Soft Partitioning Flow Tables for Virtual Networking in Multi-tenant Software Defined Networks

Ying-Dar Lin, *Fellow, IEEE*, Te-Lung Liu, *Member, IEEE*, Jian-Hao Chen, and Yuan-Cheng Lai, *Member, IEEE*

***Abstract*—In a multi-tenancy SDN environment, physical devices such as switches are shared among tenants. In addition to a centralized controller, each tenant has his own controller that manages resources allocated to the tenant. Hence, the centralized controller performs SDN resource virtualization among tenants and acts as proxy between physical resources and tenant controllers. In order to manage the flow tables of the SDN switches, two partitioning strategies are considered. Hard partitioning of flow tables allocates a fixed amount of flow entries to each tenant, but flow tables are wasted if the tenant does not actually use them. On the other hand, soft partitioning strategy shares available flow entries among tenants, resulting in higher utilization but a resource monopoly problem, i.e., flow entries dominated by some greedy tenants. To achieve high flow table utilization and avoid the resource monopoly problem, we propose a Soft-Partitioning Resource Manager (SPRM) to manage the flow table resources in a multi-tenancy SDN environment. In SPRM, the allowed number of flow entries for each tenant ranges from a lower bound which equals to the tenant's quota to an upper bound which is dynamically adjusted according to the tenant's past usage. If an incoming flow request of a tenant is beyond his lower bound but under his upper bound, it could be temporarily accepted when there are free entries available. These borrowed flow entries will later be replaced if needed. If a request of a tenant is beyond his upper bound, SPRM will select a least-recently used flow entry of the tenant and replace it with the new request. In addition, SPRM monitors flow table resources and submits modify flow entry messages directly to SDN switches without checks by the management plane as possible in order to reduce flow modification latency. As a result, SPRM could reach higher flow table utilization and lower both flow entry miss rate and Packet_in events. Experimental results show that 100% flow rejections, and 95% Packet_in events are reduced while flow modification latency is decreased by 30%, as compared to hard partitioning.**

***Index Terms*—SDN, OpenFlow, soft-partitioning, resource management, multi-tenancy**

First manuscript received on Oct. 12, 2016. Revisions received on Mar. 3, 2017. Resubmitted manuscript received on Oct 6, 2017.

This work was supported in part by Ministry of Science and y, Taiwan, and in part by Chunghwa Telecom.

Y. D. Lin and J. H. Chen are with Department of Computer Science, National Chiao Tung University, Hsinchu, Taiwan, (e-mail: ydlin@cs.nctu.edu.tw, a7533258@gmail.com).

T. L Liu is with National Center for High Performance Computing, Tainan, Taiwan, (e-mail: tlliu@narlabs.org.tw).

Y. C. Lai is with Department of Information Management, National Taiwan University of Science and Technology, Taipei, Taiwan, (e-mail: laiyc@cs.ntust.edu.tw)

I. Introduction

Network virtualization allows IT managers to consolidate multiple physical networks by dividing the network into multiple segments or creating software-only networks among virtual machines. The goal of network virtualization is to improve provision time and manageability with automation by adding new software elements. The process of network virtualization may be programmable with application programming interface (API) calls, or simply configured through web user interface (UI) or command-line interface without programmability. In the past, Virtual Private Network and other technologies are deployed to achieve network virtualization by provisioning tunnels across multiple network domains. However, these tunnels are configured with vendor-specific commands or software so that cross-vendor operation cannot be reconfigured easily for modern cloud environments. Recently, the idea of Software Defined Network (SDN) [1][2] arises by separating control and data planes. With centralized control plane and standard programmable API for configuring devices, SDN can assign traffic flows on demand, provide application-level Quality of Services, manage the networks conveniently, and enable dynamic network virtualization without vendor lock-in.

According to the survey in [3] by A. Blenk et al, when network providers perform network virtualization with SDN technology, there are two kinds of virtual network provisioning. One is traditional multi-tenant network, where network providers use SDN applications to allocate virtual networks from physical network resources. Tenants do not need SDN controller in their virtual environment and simply configure required options as in traditional networks. Current solutions include OpenDaylight Virtual Tenant Network [4], and so on. The other kind is multi-tenant SDN (a.k.a. Virtualizing the SDN Network [3]), where network providers use the controller as a proxy to virtualize SDN resources. Tenants have to install their own SDN controllers, called as tenant controller, to administrate virtual SDN switches through OpenFlow protocol and use SDN applications to manage their own virtual networks. In this architecture, a proxy controller (a.k.a SDN network hypervisor [3]) acts as the control plane for physical switches and the data plane for tenant controllers. With proxy controller, SDN network is much more flexible because multi-layer virtualization can be achieved and each tenant could choose its own controller. Current solutions include FlowVisor [5], OpenVirteX [6], CoVisor [7], ADVisor [8] and so on.

In SDN networks with proxy controller, resource partitioning is an important issue. Various tenants share the same physical resources, such as bandwidth and flow table, that require proper partitioning techniques. Flow table [1][2][3] exists in each SDN switch, which consists of a set of flow entries. For each incoming packet, if there is a matched flow entry, the action that associated with the entry will be executed; else the packet will be executed with pre-configured default action. In this paper, we focus on the flow table partitioning among the tenants.

Flow table partitioning methods can be further divided into two categories: hard partitioning and soft partitioning. In hard partitioning, each tenant has a flow table limitation, namely quota, as the upper bound, and each tenant has no authority to consume quotas from other tenants. This type of partitioning will waste flow table space when high demand tenants have insufficient quota but low demand tenants cannot share unused flow entries. On the other hand, for soft partitioning strategy, each tenant has a quota as the lower bound and flow table resources are shared among tenants. For tenants that request more flow entries, they are allowed to use them from other tenants' quotas and therefore resource wasting can be avoided. Although resources could be shared with soft partitioning, the total flow table size is still limited by hardware. When one tenant consumes a huge amount of flow entries, other tenants may not be able to set their flow entries. We refer this situation as resource monopoly problem.

In this paper, we propose a mechanism called Soft-Partitioning Resource Manager (SPRM) in a multi-tenancy SDN environment. SPRM inherits the concept of soft partitioning to obtain high flow table utilization and further tries to avoid the resource monopoly problem. That is, in addition to the quota as lower bound, we also limit each tenant with a dynamic upper bound according to tenant's past usage. If a tenant has a new request that exceeds his quota but is under the upper bound, SPRM will accept the request when there are free entries available. These flow entries may be replaced later if needed. If the new request is beyond the tenant's upper bound, we will locate a least-recently used flow entry from this tenant and then replace it with the new request. Hence, we can assure that all requests can be satisfied.

In the SDN architecture with proxy controller, flow modification requests issued from tenant controllers should be examined by the proxy controller to determine if there are flow entries available along the requested path, which causes flow modification latency. In order to save the latency, SPRM monitors the flow table usage of SDN switches and submit the tenant controller's request directly to the switches without checks as possible. Therefore, we can lower the impact of the processing time of the proxy controller.

The remainder of this paper is organized as follows. Firstly, we introduce the related backgrounds including OpenFlow, proxy controller, and management plane in Section II. Next, the problem statement is defined in Section III. In Section IV, we propose SPRM approach with detail explanation. Then we introduce our implementation and exhibit the emulation results in Section V. Finally, we make the conclusions and future work in Section VI.

II. Background

In this section, we give an overview of the OpenFlow protocol and introduce multi-tenancy SDN architecture with management plane. We also briefly describe SDN networks with proxy controller which is adopted to provide multi-tenancy SDN services. Under such scenario, resource management among various tenants is critical. Hence, we discuss and compare related works on SDN resource management.

A. *OpenFlow*

In SDN architecture, OpenFlow [13] is a well-known southbound protocol for the communication between switches and controller. An SDN switch consists of one or more flow tables. The controller can add, update, and delete flow entries in flow tables through the OpenFlow protocol. Each flow entry consists of *match field* for packet lookup and *actions* to apply to the matched packets. OpenFlow protocol defines several types of messages to manage switches. In this work, three types of messages are involved: for arriving packets that cannot match current flow entries, *Packet_in* messages are issued by switches for the controller to decide the actions to these packets. *Packet_out* messages are generated by the controller to inject a new packet to the switches. The controller also applies *Modify Flow Entry* (*OFPT_FLOW_MOD*) messages to switches for managing switch states.

Figure 1 shows the basic SDN forwarding process. In step 1, the first packet from Host 1 is received by Switch 1 and misses the flow table lookup. Next, Switch 1 sends *Packet_in* message to Controller in step 2. In step 3, after Controller receives *Packet_in* message, it processes with internal logic flow or upper layer applications and then sets flow entries to related switches according to the processing results using *OFPT_FLOW_MOD* message. Controller then sends the first packet back to Switch 1 with *Packet_out* message in step 4 and the packet is forwarded to the destination Host 2 in step 5. Finally, subsequent packets could be matched with the flow entry and are forwarded to Host 2 in step 6.

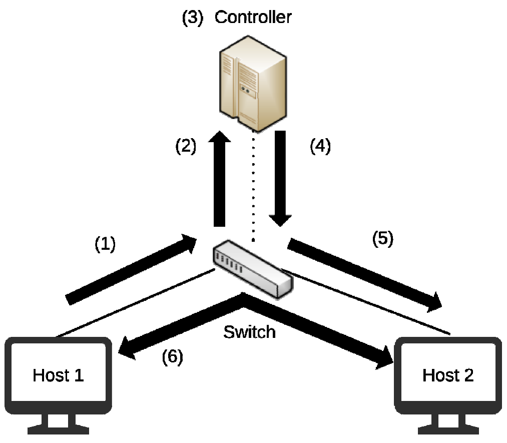

Fig. 1. Basic SDN Forwarding Process

However, flow table miss might cause huge latency because the controller needs to process more *Packet_in* and *Packet_out* messages. Therefore, it is an important indicator to illustrate network stability.

