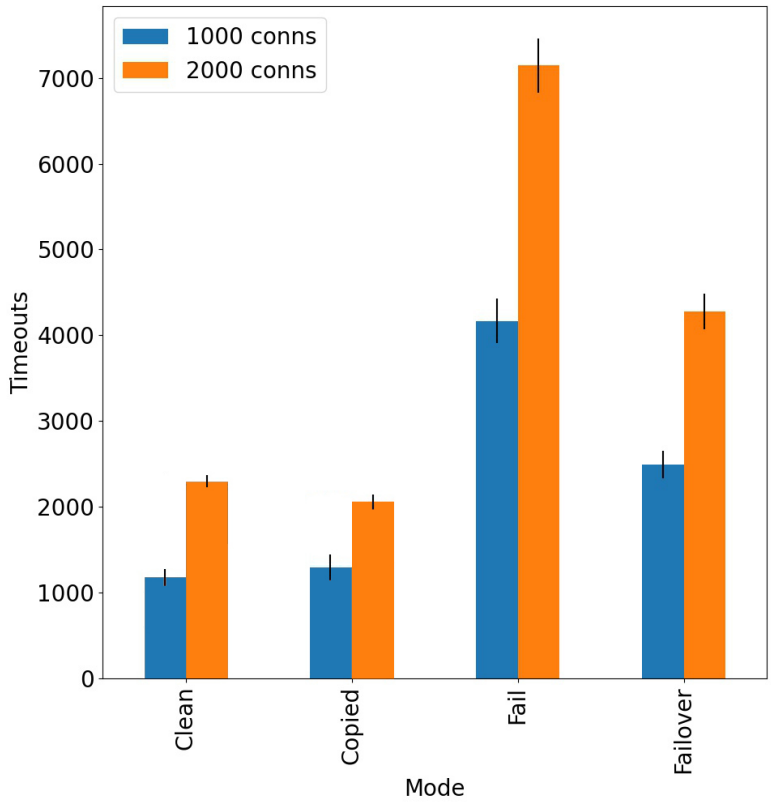

Figure 5.3: Reported timeouts of WRK connections for the 1000 and 2000 connection datasets

disrupt active connections, triggering buffer events or potentially severing connections entirely. This workload represents another form of typical web traffic, requiring a far more significant volume of persistent state with a greater risk of disruption due to their length. Compared to other DASH streaming tools, Scootplayer is relatively intolerant to disruption, lacking most video player mechanisms for resilience to disruption, and intended purely for testing rather than real-world use. The results are depicted in Figure 5.4.

Does MiMi reduce long flow timeouts? Overall, the results are consistent with the WEB workload showcasing a reduction in the rate of timeouts that occur but not a significant improvement. Starting with the 100 client set, the failure rate of connections is reduced by 18%, with a similar level for the 50 client set at 16%. The results suggest some improvement to this early approach, although the evaluation is impaired by the technical limitations of the testbed as the first iteration of these experiments, showcased in Figure 5.3 where the results for low traffic levels without

failovers still indicates timeouts due to being unable to handle any significant traffic load.

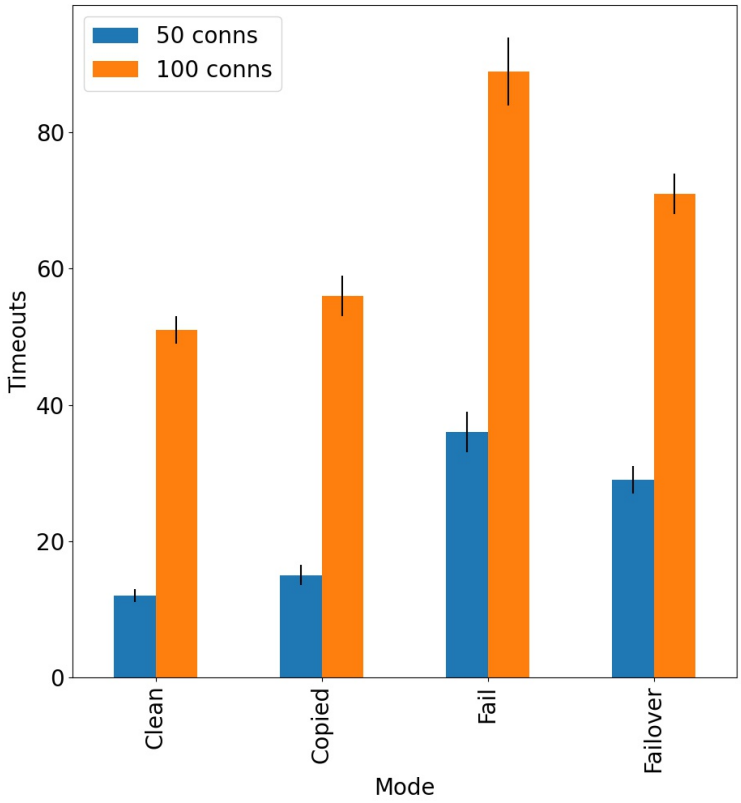

Figure 5.4: Reported timeout rates of scootplayer connections for the 50 and 100 connection datasets, with error bars generated from averaged results

5.2.4 Log-interpretation driver evaluation

The second approach, the logging interpreter/driver, derives key information from the output of the middlebox externally of the device, creating flow insertions with this information to be distributed via Redis. To evaluate the effectiveness of this second approach as well as explore the potential effects of utilising lesser-performing software as the remediation, an IPTables middlebox is used in place of the Ryu load balancer as the primary middlebox, with the redundant role now served by the Ryu instance. While both instances are software, there is a significant gap in performance regarding packet processing speeds between these two instances, serving as a stand-in replacement. To remove other potential performance mitigations, static flows replace the reactive switches in the testbed, as well as an increase in the size

of the requested file from 5.7kb to 617kb. Web page sizes have steadily increased in the last fourteen years, as reported by the HTTP archive, primarily due to the rise in the number of images and CSS elements. A typical HTTP header is within the range of 500–700 bytes, while a full webpage in 2022 averages 2.2MB, or 2200kb (archive, 2010), depicted in Figure 5.5.

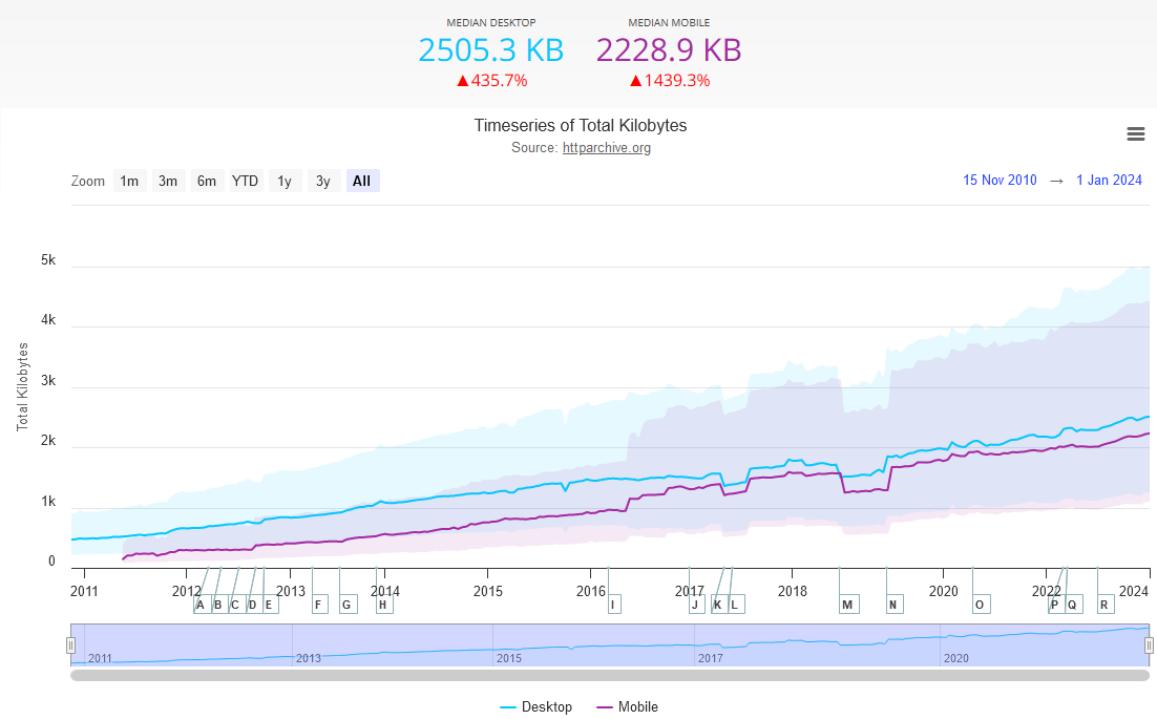

Figure 5.5: Rise in page weight (measured in KB) from 2010 to 2024 as reported by the HTTP archive (archive, 2010). This is attributed to a number of factors including the number of images used, JS elements and externally sourced elements beyond simple HTML.

The requested file is changed to not only reflect this difference between header requests and typical webpage sizes but also increase the size of the overall volume of traffic, increasing the duration of flows and the overall performance load. Additionally, as this is a kernel-level service with many optimisations to its functionality, the number of supported connections is vastly increased, and thus the volume and size of traffic are increased, reflecting the difference in performance between the tools used. Figure 5.6 presents the performance of the web workload for the four experimental scenarios and for a varying number of parallel connections.

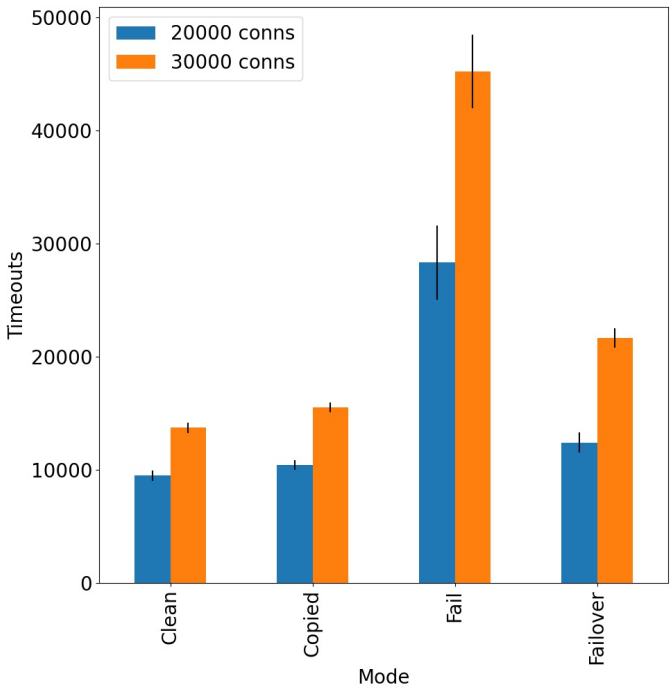

Figure 5.6: Reported timeout rates of WRK connections for the 20,000 and 30,000 connection datasets with iptable logging

Does the log interpretation reduce timeout rates? From the results, it is evident that there is a continued trend of timeout reduction with an observable drop of 56% and 60% in 20–30K connections, respectively. This is significantly improved versus the direct driver, with more pronounced results with a rise in traffic volume.

Impact of externally interpreted state on recovery The separation of the driver from the middlebox incurs no reduction in effectiveness and reduces the possible impact to performance and operations significantly.

To summarise, the log-interpreting driver offers a significant reduction in the loss of state despite interpreting its information externally through a targeted method. Furthermore, there is no clear degradation of its ability to perform its role as a result of differing performance levels between that of the primary and auxiliary middlebox, with the Ryu instance utilised as a partner technology to the logging interpreter for its flow table instantiations. This approach may require some measure of configuration and awareness of logging practices for existing middlebox hardware, but it does offer a non-modifying and performant means by which state may be retained using a Middlebox Minion. The results of the first driver approach, while an improvement, are presented

	Clean		Fail		Failover	
delay	requests	Timeouts	requests	Timeouts	requests	Timeouts
0	120,090	1177	140,788	7144	93,805	4278
0.1	120,115	1279	141,707	7113	106,995	4894
0.25	119,846	1264	145,699	7550	107,192	4804
0.5	119,594	1385	140,014	7660	106,142	4895

Table 5.3: Request throughput and total TCP failures for varying state synchronisation intervals. Frequent state synchronisation improves the overall resilience of the service.

more as an initial exploration of the topic rather than a viable approach.

5.2.5 State Synchronization Frequency

For these evaluation scenarios, the distribution of state to the middleware datastore is live, effectively updating as quickly as log output is interpreted and ensuring state is as consistent as possible across all middleboxes. This approach is motivated by the desire to ensure state liveness (up-to-date state) and reduce disruption from unexpected behaviour over restarting flows. It is not uncommon for backup systems such as this to batch state updates rather than live, however. To return to the CAP theorem, the consistency of state is sacrificed in favour of availability through an "eventually consistent" approach. A state distribution algorithm that guarantees a transaction, such as a two-phase commit, ensures the consistency of the state view across the primary and redundancy, but this is costly on processing. Past work in this domain has batched packets (Rajagopalan, Williams, and Jamjoom, 2013, Rajagopalan, Williams, Jamjoom, and Warfield, 2013) and other forms of state (Sherry, Gao, et al., 2015, Dunlap, King, et al., 2002) as discussed in section 2.5. It is worth examining whether small-scale batching would benefit the system by reducing the overall time the processor is interrupted from packet processing to update the flow tables without impairing the effectiveness of recovery. The more frequently state is synchronised between multiple instances, the greater the presumed correctness of state. Delays may hamper how quickly failure is remediated. To examine the impact of this, we replayed the WEB traffic experiment with varying levels of delay: 0.1, 0.25 and 0.5 seconds. Greater levels of delay than this are unlikely to be seen in real-world networks where even milliseconds of delay can be problematic. Each of these levels of delay are examined through the WEB workload experiment using 2000 clients via the Ryu to Ryu whitebox experiment.

Effect of batching on timeout rates The results of these experiments are depicted in Table 5.3, with a control group with no failovers in the first column. Delays through batching flows do incur an observable impact on timeout rates and the resulting increase in repeated traffic, as shown by the slight rise in the number of total requests made. These rises in request rates and timeouts amortises after the initial batching delay, however. Further delays beyond those displayed in the table show greater rates of failure, but they are beyond the scope of relevancy. The rate of flow instantiation is not sufficient to overwhelm the queue within the time frame tested, suggesting that this window of effect will differ between use cases and traffic load. The necessity of batching is also dependent on the system, such as its effect on CPU utilisation and the scale of traffic, but at least within this use case, it only incurs a minor diminishment in overall performance if one requires it.

5.2.6 Impact of middleware choice

The choice of middleware is an area of experimentation that also bears exploration. The choice of data store and middleground is up to the end user, with considerations to be made over its interaction with the state extraction drivers, ability to disseminate what it receives, and the potential latency incurred. The data store used in the experiments for the MiMi evaluation is Redis (Sanfilippo, 2009), an open-source datastore used for a variety of functions, including caching, key-value storage and message distribution. While Redis does feature a publish/subscribe mechanism, used for the distribution across all potential replicas in MiMi's evaluation, this is built atop its primary focus as a database and caching mechanism.

