TOSEC: A TCP/IP Offload based Virtual Network Security Framework in NFV Environment

Hong-wei TANG[1,2,3,*], Sheng-zhong FENG[1,3] and Xiao-fang ZHAO[2,3]

[1]Shenzhen Institute of Advanced Technology, Chinese Academy of Sciences, Shenzhen, China

[2]Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China

[3]University of Chinese Academy of Sciences, Beijing, China

*Corresponding author

Keywords: NFV, Network Security, Virtual Machine, VCPU Scheduling, TCP/IP Offload

Abstract. *There are two significant problems on NFV based virtual network security solution. One is that the traditional subnet-centered architecture cannot prevent insider attacks between virtual machines in the same subnet. The other is the performance degradation due to virtualization. Motivated by the above two points, we proposed a TCP/IP offload based virtual network security framework for NFV environment, called TOSEC. In TOSEC, network security systems are packaged in virtual machines, and are deployed on each host machine to provide security checking and filtering on network traffics for each individual virtual machine. Furthermore, we adopted a macro view on inter-VM network communication optimization. It eliminates repeated TCP/IP stack processing on virtual machines by employing TCP/IP offload technique, and securely shares Layer 7 payloads between the guest VM and related security VMs via inter-VM shared memory. Moreover, evaluations on the prototype based on KVM show that it significantly improves the communication performance of the guest VM and reduces the CPU utilization for both the guest VM and security VMs. Specifically, with one security VM deployed, the communication latency of the guest VM is reduced to 68%~48% of that in the general NFV deployment, while with two security VMs, the latency is reduced to 33%~22%.*

Introduction

Cloud computing is facing the unprecedented challenges on virtual network security [1][2][3][4], which mainly concentrate in two aspects. On the one hand, network traffics between virtual machines inside a host machine are transferred locally and are not visible to outer network. As a result, traditional monitoring or filtering tools are not applicable in cloud environment. On the other hand, due to the dynamic nature of virtual network topologies, traditional hard-wired security devices cannot meet the demands on virtual network security. Recently, solutions based on the concept of network function virtualization (NFV) [5][6] have attracted more and more attentions from both the academic and industrial communities. In these solutions, network security systems are installed in virtual machines and deployed around the border of virtual networks. The benefits to provide network security services as virtual functions are obvious, such as maximum flexibility for on-demand delivering of network security services, network isolation in multi-tenant environment, and etc [5].

Copyright © 2017, the Authors. Published by Atlantis Press.
This is an open access article under the CC BY-NC license (http://creativecommons.org/licenses/by-nc/4.0/).

In spite of the advantages of NFV, there is still big performance challenge on adopting NFV for network security, which requires capabilities of high-throughput and low-latency packet processing [7][8][9][10]. Despite the optimizations such as DPDK [11][15], VCPU pinning and NUMA-aware VM placement, improve the performance of a single device effectively [14], for example the virtual switch and the virtual machine, there is still significant overhead of the overall NFV framework. In this framework, packets are forwarded according to a predefined path among virtual machines with various security functions over virtual networks which may even span hosts. At each virtual machine, packets have to be packed and unpacked repeatedly by the TCP/IP stack and NIC drivers. And apparently, this brings pressures on virtual networks, and introduces a lot of consumption of host resources, especially the CPU resource. On the one hand, the current network security architecture is subnet-centered, in which network security systems are deployed around the boundary of subnet that only check traffics between the subnet and the outer world. This depends on trust between the servers in the same subnet. However, insider threats are more and more serious in modern network environment [12][13].

