P4MT: *Designing and Evaluating Multi-Tenant Services for P4 Switches*

Buck Chung, Chien Chen, Chien-Chao Tseng
Department of Computer Science
National Yang Ming Chiao Tung University
Hsinchu, Taiwan
buck5060.cs06g@g2.nctu.edu.tw,
chienchen, cctseng@cs.nctu.edu.tw

Jim Hao Chen, Joe Mambretti
International Center for Advanced Internet Research
Northwestern University
Chicago, Illinois 60611
Jim-Chen, j-mambretti@northwestern.edu

***Abstract*—The goal of P4MT is to leverage the role-based control mechanism in P4Runtime Specification to further enable multi-tenancy on a single P4 switch. The evaluation results show that P4MT consumes only a small percentage of ASIC space and causes a negligible increment in data plane/control plane latency. International P4 Experimental Networks (i-P4EN) has been created to realize the potential of P4 experimental networks. To perform multiple experiments for research groups, multitenancy is required to increase the efficiency of the testbed. By modifying the P4Runtime design and P4 pipeline, we designed and implemented multi-tenancy on a P4 switch. With P4MT, each tenant can select its own P4 pipeline and control the packet processing without intervening with the packet processing of other tenants. Future work includes supporting P4 Externs for programmable data plane networking, multi-tenant networking and automatic pipeline migration on P4MT.**

***Index Terms*—P4, Software Defined Networking, multi-tenant networking, programmable networking**

I. Introduction

Software Defined Networking (SDN) and the OpenFlow protocol [1] enable programmability on computer networks. P4 [2] brings future flexibility to packet processing, making it easier to support non-standard protocols. Researchers are innovating and leveraging P4 to solve network problems, but these prospective solutions need to be verified in a testbed before being deployed into production to ensure optimal performance and avoid potential problems.

Several P4 research institutions around the world formed a partnership to design and implement the international P4 Experimental Networks (i-P4EN) (Figure 1). The participating institutions can share distributed P4 resources over international research and education networks, as well as initiate international P4 research collaboration projects. The initial network scenarios employed in this research testbed are Software-Defined Network Exchanges (SDXs), Network and Cloud Testbed Networks, Science Networks, and Campus Research Networks.

These four initial network scenarios require multi-tenant support. In order to run multiple projects in P4 Experimental Networks, multiple P4 programs will need to be deployed in a P4 switch if the number of projects exceeds the number of P4 switches on one site. The P4Runtime Specification defines a multiple role design which slices the control plane for different

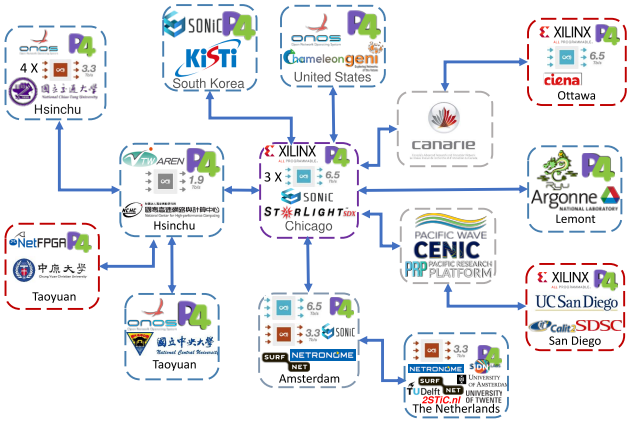

Fig. 1. International P4 Experiment Networks (i-P4EN)

roles and enables control plane redundancy [3]. However, the current P4Runtime implementation does not enable control and data plane partition to support multi-tenant services on a switch. P4MT modifies the current design of the control plane and packet processing pipeline to support a provider and multiple tenant model, where each tenant operates a logically dedicated P4 switch and decides packet process procedures with a tenant controller. (Figure 2) The testbed network can thus support the four international P4 Experimental Networks (i-P4EN) scenarios and multiple tenant specified applications.

