NetSecCC: A scalable and fault-tolerant architecture for cloud computing security

Noname manuscript No.
(will be inserted by the editor)

NetSecCC: A Scalable and Fault-tolerant Architecture for Cloud Computing Security

Jin He · Mianxiong Dong · Kaoru Ota · Minyu Fan · Guangwei Wang

Received: date / Accepted: date

Abstract Keywords security group, security inspection chain, scalability, fault tolerance, on-demand service

1 Introduction

As one of the most influential computing paradigms in recent years, cloud computing not only noticeably reduces capital expenditures, but also largely improves computational efficiency, and thus successfully attracts extensive attentions from both academia and industry. As a result, cloud computing is widely used in various IT services, including but not limited to, parallel computing, visualization, network storage technologies, load balance, utility computing, service-oriented etc.

Behind the great success and potential of cloud computing, there is a big challenge to ensure its security. Though cloud computing also involves other types of security issues (e.g., data security), network security is considered as one most prominent issue that must be solved [6] [9] [36] [42] [12] [2]. Indeed, as stated by National Vulnerability Database [15], there are 84 network vulnerabilities discovered in cloud computing by February 2013, all of which strongly threaten the security of cloud computing. Other evidences [14] [10] [34] [41] also reveal that malicious network attacks are responsible to a large number of data destruction or tampering or forgery in cloud computing. Unless network security in cloud computing is properly ensured, all cloud computing based services are exposing to a high risk of attacks.

To protect network security, a traditional architecture is to place network security devices (middleboxes [8]) at front-end of cloud computing as shown in Fig. 1. Though this traditional architecture addressed many concerns in network security, it is faulted in several important aspects. ***Lack of network security protection between virtual machines (VMs)***: Since a compromised VM easily attacks other VMs in a same hardware platform by virtual network [4] [39], However, the traditional architecture is lack of an internal network protection mechanism between VMs. ***Difficult scalability***: The traditional architecture presents such a scenario: traffic bursts and exceeds the maximum capacity of the existing deployed middleboxes at some points, while network traffic flows under normal circumstances. If we add a number of corresponding middleboxes to reduce traffic loss at peak load, it results in not only less efficient resource utilization, but also higher deployment and maintenance costs. ***Difficult fault-tolerance:*** Using hot standby (HS) in the traditional architecture can offer fault tolerance for the failed middelboxes. However, one simple enterprise network requires 640 middleboxes to protect its security [32] [33], not to mention cloud computing that will host much more complex multi-services end users. In cloud computing, if we use the same hot standby to offer fault tolerance for such a large volume of middleboxes, this will result in unsustainable costs.

J. He, M. Fan and G. Wang
Department of Computer Science, University of Electronic Science and Technology of China (UESTC), Chengdu, 611731, P.R.China
E-mail: {hejin_some, ff98, wguanwei}@163.com

M. Dong
National Institute of Information and Communications Technology, Japan
E-mail: mx.dong@ieee.org, ota@csse.muroran-it.ac.jp

K. Ota
Department of Information and Electronic Engineering, Muroran Institute of Technology, Japan
E-mail: ota@csse.muroran-it.ac.jp

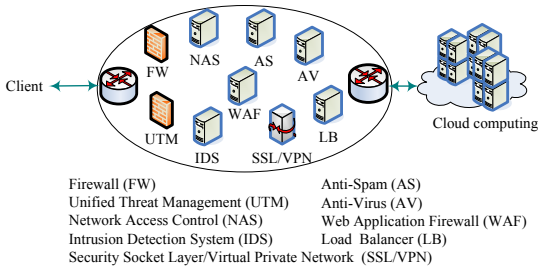


Fig. 1: The traditional architecture network security for cloud computing

Considering the above shortcomings of the traditional architecture, both industries and academies put many efforts on alternative solutions. In industry, *McAfee Security-as-a-Service* [27] merely provides Web and Email security protection in cloud computing, and thus it is lack of comprehensive multi-service protection in cloud. *Amazon Web Services* [16] only provides basic network security via a port-based firewall, and relies on third-party security vendors to provide robust network security with the granularity, control and reporting what customers need. *VMware vShield* (app, endpoint, edge, zones) [24] provides services in cloud with partial network security protection, but it is short of comprehensive and integrated capacity (e.g., encryption transmission, anti-virus). *Security as a Service* (SecaaS) [21] provides services in cloud with comprehensive security protection, including web, email and intrusion SecaaS, but it fails to include scalability and system fault-tolerance.

In academia, Wu *et al.* [40] aim to control the intercommunication among virtual machines with higher security by an embedded firewall in virtualized environment, but this method neither prevents malicious attacks from external traffic, nor provides flexible scalability and fault tolerance for the firewall. Salah *et al.* [31] propose a cloud-based security overlay network as a comprehensive protection solution for servers and end-users, but it is also lack of an effective scalability and fault-tolerance mechanism. Split/Merge [30] can be dynamically scaled out (or in) virtual middleboxes in cloud computing by SDN [11], which only focuses on load-balanced elasticity and system utilization without paying attention to preventing external and internal malicious traffic from attacking cloud services. [33] [28] well combine middleboxes with SDN to protect enterprise network security, and to provide a flexible scalability and fault-tolerance mechanism, but it is a pity that they are not suitable for cloud security.

Since it is not suitable or defective for the above efforts to protect network security of cloud computing, we propose a *NetSecCC* architecture that takes a novel approach of eliminating these disadvantages. It not only prevents external and internal malicious attacks and offers on-demand network security service for cloud users, but also is able to provide flexible scalability and fault tolerance for virtual middlebox load and failure, respectively. Experiments have further fully proved that *NetSecCC* has high performance in terms of scalability and fault tolerance, and also provides security services for cloud computing without much degraded system performance. In summary, our main contributions are as follows:

- ***An innovative architecture*** *NetSecCC* is a novel scalability and fault-tolerant security architecture using a systematic approach to properly provide security protection for cloud computing. The architecture provides balanced scalability alongside VM scale-in and scale-out for virtual middleboxes according to their loads, and offers many-to-one fault-tolerant mechanism to overcome disadvantages of the traditional HS for virtual middlebox failure.
- ***External and internal protection*** *NetSecCC* prevents malicious attacks from not only external traffic, but also internal traffic to ensure network security of cloud users' services in cloud computing.
- ***On-demand network security services*** *NetSecCC* provides on-demand network security services for different network security requirements from different services on cloud computing.

