EveryWAN White paper v1

EveryWAN White Paper
Authored by: Carmine Scarpitta, Pier Luigi Ventre, Stefano Salsano
Version: 1.0
Date: 2021:05:06

Table Of Contents

1. Introduction **2**

2. EveryWAN architecture **6**

3. EveryWAN services **8**

4. EveryEdge Node Architecture **10**

5. EveryEdgeOS and EveryWAN Orchestrator **14**

6. EveryWan Validation and Performance Evaluation platform (EVPE platform) **16**

7. Future works **17**

8. References **18**

1. Introduction

Software Defined Wide Area Network (SD-WAN) was originally proposed as an alternative solution to redesign the architecture of the WAN ([1]–[4]). Like its technology precursor Software Defined Networking [5], SD-WAN was aiming at simplifying the management and operation of the networks (with a particular focus on WAN scenarios) by decoupling the networking hardware from its control programs and using software and open APIs to abstract the infrastructure and manage the connectivity and the services. While this "softwarization" step and disaggregation of the WAN devices was possible for the operators owning and managing the entire infrastructure - nowadays, with the rise of the Cloud era, all the incumbent vendors tend to agree towards a new broader concept. An SD-WAN is an architecture that leverages SDN principles to securely build interconnections between users and the applications hosted in the clouds or in remote branches, by leveraging any combination of transport services, including low-cost and commercially available broadband access (MPLS, LTE/5G and broadband internet services) [6] [7].

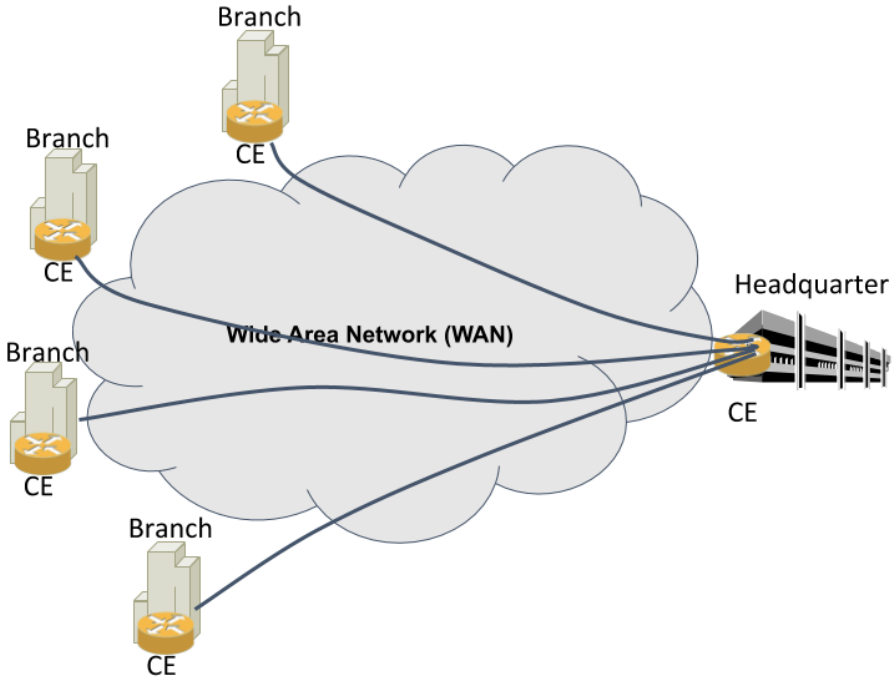

Figure 1 - WAN reference scenario

Figure 1 shows an example of a traditional WAN scenario where a customer/tenant, owning several branches, needs connectivity to interconnect the remote sites to the main office. Within the latter, there are the servers that host applications accessed by users dislocated in the remote branches. Each office can be interconnected using different service providers which typically create dedicated MPLS circuits to fulfill this requirement. MPLS is used to offer VPNs and layer 2 connectivity services and help ensure a stable connectivity. Borrowing the terminology used in the traditional WANs, the providers networks include a set of Provider Edge routers which are interconnected through a multi-terabit Core Network where MPLS is used to improve the forwarding of the IP traffic and to avoid the explosion of the routing tables. Instead, the Customer Edge (CE in Figure 1) routers represent the IP

based customer devices connected to the provider. Typically, the CEs are managed by the tenants and represent the line of demarcation with the WAN provider network. Another interesting trend is the Managed Service Provider (MSP) use case where the entire infrastructure can be managed by a third party relieving the tenants from a heavy configuration/management effort.