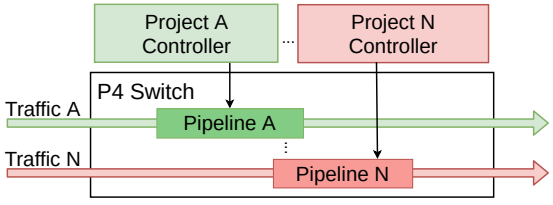

Fig. 2. The concept of multiple tenant P4 switch

II. Background & Related Works

A. P4

P4 [2] is a domain-specific language for describing packet processing procedures and header formats in the switch. By writing the P4 code and installing the code into a P4-enabled switch, new network protocols can be supported without designing a new application-specific integrated circuit (ASIC). Additionally, P4 can map lookup tables based on the purpose of the switch onto the switch's TCAM or SRAM resources,

which improves the space efficiency of the ASIC. There are three characteristics of P4:

- Protocol independence: developers can describe new or self-defined network protocols
- Target independence: P4 can run on both software switches (BMv2) or hardware switches (FPGA or ASIC)
- Field reconfigurability: packet processing procedures can be changed after deployment.

B. P4Runtime

P4Runtime [3] is the P4 control plane protocol that handles data plane management and pipeline reconfiguration. In order not to change the control message format when modifying the P4 pipeline, P4Runtime separates pipeline information (P4Info) from the control message format. P4Runtime also includes a design that separates the control plane into multiple non-overlapping roles in the switch CPU; Role Config, as presented in Figure 3, stores the separation policy. For each role, a primary-standby controller design is applied for redundancy.

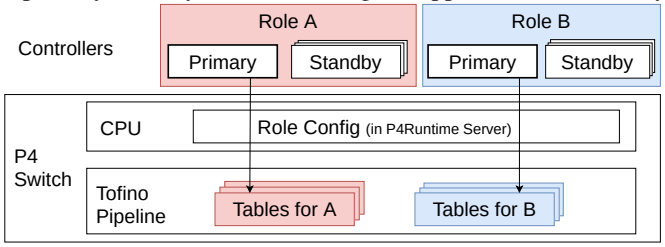

Fig. 3. Role-based P4Runtime Design

C. Related Works

FlowVisor FlowVisor [4] separates the traffic for controllers in an OpenFlow switch with Slice Policy, which defines the type of traffic. FlowVisor acts as an OpenFlow proxy, translating the flow entries from the controller and dispatching notifications from the switch according to Slice Policy. FlowVisor isolates the traffic for a slice by adding extra match keys on flow entries. Since FlowVisor acts as a proxy, it will cause a single point of failure if the proxy crashes. Furthermore, because FlowVisor isolates traffic with the value of header so slices can not manage the same IP or MAC address from different groups of hosts.

OpenVirtex To virtualize the SDN network, OpenVirtex [5] also translates OpenFlow messages in the control plane. Tenants can send the network topology and other requests to OpenVirtex, after which OpenVirtex will create a virtual network for the tenant. Compared to FlowVisor, isolation in OpenVirtex is done through address translation in the network. The MAC and IP address of the tenant are translated when entering the OpenVirtex network, and the address is recovered when leaving the OpenVirtex network. As a result, tenants can use the overlapped addresses in their traffic.

HyPer4 HyPer4 [6] designs and implements a P4 program emulator with a P4 program. Multiple P4 programs and dynamic pipeline loading are supported in the HyPer4 pipeline. The two most important parts of the HyPer4 pipeline are parser emulation and match-action table emulation. For parser emulation, the parsing length is decided by the HyPer4 parsing table. The packet is then sent back to the beginning of the pipeline with the Resubmit function in P4, and the native parser extracts the packet header. This process is repeated until there is no header left in the packet. Since Resubmit makes packets go through the pipeline more than one time, the packet processing throughput diminishes.

To emulate an arbitrary match action table, multiple match-action stages with multiple action primitives are pre-configured in a match action stage with all possible types of match and actions in HyPer4 design. About 230 tables in ASIC are used to emulate five stages and four primitives tables. [6]. Additionally, in order to match any header structure, HyPer4 concatenates headers into a single, long variable and matches the desired header key with a wide TCAM table in the match-action stage.

