SDN-Enabled EVPN-VXLAN With P4 Accelerated User Plane

Yu-Cheng Yang, Ze-Yu Jin, Li-Hsing Yen, Chien-Chao Tseng

Dept. of Computer Science, College of Computer Science, National Yang Ming Chiao Tung University, Hsinchu, Taiwan.

***Abstract*— Ethernet Virtual Private Network (EVPN), together with Virtual Extensible Local Area Network (VXLAN), allows us to interconnect multiple Ethernet segments across the Internet to form a single, private Ethernet network. This paper adopts software-defined networking (SDN) technology to implement an EVPN-VXLAN architecture. Prior works either targeted data center networks (instead of geo-distributed sites) or used low-throughput virtualized switches. By contrast, the proposed approach targets geo-dispersed sites and uses P4 switches to accelerate data-plane performance. The approach integrates SDN host detection and EVPN host learning mechanisms for efficient host tracking and ARP suppression. It also supports Multi-Tenancy and Distributed Anycast Gateway (DAG). Experimental results demonstrate enhanced User Plane efficiency and reduced host communication latency.**

Keywords—SDN, EVPN, VXLAN, P4

I. INTRODUCTION

Technologies that provide Layer-Two (L2) connectivity over a Layer-Three (L3) network are collectively known as Layer-Two Virtual Private Network (L2VPN). Particularly, Virtual Extensible Local Area Network (VXLAN) [1], an extension to Virtual Local Area Network (VLAN), uses an overlay technology to provide L2 connectivity across geographically-dispersed sites. It creates tunnels between geo-dispersed VXLAN Tunnel End Point (VTEPs), encapsulates Ethernet frames in UDP datagrams, and identifies and isolates L2 user traffic using VNIs (VXLAN Network Identifiers). However, VXLAN alone is merely a data-plane solution, where the learning and forwarding of host Media Access Control (MAC) addresses among diverse sites mainly rely on the flood-and-learn approach. This leads to substantial unnecessary ARP flooding, causing a waste of network bandwidth, a problem particularly serious when sites are separated by a wide area network (WAN). In large-scale networks, this imposes capacity constraints of MAC forwarding tables on Provider Edge (PE) routers, thereby limiting the scale of the networks. As a remedy, Ethernet Virtual Private Network (EVPN) [2] provides a control-plane solution for geo-dispersed sites by advertising MAC and IP addresses, routes, and other attributes using Multi-Protocol Border Gateway Protocol (MP-BGP). The integrated architecture, known as EVPN-VXLAN, exercises Control and User Plane Separation (CUPS).

A straightforward way to implement EVPN-VXLAN is to let each VTEP run MP-BGP to exchange MAC/IP routing information. This is most commonly used in production networks such as leaf-spine data center fabrics. Vendors also offer fabric managers or controllers to abstract and automate EVPN-VXLAN deployments. By contrast, we consider implementing EVPN-VXLAN with Software-defined networking (SDN) [3]. SDN offers a new paradigm by decoupling the control and user planes, centralizing network intelligence, providing programmability, simplifying network management, and accelerating new service deployments. SDN controllers, such as the Open Network Operating System (ONOS) [4], manage the network from a central point, enabling automation, efficient resource allocation, and improved network visibility. These features make SDN an attractive choice to implement the EVPN-VXLAN architecture. There have been several EVPN-VXLAN implementations based on SDN [5] [6] [7], but they primarily target EVPN-VXLAN deployments in data centers rather than geo-distributed WAN sites. Huang [8] proposed an EVPN-VXLAN implementation based on SDN and Network Function Virtualization (NFV) for distributed sites. However, this approach cannot meet high-throughput demands as it adopts Open vSwitch (OVS) [9] as the virtual switch on nodes, which restricts the user-plane throughput performance and causes additional consumption of CPU computing resources.

