Embedding Bandwidth-Guaranteed Network-Based Virtual Ethernet Switches in SDN Networks

Steven S. W. Lee, Kuang-Yi Li, Wei-Kai Liu, Chen-Hua Chen, How-Jen Fang, and Ting-Shan Wong

***Abstract*—In this paper, we present a design for implementing network-based virtual Ethernet switches (NVESs) on top of a physical substrate consisting of a software-defined networking (SDN) network. An NVES has the same capability as a real Ethernet switch. A user makes a request to provision an NVES by specifying the number of ports and the maximum bandwidth for each port. The substrate network considered in this study is an Open Flow network. The entire network is controlled by an Open Flow controller. The controller is responsible for performing admission control, resource allocation, routing, address learning, and spanning tree protocol. To enhance the resource utilization of the substrate network, multipath routing is applied. Although using multipath routing can significantly improve bandwidth utilization, this approach results in out-of-order packet delivery and introduces additional error-control problems. To resolve these problems, we design a multipath agent to enhance the edge switch capability. The multipath agent is responsible for bandwidth metering, flow splitting, sequence number management, and packet reordering. We evaluated our virtual switches on top of a testbed network that includes not only commercial Open Flow switches but also a GPON-based virtual Open Flow switch. Multiple NVESs share the testbed network. The experimental results show that the user experience when using an NVES is identical to using a real Ethernet switch. In addition to the provision of guaranteed bandwidth for each NVES, traffic isolation among the NVESs is achieved. All NVESs are free of interference from others in the same network.**

***Index Terms*—Multi-path routing, network-based virtual Ethernet switch, OpenFlow, software-defined networking (SDN).**

I. Introduction

BY PROVIDING highly flexible controllability, software-defined networking (SDN) is considered one of the most important technologies for new generation networks. Network virtualization enables multiple users to effectively share a physical substrate network. The deployment of virtual networks through SDNs has received much attention in recent years. A comprehensive survey on the design of SDN network virtualization can be found in [1]. Because OpenFlow is the major commercially available SDN technology, most SDN virtual networks designs are based on OpenFlow.

Manuscript received May 23, 2017; revised September 8, 2017; accepted October 13, 2017. Date of publication October 22, 2017; date of current version November 16, 2017. This work was supported in part by the Ministry of Science and Technology, Taiwan under Grants 105-2218-E-194-002 and 105-2221-E-194-015. *(Corresponding author: Steven S. W. Lee)*

The authors are with the Department of Communications Engineering, National Chung Cheng University, Chiayi 62102, Taiwan (e-mail: ieeswl@ ccu.edu.tw; nickykyli@gmail.com; liu923526@gmail.com; k951874632@gmail.com; max7325111@gmail.com; wongtingshan111@gmail.com).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JLT.2017.2765464

The simplest type of virtual network is called bandwidth slicing. FlowVisor [2] was the first OpenFlow virtual network to use bandwidth slicing, in which the topology of each virtual network is essentially the same as the substrate network. The bandwidth of each link is partitioned to support all the virtual networks. A more advanced version of virtual network design provides each user with a custom network topology [3], [4]. Because the topologies of the virtual networks and the substrate network differ, the control functions are more complex than under pure bandwidth slicing.

In addition to virtual network design, it is possible to transform a physical network into a virtual switch. In [5], we successfully implemented a GPON-based SDN-enabled virtual switch. In this design, the entire GPON network becomes a single OpenFlow switch, hiding the GPON operational details from users. Thus, users can use standard OpenFlow protocols to control and manage the virtual switch just as if it were a real OpenFlow switch, without knowing that it is a GPON network.

This work takes a further step to implement virtual Ethernet switches on top of a substrate OpenFlow network. In addition to sharing the physical substrate network to reduce costs, the benefit of providing each user with a virtual Ethernet switch stems from the simplicity of controlling and managing the device. Ethernet switches are known for their plug-and-play capability. To use Ethernet, a user needs only to connect his or her devices to the switch. No additional switch configuration is required. Just like an Ethernet switch, a virtual Ethernet switch possesses the same plug-and-play capability. Users do not need to have any SDN knowledge to use a virtual Ethernet switch. Technical details such as routing, bandwidth management, and error control are handled automatically by the network.

The term "virtual Ethernet switch" has also appeared in cloud systems such as VMware. Unlike our virtual switch, which acts on top of a substrate network, the virtual Ethernet switches in these systems function within a single physical computer and connect multiple virtual machines inside the same system. To differentiate our approach, we hereinafter term our switch a network-based virtual Ethernet switch (NVES). Because our approach must control multiple distinct nodes and manage network resources among them, our implementation of an NVES is much more challenging than that of a virtual switch in a single computer.

A substrate SDN network can contain multiple NVESs; however, for simplicity, we depict only one 4-port NVES in Fig. 1. In a real switch, each port has a constant port rate. Traffic going through any port must meet this port rate limitation. An NVES

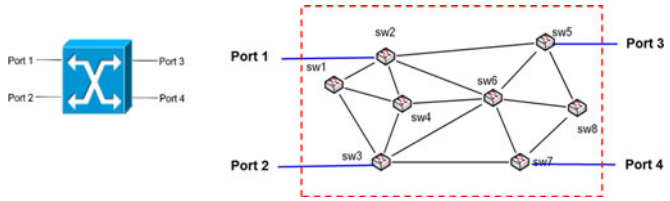

Fig. 1. Real and virtual switches: (a) a 4-port real switch and (b) a 4-port NVES implemented on a physical substrate network.

also must enforce a bandwidth usage constraint on all ports; however, unlike a real Ethernet switch, for which the port rate is limited to only a few choices (e.g., 100 Mbps or 1 Gbps), an NVES user can flexibly specify not only the desired number of ports but also the maximum bandwidth for each port in the virtual switch.

In addition to the port speed constraint, traffic isolation and a bandwidth guarantee inside a virtual switch are the key technical issues when designing an NVES. Just like a real Ethernet switch, an NVES must be non-blocking. For a user, a traffic pattern is called eligible if the input traffic at each input port and the aggregated traffic toward any output port are no greater than the port rate limitation. The virtual switch should not drop any packet when the input traffic is eligible. For example, assume that the port rate of each port in the NVES shown in Fig. 1 is 100 Mbps. If the amounts of traffic coming from port 1, port 2, and port 3 toward port 4 are 10, 20, and 30 Mbps, respectively, the traffic pattern is eligible because not only the input traffic at each port but also the aggregated traffic toward port 4 is less than 100 Mbps.

NVESs offer the benefits of network resource sharing, traffic isolation, and guaranteed bandwidth. Unlike virtual networks, which requires each user to operate their own network using a controller, virtualizing a virtual network as a virtual Ethernet switch minimizes the effort required to use it. The possible applications of NVES include sharing a substrate network among different departments in an enterprise or enabling a network operator to share a network to support a multi-tenant environment.

In summary, our major contributions are:

1) We propose a network based virtual Ethernet switch for OpenFlow networks. To our knowledge NVES is the first proposal to embed multiple-bandwidth-guaranteed virtual Ethernet switches in an OpenFlow network.
2) We design and implement the control and data plane of NVES. To enhance network bandwidth utilization, a multi-path agent is implemented at each edge switch to achieve bandwidth-guaranteed multi-path routing while eliminating the packet out-of-order problem
3) We conduct experiments to evaluate the performance of the proposed NVES. The experimental results show that the performance of an NVES is comparable to a real Ethernet switch. Although multiple NVESs share a single OpenFlow network, a guarantee of traffic isolation prevents interference between different NVESs.

The remainder of this paper is organized as follows. In Section II, we discuss related works and the design challenges for NVES. In Section III, we introduce the control and management functions for the NVES system. Section IV describes how to set up flow entries in OpenFlow switches to identify a routing in the network. Section V presents the detailed design of the proposed MP agent. In Section VI, we report the experimental results and make performance comparisons between our NVES and a real Ethernet switch. Finally, concluding remarks are presented in Section VII.

II. Related Works and Design Challenges for NVES

Virtualization models for SDN-enabled optical connectivity service providers were discussed in [6]. Two models, a single virtual node model and an abstract link model, for topology virtualization with abstracted topology have been introduced. The single virtual node model presents the whole virtualized network as one single node. This abstraction makes it easy to create connections by connecting ports and hides the complexity of the optical domain. Although the model was proposed in [6], the implementation details were missing. Our NVES shares the same concept as the single virtual node model; however, the single node model was proposed for multi-wavelength optical networks. The authors of [6] indicated that this model makes creating connections between ports nontransparent for the user due to the fact that not all combinations of ports have to be part of a valid lightpath. NVES does not suffer from the problem. The only constraint for users is the port speed.

To guarantee non-blocking for any eligible traffic pattern, we must determine the routing paths and reserve bandwidth in the substrate network. A similar problem has been considered in designing a virtual private network (VPN), and the problem is called the hose model in [7]. In the model, each VPN endpoint v is specified as the maximum total bandwidth $b_{in}(v)$ that v will send into the network at any time and the maximum total bandwidth $b_{out}(v)$ that v will ever receive from the network at any time. It is not necessary to specify the exact demands between any endpoint pair. The capacity reserved for the VPN must be sufficient for every eligible traffic pattern that is consistent with the values of $b_{in}(v)$ and $b_{out}(v)$.

By examining the routing and bandwidth reservation problem for provisioning NVES, we find that the problem is the same as the hose model problem considered in VPN networks. The output of the hose model algorithm provides the routing and required bandwidth reservation of each link to provision the virtual Ethernet switches. An algorithm for the hose model is proposed in [8] in which a pair of endpoints is configured with multiple paths to enhance the bandwidth utilization on the physical network. The algorithm determines the splitting ratios for multi-path (MP) routing. We can apply the same algorithm in designing an NVES.

OpenFlow is a technology for realizing SDN [9], [10], and numerous vendors provide OpenFlow switches. Our implementation lies on top of OpenFlow. Because the flows between any pair of two ports of an NVES can traverse multiple paths, the simplest approach is to split the traffic according to the desired ratios by OpenFlow switches. However, flow splitting is not supported by most commercial OpenFlow switches. In

addition, because different paths having different delays, directly splitting traffic on multiple paths leads to the packet out-of-order problem. In TCP, out-of-order packet arrival causes the end stations to trigger congestion control by throttling down the TCP window, resulting in significant throughput degradation.

