Design and Demonstration of High-Throughput Protocol Oblivious Packet Forwarding to Support Software-Defined Vehicular Networks

Quanying Sun, Yuhan Xue, Shengru Li, Zuqing Zhu, *Senior Member, IEEE*

***Abstract*—Software-defined networking (SDN) is a promising technology that can resolve the challenges faced by vehicular networks. However, the OpenFlow-based SDN implementations can only provide a protocol-dependent data plane. This can restrict the effectiveness of software-defined vehicular networks, since special-purpose protocols that have not been standardized in OpenFlow specifications are used frequently in vehicular networks. To address this issue, this work studies how to realize a protocol-independent data plane by leveraging the protocol oblivious forwarding (POF). Specifically, we present the design of a software-based POF switch (PVS) that supports runtime protocol customization in principle. We implement PVS in a switch box, and propose a flow table management scheme to ensure high-throughput packet forwarding. The experimental results verify that PVS can achieve line-rate packet forwarding at 10 Gbps when the packets' size is 512 bytes, and the proposed flow table management scheme is effective.**

***Index Terms*—Protocol oblivious forwarding (POF), Flow table management, Software-based SDN switch.**

I. Introduction

RECENTLY, the advantages of software-defined networking (SDN), which separates the control and data planes of a network and incorporates centralized network control and management (NC&M) for enhanced network programmability, have been verified extensively in various networks [1–12]. Moreover, the studies in [13] suggested that with the centralized NC&M, SDN could be a promising solution to the challenges faced by vehicular networks [13–17], *e.g.*, low data forwarding efficiency, unbalanced link utilization, frequent security breaches, and insufficient protocol compatibility. Fig. 1 shows the architecture of a software-defined vehicular network. The centralized SDN controller is in charge of the flow setup and packet forwarding in the data plane, which is built with SDN switches to interconnect the access points (APs). Hence, the packets from the vehicles can be forwarded correctly to realize vehicle-to-vehicle communications.

Note that, most of the existing SDN implementations are based on OpenFlow, which uses a protocol-dependent data plane with only limited programmability. Specifically, OpenFlow defines match fields based on existing network protocols (*e.g.*, IP), and thus an OpenFlow switch has to know the protocol to parse and match to the fields in a packet header. Therefore, when it comes to supporting new protocols, OpenFlow has no other choices but to add match fields constantly [18]. This, however, can restrict the effectiveness of software-defined vehicular networks, since special-purpose protocols that have not been standardized in OpenFlow specifications (*e.g.*, dedicated short-range communication (DSRC) [15]) are used frequently in vehicular networks. Hence, people are seeking for the SDN technologies that can further enhance the network programmability and realize a protocol-independent data plane, and both the protocol oblivious forwarding (POF) [19, 20] and programming protocol-independent packet processors (P4) [21] have been put forward for this purpose. Moreover, to improve the security of software-defined vehicular networks, one may expect the SDN switches to support runtime protocol customization. This is because changing the network protocol in runtime helps to make network operations more private and thus more difficult to be compromised. POF fits to this requirement in principle.

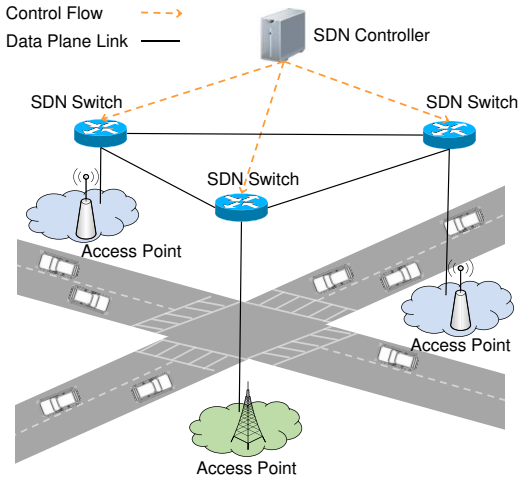


Fig. 1. Architecture of a software-defined vehicular network.

POF uses a three-tuple <*offset*, *length*, *value*> to generalize the description of match fields, and also introduces a generic flow instruction set (POF-FIS) [19] to operate on it. Here, *offset* tells the start bit-location of a match field in packets, *length* indicates the field's length in bits, and *value* describes the field's value. For instance, the *Destination IP Address* field in an IPv4 packet can be described with *offset* $= 30$ bytes and *length* $= 4$ bytes, while its *value* is the 4-byte IPv4 address.

Q. Sun, Y. Xue, S. Li and Z. Zhu are with the School of Information Science and Technology, University of Science and Technology of China, Hefei, Anhui 230027, P. R. China (email: zqzhu@ieee.org).

Manuscript received on September 30, 2017.