B. *Virtual Network and Multi-tenancy SDN*

Virtualization technique is not a new concept. Virtual memory and virtual disk are widely discussed in the past for creating virtual machines. Similarly, network virtualization could share underlying network among multiple users and is a hot issue in computer communication recently. From survey in [14], virtual local area networks (VLANs) allow multiple department of a company to share a physical LAN with isolation, while virtual private networks (VPNs) allow companies and employees to use public networks with the same level of security they enjoy in their private networks. However, solutions of these proposals are proprietary without standard API, and it is difficult to provision virtual networks dynamically. SDN, a new protocol to separate the data plane and control plane, provides standard API to the centralized control plane. With programmability feature of SDN, dynamic network virtualization can be achieved.

Multi-tenancy means that there are multiple tenants sharing the same physical resource. Similarly, multi-tenancy SDN means multiple tenants sharing the same SDN network. According to tenants' expectations to virtualized networks, we can divide multi-tenancy SDN into two kinds as discussed in Section 1. In the first kind, SDN network is virtualized by tools like OpenDayLight VTN [4] into multiple traditional networks. In another kind, SDN network is virtualized into several virtual SDN networks. Current solution of this type includes FlowVisor [5], OpenVirteX [6], CoVisor [7], and ADVisor [8]. FlowVisor is the first hypervisor controller introduced by ON.Lab. However, it misses some functions to implement full network virtualization and there are works like ADVisor [8] to overcome it. OpenVirteX [6] is the recent update of FlowVisor that could slice the network by full packet header space and hence supports fully virtualized networks.

C. *Multi-tenancy SDN Architecture with Management Plane*

In 2014, Open Network Fundation (ONF) defined the architecture for management by policy-based multi-tenancy SDN [15]. In this work, we modify this management plane with proxy controller. Figure 2 shows how management plane works in a multi-tenancy SDN. Two managers are defined in management plane: root manager and tenant manager. Root manager could send policies to coordinator in control plane. Coordinator manages the data plane and tenant networks according to the policies with data plane control function (DPCF). Tenants have their own tenant manager to manage their own virtual network. Tenant manager communicates with SDN applications and these applications send some messages we called intent to agent through API calls. Then the agent could send the message to virtualizer for validation. Finally, DPCF sends the validated OpenFlow message to the underlying data plane.

There are several advantages with SDN management plane. First, it can reduce control plane overhead because management plane could work independently from proxy controller. Also, management functions such as QoS can be modularized for several agents. In addition, third party developers could contribute their management module easily.

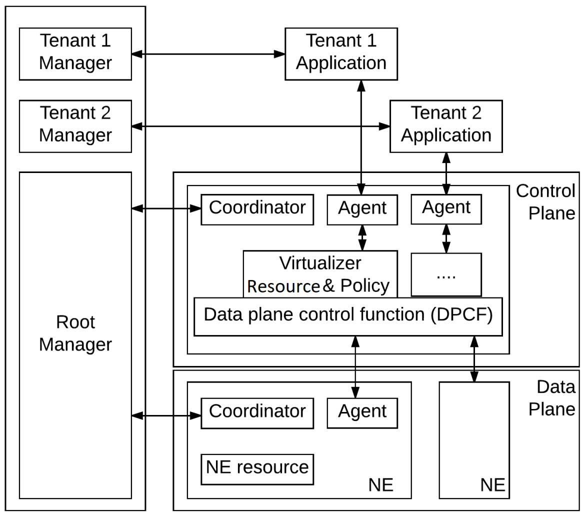

Fig. 2. Policy-based Multi-Tenancy SDN

D. *SDN Networks with Proxy Contoroller*

In the SDN network architecture with proxy controller [5] for multi-tenancy OpenFlow networks, there is a centralized controller serves as a proxy sitting between tenant controllers and physical OpenFlow switches. As illustrated figure 3, a virtualization layer is located entirely within a proxy controller. The proxy controller behaves as a virtual OpenFlow switch providing views of the virtual OpenFlow networks to tenant controllers. Meanwhile, for physical OpenFlow switches, proxy controller behaves as the OpenFlow controller.Such architecture does not require physical OpenFlow switch any additional functionality for virtualization. Currently, there are some implementations such as FlowVisor [5] and OpenVirteX [6].

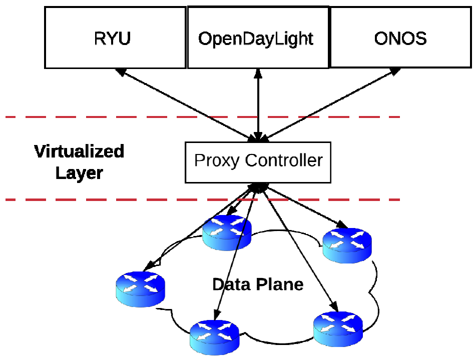

Fig. 3. SDN Networks with Proxy Controller

E. *Related Work*

In this subsection, we discuss some related works regarding SDN resource management algorithms and list the comparison in table I. We compare these proposals according to four factors: working environment, resource for share, partition type, and switch mode for management. For working environment, each proposal may works in multi-tenant or single tenant SDN environment. The resource for share could be bandwidth or flow table. As to partition type, these works could use either soft partitioning or hard partitioning method. Please note that these partitioning methods only apply to multi-tenant SDN environment only. Finally, the switch mode for management contains in-line mode, sniff mode, and hybrid mode. In in-line mode, the controller applies the flow request to the SDN switches after the management planes checks the resource availability. This will ensure the submitted flows are valid but may cause an extra latency. In sniff mode, when controller receives a flow request, it applies the requested flow to SDN switches and passes the request to management for check in parallel. The flow will be installed right away without latency in sniff mode. However, if the request is detected as invalid by management plane, it will take another extra overhead to fall back the switches to the original state. Therefore, we propose hybrid mode in our work that monitors the flow resources. If there are enough flow entries available, sniff mode will be used. When the flow table in SDN switch is full, we will use in-line mode instead. Details of the modes are given in Section IV B.

TABLE I
RELATED WORK ON SDN RESOURCE MANAGEMENT

Feature / Algorithm	Work in multi-tenant SDN	Resource sharing	Partition type	Switch mode for management
Packet schedule [10]	no	bandwidth	N/A	in-line
Bandwidth schedule with Hadoop [11]	no	bandwidth	N/A	sniff
EnterpriseVisor [12]	yes	bandwidth	soft partition	in-line
Flow Table Management [16]	no	flow table	N/A	in-line
Flow Table Management based on Forwarding Paths [17]	no	flow table	N/A	in-line
Our approach (SPRM)	yes	flow table	soft partition	hybrid

From Table 1, we could see that previous works discuss only on bandwidth management issue. In [10], the authors propose packet scheduling method that could improve bandwidth QoS and management performance. Its management algorithm works in in-line mode. Bandwidth scheduling with big data analysis is discussed in [11] and it manages the traffic in sniff mode with remote Hadoop server. However, both [10] and [11] did not apply to multi-tenancy SDN environment. For multi-tenancy SDN environment, EnterpriseVisor, which extends hypervisors with bandwidth management functions, is proposed in [12]. It could perform the soft partitioning for the bandwidth with in-line mode operation. In [16], the authors propose a flow table management mechanism. However, it does not support multi-tenancy SDN environment and works in in-line mode. The authors of [17] proposed another flow table management algorithm based on flow forwarding paths in order to reach higher performance level. Nevertheless, it operates in single tenant scenario without considering multi-tenancy.

When concerning flow table resource, most hardware SDN switches implement flow table with ternary content addressable memories (TCAM). With the limitation of capacity and power consumption considerations, A. Liu et al. proposed TCAM Razor [19] to minimize TCAM usage as possible so that we can utilize TCAM capacity more effectively. Similar techniques are applied to SDN environment. The authors in [20] investigate the effectiveness between automatic and manual rule compaction with possible side-effects. M. Rifai et al. proposed MINNIE [21] for compressing flow rules and evaluate on a real testbed. Although flow compaction approaches are not discussed in this paper, we would like to consider them in our future works.

From table I, there is no previous work discussing flow table sharing strategy in a multi-tenancy SDN environment currently. The SPRM proposed in this paper manages flow table resource with soft partitioning strategy for multi-tenancy SDN. We also propose hybrid mode that mixes in-line mode with sniff mode for better performance.

III. PROBLEM STATEMENT

In this section, we specify the notations used in this paper and formulize the problem statement. We also give a simple example that demonstrates the resource management problem with soft and hard partitioning.

A. Notation Descriptions

Table II shows the notations used in SPRM. For the parameters related to the proxy controller, we would like to identify each tenant with T and record its flow table lower bound Q.

- $T=\{t_i\}$ is the set of tenant identifications where t_i is the i-th tenant.
- $Q=\{q_i \mid q_i>0\}$ is the set of lower bound quota limitation of each tenant where q_i is the quota of t_i

For the SDN switches, there are parameters of total flow table size TFS and idle flow table size IFS.

- TFS is the total flow table size in a switch
- IFS represents number of idle flow table size in a switch.

As to the parameters for the tenants, we would like to represent the flow request sequence of the tenants by TR. For each tenant, we would like to record is current flow table usage TUE, past flow table usage TUE_{rec} and number of rejected requests TRR.

- TR represents the sequence of tenant's flow entry request.
- $TUE=\{tue_i \mid tue_i\geq 0, i>0\}$ is the set of tenant's flow usage in a switch, where tue_i is the usage of i-th tenant.
- $TUE_{rec}=\{tuer_i \mid i>0\}$ is the set of historical records of tenant's usage where $tuer_i$ is the historical usage of i-th tenant. We define $tuer_i=\{r_{i,j} \mid r_{i,j} \geq 0, j>0\}$ where $r_{i,j}$ is the tue_i in j-th second.