Other tools exist that are built for the distribution of state. Kafka (Sax, 2018) is a distributed event streaming platform that operates using a publish/subscribe model at its base, built for the purpose of continuously capturing, storing and distributing 'events' within a system as it is operating. Utilising a cluster implementation of Kafka in place of Redis, the WRK test was repeated with 2000 clients so as to observe any potential overhead changes.

Difference in latency between Kafka and Redis The results depicted in Figure 5.7 presents little observable change that does not fall outside of reasonable deviation of results from experiment to experiment. One change, the reduction in failover timeouts, may present that overhead differences between Kafka and Redis, whilst small, may still observe a beneficial reduction in latency for distributing state. The results suggest the choice of middleground technology should be taken into consideration, but the deviations between each technology may only be minor, at least within the scale of this testbed's use case.

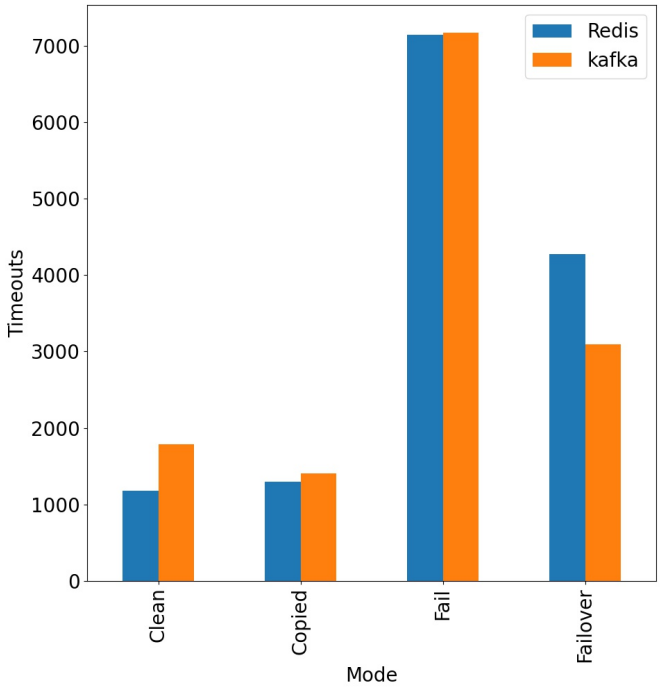

Figure 5.7: Comparison between timeout results

5.3 Katoptron performance evaluation

This section evaluates the proposed Katoptron architecture. The analysis uses network emulation to run a small-scale network topology equipped with the Katoptron service. We evaluate two aspects of the Katoptron architecture: the ability to support a wide range of middlebox types and the improvement of the service on application resilience during middlebox failures.

5.3.1 Experimental Setup

To evaluate Katoptron, a new testbed was created using the MiniNet emulation platform to emulate a topology using OpenVswitch switch instances. Depicted in Figure 5.8, it consists of a number of client and server end-hosts, each running on a distinct subnet and connected to an ingress and egress switch/router. The switches are configured using static OpenFlow rules to connect end-hosts with the Katoptron service as well as route traffic between the client and the server subnets. The Katoptron service consists of a primary and backup middlebox instance and a Click-based Katoptron filter. Both middlebox instances connect via the Katoptron filter to the ingress switch via dedicated links, while the output traffic of the middleboxes is

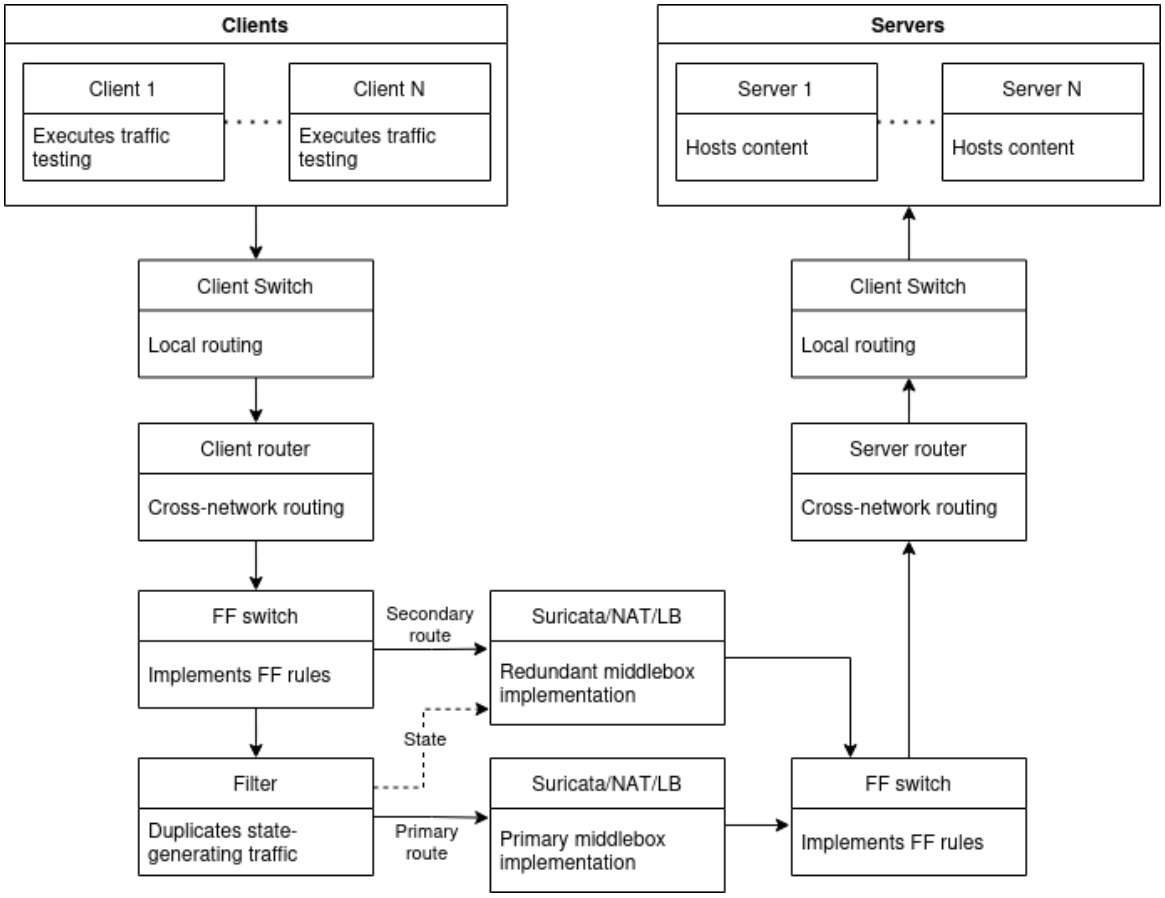

Figure 5.8: The Katoptron testing topology, consisting of a client/server model separated into subnets, with traffic passing through the gateway to the backend of servers.

forwarded to the egress switch, which connects the two middleboxes with the server end-hosts. The ingress and egress switches use static OpenFlow rules that route traffic between the client and server subnets and perform the learning operation required for service restoration during failures. During operation, traffic between the client and the server traverses the different subnets and the middlebox, while a small subset of traffic is duplicated and redirected to the redundant middlebox, whose links are disabled during failure-free operations to minimise the risk of packet duplication on end-hosts. The testbed and the evaluation were run on a Dell server (2x Intel Xeon 4114, 32gb RAM) running Ubuntu 20.04. The experiments use three distinct network traffic workloads, previously summarised in Section 5.1.3 and Table 5.2. The first two workloads, WEB and streaming, are similar but not identical to those discussed in the prior Section 5.1. The *WEB* workload generates HTTP traffic between the

Function	State	Functionalities	Min. sampling rate
NAT	hashmap	Map address space	1
IDS	thresholds	Detect attacks	5
LB	weighted buckets	Distr. new conns	1

Table 5.4: Minimum number of packets per flow needed for state determined through simple experimentation, observing for rises in failure rates for the experiment KPIs

client and the servers for evaluation of short-lived HTTP traffic, as would be typically produced by an Internet web server using different payload sizes. The *streaming* workload emulates a varying number of MPEG-DASH streams between the client and the server end-points for evaluation of long-lived HTTP traffic (video content) that forms another major chunk of typical Internet traffic. Finally, the *attack* workload replays traffic from the CICID 2017 evaluation dataset using TCPReplay. This final body of traffic is evaluated against real-world traffic that has been employed for the purposes of testing IDS detection effectiveness in prior research by others, so as to ensure the evaluation of Katoptron's ability is thoroughly evaluated.

Finally, in all experiments, we consider three experimental configurations: *Base*, executes the experiment with no failures, *Simple*, executes the experiment with failures and uses simple 1:1 middlebox redundancy, and *Katoptron*, executes the experiment with failures and uses the Katoptron architecture to improve network resilience. The first setup is used to demonstrate the performance of the application during normal operation; the second setup demonstrates the limitation of simple 1:1 redundancy; and the third configuration is used to demonstrate the improvement achieved with our Katoptron architecture.

5.3.2 Middlebox support

In order to demonstrate the generality of the Katoptron architecture, we discuss in this section the set of middlebox appliances that we provide out-of-the box support for our Katoptron strawman implementation. Our set of compatible middleboxes consists of three off-the-shelf, unmodified network functions: a NAT, an IDS and a load balancer. The selected functions cover a wide range of network operations, including packet field modifications, stateful packet forwarding and flow monitoring. Without loss of generality, the middlebox implementations we employed in our evaluation are software-based, and we use them as black-box devices, *i.e.*VNF instances are unmodified, and their implementation closely matches the behaviour of a hardware-accelerated device. Where possible, we have utilised open source tools to assist our aim of evaluating the generality and avoid overfitting to specific operational

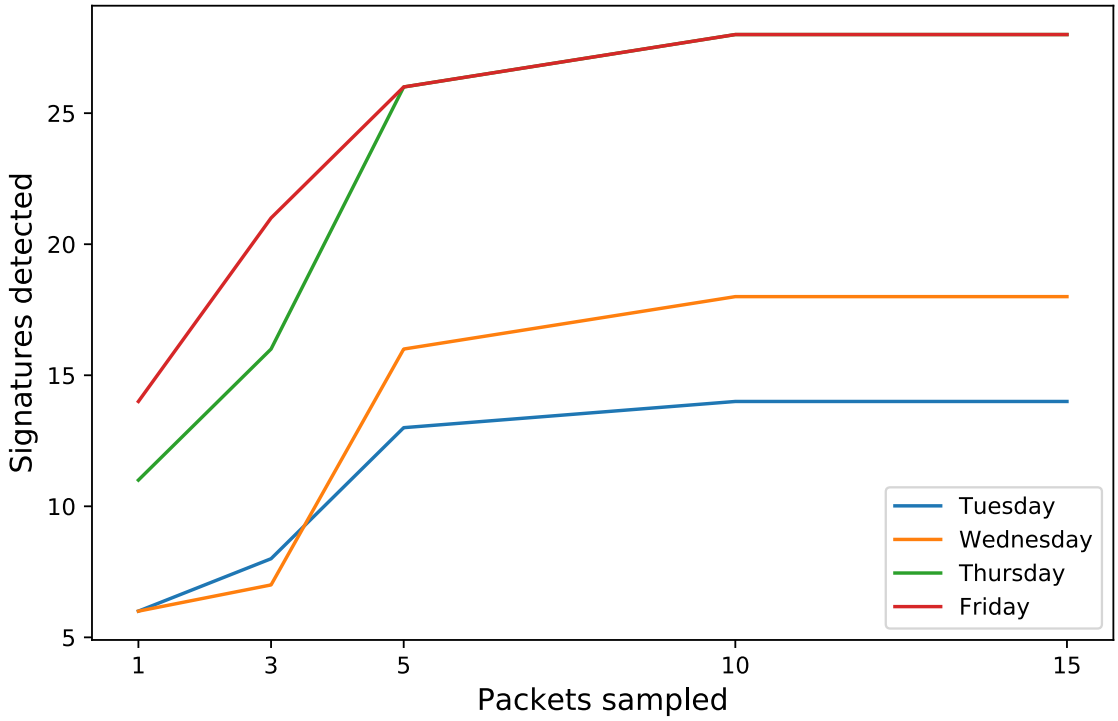

Figure 5.9: Evaluation of the impact of traffic sampling policies on the total number of signatures detected by the IDS middlebox (Suricata) and diminishing returns of greater sample sizes for the CICID 2017 datasets.

models, such as the Suricata IDS (O. I. S. Foundation, 2022). Furthermore, the filter implementation does not depend on a specific technology and could be realised using several packet processing technologies (*e.g.* the P4 language, DPDK program). Overall, our experimentation and approach showcases that Katoptron can both target many production middlebox targets and cater to physical or virtual redundancies, as well as realised in multiple technologies as demonstrated.

Our NAT middlebox uses the high-performance Click modular router and its built-in mazu-nat application. The function uses a lookup table with a 5-tuple hash (IP Proto, IP address and port source and destination) to connect the client hosts to the server network via a single IP. Furthermore, the NAT operates as a gateway for all incoming and outgoing traffic between the client and server hosts. The Click application implements a full-cone NAT and utilises a consistent hashing mechanism common across similar NAT implementations. The function state includes the entries of the lookup table and possible loss during a failure incurs processing overhead in the recalculation of the lookup state for every established connection. Our IDS function

uses the open-source Suricata software, equipped with the open emerging threats ruleset (Inc, 2021). The IDS instance monitors the traffic between the client and the server network, operating in IPS mode (the IDS receives a copy of the active traffic) and its configuration will log alerts when a rule matches a flow of packets. State is defined in this scenario as these threshold counters; if counters are lost for active connections in flight, it would be possible for false negatives to occur when thresholds are not met for each separate instance's view of the live connection, allowing malicious traffic to reach the target. Typically, emerging rules match against flow and host statistics or apply wildcard masks on the start of the application payload.

Finally, the load balance function uses the Open VSwitch switch with a fixed set of OpenFlow rules. The function maps incoming flows to a random backend server using a weighted bucket entry, which distributes traffic evenly between a fixed set of servers using a consistent hash function on the 5-tuple of incoming packets. Finally, we develop a load balancer middlebox application using an Open vSwitch configured with a static set of OpenFlow rules that spread user requests across the servers, using a simple weighted bucket approach to distribute load in a broadly even distribution via consistent hashing of the five-tuple of incoming packets.

Table 5.4 summarises the three integrated middleboxes and reports the minimum number of packets required in order to ensure sufficient state recreation between a primary and a backup instance. The sampling rates were determined through initial testing of each scenario to narrow down the minimum viable state. For the NAT and LB, only the initial SYN packets are sufficient to prime state on the backup server, with the majority of their logic beginning at connection establishment. For the IDS middlebox, we ran different traffic sampling scenarios using the attack workload and concluded that the middlebox requires a minimum of five packets to fingerprint a flow accurately. Fewer packets impair its success rate, with greater offering no additional benefit once signature rates are matched, as shown in Figure 5.9.