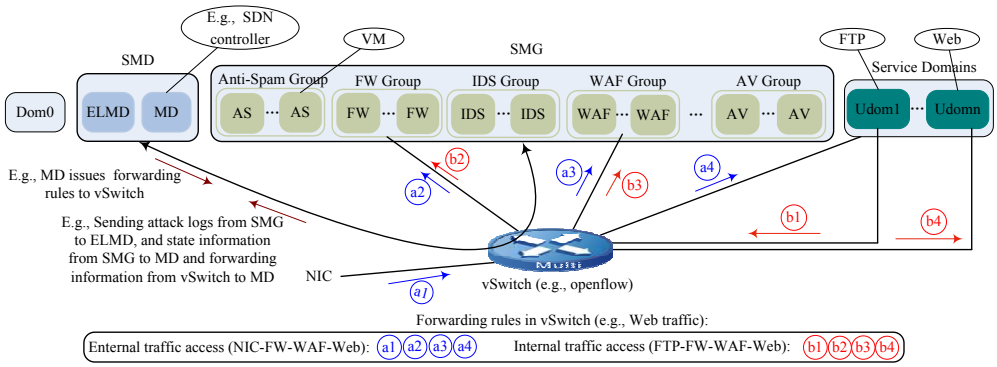
The rest of the paper is organized as follows: Section 2 provides an overview of the design of *NetSecCC*. Section 3 gives implementation details of the entire system. Section 4 shows various experimental results for evaluating the impact and performance of our system. We conclude our paper in Section 5.

2 Design

Before describing the *NetSecCC* design, we assume that hardware platform, hypervisor and VMs-OS on cloud computing are trusted, and just focus on network security of services placed on cloud computing. thereby ensuring that traffic arriving at services in service domains is secure and trusted.

2.1 Principle

In order to prevent external and internal malicious traffic from attacking cloud users' services in cloud computing, incompetence external traffic from Internet or internal traffic from VMs must be forwarded through a desired sequence of security groups in SMG (e.g., FW-WAF) to be inspected and filtered before arriving at services in service domains §(2.2).



MD, as a SDN controller, collects state information (e.g., load, failure) from every group in SMG, and receives forwarding information (e.g., traffic) from vSwich in a timely manner, and generates and issues forwarding rules to vSwitch according to SIC mapped by network security requirements of cloud users' services (To simplify, we call it SIC of cloud users' services) §(2.2), security domains topology, state information and forwarding information. VSwitch, as an openflow switcher forwards external traffic from Internet and internal traffic from VMs through the corresponding SIC according to forwarding rules in vSwtich. SMG, as a performer is comprised of various security groups (e.g., WAF group), and is responsible for filtering and inspecting incoming traffic before it is forwarded to service domains.

Preventing external and internal malicious attacks Incompetence external traffic from Internet or internal traffic from VMs, before arriving at services in service domains, must be forwarded through SMG, thereby ensuring services security. To make this process concrete, as shown in Fig. 2, we use the Web server in service domains as an example to elaborate the processing. When external traffic accesses to the Web server, it goes through NIC-FW-WAF-Web presented by blue areas to ensure that traffic arriving at the Web server is secure and trusted. When internal traffic from FTP domains accesses to Web server, it must go through FTP-FW-WAF-Web presented by red areas.

Scalability and fault tolerance Traffic is required to go through one or more security groups, each group on SIC path may suffer overload or low load or failure, so a scalable and fault-tolerant security architecture requires load-balanced dynamic elasticity and high availability in every group as shown in §(2.3). MD dynamically adjusts forwarding rules in vSwtich to achieve the purpose of load balancing and fault tolerance in each group, and updates rules in the following two stages: Initial phase, when cloud users employ their services in service domains before not running, MD generates forwarding rules in accordance with SIC of cloud users' service, security domains topology and current middleboxes load; Running phase, when middelboxs suffer overload or low load or failure, MD updates forwarding rules in vSwtich to rebalance middleboxes load for overload or low load, and provide fault tolerance for failure.

Compared with the traditional architecture, it can be observed from the *NetSecCC* work principle that it not only prevents external and internal malicious attacks and offers on-demand network security service for cloud users, but also is able to provide flexible scalability and fault tolerance for virtual middlebox load and failure. The focus of *NetSecCC* design is on-demand network security service and scalability and fault-tolerance of each group in SMG. SIC of cloud users' services focuses on on-demand security service (§2.2), while flexible scalability and efficient fault tolerance in each group (§2.3) can enhance load balancing and high availability, and improve resource utilization.

2.2 SIC

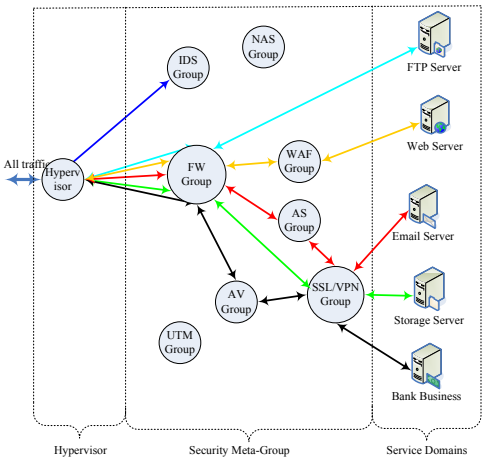


Fig. 2: Security Inspection Chains

Table 1: *NetSecCC*'s components and functions

Component	Function
Dom0	• weaken dom0 privileges, it has no permission to create/start and stop/destroy any domain in SMG. • dom0 still keeps such permissions to do with all domains in service domains and SMD, and manages resources, including scheduling time-slices, I/O quotas, etc.
SMD	It is composed of management domain (MD) and event and log management domain (ELMD). ELMD stores and manages events and logs from SMG, and provides the unified query for security managers. MD is responsible for three main functions: • create/destroy any domain in SMG. • collect state information (e.g., load, failure) from every group in SMG and receive forwarding information (e.g., traffic) from vSwich. • generate and update forwarding rules in vSwtich according to security inspection chains (SIC) §(2.2), security groups topology, virtual middlebox load and failure.
SMG	It is comprised of various security meta-groups (e.g., WAF group, IDS group, AV group). Every group includes one or more of virtual middleboxes of the same type (To simplify, we also call these virtual middleboxes security domains). Note that each virtual middlebox is installed in a standalone VM. • Security domains are responsible for traffic security inspection and filtering. • provide fault tolerant for the failed security domains by the improved Hot Standby (HS).
Service Domains	It hosts various types of Internet-based cloud users' services (e.g., FTP server, Web server).
vSwtich	It is responsible for receiving forwarding rules from MD, and forwarding external and internal traffic through security domains to be filtered and inspected.