In this paper, a VM-centered and lightweight virtual network security framework based on NFV architecture is proposed, which is called TOSEC. In this framework, network security systems are encapsulated in virtual machines (SVMs) and deployed on host machine to provide security services for the guest virtual machines. In particular, the TCP/IP stacks of guest VMs are offloaded to a dedicated virtual machine (TOVM) and Layer 7 payloads are transferred via inter-VM shared memory between them. By attaching the shared memory to the SVMs, the network security systems could perform checking on the payloads directly, so the TCP/IP stack processing is eliminated, and furthermore, the efficiency of network transmission between virtual machines is significantly improved. The main contributions of this work can be summarized as follows:

Firstly, we propose a VM-centered and lightweight virtual network security framework for NFV environment. By offloading the TCP/IP stack from guest VM, Layer 7 payloads are directly provided to network security systems in inter-VM shared memory, so that the overhead of repeatedly encapsulating and decapsulating packets through the TCP/IP stack is eliminated. Moreover, network performance is significant improved since the inter-VM communication is completely based on shared memory. Furthermore, it complies with the NFV architecture and still keeps the benefits such as flexibility, multi-tenancy, SFC support, and etc.

Secondly, a prototype system is implemented based on KVM virtualization platform, and the performance is evaluated by micro-benchmarks. The experimental results show that it achieves a higher communication performance and lower resource consumption than the native NFV architecture.

The rest of the paper is organized in the following manner: Section 2 gives a brief review of related works on NFV performance optimization and TCP/IP offloading. Section 3 gives the design and implementation of TOSEC and the CPU scheduling strategies and algorithm. Section 4 presents the experimental results, and finally Section 5 concludes the paper and mentions the future work.

Related Works

In this section, we first review the NFV solutions and then give related works on TCP/IP stack offload.

NFV and Optimization Methods

NFV provides a way to implement fundamental network services, in particular security services, in software running in virtual machines on commodity servers rather than using dedicated and expensive hardware devices [5][6]. NFV also provides advantages for network security. It facilitates the availability of network security services for different applications, users and tenants, and isolation among tenants. The security services deployed in NFV mode can be rapidly provisioned and scaled up or down as required, while in the traditional environment based on hard-wired security framework, the adaption is time-consuming and complex.

However, since the NFV solution is based on commodity servers, a probable decrease in performance has to be taken into account. The challenge is how to keep the performance degradation as small as possible, so that the effects on network latency, throughput and processing overhead are minimized. To this end, many performance optimization approaches are adopted on the VM aspect, such as pinning VCPU to physical CPU core to avoid context switching overhead and frequent cache invalidation and TLB flushing, NUMA-aware VM placement and memory allocation to VMs to minimize cross-node memory accesses, leveraging huge pages for VMs to reduce TLB miss, using SR-IOV enabled NICs to eliminate overhead on NIC emulation [14], and so on. Moreover, there are also works focusing on accelerating packet transmission on the virtual network. In [15], DPDK is used to improve the performance of virtual switch by bypassing the kernel and using polling instead of interrupt. In [10], NICs with overlay offload capability are used to perform packet encapsulation and decapsulation for overlay networks such as VXLAN, NVGRE and GENEVE in hardware, which can significantly reduce CPU load on host machine and enhance network throughput and improve transmission latency.

Overall, the current optimizations mainly targeting minimizing overhead from the virtualization layer, while the overhead due to packet transmission over TCP/IP should also not be ignored.

TCP/IP Offload

TCP/IP offload techniques have been studied for many years since it had been proposed, and now many intelligent NICs provide this kind of functionalities. Offloading the processing of TCP/IP stack to NICs can reduce the pressure of host CPU by reducing NIC interrupts to CPU and memory copies over the system bus [16][17][18][21]. Other than NICs, the TCP/IP stack can also be offloaded to elsewhere. [18] proposes a system architecture that executes the TCP/IP processing on a dedicated processor, remote node or device which is away from where the Internet server runs while providing low-overhead communication between them. The idea is also adopted in virtualization environments. [19] improves TCP throughput of virtual machines via TCP acknowledgement offload to domain 0, [20] presents the solution of offloading TCP/IP stack from guest OS to the hypervisor (vmkernel) thereby bypassing the guest OS kernel.

