systems

Article

SALMA: A Novel Middlebox Infrastructure System Based on Integrated Subnets

Amer AlGhadhban [1] and Ahmad Showail [2,3,*]

[1] Department of Electrical Engineering, University of Ha'il, Ha'il 55476, Saudi Arabia
[2] Department of Computer Engineering, Taibah University, Madinah 42353, Saudi Arabia
[3] Department of Computer Science, University of California, Irvine, CA 92697, USA
* Correspondence: ashowail@taibahu.edu.sa

Abstract: Middleboxes are critical components in today's networks. Due to the variety of network/security policies and the limitations of routing protocols, middleboxes are installed in multiple physical locations to face high traffic with few considerations for efficiency. Reducing the number of deployed middleboxes would reduce capital and operation costs. Moreover, some flows prefer to bypass one or more in-path middlebox where they provide useless services, such as payload compression for multimedia streams. These challenges can be partially tackled by network function virtualization (NFV) schemes with the costs of performance reduction and replacement expenses. Given the rapid growth and the wide adoption of software-defined networking solutions and the recent advances in managing middleboxes' configuration, the consolidation of middleboxes is becoming easier than before. We designed and evaluated SALMA, a new pre-NFV practical solution that systematically recreates the infrastructure of middleboxes by proposing the Integrated Middleboxes Subnets scheme. In this work, we attempted to reduce the number of installed middleboxes by implementing horizontal integration of middleboxes' functions, such as every pair of middleboxes being integrated into a dedicated hardware box. We support the motivation for creating SALMA with a practical survey of in-production middleboxes from 30 enterprises. Our solution addresses key challenges of middleboxes, including cost, utilization, flexibility, and load balancing. SALMA's performance has been evaluated experimentally as well.

Keywords: systems architecture; network infrastructure; middleboxes; software defined networking; network management system

check for updates

Citation: AlGhadhban, A.; Showail, A. SALMA: A Novel Middlebox Infrastructure System Based on Integrated Subnets. *Systems* **2022**, *10*, 165. https://doi.org/10.3390/systems10050165

Academic Editors: Zhan Shu and Pingyu Jiang

Received: 4 August 2022
Accepted: 19 September 2022
Published: 24 September 2022

1. Introduction

Middleboxes are widely adopted in broadband, enterprise networks, and lately, in data center networks. In today's networks, the number of deployed middleboxes is on par with the number of forwarding elements [1]. Several studies forecast continued rapid growth of the middlebox market; the cloud security market alone is expected to reach $11 billion by 2022 [2]. At the same time, the operators need to reduce the deployment costs of middleboxes to align their capital expenses with the current reduction of average revenue per user [3]. The topology-dependent nature of middleboxes and the inflexibility of routing protocols restrain prior practical initiatives from efficient middlebox deployment in today's networks [1,4,5].

To allow middleboxes to perform their tasks on particular flows, they must be carefully placed within the physical infrastructure of the enterprise network. Middlebox placement is usually done in an ad hoc manner supported by error-prone manual configuration. In reality, the physical locations of middleboxes are selected based on certain network and security strict-policies with little consideration for their efficiency [4]. As a result, multiple middleboxes of the same type are scattered inside the network to face in-scope traffic. By in-scope traffic, we mean flows that are required by security policies to be examined by a sequence of middleboxes.[1] These practices lead to low utilization of network resources and an increased network operation complexity in terms of management, troubleshooting, and maintenance. The resources available cannot be fully utilized despite the fact that the workloads are highly appropriate to do so. Moreover, certain classes of traffic, such as multimedia, need to bypass some of the in-path middleboxes, such as the WAN optimizer [3,6]. In today's networks, administrators recraft the network infrastructure and use manual configuration supported by tunneling methods to have a reasonable degree of control in order to manage network resources. Also, middleboxes may be misconfigured accidentally, which leads to network isolation and/or traffic policy violation. These drawbacks, among others, encourage researchers to propose a better solution to utilize network resources and augment the availability of middleboxes.

To address this shortcoming of utilization, the scattered middleboxes could be efficiently gathered in multiple centralized subnets. Middleboxes from the same type would be integrated together, forming one super middlebox. In turn, the super middlebox should be able to handle the traffic from other subnets, in addition to the traffic of its own subnet. Accordingly, we need a mechanism to forward the traffic from remote subnets to the right middlebox before it reaches its final destination. Researchers in pLayer [7] proposed the Layer-2 mechanism to route traffic through a sequence of middleboxes connected to a policy-based switch. Unfortunately, this solution did not address the issues of load balancing, utilization, and cost. Similarly, the authors of CoMb [4] proposed a method for middlebox function-level consolidation into a shared hardware box. In the same context, several other solutions have proposed moving middleboxes from the enterprise network to the cloud [8–10]. These efforts suffer from the high latency of routing the internal traffic to the cloud. In addition, they likely face serious hindrances in terms of resilience and security concerns, due to several challenges. Some examples are the hard-to-modify dedicated hardware middleboxes, the growing rigorous security practices and their concerns, and the serious resistance to deploying radical solutions. As an alternative, we propose a more practical solution that can make use of the available resources.

Network function virtualization (NFV) is a leap network technology that offers multiple features and open a wide range of development opportunities [11–17]. NFV leverages existing virtualization advances to abstract general-purpose hardware, facilitating the deployment of hardware-based network functions (NF) in a virtual environment. Although NFV offers those remarkable features, it is still intractable to maing the virtualized NFs function correctly on the ground. The NFV has multiple processing and practical limitations. The packet processing speed in modern OSes is 10 to 20 times slower than in a hardware network interface. The current advances in the speed of network interfaces and conventional switches that allow up to 100 Gbps add pressure on the NFV and software-based packet processing to cope with these speeds. At 100 Gpbs, the processing time of a small packet, 64B, is 6.7 nanoseconds, which is in order of one cycle in modern CPUs. This speed adds a burden on switch designers. Indeed, the vendors of conventional hardware switches struggled to meet this processing speed. They could barely find enough processing time to run critical switch features (e.g., flow classification). Moreover, when the NFV is used to perform the middleboxes' functions, the traffic needs to flow from NFs to another according to the service chaining policy. These NFs reside either in a single or in multiple machines. The packet forwarding between NFs in a single machine encountered the context switching challenge. Similarly, the latter case suffers from packet queuing and processing challenges, as the packets should go through multiple virtual and physical network interfaces. Besides that, the VM users encounter poor network connectivity due to virtualization faults and compatibility problems. These drawbacks and limitations hinder the adoption of NFV-based solutions in practice, as it has been shown in our survey. That is to say nothing of the cost of replacing middleboxes' hardware and software with new NFV-enabled devices.

In this work, we take a backward step to introduce a solution that lies between conventional middlebox implementations and NFV adoption which prepares the enterprise network infrastructure towards NFV. The presented solution aims at efficient repositioning of in-operation middleboxes by efficiently integrating them into multiple clusters close to their traffic flows. These clusters are partitioned into two main parts: (primary and backup). The primary clusters hold the most efficient middleboxes, in terms of QoS criteria, and others are grouped in the backup clusters. In this case, the out-of-service middleboxes can be replaced by NFV-based solutions. This practice would enable gradual migration from legacy middleboxes to NFV. Moreover, we built our solution to minimize adoption barriers in practice. The name of our novel framework is SALMA, and it is based on the software-defined networking (SDN) concept that addresses cost, utilization, availability, and operation complexity. The middleboxes from the same type, defined as peers, are integrated into a single powerful middlebox, and their policies are accumulated after passing validation and verification procedures. A middlebox manager is added to empower network administrators with the ability to steer traffic towards the required sequence of middleboxes. The integrated middleboxes are replicated across multiple subnets to increase their availability and to spread the traffic load among these subnets. Moreover, in the same context of the previous solutions [4,7], we aim to solve the shortcomings above by relaxing the network core from the interposer devices, maintaining the policy sequences as illustrated in [5], raising the flexibility and utilization of the middleboxes, and balancing the traffic load between them. Our solution is practical, as only edge switches, and the subnets of middleboxes must be able to communicate through OpenFlow. This is a common practice and is available in a variety of network hardware these days.

Our main contributions in this work are as follows:

- We did a survey on the current deployments and costs of middleboxes in 30 enterprises of various sizes. We found that a large number of the investigated middleboxes are highly underutilized.
- We solved the challenges raised, i.e., the policy chaining of middleboxes, topology dependence, and consolidation challenge, by proposing SALMA. This is the main contribution of this paper.
- We invented a novel encoding mechanism by exploiting a correlation between the service policy chaining and the address space in packet headers in order to allow the forwarded packets to traverse a pre-definded sequence of middleboxes.

2. Motivation

Before discussing the proposed work, we present the results of a study that we performed on the deployment of middleboxes in multiple enterprise networks to highlight their challenges and limitations.

2.1. Survey Details

We conducted a survey with around 30 enterprise networks as participants. The goal of this survey was to demonstrate the challenges and limitations in existing deployments in today's networks. We collected the number of middleboxes, the average number of employees assigned to every middlebox, and the costs of different middlebox platforms. The studied enterprises are in various business sectors. They are Internet Service Providers (ISPs), telecommunications companies, academic institutions, and financial organizations. For security reasons, we were not allowed to reveal the identities of businesses that participated in the survey. Thus, we classified the survey results based on enterprise size (i.e., large, medium, and small). Our dataset included 4 small enterprises (fewer than 1000 hosts), 11 medium enterprises (between and 1000 and 10,000 hosts), and 15 large enterprises (10,000 to 100,000 hosts). Figure 1 shows the number of middleboxes deployed in the three classes of networks of the participating enterprises. The type of middlebox is abbreviated on the x-axis as follows: WO stands for WAN optimizer, PX stands for proxy, LB stands for load balancer, VS stands for VPN server, IDS stands for intrusion detection system, and FW stands for firewall.

