Design and Implementation of Verification Based OpenFlow Hypervisor for Multi-Tenant Virtualized Network

Shun Higuchi
Graduate School of Computer and Information Science
Hosei University
Tokyo, Japan
Email: shun.higuchi.6j@stu.hosei.ac.jp

Toshio Hirotsu
Faculty of Computer and Information Science
Hosei University
Tokyo, Japan
Email: hirotsu@hosei.ac.jp

Abstract—**Cloud services that virtualize existing IT infrastructures in data centers are widely used by governments, universities, and companies. Multi-tenancy is an indispensable feature for data centers to provide a large number of isolated networks to different organizations. OpenFlow is a core technology of software defined networking (SDN) and is useful for centrally managing and controlling these networks; however, SDN is used only at the management level. It is desirable to make the flexible features of SDN/OpenFlow available to users' virtual networks. FlowVisor provides virtualized multi-tenant OpenFlow networks by coordinating multiple controllers, but it is unable to prevent conflicts among the control rules of individual virtual networks. Administrators of each tenant thus need to design the specifications of each virtual network carefully. In this paper, we propose a verification-based scheme for coordinating multiple tenants' OpenFlow networks. The scheme enables administrators to design their own virtual networks without considering conflicts with other tenants. A flow space manager manages overlaps of the address spaces and resolves conflicts between rules of different tenants; in so doing, isolation is preserved transparently for each tenant.**

Keywords–OpenFlow; Virtualization; Multi-tenant Network.

I. Introduction

With the development of server virtualization technology, cloud computing services, such as Infrastructure as a Service (IaaS), have become popular. Here, an organization's IT infrastructure is consolidated in a data center by using server virtualization and is provided through the Internet. In multi-tenant networks, one physical network is divided into many virtual tenant networks. The traffic in each virtual network needs to be isolated from the traffic in other networks. Virtual LAN (VLAN) is a popular isolation technology. IaaS providers define many virtual layer 2 tenant networks by assigning VLAN-IDs to each tenant; the tenants ' administrators then construct their own layer 3 networks by using the VLANs. On an IaaS cloud using VLAN technology, the IaaS administrator needs to change the configuration of the VLAN on all related network devices in order to modify the virtual tenant networks. However, in such environments, demand is growing for an efficient means of changing, i.e., creating, modifying and destroying, virtual networks; what is required is a more flexible virtual network construction and management method.

OpenFlow [3], which is a core technology of software-defined networking (SDN) [2], enables flexible routing control and centralized management of networks by separating the control plane from the data transfer plane. A controller defines the routing of packet forwarding, and the data plane switches transfer packets in accordance with the instructions of the controller. Since this technology has the ability to recognize and rewrite the VLAN-ID of each packet, IaaS providers can aggregate VLAN management functionalities into one controller. The OpenFlow based network architecture enables flexible virtual network management for IaaS providers; however, a tenant network is not allowed to use OpenFlow functionalities to avoid confusion with cloud management level controls. This means administrators of tenant networks can not gain the benefits of OpenFlow even if the provider uses OpenFlow technology to manage its IaaS platform. In contrast, effectively coordinating multiple OpenFlow networks would enable all tenants to manage their own virtual tenant networks by using OpenFlow on a physical network.

FlowVisor [4] is one idea for handling requests from multiple OpenFlow controllers. In FlowVisor, a proxy is placed between the OpenFlow controller and the switches, and it exchanges and manages each tenant's control messages sent between the controllers and switches. OpenFlow switches on physical networks work properly under the control of FlowVisor, which coordinates multiple tenant controllers. In FlowVisor, each isolated virtual network space is expressed as a flow space. The administrator of a flow space needs to regulate the OpenFlow controller to write flow entries under the restrictions of the allocated flow space definition. This mechanism can be used to construct a plurality of virtual OpenFlow networks, and it enables each tenant controller to control their virtual tenant networks individually; however, FlowVisor does not avoid overlaps between flow spaces. When using FlowVisor to isolate multiple OpenFlow tenant networks, each tenant network is expected to obey the flow space definition provided by the IaaS provider. This problem becomes more complicated because the flow specifications used in the networks are not always exclusive. In the case of monitoring one tenant's network from another management network, the flow specifications allocated to each network's flow space must overlap. The IaaS provider needs to define them very carefully so as not to cause unintended traffic control.

In this research, we propose a verification-based OpenFlow network virtualization based on OpenFlow hypervisor that enables the network to be freely designed by each tenant. To guarantee isolation of tenant networks, we introduce a conflict management system that uses verification of flow space definitions. When a conflict occurs, the management system detects it and resolves the conflict by rewriting a flow entry. Our approach verifies and manages overlapping parts of the flow spaces defined by individual tenants, detects conflicts between flow spaces and flow entries, and rewrites the entries to avoid conflicts in the FlowVisor. This paper describes the method of flow space verification and its implementation based on our proposed mechanisms [1]. Section II is an overview of OpenFlow/SDN technology. Section III explains the mechanism and problems of FlowVisor. Section IV outlines the proposed virtualization method based on flow entry verification, and Section V describes the method for avoiding flow entry conflicts in more detail. Section VI describes our prototype implementation and its performance evaluation. Section VII discusses our method in relation with other research. Section VIII is a conclusion that mentions future work.

