Virtualization of Table Resources in Programmable Data Plane with Global Consideration

Yuhan Xue[†], Shengru Li[†], Kai Han[†], Sicheng Zhao[†], Huibai Huang[†], Shui Yu[‡], Zuqing Zhu[†]
[†]School of Information Science and Technology, University of Science and Technology of China, Hefei, China
[‡]School of Information Technology, Deakin University, VIC 3125, Australia
Email: {zqzhu}@ieee.org

***Abstract*—In this work, we try to address the problem of memory fragmentation in ternary content addressable memory (TCAM) in programmable data plane (PDP), by designing and implementing a novel network hypervisor for PDP, namely, TPVX. TPVX realizes the virtualization of table resources in PDP with global consideration, *i.e.*, when mapping tenant flow tables to physical switches, TPVX considers their table sizes and the pre-formatted sub-tables in the physical network to improve TCAM utilization and avoid memory fragmentation. Our experimental results verify that with TPVX, the utilization of the table resources in PDP can be improved dramatically and the extra processing latency due to the newly-introduced overheads can be maintained well simultaneously.**

***Index Terms*—Network hypervisor, Programmable data plane, Ternary content addressable memory (TCAM).**

I. Introduction

Recent developments in software-defined networking (SDN) and network function virtualization (NFV) have suggested that data plane programmability would become increasingly important [1, 2]. Following this trend, people have come up with proposals like P4 [1] and protocol-oblivious forwarding (POF) [2] to make data plane more programmable and depend less on existing network protocols. Meanwhile, the innovations in semiconductor industry also speed up the development of programmable data plane (PDP) [3]. With PDP, emerging applications can offload network functions to SDN switches by leveraging NFV, for flexible deployment and enhanced traffic processing performance. However, there is no free lunch, and such scenarios would put a great pressure on the table resource management in SDN switches. It is known that the limited ternary content addressable memory (TCAM) in switches is one of the intimidating bottlenecks of SDN development [4]. The problem could become even worse when it comes to PDP. This is because flow tables with various sizes would be installed in each SDN switch, and without cautious table resource management, memory fragmentation can eat up its TCAM quickly. Therefore, there is a mismatch between the control plane requirement and data plane capability, which has to be properly addressed for PDP.

Previously, people have worked on this mismatch and proposed a few approaches [5–8]. TimeFlip [5] was designed to realize accurate time-based updates and adjust the life-time of flow tables adaptively, for increasing TCAM utilization. Nevertheless, recycling unused flow tables timely would not relieve the memory fragmentation in TCAM in principle, and since the insertion and deletion of flow tables actually become more frequent, the memory fragmentation could be even worse. In [6], the authors proposed to divide the TCAM in a switch into small blocks with a fixed width and incorporate a preprocessing mechanism to activate the blocks on-demand. Hence, the memory fragmentation in TCAM can be reduced with the fine-grained operations on it. However, segmentation and reassembly of flow tables would bring in extra overheads.

As TCAM is also a type of physical resources, its utilization can be improved by leveraging the idea of resource virtualization. HyPer4 [7] tried to use P4 to virtualize PDP, but the virtualization scheme would cause noticeable performance loss in PDP. In [8], we proposed "BigMatch" to design PVFlow, which enables flow tables with different sizes in virtual networks (VNTs) to share a matching stage in a physical POF switch efficiently. Nevertheless, since BigMatch uses wildcard matching, PVFlow actually sacrifices TCAM utilization when reducing the memory fragmentation. More importantly, the studies in [7, 8] only considered how to virtualize the table resources in one switch with PDP and embed tenant flow tables locally, but did not tried to address the problem of table resource virtualization in PDP from the network perspective, *i.e.*, taking the global information of all the switches in the network into consideration.