- *TRR={trr_i | trr_i≥0, i>0}* is the set of the number of rejected requests, where trr_i is the rejected counts of *i*-th tenant.

TABLE II
NOTATION DESCRIPTIONS

Categories	Notation	Descriptions
Proxy controller	$T=\{t_i \mid i>0\}$	The set of tenant ID
	$Q=\{q_i \mid q_i>0\}$	The set of quota for each tenant
Switch	*TFS*	The total flow table size in a switch
	IFS	The number of idle flow table size in a switch
Tenant	*TR*	Tenant sequence of entry request
	$TUE=\{tue_i \mid tue_i\geq 0, i>0\}$	The set of each tenant's flow entry usage in a switch
	$TUE_{rec}=\{tuer_i \mid i>0\}$, $tuer_i=\{r_{i,j} \mid r_{i,j}\geq 0, j>0\}$	The set of each tenant's flow entry historical usage in a switch
	$TRR=\{trr_i \mid trr_i\geq 0, i>0\}$	The set of the number of rejected requests for each tenant
Objective	$OVR=\{ovr_i \mid ovr_i\geq 0, i>0\}$	The set of overflow ratio representing dynamic upper bound of each tenant
	$Q_{res}=\{rq_i \mid rq_i\geq 0, i>0\}$	The set of reserved quota which is used to calculate *OVR*
	$SAR=\{sar_i \mid sar_i\geq 0, i>0\}$	The set of satisfaction ratio of each tenant
	PKI	The number of total *Packet_in* events.

The objectives that we would like to evaluate are also listed as follows. We will calculate the dynamic flow upper bound for each tenant *OVR*, which allows tenants to overuse their quota in order to minimize request rejections. With fewer rejections, the number of *Packet_in* events could also be minimized. We also define satisfaction ratio *SAR* for the tenants and would like to maximize it as possible.

- $OVR=\{ovr_i \mid ovr_i\geq 0, i>0\}$ represents the set of dynamic quota upper bound that we call overflow ratio for each tenants. Details are given in section IV.D.
- $Q_{res}=\{rq_i \mid rq_i\geq 0, i>0\}$ is the set of reserved quota for each tenant which is used to calculate *OVR*. Details are give in section IV.D.
- *PKI* is the number of total *Packet_in* events. If the flow table resource is not well utilized under resource wasting situation, there are more packet miss the hit of flow entry and we get higher PKI. An example is given in section III.C.
- $SAR=\{sar_i \mid sar_i\geq 0, i>0\}$ is tenant's satisfaction ratio. Obviously, it would be the ratio of allocated flow entries (tue_i) over total requesting flow entries (allocated entries + rejected counts = $tue_i + trr_i$) and represented as $\frac{tue_i}{tue_i + trr_i}$.

In addition, with SPRM, we can allocate more flow entries with dynamic overflow ratio when there are residual resources available. Therefore, there will be no rejections and the tenant usage tue_i is higher than tenant's quota limitation q_i. Under such situation, we define the satisfaction ratio as $\frac{tue_i}{q_i}$ which will be larger than 100%.

Finally, the satisfaction ratio is given as:

$$sar_i = max\left(\frac{tue_i}{q_i}, \frac{tue_i}{tue_i + trr_i}\right) \times 100\% \cdot \quad (1)$$

B. Problem Statement

This work intends to propose a mechanism to avoid the resource wasting problem in hard partitioning and the resource monopoly problem in soft partitioning. Given that the tenant set *T*, quota set *Q*, tenant usage *TUE*, tenant usage historical record TUE_{rec}, switch state *TFS* and *IFS*, and tenant request *TR*, we want to reduce *TRR* in order to (1) maximize the minimum of *SAR*, so that all tenants can reach 100% satisfaction ratio as possible, and (2) minimize the *PKI*. The constraints of the problem are (1) sum of *Q* less or equal to *TFS*, and (2) each flow entries should be specified with its idle timeout.

C. Example

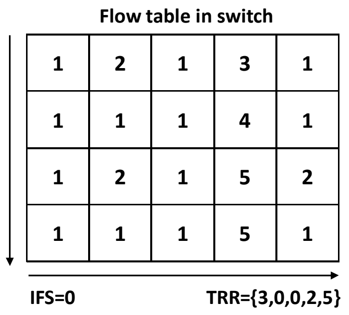

Fig. 4. Soft Partitioning Example

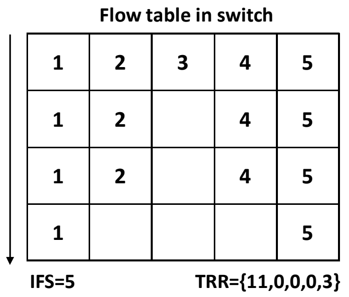

Fig. 5. Hard Partitioning Example

Figure 4 and figure 5 depict an example of resource management problem with soft partitioning and hard partitioning, respectively. Assume there are 5 tenants *T*={1, 2, 3, 4, 5}, *Q*={4, 4, 4, 4, 4}, *TFS*=20, and *TR* sequence = {1, 1, 1, 1, 2, 1, 2, 1, 1, 1, 1, 1, 3, 4, 5, 5, 1, 1, 2, 1, 1, 1, 4, 5, 5, 5, 5, 5, 4, 1}. The characters in *TR* sequence represent the corresponding tenant's request. (i.e., '1' in TR sequence represents a request from tenant '1'). Proxy controller will set entries according to *TR* sequence in first-come-first-served manner. In soft partitioning, tenants have no upper bound limitation and proxy controller only set the first twenty requests because the limitation of *TFS*. As shown in figure 4, the requests are allocated from up to down and left to right according to TR sequence. We could observe that *IFS* is zero. However, trr_5 is more than trr_1 due to its late requests. The situation results in resource monopoly problem because tenant '1' occupied most of space. On the other hand, hard partitioning defines a static upper bound limitation for each tenant. In figure 5, each tenant is reserved a column of flow entries and the requests are allocated from up to down. With hard partitioning strategy, tenant '4' can be fully satisfied (trr_4=0) and trr_5 is decreased to 3. However, both *IFS* and trr_1 are raised much higher, generating lots of *Packet_in* messages and *PKI* gets higher too. This situation results in resource wasting problem because high-usage tenants could not use the idle space that allocated to low-usage tenants.

IV. Soft-Partitioning Resource Manager (SPRM)

In this section, we state our proposed mechanism, Soft-Partitioning Resource Manager (SPRM) and its detailed operations.

A. Overview

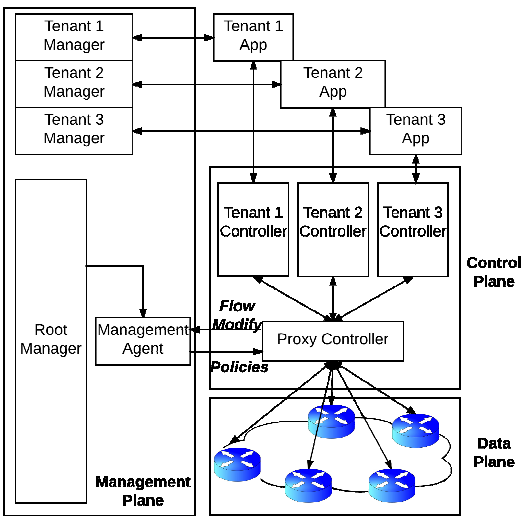

Fig. 6. Management Plane Architecture

We propose SPRM based on multi-tenancy SDN with proxy controller proposed by ONF. Figure 6 illustrates the elements for flow table resource management in this architecture. First, proxy controller receives the *Packet_in* message from data-plane switch and passes it to tenant controller in control plane. After a routing decision is determined, tenant controller sends the *OFPT_FLOW_MOD* message to proxy controller. Afterwards, this *OFPT_FLOW_MOD* message would be passed to management agent to check the availability of flow table entries. Management agent will make the decision and send a policy message to proxy controller. According to the policy, proxy controller executes the agent's decision to data-plane switch.

When a tenant sends a *OFPT_FLOW_MOD* message to the proxy controller, proxy controller forwards it to management plane to determine possible situations shown in Table III. These situations are classified according to tenant's flow usage and availability of idle entry space in the switch. In situation 1, when the switch has idle entry space and the tenant's flow usage is under its quota, a new flow entry can be set for the tenant without any problem. If the switch has idle entry space but the tenant has exhausted its quota, as situation 2, it means that the tenant wants to overflow its quota and the new entry is allowed to be set on the residual free flow space. In such case, we define an upper bound for each tenant, $OVR=\{ovr_i \mid ovr_i \geq 0\}$ where ovr_i is the upper bound of tenant *i*. If $tue_i < ovr_i$, nothing has to be done. On the other hand, the tenant's usage exceeds the upper bound and we have to release a flow entry of this tenant, which is selected by a replacement algorithm. In situation 3, switch has no idle entry space and the tenant has available entry space under its quota. It represents that other tenants have consumed this tenant's quota. We will determine a victim tenant who has the largest satisfaction ratio and replace a flow entry from victim tenant with the new requesting flow. Finally, situation 4 occurs when switch has no idle entry space and the tenant wants to overflow the quota. We will replace a flow entry from the tenant itself with the new requesting flow.

TABLE III
Four Situations When *OFPT_FLOW_MOD* Message Issued

Switch \ Tenant		**Flow usage is under tenant's quota**	
		yes	no
idle entry space available	yes	(Situation 1) Set the entry.	(Situation 2) 1. Set the entry. 2. Release tenant's flow if tenant's usage exceeds upper bound.
	no	(Situation 3) Replace victim tenant's flow.	(Situation 4) Replace tenant's flow.