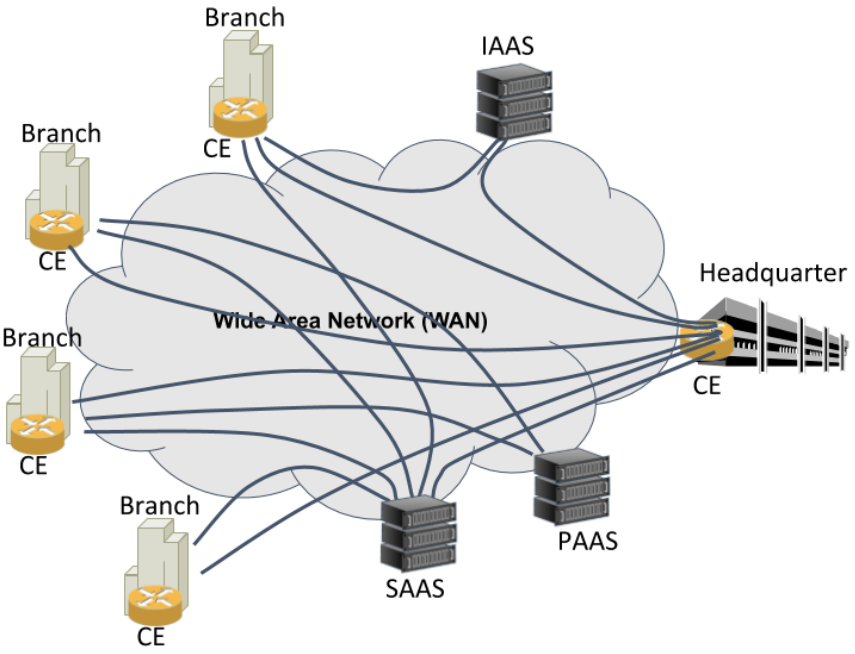

Figure 2 - SD- WAN reference scenario

With the advent of the clouds, enterprises outsource their applications and use Software-as-a-Service (SaaS) and Infrastructure-as-a-service (IaaS) from multiple cloud providers. In Figure 2, we represent the SD-WAN reference scenario, highlighting the connectivity needs of the SD-WAN customers. It is clear from the figure above that the hub-and-spoke communication model of traditional WANs (shown in Figure 1) was not designed with these concepts in mind and cannot meet the needs of today's digital businesses; because of this the user experience is poor or the costs are not sustainable. Moreover, the enterprises are interested to leverage multiple connection types across their WAN to improve application performance and reliability, and the end-user experience. Enterprises need to segment their applications and extend VLAN concepts to the cloud infrastructures creating independent slices with specific QoS and security requirements. Last but not least, customers require a simple interface (that is easy to configure and manage) and automation and orchestration features. From these requirements came the new idea behind SD-WAN which has been adopted by the majority of the vendors in the market.

With this paper we want to shed some light on the SD-WAN scenario and describe an open source implementation which can be taken as reference. We call this architecture EveryWAN. It has been designed with SDN and NFV principles in mind, and leverages Cloud best practices to deliver to the WAN customers and the MSP the same benefits and the agility of the Cloud service providers. Moreover, we strongly believe in the openness of the SDN/NFV paradigms which can ease the development of new services and can foster the innovation in the SD-WAN deployments.

OFELIA [4] in EU, GENI [3] and Internet2 [8] [9] in US were the first SDN solutions based on SDN capable switches inter-connected with a centralized controller and deployed in geographically distributed networks. OSHI [1] [2] proposed a hybrid device as a replacement to the WAN routers where IP routing and SDN coexist. Similarly, SRv6-SDN [10] provides an implementation of an IP/SDN architecture for IPv6 Segment Routing enabled WANs. Both solutions envisage a SouthBound protocol between the devices and the controller. OSHI integrated OpenFlow. Instead, SRv6-SDN implemented a new protocol that leveraged gRPC as a transport mechanism. All these solutions aimed at providing an alternative implementation of the geographical networks adopting SDN principles within the infrastructure but they cannot be considered SD-WAN solutions as meant nowday's.

The Google B4 WAN [11], [12] has likely been the first application of the SD-WAN approach to a large-scale WAN scenario. In the B4 solution the traditional distributed routing protocols coexist with a SDN/OpenFlow approach. In particular, the B4 WAN sites are interconnected with traditional routing and the SDN-based centralized Traffic Engineering solution is used to steer traffic flows across the sites. These flows are implemented as an overlay on top of basic routing. The Google B4 solution is proprietary and it is highly tailored to the needs of the Google scenario, and as such it does not represent a typical ISP WAN network. Several proprietary solutions ([13] and [7] to give some examples) already implement the SD-WAN architecture.

At the time of writing, FlexiWAN [14] is the only solution providing an open source alternative to EveryWAN. FlexiWAN envisages a more classic approach to SD-WAN with the control functionalities still running at the edge in the virtual routers. Compared with this work, we considered also the possibility of controlling the devices in a classical SDN fashion which led us to adopt an approach that is similar to OSHI and SRv6-SDN in some extents, where traditional IP protocols coexist seamlessly with a SDN SouthBound (SB) in the devices. On top the controller can leverage the SB protocol to manage and control the nodes. In this way, we can combine the fault-tolerance based on the regular IP routing together with the openness and the abstraction of a SDN control plane which can ease the development of new services and foster innovation. Moreover, thanks to this approach, the EveryWAN architecture is very flexible and can be tailored to different needs.

The source code of all the components of the EveryWAN architecture and the different tools that have been developed are published at [15]. In order to ease the initial setup of the solution for other researchers, everything has also been packaged in a ready-to-go virtual machine (available at [15]), with pre-designed example topologies that emulate different WAN scenarios, including the emulation of different broadband technologies. To the best of our knowledge, there is no such SD-WAN solution readily available in Open Source. The contributions of this paper are multifold:

- High level design of our open source SD-WAN architecture, called EveryWAN;
- Design and implementation of a virtual CE device made of open source components and built with SDN principles in mind;

- Design and implementation of a SD-WAN controller to control and program the edge devices;
- Compelling SD-WAN services like overlay networks and network slicing;
- Design and implementation of an orchestration layer through which customers or MSP can implement and manage the deployed SD-WANs;
- Design and implementation of an open reference environment to deploy and test EveryWAN and related network services.

The paper is structured as follows: Section 2 presents the high level architecture of EveryWAN. We will describe EveryWAN architecture using a bottom up approach. The design and the implementation of the vCE, called EveryEdge, is described in Section 3. EveryEdgeOS is the SDN controller that programs the edge devices, Section 4 describes the controller architecture and the southbound APIs. In Section 5, we describe the orchestrator layer, the devops and the management tools. Section 6 explains the testbed. We draw conclusions and highlight the next steps in the final Section of the paper.

2. EveryWAN architecture

EveryWAN has been designed with SDN and NFV principles in mind. CE devices are replaced by Universal Customer Premise Equipment (uCPE) boxes, that integrate computing, storage and networking on COTS hardware, giving the possibility of realizing new services as virtual functions to any site and optimizing the provisioning of the existing ones through orchestration. In general, any server providing computing, storage and network interfaces can be used as replacement of a CE equipment. This approach would also overtake the monolithic hardware trend of the legacy CE that has been a barrier for innovation for several years.