II. OpenFlow/SDN

OpenFlow is a representative architecture of software-defined networking, and it is currently being standardized. It is a next-generation network technology for cloud computing environments. An OpenFlow network consists of an OpenFlow controller responsible for routing control and an OpenFlow switch for transferring packets according to flow entries written by the controller. Hence, it is a centralized control architecture that enables centralized management of networks by separating the traditional network system into a control plane and data plane.

The controller is software, and a pair of matching fields, such as a MAC address, an IP address, a transport number, a VLAN-ID, and the actions to be performed on a packet are defined as a flow entry. Flexible routing control is enabled by transferring packets according to flow entries in the switch. If the switch has to be reconfigured in response to a change in the network configuration, the change is applied to all the switches by describing the change settings as new flow entries in the controller. This improves the manageability of the network. The controller and switch are connected by an OpenFlow channel, which is a control network using TCP/IP that is constructed separately from the data network, and they exchange control messages called OpenFlow messages through it. Through OpenFlow messages, the controller controls switches such as for writing the flow entry. In OpenFlow, since the controller controls all the switches and knows the network topology, it is possible for it to control routing flexibly such as through source routing and multi-path forwarding. Virtualizing a physical network by using OpenFlow makes it possible not only to improve the manageability of VLAN-IDs but also to ensure logical division of the network by using the packet headers of layers 1 to 4 that can be specified as a match field. OpenFlow enables its users to create a number of virtual networks beyond the usual limits of VLAN-IDs by dividing up the used address space in advance.

However, the conventional OpenFlow technology has some problems when it comes to virtualizing and controlling the OpenFlow network itself. For example, it is not possible to control each switch individually from multiple controllers in one OpenFlow network. Moreover, there is no mechanism to logically divide one OpenFlow network into multiple virtual OpenFlow networks. These problems make it impossible for a tenant to construct and control each controller or devise a virtual OpenFlow network in a multi-tenant data center that provides IaaS.

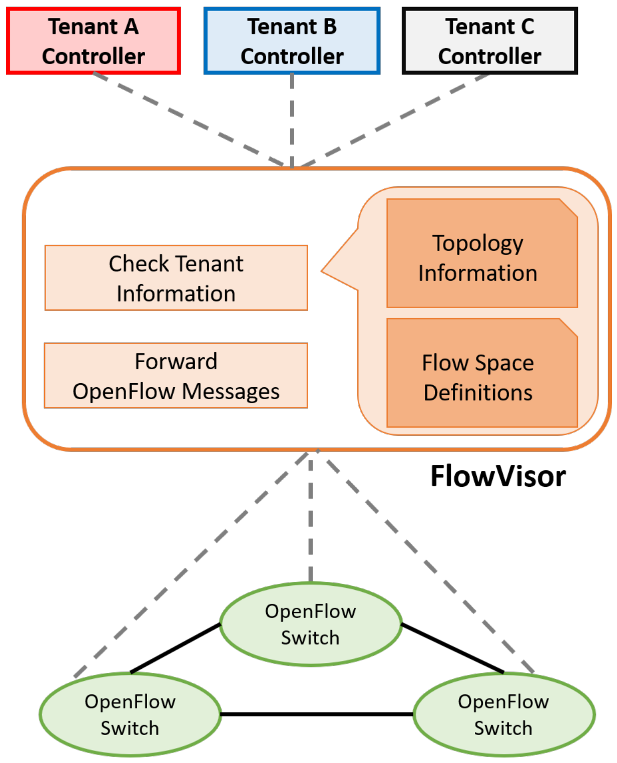


Figure 1. FlowVisor

III. FlowVisor

A FlowVisor is placed in the controller and switches, as shown in Figure 1. It operates as a proxy that transfers the OpenFlow messages necessary to control the switch from the controller. The administrator of FlowVisor defines the available network space to each tenant as a flow space and presents a flow space definition to each tenant user in some way. Tenant users create their own controllers that write flow entries in accordance with the network topologies and flow space definitions of the tenants' OpenFlow networks presented to them by the FlowVisor administrator. A tenant user can control the tenant network by connecting his/her controller to FlowVisor.

A. Flow Space

The administrator of FlowVisor defines a network space called a "flow space" that expresses the ranges of network parameters allowed to each tenant beforehand. Table I shows an example of a flow space. The flow space definition holds a slice name indicating the name of the tenant network, a DPID that indicates the OpenFlow switch ID, and a MAC address, IP address, transport number, etc., as an available match field from layer 1 to 4 in a flow entry and priority. In addition, each flow space is based on the premise that the defined network space is independent and has no overlaps. Therefore, there is no mechanism in FlowVisor for checking whether flow space conflicts exist; hence, the administrator needs to define each flow space carefully.