This work is a follow-up of Ref. [8] with an aim to enhance network performance, reduce latency, and improve scalability by leveraging programmable Programming Protocol-Independent Packet Processor (P4) [10] switches. Besides the hardware acceleration part, we also extend the original work by supporting multi-tenancy, i.e., multiple enterprise users or organizations share the network infrastructure. In the multi-tenancy environment of EVPN, the Distributed Anycast Gateway (DAG) plays a pivotal role. It offers unified network egress services, enabling tenants' traffic to share the same egress IP address. When properly integrated with intelligent control mechanisms, DAG is able to dynamically route traffic to the most suitable gateways based on network conditions and gateway loads. This efficiently balances the load across gateways, prevents overloading due to tenants' bursty traffic, optimizes resource utilization, and maintains stable network service quality. In case of gateway failures, DAG's fail-over mechanism ensures seamless traffic redirection to functioning gateways, guaranteeing service continuity for all tenants. Moreover, by providing a single shared anycast IP for network egress, DAG simplifies tenant-side network configuration, reducing the complexity of multi-tenant network management and facilitating more efficient network connection management by administrators.

The key contributions of the proposed architecture are

- Utilizing P4 hardware switches for user-plane acceleration while providing interoperability with standard EVPN-VXLAN networks.
- Implementing SDN-assisted efficient host tracking and ARP suppression that reduces overall ARP traffic and communication latency.
- Supporting multi-tenancy and DAG [11] with efficient Internet connectivity.

The remainder of this paper is organized as follows. Sec. II reviews some related works. Sec. III presents the design of the proposed architecture. Sec. IV describes the method of implementation. Sec. V reports the experimental results, and the last section concludes this paper.

II. Related Work

In [5], the authors proposed an SDN-based framework for automating EVPN deployment in SDN-based data centers (DCs). While effective, it assumes VM environments within each DC, requires OpenStack for VM deployment, and uses SDN only for the overlay network, leaving the underlying network traditional. PE routers do not implement CUPS, limiting SDN benefits. Using OVS as the virtual switch for the overlay network may hinder user-plane performance and consume node computing resources.

Ref. [6] introduced an SDN-based architecture to improve EVPN performance, particularly focusing on the Designated Forwarder (DF) selection process. An SDN controller dynamically manages DF roles across VTEPs. This work maintains EVPN as a baseline, but introduces SDN logic to dynamically control and enhance EVPN functions, making it a hybrid controller-based EVPN system. Furthermore, this work is for interconnecting multiple close, clustered data centers.

Zhao et al. [7] proposed a three-layer SDN framework to optimize VXLAN-based overlay networks. It uses an SDN controller to replace traditional BGP-based MAC/IP learning. This non-standardized design limits its interoperability, so it is best suited for private clouds and does not support multi-tenancy.

The work [8] proposed an architecture to enable SDN to participate in traditional EVPN-VXLAN networks. The architecture leverages Network Function Virtualization (NFV) [12] technology by using a software router as the MP-BGP speaker to exchange EVPN routes with external networks for host learning and advertisement. Such a design maintains interoperability with standard EVPN PEs. On the other hand, the SDN controller installs flow rules on the underlying SDN switches based on the routes received by the speaker. Additionally, an OVS edge switch is deployed at the boundary between the SDN and external networks, acting as a VTEP and traffic classifier on the user plane. However, due to the implementation principles of OVS, this architecture requires splitting the edge switch into two OVS instances to handle both traffic classification and VTEP functions.

This paper adopts a similar approach by using an existing and mature software router as the MP-BGP speaker in SDN to exchange EVPN routes with external PEs. Specifically, our approach extends this concept by incorporating P4-based user-plane acceleration.

III. Design of SDN-Enabled EVPN-VXLAN With P4

A. Design Goal

We want to design an SDN-based EVPN-VXLAN architecture that can interoperate with standard EVPN sites (interoperable with other PEs). SDN controller should be responsible for control-plane traffic and minimize unnecessary ARP flooding. SDN switches (specifically, P4 switches) should accelerate user-plane traffic and, together with the SDN controller, implement CUPS. More specifically, we aim to achieve the following goals and features:

- *Interoperability*: The design should be interoperable with traditional EVPN-VXLAN networks.
- *ARP Suppression*: The design should suppress unnecessary ARP flooding (stemming from host learning).
- *Packet Forwarding and User Plane Acceleration*: The design should accelerate the processing of VXLAN packets by an efficient user-plane traffic classification and forwarding. This is crucial for high traffic loads.
- *Multi-Tenant Internet Connectivity*: The design should provide tenant hosts with efficient and low-latency network connections and maintain the same network connections when they roam between different networks.

B. System Components

Our architecture of EVPN-VXLAN consists of an SDN Controller, EVPN Control Plane, and User Plane, as shown in Fig. 1.

