Received: 18 October 2018 | Revised: 28 February 2019 | Accepted: 28 April 2019

DOI: 10.4218/etrij.2018-0565

ORIGINAL ARTICLE

ETRI Journal WILEY

Toward manageable middleboxes in software-defined networking

Ehsan Zadkhosh[1] | **Hossein Bahramgiri**[1] | **Masoud Sabaei**[2]

[1]Information and Communication Technology Department, Malekashtar University of Technology, Tehran, Iran

[2]Department of Computer Engineering and Information Technology, Amirkabir University of Technology, Tehran, Iran

Correspondence
Hossein Bahramgiri, Information and Communication Technology Department, Malekashtar University of Technology, Tehran, Iran.
Email: bahramgiri@mut.ac.ir

Software-defined networking (SDN) acts as a centralized management unit, especially in a network with devices that operate under the transport layer of the OSI model. However, when a network with layer 7 middleboxes (MBs) is considered, current SDNs exhibit limitations. As such, to achieve a real-centralized management unit, a new architecture is required that decouples the data and control planes of all network devices. In this report, we propose such a complementary architecture to the current SDN in which SDN-enabled MBs are included along with contemporary SDN-enabled switches. The management unit of this architecture improves network performance and reduces routing cost by considering the status of the MBs during flow forwarding. This unit consists of the following two parts: an SDN controller (SDNC) and a middlebox controller (MBC). The latter selects the best MBs for each flow and the former determines the best path according to its routing algorithm and provides information via the MBC. The results show that the proposed architecture improved performance because the utilization of all network devices including MBs is manageable.

KEYWORDS
load-balancing, middlebox, network utilization, software-defined networking

1 | INTRODUCTION

Currently, networks make use of a variety of middleboxes (MBs) to provide enhanced performance and security [1]. Moreover, policy changes or the addition of new functionalities to a network usually requires additional MBs. As reported in [1], in a network with 80K users, the total number of MBs is to some extent of the order of the number of switching elements.

An MB is any device in the path of a datagram that performs functions other than the normal tasks of an IP router [2], such as route discovery [3]. MBs could terminate a flow and establish another one or/and make changes to it. They could also act as an embedded action in other devices, but they are never the destination of a session [2]. These devices can be classified based on their usage as shown in Table 1 [4].

MBs have different kinds of features [2]. Among these features, the open system interconnection (OSI) layer in which an MB operates is more important to our study. Figure 1 contains a list of MBs that are frequently in use in networks and their operational OSI layer [2]. Currently, these MBs are expensive, inflexible, and special-purpose. They are also closed source devices with no flexible management interface. Consequently, to manage the MBs of specific manufacturers, a specialized team is required. As a result, network management and evaluation become complex, expensive, and error-prone.

Given that MBs drop any unknown traffic, security improvements through new protocols can only occur by improving the existing protocols. As reported in [2], 16 out of 22 MBs require an upgrade of the existing protocol. Therefore, the newly generated traffic of most MBs should pass through tunnels. Otherwise, subsequent MBs will drop their flow [5].

To address these problems, the current solutions focus on the mounting of processes to general-purpose software on generic platforms such as servers. This facilitates new network services without hardware changes [6,7]. Software-defined networking (SDN) is a practical and extensible solution for this category.

The SDN architecture decouples data and the control unit of network devices to provide a central management unit. One of the responsibilities of this unit is to choose the path along which flow should forward, in a way that guarantees the desired security and policies of the network [8,9]. SDN provides many improvements for the network industry such as an open management interface, flexible topologies, and innovation speed-up. However, it does not provide any information on the status of MBs. Therefore, it can be stated that the current path selection in SDN is in progress without considering the MB's internal state. The absence of a complete view of the MBs' internal logic and the dynamic changes that MBs cause in packets, lead to routing ambiguity and non-optimal performance [10]. Moreover, in current networks, to control delay, link utilization should be less than 30% to 50%. Otherwise, the network is vulnerable to bursty traffic [11]. As such, current networks suffer from low utilization and poor sharing of resources.

TABLE 1 Classification of MBs

Function	Example MB
Security	Firewall, IDS, and IPS
Traffic shaping	Load balancer
Address dealing	NAT and traffic shaper
Network optimization	Cache and proxy

SDN acts based on matches in the header fields. Accordingly, although current SDN support actions up to layer 4 of the OSI model, it is incapable of managing operations that require payload checking at a reasonable speed. This is because it should process all actions in the controller (software-based). Therefore, we could classify network facilities into devices with activities up to layer 4 and those that act above this layer. Current SDNs cover the first group but vendor-specific MBs are still required for the second category. As illustrated in Figure 1, 19 out of 26 MBs have activities above layer 4, which means that current SDNs must be improved to cover all network devices.

In summary, the control unit of the current SDNs does not have a complete view of the network. Consequently, networks require a control unit with a comprehensive view in terms of increasing resource utilization, network performance, and the capability of online device addition/removal [12,13]. This controller will also reduce manual configurations and bugs [14]. As an example, consider Figure 2. In Figure 2, traffic distribution between servers only occurs when the administrator manually configures the load balancer (LB). Therefore, the addition of a new server means that the LB must be reconfigured.