TABLE I. EXAMPLES OF FLOW SPACE

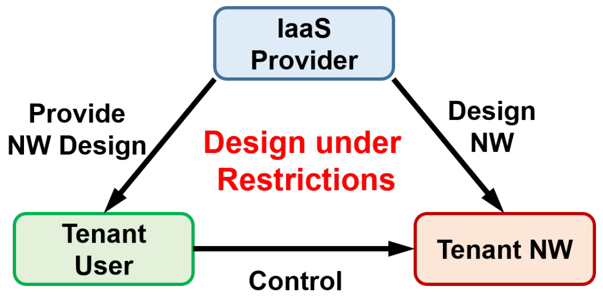
Slice	DPID	Priority	VLAN	Src MAC	Dst MAC	Src IP	Dst IP	Src TCP	Dst TCP
Tenant A	1	100	50	*	*	*	*	80, 22	*
Tenant B	1	100	50	*	*	10.0.1.0/24	*	80	*
Tenant C	1	100	50	*	*	10.0.2.0/24	*	80	*

B. FlowVisor Mechanism

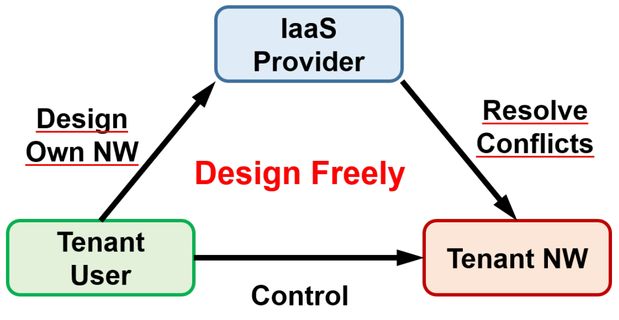
FlowVisor functions as a proxy on the OpenFlow channel and controls the transfer of OpenFlow messages between multiple controllers and switches. This function differs between the case of transferring messages from the switch to the controller, such as when sending Packet-In and Port-Status messages, and the case of forwarding messages from the controller to the switch, such as when sending the Flow-Mod message.

First, we describe the messages that are transferred from the switch to the controller. In this case, it is necessary to specify the controller to which the message pertains before transferring the message to it. As an example, a Port-Status message notifying that the physical port state of the switch has changed will affect all the tenant controllers using that port. Accordingly, FlowVisor searches for all target controllers from the topology information of each tenant network and transfers the Port-Status message to all of them. In the case of a Packet-In message, FlowVisor searches the flow space definition to specify which tenant network the packet belongs to and forwards the message to the tenant controller of the corresponding flow space.

Next, we describe the messages that are transferred from the controller to the switch. In this case, FlowVisor refers to the topology information of all the tenant networks; then it transfers the message to the target switch; it performs the same operation on every message. If a tenant user tries to send a message to a switch that does not belong to its own tenant network, the send operation fails and a message transfer error is returned to the controller.



(a) FlowVisor



(b) Ideal for IaaS

Figure 2. Network Design Policy

C. FlowVisor Problem

FlowVisor is based on the premises that the flow spaces allocated to each tenant network are independent and the tenant controller sets flow entries within the allocated flow space. If a FlowVisor administrator defines an unintended or incorrect flow space content, an unexpected network control will be executed. This problem can be discussed from a different viewpoint that the IaaS provider forces each tenant to restrict his/her network design, as shown in Figure 2(a). In contrast, if IaaS providers want to enable each tenant user to freely design their own tenant network, as shown in Figure 2(b), each tenant user should be able to define their flow space freely. This, however, leads to a problem that unintended traffic control can occur when a flow entry is written that conflicts with the flow space of another tenant. Hence, it is necessary to implement a mechanism that can check for conflicts in flow spaces and flow entries in a multi-tenant network.

Table I shows an example of conflicting flow entries, wherein tenant user A tries to write a flow entry that prohibits the SSH session such as by sending“*Src TCP = 22, action = DROP*“ to the switch with DPID = 1. In the table, the match fields of tenant A are defined as wildcard values “ * “ with the exception of Src TCP; thus, tenant user A can freely use this value. However, if a flow entry such as what is mentioned above is written, it will be applied to all packets transferred through this switch with the source TCP port number 22. Since all the packets are dropped, all SSH connections are closed, even in the other tenant networks. In this case, the packet was dropped unintentionally, however, it is possible to rewrite the packet header as a specified action and transfer it in OpenFlow. It is also possible to act in dubious or illegal ways, such as eavesdropping by transferring traffic of other tenants that are not permitted to use a server on their tenant network. In particular, it is possible to transfer the traffic of other tenants to a server on one’s own tenant network for the purpose of sniffing packets.

If a FlowVisor administrator allows each tenant user to freely design their tenant network and flow space definition, a flow space that has overlaps will cause unintended behaviors because the flow entries conflict. This is due to OpenFlow’s ability to flexibly set values such as wildcards about the L1-L4 headers in the match field. In the example mentioned above, since the tenant user can write a flow entry with wildcards other than the source TCP port number to the switch, s/he can control the traffic in unassigned flow spaces.

IV. Virtualization Based on Flow Entry Verification

We propose a new scheme to manage the virtualization of an OpenFlow network that allows each tenant to freely design his/her own network. It coordinates the controllers of the tenants' networks to make their designs work properly on a physical OpenFlow network (Figure 2(b)). The core tasks of the coordination are verification and conflict resolution. The management system verifies duplications of flow spaces assigned to each tenant, then resolves the conflict by rewriting the flow entries received from the tenant controllers. Figure 3 shows the verification process in OpenFlow Hypervisor. First, this system verifies and manages the overlapping address spaces in the owned flow space. A tenant administrator defines the combination of address spaces that is used in each tenant network as a "flow space". In addition, when a flow entry in a flow space includes address spaces overlapping those of other flow spaces, it checks for conflicts in the flow entries and rewrites the match field to guarantee separation of traffic of each tenant network. This minimizes the amount of rewriting of flow entries by applying verification and management to the flow space in advance.

