QoSE: Quality of SEcurity a network security framework with distributed NFV

Taejune Park, Yeonkeun Kim, Jaehyun Park, Hyunmin Suh, Byeongdo Hong and Seungwon Shin
School of Computing, KAIST, Republic of Korea
Email : {taejune.park, yeonk, jaehyun.park, hyunmin088, byeongdo, claude}@kaist.ac.kr

Abstract—**An effort to deploy security devices by the network provider has been increasing as the network is being exposed to various types of network attacks. However, network providers are incapable of handling all types of attacks as each security device is designed for a certain purpose. If an attack breaks out, only one particular device becomes busy in terms of resource usage while others being idle. Moreover, it is hard to adjust a level of security service with respect to the importance of network flow. To address these issues, we propose a new security solution, QoSE, which provides adaptive security services based on Network Function Virtualization (NFV). QoSE provides a capability to manage resource usage that the network flow is not concentrated on a specific node. We design QoSE considering a distributed NFV environment to avoid a single point of failure and a bottleneck problem. Our proposed solution has also shown a quick recovery from fault situation. In addition, we provide a novel resource optimization algorithm to operate security services efficiently. We have implemented a prototype system to verify our ideas and have checked that QoSE shows reasonable performance compared with a common device.**

I. Introduction

Network attacks are becoming various, intelligent, and elaborative so that the network providers need to pay more attention when deploying security devices. However, a conventional security device does not follow up the rapid changing of network attacks due to several reasons.

First, security devices are only designed to prevent certain types of attacks. For example, a DoS detector is likely to detect only network flooding attacks, but it may miss some cross-site scripting attacks targeting a web site. Hence, network providers must be aware of each specific attack and need to prepare for protecting their network from various types of attacks.

Another limitation is an inefficient resource usage in the network environment. Traditional devices are designed on the basis of an independent hardware device so that they cannot share their resources. For example, when a network is under DoS attack, the DoS Detector is the only device running actively while Firewall and IDS are idle. If Firewall or IDS can share their resources to DoS Detector, the utility of the whole network can be increased.

Lastly, there is a lack of service flexibility in a fixed network. Service inflexibility in a fixed network means that the capability provided to the user is restricted to the security services provided by the service provider in the fixed network without giving any choices. Thus, the network user can not choose the services based on their taste. For example, depending on the customer, one may need the highest level of security services or the other may prefer the lowest level of security to increase the performance.

Several types of research are studied to address these problems. SIMPLE[1] has suggested a traffic steering method for legacy devices using Software-Defined Networking(SDN). It allocates paths that packets go by way of devices while considering resource distribution. However, even if it makes a network more flexible, it could not make resource management also flexible because the kinds of devices are not changed, and the summation of its resources is fixed.

Network Function Virtualization(NFV) is a satisfying solution to overcome the limitation of physical devices. NFV is a network architecture which can make up the network services using virtualization as a software. The key advantage of NFV is the ability to virtualize the physical devices by simply adding a virtual machine(VM) whenever needed. However, NFV also carries several problems such as a single point of failure or a bottleneck problem due to its property.

In order to solve the problems for both occurring at the conventional network and those of NFV, we suggest QoSE applying the NFV technique with several NFV hosts connection as a distributed system in order to avoid a single point of failure and a bottleneck problem for ensuring high reliability. QoSE manages the resource usage in a flexible manner by referring to the network function usage. In addition, QoSE allows a user to selectively choose the services on demand. Finally, it presents the automatic recovery from the fault situation.

In this paper, we detail the design of QoSE and how it works, how each resource should be distributed and what paths should be selected among NFV nodes based on a resource optimization formula and algorithm.

II. design

The followings are the key considerations of QoSE.