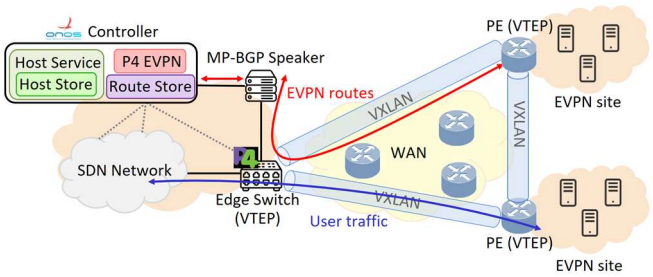

Fig. 1: System Architecture

SDN Controller: An SDN controller that runs essential APPs and provides services, including 1) P4 EVPN APP, which is responsible for EVPN's deployment and management in the SDN domain, MP-BGP Speaker's initial connection settings, synchronization of host routing info, route advertisement and reception, host learning and tracking in the EVPN network, and traffic engineering in the SDN network; 2) the ONOS Host Store, which is a database for host information in the SDN; and 3) the EVPN Route Store, which stores EVPN routes.

EVPN Control Plane: The MP-BGP Speaker is a software router deployed on a general server or in a container [13]. It is mainly responsible for exchanging EVPN routes with external PE devices and synchronizing routing information with the SDN controller.

User Plane: The P4 Edge Switch receives traffic from the SDN, MP-BGP Speaker, and external networks. It is responsible for classifying and handling different types of traffic and forwarding them to their destinations accordingly. It also acts as a VTEP that encapsulates and decapsulates VXLAN tunnel packets between sites.

IV. Implementation Details

A. SDN-enabled EVPN-VXLAN Functionalities

SDN-assisted Host Tracking: We leverage SDN to achieve efficient host tracking.

MP-BGP Speaker Setup: Initially, the administrator uploads a network configuration file for automated EVPN deployment in the SDN domain. The SDN controller, per the file's specifications, sets up a gRPC connection with the MP-BGP speaker and registers an EVPN route listener with it via the GoBGP-provided mechanism, ensuring that external EVPN routes received by the speaker are promptly notified and forwarded to the SDN controller for real-time route information synchronization.

Local Host Learning and Advertisement: Since ONOS acquires local SDN host info via packet-in ARP and the MP-BGP speaker can handle EVPN route advertisements, the SDN controller, upon learning a local host, sends the host info to the MP-BGP speaker for timely local host advertisement. As depicted in Fig. 2, the process is as follows: (0) A host sends an ARP packet. (1) The SDN switch receiving the ARP packet sends a packet-in message to the ONOS controller. (2) The SDN controller learns the host and stores its info in the ONOS Host Store. (3) The controller then sends the host info to the speaker, instructing it to generate and advertise the EVPN route for this host. (4) The speaker generates the corresponding EVPN type-2 route advertisement.

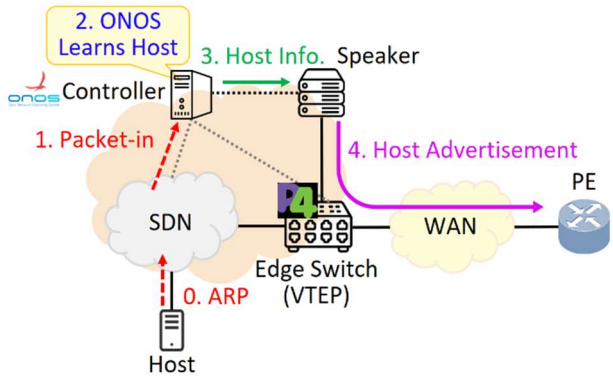

Fig. 2: Local Host Learning and Advertisement

Remote Host Learning: The SDN relies on the MP-BGP speaker to automatically push remote host EVPN routes received from external PEs to the SDN controller, achieving real-time learning of remote host information. As shown in Fig. 3, the overall process is as follows: (0) Assume a remote host sends an ARP packet to a remote PE. (1) The remote PE learns the host locally. (2) The remote PE advertises the EVPN type-2 route for the host. (3) The MP-BGP speaker receives the route, and the pre-registered EVPN Route Listener immediately pushes this route information to the SDN controller. (4) The SDN controller analyzes and converts the host route information, storing it in the EVPN Route Store, thus completing the learning of the remote host.