The flow space of our system has a different definition from that of FlowVisor. Our definition of a flow space is a set of specific elements of OpenFlow match fields. In the existing work, there is no mechanism to check for violations of the flow space definition; thus, each tenant network is able to set arbitrary values for all match field elements. In this case, a tenant user controller may cause unintended traffic controls, such as using wildcards. On the other hand, our method restricts tenants to using only the range of match fields' values that each tenant defined as a flow space previously. If the tenant controller tries to set erroneous match fields in the flow entry, our system verifies and rewrites them to fit to the flow space definition. Our system also resolves the conflict ranges of the match fields by rewriting and writing back the values of the match fields.

A. Flow Space Definition

This flow space is different from the definition of FlowVisor in Section III-A. In previous work, a flow space was defined for each switch that the tenant can control; however, here, a new flow space is defined as a combination of address spaces that the tenant can use for one tenant network. A flow space is composed of multiple rules, where each rule consists of rule IDs, flow space names, and a matching field that is available to the tenant, as shown in Table II. In the matching field, it is possible to set five kinds of header information of L2 to L4, i.e., VLAN ID, Src/Dst IP address, and Src/Dst TCP port, which are necessary for network operations. These definitions are described in JSON format, as shown in Figure 4. Each flow space has a flow space name and a set of flow definitions. A flow definition is described for each element of a match field, and it is defined as a conjunction of fields. Since one flow space is represented by one or more flow definitions, multiple flow definitions are defined as disjunctions to allow flow entries that match any one. Each tenant uses only the combination of address spaces specified in this flow space. Definition example 2 in Table II, which summarizes the examples of Figure 4, shows the following address space:

- VLAN ID = 100, Src IP = 192.168.64.0/20, Dst IP = 192.168.64.0/20, Src TCP = 80
- VLAN ID = 101, Src IP = 192.168.64.0/20, Dst IP = 192.168.64.0/20, Src TCP = 80

The tenant assigned this flow space can control the network by using these two different combinations as a match field of the flow entry. The top row of Table II shows the available address space as the match field, but the upper limit of the VLAN ID is half the original limit of 4096. This is due to securing independent address space as management space for managing duplications of flow spaces and resolving conflicts in advance. VLAN-IDs are allocated from this management space to the flow space when necessary.

B. Duplicate Flow Space Verification and Flow Entry

Now let us explain the overlap verification between flow spaces and conflicts of flow entries on the basis of the definition in the previous section. Table III lists examples of flow spaces defined for three tenants A, B, and C. Since the flow space definition of tenant A on the top row completely includes the flow spaces of tenants B and C, the flow space of A overlaps those of B and C and is not independent. On the other hand, in the flow spaces of tenants B and C are independent in Table II, and independent values are specified for any of the match fields, such as Src IP address. Since only a combination of the address spaces is used as a match field in our flow space definition, we can detect duplications by verifying the inclusion relation for each combination of address spaces.

If the flow spaces have a complete inclusion relation, one must detect and avoid conflicts of flow entries after managing any flow space duplication. In flow spaces such as in Table III, the flow entry at the top of the Table IV written from tenant A's controller will collide with the flow entries of other tenants. Table IV shows two examples, i.e., one that conflicts with other flow space definitions and another that does not conflict with any other. In the example flow entry on the upper row, the value of Src IP is 192.168.64.0/20, and it is based on the flow space of tenant A. Since it includes the range of the flow spaces of the other tenants B and C, it conflicts with their flow entries, and their traffic is also controlled by this conflicting flow entry. On the other hand, in the example flow entry on the lower row, VLAN-ID = 101, which is an independent value from the flow space of the other tenants, is set in the match field. This flow entry does not cause a conflict. As mentioned above, we must verify the inclusion relation of the value specified in the match field for each flow space. If the value includes other tenant's flow spaces, it is possible to verify and avoid conflict by allocating a new value from the free independent address space and setting it in the conflicting flow entry.

V. Conflict Verification of Flow Entry

To avoid conflicts between flow entries, we propose a two-step conflict resolution method. The first step involves checking the consistency between the address space defined in the match field of the flow entry and its own flow space. In the second step, the OpenFlow hypervisor rewrites the match field of the conflicting flow entries that overlap with the address spaces in another tenant's flow space. Only switches under the common topology of the tenants' network are targets of this operation. Part of the match field of the flow entry is

