Design and Demonstration of SDN-based Flexible Flow Converging with Protocol-Oblivious Forwarding (POF)

Daoyun Hu, Shengru Li, Nana Xue, Cen Chen, Shoujiang Ma, Wenjian Fang, Zuqing Zhu†
School of Information Science and Technology, University of Science and Technology of China, Hefei, China
†Email: {zqzhu}@ieee.org

***Abstract*—With the development of software-defined networking (SDN), people start to realize that the protocol-dependent nature of OpenFlow, *i.e.*, the matching fields are defined according to existing network protocols (*e.g.*, Ethernet and IP), will limit the programmability of forwarding plane and cause scalability issues. In this work, we focus on Protocol-Oblivious Forwarding (POF) [1], which can make the forwarding plane reconfigurable, programmable and future-proof with a protocol-independent instruction set. We design and implement a POF-based flexible flow converging (F-FC) scheme to reduce the number of flow-entries for enhanced scalability. To evaluate the POF system experimentally, we build a network testbed that consists of both commercial and software-based POF switches. Network experiments with real-time video streaming in the proposed POF system demonstrate that our POF-based F-FC approach can outperform conventional schemes.**

***Index Terms*—Software-defined networking (SDN), Protocol-oblivious forwarding (POF), Flexible flow converging (F-FC).**

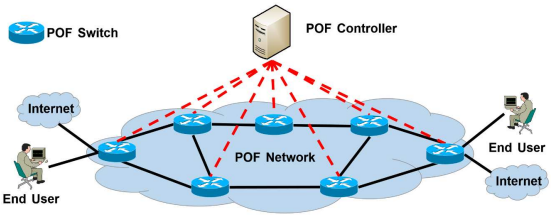
I. Instruction

Over the past decade, the fast development of the Internet has pushed the scales of end users, network devices and applications to grow exponentially. That made the Internet architecture become more and more complicated, which impedes the introduction of new protocols and slows down the support of new services. In response to these issues, software-defined networking (SDN) was proposed to make a network more programmable and application-aware, by separating its control and forwarding planes [2]. As an initial implementation of SDN, OpenFlow [3] has been developed as an open standard protocol to specify a forwarding plane abstraction together with an application programming interface (API) for forwarding devices. Specifically, OpenFlow leverages flow-based switching and enables software-defined routing, forwarding and managing by introducing a centralized controller.

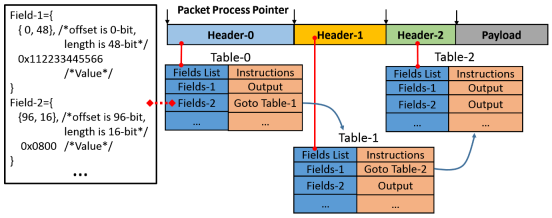
In OpenFlow, a forwarding element is abstracted as a flow table, which realizes flow processing by applying the "match-and-act" principle. The flow tables are managed by the centralized controller with an instruction set specified in OpenFlow. It is known that OpenFlow provides a power tool for the control and management of enterprise networks and has already been applied to wide-area networks (WANs) and large datacenters [4]. However, OpenFlow still has the protocol-dependent nature, *i.e.*, the matching fields in its flow tables are defined according to existing network protocols (*e.g.*, Ethernet and IP), which limits the programmability of forwarding plane and causes scalability issues [1, 5, 6]. For instance, the number of the matching fields has been increased from 12 in OpenFlow 1.0 to 44 in OpenFlow 1.5 [7]. In the flow matching, since OpenFlow switches need to understand the protocol headers to parse packets, compatibility may become a serious issue when protocols try to add or remove header fields [4]. Hence, it is desired that the programmability of networks should be further improved such that the forwarding plane can be dynamically reprogrammed to seamlessly support new protocol stacks and associated packet parsing and processing [4, 8].