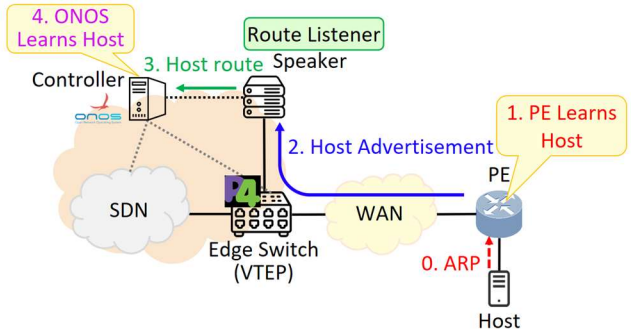

Fig. 3: Remote Host Learning

ARP Suppression: We implement Proxy ARP on the SDN controller, responding to all known host ARP via the packet-in/packet-out approach. This curtails unnecessary ARP flooding. In EVPN, upon learning local host info, PE routers advertise it to other PEs through EVPN type-2 routes, which include MAC/IP Advertisement and MAC-only Advertisement. However, when only a MAC-only Advertisement is received, Proxy ARP resorts to flooding for remote host ARP requests because of the absence of IP-MAC mapping in the EVPN Route Store, leading to superfluous flooding traffic within the site and across the provider's backbone network, thus wasting bandwidth. We utilize SDN's built-in packet-in mechanism to learn remote host information to tackle this. As illustrated in Fig. 4, by installing the appropriate flow rule on the P4 Edge Switch, ARP packets from remote hosts transmitted through the VXLAN tunnel are decapsulated and then packet-in to the SDN controller. This enables the SDN controller to automatically learn and store host information in the ONOS Host Store. Significantly, the ARP packets packet-in to ONOS are the original decapsulated host ARP packets, rendering the VXLAN tunnel transparent to ONOS. To ONOS, the remote host seems like a regular local host directly connected to the P4 Edge Switch's WAN port.

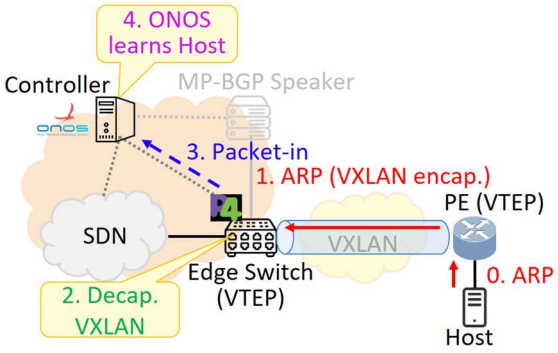

Fig. 4: Remote Host Learning on SDN User Plane

The following outlines how our architecture acquires the IP-MAC mapping of remote hosts in the MAC-only Advertisement scenario to ensure Proxy ARP functionality. Remote host sites can be of two types:

Traditional EVPN Site: Making the VXLAN tunnel transparent to the SDN controller, ARP packets from remote hosts are handled by packet-in messages. The controller learns remote host info and retrieves the IP-MAC mapping from the ONOS Host Store. This enables Proxy ARP to respond to ARP requests, avoiding ARP flooding directly.

SDN Site: Theoretically, no ARP packets from remote hosts should exist in the SDN architecture with implemented Proxy ARP. By ensuring the SDN implementation consistently advertises MAC/IP Advertisement routes with full IP-MAC mappings, Proxy ARP can always retrieve the remote hosts' IP-MAC mappings from the EVPN Route Store, effectively preventing needless ARP flooding.

DAG: In EVPN, the PE handling inter-subnet routing of user traffic acts as the default gateway. Here, we implement a DAG to support tenant host mobility. This allows tenant hosts to access the internet efficiently via the nearest PE without making changes. As depicted in Fig. 5, the DAG concept makes all gateways in the EVPN network function as one, with all PEs (gateways) in the tenant EVPN network sharing the same gateway IP. The DAG's gateway MAC can be configured in one of the following ways:

Unified Gateway MAC: A unified gateway MAC lets network devices treat DAG as a single entity. In SDN-based DAGs, the controller assigns a unified MAC address, allowing external networks to communicate without knowing the internal topology, boosting scalability. Poor internal MAC management risks conflicts.

MAC Aliasing: Gateway MAC aliases enable load balancing and failover. Mapping multiple aliases to one MAC address lets DAG distribute traffic to different nodes. Failover occurs by remapping aliases to healthy nodes [14]. In the SDN domain, we install relevant flow rules on the P4 Edge Switch for each received Default Gateway route to match the gateway MAC and offer routing services.