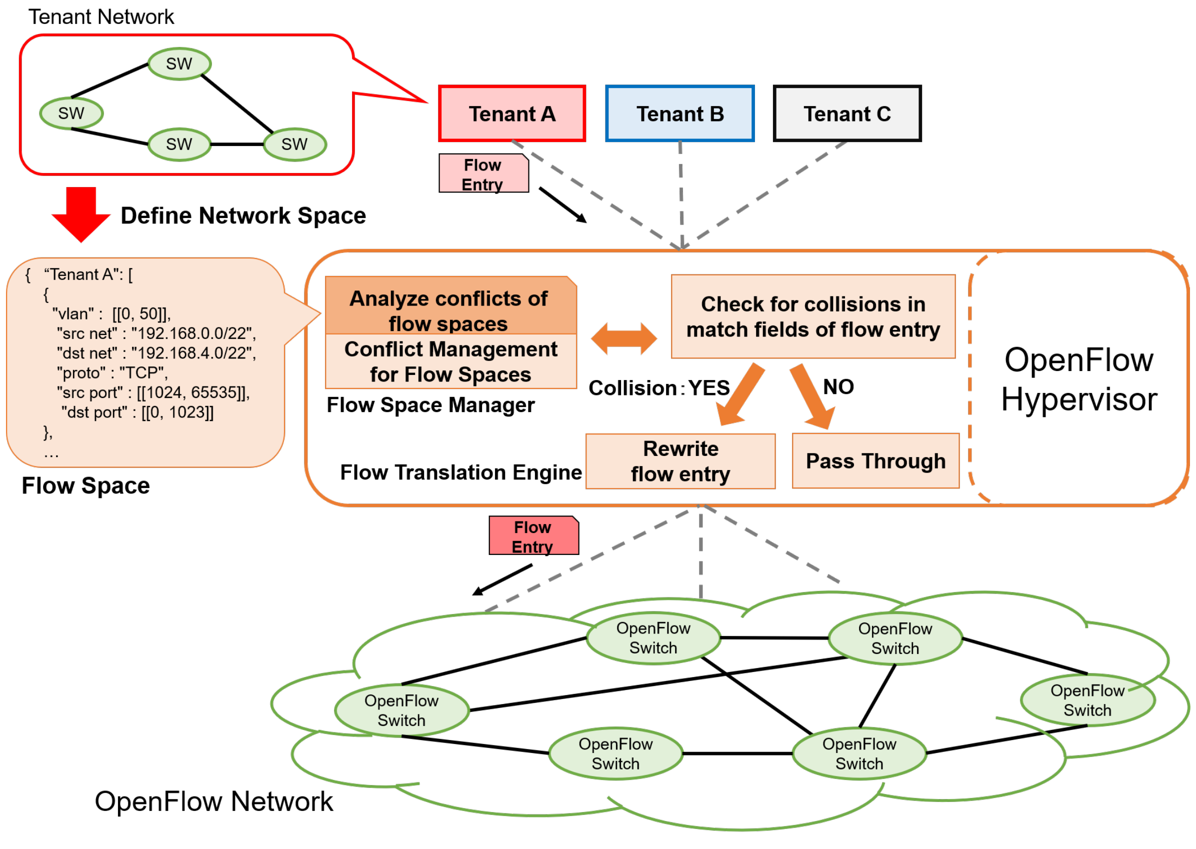


Figure 3. Proposed Architecture

TABLE II. FLOW SPACE LIMIT AND DEFINITION EXAMPLES

Rule ID	Space Name	VLAN	Src IP	Dst IP	Src TCP	Dst TCP
1	Maximum usage	0∼2047	0.0.0.0∼255.255.255.255	0.0.0.0∼255.255.255.255	0∼65535	0∼65535
2	Example 1	0∼50	192.168.0.0/22	192.168.4.0/22	1024∼65535	0∼1023
3	Example 2	0∼50	192.168.4.0/22	192.168.0.0/22	0∼1023	1024∼65535
4	Example 3	100, 101	192.168.64.0/20	192.168.64.0/20	80	*

TABLE III. EXAMPLES OF DUPLICATE FLOW SPACES

Rule ID	Space Name	VLAN	Src IP	Dst IP	Src TCP	Dst TCP
1	Tenant A	100, 101	192.168.64.0/20	192.168.64.0/20	80, 22	*
2	Tenant B	100	192.168.64.0/24	192.168.64.0/24	80	*
3	Tenant C	100	192.168.65.0/24	192.168.65.0/24	80	*

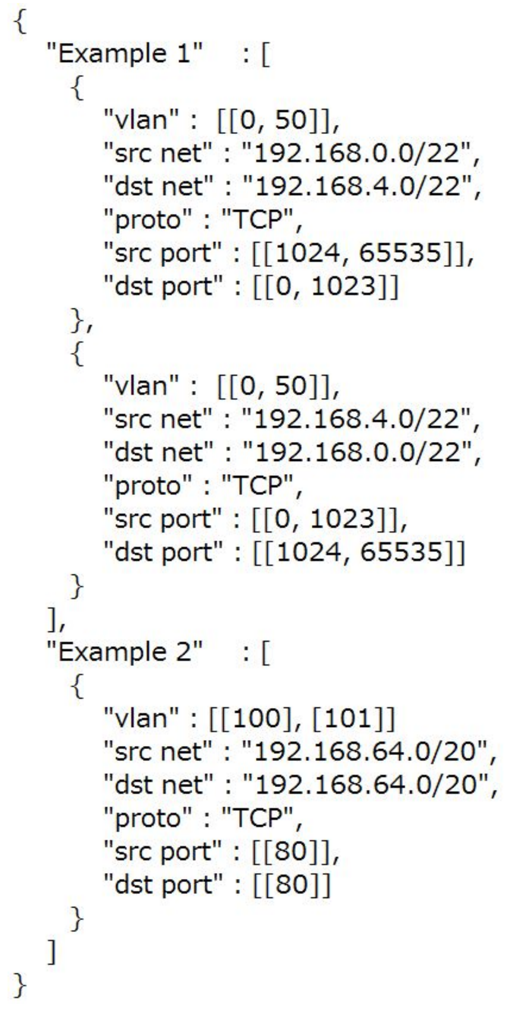
TABLE IV. EXAMPLES OF FLOW ENTRIES IN TABLE III

Entry	Match Field	Action
Conflicting Flow Entry	VLAN ID = 100 Src IP = 192.168.64.0/24 Dst IP = 192.168.64.0/24 Src TCP = 80	Output: port 2
Non-Conflicting Flow Entry	VLAN ID = 101 Src IP = 192.168.64.0/24 Dst IP = 192.168.64.0/24 Src TCP = 80	Output: port 2

converted into an independent value by using an address space not used by other tenants. At this time, the rewriting method will vary depending on the type of overlap in the topology. As a result, conflicts due to flow entries are automatically detected and avoided, while at the same time, different flow entries are prohibited in the defined flow space. These measures guarantee that the traffic of different tenant networks stays separated.