Multiple research studies have aimed to resolve the out-of-order problem for TCP using multiple paths. SACK TCP can address multiple packet losses and a SACK extension, DSACK [11], can detect and mitigate spurious retransmissions. RR-TCP [12] can adjust the dupthresh value dynamically to avoid unnecessary timeouts. MMPTCP [13] can also adaptively adjust the dupthresh value to avoid the re-ordering problem.

Although several TCP variants already handle the packet re-ordering problem, those special versions of TCP are not widely used in current operating systems, including Windows and Linux. In our NVES design, we do not wish to introduce any constraints that force users to use any specific transport protocol. Our system must guarantee in-sequence packet delivery so that users can use any transport protocol they wish without causing any out-of-order problems.

To employ bandwidth-guaranteed MP routing while eliminating the out-of-order problem, we propose embedding an MP agent in each edge switch. In our network, we classify nodes into edge switches and core switches. The edge switches are those nodes connected to user devices. For each pair of ports, the MP agent performs flow splitting at the ingress switch and traffic aggregation at the egress switch. In addition, it also provides bandwidth and error-control functions. The MP agent does not differentiate packets based upon their transport protocols. The order in which packets leave an NVES from the egress switch is exactly the same as the order in which they enter at the ingress switch.

Fig. 2 shows an example of routing via multiple paths. In this example, two virtual switches share the substrate SDN network. The routing paths between port 1 and port 4 of User A are highlighted. In Fig. 2(a), the flow-splitting ratios at each node are provided by the solution to the hose model problem. Although the hose model solution provides flow-splitting ratios only at each node, the sub-flow paths and flow-splitting ratios for each sub-flow path can be derived. We can decompose the results of Fig. 2(a) into three sub-flow paths (labeled Path 1, Path 2, and Path 3 in Fig. 2(b)). The corresponding flow-splitting ratios for the three sub-flows are 0.5, 0.2, and 0.3, respectively. Thus, we can expect the average numbers of packets traversing those sub-flows to be 5:2:3. The MP agent in Edge Switch 1 is responsible for accurately splitting the traffic on the sub-flow paths according to the desired ratios, and the MP agent in Edge Switch 7 is responsible for aggregating traffic from the three sub-flows and delivering in-sequence packets to the user. The NVES must control the bandwidth usage accurately to guarantee that no interference occurs among the flows generated by different users.

Approaches have been published in the literature to support MP routing for standard TCP connections are candidates for realizing MP agents. In other words, we can directly encapsulate the arrival packets inside a TCP packet at the ingress switch and decapsulate the original packet at the egress switch. Those

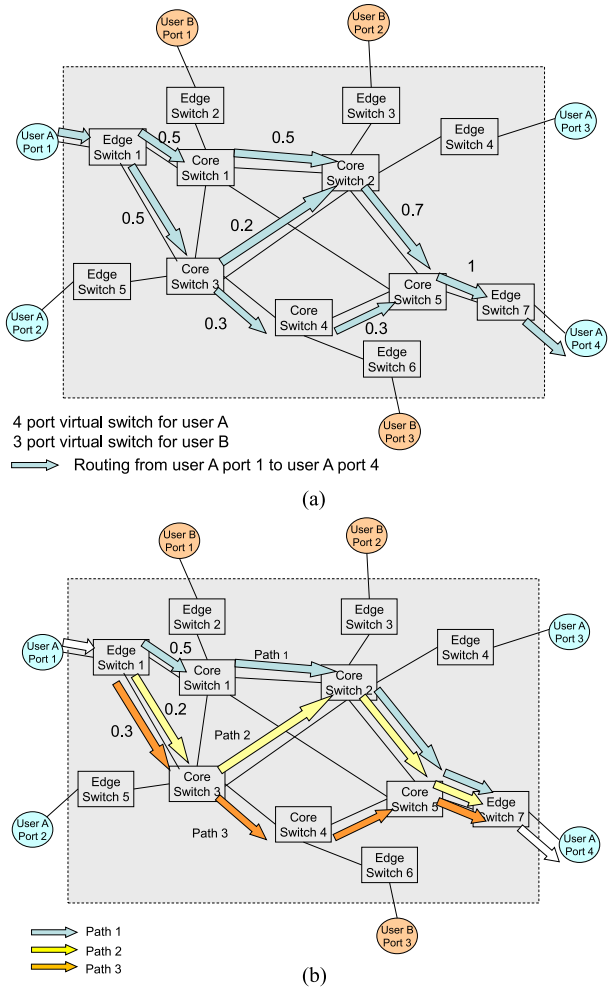

Fig. 2. An example for MP routing in an NVES. (a) Node traffic splitting ratios. (b) Derived sub-flow paths and path-splitting ratios.

candidate multi-path TCP protocols guarantee in-order packet delivery between the two end-to-end edge switches.

Flare [14] is one of the candidate approaches for MP routing. Flare employs *flowlet switching*. Flowlets are bursts of packets from a flow separated by sufficiently large time gaps. In Flare, multiple disjoint paths are provisioned between two edge nodes. At any time, only one of those paths is used to transmit TCP packets. The edge nodes are responsible for detecting the round-trip delay times of those paths. The edge node can transmit a newly arrived packet on a new path only when the time gap between the previously transmitted packet and the newly arrived packet is larger than the difference of the round-trip times between the old path and the new path. According to a set of given splitting ratios, each path takes turns transmitting a fixed quota of packets. As a result, Flare can spread traffic among multiple routing paths by following the specified splitting ratios.

CONGA [15] and HULA [16] also support MP routing. Both approaches detect network state and employ flowlet switching to realize congestion-aware load-balanced routing. To achieve high throughput, CONGA has been implemented in custom ASICs. A prototype of HULA was implemented using the recently

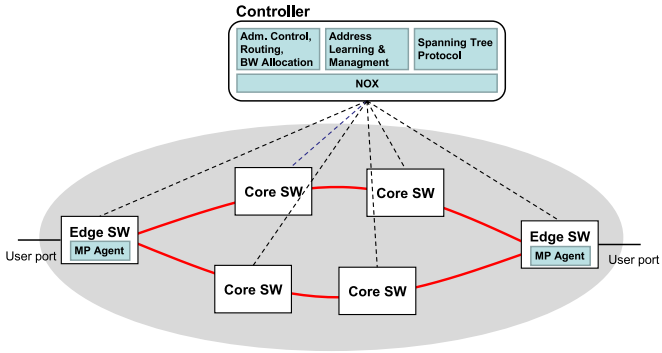

Fig. 3. Architecture of the NVES.

proposed P4 language [17]. This language facilitates implementing HULA in new generation programmable switches such as RMT [18], FlexPipe [19], and XPlant [20].

MPTCP [21] is another approach for enabling MP routing. MPTCP provides a sublayer beneath TCP to divide a TCP connection into sub-flows. Each sub-flow can take a different path in the network. A sub-flow has its own congestion and error controls. At the receiver end, the MPTCP integrates packets coming from the sub-flows to guarantee in-sequence packet transmission to the TCP layer. The study in [22] shows that MPTCP always outperforms single-path TCP.

Although Flare, CONGA, HULA, and MPTCP can use multiple paths, they have some limitations for NVES. Flare, CONGA, and HULA allow only one active path at any moment; thus, they do not fully use the capacity of the MPs. Another Flare limitation is that it requires all the paths to be mutually disjoint. MPTCP always attempts to maximize packet transmissions along the sub-flows. Because MPTCP does not use fixed ratios to split traffic among sub-flows, it is not able to take advantage of the solution obtained from the hose model. CONGA and HULA are unsuitable for NVES for the same reason. In addition, the experimental results in Section VI show that the throughput from applying MPTCP in the MP agent is unsatisfactory. Consequently, we propose a new design for MP agents in this work.

III. Control and Management

The architecture of the entire system is shown in Fig. 3. An MP agent is embedded into an edge switch to perform flow splitting at the ingress switch and traffic aggregation at the egress switch. This agent guarantees in-sequence packet delivery to the user. In the current implementation, the controller is a NOX [24] with application programs to perform control and management functions. The core switches are commercial OpenFlow switches. We implemented the edge switches on top of PCs running Open vSwitch (OVS) [25]. In this section, we describe the design of those functions in the controller. Then, we introduce the design of the MP agent in the following section.

A. Admission Control, MP Routing, and Bandwidth Allocation

When a request to provision a set of new NVES is received, the controller runs an optimization algorithm to jointly determine the admission control, routing, and bandwidth allocation for the NVES. To avoid interrupting the services of existing users, we do not reconfigure the routing and bandwidth reservations for the NVESs that are already in the network. The new request is accepted only when the substrate network has sufficient capacity. If the request is accepted, then the output of the algorithm provides the MP routing and bandwidth reservation in the network.

The routing and bandwidth reservation for the virtual switch is essentially a hose model problem. We extend the algorithm proposed in [8] to determine the MP routing, link bandwidth reservation, and flow-splitting ratios for those MPs. The algorithm includes two parts, Problem P1 and Problem P2. Each part is a solution procedure for an optimization problem. The problem input of the first part includes the routing and the flow-splitting ratios on the MPs. The solution of the first part of the algorithm is used to obtain eligible traffic demands that maximize the required bandwidth on each link. If the given MP routing and flow-splitting ratios make the worst-case demand pattern meet the physical capacity constraint on each link, then an optimal solution has been obtained, and the program terminates; otherwise, the demand pattern that violates the capacity constraint is included in the constraint set of the second part's problem. The problem solution of the second part problem determines a new set of MP routing and splitting ratios. These two problems are solved iteratively until an optimal solution to the hose model problem is obtained.

The original formulations shown in [8] consider the deployment of one VPN. We extend the model to include multiple NVESs. We also modify the second part of the model to reduce the computation time.

The notations used in the formulations are summarized below.