In this paper, we design and implement a novel network hypervisor for PDP, namely, TPVX, to realize the virtualization of table resources in PDP with global consideration. TPVX can operate on physical POF switches that support protocol-independent packet processing and forwarding. When mapping tenant flow tables to physical switches, TPVX considers their table sizes and the pre-formatted physical tables in the physical network to improve TCAM utilization and avoid memory fragmentation. Specifically, we leverage the idea of "Big-Switch" to embed the flow tables of a virtual switch onto multiple physical switches according to their table sizes. Our experimental demonstrations indicate that with TPVX, we can improve the utilization of the table resources in PDP dramatically, while the extra processing latency due to the newly-introduced overheads can be maintained well.

The rest of paper is organized as follows. Section II briefly introduces the background of table resource virtualization in PDP. The design and implementation of TPVX are described in Section III. Then, we discuss the experimental demonstrations in Section IV. Finally, Section V summarizes the paper.

II. Background and Motivations

In this section, we first introduce the concept of PDP, and then explain the virtualization scheme for it.

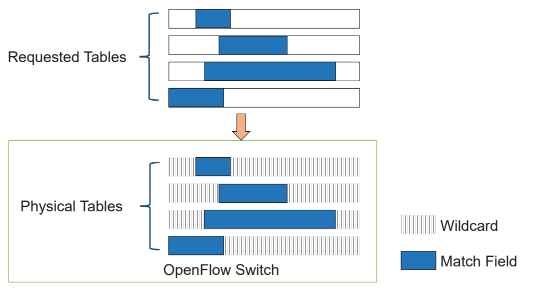
A. Programmable Data Plane

One of the major drawbacks of OpenFlow is that all of the match fields and actions are defined based on existing network protocols. Hence, the OpenFlow specifications have to be updated consistently to include more and more match fields and actions, which greatly limits the programmability of the data plane. In order to address this issue, PDP schemes such as P4 and POF have been considered in the literature [1, 2]. P4 is a high-level programming language that can be used to customize network protocols and packet processing procedures. With P4, the match tables and actions in switches would not be restricted by existing protocols anymore. Note that, P4 programs need to be compiled and loaded into target switches before the actual running of the data plane, and thus it would be difficult to redefine the packet processing pipelines during runtime. POF uses a tuple <*offset, length*> to refer to a packet field, where *offset* is the start location of the field in a packet and *length* represents its length in bits [4]. Therefore, POF switches can locate any data in a packet with the tuple <*offset, length*>, and then process it with the protocol-oblivious forwarding instruction set (POF-FIS) [2] for packet parsing and forwarding.

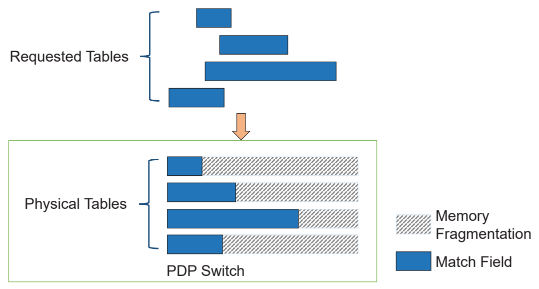
To this end, we can see that both P4 and POF allows an operator to define arbitrary packet fields/formats with various lengths. Fig. 1 provides two intuitive examples to explain the difference on the organizations of flow tables in OpenFlow-based data plane and PDP. Since each OpenFlow-based flow table includes all the match fields supported by OpenFlow, the sizes of the flow tables in Fig. 1(a) are the same even though they may care about different match fields. Hence, after being installed in an OpenFlow switch, the flow tables only facilitate the matching to the cared fields during packet forwarding, while the remaining fields are ignored with wildcards. On the other hand, the PDP switch in Fig. 1(b) uses variable-sized flow tables since there is no need to include irrelevant match fields. This, however, would lead to memory fragmentation in the physical tables of the PDP switch if they are not managed well (*i.e.*, as in Fig. 1(b)). Therefore, how to efficiently accommodate variable-sized flow tables in PDP switches to relieve memory fragmentation has become an important problem to study, which, to the best of our knowledge, has not been properly addressed yet.