Differentiated services: A key contribution of QoSE is that it can provide differentiated service level for each network flow. A network situation can be easily changed and requirements about security services are different for each traffic and each user. To adapt and meet it, we apply NFV technique. NFV provides network functions using VMs, so it can provide various security functions as a software. If networks need a new security function, networks just load a

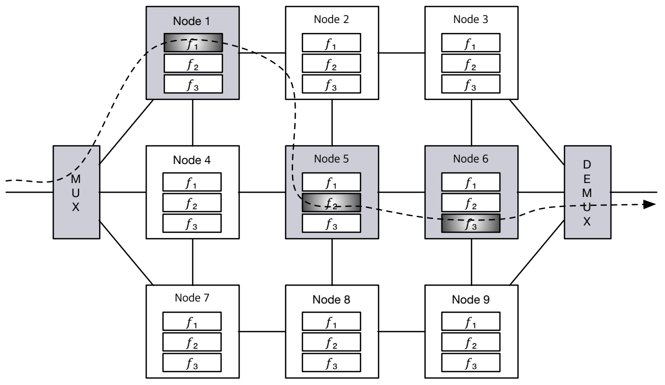


Fig. 1. QoSE consists of several NFV nodes. The flow follows the path based on function of selected nodes.

new VM. Furthermore, we provide a policy which describes what services are provided for each flow (Section II-E).

Fault recovery: If security functions come to halt suddenly, networks might be exposed to various threats. Therefore, a robustness of security functions cannot be emphasized enough. They should keep the running state and be able to recover from a fault situation as soon as possible. A single host based NFV environment is more of critical because it contains plural functions in a single device. To address it, QoSE offers a distributed NFV environment with a recovery mechanism to handle the problem correspondingly (Section II-D).

Efficient resource management: Since QoSE consists of distributed system, resource management must be considered. Otherwise, network flows may be able to be concentrated on the specific node, and it can raise a bottleneck problem or others. Therefore, the network flows must be properly distributed to nodes. To achieve this, we devise a novel resource optimaization and path selection algorithm (Section II-C).

A. Structure

As mentioned before, QoSE is a distributed NFV system. We connect several NFV nodes which have the network functions as a grid topology, and add Multiplexer(MUX) and Demultiplexer(DEMUX) at the edge of both sides for modularizing. Each node has *the same* security functions. Hence, every node can do the same works, so if one node goes wrong, other nodes can stand in.

Network flows pass through nodes from MUX to DEMUX following a decided path. For this work, we use OpenFlow which is a de-facto standard protocol of Software-Defined Networking(SDN) for controlling flows. When a new flow arrives at MUX, a notification message is sent to a QoSE controller by the *packet_in* message. Then, the QoSE controller calculates which nodes to run for which functions using the resource optimization the path selection algorithm.

Fig. 1 describes an overall example of QoSE. In this figure, nine NFV hosts have three network functions and compose 3x3 size of QoSE module with MUX and DEMUX. In this situation, we assume that a new flow comes to MUX and the controller runs the algorithms. As the result, Node 1 is selected

TABLE I
SIMPLE OPENFLOW TABLE FOR A NODE WHICH HAS TWO NETWORK FUNCTIONS.

Situation	Src	Dst	in_port	Actions	Byte counts	Packet counts
A→B, Fwd	A	B	1	output:2	*	*
A→C,f_1	A	C	1	**output:10**	*	*
	A	C	**11**	output:3	*	*
A→D, f_2	A	D	1	**output:20**	*	*
	A	D	**21**	output:4	*	*

for the network function f_1, Node 5 is selected for network function f_2, and Node 6 is selected for the network function f_3. Node 2 is also selected for relaying the flow from node 1 to node 5. After this process, the controller installs OpenFlow rules which handle the flow at selected node and then the flow is going to be processed.

B. Flow control on the node

QoSE dynamically distributes network flows into NFV nodes, which provide a set of security services, based on resource states of each node to optimize resource usages of the nodes. For this dynamic flow control, QoSE employs Open vSwitch(OVS) [2] on each node, which is a software switch mostly used for host switches of VMs [3]. Thanks to the programmability of OVS, QoSE can forward flows dynamically to a proper node according to resource usages of the system.

