Received: Added at production | Revised: Added at production | Accepted: Added at production

DOI: xxx/xxxx

RESEARCH ARTICLE

# REMEDIATE: Improving Network and Middlebox Resilience with Virtualisation

**Lyn Hill** | **Charalampos Rotsos** | **Chris Edwards** | **David Hutchison**

Computing and Communications, Lancaster University, Lancashire, United Kingdom

**Correspondence**
Lyn Hill
Email: l.hill4@lancs.ac.uk

## Abstract

The increasing demand for low-latency, high-bandwidth connectivity has introduced novel challenges to delivering strong resilience guarantees in production network environments. Closed hardware platforms, known as middleboxes, that lack visibility and support for state retention remain a key challenge for continuous service delivery during network failures. These middleboxes rarely employ recovery mechanisms of their own, inspiring renewed interest in the field of NFV in recent years due to this gap within the industry. The increasing availability of VNF capabilities in modern infrastructures offers an opportunity to exploit the flexibility of software and use hybrid architectures to improve resilience. REMEDIATE is a high-availability service that propagates state between unmodified hardware middleboxes and backup PNF or VNF appliances. The platform utilizes targeted packet mirroring to allow network devices to organically construct equivalent state and thus allow an easy transition between hardware and software. To demonstrate its viability, we have evaluated REMEDIATE against a wide range of common hardware middlebox use cases built using multiple open-source packet processing frameworks. Results show upwards of 90% matching state with no observable delay to normal traffic or impact on its functionality.

**KEYWORDS**
Networking, middlebox, resilience, fault-tolerance, NFV, VNF

## 1 | INTRODUCTION

Internet protocol design adopts a stateless design approach for the lower network layers in order to allow recovery from a wide range of failure scenarios, including packet loss, congestion, and link and hardware failures, which operationally are inevitable at all levels[1]. Redundancy is one of the most popular approaches to enhance network resilience against hardware and software failures. However, the stateful design of middleboxes limits the effectiveness of redundancy. Middleboxes maintain cross-layer protocol state to speed up protocol operations and improve scalability[2]. As a result, they increase system inter-dependence in the network and violate both the end-to-end and survivability Internet principles[3].

Hardware-accelerated middleboxes rarely offer any internal mechanisms for fault recovery or stateful failover to redundant devices[4]. Building distributed 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 latency as the upper ceiling[5] for per-packet incurred delays. Middlebox design further compounds the problem by their nature: inexpensive and replaceable hardware devices built for reliability and performance, not observation or modification. The scale of deployment[6 7] for these devices leaves simple redundancy as the only option; both a costly and imperfect solution. Despite their low cost per unit, the high volume necessary[8] for both the primary and redundant hardware as well as the difficulty in their configuration[9] and lack of standardized design[10] makes it very difficult for networks to replace these boxes where required, contrary to their intention as easily replaceable hardware solutions. This technique only ensures eventual service recovery, cannot recover lost state between middlebox instances, and prevents long-lasting service degradation.

**Abbreviations:** NFV, Network Function Virtualisation; IDS, Intrusion Detection System; TCAM, Ternary Content-Addressable Memory

  

Traffic cloning to redundant devices is an effective but expensive solution that requires extended link upgrades; unlike software network functions, which commonly use virtual networking on a single node[11], redundant hardware middleboxes are unlikely to be within the same location. In recent years, the network community adopted the Network Function Virtualisation (NFV)[12] paradigm: replacing hardware-accelerated network functions with software running on general-purpose servers. The software nature of virtual network functions (VNF) can support distributed system techniques to improve beyond hardware redundancy, by replicating state during failures. Prior research[13 5] has explored several VNF state recreation methods, including replaying, non-replaying, checkpointing, and live replication. Unlike VNFs, modifying the firmware of black-box middleboxes is impossible. This is often a pivotal motivation to replace middleboxes with VNFs; unfortunately, the significant performance gap between software and hardware dissuades this pursuit[14]. VNF instances have a considerable performance gap against specialized ASICs, even when using fast packet processing frameworks[15], limiting their applicability in performance-critical use cases, like carrier-grade NAT. Hardware middlebox usage is extensive in networks, which have grown to rely on their high performance for the sake of competitiveness and service level agreements, leaving research to develop new mechanisms that may improve resilience without sacrificing these performance gains.

This paper presents REMEDIATE (REsilient MiddlEbox Defence Infrastructure ArchiTEcture), a novel resilience architecture that enables stateful failover for unmodified software and hardware middleboxes, with support for varying device openness levels. It builds upon our prior publications Katoptron[16] and Middlebox Minions[17], expanding primarily on the work of the former. For devices offering extended logging, REMEDIATE translates log output to VNF state stored in an external database. The state can be used to quickly spin up a hot middlebox replica for appliances with partial openness. Furthermore, an efficient live state recreation mechanism allows a backup middlebox instance to reconstruct the operational state of a middlebox by processing a sample of the live network traffic. The specific contributions of this paper are summarised below:

- **High-level resilience framework:** REMEDIATE is a backward-compatible resilience architecture for grey and black-box middleboxes that uses production orchestration and cloud management services to establish persistent state across replicas through the deployment of state-preserving mechanisms.
- **State-preserving mechanisms:** A selection of external state preservation mechanisms can 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 can target both software and hardware middleboxes. This state preservation can ensure state persistence across failovers, regardless of source or technology, and can be distributed to any number of replicas of identical or differing technologies.
- **Accurate and efficient:** REMEDIATE is both highly accurate and low impact on failure-free operations. Depending on the mechanism, they can provide up to 90% of a target middlebox' state using only 1 to 1.5% of the original traffic.
- **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 redundant 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 maximize its usability.

The rest of this paper is organized as follows: We elaborate on middlebox deployment practices and present relevant research efforts on middlebox resilience (§ 2). This is followed by a presentation of the design of both the proposed system and the prototype setup (§ 3) and a performance evaluation of the framework (§ 7). Finally, we conclude our work (§ 8).

## 2 | BACKGROUND

A critical requirement for network environments is the ability to recover from network failures, including black-box middlebox failures. The section discusses the resilience challenges for blackbox middleboxes[18] and elaborates on the current state of the art.