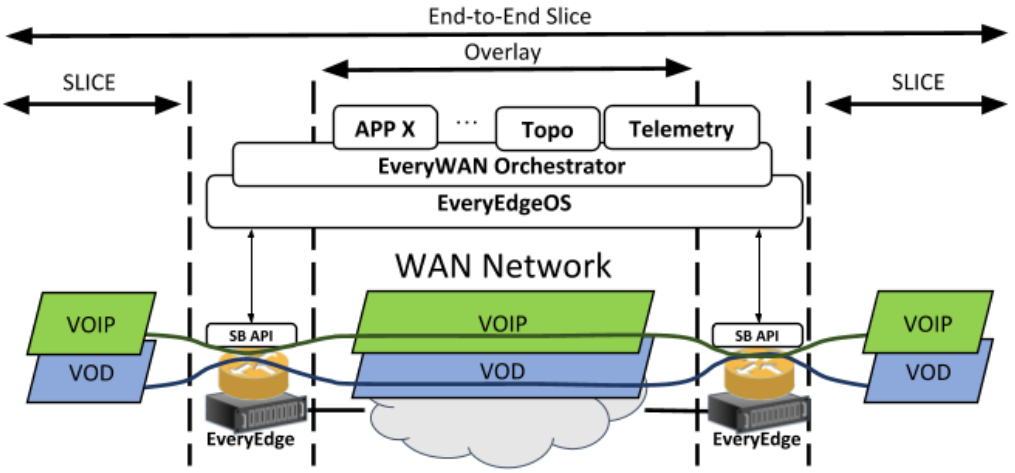

Figure 3 - EveryWAN architecture

Figure 3 shows the building blocks of the EveryWAN Architecture. Virtual CEs (vCE), the so called EveryEdge routers, are deployed as virtual network functions (VNF) and are controlled by the EveryEdgeOS. The SDN controller deals with many aspects of the device life cycle which includes not only the management and the programming but also the initial device registration, authentication and configuration leveraging a Zero Touch Provisioning (ZTP) approach. An overarching orchestrator sits on top of this SDN architecture, which orchestrates and automates the deployment of the virtual routers and of the SD-WAN services to any edge site on a network. The orchestrator offers also a GUI through which the tenants can design the network topology, configure services, manage the SD-WAN interconnections, the virtual devices and the users. The Network Operating System (NOS) and the orchestrator can run in a self-managed IAAS or in a public Cloud. Instead, the tenants will deploy the EveryEdge nodes in all the sites where SD-WAN interconnections need to be established.

We report in Figure 4 a more complex architecture where the EveryWAN control is extended to the border and the EveryEdge devices are also in charge of managing the LAN where the users are. This approach of extending the control to the LANs is typically known in literature as SD-LAN. In this particular use case, we envisage the possibility of deploying the edge devices and the SDN controllers locally in the same IAAS infrastructure available on any

SD-WAN site. There are inherent scalability benefits from having a layered architecture since each NOS controls a subset of devices. However, each deployment scenario can be customized based on the user needs.

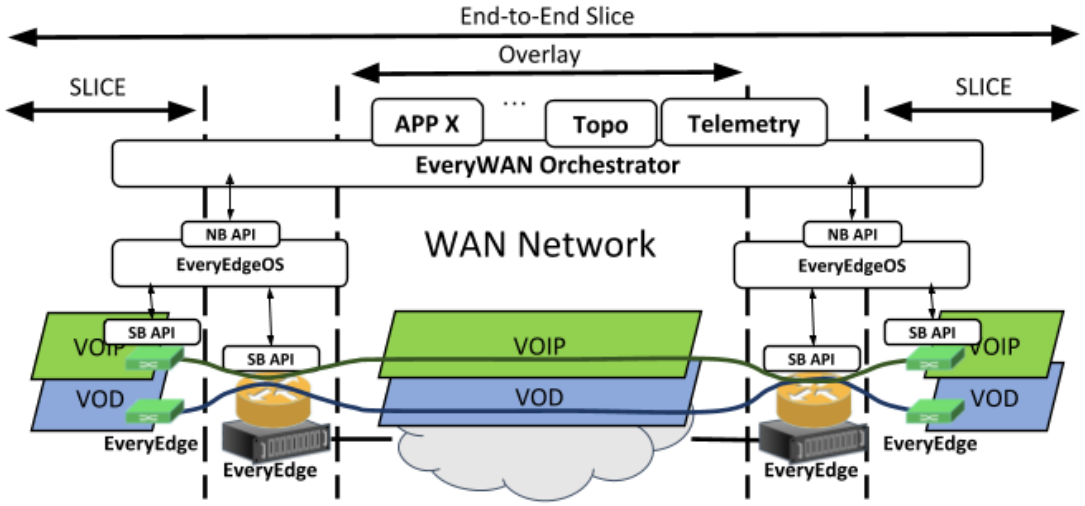

Figure 4 - Extended EveryWAN architecture

In our first design, EveryWAN does not include any IAAS management functionality and we assume that the EveryWAN users will deploy the vCE images in their self-managed uCPEs distributed across the WAN sites. The container/VM images are shipped with a minimal configuration which allows the virtual routers to reach the controller, download the configuration and complete the provisioning process.

As regards the services, the first shift is how the private interconnections are realized. PE based VPNs are now replaced by CE based VPNs, overlay technologies are used in place of legacy VPNs based on MPLS to realize end-to-end connectivity and build "virtual links" on top of the WAN pipes - which can be made secure using technologies like IPSEC [16]. With reference to Figure 3 and Figure 4, we call Slice the edge segments where the applications and users are. They are terminated in the LAN ports of the vCE. The overlays are established between virtual endpoints in the vCE and are logically terminated in the WAN interfaces of the vCEs. Finally, we define End to End Slices (E2E Slice) the composition of Slices and Overlays/Tunnels.

Existing overlay mechanisms provide the ability of building different logical instances of a multipoint network over the same WAN. This means that different applications can run on different slices and obtain the needed isolation requirements. An overlay approach has also the inherent advantage of abstracting the transport layer and being less dependent on the service providers and their networks. Thanks to this approach manyfold broadband technologies can be leveraged together with MPLS (or as a backup): overlays are not tied to a specific WAN and can use different connections, also in parallel to implement load balancing policies and guarantee better performance. WAN connections can be selected also on a schedule basis, in this case the edge nodes will classify the packets and send them accordingly to the scheduling decisions made by the SD-WAN tenant.

3. EveryWAN services

We designed and implemented a basic service called EveryWAN Overlay Network (EON) as the basic building block of the EveryWAN architecture. EON can be used to support VPN use cases (e.g. interconnect different branch offices of a company through the ISP WAN) as well as to transport traffic of specific applications. EON can be seen as an implementation of a decentralized free Internet, where the connections are established at the edge and the data is managed by the same users. The overlay is realized between end-points in vCEs belonging to the same tenant. The end-points are logical ports: Virtual Tunnel Endpoint (VTEP) on a physical port. At the time of writing, the connection is established only in a full-mesh fashion. In our current implementation, the overlays can be realized by means of different technologies which include VXLAN [17] and IPv6 Segment Routing (SRv6) [18].

