Volume 34, Issue 2, 15 February 2011 ISSN 0140-3664
ELSEVIER
computer communications
Available online at www.sciencedirect.com
ScienceDirect
Special Issue:
Open Network Service Technologies and Applications
Guest Editors: Chung-Ming Huang
Isaac Woungang
Ming-Chiao Chen

Computer Communications 34 (2011) 200–208

Contents lists available at ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

OpenGate: Towards an open network services gateway

Yaxuan Qi [a,*], Fei He [a], Xiang Wang [b], Xinming Chen [a], Yibo Xue [a,c], Jun Li [a,c]

[a] *Department of Automation, Tsinghua University, Beijing, China*
[b] *School of Software Engineering, University of Science and Technology of China, Hefei, China*
[c] *Tsinghua National Lab for Information Science and Technology, Beijing, China*

A R T I C L E I N F O

Article history:
Available online 4 April 2010

Keywords:
Open technology
Gateway
Security

A B S T R A C T

In this paper, we propose an extensible open network services gateway (OpenGate) for high-performance network processing at the edge of high-speed networks. The OpenGate system embraces recent advances of open network technologies: the performance is guaranteed by using open-standard ATCA platforms; and the extensibility is achieved by employing parallelized open source software. As an application example of OpenGate, a high-performance security gateway, OpenGate-SG, was developed using existing ATCA platforms and open source software. This system provides multiple security services, including stateful firewall, intrusion prevention and anti-virus. Experimental results show that, OpenGate-SG can achieve up to 200 Gbps stateful firewall throughput with 8 Gbps intrusion prevention and anti-virus, which is competitive to the performance of today's high-end security products. OpenGate-SG has also been tested as a security gateway for a university campus network with more than 1000 students.

1. Introduction

With the increase of Internet traffic and the multitude of network services, the need for both performance and flexibility has become the key challenge for novel network services gateways (NSG). The rapid growth of data centers, the IP convergence of telecom services, and the security requirement of enterprise networks together make the great demand for high-performance NSGs to meet the following challenges:

(1) *Performance*: New generation NSGs are commonly integrated with multiple 1 Gbps/10 Gbps interfaces. The emergence of 100 Gbps Ethernet interface will bring about further challenges for high-throughput packet processing [1,2]. In addition, a 10 Gbps NSG is typically required to support millions of concurrent connections, as well as tens of thousands connection creation rate.
(2) *Extensibility*: NSGs require the flexibility to accommodate interoperability, quality of service, dynamic algorithms, network security, and other services for multiple protocols. Due to the diversity of network services and the instability of network topology, NSGs must not only support all current requirements, but also have the extensibility to support unforeseen future requirements.

Most of today's equipment vendors design their high-end products based on proprietary hardware. Such a closed-but-fast model gives them complete control over system quality, but has become a barrier to innovation [3]. Recently, both the academic and industrial worlds initiated research programs on open network technology, which encourages research innovation and allows third parties to develop software extensions for proprietary hardware. The inherently collaborative and distributive nature of open network technology will benefit the NSG design from multiple aspects: open source software can help to quickly deploy novel services; open system standards can consolidate multiple vendors' functions into one physical box; open network architecture can reduce redundant network devices and thus save unnecessary cost. In this paper, we propose an open network services gateway (OpenGate) by exploiting recent advances of open network technologies. Main contributions of this paper are:

(1) *OpenGate design*: OpenGate is designed to meet the goals of an extensible NSG. It follows the open standard of Advanced Telecom Computing Architecture (ATCA) to guarantee the system performance and scalability, and provides multiple programming APIs to support a variety of open source software.
(2) *Example application*: As an application example, a high-performance security gateway, OpenGate-SG, was developed based on the OpenGate architecture. OpenGate-SG provides multiple network security services, including stateful firewall, intrusion prevention and anti-virus.

* Corresponding author.
E-mail addresses: yaxuan@tsinghua.edu.cn (Y. Qi), hefei06@mails.tsinghua.edu.cn (F. He), kojiroh@mail.ustc.edu.cn (X. Wang), chenxm05@mails.tsinghua.edu.cn (X. Chen), yiboxue@tsinghua.edu.cn (Y. Xue), junl@tsinghua.edu.cn (J. Li).

0140-3664/$ - see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.comcom.2010.03.035

(3) *Performance evaluation*: OpenGate-SG is tested using both traffic generator and real-life traffic traces. Experimental results show that, our system can achieve up to 200 Gbps stateful firewall throughput with 8 Gbps intrusion prevention and anti-virus. OpenGate-SG has also been tested as a security gateway for a university campus network with more than 1000 students.

The rest of the paper is organized as follows: in Section 2, we discuss the open network technologies related to our research; in Section 3, we propose the system design of OpenGate; OpenGate-SG is described in Section 4 as an application example, and its performance is evaluated in Section 5; in Section 6, we state our conclusion.

2. Background

Open technologies have different scopes in network and system research, such as *open source software*, *open system standards* and *open network architecture*. Although they focus on different research topics, their inherent collaborative and distributive natures all contribute to the success of open network technologies. In this section, we will discuss the open network technologies related to our OpenGate design.

2.1. Open network software

The open source community brainstorms, develops, enhances and maintains the open source software by collaboration and free sharing. A big success has been achieved by open source software in commoditized platforms. However, adapting open source software to a specialized hardware environment such as multi-core network processors meets the following challenges:

(1) *Parallel processing:* Because the performance of single-threaded open source software is commonly not scalable on multi-core platforms, OpenGate needs to provide an effective way to support parallel processing of open source based network services.
(2) *Load balancing:* To maximize the overall performance on multi-core platforms, the workload should be appropriately distributed among multiple processing threads. Load balancing is also required to reduce inter-dependencies of multiple processing threads for scalable parallelism.

In addition, multi-threaded programming is considered difficult even with new tools and enhanced compilers. Fortunately, network traffics have inherent independency at certain granularities, which can be exploited for scalable parallel processing [4]. In our design, OpenGate exploits this independency and provides a parallel programming model to achieve scalable performance of open source software.

