The integration of Ethernet Virtual Private Network in Kubernetes

Potter, F.J.M.
MSc Security and Network Engineering
University of Amsterdam
frank.potter@os3.nl

Supervisor: De Groot, A.
Cumulus Networks
attilla@cumulusnetworks.com

Abstract

Recent developments in data center architecture require distributing of tenants over the available hardware. This optimizes the resource utilization and thus cuts back costs. However, tenants expect a certain level of security and privacy in order to separate their resources from other tenants. Current Container Network Interfaces (CNIs) in Kubernetes use network policies to achieve this. The scalability of those policies can become an issue in an orchestrated containerized environment. This research introduces Ethernet Virtual Private Network (EVPN) as an alternative CNI in Kubernetes, that does not rely on network policies to provide secure multi-tenancy.

In order for EVPN to provide for secure multi-tenancy without network policies, an architecture based on the Kubernetes implementation was created. This architecture made use of a Virtual Routing and Forwarding (VRF) table and a virtual bridge per tenant. The VRF only contained the routes of containers that belong to the tenant, such that the tenant can only reach its own containers. Beside the L3 isolation provided by the VRF, a virtual bridge prevents malicious tenants from sniffing L2 traffic and spoofing the anycast Switched Virtual Interface (SVI) to become a Man-In-The-Middle (MITM).

The described architecture was tested in a Proof of Concept and showed that inter-host connectivity and multi-tenancy could be offered by EVPN without the use of network policies.

Keywords— EVPN, Multi-tenancy, Kubernetes, VRF, CNI

1 Introduction

Network Virtualization Overlays (NVOs) are used to provide logical separation over a shared physical network infrastructure [1]. Examples of NVOs are Virtual Extensible Local Area Network (VXLAN) and Network Virtualization using Generic Routing Encapsulation (NVGRE). NVOs were created as a response to the increasing network demand for server virtualization, and are more and more used in data center environments. Server virtualization allowed for a physical infrastructure to be shared between multiple tenants [2].

Multi-tenancy can be interesting for smaller tenants that only want a few containers that are spread over multiple hosts for redundancy. However, these smaller tenants do not want dedicated hosts, but rather share those hosts with other tenants to allow for tight packing of containers, which could optimize resource utilization and thereby reduce costs [2]. That said, sharing the host can be dangerous, since the other tenants are unknown and might be potentially malicious. Therefore, tenants expect a certain level of security and privacy in order to separate their resources from other tenants (i.e hard multi-tenancy) [3].

NVOs make use of network policies to provide for hard multi-tenancy. Network policies give control over the communication between containers and prevent one tenant its traffic to reach a different tenant [3]. However, in an orchestrated containerized environment, where one server can host hundreds of containers, the scalability of those network policies can become an issue [4].

EVPN that can be added to existing NVOs, can provide for hard multi-tenancy by logically separating the communication between pods of different tenants [5] [6] [7]. Therefore, it can prevent tenants from communicating in the first place, rather than blocking the undesired traffic later. Thus, EVPN does not rely on the network policies and the scaling issues of those network policies in a dynamic container environment.

EVPN has been adopted by Free Range Routing (FRR) in version 4.0, that was released in March 2018. FRR is an IP routing protocol suite for Linux and Unix platforms and allows for EVPN to be integrated on the host. This paper focuses on the analysis of EVPN on the host and how to integrate EVPN into an orchestrated containerized environment. The container orchestrator that is used in this research is Kubernetes, since Kubernetes was found to be the most actively used container orchestrator in the industry [8]. In order to evaluate EVPN as a multi-tenancy solution, we investigate the following research question:

"How can EVPN provide for multi-tenancy in a Kubernetes containerized environment?"

Two sub questions were derived from this question:

1. "What are the advantages of EVPN as a solution to implement multi-tenancy in a Kubernetes orchestrated container environment?"
2. "How can the recent improvements in FRR allow for EVPN to be integrated into the Kubernetes container orchestration platform?"