## 2.1 | Middlebox design

RFC 3234[19] defines a middlebox as "any intermediary device performing functions other than normal standard functions of an IP router on the datagram path between a source host and destination host". and include functions like WAN optimisers, firewalls, and IDS[20,21]. Their popularity steadily increases in production networks[20] because they allow operators to improve network performance and support new capabilities with minimal infrastructure investment. A survey[21] across 57 networks of varying sizes indicates that an average of 1946 middleboxes operate in a production network, while hundreds of middleboxes have been observed to operate in more than 1000 Internet ASes[6]. Middleboxes are an essential tool to evolve network protocol functionality at scale. A key benefit aspect of middleboxes is their simple deployment model; sold as complete units, they operate as a bump-on-the-wire, and their configuration does not depend on the overall network configuration. Treated as a product first and foremost, companies can replace devices to address their evolving operational needs. Bespoke ASICs can support many orders of magnitude higher packet processing rates than VNFs. Their many benefits shape the design of network infrastructures, but they are not without their flaws. There are a number of issues associated with their use, both for the end-user and the wider network infrastructure. To begin with, middlebox standards, like the ETSI NFV-MANO[22] and OpenConfig YANG models[23,24], focus on standards for device configuration, and we lack common deployment and design standards, which limit the ability to design generic mechanisms that improve maintenance and simplify operation *et al.*[21]. Configuration variability requires the development of bespoke management systems; 33% of faults that occur with middleboxes are due to misconfiguration, as well as policy errors rooted in poor interoperability[9].

Furthermore, middleboxes break the end-to-end Internet principle. The nature of the interference is dependent 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[2]. The impact on the end-to-end Internet principle is important for network resilience. Middleboxes maintain an extended network state, which has a detrimental impact on established flows during middlebox failures. In parallel, although middleboxes have a low cost per unit, operators must deploy a significant number of middlebox instances (>1000) to support new functionalities[6], magnifying the problem drastically. Despite the design intention as replaceable hardware, 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 the 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. This creates a unique problem: middleboxes are simultaneously replaced too often and not often enough. The failure of the end-to-end principle was in large part motivated by this pursuit of performance, shifting the functionality further onto the communication medium 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 performance levels and retain a competitive edge. Redundancy is the oldest and most simplistic form of resilient design[25], utilising multiple copies of a point of failure to minimise potential inaccessibility and downtime. For example, a WAN can limit the impact of link failures and flash crowds by overprovisioning spare link capacity, which more than doubles the operational cost during normal operation[4]. 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 for 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. State in the context of middlebox hardware typically refers to the live state maintained by the firmware (e.g., NAT translation) and the content of its TCAM and lookup tables. As hardware, they are far less susceptible to fault than software and are typically perceived as high-performance and reliable. Their blackbox design limits visibility in the internal state. This blackbox nature heavily contributes to resilience and operation issues with middleboxes, far more than transient network or hardware failure. A large-scale study[9] 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. More importantly, the study reports that middlebox redundancy is ineffective in 33% of cases for load balancers and firewalls. 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[26], with long-term failures reportedly very costly for both the industry and middlebox vendors[27]. The blackbox nature of this hardware greatly hinders the range of options for resilience and state propagation as a whole. With the scope of the problem established, there are several clear limitations to expanding blackbox resilience. Sacrificing performance with the

intent of improving recovery is inhibited by their widespread and ingrained presence, as well as their closed nature, preventing modification and awareness of internal logic. With this in mind, any remediation approach proposed must abide by a strict set of criteria: (i) No modification or replacement of the target platform; (ii) minimise overhead incurred to limit the effect on failure-free operations; (iii) guarantee acceptable correctness of recovery; and (iv) be sufficiently generic to apply to all stateful functions executed in hardware.

## 2.2 | State of the art

Related research focuses primarily on the issue of state preservation for software middleboxes and delivers added-value guarantees, like state correctness or reduced forwarding latency overheads. This has produced promising advances in the resilience of software middleboxes and NFV, but it cannot easily apply to blackbox middleboxes. Research in this field is summarised in Table 1.

| | **Benefits** | **Drawbacks** |
|---|---|---|
| **Replaying** | | |
| VM Capture [28] | Perfect state capture | Expensive, slow |
| Packet capture [13] | No missing flows | Delay, Non-deterministic |
| Coarse logging exreplay | Deterministic execution | Delay to regular traffic |
| Fine logging [5] | 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 1** Comparison of state preservation methods from both past research and current practices, organised into replaying and non-replaying techniques

Early efforts in VNF resilience exploited native virtualisation support in production cloud technologies to enable replication for software middleboxes. Remus [28] utilises built-in VM checkpointing capabilities in XEN [29] to snapshot its current memory at fixed intervals, offering weak deterministic execution guarantees. The system supports VMs exclusively and incurs significant processing overheads. Its design is one of the first generic attempts to persist the state of unmodified software middleboxes.

Relevant research efforts have also explored the development of VNF frameworks that reliably persist middlebox flow state using packet capture and duplication to redirect network flow between middlebox replicas. Building on the earlier Split/Merge framework [30], Pico [13] achieves VNF resilience by developing a flow persistence API that decouples states from the middlebox processing logic. Flows are identified as the source of the state itself, and initial packets are duplicated and buffered to be sent to multiple replicas. This approach improves state availability and incurs low computation overheads, but active state sharing increases synchronisation latency and requires application modifications. Log-based checkpointing records and replays non-deterministic instruction events (e.g., interrupts), recording events at the system-level, and mandating direct awareness of internal operations. SMP-Revirt [31] is one of the first examples of this highly efficient but invasive approach. FTMB [5] uses a similar approach, recording all processor events and packets that trigger them. As a replaying approach, the delay incurred by the initial startup of a recovery VM averages 275 ms, while checkpointing during failure-free operation averages only 30 microseconds overhead. This log-based checkpointing approach, while efficient, must be accommodated to its target through modification of the platform and is only feasible with software-based variants.