2.2. Open network platform

Based on the standardized platforms and multi-core processors, system-level open network technologies are studied in recent years [5]. The driving forces of this research are from:

(1) *Service providers:* They can deploy novel services faster and cheaper by consolidating multiple vendors' functions into a single standardized platform.
(2) *Equipment vendors:* Open system creates a software ecosystem that helps them gain new features without non-recurring engineering (NRE) costs.
(3) *Network researchers:* Open system is an ideal platform for innovation. Novel algorithms and protocols can be easily deployed and evaluated on these platforms.

With the emergence of ATCA technologies, the goal of system-level standardization for open network platforms can be achieved. ATCA is an industry standard to create a new blade and chassis optimized for network communications [6]. Based on the ATCA standard, server blades, switch cards and storage devices from different manufacturers can use the same electrical interface to work together within in the same ATCA chassis. In this paper, the proposed OpenGate NSG fully exploits the advantages of the ATCA platform to achieve both performance and scalability. More detail about the ATCA technology and open modular computing specifications can be found at the official PICMG website [6].

2.3. Open network architecture

With the evolving of enterprise networks and datacenters, traditional interconnect technology has been inefficient to adapt to the rapid changes of network traffics and topologies. Open network architectures are proposed to improve the manageability of existing network.

McKeown et al. proposed an open-protocol network switching architecture named OpenFlow [9]. OpenFlow allows researchers to run experiments on heterogeneous switches in a uniform way at line-rate, while vendors do not need to expose the internal hardware of their switches. As an application example, Ethane [10] couples OpenFlow switches with a centralized controller, trying to take control of the enterprise network. An Ethane switch sends the first packet of a flow to the controller. This controller determines the path of the flow and installs the forwarding entries at OpenFlow switches. Off-path middle-boxes can be imposed by including them in the source routes. NOX extends the ideas of Ethane and scales this centralized paradigm to very large systems [11]. In comparison, Joseph et al. proposed a policy-aware switching layer (PLayer) for datacenter networks [12]. According to their design, unmodified middle-boxes can be placed on the network path by plugging them into inter-connected policy-aware switches. Different from Ethane, policy lookup is done within the *pswitches* (fast-path) rather than the centralized controller (slow-path). The authors claimed that such a distributed approach can make it better suited for scaling to a large number of data flows and more robust against network churn and attacks.

Because OpenGate is an inter-connected distributed system working as a small cluster network, we borrow ideas from open network architectures, such as management centralization and policy-path separation, to enhance the flexibility and manageability of our system.

3. OpenGate overview

OpenGate embraces multiple open network technologies to meet the performance and flexibility requirements. Its software architecture supports existing open source software, its system architecture employs the ATCA open network technology, and its data-plane network takes advantages of open network architecture. According to Fig. 1, OpenGate contains the following processing modules:

(1) *Network processing module (NPM)*: NPM provides the external network interfaces and packet forwarding engine for OpenGate. NPM is responsible for front-end network processing, including packet forwarding, flow classification, stateful inspection and traffic management. NPM is also

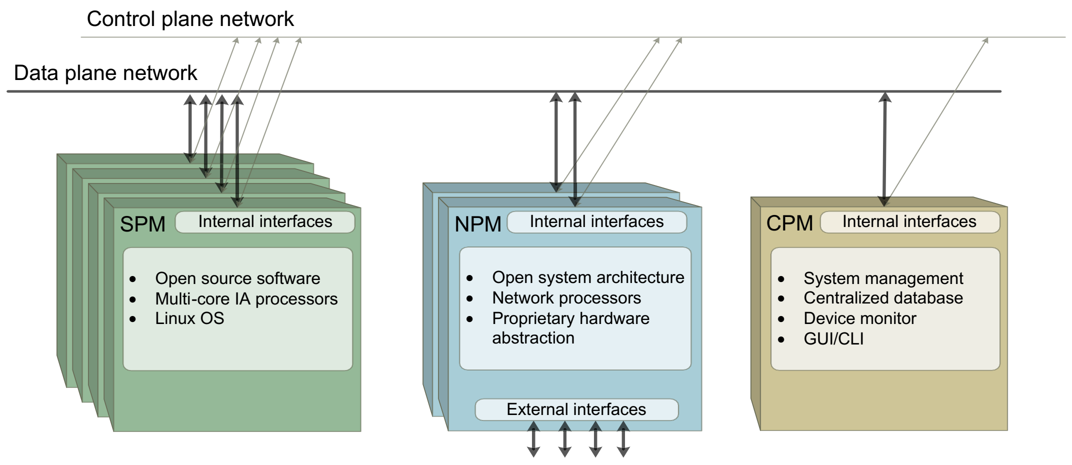

Fig. 1. OpenGate system overview.

responsible for load-balancing among back-end services processing blades. Due to the high-throughput requirement, NPM is implemented on network processor (NP) based blades.

(2) *Service processing module (SPM)*: SPM provides flexible application-level processing capability to meet the requirement for the ever-evolving network services. With the help of front-end load-balancing, the overall performance of OpenGate can be increased by inserting new SPM blades in the ATCA chassis or plugging SPM servers into the ATCA switch blades. SPM is implemented on the Intel Architecture (IA) platform to gain maximum programming flexibility.

(3) *Control processing module (CPM)*: Because OpenGate is a cluster system conducting distributed processing, a system-wide control processing module is necessary for centralized management. CPM provides an extensible management model for system initialization, monitoring and update. CPM also provides the GUI/CLI interfaces for OpenGate administrators.

3.1. OpenGate network processing module

NPM is not only responsible for packet forwarding, but also required to maintain connection states to support stateful inspection and connection-oriented load-balancing. In addition, because network I/O interfaces reside in the NPM, the overall traffic of OpenGate should be managed by NPM. Our design of OpenGate NPM contains the following building blocks:

(1) *Packet receive/transmit blocks:* These blocks are responsible for receiving and transmitting packets via multiple network interfaces in the NPM. These building blocks are commonly implemented by hardware co-processors to avoid excessive memory copies and frame reassembly.