2 Related work

2.1 Network Virtualization Overlays

In order to describe the problem statement of the increasing demand for server virtualization in data centers and the lack of the data center networks to support this increasing demand, Narten, et al., [2] created RFC 7364. Server virtualization provides numerous benefits, including higher utilization, increased security, reduced user downtime and reduced power usage. However, the underlying network also needs to support this increasing demand for server virtualization. According to Narten, et al., data center networks can overcome this increasing demand by using an NVO that provides multi-tenancy. RFC 7364 describes traffic isolation as the main requirement for multi-tenancy. Another requirement that Narten, et al., give is address space isolation. Address space isolation means that different tenants can use the same address space within isolated virtual networks. Address space isolation becomes possible with traffic isolation, such that the address space is only visible to the tenants within the same isolated virtual network [2].

RFC 7364 provides a good description of NVOs as a solution to the increasing demand for server virtualization. However, the focus of RFC 7364 is on Virtual Machine (VM) virtualization instead of container virtualization. Nevertheless, we still rely on the terms and definitions specified in RFC 7364, since these terms and definitions are also relevant in container virtualization and container networking.

2.2 EVPN as a control plane NVO

RFC 8365 [9] describes EVPN as a control plane NVO solution to address the requirements of a multi-tenant data center. The control plane is used to provide for the management of traffic flows between NVO endpoints [1]. To allow for traffic flows within the network a data plane is needed. The data plane enables the forwarding of traffic in the network, and the control plane could be used to provide for logic to the data plane. In addition, Sajassi, et al., [9] specify the isolation of network traffic per tenant, the support for a large number of tenants, extending L2 connectivity and MAC mobility as the key requirements of EVPN as a control plane NVO solution. Furthermore, Sajassi, et al., state that the scaling properties of the control plane for an NVO are extremely important.

RFC 7432 [10] introduces the Route Types (RTs) that EVPN uses to address the requirements mentioned in RFC 8365. Beside the RTs 1-4 mentioned in RFC 7432, Rabadan, et al. [11] describe the later added RT-5 (i.e. the IP Prefix Route). RT-5 can be used to advertise host routes or a group of routes without MAC addresses, whereas RT-2 (i.e. the MAC/IP Advertisement Route) advertises host routes with the corresponding MAC addresses. In a larger environment, the movement of a large number of host routes can cause scaling issues with the RT-2, and therefore the RT-5 was created [11]. The RT-5 is more suitable to meet the scaling properties that in RFC 8365 were mentioned as extremely important. However, the RT-5 does not provide for MAC mobility, since it does not contain MAC addresses.

2.3 Container networking

Makowski and Grosso [12] conducted a study into the evaluation of virtualization and traffic filtering methods for container networks. They claim that container networks in essence are NVOs, where traffic between containers is encapsulated and sent through a tunnel. Furthermore, they claim that multi-tenancy is a major feature for modern NVOs. Therefore, Makowski and Grosso compared Identifier Locator Addressing (ILA) and EVPN based on multi-tenancy and ease-of-use. From this comparison it was found that EVPN was more flexible than ILA in terms of multi-tenancy and that EVPN was a more mature solution. Moreover, they state that ILA would offer a lightweight way of creating virtual networks that spans across multiple administrative domains, but requires more development effort before it could be used in a production environment. Lastly, Makowski and Grosso state that even though EVPN and ILA are capable of creating multi-tenancy in an orchestrated containerized environment, they both lack integration with a container orchestrator (e.g. Docker Swarm or Kubernetes). However, Makowski and Grosso state that EVPN can be adopted in a container network without much development effort [12]. Therefore, in our research we extend their study by proposing an architecture and Proof of Concept (PoC) of EVPN in container networking.

3 Background

In this section we describe Kubernetes and the networking rules it needs to comply with. We also discuss the most important CNIs that offer multi-tenancy as a comparison for EVPN. After discussing those CNIs, we go in more detail on how EVPN is added to BGP and the route types it provides.

3.1 Kubernetes

Kubernetes was created in 2015 as an open source cluster manager for container platforms, such as Linux Containers (LXC), Docker, etc. [13]. Kubernetes manages container platforms by having a cluster with a master and one or more nodes. The master can schedule and scale containerized applications through an Application Programming Interface (API) that communicates with the Kubelet agent on each node, as shown in Figure 1. Each node can have a pod, which is a group of one or more containers. The pod shares resources such as storage volumes, network namespace and container specific information, which the containers in that pod can use [14].