SIC is a sequence of logical policy chains through one or more security groups (e.g., FW-WAF, FW-IDS), traffic accessing to service domains must be forwarded through the corresponding SIC to ensure the security of service domains. *NetSecCC* is able to provide suitable SIC for different network security requirements from different services, i.e., on-demand security service as shown in Fig. 3. Note that many middleboxes are stateful and need to process both directions of a session for correctness. To make this discussion concrete, we use two examples to further illustrate SIC.

Web server in service domains needs to solve these attacks from network-layer and application-layer. Attacks from network-layer include DDOS attack, syn attack, etc. Attacks from application-layer includes cross-site attacks, SQL injection, vulnerability overflow and so on. *NetSecCC* provides Web server security with SIC (FW-WAF) as shown in Fig. 3 with yellow lines, Web traffic must flow through FW and WAF to ensure the security of Web server, where FW group assures its network-lay security, WAF group offers its application-layer security, thereby ensuring that traffic reaching web service is secure.

Email server security requirements are able to protect against DDOS attack, syn attack, malicious e-mail, spam and virus e-mail, etc. Even the important emails need to be encrypted for transmission. *NetSecCC* provides email server with SIC (FW-AS-SSL/VPN), indicated in Fig. 3 with red lines, to guarantee its security. Where FW group secures network-layer security of email server, AS group filters malicious and spam e-mail to guarantee application-layer security, and SSL/VPN group provides the important emails with secure transmission.

2.3 Group Management

Incompetence external traffic from Internet or internal traffic from VMs accesses to services in service domains, MD as a SDN controller is responsible for controlling traffic to follow its corresponding SIC. While each group on SIC path is a real performer on security inspection and filtering, preventing malicious and virus attacks from arriving at services in service domains. In this process, when security domains (nodes) in some groups on SIC path suffer overload or low load or failure, *NetSecCC* needs to rebalance load in groups for overload or low load to strengthen network traffic processing capability, including increasing throughput and resource utilization, and to provide fault tolerance using hot standby for failure to improve seamless inspection and filtering, including reducing system recover time. Note that the HS is not a traditional one-to-one relationship [5] [38] [3] between active nodes and standby nodes, but a improved many-to-one relationship, that is, the state information of all active nodes is synchronized to one standby node to improve resource utilization.

When one group faces traffic overload or low load or node failure, as shown in Fig. 4, *NetSecCC* presents

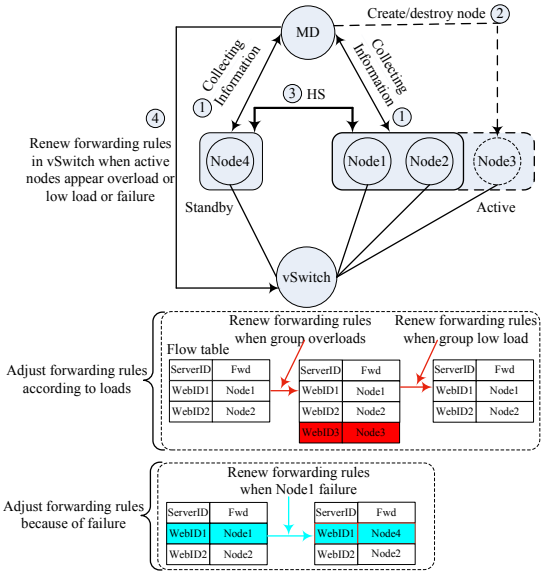


Fig. 3: Working mechanism of every group in SMG

how to deal with such a problem. MD collects and receives load and traffic information (e.g., session, load) from active nodes in real time (1); In the case of these active nodes overload or low load, MD makes such a determination according to the received information: If the load of the active nodes is not balanced, MD renews forwarding rules in vSwitch to adjust the load between the active nodes (4). If all active nodes overload, MD creates an active node (2), and dynamically generates new forwarding rules and renews those in vSwitch (4) to balance the load in active nodes. If active nodes face low load, MD may destroy an active node (2), and renew forwarding rules in vSwitch (4) to improve resource utilization. To make load balancing contrate, we present from Fig. 4 how MD adjusts flow table to rebalance load. Initially, WebID1 and WebID2 traffic is forwarded to node1 and node2, respectively, to be inspected and filtered. If WebID3 traffic arrives and goes through node1 and node2, they overload, so MD creates node3 and adds forwarding rules to route WebID3 traffic to node3. When WebID3 traffic ends, MD deletes rules forwarding traffic to node3 and destroys node3.

Since most middleboxes are stateful, a middelbox fails to result in loss of the established sessions in its memory. If a client accesses to the remote server again, the rebooted middlebox needs to reestablish a new session between client and server, resulting in a large latency. Although the traditional HS is able to solve this problem by one-to-one switch-over between active nodes and standby nodes, too many standby nodes seriously reduce resource utilization. Because the probability of middlebox failure is low, *NetSecCC* uses many-to-one mapping relationship between all active nodes and one standby node, that is, the state information in all active nodes is synchronized to a same standby node. When any active node fails, an automatic switch-over is achieved between the failed active node and the standby node, the standby node immediately plays the role of the active node, and MD changes forwarding rules to route traffic from the failed active node to the switched standby node. The improved HS not only overcomes a long latency to reboot the failed middlebox, but also improves system resource utilization. In a specific example as shown in Fig. 4, when node1 suddenly fails, node4 immediately plays the role of node1 by switch-over, and changes forwarding rules in vSwitch to route traffic from WebID1 to node4. This process is done automatically without human involvement. A more detailed process will be shown in Section 3.2.