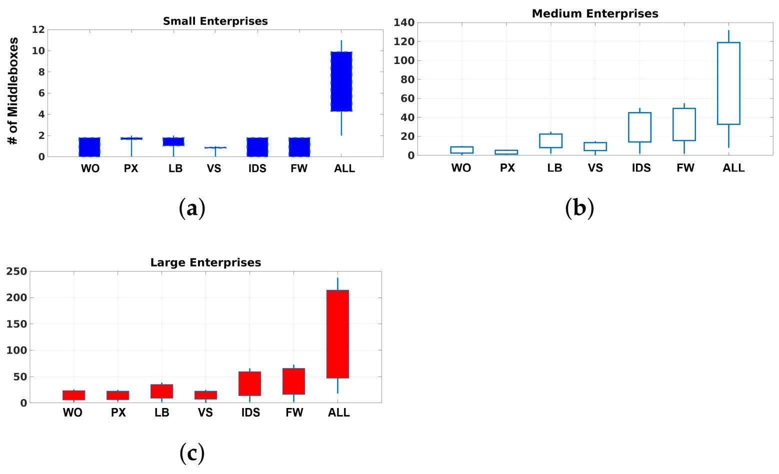


Figure 1. Number of middleboxes that are deployed in enterprise networks of various sizes. (**a**) Middleboxes in small enterprises. (**b**) Middleboxes in medium enterprises. (**c**) Middleboxes in large enterprises.

2.2. Middleboxes Cost

The cost of middleboxes is not limited to the hardware cost. In addition, the management cost of middleboxes is also high. It includes the number of assigned employees, the cost of training, the number of middlebox failures during the year, and the estimated mean cost to repair (MCTR). Table 1 lists the average middlebox price based on the network size. Additionally, the estimated MCTRs per year for the participating enterprises are illustrated in Figure 2. To be able to estimate MCTR, we asked the participating enterprises the following questions:

1. What is the average network engineer's salary?
2. How many engineers are assigned to fix a single failure?
3. What is a failure's mean time to repair?
4. What is the number of failures within a year?

Table 1. Middleboxes' prices (thousand $) for various network sizes.

	Small	Medium	Large	Very Large
Firewall	1.4–3.1	4.3–14.8	20–44	50–104
IPS	0.56–1.1	5–21	35–50	84–200
WAN Opt.	1.7–10.5	12.5–24	26–59	114–235

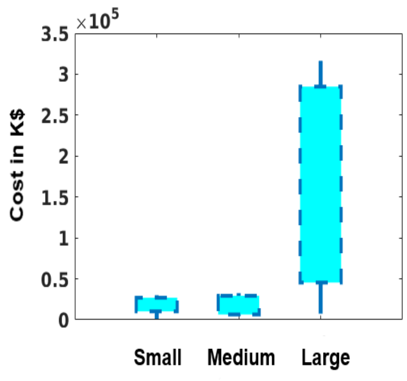


Figure 2. MCTR of all middlebox failures during a year.

From these questions, we built our MCTR estimation. Since the majority of failures are due to human-based errors, such as misconfiguration or traffic overload [1,10], we did not include the cost of spare parts in our estimation. In fact, MCTR would have been higher had the cost of the spare parts been included. It is clear from Figure 2 that the cost is positively proportional to the number of middleboxes deployed in the network. Hence, a key factor in reducing the cost of middleboxes is reducing the number of deployed middleboxes; any reduction in the number of deployed middleboxes is going to affect other costs directly. Instead of trying to reduce the cost of management by providing a sophisticated management console, or by increasing the utilization of middleboxes, it is easier to just simply lower the number of middleboxes deployed.

2.3. Utilization of Middleboxes

The collected statistics showed that multiple duplicate middleboxes of the same type, (i.e., IP firewall, IDS/IPS, or load balancer) were installed inside the same network. For example, when we closely looked at one of the investigated service-provider networks, we found that every interface that was associated with an untrusted network was connected to a firewall and an IDS. In some cases, duplicate middleboxes are installed in parallel for high availability. Thus, the utilization of the firewalls and IDSs relies on the activities of the customers in the connected branches. To demonstrate our claim, we collected the utilization statistics of four middleboxes (two firewalls and two load balancers) installed in various locations inside a service-provider network, as shown in Figure 3. In fact, we noticed that every common service chain is grouped in an independent set, so firewalls must exist in every set. This means that the maximum utilization of every set could not exceed the firewall utilization. Additionally, the datasets provided have multiple firewalls and load balancers; their utilization was similar to that displayed in the figure. The statistics were collected during October.

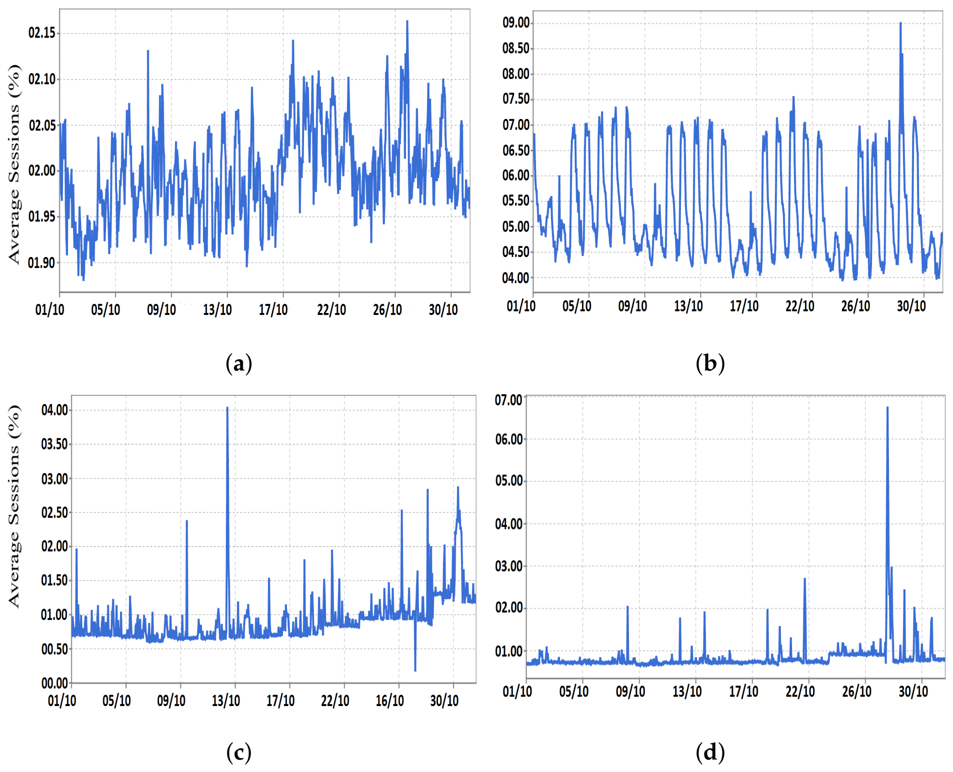


Figure 3. The utilization of four independent middleboxes installed in different locations inside a service-provider network. (**a**) Firewall 1 (FW1). (**b**) Firewall 2 (FW2). (**c**) Load balancer 1 (LB1). (**d**) Load balancer 2 (LB2).

In reality, middlebox duplication is done not only to achieve high availability, but also due to topology-dependent conditions. Typically, middleboxes are scattered all over the network in order to implement the network policy on the right traffic. This duplication practice results in low middlebox utilization as well. We found that the firewalls studied showed a serious utilization gap because of the topology-dependent conditions and routing protocol stiffness. This is due to the fact that the traffic of untrusted domains has to be monitored and examined by the firewall/IDS policies regardless of its volume. Furthermore, the available routing protocols are not flexible enough to provide a sufficient degree of control. In fact, when we analyzed the middleboxes' utilization statistics, we found variability during a small period of time (i.e., one or more days), which is aligned with previous findings [4]. However, when these groups of firewalls are integrated into a single firewall and all the in-scope traffic is forwarded to that box, the utilization will be increased, and all the overall cost (management, operation, and others) will be reduced.

2.4. Middleboxes' Placement

Due to network structure limitations and policy constraints, we found that the middleboxes in the surveyed networks are physically located between networks to be able to enforce certain policies on a particular group of services and/or end-hosts, which are defined here as in-scope systems. Despite the fact that this enables middleboxes to be on the path of in-scope traffic, it complicates the network management and hinders the deployment of recent network agility services [1,10]. In reality, middleboxes, such as load balancers, VPN servers, and WAN optimizers, which are physically installed in the path of bi-directional traffic, are often useless and may add an extra overhead to the exchanged traffic [6]. For instance, the potential speedup from deploying the load balancer is $\theta(t)$; however, the load balancing algorithm introduces an extra overhead $L_{overhead}$ to perform its load-balancing function. When the delay of $L_{overhead}$ is greater than the obtainable speedup $\theta(t)$, it is better to bypass the load balancer until the obtainable speedup $\theta(t)$ is greater than the load-balancing overhead $L_{overhead}$. In fact, researchers found benefits in bypassing the load balancer even in the return path of network traffic [18]. To give another example, let us consider the voice and video traffic as examples of delay-sensitive flows. Holding such traffic back in the WAN optimizer for compression will introduce an unnecessary delay and jitter. Additionally, the video streams are already compressed, so additional compression will not be beneficial. Consequently, VoIP or video should bypass the WAN optimizer.

3. Problem Statement

In the investigated enterprises, each group of middleboxes was installed in the face of certain traffic of particular users. Thus, these middleboxes are configured to perform certain network/security policies on the passing traffic. In an individual enterprise there are M middleboxes (number of them). Every group of middleboxes is gathered in a single set, $S_1, S_2, S_3, \dots S_n$, and the total number of sets is N. A single set contains different types of midddleboexs and is placed in one of the entry points (known as the point of presence (PoP)) of the enterprise network. These sets are implanted in the network path of a certain group of users to execute specific QoS and security policies on their traffic. For instance, the VPN users entail different QoS and security measures than the visitors for the enterprise website.

The network designers typically select the place of every set with little consideration for their optimization. Indeed, the sets' places are critical, and they suffer the same fate: the failure or delay in one of the set members impacts the performance of the whole set. One way to mitigate this challenge is by replicating the set members, which in return increases the total capacity $\mathcal{C}_m$ and reduces the utilization, and increases the expenses. Assume the utilization of a set i is $\mathcal{U}_i$, where its value is proportionally related to the traffic load of its users λ^i_E. Therefore, $\mathcal{U}_i = \frac{\lambda^i_E}{\mathcal{C}^i_m}$. In order to elevate the utilization, we need either to add new load (e.g., more users) or decrease the available capacity. The administrator does not control the users' behavior or places. In addition, the administrator has limited control over the middleboxes' processing capacity[2]. This is not only for one set, since the enterprises and ISPs have hundreds of sets to serve internal and external PoPs, which means a huge number of unused resources.