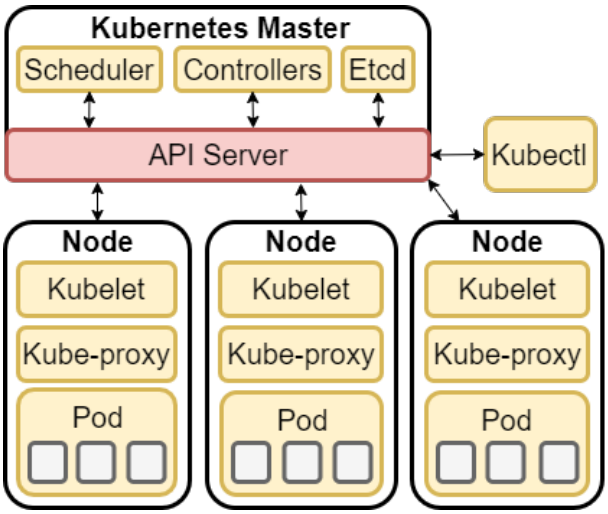

Figure 1: Kubernetes architecture [15]

Since containers in the same pod share the network namespace they can communicate with each other via localhost. Containers in separate pods are isolated from each other, unless pods use a virtual bridge to allow containers to communicate outside their pods. However, the virtual bridge does not provide inter-host communication. To allow for inter-host communication, routing functionality is needed. This could be done manually on each host, but this does not scale [14]. Therefore, inter-host communication in Kubernetes is often provided through a Container Networking Interface (CNI).

There are currently 25 different CNIs in Kubernetes, each with a different focus. Nonetheless, each CNI needs to comply to the Kubernetes networking rules [16], which are:

1. Pods on the same node must be able to communicate with other pods without the use of Network Address Translation (NAT).
2. All agents (e.g. system daemons, kubelet) running on a particular node can communicate with all the pods running on the same node.
3. Pods that use the host network must be able to contact all other pods on all other nodes without using NAT.

From the 25 CNIs 11 comply to those rules by using an NVO. The NVO is a virtual network that is built on top of the physical infrastructure (i.e. underlay) that creates tunnels between hosts by using encapsulation. However, the use of encapsulation will increase the packet size and due to the default Maximum Transmission Unit (MTU) size, the total number of frames that need to be transferred will increase. This leads to overhead in the network [17]. To overcome this overhead, the MTU size must be increased [14].

3.2 Container Network Interfaces

Before specifying how EVPN can be used as a CNI that does not rely on network policies, the disadvantages of network policies need to be specified. Therefore, the inner workings of two CNIs that rely on network policies to provide for hard multi-tenancy are described.

3.2.1 Calico

One of those CNIs is Calico. Calico runs a calico agent on each node in a Kubernetes cluster, which consists of a BIRD Internet Routing Daemon (BIRD) and a Felix agent. BIRD is a BGP routing daemon that is used to propagate routes between nodes. The Felix agent can write to the kernel's routing table and manipulate the iptables (i.e. network policies) of the node. When a pod is created, the Felix agent assigns an IP address to the pod and writes this IP address to the kernel's routing table. Then the BIRD agent advertises this new pod to other BIRD agents in the cluster. However, the use of internal BGP (iBGP) advertisements can become a limiting factor in larger deployments, since a full mesh is used with iBGP by default. As a result, the number of iBGP connections increase with N * (N - 1) / 2 for each added peer. In order to reduce this connection increase the BIRD BGP daemon can run as a BGP Route Reflector on some of the BIRD agents [14] [18].

In order for Felix and BIRD to function properly, it relies on Etcd. Etcd is a distributed Key-Value (KV) store that is used to store and share information between the different components that Calico needs to build a network [18]. On top of BGP, Calico can provide for an NVO. Calico does so by creating a tunnel between hosts and makes use of IP-in-IP (IPIP), as shown in Figure 2.