Fig. 2 describes an example on how to process IPv4 packets in SDN switches according to the working principle of POF. Firstly, the switch needs to check whether the packet is an IPv4 one. This is achieve by examining whether the value of the *Type* field (*i.e.*, $\{$*offset* $= 12$ bytes, *length* $= 2$ bytes$\}$) in the packet's *Ethernet Header* is $0x0800$. If not, the packet gets dropped directly. Otherwise, the switch continues to extract the *Destination IP Address* field (*i.e.*, $\{$*offset* $= 30$ bytes, *length* $= 4$ bytes$\}$), and uses its value to find the match rule of the packet (*i.e.*, the corresponding packet forwarding action(s)). Next, the switch needs to decrease the value of the packet's *TTL* field (*i.e.*, $\{$*offset* $= 22$ bytes, *length* $= 1$ byte$\}$) by one. If the *TTL* field becomes zero, the packet will be dropped. Otherwise, the switch recalculates and updates the *Checksum* field in the packet (*i.e.*, $\{$*offset* $= 24$ bytes, *length* $= 2$ bytes$\}$) according to the value of the whole *IPv4 Header* (*i.e.*, $\{$*offset* $= 14$ bytes, *length* $= 20$ bytes$\}$). The packet processing procedure shown in Fig. 2 indicates that with POF, the SDN switches can process and forward packets without knowing the actual network protocol in the data plane. Hence, POF makes the data plane completely protocol-independent and can support new protocols in a future-proof manner.

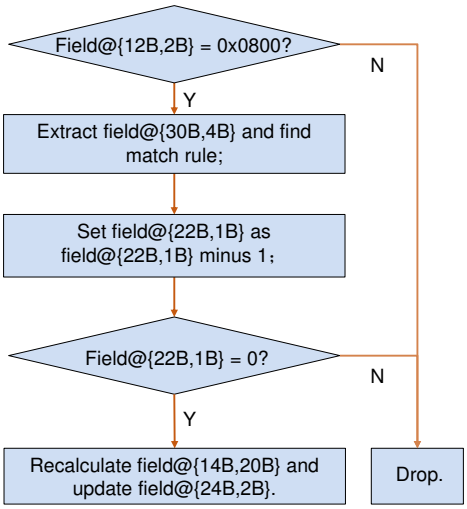


Fig. 2. Example on processing IPv4 packets according to POF.

Previously, people have discussed the working principle of POF in [18, 19], implemented POF-enabled controller and network virtualization hypervisor in [18, 22, 23], and performed functional demonstrations in [24–27]. However, the implementation and demonstration of high-throughput POF switches were still missing. Without a powerful switch like OpenvSwitch (OVS) [28], one can never fully explore the benefits of POF. OVS is an open-source software-based switch that can realize high-throughput packet forwarding based on OpenFlow. Nevertheless, since the organization and processing schemes of POF-based flow entries are significantly different from those in OpenFlow switches, the optimization techniques developed to improve the performance of OVS [28, 29] cannot be directly utilized to design high-throughput POF switches. By leveraging P4, the authors of [30] have realized a protocol-independent software-based switch, *i.e.*, PISCES. However, the switch needs to be recompiled every time when it has to support a new protocol.

In this paper, we present our design of a software-based POF switch (PVS) that supports runtime protocol customization in principle. We implement PVS in a switch box that is based on the advanced telecommunications computing architecture (ATCA), by leveraging the packet processing accelerating tool provided by data plane development kit (DPDK) [31]. Compared with the software-based POF switch that was reported in [18], PVS adopts new hardware/software designs to achieve a much higher packet forwarding throughput. We also propose a flow table management scheme, which organizes the flow entries in PVS based on their popularity, to further improve the performance. Our proposal is then experimentally demonstrated in a real network testbed. The experimental demonstrations verify that PVS can achieve a packet forwarding rate of 10 Gbps when the packets' size is 512 bytes, and the flow table management scheme can effectively improve its throughput.

The rest of this paper is organized as follows. We describe the architectural design of PVS in Section II. The flow table management scheme is discussed in Section III, and Section IV presents the experimental results on PVS. Finally, we summarize the paper in Section V.

II. Architectural Design of PVS

Fig. 3 shows the forwarding model of PVS. In this model, each packet directly gets processed by a pipeline, which is built with the multi-stage flow tables that are based on POF-FIS [22]. Specifically, the header fields of packets (*i.e.*, the flow entries' match fields) are extracted according to the POF-style three-tuple $<$*offset*, *length*, *value*$>$. Therefore, when PVS is up and working (*i.e.*, at "runtime"), POF controller can add, remove and modify flow entries for specifying match-action rules to support new protocols on-the-fly. Moreover, POF-FIS also defines the *Write-metadata* instruction, which enables flow tables to extract an arbitrary bit-block in a packet and cache it in PVS' *metadata memory* for later use in a pipeline. Hence, the packet processing and forwarding in PVS is more flexible than that in OVS [28] and PISCES [30], since the predefined packet parser is not required anymore. Consequently, PVS can support runtime protocol customization in principle. To realize the forwarding model in Fig. 3, we design the architecture of PVS as that in Fig. 4(a), which runs on a Linux system and uses DPDK to ensure high-throughput packet forwarding throughput [18].

