An Extended SDN Architecture for Network Function Virtualization with a Case Study on Intrusion Prevention

Ying-Dar Lin, Po-Ching Lin, Chih-Hung Yeh, Yao-Chun Wang, and Yuan-Cheng Lai

Abstract

In conventional software-defined networking (SDN), a controller classifies the traffic redirected from a switch to determine the path to network function virtualization (NFV) modules. The redirection generates a large volume of control-plane traffic. We propose an extended SDN architecture to reduce the traffic overhead to the controller for providing NFV. The extension includes *two-layer traffic classification in the data plane, extended OpenFlow protocol messages* and *service chaining mechanisms.* Network events are analyzed in the data plane instead of the control plane. The efficiency is evaluated with a case study of intrusion prevention. The evaluation shows that only 0.12 percent of the input traffic is handled by the controller, while 77.23 percent is handled on the controller in conventional SDN.

In contrast to traditional networks, software-defined networking (SDN) decouples the control plane from the data plane to enhance programmability and flexibility of network control [1]. In SDN, the control plane is a logically centralized controller, which communicates with the data plane via a control channel, say the OpenFlow protocol standardized by the Open Networking Foundation (ONF) [2]. SDN is suitable for managing connectivity because the controller has a global view of the network. If deploying network function virtualization (NFV) modules [3] as SDN services is desired, the controller will have to handle numerous control messages and incoming packets from the data plane to classify them or even inspect packet payloads.

This work presents an extended SDN architecture to reduce the traffic overhead to the controller and support NFV by service chaining [4]. We designed a two-tier mechanism to classify traffic on the data plane instead of the control plane as much as possible. In the first tier, a classification module, which inspects the TCP/IP and application headers, is added on the switch. If this module is unable to determine the policy to be applied, a deep packet inspection (DPI) module in the second tier serves as an NFV module and analyzes the packets that cannot be classified in the first tier. The controller determines the services to be invoked based on the classification results, and makes the decision about service routing. Both modules will retain the decision, so subsequent packets will be directly forwarded to the NFV modules designated by the controller. We implemented intrusion prevention as the example for evaluating the extended architecture.

The rest of this article is organized as follows. We first present the background of this work. Then we present the extended architecture, and evaluate its performance. Finally, we conclude this work.

Ying-Dar Lin, Chih-Hung Yeh, and Yao-Chun Wang are with National Chiao Tung University. Yao-Chun Wang is also with Chunghwa Telecom Laboratories.

Po-Ching Lin is with Nation Chung Cheng University.

Yuan-Cheng Lai is with National Taiwan University of Science and Technology.

Background

Traditional network devices have fixed firmware and functions developed by manufacturers in a closed manner, so experimenting with network innovations is usually restricted. The tightly coupled control plane and data plane also reduces the flexibility of network management. SDN is the evolutionary consequence toward the desire for a programmable network and the separation of the control plane and the data plane. OpenFlow was proposed by McKeown *et al.* to standardize the communications between the centralized controller and the managed switches in SDN [5]. The idea is inherited from previous works in programmable networks such as active networks, which allows instructions to be carried by packets and executed by network devices. After programmable networks, projects toward the separation of the control plane and the data plane, such as the forwarding and control element separation (ForCES) framework, were developed to standardize the communications between multiple control and data elements within the same network. OpenFlow followed up, and its success further promoted the wide deployment of SDN. Due to limited space, we refer the readers to [5] for a complete survey of OpenFlow development. The technical background and related work are described in the following.

OpenFlow Protocol and NFV

The OpenFlow switch specification defines the communication messages between the control plane and the data plane and their processing [2]. An OpenFlow switch handles incoming packets by matching the entries in the OpenFlow pipeline, which consists of one or multiple flow tables. The switch applies the instructions in the matched entries, and performs actions such as packet forwarding. In case of a table miss, the switch may send an `OFPT_PACKET_IN` message to the controller to ask for the actions to be applied, depending on the configuration. The controller then replies to the switch with an `OFPT_PACKET_OUT` or `OFPT_FLOW_MOD` message containing an action list. The controller can specify to add, modify, or delete flow entries on the switch. The specification also defines the messages for the controller to collect information from a switch.

