Tualatin: Towards Network Security Service Provision in Cloud Datacenters

Xiang Wang[1, 2], Zhi Liu[1, 2], Jun Li[2, 3]
[1]Department of Automation
[2]Research Institute of Information Technology, Tsinghua University
[3]Tsinghua National Lab for Information Science and Technology
Beijing, China
{xiang-wang11, zhi-liu12}@mails.tsinghua.edu.cn
junl@tsinghua.edu.cn

Baohua Yang
IBM China Research Lab, Beijing, China
baohyang@cn.ibm.com

Yaxuan Qi
Yunshan Networks Inc., Beijing, China
yaxuan@yunshan.net.cn

***Abstract*—Multi-tenant infrastructures deployed in cloud datacenters need network security protection. However, the rigid control mechanism of current security middleboxes induces inflexible orchestration, limiting the agile and on-demand security provision in virtualized datacenters. This paper presents Tualatin, a consolidated framework of delivering security services in multi-tenant datacenters. It meets security requirements of different scenarios by hardware and software co-design. Leveraging Software-Defined Networking (SDN) and OpenFlow techniques, Tualatin provides fine-grained security protection in dynamically changing network topologies, where both switches and security middleboxes are programmatically controlled by logically centralized controllers. With service-level APIs exposed, Tualatin could be easily integrated with other Cloud Management System (CMS). A proof-of-concept system has been deployed in a Tier-IV datacenter, providing customizable network security services for tenant Virtual Private Cloud (VPC) infrastructure.**

Keywords—Software-Defined Networking; Cloud Datacenter; Network Security

I. INTRODUCTION

More companies turn to Infrastructure as a Service (IaaS) to fast deploy their applications. Cloud providers rely on network security middleboxes, such as Firewall (FW), Intrusion Prevention System (IPS) and Anti-Virus Gateway (AVG), to ensure the security of tenant applications [1]. In multi-tenant cloud datacenters, the network security service should be provided with the following features:

High performance and efficient resource utilization: Tenant applications deployed in datacenters usually require high bandwidth and low latency. To protect these applications without compromising the performance, security processing should be accomplished as fast as possible. Besides, cloud providers need to weigh between hardware and software based solutions for both packet header and packet payload inspections. Cost-effective security provision is able to exploit the benefits of hybrid solutions, imposing less impact on both compute and network resources.

Disparate and intricate security management: Each tenant shall be able to independently define its own security policy with flexibility. Besides, security services involve a group of inspection chains for different flows, to perform fine-grained processing. Some functions may be even dynamically enabled. For example, encrypted traffic should be bypassed ahead of packet payload inspection middleboxes to avoid unnecessary processing, and new FW rules need to be inserted corresponding to AVG alerts to suppress malicious flows. Moreover, the security processing capacity should be able to dynamically scale up/down, keeping up with traffic load variation.

Service-level security provision and orchestration: Tenant applications have diverse security requirements at different places of datacenter network. As clouds provide *on-demand* and *pay-as-you-go* services, it is difficult to tackle these diversities within single hardware middlebox. Besides, it is also uneconomic to statically allocate dedicated resources for different tenants. Furthermore, the large scale of cloud computing prompts the orchestration transforming from device management to catalogued services coordination, to reduce labor cost.

However, today's security services are commonly delivered via standalone and specialized devices at restricted physical position. Cloud operators face the challenges of coordinating these closed devices effectively in the following aspects:

- **Inadequate hardware-software collaboration:** Hardware middleboxes could achieve great performance for a large number of security functions. However, their awkward position at Top of Rack (ToR) makes it costly to steer traffic out of physical servers for intra-server traffic inspection. On the contrary, security Virtual Machines (VMs) could be distributed within physical servers across the whole datacenter, which eliminates the needless traffic steering. But it induces the large performance penalty that consumes much more CPU and memory resources of physical servers for Deep Inspection (DI) security services.
- **Overwhelmed network control plane:** OpenFlow [2] / Software-Defined Networking (SDN) [3] holds the promise of simpler solutions for security middlebox coordination. Most existing arts dynamically setup switch flow table via logically centralized controllers to perform inspection composition, dynamic provision, and processing capacity scaling. However, different from stateless network forwarding, most security inspections are usually stateful. They require middleboxes to main-

tain traffic sessions and check all packets of each flow, to rebuild protocol states. This tremendous workload will overwhelm network controller, and therefore ends up with slow and error-prone security services.

- **Configuration and management complexity:** Today, security middleboxes from different vendors are of mostly nonstandard and proprietary configuration interfaces. Besides, the lack of programmability leads to the limited extensibility and cohesiveness of middlebox in cloud ecosystem, prohibiting automatic control and agile service provision. Existing works have shown that most network operators consider the manual misconfiguration as the typical reason of service failure [1].