Recently, several new SDN approaches have been proposed to enhance the programmability of forwarding plane, including protocol-oblivious forwarding (POF) [1] and protocol-independent forwarding (PIF) [5]. Both POF and PIF try to make the forwarding plane reconfigurable, programmable and future-proof with a protocol-independent instruction set. Meanwhile, the programming of protocol-independent packet processors (P4) [6] has been addressed to provide a framework and a high-level language for programming forwarding devices based on such an instruction set. People have designed a generic flow instruction set (FIS) for POF to make forwarding devices work as white-boxes [9]. Hence, a forwarding device can parse and process packets with the rules implemented by the POF controller and does not need any pre-knowledge on the protocol stack and associated packet handling mechanism. Consequently, the POF network can transmit packets belonging to various protocols, and the operator can even define new protocols and/or packet handling mechanisms without concerning about the compatibility in forwarding plane.

In this work, we focus on POF and demonstrate a novel SDN-based flexible flow converging scheme with it. Note that when an OpenFlow network needs to carry huge volume of traffic with many flows, the number of flow-entries implemented in the switches may increase rapidly and can cause serious scalability issues. This problem can be mitigated by compressing the flow-entries with non-prefix aggregation [10, 11]. However, it is known that non-prefix aggregation is a $\mathcal{NP}$-hard problem [10] and hence the computation time could be an issue for online operation. Previous work also suggested that one can reduce flow-entries by leveraging the flow converging scheme that classifies the flows going through the same path segment into one forwarding equivalence class (FEC) [12].



(a) Network architecture.



(b) Forwarding procedure.

Fig. 1. Overview of POF

Nevertheless, as we will show later in the paper, this scheme cannot maximize the reduction on flow-entries due to the protocol-dependent nature of OpenFlow.

We propose a POF-based flexible flow converging (F-FC) scheme to reduce the flow-entries further for better scalability, and also design the POF system to implement it. Then, we realize the proposed system in a network testbed that consists of both commercial and software-based POF switches. Networking experiments are also conducted to demonstrate F-FC and compare its performance with the conventional schemes.

The rest of the paper is organized as follows. Section II describes the working principle of POF. In Section III, we discuss the F-FC scheme and our design of the POF system to implement it. The experimental demonstration is presented in Section IV. Finally, Section V summarizes the paper.

II. Protocol-Oblivious Forwarding (POF)

A. Overview of POF

Fig. 1 shows the overview of POF. In Fig. 1(a), we can see that POF utilizes the similar architecture as that of OpenFlow, *i.e.*, a centralized controller resides in the control plane to manage the switches in the forwarding plane. However, POF uses different forwarding procedure as shown in Fig. 1(b), which is not based on the header fields defined by specific protocols. Actually, the POF switches do not need to know the forwarding protocol in advance, since the packet parsing and flow matching are directed by the POF controller through a sequence of generic key assembly and table lookup instructions [1]. This is achieved by defining the search key of a matching field as a tuple $<$*offset*, *length*$>$, where *offset* indicates the beginning position of the matching field in the packet and *length* provides the field's length. Moreover, a POF switch can also use $<$*offset*, *length*$>$ to locate the target data when it needs to manipulate the packet with certain action(s) (*e.g.*, add, delete and modify). Hence, for packet forwarding, all a POF switch needs to do is to extract the matching fields from the packet header, perform flow table lookups, and then execute the associated instructions provided by the POF controller.

B. POF Switch and Controller in Our System

Similar to the case of OpenFlow, we can use two types of POF switches to realize the functionalities described above, which are the commercial and software-based ones. In this work, we include both of them in our POF-based network testbed. For the commercial switch, we use Huawei's NE40E-X3, while the software-based switch is home-made by modifying and upgrading an existing software prototype [13].

In order to implement F-FC, we realize the POF controller by extending the POX platform [14]. Fig. 2 shows the proposed structure of the POF controller to realize F-FC. The details of the functional modules are as follows.