3 Implement

The above design elaborates the principle of *NetSecCC*. In this section, we represent the implementation of *NetSecCC* in detail. In regard of the implementation of *NetSecCC*, we focus on the implementation of MD and security group. MD provides the corresponding SIC for cloud users's services placed on cloud computing to ensure their network security, and dynamically adjusts load balancing in security groups according to the load information. Security groups not only perform security inspection and filtering, but also improve quality of security service, including resource utilization, fault tolerance, high availability. We first demonstrate the implementation of MD.

3.1 MD Implement

During the implementation of *NetSecCC*, MD as a SDN controller plays two important roles: First, it controls traffic through the corresponding SIC of cloud users' services to ensure these services security. Second, it rebalances load due to traffic overload or low load or node failure in each group. MD implements its two functions by forwarding rules in vSwitch, and the process of implementation is shown in Fig. 5. Resource manager sorts and analyzes these data from the inputs: state information from groups in SMG (e.g., CPU, session) and forwarding information from vSwitch, groups topology and SIC, and outputs the parameters as the inputs of RouteGen. RouteGen converts these parameters into forwarding rules by forwarding traffic through groups in SMG to be inspected and filtered. Until deployment of new services or security needs change in service domains, traffic overload or low load or node

failure in groups, MD will not generate or renew rules in vSwitch.

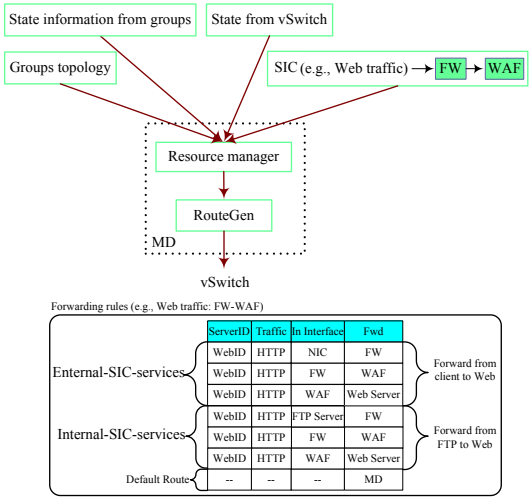


Fig. 4: MD implementation

Web server (Web security chains FW-WAF) as an example presents its forwarding rules by MD, as shown in Fig. 5. External traffic from a client to the Web server and internal traffic from the FTP server to the Web server are first forwarded to FW by vSwtich to be filtered, then forwarded to WAF to be inspected, and finally arrive at the Web server, thereby ensuring that traffic arriving at Web server is secure and trusted.

3.2 Security Group

MD described above, as a controller, performs network security inspection and filtering for network-based services in cloud computing, while security groups are regarded as an actual operator to put into effect specific security inspection. As shown in Fig. 6, we elaborate the implementation of security group in detail by focusing on load balancing and fault tolerance of each group in SMG. We first dwell on the cooperation between MD, active nodes and vSwitch to implement load balancing between active nodes. Fig. 6(a) shows their communication and work sequence.

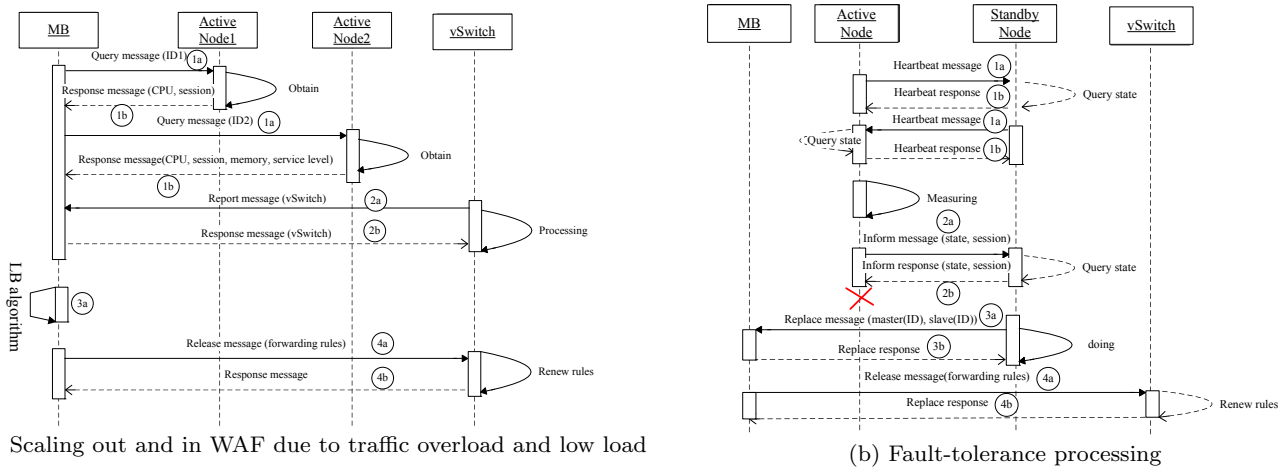
1. MD $\xrightarrow[1a]{ReqMessage_{query}(ID_{Node})}$ active nodes: ID_{Node} denotes the current active node identifier, MD send a query message to all active nodes.
2. Active nodes $\xrightarrow[1b]{ResMessage_{query}(ID_{Node},CPU,memory)}$ MD: All current active nodes respond to **the respective state information** including CPU utilization, memory usage, sessions, etc.
3. vSwitch $\xrightarrow[2a]{ReqMessage_{report}(vSwitch)}$ MD: vSwitch periodically reports the number of packets and flow size through active nodes to MD.
4. MD $\xrightarrow[2b]{ResMessage_{report}(vSwitch)}$ vSwitch: MD responds the report message to vSwitch.
5. LB algorithm: Alg (the number of packets, flow size, CPU, memory, session, etc) (3a). **Note: the current popular algorithms, for example round robin, dynamic server act and dynamic ratio-APM, may be used as the load balancing algorithm. Since the article focuses on security architecture of cloud computing, load balancing algorithm will not be described too much.**
6. MD $\xrightarrow[4a]{ReqMessage_{release}(vSwitch)}$ vSwitch: MD issues forwarding rules to vSwitch according to LB algorithm.
7. vSwitch $\xrightarrow[4b]{ResMessage_{release}}$ MD: vSwitch responds to MD for the release message.