5.3.3 NAT middlebox performance

The first scenario we explored is the NAT middlebox, where the loss of state results in the reset of active connections between clients and servers. The redundant NAT constructs its state by duplicating SYN packets from all incoming connections to the primary box. In this experiment, we use the WEB workload and vary the number of parallel HTTP connections. We run our experiment for 30 seconds and trigger a link failure halfway through the experiment. We run each experimental setup five times and report in Table 5.5 the average timeout and reset rate when running the WEB workload for two different content sizes (5KB and 627KB) and for a varying number of clients. Furthermore, we run experiments using both the streaming and the web workloads. The values displayed in Table 5.5 consist of TCP RSTs generated from

WEB workload - 5KB object

no. clients	Base		Simple		Katoptron	
	Reset	Timeout	Reset	Timeout	Reset	Timeout
50	0	0	157	159.2	126.6	98.4
100	0	0	128.8	124.8	191.8	56.2
200	0	0	344.2	359	158.2	114.2
300	1.8	12.2	503.2	492.4	256.8	252

WEB workload - 627KB content

no. clients	Base		Simple		Katoptron	
	Reset	Timeout	Reset	Timeout	Reset	Timeout
50	0	7	29.8	175.4	0.2	17.2
75	0	63.8	1.4	237.8	1.4	155.96

Table 5.5: Average HTTP resets and timeout rates during NAT middlebox failures using the WEB workload for both small (5KB) and large (627KB) objects served.

packets hitting the closed port and timeouts triggered by state loss. A rise of either, but especially timeouts, is indicative of traffic dying on the wire, with its reduction demonstrating a visible improvement in minimising observable failures for the end user.

Does Katoptron reduce short-lived state failures? At first glance, there is a clear reduction in timeouts and resets across every test. This becomes more pronounced for experiments with higher traffic rates, as showcased with the 300 clients set with an almost 50% reduction in failure rates of short-lived connections. While short connections have far less state to lose, the results demonstrate a marked improvement over simple redundancy.

Effect on overall traffic rates Inaccurately copied state could potentially cause a rise in the volume of overall traffic, as well as restarting connections from their general loss. However, there is only a minimal observed rise in request rates in each scenario (e.g. 326K requests for filtered traffic to 315K unfiltered redundancy for 300 clients, or 3.5%). This is a relatively minor rise versus typical traffic patterns and is well within tolerable limits for bandwidth use.

Lighter vs heavier traffic Short flows, like those shown in the top half of Table 5.5 using the 5KB object, are short-lived and far easier to restore when their connections are disrupted with or without Katoptron, as their loss and repeated traffic incurs far less bandwidth use versus larger flows. For the 5KB HTTP requests, there

	Base		Simple		Katoptron	
Experiment	sigs	alerts	sigs	alerts	sigs	alerts
Tuesday	14	7420	9	5073	14	6619
Wednesday	18	1106	15	1366	18	1347
Thursday	28	1434	22	1153	28	1460
Friday	28	1381	12	671	28	1235

Table 5.6: Total number of signatures and alerts raised by the redundant Suricata IDS instance for each trace, when using the Attack workload. Base represents the expected detected outcome of the IDS to the attack workload. Simple represents the results when experiencing loss of IDS and failover without state preservation. Katoptron represents the results when experiencing the same with state preservation techniques.

is a pronounced improvement on both connection resets and timeouts. For longer flows, such as the larger 627KB request, the window of potential disruption is longer. The bottom half of the table, using the 627KB object, shows a similar and clear improvement in minimising timeouts for these longer and more vulnerable flows, with a significant drop in the number of timeouts between failures with katroptron and without (simple). Overall, while short connections have far less state to lose, the results of both the lighter and heavier traffic tests showcase that even for easily restarted traffic such as simple web requests, our non-modifying improvement to blackbox redundancy has an observable benefit in reducing disruption to connections during failover.

5.3.4 IDS middlebox performance

The second scenario we explored is the IDS middlebox, where the loss of state influences the number of attacks detected by the IDS since the system will have a limited view of the active traffic. For this experiment, we use the Suricata IDS both for the primary and secondary middlebox and execute the attack workload. During each experimental run, we trigger five failures, with a 15-second gap between them. Furthermore, we run each experiment five times and report the average signature and alert results in Table 5.6. These indicate the detection of attack signatures (sigs) and how often they are observed (alerts) and are directly compared to the control/"base" set; observing the same number of signatures indicates no degradation of the ability to detect attacks.

Does the filter affect signature detection? The results show a clear continuation of detected attacks even with only 1.5 to 5% of the body of traffic and repeated

failovers disrupting detection versus non-state-preserving redundancy. These results do not significantly improve with increasing the number of packets sampled beyond five packets. When reducing the number of packets below this determined value, however, there is a marked rise in false negatives and mismatched signature detection.

short-term and long-term attacks The Tuesday and Thursday traces utilise short-term rapid attacks, necessitating repeated and aggressive failures to meet the short duration of each type of attack. As showcased by the "Simple" category in Table 5.6, is unable to detect the majority of these short-term attacks due to this difficulty of timing, but the redundant device primed with state via the filter detects all potential malicious traffic. Attacks that occurred over more significant periods of time, including all botnet and distributed attacks, show a far greater loss when this state is not preserved and likely pose a greater risk and danger to networks in real-world scenarios if they were to go unchecked.

5.3.5 Load balancer middlebox performance

The third scenario we explored is the Load Balancer middlebox,where the loss of state influences the number of connection resets per client, with the loss of state during failover forcing them to be re-established. This approach is akin to the NAT deployment, but with a far greater dependence on state due to the weights used in the decision process. This experimental configuration uses the service restoration mechanism, configured using static rules on the ingress and egress switches. It is aided further by a secondary OVS rule to the failover and a masking switch on the server side of the failover mechanism that keeps track of existing flows so that when traffic is returned to the primary path, existing connections that have already been established by the auxiliary will return through the primary rather than being dropped. The filter is formed in the same SYN configuration as the NAT experiment as well as the body of experimentation for both traffic and failover tests. The state for this scenario consists of the hash lookup tables, the loss of which is improbable to recreate outside its original moment of generation due to the weighted consideration of the current table contents. During this experiment, we fluctuate the link state to the primary middlebox every ten seconds, which triggers the Katoptron service to switch between the primary and secondary middlebox instances. As discussed in Section 5.3.1, the GST streams consist of 375 active connections divided across five clients for a total duration of 120 seconds, saturating the traffic load the testbed is able to handle. The metrics presented in Table 5.8 showcase resolution changes where the DASH format shifts its streaming bitrate to accommodate for perceived bad connectivity, as well as buffer periods and broken streams.

Test	reset	timeout
Base	26.68 ±8.37	1.92 ±0.796
Simple	324.2 ±6.172	240 ±7.886
Katoptron	50.32 ±6.161	2.2 ±0.744

Table 5.7: Results of the LB failover using the 627KB object. The number of reported TCP resets and timeout responses are shown in their respective columns, averaged from multiple runs. Results indicate a drop in rests and timeouts significantly when preserving state with katoptron over not with simple.

Test	buffer events	res changes	failed conns
Base	1	2	0
Simple	8.75	19 ±0.693	1
Katoptron	1	2	0

Table 5.8: Total count of buffering, resolution change and failed connections during Load Balancer middlebox failures with the streaming workload.

WEB workload - Small objects For the short-term streams, the difference between simple redundancy and the Katoptron is more service-pronounced than NAT, due to the weighted bucket approach establishing a small measure of non-deterministic execution. This magnifies the effects of the failure of state transfer. Table 5.7 reflects this with the significant reduction in timeouts and TCP resets between simple and Katoptron, from 324 to 50 and 240 to 2.2 timeouts respectively. With short-lived traffic, this is not a significant load on a load balancer in most use-cases, but our live capture of state ensures that even short-lived failures are minimised. This shows that the decision-making logic of this bucket is consistent to the state at that moment in time, serving as a demonstration for how Katoptron can facilitate even slightly non-deterministic decision logic that might be present in blackboxes alongside more common hash-based methods.

Effect on video streams for buffer events Table 5.8 reports the performance results of the streaming workload when processed by the load balancer middlebox. With five active streams split between the topology servers, there were no changes in both buffer events and resolution changes, despite 20 failovers randomised across the ten-minute duration of the experiment. The simple redundancy mechanism results in multiple resolution changes during each failure, with each flow forced to re-initiate connection at each failover and one forced to timeout completely. The simple redundancy/non-Katoptron instance also showed a significant increase in both the total playtime and the mean buffering time due to the persistent loss and forced

delay to re-establishing connections at each lost chunk, with a total playtime of 1047 seconds versus the expected 595.

5.4 Summary

In this chapter, the resilience framework and its mechanisms have been evaluated. Each of these evaluations aimed to test their effectiveness at transferring state, their impact on the overall system operations, including incurred latency and delay, the accuracy of their recovered state, generic applicability for both construction and their ability to target all possible middleboxes. The Key Performance Indicators (KPI) of the evaluation varies between tests and shall be broken down with the results. It is split into two sections for the two sets of contributions, MiMi and Katoptron, from Sections 5.2 and 5.3, respectively. These sections have examined the effectiveness of the white, grey and blackbox state mechanisms created and how well they meet these KPIs.

In the first section, presented in Section 5.2, the contributions of MiMi are demonstrated. As a broad summary, MiMi explores the scope of the problem of building resilient systems for blackbox middleboxes. The mechanisms developed target white and greybox scenarios using direct extraction with open-source software and interpretation of logging information respectively. The whitebox mechanism was evaluated through the direct extraction of flow table instantiation messages from a Ryu-based load balancer to a Redis datastore, acting as the middleground. The greybox mechanism was evaluated through an external interpreter acting as a syslog server receiving the logging output of an IPtables-based load balancer and using this to create OF flow table instantiation messages for a Ryu-based redundant load balancer. Furthermore, the choice of middleground and its method of distributing state was examined for its potential impact on failover, as well as the batching of state distribution.

The KPIs of the WEB workload, examining short-term flows, consist of the timeout rate of connections and increases in traffic volume. Other performance indicators include read errors, latency and request rates, as reported by WRK. The KPI of the video workload, examining long-term flows, consists of the failed connections from timeouts as reported by Scootplayer. Overall, there is an observable improvement in connection retention for both long and short flows. For the whitebox evaluation using the Ryu load balancer, there is a reduction in timeout rates for the web workload (short-term flows) of up to 40% with an increase in traffic volume of up to 17%, with no observable change in latency. The video workload (long-term flows) shows a smaller improvement, with the reduction of timeouts up to 18%. For the greybox evaluation using IPTables, the WEB workload reports a much more observable drop of 56-60% in timeout rates for 20-30K connections respectively. The

state synchronisation evaluation repeats the WEB workload of the greybox scenario, monitoring for increases in request rates and timeouts as the KPI. Results indicate a minor increase in timeout rates of 14% with a delay of 100ms, with further delays amortising with no increase in timeout rates from larger delays. Finally, the middleground impact evaluation observed for increases in timeouts from inferred latency between the two technologies, Redis and Kafka, with the results suggesting the difference in latency between technologies is negligible, at least for the testbed used. Middleboxes present a difficult goal for maintaining state across failures, but NFV offers considerable potential in its adaptability, especially when played in an auxiliary role where its comparative downsides are diminished.

Overall, the results for MiMi show an observable improvement in connection retention for both long and short flows, with state reconstructed from logging serving as an effective first step in more generic approaches to state preservation. This half of the work presented in this thesis also explores the necessary level of separation and technology required to allow NFV to work in this secondary role to a middlebox and ensure its operations are undisturbed, as well as accommodate for potential scalability and expansion.

In the second section, presented in Section 5.3, the contributions of Katoptron are demonstrated. The larger body of work for this thesis, it evaluates the effectiveness of the blackbox state mechanisms created. Katoptron establishes a targeted packet filtering approach to recreate state in a redundant middlebox by replicating sufficient traffic to instantiate flow table entries. The evaluation is divided across three different middlebox use cases, each built within a different technology. The first is a NAT built within Click evaluated against the WEB workload using two sizes of payload. Its KPIs differ slightly by reporting on both the timeout rates and reset connections. Results indicate a significant reduction in timeouts for the 5KB payloads with an up to 68% reduction in timeouts, and a 34% reduction for the larger 627KB payload. The NAT scenario also indicated an increase in traffic volume of up to 3.5% from disrupted connections attempting to re-establish.

The second scenario was an IDS using Suricata and the emerging threats signature set, evaluated using the CICID 2017 attack dataset. This evaluation was divided into four sets for the four days of PCAPs within the set, each focusing on a different set of attacks. For example, "tuesday" uses short-term DDOS connections in high volume, while other days possess heartbleed or botnet attacks over long-term flows. The KPIs for the IDS consist of signature detections where malicious traffic was observed accurately by the IDS and the number of alerts triggered per signature. Of the two, the signature is far more important as an indicator of the accuracy of the IDS's ability to detect and detect accurately. Results show no disruption to signature detection in the redundant middlebox using only 1.5% of the traffic. There is a reduction in the alert rate; however, this is to be expected with repeated triggered failovers and

does not disrupt the IDS' performance. The volume of traffic necessary to ensure the maximum detection offered diminishing returns beyond five packets and no further improvement beyond ten per-flow for this evaluation.

Finally, the third scenario consisted of a load balancer built within OVS, distributing traffic between multiple backend servers. This is evaluated using the WEB workload with the 627KB payload for short-term flows and a gstreamer parser, configured to act as a video player for an MPEG-DASH short film. The short-term flows differ slightly from the prior experimentation due to the presence of a weighted bucket distribution, which introduces some measure of non-determinism into the load distribution. Results indicate a continued reduction in timeout and reset rates, with a reduction of resets up to 84% and timeouts almost eliminated at 99%. The KPIs of the DASH stream consist of reported buffer events, resolution changes from detection of disruption to buffering from a slow or poor quality connection, and the number of failed connections. Results indicate that for these long-term video streams, no disruption could be detected by the player for middleboxes pre-populated via Katoptron, compared to an average of 9 buffer events and 19 resolution changes for simple redundancy. Furthermore, the persistent loss of state and re-establishing of connections incur a significant delay in the playtime of the video file, from 595 seconds of the expected duration to 1047 seconds.