The controller is responsible for extracting network events, collecting statistics, and analyzing payloads for service routing decisions to support NFV modules, which make up a framework that decouples network functions from proprietary hardware appliances and enable dynamic methods to construct and manage network functions with virtualization. The data plane can help the tasks and notify the controller to relieve its burden. The amount of traffic from the switch to the controller is also reduced. However, OpenFlow defines neither flow entries for various network events nor messages for passing high-level network states to the controller.

NFV also simplifies the process of service chaining. Furthermore, a network service header (NSH) [6] is metadata added to a packet for creating a service plane, and it is encapsulated in an outer header for transport.

Related Work

Companies such as Cisco, Qosmos, Juniper, Huawei, and Ericsson [7–11] have solutions for building service chains with traffic classification functions. On the contrary, we focus on extending the data plane and the OpenFlow protocol messages. The primary difference from the prior works in terms of traffic classification is the two-tier classification, the extended OpenFlow protocol messages, and service chaining mechanisms. Furthermore, the previous works do not reveal or evaluate their design in detail. Curtis *et al.* [12] designed the DevoFlow framework to keep as many flows as possible in the data plane and reduce the traffic overheads to the controller, but that work is irrelevant to traffic classification on the switch or providing NFV. Shin *et al.* [13] implemented the FRESCO architecture to support security services on the controller, but it did not reduce the burden of the controller and is irrelevant to NFV.

The Extended Architecture

The extended architecture will be presented in three subsections. First, we present an overview of this architecture, and then describe the extended OpenFlow protocol messages. Finally, we illustrate the operation with two case studies of intrusion prevention.

Overview of the Extended Architecture

The extended architecture is intended to map NFV modules to SDN. Compared with deploying NFV modules in an OpenFlow-based SDN, this architecture moves traffic classification from the controller to the data plane, and extends OpenFlow messages with a matched field of network events. The data plane reports network events to the controller after classification, and the controller then sets the extended table entries on the data plane. Thus, the controller's burden is relieved. Figure 1 illustrates the extended architecture. The OpenFlow

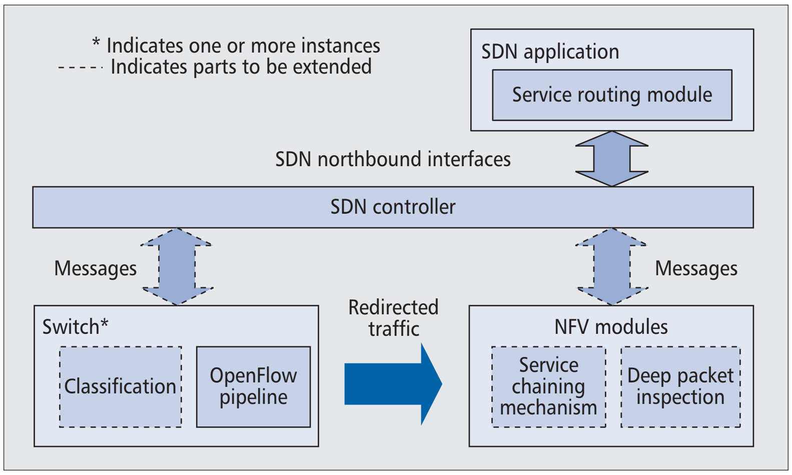

Figure 1. The extended architecture.

pipeline is extended to service chaining policies. If the switch cannot classify the traffic, the DPI module can extend the data plane and help classification. The OpenFlow messages are also augmented to support the extension. The SDN application implements the service routing logic via the northbound interfaces.

This architecture classifies incoming traffic for service chaining, and refers to network events, statistics, and packet headers to decide the service chaining policies on the data plane. The classification (CLA) module is located on the switch. Payload analysis is shifted to the DPI function as an NFV module because the analysis is too expensive to be performed on the switch. If the policy tables on the data plane are missing, the data plane will send an extended OpenFlow message to query the controller about the policies. The controller will make the decision and reply based on the network states and the packets on the data plane. OpenFlow is extended to support such communications between the control plane and the data plane. In summary, the extensions involve modifications on the switch, the NFV modules, and the extended OpenFlow to lift the limitations of the conventional SDN architecture. Other NFV modules, if any, also belong to the data plane. The service routing (SR) module on the control plane serves as the decision maker for the policies maintained on the data plane.