Past work in this area, such as CHC [32] and Reinforce [33], focusses on SFCs and how the state can be altered by any member of the chain. Reinforce's approach utilises a similar understanding of deterministic and non-deterministic actions, like SMP-Revirt, relying on programmer annotation to indicate non-deterministic state operations. 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. This checkpointing approach is costly and scales poorly with both the frequency of rate and length of the chain, but its lazy replication approach is a useful demonstration of how state is initialised by

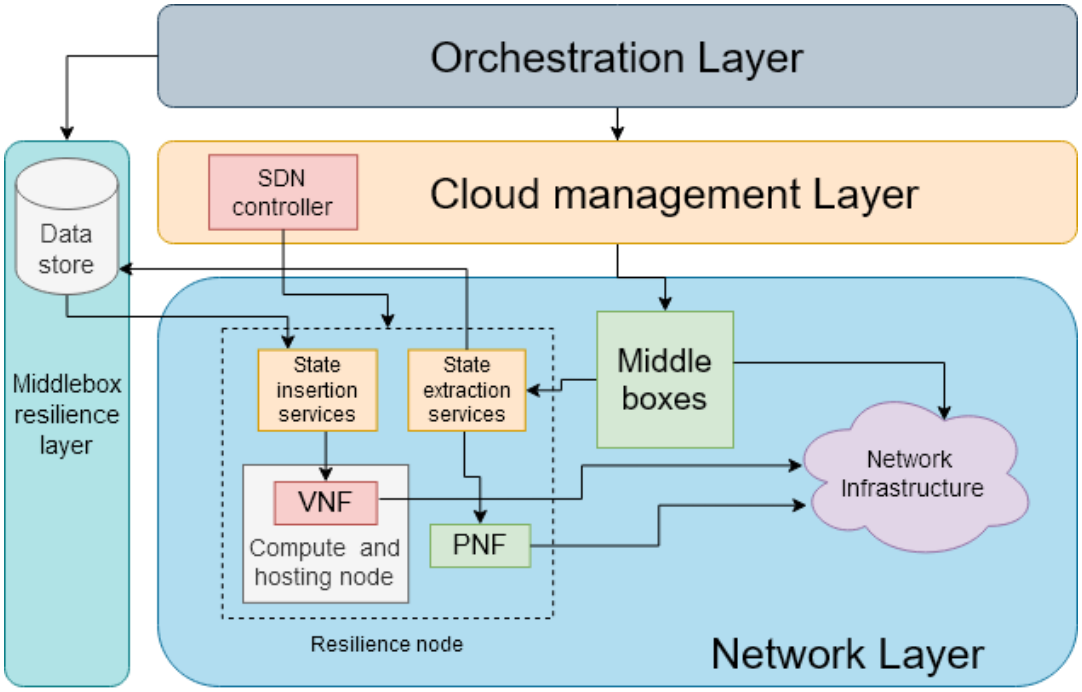


**FIGURE 1** High-level architecture overview. The middlebox resilience layer serves as the centralised logic of the framework, utilising network orchestration and cloud management services to operate the state extraction mechanisms and VNFs within the network.

the beginnings of flows and how performance can be achieved without direct modification of the target platform or per-thread logging.

## 3 | DESIGN

Given our understanding of the nature of state in blackbox middleboxes, as well as the lack of significant enforcement for consistency in software blackboxes in industry, a new argument and approach to resilience may be viable. This paper 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 areas of both research and real-world network adoption where it has otherwise languished. Past research discussed is exclusively applicable to software rather than hardware, but many of the principles remain the same; non-determinism, latency, and correctness of recovery are equally important concerns for the resilience of hardware. REMEDIATE aims to address four major challenges to hardware middlebox resilience. Firstly, it must be compatible with a wide range of unmodifies production middleboxes. Secondly, it must minimize the use of software during remediation, to limit reduced forwarding performance. Thirdly, the system should scale resource to match hardware performance when using VNFs. Finally, an effective resilience architecture should support not only failure remediation but also provide a mechanism for service recovery. This section provides an analysis of the middlebox landscape (§ 3.1, followed by a presentation of REMEDIATE (§ 3.2.

### 3.1 | Middlebox Taxonomy

As discussed in the previous section, the middlebox ecosystem is rather wide, and appliance openness varies across vendors. Device openness has a direct impact on the efficiency of any proposed architecture. In the context of our REMEDIATE architecture, we consider the following three middlebox types:

- **Whitebox middleboxes**
  Whiteboxes are devices that offer direct state visibility and programmability based on open models and interfaces. Not

all VNFs popularised in businesses are distributed in a binary format with licensing schemes. Open source efforts, like the OpenConfig initiative[34], develop standardised APIs and open tools for managing network devices. Programmability allows for modification and insertion of drivers capable of manipulating forwarding table entries or similar state. Whitebox middlebox examples include OpenFlow or P4 switches. This type of devices can support lossless service remediation with negligible performance impact.

- **Greybox middleboxes**
  Greybox moddleboxes offer observable state outputs, such as logging or diagnostics, and provide approximate information about the internal forwarding state. External drivers (*e.g.,* log parser) can recreate an operationally equivalent state in a replica VNF by processing device outputs. An example of this is interpreting the log output of a blackbox firewall and using external read and write handles of a Click[35] element to directly modify its state using the hotconfig mechanism, allowing state to effectively be transferred between routers during liveness.
- **Blackbox middleboxes**
  Blackbox middleboxes employ extensively specialized hardware that lacks any state visibility. State replication must exploit the deterministic nature of hardware design, and state replication is achievable using traffic cloning and replication. Supporting service remediation for such devices is difficult, requiring investment in redundant equipment. Traffic cloning is an equally resource-expensive operation (doubling the in-network traffic volume). To minimise the impact of this approach, traffic cloning must be optimised using traffic sampling, tailored to the oepration realised by the middlebox, that exploits the observation that middlebox state is formed based on the first flow packets. Cloning only the first few packets of each flow to pre-populate the state of a redundant middlebox offers an excellent remediation strategy that reduces service degradation during failures. 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 vary from middlebox to middlebox, although all within the range of a handful of packets.