The management plane functions include switch mode check, replacement algorithm, and determination of overflow ratio. Switch operates in sniff mode if there are idle entry spaces and in-line mode otherwise. Figure 7 shows the flowchart of management plane process. For each incoming *OFPT_FLOW_MOD* message from proxy controller, we determine the switch mode and compare tue_i with q_i. The switch operates in sniff mode if there are free entries in its flow table (*IFS* > 0). Otherwise, the switch is put in in-line mode. If the switch is in sniff mode and $tue_i < q_i$, it maps to situation 1 and the flow can be set without any process from management plane. On the other hand, when $tue_i \geq q_i$ under sniff mode, we will determine overflow ratio as in situation 2. If tue_i is larger than the dynamic upper bound, we will then release a flow entry from the requesting tenant itself. For situation 3, the switch is in in-line mode and $tue_i < q_i$. The management plane will replace the entry from tenant who has the highest satisfaction ratio, *sar*, with the new request. Finally, if $tue_i \geq q_i$ under in-line mode as in situation 4, the management plane will replace flow entries from the requesting tenant. Detail operations will be given in the following subsections.

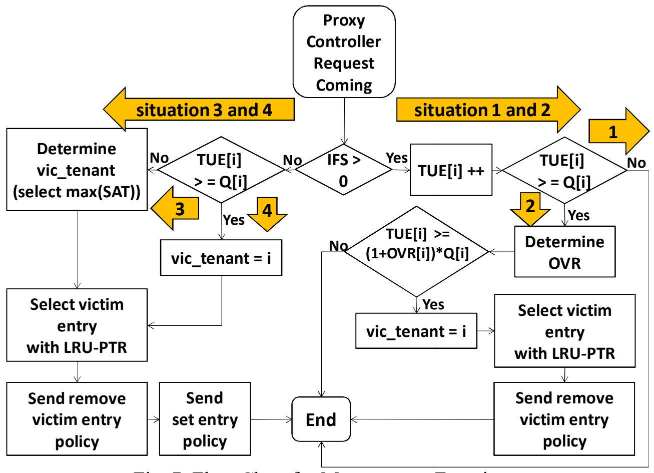
Proxy Controller Request Coming
situation 3 and 4
situation 1 and 2
1
2
3
4
Determine vic_tenant (select max(SAT))
TUE[i] > = Q[i]
IFS > 0
TUE[i] ++
TUE[i] > = Q[i]
TUE[i] >= (1+OVR[i])*Q[i]
Determine OVR
vic_tenant = i
vic_tenant = i
Select victim entry with LRU-PTR
Select victim entry with LRU-PTR
Send remove victim entry policy
Send set entry policy
End
Send remove victim entry policy
Yes
No

Fig. 7. Flow Chart for Management Functions

The complexity of SPRM could be calculated according to the four possible situations. For situation 1, the complexity is in constant O(1). In situation 2, the worst case contains overflow ratio determination and victim entry selection with LRU-PTR. Overflow ratio is calculated for each tenant within constant time and the complexity is O(|*T*|), while the victim is selected among tenant's flow entries by LRU-PTR with complexity O(tue_i), Hence, the total complexity for situation 2 is O(|*T*|)+ O(tue_i). In situation 3, the complexity includes victim tenant and victim entry selections. Similar to situation 2, the complexity for situation 3 is O(|*T*|)+ O(tue_i). In situation 4, we need only select a victim entry and the complexity is O(tue_i). So the worst case time complexity is O(|*T*|)+ O(tue_i), where *i* is the selected victim.

B. Mode

Switch mode affects how management plane deals with new flow requests. We have discussed about in-line mode and sniff mode in previous sections. The details are explained as follows

1) In-line Mode

Figure 8 shows how *OFPT_FLOW_MOD* message is proceeded in in-line mode. When a tenant sends a *OFPT_FLOW_MOD* message (step 1), proxy controller passes this message to the management agent for further checking (step 2). Then management agent sends the resulting policy to proxy controller and tenant controller that indicates whether the flow requesting could be accepted (step 3). Finally, proxy controller installs the requesting flow to the switches (step 4). In-line mode guarantees that requests could be installed properly with preventive checks. Nevertheless, it requires long processing path which might cause large flow modification latency.

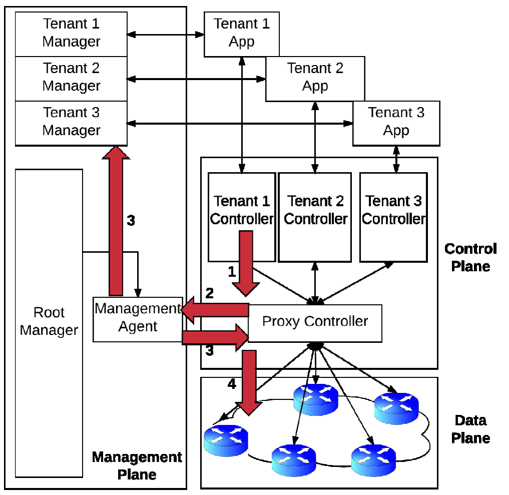

Fig. 8. Message Processing in In-line Mode

2) Sniff Mode

Figure 9 depicts the processing path of *OFPT_FLOW_MOD* message in sniff mode. When a tenant sends a *OFPT_FLOW_MOD* message (step 1), proxy controller forwards the message to SDN switches directly and mirrors the message to management agent for validation check (step 2). After policy checking, management plane send the result to both proxy and tenant controller (step 3). If there are available flow entries for the request, the connection has been already setup. Otherwise, there are some switches on the routing path do not contain enough flow entries for the request. In this case, the proxy controller will start fall-back sequence by removing those installed flow entries (step 4). In this mode, flow entries could be set right away with management plane processing simultaneously in order to save the latency caused in in-line mode. However, when the flow resource is not enough for the request, it takes more time than in-line mode in order to remove those pre-installed flows.

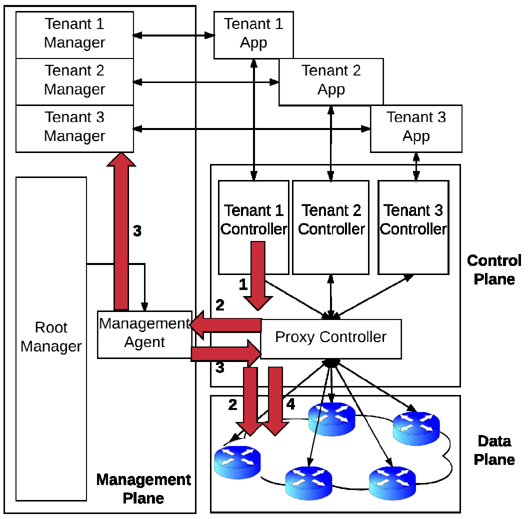

Fig. 9. Message Processing in Sniff Mode

3) Hybrid Mode

In order to take advantages of both in-line mode and sniff mode, SPRM adopts a hybrid mode that combines both modes. We monitor the flow table usage of the SDN switches. The switches with idle entry space will be turned into sniff mode for reducing flow modification latency. Alternatively, the switches with fully-used flow table are changed to in-line mode in order to avoid request rejection.

C. Least Recently Used with Partial Timeout Reset (LRU-PTR) Replacement

The replacement algorithm is one of the core modules in SPRM. The proposed SPRM executes the replacement algorithm to determine which flow entry to be replaced, so that the new requesting flow entry can be set. With the replacement algorithm, all requests from tenants are accepted, and the number of *Packet_in* events can be reduced.

SPRM chooses LRU replacement algorithm because it is a well-known algorithm with excellent performance on buffer hit over FIFO and other algorithms [19]. In order to indicate the least recently used entry, we need to have the knowledge of accurate idle time of each flow entry. However, there is no specification in OpenFlow standard to obtain such information. Therefore, we propose an approach to get approximate idle time for determining victim entry.

When LRU-PTR selects the victim entry, the most important issue is how to get an approximate idle time of each flow entry. Our solution uses OpenFlow standard flag to let switch report the idle timeout on expiration automatically. Then we reduce the idle timeout of flow entry to let switch intermittently report the idle state of the flow entry. Details are given in the following steps. First, when a tenant wants to set a flow entry with *idle_timeout* value and passes to proxy controller, proxy controller changes this to *partial_timeout*. Next, proxy controller sets an *OFPFF_SEND_FLOW_REM* flag on the newly added entry. When this entry has been idled for *partial_timeout*, it will send a remove message to proxy controller. Proxy controller records this time and sets this entry again until the original *idle_timeout* value is reached. Therefore, when a switch sends remove messages for one entry for *n* times, it means that the entry is idled for *n*×*partial_timeout* seconds.

With such solution, we can get the approximated idle time of each flow entry. If the flow entry is matched, the openflow switch should extend its lifetime with another *idle_timeout*. However, because the lifetime of the entry is now reduced to *partial_timeout* instead of *idle_timeout*, the flow entry could be removed before its real lifetime. We call this life cycle problem and describe our solution as followed:

Figure 10 illustrates the problem of incorrect entry life cycle with proposed idle time calculation. If one entry had been matched before it sends the remove message, we only reset the entry with *partial_timeout* which might reduce entry life cycle. We assume that *idle_timeout*=20 *and partial_timeout*=10. As shown in the figure, if the entry is matched at time 12, the next timeout will be 12 + 10 (*partial_timeout*) = 22 instead of 12 + 20 (*idle_timeout*) = 32. In order to solve this problem, we propose a solution that if proxy controller receives remove message from a timeout entry, it calculates *now_time – last update time*. If this value is larger than *partial_timeout*, it represents that this entry had been matched and its idle time had been reset as 0. Otherwise, we increase the timeout count by 1 and calculate the new idle time. In the same example, when controller is notified with the timeout entry on time 22, it calculates 22 (*now_time) –* 10 (*last update time*) = 12, which is greater than *partial_timeout* 10. Thus, the controller knows that the entry had been matched and its idle time had been reset as 0. Finally, the the flow entry is reset again..

