2022 IEEE 12th Annual Computing and Communication Workshop and Conference (CCWC) | 978-1-6654-8303-2/22/$31.00 ©2022 IEEE | DOI: 10.1109/CCWC54503.2022.9720797

Dynamic On-Demand Virtual Extensible LAN Tunnels via Software-Defined Wide Area Networks

Gieorgi Zakurdaev
Dept. of Systems and Computer Engineering
Carleton University
Ottawa, Canada
gieorgi.zakurdaev@carleton.ca

Mohammed Ismail
Dept. of Systems and Computer Engineering
Carleton University
Ottawa, Canada
hamood.ismail@carleton.ca

Chung-Horng Lung
Dept. of Systems and Computer Engineering
Carleton University
Ottawa, Canada
chlung@sce.carleton.ca

***Abstract*—The world of information and communications technologies continues to evolve at an exponential rate, not only inaugurating numerous breakthroughs in research, but also changing the perception of interconnectivity and exchange of information. The increasing popularity of software-defined networks (SDN) and the related technologies have shattered the realm of corporate infrastructures. The traditional approach for employees to access the corporate headquarter data centers has experienced challenges due to the significant traffic volume and delay, as more services have been moved to the cloud, e.g., Software-as-a-Service (SaaS). Tunnel-splitting can mitigate the problem, but it is mostly static. The paper proposed a dynamic on-demand tunnels approach based on Extensible LAN Tunnels (VXLAN) and SDN. The primary objectives are to reduce the load on corporate network and delay for users to access SaaS. We conducted experiments for feasibility study using Mininet and the results showed that the delay could be significantly reduced and the approach allows for a single point of policy management, which it still preserved the benefit of split tunnels.**

***Index Terms*—SDN, NFV, VXLAN, SD-WAN, Cloud, Tunnel-Splitting, Mininet**

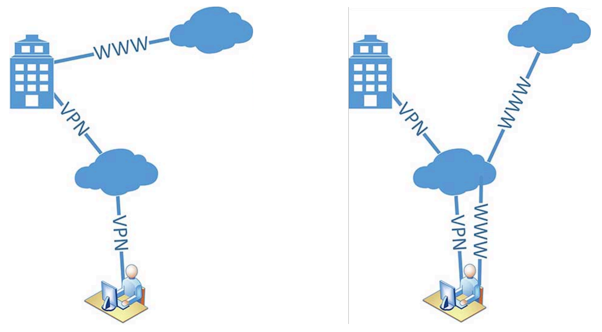
I. Introduction

Over the course of the precedent decades, the world of information and communications technologies continued to evolve at an exponential rate, not only inaugurating numerous breakthroughs in research, but also changing the entire perception of interconnectivity and exchange of information. The increasing popularity of software-defined networks (SDN) has shattered the realm of corporate infrastructures, providing historic opportunities for the development of next-generation network technologies and introducing revolutionary standards for services and applications.

In retrospect, as part of many programs, piloted in response to the innumerable COVID-19 impediments, a multitude of companies began to implement Internet-based services at an escalating rate, resulting in an impact on the overall performance of the transfer of information and data, particularly affecting link capacity and network reliability [1], [2]. Recent statistical data has shown that in comparison to January 2020, online transactions have increased by 39.7%, and conversion rates — by 40.3%, significantly straining the performance of wide and land area networks. In order to mitigate these underlying issues, service providers have introduced remediating decisions, striving to increase channel capacities. However, the scope of the remaining constraints continues to illustrate limitations of Internet availability and performance, emphasizing the inevitable contributions of online migration campaigns to the spiked demands for bandwidth, which strain the reliability of modern corporate network architectures [2], [3].

For instance, a report [4] shows that an elevated number of employees working from home presents a narrow bottleneck for enterprise traffic flows, often creating unmanageable loads on corporate servers and data centers. Specifically, 80% of the traffic from employees working remotely is for the Internet access to various cloud services (e.g., Software-as-a-Service (SaaS)) and only 20% of the traffic is for the internal services hosted at the corporate data centers. Consequently, the traditional approach for employees to access the corporate headquarter data centers, even for 3rd party cloud SaaS, has experienced challenges due to the significant traffic volume and delay via the headquarters. Hence, the motivation of the paper is to identify a functional approach to assist businesses with the discovery of alternative solutions that would mitigate the corporate network congestion problem.

One particular solution that has increased in popularity throughout the precedent years is split-tunneling [5], as depicted in Fig. 1. It plays an integral role in the improvement of the bottleneck challenge, prevailing within enterprise architectures.



(a) without a split-tunnel (b) with a split-tunnel

Fig. 1: Representation of the Split-Tunnelling Technique [5]

Although split-tunneling presented a viable approach, as the amount of Internet users grew, a number of limitations became apparent. For instance, the original idea behind split-tunneling, as shown in Fig. 1b, aimed to navigate web-oriented traffic of remote users directly towards the destination servers, without firstly relaying it through the corporate data center, as depicted in Fig. 1a, where granular inspection took place, such as monitoring, inspection, and intrusion detection. Historically, end-user traffic, dedicated to professional tasks that accessed sensitive corporate information, constituted the majority of remote employee workflows. As a result, the decision for traffic redirection was simply made based on a static criterion, which dictated the default tunnel route as the most secure path, and transferred all enterprise traffic to company headquarters for detailed analysis. On the other hand, because remote Internet traffic mostly stemmed from harmless web-browsing and personal use, the tunnel switching criterion simply mandated all web-oriented packets to form a direct connection with the destination servers, completely eliminating the need for redundant inspection at the enterprise data-centers, and thus, greatly reducing the associated load.