Input values:

N	set of nodes in the network; In addition to switch nodes, for ease of problem formulation, each NVES port is considered to be a virtual node. Each virtual node x is connected to the access port of corresponding edge switch through two directional virtual link $\hat{l}_x^{in}$ (entering node x) and $\hat{l}_x^{out}$ (leaving node x).
L	set of links in the network (including physical links and virtual links);
$L_n^{in(out)}$	set of links entering (leaving) node n;
C_l	physical capacity of link l;
K	set of existing NVESs;
γ	index for the new request NVES;
Q_k	set of ports in NVES k;
$\tilde{f}_{uv}^l$	flow splitting ratio on link l for all port pairs of existing NVESs, where $u, v \in Q_k, k \in K$;
$g_u^{in(out)}$	Input (output) bandwidth requirement of port u;

Decision variables for Problem P1:

d_{uv}	traffic demand for the port pair (u, v) that maximizes bandwidth usage on the target link;

Decision variables of Problem P2:

α	utilization of the most congested link;
f_{uv}^l	flow ratio of port pair (u, v) on link l;

Problem P1 ($l, \boldsymbol{f}$)**:**

$$\text{maximize} \sum_{k\in K\cup\{\gamma\}} \sum_{u\in Q_k} \sum_{v\in Q_k} d_{uv} f_{uv}^{l}$$

subject to:

$$\sum_{v\in Q_k} d_{uv} \le g_u^{in} \qquad \forall u \in Q_k, k \in K\cup\{\gamma\} \tag{1}$$

$$\sum_{u\in Q_k} d_{uv} \le g_v^{out} \qquad \forall v \in Q_k, k \in K\cup\{\gamma\} \tag{2}$$

$$d_{uv} = 0 \qquad \forall u \in Q_k, k \in K\cup\{\gamma\} \tag{3}$$

$$d_{uv} \ge 0 \qquad \forall u, v \in Q_k, k \in K\cup\{\gamma\} \tag{4}$$

Given the flow splitting ratio $\boldsymbol{f}$, Problem P1 involves finding an eligible traffic demand matrix that maximizes the bandwidth utilization of link l. Constraints (1), (2), and (3) specify the bandwidth limitations of each NVES port. Constraint (4) requires that the demand between any two ports be non-negative.

Problem P2 (**D**, t)**:**

$$\text{minimize } \alpha$$

subject to:

$$\sum_{k\in K\cup\{\gamma\}} \sum_{u\in Q_k} \sum_{v\in Q_k} d_{uv}^{i} f_{uv}^{l} \le \alpha C_l \quad \forall 1 \le i \le t, l \in L \tag{5}$$

$$\sum_{l\in L_n^{out}} f_{uv}^{l} - \sum_{l\in L_n^{in}} f_{uv}^{l} = 0 \quad \forall u, v \in Q_\gamma, u \ne v, \; n \in N\backslash\{u, v\} \tag{6}$$

$$\hat{f_{uv}^{l_u^{out}}} = 1 \qquad \forall u, v \in Q_\gamma, u \ne v \tag{7}$$

$$\hat{f_{uv}^{l_u^{in}}} = 0 \qquad \forall u, v \in Q_\gamma, u \ne v \tag{8}$$

$$\hat{f_{uv}^{l_v^{in}}} = 1 \qquad \forall u, v \in Q_\gamma, u \ne v \tag{9}$$

$$\hat{f_{uv}^{l_v^{out}}} = 0 \qquad \forall u, v \in Q_\gamma, u \ne v \tag{10}$$

$$f_{uv}^{l} = \tilde{f}_{uv}^{l} \qquad \forall u, v \in Q_k, k \in K, l \in L \tag{11}$$

Given a collection of demand matrices $\mathbf{D} = \{\mathrm{D}_1, \mathrm{D}_2, \ldots, \mathrm{D}_t\}$, Problem P2 finds a new flow splitting ratio that minimizes the bandwidth utilization on the most congested link. For the original second model in [8], the objective function was to minimize the weighted sum of the total bandwidth consumption. In the new model, the problem tries to spread the flows on the network to consume as little bandwidth as possible on the most congested link. Therefore, when this solution is input into the first problem, it increases the possibility of finding a feasible solution, thus reducing the number of iterations required to solve these two problems.

The left-hand side of Constraint (5) denotes the total bandwidth usage on each link l, where d_{uv}^{i} is the element at the u-th row and v-th column of matrix $\mathrm{D_i}$. Constraints (6)–(10) jointly enforce the flow conservation law. Because we do not reconfigure the NVESs that are already in the network, Constraint (11) requires that the flow splitting ratios for the existing NVESs be maintained without change.

Algorithm AC & R()

```
1.  Begin
2.    Initialize f_uv^l , ∀u, v ∈ Q_r ; /* initialize the flow ratio for
      each port pair of the new NVES. The initial value is obtained
      based on the shortest path routing algorithm */
3.    t:=1;
4.    D:=Φ;
5.    repeat
6.      flag:=True;
7.      for each link l ∈ L
8.        Solve Problem P1 (l, f) to obtain D_t;
9.        /*where D_t[u][v]:= d_uv for all port pair (u, v) */
10.       if Σ_{k∈K∪{γ}} Σ_{u∈Q_k} Σ_{v∈Q_k} d_uv f_uv^l > C_l then
11.         D:=D ⋃ {D_t};
12.         Solve Problem P2 (D, t) to obtain (α, f);
13.         if α>1 then
14.           admission-control:=Reject;
15.           return; /* new NVES rejected */
16.         else
17.           flag:=False;
18.           t:=t+1;
19.           break for loop;
20.         end else;
21.       end if
22.     end for
23.   until flag=True;
24.   admission-control:=Accept;
25. End.
```

Fig. 4. The admission control and routing algorithm.

Fig. 4 summarizes the whole algorithm. The initial routing for the new NVES is obtained by executing the shortest path algorithm. Given the flow splitting ratio $\boldsymbol{f}$, problem P1 determines the worst-case traffic demand that maximizes the bandwidth utilization of link l. When the demands violate the link capacity constraint, the demand matrix is added to the set **D**. Given **D**, the task of problem P2 is to find a new flow-splitting ratio. If any demand matrix exists that prevents problem P2 from finding a solution without violating link capacity, the program terminates and the new NVES is rejected. Otherwise, the program continues until a feasible solution is obtained.

We must reserve bandwidth on each link for each virtual switch to guarantee the required bandwidth. The amount of reserved bandwidth can be obtained from the final solution of the hose model problem; that value is the maximum bandwidth required for each link under the worst-case input eligible traffic pattern. Any user generating a traffic volume larger than the maximum reserved bandwidth must be the result of excess bandwidth usage. Therefore, the excess traffic should be dropped to prevent interference with other virtual switches.

B. Address Learning and Management

By examining the destination media access control (MAC) address of a received packet in the forwarding table, an Ethernet switch determines the correct output port to which to forward

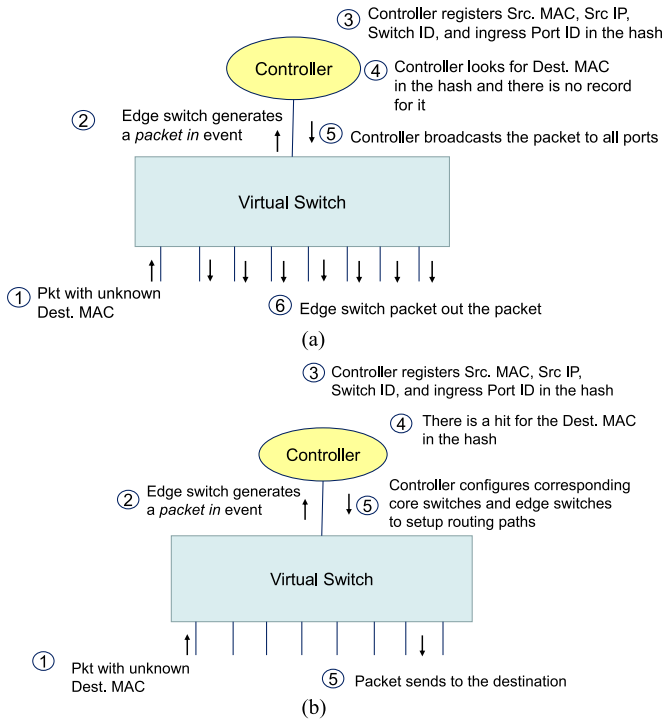

Fig. 5. The procedures for handling a packet with an unknown destination MAC address. (a) The hash table does not have a record for the destination MAC address. (b) The hash table has a record for the destination MAC address.

the packet. The technical term for the forwarding table is the forwarding information base (FIB). The FIB is maintained through the MAC learning process. As a packet is received, the switch adds its source MAC address and the corresponding port in the FIB. Switches automatically remove FIB entries after a fixed time if they have not received any further packet carrying the same source MAC address. Therefore, if a host station does not send traffic for a designated period, the switch deletes the corresponding forwarding entry.

In our NVES implementation, the controller employs a hash table to manage MAC addresses and provide switch locations and port IDs to facilitate routing. When a new MAC address is discovered at an edge switch, the source MAC address is registered in the hash table. In addition to the source MAC address, we also include the source IP address and the switch and port identifications in the hash. In our system, we borrow the source IP field to carry a routing tag to facilitate packet routing inside the network. We change the address back to its original value at the egress switch. The detailed operations will be explained in the next section. Just like a real Ethernet switch, if a record is not refreshed for a certain period of time, the record is removed from the hash table.

If an Ethernet switch receives a packet with an unknown destination MAC address, the switch broadcasts the packet to all its ports except the one that the packet came from. Our virtual switch follows the same rule. Fig. 5 depicts the operations of an NVES when an edge switch receives a packet with an unknown destination MAC address. An unknown packet means that the NVES has not configured routing paths for that flow in the network. This condition results in a table miss for the flow entries when the packet arrives at the edge switch. The edge switch encapsulates the unknown packet inside a *Packet in* message and then delivers it to the controller. When the controller receives the message, it first searches for the destination MAC address in the hash table. When there is a match, the controller can identify the location of the egress switch for the packet. Based on that condition, the controller sets up MP routing for the flow by configuring the core switches and edge switches. However, when no match for the destination MAC address exists in the hash table, the controller broadcasts the packet to all the ports of this NVES using the *packet out* command.

A user is free to change which ports his/her devices are connected to. Just like a real Ethernet switch, the NVES must detect port changes and reconfigure the routing paths accordingly. In the third step of Fig. 5, the controller not only checks whether the destination MAC appears in the hash but also checks whether the MAC address and its incoming port for the packet are the same as shown in the hash table. When a mismatch occurs, the NVES assumes that the user is using a new port for the connection; therefore, the controller not only updates the record in the hash but also reconfigures the flow entries in the OpenFlow network to change the packet delivery routes to the correct location.