III. Multi-Tenant P4 Switch

A. Design Overview

Data/Control Plane Isolation To allow multiple tenants to share a physical P4 switch, we need to enforce data plane and control plane isolation. For data plane isolation, packets are processed by tenant flow rules, and tenants are allowed to send/receive packets on the designated switch ports. For control plane isolation, the control message needs to be verified according to tenant identity; tenants cannot change the behavior on other tenant data planes. Packet In and Packet Out instructions also need to be dispatched and verified.

Pipeline Composition Because multiple projects also need to be deployed in one P4 switch, we pre-define multiple *sub-pipelines* in the P4MT P4 program. Each sub-pipeline is the P4 program used by each project and is designed to accommodate multiple tenant flow rules, which will be discussed in the section on data plane isolation. In Figure 4, Sub-pipeline is a set of flow tables that functions as a packet processing procedure; tenants share a sub-pipeline (with data plane isolation enforced) if more than one tenant need to use the same packet processing procedure. Moreover, the enrollment of a new tenant in P4MT cannot intervene the existing packet processing of other tenants.

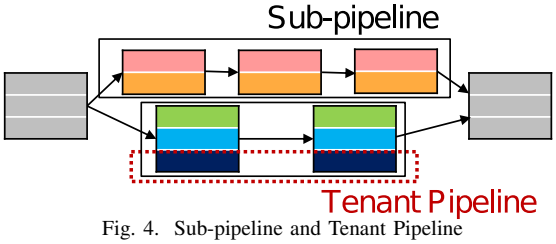

Fig. 4. Sub-pipeline and Tenant Pipeline

Role There are two kinds of roles in P4MT: provider and tenant. The provider is the administrator who deploys P4 pipelines, handles tenant pipeline requests, and sets up an isolation rule for the tenant. The tenant then processes packets according to provider administrative rules and tenant flow rules.

ID-based Isolation To identify each tenant and sub-pipeline in P4MT, an unique identifier is assigned to each tenant

and a sub-pipeline unique ID (sub-pipeline ID) to each sub-pipeline. We reference the concept of Role ID in P4Runtime Specification [3] for tenant identification.

Operation Scenario We describe the process of tenant pipeline allocation by the provider in the following description and Figure 5.

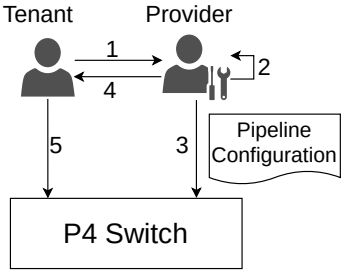

Fig. 5. Tenant Pipeline Allocation

1) Tenants apply pipelines from the provider with Pipeline Configuration, which includes the sub-pipeline ID and designated switch ports. Switch ports are arranged in advance by the provider according to tenant requirements and provider hardware deployment restrictions.
2) The provider verifies the availability of the requested sub-pipelines and switch ports to fulfill the pipeline application; it then assigns the tenant a Role ID.
3) The provider sets up the tenant data plane access rule.
4) The provider replies to the tenant with Role ID and its approval of the pipeline application.
5) The tenant connects to the P4 Switch by using P4Runtime with tenant Role ID.

B. Data Plane Isolation

To process packets for multiple tenants in the switch, we need to 1) identify who owns the packet and 2) process the packet according to the tenant flow rule.

Isolation Enforcement In Step 3 of the Operation Scenario listed above, the provider translates Pipeline Configuration into data plane flow rules. To classify incoming packets for the tenant, the flow table (*Classification and Dispatch Table*) is placed at the beginning of the P4MT pipeline, which tags the packets with Role IDs according to the tenant switch port. This flow table also tags packets with sub-pipeline IDs to dispatch packets to the sub-pipelines.

After a packet is processed by the tenant pipeline, the tenant may send the packet to the switch port of another tenant. As a result, another table (*Isolation Validation Table*) is appended at the end of P4MT pipeline to enforce pipeline configuration, dropping the packet if it violates the pipeline configuration.