- **POF Manager**: It works as the arbiter in the POF controller and other modules talk with it to realize certain functions for network control and management (NC&M). Meanwhile, POF Manager monitors the network status proactively by checking TED periodically and when a path switching is needed (*e.g.*, a network failure happens), it instructs the path computation module (PCM) to calculate the new forwarding path and invokes FPM to implement the path switching. Besides, POF Manager works with a web-based graphic user interface (GUI) to illustrate the network status, and the operator can retrieve information from it through an external network management system (NMS).
- **FPM (Flow Provision Module)**: It interacts with POF switches to manage the forwarding paths of flows and is also responsible for encoding/decoding POF messages. For example, during the flow setup, FPM receives the *Packet-In* message[1] from the ingress switch and forwards the information to POF Manager for path computation. After obtaining feedback from POF Manager, it encodes the flow-entries in *Flow-Mod* messages and sends them to related switches to provision the flow. FPM also lets POF Manager update the records of flows in the traffic engineering database (TED).
- **PCM (Path Computation Module)**: It receives path computation tasks from POF Manager and optimizes each packet flow's forwarding path to achieve F-FC. Upon receiving a task, PCM calculates the path based on the current network status in TED. When the path-computation is done, it checks whether the F-FC scheme needs to be applied. If yes, it generates the F-FC labels for the flow and saves the label-flow mapping in the label database (LDB). Otherwise, the flow will be forwarded independently. The F-FC labels will be explained in detail in Section III. Finally, PCM returns the path computation results to POF Manager, which will then instruct FPM to build and send the corresponding flow-entries.
- **TED (Traffic Engineering Database)**: It stores the network status, including active POF switches, connectivity among them, bandwidth usage on each link, and the information of in-service flows. The network abstraction

[1]Note that the *Packet-In* and *Flow-Mod* messages here refer to the POF messages that are extended from those with the same names in OpenFlow.

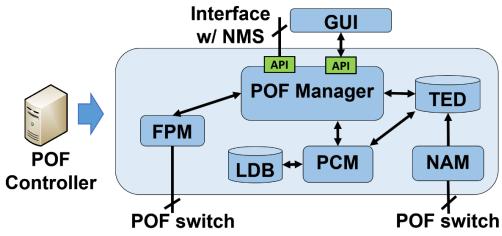


Fig. 2. Proposed structure of POF controller.

module (NAM) and POF Manager update TED in real-time to make it contain the most-updated information.

- **NAM (Network Abstraction Module)**: It collects the topology information, abstracts the POF switches, and updates the network status in TED proactively.
- **LDB (Label Database)**: It stores the label-flow mapping for F-FC and assists PCM to generate the F-FC labels.

III. Flexible Flow Converging (F-FC) with POF

A. Flow Converging with FEC

The flow converging (FC) scheme that classifies the flows going through the same path segment into one FEC was originally defined in multi-protocol label switching (MPLS) [15], which is supported by the latest OpenFlow specification. Hence, the OpenFlow controller can classify flows into FECs and assign a MPLS label to each of them [12]. However, as OpenFlow is protocol-dependent, it can only leverage one MPLS label per flow for FC (*i.e.*, no matter how many labels we assign to a flow, the OpenFlow switches will only match the first one), which restricts the scheme's flexibility. Specifically, we have the dilemma that if we classify too many flows into one FEC, the granularity of traffic engineering would become coarse, but if one FEC only contains a few flows, the number of flow-entries could not be reduced effectively.

B. POF-based Flexible Flow Converging (F-FC)

In the proposed F-FC scheme, we use two labels per flow to distinguish the flows in a hierarchic manner. Specifically, we use the first label to identify a converged super-flow, while the individual flows in the super-flow is distinguished by the second label. POF switches can determine the forwarding action(s) of a flow by matching only the first label or both ones, depending on the flow-entries provided by the POF controller. Hence, the POF switches on the shared path segment can forward the super-flow based on the first label and individual flows can be extracted from the super-flow easily by conducting forwarding action(s) based on both labels. More importantly, as the flow-entries can be updated by the POF controller dynamically, we can adjust the F-FC schemes of the flows adaptively, according to the network status.

In order to fully explore the flexibility on packet forwarding provided by POF, we make POF switches perform the packet format conversion shown in Fig. 3 on the flows that need to go through F-FC. Specifically, we first replace the Ethernet header with two F-FC labels, which have variable lengths. Each F-FC label contains two fields, *i.e.*, the *length* and *value* fields. The *length* field occupies 4 bits and indicates the length

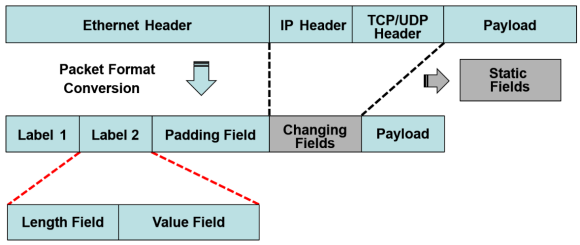


Fig. 3. Packet format conversion for F-FC.