Like a real Ethernet switch, an NVES should be able to handle packets with both broadcast and multicast addresses. Multicast traffic is handled by the same procedures as broadcast traffic. In the next paragraph, we describe how to handle broadcast traffic.

A packet with a destination MAC address of "*FFFFFFFF FFFF*" is called a broadcast packet, which is sent to all NVES ports. In our implementation, if an incoming broadcast packet has not been seen previously by the NVES, the packet is sent to the controller. After completing the MAC learning process, the packet is sent to all the NVES ports by the controller using the OpenFlow *packet out* command. To reduce the future loading on the controller handling this broadcast flow, the controller sets up paths in the OpenFlow network so that any further traffic belonging to the same broadcast flow is handled by the OpenFlow network directly.

The detailed procedure for handling a broadcast packet is shown in Fig. 6. As shown in Fig. 6(a), if no matched flow entry exists for this incoming packet, a *packet in* event is triggered at the edge switch. The packet is sent to the controller, which registers its addresses and IDs in the hash table. The controller then sends a copy of the packet to each output port of the NVES. In addition, the controller configures the corresponding flow entries in the substrate network to set up routing paths for this broadcast flow. Subsequently, as shown in Fig. 6(b), there will be a match at the edge switch the next time such a broadcast packet arrives. Thus, an OpenFlow network can directly handle the broadcast flow without involving the controller. This approach significantly reduces the controller burden for handling broadcast packets.

C. Spanning Tree Protocol

We do not preclude users from connecting their NVES to other real Ethernet switches. In that case, an NVES is participating in an Ethernet network and must execute the IEEE

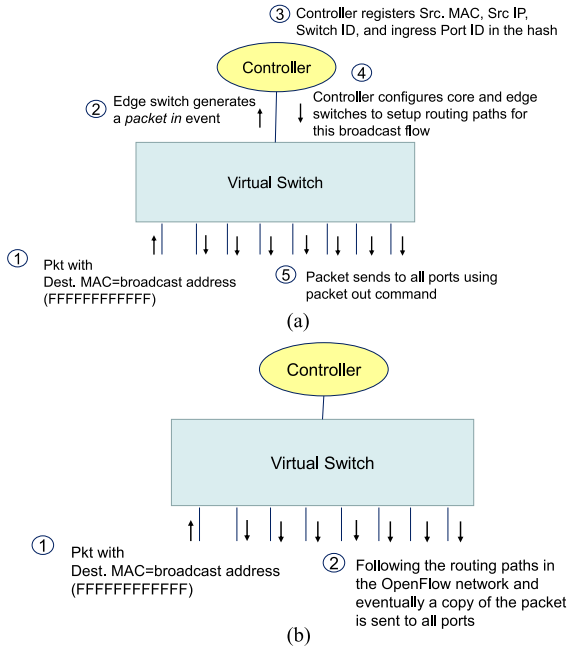

Fig. 6. The procedure for handling a packet with a broadcast address in the destination address field. (a) There is no flow entry at the edge switch for the broadcast packet. (b) There is a flow entry at the edge for the broadcast packet.

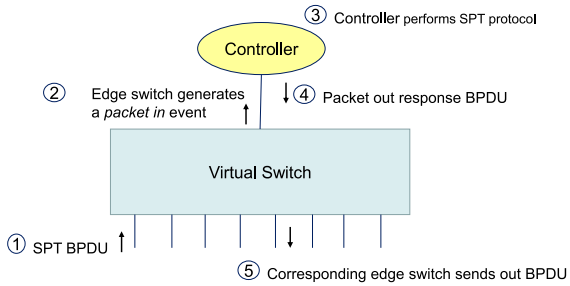

Fig. 7. The procedures for handling an STP BPDU.

802.3 spanning tree protocol (STP) [26] to guarantee loop-free forwarding. In our system, the spanning tree module in the controller is responsible for analyzing and responding to the spanning tree bridge protocol data unit (BPDU). Fig. 7 shows the operations for an NVES to handle the BPDU. Because each BPDU is carried by a frame with MAC address = *01 80 c2 00 00 00 (or 01:00:0C:CC:CC:CD)*, the controller installs the following flow entry at each edge switch to intercept all BPDUs and redirect them to the controller.

Match Dest. MAC = 01 80 c2 00 00 00

Action send the BPDU packet to the controller

Based on the results of running STP, the controller not only transmits BPDUs to its corresponding neighbor Ethernet switches but also configures NVES to prevent any traffic from going through blocked ports to guarantee loop-free forwarding in an Ethernet network.

IV. Routing Tag Assignment and Setting Flow Entries

As described in the previous section, an NVES applies MAC learning to enable the system to identify a flow. To facilitate routing in the core network, each sub-flow path that connects a pair of ports in an NVES is assigned a path ID. The ingress switch attaches a path ID to each incoming data packet; then, core switches makes their routing decisions based only on the path ID. Moreover, the ingress switch assigns each packet a sequence number to help with packet re-ordering at the egress switch.

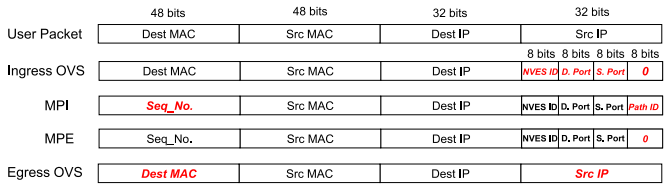

Fig. 8. Address fields used for routing and in-sequence control in an NVES.

We implemented our edge switches on PCs running OVS. An MP agent is embedded on top of OVS to perform flow splitting at the ingress switch and traffic aggregation at the egress switch. At the ingress switch, the MP agent (abbreviated as MPI) and the OVS jointly assign a routing tag and sequence number for each incoming packet. The routing tag is a path ID attached to the packet that identifies the sub-flow path inside the network. At the egress switch, the MP agent (called MPE) and the OVS remove the path ID and the sequence number. More specifically, for each packet, the path ID and sequence number are carried in the source IP and destination MAC fields, respectively. Although the original source IP address and destination MAC address are replaced with the path ID and sequence number at the ingress switch, their original values are recovered at the egress switch.

In our design, we take full advantage of the redundancy between the source (destination) IP and source (destination) MAC. When the controller performs the address learning process, it knows the mapping between IP address and MAC address. Thus, given either value, we can determine the other. Accordingly, inside the NVES, we can adopt one field for control purposes and change it back to the original value at the egress switch. The detailed operations for address field replacements in both edge switches are shown in Fig. 8. Because core switches do not modify any packet fields, Fig. 8 does not include any core switches.

After a routing has been determined, the controller downloads the flow entries to the OVSs at both edge switches and the corresponding core switches. The controller also configures the MPI and the MPE modules for sub-flow scheduling. The OVS at the ingress switch is responsible for replacing the source IP field of an incoming packet with an NVES ID, D. port ID, and S. port ID. Those IDs are 8 bits long. The NVES ID is used to identify the virtual switch, while the D. port and S. port are the destination port ID and source port ID of the NVES, respectively. For instance, suppose a packet has the source MAC a, the destination MAC b, the source IP c, and the destination IP d. Assuming that this packet enters NVES 2 from port 3 and should be switched to port 4, the flow entry at the OVS of the ingress switch is as follows:

Match: "Src MAC field = a and Dest MAC field = b", Action: Write "NVES ID = 2, D. port ID = 4, S. port ID = 3, and path ID = 0" to Src IP field".

After attaching the NVES ID, D. port ID, and S. port ID, the ingress OVS sends the packet to its MPI module to determine the corresponding sub-flow path for routing. At the MPI module, each packet is assigned a sequence number (Seq. No.). The number is inserted into the destination MAC field of the packet. According to the routing path, MPI also inserts the Path ID into the last 8 bits of the source IP address field. In this example, we assume that the MPI assigns path ID $= j$ for the flow.

By checking the NVES ID and Path ID, a core switch can determine how to forward the packet. A core switch does not change any field of a received packet. Following the previous example, the controller installs the following flow entry to enable the core switch to deliver the example packet to its output port k:

Match: "Src IP field: NVES ID = 2, D. port ID = 4, S. port ID = 3 and path ID = j", Action: "output the packet to port k".

Finally, the egress OVS rewrites the original destination MAC and source IP back to the packet. Assume that the user port is at the q-th port of the OVS; then, the flow entry at the OVS becomes

Match: "Src MAC = a and Dest IP = d", Action: Write "Dest MAC = b and Src IP = c" and output the packet to port q.

As a result, a user can receive a packet with the address fields as it enters the NVES at the ingress switch.

Note that the MPI agent at the ingress switch does not examine the transport layer protocol type of the incoming packets. The agent only attaches the path ID and sequence number for the incoming packets. Based on the sequence number, the MPE at the egress switch performs packet re-ordering to guarantee in-sequence packet delivery to the users. Accordingly, the order of packets leaving the NVES is the same as that of the packets entering the NVES. We present the detailed design of the proposed MP agent in the next section.

V. Design and Implementation of a BSRG MP Agent

The goal of the proposed MP agent is to achieve Bandwidth and Splitting Ratio Guaranteed (BSRG) MP routing. We embed a BSRG MP agent in each edge switch to perform bandwidth metering, flow splitting, sequence number control, and packet re-ordering. Because the flow splitting on MPs is performed at edge switches, the core switch needs only to enforce bandwidth control by dropping packets to prevent excess bandwidth usage. In this section, we present the detailed design and implementation of the functions inside the MP agent.

Fig. 9 shows the architecture of a BSRG MP agent. Each agent includes an MPI module and an MPE module. There are three types of ports in an edge switch. Two are physical ports, including the ports connecting to user devices and the ports connecting to the core switches. The other ports are virtual ports. Each virtual port represents a sub-flow going through this edge switch. At the ingress switch, any packet coming from the user is delivered to the MPI module. This module spreads a stream of incoming packets onto multiple sub-flow paths based on the hose model solution. At the egress switch, the MPE receives the packets from the multiple sub-flows and transmits them to the end hosts in the correct order.