However, OVS cannot forward flows to a proper VM serving a desired security service in a node since it does not have the semantics of flows and VMs. To resolve this challenge, we use a simple trick on switch ports of the software switch. QoSE notifies the semantics of VMs to OVS by assigning an identifier to each VM and using it as a port number of corresponding VMs. Furthermore, QoSE separates an input and an output interface so that QoSE guarantees whether a flow is processed or not by the *in_port* number and gives the semantics of flows to the software switch.

To give a concrete example of this, Table I describes a flow table of a node. In this node, the *input/output* port numbers of the function f_1 are *10/11*, of the function f_2 are *20/21*. A flow which goes from A to B is simply indicated as an output in the flow table(Situation A→B, Forwarding). However, when a flow which goes from A to C needs to visit the network function f_1, the flow output port indicates the input port of the network function f_1(Port 10). Then, the other rule is added for this flow that the input port becomes the output port of the network function f_1(Port 11) and output port indicates the next node. Both rules indicate the flow which goes from A to C via the function f_1(Situation A→C, f_1). A flow which goes from A to D via the function f_2 is similar to the flow which goes from A to C. Instead, its output port and input port are for function f_2(Port 20/21)(Situation A→D, f_2).

Using this method, a bandwidth information of each function can be easily obtained because the OpenFlow Table records the statistic information automatically per a rule. This statistic information can be used for estimating the resource usage of functions.

C. Resource optimization

Since QoSE is kind of a distributed system, the efficient resource distribution has to be an essential factor. Traffic is partly distributed to every node by resource usage and should be processed well without idle resources. To achieve this work, we have checked a correlation of resource usage and the amount of traffic at first.

According to Cherkasova et al. [4], the CPU usage of devices is proportional to the amount of traffic. They measured the CPU usage per network I/O rate on Xen hypervisor and found that CPU overhead is increased when the network bandwidth is increased. As a consequence, it is common that network functions use more resources when process more traffic. Thus, if we profile the correlation of resource usage and the amount of traffic in advance, we can predict how much resources are needed and how many traffic can be processed by remained resources. We can also calculate how many resources are allocated per bandwidth.

D. Fault Recovery

Detecting and recovering from fault states is the important issue. We classify two fault states which can be occurred: *i)* node failures and *ii)* link failures. First, we detect node failures by OpenFlow echo messages. A SDN controller sends an *echo* message to check switch states regularly. If the controller does not receive any reply message from the switch, the switch is regarded as a failure. Likewise, nodes of QoSE use OVS basically, we can apply this mechanism at QoSE. Next, to detect link failures, we use Link Layer Discovery Protocol(LLDP). If a connection between two nodes, the controller receives an alert message LLDP.

If the controller detects fault states, a fault recovery process runs. It has two steps. First, the QoSE controller removes all flow information related to the fault node. Then, the controller re-calculates available resources because total resource amounts can be changed. After it, the controller re-runs the path selection algorithm based on re-calculated resource usage for the removed flows.

E. Service policy

It is important to provide differentiated service in point of a resource efficiency and a service quality because the requirement about security services is different. To provide differentiated services, we introduce a service policy which describes what flows, what service they take, and how much traffic they use. The policy can be classified into two types; *i)* general policy and *ii)* private policy. A general policy is a default policy of the network that all traffic is influenced. A network administrator sets the default function rates provided to users. For example, the network administrator can adjust the usage rate of functions as 6:4 or 3:7 depending on the situation.

A private policy is commonly similar to the general policy, but it is only for specific flows. It is free from general policy and has a higher priority than the general policy. Therefore, it is calculated and allocates resources before the general policy.