B. Virtualization of PDP

As we have explained in the previous section, table resource virtualization can be utilized to address the memory fragmentation in Fig. 1(b). However, the dilemma is that if we still try to maintain the "one-to-one" mapping between virtual switches and physical switches as in normal virtual network embedding (VNE) problems [9, 10], we can never ensure that all the tenant flow tables on a virtual switch would have the same size. Therefore, the memory fragmentation will always exist. Note that, even though we can partition the TCAM on a PDP switch into sub-tables with different sizes and try to organize the tenant flow tables nicely to reduce memory fragmentation [11], predicting the requirements on different sized flow tables would be necessary but tricky since the TCAM is usually limited and we can hardly repartition it during runtime. Therefore, in this work, we will extend the network virtualization systems developed in [8, 12] to address the problem of table resource virtualization in PDP from the network perspective, *i.e.*, leveraging the idea of "Big-Switch" to embed the flow tables of a virtual switch onto multiple physical switches according to their table sizes.



(a) Flow tables in OpenFlow-based data plane



(b) Flow tables in PDP

Fig. 1. Organizations of flow tables in different data planes.

III. System Design and Implementation

In this section, we first explain the design principle of TPVX based on Big-Switch, and then elaborate on the implementation to describe its functional modules in details.

A. Design Principle

Fig. 2 shows our design principle of table resource virtualization in PDP with global consideration. The VNE scheme is illustrated in Fig. 2(a), where TPVX organizes each virtual switch in the tenant VNT as a Big-Switch and then embeds it onto multiple PDP switches in the physical network. To minimize memory fragmentation in the table resources in PDP switches, TPVX partitions the table resources in different PDP switches into sub-tables with different sizes, as shown in Fig. 2(b), and installs the tenant flow tables in them according to the table sizes. Meanwhile, for the case in which the size of a tenant flow table does not match with any of the table sizes in the PDP switches, we design a table padding scheme to pad enough wildcard bits to it for making the

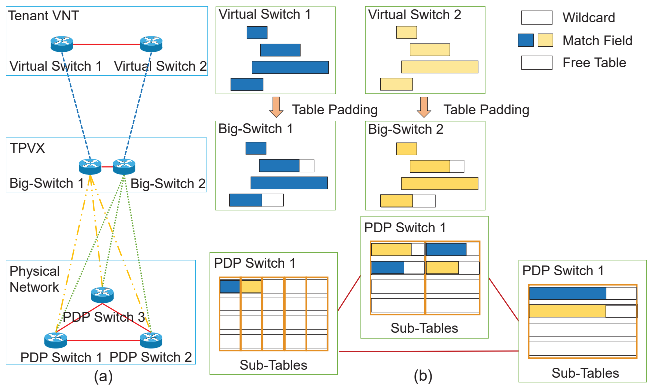


Fig. 2. Table resource virtualization in PDP with global consideration.

right table size. After the tenant flow tables having been installed in the PDP switches, they can be used to process the traffic running in the tenant VNT. Specifically, the PDP switches in a Big-Switch will cooperate to achieve cross-switch traffic processing. Meanwhile, to minimize the extra latency due to the cross-switch processing, we redesign the internal connections of each Big-Switch and make the PDP switches in it form a complete graph.

Note that, the concept of Big-Switch has also been realized in OpenVirteX (OVX) [13], but our design here is different from OVX in three aspects: 1) OVX is based on OpenFlow and thus does not support PDP switches, while our TPVX is designed to operate on PDP switches; 2) OVX cannot partition a physical switch to use the table resources on it to carry multiple virtual switches, while TPVX does not have such restriction; and 3) OVX makes each physical switch store all the flow tables of its Big-Switch, which causes unnecessary flow table duplications, while TPVX only stores a tenant flow table once in a physical switch with the proper table size.

B. Packet Format and Packet Processing Procedure

In this work, we assume that all the PDP switches in the physical network are POF switches [14]. In order to facilitate network virtualization, we define two types of flow tables in each POF switch, *i.e.*, the control tables and tenant tables. The control tables are used to dispatch packets from/to hosts to enter/leave the corresponding tenant VNTs, which are initialized during switch initialization and updated every time when a new tenant VNT has been created. While the tenant tables are the flow tables from the SDN controller of a tenant VNT, and they will be installed in different POF switches according to their table sizes, as explained in Fig. 2(b).