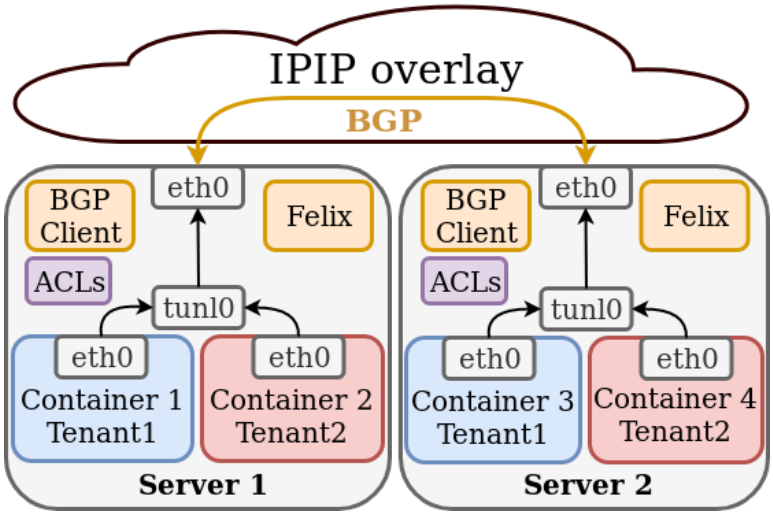

Figure 2: Calico multi-tenancy [14]

IPIP encapsulates the IP address of the container as the inner header and the IP address of the node as the outer header. Calico uses BGP to the host, which allows for L3 multi-tenancy. Nonetheless, Calico needs network policies to provide tenant isolation. Calico does so through the Kubernetes network policy API that makes use of the standard Linux network filtering technology iptables [14]. The disadvantage of iptables is that it gets quite big with thousands of containers. For example when a packet matches the last rule in the iptables filtering table, a traversal of the whole table is required, which is not very efficient. To avoid this issue, one can only check the first packet of a new connection with the current applied policies. However, the other packets of the connection are then accepted without verification. As a result, a network policy change will not affect any ongoing connections. Only when an ongoing connection has expired and was reestablished, the new policy has effect. [14]. Moreover, in a containerized environment where containers can have short lifetimes, an IP address first belonging to a container of one tenant can later be reassigned to a different container of a different tenant. Therefore, iptables is not designed for a large multi-tenant orchestrated container environments.

Another disadvantage of Calico is that IPIP is not a frequently used payload. However, the advantage of IPIP is that it only adds a 20 bytes IP header. Meaning that if the underlaying hardware supports IPIP, it is an ideal low overhead encapsulation solution [14].

3.2.2 Cilium

The other CNI that relies on network policies is Cilium. Cilium addresses the network policy management that has become a challenge in orchestrated container environments, where containers can have short lifetimes. To address these shortcomings Cilium uses extended Berkeley Packet Filter (eBPF) instead of iptables [4]. The eBPF is a Linux kernel bytecode interpreter created to filter network packets in kernel space [4]. Developers can write a program for eBPF, load it into memory, and then run it when certain events happen. Thus, with eBPF one can filter packets in a more granular and fine-grained manner [4].

In addition, Cilium focuses on labels as identity of containers, rather than IP addresses that can change frequently. Labels are KV pairs that are attached to objects such as pods or containers and are intended for users to identify attributes. Cilium keeps track of these labels and their identity by mapping them in a KV store. These labels allow Cilium to isolate pods by limiting access to certain pods based on their identity [14]. Cilium uses VXLAN as an NVO to allow for L2 multi-tenancy as shown in Figure 3.

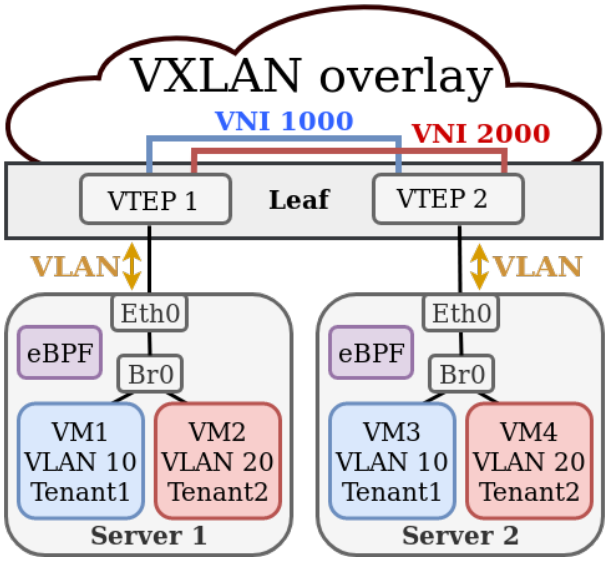

Figure 3: Cilium multi-tenancy [14]