Note that, to improve its throughput, OVS builds a fast path that consists of *Microflow Cache* and *Megaflow Cache* [28, 29]. Specifically, when it receives a new packet flow, OVS creates an entry in both *Microflow Cache* and *Megaflow Cache*, and when they are full, new flows would be processed in the slow path. However, the cache management in OVS does not consider the popularity of flow tables, and thus when there are multiple simultaneous flows, the packet arrival sequence can affect its forwarding performance [29]. Moreover, the flexible match fields in POF would make the table cross-product computation for *Megaflow Cache* much more complex, even

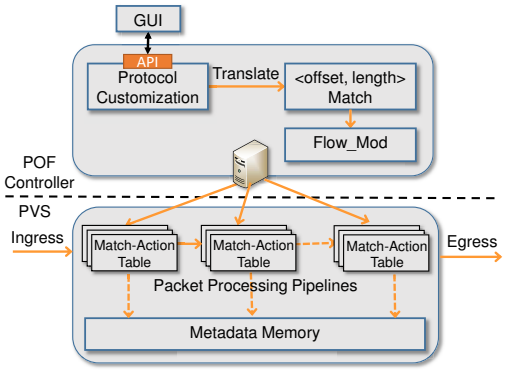


Fig. 3. Forwarding model of PVS.

though it is already very complex in OVS [28]. Considering these issues, we design the modules for *Cold Tables (C-Tbl)*, *Hot Tables (H-Tbl)*, *Exact Flow Entry Generation (EFE-Gen)*, and *Packet Processing (Pkt-Proc)* in PVS.

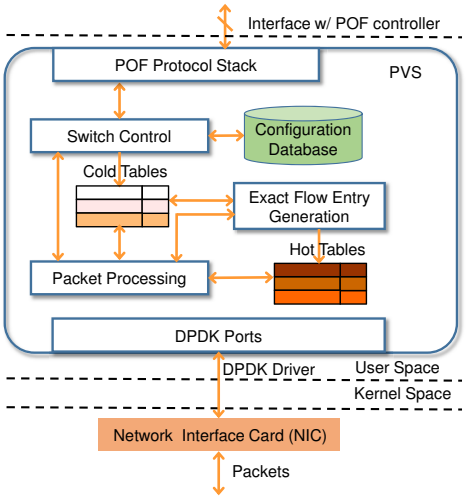
When a packet is received from the DPDK ports, *Pkt-Proc* looks up *H-Tbl*. If an entry can be found there for the packet, it processes the packet according to the matched entry. Otherwise, *Pkt-Proc* continues to check *C-Tbl* for the packet, and if it still cannot find any entry there, it will tell the *Switch Control (Sw-Ctrl)* module to encode the packet in a *Packet_In* message to send to the POF controller. Here, *H-Tbl* stores the flow entries generated by *EFE-Gen*, which are all for exact match. Hence, *Pkt-Proc* can search for flow entries in *H-Tbl* quickly with hash match. On the other hand, *C-Tbl* stores the flow entries that are just received from the POF controller through *Sw-Ctrl*. These flow entries may have masks, and for each flow entry, the mask can be discontinuous in terms of bit-location. Therefore, matching to an entry in *C-Tbl* would be slower than that in *H-Tbl*, since both the longest prefix match (LPM) and hash match are not feasible. The hash match in *H-Tbl* has a complexity of $O(1)$, and the complexity of searching *H-Tbl* is $O(n)$, where n is the number of entries in it. The complexity of entry match in *C-Tbl* is $O(m)$, where m is the number of bits to match in each entry, while the search complexity of *C-Tbl* is the same as that of *H-Tbl*.

EFE-Gen generates an exact flow entry based on an entry in *C-Tbl* and stores it in *H-Tbl*, when *Pkt-Proc* determines that the entry in *C-Tbl* is popular according to the history of packet processing. *Sw-Ctrl* manages all the functional modules in PVS, and it is also responsible for initializing the local resources in PVS and executing the POF control messages received from the POF controller at runtime. When PVS is handshaking with the POF controller, *Sw-Ctrl* gets the configuration of PVS (*e.g.*, the settings of switch ports and flow table capacity) from the *Configuration Database (Cfg-DB)* and stores the configuration received from the controller (*e.g.*, the settings of *H-Tbl* and *C-Tbl*) in *Cfg-DB*. The *POF Protocol Stack* module is in charge of the communications with the POF controller, *i.e.*, parsing the control messages from the POF controller and encapsulating control messages to send to the POF controller.

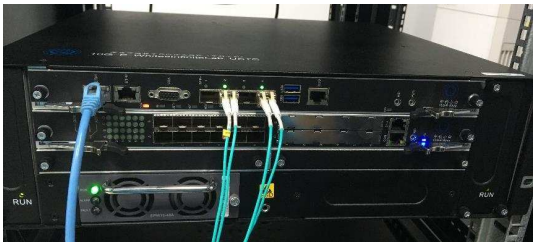
We implement PVS in a switch box that is based on ATCA and runs Linux. As shown in Fig. 4(b), the PVS system uses a standard 3U chassis that consists of two linecards, *i.e.*, for computing and switching, respectively. The switch box is equipped with 12 10GbE ports and two 1GbE ports.