The initial network configuration file can specify both DAG gateway MAC settings. Implementing DAG enables each PE to function as a default gateway. It offers optimal routing, supports host mobility, and provides efficient internet access for any connected tenant host.

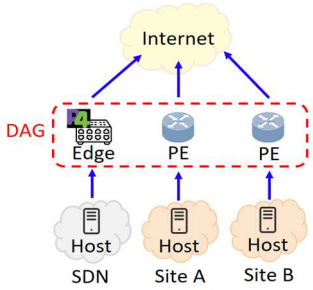

Fig. 5: Distributed Anycast Gateway

B. P4 Switch Pipeline

Our architecture uses hardware offloading by introducing a P4 Edge Switch for User Plane acceleration. This section introduces the Pipeline design implemented with P4. P4 is a domain-specific programming language that specifies how User Plane devices process packets. independence (programmers need not know underlying hardware specifics), and reconfigurability (programmers can redefine packet parsing and processing rules).

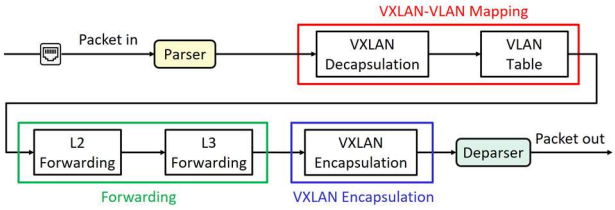

Fig. 6: P4 Switch Pipeline Overview

The Pipeline in this paper is written using P4 based on the Tofino Native Architecture (TNA) [15]. The overall Pipeline design is shown in Fig. 6. The Pipeline is divided into the following three stages based on major functions:

1) VXLAN-VLAN Stage . This stage primarily handles the initial VXLAN-VLAN conversion of incoming packets, ensuring that packets forwarded to the Forwarding Stage carry VLAN tags. It includes two tables. One is VXLAN Decapsulation Table, which *determines whether to decapsulate based on VNI and VTEP IP. The VTEP IP must match the P4 Edge Switch's VTEP IP for decapsulation. If decapsulation is needed, the VLAN tag is added to the packet based on the VNI*. The other is VLAN Table, which adds the corresponding VLAN tag to packets that do not already have one, based on the In_port. For packets from the MP-BGP Speaker or remote PE that do not belong to any tenant, the default behavior is to add a Native VLAN tag of 1 for subsequent matching.

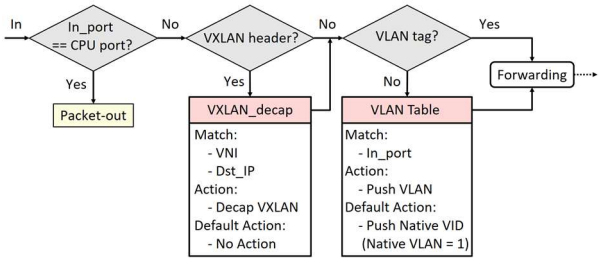

Fig. 7: VXLAN-VLAN Mapping Stage

2) Forwarding Stage. This stage handles packet classification and forwarding logic, and consists of L2 and L3 forwarding tables. L2 Forwarding Table classifies and forwards packets based on VLAN ID, Ethernet Type, and destination MAC. If the destination is a Gateway MAC, the Routing Tag is set to 1, indicating L3 routing in the next step. For packets with a Routing Tag of 1, L3 Forwarding Table forwards the packets based on the destination IP and performs the corresponding hop-by-hop MAC replacement.

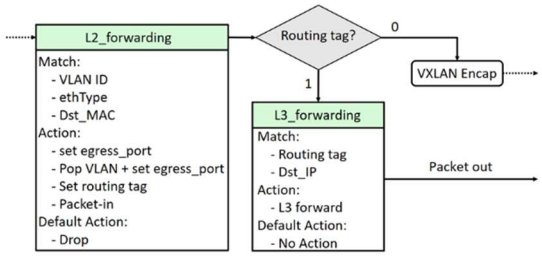

Fig. 8: Forwarding Stage

3) VXLAN Encapsulation Stage. This stage adds the VXLAN header to packets, which includes only one table: VXLAN Encapsulation Table. It calculates the inner packet length in the pipeline and encapsulates tenant user packets with the corresponding VXLAN header based on VLAN ID and destination MAC address before sending them out.