To address these issues, this paper presents Tualatin, a consolidated framework of elastic security service provision for virtualized datacenters. New security requirements of virtual network protection are investigated based on the Virtual Private Cloud (VPC) [4] service model that is an enterprise-grade cloud solution. Security policies of *intra-Virtual Network (intra-VN)*, *inter-Virtual Network (inter-VN)* and *Internet-Virtual Network (Internet-VN)* are enforced via the co-design of software and hardware. Tualatin leverages OpenFlow / SDN techniques to perform fine-grained traffic classification that facilitates lightweight inspection and brings trusted traffic bypassing forward. With the newly introduced *Security Workload Scheduler (SWS)*, Tualatin enables the composition of application-level inspections and the scaling of processing capacity. Tualatin controller exposes service-level APIs to coordinate with Cloud Management System (CMS) for network security service orchestration. The main contributions of this work are:

- **Security requirement categorization of modern IaaS:** Based on our experience of real deployment, VPC service delivery with multiple virtual networks requires new security protection features. Different from the emphasis of network security service provision at enterprise network gateway, we argue that it is inevitable for VPC to protect internal networks additionally.
- **System design and implementation:** This paper presents both system and software architectures of Tualatin. The deployment cohesiveness of intra-VN motivates dynamic security provision, to meet the performance requirement of tenant applications. Inter-VN security is enhanced by the elastic composition and scaling of security inspections, which adapts to the variation of tenant applications. Internet-VN security inspection is offloaded at datacenter gateway, enabling hardware acceleration.
- **System deployment:** The prototype system has been implemented and integrated with the LiveCloud CMS [5]. Being evaluated in a Tier IV datacenter, it provides tenants with *Secure VPC* service. The system not only prevents attacks from the Internet, but also isolates abnormal traffic within multi-tenant cloud infrastructure.

II. Related Work

The dynamics and large-scale properties of cloud datacenter promote infrastructure to open programming interfaces, to enable automation and reduce operational expenditure (OPEX).

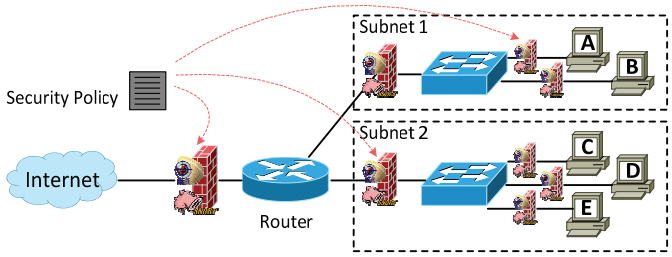

Fig. 1. Tenant view of secure VPC service

Borrowing the idea and achievements of open networking movement, several recent arts have addressed the management issues of generic middlebox. And the proposed solutions generally fall into the following three categories:

Programmable middlebox with open interfaces: CoMb [6] and xOMB [7] propose scalable middlebox architectures with programming interfaces. They implement middleboxes on commodity servers and operating systems with function modularity design. These proposed solutions show the feasibility of high performance, which are suitable for the implementation of security middleboxes deployed in cloud datacenters. SDMBN [8] takes a further step to the philosophy of middlebox interface abstraction. It groups middlebox states into four classes and abstracts interfaces for them respectively. All these works solve the middlebox intrinsic inefficiency of both resource usage and management. Nevertheless, these works rarely involve either the coordination of other networking elements or the deployment issues in security provision scenario.

OpenFlow application based security function implementation: FRESCO [9] presents a framework of developing OpenFlow based security applications. The implemented security functions run as modules on OpenFlow controller. This kind of solution profits from controller's rich APIs, but will probably saturate the control plane. Because security inspections, especially DI, require system control plane to scan all packets in each flow, not merely the first packet. Besides, its maintenance of flow states requires arduous engineering to support stateful applications, as the OpenFlow protocol provides little abstraction at this level.

SDN traffic steering based middlebox coordination: Several works [10] [11] [12] dynamically setup switch flow table to redirect the traffic traversing different middleboxes in desired sequence. Similar to our ideas, these solutions bring more flexibility and efficiency to security middlebox coordination. However, these works seldom address the support of multi-tenancy in cloud datacenters. Within these datacenters, flow-encapsulation based network virtualization [13] is applied to isolate traffic of different tenant, which conceals tenant original address space and traffic content from middleboxes. Besides, this type of solution merely relies on the OpenFlow switch to composite stateful services, which requires reverse engineering on packet payload to fetch application semantics. We argue that it may reduce the deployment feasibility in critical security scenarios, in which case our proposed SWS outperforms.

III. Secure Virtual Private Cloud

The Tualatin design begins by illustrating VPC model and discussing the security requirements of tenant application. Fig-

ure 1 shows the typical view of tenant's cloud infrastructure. This view is a general abstraction of existing arts [4] [5] [14].

A. Overview

VPC provides tenant VMs with more deployment flexibility. It defines a hierarchical topology for VM connectivity. Each tenant has one or more virtual L2 switches, i.e., virtual networks, over which L3 subnets could be assigned accordingly. All L3 subnets and the Internet are connected with each other via a virtual router. Moreover, every tenant is able to assign arbitrary L2 and L3 address spaces for VMs. Traffics of different virtual networks are isolated from each other via tagging or tunneling schemes [15] [16] [17]. Thus, the deployment of tenant application could be decoupled from rigid physical network restrictions, providing more flexibility.