Architecture and Implementation

Overall Architecture

On the basis of NFV, TOSEC aims to provide a VM-centered and lightweight network security framework for virtual machines. Different from the traditional way in which security systems are deployed around the border of the network, under TOSEC network security systems are deployed on each host machine and provide security services for each individual guest VM. Furthermore, it employs TCP/IP offloading technique to eliminate the overhead of repeated TCP/IP stack processing on each VM, and shorten the communication path between VMs. The overall architecture is briefly shown in Fig. 1. The prototype system is implemented on Qemu/KVM. TOSEC is composed of the following components:

- *Guest VM*, where application runs. The network stack of guest operating system is offloaded to a dedicated VM. Socket compatible APIs are provided in the guest VMs.
- *TCP Offload Engine (TOE)*, which manages network connections and processes incoming and outgoing packets on the local TCP/IP stack on behalf of guest VMs. It runs in a multi-threading mode in a dedicated VM which is called *TOVM*. For each guest VM, there are dedicated TOE working threads serving the network processing.
- *Security Virtual Machine (SVM)*, which is the encapsulation of a network security system together with the runtime and operating system in a virtual machine. It provides stateless security service for guest VMs on the host.
- *Remote Procedure Call (RPC) Channel*, which provides inter-VM communications for commands and notifications.

Definitions

- *Layer 7 Data Stream*

A Layer 7 data stream is composed of sequential Layer 7 payloads, such as HTTP packets, in a certain direction (incoming or outgoing), over a TCP or UDP connection of a guest VM via the TOE. An outgoing stream originates from the guest VM, goes through several SVMs, encapsulated by the TOE, and finally flows out to the outer world. While an incoming stream goes through the reverse path. When a TCP or UDP connection of a guest VM is established, a unique ID is assigned to the stream for differentiating it from other streams.

- *SVM chaining*

Just like SFC, SVMs can also be chained together to protect a guest VM. For example, to filter the traffics to a web server on the guest VM, SVMs of type WAF, IPS and Anti-Virus are deployed in sequential order along the path of the stream, which form a SVM chain. Specially, a SVM can be linked into multiple chains and provides protection for guest VMs of different users or even different tenants.

- *Stream Table*

Stream table is used to specify the routes for streams. When a stream is generated, the corresponding routing rules are distributed to the local stream tables on each VM along the path. After a VM has completed the processing of a payload, it queries the local stream table to find the next VM to process or accept payload. Furthermore, stream table is the basis to implement SVM chaining.

TCP/IP Stack Offload

The cooperation of guest VM and TOE on TCP/IP stack processing is based on RPC mechanism which is composed of RPC stub, RPC server, and communication channel based on inter-VM shared memory. There is a RPC stub and a RPC server on each VM. The inter-VM shared memory is divided into two parts, the control buffers and the data buffers. The control buffers are organized into rings, and there is a dedicated ring for each VM, including the TOVM, SVMs and guest VM. While the data buffers are provisioned for each guest VM. The data buffers are further subdivided into a send buffer pool for outgoing stream and a receive buffer pool for incoming stream.

To execute a remote procedure, the RPC stub allocates a buffer from the control ring, fills command and parameters into the buffer, and notifies the RPC server to consume it by sending an inter-VM interrupt. Then the RPC server calls the corresponding server function according to the command and parameters in the buffer, and fills return values into the buffer after servicing the command. Finally, until the RPC stub gets return values from the RPC server, the buffer is released back to the control ring. Furthermore, a socket compatible user-level API library is provided to applications on guest VMs. It simply forwards socket related operations to the TOE via RPC.