III. Flow Table Management in PVS

In this section, we describe the proposed flow table management scheme for moving flow entries between *C-Tbl* and *H-Tbl* with the assistance of *EFE-Gen*.



(a) Architecture of PVS.



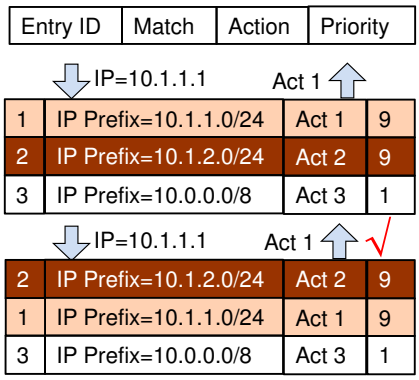
(b) Picture of PVS system.

Fig. 4. Design and implementation of PVS.

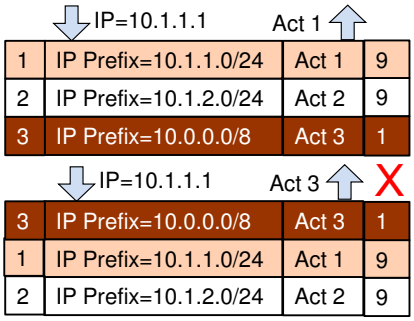
A. Problem Description

It is known that the packet forwarding performance of an SDN switch can be significantly affected by its flow table look-up scheme. This also applies to our PVS. Similar as OVS, PVS also takes the priority of flow entries into consideration. This means that if a packet matches to multiple flow entries, PVS processes it according to the one with the highest priority and ignores the remaining entries. Hence, sorting the flow entries in descending order of their priority would help to reduce the complexity of average flow table look-up. Nevertheless, in addition to the priority, the popularity of flow entries would also affect the complexity of flow table look-up. For instance, if the packets of an elephant flow match to an entry whose priority is low, PVS has to go through most of the entries for

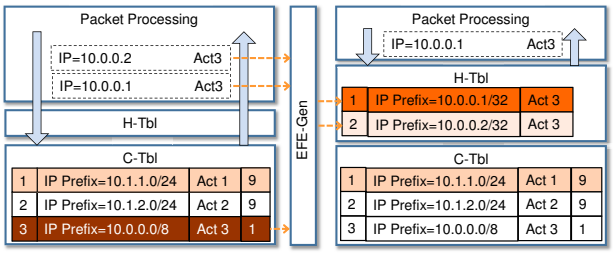
a match frequently. Therefore, the organization of flow entries in PVS should be adjusted dynamically according to not only their priority but also their popularity, which is actually our motivation to introduce *C-Tbl* and *H-Tbl* in PVS.



(a) Sort flow entries in a table (correct scenario).



(b) Sort flow entries in a table (incorrect scenario).



(c) Update flow entries in *H-Tbl*.

Fig. 5. Examples on flow table management.

Note that, a straightforward way to adjust the organization of flow entries dynamically in PVS is to sort them first by the priority and then by the popularity [32]. For example, in Fig. 5(a), *Entry* 2 is the most popular one in the table[1] and its priority is also the highest. Hence, after two rounds of sorting, it takes the first place in the table while *Entry* 1 becomes the second. Then, when a packet with IP address "10.1.1.1" comes in, the match result is correct as indicated in Fig. 5(a). However, this might not always be the case. For instance, in Fig. 5(b), even though the priority of *Entry* 3 is lower than that of *Entry* 1, it is more popular than *Entry* 1. Then, after two rounds of sorting, if PVS just takes the first match and then processes the packet with IP address "10.1.1.1" using the matched action of *Entry* 3, the action would be incorrect. A similar issue has been pointed out in [32] too, but the authors considered it as an open question for their future work.

The aforementioned issue comes from the dilemma that fast table look-up and exact flow entry search can hardly be realized in a single table. Therefore, we come up with a scheme to selectively generate exact flow entries from *C-Tbl* with *EFE-Gen* and store them in *H-Tbl*. The proposed scheme is illustrated in Fig. 5(c), where *C-Tbl* stores all the flow entries that can have masks and are sorted by the priority. When PVS determines that a flow entry in *C-Tbl* is popular, *EFE-Gen* generates one or more exact flow entries from it and stores the result in *H-Tbl*. All the exact flow entries in *H-Tbl* are sorted by the popularity, and thus they can be searched quickly with hash match to accelerate packet forwarding. Note that, since the packets from multiple flows can match to the same masked flow entry in *C-Tbl*, *EFE-Gen* can generates multiple exact flow entries from the masked one and put them in *H-Tbl*, as shown in Fig. 5(c). In addition to this negative case, we also need to address the situations in which *H-Tbl* is full or the entries in *H-Tbl* is not popular anymore. Therefore, we need to update the flow entries in *H-Tbl* dynamically.