(2) *Packet processing blocks:* Packet processing in NPM is implemented at different granularities. At *packet-level*, incoming packets are handled by per-packet processing blocks, such as protocol parsing and header validation. When session (bi-direction TCP/ICMP/pseudo-UDP connection) is established, packets are processed at *session-level*, i.e. packets with the same header (for specific fields, such as 5-tuple fields) are processed in the same way. In addition, for traffic management tasks, such as traffic shaping and rate limiting, packets are processed at *flow-level*, i.e. all packets match the same policy or belong to the same application type are treated as a flow and take the same traffic management actions. Packet processing blocks are running at core-affinitive threads for high-throughput fast-path processing.

(3) *Local management blocks:* Local management blocks are responsible for NPM resource management, blade initialization, policy update and system-wide communication. Local management blocks are running on a slow-path processor with Linux OS support.

As the front-end of a network system, the packet processing performance of NPM is critical. Linux-based software programming model for L2–L4 processing cannot guarantee the line-rate forwarding performance on NPM. Therefore, OpenGate runs an optimized network stack (based on Linux Netfilter) on NPM and provides a series of NPM APIs for developers to add novel NPM functions. For *packet-level* processing, OpenGate provides task-based APIs: each of the incoming packets, system events, and control messages is defined as a task; each task is associated with a task-descriptor, which contains the metadata of the task, the memory address to access the task, and the processing state of the task. To add a new processing function for a certain type of packets or system events, OpenGate developers only need to register a new task type and provide with its processing callbacks to the main loop of OpenGate network stack using NPM APIs. For *session-level* and *flow-level* processing, OpenGate network stack uses an extended session table to store not only the packet header and connection states of the session, but also a service chain to specify the packet processing sequence. A service chain contains two parts: for NPM local processing, it provides a function vector (a pointer list of functions with the same types of parameters and return values) to specify the building blocks required by the session; for remote processing on back-end services blades, it uses a blade-identifier list to specify which blade to forward to for certain types of service. Further, to support efficient packet switching among NPMs and SPMs, a next-hop identifier is encoded in the packet header for data-plane transmission. Back-end blades can thus determine the next-hop blade by decoding the identifier. To add new functionalities at session/flow-level, OpenGate developers can use OpenGate APIs to register new service chain and provide corresponding processing functions to the main loop of the OpenGate network stack.

3.2. OpenGate service processing module

The main objective of SPM is to provide various types network services. To guarantee the efficiency for new service deployment and the flexibility for service update, SPM contains the following building blocks:

(1) *Multi-threaded service blocks*: Service blocks are running at Linux user-space as independent (lock-free) threads. A thread-isolator guarantees the lock-free parallel processing mode by separating their input traffic flows. Incoming packets are dispatched to multiple service threads according to flow-level load-balancing strategies [13].
(2) *Fast-path routing blocks*: These blocks are implemented as kernel modules to support fast-path policy-based packet forwarding [12].
(3) *Local management blocks*: These blocks are responsible for local management such as SPM configuration and communication with CPM.

Although IA-based platforms has better programmability than network processor platforms, the code efficiency for multi-core processing, as well as the interaction with front-end NPM, make it still challenging to provide an efficient SPM programming model. By exploiting the flow-level parallelism of network processing [4], we propose a simple and flexible programming model for OpenGate SPMs.

At Linux user-space, *thread-isolated processing* is adopted by OpenGate SPMs. A thread-isolator is designed to serve as a traffic dispatcher, as well as a service manager. It gets packets from kernel space via *netfilter* queues, and then dispatches them (at flow-level) to different service threads according to user-defined load-balancing strategies. Because the input traffic has been isolated by the thread-isolator, service threads work independently with each other. To add new service threads, OpenGate developers only need to modify the single-threaded source code to meet the interface of OpenGate SPM thread-isolator. In addition, to avoid unnecessary memory duplication, developers can register memory initialization functions to the thread-isolator for read-only memories shared among multiple service threads. The thread-isolator will allocate new input queues to support new service threads and will also initialize and update shared memory for them.

After processing the incoming packets, SPM needs to forward them back to NPMs. Note that, using the external routing table (which may have millions of routing entries) to forward packets is not a scalable solution on a Linux-based IA platform. Therefore we propose a *policy-based forwarding* scheme for efficient and scalable data-plane forwarding. According to NPM processing, each data-plane packet has an identifier encoded in its packet head. When the packet is received by a SPM, it will be decoded by a kernel module inserted between Ethernet driver and Linux *ip_rev* process. Based on the identifier, the SPM can determine the next-hop NPM, and the packets will be correctly forwarded to the NPM via Linux policy-routing [14]. The kernel module for policy-based forwarding is also reprogrammable to meet different requirements of data-plane packet switching.

3.3. OpenGate control processing module

CPM provides a centralized system management for the OpenGate platform, including device monitoring, system configuration and GUI/CLI. The key challenge for CPM design is how to correctly and effectively handle various kinds of system events and configurations. In our design, a message-based processing model is proposed, which contains:

(1) *Centralized database*: A centralized database is crucial for distributed system. The database is used not only for configuration storage, but also for the consistence of data access.
(2) *Message handlers*: OpenGate CPM provides a message-based programming model. Each registered message corresponds to a series of message handlers. Every handler is responsible for a certain processing, such as database update, GUI/CLI display, and policy setup. As a centralized control model, CPM message handlers send CPM messages to other blades, which will be processed by NPM and SPM daemons for local updates.
(3) *Communication daemons*: Every blade in OpenGate has communication daemons to interact with each other. These daemons receive and parse remote messages, call local message handlers, and report local processing results. In our implementation, all communication messages are sent through control-plane network and the format of the massages are described by XML.

OpenGate also provide programming APIs for CPMs, including database access, message parser/sender, and GUI/CLI interfaces. OpenGate developers can add new control functions by registering new message types to the message parser with its corresponding message handlers. The handlers can then access the centralized database and communicate with NPMs and SPMs by calling corresponding OpenGate CPM APIs.

4. An example application: OpenGate-SG

As an application example, we develop an OpenGate Security Gateway (OpenGate-SG) on the proposed architecture. OpenGate-SG provides multiple network security services, including stateful firewall, intrusion prevention and anti-virus. NPMs and CPMs of OpenGate-SG are implemented in a 12U 14-slot ATCA chassis. SPMs are developed on 1U servers and connected with ATCA chassis through two aggregation switches, one for data-plane transmission and the other for control-plane communication. All incoming packets are first processed by NPMs for *stateful inspection* (SI) as firewalling. Then, according to user-defined policies, packets need *deep inspection* (DI) are sent to SPMs for intrusion prevention and anti-virus. After both SI and DI, legitimate packets are sent out from the NPM external interfaces, while malicious ones are dropped. CPM provides the GUI/CLI interfaces for system monitoring and policy configurations. Fig. 2 is the system overviews of OpenGate-SG and Table 1 shows the component specifications.