These tunnelling mechanisms allow us to build an even more powerful construct over the EON service that is typically called in the industry as Network Slicing. Network slicing provides the means of building several instances of virtual networks over the same WAN connection. A typical usage of this functionality is the service oriented SD-WAN where tenants can redirect the traffic of a specific application over a defined Slice. In this way, different applications can run in isolation and still share the same connectivity. We represent in Figure 5 and in Figure 6 the different types of slicing that are supported by EON at the time of writing: Switched End-to-end Slice and Routed End-to-end Slice.

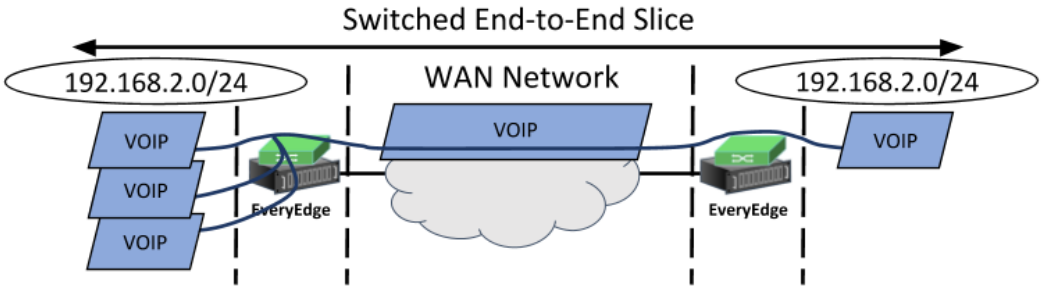

Figure 5 - Layer 2 Slices

The proposed Layer 2 Slice (represented in Figure 5) guarantees the IP endpoints to be directly interconnected as if they were in the same Ethernet LAN and sending each other arbitrary packets including Layer 2 protocols like ARP and NDP. This use case can be leveraged each time there is a need to preserve the original content of the user packets. Instead, the Routed slice, shown in Figure 6, is a simplified implementation of a L3VPN where the users attached to the remote sites belong to different broadcast domains and each site's endpoint acts as gateway for these broadcast domains. It is not meant to allow the served end-points to send packets with arbitrary Ethertype since only Layer 3 traffic will be transported across the remote sites. Moreover, in this scenario it is possible to set up several broadcast domains behind the slice endpoints using a multi-subnet configuration (Figure 6).

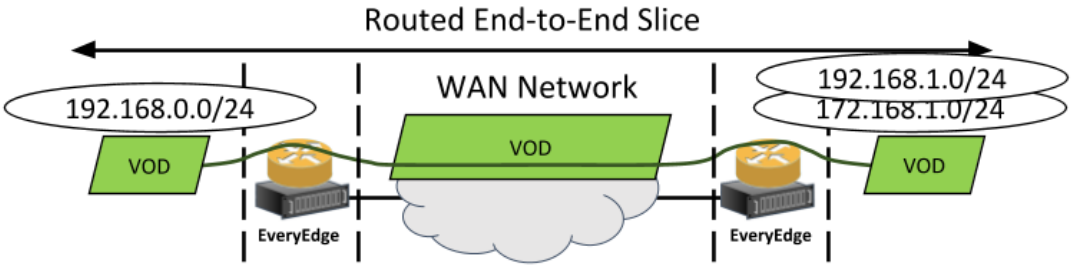

Figure 6 - Routed Slices

EveryEdge nodes can leverage multiple WAN connections to forward the traffic of the EON local slices. Overlays are not tied to a specific WAN: vCE can select which WAN interface to use for the forwarding of the traffic belonging to a particular slice based on a customer defined policy. An important mechanism for implementing this slice scheduling is the ingress classification performed by the vCE. In our first implementation, it can be either based on the physical input port or on a virtual input port - we use the so called VRF lite approach [19] where each interface is individually mapped to a Slice.

4. EveryEdge Node Architecture

We have built our Open Source virtual CE, the so called EveryEdge router, combining a Programmable IP Forwarding Engine (P-IPFE), an IP routing daemon (IPRE) and a Southbound API (SB API). The EveryEdge architecture foresees the coexistence of a local control logic based on distributed IP routing and of a classic SDN design in which the node implements a Southbound API towards a SDN controller. Regarding the P-IPFE, it is programmable in the sense that the SDN controller leveraging a Southbound protocol can instruct the nodes and program the forwarding entries in its Forwarding Information Base. Similar solutions have been already proposed in literature, i.e. [2] and [10]; others have been rolled out in production recently [20]. While these solutions, often referred to as hybrid IP/SDN, have been applied in datacenter fabric or in WAN scenarios as replacement of the WAN routers, the novelty of our approach lies in proposing and utilizing such hybrid approach for the commoditization of the CE routers.