Security VM

The inter-VM shared memory between the guest VM and the TOE is attached to the related SVMs where network security systems are installed. So they could directly perform security checking on the Layer 7 payloads without decapsulating packets through TCP/IP stack. A SVM is composed of three parts, as shown in Fig. 2:

- *Working threads*: For each guest VM, one or more dedicated working threads are employed on the SVM.
- *Manager*: It receives notification from the previous VM, and dispatches the payload to the corresponding working threads according to the related guest VM. After the working thread completes the checking, the manager helps to search the stream table to find the next VM to forward the processing. If the payload does not pass the security checks, the manager blocks the stream, logs the security threats, and notifies the administrator.
- *Network security system*: There needs minor porting effort on traditional network security systems to be adapted for the TOSEC framework. For example, the input for traditional network security systems is generally Layer 2 or Layer 3 packets, while in TOSEC, the input is Layer 7 payloads, so the input processing logic should be simplified accordingly.

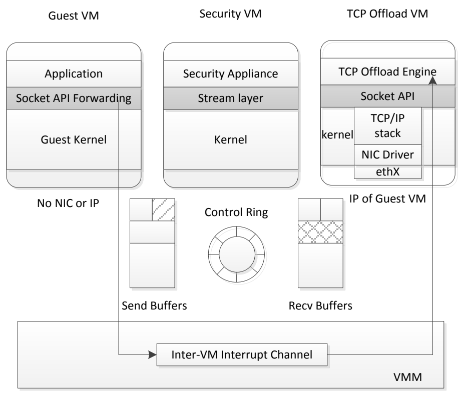


Figure 1. Overall architecture of TOSEC

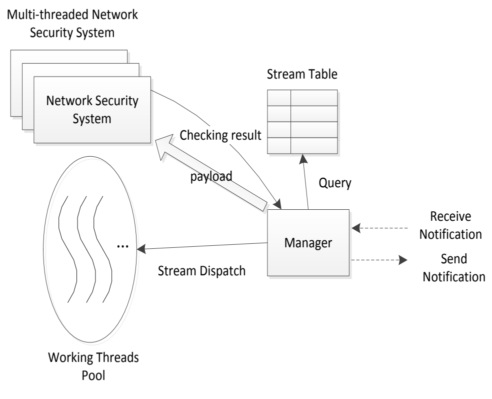


Figure 2. Architecture of security VM

Experimental Evaluations

Methodology

The prototype system is based on KVM virtualization. CentOS 7.1.1503 with linux kernel (version 3.10.0) is adopted on both host and guest VM. The experimental setup is comprised of four Sugon I620-G20 servers, which are dual-socket, 10-cores per socket servers equipped with Intel Xeon E5-2650 V3 CPUs running at 2.3GHz. The host server includes 32GB of memory, and an Intel I210 1Gb NIC. The virtual machines are configured with 4 VCPUs and 8GB memory. Netperf is used to measure the communication latency and CPU utilization of the VMs. In the following experiments, request-response mode of Netperf is switched on by specifying TCP_RR in the command line.

Benchmark Results

Apparently, the more network security systems are deployed on the path of the stream, the higher latency will be introduced on payload transmission. To verify the effect of the optimal design of TOSEC on communication performance between security VMs and target VM, three experiments are performed in this subsection. Experiment a) measures the basic communication performance of VM with TCP/IP stack offload, while experiment b) and c) measure the communication performance of VM with one or two security VMs deployed to filtering the traffic respectively.

a) Communication Performance of VM with TCP/IP Stack Offload

Firstly, the Layer 7 payload transmission latency is measured on virtual machine with TCP/IP stack offloaded. To make a comparison between it and that of virtual machine with local TCP/IP stack, the testbed is setup as Fig. 3. A physical machine (H2) is used as the communication peer for the target VM (V) which is deployed on another physical host machine (H1). The physical machines are interconnected with a 10G Ethernet.