In this report, we investigate the development of an SDN architecture in which not only the data and control planes of

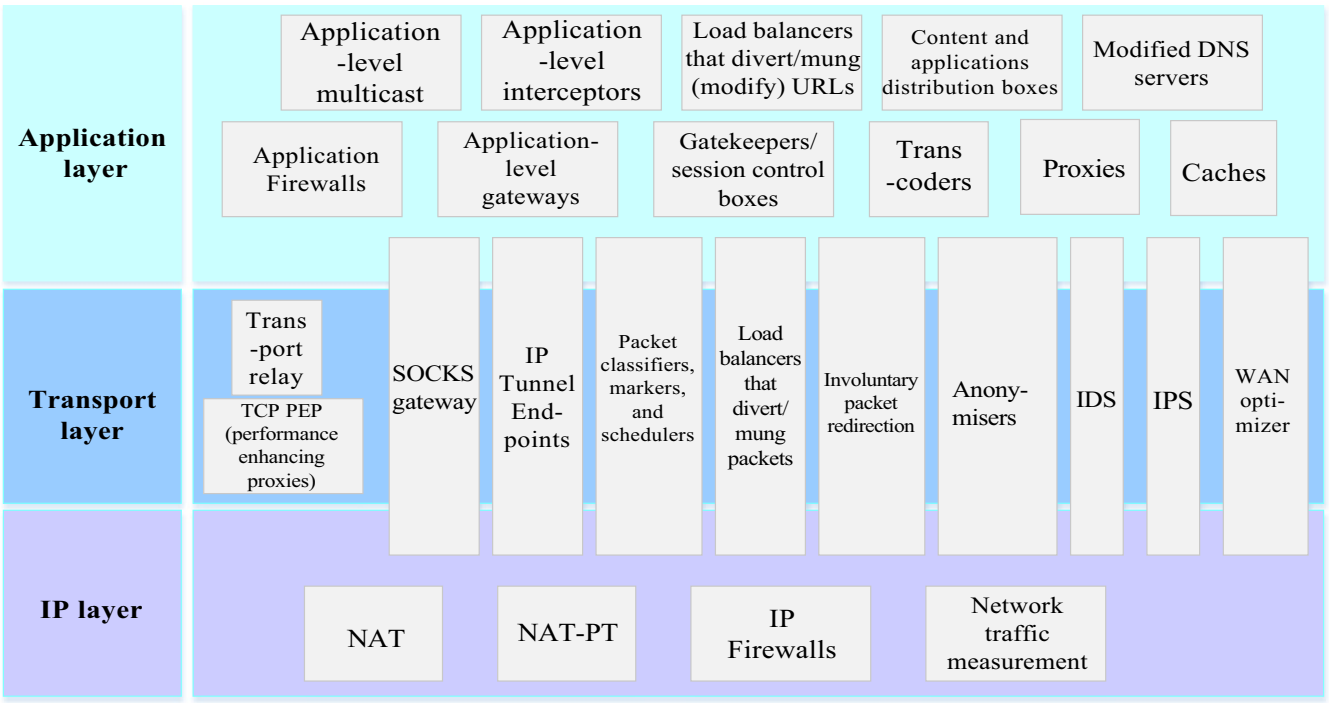

FIGURE 1 List of MBs

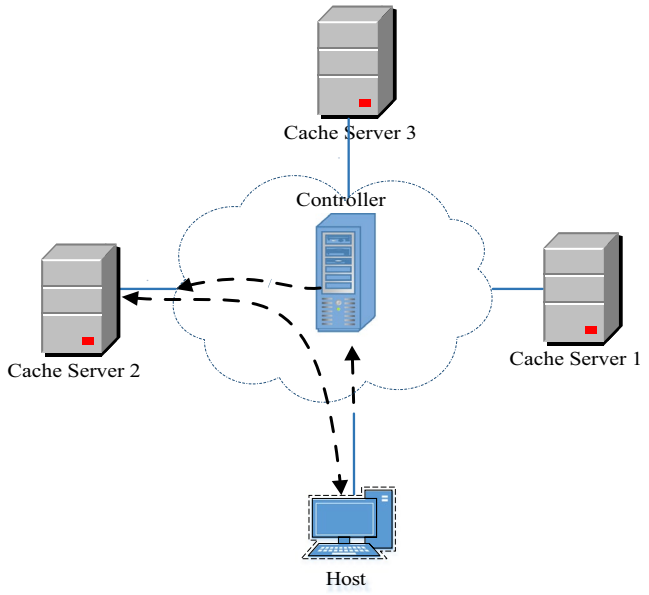

FIGURE 2 Distributed servers

switches are decoupled, but also MBs are SDN-based. In particular, we present an SDN controller that has access to the status of all network devices (layer one up to seven of the OSI model). This implies that the control unit has an overview of all network devices. Moreover, we implemented a routing algorithm that considers the state of the MBs including their load and total available resources during path selection. This architecture provides enhanced resource utilization and load balancing without any manual configuration in MBs and switches.

The remainder of this report is organized into several sections. Related works are reviewed in Section 3. Section 4 introduces the proposed architecture and a mathematical description of the model is given in Section 5. Sections 6 and 7 describe the implementation of the model and the results, respectively. Finally, the main conclusions of the study are presented in Section 8.

2 | RELATED WORK

Previous studies indicate that various kinds of policy chains are in use in different networks [1,15]. These chains can be called service function chains (SFCs) [15,16]. The proposed architecture in [15] attempts to find the path with a minimum cost for each SFC. This architecture is useful in traditional networks with fix places for MBs, and static paths. The main problem of this architecture is a lack of flexibility. Moreover, it does not make use of a central controller.

In the case where a central control unit is considered for MBs, [17] presents a low-performance architecture that uses software-based MBs. Reference [18] proposed an SDN-based policy enforcement layer to manage the traffic of MBs. This layer provides an automatic translation of policies into forwarding rules. Reference [19] proposed SDN-based MBs without implementation. Several other studies have investigated the use of SDN in the implementation of MBs, but they did not consider resource utilization and network performance [20,21]. There are many other proposed solutions for the improvement in the performance of networks that contain MBs. However, most of these methods did not consider the decoupling of the data and control planes of MBs [15,18–21]. We categorized these solutions as follows:

- Labeling: this employs unused fields of the packet header (such as MPLS in local networks) to label the flow of a specific policy chain [20]. These labels help the MBs to differentiate between different flows of different policies. Although this method could improve path selection, it is not scalable and lacks flexibility.
- MBs placement: in this method, if any change or failure occurs in network devices, new manual placement and rule setting should be manually performed by the network administrator or automatically by the network function virtualization (NFV) [7]. As previously indicated, these methods did not consider layer decoupling and the evolution of MBs [22,23].
- MB consolidation: although the goal of this method was to improve the placement of MBs, it does not consider their extension and improvement [19].
- Forward all actions to the controller: in this method, all current functional devices are cast aside. Moreover, this approach is software-based and is inhibited by low performance [24].
- Using verification tools: the purpose of verification tools such as HSA [13] and VeriFlow [12] is to verify the correct enforcement of policies. These tools are not intended for MB improvement and network performance optimization.