B. Flow Table Management Algorithm

The flow table management scheme in PVS includes three steps, which are executed periodically with a fixed interval T. Specifically, we determine the popular entries in *C-Tbl* with the procedure in *Algorithm* 1. When the system first starts, *Line* 1 initializes the counters of all the flow entries in *C-Tbl* as zero and stores them in the array $c_0[\,]$. Then, in the for-loop that covers *Lines* 2-10, we find and update the popular flow entries in *C-Tbl* every time period T. Specifically, the for-loop covering *Lines* 3-9 first dumps and records the counters of all the flow entries in $c[\,]$, and then find the popular entries by comparing each entry's counter with a threshold η. Note that, we refer to the number of entries in *C-Tbl* as *C-Tbl.entryNum*. *Lines* 5-7 compare the current and previous values of the counter of each entry. If the increment on the counter is larger than η, *Line* 6 sets the *hot-flag* of the entry as TRUE. *Line* 8 updates the counter's value in $c_0[i]$.

Here, the threshold η is obtained in an adaptive manner as shown in *Algorithm* 2. First of all, *Line* 1 initializes η as the average popularity of the entries (*i.e.*, the average value of their counters) in *C-Tbl*. Then, after each time period T, *Line* 3 obtains the popularity of the entry that is the closest to the position of bottom 10% in *H-Tbl* as ε. *Lines* 4-8 update η. Specifically, if *H-Tbl* is not full, *Line* 5 sets the threshold as $\eta = \min(\eta, \varepsilon)$, and thus we can insert as many popular exact flow entries in *H-Tbl* as possible. Otherwise, we should update *H-Tbl* in a slower manner and hence, the threshold is set as $\eta = \varepsilon$ in *Line* 7. Next, after obtaining the popular entries in

[1]In Fig. 5, we use the darkness of its color to denote the popularity of a flow entry, *i.e.*, the darker its color is, the higher its popularity is.

C-Tbl, we use *EFE-Gen* to generate exact flow entries out of them. Finally, we update the entries in *H-Tbl*. Specifically, we replace the entries in *H-Tbl* with the newly-generated ones from *C-Tbl*, whose popularity is higher. Note that, to ensure the completeness of *C-Tbl*, we would not delete an entry from it even if one or more exact flow entries have been generated out of it and inserted in *H-Tbl*.

Algorithm 1: Find Popular Flow Entries in *C-Tbl*

```
1  initialize all the entries' counters in array c_0[ ] as 0;
2  for every time period T do
3      for every entry i ∈ [0, C-Tbl.entryNum] do
4          dump and record the entry's counter in c[i];
5          if c[i] − c_0[i] > η then
6              set hot-flag of the entry as TRUE;
7          end
8          c_0[i] = c[i];
9      end
10 end
```

Algorithm 2: Update Self-Adaption Threshold η

```
1  initialize η as the average popularity of entries in C-Tbl);
2  for every time period T do
3      set ε as the popularity of the entry closest to the
       position of bottom 10% in H-Tbl;
4      if H-Tbl is not full then
5          η = min{η, ε};
6      else
7          η = ε;
8      end
9  end
```

Algorithm 3: Flow Table Management

```
1  entry = LookUp(H-Tbl) with hash;
2  if entry = NULL then
3      entry = LookUp(C-Tbl);
4      if (entry ≠ NULL) AND (its hot-flag is TRUE) then
5          generate exact flow entry entry' out of entry;
6          insert entry' in H-Tbl;
7          set hot-flag of entry as FALSE;
8      end
9  end
10 if entry ≠ NULL then
11     process packet according to the action of entry;
12 else
13     send Packet_In to POF controller;
14 end
```

Algorithm 3 shows the overall procedure of flow table management in each operation. When a packet arrives, PVS first checks *H-Tbl* and if a match can be found, it processes the packet accordingly and skips *C-Tbl* to avoid unnecessary search there. Otherwise, PVS will check *C-Tbl* for a match. Note that, in *Line* 7, we set the *hot-flag* of an entry in *C-Tbl* as false, if an exact flow entry has been generated out of it and inserted in *H-Tbl*. This is performed to avoid to generate duplicate exact flow entries in subsequent operations.

IV. Performance Evaluation

In this section, we conduct experiments in a real network testbed to evaluate the performance of PVS as shown in Fig. 6. The testbed uses a POF controller which is programmed based on POX [18] to control the PVS in the data plane. We implement PVS in a switch box (*i.e.*, ATCA). And a commercial traffic generator (*i.e.*, BigTao with two 10GbE ports) is utilized to generate the testing traffic.

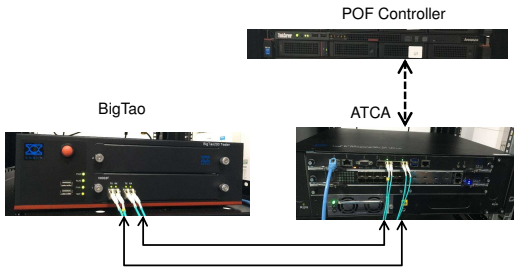


Fig. 6. The topology of the experiments.