Overall, the results show a significant improvement in retaining state across instances even with a radically reduced volume of traffic to build state from the targeted replicas, achieving three major goals: to provide an effective and accurate failover using only minimal data without modification to the target device. The total volume of traffic needed to regenerate traffic varies between targets, with a measured reduction of volume needed for only SYN packets totaling 98.5%. For state derived from observing entire flows, the first five packets of each flow total 4.8% of traffic, or a reduction of 95.2% of the total volume. This radical reduction in overall traffic needed to maintain a hot replica is a strong contribution to Katoptron's viability in real-world systems where diverse systems may require significant transit between nodes, even paired redundancies. The accuracy of this state is also demonstrated clearly in the results displayed across the three target middleboxes. The lack of modification necessary to ensure this degree of effective stateful failover is a significant step beyond existing work within this domain, providing an elegant and easily deployable drop-in solution. This is further emphasised by its demonstration in multiple technologies and its design, which is sufficiently high-level as not to be tied to any particular technology or language feature.

In summary, this evaluation demonstrates the feasibility of replicating state across blackbox hardware and software without modification efficiently and effectively. The aim of this thesis was to create a generic, all-purpose approach to establish persistent failover between network functions for both software and hardware. The goals of this

thesis have been met and presented within this section, including:

- **Ability to support both software and hardware**
  Both Katoptron and MiMi present use cases for targeting blackboxes in both software and hardware effectively, with proven results to showcase their ability to extract and distribute state to both duplicate devices and differing redundancies.

- **Support for unmodified blackboxes**
  The logging interpretation driver and the packet filter both necessitate no modification to the target device, especially that of the filter, with no reduction in their overall accuracy at the re-established state or issues involving non-determinism.

- **Minimise overhead incurred**
  Per-packet overhead is incurred through checkpointing and logging mechanisms that interfere with the flow of traffic. To minimise the impact of this approach, the body of work for this thesis has focused on non-checkpointing mechanisms and live replication as well as indirect state sources from the packet flow to prevent interference.

- **Guarantee sufficient correctness of recovery**
  The methods explored create an approximation of the original middlebox's state rather than a perfect copy as prior research has pursued, either through log interpretation or traffic filtering. Inaccurately replicated state on the redundant middlebox could impair recovery to a greater degree than its loss. However, the evaluation of both MiMi and Katoptron indicate both reduced timeout rates and more successfully transitioned connections, especially demonstrated in section 5.3.4 and the signature detection for the Suricata IDS. This suggests that the state created is sufficiently correct as to improve failover despite its "lazy failover" approach.

- **Technology-agnostic**
  To be adoptable to real-world networks and deployments, the systems developed must be both applicable to multiple kinds of middleboxes, especially those still implemented in hardware, and able to be created in a variety of technologies. Both of these requirements have been met across the two bodies of work, their designs sufficiently high level so as not to require any mechanism specific to one technology.

Chapter 6

Conclusions

The Internet today forms the lynchpin of modern communication systems, intersecting with all areas of modern society. With its integration in all aspects of modern society, from business and enterprise networks to critical support infrastructure, its resilience against disruption and failure has become a critical aspect of its design and propagation (Cisco, 2018). The over-reliance on proprietary hardware and software to deliver network features, typically in the name of performance or security, has created a vulnerability in the resilience of the infrastructure. The inability to preserve state through even common techniques such as redundancy greatly inhibits recovery efforts in minimising the visibility of failure to its users (Sherry and Ratnasamy, 2012). Their use to fulfil performance requirements for both service and contractual obligations has made them further ingrained, making this problem difficult to overcome; middleboxes have become too necessary and widespread as to be easily replaced as they were originally intended (S. Huang et al., 2017).

This thesis has tackled these challenges and presents a solution that consists of a middlebox resilience framework capable of preserving state in an extensible and distributable fashion without modification or replacement of the original hardware. This is formed of Remediate that offers two mechanisms (Hill, Rotsos, Fantom, et al., 2022, Hill, Rotsos, Edwards, et al., 2024), each targeting greybox software or blackbox hardware, alongside a scalable distribution approach to multiple replicas and an efficient high-level state filtering mechanism. This thesis has demonstrated its viability and general application for use across different technologies and common network functions with no observed degradation in effectiveness for both long- and short-term flows. Together, these two systems allow for the retention of state for all possible configurations of target devices and their redundancies (e.g., 1:N, 1:1, software to hardware, hardware to software etc.). Networking infrastructure is made with the understanding that failure is an inevitable part of its operation at every level, from intermittent packet loss to significant hardware faults (J. P. G. Sterbenz et al.,

2010) and must be built with a mindset focused upon resilience and continuation of service rather than high-grain performance. Katoptron has been built with this mentality in mind, using a low-cost and non-complex means of retaining state via filtering traffic across live blackboxes to enhance the resilience of critical areas of failure without impairing its normal operations. This filtering approach is especially powerful, enabling blackbox hardware to remain in place and incurring a minimal impact on operations versus existing research, allowing for efficient live replicas to operate with up-to-date state without the delay and overhead incurred by replaying approaches.

The evaluation of Katoptron further validates the argument that state requires minimal information from the target blackbox to ensure continuation of service during failovers, allowing it to minimise both its invasiveness through the need for modification or replacement and its impact on failure-free operations. In summary, MiMi and Katoptron improve upon the existing work within the domain of resilience for middleboxes, advancing beyond current practices in a new direction. Katoptron has been shown to effectively replicate state to provide a reasonable degree of resilience against failures with only 1.5% of the traffic, removing any need for incurred delay, modification of the target or advocating for its replacement. Furthermore, its backwards compatibility and scalability with MiMi allow for a great deal of flexibility for deploying in real-world networks with minimal modification. As middlebox usage continues to grow and change within real-world networks, the approaches to improving their resilience will also change, with future work expanding upon the capabilities of Katoptron and the contributions of this thesis to allow it to target any and all possible middleboxes, networks, and use cases.

6.1 Thesis Contributions

It is worth repeating the research questions first posed by the introduction now, to examine how the thesis and its contributions have addressed them:

- What are the limitations of existing approaches to enabling greater resilience for middleboxes, especially in regards to hardware and blackbox devices, and why are they not adopted outside of research?
- Can state be created externally without observation of the interior operations of a middlebox that is both sufficiently timely and accurate as to provide effective failover for a middlebox, regardless of the level of observability or technology it is implemented in?
- How can a proposed solution to the problem of state preservation be implemented into existing infrastructure without the requirement of replacing

existing infrastructure or disrupting the behaviour of normal operations outside of network failure in a quick and reliable fashion?

The first research question is to explore the domain of middlebox resilience in research and establish its limitations, especially in regards to adoption and use. The second and third questions build upon the now-established limitations of research, especially in regards to its lack of use. These limitations are namely the need to replace infrastructure and the loss of performance this might cause. This thesis aims to expand upon the challenge of enabling resilience for grey and blackboxes, a subset of middlebox technology typically deployed in hardware whose inner workings are unknown to the external user. This prevents the direct programmability of internal decision-making or memory, as well as the interpretation and recreation of the decision-making logic of these middleboxes. These challenges are highlighted by the existing work in this domain, focusing instead on modification or replacing blackboxes with accessible hardware or virtualisation (Sherry, Gao, et al., 2015, Rajagopalan, Williams, and Jamjoom, 2013, Panda et al., 2016). As a result, the design and implementation of a backwards-compatible and generically applicable resilience framework has been created, which, combined with a highly scalable approach to distributing state, has pursued a new direction that has improved upon the areas that otherwise limit the deployability of past research in real-world networks. This thesis has contributed the following:

- Described the history of how middleboxes first evolved and the scope of the problem in modern network infrastructure
- Created a taxonomy of approaches for middlebox resilience and identified gaps in existing research
- Established a novel approach to recreating state across grey and blackboxes by exploiting the awareness of their technical limitations for non-determinism
- Design for a set of state recovery mechanisms that do not require replacement or modification of existing middleboxes
  - whitebox resilience mechanism that extracts flow table instantiation commands from open-source whiteboxes and propagates them to other modifiable redundant whiteboxes
  - greybox resilience mechanism that uses log interpretation and external serialisers to establish equivalent state to be instantiated in redundant whiteboxes

  - blackbox resilience mechanism that uses targeted traffic filtering to recreate deterministic state using only a subset of the overall traffic distributed to redundant blackboxes

- Utilised off-the-shelf technologies to create a scalable distribution for state that also serves as an abstraction layer to obscure awareness of redundancy from primary/target middlebox

- Created proof-of-concept implementations of the resilience mechanisms for the purposes of evaluation

  - A client/server testbed for servicing traffic to pass through middleboxes at the gateways to the backend network
  - An inserted state serialiser to extract OpenFlow flow table instantiation commands
  - A set of log interpreter and state serialiser drivers to interpret greybox output and generate flow table instantiation commands, as well as the different drivers to receive them
  - A set of state filters in multiple NFV technologies, evaluated against multiple common hardware middlebox types

The contributions listed are a significant step towards enabling greater resilience in middleboxes. State preservation and transfer mechanisms are uncommon and complex in hardware, but advocating for their replacement (Sherry, Hasan, et al., 2012) with more accessible hardware is an infeasible position due to their numbers and how heavily relied upon they are. The research in this thesis allows for state preservation and transfer across a scalable number of replicas, removing the concern for their replacement or interference and providing a solution to the challenges that this area of networking faces. Much of the body of existing research on middlebox resilience concerns itself with minimising non-deterministic execution and can be roughly staged by their increasing evasiveness and improved effectiveness as they handle a greater number of potential variables. This presumption of modification is infeasible in real-world deployments, however, and has always served as a barrier to the adoption of NFV in areas concerning middlebox usage (Sherry and Ratnasamy, 2012). Through analysis of the typical operations and limitations of hardware implementations, the work of Katoptron establishes a direct, non-interfering approach to recreating state that exploits the argument that the Internet is inherently fallible and the need to limit non-determinism to this degree is unnecessary, as showcased in its evaluation.

This greatly reduces the barrier to entry and increases the potential for the designs of this thesis and future work like it to be adopted. The key motivation driving

this research concerned the further adoption of NFV in modern networks, with resilience providing a key gap in this domain. Its backwards-compatible approach can be implemented in a number of technologies, as this thesis has demonstrated, with future improvements possible in both hardware offloading and refinement of the filtering approach. The other body of work, MiMi, establishes this concept by first targeting software using an initial example of interpreting logging output, although it can realise many more. The largest contribution is the framework, establishing the means by which a middlebox may be logically separated so that no interaction takes place between the redundancy framework and the target itself. A scalable distribution mechanism leverages third-party datastore tools common to networks, allowing for significant versatility in configuration. The ability to provide redundancy to a variety of possible configurations and a number of replicas in each direction when the two projects are used in conjunction showcases just how flexible the contributions of this research may be. Overall, the work of this thesis covers a wide scope of middlebox use cases, but it is not without its limitations.

6.2 Criticisms and limitations

There are a number of points of criticism that can be raised on the work of REMEDIATE, and are worth highlighting here in the conclusion. These vary from the design, the problem it seeks to solve and its overall applicability, and finally the execution of the evaluation itself.

6.2.1 Feasibility and scope

Firstly, as network functions continue to grow in complexity, so will their operations and the means by which state is created. The proposed solution offers IP-layer resilience and is built with the expectation that hardware middleboxes will continue to operate within the bands of purely TCP traffic. However, this is not guaranteed, and the emergence of protocols such as QUIC pose a problem for the existing infrastructure, as well as for the ability of new protocols to gain traction. If they continue to limit the scope of what is achievable in current networks, the infrastructure will most likely change, and with it greatly diminish the scope and viability of technologies such as REMEDIATE. In its current state, REMEDIATE offers the ability to provide stateful failover for a number of popular mechanisms but by no means all, and only at this TCP level. Network functions that operate at higher layers, such as the HAProxy load balancer, cannot be supported. Hardware middleboxes that rely more extensively on the use of the co-processor and information that cannot be inferred, such as internal clocks, will reduce the scope of potential targets that this research applies to. Beyond its scope, another potential barrier is its deployability.

The execution of REMEDIATE in a real-world context is left flexible and up to the user, which makes evaluating its impact on a system difficult. Latency incurred by bumps in the wire is the greatest potential impact it might have, especially in regards to packet duplication from write operations. The low tolerance for additional latency however would likely restrict deployments to hardware offloaded approaches to minimise the potential risk. Despite these criticisms, REMEDIATE offers a broad solution for some, if not all, current middlebox use cases, and further evolution of the concepts explored in this thesis would likely expand on its effectiveness. Some examples will be discussed below in Section 6.3.

6.2.2 Security and points of failure

There are a number of potential security criticisms and points of failure that warrant further exploration that fell outside of the scope of this thesis. The mechanisms used to extract state from blackboxes have the potential to expose information that may cause risk to network devices, which is an important area that needs to be explored more thoroughly. Observation of extracted logs and duplicated traffic travel through non-encrypted mediums, and their use in secure systems could expose this. Its use within closed networks somewhat minimises this risk, especially as its intended role is to function in a purely redundant system. Instead, a greater risk is posed by its scalability and as a point of failure. One key point of criticism is that REMEDIATE fails to address the scalability issue of redundancy systems raised by past work sufficiently. In these publications, the concern stemmed more towards one-to-one redundancies, which REMEDIATE avoids by allowing for a one-to-many or many-to-many distribution, but does so through the introduction of singular points of failure. The loss of the filter for example, or the disruption of message passing systems used to distribute state would disrupt REMEDIATE's operations significantly, and it is not built in such a way as to handle these failures. This is once more mitigated by the notion that it is the redundant system, and there are limits to the degree that redundancy should be pursued for all entities in the network.

6.2.3 Evaluation

The evaluation of REMEDIATE occurred as its components were produced across the duration of my research, and have a number of weaknesses in how it was executed. Middlebox Minions, or MiMi, was the first publication produced and the earliest work included in this thesis, and its evaluation suffers because of this. The tooling chosen for its evaluation, as well as its testbed, was subpar, and this is reflected in the technical limitations that constrain the earliest results. The number of dropped packets, as well as timeout rates, is far higher than a real-world system should reflect and warranted

rapid adjustment even within that publication to correct. Given motivation, the developments and improvements made on the testbed overall by the time of Katoptron would have been well-placed to correct these issues with MiMi. However, the approach that it employs is decidedly less effective than the filtering, which was surprising in its results.

The overall testbed and use case scenario were discussed in peer reviews for the two publications. Namely, the applicability or scope of this research given the scenario used across all experiments. The client-server model of requesting information, be it web pages, video streams and so on, represents a significant majority of typical traffic on the Internet as a whole, and especially from the perspective of the typical end-user. This model was both simple to implement and wide reaching in its applicability, and allowed for the evaluation of a number of network functions that are typically used. For example, the use of DASH was chosen as it is an adaptive streaming technique, so disruptions in the medium (*i.e.*loss of packets/traffic from failover) directly alter the current bitrate being streamed, which provides a very clear indication of packet loss and congestion detection.

The evaluation was by no means exhaustive however, and represents a selection of the best results encountered during experimentation. For example, one technology that was evaluated but offered little to no benefit were those of proxy caches. A proxy cache retains copies of content as a local repository that are being requested by more than one end-user, and are employed to reduce the overall level of traffic. The cache is populated by the original request made, and is retained for future requests of the same content. To prepopulate a proxy cache, the files themselves must be transferred, but mechanisms such as Katoptron only allow for the initial request to be made again to be cached, which would inevitably happen if the original was lost. Overall, REMEDIATE's evaluation is limited both by the technology available and time, but the publications provide clear evidence that the testbed and use cases were adjusted progressively to correct the faults that could be resolved and eliminate disruptions to the results.