Based on proposed solutions and their drawbacks, there is the need for a new architecture with SDN-enabled MBs. This solution should also consider network performance while forwarding flows. The next section will discuss such a solution.

3 | PROPOSED ARCHITECTURE

As discussed earlier, to establish a central and integrated management unit for the entire network, it is necessary to decouple the data and control planes of MBs. In the current MBs, the control and data plane are integrated and vendor-specific. Ideally, in the decoupled version of MBs, the control plane will have a comprehensive and centralized view of all different MBs (for example, firewalls, network address translators [NATs], and intrusion detection systems [IDSs]). Figure 3 shows this ideal architecture of MBs. In

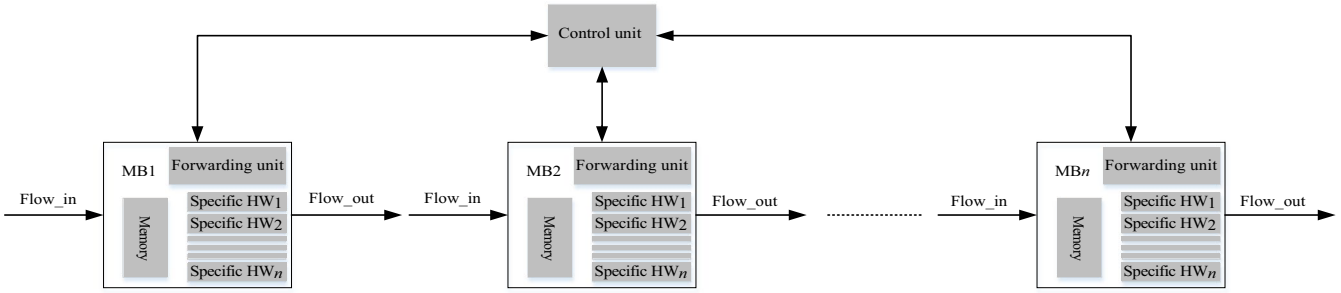

FIGURE 3 Ideal architecture of MBs

this case, there is no need to have different kinds of MBs. The decoupled data plane could play the role of various MBs according to configurations that are assigned by the centralized control plane. In the ideal architecture, to have different MBs, the controller could activate the appropriate specific hardware. In this way, a general-purpose MB could be changed to a specific-purpose MB, based on the network requirements.

According to the ideal architecture of MBs in an ideal network, all switches and MBs could have the same hardware and the control plane determines the role (a switch or a specific MB) of each piece of hardware based on the network policies. One of the obstacles of transitioning toward this ideal network is the need to replace all currently functional MBs during implementation. This replacement is expensive and it is a long-term decision; moreover, after initial role assignment, shifting from a switch to an MB could cause disorder in network's functionality. Considering a switch with thousands of rules for different flows, shifting from a switch to an MB means that all the rules should be omitted. The same problem could occur when changing an MB to a switch. As a result, the controller should carefully analyze policy changes and their consequences. Therefore, the proposed architecture of Figure 4 is more practical than the ideal architecture.

As previously discussed, the illustrated architecture in Figure 4 is the proposed SDN-based architecture. In this architecture, the control unit has access to MBs and it considers the network as a combination of two networks: switches and MB networks. Each of these networks has a specific controller. The middlebox controller (MBC) is responsible for the MBs' network and the SDN controller (SDNC) is responsible for the switching network. In this architecture, path selection is achieved according to the status of both networks. The following subsections describe the procedure of flow processing in our proposed architecture.

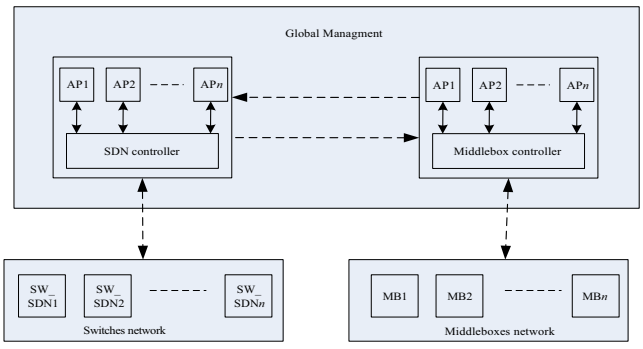

FIGURE 4 Proposed architecture

3.1 | Flow processing in the proposed architecture

We present an implementation of the proposed architecture based on the following assumptions. All flows entered the network through a switch. Switches and MBs could connect to each other through switches and MBs are capable of performing all the functionalities MBs (such as load balancing, NATing, and firewalling). Based on the arrival of a new flow, a classifier categorizes them based on network policies. Then according to the classification output, the controller selects the best route and sets rules on switches and MBs along this route. Finally, these devices forward the flow based on the inserted rules.

3.2 | Inside the control unit

As shown in Figure 5, each switch that receives any new flow sends it to the SDNC. The SDNC will then send the flow to the MBC for further inspection to determine whether the flow requires operations above layer 4. The MBC makes this decision according to the previously configured policies. If the MBC declares that no such functionalities are required, the SDNC will only set rules on switches to forward that flow; however, when a flow requires operations above the layer 4 MBs, the MBC should select the best MB(s) based on some criteria and inform the SDNC of the selection. Consequently, the MBC cooperates with the SDNC in deciding the best path. In this procedure, we could divide the flow from host A to B into two flows given that MB-X is the selected MB by MBC.