TABLE II
NOTATIONS IN RESOURCE OPTIMIZATION FORMULATION

Node	
N	Set of NFV nodes which forms a QoSE module
n	NFV node in a QoSE module where $n \in N$
Hardware Resource	
R	Set of hardware resources of node
r	Hardware resources of node where $r \in R$
r^{max}	Capacity of resource r
Bandwidth	
b_{mn}	Transferred bandwidth from node m to n, then $b_{mn} \neq b_{nm}$
b_n	Sum of incoming bandwidth to node n s.t. $\sum_m b_{mn} = b_n$
$b_{n,i}$	Allocated bandwidth for security function i on node n
B_{mn}	Link capacity between node m and n, then $B_{mn} = B_{nm}$
Security Function	
f_i	Security function i deployed in each node
f_i^{rate}	Predefined allocation ratio of security function i
$f_{n,i}^{r}(b)$	Usage of resource r when security function i on node n processes bandwidth b

III. Resource management and Path selection

A. The resource optimzation formulation

In this section, we describe how QoSE optimizes resource usages of a QoSE module by solving an optimization problem to achieve an efficient resource management. As we mentioned in Section II-C, the resource usage of security functions depends on the amount of traffic bandwidth. Therefore, we design a optimization formula which calculates optimal bandwidth distribution without exceeding the resource capacity of each NFV node.

1) Definition: Prior to designing the optimization formula, we first define some notations as denoted in Table II. As a QoSE module consists of a set of NFV nodes N, QoSE can distribute incoming traffic based on the resource usage of each node n in the module.

A resource r is a type of hardware resources in each node, for instance, $r \in \{CPU, Memory\}$ when referenced resources are CPU and memory of nodes. Since hardware resource has a limit, the usage of each resource r should not exceed its capacity r^{MAX}.

Since resource usages depend on the amount of incoming bandwidth to each node, we separate a bandwidth b from a set of resources R. A bandwidth b_{mn} refers transferred bandwidth from node m to n, thus a total bandwidth between two nodes m and n becomes $b_{mn} + b_{nm}$, which cannot exceed a link capacity B_{mn} between them.

All nodes have same security functions, and a specific number is assigned like f_i, for example, Firewall can be written as f_1, and IDS can be f_2. f_i^{Rate} is defined by policies and it denotes how much the ratio of functions are allocated at the QoSE module. $f_{n,i}^{r}(b)$ refers the usage of resource r when function i of node n processes bandwidth b. If IDS is running on node 3 processes 30Mbps traffic, CPU usage of the IDS can be described as $f_{3,2}^{CPU}(30)$.

The example of notations is described in Fig. 2 that refers variables of a 2x2 module. Dotted line box in Fig. 2 refers variables which are defined in the node n.

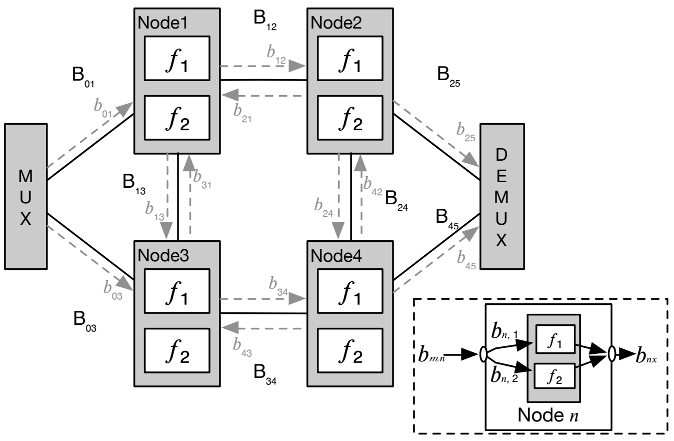


Fig. 2. Variables of 2x2 size QoSE module which has two network functions.

2) *Optimal formulation:* With above variables, we establish a formulation like following.

Maximize

$$\sum_n \sum_i b_{n,i} \tag{1}$$

Subject to

$$\forall n \in N : \sum_m b_{mn} = \sum_o b_{no} \tag{2}$$

$$\sum_m b_{MUXm} = \sum_o b_{oDEMUX} \tag{3}$$

$$\forall m, n \in N : b_{mn} + b_{nm} \leq B_{mn} \tag{4}$$

$$\forall r \in R, n \in N : \sum_i f^r_{n,i}(b_{n,i}) \leq r^{MAX}_n \tag{5}$$

$$\forall i \in I : \sum_n b_{n,i} \leq \sum_n \sum_j b_{n,j} * f^{Rate}_i \tag{6}$$