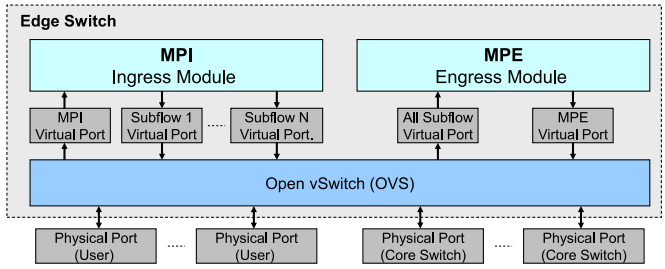

Fig. 9. Architecture of the proposed BSRG MP agent.

The details of MPI and MPE are described in detail below.

A. MPI Module

The MPI module includes the following three functions:

1) *MPI Packet Receiver*: The receiver performs queue management to handle incoming packets from the MPI virtual ports.
2) *Scheduler*: Based on the splitting ratios for the sub-flow paths, the scheduler determines the routing path for each incoming packet. The scheduler also prepares a sequence number and path ID for each packet. For each pair of ports in an NVES, the ingress and the egress switches perform the same scheduling algorithm. Therefore, both switches have the same mapping between the packet sequence number and the path ID. An egress switch fully understands which sub-flow path a packet should have come from. This approach eliminates the need for synchronization between the ingress switch and the egress switch.
3) *MPI Packet Transmitter:* This function delivers packets to the correct OVS ports.

B. MPE Module

The MPE module includes the following six functions:

1) *MPE Packet Receiver*: The receiver is responsible for receiving incoming packets from the virtual port. This function performs queue management for incoming packets.
2) *Scheduler*: The scheduler is the same as that used in the MPI module.
3) *Error Control*: Because the MPE must guarantee in-sequence packet delivery to the user, an error-control mechanism is required to prevent an infinite wait for a lost packet. Because different sub-flows have different delays, a packet with sequence number i could arrive at the MPE before a packet with a sequence number smaller than i. In the MPE, each sub-flow has its own arrival queue. When a new packet arrives, the MPE examines the arrival queues to see whether there is a missing packet with a sequence number smaller than the just-arrived packet. We denote the deferred packet arrival on sub-flow q to be $\sigma(q)$. Deferred packet arrival $\sigma(q)$ represents the degree of out-of-sequence packet arrival on sub-flow q. For example, $\sigma(q) = 3$ means that packet i carried by sub-flow q is still missing when packet $i+3$ carried by sub-flow p $(p \neq q)$ has

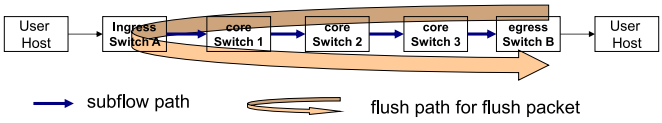

Fig. 10. Example of a flush path for packet loss detection.

arrived. In other words, sub-flow q has a missing packet that is 3-packets late to arrive in its correct order. When a new incoming packet arrives, MPE updates σ for all the sub-flows. When the $\sigma(q)$ value is larger than a threshold value, $\sigma_{max}(q)$, it is likely that the missing packet carried by sub-flow q is lost inside the network. A flushing mechanism is activated to check whether the missing packet in sub-flow q is actually lost or just late. The reference threshold value σ_{max} for packet loss judgment can be predetermined by the network manager or acquired through a learning process. In the current version of our implementation, this value is learned dynamically to reflect the delay situation of different sub-flow paths based on the statistics of recent packet arrival.

4) *Flush*: Each sub-flow path has a corresponding flush path. When a new sub-flow path is initialized, the OpenFlow controller also initializes its flush path. The flush path is a loopback path that follows the same physical route as the sub-flow. A flush packet traverses the flush path to make sure that a data packet is actually lost. A flush packet is identified by its flush path ID in its packet header. Fig. 10 shows an example. In this example, the sub-flow path is IS A→ CS 1→CS 2→CS3→ES B, and the flush path for the sub-flow is ES B→CS 3→CS 2→ CS 1→IS A → CS 1→CS 2→CS3→ES B. The flush packet is assigned the highest priority in the direction from the egress switch to the ingress switch. An OpenFlow switch transmits the flush packet with minimum delay toward the ingress switch. In other words, the path segment ES B→CS 3→CS 2→ CS 1→IS A is assigned the highest priority so that the flush packet will arrive at the ingress switch with minimum delay. The priority of the remaining segment of the flush path from the ingress switch toward the egress switch is the same as its sub-flow path. Because the flush packet follows the same path in the direction from the ingress switch to the egress switch, the MPE knows that a data packet k has been lost if it is not received before the flush packet returns to the egress switch. Thus, the MPE can skip packet k and deliver packet $k+1$ to the end host. If the lost packet is a TCP packet, this packet will be handled by the end-to-end TCP in the two user hosts. Using this mechanism, the egress switch can avoid waiting for a lost packet indefinitely.

5) *Receive Timer*: This timer is used to prevent deadlock when a packet loss occurs near the end of a TCP connection. Because our flush-based error-control packet is triggered by an incoming packet, if a packet loss occurs at the end of a TCP connection, the flush mechanism would never be activated because no further packet arrivals would occur. To resolve this problem, we implement a receive timer at the MPE. When a packet is delivered to the end user, we reset the timer. When the timer expires, the packets that have already arrived at the MPE are unconditionally sent to the user. The receive timer is also used to prevent loss of a flush packet.

6) *MPE Packet Transmitter:* This function delivers packets to the MPE virtual port for packet transmission to the user host.

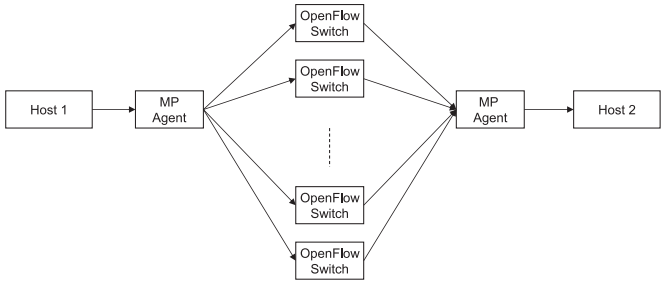

Fig. 11. Experimental configuration for throughput comparisons using different MP agents.

VI. Experimental Results

We conducted three sets of experiments to evaluate the performance of the proposed NVES system. In the first set of experiments, we compared the throughput performances using MP agents implemented with different technologies. In the second set of experiments, we embedded our NVES on top of an OpenFlow testbed to evaluate the performance of its data plane for bandwidth control accuracy and traffic isolation among virtual switches. In the final set of experiments, we examined the performance of the control plane in various scenarios and measured the response times. We also conducted pressure tests to evaluate the performance under heavy loads on the control plane.

A. Throughput Comparisons Using Different Approaches to Implement MP Agents

We implemented MP agents based on Flare, MPTCP, and the proposed BSRG approach. Flare requires the routing paths to be mutually disjoint. MPTCP cannot spread traffic on subflows following a given set of splitting ratios; it always tries to maximize the throughput for each sub-flow. To meet those specific requirements for Flare and MPTCP, we used OpenFlow switches to set up mutually disjoint paths. The bandwidth of each path is controlled by the meter function in OpenFlow switches. In other words, the disjoint path constraint for Flare and the splitting ratio constraint for MPTCP are enforced by the physical network, not by the MP agent itself.

We used HP5900 OpenFlow switches to set up the experiment shown in Fig. 11. Host 1 uses the standard TCP protocol to transmit a 5-GB file to Host 2. The MP agent is responsible for performing in-sequence packet delivery among the multiple paths. In this experiment, we evaluated the throughput on two and four paths. The splitting ratios were 1:1, 1:4, and 1:9 for the two-path case and 1:1:1:1, 1:4:6:9, and 1:4:9:36 for the four-path case. The total capacity for each experiment

TABLE I
PERFORMANCE COMPARISONS AMONG FLARE, MPTCP, AND THE PROPOSED BSRG MP AGENTS ON TWO DISJOINT PATHS

Target Subflow Ratio	Multipath Method	Throughput (Mbps)	Completion time (sec)	Subflow 1 Ratio	Subflow 2 Ratio
1:1	FLARE	92.9	462.3	0.4997	0.5003
	MPTCP	94.2	456	0.4999	0.5001
	BSRG MP	94.1	456.4	0.5000	0.5000
1:4	FLARE	83.3	515.5	0.2001	0.7999
	MPTCP	92.1	466.2	0.2017	0.7983
	BSRG MP	93.7	458.4	0.2000	0.8000
1:9	FLARE	81.3	528	0.1145	0.8855
	MPTCP	78.1	550.1	0.1240	0.8760
	BSRG MP	93.4	459.7	0.1000	0.9000

TABLE II
PERFORMANCE COMPARISONS AMONG FLARE, MPTCP, AND THE PROPOSED BSRG MP AGENTS ON FOUR DISJOINT PATHS

Target Subflow Ratio	Multipath Method	Throughput (Mbps)	Completion time (sec)	Subflow 1 Ratio	Subflow 2 Ratio	Subflow 3 Ratio	Subflow 4 Ratio
1:1:1:1	FLARE	88.7	484.4	0.2499	0.2500	0.2500	0.2501
	MPTCP	93.6	458.7	0.2506	0.2494	0.2497	0.2502
	BSRG MP	94.1	456.5	0.2500	0.2500	0.2500	0.2500
1:4:6:9	FLARE	87.5	491	0.0502	0.2000	0.3000	0.4499
	MPTCP	92.5	464.1	0.0532	0.2051	0.3053	0.4364
	BSRG MP	94	457.1	0.0500	0.2000	0.3000	0.4500
1:4:9:36	FLARE	74.3	577.8	0.0202	0.0802	0.1801	0.7195
	MPTCP	89.7	478.6	0.0225	0.0863	0.1929	0.6983
	BSRG MP	94	457.1	0.0200	0.0800	0.1800	0.7200

was set to 100 Mbps. Therefore, for a connection with sub-flow ratios x:y in the two-path case, the corresponding meters for those two sub-flows became $100 \times x/(x + y)$ Mbps and $100 \times y/(x + y)$ Mbps. In addition to the observed traffic, we also injected background traffic to fully occupy the remaining capacity of each link.