The MBC could select an MB based on several criteria such as delay, throughput, and so on. In all of these criteria, the resource constraint of the MBs is an important parameter;

therefore, the MBC considers this parameter to choose MBs. As such, the MBC considers CPU usage, free memory, and available bandwidth of each MB as constraints for the selection of an MB. Each MB reports its CPU and memory usage to the MBC. The status of bandwidth usage could be calculated from each MB's network interface card (NIC), and the cost of bandwidth could be based on the IEEE STP protocol [25].

Following this procedure will lead to balanced utilization of network resource which affects the total delay, the packet loss rate, and other criteria of the network. As such, we use network resources optimally, to optimize the entire network utilization. The improvements achieved by implementing our proposed architecture can be summarized as follows:

- Access to the MBs internal status.
- Real network-wide view including the status of MBs.
- Consideration of resource constraints and capabilities of devices while selecting the best path.
- Optimization of resource usage.
- Maximization of network utilization.

4 | MATHEMATICAL MODELING

Consider $M=\{m_1, m_2, \ldots, m_{|M|}\}$, $S=\{s_1, s_2, \ldots, s_{|S|}\}$, and $P=\{p_1, p_2, \ldots, p_{|P|}\}$ as the set of MBs, switches, and all assigned policies in the network, respectively. The control unit forward each flow according to a policy. Therefore, when a flow arrives, the first duty of the control unit is to categorize it based on its policy set. In the proposed architecture, the MBC is responsible for calculating the set of best MBs for each flow (based on the related policy). The MBC could calculate this set by taking several criteria into account, such as the load of the MBs. It then passes the best set to the SDNC, which then determines the best path (a combination of switches and best set of MBs) by using its path selection algorithm. Therefore, to process each policy in $P_k \in P$, the sequence of required MBs (M^{p_k}) is selected by the MBC.

$$M^{p_k}=\left[m_1^{p_k}, m_2^{p_k}, \ldots, m_{|M^{p_k}|}^{p_k}\right].$$

The SDNC selects the required switches (S^{p_k}) for that policy.

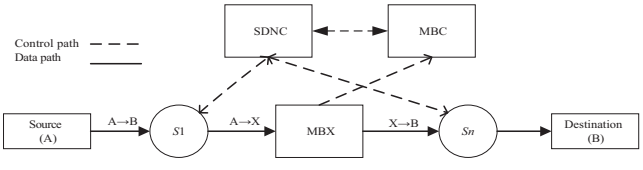

FIGURE 5 Processing procedure of a flow

$$S^{p_k}=\left[s_1^{p_k}, s_2^{p_k}, \ldots, s_{|S^{p_k}|}^{p_k}\right].$$

It should be noted that each policy could have different M^{p_k} and S^{p_k}. By calculating the set of the best MBs, the MBC attempts to minimize the maximum load of MBs to improve the network performance. Therefore, the objective of the management unit can be summarized in (1) as follows:

$$\forall m \in M: \text{Min}\left(\text{Max}\left(L_m\right)\right), \quad (1)$$

where L_m is the load of the mth MB that will be defined later in this section. To achieve this goal, the MBC requires the following information:

- MBs' functionality (functionalities): each MB could play more than one role. The MBC declares the functionality of each MB based on the policy set and status of the network. After the first role assignment, the MBC should memorize the role of each device.
- MBs' resources: hardware capability of each MB (m_i) is represented as R_{m_i} and consisting of three components:
  - $R_{m_i}^{\text{mem}}$: the total memory of m_i.
  - $R_{m_i}^{\text{cpu}}$: the total processing power of m_i.
  - $R_{m_i}^{\text{bw}}$: the total bandwidth of m_i.
- Load of each MB is represented as L_{m_i} ($0 \le L_{m_i} \le 1$) and consisting of three components:
  - $L_{m_i}^{\text{mem}}$: the memory usage in m_i.
  - $L_{m_i}^{\text{cpu}}$: the CPU usage in m_i.
  - $L_{m_i}^{\text{bw}}$: the bandwidth usage in m_i.
- Network Topology.
- Network policies.

Based on this information, the MBC selects MBs based on the following assumptions:

$$R_{m_i}=R_{m_i}^{\text{mem}}+R_{m_i}^{\text{cpu}}+R_{m_i}^{\text{bw}},$$

$$L_{m_i}=L_{m_i}^{\text{mem}}+L_{m_i}^{\text{cpu}}+L_{m_i}^{\text{bw}},$$

$$L_{m_i}^{\text{bw}}=T_{m_i}^{\text{in}}+T_{m_i}^{\text{out}},$$

where $T_{m_i}^{\text{in}}$ and $T_{m_i}^{\text{out}}$ are the input and output traffic of the m_i, respectively. Considering these assumptions, the following constraints are imposed to achieve Min(Max(L_m)):

$$L_{m_i}^{\text{mem}} \le R_{m_i}^{\text{mem}},$$

$$L_{m_i}^{\text{cpu}} \le R_{m_i}^{\text{cpu}},$$

$$L_{m_i}^{\text{bw}} \le R_{m_i}^{\text{bw}},$$

TABLE 2 Summary of symbols used in the proposed model

Symbol	Description
P	Set of all policies
M	Set of all MBs
S	Set of all switches
L_{m_i}	Load of ith MB
R_{m_i}	Total resources of ith MB
$T_{m_i}^{\text{in}}$	Input traffic of ith MB
$T_{m_i}^{\text{out}}$	Output traffic of ith MB
$L_{m_i}^{\text{bw}}$	Bandwidth usage in ith MB
$L_{m_i}^{\text{cpu}}$	CPU usage in ith MB
$L_{m_i}^{\text{mem}}$	Memory usage in ith MB
$R_{m_i}^{\text{mem}}$	Total available Memory in ith MB
$R_{m_i}^{\text{cpu}}$	Total available CPU in ith MB
$R_{m_i}^{\text{bw}}$	Total available bandwidth in ith MB
P_k	kth policy combination
Cost(P_k)	Cost of kth policy
M^{P_k}	Sequence of required MBs in P_k
S^{P_k}	Sequence of required switches in P_k
$L_{m_i}^{P_k}$	Added load to ith MB by processing kth policy

where $L_m^{P_k}$ represents the amount of load caused by P_k on m, and $|S^{P_k}|$ represents the number of required switches for P_k. The cost of forwarding flows of P_k is defined as follows:

$$\text{Cost}\left(P_k\right) = \sum_{m \in M^{P_k}} \left(L_{m_i}^{P_k} + 1\right) + |s^{P_k}|. \tag{2}$$

This equation consists of two parts. The first part represents the number of required MBs added based on their loads for forwarding P_k, and the second part represents the number of used switches that should be added to the cost of P_k. The number of switches and MBs are crucial aspects related to the cost of P_k, given that the objective is to obtain the shortest available path with the minimum load on the MBs. All the symbols used in the proposed model are summarized in Table 2.