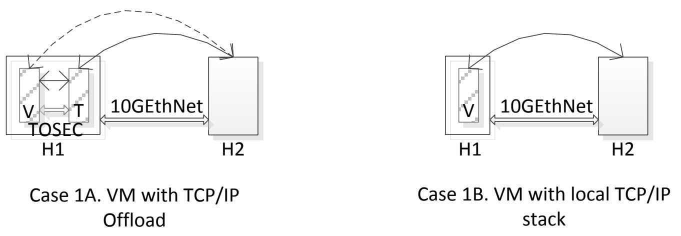

V
T
TOSEC
10GEthNet
H1
H2
Case 1A. VM with TCP/IP Offload
V
10GEthNet
H1
H2
Case 1B. VM with local TCP/IP stack

Figure 3. Testbed setup for evaluating the communication latency with TCP/IP offload

The comparison on communication latency achieved in the two deployment cases is shown in Fig. 4. Overall, VM with TCP/IP stack offload (Case 1A) has a higher latency than VM with local TCP/IP stack (Case 1B). Moreover, for both the two cases, the latency keeps flat at about 132 us (Case 1A) and 78~95us (Case 1B) respectively when the payload size is no more than 1024 bytes. When the payload size is more than 1024, the payload will be fragmented because the MTU of the link layer is 1500 bytes, so the load on traffic processing increases dramatically with the increase of payload size. The CPU utilization rates of the VMs during the benchmarking are depicted in curves in Fig. 5. Because there is no TCP/IP stack processing in the target VM in Case 1A, the CPU utilization in it is very low. Furthermore, the CPU utilization of the TOE VM increases as large payloads are fragmented, that needs more CPU cycles to handle NIC interrupts and reassemble packets.

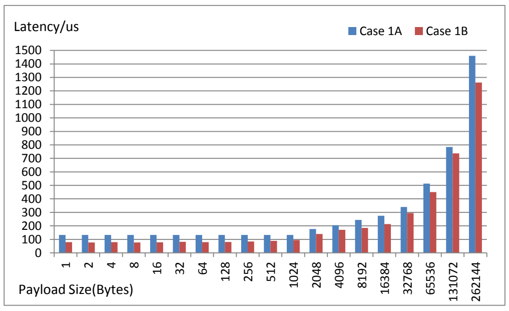


Figure 4. Comparison on communication latency

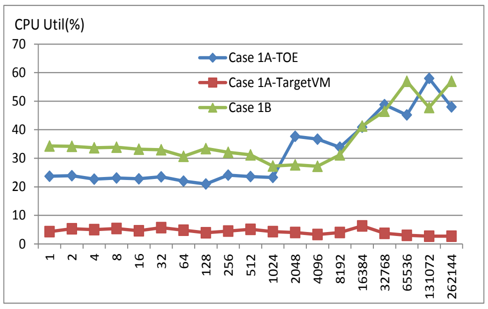


Figure 5. Comparison on CPU utilization of VMs

b) Communication Performance of VM with one Security VM deployed

Fig. 6 shows the schematic diagram of the testbed for measuring and comparing the communication latency of VM with only one security VM deployed. In Case 2A, the SVM is deployed under the TOSEC framework, while in Case 2B and Case 2C, the SVMs are both deployed in NFV manner. Specifically, Case 2B presents that the SVM and the target VM are deployed on the same physical host machine, and the communication between them is based on the virtual bridge on the host machine. In Case 2C, the SVM and the target VM are deployed on different host machines which are interconnected by a 10G Ethernet.

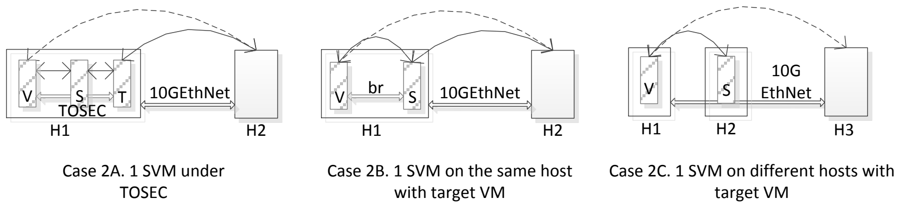
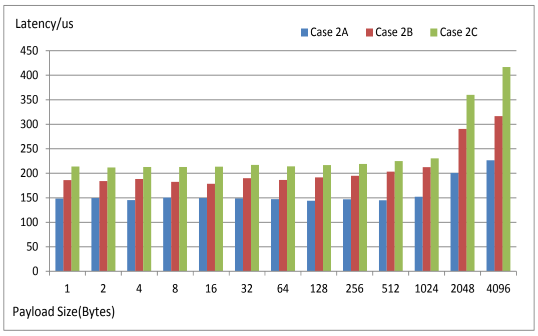