The packet processing procedure in a POF switch is shown in Fig. 3. It can be seen that there are two control tables on each POF switch, *i.e.*, *Control Tables* 0 and 1. Here, *Control Table* 0 is the first flow table that each packet encounters in a POF switch. It first checks whether the destination MAC address (DMAC) of the packet matches to a host that is directly connected to the POF switch. If yes, this is the last hop of the packet and it will make the POF switch output the

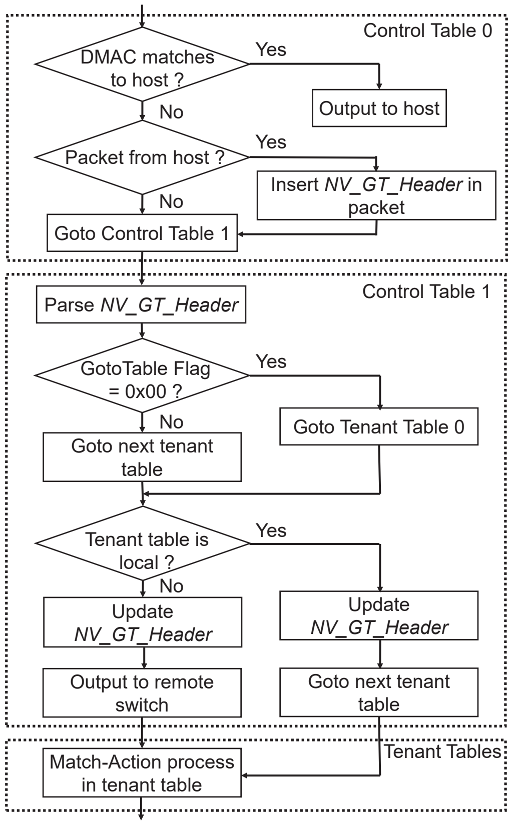


Fig. 3. Packet processing procedure in each physical POF switch.

packet to the host. Otherwise, it checks whether the packet is directly from a host. If yes, it inserts an *NV_GT_Header* field before the Ethernet header of the packet, as shown in Fig. 4.

The *NV_GT_Header* field is used to mark the packet for its tenant VNT and assist possible subsequent cross-switch traffic processing. Here, the *NV_GT_Header* field is designed to contain 4 bytes, where each of its four subfields uses one byte. The first subfield is *GotoTable Flag*, which is used to indicate that immediately after the control tables, whether the packet should be processed by the *Tenant Table* 0 of its tenant (*i.e.*, the first tenant table in the pipeline) or not. Note that, as we have explained above, the table resource virtualization by TPVX can put the tenant tables that are for the same virtual switch on different POF switches. This is reason why *GotoTable Flag* is needed. Specifically, if the packet will be processed by the *Tenant Table* 0 of its tenant after the control tables, its *GotoTable Flag* is set to 0x00, and 0x0f otherwise. The next subfield is *Next Table ID*, which stores the ID of the next tenant table if *GotoTable Flag* is 0x0f. Note that, to achieve smooth cross-switch traffic processing, TPVX numbers each tenant table on the same virtual switch with a unique ID according to their sequence in the pipeline, when storing them in different POF switches. The remaining two subfields in *NV_GT_Header* are *Tenant ID* and *Virtual Switch ID*, respectively, which are used to identify the virtual switch that the packet is currently being processed in. Since a

POF switch can carry multiple virtual switches' tenant tables, we need these two subfields to dispatch the packet to the right pipeline. When a packet first comes in from a host, we use its input port on the ingress switch to determine its *Tenant ID*. *NV_GT_Header* is only used when the packet is in a tenant VNT, and will be removed when it is forwarded to a host.

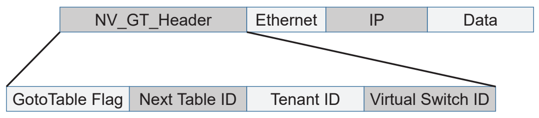


Fig. 4. Packet format used by TPVX for network virtualization.