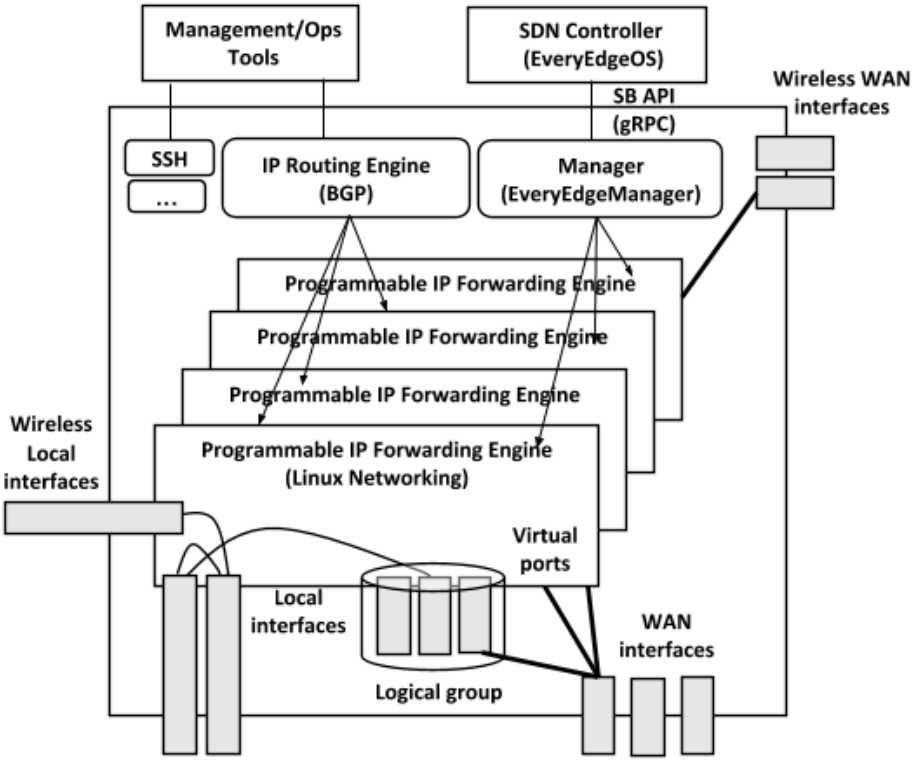

Figure 7 - EveryEdge node architecture

Figure 7 shows the high level architecture of the EveryEdge node and its main components. Each P-IPFE is connected to a number of local interfaces and a number of virtual ports. These two groups participate both in a specific Slice. The traffic is forwarded from the local interfaces to the virtual ports and vice versa according to L2/L3 rules. The local ports face always the edge network where the users requiring connectivity are attached to and transport the traffic of a specific service. The virtual interfaces are Virtual Tunnel Endpoint (VTEP); they are responsible for encapsulating the Ethernet frames coming from the users and decapsulating the user frames from the packets coming from the WAN interfaces. Tunnels are established between VTEPs on different vCE to realize an E2E Slice. The virtual

ports are closely related to the WAN interfaces that are available in the bare metal servers; they can be of different types and can give access to different types of connections which can span from low-cost and commercially available broadband access (LTE/5G and broadband internet services) to a MPLS core. There is always a two-way relation between a virtual port and a WAN interface. The choice of using a given WAN is always defined by the EveryWAN customer. Additionally, if several virtual ports are available in a given P-IPFE, a logical group can be instantiated by the SDN controller and advanced forwarding behaviors can be leveraged by the traffic.

All the ports, except for the WAN interfaces, can be organized as Logical Groups; each one implementing a different type of forwarding. We defined in our architecture the following behaviors: i) Hashed; ii) Weighted Hashed; iii) Failover; iv) Broadcast; v) Intelligent. Interfaces belonging to a logical group are seen as a single forwarding entity exhibiting a specific behavior. For example, a Hashed group implements an Equal Cost Multi Path forwarding that leverages the interfaces participating in the group. A Weighted Hashed allows to prefer some interfaces with respect to others assigning different weights to the virtual ports. Failover groups can be used to implement fast recovery mechanisms. Many to many communications are implemented using Broadcast groups. Finally, Intelligent groups are used together with an active monitoring mechanism to select each time the best interface according to some objectives specified by the tenant. The group type is strictly dependent on the type of service that has to be supported - for example a Broadcast group is used to implement a L2 Slice. In our first implementation, EveryWAN does not include the support for Logical Groups. We plan to add this functionality in the future.

We have realized EveryEdge node leveraging commodity hardware and open source software. We did not reinvent the wheel but when needed we built from scratch the missing functionalities. The IP forwarding engine is implemented using the Linux networking. We have used a general purpose distribution of the Linux OS, the only requirement is to have a recent kernel version (at least 4.14) in order to have native support for VXLAN operations in the kernel space. We have leveraged the approach designed in [10] to make it programmable.

We have also used several virtualization technologies offered by the Linux kernel to build up our slicing mechanisms and multi tenancy. In particular, the VRFs are used to construct several instances of P-IPFE logically decoupled. Virtual interfaces and VTEPs drivers are involved to instantiate virtual Ethernet interfaces that encapsulate/decapsulate traffic using a specific tunneling technology. A Routing Engine and the EveryEdge controller sit on top of the P-IPFE instances and implement proper isolation mechanisms to make sure that the P-IPFEs are separated “jails”. In this way, we are able to guarantee that the routing information of a P-IPFE instance cannot be mixed up with the Routing Information Base (RIB) of another P-IPFE. Nor the forwarding of P-IPFE X can interfere with the forwarding decision taken in P-IPFE Y, for example by overriding the rules.

The EveryEdge node with its virtualization mechanisms guarantees the IP applications running on top of the Slices to be directly interconnected as if they were using their own routers. Moreover, from an architectural standpoint this arrangement closely resembles a

chassis based router having the dataplane cards and the routing engine in the same rack. The P-IPFEs are essentially the line-cards and the backplane of the chassis router, while the IPRE together with the EveryEdgeOS represent the redundant route processors.

We foresee the coexistence between the control logic based on distributed routing control protocols and the SDN approach: BGP programs the VRFs using its internal logic and allows the vCE at the edge of the overlay to exchange the basic reachability information, in this way the IP forwarding can be always used as default choice by the services running in the network slices. We have integrated BGP implementation of Free Range Routing project [21] in our node as IPRE. The decisions taken by the routing protocol may be overridden by the SDN controller which programs the nodes leveraging the Southbound API (SB API) exposed by the EveryEdge node itself. Controller and devices talk to each other using a protocol based on gRPC technology [22] .

Another important component of this architecture is the EveryEdgeManager. It takes care of the initial configuration of the node: implements a ZTP approach which includes also the download of the bootstrap configuration of the routing daemon; it authenticates the device with the controller and implements NAT traversal protocols with the help of additional control plane components. It supports in-band and out-of-band connectivity and it can establish insecure or secure channels with the control plane. This wide range of choices avoid the need of setting up a separate out-of-band network and the same WAN interfaces can be used to reach the EveryEdgeController.

Moreover, the EveryEdgeManager acts as a mediator interacting on the south with the P-IPFE instances and on the north with the SDN controller: it translates the messages received over the Southbound API into actions to be sent to the kernel components. The communication library used to enable the communication with Linux kernel is based on the open source project pyroute2 [23], a pure python netlink library that has been properly extended to support our needs. Last but not least, a number of management/operational protocols are used for the daily check routines, to keep alive the sessions with the controller and avoid the expiration of the NAT bindings.

The virtual slices are implemented through end-to-end tunnels established between VTEPs and then mapping local interfaces and VTEPs into a specific slice; we do not expect to change how IP networks operate in these days. Therefore, we decided to use VXLAN and IPv6 Segment Routing (SRv6) encapsulation as a coexistence mechanism. This allows us to interoperate naturally with existing IPv4/IPv6. The only caveat is that SRv6 requires a WAN network at least supporting IPv6 transport.