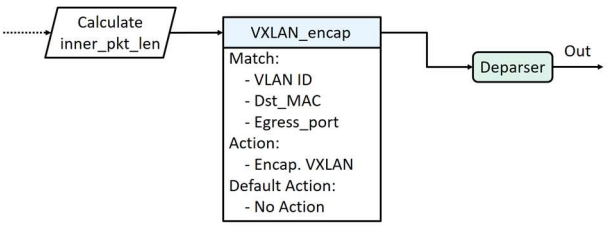

Fig. 9: VXLAN Encapsulation Stage

V. Performance Evaluation

We performed functional verification and performance evaluation of the proposed architecture. TABLE I details the hardware specifications. Fig. 10 depicts the environmental setup, where our implementation is on the left-hand side. PE2/PE3 on the right-hand side with FRR [16] are remote traditional EVPN sites. A Next-hop router connects host H_i in the middle to emulate the Internet.

The entire environment utilizes two servers and one P4 switch. One ONOS server was responsible for running the SDN controller. Except for the SDN controller, P4 Edge Switch, and H_i, all other components were simulated and ran in Docker [17] containers on another Container Server. H_i was simulated by creating a namespace on the Container Server and binding it to a network card on the server. All ports connected to the P4 Edge Switch used 10GbE links.

TABLE I: Hardware Specifications

Machine	CPU	Memory	OS	Note
ONOS Server	Intel i7-10700	32GB	Ubuntu 18.04.5 LTS (Kernel 5.4.0-150-generic)	SDN controller
Container Server	Intel Xeon CPU E5-2630 v4 @ 2.20GHz + 128G	126GB	Ubuntu 20.04.5 LTS (Kernel 5.4.0-144-generic)	NIC: Intel Ethernet Controller X540-AT2 10 Gbps
P4 Edge Switch	Edgecore Wedge 100BF-32QS (Intel Tofino BFN-T10-032Q Switch, Xeon D-1548/48G), 10 Gbps QSFP28 links Barefoot SDE Version: 9.3.0			

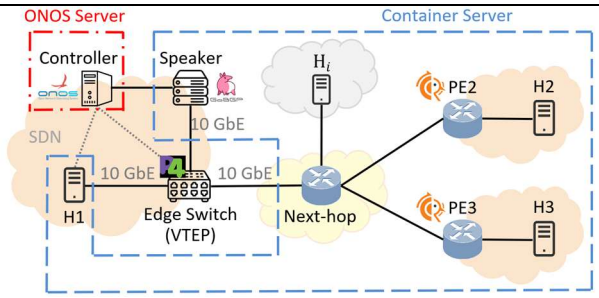

Fig. 10: Environment Setup

A. *VXLAN Tunneling Performance*

We verified the performance of the P4 Edge Switch as the VTEP in the proposed architecture and compared it with OVS.

1) End-to-End Latency. We tested the latency of the P4 Edge Switch to evaluate its efficiency concerning VXLAN encapsulation and decapsulation. Internet Control Message Protocol (ICMP) packets were sent from H1 to H2 using the PING command, encapsulated with a VXLAN header by the P4 Edge Switch, transmitted to PE2 for decapsulation, and returned. We recorded the Round Trip Time (RTT) to identify significant latency. As shown in TABLE II, the P4 Edge Switch maintained low latency and mean deviation, indicating stable and fast performance. The OVS, simulated on the same server, showed low delay due to the absence of physical link latency.

TABLE I: Latency Test

	Latency	
	OVS	P4
MIN.	0.058 ms	0.093 ms
AVG.	0.101 ms	0.138 ms
MAX.	0.834 ms	0.186 ms
MDEV.	0.168 ms	0.024 ms

2) Throughput. We also measured the throughput of the P4 Edge Switch and compared it against OVS as a VTEP. H1 generated UDP traffic with varying payload sizes using iPerf2 [18], which the P4 Edge Switch encapsulated with VXLAN headers and sent to PE2. Fig. 11 shows the results. With smaller payloads, server limitations prevented full bandwidth utilization, but encapsulation increased the Tx throughput. As a software switch, OVS was limited by server hardware and could not handle high traffic volumes or achieve line-rate speeds. In contrast, the P4 Edge Switch consistently reached line-rate speeds with larger payloads, effectively utilizing the bandwidth.