The CLA and DPI modules classify incoming traffic for various NFV modules according to network events. The former extracts network events from the transport layer and application headers of incoming packets, while the latter extracts from the entire packets, particularly the payloads, after packet reassembly. Both modules extend the OpenFlow pipeline to classify traffic to different policies based on the extended *Match* field composed of network events, besides the original OpenFlow *Match* field. The SR module accesses the statistics of network events, and decides the paths of NFV modules for various network events. The traffic will then be redirected to the NFV modules on the path. The SR module may modify the policies on the CLA and DPI modules due to the changes of network states.

CLA Module: At the start, the switch initializes the policy table based on the policy modification messages from the controller. If an incoming packet is from the controller, this module processes it based on the message type; otherwise, this module extracts the network event inside it. This module then matches the network event against the service chaining policy table to find the policy to be applied, and forwards the packet to the original OpenFlow pipeline for connectivity services, to the DPI module for deep packet inspection, to some other module, etc. The classification in the current design involves matching the TCP/IP and application headers with patterns specified by the controller. The other traffic classification techniques in the literature, e.g. those based on statistical features, may also be implemented in future work. If the table-miss policy is matched, the module stores the packet into a buffer and queries the SR module on the controller for the policy to enforce. The subcomponents of this module are described as follows.

Initializer: The default policies vary with different service routing logics. The module asks the SR module for the default policies to be configured, and initializes the policy table with them.

Control/Data Packets Separator: This separator checks whether the packets are control messages from the controller or data packets. The control message handler will process the former, and the consequent analysis procedures will process the latter.

TCP/IP Header Analyzer: This analyzer extracts the event in the TCP/IP header of an incoming packet, e.g. a SYN flag. The *Event* and the *Match* fields defined in the OpenFlow specification will be matched against the policy table. Three default policies are checked before the other ones. The first checks whether the application analyzer should analyze the application header of the packet. The second checks whether the NFV modules should process the packet. If they should not, the packet is handed to the conventional OpenFlow pipeline. The third checks whether the packet needs to be forwarded to the DPI module. If none of the policies are matched, the switch will ask the SR module.

Application Analyzer: This analyzer parses the application header and matches against the predefined patterns to extract the event. The packet may need to be processed by the NFV modules, refer to the policy suggested by the SR module, or be decided by the DPI module, which extracts the events after payload inspection.

Control Message Handler: This handler processes the packets of control messages. It may apply actions to packets, update the policy table, or reply the statistics to the controller.

Statistics Handler: The statistics are updated according to the policy after the incoming packet is processed. The SR module can configure the frequency or the condition for reporting the statistics from the CLA.

DPI Module: The DPI module inspects the packet payloads designated by the CLA module. Its subcomponents are described as follows.

Initializer: This module is initialized by the default policies that depend on the patterns to be inspected.

Application Payload Analyzer: This module extracts the events from application payloads by deep packet inspection. The events and the *Match* fields defined in the OpenFlow specification are matched against the policy table. If no match is found, this module will ask the SR module for the policy to be applied; otherwise, it applies the matched policy, which indicates whether the incoming packets need more NFV modules or not. The other subcomponents, i.e. control/data packets separator, control message handler, and statistics handler, are similar to those in the CLA module. Unlike CLA, the DPI module maintains the policies for the entire packet payloads.

SR Module: This module maintains network states according to the network events, statistics, and packets from the data plane. If an incoming packet is a message of OpenFlow connection establishment, this module initializes the policy table on the switch. If it is a message like OFPT_PACKET_IN in Openflow [2], this module will reply with the messages to update the policy table in the data plane. This module also updates the network states by the statistics reported from the data plane.