As for the ingress classification, we support, at the time of writing, a VRF lite approach where each interface is individually mapped to a slice and enslaved to the VRFs serving that slice. Additionally, if local trunk ports are available in the edge nodes, VLAN interfaces can be used in place of the physical local interfaces. As regards the operations at the egress, the edge router will extract the traffic from the tunnel and will forward it to the appropriate VRF. Finally, it will be delivered to the users according to L2/L3 rules defined for the slice.

The creation of the tunnels and the E2E slices is always initiated by the EveryEdgeController which receives the configuration on its NorthBound API. It allocates the slice specific information like the segment IDs (each tunneling mechanism has its own concept) and then translates the configuration in a set of commands to be performed on the participating vCEs. The entire end-to-end process is orchestrated by the EveryEdgeController.

5. EveryEdgeOS and EveryWAN Orchestrator

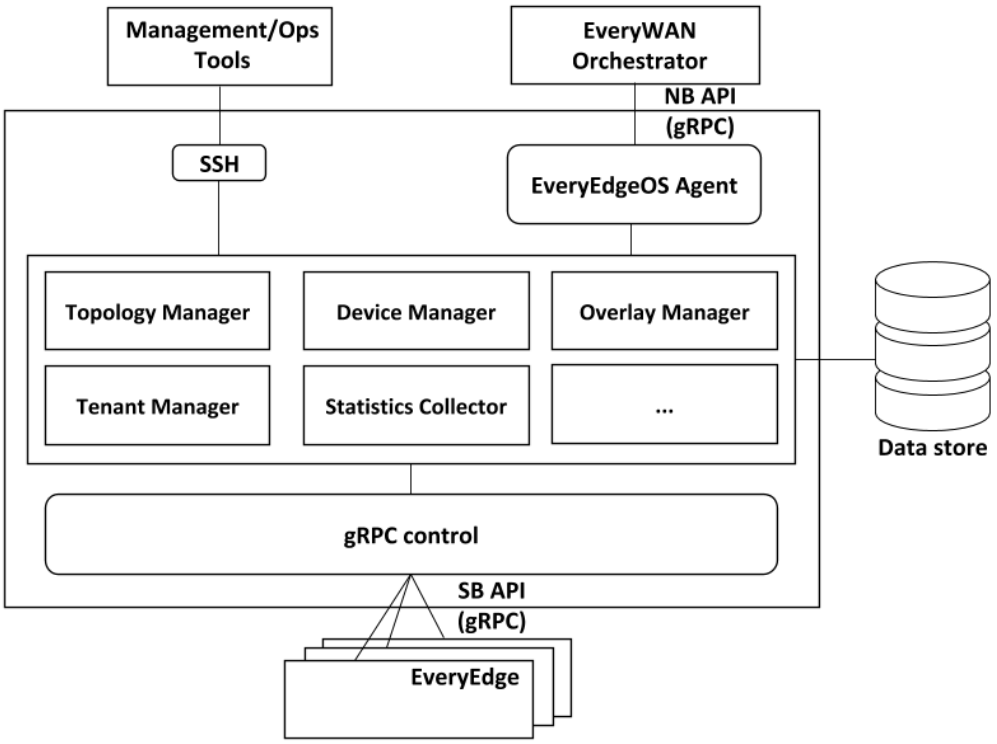

Figure 8 - SD-WAN controller architecture

We have realized from scratch a prototype of a SD-WAN controller, called EveryEdgeOS. EveryEdgeOS is responsible for the registration of the tenants, the registration and authentication of the EveryEdge devices and the provisioning of the services requested by the tenants. The key components of EveryEdgeOS are shown in Figure 8. At the highest level there is the EveryEdgeOS Agent, which acts as a mediator interacting on the south with the controller modules and on the north with the EveryWAN orchestrator. The EveryEdgeOS Agent translates the commands received from the orchestrator into actions to be sent to the EveryEdgeOS modules. In our current design, the controller has five modules: i) the Topology Manager (TM), which is responsible for building and maintaining an updated view of the network topology; ii) the Device Manager (DM), which deals with many aspects of the EveryEdge device life cycle, such as the device registration and authentication; iii) the Overlay Manager (OM), which computes paths and tunnels needed to implement the requested overlay by using the topology graph provided by the TM and the devices information provided by the DM; iv) Tenant Manager (TeM), which is responsible for the tenants registration and configuration; v) Statistics Collector (SC), which collects and reports statistics on the overlay networks and the devices. In the proposed architecture, all the controller modules are designed to be stateless. All the information including statistics, overlays, tenants, devices and the network graph is persisted on a data store, which can be either external or internal to the controller. Storing data externally to the controller has advantages from the resiliency point of view, as the controller failures do not affect data. In our current implementation, we store data on an external MongoDB database [24].

At the lowest level, the EveryEdgeOS interacts with the EveryEdge devices to program and control them. The communication between the EveryEdgeOS and the EveryEdge devices is handled via an interface called Southbound API. We decided to extend the Southbound API proposed in [10] to support the functionalities needed to control an EveryEdge device, such as the setup of VXLAN tunnels and the configuration of the routing tables. In [10], the authors have compared four different implementations: gRPC, REST, NETCONF, SSH/CLI. In this work we have adopted the gRPC solution for the API implementation.

On top of the SD-WAN controller there is an orchestrator called EveryBOSS, which is shown in Figure 9.