$$b \geq 0 \tag{7}$$

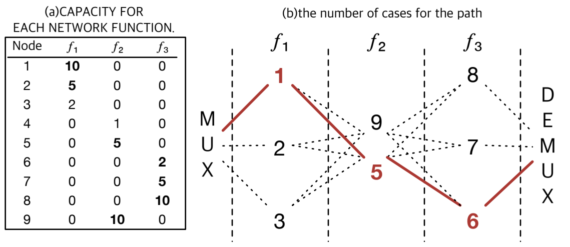
The goal of this formulation is calculating maximum traffic amounts per security function on a node without idle hardware resource of the node. In other words, the formulation maximizes *a bandwidth summation of each security function of each node*, and it can be written as equation (1).

Next we have to define conditions of the formulation to find a feasible solution. Equation (2) refers the amount of incoming traffic of node n that should be the same with the amount of outgoing traffic from the node n. Sometimes, an amount of output traffic might be different with input traffic due to drop, but we should consider that traffic is always processed without any problem(= There is NO drop). Therefore, we assume that there is no traffic loss in any node. Similarly, the total amount of incoming traffic to QoSE system should be the same with the total amount of outgoing traffic, so equation (3) is added to the formulation.

In a point of resource limitation, traffic passing through a link between node m and n has not to over a link capacity B_{mn}, thus equation (4) should be satisfied. Likewise, security functions of each node have not to over resource capacities, so a solution has to satisfy equation (5). In order to apply service policies for each security function, a sum of $b_{n,i}$ for security function i that is allocated in each node should be lower than predefined rate f^{Rate}_i of total bandwidth in QoSE, as equation (6) shows. Finally, amounts of allocated bandwidth must be a positive number, so equation (7) is added to the formulation. Table III is an example result of this formulation with following variables.

TABLE III
EXAMPLE OF A SOLUTION FOR THE OPTIMAL FORMULATION

Node	Total	f_1	f_2	CPU^{MAX}	f^{CPU}_1	f^{CPU}_2
1	10.00	10.00	0	100	100	0
2	10.00	10.00	0	100	100	0
3	13.53	6.47	7.06	100	65	35
4	13.53	6.47	7.06	100	65	35
Total	47.06	32.94	14.12	400	330	70


(a)CAPACITY FOR
EACH NETWORK FUNCTION.

Node	f_1	f_2	f_3
1	**10**	0	0
2	**5**	0	0
3	2	0	0
4	0	1	0
5	0	**5**	0
6	0	0	**2**
7	0	0	**5**
8	0	0	**10**
9	0	**10**	0

Fig. 3. Example of (a)a capacity for each network function and (b)the number of cases. Among available nodes, the red solid line path 'MUX-1-5-6-DEMUX' has the lowest hop count in this case.

Variables of Table III: For all nodes, B_{mn}=100Mbps, $f^{CPU}_{n,1}(b)$=10b, $f^{CPU}_{n,2}(b)$=10b, CPU^{MAX}_n is 100, and a function rate of f_1 and f_2 is 7:3.

B. Path selection algorithm

In order to find an optimal path of each flow, QoSE calculates the shortest path using its own path selection algorithm. Before path selection, QoSE solves the formula described in the previous section when a flow comes to MUX. After the bandwidth of each security functions in each node is resulted by the formula, the path selection algorithm selects a node sequence to traverse security functions which are written on the policy of the targeted flow. It selects the shortest path which contains the smallest hop.