Pipeline for Multiple Tenants The tenant packet is processed by one of the sub-pipelines. Since a sub-pipeline is required to accommodate multiple tenant pipelines, there are two possible table designs to support this: 1) pre-defined multiple sets of tables for each sub-pipeline. 2) one set of tables for each sub-pipeline and tenants sharing the tables.

The advantage of the first design is that the flow entries are isolated in different tables. However, pre-configured tables waste the capacity of the switch ASIC if few tenants are using the sub-pipeline. Moreover, if the numbers of tenants exceed the numbers of pre-configured sub-pipeline slots, P4 switch needs to undergo flushing and re-configuration to increase the number of tables. This will lead to pause in packet processing and the need to re-install tenant flow entries.

In contrast, sharing tables consume minimal ASIC capacity for tenants, and no reconfiguration is required when a new tenant enrolls. One disadvantage of this design is that tenants share the capacity for the maximum number of flow entries in a table. Furthermore, access control mechanism on the data/control plane is necessary in order to prevent tenants manipulating the packets of other tenants. Nevertheless, considering all of the aforementioned features, we adopted a *shared table* design to avoid having to halt all packet processing or delete tenant flow entries that affect the availability of the testbed.

For data plane isolation, tables match both the Role ID of the packet and the key of the tenant entry to ensure that the packet is processed by tenant flow entry. Control plane isolation is also accomplished with Role ID (Figure 6); this is discussed in next section.

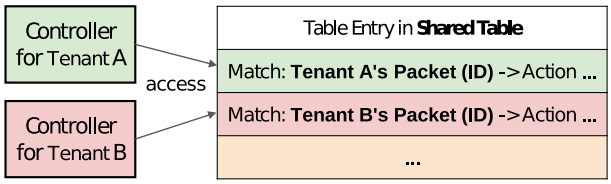

Fig. 6. Isolation on Shared Table

P4MT Pipeline Design As shown in Figure 7, P4MT includes multiple kinds of sub-pipelines that *shared tables* to accommodate tenants flow entries. To enforce isolation, *Classification and Dispatch* is placed at the beginning of the pipeline while *Isolation Validation* is placed at the end. These two tables are categorized as *non-shared table* because they function solely as access control; these type of tables can only be accessed by the provider.

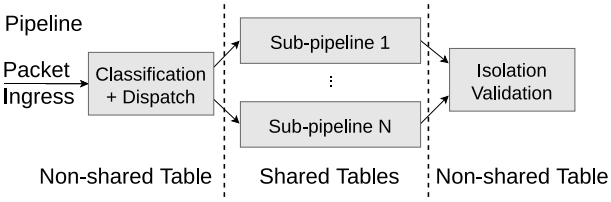

Fig. 7. P4MT Pipeline

C. Control Plane Isolation

Table Access Control Since a non-shared table can only be accessed by the provider, we verify the control message on the switch CPU (P4Runtime Server) and prevent tenants from modifying entries in non-shared tables. For the provider to assign the type of tables in the P4Runtime server and the P4Runtime server to verify the control message, we reference the Role Config design in P4Runtime Specification [3] to design our Role Config format and implement it in the P4Runtime server. Figure 8 demonstrates how table access control works.

Table Entries Access Control Enabling multiple tenants to access shared tables will result in tenants modify the

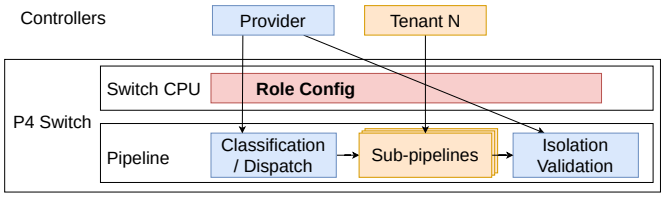

Fig. 8. Table Access Control

flow entries of other tenants. As a result, we modified the P4Runtime server to record the Role ID when inserting flow entries. If a tenant tries to modify, delete or read entries of another tenant, the P4Runtime server will deny this action.