5 | IMPLEMENTATION AND EVALUATION

To implement and test the functionality of the proposed architecture, we used the following tools in a Linux environment:

- *Mininet*: an emulator for creating and testing a realistic SDN-based network. For real-world testing and performance evaluation, the developed and tested codes on *Mininet* for a controller, modified switch, or host can be transitioned to a real system with minimal changes. Importantly this means that a design that works in *Mininet* can usually be transitioned directly to hardware switches for line-rate packet forwarding [26].
- *POX* Controller: an SDN controller with OpenFlow support. *POX* is Python-based and open source [27].
- Snort: an open source IDS that facilitates the capture of packets on a specific interface in addition to the analysis of their headers and the payload [28].

To implement the proposed architecture on *mininet*, a topology that consists of seven hosts was used. Two of these hosts are servers and the others play the role of clients. This topology consists of two networks. The first network (Net1) has m MBs and one server (MB1-1 to MB1-m and server1) while the second network (Net2) has n MBs and one server (MB2-1 to MB2-n and server 2). All switches and MBs are connected to the controller as shown in Figure 6.

Although having different policies is common in networks, in our implementation, only two policies are implemented: $H_n \rightarrow \text{IDS} \rightarrow H_m$ and $H_n \rightarrow \text{IDS} \rightarrow \text{SW} \rightarrow \text{IDS} \rightarrow H_m$. These policies dictate that traffic should pass through an IDS in each network.

Our implemented MBs are just IDSs and these MBs are customized versions of *mininet's* switches. We can run *snort* as a command in these switches and transform them into an IDS.

As mentioned in the previous chapter, the control unit requires the network topology to calculate the best path. To inform the controller about the topology, we use the *net* module of the *mininet*. In this way, as soon as we create our topology on the *mininet*, the *mininet* will provide the topology for the *POX*. After receiving the topology, the MBC recognizes MBs. To calculate the load and capabilities of each MB, we use the *psutil* package. Using this package, the MBC could have access to the internal status of MBs at any time (for example, the CPU usage of each MB). In this way, the management unit could view any topology or load changes of MBs.

In practical scenarios, given that the CPU and RAM utilization of the SDN devices can become a crucial issue; the controller can use secure shell (SSH) or other connections

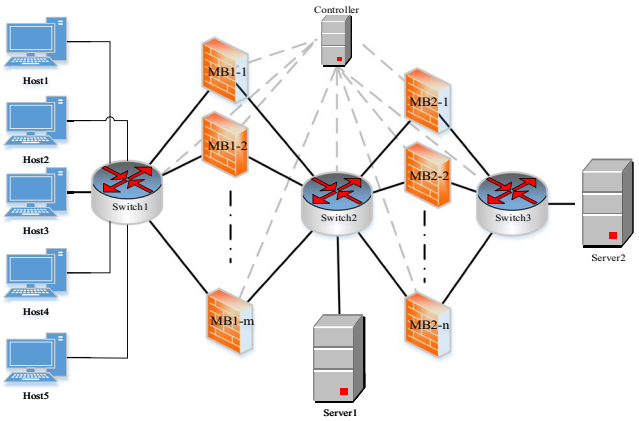

FIGURE 6 Test topology

to execute commands on SDN devices. In this way, the controller could have access to the CPU and RAM status of the devices.

We also implemented a path routing algorithm based on *Dijkstra* in the management unit of the proposed architecture as shown in Figure 7.

As illustrated in Figure 7, the *findBestPath(src, dst)* method works based on a weighted graph. This graph is the weighted version of the base graph that is passed to the controller by *mininet's net* module. To convert the base graph to a weighted version, we calculated the load of the MBs as shown in Figure 8, and then the edges that are connected to MBs based on the calculated loads were updated.

The path selection algorithm calculates the best path with minimum cost for all flows. The weighted version of the topology consists of the cost of traversing from any device (host, switch or MB) to any other device. Given that this weighted topology is accessible by the controller, any load changes in the MBs dramatically changes the weighted topology.

It should also be noted that based on the implemented path selection algorithm, we implicitly omitted unwanted topological loops and broadcasts. As noted in [8] and [29], approximately 30% to 40% of the network traffic is broadcast messages such as address resolution protocol (ARP) and dynamic host configuration protocol (DHCP).

```
def findBestPath(src, dst):
    weightedGraph = Graph()
    for node1 in distances.keys():
        for node2 in distances[node1].keys():
            weightedGraph.add_edge(node1, node2, {'cost': distances
[node1][node2]})
    cost_func = lambda u, v, e, prev_e: e['cost']
    return find_path(weightedGraph, src, dst, cost_func=cost_func)
```

FIGURE 7 Path routing algorithm of the proposed architecture

```
def updateDistance():
    for node in middelboxList.keys():
        middelbox = middelboxList[node]
    cpuUsage = middelbox.cpu_percent()
    middelboxDistance[node]= cpuUsage + 1 for node1 in
distances.keys():
        for node2 in distances[node1].keys():
            if node2 in middelboxList.keys():
                distances[node1][node2] = middelboxDistance[node2]
            if node1 in middelboxList.keys():
                distances[node1][node2] = middelboxDistance[node1]
```

FIGURE 8 Script for load calculation of MBs and updating of the weighted graph

To test our proposed architecture, we developed three python scripts. The first script creates the depicted topology in Figure 6, while the second and third scripts are the *client.py* and *server.py*. The client sends packets to the server with predefined port numbers. Our servers on the other side listen to the requests from the clients.