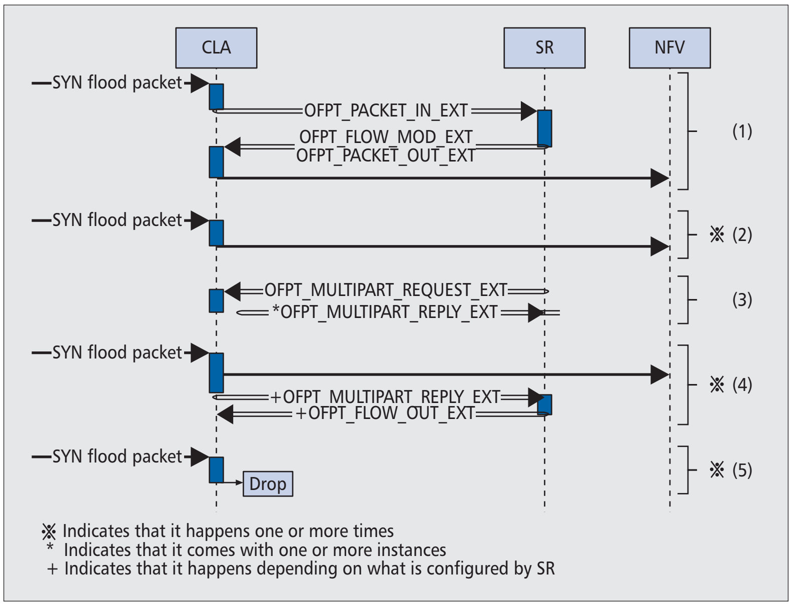
CLA
SR
NFV
SYN flood packet
OFPT_PACKET_IN_EXT
OFPT_FLOW_MOD_EXT
OFPT_PACKET_OUT_EXT
(1)
SYN flood packet
※ (2)
OFPT_MULTIPART_REQUEST_EXT
*OFPT_MULTIPART_REPLY_EXT
(3)
SYN flood packet
※ (4)
+OFPT_MULTIPART_REPLY_EXT
+OFPT_FLOW_OUT_EXT
SYN flood packet
Drop
※ (5)
※ Indicates that it happens one or more times
* Indicates that it comes with one or more instances
+ Indicates that it happens depending on what is configured by SR

Figure 2. The scenario of detecting SYN flooding.

Extended OpenFlow Protocol Messages

We added the concept of network events into the original OpenFlow protocol. The original *Match* field was extended with the *Event List* field. We also extended the `OFPT_PACKET_IN`, `OFPT_FLOW_MOD` messages and the original OpenFlow table entries, as well as the `OFPT_PACKET_OUT` message with the action list of the *Apply-Actions* instruction in OpenFlow. The added action defines the path identification of the NFV modules. The actions of the instruction field in the original OpenFlow table entry were also extended. For the `OFPT_MULTIPART_REQUEST` and `OFPT_MULTIPART_REPLY` messages, we extended `OFP_MULTIPART_TYPE` for the statistics of the extended service chaining policy.

Cases of Intrusion Prevention

We use two cases of intrusion prevention for the study. The first case is detecting SYN flooding from the TCP/IP headers. The second is detecting SQL injection and cross-site scripting (XSS) attacks for web applications. The messages in the form of `OFPT_*_EXT` support high-level network events and service chaining, and are the extended version of their counterparts without the suffix _EXT in the OpenFlow specification.

SYN Flooding Attack: Figure 2 presents the stages in the sequence diagram. The CLA module extracts events from the TCP flags. In stage (1), this module extracts the SYN event from the first SYN packet, and sends an `OFPT_PACKET_IN_EXT` message with this event to the SR module because there are no corresponding policies in this module. After the SR module replies with the `OFPT_PACKET_OUT_EXT` and `OFPT_FLOW_MOD_EXT` messages, the policy to detect SYN flooding attack is set in CLA to handle the SYN packet, which is still forwarded to the subsequent NFV modules before being detected as an attack. The destination IP address and port of a SYN packet are critical fields in the policy, and the other fields are wildcard. In stage (2), the following SYN packets to the same destination IP address and port will