In light of the era of SaaS and Infrastructure-as-a-Service (IaaS), which shifted the fundamental business needs of many industries towards cloud-oriented solutions, fully reliant on web-based traffic, the initial design of the split-tunneling architecture became less efficient. Particularly, an apparent need to modify the tunnel-switching decision presented a point of concern, and an ultimate requirement for a dynamic structure was deemed essential. Consequently, the main objective of the paper is to demonstrate potential uses-cases for a dynamic model of communication using Extensible LAN Tunnels (VXLAN) and SDN, where strategic routing of corporate data through an appropriate path could be achieved based on the on-demand needs, rather than a static set of firewall rules. Intending to approach the goal, this paper identified areas for potential improvement and presented experimental evaluation using Mininet. The results showed that proposed approach could significantly reduce the latency.

The rest of the paper is organized as follows: §II describes some related work, §III presents the design, and §IV shows some results, which are then followed by §V with the conclusions and discussion of future work.

II. Related Work

Several technical areas are related to the paper. This section highlights a few closely related topics.

Network Function Virtualization (NFV) and *Software Defined Networking (SDN)* have become popular. With NFV, standard resources, designed for computational, storage, and network-oriented functions can be virtualized and placed on commercial off-the-shelf (COTS) hardware. The introduction of virtualized components infers that portions of the available resources can be allocated to virtual machines (VMs). In that manner, multiple VMs can run on a single server and scale to consume the remaining unused resources, upon demand [6].

SDN is defined to represent a network architecture that allows a network to be intelligently and centrally controlled. This enables the operators to manage the entire architecture consistently and holistically, regardless of the underlying technologies. SDN enables the programming of network behavior in a centrally controlled manner, through software applications, which use open APIs. By expanding traditionally closed network platforms and implementing a common SDN control layer, operators can manage the complete network and the associated connected devices consistently, regardless of the complexity of the underlying infrastructure. SDN allows a system to acquire scalable characteristics, while being manageable and easily kept under control. Additionally, the term "SD-WAN" is often used in correlation to the concept, referring to the SDN functionalities within a Wide Area Network (WAN), rather than a Local Area Network (LAN) [7]. SDN has four critical areas that can bring critical differences within an organization: network programmability, logically centralized intelligence and control, abstraction of the network, and openness [7], [8].

Yang et al. [8] presented challenges that existing WANs face. In addition, they highlighted opportunities and emerging techniques, such as NFV and machine learning, to support the evolution of SD-WAN. The authors in [9] proposed an approach to minimize the total route update cost on all flows by formulating the problem as an integer linear programming optimization problem.

Secure Access Service Edge (SASE) is defined as a cloud-delivered service that combines network and security functions with SD-WAN capabilities to support the dynamic, secure access needs of today's hybrid organizations, such as Cloud Access Security Broker (CASB), Firewall-as-a-Service (FWaaS), and Zero-Trust Network Access (ZTNA). SASE also acts as a gateway for Virtual Private Network (VPN) solutions, while simultaneously remaining flexible, cost-efficient, quick, and simple to implement [10].

Virtual Extensible LAN (VXLAN) is an encapsulation protocol that provides data center connectivity by introducing overlay tunneling technique that stretches Layer 2 Ethernet connections over underlying Layer 3 networks using UDP. VXLAN solves the scalability limitations of traditional VLANs, as well as providing other benefits absent from VLAN technologies. Each Layer 2 subnet can be uniquely identified by a VXLAN network identifier (VNI), thoroughly segmenting the traffic. Each VNI in VXLAN has 24 bits, which enables to support 16 million isolated networks, where traditional VLAN only uses 12 bits to support a maximum of 4096 isolated networks. Finally, VXLAN is capable of performing encapsulation and decapsulation of packets at various end-points, intending to support devices unable to act as virtual interfaces on their own [11].

III. System Design and Implementation

This section presents the design that combines the concepts described in §II. Specifically, as the main objective involves the implementation of networking functionalities within a software-defined virtual network environment, hence, NFV and SDN/SD-WAN functionalities are leveraged.

As previously mentioned, it is evident that many businesses are frequently concerned with confidentiality and integrity aspects of their information; as a result, they tend to implement inspection sites on centrally located servers, which are assigned verification and record-keeping tasks, completed prior to the further transmission of traffic. These include not only web-oriented traffic flows, but also cloud-based services, such as SaaS, or video calls, which must be ultimately transmitted to according destinations. Although the motivation for such designs shows great advantages in terms of security, there exist many scenarios, where deep corporate-level inspection is simply not essential due to the lower level of sensitivity for a particular category of user traffic.

Alternatively, network traffic of a certain nature may already be subjected to transmission that uses protocols, that by design, are furnished with sufficient security measures. In this case, routing traffic through centralized inspection sites can significantly increase network load, congestion, and delay, without providing a functional advantage as a result [5].

Introducing a dynamic on-demand tunneling architecture could serve as a mitigating factor for corporate network load-balancing. The dynamic on-demand tunneling can allow for two unique devices to establish a direct route between their end points, while simultaneously preserving an alternative fail-safe path, mapped through the headquarters, which could also serve as the principal route for all end-user traffic until the direct tunnel is created.

As such, the proposed schematic aims to divide the solution into two main use-cases. The first instance involves a default, fail-safe path that routes the end-user traffic through the corporate headquarters, while the second instance involves a direct path, created and established on-demand, based on specific underlying criteria. This is achieved by leveraging VXLAN capabilities to form both of the static tunnel routes, and by using Secure Shell (SSH) for the administration of dynamic tunnel switching, completed between the two paths, and automated using an SDN controller function [12].

A. Default Tunnel

In order to highlight the first use case scenario, focused on the default tunneling technique, a specific design was developed, as depicted in Fig. 2. The architecture envisioned three unique environments, each represented by a VM, and intended to emphasize three separate networks. VM1 was dedicated to the client's "subnet", where VM1's operating system was used to create a Mininet virtual topology, containing one virtual client entity connected to one virtual switch. Similarly, VM2 was used to imitate the server's "subnet", creating an additional Mininet topology, also encapsulating one virtual host and one virtual switch. Lastly, VM3 was used to generate a final Mininet topology, which only contained one virtual switch and no hosts, representing the mediator corporate headquarters.