6.3 Future work

With the growing use of NFV in real-world networks, there is interest in further expanding the potential roles in networking that may be replaced or augmented by its use. The research conducted has the capacity to be expanded upon in several directions, chiefly in regards to service function chains, performance, management, and other types or means of state reconstruction. These will be summarised below:

6.3.1 Expanding awareness to Service Function Chains

Remediate targets individual middleboxes for the purposes of failover, for both greyboxes with MiMi and blackboxes with Katoptron, further reinforced by their need to be fitted to a minor degree to their targets. Modern networks typically chain network functions together to form the gateway to networks or as a packet processing pipeline, known as Service Function Chains (SFC). Prior research regarding SFCs (Khalid et al., 2019) has explored the recreation of state and the complexities involved, with discussion on how state is transformed by the additional complexity of each stage of the pipeline versus just a single box. Furthermore, criticism from past work in this domain is that ad-hoc and individual solutions, like those proposed in this thesis, are not scalable for enterprise networks due to the volume of middleboxes involved (Kulkarni et al., 2020). It would be worth exploring Remediate's effectiveness when expanded to target entire chains at once, especially in regards to its feasibility and the impact it might incur. For example, filters from Katoptron could be inserted in-between each NF within the chain rather than simply at the start, or the decisions made by multiple chained greyboxes may require more complex logic to properly interpret and replicate in software rather than treat the boxes as purely independent elements. In summary, SFCs represent one area of increased complexity that future research should explore to examine its effectiveness and its need to adapt to more complex environments and networks.

6.3.2 NFV management, scaling and integration

As NFV adoption grows, the potential to incorporate virtualised techniques for remediation and recovery grows. The initialisation and utilisation of VNFs in this design and experimentation during the proof-of-concept evaluations were established manually, created on a per-box basis, with some experimentation given towards scaling replicas for both performance loss mitigation and non-deterministic testing. The integration of Remediate into VNF platforms would be an appropriate step towards its further use. The development of either APIs, integrations or adoptions of one specific technology is one such course of action worth exploring, such as via Kubernetes using labels to identify the pod elements. This could be taken further from virtualisation deployment methods to larger frameworks. One such example is the Open Digital Architecture ODA model, or "Open Digital Architecture", which proposes a framework of interoperable APIs and microservices for all services to replace existing approaches to deployments (Forum, 2021). Beyond the deployment and integration with both existing and future technologies, the need for scaling and performance is an existing major concern for virtually deployed systems, with AWS and kubernetes popular for their automatic resource scaling. There are limitations to the extent that scaling virtualisation may offset the performance loss experienced by

using a software fallback or redundancy to a physical primary middlebox (C. Wang, Spatscheck, Gopalakrishnan, Y. Xu, et al., 2016), but further exploration of the effectiveness of this research when scaled to significantly larger numbers than was feasible within the course of this thesis would be of interest as a body of future research and development.

6.3.3 State recreation approaches

The filtering mechanism is efficient in its design, proving effective yet easy to replicate in a multitude of technologies. Further work in this area may be able to reveal further refinements to its technique, reducing the load of duplicate traffic further. This filter is not the only viable approach, however, and further work in this domain would be especially productive in discovering new non-modifying approaches that may refine its design further. Areas where this improvement would be of significant benefit include consistent monitoring, where observation is better over a connection's lifespan, such as an IDS or IPS, or reducing the volume of information necessary to recreate the state even further. There are several possible directions this might take that offer potential. Sketching algorithms are a proposed alternative to monitoring networks using data structures to estimate statistics of a streaming workload (Liu et al., 2019). Rather than consuming bandwidth with high-fidelity monitoring (Alizadeh et al., 2014, García-Teodoro et al., 2009) sketching proposals utilise hashing techniques to estimate the size and volume of traffic, providing an approximate view of the network without consuming network resources.

This can include heavy hitter detection (Ben-Basat et al., 2018, Schweller et al., 2004, Sivaraman et al., 2017), per-flow frequency (Charikar et al., 2004) and traffic pattern estimation (Krishnamurthy et al., 2003). There are several examples of this technology in modern literature (Q. Huang, Lee, et al., 2018, Liu et al., 2019) originally targeting physical switches with and now with a transition more towards implementing techniques into the growing presence of software switches and NFV/SDN (M. Yu et al., 2013, Q. Huang, Jin, et al., 2017, T. Yang et al., 2018). The advantages of this are clear in their minimal footprint on resource usage when their accuracy is sufficient for real-time use and the parallel of their use is clear for the purposes of preserving state; if certain information can be predicted using a minimal degree of information derived from traffic flows, it may be possible to expand this concept further to encompass middleboxes so that their existing state can be both modelled and replicated using predictive techniques, further reducing the impact on the system in resource consumption and interference with failure-free operations.

6.3.4 AI and machine learning

The proposed use of sketching algorithms to predict the behaviour of blackboxes and the decisions they will make could be defined as a form of state machine. Typically, these behaviour models are rather simplistic, and the middleboxes that the work of this thesis has targeted are not stateless, producing the same result given the same input. Machine learning could bridge this gap instead, creating models based on data collection and reinforcement learning (M. Wang et al., 2018). Through observation of incoming and outgoing traffic and the operations that the box is expected to perform, middleboxes could potentially be modelled and their behaviour replicated externally. The limited variability of hardware middlebox design is beneficial in this regard. Machine learning generates models through a series of steps, following a cycle of problem formulation, data collection, data analysis, model construction and finally model validation. The use of AI and machine learning in networking was limited until recent years due to a number of issues, including the diversity of network designs from their components to their layouts and the volume of information necessary for their operations to accurately model the network's operations and react appropriately to its ever-changing traffic (Boutaba et al., 2018). The growth of SDN into enterprise networks allows a great deal of potential control over its operations; however, a number of use cases for ML in networking have developed in recent years. This includes traffic prediction (Samira et al., 2010, Bermolen et al., 2008) and classification (Sun et al., 2010), as well as more complex operations such as congestion control (Fonseca et al., 2005) and QoE management (Shaikh et al., 2010, Fiedler et al., 2010).

Traffic prediction, as a product of ML, can be broken down into its steps: collecting traffic traces with flow statistics, observation of trends in significant volumes to determine the decision-making logic, training a Hidden-Markov model with Kernel Bayes Rule and Recurrent Neural Networks, and finally inputting live traffic traces and evaluating the accuracy of its results (M. Wang et al., 2018). Work in this domain is generally limited more to traffic volume estimation such as network tomography (A. Chen et al., 2007), but there is a body of work dedicated to generating simulated flow statistics and similar table counts (Z. Chen et al., 2016). Machine learning is a wide area of research and is out of the scope of this thesis, but its potential for simulating flow information accurately provides a future avenue of research worth examining further for generating state for redundant units without the need for duplicating packet processing with live replaying.

6.3.5 Non-TCP based statefulness

The focus of both this thesis and the work prior to it has been almost exclusively on TCP traffic; it forms the majority of traffic within the Internet by a significant margin over UDP, but these are not the only protocols employed in networking today.

Newer transport layer protocols such as Quick UDP Connections (QUIC) (Iyengar et al., 2021) have seen growing adoption beyond their initial drafts, with a number of IETF standards now set (Thomson and Turner, 2021, Thomson, 2021), and this may pose a problem for the techniques explored thus far. QUIC, established by Google in 2012, utilises multiplexing and UDP to divide the traffic across multiple independent streams to be sent to the target end. This approach establishes some measure of redundancy for active connections on a per-stream basis and eliminates the need for some of the TCP's mechanisms, namely its handshake process and other mechanisms that might incur delay in the name of guaranteeing transmission. QUIC aims to provide the same level of guarantees as TCP without this additional incurred latency through multiplexing, with UDP used as its base. This new protocol presents a number of issues for the work pursued. Firstly, the majority of middleboxes in use today drop UDP traffic, which includes QUIC, limiting its adoption to a degree. This is a known problem of middleboxes and their contribution to network ossification, with their need to be aware of the network protocols for the traffic they operate upon a well-established problem. There are a number of proposed solutions to this problem, with QUIC adopting a fallback mechanism to TCP and HTTP/2 if packet loss is observed with the initial QUIC connection. The multiplexed streams of even an individual traffic flow add a significant degree of complexity to the proposed designs, as none of the approaches used for TCP traffic can be employed here. These streams cannot be externally identified and handled as an observable multiplexed connection; connection IDs, the markers used to indicate the individual streams are a part of the same connection, are explicitly designed so as not to be identifiable to an outside observer (such as a middlebox breaking the end-to-end principle) to accomplish this.

One advantage is that any streams received that are inconsistent with the rest of the stream's state, such as falling out of the idle timeout window or having already been rebroadcast, are dropped immediately and will not cause any confusion in the protocol. Streams can potentially be treated as separate flows, although this is difficult to evaluate without testing. Loss of a single stream can be recovered at the end points of the connection, but there are a number of in-built mechanisms to this protocol that the operations of something like Katoptron might interfere with. These mechanisms include handling connection migration, loss detection and path validation, which, while highly beneficial to the traffic itself, adds a degree of complexity to the work performed in this thesis. Unexpected behaviour such as redirection without resetting a stream from either end poses a number of potential risks that would require significant exploration. The use of QUIC is likely to grow, as are other protocols that attempt to circumvent the ossification of the networking infrastructure. This is predominantly a problem for middleboxes themselves and how they handle non-TCP traffic, but equally challenges certain presumptions about the nature of traffic received by my work. It will require consideration and potentially need to be incorporated into any future

work pursued in this domain, depending on the continued presence of middleboxes in the network.

References

Abhishek, Rohit, David Tipper, and Deep Medhi (2020). "Resilience of 5G Networks in the Presence of Unlicensed Spectrum and Non-Terrestrial Networks". In: *2020 16th International Conference on the Design of Reliable Communication Networks DRCN 2020*. DOI: 10.1109/DRCN48652.2020.1570604438.

Alizadeh, Mohammad, Tom Edsall, Sarang Dharmapurikar, Ramanan Vaidyanathan, Kevin Chu, Andy Fingerhut, Vinh The Lam, Francis Matus, Rong Pan, Navindra Yadav, and George Varghese (2014). "CONGA: Distributed Congestion-Aware Load Balancing for Datacenters". In: *Proceedings of the 2014 ACM Conference on SIGCOMM*. SIGCOMM '14. Association for Computing Machinery. DOI: 10.1145/2619239.2626316.

Allman, Mark (2003). "On the Performance of Middleboxes". In: *Proceedings of the 3rd ACM SIGCOMM Conference on Internet Measurement*. IMC '03. Association for Computing Machinery. DOI: 10.1145/948205.948246.

Amazon (2019). *Resilience in Amazon EC2 Auto Scaling*. URL: https://docs.aws.amazon.com/pdfs/autoscaling/ec2/userguide/as-dg.pdf#disaster-recovery-resiliency.

archive, HTTP (2010). *HTTP archive: page weight*. URL: https://httparchive.org/reports/page-weight?start=2017_05_15&end=latest&view=list.

Arcilla, Alex and Tony Palmer (2019). *Achieving Predictably High Performance for Real-world Data Center workloads*. URL: https://docs.broadcom.com/doc/12395356.

Atlas, Alia, George Swallow, and Ping Pan (2005). *Fast Reroute Extensions to RSVP-TE for LSP Tunnels*. RFC 4090. DOI: 10.17487/RFC4090. URL: https://www.rfc-editor.org/info/rfc4090.

Barham, Paul, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield (2003a). "Xen and the Art of Virtualization". In: *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*. SOSP '03. Association for Computing Machinery. DOI: 10.1145/945445.945462.

— (2003b). "Xen and the Art of Virtualization". In: *SIGOPS Oper. Syst. Rev.* 37. DOI: 10.1145/1165389.945462.

Belyaev, M. and S. Gaivoronski (2014). "Towards load balancing in SDN-networks during DDoS-attacks". In: *2014 International Science and Technology Conference (Modern Networking Technologies) (MoNeTeC)*. DOI: 10.1109/MoNeTeC.2014.6995578.

Ben-Basat, Ran, Xiaoqi Chen, Gil Einziger, and Ori Rottenstreich (2018). "Efficient Measurement on Programmable Switches Using Probabilistic Recirculation". In: *2018 IEEE 26th International Conference on Network Protocols (ICNP)*. IEEE. DOI: 10.1109/icnp.2018.00047.

Benjamin Rainer Stefan Lederer, Christopher Müller and Christian Timmerer (2012). *Big Buck Bunny MPEG-DASH testing*. https://dash.itec.aau.at/dash-js/.

Bermolen, Paola and Dario Rossi (2008). "Support vector regression for link load prediction". In: *2008 4th International Telecommunication Networking Workshop on QoS in Multiservice IP Networks*. DOI: 10.1109/ITNEWS.2008.4488164.

Bierman, Andy (2011). *Guidelines for Authors and Reviewers of YANG Data Model Documents*. RFC 6087. DOI: 10.17487/RFC6087. URL: https://www.rfc-editor.org/info/rfc6087.

— (2018). *Guidelines for Authors and Reviewers of Documents Containing YANG Data Models*. RFC 8407. DOI: 10.17487/RFC8407. URL: https://www.rfc-editor.org/info/rfc8407.

Bosshart, Pat, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker (2014). "P4: Programming Protocol-Independent Packet Processors". In: *SIGCOMM Comput. Commun. Rev.* 44. DOI: 10.1145/2656877.2656890.

Boutaba, R., Mohammad Salahuddin, Noura Limam, Sara Ayoubi, Nashid Shahriar, Felipe Estrada-Solano, and Mauricio Caicedo (2018). "A Comprehensive Survey on Machine Learning for Networking: Evolution, Applications and Research Opportunities". In: *Journal of Internet Services and Applications* 9. DOI: 10.1186/s13174-018-0087-2.

Bressoud, T. C. and F. B. Schneider (1995). "Hypervisor-Based Fault Tolerance". In: *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles*. SOSP '95. Association for Computing Machinery. DOI: 10.1145/224056.224058.

Brim, Scott W. and Brian E. Carpenter (2002). *Middleboxes: Taxonomy and Issues*. RFC 3234. DOI: 10.17487/RFC3234. URL: https://www.rfc-editor.org/info/rfc3234.

Broadbent, Matthew (2015). *Scootplayer*. https://github.com/broadbent/scootplayer.

Broadcom (2024). *Broadcom Trident 4-X7 Diagram*. URL: https://www.servethehome.com/broadcom-trident-4-x7-for-400gbe-networking-launched/ (visited on 01/16/2024).