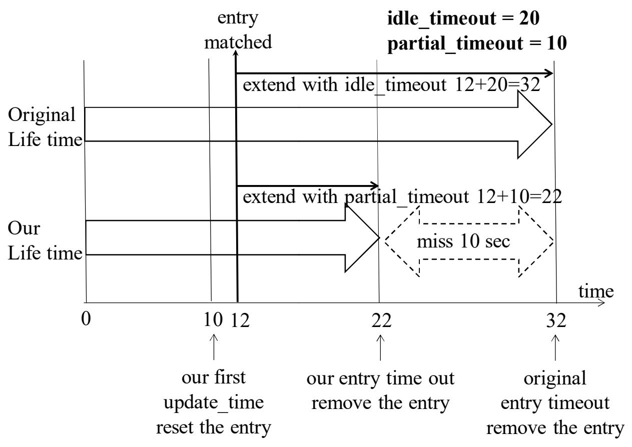
entry matched
idle_timeout = 20
partial_timeout = 10
extend with idle_timeout 12+20=32
Original Life time
extend with partial_timeout 12+10=22
Our Life time
miss 10 sec
time
0
10 12
22
32
our first update_time reset the entry
our entry time out remove the entry
original entry timeout remove the entry

Fig. 10. Life Cycle problem

Finally, LRU-PTR chooses the entry with the largest idle time as the victim entry. Figure 11 shows the pseudo code of LRU-PTR algorithm. With LRU-PTR, we do not need to ask switches for idle status of each entry every time. Each flow entry in the switch could send remove message to proxy controller to estimate the approximate idle time automatically. It efficiently improves the efficiency of selecting victim entries for replacement.

Algorithm 1 LRU-PTR solution

```
1:  if $NowTime - UpdateTime > PartialTimeout$ then
2:      $UpdateTime = NowTime$
3:      $IDLE = PartialTimeout$
4:      $Resetentry()$
5:  else
6:      $UpdateTime = NowTime$
7:      $RemoveTimes = RemoveTimes + 1$
8:      $IDLE = IDLE + PartialTimeout$
9:      if $RemoveTimes * PartialTimeout > IdleTimeout$ then
10:         $Removeentry()$
11:     else
12:         $Resetentry()$
13:     end if
14: end if
15: $SELECT(MAX(IDLE))$
```

Fig. 11. Pseudo Code of LRU-PTR Algorithm

D. Usage Range: Overflow Ratio

As discussed in previous sections, an upper bound is set to limit the number of flow entries used by a tenant. This upper bound is derived from another parameter, overflow ratio. Basically, we want to allow tenants to use more flow entries than their quota to increase the flow table utilization. However, without the limitation of an upper bound, some greedy tenant will occupy most of flow entries, causing that switches enter in-line mode easily and suffered from long flow modification latency.

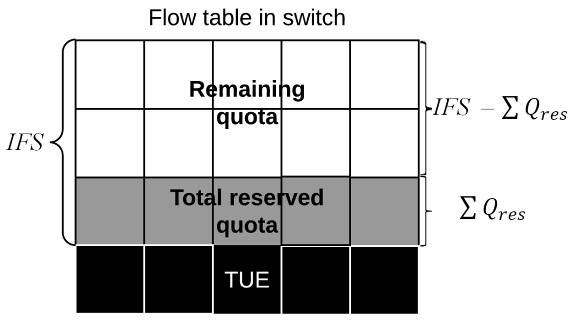

Fig. 12. Flow table allocation

Figure 12 shows the flow table allocation in a SDN switch. The black area represents the entries that already used by tenants (TUE) and the rest are the idle entries (IFS). To effectively allocate the IFS among the tenants, we firstly reserve part of IFS for each tenant according to their past average usage, which is called reserved quota. The total reserved quota is the summation of every tenant's reserved quota denoted as$\sum Q_{res}$ and depicted as grey area in Fig.12. Finally, we could locate the size of remaining quota with $IFS - \sum Q_{res}$, which is the white area in Fig. 12. The remaining quota could be allocated for calculating overflow ratio among the tenants.

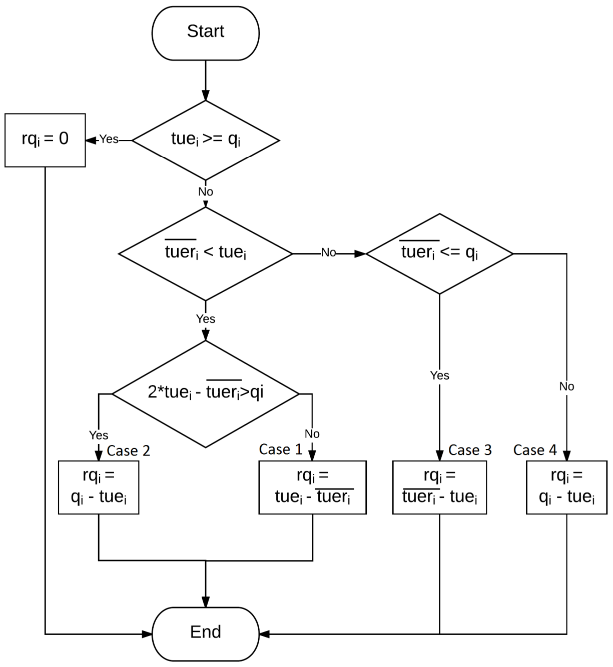
Start
$tue_i >= q_i$
Yes
$rq_i = 0$
No
$\overline{tuer_i} < tue_i$
No
$\overline{tuer_i} <= q_i$
Yes
$2*tue_i - \overline{tuer_i} > q_i$
Yes
No
Yes
No
Case 2
Case 1
Case 3
Case 4
$rq_i = q_i - tue_i$
$rq_i = tue_i - \overline{tuer_i}$
$rq_i = \overline{tuer_i} - tue_i$
$rq_i = q_i - tue_i$
End

Fig. 13. Calculation of Reserved Quota for Tenant

Figure 13 shows how to calculate reserved quota *rq* of each tenant. Firstly, if the tenant already overuse his quota ($tue_i >= q_i$), we will not reserve more quota for him by setting $rq_i=0$. Next, we calculate rq_i with q_i and tue_i in different cases.

- Case 1: If tenant's current usage is more than his past average usage $(\overline{tuer_i} < tue_i)$, it means that the tenant is going to submit more flow entries than past. Hence, we assume that the tenant will request another amount of $tue_i - \overline{tuer_i}$ in the future and allocated $rq_i = tue_i - \overline{tuer_i}$ for him.
- Case 2: In Case 1, we would like to confirm that the total allocated quota for the tenant ($tue_i + rq_i = tue_i + (tue_i - \overline{tuer_i}) = 2 \times tue_i - \overline{tuer_i}$) does not exceed his quota q_i. Therefore, if $2 \times tue_i - \overline{tuer_i} > q_i$, we could only allocate up to q_i for the tenant so that $rq_i = q_i - tue_i$.
- Case 3: If tenant's current usage is no more than his past average usage $(\overline{tuer_i} \geq tue_i)$, we would like to reserve $rq_i = \overline{tuer_i} - tue_i$ for the tenant so that his quota could reach up to his past average usage.
- Case 4: In Case 3, if the tenant's past average $\overline{tuer_i}$ exceeds his quota q_i, we could only allocate up to q_i for the tenant so that $rq_i = q_i - tue_i$.

After we determine each tenant's reserved quota rq_i, we could calculate the total reserved quota (the grey area in Fig. 12.) as

$$\sum Q_{res} = \sum_{i=1}^{|T|} (rq_i) \cdot \tag{2}$$

After we reserved some quota to other tenants, then we determine $IFS - \sum Q_{res}$ as the remaining quota that can be allocated (the white area in Fig. 12.). The following three policies are discussed for allocating remaining quota to the tenant.

1) *Total Overflow*

Total overflow is a simple allocation method. We expect no more tenant wants to use the remaining quota. In this policy, we allocate all remaining quota among the tenants who want to overflow its quota and the formula is given as

$$ovr_i = \frac{(IFS - \sum Q_{res})}{\#\text{ of Tenants with overflow request}} \times 100\% \cdot \tag{3}$$

2) *Equal Overflow*

In Equal overflow policy, we expect that some tenants still want to use the remaining quota and avoid huge remaining quota being allocated to few tenants. So the remaining quota is equally allocated among all tenants and the calculation of overflow ratio is

$$ovr_i = \frac{(IFS - \sum Q_{res})}{|T|} \times 100\% \cdot \tag{4}$$

3) *Weighted Overflow*

The weighted overflow is adopted when the tenants are assigned with different quotas. In this policy, we allocate the remaining quota to tenants according to the ratio of each tenant's quota. Hence, those tenants with larger quota will be allowed with larger overflow ratio and the formula is calculated as

$$ovr_i = (IFS - \sum Q_{res}) \times \frac{q_i}{TFS} \times 100\% \ . \qquad (5)$$

Figure 14 depicts an example of calculating overflow ratios. Assume there has 5 tenants T={A, B, C, D, E}, Q={5, 3, 4, 4, 4}, TFS = 20, TR sequence = {A, A, A, A, B, A, B, A}and average *tuer* of tenant B is 1. When the flow table is allocated as shown in figure 14, there is only one tenant A wants to overuse the remaining quota. We could calculate *rq* of each tenant, Q_{res}={0, 1, 0, 0, 0}. Then $\sum Q_{res}$=1, so IFS - $\sum Q_{res}$=12 for the remaining quota. Finally, we could calculate *ovr* of tenant A according to the formula of each overflow ratio policy. If the flow table utilization goes up, the *ovr* of the same tenant will drop down. When *ovr* drops down, tenant could not overuse the remaining quota and has to release its own flow entries for new requests.