B. Security Requirements

To provide flexible and fine-grained network security services, multiple policy definition points should be exposed. In Figure 1, intra-VN, inter-VN and Internet-VN security services could be defined at VM's access point, subnet gateway and VPC gateway, respectively. Security requirements for these three scenarios are categorized as follows:

Intra-VN: As traffic from infected VMs may impact on other regular VMs, we argue that it is necessary to provide intra-VN security in cloud datacenters. From both tenant and cloud provider's view, VMs of the same virtual network are usually clustered physically for performance consideration. Besides, VM live migration also occurs frequently and technically within the same virtual network [18]. Therefore, only a few lightweight inspections, e.g., packet header inspections, are provided, to avoid application performance and VM migration overhead. And heavyweight inspections, e.g., packet payload inspections, need to be dynamically activated only when suspicious activities happen, e.g., numerous semi-opened TCP flows are reported by NetFlow [19] traffic analyzer. The requirement could be described as: *To agilely and proactively enable security mechanisms within dynamic virtual networks.*

Inter-VN: Facing with the application expansion, tenants will require more VMs and subnets in VPC. The expansion firstly results in the increase of traffic volume to be inspected, where security processing capacity should be raised accordingly. Besides, it also results in the diversity of security protection among multiple virtual networks. For example, virtual network for database server deployment may need higher level security protection than that for web server deployment. Moreover, the compositing, load-balancing and scaling-out of security functions make the security provision even more complicated, since security control plane ought to parse and maintain application semantics. The requirement could be summarized as: *To efficiently deliver fine-grained security inspections, and elastically scale the processing capacity during the application expansion.*

Internet-VN: Tenant VPC gateway is the critical point that may suffer high-volume attacks from the Internet, such as Distributed Denial of Service (DDoS). Besides, mal-tenants in the same cloud may also launch attacks to other legitimate tenants. *We argue that it is necessary and suitable to leverage hardware devices to offload and accelerate the security inspection.*

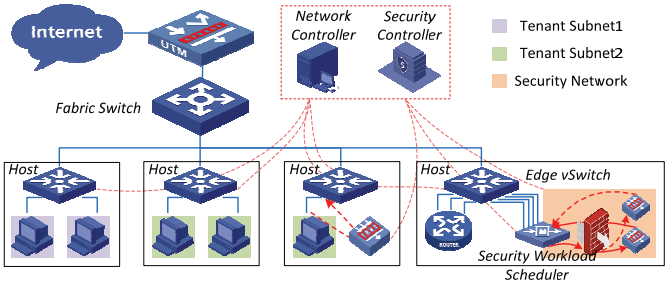

Fig. 2. Physical deployment of Tualatin system

Based on the above discussion, both security requirements and physical deployments of each scenario are quite different from those of the others. Thus, it is difficult and inefficient to deliver security service by either standalone hardware or pure software solution. We believe that a hybrid security mechanism is more appropriate to meet the overall security requirements. This philosophy guides the entire Tualatin design. In the next section, Tualatin architectures are presented in details to elaborate how these fundamental requirements are tackled.

IV. Tualatin Design

In order to clearly describe the design of Tualatin, the framework is demonstrated in two types of architecture: system and software. The system architecture describes the enforcement of security policies at data plane in physical topology. Security policies for these three scenarios discussed above should be separately processed by specific mechanisms at appropriate places, which could contribute to efficient resource utilization. The software architecture, which is more important, presents the security processing logic of the system control plane. In Tualatin, network security is packed and delivered as a service. It cooperates with other service modules of CMS by exposing service-level APIs, which highly facilitates the service provision and orchestration.

A. System Architecture

In cloud datacenters, network virtualization technology [13] decouples network services from hardware devices. Therefore, VMs carrying tenant applications could be deployed at arbitrary places. However, in real deployment, network forwarding capacity and computer hardware parameters may limit the VM distribution. For example, commodity switches cannot sustain large east-west traffic in case of VMs in the same virtual network distributed among different servers. VM live migration usually requires source and destination servers having the same CPU features. Besides, network services, such as tenant virtual routers, could be delivered via dedicated hardware that supports multi-tenancy, but it is probably a long-term solution. Therefore, we argue that the system architecture design should take these realities into account. Figure 2 depicts Tualatin system architecture via the physical deployment of tenant application shown in Figure 1. VMs in the same virtual network are normally assigned to adjacent locations, i.e., within the same server or rack. This strategy ensures the performance of tenant applications in practice. The tenant virtual router is implemented by Network Functions Virtualization (NFV) [20] enabled software entities, e.g., VM or service daemon.

For intra-VN security, as VMs migrate frequently, security policies are *per-source* managed [21]. This methodology can reduce the complexity of policy maintenance during VM migration. Benefiting from arbitrary matching on packet header fields of OpenFlow protocol, Tualatin employs *OpenFlow compatible* software switches [22] to facilitate the enforcement of intra-VN security policies at the hypervisor of each host. For packet header inspections, such as Access Control List (ACL) and Quality of Service (QoS) policies, Tualatin directly employs the supported features and OpenFlow entries of software switch to execute the policy. For packet payload inspection, Tualatin redirects the traffic to security VMs via fine-grained OpenFlow entries, which is enabled *on demand*.

For inter-VN security, the cluster of diverse detection engines is employed to deal with the scalability and composition of security inspections. The engine cluster is deployed close to the tenant virtual router, avoiding unnecessary traffic steering that consumes network bandwidth. *Security workload scheduler* is introduced as the frontend of multiple VM-based detection engines in deployment. It classifies traffic at application level, and maintains sessions for composition and load-balancing. Within the applied virtual security network, the scheduler redirects traffic across security engines in desired sequence. It is also convenient to plug in or unplug engines to the virtual security network to regulate the overall processing capacity. This design brings more flexibility, and its rationality will be discussed in the following software architecture part.