## 3.2 | REMEDIATE Architecture

REMEDIATE (REsilient MiddlEbox Defence Infrastructure ArchiTEcture) is a generic state recovery system for unmodified middleboxes. Figure 1 presents a high-level overview of the system components. The system is organized into 3 layers: The infrastructure, which includes the network and compute resources of the network; the management, which contains the services used by the operators to configure the infrastructure; and the middlebox resilience, which includes the services supporting the REMEDIATE functionality. The middlebox resilience layer interfaces with existing orchestration and cloud management services and ensures that the state-preserving mechanisms discussed in Section 4 establish a persistent state between primary and redundant instances.

### 3.2.1 | Resilience Node

The resilience node is a network element that pairs primary middleboxes with virtual or physical redundant middleboxes, aided by state-preserving mechanisms as proposed above. It delivers a bi-directional stateful failover mechanism that synchronizes the state between the primary and the redundant middlebox without replacing the existing/underlying infrastructure. The resilience node abstracts the different remediation techniques and its implementation depends on the openness of the primary middlebox. 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 offers 3 functionalities: state extraction, state insertion and redundancy. The first two functionalities ensure the middlebox state is synchronized with the data store, while the redundancy functionality is responsible to ensure smooth traffic transition between primary and redundant middleboxes during failures.

The components of the resilience node are depicted in Figure 1. A redundant middlebox could be a VNF hosted in compute infrastructure or implemented physically as a PNF. The VNFs, used for filtering and traffic redirection, are managed by the network-wide SDN controller. PNFs are more static, and we use agents (e.g., juju actions) for monitoring and configuration. 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, 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 remove direct interaction from the primary to the redundancy; 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.

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 its viability. 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 using a state repository instance.

### 3.2.2 | State Repository

The state repository is a service that stores whitebox and greybox middlebox state in a logically centralised location. Bandwidth is an important metric in network operations, especially regarding its use from a business perspective. In principle, bandwidth capacity can scale infinity as the volume of traffic grows, but in practice, there are physical and economic limits that must be considered. Minimising the increase in bandwidth use and latency incurred for state preservation by 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 here, 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. State transfer may vary in its scope and direction, with the number of entities on either side of the primary:redundancy balance highly flexible.

### 3.2.3 | REMEDIATE - Middlebox Resilience layer

The logic and core operations of REMEDIATE are represented by the external middlebox resilience layer presented in Figure 1. 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 controllers 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 is represented as an external datastore, incorporated into the resilience layer, and operates independently of direct control beyond its initial setup. 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.

# 4 | IMPLEMENTATION

In the previous section, a high-level design for REMEDIATE presented 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. This section details the implementation of both the software-targeted state extract system "Middlebox Minions (MiMi)" and the hardware-targeted state recreation filtering system "Katoptron".

## 4.1 | 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[36] or a P4 switch[37] 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 FRR paths with RSVP-TE signaling[38]. These approaches offer different trade-offs with respect to hardware requirements and failure detection guarantees. More advanced mechanisms, such as heartbeat monitors[39], 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[40] or LACP[41].

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.

## 4.2 | Points of Failover proof of concept

The two mechanisms both utilise OpenFlow's inbuilt group tables to enable failover using a set of static links to be swapped to in a progressive chain in case of link breakage/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.

# 5 | WHITE AND GREYBOX RESILIENCE

Middlebox Minions (MiMi) is a VNF resilience framework that provides methods for white and greybox stateful failover, as well as the scalable state distribution for the overall REMEDIATE framework. 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[42] 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[38]. The architecture is divided into three layers: management, drivers and middleboxes/VNFs. Firstly, the management layer orchestrates all of the operations of the VNFs, including liveness and failover. The management 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 paper, 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 third-party storage presented 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.

## 5.1 | State drivers

The state drivers are the means by which state is extracted or generated for the redundant VNFs and are 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.

### 5.1.1 | Log interpretation

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 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 are 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[43], 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. This mechanism, whilst fitted, is fairly simplistic to implement 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 our implementation and evaluation, we focus on OpenFlow and NetFilter[44] 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.

### 5.1.2 | 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[45] and P4[37] 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.

## 5.2 | 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 limit any need to modify or interfere in its operations, to allow for a scalable and centralised distribution point for the redundant replicas and 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.

# 6 | BLACKBOX RESILIENCE

Katoptron[16] (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

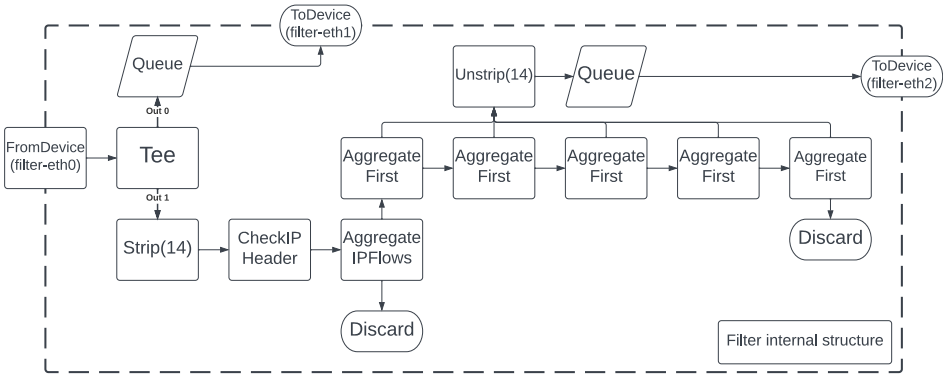


**FIGURE 2** Five packet aggregator filter in Click

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, while dissuading utilising circuit space on pseudo-random generation or system clocks and instead deriving state from received input. This greatly minimises potential non-determinism, allowing Katoptron to exploit a "lazy correctness" approach highlighted by Reinforce[33] 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 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.

## 6.1 | 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. The 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. 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 be mildly stateful and maintain awareness of flows that it is monitoring. An example of a filter implemented in Click is presented in Figure 2.