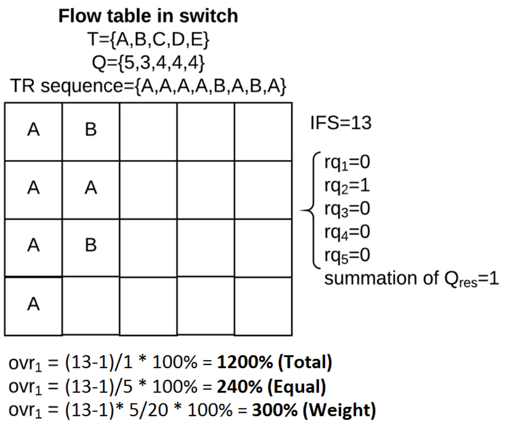
Flow table in switch
T={A,B,C,D,E}
Q={5,3,4,4,4}
TR sequence={A,A,A,A,B,A,B,A}
A
B
A
A
A
B
A
IFS=13
rq1=0
rq2=1
rq3=0
rq4=0
rq5=0
summation of Qres=1
ovr1 = (13-1)/1 * 100% = 1200% (Total)
ovr1 = (13-1)/5 * 100% = 240% (Equal)
ovr1 = (13-1)* 5/20 * 100% = 300% (Weight)

Fig. 14. Overflow Ratio Example

V. Experiments and Results

In this section, we describe our experiment environment and experimental results.

A. Implementation

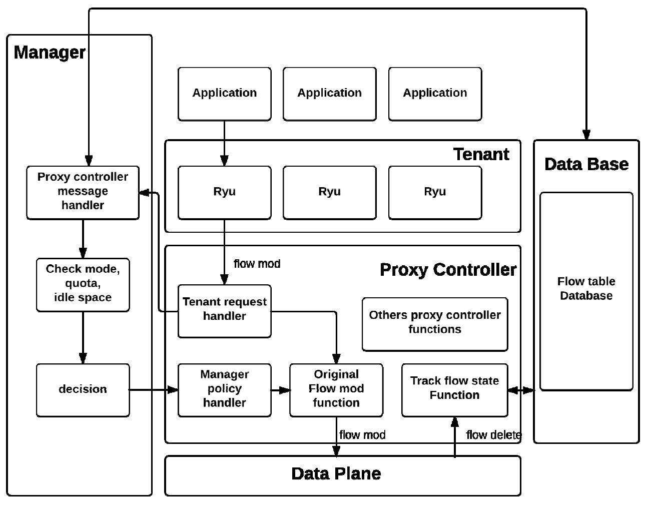

Fig. 15. System Architecture

Figure 15 shows the system architecture of the experiments. We implement the system based on OpenFlow 1.0 [13]. Ryu [22] is deployed as controllers for tenants. We choose OpenVirteX [6] as proxy controller and modify its functions for the communication with SPRM. We also add features that deal with management plane policies and idle timeout messages for tracking the flow states. The key functions of SPRM such as the replacement algorithm and overflow ratio calculation could be implemented on a remote server to reduce proxy controller's overhead.

When a tenant application sends a flow modify message to proxy controller, the tenant request handler forwards the request to SPRM through MQTT [23] protocol. SPRM then uses JSON-RPC API of OpenVirteX to get tenant's information. After checking tenant state and making decisions of replaced entries, SPRM sends the policies back to proxy controller through MQTT. Finally, OpenVirteX communicates with switches according to the policies.

B. Environment Setup

There are two servers equipped with Intel core i5-4440 CPU 3.10GHz and 24GB DRAM installed with VMWare Workstation for the emulation environment. Several virtual machines are allocated with two core processors and 2GB DRAM on these servers. The first server emulates 5 virtual machines as tenant controllers. Another server emulates the system elements with 3 virtual machines, including SPRM management plane, OpenVirteX as proxy controller, and MQTT broker for the communication between OpenVirteX and SPRM. Finally, we use Mininet [24] to emulate the physical network and combine it with the elements described above.

TABLE IV
System Default Settings

Categories	Field	Default Values
Tenant	$\|T\|$	5
	Q	[100, 200, 400, 200, 100]
Switch	*TFS*	1000
Ryu application	*Flow entry idle_timeout*	30 s
SPRM	*LRU-PTR partial_timeout*	10 s
	Overflow ratio policy	Total overflow
Traffic generator: Ostinato	*Source IP*	10.0.0.1 ~ 10.0.0.5 (μ=3, σ=0.6)
	Destination IP	10.0.0.6 ~ 10.0.3.255 (μ=512, σ=300)
	Number of Connections	40000
	Inter-arrival time between connections	5 ms
	Packet Rate	1000 packets/s for 50 packets

The default settings and configurations of our test cases are shown in Table IV. The number of tenant $|T|$ is 5 and the quota set Q for these tenants is [100, 200, 400, 200, 100]. The flow table size in a switch *TFS* is 100. The idle timeout for each new flow entry is 30 seconds. We set *partial_timeout* as 10 seconds and select total overflow policy as default policy. To dynamically create connections, Ostinato [25] is adopted as packet generator. We use normal distribution to randomly generate source and destination IPs. For traffic source, there are 5 source IPs ranges from 10.0.0.1 to 10.0.0.5 and we set the mean μ to 3 and standard deviation σ to 0.6 as default. The range of destination IP is from 10.0.0.6 to 10.0.3.255, the default value of μ is 512 and σ is 300. Total 40,000 connections are generated and there is a 5 ms delay between connections. Each connection transfers 50 packets with rate 1,000 packet/second, which lasts 50/1,000 = 0.05s = 50ms. Hence, our total simulation time = 40,000 connection * 50ms + 39,999 intervals * 5ms = 2,199,995ms = 36.67min. In order to observe flow modification latency, Cbench [26] is used to compare the overhead.

C. *Experiment Results*

In this section, we provide the experiment results to observe the values of *SAR* and *PKI*. We compare *SAR* and *PKI* among OVX, hard partitioning, and SPRM. For SPRM, we perform the simulations with different overflow ratio policies for comparison. Finally, the performance of flow modification latencies is discussed.

1) *Comparison among OpenVirteX, Hard Partitioning and SPRM*

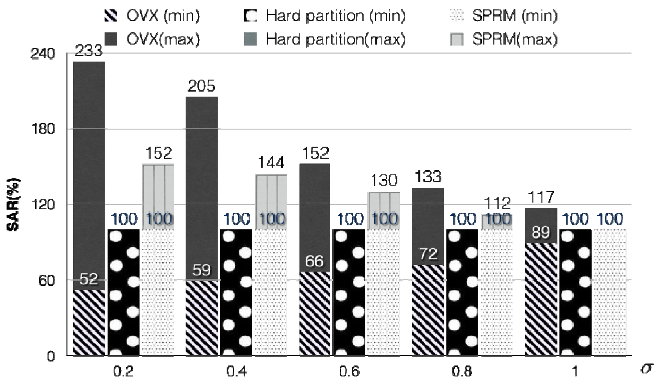

Fig. 16. Results of Satisfaction Ratio

Figure 16 shows maximum and minimum *SAR* of tenants among OpenVerteX without flow table management functions (denoted as OVX), hard partitioning, and proposed SPRM. The x-axis represents the standard deviation σ , of normal distribution. With smaller σ, the distribution of TR is more uneven and most of the traffic requests come from specific tenants. As a result, the resource monopoly problem is more serious. Hence, we can investigate the performance of SAR under different distribution patterns of source and destination IPs by assigning different values of σ. We can observe that SPRM outperforms OVX because the minimum *SAR* of SPRM can reach 100% while the minimum *SAR* of OVX is only 52% when σ=0.2. That means, under the situation with few greedy tenants, these greedy tenants dominate the resources with OVX easily. SPRM, on the other hand, limits these greedy tenants by calculating overflow ratio and could fairly allocate the resources among tenants. From the results we can assure that SPRM could eliminate the resource monopoly problem effectively. Although hard partitioning strategy could deal with this issue, it lacks of the flexibility of overflow mechanism, so that the maximum of *SAR* cannot be higher than 100%.