The goal was to implement all three of the VMs on separate subnets to provide a clear differentiation between the Mininet environments. As a result, we simulated three unrelated networks, potentially located on opposite sides of the world. As such, the default path aimed to configure two segments of VXLAN tunnel, where one interconnected the switch hosted on VM1 with the switch hosted on VM3, and sequentially, the second interconnected the switch hosted on VM3 with the switch hosted on VM2. Finally, a number of interfaces were initialized on each VM, following the model represented on the diagram, where the Ethernet interface on each switch was associated with the virtual host, while the virtual interface was associated with the outbound VXLAN connection.

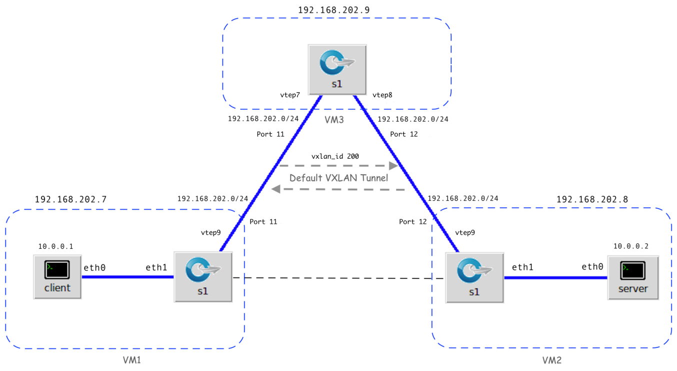


Fig. 2: Architecture of the Default Tunnel

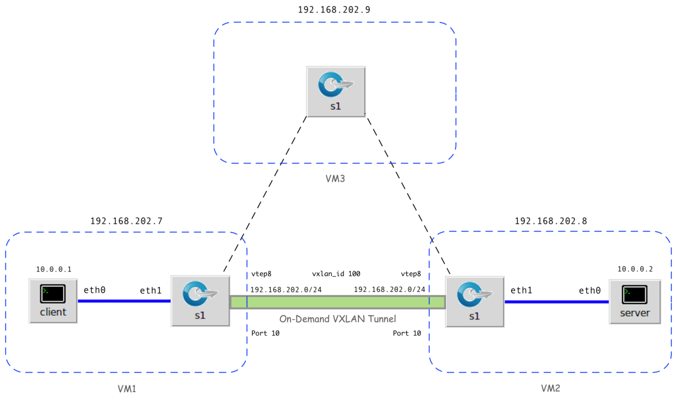


Fig. 3: Architecture of the On-Demand Tunnel Scenario

B. On-Demand Tunnel

The second use case scenario focused on the on-demand tunneling technique, where a direct connection was established between the client and the server hosts, and consequently, between VM1 and VM2. The revised architecture is depicted in Fig. 3. In this case, an additional interface was added for the switches hosted on VM1 and VM2, dedicated strictly for the on-demand VXLAN tunnel link. As a result, the overall architecture was pre-configured statically, for both available paths. Precisely, sourcing the flow table entries at each switch, which were unique for each VXLAN tunnel identification number, the network packets could potentially flow over the default tunnel route, or alternatively, over the on-demand tunnel route.

In order to achieve the dynamic behavior of the tunnel switching, a piece of code was developed, intending to act as a controller for the virtual switches on VM1 and VM2. In particular, the code was designed to initiate SSH connections to both VMs upon demand, delete existing flow table entries with the old tunnel ID, and flush new entries with the updated tunnel ID. As a result, it achieves the intended purpose by decreasing the load on the median host and completing the data transfer at a faster pace. Aiming to preserve simplicity within the proof-of-concept model, a specific time interval was configured to present the triggering factor for the program's initiation, in order to imitate dynamic congestion control or load reduction measures. However, the program was designed in a scalable manner, leaving plenty of room for more sophisticated decisive measures, such as specific security parameters.

C. Implementation

VMware Fusion was chosen to serve as the software hypervisor and Mininet was used for topology setup. Open vSwitch was then configured with the following command for the client VM1 with IP address `192.168.202.7`:

```
mininet>   sh ovs-vsctl add-port s1 vtep9 -- set interface vtep9 type=vxlan
        option:remote_ip=192.168.202.9 option:key=flow ofport_request=11
```

The first line created a new interface called `vtep9` on VM1 and specified the channel type to be `vxlan`, established towards a remote IP of VM3 on OpenFlow `port 11`. Similarly, `vtep8` for a `vxlan` channel on VM2's IP address on `port 10` can be created. The final step required the implementation of the preliminary OpenFlow table, which was designed to manage VM1's Open vSwitch traffic by directing it through the default VXLAN tunnel path using according flow entries. The following demonstrates the complete flow table for VM1:

```
table=0,priority=200,in_port=1,actions=set_field:200->tun_id,resubmit(,1)
table=0,priority=100,actions=resubmit(,1)

table=1,tun_id=100,dl_dst=00:00:00:00:00:01,actions=output:1
table=1,tun_id=100,dl_dst=00:00:00:00:00:02,actions=output:10
table=1,tun_id=100,arp,nw_dst=10.0.0.1,actions=output:1
table=1,tun_id=100,arp,nw_dst=10.0.0.2,actions=output:10
table=1,tun_id=200,dl_dst=00:00:00:00:00:01,actions=output:1
table=1,tun_id=200,dl_dst=00:00:00:00:00:02,actions=output:11
table=1,tun_id=200,arp,nw_dst=10.0.0.1,actions=output:1
table=1,tun_id=200,arp,nw_dst=10.0.0.2,actions=output:11
table=1,priority=100,actions=drop
```

The entries in the upper-hand table (`table 0`) ensured that, by default, any outbound traffic flowing through the switch on `port 1` is assigned a value of `200` for the VXLAN tunnel ID, `tun_id`. The entries in the lower-hand table (`table 1`) ensured that based on the `tun_id` and the destination MAC address, the traffic is directed either inbound (to `port 1`), or outbound (to `port 10` or `port 11`). Precisely, the traffic was intended to flow through `port 11` (default tunnel path) if the assigned `tun_id` was `200`, and through `port 10` (on-demand tunnel path) if the `tun_id` was `100`. However, since the entries of `table 0` dictated that all outbound traffic was assigned a `tun_id` equal to `200`, the packets could only flow through the default path at that point.