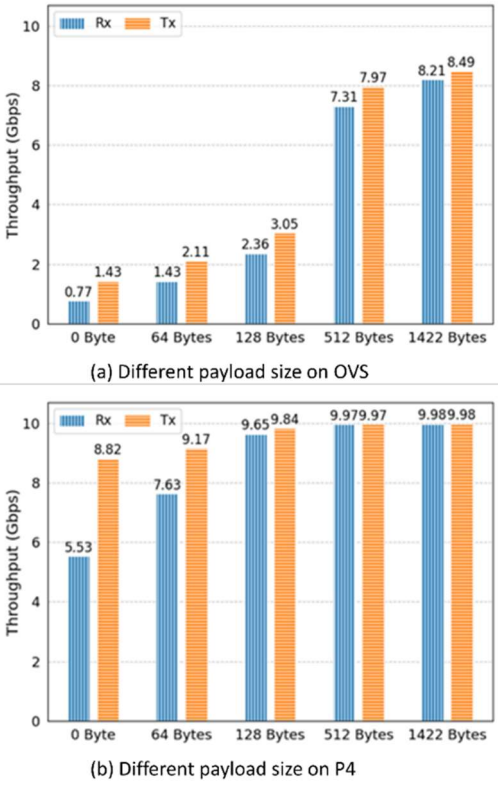

Fig. 11: VXLAN Throughput

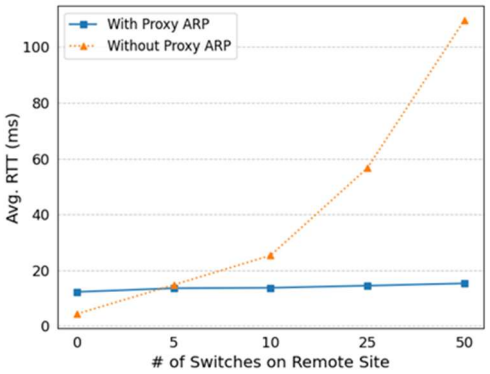

Fig. 12: ARP Latency Measurement

B. *ARP Latency Measurement*

We compared the latency of ARP packets on the User Plane with and without Proxy ARP. The experiment used Mininet [19] to simulate varying numbers of switches between PE2 and H2, representing different network topologies. Each link between switches and routers was set with a 1 ms delay and 0.2 ms jitter. H1 sent ARP requests to H2 using ARPING, and the RTT of the packets was observed, assuming SDN had already learned H2's information. Fig. 12 shows that without Proxy ARP, ARP requests traveled via VXLAN to the remote PE2 and then to H2, resulting in delays influenced by network

complexity and latency, along with increased ARP traffic. With Proxy ARP, ARP requests were handled directly by the SDN controller via packet-in/packet-out, resulting in more stable delays influenced only by the SDN controller's capacity and the switch-to-controller latency. Proxy ARP also reduced unnecessary ARP traffic, saving network bandwidth.

C. Default Gateway Performance

We analyzed the performance of different default gateway scenarios with host mobility using Mininet to simulate varying numbers of routers between the Next-hop router and each site, respectively, as shown in Fig. 13. Each link had a 1 ms delay and 0.2 ms jitter. We compared H3's RTT when sending ICMP packets to H_i in three scenarios:

- *Original Gateway* (*OG*)*:* H3 accessed the Internet through PE3 from its original location.
- *Remote Centralized Gateway* (*RCG*)*:* With the Default Gateway at PE2, H3 moved to the SDN domain. The P4 Edge Switch encapsulated H3's traffic with a VXLAN header to PE2 and then routed the traffic to H_i. The return traffic followed the same path back.
- *DAG:* H3 moved to the SDN domain and directly accessed the Internet through the nearest P4 Edge Switch.

Fig. 14 shows the average RTT from H3 to H_i (P4 edge switch, PE2, PE3) for these scenarios. The *x*-axis represents the number of router hops, and the *y*-axis shows the average RTT. Despite equal hops to H_i, the RCG scenario shows increased RTT as the hop count between the P4 Edge Switch and PE2 increased, highlighting sub-optimal routing. In contrast, the DAG scenario maintained low and stable RTT by routing traffic through the nearest P4 Edge Switch. This demonstrates its efficiency in avoiding sub-optimal routing and providing better Internet connectivity for mobile hosts.