For better understanding, an example scenario of the path selection scenario is following. Assuming that there is 3x3 QoSE system which provides three security functions. A flow coming to QoSE wants to use every functions f_1, f_2, f_3 in a bandwidth of *2Mbps* as its policy. In this situation, suppose that the result of solving the formulation is as illustrated in Fig. 3-(a). According to the result, nodes which accept f_1 are $\{1, 2, 3\}$, f_2 are $\{5, 9\}$ (Node 4 is NOT available because we need more than *2Mbps* bandwidth in this example.) and f_3 are $\{6, 7, 8\}$. After that, ordering it in descending order by the capacity, the results of all possible paths are described in Fig. 3-(b). Among the possible number of paths, an optimal path which has lowest hop count is the solid line that traversing sequence of nodes as '1-5-6'. Although the path traversing sequence of nodes '2-5-6' also contains the lowest hop count, it is lower priority due to the having the lower capacity. Consequently, the

Algorithm 1 Path selection algorithm

```
1: $functions[] \leftarrow$ readFunctionList()
2: for $i$ in range(funcLength) do
3:     $availNodes \leftarrow$ getAvailableNodes($functions[i]$)
4:     $list \leftarrow$ NodeOrdering($availNodes, functions[i]$)
5:     $Nodes[i] \leftarrow list$
6: $fList \leftarrow [[MUX], Nodes[1], Nodes[2], ..., [DEMUX]]$
7: FINDPATH($fList, 0, [], ret$)
8:
9: procedure FINDPATH($fList, depth, step, ret$)
10:     if $depth$ == length($fList$) then
11:         $dist \leftarrow 0$
12:         $path \leftarrow []$
13:         for $i$ in range($depth - 1$) do
14:             $d, p \leftarrow$ getHopCounts($step[i], step[i+1]$)
15:             $dist \leftarrow dist + d$
16:             $path$.append($p$)
17:         return $dist, path$
18:     else
19:         $f \leftarrow fList[depth]$
20:         for $i$ in range(length($f$)) do
21:             $step$.append($f[i]$)
22:             $d, p \leftarrow$ FINDPATH($fList, depth, step, ret$)
23:             if $d < ret.dist$ then
24:                 $ret.dist \leftarrow d$
25:                 $ret.path \leftarrow p$
26:                 $ret.step \leftarrow step$
27:     return $ret$
```

selected shortest path which connects from MUX to DEMUX containing nodes as the sequence of '1-5-6' is selected as the final path. Detail of path selection algorithm is illustrated in Algorithm 1.

IV. Implementation

We have implemented a prototype of QoSE in Python. A QoSE controller is built on POX [5] which is one of the popular SDN controllers written in Python. The QoSE controller uses a PuLP Python module [6] as an LP solver to calculate optimal resource distribution. To build QoSE nodes, we have used Raspberry Pi [7], a low price single board PC. Even though a cost of Raspberry Pi is affordable to build our distributed system, it has limited hardware performance to run multiple commercial VMs for constructing an NFV environment. Instead, we use Mininet [8] which supports process-based virtualization to create a lightweight VMs.

V. Evaluation

To evaluate QoSE, we have built a QoSE module by connecting four Raspberry Pis that provide two security services (Firewall [9] and IDS [10]) in a grid topology (2x2 shape). Then, we have investigated how effective QoSE distributes flows according to resource states of nodes without degrading performance by measuring throughput and how fast QoSE recovers system faults such as a link disconnection. Furthermore, we have constructed a different size of a QoSE module with nine Raspberry Pis (3x3 shape) to show the scalability of QoSE.

We also have performed the same evaluation for a conventional single NFV system to show the effectiveness of our