In this work we aim at providing the network administrator enough governance on these two factors. The main obstacle to optimizing the performance of the middleboxes is the network's structure. To overcome this issue, the sets of middleboxes are decoupled from their places and regathered in new sets. The decoupling technique introduces multiple challenges. First, how does the solution engage the users with their corresponding set of middleboxes? Second, how does the solution forward the E^i traffic to the set S_i over the enterprise network? Third, how does the solution make sure the new set have enough capacity to handle the aggregated traffic load? Finally, the solution needs to optimize the assignment process between the traffic load and the sets of the middleboxes. We elaborate more on these concerns in Section 4.

Instead of randomly reassigning the group of users to the middleboxes' sets, we need to search for the optimal set that satisfies certain constraints, such as capacity and the security/network policy. The target optimal function is:

The first constraint, the capacity of the new sets, is enough to serve the total traffic load from all sets.

$$\sum_{i=1}^{N} \lambda_E^i \leq \mathcal{C}_L \tag{1}$$

We can rewrite this constraint to be as follows:

$$\mathbf{E}_\tau \leq \mathcal{C}_{QoS}^L \tag{2}$$

Assume the matrix $\boldsymbol{E}_\tau$ has all the traffic workload on previous sets before the new arrangement, and the $\mathcal{C}_{QoS}^L$ matrix has the capacity of all individual middleboxes in every new set. Since the traffic load is different during a working day, $\mathbf{E}$ is a fat matrix which has the time with a resolution of τ unit of time as the column index. For instance, column $(index = 1)$ has the traffic workload on every set during the τ_1 unit of time[3]. The second constraint is the processing delay: $\boldsymbol{\mathcal{D}}_s$ should not exceed a certain QoS delay, $\boldsymbol{\mathcal{D}}_{QoS}$.

$$\boldsymbol{\mathcal{D}}_s^\tau \leq \boldsymbol{\mathcal{D}}_{QoS} \tag{3}$$

The delay issue commonly arises when the middleboxes are highly utilized, which was our goal in solving this problem. Accordingly, the following binary LP model was formulated to maximize the load on the subnets of middleboxes such that the involved constraints are satisfied.

$$S^* = \max_i \sum_{i=1}^{M} x_{i,j}. \tag{4}$$

s.t.

$$x_{i,j} \in \{0,1\}.$$

where M is the number of new sets. In this case, the model assigns as much traffic load as it can to one of the sets S_i, $i \in M$, until one of the QoS constraints is not satisfied. This practice is a well-known technique in the literature to increase the utilization and reduce the total power consumption. The essential weakness of this technique is the inevitability of assigning the same load to more than one set. To overcome this issue, the following constraint has been added.

$$\sum_{j=1}^{N} x_{i,j} \leq 1 \tag{5}$$

This constraint is to enforce only one assignment for every traffic load.

4. SALMA Data-Plane

After promoting the idea of middlebox integration by presenting some of the shortcomings of middlebox deployment in current enterprise networks, we are now ready to discuss how these shortcomings are addressed in our work. Our proposed solution has two types of data-plane devices: the edge switches and the middlebox subnet switches. The incoming flows to any of the edge switches can be classified into in-scope and out-scope flows. The former need to be forwarded to one of the subnet replicas of the middleboxes, whereas the latter are not required to be forwarded to the middlebox subnet. Instead, these flows should be forwarded directly to the destination. The edge data-plane devices must be able to recognize which flow belongs to which class, and this is the duty of the middlebox manager, as explained in detail in the next section. When an unknown flow is received by an edge data-plane device, it sends a flow setup request to the middlebox manager, which in turn installs a flow entry on the edge switch to decide whether this flow is an in-scope or out-scope flow. In the case of in-scope flow, the data plane forwards the flow packets to the middlebox subnet. Otherwise, the packets are forwarded to the destination. In simple words, the middlebox manager classifies the unknown flows based on the policy configuration that was installed by the network administrator into the policy sequence module. The second type of data-plane device is the middleboxes' subnet switch. When the middlebox manager receives the flow-setup request from the edge switch and considers it as in-scope flow, it configures the middleboxes' subnet switch with multiple flow entries to maintain the policy sequence. At the same time, it checks the collected statistics and the number of assigned flows to each middlebox's subnet replicas to balance the traffic load among them.

5. Middlebox Manager

One of the key considerations in the design of our solution is to make it topology-independent. The network traffic routing decisions and operator specific policies must be set by a central entity, and the middleboxes must be plugged out from a traffic link. The middlebox manager is the one responsible for the edge switches' configuration. As explained in the previous section, this special configuration enables edge switches to route in-scope traffic to the subnet of the middleboxes. Furthermore, the middlebox manager is responsible for balancing the traffic load between the available subnets of the middleboxes, and configures their switch with the right flow entries to maintain the policy sequence. Figure 4 shows the main modules that form the middlebox manager panel, including the network topology database, load balancer, and policy sequence module. The figure also shows how these modules interact with each other and how they interact with the other data-plane devices. In the following paragraphs, we will describe some of these modules in detail.

Topology Database Module: The middlebox manager collects the network topology information from OpenFlow Discovery Protocol (OFDP) messages, which are similar to Link Layer Discovery Protocol (LLDP) messages [19]. The data-plane devices in the network exchange these messages to know more about the data-plane devices and the connected hosts. Fortunately, non-OpenFlow switches can still forward OFDP messages without reacting to them. In fact, the middlebox manager uses these messages to build the network topology. Typically, the link-state routing protocol builds the network topology database in production networks, which is used by the middlebox manager whenever it is needed.

Load Balancer Module: The edge switches and the middlebox manager communicate through the OpenFlow protocol. The middlebox manager is involved in every flow-setup, which provides it with a high degree of visibility in network activities. This visibility allows the middlebox manager to do the load balancing efficiently between the replicated subnets of the middleboxes. Actually, the motivation of load balancing goes beyond distributing the load between available resources. For example, proper load balancing helps prevent denial of service concerns, such as unacceptable delays in the order of milliseconds that could easily disturb multimedia services. Since the service times of the middleboxes are almost the same, we decided to use a random flow assignment method to spread the load among available subnets of the middleboxes. When the middlebox manager receives a flow-setup request of an in-scope flow, it randomly selects the middleboxes' subnet and builds an end-to-end tunnel between the edge switch and the selected subnet of the middleboxes.

Policy Sequence Module: The network administrator is responsible for configuring the middlebox manager with the network policy classes, similar to the ones shown in Figure 5. This figure shows several examples of policy sequences. These sequences could easily be classified into multiple classes where each class contains the list of middleboxes that must be visited, and the order of these visits is by the flows that match this class.

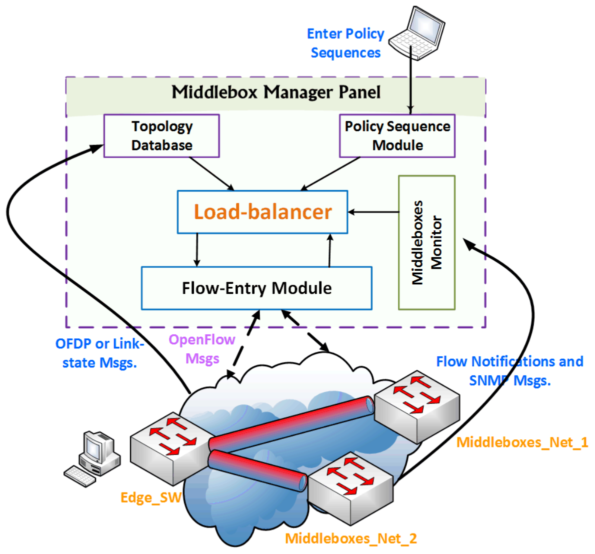
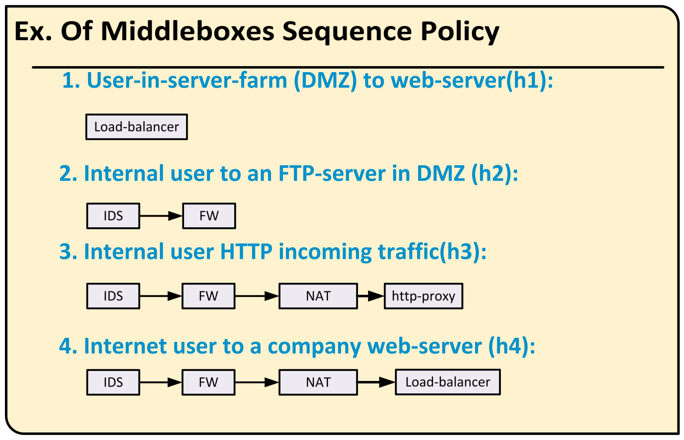


Figure 4. The middlebox manager panel.



Ex. Of Middleboxes Sequence Policy

1. User-in-server-farm (DMZ) to web-server(h1):

   Load-balancer

2. Internal user to an FTP-server in DMZ (h2):

   IDS → FW

3. Internal user HTTP incoming traffic(h3):

   IDS → FW → NAT → http-proxy

4. Internet user to a company web-server (h4):

   IDS → FW → NAT → Load-balancer

Figure 5. An example of variations in policy sequences.

6. Implementation

To have a scalable integrated design of middleboxes, we need to address several challenges, namely, a policy challenge, a consolidation challenge, and a load-balancing challenge. In the following lines, we describe how these challenges are tackled in our implementation.

6.1. Policy Challenge

The proposed solution needs to maintain the policy sequence where the flow visits the right middleboxes in the right sequence. In today's network, when a flow needs to be examined by a sequence of middleboxes (e.g., firewall $\rightarrow$ IDS $\rightarrow$ load balancer), these middleboxes are installed in the flow path. In this section, we shed light on the policy sequence algorithm implemented in our solution. The algorithm defines how the edge switches will forward the in-scope traffic to the middleboxes' subnet. It also explains how the middleboxes' subnet switch maintains the policy sequence. Furthermore, we show in this section how the proposed algorithm handles both loop and loop-free scenarios. Let us assume the following sequence of middleboxes is installed physically between endpoints to face their exchanged flows:

$$\text{FW} \rightarrow \text{IDS} \rightarrow \text{Proxy}.$$

When these middleboxes are moved from one location to another inside the enterprise network, it is necessary to forward the exchanged bidirectional flows between the endpoints to the right middleboxes. In addition, the bidirectional flows must visit the middleboxes while following the right sequence.