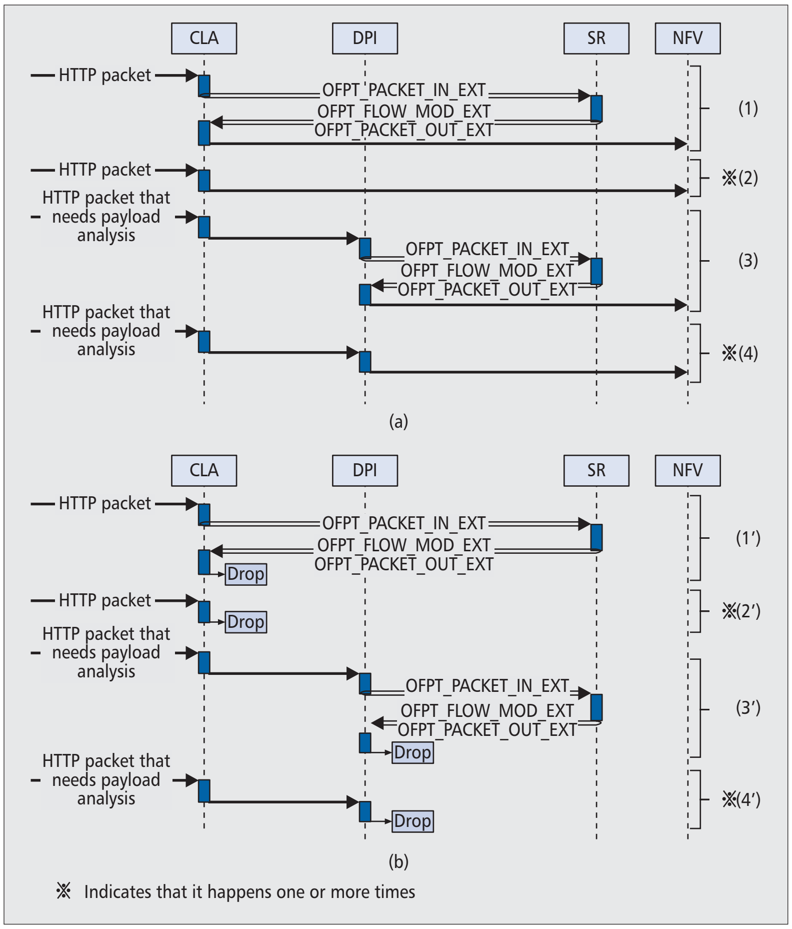

Figure 3. The scenario for web application attacks.

match this policy. By sending an OFPT_MULTIPART_REQUEST_EXT message in stage (3), the SR module inquires about the count of matched packets in incomplete connections (i.e. the connections without the complete three-way handshake), which are reported with one or more OFPT_MULTIPART_REPLY messages from CLA. In stage (4), the SR module detects that the count increases in a short time from the statistics, and realizes the occurrence of SYN flooding. It then sends an OFPT_FLOW_MOD_EXT message to the CLA module to modify the policy into a DROP action. After that, in stage (5), all the SYN flooding packets will match this policy and be dropped by CLA.

Web Application Attack: Attacks on web applications such as SQL injections usually come with malicious patterns in HTTP requests. Figure 3 presents an example, where Fig. 3a and Fig. 3b present the scenarios without and with malicious patterns in the HTTP packets. If malicious patterns are present in the HTTP headers, the CLA module can parse the headers and check for malicious patterns. However, if malicious patterns are in the HTTP payloads, the DPI module will inspect the payloads to check for them.

In stage (1) of Fig. 3a, the CLA module does not extract any events about malicious patterns in the HTTP packet, while in stage (1') of Fig. 3b, the module identifies malicious patterns. In either case, this module sends an OFPT_PACKET_IN_EXT message to the SR module because the corresponding policy has not been set, and receives OFPT_PACKET_OUT_EXT and OFPT_FLOW_MOD_EXT messages from the controller. The messages instruct CLA to forward the HTTP packet to the subsequent NFV modules and set the corresponding policy in stage (1), and to drop the packet in stage (1'), since a DROP action is set for the events about malicious patterns. The subsequent HTTP packets arriving at CLA in stage (2) and (2') will be forwarded to NFV modules or dropped directly.

In stages (3) and (3'), suppose an HTTP packet with an application payload arrives at the CLA module, with the default policies set to forward such packets that need DPI to the DPI module. After payload inspection, the DPI module will send an OFPT_PACKET_IN_EXT message to the SR module because of no corresponding policies, and then sets the policies from the OFPT_PACKET_OUT_EXT and OFPT_FLOW_MOD_EXT messages. The packet with malicious patterns will be dropped in stage (3'), or forwarded to the NFV modules in stage (3) otherwise. The DPI module will not ask the SR module for the following HTTP packets that need DPI after the policies have been set. In stage (4), the DPI module sends the incoming packets to NFV modules while in stage (4'), and it drops the incoming packets directly by the policy for malicious patterns.

Experiment and Analysis

Experiments for Throughput Analysis

In the first experiment, the performance of transmission (1) with classification in the control plane and (2) with classification in the data plane were compared. We used mininet (mininet.org) and a Ryu controller (osrg.github.io/ryu) to build OpenFlow-based SDN for a network service. We also implemented a Ryu application to handle service routing by the TCP header. Two hosts simulate the client and the server, between which are three switches controlled by Ryu, numbered from 1 to 3 from the client. Between switches 1 and 2 is a firewall running iptables (allowing web traffic or SYN packets) or a direct link, and between switches 2 and 3 is a Snort IDS with default rules or a direct link. Thus, four types of service paths are possible with this topology.