4.1. NPM implementation on OpenGate-SG

OpenGate-SG NPMs are implemented on Radisys ATCA-7220 blades. Each of them has dual OCTEION5860 16-MIPS-core network processors [19] and four 10 Gbps SFP+ Ethernet interfaces. According to Fig. 3, an incoming packet is first processed at packet-level for L2–L3 decapsulation and validation. Packet-level processing blocks also check where the packet comes from (to support remote service chain) and whether the packet needs slow-path processing (for exceptional and control packets). All per-packet processing results are stored in the task-descriptor to trigger the session-level and flow-level service chain. If the packet is the first one of a flow, a new session entry is inserted into the session table after the packet passes all session creation processing. Different from previous work by Turner et al. [7], session creation in OpenGate-SG is implemented in the fast-path. This is important because a security device without high connection establishing rate is very vulnerable to deny-of-services (DoS) attacks. Follow-up packets belong to the same session are processed according to the service chain stored in the session table. We use OpenGate NPM APIs to register a series of NPM services in the chain, such as network address translation (NAT), IPSec VPN, and TCP state val-

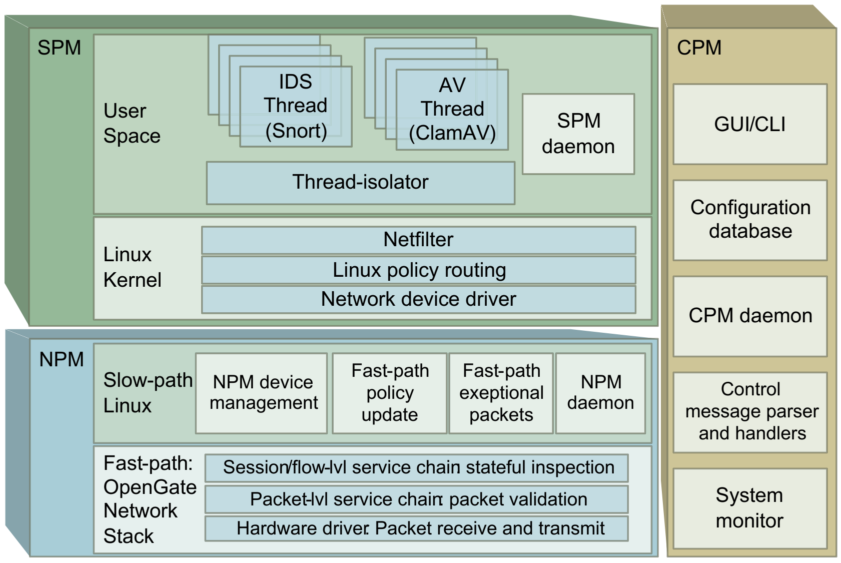

Fig. 2. OpenGate-SG system overview.

Table 1
OpenGate-SG components.

Module	Function	Hardware
Chassis	Blades carrier	Radisys ATCA-6010 [17], 12U, 14Slots Platform
	Switch fabric	1G/10 Gbps base/fabric Ethernet switch
NPM	Stateful inspection	Radisys ATCA-7220 [18], dual 16-MIPS-core
	Traffic management	OCTEON5860 NP, BusyBox Linux
SPM	Snort 2.8 inline IPS [15]	Intel IA blades, dual quad-core XEON E5335
	ClamAV 0.95 anti-virus [16]	Redhat Enterprise Linux 4
CPM	System management	Intel MPCBL-0040 ATCA blade
	GUI/CLI	Debian Linux

idation (see Fig. 3). Remote service chain includes Snort-based intrusion detection and ClamAV-based anti-virus. Packets need deep inspections are sent to a certain SPM according to NPM load-balancing decision. We use a weighted 5-tuple flow-direction-agnostic hash to dynamically balance the traffic among all SPMs: the workload of each SPM, such as CPU/Memory usage, traffic throughput, are periodically reported to the CPM, and the CPM will update the load-balance weights table on NPM if a sensible change of workload distribution is observed. Note that, because this feedback load-balance strategy only takes effect on new flows, packets belong to existed flows will still be forwarded to their original destination SPM, which guarantees the completeness and correctness of TCP reassembly required by deep inspection.

4.2. SPM implementation on OpenGate-SG

SPMs are implemented on 1U Intel IA servers with dual quad-core XEON E5335 processors. Each SPM has 2 gigabits Ethernet interfaces, one for data-plane interconnection and the other for control-plane communication. According to Fig. 4, the processing path of data-plane packets is determined by the policies on NPMs (policy-path separation [8]), so the SPMs must return the packet to its original NPM after processing it. This can be realized by adding less than 100 lines of C codes to the policy-based forwarding module provided by OpenGate (refer to Section 3.2). The modified kernel module compares the source MAC of the packet with all NPM data-plane interfaces' MACs and then set the *fwmark* field in the *sk_buff* of the packet with a next-hop identifier. Each identifier corresponds to a routing table, which contains only one route entry with the default gateway to a specific NPM data-plane interface. Note that, because the number of NPM data-plane interfaces is comparably small (on the order of ten), the efficiency of MAC lookup and routing decision is guaranteed.

After kernel processing, data-plane packets are sent to user-space service threads via netfilter queues (NFQueues). The OpenGate SPM thread-isolator is responsible for receiving data-plane packets from kernel space. Based on the packet header, the thread-isolator calculates a flow-direction-agnostic consistent hash value [4] to determine the flow-ID of the packet. Then according to SPM load-balance strategy, the thread-isolator sends all packets of a certain flow to the same service thread, which runs Snort2.8 [15] and ClamAV0.95 [16] (in inline mode) for intrusion prevention and anti-virus. Because all service threads run independently with each other, the overall memory usage will grow linearly as the number of threads increases. For example, each Snort2.8 thread requires near 2 GB memory if the whole IPS signatures are turned on. To reduce memory usage, we register read-only shared memory (signature lookup data structures) of Snort and ClamAV to the OpenGate thread-isolator, which will initialize and update these memories for all the threads. Experimental results show that, with memory registration, 7 Snort threads only consume 2.4 GB memory, which is far less than the 2 GB * 7 = 14 GB memory usage without memory sharing. This result is in accordance with a previous work proposed by Schuff et al. [20].