Back to the procedure in Fig. 3, we can see that *Control Table* 0 sends the packet to *Control Table* 1 after the processing in it has been done. *Control Table* 1 first parses the packet's *NV_GT_Header*, and then uses the obtained information to direct it to the next tenant table in its pipeline. Specifically, *Control Table* 1 determines whether the next tenant table for the packet is on the local switch or not. If yes, it uses the *GotoTable* action to send the packet to the local tenant table. Otherwise, the packet will be forwarded to the remote switch that contains its next tenant table with the *Output* action. Meanwhile, the information in the packet's *NV_GT_Header* gets updated accordingly, and then the packet will be processed by the tenant tables in its pipeline in sequence.

C. System Implementation

The system architecture of TPVX is shown in Fig. 5. On the top of TPVX, there are the tenant controllers, which control their virtual switches through TPVX. The tenant flow tables from the controllers are translated by TPVX and installed in a proper POF switch according to their table sizes. It can be seen that TPVX mainly consists of two modules, *i.e.*, the table management and entry management modules.

1) Table Management: The main functionality of this module is to install tenant tables with different sizes in suitable POF switches, by leveraging the idea of Big-Switch. Note that, in POF, tenant tables are actually installed by the controller with *TableMod* messages, and thus TPVX can determine

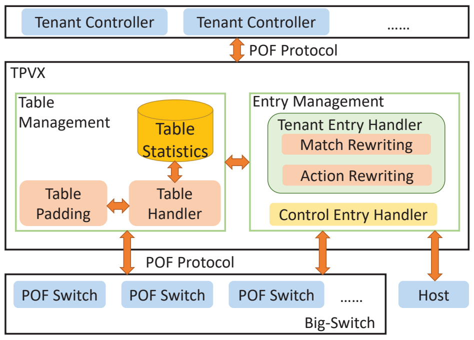


Fig. 5. System architecture of TPVX.

where to install tenant tables based on the table size in their *TableMod* messages and the following two constraints.

- **Sub-table Size Constraint**: In this work, we partition the table resources (*i.e.*, TCAM) in POF switches into sub-tables that have sizes as 32, 64 and 128 bits, by considering the size distribution of possible tenant tables. Hence, each tenant table is mapped to a physical sub-table by rounding its table size to the nearest sub-table size to the longer end. Meanwhile, the table padding process will insert enough wildcard bits at the end of the tenant table to make its size the same as the sub-table size.
- **Sub-table Capacity Constraint**: Since the table resource in each POF switch is limited, the number of entries that a physical sub-table can support is also restricted. Therefore, we implement a table statistics process in the table management module to record the number of used entries on each POF switch. When receiving a *TableMod* message, TPVX checks the number of required entries in it and installs the tenant table in a POF switch that has both proper sub-table size and enough available entries.

After taking care of the two constraints, the table handler in the table management module dispatches a tenant table to the selected POF switch in the corresponding Big-Switch. In the meantime, after processing each *TableMod* message for a new tenant table, the table handler instructs the entry management module to generate and insert corresponding control entries in the related control tables in the POF switches.

2) Entry Management: This module ensures that the entries in each tenant table can facilitate traffic processing and forwarding in POF switches correctly. In the tenant entry handler, we use match rewriting to mask the unnecessary bits in match fields and make sure that the change on table sizes caused by the table padding would not affect the field matching in POF switches. Then, action rewriting handles the actions and instructions in the entries, especially for the *Output* action and the *GotoTable* instruction. For the cases in which a packet needs to be forwarded to another virtual switch or a host, the tenant controller encodes the *Output* action in the corresponding entry. Hence, for the *Output* action, the action rewriting needs to replace the virtual port ID in the entry from the tenant controller to the actual physical port ID on the related POF switch, and for the latter case (*i.e.*, the next hop is a host), it needs to add a *DelField* action there to remove the packet's *NV_GT_Header*. On the other hand, the tenant controller encodes the *GotoTable* instruction in an entry to move a packet in its processing pipeline in sequence. Therefore, since TPVX may disassemble the pipeline and install the tenant tables in it in different POF switches, the action rewriting needs to first determine whether the next tenant table in the pipeline is local and then modify the *GotoTable* instruction in the entry accordingly. Specifically, if the next tenant table is local, action rewriting only needs to modify the *GotoTable* instruction to point to the next tenant table in the local switch. Otherwise, it needs to rewrite the

GotoTable instruction to an *Output* action to direct the packet to the next tenant table in a remote switch.