Next, as show in Fig. 6(b), the communication and work sequence between active nodes and standby node will be elaborated to prepare for fault-tolerant on account of any active node failure.

1. Active or standby node $\xrightarrow[1a]{ReqMessage_{heartbeat}(ID_{Node})}$ standby or active node: they probe each other to determine whether the other is alive.
2. Standby or active node $\xrightarrow[1b]{ResMessage_{heartbeat}(ID_{Node})}$ active or standby node: they respond to each other probes.
3. Active node $\xrightarrow[2a]{ReqMessage_{inform}(ID_{Node},state,session)}$ standby node: The renewed information (such as state, session) is replicated from active nodes to standby node in real time.
4. Standby node $\xrightarrow[2b]{ReqMessage_{inform}(ID_{Node})}$ active node: Standby node responds to active nodes for the renewed message.

Finally, as show in Fig. 6(b), Hot Standby switch-over processing will be explained in accordance with an active node failure, and the sequence and communication between MD and standby node will be present to implement fault-tolerance. That is, if any active node fails, the standby node immediately plays the role of that failed active node.

1. Standby node $\xrightarrow[3a]{ReqMessage_{replace}(ID_{active},ID_{standby})}$ MD: if probing that the active node has failed, the standby node sends a replacement message to MD to switch over between the active node and the standby node.
2. MD $\xrightarrow[3b]{ResMessage_{replace}}$ Standby node: MD responds to the standby node for switch-over.



(a) Scaling out and in WAF due to traffic overload and low load

(b) Fault-tolerance processing

Fig. 5: Load balancing and fault-tolerance processing in each group

3. MD $\xrightarrow[4a]{ReqMessage_{release}(vSwitch)}$ vSwitch: MD renews forwarding rules to vSwitch after switch-over. The standby node becomes the active node responsible for inspecting and filtering the received traffic.
4. vSwitch $\xrightarrow[4b]{ResMessage_{release}}$ MD: vSwitch responds to MD for the renewed message.

To summarize, MD guides security groups to put into effect security inspection for traffic, thereby ensuring that traffic arriving at service domains is secure and trusted, while security groups are the specific implementer of security inspection. They complement each other to achieve security protection of cloud computing.

Table 2: The list of open source security middleboxes

Middlebox Name	Open Source Software
FW	IPFire [7]
WAF	ModSecurity [13]
SSL/VPN	OpenSSL [20]
AS	PacketFence [35]

4 Evaluation

In this section, we evaluate NetSccCC with the following goals:

- evaluate *NetSecCC*'s ability to prevent external and internal malicious traffic from attacking cloud users' services (§4.1).
- evaluate *NetSecCC*'s ability to provide dynamic scalability to complex real world middleboxes, and measure the gain in resource utilization when scaling in a deployment with *NetSecCC* (§4.2).
- evaluate *NetSecCC*'s fault-tolerance ability compared with three different fault-tolerance cases when one or more of the active nodes fail (§4.3).
- quantify system overhead with *NetSecCC* compared with the case without security protection in cloud computing (§4.4).

Experimental environment Cloud platform was conducted on a Dell Server with 8 core, 3.42 GHz Intel CPU, 16GB memory. The XEN hypervisor version is 3.4.2, the dom0 system is fedora 16 with kernel version 2.6.31. We used a 64bit fedora Linux with kernel version 2.6.27 as guest OS, vSwitch bandwidth is 1 Gigabit Ethernet; *NetSecCC* uses open source security middleboxes as shown in Table 2.

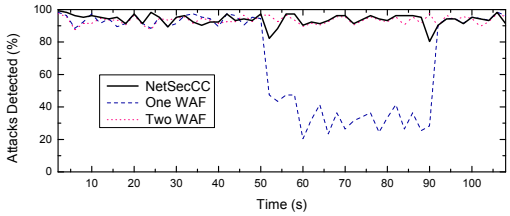
4.1 Protection

Table 3 shows *NetSecCC*'s ability to prevent external and internal malicious traffic from attacking Web server in service domains. Our experimental environment is that a Web server [26] was installed on a standalone VM in service domains, we have simulated different types of malicious external traffic from Internet and internal traffic from other VMs generated by attack tools (e.g., SQL Inject Me, HackBar), as shown in Table 3, to attack the Web server. Three scenarios are compared to protect against these malicious attacks: *NetSecCC*, *Wu et al.* that control the inter-communication among virtual machines with higher security by the embedded firewall in virtualized environment, and *the traditional architecture* that places network security middleboxes at front-end of cloud computing to protect their network security, as shown in Fig. 1.

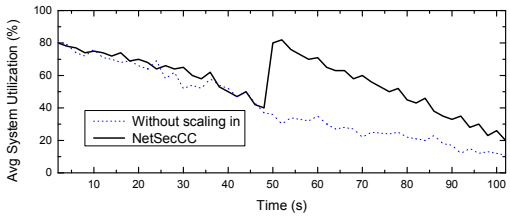
Table 3: Comparison results of the three protection: Wu *et al.* just prevent malicious internal traffic between VMs; the traditional architecture is just able to prevent malicious external traffic; While *NetSecCC* can address the attacks from external and internal traffic.