A. Consistency Check with Flow Spaces

When the OpenFlow hypervisor receives a Flow-Mod message from a tenant controller, it check whether the match field included in the message deviates from the tenant's flow space definition. To do so, it simply checks the range of the address space for values other than wildcards in the match field to see if they go beyond the range defined in the flow space. If a flow entry with a value beyond that of the flow



```
{
   "Example 1"    : [
     {
        "vlan" :  [[0, 50]],
        "src net" : "192.168.0.0/22",
        "dst net" : "192.168.4.0/22",
        "proto" : "TCP",
        "src port" : [[1024, 65535]],
        "dst port" : [[0, 1023]]
     },
     {
        "vlan" :  [[0, 50]],
        "src net" : "192.168.4.0/22",
        "dst net" : "192.168.0.0/22",
        "proto" : "TCP",
        "src port" : [[0, 1023]],
        "dst port" : [[1024, 65535]]
     }
   ],
   "Example 2"    : [
     {
        "vlan" : [[100], [101]]
        "src net" : "192.168.64.0/20",
        "dst net" : "192.168.64.0/20",
        "proto" : "TCP",
        "src port" : [[80]],
        "dst port" : [[80]]
     }
   ]
}
```

Figure 4. JSON Format for Flow Space

space definition is written, the OpenFlow hypervisor discards the Flow-Mod message and sends an error for the message to the tenant controller.

Subsequently, for flow entries that have passed the consistency check, the possibility of conflicting flow entries is checked by using the flow space management information. If the flow entry has no conflicts, the Flow-Mod message is simply transferred to the OpenFlow switch. By contrast, when flow entries conflict, our OpenFlow hypervisor rewrites the flow entry according to the rules discussed in the next section and forwards the rewritten Flow-Mod message.

B. *Rewriting Flow Entries*

If two tenants have common areas in their network topologies and address spaces in the flow definitions of their flow spaces, the flow entries may conflict on the switches in the common topology. In this case, it is necessary to rewrite a tenant's flow entry and to inject a flow entry to convert the packet header into one that resolves the conflict. When rewriting the flow entries so as not to conflict with other tenants' flow spaces, the hypervisor needs to determine an address space without the conflict. Two rewriting methods can be used; the choice is based on the size of the common area of the topology with other tenants. Below, we explain these methods using the flow spaces of Tables III and the topology of Figure 5 as an example.

1) Partial Case: If the common area covers only part of the topology, we resolve the conflict by using NAT (Network Address Translation) with unused IPv4 address blocks. First, NAT flow entries with unused IPv4 address blocks are set at the edge switch of the common part of the tenant network. These NAT entries convert the collision addresses of all packets in the tenant network temporarily into different address blocks at the edge of the common area. All of the original flow entries from the tenant controller need to be modified to fit the NAT address blocks before they are transferred to the switches.

In Figure 5, SW2 is located in the common area of the topologies of tenant A and tenant B. When the controller of tenant B tries to write a flow entry that conflicts with tenant A to SW2, the OpenFlow hypervisor sets the NAT flow entry to SW3 and SW6 in advance. An example of a NAT flow entry installed in the switches is shown in Table V(a). In this example, the IPv4 address block of tenant B collides with the one of tenant A and is converted into an unused IPv4 address block. In this example, for the packet header of tenant B, the IPv4 address described in the flow space of Table III is rewritten to an unused address 10.168.64.0/24. In SW3 and SW6, this flow entry and a flow entry to convert back to the original packet header are set to each. The match fields of all subsequent OpenFlow messages for the switches in the tenant network are rewritten in order to use the VLAN-ID to determine the tenant's traffic.

2) Overall Case: If the topology of a tenant network is fully included in one of another network, the OpenFlow hypervisor rewrites the flow entry which uses the VLAN-ID reserved for the management address space, as explained in Section IV-A. First, the flow entry for assigning or converting the VLAN-ID is installed in the switches in the common area. This flow entry assures that a different VLAN-ID is assigned to each tenant on the basis of the ingress physical port number. Subsequently, we rewrite the VLAN-ID of the match field for the flow entry that the tenant writes and forward it to the switch.

In Figure 5, the topology of tenant C is entirely included in the one of tenant A. In this case, a flow entry for rewriting the VLAN-ID is installed in SW1 and SW4, that are in the included tenant network, where the flow entries may conflict. If the flow space of tenant C, which is the included tenant network, uses VLAN, the flow entry that converts the value of the VLAN-ID is installed. If it does not use VLAN, a flow entry which pushes a VLAN-ID is installed. The first row of Table V(b) shows the former case, and the second row shows the latter. The OpenFlow hypervisor converts all OpenFlow messages received from the controller of tenant C into ones that use the VLAN-ID assigned for tenant C before transferring them to SW1 and SW4.