The control entry handler is in charge of the two control tables on each POF switch to add/remove control entries in them for tenant VNTs. For instance, when a host in a tenant VNT first comes online and gets detected by TPVX, TPVX will use the control entry handler to generate two control entries for it and install them in the switch that the host directly connects to. The control entries are for the traffic from/to the host, *i.e.*, to forward packets correctly and add/remove *NV_GT_Header* on them, respectively.

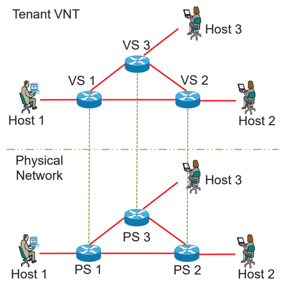
IV. Experimental Demonstration

To evaluate the performance of our TPVX, we implement it in a practical network system and conduct experiments. In the experimental setup, the data plane includes three POF switches and three hosts (as shown in Fig. 6). Each POF switch is based on our homemade software POF switch [14] and runs on a high-performance Linux server. The control plane, *i.e.*, the tenant controller [15] and TPVX, also runs on a Linux server. The experiments measure the table resource usage, data plane latency, and processing latency of TPVX.

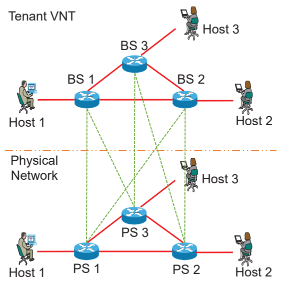
A. Table Resource Usage

As shown in Fig. 6, we consider two scenarios in the experiments. The Single-Switch scenario in Fig. 6(a) is the benchmark that still maintains the "one-to-one" mapping between virtual and physical switches (*i.e.*, VS' and PS', respectively), while the Big-Switch scenario in Fig. 6(b) is the one used in our TPVX, where each VS gets mapped to a Big-Switch (BS) that includes two PS'. Note that, since we do not have access to hardware PDP switches that contain TCAM, we can only emulate the PDP switches with our software POF switch and allocate certain memory space in it to simulate TCAM. Here, we assume that the table width of the TCAM is 40 bits and it can totally carry 4000 entries [11], which means that the total size of the table resource on a POF switch is $40 \times 4000 = 160$ Kbits. Then, the tenant controller tries to install tenant tables in VS' in the tenant VNT. Here, we assume that each tenant table contains 500 entries and its table size is randomly selected from $\{32, 48, 56, 60, 64, 108\}$ bits. The table sizes are selected according to the sizes of common match fields and their combinations, such as the MAC address, IPv4 address, switch port ID, *etc*. Then, we arrange the POF switches in different configurations and test how many tenant tables can be successfully installed in them.

The results on the average number of installed tenant tables are shown in Fig. 7. Here, the configuration "128/128/128" means that we use the Big-Switch scenario and the table resources on the three POF switches are partitioned into sub-tables with a size of 128 bits. The results indicate that no matter how we arrange the sub-tables on the POF switches, the Big-Switch scenario in our TPVX can carry much more tenant tables than the conventional Single-Switch scenario, with the same amount of table resources in the physical network. This confirms that our TPVX can reduce the memory fragmentation



(a) Single-Switch scenario.



(b) Big-Switch scenario.

Fig. 6. Experimental setup.

in the TCAM of physical switches and greatly improve the utilization on TCAM. Meanwhile, it is interesting to notice that in the Big-Switch scenario, different arrangements of the sub-tables on POF switches actually lead to significantly different performance on the TCAM utilization. Specifically, since most of the tenant tables in the experiments have a size that is close to 64 bits, the arrangement of "64/64/128" achieves the largest number of installed tenant tables. Hence, the results also suggest that the performance of TPVX cannot be optimized without a reasonable estimation on the distribution of tenant table sizes, which will be considered in our future work.