4.3. CPM implementation on OpenGate-SG

OpenGate-SG CPM is implemented on Intel MPCBL-0040 ATCA blades. Implementation of control processing functions is shown in Figs. 5 and 6. A *centralized_database* (using MySQL) is used to store system configurations. The *message_parser* parses the XML-based control messages and calls corresponding *message_handlers* we have registered to conduct control processing. After that, the *message_sender* sends the update messages to NPM and SPM daemons for system update. If necessary, the *message_sender* also sends processing results to the GUI/CLI applications to display new configurations. For example, when a new routing entry is added from the GUI interface, the *message_parser* will receive a

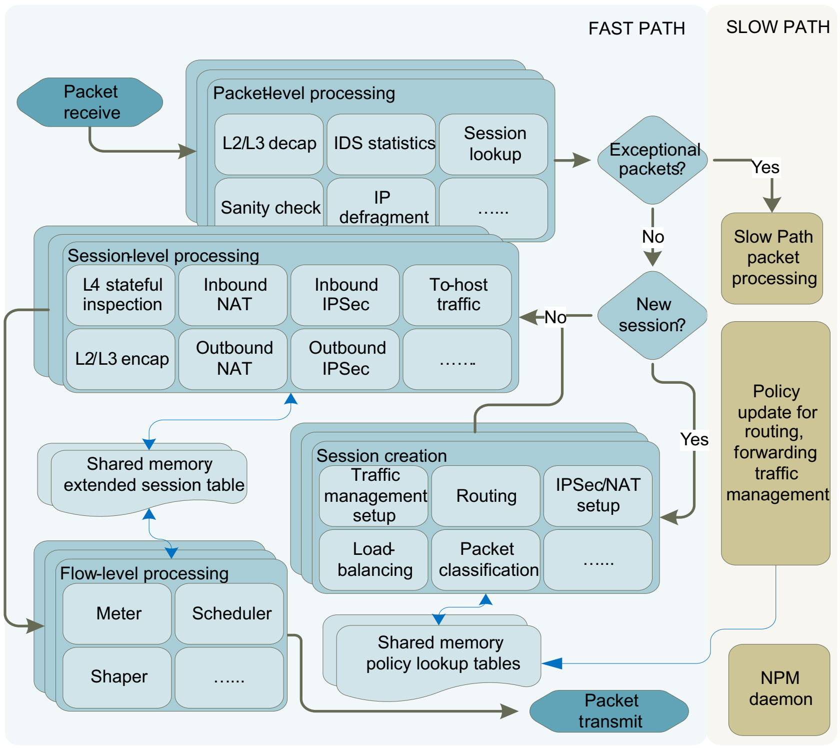

Fig. 3. OpenGate-SG NPM.

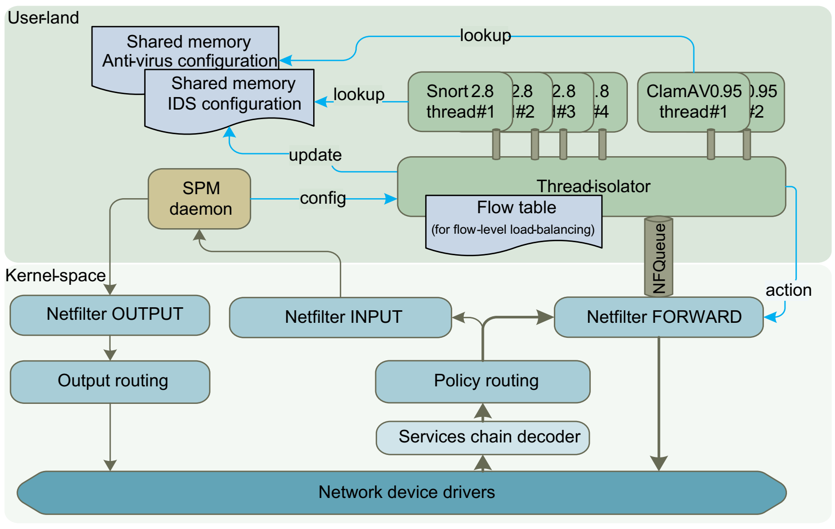

Fig. 4. OpenGate-SG SPM.

control message from the GUI applications. It will then call the *NPM_route_update* handler, which will first update the *configuration_database*, and then send an update message by calling *message_sender* API to all NPMs to update their routing tables. After that, the *message_sender* will invoke the GUI application to display the new routing table.

5. Performance evaluation

5.1. Test overview

To verify the efficiency and extensibility of OpenGate, we have done extensive hardware tests. The performance tests include *stateful inspection* tests for L2–L4 and *deep inspection* tests for L2–L7 processing. We also compare OpenGate-SG with state-of-art products from both academic and industrial worlds to provide a baseline for the performance evaluation. OpenGate-SG has also been tested as a security gateway for a university campus network with more than 1000 students.