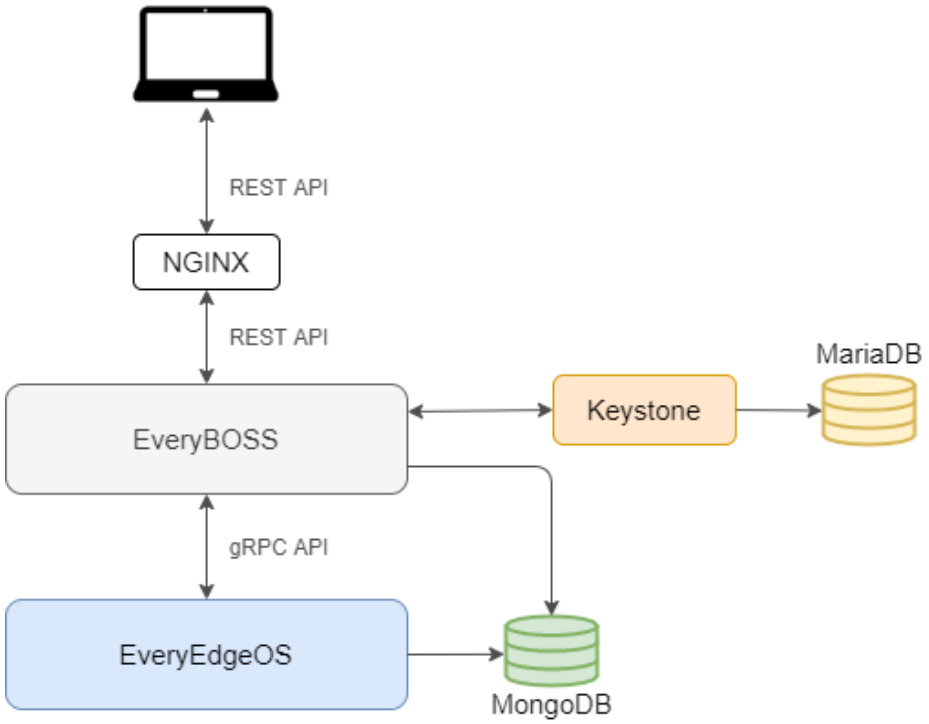

Figure 9 - SD-WAN orchestrator and EveryGUI

The orchestrator offers a GUI called EveryGUI through which the tenants can design the network topology, configure the services, manage the SD-WAN interconnections, the virtual devices and the users. The communication between the EveryBOSS and the EveryGUI is handled via a REST interface. The commands received from the GUI are sent to the SD-WAN controller through the Northbound interface exposed by the SD-WAN controller. Moreover, the EveryBOSS interacts with a Keystone instance to support user registration and authentication functionalities.

6. EveryWan Validation and Performance Evaluation platform (EVPE platform)

We have built an open reference environment on top of the Mininet [25] emulator to deploy and test EveryWAN and related network services. We call this environment EveryWan Validation and Performance Evaluation platform (EVPE). The EVPE includes pre-designed example topologies which support the emulation of different WAN scenarios, including the different broadband technologies and NATed environments.
With EVPE it is possible to measure various performance indicators inside a topology. In the dataplane it can compare the throughput and the CPU/memory usage between VXLAN and SRv6. It is also possible to measure different control plane aspects, such as the time needed to configure various services or the scalability.
Figure 10 shows the example of a SD-WAN scenario emulated in the EVPE.

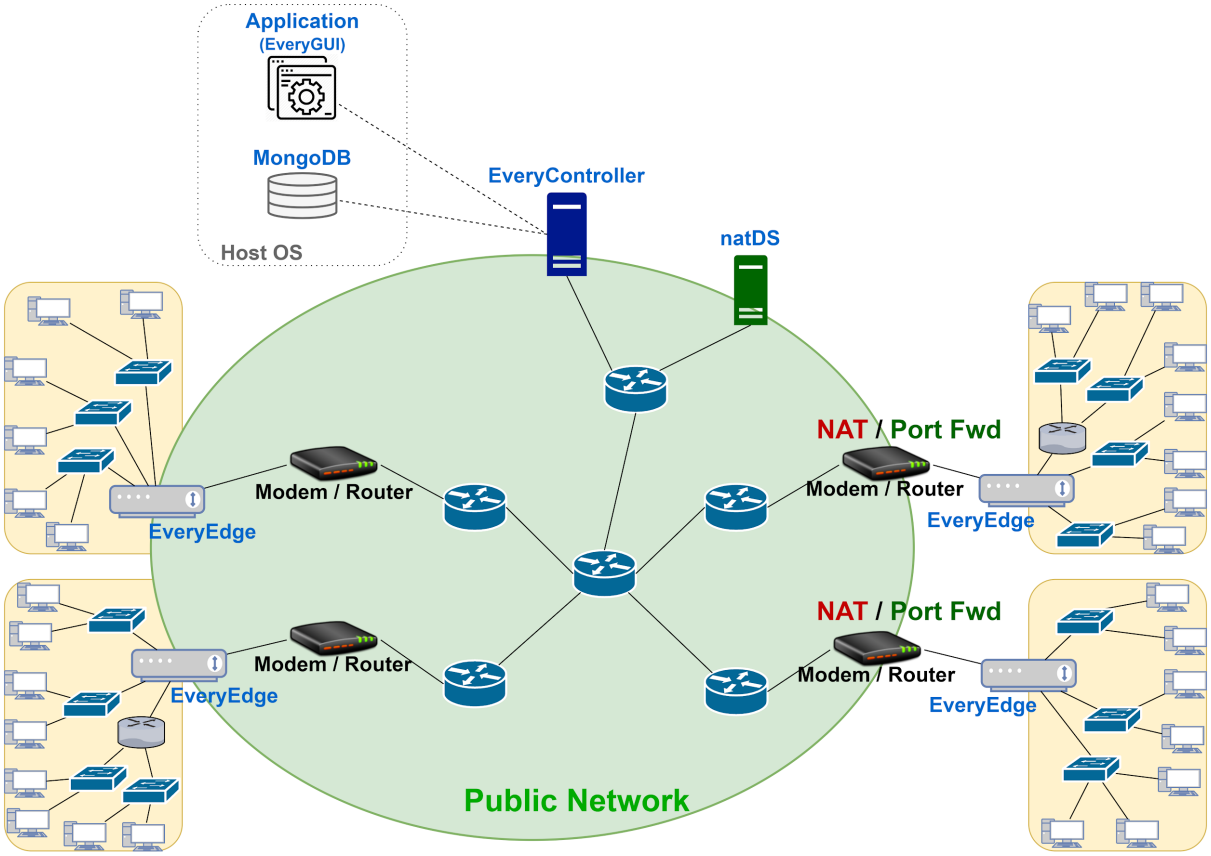

Figure 10 - Emulated topology

7. Future works

There are various improvements and new functionalities that can be included in the overall architecture in the future. Some of them are crucial for the functioning of the final architecture, some others could just add more features.
In the future, we plan to also support the composition of groups, since a group can be abstracted as a single logical interface and tied together to another group.
We plan to explore in the next iteration of the project the integration of programmable virtual switches like Open vSwitch [26] to implement advanced forwarding behaviors and "cross-layer" classifications of the packets.
We plan to extend the ingress classification adding the capability to redirect TCP/IP flows or leverage a “cross-layer” approach, similar to OpenFlow matching rules, that offers a great flexibility and can classify packets by considering packet headers at different protocol levels.
We are working to support different types of configurations like hub and spoke and will provide the users with means to specify the endpoints to be interconnected.
In the future, we will explore the possibility of automating the deployment process of the VMs/Containers through an integration of resources orchestrators like OpenStack [27] and Kubernetes [28].