The experimental results for the two-path case shown in Table I indicate that our BSRG MP has the best performance in terms of throughput. The agent accurately follows the given ratios to spread the traffic on those paths. Although Flare can use multiple paths for packet transmission, it is unable to fully use the available network bandwidth. The performance of the MPTCP-based MP agent strongly depends on the available bandwidth of the sub-flows. We observed that the throughput of MPTCP is severely degraded when the difference of the available bandwidths between the two paths becomes large. This situation was also observed in [23]. In our application, the MPTCP-based MP agent can be viewed as a standard TCP overlaid on an MPTCP network. Due to the larger delay experienced in the overlay network, the performance is even worse than in the case that directly applies MPTCP at both end hosts. This limitation prevents using MPTCP to implement the MP agents. Because the four-path results are similar results to those of the two-path cases, we present the experimental results in Table II as a reference.

B. Performance Evaluation of MP Agent

We further examine the performance of the MP agent in different scenarios using the topology shown in Fig. 12, which includes two edge switches (S1 and S4) and two HP 5900 switches (S2 and S3). We evaluate the performance between the pair of

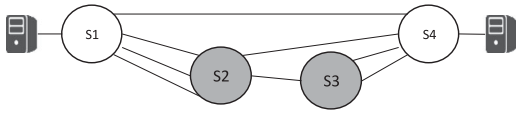

Fig. 12. Experimental configuration for performance evaluation of MP agent.

edge switches. The two edge switches are off-the-shelf commodity PCs. Each PC has an Intel Core i7-7700K processor running at 4.50 GHz. This processor includes a quad core CPU. The system is supported by 8 GB DDR4 of on-board memory. The operating system is Ubuntu 14.04 with OVS 2.3.9.

We first examine the performance as the number of connections supported by each NVES increases. In this experiment, the edge switch includes four NVESs. Each NVES is assigned a 50 Mbps port at each edge switch. For ease of rate control, the UDP protocol is applied. The connection of each NVES is supported by four subflows traversing through 1 hop, 2 hops, or 3 hops between the two edge switches. The topology shown in Fig. 12 is used to generate various routing patterns for the subflows. Some of the subflows are aggregated to a common link and some are delivered to different links after traversing a core switch. The results shown in Fig. 13(a) indicate that increasing the number of connections for each NVES does not degrade the performance at the MP agent.

In the second experiment, we examined the MP agent by increasing the number of NVESs. In this experiment, each NVES was assigned a port speed of 50 Mbps. The results shown in Fig. 13(b) indicate that the edge switch can support up to three NVESs without performance degradation. When the number of NVES reaches five, the throughput is reduced due to the limited computational power of the PCs.

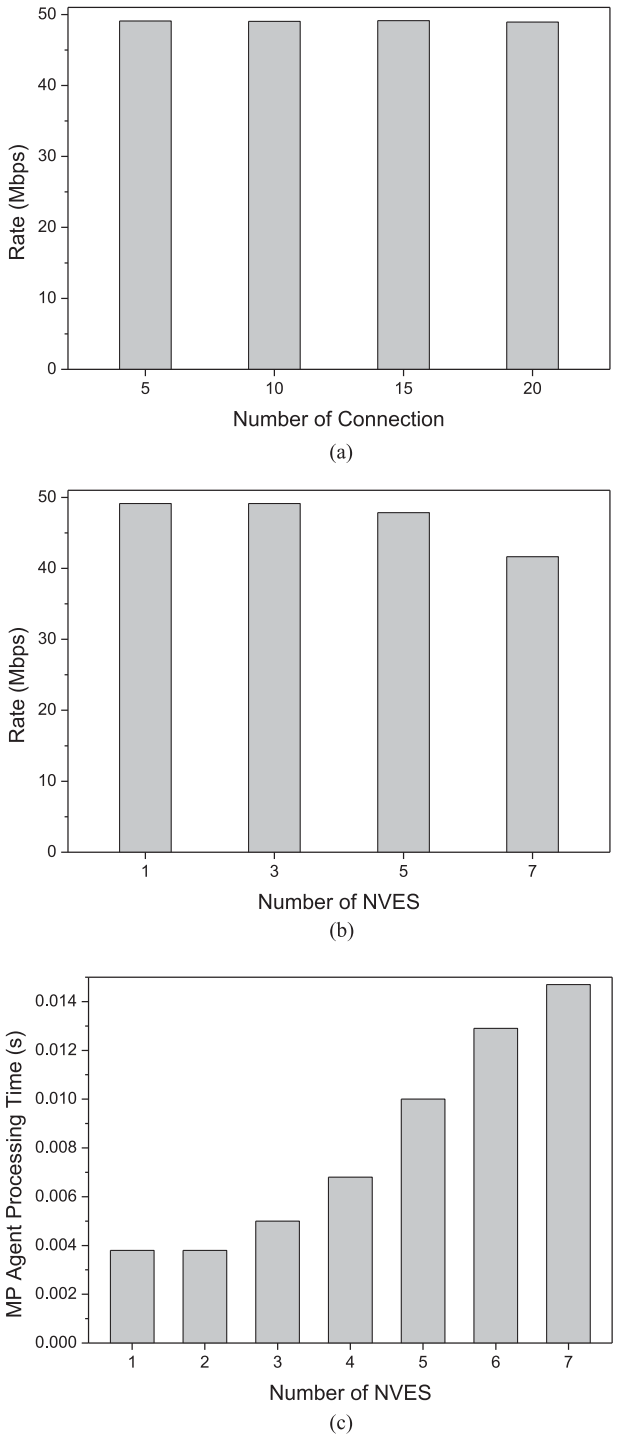

Fig. 13. Experimental results of the performance evaluation of MP agents. (a) Throughput under different numbers of connections. (b) Throughput under different number of NVESs. (c) Processing delay under different NVESs.

In the third experiment, we examined the average processing delay of packets traversing through an ingress edge switch and an egress switch. The results are shown in Fig. 13(c). When the number of NVES is small, the delay is less than 10 ms. However, when the number of NVES is larger than five, the lack of computational power causes increases in the delay.

Next, we discuss the queue required to support the MP agent. In the current implementation, the CPU examines the network interface by invoking the libpcap 1.59 system call to check whether any incoming packet exists. MPI and MPE run on different threads. Because MPI handles one packet at a time, only a small buffer is needed. The major memory consumption comes from the MPE. Because the MPE needs to handle the out of order problem, packets must be kept in a dynamic linked list before delivery to the user. The maximum buffer required to support one subflow in MPE is $b = \sigma_{max} \times l_{\max} + t_{flush} \times r_{sub\_flow}/8$ byte, where σ_{max} is the maximum out-of-order tolerance defined in Section V-B, $l_{\max}$ is the maximum packet length (i.e., 1518 bytes), t_{flush} is the round-trip delay for the flush packet, and r_{sub_flow} is the rate of an incoming subflow. Therefore, the maximum buffer for a single subflow is 258 KB when $\sigma_{max} = 5$ packets, $t_{flush} = 20$ ms, and the subflow rate is 100 Mbps.

C. Performance of NVES Data Plane

To evaluate the performance of the NVES, we implemented an OpenFlow testbed and embedded NVES into the network. The substrate OpenFlow network is shown in Fig. 14. In the network, there are 10 core switches and 12 edge switches. The core switches (CS1–CS10) are HP 5900 OpenFlow switches. There are two types of edge switches. Edge switch (ES1) is a GPON-based OpenFlow enabled switch. The detailed implementation of this switch can be found in [5]. The other edge switches are PC-based software switches running OVS. We included our BSRG MP agent in every OVS edge switch and GPON-based edge switch to make them NVES-enabled edge switches. The system controller is a Linux-based PC running the NOX OpenFlow control program [24]. We included the admission control and routing module, the address learning and management module, and the STP module in the application layer on top of the NOX controller. We consider seven NVESs sharing the substrate OpenFlow network. For simplicity, all ports of a NVES have the same rate limitation. The detailed rates for each NVES port are listed in Fig. 14.

The optimization algorithm shown in Section III is used to determine the MP routing, flow-splitting ratios, and bandwidth reservation for each NVES. The results are configured into the substrate network; thus, the required bandwidth is guaranteed for all seven NVESs. The meters in the core OpenFlow switches guarantee the bandwidth requirement of each NVES. Because the software switches in the edge network do not provide meter functions, we implemented a leaky-bucket-based meter to regulate the rate limitation at the access ports in each edge switch. Packets that use in excess of the allocated bandwidth are dropped by the meters in the core and the edge switches to prevent interference with other NVESs.

We first examine only NVES 1 to see whether it operates correctly under various input situations. Three cases are considered here: (1) the data rate at each input port and the aggregated rate at the observed output port are within the port rate limitation; (2) the rate at each input port is within the rate limitation but the aggregated rate at the destination output port is larger than the

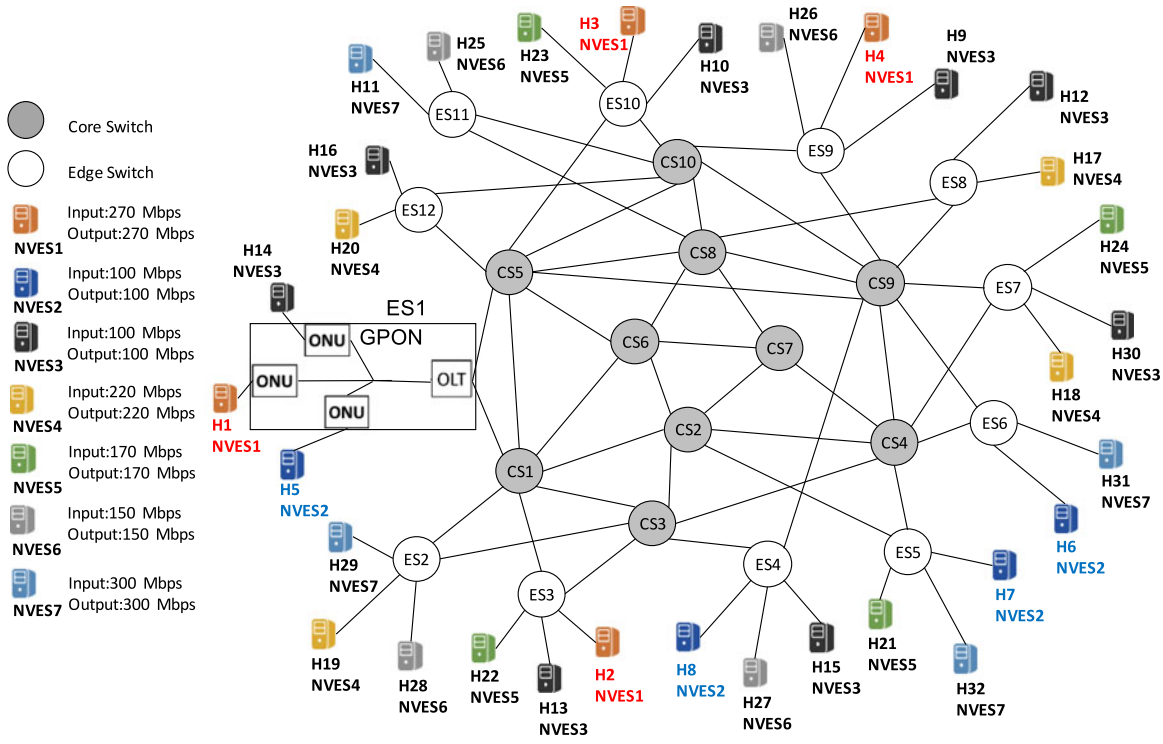

Fig. 14. Experimental network.