We simulated each scenario by adjusting the SDN application design. For classification in the control plane, the scenario was as follows. The switches handle incoming traffic by sending each incoming packet with an OFPT_PACKET_IN message to the controller, which then replies with an OFPT_PACKET_OUT message to forward the packet. As a result, each packet needs to pass through the controller and be classified by its TCP header to emulate classification in the control plane.

On the other hand, for classification in the data plane, the scenario was as follows. Switch 1 handles incoming client traffic by sending the first packet with an OFPT_PACKET_IN message to the controller, which then sends an OFPT_FLOW_MOD message with the OFPFC_ADD command to switche 1, switch 2, and switch 3 to forward the packets. As a result, the rest of the packets do not pass through the control plane because of the classification on each switch. We used Iperf (iperf.fr) to send traffic with destination TCP port 80 through each service path in the above two scenarios, and measured the TCP throughput in 30 seconds. Iperf sent data in one connection with a default TCP window size of 85.3 Kbytes. The bandwidth of network interfaces of the switches is 525 Mb/s. The experimental results demonstrate the relationship between transmission performance and the location of classification.

In this experiment, the throughput is presented for four service paths. The first service path is composed of no services; the second of firewall only; the third of IDS only; the

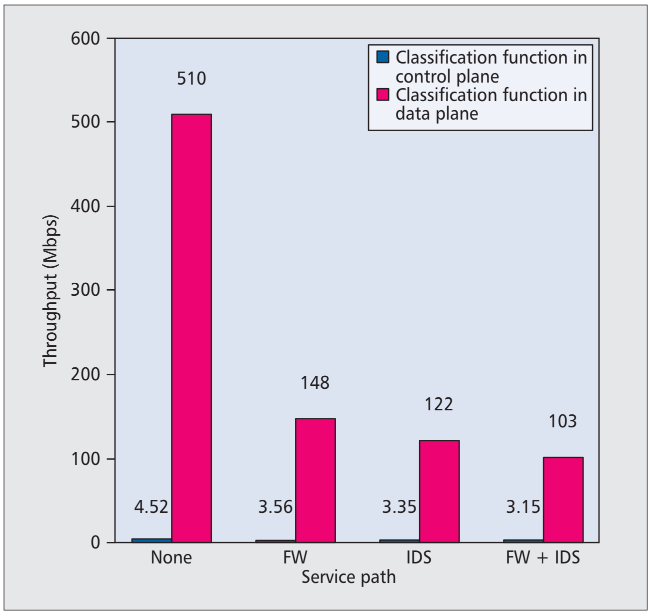

Figure 4. The throughput in various scenarios.

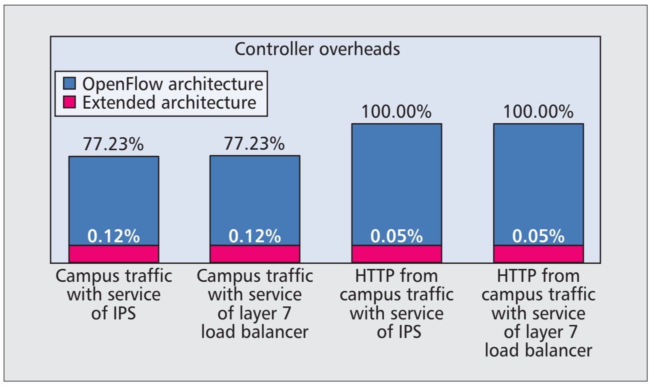

Figure 5. The comparison of controller overheads between the OpenFlow and the extended architectures.

fourth is composed of firewall and IDS. In Fig. 4, we compare the throughput in the scenarios of classification in the control plane and in the data plane. In the first scenario, each packet needs to pass through the control plane and to be classified by its TCP header. In the second, only the first packet passes through the control plane and the rest of the packets are classified by the data plane and redirected to the service path. The short transmission path with the classification in the data plane significantly increases the throughput.

Experiments for Analysis of Controller Overhead and Redirection Ratios

In the second experiment, we implemented the modules of the extended SDN architecture with a C/C++ program for traffic simulation and analysis. We ran Endace NinjaBox with Endace DAG Cards (www.emulex.com) to capture the input traffic from the campus network of National Chiao Tung University. The traffic included 1,224,895 packets. The TCP traffic accounted for 77.23 percent, and 23.63 percent of the packets come with the source or destination port number 80. The amount of traffic to be processed by the SR, DPI, and NFV modules can be derived by analyzing the input traffic, and the results are compared with those in the conventional OpenFlow-based network.