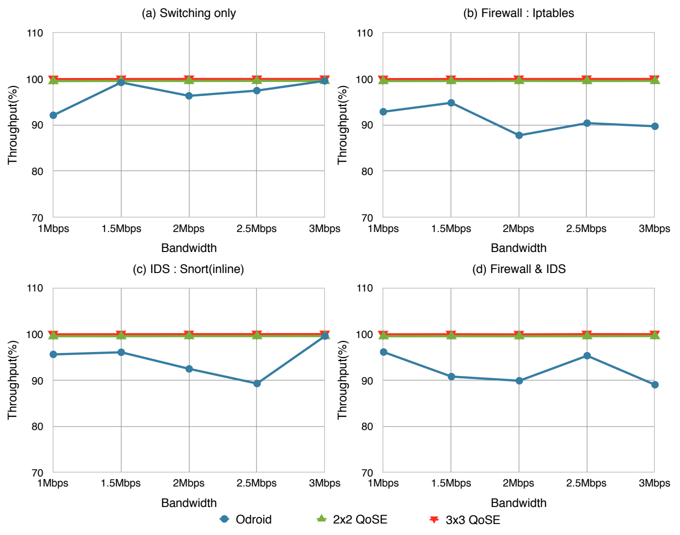


Fig. 4. Throughput of QoSE and Odroid-XU3 in four test cases

distributed system. We have devised a single NFV environment on Odroid-XU3 [11] which is another single board PC with higher performance than Raspberry Pi with the same security services. We argue that it is reasonable to compare it with 2x2 QoSE in terms of costs since the price of a single Odroid-XU3 ($179) amounts to the price of four Raspberry Pis ($35 each).

A. Profiling

Before evaluating our system, we have measured resource usage of the two security functions first because QoSE controller requires $f_i^r(b)$ to calculate optimal resource distribution. We have run iptables and Snort with 1000 rules on Raspberry Pi. We have sent random TCP flows to the node and monitored resource usage of each function through a Linux `top` command. We also increase the amount of traffic linearly to draw a correlation between resource usage of each function and the volume of traffic. A linear regression model of CPU usage of each function and determination are below (R^2 means the coefficient of determination).

$$f_{snort}^{CPU}(b) = 0.024b + 0.5992,\ R^2 = 0.9263$$
$$f_{iptabls}^{CPU}(b) = 0.043b + 0.4988,\ R^2 = 0.9617$$

Limitation for evaluation Through this measurement, we found that Raspberry Pi can not cover traffic more than 3Mbps due to a limitation of CPU performance. Thus, we have limited the amount of traffic up to 3Mbps in our evaluation.

B. Throughput

As a first evaluation, we measure throughput of QoSE to show how effective our system processes traffic without performance degradation. We connect a QoSE module between a sender and a receiver host, and compare the number of sent packets with the number of received packets by sending random TCP packets from the sender to the receiver host. Moreover, we perform four different evaluation cases to show detailed evaluation according to the number and variety of security services in QoSE.

TABLE IV
FAULT RECOVERY TIME

Fault recovery time	Min	Avg	Max	Stdevp
	0.99	1.50	2.01	0.52

Fig. 4 describes the measured throughput of QoSE in four cases: (a) switching without passing any security service, (b) passing Firewall, (c) passing IDS, and (d) passing both Firewall and IDS. In the all cases, both scales of QoSE show almost 100% of throughput while Odroid has less than 90% of throughput in the worst case.

The results demonstrate that the distributed architecture of QoSE with our resource optimization algorithm is suitable than the centralized architecture of the conventional NFV since it distributes system loads to avoid bottleneck of each node while Odroid suffers from bottleneck and performance degradation.

C. Fault Recovery Time

Next, we measure the fault recovery time to evaluate how fast QoSE detects and recovers system faults to provide stable security services. We disconnect one link in a module during sending packets and measure a time when the receiver host receives packets after the link disconnection. Table IV is the fault recover time of QoSE and shows QoSE recovers the fault within 2 seconds. Since it depends on the configuration time of the OpenFlow echo message, the recovery can be done faster by decreasing the terms of the echo message.

VI. RELATED WORK

Controlling distributed security functions: Cloudwatcher [12] proposes routing strategies how network flows optimally traverse the security devices. Although both QoSE and Cloudwatcher use the path selection algorithm to traverse security functions, Cloudwatcher does not consider of which security devices should be traversed for load balancing of the devices. Sekar *et al.* [13] propose a method to control IDS and IPS for a large scale network. For an efficient management of IDS and IPS, they devise a formula that models constraints such as resources, placement and network-wide objectives. SIMPLE [1] concentrates on the load balancing of legacy middleboxes with network policy enforcements. In contrast with SIMPLE, CoMb [14] proposes customized API for a consolidated middlebox design in a commodity hardware. More recently, SOL [15] tries to abstract the optimization process for SDN applications developer. One may look at QoSE as previous middlebox management researches in a manner of resource management; there is a clear distinction apart from those. We focused on a specialized network architecture for security services which operates as one powerful middlebox that has not been considered in early works.