The middlebox manager steers the in-scope traffic to one of the preconfigured tunnels between edge switches and the subnets of the middleboxes, as shown in Figure 6. The selection criteria of middleboxes' subnets is based on the load-balancing algorithm and the policy sequence. As illustrated in Figure 6, the physical arrangement of middleboxes in the subnet cannot satisfy all the policy sequences. For instance, the incoming HTTP traffic must follow the subsequent sequence: IDS → FW → Proxy, and the sequence of middleboxes visited in the case of outgoing Internet is FW → IDS. Thus, we can not physically reorder the middleboxes in the subnet to satisfy all sequences. To solve this issue, we propose to install the middleboxes in the subnet in any order. Since this order is known to the middlebox manager, it will be responsible for installing the appropriate flow entries to maintain the appropriate sequence. Unfortunately, there are only a few fields in the IP and MAC headers that can be utilized for this purpose. Flags such as DSCP and ECN are used by other solutions, such as the WAN optimizer and the QoS configuration in routers. Furthermore, end-to-end flows and middleboxes, such as IDS, proxy, and firewall, are sensitive to L4 fields (e.g., destination port number). Nevertheless, we propose two alternatives to maintain the policy sequence.

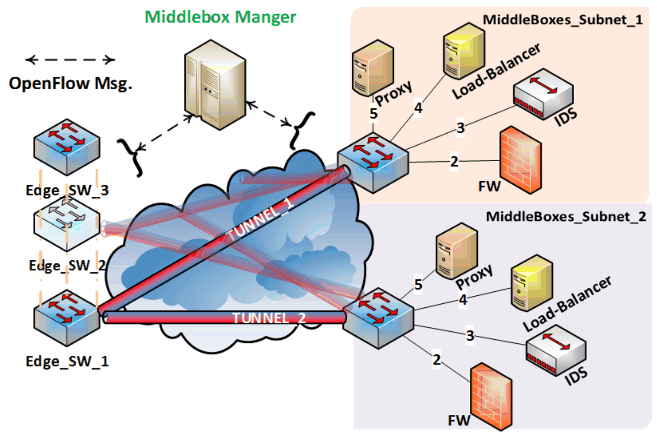


Figure 6. A sample network topology of SALMA showing in-scope traffic steering between edge switches and the subnets of the middleboxes.

6.1.1. SALMA v1

The idea here is to utilize the IP/MAC source and destination fields for policy sequencing by installing all the flow entries in the same flow-table. By using the multiple flow-table features in the OpenFlow protocol and the TTL field in the IP header, flow entries in each middlebox are assigned to a certain flow-table, (e.g., the middlebox in outport#2 is assigned to table = 2). The TTL value is used to point out the port number of the next middlebox or final exit. The incoming flow will be handled by table = 0, then resubmitted to the appropriate table according to the intended policy sequence. This process will continue until the flow exit from the last middlebox is in its sequence. The flow entries in the tables other than table = 0 are used to reset the TTL value and forward the flow to the corresponding middlebox. Moreover, we assigned a particular table for each middlebox to facilitate the flow management and speedup of the flow entry's lookup, since routing tables in large networks might contain more than 100,000 entries [20].

6.1.2. SALMA v2

In this version of the algorithm, we encode the policy sequence into the packet address, such as a MAC or IP address. The packet address is replaced by the output ports of the middlebox to be visited. To able to do that, the middlebox manager must configure the middleboxes' subnet switch to forward every incoming flow. This configuration needs to precisely follow the service-chaining policy. This configuration introduces a new communication overhead. In order to optimize it, we exploit a correlation between the service policy chaining and the address space in packet headers.

The service-chaining policy requires that each and every flow is examined by particular middleboxes. Each middlebox is connected to the subnet switch via a specific network port. The middleboxes in the chain are identified neither by their names, nor by their functions. Instead, the subnet switch will identify them by their outgoing ports, which are encoded into the packet headers. The most efficient method for performing the encoding task is utilizing the IP/MAC rewriting feature. Accordingly, the proposed technique encodes the service chaining into the MAC address.

The edge router performs the address rewriting task of the outgoing edge-to-middlebox packets. On the other hand, the middleboxes' subnet switch performs the decoding, i.e., returning back the original MAC address. For example, let us assume that the firewall is connected to port number 20 and the IDS is connected to port number 21. As such, the corresponding MAC address is 20:21:1:0:0. The port number 1 is the middleboxes' subnet exit port. The system is free to select either the source, the destination, or both MAC addresses for encoding. However, the selection needs to be synchronized among all parties in the system.

We simplify the algorithm's mechanics to make it easily adopted by in-production networks. Upon the arrival of a new flow, the edge switch performs service-chain encoding, as instructed by the middlebox manager. At the same time, the middlebox manager stores the original address information of this flow. This information is used to configure the middleboxes' subnet switch to perform the decoding task for this flow precisely. Then, the added rules are removed to free some space for new rules and avoid any unplanned conflicts. The middlebox manager has the necessary information to efficiently select the address, fulfilling the demand of the security policy.

Finally, we would like to clarify how the the middleboxes' subnet switch performs the decoding to return back the original MAC address. When an unknown flow arrives at an OpenFlow switch, a packet-in message is sent to the controller. Additionally, the OpenFlow protocol has a feature to enable data-plane devices to send a notification to the controller upon the arrival of a particular flow(s). When a flow that must get examined by the middleboxes arrives at a switch, which is usually a PoP switch, the controller is notified. These notifications have enough information about the flows, including their IP and MAC addresses. The controller reads its service chain database to learn which middleboxes need to be visited by this flow and in which sequence. Accordingly, the MAC address is structured, and a rewriting command is sent to the switch. Simultaneously, a rewriting command is sent to the middleboxes' subnet switch to modify the flow packets with their original MAC addresses before forwarding them out of the middleboxes' subnet. Moreover, the controller, alongside the rewriting commands, configure the PoP switch to forward the flow toward the selected middleboxes' subnet.

6.2. Consolidation Challenge

We have concluded from the survey analysis in Section 2 that several copies of the same middlebox type are installed in various locations in current enterprise networks. Due to multiple reasons, the enterprises in our survey have several middleboxes of the same type (i.e., firewall, IDS, and load balancer) installed in different places inside their networks. In fact, this finding is in agreement with recent surveys [1,4]. This practice causes middleboxes to be poorly utilized, as shown in detail in the previous sections. Furthermore, service providers typically have thousands of rules in their firewalls. The number of rules is huge due to the new proliferation of communication technologies and services and the notable increase in Internet users. Every account manager pushes the IT team to provide the service for his/her client ASAP. This unhealthy practice forces the firewall administrators to turn a blind eye on some quality steps and add new rules with low credibility, which increases the likelihood of creating anomaly rules. As a response to these unhealthy practices, researchers invent efficient tools to detect these rules and remove them. We expect the administrators to add in their middleboxes a maintenance procedure step to check for anomaly rules.

In this work, we propose to integrate middleboxes from the same type into a single powerful device. In reality, the consolidation of multiple middleboxes' functionality into a single general-purpose hardware is not new in the literature. For instance, most of the enterprises in our survey integrate both firewall and NAT functions into a single hardware box. Similarly, the load balancer and IDS are usually integrated when dealing with web traffic. However, the goal of this work is to integrate multiple middleboxes of the same type, such as multiple firewalls. This integration is the cause of what we called the consolidation challenge. In fact, the rules of similar devices can be manually examined. However, we expect the middleboxes, particularly firewalls, as we mentioned above, have thousands of rules, so the manual reading is not efficient even with a keen eye. We believe the market has multiple efficient anomaly-rule-detection tools, such as ManageEngine, solarwinds, algosec, Cisco CLI analyzer, and WallParse. Similar middleboxes (e.g., firewalls) are grouped in one file and then examined by these tools to detect and remove anomaly rules. To the best of our knowledge, the consolidation challenge has not been addressed in the literature. In this section, we discuss the middlebox consolidation procedure and how the proposed solution addresses its challenges.

Most of the middleboxes are configured with a sequence of policies, such as the access control list (ACL) in the case of firewalls, which must be examined for every incoming packet. For example, the firewall policies are designed to examine certain fields of a packet header, and subsequently either forward the packet when it has a "permit" action or drop the packet when it has a "deny" action. To consolidate multiple peers of middleboxes, we need to integrate their policy rules. However, the policy rules after integration should be validated and verified against duplication and conflict incidents using a well-known procedure and tool [21]. Additionally, on some occasions, the middlebox must not be integrated with its peers for performance or security reasons, which is defined here as the non-integrable middlebox challenge. Additionally, the middlebox needs to keep its configuration context, such as an ACL of a classified customer, away from other middleboxes.

When we integrated the policies of multiple middleboxes of the same type, we found some of the examined middleboxes already having repeated rules that perform the same action. As a result, we followed a verification procedure to eliminate these duplicates. The implemented verification procedure included the following:

1. Middlebox equivalence checking: to examine the new integrated policy rules that are installed into the unified middlebox having all the previous individual middlebox rules.
2. Middlebox rule redundancy checking: to eliminate redundant rules after integration.
3. Middlebox rule conflict checking: to detect and eliminate conflict rules after integration.

To solve the non-integrable middlebox challenge, we used both hardware and software features of middleboxes. From our investigation, we found that most firewalls are supported by several configuration management features that facilitate the partitioning of a single large ACL list into multiple small lists. These lists are installed into different tables. A configurable classifier in the first table is responsible for classifying incoming traffic and resubmitting it to the right list. This feature is similar to the multiple tables feature in OpenFlow switches, where the first table, table = 0, is the classifier, and the other tables contain the ACL lists. In this work, we utilized this feature to configure each table with the needed context. For example, when the ACLs of firewall-1 and firewall-2 cannot be integrated with each other, the ACLs of these two firewalls are configured in table = 1 and table = 2, respectively. The lists of other firewalls can then be integrated in table = 3. Additionally, the integrated ACL lists need to pass the validation and verification test.This feature has an additional advantage: the reading of the ACL list is faster in cases of small lists.