At first, CLA reads the input traffic, and then forwards the output traffic to the SR, DPI, and NFV modules. DPI processes the traffic from CLA, and then forwards the output traffic to the SR and NFV modules. The transmission delay between the data plane and the control plane was negligible, and the lifetime of the service chaining policy entries was set to infinity. We can also learn how much traffic went to the NFV modules by checking the traffic from CLA to DPI and the other NFV modules.

We study the controller overhead and the redirection ratio for the NFV modules in this experiment. The former is the amount of traffic from the data plane to the control plane, and the latter is the amount of traffic going to DPI and the other NFV modules. The experiment involved four scenarios. The first enabled an intrusion prevention system (IPS) for the campus traffic; the second enabled a layer-7 load balancer for the same traffic; the third enabled IPS for only the HTTP traffic retrieved from the campus traffic; the fourth enabled a layer-7 load balancer for the same traffic. We also enabled the aforementioned services for detecting SYN flooding and web application attacks.

In Fig. 5, the controller overheads are compared between the result from the OpenFlow-based network and that from the extended architecture. We also analyzed the events in the TCP packets. In the first two scenarios, since the data plane cannot extract the events, all the TCP packets, which account for 77.23 percent of the input traffic, will be redirected to the controller for event extraction. However, the extended data plane can extract the events, and only 0.12 percent of the input traffic is processed by SR for the policy-miss cases. In the third and fourth scenarios, since the data plane cannot extract the HTTP events, the input (pure HTTP) traffic all needs to be redirected to the controller. However, the extended data plane can extract the application layer events, and only 0.054 percent and 0.053 percent of the input traffic is processed by SR.

Figure 6 presents the redirection ratios in the four scenarios. We consider HTTP POST requests and HTTP responses with status code 200 as the packets to be inspected in the application payload, and DPI is the destined NFV module for these packets. We found that only 0.06 percent of the input traffic needed to be sent to DPI for traffic classification in the first two scenarios, and the redirection ratios to the NFV modules were 23.63 percent and 0.42 percent. The amount of traffic to the NFV modules varied with the enabled NFV modules. In the first scenario, all the TCP packets with source port or destination port of 80 were redirected to the IPS service. On the contrary, only the packets to the web application server with load balance service enabled needed to be redirected to NFV modules in the second scenario.

In the third and fourth scenarios, the input traffic is all HTTP, all of which was redirected to NFV modules in the third scenario for IPS service, while that to the web application server was redirected to the NFV modules in the fourth scenario. The results show only the necessary traffic will be redirected to the NFV modules.

Conclusions

Compared with OpenFlow-based SDN, the extended architecture causes little controller overhead for providing NFV modules. The CLA module on the switch classifies most traffic, and the DPI module is responsible for the rest that cannot be classified by CLA. We studied the controller overhead and the redirection ratios in various scenarios. The results show that controller overhead can be reduced from 77.23 percent to 0.12 percent, demonstrating the feasibility of this extended

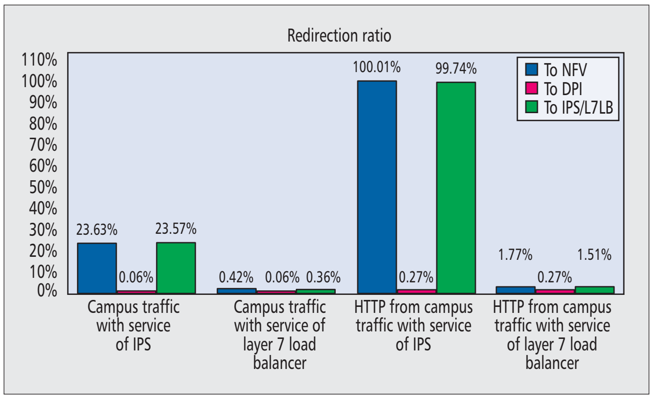

Figure 6. The redirection ratios in the four scenarios.

architecture. The throughput is improved from a few Mb/s to more than 100 Mb/s because of the short transmission path. Moreover, only the necessary traffic will be redirected to the NFV modules, so the redirection ratios vary with the composition of traffic types and the required NFV modules. Therefore, the extended architecture is a feasible approach to extend the OpenFlow specification for supporting NFV modules in SDN in the future.