The setup for the server on VM2 is similar. The resulting flow table for VM2 is shown as follows:

```
table=0,priority=200,in_port=1,actions=set_field:200->tun_id,resubmit(,1)
table=0,priority=100,actions=resubmit(,1)

table=1,tun_id=100,dl_dst=00:00:00:00:00:02,actions=output:1
table=1,tun_id=100,dl_dst=00:00:00:00:00:01,actions=output:10
table=1,tun_id=100,arp,nw_dst=10.0.0.2,actions=output:1
table=1,tun_id=100,arp,nw_dst=10.0.0.1,actions=output:10
table=1,tun_id=200,dl_dst=00:00:00:00:00:02,actions=output:1
table=1,tun_id=200,dl_dst=00:00:00:00:00:01,actions=output:12
table=1,tun_id=200,arp,nw_dst=10.0.0.2,actions=output:1
table=1,tun_id=200,arp,nw_dst=10.0.0.1,actions=output:12
table=1,priority=100,actions=drop
```

The first line of `table 0` enforced the assignment of `tun_id` to `200` for all outbound traffic. In this case, OpenFlow `port 12` represented the default tunnel path, while `port 10` continued to represent the on-demand tunnel, which remained unused at this point.

The ultimate step of the setup focused on the configuration of the mediator environment, initialized on VM3. The Open vSwitch on VM3 was configured as follows:

```
mininet>   sh ovs-vsctl add-port s1 vtep7 -- set interface vtep7 type=vxlan
        option:remote_ip=192.168.202.7 option:key=flow ofport_request=11
```

```
mininet>   sh ovs-vsctl add-port s1 vtep8 -- set interface vtep8 type=vxlan
        option:remote_ip=192.168.202.8 option:key=flow ofport_request=12
```

The interface `vtep7` was channeled to VM1 on OpenFlow `port 11`, while the interface `vtep8` mirrored the configuration on VM2, with `port 12` assigned accordingly. The flow table is depicted as follows:

```
table=0,actions=resubmit(,1)

table=1,tun_id=200,dl_dst=00:00:00:00:00:02,actions=output:12
table=1,tun_id=200,arp,nw_dst=10.0.0.2,actions=output:12
table=1,tun_id=200,dl_dst=00:00:00:00:00:01,actions=output:11
table=1,tun_id=200,arp,nw_dst=10.0.0.1,actions=output:11
table=1,priority=100,actions=drop
```

In this case, all actions targeting `tun_id=100` were removed, since no network traffic with that VXLAN identification value was ever destined to flow through VM3. Likewise, the flow entry that initially assigned the tunnel ID in `table 0` was also removed, and the entries in `table 1` were adjusted to match the according ports for the forwarding task.

D. Tunnel Switching

Up to this point, the implementation process involved static configurations for the client, server and mediator, designed to remain unchanged throughout the remaining stages of the experiment. However, in order to attain the principal goal of the project, precisely, dynamic tunnel switching, a program was designed with SSH for automatic connection setup. In fact, the expected functionality of the solution aimed to update the respective flow table entries on VM1 and VM2, simulating an OpenFlow controller.

In essence, the program performed the deletion of old Open vSwitch flow table entries on VM1 and added the corresponding updated flow entries — all while executing the commands natively, in the background. In fact, the updated flow entries for both VM1 and VM2 were practically identical to the old entries, except the first line:

```
table=0,priority=200,in_port=1,actions=set_field:100->tun_id,resubmit(,1)
```

In comparison to the old flow table entries for VM1, the only variation of the updated entries is the very first line, where the `tun_id` identification was changed from `200` to `100`, the VXLAN network identifier (VNI), as shown in Fig. 3.

IV. Results and Analysis

In order to ensure that the dynamic tunneling design functioned according to the expectations, the results were collected and analyzed using Wireshark.

A. *Experiment Launch on VM1*

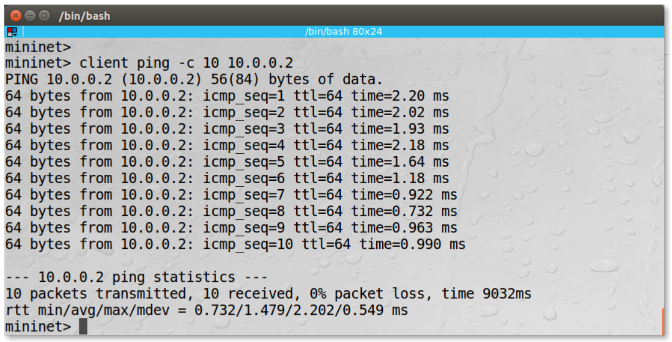
A `ping` from the client to server was executed in the Mininet terminal on VM1 for the initial test:

```
mininet>   client ping -c 10 10.0.0.2
```

In essence, a set of 10 `ping` requests were initiated from the client host, on VM1, to the server host, on VM2. After several seconds, in a different terminal of VM1, the following command was executed to run a Python program:

```
$   sudo python dynamic-tunnel.py
```