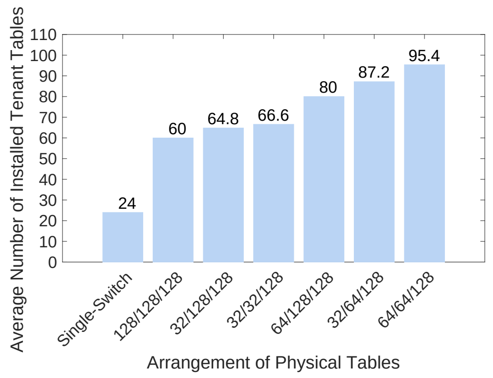


Fig. 7. Results on number of installed tenant tables in the physical network.

B. Packet Processing Latency

As our TPVX actually introduces additional packet processing overhead in the data plane, we conduct experiments to compare the Single-Switch and Big-Switch scenarios in terms of packet processing latency to quantify the overhead. The experiments use a commercial traffic analysis equipment to send and receive packets and apply the standard scheme in [16] to measure the packet processing latency. Here, we consider the cases in which each packet experiences different lengths of processing pipelines (*i.e.*, being processed by different numbers of tenant tables) end-to-end in its tenant VNT. There are four experimental schemes:

- *Single-Switch 1-Hop* means that the tenant tables for each packet only get installed in one POF switch, with the Single-Switch scenario.

- *Single-Switch 3-Hop* means that the tenant tables for each packet can be finished within three hops at most, with the Single-Switch scenario.
- *Big-Switch 3-Hop* means that the tenant tables for each packet can be finished within three hops at most, with the Big-Switch scenario.
- *Big-Switch Worst* means that each of the tenant tables for the packets leads them to be redirected to a remote POF switch for processing, with the Big-Switch scenario.

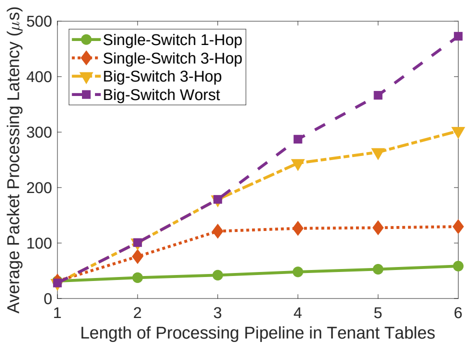


Fig. 8. Results on packet processing latency.

The experimental results are plotted in Fig. 8. As expected, the packet processing latency of the Big-Switch based schemes is longer than that of the Single-Switch based ones. However, the additional overhead is still reasonably small, if we consider the fact that the transmission latency on links is also included in the packet processing latency. Meanwhile, we notice that the latency of *Big-Switch Worst* increases much faster than that of *Big-Switch 3-Hop*, due to the frequent packet redirections among the POF switches. This suggests that when laying out the packet processing pipeline of a tenant VNT with TPVX, one should carefully avoid frequent packet redirections.

C. Control Plane Latencies for Table Virtualization

Finally, we use cbench [17] to measure the control plane latencies for table virtualization, *i.e.*, the time used by our TPVX to process a *TableMod* or *FlowMod* message. Here, we still consider the Single-Switch scenario as the benchmark. The results are listed in Table I. It can be seen that even though the Big-Switch scenario increases the control plane latencies due to the additional operations on the messages, the increases are still relatively small and would not cause significant performance degradation on the network virtualization system.

TABLE I
AVERAGE CONTROL PLANE LATENCIES (MSEC)

	per *TableMod* message	per *FlowMod* message
Single-Switch	0.464	0.600
Big-Switch	1.081	1.110

V. CONCLUSION

In this paper, we designed and implemented TPVX as a novel network hypervisor for PDP, to realize the virtualization of table resources in PDP with global consideration. Specifically, when mapping tenant flow tables to physical switches, TPVX considered their table sizes and the pre-formatted sub-tables in the physical network to improve TCAM utilization and avoid memory fragmentation. Our implementation leveraged the idea of "Big-Switch" to embed the tenant tables of a virtual switch onto multiple physical switches according to their table sizes. The experimental results confirmed that with TPVX, we can improve the utilization of the table resources in PDP dramatically, while the extra processing latency due to the newly-introduced overheads can be maintained well.