For Internet-VN security, the processing is offloaded to hardware devices. As all tenant virtual routers are connected to the Internet, sharing the same Internet address space, existing hardware security middleboxes that seldom acquire rich and direct support of multi-tenancy could outperform at this place. With plenty of security checking tasks in common, different tenants could share the computing resources of hardware security middleboxes. Thus, it is reasonable for cloud providers to employ Unified Threat Management (UTM) [23], Next Generation Firewall (NGFW) [24] or other series of hardware security middleboxes to protect the infrastructure away from large-volume attack traffic.

B. Software Architecture

Tualatin software architecture design follows the *shared nothing* principle [25], which CMS mostly employs to achieve scalability and elasticity. Different functions are abstracted as services, and interact with each other via explicit APIs. In this work, we argue that it is also necessary to deliver *Network Security as a Service (NSaaS)* that is coordinated by CMS. As Figure 2 shows, standalone security controller, as well as network controller and other devices, are connected to each other via datacenter control plane. The control application running on the security controller monitors cloud states and provides elastic processing capacity of different security inspections.

1) Tualatin control plane

The Tualatin control plane is mainly responsible for compositing different types of security inspections and delivering security detection capacity, which requires proactivity, scalability and agility. Figure 3 shows the design of Tualatin control plane, along with the other cooperating services. All these services are equally under the scheduling of CMS orchestration tools. Each service manages specific infrastructure resources, and provides corresponding abstracted functions that are not overlapped. Thus, the other services could invoke the exposed APIs to request resources for their internal usage, leaving the collision of resource management solved in single encapsulated service. For instance, *network security service* could register security VM images to the *image service* in advance, and also could request one virtual network from *network service* for the connectivity among SWS and multiple detection engines.

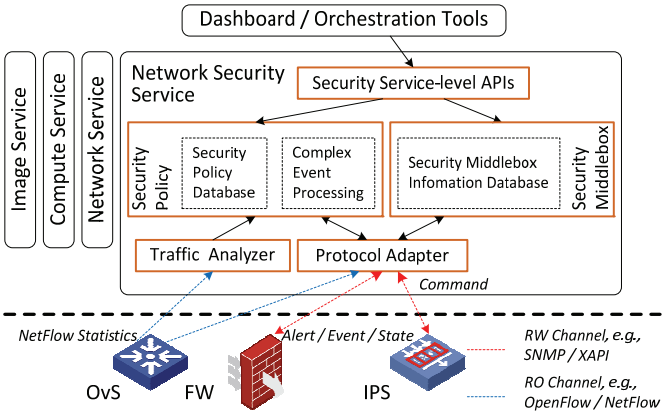

Fig. 3. Tualatin control plane design

In order to deliver network security service with early prediction, Tualatin control logic tries to make more precise decision based on the comprehensive analysis of entire infrastructure states. Therefore, hardware and software security middleboxes, as well as switches and other manageable devices, are monitored. Following the shared nothing principle, Tualatin control logic has full control permissions of security middleboxes and only receives notifications from other devices. For example, Tualatin controller builds Read Only (RO) channels over OpenFlow and NetFlow protocols to sniff network states, and supervises security middleboxes via Read Write (RW) channels. The *protocol adapter* maintatins these channels and sends heartbeats periodically to keep alive. As large-volume NetFlow statistics may impact on critical operations transmitted on other channels, standalone *traffic analyzer* is employed in real deployment, which aggregates raw NetFlow statistics for the following processing.

The essential parts of Tualatin control logic are modules of *security policy* and *security middlebox*. The former module manages security policies defined by tenants, and mainly consists of two elements: *security policy database* and *Complex Event Processing (CEP)* engine. The policy format has three fields: *i) flow pattern*: the matching condition of packet header fields, which is supported by OpenFlow; *ii) state condition*: the temporal and statistical matching condition which is supported by CEP query language; *iii) action*: ACL / QoS actions or the sequence of multiple inspections that are supported by security middleboxes. Once tenants assign the state condition in security policy, Tualatin control logic will insert corresponding query statements in CEP engine. The defined policies are stored into the security policy database. The CEP engine takes notified alert / event / state and aggregated NetFlow statistics as the input to exploit the abnormalities, which triggers dynamic se-

TABLE I. TAXONOMY OF NEWLY INTRUDUCED APIS

Service	Class	Entity
Network	CRUD	FlowSlice
Network Security	CRUD	SecurityPolicy
Network Security	CRUD / Attach / Detach	SecurityEngine

curity provision. The latter module stores the configuration and running states of security middleboxes that are actually deployed into the *security middlebox information database*. The stored information is primarily composed of the type, location and signature sets of middleboxes, the binding information of security policy and running statistics, e.g., CPU load and memory usage. To alleviate the IO pressure of transactional database, Tualatin employs NoSQL[1] techniques to cache the dynamically changing information, especially the running statistics.

Two categories of APIs for the management of security middleboxes and security policies are abstracted and exposed on the basis of two modules mentioned above. Tualatin control logic conceals the details of tedious internal processing from end users, which facilitates the orchestration. For example, when defining the security policy with the action of multiple inspections, tenants only need to specify the inspection type in desired sequence. Tualatin control logic is responsible for determining the number and location of security middleboxes to be booted. The exposed APIs will be introduced in the following service orchestration part.