8. References

[1] S. Salsano, P. L. Ventre, L. Prete, G. Siracusano, M. Gerola, and E. Salvadori, “OSHI - Open Source Hybrid IP/SDN Networking (and its Emulation on Mininet and on Distributed SDN Testbeds),” *2014 Third European Workshop on Software Defined Networks*. 2014 [Online]. Available: http://dx.doi.org/10.1109/ewsdn.2014.38

[2] S. Salsano *et al.*, “Hybrid IP/SDN Networking: Open Implementation and Experiment Management Tools,” *IEEE Transactions on Network and Service Management*, vol. 13, no. 1. pp. 138–153, 2016 [Online]. Available: http://dx.doi.org/10.1109/tnsm.2015.2507622

[3] “GENI.” [Online]. Available: http://www.geni.net. [Accessed: 30-Apr-2021]

[4] M. Suñé *et al.*, “Design and implementation of the OFELIA FP7 facility: The European OpenFlow testbed,” *Computer Networks*, vol. 61. pp. 132–150, 2014 [Online]. Available: http://dx.doi.org/10.1016/j.bjp.2013.10.015

[5] T. Sloane, “Software-Defined Networking: The New Norm for Networks - Open Networking Foundation,” 02-May-2013. [Online]. Available: https://opennetworking.org/sdn-resources/whitepapers/software-defined-networking-the-new-norm-for-networks/. [Accessed: 30-Apr-2021]

[6] *Benefits of SD-WAN - Cisco SD-WAN*. 2020 [Online]. Available: https://www.cisco.com/c/en/us/solutions/enterprise-networks/sd-wan/what-is-sd-wan.html. [Accessed: 05-May-2021]

[7] R. Nguyen, “SD-WAN Explained,” 12-Dec-2018. [Online]. Available: https://www.silver-peak.com/sd-wan/sd-wan-explained. [Accessed: 30-Apr-2021]

[8] “Home - Internet2,” 05-Dec-2019. [Online]. Available: https://internet2.edu/. [Accessed: 30-Apr-2021]

[9] “Website.” [Online]. Available: https://www.internet2.edu/communities-groups/advanced-networking-groups/software-defined-networking-group/. [Accessed: 30-Apr-2021]

[10] P. L. Ventre, M. M. Tajiki, S. Salsano, and C. Filsfils, “SDN architecture and southbound APIs for IPv6 segment routing enabled wide area networks,” *IEEE Trans. Netw. Serv. Manage.*, vol. 15, no. 4, pp. 1378–1392, Dec. 2018.

[11] S. Jain *et al.*, “B4,” in *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM*, Hong Kong China, 2013, doi: 10.1145/2486001.2486019 [Online]. Available: https://dl.acm.org/doi/10.1145/2486001.2486019

[12] C.-Y. Hong *et al.*, “B4 and after,” *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. 2018 [Online]. Available: http://dx.doi.org/10.1145/3230543.3230545

[13] *Benefits of SD-WAN - Cisco SD-WAN*. 2020 [Online]. Available: https://www.cisco.com/c/en/us/solutions/enterprise-networks/sd-wan/what-is-sd-wan.html. [Accessed: 30-Apr-2021]

[14] “SD-WAN & SASE - The World’s First Open Source,” 22-Jan-2021. [Online]. Available: https://flexiwan.com/. [Accessed: 30-Apr-2021]

[15] “EveryWAN.” [Online]. Available: https://github.com/everywan-io. [Accessed: 30-Apr-2021]

[16] K. Seo and S. Kent, “Security Architecture for the Internet Protocol,” Dec. 2005 [Online]. Available: https://tools.ietf.org/html/rfc4301. [Accessed: 30-Apr-2021]

[17] M. Mahalingam *et al.*, “Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks.” 2014 [Online]. Available: http://dx.doi.org/10.17487/rfc7348

[18] S. Previdi, D. Voyer, S. Matsushima, C. Filsfils, J. Leddy, and D. Dukes, “IPv6 Segment Routing Header (SRH),” Mar. 2020 [Online]. Available: https://tools.ietf.org/html/rfc8754. [Accessed: 30-Apr-2021]
[19] “Intro to VRF lite.” [Online]. Available: https://packetlife.net/blog/2009/apr/30/intro-vrf-lite/. [Accessed: 30-Apr-2021]
[20] A. Andreyev, “Introducing data center fabric, the next-generation Facebook data center network - Facebook Engineering,” 14-Nov-2014. [Online]. Available: https://engineering.fb.com/2014/11/14/production-engineering/introducing-data-center-fabric-the-next-generation-facebook-data-center-network/. [Accessed: 06-May-2021]
[21] “FRRouting.” [Online]. Available: https://frrouting.org/. [Accessed: 30-Apr-2021]
[22] “gRPC.” [Online]. Available: https://grpc.io/. [Accessed: 30-Apr-2021]
[23] “pyroute2.” [Online]. Available: https://pypi.org/project/pyroute2/. [Accessed: 30-Apr-2021]
[24] “The most popular database for modern apps.” [Online]. Available: https://www.mongodb.com. [Accessed: 06-May-2021]
[25] “Website.” [Online]. Available: https://mininet.org. [Accessed: 30-Apr-2021]
[26] “Open vSwitch.” [Online]. Available: http://openvswitch.org. [Accessed: 30-Apr-2021]
[27] “Open Source Cloud Computing Infrastructure - OpenStack.” [Online]. Available: https://www.openstack.org/. [Accessed: 30-Apr-2021]
[28] “Production-Grade Container Orchestration.” [Online]. Available: https://kubernetes.io/. [Accessed: 30-Apr-2021]