To evaluate the performance of the proposed architecture, we measured the CPU load of MBs by passing base topology and weighted topology to the path selection algorithm. By using the base topology, the path selection algorithm only considers hop counts from the source to the destination to calculate the best path. However, considering the weighted version of the topology, the path selection algorithm will include the status of each MB to calculate the best path.

Referring to [11] we attempted to maintain the total utilization of each MB below 80%. Therefore, any MB with utilization at approximately 80% was not selected by MBC. Considering this threshold there are two options: the selection and use of a path until one of its related MBs reaches 80% utilization or the selection of paths such that the load increase in all MBs in a coordinated manner. We chose the second option in our implementation.

6 | RESULTS AND DISCUSSION

To test our proposed architecture with the topology depicted in Figure 6, we designed two categories of scenarios. In the first case, we considered Net1 with five MBs. In the second category, we consider the entire topology with different numbers of MBs in Net1 and Net2. In these scenarios, we configured MBs and switches to flush away their internal rules after 300 seconds (hard-timeout). The clients initiate continuous requests for one hour to multi-thread servers. Clients emit requests every 3 seconds. We monitored the total network status and CPU usage of MBs, during the clients' connections. In this version of our implementation, we ignored the RAM usage and bandwidth utilization.

In the first scenario of the first category, there were two simultaneous flows to the server1 ($f1$ and $f2$), from two clients ($c1$ and $c2$). When the client $c1$ makes the first request at time 0, all MBs of Net1 are free of any load. Therefore, given that there is no rule for the switches and MBs, the switch forwards the first packet of $f1$ to the controller. Based on our path selection algorithm and the information provided by the MBC, the SDNC will select the path that contains MB1-1 as the best path.

After 3 seconds, $f2$ arrives. At this time, the load of MB1-1 is approximately 25%, and other MBs of Net1 have no load. As a result, MBC chooses MB1-2 for the processing of $f2$. After

the timeout, $f1$ requires path selection one more time. At this time, given that MB1-1 is busy with the remaining processes of the previous process and MB1-2 is busy with $f2$, MBC chooses MB1-3 as the best path for $f1$. The same is true for $f2$ when the rules expire for MB1-2. Therefore, the SDNC chooses MB1-4 to process $f2$. In this scenario, in the first episode (300 seconds), MB1-1 and MB1-2 are the best MBs for $f1$ and $f2$, while in the second episode, MB1-3 and MB1-4 are the selected MBs for $f1$ and $f2$. This scenario is repeated in the next episodes. Figure 9 shows the average CPU usage of MBs during the test time.

In the second scenario of the first category, three clients send concurrent requests to server1. In the first episode, the MBC chooses MB1-1, MB1-2, and MB1-3 for $f1$, $f2$, and $f3$, respectively. In the second episode, MB1-1 and then MB1-2 (with a 3-second delay) flush away their internal rules. Therefore, the path selection algorithm chooses the best paths for flow 1 and 2. Given that MB1-1 and MB1-2 are still busy with previously assigned processes, the algorithm chooses MB1-4 and MB1-5 for them. As depicted in Figure 10, three MBs are busy in each episode and the processing load is distributing among all the MBs.

In the third scenario of the first category, we consider five concurrent flows ($f1$, $f2$, $f3$, $f4$, and $f5$) to server1. In all episodes of this scenario, MB1-1 to MB1-5 is chosen for $f1$ to $f5$, respectively. As depicted in Figure 11, the load of all MBs is in the same order.

Based on the results of the scenarios in the first category, the average load of MBs in our architecture with two, three, and five concurrent flows could be summarized as Figure 12.

As previously mentioned, in the second category of our testing scenarios, we consider the entire topology of Figure 6 with a different number of MBs in Net1 and Net2 as depicted in Table 3. In this category, we send five concurrent flows from clients to server1 or server2, for an hour.

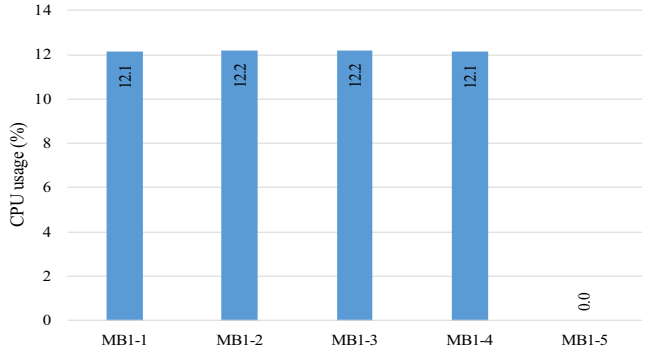

FIGURE 9 Average MBs' CPU usage with two simultaneous flows using weighted topology

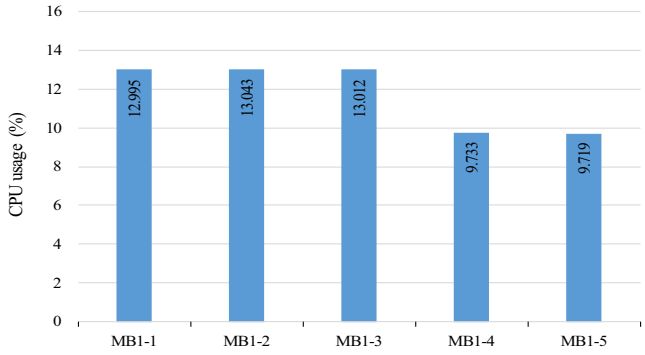

FIGURE 10 Average CPU usage of MBs with three simultaneous flows using weighted topology