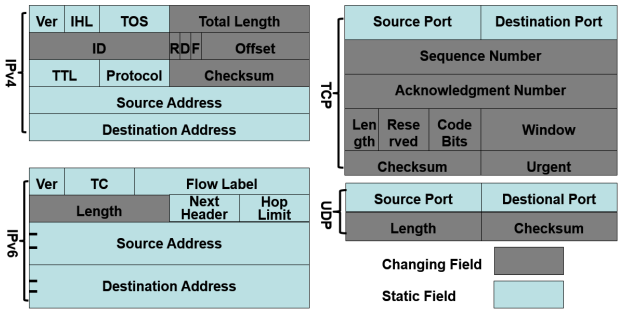


Fig. 4. Changing and static fields in IP/TCP/UDP headers.

of the *value* field in bytes. As the total length of the Ethernet header is 14 bytes, we use the *padding* field to stuff it if the two F-FC labels cannot fully fill it. The reason why we use variable-length F-FC labels here is that they bring in more flexibility. For instance, the label length can be determined based on the number of flows in the network, or the labels in different lengths are used to identify different services.

After replacing the Ethernet header with the F-FC labels, the packet format conversion performs a header compression to improve the packet flow's transmission efficiency. Before performing the header compression, we categorize the fields in IP/TCP/UDP protocol headers into two classes, *i.e.*, the changing fields and the static fields. Basically, as shown in Fig. 4, the static fields will remain unchanged when a packet being forwarding through the POF network, while the packet forwarding can vary the value of a changing field. Hence, we make the ingress POF switch use the header compression to extract all the static fields, which will be reinserted into the packets at the egress switch.

C. F-FC Algorithm

For proof-of-concept demonstration, we develop a straightforward algorithm to determine the F-FC schemes for flows. Basically, when a flow initially arrives at an ingress POF switch, the POF controller calculates the shortest feasible path for it to go through the POF network. Then, the controller checks the switches on the path one-by-one to find whether there is a flow or a super-flow that shares certain path segment with this flow and we can apply the F-FC scheme. For instance, as shown in Fig. 5, the *Flows* 7→94 and 8→94 can be converged at *Node* 37, while the super-flow of these two flows can be further converged with *Flow* 1→94 at *Node* 44. Here, for the path segment 37→44, the two flows share the same first F-FC label to indicate the super-flow, while their second F-FC labels are different to distinguish them. Similarly,

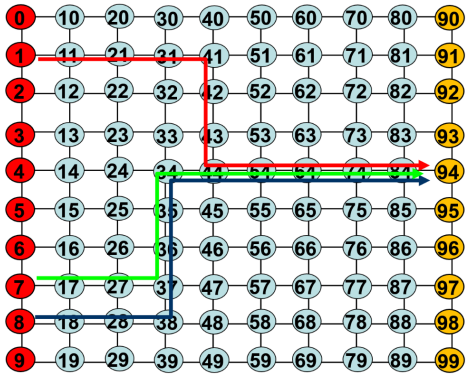


Fig. 5. Network topology of F-FC simulation.

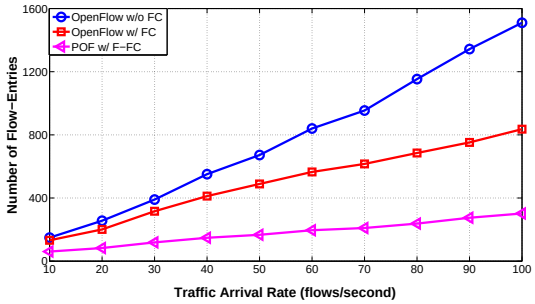


Fig. 6. Simulation results on number of flow-entries.

for the path segment 44→94 , the three flows share the same first F-FC label but use different second F-FC labels.

To show the effect of F-FC on flow-entry reduction, we conduct a simple simulation with the 10×10 grid topology in Fig. 5. The simulation randomly chooses the ingress and egress nodes of flows from *Nodes* $\{0, \cdots, 9\}$ and *Nodes* $\{90, \cdots, 99\}$, respectively. The flows are dynamically generated according to the Poisson traffic model. Here, we compare the flow-entries used in three schemes, *i.e.*, OpenFlow without FC, OpenFlow with FC, and POF with F-FC. Fig. 6 shows the simulation results and we observe that POF with F-FC uses the least number of flow-entries among the three scenarios.