VI. Implementation

We implemented an OpenFlow hypervisor prototype. The core modules of the system were the flow space manager and flow translation engine. We measured and evaluated simple benchmarks individually for each of these prototypes. In this section, we describe the implementation of each module of the

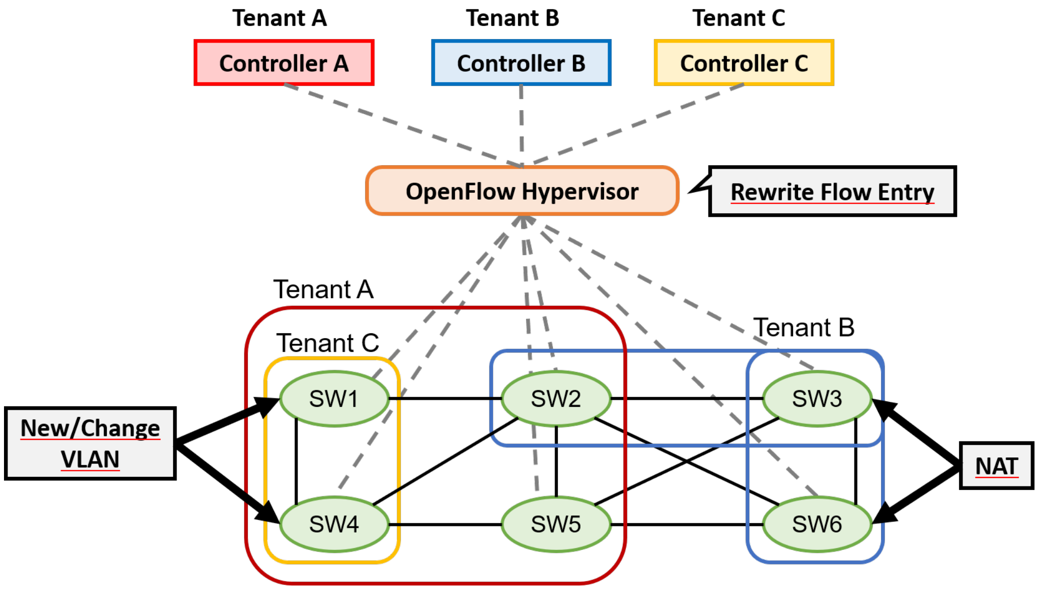


Figure 5. Rewriting Flow Entry

TABLE V. EXAMPLES OF FLOW ENTRIES FOR REWRITING

(a) NAT Flow Entry

Entry	Match Field	Action
NAT Flow Entry	VLAN-ID = 100 Src IPv4 = 192.168.64.0/24 Dst IPv4 = 192.168.64.0/24 Src TCP = 80	Set Field: Src IPv4 = 10.168.64.0/24, Dst IPv4 = 10.168.64.0/24

(b) VLAN Flow Entry

Entry	Match Field	Action
New VLAN Flow Entry	In_Port = 2	Push VLAN: 2049
Change VLAN Flow Entry	In_Port = 2 VLAN ID = 100 Src IP = 192.168.65.0/24 Dst IP = 192.168.65.0/24 Src TCP = 80	Set Field: VLAN-ID: 2049

prototype system and the preliminary performance evaluation for each.

A. *Flow Space Manager*

The flow space manager holds definitions of the given flow space and investigates in advance where flow entries can collide in it. Here, the flow space is defined as shown in Figure 4; the manager analyzes it and holds flow definitions for each flow space. At this time, in each flow definition, an address space that has overlapping address blocks with one of the other flow spaces in all match fields may cause a conflict.

The flow definition is managed using the hash of the source IP address space with the network address of a 24-bit prefix as the key. In this case, if the source IP address space of the flow definition is smaller than /24, the network address of the /24 network including it is used as the key. If it is larger than /24, the network addresses of all /24 networks are registered as multiple entries.

The manager prototype was implemented in Ruby 2.3, and the overhead of flow registration was measured. In particular, we measured the overhead of flow registration of 5000 flow spaces to the manager. Figure 6 shows the results of the evaluation measured for an Intel Core-i7 3.6GHz with 16GB memory. The graph with the plus signs (+) shows the case of 5000 flow spaces without any conflicts, while the one with the cross signs (x) shows that of 5000 flow spaces containing one conflict for each. Considering that the flow space registration is relatively infrequent, this result indicates an acceptable level of performance.

B. *Flow Translation Engine*

The flow translation engine receives flow entries that tenant controllers try to write to the OpenFlow switch and rewrites them to avoid conflicts by using the algorithm described in Section V-B. In OpenFlow, the controller sends a Flow-Mod message to an OpenFlow switch for changing the flow entry on each switch. The flow translation engine parses this Flow-Mod message and transfers it after rewriting the packet data such as the match field of the flow entry.