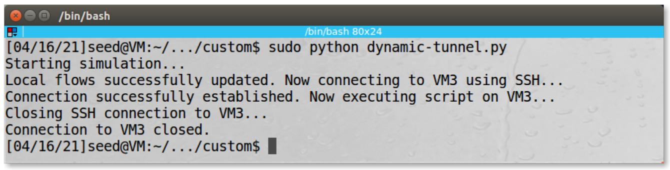
With this command, the Python program was executed about halfway through the sequence of `ping` requests, aiming to switch the tunnel path from *default* to *on-demand* after several seconds. Fig. 4 and Fig. 5 depict the resulting outcome.



```
mininet>
mininet> client ping -c 10 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=2.20 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=2.02 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=1.93 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=2.18 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=1.64 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=1.18 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=0.922 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=0.732 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=0.963 ms
64 bytes from 10.0.0.2: icmp_seq=10 ttl=64 time=0.990 ms

--- 10.0.0.2 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9032ms
rtt min/avg/max/mdev = 0.732/1.479/2.202/0.549 ms
mininet>
```

Fig. 4: Initiation of `ping` Requests from Client on VM1 to Server on VM2



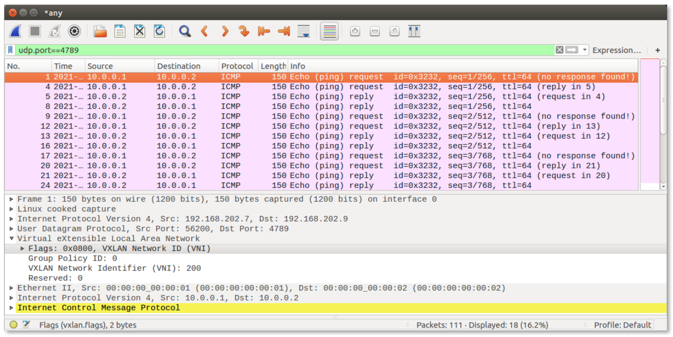
```
[04/16/21]seed@VM:~/.../custom$ sudo python dynamic-tunnel.py
Starting simulation...
Local flows successfully updated. Now connecting to VM3 using SSH...
Connection successfully established. Now executing script on VM3...
Closing SSH connection to VM3...
Connection to VM3 closed.
[04/16/21]seed@VM:~/.../custom$
```

Fig. 5: Execution of the Dynamic Python Program on VM1

It can be observed that the experiment completed successfully. In Fig. 4, it can be seen that the round-trip time (RTT) of the ping requests significantly decreased about halfway through the execution, as expected. Precisely, it can be noted that the minimum RTT, which showed to be `0.732` ms, was less than half of the maximum RTT, which showed to be `2.202` ms, demonstrating significant improvements in delays after the direct tunnel was enabled.

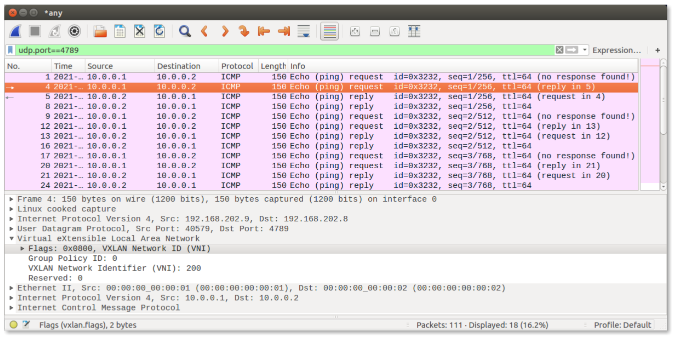
B. *Experimental Results on VM3*

During the experiment, an instance of Wireshark was initialized on VM3. Fig. 6 depicts the results produced from capturing the packets on interface `any`, filtered by `udp.port ==4789`, to isolate VXLAN traffic.



```
udp.port==4789
No.  Time    Source    Destination Protocol Length Info
   1 2021-… 10.0.0.1  10.0.0.2    ICMP     150 Echo (ping) request  id=0x3232, seq=1/256, ttl=64 (no response found!)
   4 2021-… 10.0.0.1  10.0.0.2    ICMP     150 Echo (ping) request  id=0x3232, seq=1/256, ttl=64 (reply in 5)
   5 2021-… 10.0.0.2  10.0.0.1    ICMP     150 Echo (ping) reply    id=0x3232, seq=1/256, ttl=64 (request in 4)
   8 2021-… 10.0.0.2  10.0.0.1    ICMP     150 Echo (ping) reply    id=0x3232, seq=1/256, ttl=64
   9 2021-… 10.0.0.1  10.0.0.2    ICMP     150 Echo (ping) request  id=0x3232, seq=2/512, ttl=64 (no response found!)
  12 2021-… 10.0.0.1  10.0.0.2    ICMP     150 Echo (ping) request  id=0x3232, seq=2/512, ttl=64 (reply in 13)
  13 2021-… 10.0.0.2  10.0.0.1    ICMP     150 Echo (ping) reply    id=0x3232, seq=2/512, ttl=64 (request in 12)
  16 2021-… 10.0.0.2  10.0.0.1    ICMP     150 Echo (ping) reply    id=0x3232, seq=2/512, ttl=64
  17 2021-… 10.0.0.1  10.0.0.2    ICMP     150 Echo (ping) request  id=0x3232, seq=3/768, ttl=64 (no response found!)
  20 2021-… 10.0.0.1  10.0.0.2    ICMP     150 Echo (ping) request  id=0x3232, seq=3/768, ttl=64 (reply in 21)
  21 2021-… 10.0.0.2  10.0.0.1    ICMP     150 Echo (ping) reply    id=0x3232, seq=3/768, ttl=64 (request in 20)
  24 2021-… 10.0.0.2  10.0.0.1    ICMP     150 Echo (ping) reply    id=0x3232, seq=3/768, ttl=64
▸ Frame 1: 150 bytes on wire (1200 bits), 150 bytes captured (1200 bits) on interface 0
▸ Linux cooked capture
▸ Internet Protocol Version 4, Src: 192.168.202.7, Dst: 192.168.202.9
▸ User Datagram Protocol, Src Port: 56200, Dst Port: 4789
▾ Virtual eXtensible Local Area Network
  ▸ Flags: 0x0800, VXLAN Network ID (VNI)
    Group Policy ID: 0
    VXLAN Network Identifier (VNI): 200
    Reserved: 0
▸ Ethernet II, Src: 00:00:00_00:00:01 (00:00:00:00:00:01), Dst: 00:00:00_00:00:02 (00:00:00:00:00:02)
▸ Internet Protocol Version 4, Src: 10.0.0.1, Dst: 10.0.0.2
▸ Internet Control Message Protocol
Flags (vxlan.flags), 2 bytes          Packets: 111 · Displayed: 18 (16.2%)          Profile: Default
```