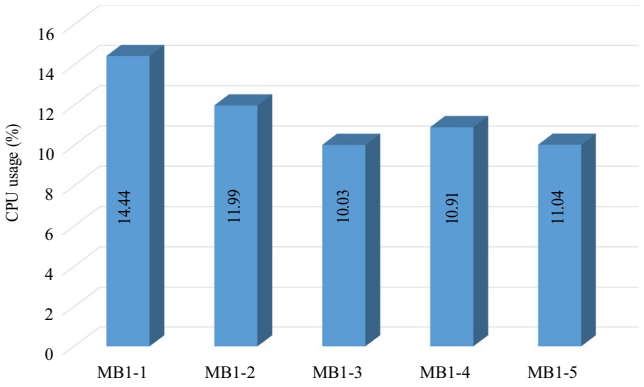

FIGURE 11 CPU usage of MBs with five simultaneous flows using weighted topology

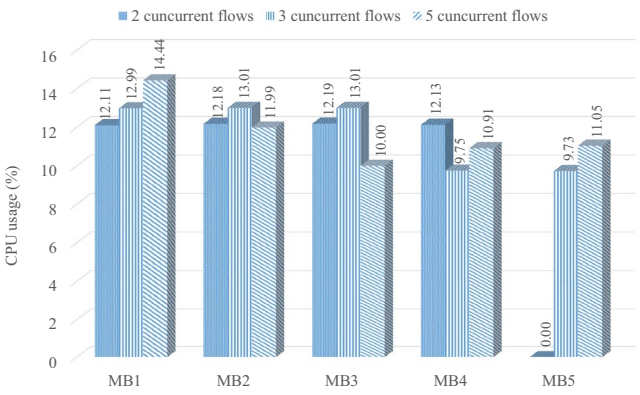

FIGURE 12 Average load of MBs considering their internal status

TABLE 3 Number of MBs in Net1 and Net2

No. MBs in Net1	No. MBs in Net2	Scenario name
1	0	S1-0
5	0	S5-0
2	0	S2-0
5	2	S5-2
4	3	S4-3
3	4	S3-4

TABLE 4 Load of MBs in different scenarios

Scenario name	Net1					Net2			
	L_{MB1-1}	L_{MB1-2}	L_{MB1-3}	L_{MB1-4}	L_{MB1-5}	L_{MB2-1}	L_{MB2-2}	L_{MB2-3}	L_{MB2-4}
S1-0	48.21	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
S5-0	14.44	11.99	10.03	10.91	11.04	N/A	N/A	N/A	N/A
S2-0	37.83	36.59	N/A	N/A	N/A	N/A	N/A	N/A	N/A
S5-2	15.88	13.96	13.18	12.94	12.01	35.44	34.79	N/A	N/A
S4-3	20.02	17.12	14.57	13.75	N/A	22.60	23.11	21.57	N/A
S3-4	22.69	19.77	18.25	N/A	N/A	16.39	14.95	16.73	15.47

When the number of MBs is zero in Net2, this implies that the clients' requests are toward server1; otherwise, server2 is the destination of the flows. Table 4 shows the average load of the MBs for different scenarios of this test. In this table, extra MBs are marked with N/A (not available). As shown in this table, it is considered that the CPU constraint in path selection could result in a fair load distribution.

After testing the proposed architecture with the weighted topology, we also examine it with the base topology. In this examination, we considered Net1 with five MBs and five concurrent flows. The results show that all flows are forwarded through MB1 and the load of this MB is approximately 50% during the test time. However, all four other MBs are free of load and are useless. The result of this test is in the same range as the result of S1-0, for which we only have an MB. In this situation, increasing the number of concurrent flows will only increase the MB1's CPU usage. Considering that MB1 has limited CPU power, this method of path selection makes MB1 a point of failure.

Considering Figure 13 and the load distribution of S5-0 in Table 4, the results justify that the decreased load of MB1 in our architecture, as shown in Figure 13, offloads among other MBs of S5-0. Therefore, when we include the status of MBs in path selection, network resource utilization is approximately 30% more optimize than when we ignore the status of the MBs. As such, using the dynamicity of SDN in all network devices results in better resource utilization. It can also be noted that ignoring the status of MBs in network decisions or using static configurations could cause a bottleneck and poor resource utilization. Therefore, in a real situation with more concurrent flows, ignoring the status of MBs leads to long queues, long delays, more packet losses, and poor network performance.

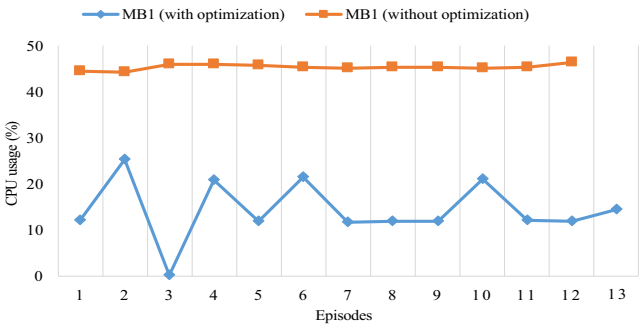

FIGURE 13 MB1 CPU usage with five simultaneous flows with and without considering the status of MB in path selection

7 | CONCLUSION

The current structure of SDN is unable to provide a centralized management unit in a network with layer 7 MiddleBoxes (MBs). As a result, to improve the network and to make it more manageable, in this report we proposed a new architecture that decouples the data and control plane of all network devices. To improve the performance of networks, this architecture considers MBs in addition to switches during the path selection procedure. The control plane of this architecture consists of two separated parts: SDN Controller (SDNC) and middlebox controller (MBC). The MBC is responsible for selecting the best MBs for flows, based on the predefined policies and status of MBs. The MBC chooses MBs, in a way that maximizes network utilization. We implemented and evaluated our architecture on *mininet*. In our implementation, MBs were an evolved version of OpenFlow switches. The results indicated that considering the status of MBs in flow forwarding improves CPU utilization by approximately 30%. In our implementation, we did not consider RAM usage and bandwidth utilization. Considering these constraints in path selection could lead to improved network performance. In future work, we will consider memory and bandwidth utilization in flow forwarding. We will also compare our weighted path selection algorithm with more dynamic load balancing algorithms such as Round Robin.