TABLE III
INPUT RATES AT SOURCE AND EXPECTED RECEIVING RATE AT DESTINATION

Case	Output rate at Host 2	Output rate at Host 3	Expected Input rate at Host 1
1	100	110	210
2	120	200	270
3	280	350	270

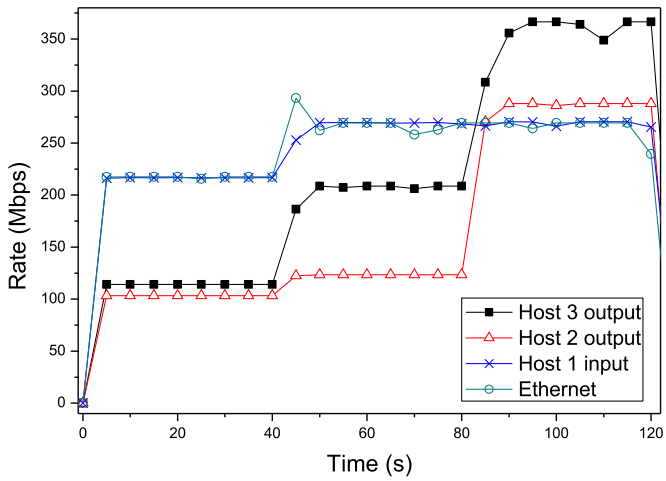

Fig. 15. Experimental results for bandwidth control on NVES 1.

rate limitation; and (3) both traffic arriving at the input port and the aggregated traffic at the output port are larger than the rate limitation.

In this set of experiments, the source stations are Host 2 and Host 3, and the destination station is Host 1. We examine the amount of traffic entering the network from Host 2 and Host 3 and the total traffic leaving the network toward Host 1. We use iperf [27] to generate UDP packets to facilitate control of the rates at the input ports. The detailed input patterns are shown in Table III and Fig. 15 presents the experimental results. From 0 s to 40 s, Host 2 and Host 3 generate 100 Mbps and 110 Mbps traffic, respectively. The pattern meets the Case (1) situation. Because traffic arriving at each input port is within the rate limitation (270 Mbps for NVES 1), we can expect no packet drops inside the network. The experimental results show that the data rate received at Host 1 is close to 210 Mbps, which is exactly the summation of the traffic coming from both hosts.

We further examine Case (2) during the period from 40 s to 80 s. In this case, the offered loads from Host 2 and Host 3 are 120 Mbps and 200 Mbps, respectively. Both flows are under their individual input bandwidth limitation. However, the aggregated 320 Mbps is larger than the 270 Mbps limitation for the destination port. Fig. 15 indicates that the traffic received by Host 1 is close to 270 Mbps, implying that the excess traffic is successfully dropped inside the virtual switch.

We further examine the Case 3 situation in the period between 80 s and 120 s. The input traffic generated from Host 2 and Host 3 is 280 Mbps and 350 Mbps, respectively. Each is larger than their input limitation. The total traffic received by Host 1 is only 270 Mbps. The experimental results confirm that the system can successfully enforce bandwidth usage requirements for all three cases.

We also conducted an experiment on a real Ethernet switch to test the same scenarios shown in Table III. The Ethernet

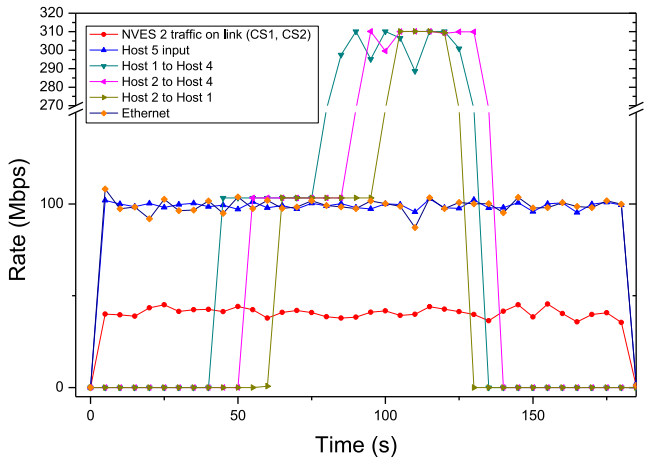

Fig. 16. Experimental results from testing for interference between NVES 1 and NVES 2.

TABLE IV
CONTROL PLANE RESPONSE TIME

Case	Average Time (ms)	STD	95% CI
1. Time to broadcast an unknown packet from a source when the source MAC address of the packet has been registered in the hash table of the controller	0.1925	0.0717	0.0199
2. Time to broadcast an unknown packet from a source when no record for the source MAC address exists in the hash table	1.5341	0.2993	0.0830
3. Connection setup time between Host 7 and Host 8 (including ARP)	12.0528	1.5314	0.4245
4. Port change	1.2092	0.0277	0.0077

switch used in the experiment is an HP 5120 switch. Because the link rate of the 5120 switch is 1 Gbps, we used another HP 5900 switch to throttle the link rate to 270 Mbps. The results are also plotted in Fig. 15. The total received traffic by Host 1 is similar between a real switch and an NVES. The only clear difference occurs during the transient period at approximately 40 s. This effect occurs because the responses of both meters used in those two switches are different. In the Ethernet switch case, the meter is provided by HP 5900. In the virtual switch, we used a leaky-bucket-based meter at the edge switches that we coded. Therefore, the meters have different transient responses for input traffic.

The final set of experiments tested whether a virtual switch is immune to interference from the network usage of another virtual switch. Here, a TCP connection exists for each port pair belonging to NVES 2. Because there are 4 ports in NVES 2, we have 12 TCP connections in total. We examine the traffic received by Host 5 of NVES 2 to see whether it is influenced by the traffic belonging to NVES 1. All the NVES 1 flows are UDP; thus, we can accurately control the amount of NVES 1 traffic. The experimental results are included in Fig. 16.

Based on the hose model solution, the communication pairs (Host 1, Host 4), (Host 2, Host 4), and (Host 2, Host 1) of NVES 1 go through Link (CS1, CS2). The communication pairs (Host 5, Host 8) and (Host 8, Host 7) of NVES2 also pass through this link. Because Link (CS1, CS2) carries the largest number of flows, we also observe the bandwidth usage of NVES2 on this link.

We dynamically change the input traffic of (Host 1, Host 4), (Host 2, Host 4), and (Host 2, Host 1) of NVES 1 over a wide range, from 0 Mbps to 300 Mbps, to discover whether NVES 2 is influenced. The traffic of NVES 2 on Link (CS1, CS2) is stable, remaining at approximately 40 Mbps. This value is not influenced by the changing patterns of NVES 1. Moreover, the total data received by Host 5 remains almost constant at 100 Mbps. The results confirm that NVES 2 is free of interference from the bandwidth usage of NVES 1.

In addition to using NVES, we also implemented our experiments using a real Ethernet switch. These experimental results are also included in Fig. 16. The virtual switch and real switch exhibit similar performance, indicating that our NVES system can successfully emulate a real Ethernet switch.

D. Performance of NVES Control Plane

We further examine the performance of the controller. When an NVES receives a packet with an unknown destination MAC address, the controller broadcasts the unknown packet to all its ports except the one from which the unknown packet arrived. We ran this experiment 50 times. Table IV lists the experimental results, including average time, standard deviation, and 95% confidence interval. The results show that the average time required by our testbed network to broadcast a packet with an unknown destination MAC address is 0.1925 ms or 1.5341 ms—depending on whether the source MAC address is known to the controller. When the source MAC address is unknown to the controller, the extra time arises from the address registration to the hash table in the controller.

In the second experiment, we examined the worst-case connection setup time between two hosts. Both hosts are unknown to each other, and there are no records in the hash table for these two hosts. We issue a ping message from Host 7 to Host 8 and measure the time elapsed until the reply is received by Host 7. Because both hosts are unknown to each other, Host 7 must send an address resolution protocol (ARP) packet to query the MAC address of Host 8 first. Through the MAC learning process, the controller learns the MAC addresses and IP addresses of both hosts. Accordingly, the controller sets up the routing paths for the connection. This process takes only 12.0528 ms on average (including the end host processing of the ARP packet) to complete the unknown packet flooding, MAC learning, and flow entry configuration to set up the connection.

Ethernet switches are known for their plug-and-play capability. When a host changes its connecting port, the switch should respond quickly. In the third experiment, we examined the time required for an NVES to respond to a port change. This experiment was implemented in NVES 2. The results show that the

TABLE V
CONNECTION SETUP TIME UNDER DIFFERENT REQUEST INTERVALS

Inter-arrival Time (ms)	No. Connection Loss	Average Time (ms)
50	0	4.2
10	0	4.4
5	0	4.5
1	0	4.8

time required to perform a port change is 1.2092 ms. In the normal case, a port change means a user has moved his/her device from one location to another location; thus, the time required for our NVES to respond to a port change is sufficiently fast and does not generate a noticeable delay for a user.