Brodkin, Jon (2012). *Why Gmail went down: Google misconfigured load balancing servers*. http://static.googleusercontent.com/external_content/untrusted_dlcp/www.google.com/en/us/appsstatus/ir/plibxfjh8whr44h.pdf. URL: %5Curl%7Bhttps://arstechnica.com/information-technology/2012/12/why-gmail-went-down-google-misconfigured-chromes-sync-server/%7D.

Buldyrev, Sergey V, Roni Parshani, Gerald Paul, H Eugene Stanley, and Shlomo Havlin (2010). "Catastrophic cascade of failures in interdependent networks". In: *Nature* 464. DOI: 10.1038/nature08932.

Burrows, Mike (2006). "The Chubby lock service for loosely-coupled distributed systems". In: *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*. OSDI '06. USENIX Association. DOI: 10.5555/1298455.1298487.

Caesar, Matthew, Donald Caldwell, Nick Feamster, Jennifer Rexford, Aman Shaikh, and Jacobus Van Der Merwe (2005). "Design and implementation of a routing control platform". In: *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*.

Cao, Lianjie, Puneet Sharma, Sonia Fahmy, and Vinay Saxena (2015). "NFV-VITAL: A framework for characterizing the performance of virtual network functions". In: *2015 IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN)*. DOI: 10.1109/NFV-SDN.2015.7387412.

Casado, Martin, Michael J Freedman, Justin Pettit, Jianying Luo, Nick McKeown, and Scott Shenker (2007). "Ethane: Taking control of the enterprise". In: *ACM SIGCOMM computer communication review* 37. DOI: 10.1145/1282427.1282382.

Casazza, Marco, Mathieu Bouet, and Stefano Secci (2019). "Availability-driven NFV orchestration". In: *Comput. Netw.* 155. DOI: 10.1016/j.comnet.2019.02.017.

Cetinkaya, Egemen K and James PG Sterbenz (2013). "A taxonomy of network challenges". In: *2013 9th International Conference on the Design of Reliable Communication Networks (DRCN)*. IEEE.

Charikar, Moses, Kevin Chen, and Martin Farach-Colton (2004). "Finding frequent items in data streams". In: *Theoretical Computer Science* 312. Automata, Languages and Programming. DOI: https://doi.org/10.1016/S0304-3975(03)00400-6. URL: https://www.sciencedirect.com/science/article/pii/S0304397503004006.

Chen, A., J. Cao, and T. Bu (2007). "Network Tomography: Identifiability and Fourier Domain Estimation". In: *IEEE INFOCOM 2007 - 26th IEEE International Conference on Computer Communications*. DOI: 10.1109/INFCOM.2007.218.

Chen, Yang and Jie Wu (2018). "NFV Middlebox Placement with Balanced Set-up Cost and Bandwidth Consumption". In: *Proceedings of the 47th International Con-*

ference on Parallel Processing. ICPP '18. Association for Computing Machinery. DOI: 10.1145/3225058.3225068.

Chen, Zhitang, Jiayao Wen, and Yanhui Geng (2016). "Predicting future traffic using Hidden Markov Models". In: *2016 IEEE 24th International Conference on Network Protocols (ICNP)*. DOI: 10.1109/ICNP.2016.7785328.

Chiesa, Marco, Roshan Sedar, Gianni Antichi, Michael Borokhovich, Andrzej Kamisiński, Georgios Nikolaidis, and Stefan Schmid (2019a). "PURR: A Primitive for Reconfigurable Fast Reroute: Hope for the Best and Program for the Worst". In: *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies*. CoNEXT '19. New York, NY, USA: Association for Computing Machinery. DOI: 10.1145/3359989.3365410.

— (2019b). "PURR: a primitive for reconfigurable fast reroute: hope for the best and program for the worst". In: *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies*. CoNEXT '19. Association for Computing Machinery. DOI: 10.1145/3359989.3365410.

Cholda, Piotr, Anders Mykkeltveit, Bjarne E Helvik, Otto J Wittner, and Andrzej Jajszczyk (2007). "A survey of resilience differentiation frameworks in communication networks". In: *IEEE Communications Surveys & Tutorials* 9. DOI: 10.1109/COMST.2007.4444749.

Cisco (2013). *Cisco's One Platform Kit (onePK)*. https://blogs.cisco.com/security/ciscos-onepk-part-1-introduction.

— (2018). *Cisco Annual Internet Report*. Tech. rep. URL: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html.

— (2020). *Cisco Cloud Services Platform 5000 Series*. https://www.cisco.com/c/en/us/products/collateral/switches/cloud-services-platform-5000/nb-06-csp-5k-data-sheet-cte-en.html.

cisco (2023). *Catalyst Switched Port Analyser*. URL: https://www.cisco.com/c/en/us/support/docs/switches/catalyst-6500-series-switches/10570-41.html (visited on 2023).

Clark, Christopher, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield (2005). "Live Migration of Virtual Machines". In: *Proceedings of the 2nd Conference on Symposium on Networked Systems Design amp; Implementation - Volume 2*. NSDI'05. USENIX Association. DOI: 10.5555/1251203.1251223.

Colman-Meixner, Carlos, Chris Develder, Massimo Tornatore, and Biswanath Mukherjee (2016). "A survey on resiliency techniques in cloud computing infrastructures and applications". In: *IEEE Communications Surveys & Tutorials* 18. DOI: 10.1109/COMST.2016.2531104.

Corbató, Fernando J (1963). *The Compatible Time-Sharing System: A Programmer's Guide*. The MIT Press.

Cully, Brendan, Geoffrey Lefebvre, Dutch Meyer, Mike Feeley, Norm Hutchinson, and Andrew Warfield (2008). "Remus: High Availability via Asynchronous Virtual Machine Replication". In: *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*. NSDI'08. USENIX Association. DOI: 10.5555/1387589.1387601.

David Hutchison, James P. G Sterbenz (2015). *Resilinets*. URL: https://resilinets.org/main_page.html.

Dell (2015). Tech. rep. URL: https://i.dell.com/sites/doccontent/business/large-business/en/documents/BeyondFiveNines-availability-AchievingHighAvailabilty-with-DellCompellentStorageCenter.pdf.

Detal, Gregory, Benjamin Hesmans, Olivier Bonaventure, Yves Vanaubel, and Benoit Donnet (2013). "Revealing Middlebox Interference with Tracebox". In: *Proceedings of the 2013 Conference on Internet Measurement Conference*. IMC '13. Association for Computing Machinery. DOI: 10.1145/2504730.2504757.

Diffie, W. and M. Hellman (1976). "New directions in cryptography". In: *IEEE Transactions on Information Theory* 22. DOI: 10.1109/TIT.1976.1055638.

Dong, YaoZu, Wei Ye, YunHong Jiang, Ian Pratt, ShiQing Ma, Jian Li, and HaiBing Guan (2013). "COLO: COarse-Grained LOck-Stepping Virtual Machines for Non-Stop Service". In: *Proceedings of the 4th Annual Symposium on Cloud Computing*. SOCC '13. Association for Computing Machinery. DOI: 10.1145/2523616.2523630.

Dunlap, George W., Samuel T. King, Sukru Cinar, Murtaza A. Basrai, and Peter M. Chen (2002). "ReVirt: Enabling Intrusion Analysis through Virtual-Machine Logging and Replay". In: *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (Copyright Restrictions Prevent ACM from Being Able to Make the PDFs for This Conference Available for Downloading)*. OSDI '02. USENIX Association. ISBN: 9781450301114.

Dunlap, George W., Dominic G. Lucchetti, Michael A. Fetterman, and Peter M. Chen (2008). "Execution replay of multiprocessor virtual machines". In: *Proceedings of the Fourth ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*. VEE '08. Association for Computing Machinery. DOI: 10.1145/1346256.1346273.

Elliott, Chip (2008). "GENI - global environment for network innovations". In: *2008 33rd IEEE Conference on Local Computer Networks (LCN)*. DOI: 10.1109/LCN.2008.4664143.

Ellison, Robert J, David A Fisher, Richard C Linger, Howard F Lipson, Thomas Longstaff, and Nancy R Mead (1997). "Survivable network systems: An emerging discipline". In.

Ellison, Robert J, David A Fisher, Richard C Linger, Howard F Lipson, Thomas A Longstaff, and Nancy R Mead (1999). "Survivability: Protecting your critical systems". In: *IEEE Internet Computing* 3. DOI: 10.1109/4236.807008.

Elwalid, A., C. Jin, S. Low, and I. Widjaja (2001). "MATE: MPLS adaptive traffic engineering". In: *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No.01CH37213)*. Vol. 3. DOI: 10.1109/INFCOM.2001.916625.

ETSI (2012). "ETSI GS NFV 006 V4.4.1 (2022-12) Network Functions Virtualisation (NFV) Release 4; Management and Orchestration; Architectural Framework Specification". In: *Architectural Framework. sl: ETSI*.

— (2014a). "ETSI GS NFV-MAN 001: "Network Functions Virtualisation (NFV); Management and Orchestration"". In: *Architectural Framework. sl: ETSI*.

— (2014b). "ETSI GS NFV-SWA 001: "Network Functions Virtualisation (NFV); Virtual Network Functions Architecture"". In: *Architectural Framework. sl: ETSI*.

— (2016). "ETSI GS NFV-REL 003: "Network Functions Virtualisation (NFV); Resiliency Requirements"". In: *Architectural Framework. sl: ETSI*.

ETSI, N (2013). "Etsi gs nfv 002 v1. 1.1 network functions virtualization (nfv)". In: *Architectural Framework. sl: ETSI*.

F5 (2011). *2011 ADC Security Survey Global Findings*. URL: %5Curl%7Bhttps://www.slideshare.net/f5dotcom/2011-f5-adc-security-survey-global-slide-share%7D.

Fayazbakhsh, Seyed Kaveh, Vyas Sekar, Minlan Yu, and Jeffrey C. Mogul (2013). "FlowTags: Enforcing Network-Wide Policies in the Presence of Dynamic Middlebox Actions". In: *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*. HotSDN '13. Association for Computing Machinery. ISBN: 9781450321785. DOI: 10.1145/2491185.2491203.

Feamster, Nick, Jennifer Rexford, and Ellen Zegura (2014). "The Road to SDN: An Intellectual History of Programmable Networks". In: *SIGCOMM Comput. Commun. Rev.* 44. DOI: 10.1145/2602204.2602219.

Fiedler, Markus, Tobias Hossfeld, and Phuoc Tran-Gia (2010). "A Generic Quantitative Relationship between Quality of Experience and Quality of Service". In: *Netwrk. Mag. of Global Internetwkg.* 24. DOI: 10.1109/MNET.2010.5430142.

Fonseca, N. and M. Crovella (2005). "Bayesian packet loss detection for TCP". In: *Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies.* Vol. 3. DOI: 10.1109/INFCOM.2005.1498462.

Fortz, B. and M. Thorup (2002). "Optimizing OSPF/IS-IS weights in a changing world". In: *IEEE Journal on Selected Areas in Communications* 20. DOI: 10.1109/JSAC.2002.1003042.

Fortz, Bernard and Mikkel Thorup (2000). "Internet traffic engineering by optimizing OSPF weights". In: *Proceedings IEEE INFOCOM 2000. conference on computer communications. Nineteenth annual joint conference of the IEEE computer and communications societies (Cat. No. 00CH37064)*. Vol. 2. IEEE. DOI: 10.1109/INFCOM.2000.832225.

Forum, TM (2021). *Open Digital Architecture*. https://www.tmforum.org/oda/.

Foundation, Linux (2015). *Data Plane Development Kit (DPDK)*. URL: http://www.dpdk.org.

Foundation, Open Information Security (2022). *Suricata - Open Source IDS/IPS*. URL: https://suricata.io/.

García-Teodoro, Pedro, Jesús Díaz-Verdejo, Gabriel Maciá-Fernández, and Enrique Vázquez (2009). "Anomaly-based network intrusion detection: Techniques, systems and challenges". In: *Computers and Security* 28. DOI: 10.1016/j.cose.2008.08.003.

Gavras, Anastasius, Arto Karila, Serge Fdida, Martin May, and Martin Potts (2007). "Future internet research and experimentation: the FIRE initiative". In: *SIGCOMM Comput. Commun. Rev.* 37. DOI: 10.1145/1273445.1273460.

Gerhards, Rainer (2009). *The Syslog Protocol*. RFC 5424. DOI: 10.17487/RFC5424. URL: https://www.rfc-editor.org/info/rfc5424.

Gilbert, Seth and Nancy Lynch (2002). "Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services". In: *SIGACT News* 33. DOI: 10.1145/564585.564601.

— (2012). "Perspectives on the CAP Theorem". In: *Computer* 45. DOI: 10.1109/MC.2011.389.

Glozer, Will (Dec. 30, 2023). *wrk - a HTTP benchmarking tool*. URL: https://github.com/wg/wrk.

Gude, Natasha, Teemu Koponen, Justin Pettit, Ben Pfaff, Martín Casado, Nick McKeown, and Scott Shenker (2008). "NOX: Towards an Operating System for Networks". In: *SIGCOMM Comput. Commun. Rev.* 38. DOI: 10.1145/1384609.1384625.

Hajar, Hantouti, Nabil Benamar, Tarik Taleb, and Abdelquoddouss Laghrissi (2018). "Traffic Steering for Service Function Chaining". In: *IEEE Communications Surveys and Tutorials* PP. DOI: 10.1109/COMST.2018.2862404.

Haleplidis, Evangelos, Kostas Pentikousis, Spyros Denazis, Jamal Hadi Salim, David Meyer, and Odysseas Koufopavlou (2015). *Software-Defined Networking (SDN): Layers and Architecture Terminology*. RFC 7426. DOI: 10.17487/RFC7426. URL: https://www.rfc-editor.org/info/rfc7426.

Handigol, Nikhil, Srini Seetharaman, Mario Flajslik, Nick McKeown, and Ramesh Johari (2009). "Plug-n-Serve: Load-Balancing Web Traffic using OpenFlow". In: vol. 4.

Hantouti, Hajar, Nabil Benamar, Tarik Taleb, and Abdelquoddous Laghrissi (2019). "Traffic Steering for Service Function Chaining". In: *IEEE Communications Surveys and Tutorials* 21. DOI: 10.1109/COMST.2018.2862404.

Harris, Chandler (2011). *Data center outages generate big losses*. URL: https://www.informationweek.com/it-sectors/data-center-outages-generate-big-losses.

Hill, Lyn, Charalampos Rotsos, Chris Edwards, and David Hutchison (2024). "Katoptron: Efficient State Mirroring for Middlebox Resilience". In: *NOMS 2024-2024 IEEE Network Operations and Management Symposium*, pp. 1–9. DOI: 10.1109/NOMS59830.2024.10575815.

Hill, Lyn, Charalampos Rotsos, Will Fantom, Chris Edwards, and David Hutchison (2022). "Improving network resilience with Middlebox Minions". In: *NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium*. DOI: 10.1109/NOMS54207.2022.9789819.