ORCID

Hossein Bahramgiri https://orcid.org/0000-0001-6374-2173

REFERENCES

1. V. Sekar et al., *The middlebox manifesto*, in Proc. ACM Workshop Hot Topics Netw., Cambridge, MA, USA, Nov. 2011, pp. 21:1–6.
2. RFC3234, *Middleboxes: Taxonomy and issues*, 2002.
3. RFC1812, *Requirements for IP version 4 routers*, 1995.
4. SIGCOMM, *ACM SIGCOMM workshop on Hot Topics in Middleboxes and Network function virtualization-HotMiddlebox*, 2015, available at http://conferences.sigcomm.org/sigcomm/2015/hotmiddlebox.php.
5. M. Honda et al., *Is it still possible to extend TCP*?, in Proc. ACM SIGCOMM Conf. Internet Meas. Conf., Berlin, Germany, Nov. 2011, pp. 181–194.
6. ONF, *OpenFlow® Switch Specification Ver 1.5.1*, available at https://www.opennetworking.org/technicalcommunities/areas/specification
7. C. Cui et al., *Network functions virtualisation*, Course.Ipv6.Club.Tw.
8. M. Casado et al., *Ethane*, ACM SIGCOMM Comput. Commun. Rev. **37** (2007), no. 4, 1–12.
9. D.A. Joseph, A. Tavakoli, and I. Stoica, *A policy-aware switching layer for data centers*, ACM SIGCOMM Comput. Commun. Rev. **38** (2008), no. 4, 51–62.
10. A. Gember et al., *Stratos: A network-aware orchestration layer for virtual middleboxes in clouds*, 2013, arXiv:1305.0209.
11. C. -Y. Hong et al., *Achieving high utilization with software-driven WAN*, in Proc. ACM SIGCOMM 2013 Conf. SIGCOMM, Hong Kong, China, Aug. 2013, pp. 15–26.
12. A. Khurshid et al., *Veriflow: Verifying network-wide invariants in real time*, ACM SIGCOMM Comput. Commun. Rev. **42** (2012), no. 4, 467–472.
13. P. Kazemian, G. Varghese, and N. McKeown, *Header space analysis: Static checking for networks*, in Proc. USENIX Conf. Netw. Syst. Des. Implement, San Jose, CA, USA, Apr. 2012, p. 5.
14. A. Gember, T. Benson, and A. Akella, *Challenges in unifying control of middlebox traversals and functionality*, in Proc. Large-Scale Distributed Syst. Middleware (LADIS), Madeira, Portugal, 2012, pp. 1–2.
15. IETF, *Service function chaining: Framework & architecture*, Internet-Draft, 2014, pp. 1–24.
16. IETF, *Service function chaining problem statement*, Internet-Draft, 2015, pp. 1–19.
17. V. Sekar et al., *Network-wide deployment of intrusion detection and prevention systems*, in Proc. Int. Conf. (Co-NEXT), Philadelphia, PA, USA, 2010, pp. 18:1–12.
18. Z.A. Qazi et al., *SIMPLE-fying middlebox policy enforcement using SDN*, ACM SIGCOMM Comput. Commun. Rev. **43** (2013), no. 4, 27–38.
19. V. Sekar et al., *Design and Implementation of a Consolidated Middlebox Architecture*, in Proc. USENIX Conf. Netw. Syst. Des. Implement., San Jose, CA, USA, Apr. 2012, pp. 1–14.
20. S.K. Fayazbakhsh et al., *FlowT ags: enforcing network-wide policies in the presence of dynamic middlebox actions*, in Proc. ACM SIGCOMM Workshop on Hot topics Softw. Defined Netwo. – HotSDN, Hong Kong China, Aug. 2013, pp. 19–24 .
21. P. Patel et al., *Ananta: Cloud scale load balancing*, ACM SIGCOMM Comput. Commun. Rev. **43** (2013), no. 4, 207–218.
22. J. Sherry et al., *Making middleboxes someone else's problem*, ACM SIGCOMM Comput. Commun. Rev. **42** (2012), no. 4, 13.
23. W. Ma et al., *SDN-based traffic aware placement of NFV middleboxes*, IEEE Trans. Netw. Serv. Manag. **14** (2017), no. 3, 528–542.
24. E. Kohler et al., *The click modular router*, ACM Trans. Comput. Syst., **18** (2000), no. 3, 263–297.
25. IEEE, *802.1D-2004 - IEEE Standard for Local and metropolitan area networks: Media Access Control (MAC) Bridges*, 2004.
26. mininet, available at https://github.com/mininet/mininet
27. pox, available at https://github.com/noxrepo/pox
28. snort, available at https://www.snort.org/
29. T. Koponen et al., *Onix: A distributed control platform for large-scale production networks*, in Proc. USENIX Symp. Operating Syst. Des. Implement., Vancouver, Canada, Oct. 2010, pp. 1–6.

AUTHOR BIOGRAPHIES

Ehsan Zadkhosh received his MS degree in computer engineering from Razi University, Kermanshah, Iran in 2014 and he is pursuing his PhD degree at Malekashtar University of Technology. His research interest includes Software Defined Networking, Internet of Things and performance modeling.

Hossein Bahramgiri received the BS and MS degrees in 2000 and 2003 respectively, from Sharif University of Technology, Tehran, Iran, and the PhD degree in 2010 from the University of Tehran, Iran all in Electrical Engineering. He has been an assistant professor at the Malekashtar University of Technology since 2016, and his research area includes information theory and security and communication networks.

Masoud Sabaei received his BS degree from Esfahan University of Technology, Esfahan, Iran, and his MS and PhD form Amirkabir University of Technology (Tehran Polytechnic), Tehran, Iran, all in the field of computer engineering in 1992, 1995 and 2000, respectively. Dr. Sabaei has been a professor in the computer engineering department at Amirkabir University of Technology (Tehran Polytechnic), Tehran, Iran since 2002. His research interests include software defined networking, internet of things, wireless networks, and telecommunication network management.