2) Security workload scheduler

Before introducing the SWS, it is necessary to review the processing of traditional security mechanisms. Current hardware security middleboxes are usually deployed as *bump in the wire*. The first packet of every flow is forced to enter a device and goes through the following stages:

- **Policy lookup:** It figures out which policy the current flow matches, and decides whether this device should inspect or bypass the flow.
- **Session construction:** It computes the processing chain of the current flow and constructs session entry, which is based on the policy action, protocol semantics, and security engine status.

There are two key observations during these stages. Firstly, security middleboxes have to allocate resources to maintain the bypassed sessions, which may cost too much memory usage. Secondly, security middleboxes should keep tracking the protocol semantics of the inspected flows at application level, which increases the complexity of system control plane, comparing to that of forwarding only. Security policies are predefined statically, but flow sessions are constructed dynamically. As a consequence, it is practical to leverage OpenFlow rich matching capability to separate trusted and un-trusted flows. But the same solution is not suitable for inspection composition and processing capacity scaling. For example, to deal with FTP-like protocols that establishes multiple 5-tuple (source IP, destination IP, source port, destination port, and IP protocol)

[1] Redis. http://www.redis.io
Memcached. http://www.memcached.org

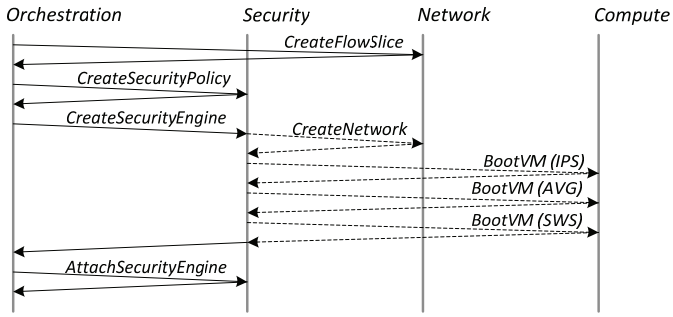

Fig. 4. Sequence of API calls for composited security inspection

flows in single transaction, security middleboxes need to scan every packet during control phase to find out the transmission flow in data phase, and some inspections may even need both of these flows to be directed to the same detection engine. If OpenFlow switches are employed for these tasks, it will consume a great deal of flow entries to store the fine-grained sessions. Besides, facing with the explosion of mobile applications, it is impractical for OpenFlow compatible switches to keep identifying ever-increasing applications [26]. In another way, the identification task could be implemented on OpenFlow controller, but the control plane will be overwhelmed to inspect every data plane packet. The essential behind these differences is the much higher complexity of security middlebox control plane, and the existing network L2 / L3 oriented SDN architecture design cannot sustain the overload.

To address the problem, SWS is introduced, which is a VM-based dispatcher sitting in data plane. It offloads part of security control plane tasks, especially the work in session construction stage. Thus, it provides network security service with more efficiency, flexibility and scalability. The SWS connects multiple virtual networks of the same tenant, as well as virtual security network, illustrated as Figure 2 shows. The SWS has two pairs of virtual interfaces, which are both connected to the OpenFlow compatible software switch at hypervisor. Each pair is assigned to the same virtual network, inspecting inbound and outbound traffic at the virtual network gateway. Security policies, which specify the flow that needs security inspections provided by security middleboxes, are deployed on the software switch that SWS connects. When packets arrive at the switch and match the flow pattern of security policy, the switch steers packets to SWS. The SWS is responsible for directing packets to go through appropriate inspection engines in tenant-defined sequence, without breaking protocol semantics. Security middlebox providers could design their own efficient session data structure and deploy multiple schedule strategies on SWS, which brings more flexibility to security control plane design.

C. Service Orchestration

Tualatin control plane exposes service-level APIs to facilitate the security provision. These APIs could be catalogued into two groups: the one is security middlebox manipulation APIs, and the other is security policy definition APIs. Besides, the network service APIs are also expanded to support fine-grained traffic steering. Table 1 lists the taxonomy of these APIs.

FlowSlice is introduced to define flows that require security inspections. Its creation takes the interface that connects tenant

VM and the matching pattern of packet header fields as inputs, and outputs the generated unified identifier. Specifically, network control logic installs the corresponding flow entry on OpenFlow switch with forwarding action. As described above, *SecurityPolicy* defines the matching conditions and actions. If the action belongs to ACL or QoS scope, the control logic invokes FlowSlice APIs to update the corresponding action of flow entry. If the action is multiple inspections supported by security middleboxes, the orchestration tool needs to further request *SecurityEngine* that is associated with the SecuirtyPolicy instance. The creation of SecurityEngine mainly involves the request for virtual security network and the bootup of security middleboxes. Their runtime parameters, such as signature sets, are pre-assigned in the action field of security policy. Each group of these newly introduced APIs supports the operations of Create, Read, Update and Delete (CRUD). SecurityEngine is further endowed with Attach / Detach interfaces.