Security enhancement using SDN/NFV: AvantGuard [16] proposes a secure OpenFlow switch architecture to prevent network attacks. Although there is no consideration of the hardware supporting security functions in the current QoSE design, QoSE can be extended in future work by offering both hardware support as well as fast attack detection. Braga et al. [17] propose a DDoS attack detection scheme using the flow statistic information in the flow table of OpenFlow-enabled switches. Our work is orthogonal to this research since their work focuses on a northbound application level of detection in contrast to the detection by the independent security device.

VII. CONCLUSION

QoSE is a new type of system using SDN/NFV to manage the network resources flexibly and increase the network utility by sharing the idle resource. It can also recover from fault situation automatically, and provide a capability to the users by selectively choosing the services based on their demand. Proposed optimal formula and path selection algorithm showed how to manage the network resources and control the network flows in the distributed NFV environment. Our contribution has presented that QoSE enables network providers to efficiently meet the security objectives for a secure network environment.

REFERENCES

[1] L. C. R. M. V. S. Zafar Ayyub Qazi, Cheng-Chun Tu and M. Yu, "Simple-fying middlebox policy enforcement using sdn," in *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, 2013.

[2] Open vSwitch, "An Open Virtual Switch," http://openvswitch.org/.

[3] Xen, "Xen Wiki - Networking," http://wiki.xenproject.org/wiki/Xen_Networking#Open_vSwitch.

[4] L. Cherkasova and R. Gardner, "Measuring cpu overhead for i/o processing in the xen virtual machine monitor," in *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, ser. ATEC '05. Berkeley, CA, USA: USENIX Association, 2005, pp. 24–24.

[5] POX, "Python Network Controller," http://www.noxrepo.org/pox/about-pox/.

[6] PuLP, "Python linear programming solver," https://pypi.python.org/pypi/PuLP.

[7] Raspberry Pi, "Single board PC," https://www.raspberrypi.org/.

[8] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: Rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, ser. Hotnets-IX. New York, NY, USA: ACM, 2010, pp. 19:1–19:6.

[9] iptables, "Linux firewall service," http://www.netfilter.org/projects/iptables/index.html.

[10] Snort, "Network Intrusion Detection System," https://www.snort.org/.

[11] Odroid, "Single board PC," http://www.hardkernel.com/main/main.php.

[12] S. Shin and G. Gu, "Cloudwatcher: Network security monitoring using openflow in dynamic cloud networks (or: How to provide security monitoring as a service in clouds?)," in *Proceedings of the 7th Workshop on Secure Network Protocols (NPSec12), co-located with IEEE ICNP12*, October 2012.

[13] V. Sekar, R. Krishnaswamy, A. Gupta, and M. K. Reiter, "Network-wide deployment of intrusion detection and prevention systems," in *Proceedings of the 6th International COnference*, ser. Co-NEXT '10. New York, NY, USA: ACM, 2010, pp. 18:1–18:12.

[14] V. Sekar, N. Egi, S. Ratnasamy, M. K. Reiter, and G. Shi, "Design and implementation of a consolidated middlebox architecture," in *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 24–24.

[15] V. Heorhiadi, M. K. Reiter, and V. Sekar, "Accelerating the development of software-defined network optimization applications using sol," *arXiv preprint arXiv:1504.07704*, 2015.

[16] S. Shin, V. Yegneswaran, P. Porras, and G. Gu, "Avant-guard: Scalable and vigilant switch flow management in software-defined networks," in *Proceedings of the 20th ACM Conference on Computer and Communications Security (CCS13)*, November 2013.

[17] R. S. Braga, E. Mota, and A. Passito, "Lightweight ddos flooding attack detection using nox/openflow," in *Proceedings of the 35th Annual IEEE Conference on Local Computer Networks*, ser. LCN, 2010.