5.2. Stateful inspection performance

In this test, the performance of stateful firewall throughput and session creation rate is evaluated. According the OpenGate architecture, this evaluation mainly depends on the NPM performance. Since there are totally 16-MIPS-cores of each OCTEON5860 processor on the NPM, and we use 15 of them to run fast-path code, leaving one for slow-path processing (where a busy-box Linux is

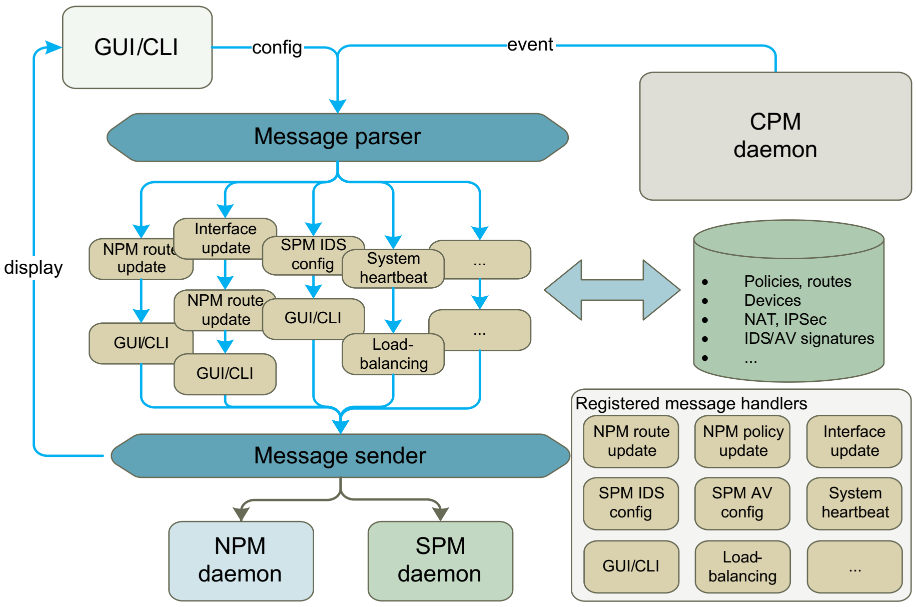

Fig. 5. OpenGate-SG CPM.

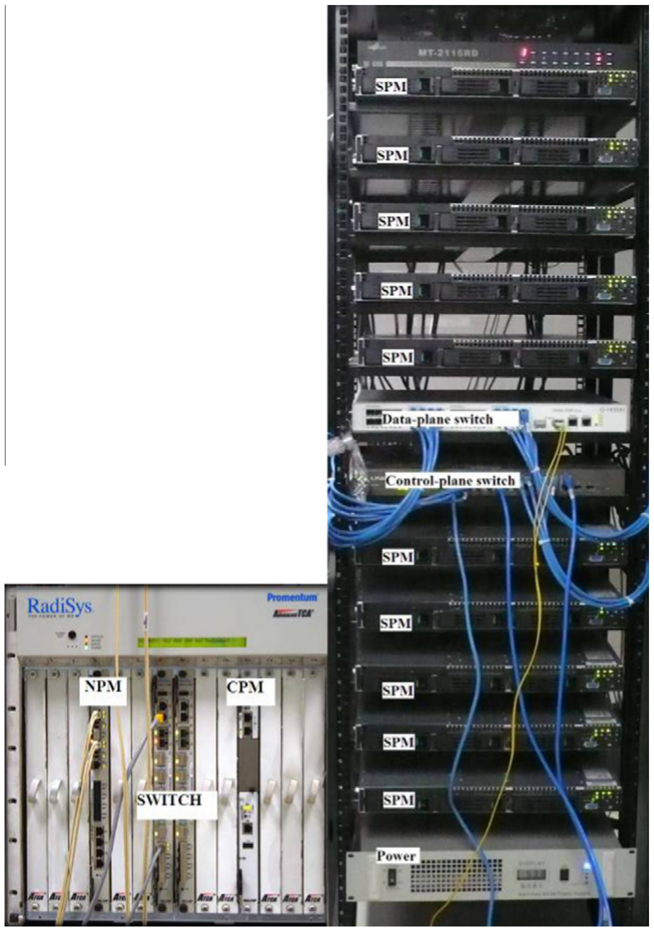

Fig. 6. Prototype of OpenGate-SG.

running for initialization and management). Because the NPM fast-path code is run-to-completion (RTC) and each core uses simple executive mode, i.e. all the fast-path cores can work independently, we can control the number of cores participating in processing to test speedups. A SmartBit packet generator (SmartBit6000) is used to generate 10 Gbps bi-directional traffic. To make a strict and objective test, packet size is set to 64B (minimum Ethernet packet size) and the packet loss rate is set to *zero*.

Figs. 7 and 8 are the results of firewall throughput test. In our prototype, each session entry has 256B size and totally 2M session entries are supported. The input traffic is 1,000,000 UDP flows (varied in 5-tuple) with 64B packet size. We can see from Fig. 7 that as the number of cores increases, the firewall throughput grows linearly. The throughput finally reaches near line-rate with 11 cores.

Fig. 9 shows the session creation rate on NPM. We use SmartBit to generate 64B TCP SYN packets with different 5-tuple headers, so that NPM will create a new session for each of them. By monitoring the SYN packet receiving rate on SmartBit, we can figure out the session creation rate on the NPM (without successful session creation, SYN packets will be dropped by the NPM). According to Fig. 9, the session creation rate reaches near 3 millions new session/second with only 6 cores (due to SmartBit performance, we have no results with more cores). Compared to today's high-end firewall products, which commonly have 10–100K new session/second rates, the session creation rate of OpenGate-SG is extremely high. Main reasons for such a high-performance session creation rate are the fast-path session creation mechanism and an efficient policy lookup algorithm [22].

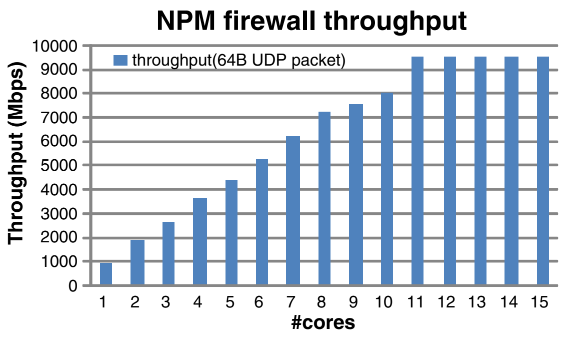

Fig. 7. NPM SI throughput.

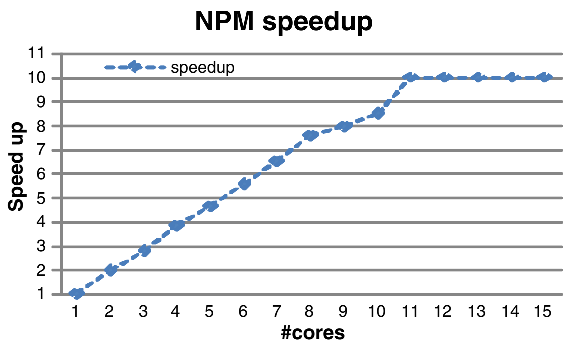

Fig. 8. NPM SI speedup.