Figure 4 shows the orchestration steps for SWS based Web traffic inspection which composites both IPS and AVG services. It is assumed that tenant VPC infrastructure has already been deployed. The orchestration tool firstly requests one flow slice capturing inbound and outbound Web traffic at router's interface which connects tenant subnet. Then the requests for security policy and engines that cascade IPS and AVG with static state condition and Web flow pattern are sent to the network security service. The control logic checks the state condition field of security policy, and immediately invokes network and compute services to apply for one virtual security network and VMs of both IPS and AVG. After succeeding, the security engine is patched into the applied flow slice.

V. IMPLEMENTATION

The prototype system is implemented and evaluated on generic X86-64 commodity servers. In this section, data plane and control plane implementations are presented separately.

Tualatin data plane: Xen Cloud Platform (XCP) is employed to build the virtualization environments. Open vSwitch is delivered within this platform, acting as the access switch of VMs. Overlay network virtualization mechanism based on GRE protocol is developed to implement Tualatin network data plane. Open vSwitch is responsible for encapsulating and transmitting traffic. Fine-grained flow steering, normal L2 forwarding and virtualization tunneling are executed on three individual bridges, respectively. Although these functions could be implemented using single bridge [13], our method makes network forwarding and security orthogonal, which has less modification of existing virtual network service. Different security inspection engines are built on open source software[2] and delivered in VM. Several modifications are carried out to expose the device states and enable the event notifications. Each security VM has a pair of network interfaces which differentiates inbound and outbound traffic, and is deployed in transparent mode.

Tualatin control plane: Tualatin controller is architected for multi-thread mode and implemented in C language. It abstracts the unified processing model, and multiple protocols could be flexibly supported as plugins. Currently, the controller supports OpenFlow and OVSDB[3] protocol to manage switches, XAPI[4] protocol to manipulate VMs, and SNMP[5] protocol to interact with legacy hardware devices. Ntop[6] is employed as traffic analyzer which runs on standalone physical server. Dynamic security policy processing logic is implemented on the basis of open source CEP engine. Both relational database and NoSQL key-value storage are employed to store critical configuration and instantaneous states, respectively. The SWS derived from our previous work [23] is optimized for transparent transmission mode. It maintains two-wing flow sessions, parses application protocols, composites inspections and distributes workload according to the detection engine states.

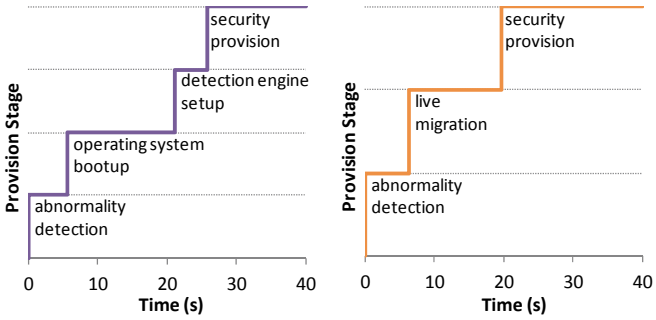

Fig. 5. Intra-VN security provision time for cold start and hot migration

VI. EVALUATION

In this section, we evaluate the effectiveness of Tualatin design and the performance of Tualatin control logic. The prototype controller is deployed on a single Intel i7-4770 3.4GHz CPU workstation with 32GB DDRIII memory and CentOS 6.5 64-bit operating system. The virtualization platform on which SWS and other security VMs run is deployed on multiple two 4-core Intel Xeon E5620 2.4GHz CPUs servers with 48GB DDRIII memory. The local storage of physical server is SATA hard disk. The release version of XCP is 1.6, and the Open vSwitch is updated to version 1.7.3.

A. Agility of intra-VN security provision

The provision time, which starts from the launch of malicious behavior and ends up with the completion of security middlebox setup, is measured to evaluate the agility of intra-VN security provision. We conduct two tests for different provision methods: the one is cold start from local server; the other is hot migration from remote server. The provision time of the former test mainly consists of abnormality detection time, operating system bootup time, and detection engine setup time. The provision time of the latter test is primarily determined by abnormality detection time and migration time. Since Tualatin control logic only updates the action field of flow entries, the flow setup time could be ignored in these tests. During the testing, intra-VN IPS inspection will be enabled when TCP SYN

[2] Snort. http://www.snort.org
ClamAV. http://www.clamav.net

[3] OVSDB. http://tools.ietf.org/rfc/rfc7047.txt
[4] XAPI. http://www.xenserver.org
[5] NetSNMP. http://www.net-snmp.org
[6] ntop. http://www.ntop.org