# 7 | EVALUATION

This section evaluates the designs of the resilience framework and the implementation of the two mechanisms prototype implementations. Firstly, we describe the topologies and workload used to evaluate the resilience of each separate project, with further details in their respective sections. Secondly, the filter approach of Katoptron is evaluated in multiple implementation technologies and multiple target middlebox functions. Finally, 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.

## 7.1 | Testbed environment

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. These experiments were executed on a Dell server (dual socket Xeon 4114, 20 cores, 32 GB of RAM, Ubuntu 18.04) using the Mininet platform.

| 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 2** Minimum sampling rates for state determined through simple experimentation

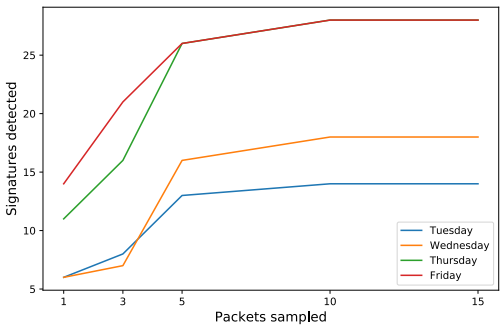


**FIGURE 3** Impact of traffic sampling on detected signatures in the CICID2017 dataset.

The testbed environment 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. Our strawman implementation uses Mininet[46], a network emulation tool that allows for rapid testing and development of SDN technologies, like OpenFlow[36]. We developed two Mininet topologies to test the two remediation approaches. The MiMi topology uses a singular topology with a client/server network split into separate subnets, joined by a load balancer. The Katoptron topology 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.

## 7.2 | Workloads

The evaluation utilises two common workload models, short and long flows, to emulate typical Internet traffic workloads. Katoptron uses a third, which will be detailed last. 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, and 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). 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, we use two WRK statistics: connection errors, which occur when a TCP connection is reset and timeouts, which count the number of TCP connections that terminate due to HTTP read timeouts. 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. This workload uses MPEG-DASH streams of the "Big Buck Bunny"[47] video in the highest quality (8000 kbit) segmented in 1 second chunks, with chunk sizes varying between 100 kb and 1.4 mb and quality representations split six ways from 2500 kbit to 8000 kbit. We use Scootplayer[48] for the Mimi evaluation and a dummy gstreamer plugin that does not perform any video decoding for Katoptron to emulate the video clients. In both scenarios, the servers use the lighttpd[49] HTTP server (v.1.4.45) to emulate the server. 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 are from the CICID 2017 dataset[50] and consist of labelled network traces from a real network, split across four separate days. Each day contains a unique attack scenario: 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. 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.

## 7.3 | Blackbox remediation evaluation

This section evaluates the proposed Katoptron mechanism. 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. The Katoptron testbed is configured in the client/server model discussed previously, using static OpenFlow rules to connect end-hosts with the Katoptron service as well as route traffic between the 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 forwarded to the egress switch, which connects the two middleboxes with the server end-hosts. 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. Finally, in all experiments, we consider three experimental configurations: *Base*, which executes the experiment with no failures; *Simple*, which executes the experiment with failures and uses simple 1:1 middlebox redundancy; and *Katoptron*, which 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.

### 7.3.1 | Middlebox support

In order to demonstrate the generality of the REMEDIATE Katoptron mechanism, we evaluate against three common middlebox scenarios. These representative middleboxes are off-the-shelf unmodified network functions consisting of a NAT, an IDS and a load balancer. The chosen functions cover a wide range of traffic processing scenarios still realised using ASIC acceleration in modern networks, including packet field modifications, stateful forwarding and flow monitoring. For testing, the middlebox implementations employed in our evaluation are software-based but are treated as black-box devices, *i.e.*VNF instances are unmodified and their implementations closely match the behaviour of a hardware-accelerated device, so as not to impact the evaluation of Katoptron's generality. Where possible, we have utilised open-source tools to avoid overfitting to specific operational models such as the Suricata[51] IDS. Furthermore, the filter's implementation is not dependent on a specific technology and can be realised using several packet processing technologies, such as P4 and DPDK.

Our NAT middlebox, based on the Click modular mazu-nat program, uses a lookup table with a 5-tuple hash to connect the clients to the servers via a single IP. The NAT operates as a gateway for all 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 state consists of the lookup table entries, with their loss during a failure incurring processing overhead from the recalculation of the lookup for every lost connection. Our IDS function uses the open-source Suricata software, equipped with the open emerging threats ruleset[52]. The IDS monitors the traffic between the client and servers, operating in IPS mode (the IDS receives a copy of the active traffic) and generating alerts when traffic flows match malicious signatures. State is defined in this scenario as these threshold counters: if a counter is reset for an active connection, false negatives can occur where thresholds are no longer met but would have been, allowing malicious traffic to reach the target. Typically, rules match against flow and host statistics or apply wildcard masks on the start of the application payload. Finally, the load balancer is an OVS switch with a fixed set of OpenFlow rules to map incoming flows to a backend server using consistent hashing of the 5-tuple and weighted buckets, distributing traffic in a broadly even spread. State is defined in this scenario as the loss of the forwarding table entries, which are mildly stateful due to the weighted buckets affecting the consistent hashing placement.

Table 2 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. Sufficiently accurate can be broadly defined as state that does not significantly deviate from that of the original boxes to a degree that causes disruption in live connections. Non-determinism is one such example of this, where potential deviation will create inaccurate state that causes greater disruption than if it had simply been lost and re-established. 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 accurately fingerprint a flow. Fewer than five impair its success rate, while greater than five offered no additional benefit once signature rates are matched, as shown in Figure 3.

### 7.3.2 | 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 via duplicating SYN packets from all incoming connections to the primary box. In this experiment, we use the HTTP 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, with five runs for each setup to calculate an average. The results are presented in 3, including the average timeout and connection resets reported by WRK for the two content sizes (5KB and 627KB) and varying numbers of clients. WRK utilises non-blocking sockets and has limited tolerance

HTTP 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 |

HTTP 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 3** Average HTTP resets and timeout rates during NAT middlebox failures using the HTTP workload for both small (5KB) and large (627KB) objects served.

| | 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 4** Total number of signatures and alerts raised by the redundant Suricata IDS instance for each trace, when using the Attack workload.

for latency. Connection resets indicate TCP establishment has been delayed by retransmissions, while timeouts are reported when page transfers exceed two seconds. Both of these reported metrics are a result of packet retransmissions from loss of state, and a rise in either, especially timeouts, is indicative of failed transfer of live connections.