D. Operation Procedure of F-FC

Fig. 7 illustrates the procedure of forwarding a TCP/IP packet flow through the POF network using the F-FC scheme.

- **Step 1**: The packet flow that comes from an end users arrives at an ingress switch of the POF network.
- **Step 2**: The ingress switch checks and finds that there is no matching flow-entry for the flow. Hence, it sends a *Packet-In* message to the POF controller and asks for the instructions on forwarding the flow.
- **Step 3**: FPM in the POF controller receives the POF message and forwards the information to POF Manager for path computation.
- **Step 4**: POF Manager invokes the path computation in PCM, which obtains the forwarding path based on the current network status in TED and also determines whether the F-FC scheme needs to be applied. If yes, it generates the F-FC labels for the flow and stores the label-flow mapping in LDB.

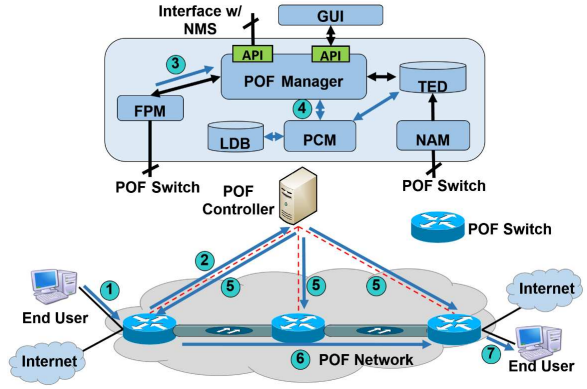


Fig. 7. Procedure for flow forwarding with F-FC in POF Network.

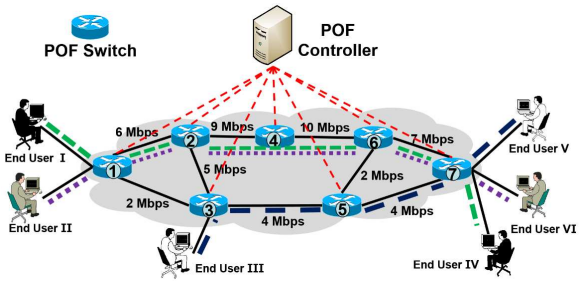


Fig. 8. Experimental setup.

- **Step 5**: POF Manager instructs FPM to encode the flow-entries for all the switches on the forwarding path, and FPM sends the flow-entries out with *Flow-Mod* messages to provision the flow.
- **Step 6**: At the ingress switch, the flow packets go through the packet format conversion for F-FC (assuming F-FC needs to be applied), and each intermediate switch forwards the packets according to the flow-entries provided by the POF controller.
- **Step 7**: Finally, at the egress switch, the packets are recovered to their original format by going through the reversed operation for the packet format conversion.

IV. Experimental Demonstrations

We implement the aforementioned POF system in a network testbed that consists of both commercial and software-based POF switches. As shown in Fig. 8, there are 7 POF switches in the testbed. Among them, *Nodes* 1-6 are realized with running software-based POF switch on high-performance Linux servers (Lenovo ThinkServer RD540), and *Node* 7 is the hardware POF switch that is based on Huawei's NE40E-X3. The POF controller is also implemented on an independent Linux server, which is directly connected to all the POF switches.

A. Experiments for Function Verification

We first perform a serial of simple experiments to verify that the POF system can perform packet forwarding as designed. Specifically, we use an IXIA traffic generator to send IP/UDP packets from *Node* 1 to *Node* 7 and the input throughput before *Node* 1 is fixed at 5 Mbps. Then, we change the input IP packet length from 128 to 1500 Bytes, and check the bandwidth utilization in the POF network when the packet

format conversion described in Section III-B is applied at the ingress POF switch on *Node* 1.