Vulnerability	Attack Tool	Wu *et al.* [40]		Traditional Architecture		*NetSecCC*	
		Ext.	Int.	Ext.	Int.	Ext.	Int.
Cross-site scripting	ZAP [25]	×	√	√	×	√	√
Cross-site request forgery	ZAP	×	√	√	×	√	√
Input validation	Nikto/Wikto [19]	×	√	√	×	√	√
SQL injection	SQL Inject Me [22]	×	√	√	×	√	√
Information leak	Tamper Data [23]	×	√	√	×	√	√
Authentication issues	HackBar [17]	×	√	√	×	√	√
Path traversal	HackBar	×	√	√	×	√	√

The results of experiments are shown in Table 3. The solution provided by Wu *et al.* is just able to protect inter-communication between virtual machines in the same platform. However, it can not protect against external malicious attacks. The traditional architecture is now a popular way to provide protection for external network security of cloud computing, but it can not protect internal communication between virtual machines, thus resulting in the serious problem that the malicious virtual machine can attack and control the other virtual machines in the same platform to destroy the entire cloud system. While *NetSecCC*, in the contrary, protects against not only malicious attacks from external traffic, but also attacks from internal traffic to ensure network security of cloud users' services in cloud computing, thus overcoming the shortcomings of Wu *et al.* and the traditional architecture.



(a)



(b) System utilization when scaling FW in for low load

Fig. 6: Scaling security groups out and in to test *NetSecCC*'s scalability according to traffic overload and low load

4.2 Scalability

Fig. 7(a) shows *NetSecCC*'s ability to dynamically scale WAF out and in during a load burst. Our experimental environment is that a Web server is installed on a standalone VM in service domains, 30 clients in the form of a continuous sequence of POST requests access to the Web server, the requests contain SQL injection and cross-site scripting attacks, and each client generates 80 requests/second. We inject a load burst 50 seconds into the experiment by introducing an additional 30 clients, and the load burst lasts 40s. Three scenarios are compared: a single WAF instance that handles the entire load burst, a pair of WAFs that share load (flows are assigned to each WAF in a round-robin fashion) and *NetSecCC*. *NetSecCC* scenario begins with a single WAF. When overloaded, *NetSecCC* creates a new WAF to split Web traffic.

As shown in Fig. 7(a), until the load burst at t = 50s, all the three scenarios have a 100% detection rate. During the load burst, the performance of the single WAF degrades drastically because packets are dropped and attacks are missed. The two WAFs do not experience any degradation as they have enough capacity and the load is well balanced between the two WAFs. While *NetSecCC* creates a new WAF to split Web traffic according to the load burst at t = 50s, the following problem is caused: packets are dropped and attacks are missed. However, the detection rate quickly rises because the two WAFs have enough capacity for the load burst. After the load burst (t = 90s), *NetSecCC* detects a drop in load due to the destroying of one WAF. *NetSecCC* therefore enables WAF to handle the load burst without wasting resources by running two WAFs throughout the entire experiment.

Fig. 7(b) shows system utilization between *NetSecCC* and a pair of FWs that share load in a round-robin fashion. Our experimental environment is that a UDP

server is installed on a standalone VM in service domains, 100 UDP clients continuously send UDP packets to the UDP server, and each client evenly generates from 8M requests/second to 1M requests/second within 100s in the descending way. Initially, *NetSecCC* has two FWs to share UDP traffic. When traffic declines, in the first 50s, the system utilization of *NetSecCC* is the same as that of this pair of FWs, decreased from 80% to 50%. However, *NetSecCC* utilization burst reaches 80% at t = 50s, this is mainly because that *NetSecCC* is configured with a scale-in policy that is triggered once one FW load falls below 50. That is, at low load, one FW is destroyed, and all traffic is forwarded to the other FW. That is, at this moment, only one FW in *NetSecCC* is responsible for filtering and inspecting all traffic, while a pair of FWs remain two FWs, thereby improving system utilization. After 50s, *NetSecCC*'s system utilization decreases from 80% to 20% with gradually decreasing traffic, while the system utilization of this pair of FWs decreases to 10%.

4.3 Fault Tolerance

One dynamic scenario is considered in Fig. 4. When node3 fails, it is needed to recover the normal running network, and what we are interested in is the recovery time [37]. Three scenarios are compared: Remus [1], RP [29] and *NetSecCC*. The following shows three different fault-tolerance configurations:

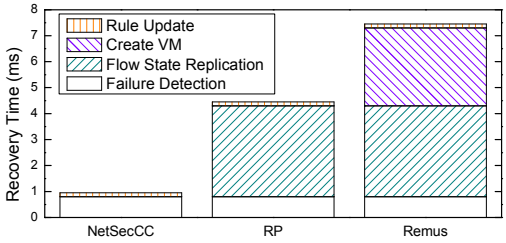


Fig. 7: Recovery time in the case of a virtual middlebox failure

Remus uses a VM-level fault-tolerance technique that provides transparent failure recovery. The flow state of node3 is constantly replicated by Remus to node4 in the same group, and node3's passive backup copy created by Remus is denoted as node3-B. When node3 failure is detected, Remus creates node3-B, and constantly replicates the flow state of node3 from node4 to node3-B, and vSwitch forwards the flows accordingly to node3-B.

RP uses a fine-grained flow-level fault-tolerance technique. When node3 fails, the flow state of node3 is constantly replicated by RP to node1 and node2 in the same group, and the SDN controller immediately activates the relevant flow states in node1 and node2 and vSwitch forwards the flows accordingly.

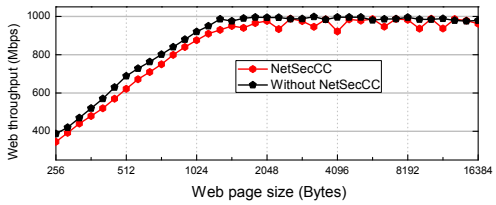
Fig .8 shows recovery time of each fault-tolerance strategy, including failure detection time, flow state replication time, creation VM time and rule update time. In regard of Remus, the overhead is prohibitively high, and the main overhead comes from the flow state replication and the creation VM. Where the time overhead of the creation VM is the time to establish VM and middlebox, the root cause of the performance degradation during the flow state replication arises from suspending and resuming the entire VM by a VM-level fault-tolerance technique. In regard of PR, the recovery time is very close to approximate 5ms, its main overhead also comes from the flow state replication. This is due to the fact that in order to failover a set of flows of the failed node to the active nodes without disrupting end-to-end connectivity, when finding node failure, PR takes a long time to replicates relevant flow states. In regard of *NetSecCC*, during the operation, the new and renewing flow state of the active nodes has being synchronized to the standby node in a timely manner, so *NetSecCC*'s fault tolerance overhead just takes failure detection time and rule update time, thus reflecting a higher recovery efficiency.