Packet In and Packet Out Packet In and Packet Out is an important function for the controller to realize changes in the network. In P4Runtime, Packet In sends packets from the P4 switch to the primary controller, while Packet Out is receiving packets from the primary controller sent by the P4 switch. For a multiple-tenant P4 switch, a Packet In from the data plane needs to be dispatched to the tenant controller and a Packet Out needs to be verified if its egress destination violates pipeline configuration. In the P4Runtime Specification, controller metadata will contain additional information for Packet In [3]. Since the packet in the data plane is tagged with a Role ID, the Role ID is also tagged in the controller metadata at the end of data plane. The P4Runtime Server dispatches the packet according to the controller metadata (Figure 9)). On the other hand, the controller tags the packet with the packet destination e.g., the egress port on a switch as well as the Role ID. Role ID in the data plane is assigned at the beginning of the pipeline according to controller metadata. Finally, Isolation Validation in the data plane verifies packet destinations to prevent tenants from sending packets to the switch ports of other tenants. The operation of P4MT Packet In/Out is shown in Figure 9 and Figure 10.

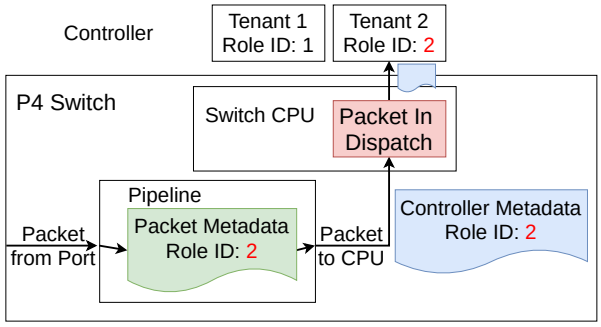

Fig. 9. Packet-In in P4MT

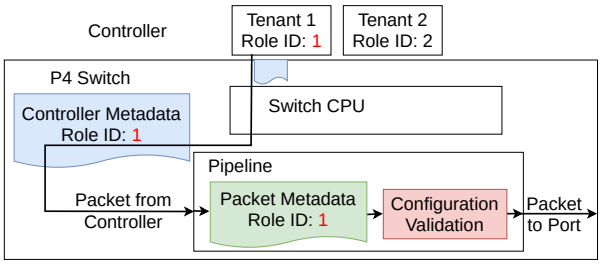

Fig. 10. Packet-Out in P4MT

IV. Evaluation

A. ASIC Space Overhead

Since there are limited resources (such as the number of tables) on P4 ASIC, it is necessary to evaluate the impact that multiple tenants and sub-pipelines have on the ASIC resources. The targets of our evaluation in the P4MT P4 program include:

- Bit width of the ternary match (consumes TCAM)
- Bit width of the exact match (consumes SRAM)
- Bit width of the action parameter (consumes ALU)
- Number of tables

We also estimated the total ASIC space and compared it with the overhead of multiple tenants. For ASIC capacity, we used Tofino from Barefoot as a reference, although the real capacity and ASIC design of Tofino are protected by an NDA. As a result, we used the following method to estimate ASIC capacity. We counted the evaluation targets of resource usage on one of the P4 example programs from the Tofino software development kit. The P4 Program is compiled and the report from compiler shows the percentage of evaluation target usage that is also generated. We then estimate the total ASIC capacity.

TABLE I
ASIC Space Evaluation Result

	Overhead of multi-tenant	Basic.p4
Ternary match width	159 bits (2.71%)	216 bits (3.69%)
Exact match width	18+9*3 bits (0.39%)	0 bits (0%)
Action width	18 bits (1.04%)	27 bits (1.56%)
Table number	2 (1.04%)	3 (1.56%)

To evaluate the maximum number of sub-pipelines in the ASIC, We increased the number of sub-pipelines in the P4MT P4 program until pipeline compiling failed. We used the Basic.p4 pipeline from ONOS [7] as a sub-pipeline for this evaluation. Table I shows the estimated results. The ternary match uses the highest percentage of ASIC capacity at 2.71%. The max number of sub-pipelines is 13.