For the final set of experiments, we performed a pressure test on our NVES. We do not preclude users from using our NVES in a large Ethernet network. In such cases, an NVES must accommodate numerous flows.

In this experiment, we randomly selected a pair of ports (*i*, *j*) to set up a connection. Then, we injected a random packet from port *i*. The address fields in a test packet are new to the system; thus, no record exists in the hash table for the packet. The input of the packet triggers the MAC learning process, and the packet is broadcast to all the ports of the same NVES. When the packet is received by port *j*, we re-inject the packet into the network with its source address and destination address swapped. Consequently, the packet should be sent back to the transmitter at port *i* if the network can successfully set up a connection for it. We measure the elapsed time from the packet entering port *i* to the packet returning to port *i*.

We sequentially generated 4000 connection requests to examine the controller response time. By adjusting the inter-arrival time between two connective requests, different pressures can be applied to the controller. The experimental results are shown in Table V.

No connection losses occurred during this experiment. As the inter-arrival time decreases, the average connection setup time increases slightly. During our experiments, the worst-case configuration time was 27.7 ms. Because such heavy connection requests are unlikely to arrive in such a short period of time in real environments, we consider that this experiment demonstrates that the control plane is qualified for operations under normal usage.

VII. CONCLUSION AND DISCUSSION

In this paper, we presented an architectural design and system implementation for embedding NVESs in a substrate OpenFlow network. Each NVES has plug-and-play capability and can provide guaranteed bandwidth for its users. Traffic isolation is achieved; therefore, the traffic of an NVES is not influenced by other NVESs sharing the same network. We designed a new MP agent to achieve multi-path routing while eliminating the packet out-of-order problem. Just like a real Ethernet switch, our NVES system performs the MAC learning process to recognize the mappings between MAC addresses and port locations. The experimental results demonstrate that the throughput performance of the proposed NVES is similar to that of a real Ethernet switch. The results also reveal that an NVES is free from interference resulting from the network usage of other NVESs. It takes only a few milliseconds for an NVES to set up a new connection and react to a port change. Moreover, we also connected NVES systems to an existing Ethernet network. With the help of the STP module in the controller, an NVES can cooperate with real Ethernet switches without generating any traffic loops. A pressure test demonstrated that the proposed NVES system can handle a huge number of flows within a short period of time. These results confirm that the proposed NVES system is comparable to a real Ethernet switch.

The experimental results successfully demonstrated the proof-of-concept for the NVES system. Due to the shortage of computing power on the commodity PCs used here, the edge switches were unable to support a large number of NVESs. Beyond using more PCs to share the load, another solution to the problem is to implement edge switches on high performance computing servers. Another possibility is to implement edge switch functions on dedicated add-on hardware such as an FPGA. One future research direction is to implement an edge switch using the P4 language to facilitate targeting our system on new programmable high-throughput data planes [18]–[20], [28], [29].

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers for their helpful and constructive comments, which greatly contributed to improving the quality of this paper.

REFERENCES

[1] A. Blenk, A. Basta, M. Reisslein, and W. Kellerer, "Survey on network virtualization hypervisors for software defined networking," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 1, pp. 655–685, Jan. Apr. 2016.
[2] R. Sherwood *et al.*, "FlowVisor: A network virtualization layer," Open Flow Tech. Rep., vol. 2009, Oct. 2009. [Online]. Available: http://archive.openflow.org/downloads/technicalreports/openflow-tr-2009-1-flowvisor.pdf
[3] E. Salvadori, R. D. Corin, A. Broglio, and M. Gerola, "Generalizing virtual network topologies in OpenFlow-based networks," in *Proc. IEEE Global Telecommun. Conf.*, 2011, pp. 1–6.
[4] S. Lee, K. Li, K. Chan, Y. Chung, and G. Lai, "Design of bandwidth guaranteed OpenFlow virtual networks using robust optimization," in *Proc. IEEE Global Telecommun. Conf.*, 2014, pp. 1916–1922.
[5] S. Lee, K. Li, and M. Wu, "Design and Implementation of a GPON-based virtual openflow-enabled SDN switch," *J. Lightw. Technol.*, vol. 34, no. 10, pp. 2552–2561, May 2016.
[6] A. Autenrich *et al.*, "Evaluation of virtualization models for optical connectivity service providers," in *Proc. Int. Conf. Opt. Netw. Des. Model.*, 2014, pp. 264–268.
[7] N. Duffield, P. Goyal, and A. Greenberg, "A flexible model for resource management in virtual private networks," in *Proc. ACM SIGCOMM*, 1999, pp. 95–108.
[8] T. Erlebach and M. Ruegg, "Optimal bandwidth reservation in Hose-model VPNs with multi-path routing," in *Proc. IEEE Int. Conf. Comput. Commun.*, 2004, pp. 2275–2282.
[9] N. McKeown *et al.*, "OpenFlow: Enabling innovation in campus networks," in *Proc. ACM SIGCOMM*, 2008, pp. 69–74.
[10] OpenFlow Switch Specification Version 1.3.0. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.0.pdf

[11] S. Floyd, J. Mahdavi, M. Mathis, and M. Poldolsky, "An extension to the selective acknowledgement (SACK) options for TCP," IETF RFC, 2883, 2000.
[12] M. Zhang, B. Karp, S. Floyd, and L. Paterson, "RR-TCP: A reordering-robust TCP with DSACK," in *Proc.11th IEEE Int. Conf. Netw. Protocols*, 2003, pp. 95–106.
[13] M. Kheirkhah, I. Wakeman, and G. Parisis, "MMPTCP: A multipath transport protocol for data centers," in *Proc. IEEE Int. Conf. Comput. Commun*, 2016, pp. 1–9.
[14] S. Kandula, D. Katabi, S. Sinha, and A. Berger, "Dynamic load balancing without packet reordering," in *Proc ACM SIGCOMM*, 2007, pp. 53–62.
[15] M. Alizadeh *et al.*, "CONGA: Distributed congestion-aware load balancing for datacenters," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, pp. 503–514, Aug. 2014.
[16] N. Katta, M. Hira, C. Kim, A. Sivaraman, and J. Redford, "HULA: Scalable load balancing using programmable data planes," in *Proc. Symp. SDN Res.*, 2016, Paper 10.
[17] P. Bosshart *et al.*, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 87–95, Jul. 2014.
[18] P. Bosshart *et al.*, "Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN," *SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 99–110, Jul. 2013.
[19] "Intel FlexPipe." 2013. [Online]. Available: http://www.intel.com/content/dam/www/public/us/en/documents/product-briefs/ethernet-switch-fm6000-series-brief.pdf
[20] "Cavium and XPliant introduce a fully programmable switch silicon family scaling to 3.2 terabits per second." 2014. [Online]. Available: http://tinyurl.com/nzbqtr3
[21] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure, "TCP extensions for multipath operation with multiple addresses," IETF RFC 6824, 2013.
[22] A. Greenberg *et al.*, "VL2: A scalable and flexible data center network," in *Proc. ACM SIGCOMM*, 2009, pp. 51–62.
[23] C. Raiciu *et al.*, "How hard can it be? Designing and implementing a deployable multipath TCP," in *Proc. 9th USENIX Conf. Netw. Syst. Des. Implementation*, 2012, p. 29.
[24] NOX. 2012. [Online]. Available: https://github.com/noxrepo/nox
[25] OpenvSwitch. 2015. [Online]. Available: http://openvswitch.org/
[26] *Media Access Control (MAC) Bridges*, IEEE 802.1D, 1998.
[27] Iperf, The TCPUDP and SCTP network bandwidth measurement tool. 2014. [Online]. Available: https://iperf.fr/
[28] G. Bianchi, M. Bonola, A. Capone, and C. Cascone, "Openstate: Programming platform-independent stateful openflow applications inside the switch," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 2, pp. 45–51, 2014.
[29] A. Sivaraman *et al.*, "Packet transactions: high-level programming for line-rate switches," in *Proc. ACM SIGCOMM*, 2016, pp. 15–28.

Steven S. W. Lee received the Ph.D. degree in electrical engineering from National Chung Cheng University, Chiayi, Taiwan, in 1999. From 1999 to 2008, he was with Industrial Technology Research Institute, Taiwan, where he was the Leader of Intelligent Optical Networking Project and a Section Manager of the Optical Communications Department. He was also a Research Associate Professor of National Chiao Tung University, Taiwan, during 2004–2007. In 2008, he joined National Chung Cheng University, Taiwan, where he is currently a Professor in the Department of Communications Engineering. He has published more than 80 papers in international journals and conferences and is a coauthor of 20 international patents in the areas of broadband communications. His research interests include optical and broadband networking, network optimization, green network, and software-defined networking.

Kuang-Yi Li received the B.S. degree in information management from Saint John's University, Taipei, Taiwan, in 2010. He is currently working toward the Ph.D. degree in the Department of Communications Engineering, National Chung Cheng University, Chiayi, Taiwan. He has published more than 20 papers in journals and conferences. His research interests include network optimization, network survivability, green network, and software-defined networking. He received the Best Student Paper Award at IEEE CloudNet 2015.

Wei-Kai Liu received the B.S. degree in computer science and information engineering from Nanhua University, Chiayi, Taiwan, in 2014, and the M.S. degree in communications engineering from National Chung Cheng University, Chiayi, Taiwan, in 2016. He is currently an Engineer at DrayTek, Hsinchu, Taiwan. His main research focuses on the area of communication networks.

Chen-Hua Chen received the B.S. degree in computer science and information engineering from Nanhua University, Chiayi, Taiwan, in 2015, and the M.S. degree in communications engineering from National Chung Cheng University, Chiayi, Taiwan, in 2017. His research interests include software-defined networking and embedded systems.

How-Jen Fang received the B.S. degree in electrical engineering from National Chung Cheng University, Chiayi, Taiwan, in 2015, and the M.S. degree in communications engineering from National Chung Cheng University, Min-Hsiung, Taiwan, in 2017. He is going to be a Firmware Engineer at LITE-ON Technology. His main research focuses on the area of communication networks.

Ting-Shan Wong received the B.S. degree in communications engineering from National Chung Cheng University, Chiayi, Taiwan, in 2015. He is currently working toward the Ph.D. degree in electrical engineering at National Chung Cheng University, Chiayi, Taiwan. His main research interests focuses on the area of communication networks.