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, 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. 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 for 300 clients). This is well within acceptable limitations, demonstrating that in this scenario, this problem is not occurring. Short flows like those in the top half of Table 3 are only mildly stateful, but to evaluate significantly longer flows, the bottom half demonstrates that Katoptron's reduction in timeouts is consistent, if not more pronounced, on retaining active connections across redundancies. 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.

### 7.3.3 | 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. During each experimental run, we trigger 5 failures, with a gap of 15 seconds between them. Furthermore, we run each experiment 5 times and report the average signature and alert results in Table 4. 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. 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. The number of signatures detected remains identical, with an expected drop in alerts due to the disruption of long-running streams that can trigger multiple alerts per detection. The matching number of observed signatures is far more important and indicative of correct IDS detection behaviour. 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 4, it 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.

| 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** Average HTTP resets and timeouts rates during Load Balancer middlebox failures, using the HTTP workload with large content sizes (627KB).

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

**TABLE 6** Total count of buffering, resolution change and failed connections during Load Balancer middlebox failures with the streaming workload.

### 7.3.4 | Load balancer middlebox performance

The third scenario is the load balancer middlebox, where loss of state would affect the number of connection resets, akin to the NAT deployment, but with far greater dependence on state due to the weights used in the decision logic. This experimental configuration uses the service restoration mechanism configured using static rules on the ingress and egress switches. 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. The metrics presented in Table 6 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.

For the short-term streams, the difference between simple redundancy and Katoptron is more service-pronounced than NAT, due to the weighted bucket approach establishing a small measure of non-deterministic execution. With the live replay approach utilised by our filter, however, the results depicted in Table 5 show that the decision-making logic of this bucket is consistent with 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. Table 6 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 the connection at each failover, and one is 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.

## 7.4 | Greybox remediation evaluation

This section evaluates the performance of the MiMi resilience mechanism. 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.

### 7.4.1 | Experimental Setup

The experimental topology, depicted in Figure 4, consists of multiple clients accessing a load-balanced HTTP service supported by a set of servers separated on subnets. The load balancers are implemented as Ryu applications using a round-robin distribution of traffic flows. 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.

For each experiment, the results are divided into four scenarios: the base scenario without failures, a control with the mechanism enabled to observe for impact, failures without replication, and finally failures with replication enabled. These are labelled clean, copied, fail and failover, respectively. For the WEB workload, the experiment runs for five minutes with a link failure triggering every 30 seconds, totalling five state transfers. For the DASH workload, 15 failures are triggered at the same evenly

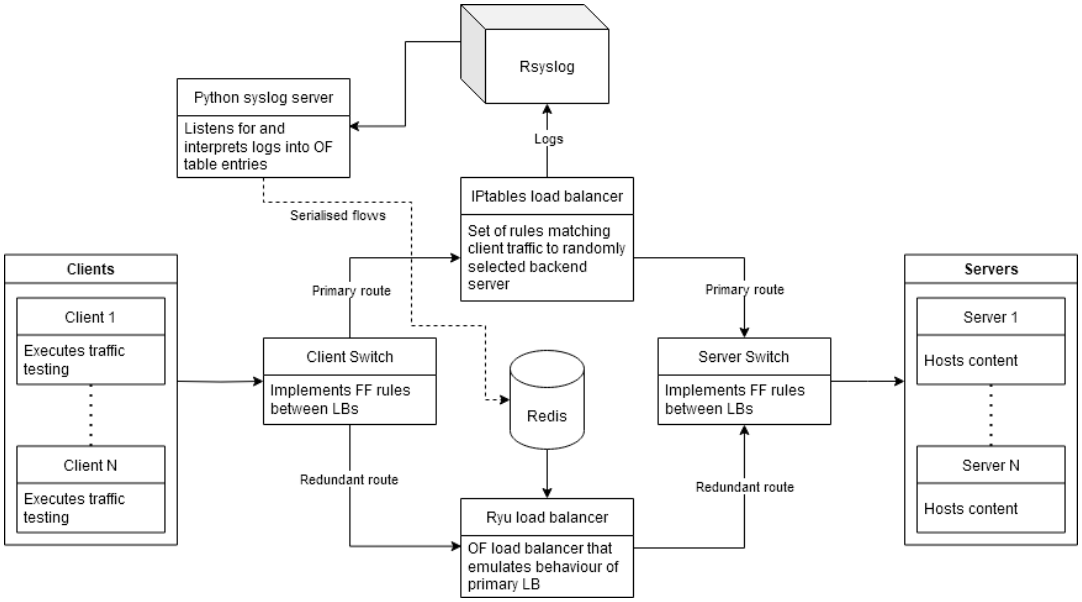


**FIGURE 4** Experiment topology showcasing the syslog variant of the evaluation, emulating a caching service and load balancer.

spaced 30-second intervals. It is worth noting that state is only mirrored in one direction, from the primary middlebox to the minion VNF. State is therefore lost when restoring performance to the original primary. 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.

### 7.4.2 | 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 3.1, 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 middleground[53]. 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 in 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.

### 7.4.3 | Direct extraction evaluation

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

*WEB workload*

The web 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. The results are depicted in Figure 5, 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 totalling 5 triggered failovers. Overall, the web workload is robust against disruption as anticipated, with two trends emerging: 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%. Timeouts incur packet retransmissions, 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 backwards compatibility of this technique.