B. Performance

Because we implement P4MT on Tofino ASIC, and because P4MT does not require packets to be processed by pipelines multiple times (i.e., resubmission and recirculation), the increment in data plane latency can be neglected. To evaluate the performance of the P4Runtime Server on P4 Switch, namely our modified control plane, we measured the instruction latency and control packet throughput with 1, 5, 10 and 15 tenant(s) and compared the results with the original design. The maximum number of controllers in the P4Runtime Specification is 16, with one primary and 15 backup controllers. As a result, the maximum number of tenants is 15, with one provider and 15 tenant controllers. Furthermore, the P4 switch is equipped with enough computational resource to accommodate the provider and tenants controllers, which reduces the number of servers we use (Figure 11).

A P4 switch (Inventec D5254, 8 CPU threads, and 8 GB RAM) and two servers, each equipped with 24 CPU threads and 128 GB RAM, were used for setting up the experiment

environment (Topology and Environment are shown in Figure 11).

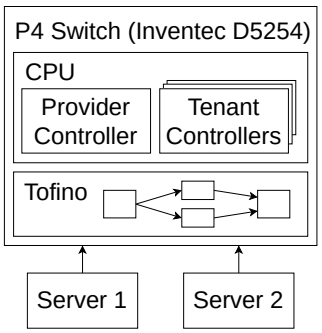

Fig. 11. Experiment Environment

Flow Insertion Latency To enforce table access control and table entry isolation, we modify the P4Runtime server to verify flow insertion/modification/deletion. Thus, we measured flow insertion latency by sending the request 300 times from one of the tenant controllers, waiting for 0.1 seconds between the last response and the next request. During the experiment, only one tenant controller send out flow insertion requests. The rest of the controllers were connected to the switch but inactive. The results are shown in Table II below. 3.09% is the maximum latency difference between the original and the modified P4Runtime server; average latency increased while the latency of the tenant controller increased. However, the latency difference between the original and the other 15 tenants is only 0.017 ms, which is acceptable.

TABLE II
FLOW INSERTION LATENCY

(ms)	Original	1 tenant	5 tenants	10 tenants	15 tenants
Min	0.386	0.394	0.391	0.415	0.409
Average	0.556	0.558	0.56	0.571	0.573
Max	0.833	0.788	0.801	0.832	0.836
Stdev	0.083	0.073	0.078	0.077	0.083
Increased Rate of Average		0.36%	0.69%	2.76%	3.09%

Packet In and Out Latency Packet In packets are dispatched by the P4Runtime server to the tenant controller, so an increase in latency is expected when the number of tenants increases. Packet Outs are verified by the Isolation Validation table on the data plane, whose latency remains the same. To measure Packet In latency accurately, the sender and receiver need to be in the same program to avoid a problem with time synchronization between two servers/programs. As a result, a round trip packet is sent by the tenant controller. The process is as follows (Figure 12). A Packet Out is sent by one of the tenant controllers, which specified a Role ID and the egress port as "CPU Port." The P4 pipeline then sends the packet back to the tenant controller as a Packet In. We measured the latency 300 times and waited 0.1 seconds between every two measurements.

The results are shown in Table III. 4.74% (0.034 ms) is the maximum difference between the original and the modified P4Runtime server.

Packet In Throughput Packet In packets are dispatched in the P4Runtime server, which compares the Role IDs in the control metadata with the ones in the controller connection list. The workload of the P4Runtime server increases when more tenants are connected to the server because there are more Role IDs in the controller connection to compare. We evaluated the maximum Packet In throughput to see if the modification impacts the P4Runtime server performance.