6.3. Classifier-Shim Layer

In case the configurable classifier feature is not supported by the middlebox, we have two alternatives. First, if the middlebox is an application software, then we can integrate that middlebox with its peer in general purpose hardware. The incoming packets to this hardware need to be forwarded to the right middlebox platform. We introduce a classifier-shim layer (cshim) that is responsible for classifying the incoming traffic and forwarding it to the right middlebox platform. This shim layer is similar to the policy shim layer in CoMb [4]. In reality, they designed this layer to maintain the middleboxes' policy sequence, and we designed it to classify and forward the traffic. For example, let us assume having a load-balancer platform that cannot to be integrated with other middleboxes. We can simply install the integrable load balancers and the non-integrable load balancer in general purpose hardware where the cshim classifies and forwards the incoming traffic to the right load balancer, as displayed in Figure 7. The second solution is to utilize the availability of network ports in the middleboxes' subnet switch to plug out the non-integrable middlebox and plug it into any available port in the switch. In this case, the middlebox manager is responsible for classifying the incoming traffic and configuring the middleboxes' switch on the fly to forward the incoming traffic to the right middlebox.

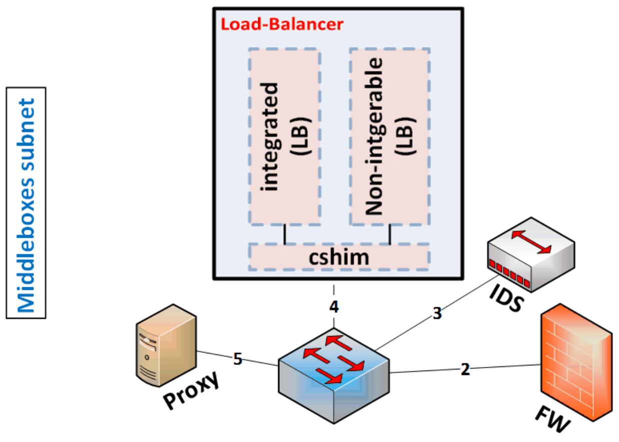


Figure 7. Consolidating integrable and non-integrable (LB = load balancers). This is an example that can be generalized to other middleboxes.

The cshim layer can be implemented by using IP-Table service of the Linux kernel or in more sophisticated manner by coding a kernel module. In this work, we go for the latter case. A kernel module has been written to enable our solution with a wide spectrum of features. The flow can be classified according to legacy features (e.g., source and destination MAC/IP addresses, and transport layer ports) or advance features (e.g., flow size: elephant or mouse). The legacy classifications can easily be done in both the kernel module and IP-Table. The advanced classification needs deep inspection of forwarded packets, where the IP-Table has no leverage.

In this work, we used the NetFilter Linux feature to write the cshim layer kernel module. Moreover, the kernel module can be uploaded into either the host kernel or the kernel of the Openvswitch/hypervisor. We tested the uploading of the cshim layer into the Mininet host and the OvS; the switch in Mininet is an Openvswitch. However, the results were collected from the former case. The kernel of these two systems provide a degree of control for exchanging packets and full access to their headers and unencrypted payload. The cshim layer has a hashtable to store the flow information and statistics. When the flow size exceeds a certain threshold value, 1MB, it is considered elephant flow [11–17]. We used the aggregated topology. The size of mouse flows is randomly selected from 10 KB to 1 MB, whereas the elephant flows range is 10–128 MB.

In our cshim layer, we tested our module using this feature, and tested its speed and accuracy when implementing this feature. We generated flows of mouse-size-only, elephant-size-only, and a mix of mouse and elephant flows. The number of flows in every test is 600. During the mix scenario, the percent of elephant flow was 10%. The cshim layer achieved 100% accuracy in all of the tested scenarios. This accuracy is due to its privileged location where it resides in the face of the transmitted flows, which facilitates its goal in measuring the flow size. The speed of detection on average is 14 ms.

6.4. Load-Balancing Challenge

The motivations of load balancing are not bonded by spreading the traffic load among available resources. In addition, the value of middleboxes' resources and denial of service concerns include unacceptable delays that could reach milliseconds for multimedia services. In this work, we used an optimal flow assignment method to spread the load among the available subnets of the middleboxes. When the middlebox manager receives a flow-setup request of in-scope flow, it selects the middleboxes' subnet based on the optimization model that is explained in Section 3, and builds an end-to-end tunnel between the edge switch and the selected middleboxes' subnet.

7. Evaluation

In this section, we evaluate our proposed solution through various sets of experiments. Before starting our extensive evaluation, we built a proof of concept for the SALMA; the topology is depicted in Figure 6 and the policies in Figure 5. After that, we evaluated SALMA using larger, more realistic topologies, such as Internet2, Geant, and the abstracted enterprises network, as shown in Figure 8, to be able to evaluate the performance on various topologies. Moreover, we evaluated the overhead of SALMA on the abstracted enterprise network using various scenarios.

7.1. Proof of Concept

We built a topology similar to the one shown in Figure 6 by using Mininet [22]. We modified POX [23] to act as the middlebox manager. In this experiment, the hosts and the destination server were installed in the same subnet. However, the traffic was steered to the middleboxes' subnet to be examined by the intended middleboxes based on the policy sequences explained in Figure 5. The results of average TCP throughput for both SALMAv1 and SALMAv2, with and without the load balancer, are shown in Figure 9. In this scenario, the iperf [24] communication is sequential from h1 to h2, then h3, and then h4. For this reason, the impact of the load-balancing solution is demonstrated in h3 and h4 results, where the network got enough load.

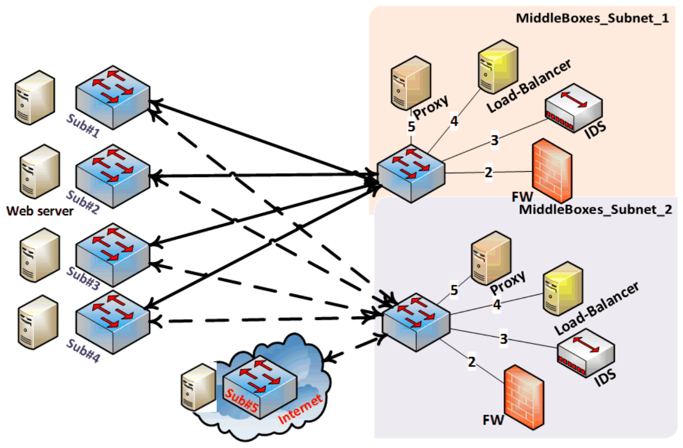


Figure 8. The enterprise topology after abstraction.

7.2. SALMA Evaluation

In this section, we evaluate the effectiveness of our proposed solution in terms of throughput and utilization. The network was composed of 10 subnets. Every subnet had 7 middleboxes, and the arrival-rate followed an exponential distribution with a mean of 20 milliseconds, with the number of flows being 60 for every subnet. The individual flow randomly visited between 3 and 7 middleboxes. After the aggregation, we had only a single subnet with 7 middleboxes, and the arrival-rate followed an exponential distribution with a mean of 2 milliseconds; the number of flows was 625. Every individual flow randomly visited between 3 and 7 middleboxes. Evaluations were conducted using Mininet emulator [22] and the POX [23] controller on a machine with 4$\times$ (2.5 GHz Intel i7 CPU) processors and 8 GiB memory. Each switch in the above-mentioned topologies had 7 hosts, unless stated otherwise. Iperf was used to generate flows whose arrival followed an exponential distribution. The Iperf permits specifying the flow size and communicating units. The hosts in Mininet were divided into two separate sets (clients and servers), and Iperf was configured to loop these two sets till all the members were visited and the launched flows were completed. The clients and the servers were always selected from different subnets. The flow IDs were directly collected from the Mininet and stored in another file for comparison purposes. Our evaluation was based on network instances (i.e., one-shot). A new instance was run after ensuring that all flows of the ongoing instance had arrived. Linux monitoring features were used to monitor the processing of evaluation components such as Iperf PID of every running flow.

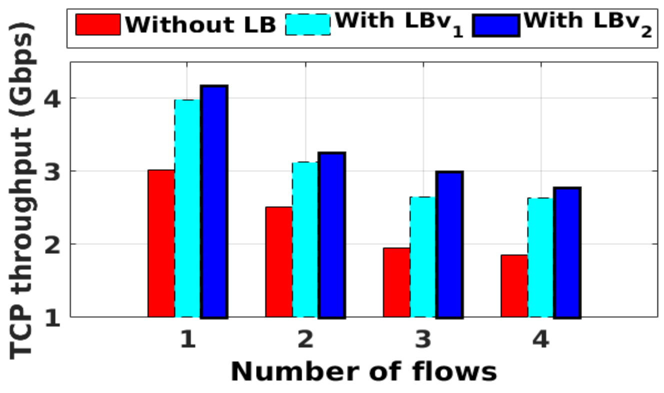


Figure 9. The average TCP throughput. LB = load balancer.

We start by evaluating the throughput gained by our algorithm before and after the aggregation of the middleboxes in one subnet. Figure 10 shows the average throughput of SALMA overtime compared to the normal approach of distributed middleboxes. On average, SALMA achieved 100.5 Mbps compared to only 4.1 Mbps for the case of the distributed middleboxes approach. To understand the throughput behavior, we analyzed the TCP statistics, including the congestion window (CWND) and the round trip time (RTT) of the TCP flows for the two scenarios. On average, the TCP congestion window in the case of SALMA was 22.85 bytes. The higher CWND value in the case of SALMA, as shown in Figure 11, is directly related to the higher throughput that we saw in Figure 10. We also did a comparison of RTT between SALMA and the distributed middleboxes approach, as shown in Figure 12. The mean RTT of SALMA is almost half of the mean RTT in the case of distributed middleboxes. Moreover, the latter have a higher maximum RTT of 370 ms compared to only 220 ms in the case of SALMA. We also compare the average CPU load of the middleboxes after and before the aggregation to prove that SALMA is going to increase the overall utilization. It is clear from Figure 13 that SALMA indeed increases the overall utilization. Table 2 lists average throughput, congestion window, and round trip time for SALMA, compared to the conventional method.

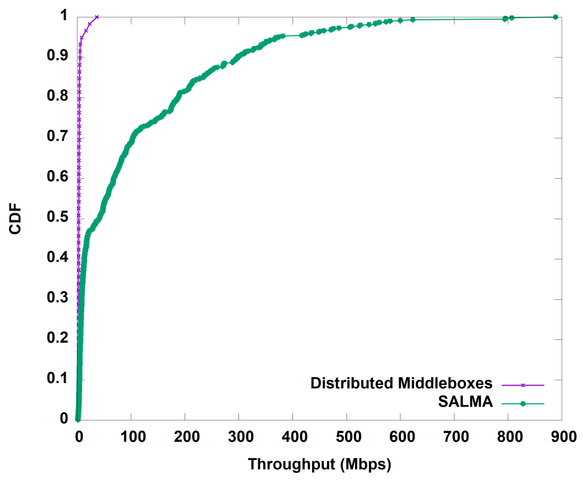


Figure 10. Throughput of both SALMA and the distributed middleboxes approach.