A. Packet Forwarding Throughput

We first measure the single-port packet forwarding throughput of the PVS system. The experiments use the traffic generator to send a 10 Gbps flow to PVS, change its packet size from 64 to 1500 bytes, let the controller install one entry in *C-Tbl*, and measure the throughput of PVS.

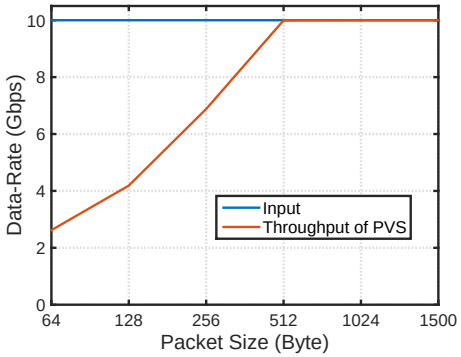


Fig. 7. Single-port packet forwarding throughput.

Fig. 7 shows the experimental results. It can be seen that the PVS system can achieve line-rate forwarding (*i.e.*, a throughput of 10 Gbps) when the packet size is longer than 512 bytes, which corresponds to a packet processing rate of 2.44×10^6 pps. However, when the packet size keeps decreasing, the throughput of PVS decreases. This is because when the packet size decreases, the PVS system needs to

process more packets every second. Note that, even though the throughput of PVS is around 2.6 Gbps when the packet size is 64 bytes, the packet processing rate is actually higher than that of the case with 512-byte packets. This is just a preliminary demonstration of our PVS system to confirm that it has the potential to achieve high-throughput protocol oblivious packet forwarding. The current PVS can still be optimized in a few perspectives, *e.g.*, incorporating parallel processing with multi-core, which will be addressed in our future work.

B. Impact Factors of Packet Forwarding Throughput

As PVS runs flow table management periodically with a fixed interval T, its throughput can be affected by T too. Hence, we send 100 flows with a total data-rate of 10 Gbps to PVS, change T within $[0.001, 1]$ ms, and obtain the throughput results in Fig. 8. We observe that when T is shorter than 0.2 ms, the frequent flow table management operations would affect the throughput of PVS significantly, but when T approaches to 1 ms, flow table management would not degrade the throughput of PVS anymore.

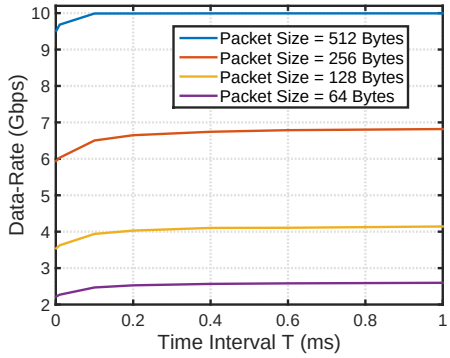


Fig. 8. Impact of flow table management interval T.

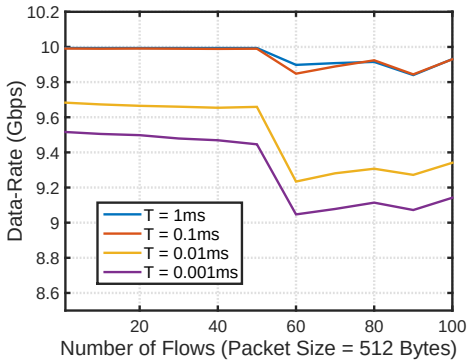


Fig. 9. Impact of the number of flows.

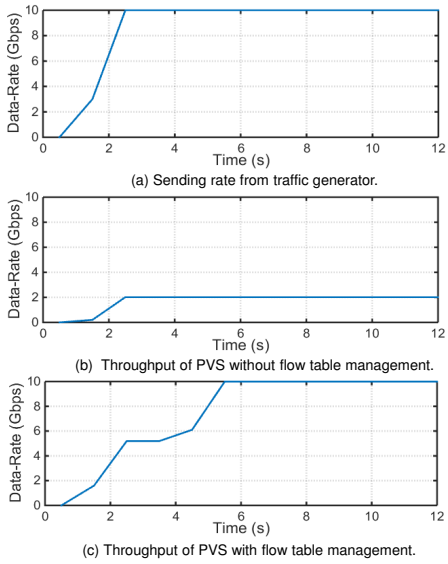


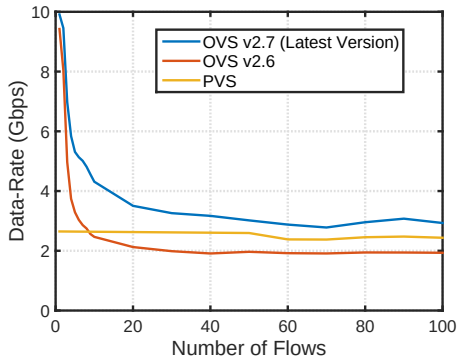
Fig. 10. Performance of flow table management.