V
S
T
TOSEC
10GEthNet
H1
H2
Case 2A. 1 SVM under TOSEC
V
br
S
10GEthNet
H1
H2
Case 2B. 1 SVM on the same host with target VM
V
S
10G EthNet
H1
H2
H3
Case 2C. 1 SVM on different hosts with target VM

Figure 6. Testbed setup for evaluating the communication latency with 1 SVM

The latency of transmitting Layer 7 payloads between the target VM and the external physical host is shown in Fig. 7. Overall, the results approve that the performance under TOSEC outperforms that in the general NFV solution. Specifically, TOSEC reduces the communication latency to 68%~48% of that in the general NFV solution. The CPU utilization of each VM is shown in the curves in Fig. 8. Because the TCP/IP stack is offloaded from the target VM and the SVM, the CPU utilization of these two VMs (Case 2A-SVM and Case 2A-TragetVM) is much lower than that in Case 2B and Case 2C, and keeps flat with the increase of the payload size. In particular, in Case 2B, the CPU utilization of the SVM and the target VM changes dynamically between 20% to 100%, and it cannot achieve a stability when the payload size is larger than 1024 bytes.

c) Communication Performance of VM with 2 Security VMs deployed

The testbed setup for this experiment is shown in Fig. 9. In case 3A, the SVMs are deployed under TOSEC framework. For comparison with TOSEC, other two deployments that follow the general NFV solution are also proposed. In case 3B, the SVMs and the target VM are on the same host, and communicate with each other via TCP/IP over the virtual bridge. In case 3C, the three VMs are distributed onto different hosts, communicating over the 10G Ethernet.



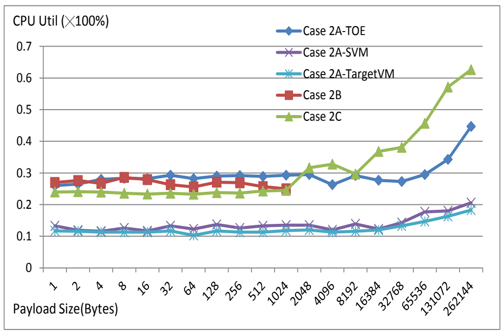


Figure 7. Communication latency of VM with 1 SVM   Figure 8. CPU Utilization of VMs with 1 SVM

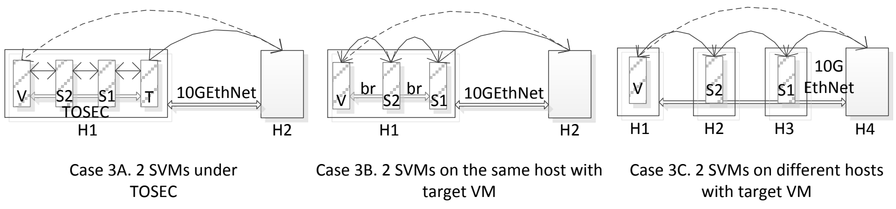
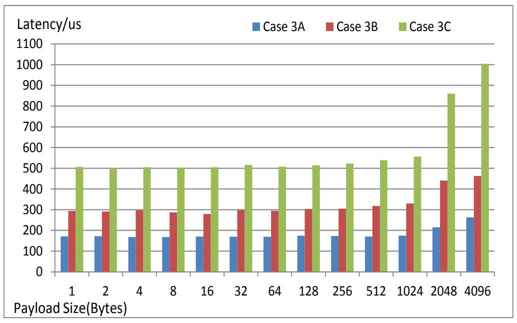