Acknowledgment

This work was supported in part by the Ministry of Science and Technology (MoST), Chung-Hwa Telecom (CHT), and the Industrial Technology Research Institute (ITRI) in Taiwan, and also in part by Mediatek Inc.

References

[1] ONF, "SDN Architecture Overview," Dec. 12, 2013.
[2] ONF, "OpenFlow Switch Specification version 1.4.0," Oct. 14, 2013.
[3] ETSI, "Network Functions Virtualization — Introductory White Paper," http://portal.etsi.org/NFV/NFV_White_Paper.pdf.
[4] P. Quinn, Ed, *et al.*, "Service Function Chaining Problem Statement," IETF Internet-Draft, draft-ietf-sfc-problem-statement-02.txt, Feb. 14, 2014.
[5] A. Lara, A. Kolasani, and B. Ramamurthy, "Network Innovation using OpenFlow: A Survey," *IEEE Commun. Surveys & Tutorials*, vol. 16, issue 1, 1st Quarter, 2014, pp. 493–512.
[6] P. Quinn, *et al.*, "Network Service Header," IETF Internet-Draft, draft-quinn-sfc-nsh-02.txt, Feb. 14, 2014.
[7] Cisco, "Enabling Service Chaining on Cisco Nexus 1000V Series," http://www.cisco.com/c/en/us/products/collateral/switches/nexus-1000v-switch-vmware-vsphere/white_paper_c11-716028.pdf.
[8] Qosmos, "Service-Aware Network Architecture Based on SDN, NFV, and Network Intelligence," http://www.qosmos.com/wp-content/uploads/2014/01/Intel_Qosmos_SDN_NFV_329290-002US-secured.pdf.
[9] Juniper Networks, "Contrail Architecture," http://www.juniper.net/us/en/local/pdf/whitepapers/2000535-en.pdf.
[10] Huawei, "Enabling Agile Service Chaining with Service Based Routing," http://www.huawei.com/ilink/en/download/HW_308622.
[11] W. John *et al.* "Research Directions in Network Service Chaining," *IEEE SDN for Future Networks and Services (SDN4FNS)*, 2013.
[12] A. R. Curtis *et al.*, "DevoFlow: Scaling Flow Management for High-Performance Networks," *ACM SIGCOMM*, Aug. 2011.
[13] S. Shin *et al.*, "Fresco: Modular Composable Security Services for Software-Defined Networks," *Network & Distributed System Security Symp. (NDSS)*, Feb. 2013.

Biographies

YING-DAR LIN [F'13] is a distinguished professor at National Chiao Tung University (NCTU). He received his Ph.D. from UCLA in 1993. He was a visiting scholar at Cisco in 2007–2008. He directs the Network Benchmarking Lab (NBL), an approved lab of the Open Networking Foundation (ONF). His work on "multi-hop cellular" has been highly cited and standardized. He is an IEEE Distinguished Lecturer, and ONF research associate, and serves on several editorial boards. He co-authored *Computer Networks: An Open Source Approach* (McGraw-Hill, 2011).

PO-CHING LIN received the M.S. and Ph.D. degrees in computer science from National Chiao Tung University, Hsinchu, Taiwan, in 2001 and 2008, respectively. He joined the faculty of the Department of Computer and Information Science, National Chung Cheng University (CCU), Chiayi, Taiwan, in August 2009. He is currently an assistant professor. His research interests include network security, network traffic analysis, and performance evaluation of network systems.

CHIH-HUNG YEH received his master's degree in computer science from National Chiao Tung University in 2014. He is currently fulfilling his alternative military service, which will end in September 2015, as a project assistant under the Information and Communication Technology Project of Taiwan ICDF in the Taiwan Technical Mission in Saint Christopher and Nevis. His research interests include network architecture, software-defined networking and cloud computing.

YAO-CHUN WANG is an associate researcher at Chunghwa Telecom (CHT). He received his M.S. degree from National Chiao Tung University (NCTU) in 2011. He is a Ph.D. student at National Chiao Tung University (NCTU).

YUANG-CHENG LAI received the Ph.D. degree in computer science from National Chiao Tung University in 1997. He joined the faculty of the Department of Information Management at National Taiwan University of Science and Technology in 2001, and he has been a professor since 2008. His research interests include wireless networks, network performance evaluation, network security, and social networks.