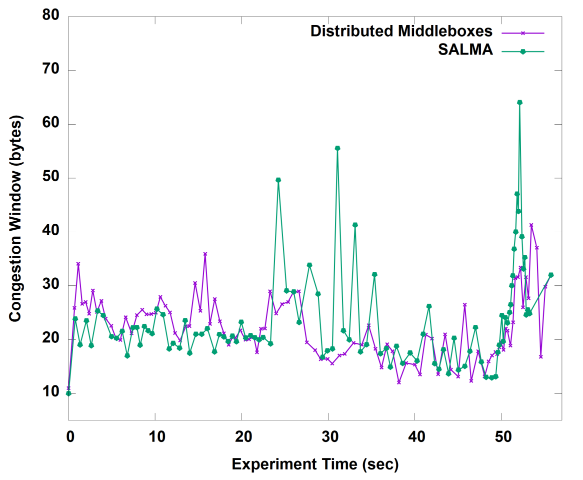


Figure 11. TCP statistics for SALMA and the distributed middleboxes approach.

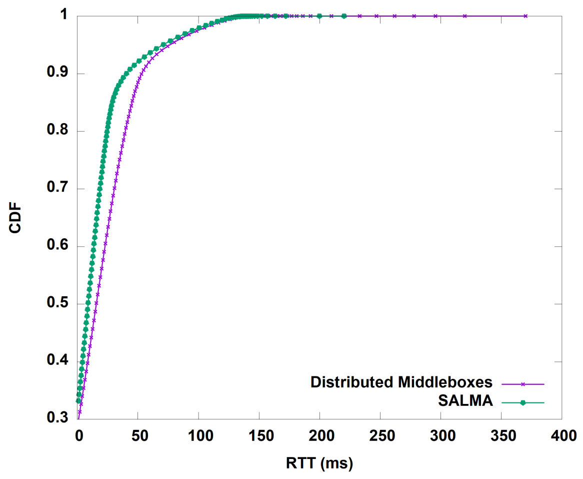


Figure 12. Comparison of round trip time (RTT) between SALMA and the distributed middleboxes approach.

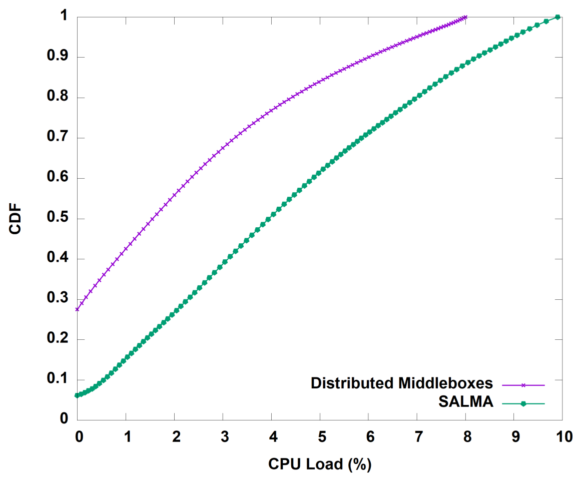


Figure 13. Percentage of CPU load for both SALMA and the distributed middleboxes approach.

Table 2. Average throughput, congestion window, and round trip time for SALMA compared to the conventional method.

Metric	SALMA	Distributed Middleboxes
Throughput	100.5 Mbps	4.1 Mbps
Congestion Window	22.85 bytes	22.47 bytes
Round Trip Time (RTT)	8.9 ms	15.7 ms

7.3. Overhead Analysis

In this experiment, we focused on three key benchmark metrics: the time to setup the network with proactive flow entries, the southbound communication overhead between the controller and the switches, and the flow setup delay of every new flow. We used Mininet to build large topologies, including Internet2/Abilene, Geant, and an abstracted enterprise network for one of the large enterprises that participated in our survey, as shown in Figure 8. We built the topology using Mininet where end nodes were connected via 1 Gbps links. In all the topologies, we gathered all the middleboxes into two subnets for redundancy and load balancing. The network setup details are shown in Table 3.

Table 3. The network setup overhead in terms of time to install proactive flow entries and southbound communication overhead.

Topology	Switches	Hosts	MBs	R_v1	R_v2	Setup (s)	Overhead (KB)
Internet2	13	39	10	152	202	≈0.043	≈15, 20
Geant	26	96	10	268	318	≈0.048	≈26, 31
Enterprise	9	130	10	88	138	≈0.038	≈8, 13

Network setup time: In reactive mode, we needed to install proactive flow entries to build a tunnel between every switch and the two middleboxes' subnets. The reactive flow entries were also used to dynamically balance the traffic load among the two subnets of middleboxes. Furthermore, we proactively used the installed flow entries to maintain the policy sequence of the flows that exit from the proxy. Since SALMA needs only a few flow entries, the setup time was in the range of 30 ms, as shown in Table 3.

The southbound communication overhead: Since the sizes of the OpenFlow messages are small and since only a few of them are needed to setup the network, the communication overhead between the switches and the middlebox manager is small. Table 3 shows that the communication overhead is in the range of few kilobytes.

Flow setup delay: In order to dynamically assign the new flows to one of the middleboxes' subnets, the middlebox manager is involved in the flow setup process. We built a simple topology in Mininet to test the flow-setup process's delay and compare it with the proactive mode. The network was composed of only a single switch, a controller, five hosts (h1–h5), and one server (h6). We tested the effect of the flow-setup process on round-trip time; where the flow entry expiration time was 55 s. The flow-setup delay was zero in the proactive mode, since the flow entries were already installed, and there was no need to communicate with the controller to set up a flow. The comparison is shown in Figure 14. The average flow-setup delay for the reactive scenario was 3.3 ms.

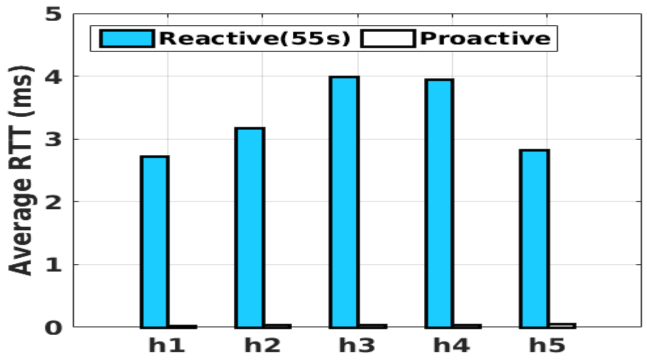


Figure 14. The effect of flow-setup on round trip time.

8. Related Work

In the literature, middleboxes attracted a remarkable amount of research. These research works span over three main categories: function-level integration, traffic detouring, and resource optimization. We also briefly discuss middlebox integration tools towards the end of this section.

Function-level Integration:

The idea of function-level integration was invented by the authors of CoMb [4]. From a practical survey on a very large enterprise, they found that the enterprise network presents low utilization of middleboxes' hardware resources. To increase the utilization, they proposed a scheme to integrate more than one middlebox type into a single general-purpose hardware device. For instance, the proxy server, WAN optimizer, VPN server, firewall, and IDS can be combined into a single hardware box to share their resources. A spatial load-balancing mechanism was proposed, along with a shim layer to maintain the policy sequence. A similar solution to CoMb is xOMB [25], an eXtensible Open MiddleBox software framework, which focuses on middleboxes' programmability and flexibility on commodity servers.

Similarly, a new extension of Click [26] named ClickOS [27] is a virtualized middlebox platform that is able to run hundreds of parallel middleboxes' functions without introducing significant delay. On top of KVM and Intel DPDK library, Hwang et al. introduced a high performance network virtualization platform to enable seamless integration of data plane functions with virtual machines [28]. Li et al. introduced Rubik [29], which is a domain-specific language similar to P4 [30], made to enable efficient programming and easy customization of middleboxes' functions. It is very clear that integrating different functions of middleboxes into a single machine causes multiple challenges, including security, availability, and integrity. Precisely, this type of integration results in having the super middlebox as a single point of failure due to misconfiguration, software bugs, or malicious breaches. This is what SALMA avoided.

Middleboxes in their current state need top-down development, including their placement in the network [31] and their stack code [4,25,27–29]. The middleboxes' replacement enforces integration either physically, as in SALMA, or functionally, as in CoMb [4]. The integration at the function level requires tackling the issues of packet latency [31], security [32], and compatibility [11]. The researchers mitigated these challenges by improving the packet processing speed [11,33,34] or migrating some functions out of the middleboxes hardware [35]. Moreover, the adoption of network virtualization technologies and programmable appliances enabled network developers to invent new protocols and services [29]. On the contrary, the middleboxes need to parse and read the packets of these protocols without redesigning them from scratch [29,36,37].

Traffic Detouring: In traffic detouring schemes, there are two approaches. The first approach keeps the middleboxes in their locations and migrates the traffic. In the other approach, the middleboxes are migrated to the cloud, and the internal traffic is detoured through the Internet. The authors of pLayer [7] proposed the Layer-2 mechanism to route traffic through a sequence of middleboxes connected to a policy-based switch. In their solution, they plugged out the middleboxes in the flow path and plugged them into the policy-based switch, where they were re-programmed to maintain the sequence policy. Even though this solution addresses several middlebox challenges, it did not tackle the following: load-balancing, utilization, and cost challenges. SIMPLE [10] is a middlebox control-plane layer that enables network administrators to enter the middleboxes' policy chain in a friendly manner; the underlying modules (ResMgr, DynHandler, and RuleGen) convert these policies into OpenFlow forwarding rules after taking into consideration the network topology, switch TCAM capacity, and middlebox processing limitations. StEERING [3], on the other hand, exploits SDN features to provide the network operators the flexibility to route the traffic between middleboxes while they are in their places. The authors of FlowTags [5] proposed a solution to provide the network with a flow-tracking mechanism to secure coherent policy enforcement with the existence of dynamic traffic modifications. We followed the same context of these solutions as we utilized the decoupling of control and data planes in our solution.