4.4 Performance Overhead

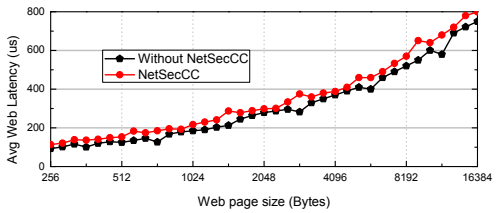
To evaluate *NetSecCC*'s system performance overhead, throughput and latency that are important indicators of system performance are used as evaluation criteria. Although this way without *NetSecCC* is higher efficient than one with *NetSecCC*, if no protective measures are taken to protect cloud computing security, it may lead to incalculable losses. Therefore, it is necessary to protect network security of cloud computing to defend various attacks from the network. Even if *NetSecCC* is selected to protect cloud computing security, it is also necessary to consider whether its performance overhead can be accepted. IXIA [18] is used as a performance testing tool to evaluate *NetSecCC*'s performance overhead, comparing both with and without network security in cloud computing. Next, Two experiments are used to evaluate the performance impact with *NetSecCC*.

For Web page access as our first experiment, IXIA is used both as a customer and as a server to test with and without *NetSecCC*. The results of this experiment show in Fig. 9 that *NetSecCC* has little impact on system performance, compared with the case without *NetSecCC*, *NetSecCC* imposes 9.3% of average latency

overhead (ranging from 6.4% to 13.9%) and 0.4% of average throughput drop(ranging from 0 to 3.7%). There are two main reasons. *First*, since SIC (FW-WAF) of web page access is composed of FW group and WAF group, Web traffic must go through FW group and WAF group to be inspected and filtered before being forwarded to the Web server in service domains. In this process, traffic is required to match hundreds of filtering rules in FW and thousands of signatures in WAF. This will take some time, resulting in the increased latency and the decreased throughput. *Second*, since FW group and WAF group share the same hardware resources (e.g., CPU, memory) in the same virtual platform, context switches between FW group and WAF group cause cache invalidations, which is very expensive. In the case without *NetSecCC*, Web traffic directly accesses to the Web server to avoid inspection in terms of system overhead. Therefore, compared with the cases without *NetSecCC*, latency becomes longer with *NetSecCC*, throughput is suffered from the impact of latency, but overall system performance with *NetSecCC* is within the acceptable range($\leq$ 9.3%).



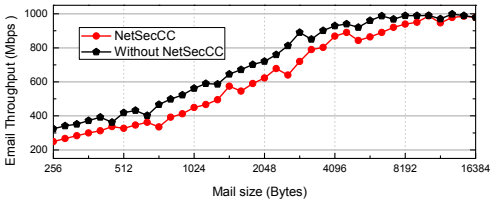
(a) Throughput comparison of web page access



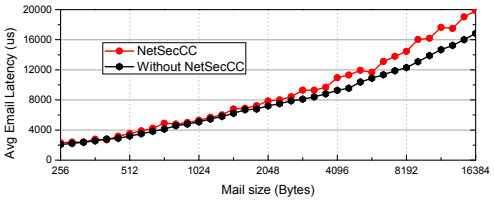
(b) Latency comparison of web page access

Fig. 8: Performance comparison results of cases with *NetSecCC* and without *NetSecCC* by Web page access

For Email access as our second experiment, IXIA is used to test performance overhead with *NetSecCC*. The results of this experiment show in Fig. 10 that even encrypted emails with *NetSecCC* are just slightly affected. Compared with the case without *NetSecCC*, specific data with *NetSecCC* on the performance overhead is shown below: the average cost of latency is 11.1% (ranging from 9.2% to 13.7%), and the average cost of throughput is 5% (ranging from 0 to 11.1%). For security services, such a performance overhead ($\leq$ 11.1%) is perfectly acceptable.



(a) Throughput comparison of mail access



(b) Latency comparison of encrypted mail access

Fig. 9: **Performance comparison results of cases with *NetSecCC* and without *NetSecCC* by mail access**

In summary, by the comparison of two cases both with and without *NetSecCC* in cloud computing, *NetSecCC* is able to provide adequate network security protection for cloud computing, but not at the cost of sacrificing the high price of system performance. The two experiments have further indicated that *NetSecCC* scheme can provide efficient comprehensive network protection for cloud computing.

5 Conclusion

In this paper, we introduce a novel solution *NetSecCC* to ensure network security in cloud computing. In particular, this new solution is carefully designed to provide protection against both external and internal attacks, to provide flexible scalability and to achieve high and effective capability of fault-tolerance. Our extensive experimental results validates all these characteristics, and thus *NetSecCC* address the all three major defects known in a traditional solution. It also provides a more comprehensive protection for cloud computing, and opens a new door in network security research.

Acknowledgements This work is partially supported by NSFC Grant No. 61450110085, the Open Research Project of the State Key Laboratory of Industrial Control Technology, Zhejiang University, China (No. ICT1407), JSPS KAKENHI Grant Number 25880002, 26730056 and JSPS A3 Foresight Program.