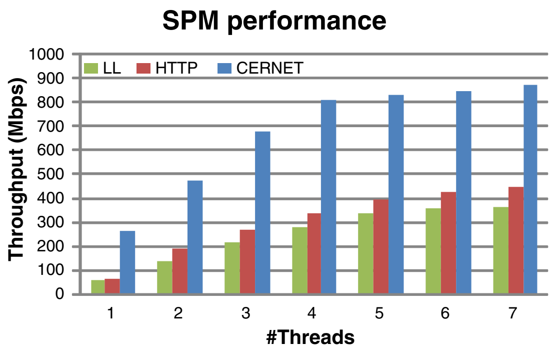

Fig. 10. SPM DI throughput.

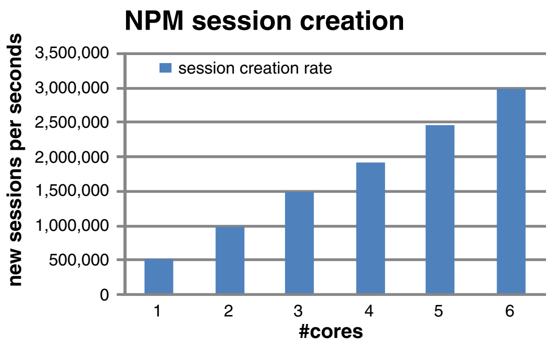

Fig. 9. NPM Session creation rate.

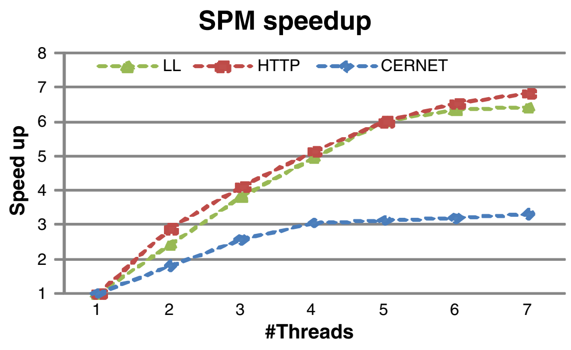

Fig. 11. SPM DI speedup.

5.3. Deep inspection performance

The software we used for deep inspection includes the open source Snort2.8 and ClamAV0.95. The source code of thread-isolator is an extended version of *dispatcher.c* in Snort3.0 beta [15]. In our test, Snort2.8 is configured to support all important security features, including Stream5 and HTTP Inspect preprocessors. All rules (over 10,000) of Snort2.8 and ClamAV0.95 are used in our test. They are compiled using the Aho-Corasick Full algorithm for fast processing, consuming 2.6 GB memory space for 7 threads (with memory sharing). Test traffic includes both real-life traffic traces and synthetic HTTP data (shown in Table 2). All these traces are sent to the OpenGate-SG using TCP replay [23].

Figs. 10 and 11 are the DI performance of a single SPM blade. We can see from these figures that the performance grows near linearly with the number of threads (because we set core affinitive for service threads, the number of threads is the same as the number of cores). With 4 cores, a single SPM reaches more than 800 Mbps performance for the CERNET traffic. While for HTTP and Lincoln Lab traces, one SPM only achieves about 400 Mbps performance even with 7 cores. This is mainly because the number of attack and virus signatures for HTTP traffic scanning is much larger than those of other types of traffic [15,16].

Table 2
Packet traces for IPS/AV.

	LL	HTTP	CERNET
Source	MIT Lincoln Lab [21]	Generated by software	Tsinghua University RIIT
Date	1999-4	2009-6	2008-8
Size (MB)	519	–	700
Average packet size (byte)	159	792	664
HTTP packets (%)	51.5	100	5.8

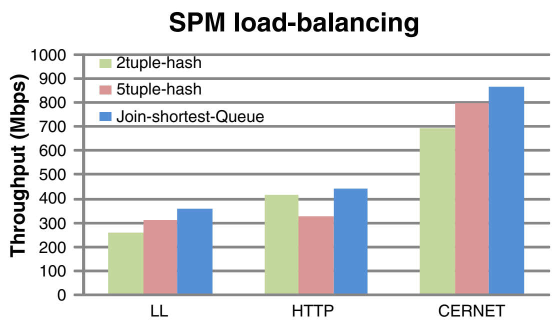

Fig. 12. SPM load-balancing.

Fig. 12 shows the impact of different load-balancing schemes on SPM. Three load-balancing schemes are implemented for comparison in our prototype: 2tuple-hash (source IP and destination IP) and 5tupe-hash (source IP, destination IP, source Port, destination Port, transport-layer Protocol) are hash-based static-binding schemes, i.e. packets with the same hash value will be processed by the same thread. In comparison, Join-the-Shortest-Queue (JSQ [24]) is a dynamic-binding scheme. A new packet flow (with the same 5tuple header) will be processed by the thread with shortest input queue length. According to Fig. 12, JSQ always performs better than 2tuple-hash and 5tuple-hash schemes. This is mainly because JSQ can dynamically distribute traffic flows according to the current statues of the service threads. As a tradeoff, JSQ needs to maintain a flow table to store existing flow-thread bindings.

5.4. Related work

Based on a standardized 12U 14-Slot ATCA chassis (e.g. Radysis ATCA-6010), 10 NPMs (other 4 slots left for 2 switch blades and 2

Table 3
OpenGate-SG vs. commercial products.

System	MAX SI	MAX DI
Fortigate-5140	70 Gbps (with 14 ATCA modules)	4.2 Gbps IPS with AV
SRX-5860	137 Gbps	8 Gbps IPS without AV
OpenGate-SG	200 Gbps (with 10 NPMs)	4–8 Gbps IPS with AV

CPMs) can be supported. According to our test, each NPM has 20 Gbps (10 Gbps bi-directional) SI performance and each SPM has up to 800 Mbps DI performance. So the overall system can achieve up to 200 Gbps SI and 8 Gbps DI performance.