Other solutions took a more radical direction by moving middleboxes from inside the enterprise network to the cloud and directing the internal traffic through the Internet to the middleboxes located in the service provider's network [8,9]. Additionally, they proposed several solutions to mitigate the extra delay of routing the internal traffic to the cloud. Given the current and forecast size of the middlebox market [2], the hard-to-modify dedicated hardware middleboxes, the growing rigorous security practices, and the severe resistance to deploying radical solutions, the valuable solutions above will likely face serious hindrances in terms of resilience and security concerns, to being adopted. As an alternative, we propose a more practical solution that can make use of the resources available. In most cases, the available resources of switch hardware are enough to install the OpenFlow protocol. Additionally, the powerful hardware resources of available middleboxes can be utilized to integrate other middleboxes of the same type. In the worst case, the proposed solution can have more than one middlebox of the same type in the same subnet, and the middlebox manager will classify and steer the traffic of each type.

Resource Optimization: The invention of NFV and middlebox function virtualization enabled researchers to reduce the end-to-end delay by optimizing the middleboxes' policy-chain assignments [12]. This vital dimension of research started by tackling the keystone challenge, which is the modeling of the policy chain [38–43]. Mehraghdam et al. [38] formulated the policy chain by utilizing a context-free language. Zhang et al. [44] modeled the placement of virtual functions as a variant of the famous discrete optimization problem (i.e., bin-packing problem). The researchers found that there is a constant relation between optimizing policy-chain assignment and middleboxes' placement. Liu et al. [45] modeled the middlebox placement problem as a binary programming problem which is NP-hard. The authors proposed two heuristic solutions: greedy algorithm and simulated annealing. Pei et al. [46] benefited from deep reinforcement learning in finding the optimal solution for the middlebox placement problem. Ma et al. [47] introduced a new optimization model for the middlebox placement problem that takes into account the effects of traffic variability. Their main objective was augmenting the link utilization. Further details could be found in [31].

As an alternative, we propose a more practical solution that aims to meet the goals mentioned earlier (high utilization and cost reduction) through minimal adoption constraints. SALMA re-employs the resources available in a more efficient manner. Enterprise networks have several sizes of middleboxes scattered all over the network. These middleboxes are gathered in multiple subnets and deliberately integrated to serve all expected flows.

Middlebox Integration Tools: The firewall rules verification procedure is employed to verify that the existing firewall rules permit all packets that need to be permitted and deny all packets that need to be denied. In cases of redundancy, the firewall rules are examined to verify that no rule in a firewall is redundant. In this research field, there is much research on integrating the rules of firewalls and verifying their validity [48–50]. The authors of [48] proposed a firewall compressor by using dynamic programming techniques. In the same context, the authors of [49] formulated the firewall rules as a quantified Boolean formula (QBF) optimization problem, and introduced a new solution that can discover redundant rules in a given ACL. The authors of [50] proved that the firewall verification procedure and redundancy checking are in fact equivalent and give the same results. In this work, we followed the same procedure when we integrated the firewalls' ACLs.

Traffic Management and SDN: Researchers proposed utilizing SDN to achieve centralized and efficient control of the traffic of data centers, enterprises, and wide area networks [51]. Since the users have different needs, their traffic presents unequal characteristics. SDN services showed remarkable performance figures in identifying the flow classes and managing them accordingly [52,53]. On the other hand, the network designers expect the traffic to show a uniform workload between subnets/racks. However, Google, Facebook, and Microsoft researchers discovered a biased traffic workload, and the uniform structure of a wired cloud network is not efficient in dealing with it [54]. As a remedy, wireless networks (mmWave [55] and FSO [56,57]) were used to offer high-speed yet elastic topology. The topology reconfiguration needs to supply rapid responses and deliberate routing decisions to optimally utilize the available resources. Consequently, SDN is used to collect network statistics and offer fine-grained topology management. Moreover, wireless networks are used for non-data traffic, such as network management traffic [58,59].

9. Conclusions and Future Directions

Middleboxes are considered an essential part of current network infrastructure, since they provide critical services, such as security, performance enhancement, and traffic shaping. We found that the cost of middleboxes in today's networks is close to the cost of L3 routers. Furthermore, middleboxes are topology-dependent, as they usually reside between trusted and untrusted networks. Hence, their availability is a serious concern and has a severe impact on network operation. Given the rapid growth of SDN solutions and the recent advances in middlebox configuration management, the consolidation of middleboxes is becoming easier than before. In this work, we introduced a new practical solution, called *SALMA*, that addresses several challenges associated with middleboxes, including cost, low utilization, load balancing and topology-dependent deployment. Middleboxes of the same type are consolidated into a single middlebox, and the policy sequence is encoded in the packet address and into the separated tables of the middleboxes' subnet switch. Various types of middleboxes are aggregated into multiple subnets, and the middlebox manager is responsible for forwarding the traffic to the right middleboxes' subnet, and balancing the traffic load among them.

In the future, we are planning on studying the optimal placement of the middleboxes' subnets and the optimal members in every subnet. Additionally, it is interesting to look at the possibility of integrating SALMA with the emerging network security solutions, such as the intelligent security operations center (iSOC).

Author Contributions: Conceptualization, A.A. and A.S.; methodology, A.A.; software, A.A.; validation, A.A. and A.S.; formal analysis, A.A.; data curation, A.S.; writing—original draft preparation, A.A.; writing—review and editing, A.S.; visualization, A.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

Notes

1 On the contrary, out-scope flows are the ones that the security policies do not require to be examined by any of the middleboxes.
2 The administrator can use power management applications to optimize the middleboxes' power consumption.
3 The value of τ depends on the workload time resolution, which appears on the system logs, as shown in previous figures.

References

1. 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 Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, 2012, SIGCOMM '12, Helsinki, Finland, 13–17 August 2012; pp. 13–24.
2. Cloud Security Market to Be Worth $12 Billion by 2022, Here's Why. Available online: https://www.techrepublic.com/article/cloud-security-market-to-be-worth-12-billion-by-2022-heres-why/ (accessed on 3 August 2022).
3. Zhang, Y.; Beheshti, N.; Beliveau, L.; Lefebvre, G.; Manghirmalani, R.; Mishra, R.; Patneyt, R.; Shirazipour, M.; Subrahmaniam, R.; Truchan, C.; et al. StEERING: A software-defined networking for inline service chaining. In Proceedings of the 2013 21st IEEE International Conference on Network Protocols (ICNP), Goettingen, Germany, 7–10 October 2013; pp. 1–10.
4. Sekar, V.; Egi, N.; Ratnasamy, S.; Reiter, M.K.; Shi, G. Design and Implementation of a Consolidated Middlebox Architecture. In Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation, NSDI'12, San Jose, CA, USA, 25–27 April 2012; p. 24.
5. Fayazbakhsh, S.K.; Chiang, L.; Sekar, V.; Yu, M.; Mogul, J.C. Enforcing Network-wide Policies in the Presence of Dynamic Middlebox Actions Using Flowtags. In Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation, NSDI'14, Seattle, WA, USA, 2–4 April 2014; pp. 533–546.
6. Patel, P.; Bansal, D.; Yuan, L.; Murthy, A.; Greenberg, A.; Maltz, D.A.; Kern, R.; Kumar, H.; Zikos, M.; Wu, H.; et al. Ananta: Cloud Scale Load Balancing. In Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM, SIGCOMM '13, Hong Kong, China, 12–16 August 2013; pp. 207–218.
7. Joseph, D.A.; Tavakoli, A.; Stoica, I. A Policy-aware Switching Layer for Data Centers. In Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication, SIGCOMM '08, Seattle, WA, USA, 17–21 August 2018; pp. 51–62.
8. Son, J.; Buyya, R. A Taxonomy of Software-Defined Networking (SDN)-Enabled Cloud Computing. *ACM Comput. Surv.* **2018**, *51*, 59. [CrossRef]
9. Nobach, L.; Hohlfeld, O.; Hausheer, D. New Kid on the Block: Network Functions Visualization: From Big Boxes to Carrier Clouds. *SIGCOMM Comput. Commun. Rev.* **2018**, *46*, 7. [CrossRef]
10. Qazi, Z.A.; Tu, C.C.; Chiang, L.; Miao, R.; Sekar, V.; Yu, M. SIMPLE-fying Middlebox Policy Enforcement Using SDN. In Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM, SIGCOMM '13, Hong Kong, China, 12–16 August 2013; pp. 27–38.
11. Fei, X.; Liu, F.; Zhang, Q.; Jin, H.; Hu, H. Paving the Way for NFV Acceleration: A Taxonomy, Survey and Future Directions. *ACM Comput. Surv.* **2020**, *53*, 73. [CrossRef]
12. Fulber-Garcia, V.; Duarte, E.P.; Huff, A.; dos Santos, C.R. Network Service Topology: Formalization, Taxonomy and the CUSTOM Specification Model. *Comput. Netw.* **2020**, *178*, 107337. [CrossRef]
13. Savi, M.; Tornatore, M.; Verticale, G. Impact of processing costs on service chain placement in network functions virtualization. In Proceedings of the 2015 IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN), San Francisco, CA, USA, 18–21 November 2015; pp. 191–197. [CrossRef]
14. Cerović, D.; Del Piccolo, V.; Amamou, A.; Haddadou, K.; Pujolle, G. Fast Packet Processing: A Survey. *IEEE Commun. Surv. Tutor.* **2018**, *20*, 3645–3676. [CrossRef]
15. Li, P.; Wu, X.; Ran, Y.; Luo, Y. Designing Virtual Network Functions for 100 GbE Network Using Multicore Processors. In Proceedings of the 2017 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS), Beijing, China, 18–19 May 2017; pp. 49–59. [CrossRef]
16. Li, Y.; Miao, R.; Kim, C.; Yu, M. FlowRadar: A Better NetFlow for Data Centers. In Proceedings of the 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16), Santa Clara, CA, USA, 16–18 March 2016; pp. 311–324.
17. Palkar, S.; Lan, C.; Han, S.; Jang, K.; Panda, A.; Ratnasamy, S.; Rizzo, L.; Shenker, S. E2: A Framework for NFV Applications. In *Proceedings of the 25th Symposium on Operating Systems Principles*; Association for Computing Machinery: New York, NY, USA, 2015; pp. 121–136. [CrossRef]
18. Kancherla, M.; Jayant, J.; Sengupta, A. Bypassing a Load Balancer in a Return Path of Network Traffic. US Patent 10,212,071, 19 February 2019.