Meantime, the number of flows can affect the performance of flow table management. This is because when a packet arrives, PVS would need to search *H-Tbl*, and the search complexity of *H-Tbl* is $O(n)$, where n is the number of flow entries in it. Hence, when there are more concurrent flows, the number of flow entries in *H-Tbl* becomes larger, which would decrease the throughput of PVS. Fig. 9 shows the experimental results on the impact of the number of flows on the throughput of PVS. Here, we set T as $\{1, 0.1, 0.01, 0.001\}$ msec, and select the packet size as 512 bytes. We can see that when T is shorter than 1 msec, the number of flows has a larger impact on the throughput of PVS than that in the cases when T is longer than 1 msec. The similar trend can also be observed in the experiment results shown in Figs. 11(a) and 11(b).

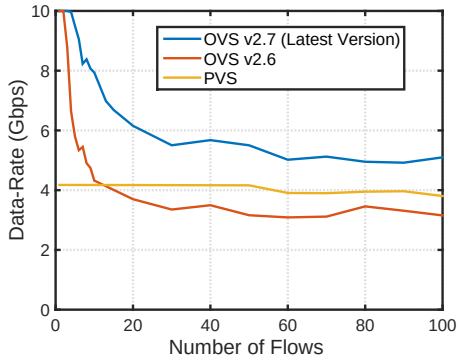
C. Performance of Flow Table Management

Next, we conduct an experiment to further verify the effectiveness of flow table management. The POF controller first installs 100 flow entries in PVS, and then we let the traffic generator send 100 flows, each of which is at 100 Mbps and uses a packet size of 512 bytes. Each flow matches to an entry in PVS, and we set $T = 3$ seconds for checking the changes brought by flow table management in an observable time scale. Fig. 10(a) shows the sending rate of the traffic generator. The throughput of PVS without the flow table management is plotted in Fig. 10(b), which indicates that the throughput of PVS can only reach around 2 Gbps. This is because without the flow table management, PVS has to go through all the flow entries in *C-Tbl* to get matches for certain packets, which increases the complexity of packet processing and thus limits its throughput. Nevertheless, when the flow table management is in place, the throughput of PVS increases to around 10 Gbps after 3 seconds in Fig. 10(c). This verifies that generating exact flow entries and inserting them in *H-Tbl* do effectively reduce the complexity of packet processing in PVS.

We also compare PVS with OVS in terms of packet forwarding rate, and Fig. 11 shows the results. Here, the experiments install 100 flow entries in an SDN switch (*i.e.*, OVS v2.6, OVS v2.7, or PVS), and let the traffic generator pump 1 to 100 flows to it to emulate various traffic conditions. The total data-rate is 10 Gbps, which is split over the flows equally in each experiment. We set the packet size as 64 and 128



(a) Packet size equals 64 bytes.



(b) Packet size equals 128 bytes.

Fig. 11. Performance comparison of OVS and PVS.

bytes to get the experimental results in Figs. 11(a) and 11(b), respectively. It can be seen that the performance trends of OVS v2.7, OVS v2.6 and PVS are similar in Figs. 11(a) and 11(b). The forwarding rate of PVS remains nearly unchanged because it adopts hash match in *H-Tbl*. However, the forwarding rate of OVS decreases with the number of flows and eventually becomes comparable to that of PVS. This is because OVS uses bitwise comparison for the entry matching in its fast path, which is less time-efficient than the hash match in *H-Tbl*.

V. Conclusion

In this work, we studied how to realize a protocol-independent data plane to better support software-defined vehicular networks, by leveraging POF. The design of a software-based PVS that supports runtime protocol customization in principle was first presented. Then, we discussed the implementation of PVS in a switch box based on ATCA, and proposed a flow table management scheme to ensure high-throughput packet forwarding. Our proposal was implemented in a network testbed for experimental demonstrations. The experimental results verified that PVS achieves line-rate packet forwarding at 10 Gbps when the packets' size is 512 bytes, and the proposed flow table management scheme is effective.

Acknowledgments

This work was supported in part by the NSFC Project 61371117, the SPR Program of the CAS (XDA06011202), the Key Project of the CAS (QYZDY-SSW-JSC003), and the NGBWMCN Key Project (2017ZX03001019-004).

References

[1] D. Kreutz *et al.*, "Software-defined networking: A comprehensive survey," *Proc. IEEE*, vol. 103, pp. 14–76, Jan. 2015.

[2] Z. Zhu, S. Li, and X. Chen, "Design QoS-aware multi-path provisioning strategies for efficient cloud-assisted SVC video streaming to heterogeneous clients," *IEEE Trans. Multimedia*, vol. 15, pp. 758–768, Jun. 2013.

[3] W. Lu *et al.*, "Implementation and demonstration of revenue-driven provisioning for advance reservation requests in OpenFlow-controlled SD-EONs," *IEEE Commun. Lett.*, vol. 18, pp. 1727–1730, Oct. 2014.

[4] S. Li *et al.*, "Flexible traffic engineering (F-TE): When OpenFlow meets multi-protocol IP-forwarding," *IEEE Commun. Lett.*, vol. 18, pp. 1699–1702, Oct. 2014.