Fig. 9(a) shows the Wireshark capture of the Ethernet frame of an IPv6 packet before *Node* 1. It can be seen that the frame length is 290 Bytes, which include 14 Bytes for the Ethernet header and 276 Bytes for the IP packet. The Wireshark capture for the same packet that is received on *Node* 7 is illustrated in Fig. 9(b). First of all, the fact that we can receive the packet on *Node* 7 confirms that the POF system can correctly forward the packet. Since the packet uses a non-standard header format that does not comply with any existing protocols, the Wireshark capture in Fig. 9(b) verifies that the POF system can perform protocol-independent packet forwarding. Moreover, we observe that the frame length gets reduced to 248 Bytes in Fig. 9(b), which indicates that the header compression also works as expected.

Fig. 9(c) shows the experimental results on bandwidth utilization in the POF network for different input IP packet lengths. Here, we test both IPv4 and IPv6 packets. It can be seen that the packet format conversion can reduce the bandwidth utilization in the POF network significantly, and hence the bandwidth efficiency of packet forwarding gets improved effectively. We also notice that the bandwidth utilization increases with the packet length. This is because when the packet length is longer, we transmit less number of packets per unit time and hence the effectiveness of header compression on bandwidth reduction becomes less. For the reason that IPv6 header is longer than IPv4 header (*i.e.*, IPv6 header can be compressed more), the bandwidth utilization of IPv6 packets is less than that of IPv4 ones.

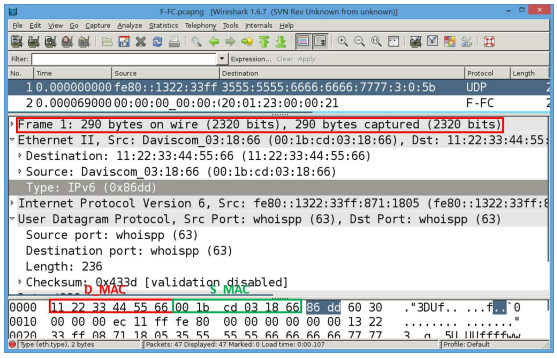
B. Experiments for Video Streaming with F-FC

We then perform experiments on video streaming with F-FC to demonstrate the advantages of the proposed POF system further. As illustrated in Fig. 8, we consider six end users who are using the POF network that has the bandwidth limitation marked on each link. Here, *Users* I and II connect to *Node* 1, *User* III connects to *Node* 3, and *Users* IV, V and VI connect to *Node* 7. The experimental scenario is as follows.

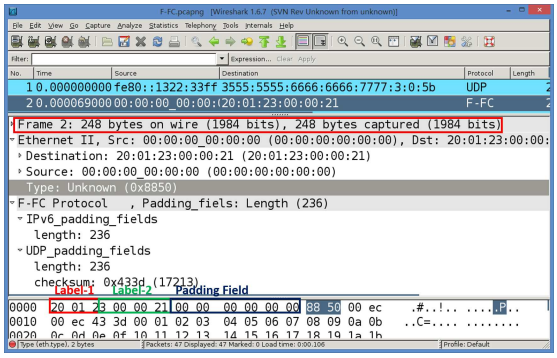
- **Step 1**: At $t = 0$ second, we start three packet flows in the POF network. *Flow* 1 is from *User* I to *User* IV for 2 Mbps file transfer, *Flow* 2 is from *User* II to *User* VI for 3.5 Mbps file transfer, and *Flow* 3 is from *User* III to *User* V for 3.5 Mbps video streaming (H.264 video sequence encoded as 1080P). The end users encapsulate all the flow packets in the IPv4/UDP format before sending them in.
- **Step 2**: At $t = 15$ seconds, we have a new *Flow* 4 that joins in and uses up all the 4 Mbps capacity on *Link* 3→5.

We compare three schemes in the experiments, *i.e.*, OpenFlow without FC, OpenFlow with FC, and POF with F-FC.

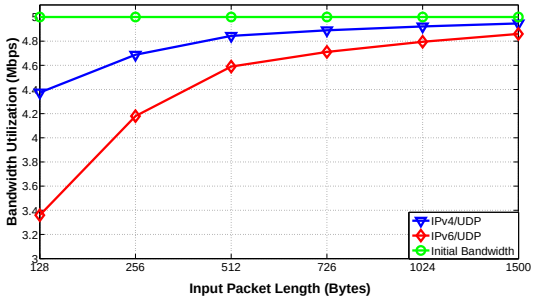
1) OpenFlow without FC: In this scheme, we downgrade the POF system to an OpenFlow one that does not have FC support. Then, the experimental scenario is as follows. Initially, *Flows* 1-3 are forwarded with the paths illustrated in Fig. 8. Then, when *Flow* 4 joins, the controller detects the congestion



(a) Wireshark capture for input IPv6 packet before *Node* 1.