19. OpenFlow Discovery Protocol and Link Layer Discovery Protocol. Figshare. Available online: https://groups.geni.net/geni/wiki/OpenFlowDiscoveryProtocol#WhatisLLDP (accessed on 3 August 2022).
20. Mahajan, R.; Wetherall, D.; Anderson, T. Understanding BGP Misconfiguration. *SIGCOMM Comput. Commun. Rev.* **2002**, *32*, 3–16. [CrossRef]
21. Network Firewall Security Management Software. Available online: https://www.solarwinds.com/security-event-manager/use-cases/firewall-security-management (accessed on 3 August 2022).
22. Handigol, N.; Heller, B.; Jeyakumar, V.; Lantz, B.; McKeown, N. Reproducible Network Experiments Using Container-based Emulation. In Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies, CoNEXT '12, Nice, France, 10–13 December 2012; pp. 253–264.
23. POX. Available online: https://github.com/noxrepo/pox (accessed on 3 August 2022).
24. iPerf. Available online: https://iperf.fr/ (accessed on 3 August 2022).
25. Anderson, J.W.; Braud, R.; Kapoor, R.; Porter, G.; Vahdat, A. xOMB: Extensible Open Middleboxes with Commodity Servers. In Proceedings of the Eighth ACM/IEEE Symposium on Architectures for Networking and Communications Systems, ANCS '12, Austin, TX, USA, 29–30 October 2012; pp. 49–60.
26. Kohler, E.; Morris, R.; Chen, B.; Jannotti, J.; Kaashoek, M.F. The Click Modular Router. *ACM Trans. Comput. Syst.* **2000**, *18*, 263–297. [CrossRef]
27. Martins, J.; Ahmed, M.; Raiciu, C.; Olteanu, V.; Honda, M.; Bifulco, R.; Huici, F. ClickOS and the Art of Network Function Virtualization. In Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation, NSDI'14, Seattle, WA, USA, 2–4 April 2014; pp. 459–473.
28. Hwang, J.; Ramakrishnan, K.K.; Wood, T. NetVM: High Performance and Flexible Networking Using Virtualization on Commodity Platforms. *IEEE Trans. Netw. Serv. Manag.* **2015**, *12*, 34–47. [CrossRef]
29. Li, H.; Wu, C.; Sun, G.; Zhang, P.; Shan, D.; Pan, T.; Hu, C. Programming Network Stack for Middleboxes with Rubik. In Proceedings of the 18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21), online, 12–14 April 2021; pp. 551–570.
30. language, P. Available online: https://p4.org/ (accessed on 3 August 2022). .
31. Kaur, K.; Mangat, V.; Kumar, K. A comprehensive survey of service function chain provisioning approaches in SDN and NFV architecture. *Comput. Sci. Rev.* **2020**, *38*, 100298. [CrossRef]
32. Poh, G.S.; Divakaran, D.M.; Lim, H.W.; Ning, J.; Desai, A. A Survey of Privacy-Preserving Techniques for Encrypted Traffic Inspection over Network Middleboxes. *arXiv* **2021**, arXiv:2101.04338.
33. Liu, G.; Ren, Y.; Yurchenko, M.; Ramakrishnan, K.K.; Wood, T. Microboxes: High Performance NFV with Customizable, Asynchronous TCP Stacks and Dynamic Subscriptions. In Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication, SIGCOMM '18, Budapest, Hungary, 20–15 August 2018; pp. 504–517. [CrossRef]
34. Tootoonchian, A.; Panda, A.; Lan, C.; Walls, M.; Argyraki, K.; Ratnasamy, S.; Shenker, S. ResQ: Enabling SLOs in Network Function Virtualization. In Proceedings of the 15th USENIX Conference on Networked Systems Design and Implementation, NSDI'18, Renton, WA, USA, 9–11 April 2018; pp. 283–297.
35. Li, Y.; Chen, M. Software-Defined Network Function Virtualization: A Survey. *IEEE Access* **2015**, *3*, 2542–2553. [CrossRef]
36. Jamshed, M.; Moon, Y.; Kim, D.; Han, D.; Park, K. MOS: A Reusable Networking Stack for Flow Monitoring Middleboxes. In Proceedings of the 14th USENIX Conference on Networked Systems Design and Implementation, NSDI'17, Boston, MA, USA, 27–29 March 2017; pp. 113–129.
37. Khalid, J.; Gember-Jacobson, A.; Michael, R.; Abhashkumar, A.; Akella, A. Paving the Way for NFV: Simplifying Middlebox Modifications Using StateAlyzr. In Proceedings of the 13th Usenix Conference on Networked Systems Design and Implementation, NSDI'16, Santa Clara, CA, USA, 16–18 March 2016; pp. 239–253.
38. Mehraghdam, S.; Keller, M.; Karl, H. Specifying and placing chains of virtual network functions. In Proceedings of the 2014 IEEE 3rd International Conference on Cloud Networking (CloudNet), Luxembourg, 8–10 October 2014; pp. 7–13. [CrossRef]
39. Moens, H.; Turck, F.D. VNF-P: A model for efficient placement of virtualized network functions. In Proceedings of the 10th International Conference on Network and Service Management (CNSM) and Workshop, Rio de Janeiro, Brazil, 17–21 November 2014; pp. 418–423. [CrossRef]
40. Sahhaf, S.; Tavernier, W.; Rost, M.; Schmid, S.; Colle, D.; Pickavet, M.; Demeester, P. Network service chaining with optimized network function embedding supporting service decompositions. *Comput. Netw.* **2015**, *93*, 492–505. doi: 10.1016/j.comnet.2015.09.035. [CrossRef]
41. Moens, H.; De Turck, F. Customizable Function Chains: Managing Service Chain Variability in Hybrid NFV Networks. *IEEE Trans. Netw. Serv. Manag.* **2016**, *13*, 711–724. [CrossRef]
42. Li, Y.; Xuan Phan, L.T.; Loo, B.T. Network functions virtualization with soft real-time guarantees. In Proceedings of the IEEE INFOCOM 2016—The 35th Annual IEEE International Conference on Computer Communications, San Francisco, CA, USA, 10–14 April 2016; pp. 1–9. [CrossRef]
43. Eramo, V.; Miucci, E.; Ammar, M.; Lavacca, F.G. An Approach for Service Function Chain Routing and Virtual Function Network Instance Migration in Network Function Virtualization Architectures. *IEEE/ACM Trans. Netw.* **2017**, *25*, 2008–2025. [CrossRef]

44. Zhang, Q.; Xiao, Y.; Liu, F.; Lui, J.C.; Guo, J.; Wang, T. Joint Optimization of Chain Placement and Request Scheduling for Network Function Virtualization. In Proceedings of the 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS), Atlanta, GA, USA, 5–8 June 2017; pp. 731–741. [CrossRef]
45. Liu, J.; Li, Y.; Zhang, Y.; Su, L.; Jin, D. Improve Service Chaining Performance with Optimized Middlebox Placement. *IEEE Trans. Serv. Comput.* **2017**, *10*, 560–573. [CrossRef]
46. Pei, J.; Hong, P.; Pan, M.; Liu, J.; Zhou, J. Optimal VNF Placement via Deep Reinforcement Learning in SDN/NFV-Enabled Networks. *IEEE J. Sel. Areas Commun.* **2020**, *38*, 263–278. [CrossRef]
47. Ma, W.; Beltran, J.; Pan, D.; Pissinou, N. Placing Traffic-Changing and Partially-Ordered NFV Middleboxes via SDN. *IEEE Trans. Netw. Serv. Manag.* **2019**, *16*, 1303–1317. [CrossRef]
48. Liu, A.X.; Torng, E.; Meiners, C.R. Firewall Compressor: An Algorithm for Minimizing Firewall Policies. In Proceedings of the INFOCOM 2008—The 27th Conference on Computer Communications, Phoenix, AZ, USA, 13–18 April 2008.
49. Zhang, S.; Mahmoud, A.; Malik, S.; Narain, S. Verification and synthesis of firewalls using SAT and QBF. In Proceedings of the 2012 20th IEEE International Conference on Network Protocols (ICNP), Austin, TX, USA, 30 October–2 November 2012; pp. 1–6.
50. Acharya, H.B.; Gouda, M.G. Firewall verification and redundancy checking are equivalent. In Proceedings of the INFOCOM, Shanghai, China, 10–15 April 2011; pp. 2123–2128.
51. Benzekki, K.; El Fergougui, A.; Elbelrhiti Elalaoui, A. Software-defined networking (SDN): A survey. *Secur. Commun. Netw.* **2016**, *9*, 5803–5833. [CrossRef]
52. AlGhadhban, A.; Shihada, B. FLight: A Fast and Lightweight Elephant-Flow Detection Mechanism. In Proceedings of the 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS), Vienna, Austria, 2–6 July 2018; pp. 1537–1538. [CrossRef]
53. Bouacida, N.; Alghadhban, A.; Alalmaei, S.; Mohammed, H.; Shihada, B. Failure mitigation in software defined networking employing load type prediction. In Proceedings of the 2017 IEEE International Conference on Communications (ICC), Paris, France, 21–25 May 2017; pp. 1–7. [CrossRef]
54. Roy, A.; Zeng, H.; Bagga, J.; Porter, G.; Snoeren, A.C. Inside the Social Network's (Datacenter) Network. *SIGCOMM Comput. Commun. Rev.* **2015**, *45*, 123–137. [CrossRef]
55. Terzi, C.; Korpeoglu, I. 60 GHz wireless data center networks: A survey. *Comput. Netw.* **2021**, *185*, 107730. [CrossRef]
56. Celik, A.; AlGhadhban, A.; Shihada, B.; Alouini, M.S. Design and Provision of Traffic Grooming for Optical Wireless Data Center Networks. *IEEE Trans. Commun.* **2019**, *67*, 2245–2259. [CrossRef]
57. AlGhadhban, A.; Celik, A.; Shihada, B.; Alouini, M.S. LightFDG: An Integrated Approach to Flow Detection and Grooming in Optical Wireless DCNs. *IEEE Trans. Netw. Serv. Manag.* **2020**, *17*, 1153–1166. [CrossRef]
58. AlGhadhban, A. F4Tele: FSO for data center network management and packet telemetry. *Comput. Netw.* **2021**, *186*, 107711. [CrossRef]
59. AlGhadhban, A. FSO Clusters for Data Center Network Management and Packet Telemetry. In *Proceedings of the SIGCOMM '20 Poster and Demo Sessions*; Association for Computing Machinery: New York, NY, USA, 2020; pp. 9–11.