V
S2
S1
T
TOSEC
10GEthNet
H1
H2
Case 3A. 2 SVMs under TOSEC
br
Case 3B. 2 SVMs on the same host with target VM
10G EthNet
H3
H4
Case 3C. 2 SVMs on different hosts with target VM

Figure 9. Testbed setup for evaluating the communication latency with 2 SVMs

The comparisons on communication latency in the three cases are shown in Fig. 10. Under TOSEC, when the payload size is under 1024 bytes, the latency keeps flat at about 170 us. It increases dramatically with payloads larger than 1024 bytes, and the performance gap between the TOSEC and the general NFV also becomes wider. By contrast, the communication latency in TOSEC is reduced to 33%~22% of that in the NFV deployment. The comparison on CPU utilization of the VMs during benchmarking is shown in Fig. 11. The three VMs deployed in the general NFV framework have the similar load, so the CPU utilization rates of them are also similar. To simplify the figure, only one curve of the VMs is depicted in the figure (Case 3B and Case 3C). Under TOSEC, the CPU utilization of the TOE VM is similar to that in Case 3C, while the CPU utilization of the SVM and the target VM are very low because there is no TCP/IP stack processing and device driver interactions. In addition, for Case 3B, when the payload size is larger than 1024 bytes, the CPU utilization goes up to 100%. That is because the transmitting capacity of the virtual bridge between the VMs surpasses the processing capacity of the guest OS, which leads the guest OS to be busy processing payloads.



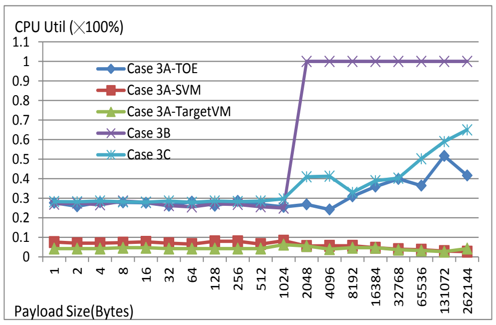


Figure 10 Communication latency of VM with 2 SVMs   Figure 11. CPU Utilization of VMs with 2 SVMs

Conclusion

NFV provides opportunities to virtual network security in cloud environment with the advantages such as flexibility, multi-tenancy, isolation, reduced hardware investment, and etc. Currently, some virtual network security solutions based on NFV architecture have already appeared. This paper aims to optimize the communication between virtual machines providing network security services and the guest virtual machines under protection from a macro view, and to improve the network security for virtual machines in a non-trustful network environment. Motivated by the above points, we propose TOSEC, a TCP/IP offload based virtual network security framework in NFV environment, and implement a prototype system on KVM virtualization platform. In TOSEC, various network security systems are deployed locally on the host machine providing protections around each individual virtual machine, so that all traffics flowing in to or out from the virtual machine are checked by the network security systems. Moreover, TCP/IP offload technique is employed to eliminate repeated TCP/IP stack processing from the virtual machines where applications or network security systems run. Finally, the evaluation results show that it significantly improves the communication performance and reduces CPU utilization of the virtual machines.

Our future work includes three parts. The first is to port common network security systems to TOSEC, such as WAF, IPS, Anti-Virus Wall, Database Access Auditor, and etc. Secondly, we are going to implement the TOE in the VMM to eliminate layer-2 packet forwarding between the virtual machine and the VMM, which might further improve the communication performance. At last, TOSEC complicates the live migration of guest VMs, we will try to solve this problem.

Acknowledgement

We would like to thank the anonymous reviewers and editors for their valuable suggestions and comments to improve the quality of this paper.