Honda, Michio, Yoshifumi Nishida, Costin Raiciu, Adam Greenhalgh, Mark Handley, and Hideyuki Tokuda (2011). "Is It Still Possible to Extend TCP?" In: *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference*. IMC '11. Association for Computing Machinery. ISBN: 9781450310130. DOI: 10.1145/2068816.2068834.

Hopps, Christian (2000). *Analysis of an Equal-Cost Multi-Path Algorithm*. RFC 2992. DOI: 10.17487/RFC2992. URL: https://www.rfc-editor.org/info/rfc2992.

Huang, Qun, Xin Jin, Patrick P. C. Lee, Runhui Li, Lu Tang, Yi-Chao Chen, and Gong Zhang (2017). "SketchVisor: Robust Network Measurement for Software Packet Processing". In: *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. SIGCOMM '17. Association for Computing Machinery. DOI: 10.1145/3098822.3098831.

Huang, Qun, Patrick P. C. Lee, and Yungang Bao (2018). "Sketchlearn: Relieving User Burdens in Approximate Measurement with Automated Statistical Inference". In: *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. SIGCOMM '18. Association for Computing Machinery. DOI: 10.1145/3230543.3230559.

Huang, Shan, Félix Cuadrado, and Steve Uhlig (2017). "Middleboxes in the Internet: A HTTP perspective". In: *2017 Network Traffic Measurement and Analysis Conference (TMA)*. DOI: 10.23919/TMA.2017.8002906.

Huawei (2022). *Link Aggregation Control Protocol*. URL: https://support.huawei.com/enterprise/en/doc/EDOC1100086560#EN-US_TOPIC_0169439602.

IBM (2024). *PowerHA® SystemMirror® cluster heartbeat over TCP*. URL: https://www.ibm.com/docs/en/powerha-aix/7.2?topic=heartbeating-over-tcpip-storage-area-networks (visited on 2024).

IDC (2021). *The business value of NVIDIA ethernet switch solutions for managing and optimising network performance*. Tech. rep. URL: https://resource.nvidia.com/en-us-ethernet-switching/idc-biz-value-ethernet-swtich-wp.

IEC (2016). *Industrial communication networks - High availability automation networks - Part 3: Parallel Redundancy Protocol (PRP) and High-availability Seamless Redundancy (HSR)*. en. Tech. rep. International Electrotechnical Commission. URL: https://webstore.iec.ch/publication/24447.

Inc, Proofpoint (2021). *Emerging Threats Ruleset*. https://rules.emergingthreats.net/.

Ingham, Kenneth, Kenneth Consulting, and Stephanie Forrest (2002). "A history and survey of network firewalls". In: *University of New Mexico, Tech. Rep.*

initiative, OpenConfig (2024). *Open Config*. URL: https://www.openconfig.net/ (visited on 2024).

Iyengar, Jana and Martin Thomson (2021). *QUIC: A UDP-Based Multiplexed and Secure Transport*. RFC 9000. DOI: 10.17487/RFC9000. URL: https://www.rfc-editor.org/info/rfc9000.

Jain, Sushant, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, Jon Zolla, Urs Hölzle, Stephen Stuart, and Amin Vahdat (2013). "B4: Experience with a Globally-Deployed Software Defined Wan". In: *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*. SIGCOMM '13. Association for Computing Machinery. DOI: 10.1145/2486001.2486019.

Juan-Marin, Ruben de, Hendrik Decker, and Francesc D. Munoz-Esco (2007). "Revisiting Hot Passive Replication". In: *The Second International Conference on Availability, Reliability and Security (ARES'07)*. DOI: 10.1109/ARES.2007.126.

Julkunen, H. and C.E. Chow (1998). "Enhance network security with dynamic packet filter". In: *Proceedings 7th International Conference on Computer Communications and Networks (Cat. No.98EX226)*. DOI: 10.1109/ICCCN.1998.998786.

Juniper (2009). *Junos Space Network Management Platform*. URL: %5Curl%7Bhttps://www.juniper.net/gb/en/products/sdn-and-orchestration/junos-space-platform.html%7D.

Kang, Nanxi, Monia Ghobadi, John Reumann, Alexander Shraer, and Jennifer Rexford (2015a). "Efficient Traffic Splitting on Commodity Switches". In: *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies*. CoNEXT '15. Association for Computing Machinery. DOI: 10.1145/2716281.2836091.

— (2015b). "Efficient traffic splitting on commodity switches". In: *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies*. CoNEXT '15. Association for Computing Machinery. DOI: 10.1145/2716281.2836091.

Kent, Stephen Thomas (1976). "Encryption-based protection protocols for interactive user-computer communication over physically unsecured channels." PhD thesis. Massachusetts Institute of Technology.

Kerrisk, Michael (2013). *Namespaces in operation, part 5: User namespaces*. URL: https://lwn.net/Articles/532593/.

Khalid, Junaid and Aditya Akella (2019). "Correctness and Performance for Stateful Chained Network Functions". In: *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. DOI: 10.5555/3323234.3323276.

Kneschke, Jan (2003). *Lighttpd: a light-weight web server*. https://www.lighttpd.net/.

Kohler, Eddie, Robert Morris, Benjie Chen, John Jannotti, and M. Frans Kaashoek (2000). "The click modular router". In: *ACM Trans. Comput. Syst.* 18. DOI: 10.1145/354871.354874.

Krishnamurthy, Balachander, Subhabrata Sen, Yin Zhang, and Yan Chen (2003). "Sketch-Based Change Detection: Methods, Evaluation, and Applications". In: *Proceedings of the 3rd ACM SIGCOMM Conference on Internet Measurement*. IMC '03. Association for Computing Machinery. DOI: 10.1145/948205.948236.

Kulkarni, Sameer G, Guyue Liu, K. K. Ramakrishnan, Mayutan Arumaithurai, Timothy Wood, and Xiaoming Fu (2020). "REINFORCE: Achieving Efficient Failure Resiliency for Network Function Virtualization Based Services". In: vol. 28. DOI: 10.1145/3281411.3281441.

Lamport, Leslie (2001). "Paxos Made Simple". In: *ACM SIGACT News (Distributed Computing Column) 32, 4 (Whole Number 121, December 2001)*. URL: https://www.microsoft.com/en-us/research/publication/paxos-made-simple/.

Lantz, Bob and Brandon Heller (2009). *Mininet: An Instant Virtual Network on your Laptop*. http://mininet.org/.

Larry Peterson, Carmelo Cascone, Brian O'Connor, Thomas Vachuska, and Bruce Davie (2016). *Software-Defined Networks: A Systems Approach*. en. Standard. URL: https://sdn.systemsapproach.org/index.html.

LeBlanc, T. J. and J. M. Mellor-Crummey (1987). "Debugging Parallel Programs with Instant Replay". In: *IEEE Trans. Comput.* 36. DOI: 10.1109/TC.1987.1676929.

Leiner, Barry M., Vinton G. Cerf, David D. Clark, Robert E. Kahn, Leonard Kleinrock, Daniel C. Lynch, Jon Postel, Larry G. Roberts, and Stephen Wolff (2009). "A brief history of the internet". In: *SIGCOMM Comput. Commun. Rev.* 39. DOI: 10.1145/1629607.1629613.

Leiner, Barry M., Vinton G. Cerf, David D. Clark, Robert E. Kahn, Leonard Kleinrock, Daniel C. Lynch, Jon Postel, Lawrence G. Roberts, and Stephen S. Wolff (1997). "The past and future history of the Internet". In: *Commun. ACM* 40. DOI: 10.1145/253671.253741.

Levin, Dan, Marco Canini, Stefan Schmid, Fabian Schaffert, and Anja Feldmann (2014). "Panopticon: Reaping the Benefits of Incremental SDN Deployment in Enterprise Networks". In: *2014 USENIX Annual Technical Conference (USENIX ATC 14)*. USENIX Association. ISBN: 978-1-931971-10-2. URL: https://www.usenix.org/conference/atc14/technical-sessions/presentation/levin.

Li, Dawn, Bruce A. Cole, Phil Morton, and Tony Li (1998). *Cisco Hot Standby Router Protocol (HSRP)*. RFC 2281. DOI: 10.17487/RFC2281. URL: https://www.rfc-editor.org/info/rfc2281.

Lira, Victor, Eduardo Tavares, Stenio Fernandes, Paulo Maciel, and Ricardo M.A. Silva (2013). "Virtual Network Resource Allocation Considering Dependability Issues". In: *2013 IEEE 16th International Conference on Computational Science and Engineering*. DOI: 10.1109/CSE.2013.58.

Liu, Zaoxing, Ran Ben-Basat, Gil Einziger, Yaron Kassner, Vladimir Braverman, Roy Friedman, and Vyas Sekar (2019). "Nitrosketch: Robust and General Sketch-Based Monitoring in Software Switches". In: *Proceedings of the ACM Special Interest Group on Data Communication*. SIGCOMM '19. Association for Computing Machinery. DOI: 10.1145/3341302.3342076.

Lyons, R. E. and W. Vanderkulk (1962). "The Use of Triple-Modular Redundancy to Improve Computer Reliability". In: *IBM Journal of Research and Development* 6. DOI: 10.1147/rd.62.0200.

Martins, Joao, Mohamed Ahmed, Costin Raiciu, Vladimir Olteanu, Michio Honda, Roberto Bifulco, and Felipe Huici (2014). "ClickOS and the Art of Network Function Virtualization". In: *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*. Seattle, WA: USENIX Association. DOI: 10.5555/2616448.2616491. URL: https://www.usenix.org/conference/nsdi14/technical-sessions/presentation/martins.

Marvell (2024). *Data flow architecture*. http://www.marvell.com/network-processors/technology/data-flow-architecture/. (Visited on 2024).

Mather, Michael (2018). "A Roadmap for Cybersecurity Research". In.

McDaniel, Patrick and Stephen McLaughlin (2009). "Security and privacy challenges in the smart grid". In: *IEEE security & privacy* 7. DOI: 10.1109/MSP.2009.76.

McKeown, Nick, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner (2008a). "OpenFlow: Enabling Innovation in Campus Networks". In: *SIGCOMM Comput. Commun. Rev.* 38. DOI: 10.1145/1355734.1355746.

— (2008b). "OpenFlow: enabling innovation in campus networks". In: *ACM SIGCOMM computer communication review* 38. DOI: 10.1145/1355734.1355746.

McQuillan, J., I. Richer, and E. Rosen (1980). "The New Routing Algorithm for the ARPANET". In: *IEEE Transactions on Communications* 28. DOI: 10.1109/TCOM.1980.1094721.

McQuillan, John M and David C Walden (1977). "The ARPA network design decisions". In: *Computer Networks (1976)* 1. DOI: https://doi.org/10.1016/0376-5075(77)90014-9.

Merwe, Jacobus E van der and Ian M Leslie (1997). "Switchlets and dynamic virtual ATM networks". In: *Integrated Network Management V: Integrated management in a virtual world Proceedings of the Fifth IFIP/IEEE International Symposium on Integrated Network Management San Diego, California, USA, May 12–16, 1997*. Springer. ISBN: 0412809605.

Needham, Roger M. and Michael D. Schroeder (1978). "Using Encryption for Authentication in Large Networks of Computers". In: *Commun. ACM* 21. DOI: 10.1145/359657.359659.

Niknami, Nadia, Avinash Srinivasan, Ken St. Germain, and Jie Wu (Dec. 2023). "Maritime Communications—Current State and the Future Potential with SDN and SDR". In: *Network* 3, pp. 563–584. DOI: 10.3390/network3040025.

Nokia (2022). *Nokia Service Router Linux*. Tech. rep. URL: https://onestore.nokia.com/asset/207598ga=2.11886007.223215410.1674487897-1044014544.1674487897.

Nvidia (2022). *DPU power efficiency*. Tech. rep. URL: https://resources.nvidia.com/en-us-telecom-wireless-resource/nvidia-dpu-power-eff-1?lx=u_4HIc&contentType=white-paper.

Ocean, Digital (2022). *Digital Ocean Reserved IPs*. URL: https://docs.digitalocean.com/products/networking/reserved-ips/details/.

Ongaro, Diego and John Ousterhout (2014). "In Search of an Understandable Consensus Algorithm". In: *Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference*. USENIX ATC'14. USENIX Association. DOI: 10.5555/2643634.2643666.

OpenFlow data-plane abstraction (OF-DPA): Abstract switch specification (2014). https://docs.broadcom.com/doc/12378911.

Org, Netfilter (2004). *The Netfilter Project*. URL: https://www.netfilter.org/.

Panda, Aurojit, James Murphy McCauley, Amin Tootoonchian, Justine Sherry, Teemu Koponen, Syliva Ratnasamy, and Scott Shenker (2016). "Open Network Interfaces for Carrier Networks". In: *SIGCOMM Comput. Commun. Rev.* 46. DOI: 10.1145/2875951.2875953.

Paxson, Vern (1999). "Bro: a system for detecting network intruders in real-time". In: *Comput. Netw.* 31. DOI: 10.1016/S1389-1286(99)00112-7.

Pfaff, Ben, Justin Pettit, Teemu Koponen, Ethan J. Jackson, Andy Zhou, Jarno Rajahalme, Jesse Gross, Alex Wang, Jonathan Stringer, Pravin Shelar, Keith Amidon, and Martín Casado (2015). "The design and implementation of open vSwitch". In: *Proceedings of the 12th USENIX Conference on Networked Systems*

Design and Implementation. NSDI'15. USENIX Association. DOI: 10.5555/2789770.2789779.

Postel, J. (1981). *NCP/TCP transition plan*. RFC 801. DOI: 10.17487/RFC0801. URL: https://www.rfc-editor.org/info/rfc801.

Potharaju, Rahul and Navendu Jain (2013). "Demystifying the Dark Side of the Middle: A Field Study of Middlebox Failures in Datacenters". In: *Proceedings of the 2013 Conference on Internet Measurement Conference*. IMC '13. Association for Computing Machinery. DOI: 10.1145/2504730.2504737.

Poutievski, Leon, Omid Mashayekhi, Joon Ong, Arjun Singh, Mukarram Tariq, Rui Wang, Jianan Zhang, Virginia Beauregard, Patrick Conner, Steve Gribble, Rishi Kapoor, Stephen Kratzer, Nanfang Li, Hong Liu, Karthik Nagaraj, Jason Ornstein, Samir Sawhney, Ryohei Urata, Lorenzo Vicisano, Kevin Yasumura, Shidong Zhang, Junlan Zhou, and Amin Vahdat (2022). "Jupiter evolving: transforming google's datacenter network via optical circuit switches and software-defined networking". In: *Proceedings of the ACM SIGCOMM 2022 Conference*. SIGCOMM '22. Association for Computing Machinery. DOI: 10.1145/3544216.3544265.

Qu, Long, Chadi Assi, Khaled Shaban, and Maurice Khabbaz (2016). "Reliability-aware service provisioning in NFV-enabled enterprise datacenter networks". In: *2016 12th International Conference on Network and Service Management (CNSM)*. DOI: 10.1109/CNSM.2016.7818411.