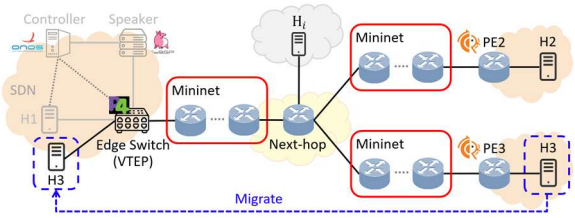

Fig. 13: Environment Setup for Default Gateway Performance Test

VI. Conclusions

This paper presents an SDN-enabled EVPN-VXLAN architecture with a P4-accelerated User Plane. It can interoperate with traditional EVPN, using SDN for features like host tracking and faster convergence. Proxy ARP cuts ARP traffic and resolves IP-MAC mapping issues. Experiments show it has low, stable latency. P4 hardware switches on the User Plane speed up processing, handle VXLAN well, and outperform OVS in high-traffic scenarios. The DAG offers low-latency internet access for tenant hosts.

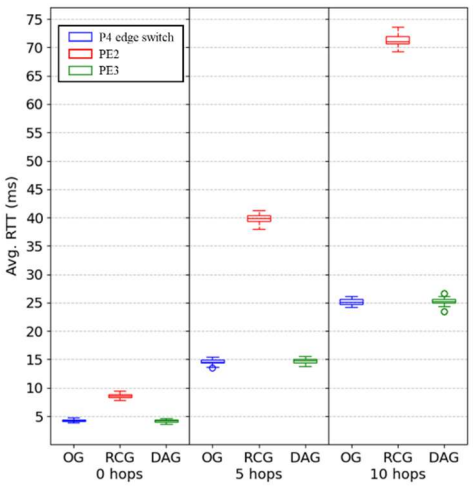

Fig. 14: Environment Setup for Default Gateway Performance Test

VII. References

[1] "Virtual Extensible LAN," [Online]. Available: https://en.wikipedia.org/wiki/Virtual_Extensible_LAN.

[2] "RFC 7432 - BGP MPLS-Based Ethernet VPN," Feb 2015. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc7432.

[3] "Software-Defined Networking," [Online]. Available: https://en.wikipedia.org/wiki/Software-defined_networking..

[4] P. Berde et al., "ONOS: Towards an Open, Distributed SDN OS," *Proceedings of the third workshop on Hot topics in software defined networking, ACM,* 2014.

[5] Kyoomars Alizadeh Noghani, Cristian Hernandez Benet, Andreas Kassler, Antonio Marotta, Patrick Jestin, Vivek V. Srivastava, "Automating Ethernet VPN Deploymnet in SDN-based Data Centers," in *2017 Fourth International Conference on Software Defined Systems (SDS)*, 2017.

[6] K. A. Noghani and A. Kassier, "SDN enhanced Ethernet VPN for data center interconnect," in *IEEE 6th International Conference on Cloud Networking*, Prague, Czech Republic, 2017.

[7] Z. Zhao, F. Hong and R. Li, "SDN Based VxLAN Optimization in Cloud Computing Networks," *IEEE Access,* pp. 23312–23319, 12 October 2017.

[8] Yihsuan Huang, *SDNFV-enabled EVPN Interconnection*, Master Thesis, National Chiao Tung University, 2021.

[9] "Open vSwitch," [Online]. Available: https://www.openvswitch.org.

[10] "P4 Language and Related Specifications," [Online]. Available: https://p4.org/specs/.

[11] "Integrated Routing and Bridging in Ethernet VPN (EVPN)," [Online]. Available: https://datatracker.ietf.org/doc/rfc9135/.

[12] "Network function virtualization," [Online]. Available: https://en.wikipedia.org/wiki/Network_function_virtualization.

[13] "GoBGP," [Online]. Available: https://github.com/osrg/gobgp.

[14] "BGP Extended Communitites Attribute," [Online]. Available: https://datatracker.ietf.org/doc/html/rfc4360.

[15] "Barefoot Tofno," [Online]. Available: https://www.intel.com/content/www/us/en/products/details/network-io/intelligent-fabric-processors/tofino.html.

[16] "FRRouting," [Online]. Available: https://frrouting.org/.

[17] "Docker," [Online]. Available: https://www.docker.com/.

[18] "iPerf," [Online]. Available: https://iperf.fr/iperf-doc.php.

[19] "Mininet," [Online]. Available: https://github.com/mininet/mininet.