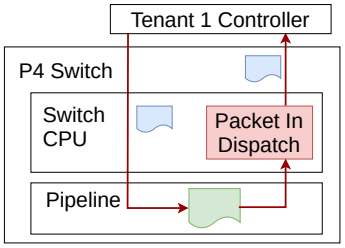

Fig. 12. Packet In and Packet Out Latency

TABLE III
PACKET IN AND PACKET OUT LATENCY

(ms)	Original	1 tenant	5 tenants	10 tenants	15 tenants
Min	0.396	0.432	0.437	0.445	0.427
Average	0.718	0.725	0.752	0.748	0.745
Max	1.268	1.118	1.293	1.302	1.082
Stdev	0.266	0.394	0.462	0.305	0.344
Increased Rate of Average		1.00%	4.74%	4.21%	3.85%

To measure the maximum Packet In throughput, we connect one server to the switch to generate traffic and install flow rules to redirect the traffic to one of the tenant controllers. The tenant controller waits for 30 seconds at the start of this evaluation for stable throughput. We then measure the average throughput for 10 minutes.

The results are shown in Figure 13. Before 1380 PPS (packets per second), the server sending rate and the controller receiving rate are linear, although the receiving throughput does not increase as much as the sending rate. On the other hand, the modified P4Runtime server shares a similar characteristic with the original one but has a slightly lower throughput than the original.

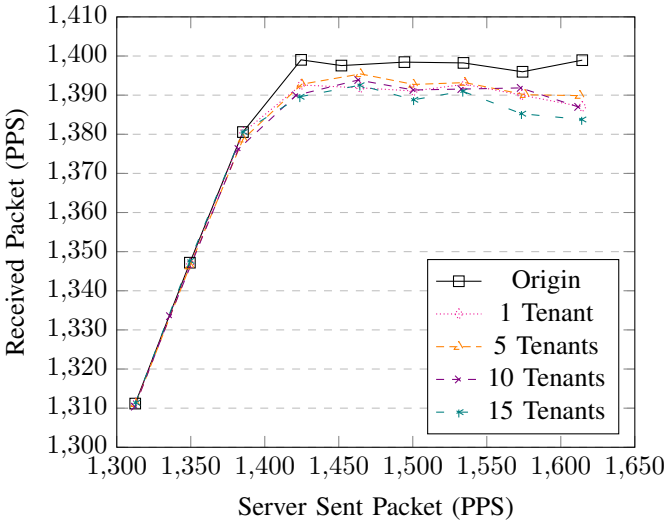

Fig. 13. Packet In Throughput

C. P4MT-enabled P4 Network

Since the goal for P4MT is to accommodate multiple projects on a P4 network, we emulated a P4MT-enabled

network with two tenants by using software switches and mininet. Tenant 1 consists of Hosts 1-1, 1-2, and 1-3, while Tenant 2 consists of Hosts 2-1 and 2-2. Three P4 switches are connected, as shown in Figure 14. There are two types of sub-pipelines on the switches, and a tenant can choose either of them to process its packets. For Tenant 1 traffic, the packet goes through the load balancer on P4 Switch 1, changes its source MAC address on Switch 2 and 3, then arrives at Host 1-2 or 1-3. The destination MAC address is changed when the packet is being send back to Host 1-1. For Tenant 2 traffic, a simple switch function is performed on both Switch 1 and Switch 2. Host 2-1 and Host 2-2 can ping each other. For this test net, each switch port is designated to the traffic of a specific tenant. The switch ports marked in red in Figure 14 belongs to Tenant 1; the ones in blue belongs to Tenant 2. Although Tenant 1 and 2 both use the same sub-pipeline on Switch 2, it is not possible to intervene with the traffic of one another. As mentioned previously, each switch will tag tenant packets with their associated Role IDs when they enter the designated port of the tenant. Therefore, the sub-pipeline tables of Switch 2 will identify the Role IDs to prevent intervention among different tenant flows.