ACKNOWLEDGMENT

This work was supported in part by the NGBWMCN Key Project under Grant No. 2017ZX03001019-004 and the Key Project of the CAS (QYZDY-SSW-JSC003).

REFERENCES

[1] P. Bosshart *et al.*, "P4: Programming protocol-independent packet processors," *Comput. Commun. Rev.*, vol. 44, pp. 87–95, Jul. 2014.
[2] S. Li *et al.*, "Protocol oblivious forwarding (POF): Software-defined networking with enhanced programmability," *IEEE Netw.*, vol. 31, pp. 12–20, Mar./Apr. 2017.
[3] P. Bosshart *et al.*, "Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN," in *Proc. of ACM SIGCOMM 2013*, pp. 99–110, Aug. 2013.
[4] S. Li *et al.*, "Improving SDN scalability with protocol-oblivious source routing: A system-level study," *IEEE Trans. Netw. Serv. Manag.*, vol. 15, pp. 275–288, Mar. 2018.
[5] T. Mizrahi, O. Rottenstreich, and Y. Moses, "TimeFlip: Scheduling network updates with timestamp-based TCAM ranges," in *Proc. of INFOCOM 2015*, pp. 2551–2559, Apr. 2015.
[6] W. Li, X. Li, and H. Li, "MEET-IP: Memory and energy efficient TCAM-based IP lookup," in *Proc. of ICCCN 2017*, pp. 1–8, Jul. 2017.
[7] D. Hancock and J. Merwe, "HyPer4: Using P4 to virtualize the programmable data plane," in *Proc. of CoNEXT 2016*, pp. 35–49, May 2016.
[8] S. Li, K. Han, H. Huang, and Z. Zhu, "PVFlow: Flow-table virtualization in POF-based vSDN hypervisor (PVX)," in *Proc. of ICNC 2018*, pp. 1–5, Mar. 2018.
[9] L. Gong and Z. Zhu, "Virtual optical network embedding (VONE) over elastic optical networks," *J. Lightw. Technol.*, vol. 32, pp. 450–460, Feb. 2014.
[10] L. Gong, H. Jiang, Y. Wang, and Z. Zhu, "Novel location-constrained virtual network embedding (LC-VNE) algorithms towards integrated node and link mapping," *IEEE/ACM Trans. Netw.*, vol. 24, pp. 3648–3661, Dec. 2016.
[11] L. Jose, L. Yan, G. Varghese, and N. McKeown, "Compiling packet programs to reconfigurable switches," in *Proc. of NSDI 2015*, pp. 103–115, 2015.
[12] S. Li *et al.*, "SR-PVX: A source routing based network virtualization hypervisor to enable POF-FIS programmability in vSDNs," *IEEE Access*, vol. 5, pp. 7659–7666, 2017.
[13] A. Al-Shabibi *et al.*, "OpenVirteX: Make your virtual SDNs programmable," in *Proc. of HotSDN 2014*, pp. 25–30, Aug. 2014.
[14] Q. Sun, Y. Xue, S. Li, and Z. Zhu, "Design and demonstration of high-throughput protocol oblivious packet forwarding to support software-defined vehicular networks," *IEEE Access*, vol. 5, pp. 24 004–24 011, 2017.
[15] D. Hu *et al.*, "Flexible flow converging: A systematic case study on forwarding plane programmability of protocol-oblivious forwarding (POF)," *IEEE Access*, vol. 4, pp. 4707–4719, 2016.
[16] S. Bradner. (1999) RFC 2544: Benchmarking methodology for network interconnect devices. [Online]. Available: https://tools.ietf.org/html/rfc2544.
[17] POF Cbench Tool. [Online]. Available: https://github.com/USTC-INFINITELAB/pof-cbench