(b) Wireshark capture for POF packet received on *Node* 7.



(c) Experimental results on bandwidth utilization in POF network.

Fig. 9. Results from the experiments for function verification.

on *Link* 3→5, switches *Flow* 1 to path 1→2→4→6→5→7, and makes *Flow* 3 take the path 3→2→4→6→7. Hence, the three flows can all be transferred successfully. However, in order to forward each flow correctly, each related switch has to install an independent flow-entry for it. For instance, after $t = 15$ seconds, there will be three flow-entries in each switch that is on path segment 2→4→6.

2) OpenFlow with FC: In this scheme, we downgrade the POF system to an OpenFlow one that supports FC with MPLS labels. Then, at $t = 0$ second, *Flows* 1 and 2 are converged as a super-flow and transmitted through path 1→2→4→6→7. Meanwhile, *Flow* 3 uses path 3→5→7. When *Flow* 4 joins, we cannot diverge *Flows* 1 and 2 at *Node* 6, because their super-flow is identified by a single flow-entry[2]. Then, *Flow* 3

[2]Note that we actually can diverge *Flows* 1 and 2 here, if we make the switch on *Node* 6 to match the source and destination IP addresses of the flows. However, this will make the number of flow-entries increase and let the flow-matching become as complicated as that in OpenFlow without FC. Hence, we assume that OpenFlow with FC will not perform those actions.

TABLE I
NUMBER OF FLOW-ENTRIES IN VIDEO STREAMING EXPERIMENTS.

	OpenFlow w/ FC	OpenFlow w/o FC	POF w/ F-FC
Before TE	10	13	10
After TE	13	16	11

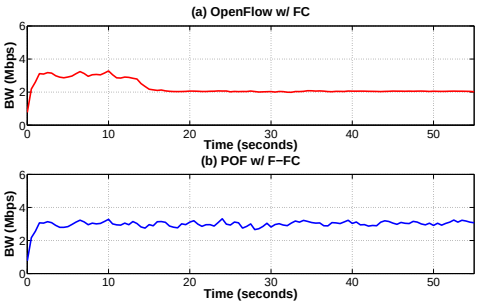


Fig. 10. Results on received bandwidth at *User* V for video streaming.

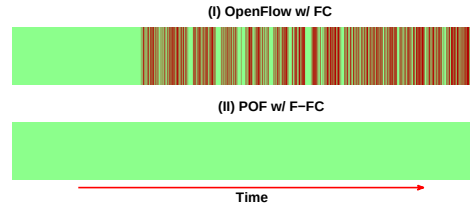
can only be switched to path 3→2→4→6→5→7. However, since the bandwidth capacity of *Link* 6→5 is 2 Mbps, the quality of video streaming gets degraded significantly.

3) POF with F-FC: Initially, the POF system serves the flows as the OpenFlow systems, and *Flows* 1 and 2 are converged at *Node* 1 with F-FC. When *Link* 3→5 becomes congested at $t = 15$ seconds, the POF controller assigns *Flows* 2 and 3 to paths 1→2→4→6→5→7 and 3→2→4→6→7, and the F-FC scheme can still be applied, *i.e.*, *Flows* 1 and 2 are converged at *Node* 1, *Flows* 1-3 are converged at *Node* 2. For example, on *Link* 2→4, we only use one flow-entry to switch the super-flow of *Flows* 1-3 with the first label, and at *Node* 6, we match both labels simultaneously to diverge *Flow* 1 from the super-flow and route it to use path segment 6→5→7.