[5] C. Chen *et al.*, "Demonstrations of efficient online spectrum defragmentation in software-defined elastic optical networks," *J. Lightw. Technol.*, vol. 32, pp. 4701–4711, Dec. 2014.

[6] Z. Zhu *et al.*, "OpenFlow-assisted online defragmentation in single-/multi-domain software-defined elastic optical networks," *J. Opt. Commun. Netw.*, vol. 7, pp. A7–A15, Jan. 2015.

[7] N. Xue *et al.*, "Demonstration of OpenFlow-controlled network orchestration for adaptive SVC video manycast," *IEEE Trans. Multimedia*, vol. 17, pp. 1617–1629, Sept. 2015.

[8] Z. Zhu *et al.*, "Demonstration of cooperative resource allocation in an OpenFlow-controlled multidomain and multinational SD-EON testbed," *J. Lightw. Technol.*, vol. 33, pp. 1508–1514, Apr. 2015.

[9] W. Fang *et al.*, "Joint defragmentation of optical spectrum and IT resources in elastic optical datacenter interconnections," *J. Opt. Commun. Netw.*, vol. 7, pp. 314–324, Mar. 2015.

[10] X. Chen *et al.*, "Flexible availability-aware differentiated protection in software-defined elastic optical networks," *J. Lightw. Technol.*, vol. 33, pp. 3872–3882, Sept. 2015.

[11] S. Li, W. Lu, X. Liu, and Z. Zhu, "Fragmentation-aware service provisioning for advance reservation multicast in SD-EONs," *Opt. Express*, vol. 23, pp. 25 804–25 813, Oct. 2015.

[12] X. Chen *et al.*, "Incentive-driven bidding strategy for brokers to compete for service provisioning tasks in multi-domain SD-EONs," *J. Lightw. Technol.*, vol. 34, pp. 3867–3876, Aug. 2016.

[13] Z. He, J. Cao, and X. Liu, "SDVN: enabling rapid network innovation for heterogeneous vehicular communication," *IEEE Netw.*, vol. 30, pp. 10–15, Jul. 2016.

[14] Z. Ning *et al.*, "A cooperative quality-aware service access system for social internet of vehicles," *IEEE Internet Things, in Press*, 2017.

[15] K. Hafeez *et al.*, "Performance analysis and enhancement of the DSRC for VANET's safety applications," *IEEE Trans. Veh. Technol.*, vol. 62, pp. 3069–3083, Sept. 2013.

[16] Z. Ning *et al.*, "Vehicular social networks: Enabling smart mobility," *IEEE Commun. Mag.*, vol. 55, pp. 49–55, Apr. 2017.

[17] ——, "Energy-aware cooperative and distributed channel estimation schemes for wireless sensor networks," *Int. J. Commun. Syst.*, vol. 30, p. e3074, Mar. 2017.

[18] S. Li *et al.*, "Protocol oblivious forwarding (POF): Software-defined networking with enhanced programmability," *IEEE Netw.*, vol. 31, pp. 12–20, Mar./Apr. 2017.

[19] J. Yu *et al.*, "Forwarding programming in protocol-oblivious instruction set," in *Proc. of ICNP 2014*, pp. 577–582, Oct. 2014.

[20] S. Li *et al.*, "Improving SDN scalability with protocol-oblivious source routing: A system-level study," *IEEE Trans. Netw. Serv. Manag., in Press*, 2017.

[21] P. Bosshart *et al.*, "P4: Programming protocol-independent packet processors," *Comput. Commun. Rev.*, vol. 44, pp. 87–95, Jul. 2014.

[22] 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.

[23] 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.

[24] D. Hu *et al.*, "Design and demonstration of SDN-based flexible flow converging with protocol-oblivious forwarding (POF)," in *Proc. of GLOBECOM 2015*, pp. 1–6, Dec. 2015.

[25] S. Li, D. Hu, W. Fang, and Z. Zhu, "Source routing with protocol-oblivious forwarding (POF) to enable efficient e-health data transfers," in *Proc. of ICC 2016*, pp. 1–6, Jun. 2016.

[26] 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.

[27] K. Han *et al.*, "Leveraging protocol-oblivious forwarding (POF) to realize NFV-assisted mobility management," in *Proc. of GLOBECOM 2017*, pp. 1–6, Dec. 2017.
[28] B. Pfaff *et al.*, "The design and implementation of Open vSwitch," in *Proc. of USENIX NSDI 2015*, pp. 117–130, May 2015.
[29] L. Molnar *et al.*, "Dataplane specialization for high-performance OpenFlow software switching," in *Proc. of ACM SIGCOMM 2016*, pp. 539–552, Aug. 2016.
[30] M. Shahbaz *et al.*, "PISCES: A programmable, protocol-independent software switch," in *Proc. of ACM SIGCOMM 2016*, pp. 525–538, Aug. 2016.
[31] DPDK: Data Plane Development Kit. [Online]. Available: http://dpdk.org/
[32] H. Chen and T. Benson, "The case for making tight control plane latency guarantees in SDN switches," in *Proc. of ACM SOSR 2017*, pp. 150–156, Apr. 2017.