The flow translation engine was implemented in Python 3.6

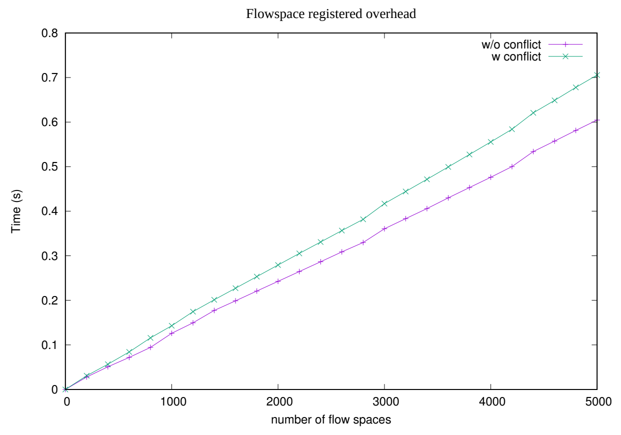


Figure 6. Execution Time of Flow Registration

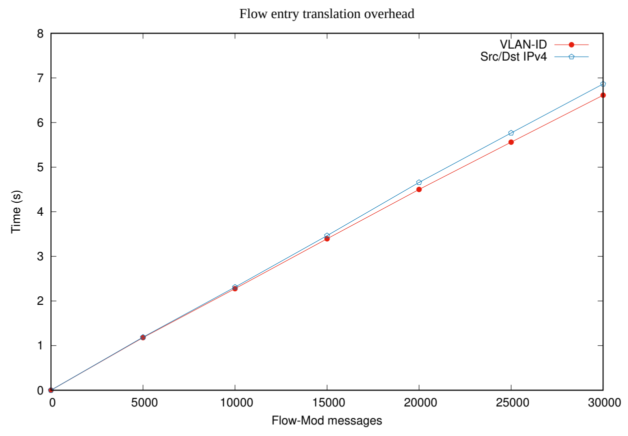


Figure 7. Overhead of Flow Translation

and Ryu SDN Framework 4.16, and the overhead of rewriting the match field of the flow entries was measured. In particular, we measured the execution time of translation for two rewriting patterns with 30000 Flow-Mod messages. The first pattern worked to evaluate the overhead for rewriting the original VLAN-ID to a different value, while the second pattern worked to evaluate rewriting the source and destination IPv4 addresses. The results, as measured by a computer with an Intel Core-i7 3.6GHz and 16 GB memory are shown in Figure 7. This engine could rewrite 30000 Flow-Mod messages in about 13 seconds for both patterns. This means that the average flow translation overhead was 0.22 ms per Flow-Mod message. This overhead is required only when a new flow space is added. Therefore, these results show that this engine has sufficient performance.

VII. Discussion

We proposed an OpenFlow network virtualization scheme that allows each tenant to freely use OpenFlow technology in a multi-tenant network environment. The core module of our scheme is the verification-based virtualization management of OpenFlow networks. The proposed OpenFlow hypervisor manages the flow spaces defined by each tenant beforehand, detects overlaps of the flow space definitions, resolves conflicts by rewriting the match fields' values, and monitors for erroneous control messages violating the flow space definitions. The designers of tenant networks can use it to freely design their own networks by defining network address ranges such as the IP address.

FlowVisor expects that the individual flow spaces never overlap; thus, it does not verify whether conflicts occur between flow spaces. This means that conflict avoidance among flow spaces is left to the responsibility of the tenant's administrators. From this point of view, it seems reasonable to view it as a network partitioning technique rather than a virtualization. Sköldström [5] et al. propose a virtualization method that uses FlowVisor as a relay network of a wide area network. They focus on resource management, whereas our research mainly deals with mapping to lower-layer network separation technology such as MPLS. The virtualization method of Yamanaka et al. [6] works by assigning and tagging a specific MAC address for each virtual network at the edge of the network. This method restricts flow definitions to those that can be described by each tenant.

Our method enables each tenant to define its virtual networks freely and guarantees isolation of the tenant networks automatically. Using our scheme, established TCP/IP networks can be migrated to a datacenter where flexible controls can easily be introduced using OpenFlow technology. Even when the backend of the IT infrastructure of the current organization is moved to the cloud environment, it will be possible to provide both flexible network control and ease of design like that of a conventional network.

VIII. Conclusion

We proposed a virtual network management system that maximizes the ability of OpenFlow virtualization by using verification of the flow space definition. The method enables individual tenant networks to be freely designed in a multi-tenant network environment and ensures isolation among them. This makes it possible for IaaS providers to provide a flexible tenant network in which OpenFlow technology is freely used for and by each tenant user. A preliminary evaluation of a prototype shows that the proposed flow space management has sufficient performance.

Acknowledgment

This work was supported by the Japan Society for the Promotion of Science (JSPS KAKENHI Grant Number JP15K00138).

References

[1] S. Higuchi and T. Hirotsu, "A Verification Based Flow Space Management Scheme for Multi-Tenant Virtualized Network," The Eleventh International Conference on Digital Society and eGovernments (ICDS), pp. 24-29, March 2017.

[2] N. McKeown, "Software-defined networking," INFOCOM keynote talk, vol. 17, no. 2, pp. 30-32, 2009.

[3] N. McKeown et al., "OpenFlow: enabling innovation in campus networks," ACM SIGCOMM Computer Communication Review, vol. 38, Issue 2, pp. 69-74, April 2008.

[4] R. Sherwood et al., "FlowVisor: A Network Virtualization Layer," Tech. Rep. OPENFLOW-TR-2009-01, OpenFlow Consortium, October 2009.

[5] P. Sköldström and K. Yedavalli, "Network Virtualization and Resource Allocation in OpenFlow-based Wide Area Networks," IEEE International Conference on Communications (ICC), pp. 6622-6626, June 2012.

[6] H. Yamanaka, S. Ishii, and E. Kawai, "Realizing Virtual OpenFlow Networks by Flow Space Virtualization," IEICE Technical Report, Network Systems, vol. 112, no. 85, pp. 67-72, June 2012.