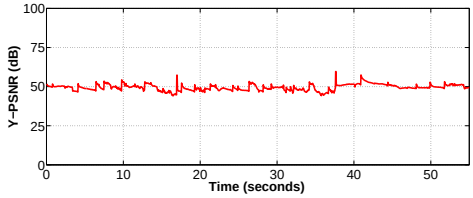
Table I summarizes the numbers of flow-entries used in the experiments, which still shows that POF with F-FC uses the least number of flow-entries. Fig. 10 shows the results on the received bandwidth at *User* V for video streaming. We can see that for OpenFlow with FC, the bandwidth gets reduced to 2 Mbps due to the bandwidth limitation on *Link* 6→5, which will degrade the quality of video streaming significantly. As shown in Fig. 11(a), after $t = 15$ seconds, the bandwidth limitation causes a lot of packet losses on *Flow* 3, which make most of the received video frames not decodable (here, a red bar means that a frame is not decodable due to severe packet loss). On the other hand, due to the F-FC scheme can arrange the forwarding paths in a better way, the bandwidth of *Flow* 3 will not decrease after $t = 15$ seconds in the POF network, and all the received video frames are decodable. The results on the luminance component's peak signal-to-noise ratio (Y-PSNR) of received video in Fig. 11(b) also indicate that the proposed POF system does not have any adverse effect on the video streaming. Note that for OpenFlow with FC, the severe packet losses after $t = 15$ seconds make the Y-PSNR results really low and hence we do not plot them to save space.

V. CONCLUSION

In this work, we designed and implemented a POF-based flexible flow converging (F-FC) scheme to reduce the flow-entries for enhanced scalability. The proposed system was



(a) Results on received video's decodability.



(b) Results on Y-PSNR of the received video in POF with F-FC.

Fig. 11. Results on performance of the received videos.

evaluated experimentally in a network testbed that consisted of both commercial and software-based POF switches. Network experiments with real-time video streaming demonstrated that the proposed POF system outperformed conventional schemes.

ACKNOWLEDGMENTS

This work was supported in part by the NSFC Project 61371117, the Fundamental Research Funds for the Central Universities (WK2100060010), Natural Science Research Project for Universities in Anhui (KJ2014ZD38), and the Strategic Priority Research Program of the CAS (XDA06011202).

REFERENCES

[1] H. Song, "Protocol-oblivious forwarding: Unleash the power of SDN through a future-proof forwarding plane," in *Proc. of ACM HotSDN 2013*, pp. 127–132, Aug. 2013.
[2] Software-defined networking (SDN) definition. [Online]. Available: https://www.opennetworking.org/sdn-resources/sdn-definition
[3] N. McKeown *et al.*, "Openflow: Enabling innovation in campus networks," *Comput. Commun. Rev.*, vol. 38, pp. 69–74, Feb. 2008.
[4] D. Kreutz *et al.*, "Software-defined networking: A comprehensive survey," *Proc. IEEE*, vol. 103, pp. 14–76, Jan. 2015.
[5] Protocol Independent Forwarding. [Online]. Available: https://www.opennetworking.org/protocol-independent-forwarding
[6] P. Bosshart *et al.*, "P4: Programming protocol-independent packet processors," *Comput. Commun. Rev.*, vol. 44, pp. 87–95, Jul. 2014.
[7] OpenFlow Switch Specifications. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.0.noipr.pdf
[8] H. Farhady, Lee, and A. Nakao, "Software-defined networking: A survey," *Compt. Netw.*, vol. 81, pp. 79–95, Feb. 2015.
[9] J. Yu *et al.*, "Forwarding programming in protocol-oblivious instruction set," in *Proc. of ICNP 2014*, pp. 577–582, Oct. 2014.
[10] C. Meiners, A. Liu, and E. Torng, "Bit weaving: a non-prefix approach to compressing packet classifiers in TCAMs," *IEEE/ACM Trans. Netw.*, vol. 20, pp. 488–500, Apr. 2012.
[11] S. Luo, H. Yu, and L. Li, "Fast incremental flow table aggregation in SDN," in *Proc. of ICCCN 2014*, pp. 1–8, Aug. 2014.
[12] B. Zhang, J. Bi, and J. Wu, "AFEC: A method of aggregating forwarding equivalence classes based on overlapped paths," in *Proc. of ICNP 2012*, pp. 1–2, Nov. 2012.
[13] POFSwitch Introduction. [Online]. Available: http://www.poforwarding.org/document/POFSwitch_Introduction.pdf
[14] POX Wiki. [Online]. Available: https://openflow.stanford.edu/display/ONL/POX+Wiki
[15] F. Le Faucheur, "IETF multiprotocol label switching (MPLS) architecture," in *Proc. of ICATM 1998*, pp. 6–15, Jun. 1998.