Rajagopalan, Shriram, Dan Williams, and Hani Jamjoom (2013). "Pico Replication: A High Availability Framework for Middleboxes". In: *Proceedings of the 4th Annual Symposium on Cloud Computing*. SOCC '13. Association for Computing Machinery. DOI: 10.1145/2523616.2523635.

Rajagopalan, Shriram, Dan Williams, Hani Jamjoom, and Andrew Warfield (2013). "Split/Merge: System Support for Elastic Execution in Virtual Middleboxes". In: *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*. nsdi'13. USENIX Association.

Roberts, L.G. (1978). "The evolution of packet switching". In: *Proceedings of the IEEE* 66. DOI: 10.1109/PROC.1978.11141.

Roesch, Martin (1999). "Snort - Lightweight Intrusion Detection for Networks". In: *Proceedings of the 13th USENIX Conference on System Administration*. LISA '99. USENIX Association.

Rooney, Sean (1997). "The Hollowman an innovative ATM control architecture". In: *Integrated Network Management V: Integrated management in a virtual world Proceedings of the Fifth IFIP/IEEE International Symposium on Integrated Network Management San Diego, California, USA, May 12–16, 1997*. Springer. DOI: 10.1007/978-0-387-35180-3_28.

Rybczynski, Tony (2009). "Commercialization of packet switching (1975-1985): A Canadian perspective [History of Communications]". In: *IEEE Communications Magazine* 47. DOI: 10.1109/MCOM.2009.5350364.

Saltzer, J. H., D. P. Reed, and D. D. Clark (1984). "End-to-End Arguments in System Design". In: *ACM Trans. Comput. Syst.* 2. DOI: 10.1145/357401.357402.

Samira, Chabaa, Abdelouhab Zeroual, and Jilali Antari (2010). "Identification and Prediction of Internet Traffic Using Artificial Neural Networks". In: *JILSA* 2. DOI: 10.4236/jilsa.2010.23018.

Sampaio, Lauren S. R., Pedro H. A. Faustini, Anderson S. Silva, Lisandro Z. Granville, and Alberto Schaeffer-Filho (2018). "Using NFV and Reinforcement Learning for Anomalies Detection and Mitigation in SDN". In: *2018 IEEE Symposium on Computers and Communications (ISCC)*. DOI: 10.1109/ISCC.2018.8538614.

Sanfilippo, Salvatore (2009). *Redis: Remote Dictionary Server*. https://github.com/redis/redis.

Sarrar, Nadi, Steve Uhlig, Anja Feldmann, Rob Sherwood, and Xin Huang (2012). "Leveraging Zipf's Law for Traffic Offloading". In: *SIGCOMM Comput. Commun. Rev.* 42. DOI: 10.1145/2096149.2096152.

Sax, Matthias J. (2018). "Apache Kafka". In: *Encyclopedia of Big Data Technologies*. Ed. by Sherif Sakr and Albert Zomaya. Cham: Springer International Publishing, pp. 1–8. DOI: 10.1007/978-3-319-63962-8_196-1.

Scales, Daniel, Mike Nelson, and Ganesh Venkitachalam (2010). "The design of a practical system for fault-tolerant virtual machines". In: *Operating Systems Review* 44. DOI: 10.1145/1899928.1899932.

Schweller, Robert, Ashish Gupta, Elliot Parsons, and Yan Chen (2004). "Reversible Sketches for Efficient and Accurate Change Detection over Network Data Streams". In: *Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement*. IMC '04. Association for Computing Machinery. DOI: 10.1145/1028788.1028814.

Sekar, Vyas, Norbert Egi, Sylvia Ratnasamy, Michael K. Reiter, and Guangyu Shi (2012a). "Design and Implementation of a Consolidated Middlebox Architecture". In: *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*. NSDI'12. USENIX Association. DOI: 10.5555/2228298.2228331.

— (2012b). "Design and implementation of a consolidated middlebox architecture". In: *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*. NSDI'12. USENIX Association. ISBN: 978-931971-92-8.

Sekar, Vyas, Sylvia Ratnasamy, Michael K. Reiter, Norbert Egi, and Guangyu Shi (2011). "The Middlebox Manifesto: Enabling Innovation in Middlebox Deployment". In: *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*.

HotNets-X. Association for Computing Machinery. DOI: 10.1145/2070562.2070583.

Serrano, Martin, Michael Boniface, Monique Calisti, Hans Schaffers, John Domingue, Alexander Willner, Chiara Petrioli, Federico M Facca, Ingrid Moerman, Johann M Marquez-Barja, et al. (2022). "Next generation internet research and experimentation". In: *Building the Future Internet through FIRE*. River Publishers.

Shaikh, Junaid M., Markus Fiedler, and Denis Collange (2010). "Quality of Experience from user and network perspectives". In: *annals of telecommunications - annales des télécommunications* 65. DOI: 10.1007/s12243-009-0142-x. URL: https://api.semanticscholar.org/CorpusID:14051498.

Sharafaldin, Iman, Arash Habibi Lashkari, and Ali Ghorbani (2018). "Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization". In: *International Conference on Information Systems Security and Privacy*. DOI: 10.5220/0006639801080116.

Sherry, Justine, Peter Xiang Gao, Soumya Basu, Aurojit Panda, Arvind Krishnamurthy, Christian Maciocco, Maziar Manesh, João Martins, Sylvia Ratnasamy, Luigi Rizzo, and Scott Shenker (2015). "Rollback-Recovery for Middleboxes". In: *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*. SIGCOMM '15. Association for Computing Machinery. DOI: 10.1145/2785956.2787501.

Sherry, Justine, Shaddi Hasan, Colin Scott, Arvind Krishnamurthy, Sylvia Ratnasamy, and Vyas Sekar (2012). "Making Middleboxes Someone Else's Problem: Network Processing as a Cloud Service". In: *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*. SIGCOMM '12. Association for Computing Machinery. DOI: 10.1145/2342356.2342359.

Sherry, Justine and Sylvia Ratnasamy (2012). *A Survey of Enterprise Middlebox Deployments*. Tech. rep. URL: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2012/EECS-2012-24.html.

Sivaraman, Vibhaalakshmi, Srinivas Narayana, Ori Rottenstreich, S. Muthukrishnan, and Jennifer Rexford (2017). "Heavy-Hitter Detection Entirely in the Data Plane". In: *Proceedings of the Symposium on SDN Research*. SOSR '17. Association for Computing Machinery. DOI: 10.1145/3050220.3063772.

Sterbenz, James P. G., David Hutchison, Egemen K. Çetinkaya, Abdul Jabbar, Justin P. Rohrer, Marcus Schöller, and Paul Smith (2010). "Resilience and Survivability in Communication Networks: Strategies, Principles, and Survey of Disciplines". In: *Comput. Netw.* 54. DOI: 10.1016/j.comnet.2010.03.005.

Sterbenz, James PG, Rajesh Krishnan, Regina Rosales Hain, Alden W Jackson, David Levin, Ram Ramanathan, and John Zao (2002). "Survivable mobile wireless

networks: issues, challenges, and research directions". In: *Proceedings of the 1st ACM workshop on Wireless security*. DOI: 10.1145/570681.570685.

Stouffer, Keith, Joe Falco, and Karen Kent (2006). "Guide to supervisory control and data acquisition (SCADA) and industrial control systems security". In: *NIST Special Publication* 800. DOI: 10.6028/NIST.SP.800-82r2.

Sun, Runyuan, Bo Yang, Lizhi Peng, Zhenxiang Chen, Lei Zhang, and Shan Jing (2010). "Traffic classification using probabilistic neural networks". In: *2010 Sixth International Conference on Natural Computation*. Vol. 4. DOI: 10.1109/ICNC.2010.5584648.

Systems, Cisco (1998). *Port Aggregation Protocol*. URL: https://www.ieee802.org/3/trunk_study/april98/finn_042898.pdf.

Tank, Gunjan P, Anmol Dixit, and Alekhya Vellanki (2017). "Software Defined Networks: The New Norm for Networks". In: URL: https://api.semanticscholar.org/CorpusID:53052362.

Taymans, Wim (1999). *Gstreamer - Open source multimedia framework*. URL: https://gstreamer.freedesktop.org/.

Team, Netfilter Core (1998). *Iptables*. URL: https://www.netfilter.org/projects/iptables/index.html (visited on 1998).

Tennenhouse, D.L., J.M. Smith, W.D. Sincoskie, D.J. Wetherall, and G.J. Minden (1997). "A survey of active network research". In: *IEEE Communications Magazine* 35. DOI: 10.1109/35.568214.

Thomson, Martin (2021). *Version-Independent Properties of QUIC*. RFC 8999. DOI: 10.17487/RFC8999. URL: https://www.rfc-editor.org/info/rfc8999.

Thomson, Martin and Sean Turner (2021). *Using TLS to Secure QUIC*. RFC 9001. DOI: 10.17487/RFC9001. URL: https://www.rfc-editor.org/info/rfc9001.

Vlacheas, P, V Stavroulaki, P Demestichas, S Cadzow, S Gorniak, and D Ikonomou (2011). "Ontology and taxonomies of resilience, version 1". In: *Proc. Eur. Netw. Inf. Secur. Agency (ENISA) Rep.*

VMware (2023). *Utility Substation Virtual Protection, Automation, and Control (vPAC) Ready Infrastructure*. URL: https://docs.vmware.com/en/VMware-Edge-Compute-Stack/services/utility-substation-vpac-ready-infrastructure.pdf.

Wang, An, Yang Guo, Fang Hao, T.V. Lakshman, and Songqing Chen (2014). "Scotch: Elastically Scaling up SDN Control-Plane Using VSwitch Based Overlay". In: *Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies*. CoNEXT '14. Association for Computing Machinery. DOI: 10.1145/2674005.2675002.

Wang, Chengwei, Oliver Spatscheck, Vijay Gopalakrishnan, and David Applegate (2016). "Toward High-Performance and Scalable Network Functions Virtualization". In: *IEEE Internet Computing* 20. DOI: 10.1109/MIC.2016.111.

Wang, Chengwei, Oliver Spatscheck, Vijay Gopalakrishnan, Yang Xu, and David Applegate (2016). "Toward High-Performance and Scalable Network Functions Virtualization". In: *IEEE Internet Computing* 20. DOI: 10.1109/MIC.2016.111.

Wang, Hao, Haiyong Xie, Lili Qiu, Yang Richard Yang, Yin Zhang, and Albert Greenberg (2006). "COPE: traffic engineering in dynamic networks". In: *SIGCOMM Comput. Commun. Rev.* 36. ISSN: 0146-4833. DOI: 10.1145/1151659.1159926.

Wang, Mowei, Yong Cui, Xin Wang, Shihan Xiao, and Junchen Jiang (2018). "Machine Learning for Networking: Workflow, Advances and Opportunities". In: *IEEE Network* 32. DOI: 10.1109/MNET.2017.1700200.

Wang, Zhaoguang, Zhiyun Qian, Qiang Xu, Zhuoqing Mao, and Ming Zhang (2011a). "An Untold Story of Middleboxes in Cellular Networks". In: *Proceedings of the ACM SIGCOMM 2011 Conference*. SIGCOMM '11. Association for Computing Machinery. DOI: 10.1145/2018436.2018479.

— (2011b). "An Untold Story of Middleboxes in Cellular Networks". In: *SIGCOMM Comput. Commun. Rev.* 41. DOI: 10.1145/2043164.2018479.

White Paper, NFV (2012). "Network Functions Virtualisation: An Introduction, Benefits, Enablers, Challenges and Call for Action. Issue 1".

Worster, Tom, avri doria, Fiffi A. Hellstrand, and Kenneth Sundell (2002). *General Switch Management Protocol (GSMP) V3*. RFC 3292. DOI: 10.17487/RFC3292. URL: https://www.rfc-editor.org/info/rfc3292.

Xu, Dahai, Yizhi Xiong, Chunming Qiao, and Guangzhi Li (2004). "Failure protection in layered networks with shared risk link groups". In: *IEEE network* 18. DOI: 10.1109/MNET.2004.1301021.

Xu, Jielong, Jian Tang, Kevin Kwiat, Weiyi Zhang, and Guoliang Xue (2012). "Survivable Virtual Infrastructure Mapping in Virtualized Data Centers". In: *2012 IEEE Fifth International Conference on Cloud Computing*. DOI: 10.1109/CLOUD.2012.100.

Yang, Lily, Todd A. Anderson, Ram Gopal, and Ram Dantu (2004a). *Forwarding and Control Element Separation (ForCES) Framework*. RFC 3746. DOI: 10.17487/RFC3746. URL: https://www.rfc-editor.org/info/rfc3746.

— (2004b). *Forwarding and Control Element Separation (ForCES) Framework*. RFC 3746. DOI: 10.17487/RFC3746. URL: https://www.rfc-editor.org/info/rfc3746.

Yang, Tong, Jie Jiang, Peng Liu, Qun Huang, Junzhi Gong, Yang Zhou, Rui Miao, Xiaoming Li, and Steve Uhlig (2018). "Elastic Sketch: Adaptive and Fast Network-Wide Measurements". In: *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. SIGCOMM '18. Association for Computing Machinery. DOI: 10.1145/3230543.3230544.

Yu, Hongfang, Vishal Anand, Chunming Qiao, and Gang Sun (2011). "Cost Efficient Design of Survivable Virtual Infrastructure to Recover from Facility Node

Failures". In: *2011 IEEE International Conference on Communications (ICC)*. DOI: 10.1109/icc.2011.5962604.

Yu, Hongfang, Chunming Qiao, Vishal Anand, Xin Liu, Hao Di, and Gang Sun (2010). "Survivable Virtual Infrastructure Mapping in a Federated Computing and Networking System under Single Regional Failures". In: *2010 IEEE Global Telecommunications Conference GLOBECOM 2010*. DOI: 10.1109/GLOCOM.2010.5683951.

Yu, Hongfang, Chunming Qiao, Jianping Wang, Lemin Li, Vishal Anand, and Bin Wu (2014). "Regional failure-resilient virtual infrastructure mapping in a federated computing and networking system". In: *Journal of Optical Communications and Networking* 6. DOI: 10.1364/JOCN.6.000997.

Yu, Minlan, Lavanya Jose, and Rui Miao (2013). "Software Defined Traffic Measurement with OpenSketch". In: *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*. nsdi'13. USENIX Association. DOI: 10.5555/2482626.2482631.

Zilberman, Noa, Yury Audzevich, Georgina Kalogeridou, Neelakandan Manihatty-Bojan, Jingyun Zhang, and Andrew Moore (2015). "NetFPGA: Rapid Prototyping of Networking Devices in Open Source". In: *SIGCOMM Comput. Commun. Rev.* 45. DOI: 10.1145/2829988.2790029.

Zilberman, Noa, Philip M. Watts, Charalampos Rotsos, and Andrew W. Moore (2015). "Reconfigurable Network Systems and Software-Defined Networking". In: *Proceedings of the IEEE* 103. DOI: 10.1109/JPROC.2015.2435732.