Fig. 6: Wireshark Analysis on VM3 (Part 1)



```
udp.port==4789
No.  Time    Source    Destination Protocol Length Info
   1 2021-… 10.0.0.1  10.0.0.2    ICMP     150 Echo (ping) request  id=0x3232, seq=1/256, ttl=64 (no response found!)
→  4 2021-… 10.0.0.1  10.0.0.2    ICMP     150 Echo (ping) request  id=0x3232, seq=1/256, ttl=64 (reply in 5)
←  5 2021-… 10.0.0.2  10.0.0.1    ICMP     150 Echo (ping) reply    id=0x3232, seq=1/256, ttl=64 (request in 4)
   8 2021-… 10.0.0.2  10.0.0.1    ICMP     150 Echo (ping) reply    id=0x3232, seq=1/256, ttl=64
   9 2021-… 10.0.0.1  10.0.0.2    ICMP     150 Echo (ping) request  id=0x3232, seq=2/512, ttl=64 (no response found!)
  12 2021-… 10.0.0.1  10.0.0.2    ICMP     150 Echo (ping) request  id=0x3232, seq=2/512, ttl=64 (reply in 13)
  13 2021-… 10.0.0.2  10.0.0.1    ICMP     150 Echo (ping) reply    id=0x3232, seq=2/512, ttl=64 (request in 12)
  16 2021-… 10.0.0.2  10.0.0.1    ICMP     150 Echo (ping) reply    id=0x3232, seq=2/512, ttl=64
  17 2021-… 10.0.0.1  10.0.0.2    ICMP     150 Echo (ping) request  id=0x3232, seq=3/768, ttl=64 (no response found!)
  20 2021-… 10.0.0.1  10.0.0.2    ICMP     150 Echo (ping) request  id=0x3232, seq=3/768, ttl=64 (reply in 21)
  21 2021-… 10.0.0.2  10.0.0.1    ICMP     150 Echo (ping) reply    id=0x3232, seq=3/768, ttl=64 (request in 20)
  24 2021-… 10.0.0.2  10.0.0.1    ICMP     150 Echo (ping) reply    id=0x3232, seq=3/768, ttl=64
▸ Frame 4: 150 bytes on wire (1200 bits), 150 bytes captured (1200 bits) on interface 0
▸ Linux cooked capture
▸ Internet Protocol Version 4, Src: 192.168.202.9, Dst: 192.168.202.8
▸ User Datagram Protocol, Src Port: 40579, Dst Port: 4789
▾ Virtual eXtensible Local Area Network
  ▸ Flags: 0x0800, VXLAN Network ID (VNI)
    Group Policy ID: 0
    VXLAN Network Identifier (VNI): 200
    Reserved: 0
▸ Ethernet II, Src: 00:00:00_00:00:01 (00:00:00:00:00:01), Dst: 00:00:00_00:00:02 (00:00:00:00:00:02)
▸ Internet Protocol Version 4, Src: 10.0.0.1, Dst: 10.0.0.2
▸ Internet Control Message Protocol
Flags (vxlan.flags), 2 bytes          Packets: 111 · Displayed: 18 (16.2%)          Profile: Default
```

Fig. 7: Wireshark Analysis on VM3 (Part 2)

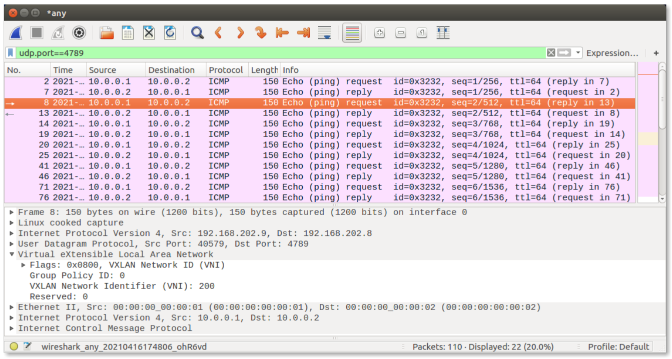
It can be seen that the source and destination IP addresses match the ones of the virtual client (`10.0.0.1`) and the virtual server (`10.0.0.2`) hosts. Additionally, on the IP layer (the grey area in the lower half), it can be observed that the source IP is `192.168.202.7`, while the destination IP is `192.168.202.9` for VM3, matching the respective VMs. Finally, the VNI also matches the `tun_id` of the default tunnel path (`200`), as expected. It can also be noted that this particular ICMP request did not receive a direct response, as it was forwarded to VM2, instead of issuing a reply from VM3.

Next, Fig. 7 depicts an analysis of the sequential request from VM3 to VM2. In this case, it is evident that the switch on VM3 (*src.:* `192.168.202.9`) has successfully forwarded the traffic to VM2 (*dest.:* `192.168.202.8`), and this time, the response for the ICMP request was successfully received.

C. Experimental Results on VM2

Another instance of Wireshark was also initialized on VM2, intending to record the details of the tunnel switching process. Fig. 8 depicts the results produced from capturing the packets on interface `any`, filtered by `udp.port==4789`, to isolate VXLAN traffic.