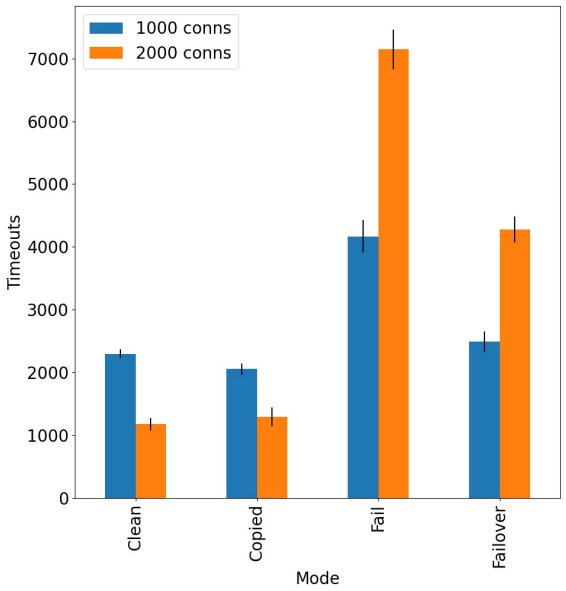


**FIGURE 5** Reported timeouts of WRK connections for the 1000 and 2000 connection datasets

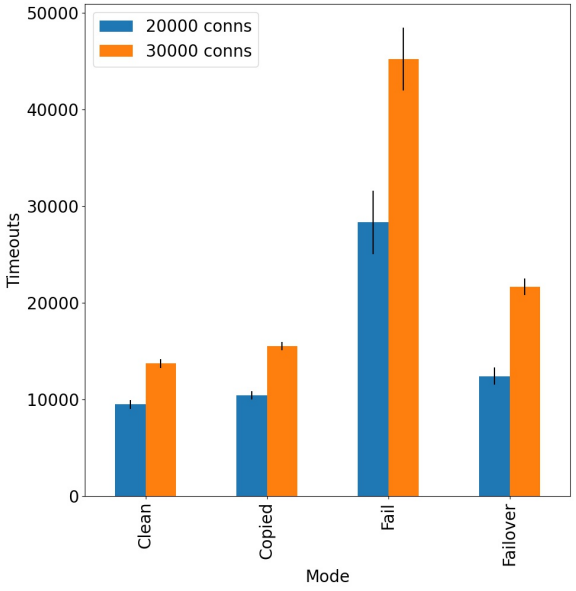


**FIGURE 6** Reported timeout rates of WRK connections for the 20,000 and 30,000 connection datasets with iptable logging

*DASH workload*

The second workload consists of MPEG-DASH video streams using Scootplayer[48], generating a small number of long-term flows where the loss of state will 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. 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 limitations of the testbed.

### 7.4.4 | 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 5.7 kb to 617 kb. 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 2.2 MB, or 2200 kb[54]. The object size increase reflects the difference between header requests and typical webpage sizes and allows us to evaluate a larger overall traffic volume, thus increasing the duration of flows and the impact of state loss. 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 6 presents the performance of the web workload for the four experimental scenarios and for a varying number of parallel connections. 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. The separation of the driver from the middlebox incurs no reduction in effectiveness and reduces the possible impact on 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 accuracy 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.

## 8 | CONCLUSIONS

Networking infrastructure is built with the understanding that failure is an inevitable part of its operation at every level, from intermittent packet loss to significant hardware faults, and must be built with a mindset focused on resilience and continuation of service alongside high-speed performance. The over-reliance on proprietary tools to deliver network features in the name of performance or security has created a vulnerability in the resilience of the infrastructure from its inability to preserve state[21]. This paper presents REMEDIATE, 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 two mechanisms from our prior work[17,16], targeting greybox software or blackbox hardware respectively, alongside a scalable distribution approach to multiple replicas and an efficient high-level state filtering mechanism. It 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.). In summary, REMEDIATE upon the existing work within the domain of resilience for middleboxes, effectively replicating 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. Our future work will examine the expansion of these techniques into the domain of intelligent orchestration and closed loop automation as 5G paves the way for an increasingly rapid adoption of SDN in modern networking.

## REFERENCES


1. Hutchison D, Pezaros D, Rak J, Smith P. On the Importance of Resilience Engineering for Networked Systems in a Changing World. *IEEE Communications Magazine*. 2023;61(11):200-206. doi: 10.1109/MCOM.001.2300057
2. Detal G, Hesmans B, Bonaventure O, Vanaubel Y, Donnet B. Revealing Middlebox Interference with Tracebox. In: IMC '13. Association for Computing Machinery 2013; New York, NY, USA:1–8
3. Saltzer JH, Reed DP, Clark DD. End-to-End Arguments in System Design. *ACM Trans. Comput. Syst.*. 1984;2(4):277–288. doi: 10.1145/357401.357402
4. Jain S, Kumar A, Mandal S, et al. B4: Experience with a Globally-Deployed Software Defined Wan. In: SIGCOMM '13. Association for Computing Machinery 2013; New York, NY, USA:3–14
5. Sherry J, Gao PX, Basu S, et al. Rollback-Recovery for Middleboxes. In: SIGCOMM '15. Association for Computing Machinery 2015; New York, NY, USA:227–240
6. Huang S, Cuadrado F, Uhlig S. Middleboxes in the Internet: A HTTP perspective. In: 2017:1-9
7. Sherry J, Ratnasamy S, At JS. A survey of enterprise middlebox deployments. tech. rep., UC Berkeley; : 2012.
8. Sekar V, Egi N, Ratnasamy S, Reiter MK, Shi G. Design and Implementation of a Consolidated Middlebox Architecture. In: NSDI'12. USENIX Association 2012; USA:24
9. Potharaju R, Jain N. Demystifying the Dark Side of the Middle: A Field Study of Middlebox Failures in Datacenters. In: IMC '13. Association for Computing Machinery 2013; New York, NY, USA:9–22