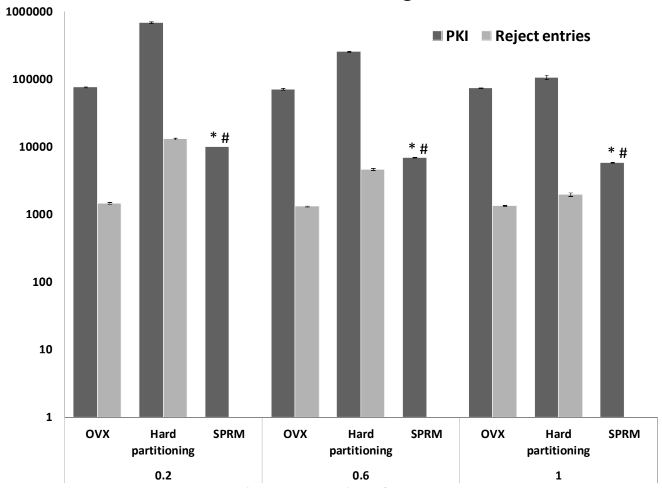

Fig. 17. Results of PKI

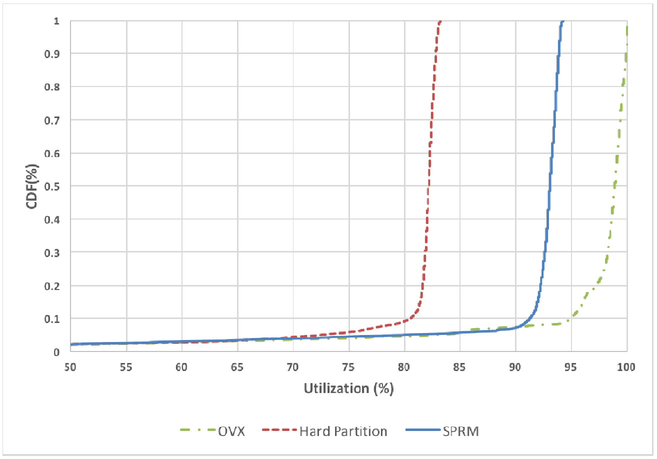

Fig. 18. Result of Flow Table Utilization

Figure 17 presents the *PKI* for each algorithm. We run 10 tests for each case. Standard deviations of the cases are the calculated and depicted on the bars. In addition, we calculate probability values (p-values) for statistical measurement. * means p-value < 0.01 when comparing SPRM with OVS; # means p-value < 0.01 when comparing SPRM with hard partitioning strategy. We can see that both * and # are true on different cases which means that it is improbable for PKI performances of both OVX and hard partitioning to match SPRM. From the figure, we can observe that the performance of SPRM is better than OVX and hard partitioning. Because quota is the upper bound in hard partitioning strategy, those tenants who want to request new flow entries over their quota will be rejected. On the other hand, there is no quota limitation in OVX and the requests will be rejected when flow table utilization reaches 100%. In SDN networks, these rejected entries will cause large *PKI* that enlarges controller's burden. In

SPRM, we design the replacement algorithm to reduce the rejection rate. From the results, we learn that SPRM reduces 95% of *PKI* and 100% of rejected entries.

We illustrate the utilization of flow table with CDF as y-axis in figure 18. Hard partitioning could only reach up to 82.5% because quota cannot be shared among tenants. OVX can achieve 100% utilization because there is no flow table management mechanism and every tenant can acquire flow entries if available. The proposed SPRM reaches 93% utilization which is much better than hard partitioning and close to OVX. However, since OVX suffers from large amount of rejections and *PKI*, we think SPRM is the best choice among these strategies.

2) *Comparison among different policies of SPRM*

We have learned that SPRM outperforms both OpenVirteX and hard partitioning on both *SAR* and *PKI*. In this part, we would like to analyze the performance among several SPRM policies and find the best setting of SPRM.

In this part, we use *partial_timeout* value as x-axis to find the best LRU-PTR configuration. In addition, we also collocate with each overflow ratio policies to observe the performance. Figure 19 depicts the minimum and maximum *SAR* of all tenants under different SPRM policies. Obviously, minimum *SAR* of each policy reaches 100%. It means that there will be no resource monopoly problem with SPRM. We also observe that the maximum *SAR* of policies with total overflow performs better than the others. The reason is that total overflow allocates more flow table size only to the tenant who wants to overuse. As a comparison, equal overflow allocates flow table for the other tenants without requesting. Because of the characteristic, *SAR* in equal overflow is less than total overflow and weight overflow. In this figure, we could also discover that there has no correlation between *partial_timeout* value and SAR performance because *partial_timeout* might not affect the tenant's usage range.

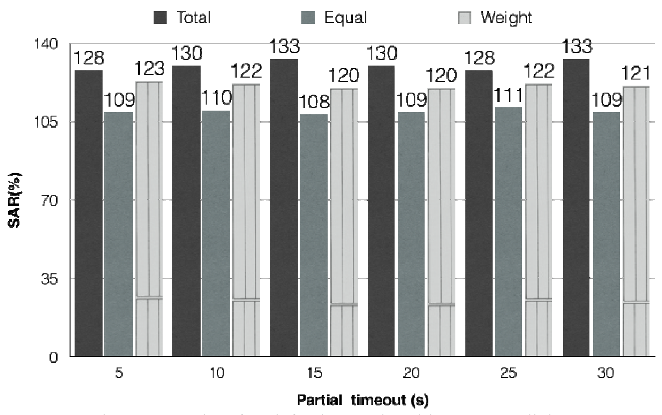

Fig. 19. Results of Satisfaction Ratio with SPRM Policies

Figure 20 depicts the *PKI* performance under different policies. We run 10 tests for each case. Standard deviations of the cases are the calculated and depicted on each dot. Again, we calculate p-values for statistical measurement. * means p-value < 0.05 when comparing total overflow policy with weighted overflow policy; # means p-value < 0.05 when comparing total overflow policy with equal overflow policy. We can see that # is true for all cases, which means that it is improbable for equal overflow policy to match total overflow policy. * is false when *partial_timeout* is 15 and 20, which means it is probable for weighted overflow policy to match total overflow policy on these 2 cases. For *PKI*, *partial_timeout* affects the LRU-PTR performance apparently. When we use smaller *partial_timeout*, it will generate more flow entry reset events which cause controller and switch processing overhead. On the other hand, if we set a larger *partial_timeout*, the idle time we calculate for each entry would not be approximated to the real idle time. Therefore, we want to find a best *partial_timeout* value and overflow ratio policy. From the result, *partial_timeout*=15 in our scenario has the minimal PKI. If we set it to 10 seconds, the *PKI* is raised because of the entry gap. On the other hand, if we set a larger *partial_timeout*, *PKI* is also raised because the estimated idle time is not close to real idle time. Figure 20 also shows that total overflow policy is better than equal and weighted overflow policies. The reason is that there are more flow spaces reserved for tenants with lower usage in both equal and weighted overflow policies. Therefore, few flow spaces are reserved for high-demand tenants, resulting in poor performance.

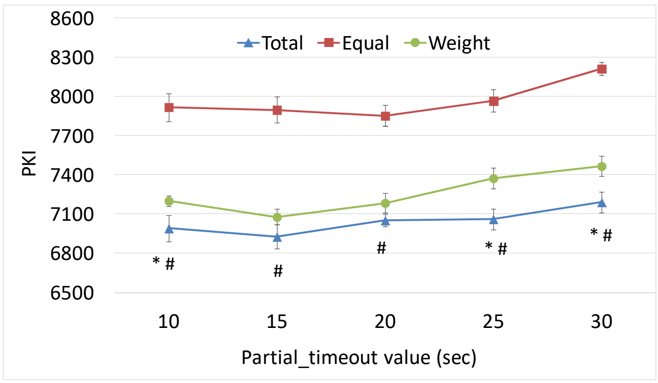

Fig. 20. Results of PKI with SPRM Policies

3) *Flow Modification Latency*

From previous results, we have showed that SPRM outperforms OVX and hard partitioning. However, proxy controller architecture adopted in SPRM may cause higher flow modification latency. In this part, we illustrate the result regarding to flow modification latency and explain how overflow ratio affects it.

From figure 21 we can see that OpenVirteX proxy controller raises the latency with 0.5 ms. If we adopt flow table management with in-line mode, the overhead increases 3 times higher. On the other hand, the latency with sniff mode is very close to OpenVirteX, and the performance of hybrid mode equals to sniff mode. Because SPRM with overflow ratio pre-reserves flow entries for potential tenants and hence our hybrid mode will stays in sniff mode, which could be observed in figure 22.

We can observe how many flow requests might be processed in in-line mode with different policies in figure 22. There are 30% requests of LRU-PTR without overflow ratio processed in in-line mode. If we deploy overflow ratio policies, these requests would be scattered into situations that switches have enough flow table space and all requests are processed in sniff mode. This is because that if there is no upper bound limited by overflow ratio, the utilization of flow table reaches 100% easily with heavy tenants. On the other hand, the overflow ratio reserves spaces to tenant with small traffics and the switch can work in sniff mode at all times.

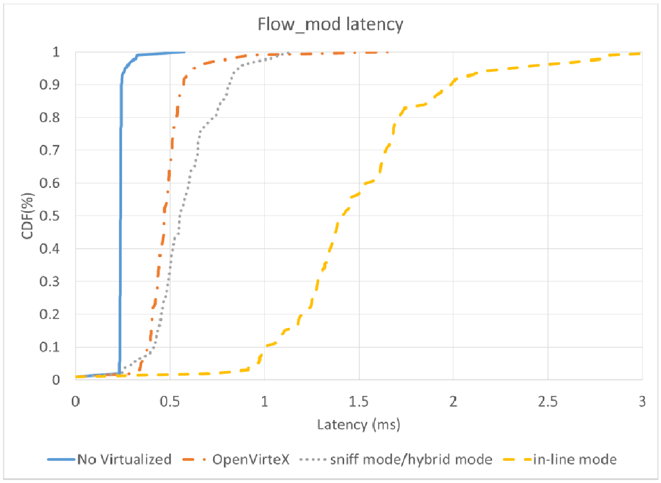

Fig. 21. Results of Flow Modification Latency

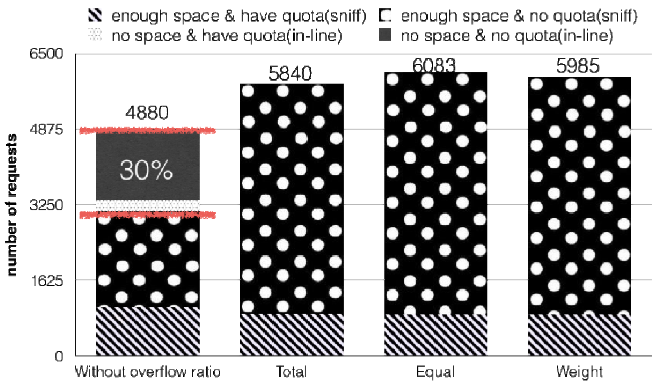

Fig. 22. Occupancy of four Situations

VI. CONCLUSIONS AND FUTURE WORK

In this paper, we proposed SPRM for managing the flow table resource in a multi-tenancy SDN environments. SPRM inherits the concept of soft partitioning to maintain high flow entry utilization. To avoid a resource monopoly problem and speedup flow modification latency, SPRM extra adopts three novel concepts: *hybrid mode*, *usage range,* and *LRU-PTR replacement*. We also implemented SPRM in a managing server and modified OpenVirteX as a proxy controller, which cooperates with SPRM.