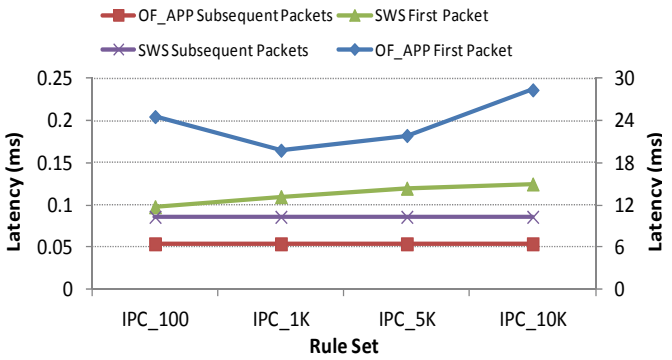

Fig. 6. Network latency comparison on different scales of rule set

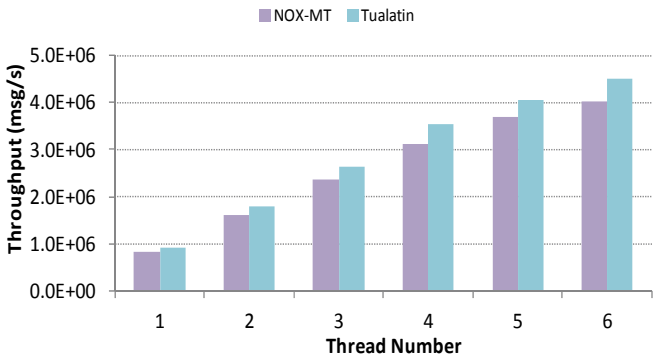

Fig. 7. Tualatin IO throughput on different number of threads

flood is detected by traffic analyzer, and nmap[7] is employed for port scanning. Figure 5 shows provision stages over time for cold start and hot migration. It is observed that Tualatin control logic takes about 5*s* to detect abnormality, which is the finest aggregation cycle for traffic analyzer. Overall, Tualatin is able to deliver intra-VN security within 30*s* for cold start case, which could be regarded as the worst case in current deployment. The usage of solid-state drive (SSD) will boost the speed. The provision time of hot migration is about 20*s* when using 1*Gbps* link connection, implying faster provision speed when using 10*Gbps* Ethernet.

B. *Security Workload Scheduler Efficiency*

The SWS is introduced to simplify the implementation of network security control plane, but it also brings one more hop in packet transmission. From the above discussion, it is obvious that SWS eliminates massive exact-matching switch flow entries that are used for inspection composition, load-balancing and processing capacity scaling. Besides, the SWS could leverage wildcard-matching switch flow entries to bypass unwanted traffic ahead, alleviating the inspection workload. Thus, only the evaluation of introduced latency is conducted on multiple scales of security policy that are generated by ClassBench[8]. As a comparison, an OpenFlow based ACL application is also implemented, which runs on remote controller. The result is shown in Figure 6 (The first packet latency of OpenFlow ACL application corresponds to the right axis). It could be observed that SWS based solution eliminates the latency of first packet accessing the remote controller, which jitters around 25*ms*. And the cost is the latency increase of subsequent packets. It also shows that the latency overhead of extra hop across the SWS is within 40 μs which could be negligible.

C. *Controller Performance*

The throughput of Tualatin controller IO and CEP logic is measured for the evaluation of controller performance. Based on the deployment experience of our previous controller [5], the efficiency of IO logic affects the controller overall performance to a large extend. Since most control operations usually wait for the readiness of required resources, Tualatin controller is implemented in fully asynchronous mode. Thus, a large portion of controller workload is imposed on the IO processing of cloud events. In order to compare with existing controller framework, cbench[9] is employed to evaluate the processing capacity of OpenFlow protocol events. As other supported protocols share majority of the processing logic, the evaluation result could be regarded as universal. Figure 7 depicts the average controller throughput comparing with multi-thread NOX[10] on *packet_in* processing of learning switch usage. This test case is the standard testing that reflects OpenFlow controller IO capacity. It could be observed that the Tualatin controller throughput is linear along with the increase of thread number. The throughput with 6 working threads exceeds 4.5M *msg/s*, which outperforms the one of multi-thread NOX. The CEP performance is mainly impacted by both the number and the complexity of query statements (state condition field in Tualatin security policy). Standalone statistical query is simpler than the one with additional temporal factor. Tualatin control logic tries to translate security policies that are of the same or similar state conditions into single query statement, which could reduce the workload of CEP. Figure 8 shows the CEP throughput of two different query types on multiple query statements. The results are for single CEP engine. Along with the increase of query number, the throughput is reduced accordingly. For the more complex statistical plus temporal query, the throughput is about 257K *event/s* that is acceptable in the deployment of moderate scale. Future work is needed to promote its processing capacity.

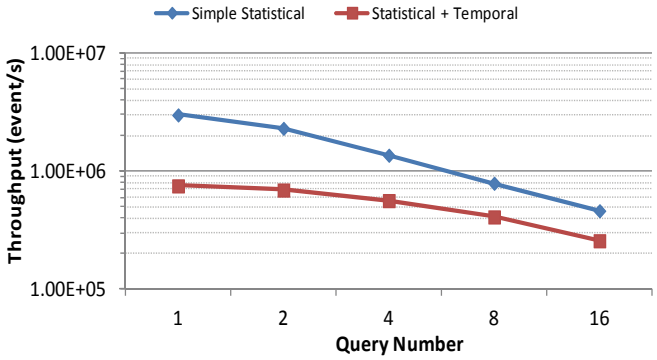

Fig. 8. Tualatin CEP throughput on different number of queries

[7] nmap. http://www.nmap.org
[8] ClassBench. http://www.arl.wustl.edu/classbench/
[9] cbench. git://gitosis.stanford.edu/oflops.git
[10] NOX. git://github.com/noxrepo/nox.git

VII. Conclusion and Future Work

This paper presents Tualatin, a consolidated architecture to provide network security services for tenant cloud infrastructures. Its hardware-software co-design satisfies diverse security requirements of VPC service model with more efficient resource usage. Combining SDN and the newly introduced SWS, Tualatin could composite complex security inspections with scalability and agility. Evaluations show that Tualatin could deliver security provision agilely, with little impact on data plane latency. And the controller also achieves high throughput.