In this first scenario, it can be seen that the source and destination IP addresses match the ones of the virtual client (`10.0.0.1`) and the virtual server (`10.0.0.2`) hosts. However, on the IP layer, it can be observed that the source IP is now `192.168.202.9` (for VM3), while the destination IP is `192.168.202.8` (for VM2), matching the VMs involved on the second segment of the default tunnel path. Finally, the VNI also matches the `tun_id` of the default tunnel path (`200`), as expected. In this case, all of the ICMP requests received according responses, since this time, the packet capture was completed on VM2, rather than VM3.

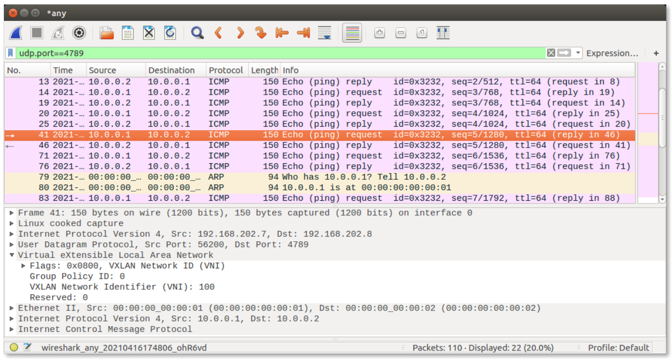


```
No.   Time     Source     Destination Protocol Length Info
    2 2021-… 10.0.0.1   10.0.0.2    ICMP      150 Echo (ping) request  id=0x3232, seq=1/256, ttl=64 (reply in 7)
    7 2021-… 10.0.0.2   10.0.0.1    ICMP      150 Echo (ping) reply    id=0x3232, seq=1/256, ttl=64 (request in 2)
→   8 2021-… 10.0.0.1   10.0.0.2    ICMP      150 Echo (ping) request  id=0x3232, seq=2/512, ttl=64 (reply in 13)
←  13 2021-… 10.0.0.2   10.0.0.1    ICMP      150 Echo (ping) reply    id=0x3232, seq=2/512, ttl=64 (request in 8)
   14 2021-… 10.0.0.1   10.0.0.2    ICMP      150 Echo (ping) request  id=0x3232, seq=3/768, ttl=64 (reply in 19)
   19 2021-… 10.0.0.2   10.0.0.1    ICMP      150 Echo (ping) reply    id=0x3232, seq=3/768, ttl=64 (request in 14)
   20 2021-… 10.0.0.1   10.0.0.2    ICMP      150 Echo (ping) request  id=0x3232, seq=4/1024, ttl=64 (reply in 25)
   25 2021-… 10.0.0.2   10.0.0.1    ICMP      150 Echo (ping) reply    id=0x3232, seq=4/1024, ttl=64 (request in 20)
   41 2021-… 10.0.0.1   10.0.0.2    ICMP      150 Echo (ping) request  id=0x3232, seq=5/1280, ttl=64 (reply in 46)
   46 2021-… 10.0.0.2   10.0.0.1    ICMP      150 Echo (ping) reply    id=0x3232, seq=5/1280, ttl=64 (request in 41)
   71 2021-… 10.0.0.1   10.0.0.2    ICMP      150 Echo (ping) request  id=0x3232, seq=6/1536, ttl=64 (reply in 76)
   76 2021-… 10.0.0.2   10.0.0.1    ICMP      150 Echo (ping) reply    id=0x3232, seq=6/1536, ttl=64 (request in 71)

▶ Frame 8: 150 bytes on wire (1200 bits), 150 bytes captured (1200 bits) on interface 0
▶ Linux cooked capture
▶ Internet Protocol Version 4, Src: 192.168.202.9, Dst: 192.168.202.8
▶ User Datagram Protocol, Src Port: 40579, Dst Port: 4789
▼ Virtual eXtensible Local Area Network
  ▶ Flags: 0x0800, VXLAN Network ID (VNI)
    Group Policy ID: 0
    VXLAN Network Identifier (VNI): 200
    Reserved: 0
▶ Ethernet II, Src: 00:00:00_00:00:01 (00:00:00:00:00:01), Dst: 00:00:00_00:00:02 (00:00:00:00:00:02)
▶ Internet Protocol Version 4, Src: 10.0.0.1, Dst: 10.0.0.2
▶ Internet Control Message Protocol

wireshark_any_20210416174806_ohR6vd     Packets: 110 · Displayed: 22 (20.0%)     Profile: Default
```