The experiment results show that OpenVirteX without management plane suffers from the resource monopoly problem and not all tenants are satisfied. However, hard partitioning strategy causes a lot of *Packet_in* events and increases controller's overhead. From our experiment results, we could choose the total overflow policy with SPRM in order to reach better performance. In addition, the minimum of *SAR* with SPRM could reach 100%, which performs 48% better than OpenVirteX without management. SPRM also saves 100% rejections and reduces 95% *PKI* messages as compared with hard partitioning strategy. As to flow table utilization, we could observe that SPRM could reach up to 93%, which is higher than 82.5% of hard partitioning strategy and solves the resource wasting problem. Hence, SPRM not only solves these two problems but also reduces at least 30% in-line mode requests to avoid high flow modification latency by using overflow ratio.

In a multi-tenancy environment, some implementations of proxy controller could provide big switch feature, which represents that multiple physical switches are mapped as a single virtual switch. How to manage a big switch is a great challenge because a single flow entry on the virtual switch will be mapped to multiple flow entries on multiple physical switches. We believe that there are more issues to study with big switch feature enabled in the future.

REFERENCES

[1] S. Sezer, S. Scott-Hayward, P.-K. Chouhan, B. Fraser, D. Lake, J. Finnegan, N. Viljoen, M. Miller and N. Rao, "Are we ready for SDN? Implementation challenges for software-defined networks," *IEEE Communications Magazine*, vol. *51*, no. *7*, pp. *36-43*, July 2013.

[2] A. Lara, A. Kolasani, B. Ramamurthy, "Network innovation using openflow: A survey," *IEEE Communications Surveys & Tutorials*, vol. 16, no 1, 2014, pp. *493-512*.

[3] A. Blenk, A. Basta, M. Reisslein, W. Kellerer, "Survey on Network Virtualization Hypervisors for Software Defined Networking," *IEEE Communications Surveys & Tutorials*, vol. 18, no 1, 2016, pp. *655-685*.

[4] "OpenDaylight VTN Gerrit Project," [Online]. Available: https://github.com/opendaylight/vtn.

[5] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown and G. Parulkar, "Flowvisor: A network virtualization layer," *OpenFlow Switch Consortium*, Tech. Rep, pp. *1-13*, 2009.

[6] Al-Shabibi, Ali, D. Leenheer, Marc, Gerola and Matteo, "OpenVirteX: Make your virtual SDNs programmable," *Proceedings of the third workshop on Hot topics in software defined networking*, 2014, pp. *25-30*.

[7] X. Jin, J. Gossels, J. Rexford and D. Walker, "Covisor: A compositional hypervisor for software-defined networks," *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, 2015, pp. *87-101*.

[8] E. Salvadori, R. Doriguzzi Corin, A. Broglio, M. Gerola, "Generalizing Virtual Network Topologies in OpenFlow-Based Networks," *2011 IEEE Global Telecommunications Conference (GLOBECOM 2011)*, Kathmandu, 2011, pp. 1-6.

[9] S. Gebert, M. Jarschel, S. Herrnleben, T. Zinner, P. Tran-Gia, "Table Visor: An Emulation Layer for Multi-table Open Flow Switches," *2015 Fourth European Workshop on Software Defined Networks*, Bilbao, 2015, pp. 117-118.

[10] A. Ishimori, F. Farias, E. Cerqueira and A. Abelem, "Control of multiple packet schedulers for improving QoS on OpenFlow/SDN networking," *Second European Workshop on Software Defined Networks (EWSDN)*, 2013, pp. *81-86*.

[11] P. Qin, B. Dai, B. Huang and G. Xu, "Bandwidth-aware scheduling with sdn in hadoop: A new trend for big data," *IEEE Systems Journal*, pp. *1-8*, March 2014.

[12] J.-L. Chen, Y.-W. Ma, H.-Y. Kuo and W.-C. Hung, "Enterprise visor: A Software-Defined enterprise network resource management engine," *IEEE/SICE International Symposium on System Integration (SII)*, 2014, pp. *381-384*.

[13] "OpenFlow Switch Specification version 1.0.0," [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.0.0.pdf.

[14] R. Jain and S. Paul, "Network virtualization and software defined networking for cloud computing: a survey," *IEEE Communications Magazine*, vol. *50*, no. *14*, pp. *24-31*, November 2013.
[15] "ONF SDN Architecture Issue 1.1," [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/TR-521_SDN_Architecture_issue_1.1.pdf.
[16] J. Kohn, D. Rowell, F. Shi and G. Aybay, "Managing a Flow Table". U.S. Patent 9065724, Jun. 23, 2015.
[17] D. Kim and B.-D. Lee, "An Efficient Flow Table Management Scheme for SDNs Based On Flow Forwarding Paths," *Advanced Science and Technology Letters*, vol.*135*, pp.*88-93*, 2016.
[18] Dan, Asit, Towsley and Don, "An approximate analysis of the LRU and FIFO buffer replacement schemes," *ACM SIGMETRICS conference on Measurement and modeling of computer systems*, 1990, pp. *143-152*.
[19] A. X. Liu, C. R. Meiners, E. Torng, "TCAM Razor: A Systematic Approach Towards Minimizing Packet Classifiers in TCAMs," *2007 IEEE International Conference on Network Protocols*, Beijing, 2007, pp. 266-275.
[20] C. Yu, C. Lumezanu, H. V. Madhyastha, and G. Jiang, "Characterizing Rule Compression Mechanisms in Software-defined Networks," *17th International Conference on Passive and Active Measurement (PAM2016)*, Greece, 2016.
[21] M. Rifai, N. Huin, C. Caillouet, F. Giroire, D. Lopez-Pacheco, J. Moulierac, and G. Urvoy-Kelle, "Too many SDN rules? Compress them with MINNIE," *2015 IEEE Global Telecommunications Conference (GLOBECOM 2015)*, San Diego, 2015.
[22] "Ryu," [Online]. Available: https://osrg.github.io/ryu/.
[23] "Wikipedia for MQTT protocol," [Online]. Available: https://en.wikipedia.org/wiki/MQTT.
[24] Mininet, [Online]. Available: http://www.mininet.org.
[25] Ostinato, [Online]. Available: http://ostinato.org/.
[26] R. Sherwood, Cbench Controller Benchmarker, [Online]. Available: http://www.openflow.org/wk/index.php/Oflops.
[27] R. C. S. Morling and G. D. Cain, "MININET: a packet-switching mini computer network for real-time instrumentation," *AIM International Meeting of Minicomputer and Data Communications*, Jan. 1975.

Ying-Dar Lin (F'13) is a Distinguished Professor of computer science at National Chiao Tung University (NCTU), Taiwan. He received his Ph.D. in computer science from the University of California at Los Angeles (UCLA) in 1993. He was a visiting scholar at Cisco Systems in San Jose, California, during 2007–2008. Since 2002, he has been the founder and director of Network Benchmarking Lab (NBL, www.nbl.org.tw), which reviews network products with real traffic and has been an approved test lab of the Open Networking Foundation (ONF) since July 2014. He also cofounded L7 Networks Inc. in 2002, which was later acquired by D-Link Corp. His research interests include network security and wireless communications. His work on multihop cellular was the first along this line, and has been cited over 650
times and standardized into IEEE 802.11s, IEEE 802.15.5, IEEE 802.16j, and 3GPP LTE-Advanced. He is also an IEEE Distinguished Lecturer (2014–2015) and an ONF Research Associate. He currently serves on the editorial boards of several IEEE journals and magazines. He published a textbook, Computer Networks: An Open Source Approach (www.mhhe.com/lin), with Ren-Hung Hwang and Fred Baker (McGraw-Hill, 2011).

Te-Lung Liu (M'99) received the B.S. and Ph.D. degrees in computer science from National Tsing Hua University, Hsinchu, Taiwan, R.O.C., in 1995 and 2002, respecively. He is currently a Research Scientist in National Center for High-Performance Computing, Tainan, Taiwan, R.O.C. He is also a Team Member of the Taiwan Advanced Research and Education Network (TWAREN) and Research Associate of Open Networking Foundation (ONF). Now He is working on SDN Testbed in Taiwan. His current research interests include Software Defined Networking, Future Internet, optical networks, and network design.

Jian-Hao Chen received the B.S. degree in computer science information engineering from National Chiayi University, Chiayi, Taiwan, in 2014 and the M.S. degree in computer science engineering from National Chiao Tung University, Hsinchu, Taiwan, in 2016. His research interest includes the development of Software Defined Networking and network virtualization technology. Mr. Chen also has a lot of developing experience of web application with SDN and real time web services.

Yuan-Cheng Lai received the Ph.D. degree in Computer Science from National Chiao Tung University, Taiwan, in 1997. In August 2001, he joined the faculty of the Department of Information Management at National Taiwan University of Science and Technology, Taiwan, where he had been a professor since February 2008. His research interests include wireless networks, network performance evaluation, network security, and Internet applications.