10. Sherry J, Hasan S, Scott C, Krishnamurthy A, Ratnasamy S, Sekar V. Making Middleboxes Someone Else's Problem: Network Processing as a Cloud Service. In: 2012:13–24.
11. White Paper N. Network Functions Virtualisation: An Introduction, Benefits, Enablers, Challenges and Call for Action. Issue 1. tech. rep., ETSI; : 2012.
12. Yang L, Anderson TA, Gopal R, Dantu R. Forwarding and Control Element Separation (ForCES) Framework. RFC 3746; 2004
13. Rajagopalan S, Williams D, Jamjoom H. Pico Replication: A High Availability Framework for Middleboxes. In: SOCC '13. Association for Computing Machinery 2013; New York, NY, USA
14. Wang C, Spatscheck O, Gopalakrishnan V, Applegate D. Toward High-Performance and Scalable Network Functions Virtualization. *IEEE Internet Computing*. 2016;20:10-20. doi: 10.1109/MIC.2016.111
15. Linux Foundation . Data Plane Development Kit (DPDK).; 2015.
16. Hill L, Rotsos C, Edwards C, Hutchison D. Katoptron: Efficient State Mirroring for Middlebox Resilience. In: 2024:1-9
17. Hill L, Rotsos C, Fantom W, Edwards C, Hutchison D. Improving network resilience with Middlebox Minions. In: 2022:1-5
18. Hutchison D, Sterbenz JP. Architecture and design for resilient networked systems. *Computer Communications.* 2018;131:13-21. COMCOM 40 yearsdoi: https://doi.org/10.1016/j.comcom.2018.07.028
19. Brim SW, Carpenter BE. Middleboxes: Taxonomy and Issues. RFC 3234; 2002
20. Wang Z, Qian Z, Xu Q, Mao Z, Zhang M. An Untold Story of Middleboxes in Cellular Networks. *SIGCOMM Comput. Commun. Rev.*. 2011;41(4):374–385. doi: 10.1145/2043164.2018479
21. Sherry J, Ratnasamy S. A Survey of Enterprise Middlebox Deployments. Tech. Rep. UCB/EECS-2012-24, ; : 2012.
22. ETSI . ETSI GS NFV 006 V4.4.1 (2022-12) Network Functions Virtualisation (NFV) Release 4; Management and Orchestration; Architectural Framework Specification. *Architectural Framework. sl: ETSI.* 2012.
23. Bierman A. Guidelines for Authors and Reviewers of YANG Data Model Documents. RFC 6087; 2011
24. Bierman A. Guidelines for Authors and Reviewers of Documents Containing YANG Data Models. RFC 8407; 2018
25. Sterbenz JPG, Hutchison D, Çetinkaya EK, et al. Resilience and Survivability in Communication Networks: Strategies, Principles, and Survey of Disciplines. *Comput. Netw.*. 2010;54(8):1245–1265. doi: 10.1016/j.comnet.2010.03.005
26. Allman M. On the Performance of Middleboxes. In: IMC '03. Association for Computing Machinery 2003; New York, NY, USA:307–312
27. Harris C. Data center outages generate big losses.; 2011.
28. Cully B, Lefebvre G, Meyer D, Feeley M, Hutchinson N, Warfield A. Remus: High Availability via Asynchronous Virtual Machine Replication. In: 2008.
29. Barham P, Dragovic B, Fraser K, et al. Xen and the Art of Virtualization. In: SOSP '03. Association for Computing Machinery 2003; New York, NY, USA:164–177
30. Rajagopalan S, Williams D, Jamjoom H, Warfield A. Split/Merge: System Support for Elastic Execution in Virtual Middleboxes. In: nsdi'13. USENIX Association 2013; USA:227–240.
31. Dunlap GW, Lucchetti DG, Fetterman MA, Chen PM. Execution replay of multiprocessor virtual machines. In: VEE '08. Association for Computing Machinery 2008; New York, NY, USA:121–130
32. Khalid J, Akella A. Correctness and Performance for Stateful Chained Network Functions. In: 2019
33. Kulkarni SG, Liu G, Ramakrishnan KK, Arumaithurai M, Wood T, Fu X. REINFORCE: Achieving Efficient Failure Resiliency for Network Function Virtualization Based Services. In: . 28. 2020:695-708
34. initiative O. Open Config. ; .
35. Martins J, Ahmed M, Raiciu C, et al. ClickOS and the Art of Network Function Virtualization. In: USENIX Association 2014; Seattle, WA:459–473
36. McKeown N, Anderson T, Balakrishnan H, et al. OpenFlow: enabling innovation in campus networks. *ACM SIGCOMM computer communication review.* 2008;38(2):69–74. doi: 10.1145/1355734.1355746
37. Bosshart P, Daly D, Gibb G, et al. P4: Programming Protocol-Independent Packet Processors. *SIGCOMM Comput. Commun. Rev.*. 2014;44(3):87–95. doi: 10.1145/2656877.2656890
38. Chiesa M, Sedar R, Antichi G, et al. PURR: a primitive for reconfigurable fast reroute: hope for the best and program for the worst. In: CoNEXT '19. Association for Computing Machinery 2019; New York, NY, USA:1–14
39. IBM . PowerHA® SystemMirror® cluster heartbeat over TCP. ; .
40. Systems C. Port Aggregation Protocol.; 1998.
41. Huawei . Link Aggregation Control Protocol.; 2022.
42. Atlas A, Swallow G, Pan P. Fast Reroute Extensions to RSVP-TE for LSP Tunnels. RFC 4090; 2005
43. Gerhards R. The Syslog Protocol. RFC 5424; 2009
44. Org N. The Netfilter Project.; 2004.
45. Kohler E, Morris R, Chen B, Jannotti J, Kaashoek MF. The click modular router. *ACM Trans. Comput. Syst.*. 2000;18(3):263–297. doi: 10.1145/354871.354874
46. Lantz B, Heller B. Mininet: An Instant Virtual Network on your Laptop. http://mininet.org/; 2009.
47. Benjamin Rainer CM, Timmerer C. Big Buck Bunny MPEG-DASH testing. https://dash.itec.aau.at/dash-js/; 2012.
48. Broadbent M. Scootplayer. https://github.com/broadbent/scootplayer; 2015.
49. Kneschke J. Lighttpd: a light-weight web server. https://www.lighttpd.net/; 2003.
50. Sharafaldin I, Habibi Lashkari A, Ghorbani A. Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization. In: 2018:108-116
51. Foundation OIS. Suricata - Open Source IDS/IPS.; 2022.
52. Jonkman M. Emerging Threats. ; .
53. Sanfilippo S. Redis: Remote Dictionary Server. https://github.com/redis/redis; 2009.
54. archive H. HTTP archive: page weight.; 2010.