In comparison to the Overlay Hosting Service (OHS) system proposed by Turner et al., which is the first example of ATCA-based open platforms to provide a reprogrammable high-performance overlay platform for PlanetLab users [7]. They use network processor (Intel IXP 2800) based ATCA blades for fast-path network processing, while general-purpose processor (Intel XEON) based blades are reserved for slow-path and control processing. Two example applications, IPv4 forwarding and Internet indirection [8], were implemented on their platform, which are designed for core routers to do *packet-level* processing at the backbone of the Internet. Compared to OHS, OpenGate-SG achieves 4 times higher throughput (20 Gbps vs. 5 Gbps), and at the same time provides *application-level* services.

Table 3 is a performance comparison of OpenGate-SG with today's most powerful commercial products, Juniper SRX-5860 [25] and Fortinet Fortigate-5140 [26]. As new-generation ATCA products will be available, such as OCTEONII based NP blades and XEON5500 based IA blades, we believe that the performance of OpenGate-SG could be further improved without redesigning the software architecture.

6. Conclusion

In this paper, we propose an extensible open network services platform for high-speed network processing, OpenGate, which embraces recent advances of open network technologies, including open source software, open system standards and open network architecture. As an ATCA-based security gateway, OpenGate-SG is implemented and tested. Experimental results show that, the OpenGate-SG can achieve up to 200 Gbps stateful firewall performance together with 8 Gbps deep inspection performance, which is competitive to today's most powerful commercial products. Our future work includes implementation of more network applications on the OpenGate platform. The load-balance strategy also needs to improve if different types of NPMs and SPMs are implemented on the same OpenGate system. Another topic is power-aware processing, i.e. how to control the overall power consumption while making best use of all the NPM and SPM resources. In summary, we believe the research on open network technologies is significant and valuable, and will bring about mutual benefit for both academic and industrial worlds.

References

[1] S. Hauger, T. Wild, A. Mutter, A. Kirstädter, K. Karras, R. Ohlendorf, F. Feller, J. Scharf, Packet processing at 100 Gbps and beyond – challenges and perspectives, in: Proceedings of ITG Symposium on Photonic Networks, 2009.
[2] EX8216 switch hardware overview. Available from: <http://www.juniper.net/techpubs/en_US/release-independent/junos/topics/concept/ex8216-hardware-overview.html>.
[3] J.C. Mogul, P. Yalagandula, J. Tourrilhes, R. McGeer, S. Banerjee, T. Connors, P. Sharma, API design challenges for open router platforms on proprietary hardware, in: Proceedings of PRESTO'08, 2008.
[4] V. Paxson, K. Asanović, S. Dharmapurikar, J. Lockwood, R. Pang, R. Sommer, N. Weaver, Rethinking hardware support for network analysis and intrusion prevention, in: Proceedings of HotSec'06, 2006.
[5] P. Crowley, D. McAuley, T. Woo, J. Turner, C. Kalmanek, Open router platforms: is it time to move to an open routing infrastructure? in: Proceedings of ANCS'07, 2007.
[6] AdvancedTCA core short form specification. Available from: <http://www.picmg.org/v2internal/newinitiative.htm>.
[7] J.S. Turner, P. Crowley, J. DeHart, A. Freestone, B. Heller, F. Kuhns, S. Kumar, J. Lockwood, J. Lu, M. Wilson, C. Wisema, D. Zar, Supercharging planetlab: a high performance, multi-application, overlay network platform, in: Proceedings of SIGCOMM'07, 2007.
[8] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, S. Surana, Internet indirection infrastructure, in: Proceedings of SIGCOMM'02, 2002.
[9] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, J. Turner, OpenFlow: enabling innovation in campus networks, ACM SIGCOMM Computer Communication Review 38 (2) (2008) 69–74.
[10] M. Casado, M.J. Freedman, J. Pettit, J. Luo, N. McKeown, S. Shenker, Ethane: taking control of the enterprise, in: Proceedings of SIGCOMM'07, 2007.
[11] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, S. Shenker, NOX: towards an operating system for networks, ACM SIGCOMM Computer Communication Review 38 (3) (2008) 105–110.
[12] D.A. Joseph, A. Tavakoli, I. Stoica, A policy-aware switching layer for data centers, in: Proceedings of SIGCOMM'08, 2008.
[13] Y. Qi, B. Xu, F. He, B. Yang. J. Yu, J. Li, Towards high-performance flow-level packet processing on multi-core network processors, in: Proceedings of ANCS'07, 2007.
[14] C. Benvenuti, Understanding Linux Network Internals, O'Reilly Media, 2005.
[15] Snort. Available from: <http://www.snort.org/>.
[16] ClamAV. Available from: <http://www.clamav.net/>.
[17] Radisys ATCA-6010. Available from: <http://www.radisys.com/Products/ATCA/ATCA-Systems/Promentum-ATCA-SYS-6010.html>.
[18] Radisys ATCA-7220. Available from: <http://www.radisys.com/Products/ATCA/Processing-Modules/Promentum-ATCA-7220.html>.
[19] Cavium OCTEON-5860. Available from: <http://www.caviumnetworks.com/OCTEON-Plus_CN58XX.html>.
[20] D.L. Schuff, Y.R. Choe, V.S. Pai, Conservative vs. optimistic parallelization of stateful network intrusion detection, in: Proceedings of PPOPP'07, 2007.
[21] DARPA intrusion detection data sets. Available from: <http://www.ll.mit.edu/mission/communications/ist/corpora/ideval/data/index.html>.
[22] Y. Qi, L. Xu, B. Yang, Y. Xue, J. Li, Packet classification algorithms: from theory to practice, in: Proceedings of INFOCOM'09, 2009.
[23] TcpReplay. Available from: <http://tcpreplay.synfin.net/trac/>.
[24] V. Gupta, M. Harchol-Balter, K. Sigman, W. Whitt, Analysis of join-the-shortest-queue routing for web server farms, in: Proceedings of Performance'07, 2007.
[25] Juniper SRX 5800 test report. Available from: <http://www.networkworld.com/reviews/2009/022309-juniper-firewall-test.html>.
[26] Fortine FortiGate-5140 data sheet. Available from: <http://www.fortinet.com/doc/FGT5000Series.pdf>.