In cloud datacenters, network virtualization solutions encapsulate tenant traffic, thus tenant topology are free from the restriction of physical topology. It speeds up the deployment of tenant application, but at the same time makes it more challenging to protect the overall network infrastructure. Tualatin delivers security services to tenant virtual networks, while the physical carrier network may still be vulnerable to attack risks. In practice, though existing security technologies could ensure the protection in most cases, the optimal solution that meets all requirements is still an open question. The open source cloud computing platform also addresses network security service provision recently [27] and is developing its extensible framework. It is worth integrating our work into the open ecosystem. Additionally, besides network security, services that other middleboxes currently provide still need adjustments to accommodate to the virtualized environment. A unified SDN [28] platform may afford more efficient and convenient ways. We identify these issues as our future works.

Acknowledgment

This work is supported by IBM Global Shared University Research (SUR) Program.

References

[1] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar. Making middleboxes someone else's problem: network processing as a cloud service. In Proc. of SIGCOMM, 2012.

[2] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. OpenFlow: enabling innovation in campus networks. In SIGCOMM CCR, 2008.

[3] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, and S. Shenker. Onix: a distributed control platform for large-scale production networks. In Proc. of OSDI, 2010.

[4] Amazon Virtual Private Cloud. http://aws.amazon.com/vpc/

[5] X. Wang, Z. Liu, Y. Qi, and J. Li. LiveCloud: a lucid orchestrator for cloud datacenters. In Proc. of CloudCom, 2012.

[6] V. Sekar, N. Egi, S. Ratnasamy, M. Reiter, and G. Shi. Design and implementation of a consolidated middlebox architecture. In Proc. of NSDI, 2012.

[7] J. W. Anderson, R. Braud, R. Kapoor, G. Porter, and A. Vahdat. xOMB: extensible open middleboxes with commodity servers. In Proc. of ANCS, 2012.

[8] A. Gember, P. Prabhu, Z. Ghadiyali, and A. Akella. Toward software-defined middlebox networking. In Proc. of HotNets, 2012.

[9] S. Shin, P. Porras, V. Yegneswaran, M. Fong, G. Gu, and M. Tyson. FRESCO: Modular composable security services for software-defined networks. In Proc. of NDSS, 2013.

[10] Z. A. Qazi , C. Tu, L. Chiang, R. Miao, V. Sekar, and M. Yu. SIMPLE-fying middlebox policy enforcement using SDN. In Proc. of SIGCOMM, 2013.

[11] K. Wang, Y. Qi, B. Yang, Y. Xue, and J. Li. LiveSec: towards effective security management in large-scale production networks. In Proc. of ICDCSW, 2012.

[12] A. Gember, A. Krishnamurthy, S. St. John, R. Grandl, X. Gao, A. Anand, T. Benson, A. Akella, and V. Sekar. TR. Stratos: a network-aware orchestration layer for middleboxes in the cloud, 2013.

[13] T. Koponen, K. Amidon, P. Balland, M. Casado, A. Chanda, B. Fulton, I. Ganichev, J. Gross, N. Gude, P. Ingram, E. Jackson, A. Lambeth, R. Lenglet, S. Li, A. Padmanabhan, J. Pettit, B. Pfaff, R. Ramanathan, S. Shenker, A. Shieh, J. Stribling, P. Thakkar, D. Wendlandt, A. Yip, R. Zhang. Network Virtualization inMulti-tenant Datacenters. VMWare TR-2013-001E, 2013.

[14] J. Mudigonda, P. Yalagandula, J. Mogul, B. Stiekes, and Y. Pouffary. NetLord: a scalable multi-tenant network architecture for virtualized datacenters. In Proc. of SIGCOMM, 2011.

[15] STT: A Stateless Transport Tunneling Protocol for Network Virtualization. http://tools.ietf.org/id/draft-davie-stt-04.txt

[16] NVGRE: Network Virtualization using Generic Routing Encapsulation. http://tools.ietf.org/id/draft-sridharan-virtualization-nvgre-04.txt

[17] VXLAN: A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks. http://tools.ietf.org/id/draft-mahalingam-dutt-dcops-vxlan-08.txt

[18] V. Soundararajan and K. Govil. Challenges in building scalable virtualized datacenter management. In SIGOPS OSR, 2010.

[19] NetFlow. http://www.ietf.org/rfc/rfc3954.txt

[20] Network Functions Virtualisation. http://portal.etsi.org

[21] M. Moshref, M. Yu, A. Sharma, and R. Govindan. Scalable rule management for data centers. In Proc. of NSDI, 2013.

[22] Open vSwitch. http://www.openvswitch.org

[23] Y. Qi, F. He, X. Wang, X. Chen, Y. Xue, and Jun Li. OpenGate: towards an open network services gateway. In Computer Communications, 2010.

[24] Next-Generation Firewalls (NGFWs). http://www.gartner.com/it-glossary/next-generation-firewalls-ngfws

[25] BasicDesignTenets. https://wiki.openstack.org/wiki/BasicDesignTenets

[26] OpenAppID. http://newsroom.cisco.com/release/1354502

[27] Neutron. https://wiki.openstack.org/wiki/Neutron

[28] OpenDaylight. http://www.opendaylight.org/