References

1. Brendan Cully, Geoffrey Lefebvre, Dutch Meyer, Mike Feeley, Norm Hutchinson, and Andrew Warfield. Remus: High availability via asynchronous virtual machine replication. In *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, pages 161–174. San Francisco, 2008.
2. Mianxiong Dong, Kaoru Ota, He Li, Suguo Du, Haojin Zhu, and Song Guo. Rendezvous: towards fast event detecting in wireless sensor and actor networks. *Computing*, pages 1–16, 2013.
3. Mianxiong Dong, Kaoru Ota, Man Lin, Zunyi Tang, Suguo Du, and Haojin Zhu. Uav-assisted data gathering in wireless sensor networks. *The Journal of Supercomputing*, pages 1–14, 2014.
4. Adrian J Duncan, Sadie Creese, and Michael Goldsmith. Insider attacks in cloud computing. In *Trust, Security and Privacy in Computing and Communications (TrustCom), 2012 IEEE 11th International Conference on*, pages 857–862. IEEE, 2012.
5. BIG-IP Configuring High Availability F5 Networks Inc. `http://support.f5.com/kb/enus/products/big-ip_ltm/manuals/product/tmos_management_guide_10_0_0/tmos_high_avail.html`.
6. Diogo AB Fernandes, Liliana FB Soares, João V Gomes, Mário M Freire, and Pedro RM Inácio. Security issues in cloud environments: a survey. *International Journal of Information Security*, pages 1–58, 2013.
7. IPFire. `http://www.ipfire.org/`.
8. Dilip Joseph and Ion Stoica. Modeling middleboxes. *Network, IEEE*, 22(5):20–25, 2008.
9. Hongwei Li, Xiaodong Lin, Haomiao Yang, X Liang, R Lu, and X Shen. Eppdr: An efficient privacy-preserving demand response scheme with adaptive key evolution in smart grid. *IEEE Transactions on Parallel and Distributed Systems*, page 1, 2013.
10. Hongwei Li, Rongxing Lu, Liang Zhou, Bo Yang, and X Shen. An efficient merkle-tree-based authentication scheme for smart grid. *Systems Journal, IEEE*, pages 655–663, 2013.
11. Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, 2008.
12. Peter Mell and Timothy Grance. The nist definition of cloud computing (draft). *NIST special publication*, 800(145):7, 2011.
13. ModSecurity. `http://www.modsecurity.org/`.
14. Ali Mohammed, Sachin Sama, and Majeed Mohammed. *Enhancing Network Security in Linux Environment*. PhD thesis, Halmstad University, 2012.
15. NVD. `http://nvd.nist.gov/`.
16. AWS [Online]. `http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-network-security.html`.
17. HackBar [Online]. `https://addons.mozilla.org/en-us/firefox/addon/hackbar/`.
18. IXIA [Online]. `http://www.ixiacom.com/`.
19. Nikto [Online]. `http://www.netsecurity.com`.
20. OpenSSL [Online]. `http://www.openssl.org/`.
21. SecaaS [Online]. `https://cloudsecurityalliance.org/research/secaas/`.
22. SQL Inject [Online]. `https://addons.mozilla.org/en-US/firefox/addon/sql-inject-me/`.
23. Tamper Data [Online]. `https://addons.mozilla.org/en-us/firefox/addon/tamper-data/`.
24. VMware [Online]. `http://www.vmware.com/`.
25. Zap [Online]. `https://code.google.com/p/zaproxy/`.
26. Apache HTTP Server Project[Online]. `http://httpd.apache.org/`.
27. McAfee SaaS Email Protection and Web Protection. `http://www.mcafee.com/us/products/security-as-a-service/index.aspx`.
28. Zafar Ayyub Qazi, Cheng-Chun Tu, Luis Chiang, Rui Miao, Vyas Sekar, and Minlan Yu. Simple-fying middlebox policy enforcement using sdn. In *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM*, pages 27–38. ACM, 2013.
29. Shriram Rajagopalan, Dan Williams, and Hani Jamjoom. Pico replication: a high availability framework for middleboxes. In *Proceedings of the 4th annual Symposium on Cloud Computing*, page 1. ACM, 2013.
30. Shriram Rajagopalan, Dan Williams, Hani Jamjoom, and Andrew Warfield. Split/merge: System support for elastic execution in virtual middleboxes. In *Proceedings of the 10th USENIX conference on Networked Systems Design and Implementation*, pages 227–240. USENIX Association, 2013.
31. Khaled SALAH, Jose M ALCARAZ CALERO, Sherali ZEADALLY, Sameera AL-MULLA, and Mohammed ALZAABI. Using cloud computing to implement a security overlay network. *IEEE security & privacy*, 11(1):44–53, 2013.
32. Vyas Sekar, Norbert Egi, Sylvia Ratnasamy, Michael K Reiter, and Guangyu Shi. Design and implementation of a consolidated middlebox architecture. In *Proc. NSDI*, 2012.
33. Vyas Sekar, Sylvia Ratnasamy, Michael K Reiter, Norbert Egi, and Guangyu Shi. The middlebox manifesto: enabling innovation in middlebox deployment. In *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*, page 21. ACM, 2011.
34. Justine Sherry, Shaddi Hasan, Colin Scott, Arvind Krishnamurthy, Sylvia Ratnasamy, and Vyas Sekar. Making middleboxes someone else's problem: Network processing as a cloud service. *ACM SIGCOMM Computer Communication Review*, 42(4):13–24, 2012.
35. SpamAssassin. `http://spamassassin.apache.org/`.
36. S Subashini and V Kavitha. A survey on security issues in service delivery models of cloud computing. *Journal of Network and Computer Applications*, 34(1):1–11, 2011.
37. Nikolay Topilski, Jeannie R Albrecht, and Amin Vahdat. Improving scalability and fault tolerance in an application management infrastructure. In *LASCO*, 2008.
38. High Availability Reference Guide Vyatta Inc. `http://www.vyatta.com/downloads/documentation/VC6.5/Vyatta-HA_6.5R1_v01.pdf`.
39. Zhi Wang, Chiachih Wu, Michael Grace, and Xuxian Jiang. Isolating commodity hosted hypervisors with hyperlock. In *Proceedings of the 7th ACM european conference on Computer Systems*, EuroSys '12, pages 127–140, New York, NY, USA, 2012. ACM.
40. Hanqian Wu, Yi Ding, Chuck Winer, and Li Yao. Network security for virtual machine in cloud computing. In *Computer Sciences and Convergence Information Technology (ICCIT), 2010 5th International Conference on*, pages 18–21. IEEE, 2010.
41. Yue Wu, Joseph P Noonan, and S Agaian. Binary data encryption using the sudoku block cipher. In *Systems Man and Cybernetics (SMC), 2010 IEEE International Conference on*, pages 3915–3921. IEEE, 2010.
42. Dimitrios Zissis and Dimitrios Lekkas. Addressing cloud computing security issues. *Future Generation Computer Systems*, 28(3):583–592, 2012.