Fig. 8: Wireshark Analysis on VM2 (Part 1)



```
No.   Time     Source     Destination Protocol Length Info
   13 2021-… 10.0.0.2   10.0.0.1    ICMP      150 Echo (ping) reply    id=0x3232, seq=2/512, ttl=64 (request in 8)
   14 2021-… 10.0.0.1   10.0.0.2    ICMP      150 Echo (ping) request  id=0x3232, seq=3/768, ttl=64 (reply in 19)
   19 2021-… 10.0.0.2   10.0.0.1    ICMP      150 Echo (ping) reply    id=0x3232, seq=3/768, ttl=64 (request in 14)
   20 2021-… 10.0.0.1   10.0.0.2    ICMP      150 Echo (ping) request  id=0x3232, seq=4/1024, ttl=64 (reply in 25)
   25 2021-… 10.0.0.2   10.0.0.1    ICMP      150 Echo (ping) reply    id=0x3232, seq=4/1024, ttl=64 (request in 20)
→  41 2021-… 10.0.0.1   10.0.0.2    ICMP      150 Echo (ping) request  id=0x3232, seq=5/1280, ttl=64 (reply in 46)
←  46 2021-… 10.0.0.2   10.0.0.1    ICMP      150 Echo (ping) reply    id=0x3232, seq=5/1280, ttl=64 (request in 41)
   71 2021-… 10.0.0.1   10.0.0.2    ICMP      150 Echo (ping) request  id=0x3232, seq=6/1536, ttl=64 (reply in 76)
   76 2021-… 10.0.0.2   10.0.0.1    ICMP      150 Echo (ping) reply    id=0x3232, seq=6/1536, ttl=64 (request in 71)
   79 2021-… 00:00:00_… 00:00:00_…  ARP        94 Who has 10.0.0.1? Tell 10.0.0.2
   80 2021-… 00:00:00_… 00:00:00_…  ARP        94 10.0.0.1 is at 00:00:00:00:00:01
   83 2021-… 10.0.0.1   10.0.0.2    ICMP      150 Echo (ping) request  id=0x3232, seq=7/1792, ttl=64 (reply in 88)

▶ Frame 41: 150 bytes on wire (1200 bits), 150 bytes captured (1200 bits) on interface 0
▶ Linux cooked capture
▶ Internet Protocol Version 4, Src: 192.168.202.7, Dst: 192.168.202.8
▶ User Datagram Protocol, Src Port: 56200, Dst Port: 4789
▼ Virtual eXtensible Local Area Network
  ▶ Flags: 0x0800, VXLAN Network ID (VNI)
    Group Policy ID: 0
    VXLAN Network Identifier (VNI): 100
    Reserved: 0
▶ Ethernet II, Src: 00:00:00_00:00:01 (00:00:00:00:00:01), Dst: 00:00:00_00:00:02 (00:00:00:00:00:02)
▶ Internet Protocol Version 4, Src: 10.0.0.1, Dst: 10.0.0.2
▶ Internet Control Message Protocol

wireshark_any_20210416174806_ohR6vd     Packets: 110 · Displayed: 22 (20.0%)     Profile: Default
```

Fig. 9: Wireshark Analysis on VM2 (Part 2)

Next, Fig. 9 presents the second scenario, highlighting the post-tunnel-switch ICMP request, which occurred several seconds after the one shown in Fig. 8. Some noticeable differences can be observed. Primarily, although the main source and destination IP addresses remained `10.0.0.1` and `10.0.0.2`, on the IP layer (lower half), the source has changed to `192.168.202.7`, respective with VM1's IP. In addition, the VNI also changed from `200` to `100`, highlighting the dynamic tunnel switch.

As a result, we concluded that the system functioned according to the expectations, and provided benefits, in agreement with the proposed dynamic on-demand tunnelling architecture.

V. Conclusions and Future Directions

The increasing demand for technologies designed to improve the availability and the end-user experience, especially within corporate networks, brings light to many innovative ideas, particularly with the expanding SDN infrastructure. The approach proposed in this paper aims to improve the existing network designs by reducing the load on corporate network circuitry with dynamic on-demand tunnels. The result showed that the latency could be significantly reduced. The improvement could be even larger in a real large SD-WAN environment. The approach leveraged the concept of SDN and preserved the advantages of the split tunnels.

We are currently extending the work using multiple open-source data centers in the world to emulate large WANs. In addition, we are evaluating the idea of Kubernetes for higher reliability and efficiency. The approach is also useful for advanced learning and prototyping in SD-WAN related techniques and research.

References

[1] "Covid-19 impact on internet performance — case study of afghanistan, nepal, and sri lanka," Internet Society, Tech. Rep., March 2021. [Online]. Available: https://www.internetsociety.org/wp-content/uploads/2020/12/Asia-Covid-report-EN-March-2021.pdf

[2] R. De', N. Pandey, and A. Pal, "Impact of digital surge during covid-19 pandemic: A viewpoint on research and practice," *International Journal of Information Management*, vol. 55, p. 102171, 2020, impact of COVID-19 Pandemic on Information Management Research and Practice: Editorial Perspectives. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0268401220309622

[3] J. Clement. Year-on-year change in online traffic worldwide as of may 2021. [Online]. Available: https://www.statista.com/statistics/1105495/coronavirus-traffic-impact/

[4] L. Miller, "Secure access service edge (sase) for dummies," Cisco, Tech. Rep., August 2020. [Online]. Available: https://www.secureitstore.com/datasheets/ebooks/new-secure-access-service-edge-sase-for-dummies-ebook.pdf

[5] O. K. Aslim, "What is split tunneling? should you allow it?" *Cub Cyber*, October 2020. [Online]. Available: https://www.cubcyber.com/what-is-split-tunneling-cmmc

[6] "What is nfv (network functions virtualization)? definition," *SDxCentral Studios*, August 2013. [Online]. Available: https://www.sdxcentral.com/networking/nfv/definitions/whats-network-functions-virtualization-nfv/

[7] "What is sdn?" Ciena, Tech. Rep., December 2019. [Online]. Available: https://www.ciena.com/insights/what-is/What-Is-SDN.html

[8] Z. Yang, Y. Cui, B. Li, Y. Liu, and Y. Xu, "Software-defined wide area network (sd-wan): Architecture, advances and opportunities," in *2019 28th International Conference on Computer Communication and Networks (ICCCN)*, July 2019, pp. 1–9.

[9] D. Zad Tootaghaj, F. Ahmed, P. Sharma, and M. Yannakakis, "Homa: An efficient topology and route management approach in sd-wan overlays," in *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, July 2020, pp. 2351–2360.

[10] "What is sase?" *Cyberpedia*, April 2020. [Online]. Available: https://www.paloaltonetworks.com/cyberpedia/what-is-sase

[11] "Ip fabric evpn-vxlan reference architecture," Juniper Networks, Inc., Tech. Rep., January 2019. [Online]. Available: https://www.juniper.net/content/dam/www/assets/reference-architectures/us/en/ip-fabric-evpn-vxlan-reference-architecture.pdf

[12] T. Ylönen, "Ssh: Secure login connections over the internet," in *Proceedings of the 6th Conference on USENIX Security Symposium, Focusing on Applications of Cryptography - Volume 6*, ser. SSYM'96. USA: USENIX Association, 1996, p. 4.