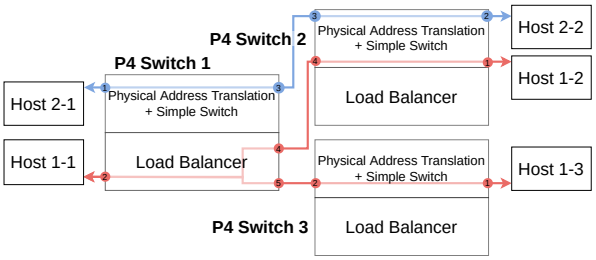

Fig. 14. P4MT-enabled P4 Network

V. Conclusion, Discussion and Future Work

We designed, implemented and evaluated P4MT as a multiple tenant solution for a P4 switch. We leveraged Role ID in P4Runtime to tag and isolate tenant traffic in the data plane. Tenant traffic is only processed by tenant flow rules in the shared tables. On the control plane, access control is set up to enforce Role Config and Pipeline configuration, which prevents tenants from modifying flow entries in the non-shared tables. Furthermore, P4MT includes multiple P4 programs and can dispatch Packet Ins to the right tenant controller and verify the Packet Out destinations. We evaluated the overhead of multiple tenants on ASIC capacity and the performances of data plane latency, flow insert latency, Packet In/Out latency and Packet In throughput. The result shows that the overhead is negligible.

In P4MT, a packet is dropped if it violates the switch port usage in the Isolation Validation table. However, a tenant may accidentally set the wrong switch port on the packet and be confused when the packet is dropped without any notice.

Currently, P4 Externs are not available. Unlike flow entries, P4 Externs have more complex mapping to tenants, which also requires additional design and implementation.

Moreover, since P4MT emulates multiple pipeline functions by combining multiple pipelines into a single pipeline, the structures and resources for parser and match-action tables from different pipelines might conflict with each other. This could be migrated automatically in future works.

Current designs still require manual operation for pipeline migration, meaning that an operational plan from pipeline development to deployment on the P4 network is needed. For pipeline development, the testbed provider shall provide an emulated P4 network, such as mininet or BMv2 in a virtual machine, for a developer. After the developer submits the P4 program (pipeline) to the provider, the provider migrates and deploys the pipeline on a simple P4 network. The developer needs to verify the pipeline on the provider site because pipeline behavior could change after the environment is altered due to the migration. After migrating pipelines from different projects, the pipeline can then be deployed to the P4 network, e.g., i-P4EN.

Finally, a better OAM interface is also required. Isolation rules on data and control planes are currently manually installed, which increases the complexity of the production environment. In addition, a management function user interface for the tenant is desired. This can provide non-critical configurations for a tenant (such as choosing another sub-pipeline) and reduces the amount of manual work by the provider.

Acknowledgment

This work was supported in part by NSF IRNC Grant Award #1450871, NSF Research Infrastructure Grant CNS-1419138, USA; in part by The Featured Areas Research Center Program within the Framework of the Higher Education Sprout Project by the Ministry of Education, Taiwan; in part by the Ministry of Science and Technology, Taiwan, under Grant 109-2221-E-009-077 and 110-2221-E-A49 -044 -MY3; and in part by the Ministry of Economic Affairs, Taiwan, under Grant 107-EC-17-A-02-S5-007.

References

[1] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. "OpenFlow: enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.* April 2008, pp. 69-74.

[2] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming Protocol-independent Packet Processors," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 87–95, Jul. 2014.

[3] The P4.org API Working Group, "P4runtime Specification version 1.0.0," Jan. 2019. [Online]. Available: https://s3-us-west-2.amazonaws.com/p4runtime/docs/v1.0.0/P4Runtime-Spec.html

[4] 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*, vol. 1, p. 132, 2009.

[5] A. Al-Shabibi, M. De Leenheer, M. Gerola, A. Koshibe, G. Parulkar, E. Salvadori, and B. Snow, "OpenVirteX: Make Your Virtual SDNs Programmable," in *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN '14. New York, NY, USA: ACM, 2014, pp. 25–30

[6] D. Hancock and J. van der Merwe, "HyPer4: Using P4 to Virtualize the Programmable Data Plane," in *Proceedings of the 12th International on Conference on Emerging Networking EXperiments and Technologies*, ser. CoNEXT '16. New York, NY, USA: ACM, 2016, pp. 35–49

[7] "ONOS - ONOS - Wiki." Nov. 2019. [Online]. Available: https://wiki.onosproject.org