References

[1] H. Wu, Y. Ding, C. Winer, L. Yao, Network Security for Virtual Machine in Cloud Computing, International Conference on Computer Sciences and Convergence Information Technology, (2010) 18-21

[2] Pratishtha, Pratishtha, Threats to Virtual Network Security, DRISTI(1) (2012) 16-19

[3] L. R. Bays, R. R. Oliveira, M. P. Barcellos, L. P. Gaspary, E. R. M. Madeira, Virtual network security: threats, countermeasures, and challenges, Journal of Internet Services and Applications 6 (2015) 1-19

[4] F. Hao, T. V. Lakshman, S. Mukherjee, H. Song, Secure Cloud Computing with a Virtualized Network Infrastructure. Usenix Conference on Hot Topics in Cloud Computing 35 (2010) 57-61

[5] Network Functions Virtualization. An Introduction, Benefits, Enablers, Challenges & Call for Action, https://portal.etsi.org/nfv/nfv_white_paper.pdf

[6] OPNFV, https://www.opnfv.org/

[7] L. Cao, P. Sharma, S. Fahmy, V. Saxena, NFV-VITAL: A Framework for Characterizing the Performance of Virtual Network Functions, Network Function Virtualization and Software Defined Network (NFV-SDN) (2015)

[8] M. Asawa, NFV: A Dynamic, Multi-Layer Resource Optimization Challenge, http://eecs.umich.edu/eecs/about/articles/2016/Celebrating-a-Leader-In-Control-Systems/presentations/asawa.pdf

[9] X. Ge, Y. Liu, D. H. C. Du, L. Zhang, H. Guan, J. Chen, Y. Zhao, X. Hu, OpenANFV: Accelerating Network Function Virtualization with a Consolidated Framework in OpenStack. ACM SIGCOMM Computer Communication Review 44 (2015) 353-354

[10] Using Hardware to Improve NFV Performance (White Paper), http://www.mellanox.com/related-docs/whitepapers/WP_heavyreading-NFV-performance.pdf

[11] Intel DPDK, http://sdn.calsoftlabs.com/competencies/intel-dpdk.html

[12] R. Koch, B. Stelte, M. Golling, Attack Trends in Present Computer Networks, International Conference on Cyber Conflict (2012) 1-12

[13] S. Cates, Combatting Insider Threats, Sports Car (2013)

[14] A Path to Line-Rate-Capable NFV Deployments with Intel Architecture and the OpenStack Kilo Release, http:// www.intel.la/content/dam/www/ public/us/en / documents/ white-papers/open-stack-Kilo-paper-v2.pdf

[15] V. Jardin, High Performance NFV Infrastructure (NFVI): DPDK Host Applications with neutron/ OpenStack and VNF Acceleration, https:// events. linuxfoundation.org/sites/events/files/slides/Openstack-v4_0.pdf

[16] J. C. Mogul, TCP offload is a dumb idea whose time has come, Hotos'03: Workshop on Hot Topics in Operating Systems 9 (2003) 25-30

[17] D. Freimuth , E. Hu, J. Lavoie, R. Mraz, E. Nahum, P. Pradhan, J. Tracey, Server Network Scalability and TCP offload, Conference on Usenix Technical Conference (2005) 209-222

[18] M. Rangarajan, A. Bohra, K. Banerjee, E. V. Carrera, R. Bianchini, L. Iftode, TCP Servers: Offloading TCP Processing in Internet Servers. Design, Implementation, and Performance (2010)

[19] A. Kangarlou, S. Gamage, R. R. Kompella, D. Xu, vSnoop: Improving TCP Throughput in Virtualized Environments via Acknowledgement Offload. (2010)1-11.

[20] M. Nelson, M. Mahalingam, R. Arunachalam, TCP/IP offloading for virtual machines, U.S. Patent 7424710 B1 (2008)

[21] B. S. Ang, An evaluation of an attempt at offloading TCP/IP protocol processing onto an i960rn-based NIC, TCP/IP networking: Intelligent Network Interface (2001)