In L2 multi-tenancy a VLAN is connected to a Virtual Network Identifier (VNI). In this example Tenant1 has VM1 on Server1 and VM3 on Server2, which both are in VLAN 10 that is connected to VNI 1000. Tenant2 has VM2 on Server1 and VM4 on Server2, which both are in VLAN 20 that is connected to VNI 2000. The VNIs span across the Virtual Tunnel End Points (VTEPs) to provide for the overlay [19]. Note that the tenants share the same bridge and therefore L2 traffic can be sniffed. However, if Cilium applies the earlier mentioned labels for pod isolation on the bridge, than Cilium can prevent this.

Makowski and Grosso [12] compared the performance of iptables and eBPF, and found that there was a slight filtering performance increase with eBPF. However, this performance increase was not very noticeable. Therefore, it seems that eBPF also has a negative effect on the network performance. Another disadvantage of Cilium is that VXLAN as overlay adds 50-54 bytes of overhead to the frame (Figure 4). This overhead needs to be considered with the MTU size (standard 1500 bytes), since it can cause fragmentation. Fragmentation puts extra load on the CPU and should therefore be avoided. However, when the underlaying hardware supports VXLAN offloading, the performance impact of the VXLAN overhead can be reduced [14].

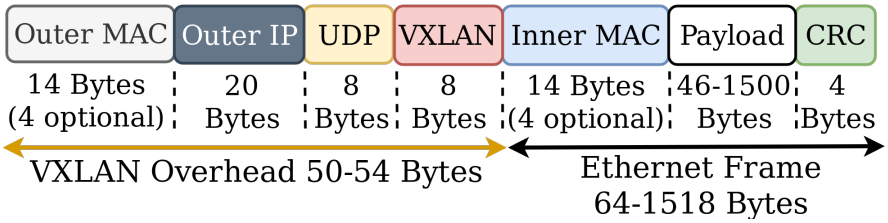

Figure 4: VXLAN Frame Format [20]

3.3 EVPN

EVPN, that does not rely on network policies, is added as an extended community to MultiProtocol-BGP (MP-BGP). EVPN was introduced as a new BGP Network Layer Reachability Information (NLRI), which is the EVPN NLRI as specified in RFC 7432 [10]. The EVPN NLRI makes use of the MP-BGP extensions Address Family Identifier (AFI) and Subsequent Address Family Identifier (SAFI) that have been defined for EVPN. For EVPN the AFI is 25 (i.e. L2 VPN) and the SAFI is 70 (i.e. EVPN) [10]. In the EVPN NLRI a Route Type can be specified. RFC 7432 describes the first four RTs and a draft of the IETF describes the later added RT-5 [11]. The five RTs are depicted in Table 1.

Table 1: EVPN Route Types [10], [11]

Type	Description
RT-1	Ethernet Auto-discovery Route
RT-2	MAC/IP Advertisement Route
RT-3	Inclusive Multicast Ethernet Tag Route
RT-4	Ethernet Segment Route
RT-5	IP Prefix Route

From this list RT-2, RT-3 and RT-5 are used in this research. The RT-2 advertisement route, as shown in Table 2, can be used to advertise MAC/IP bindings that are learnt locally [21]. In a RT-2, the EVPN Instance (EVI) is used to connect multiple nodes over an underlay, and the Route Distinguisher (RD) is used to specify the EVI the update is meant for. The Ethernet Segment Identifier (ESI) field is used in multihoming situations, where the VTEP informs the other EVIs there are multiple paths to reach the advertised MAC/IP address, and the MultiProtocol Label Switching (MPLS) label fields are used for the L2VNI and L3VNI [22]. The L2VNI is used for bridging between similar VNIs and the L3VNI is used for routing between different VNIs [21].

Table 2: EVPN RT-2 [10]

Field	Size	Description
Route Distinguisher (RD)	8 octets	RD of an EVPN Instance
Ethernet Segment Identifier (ESI)	10 octets	ESI of a connection to a peer
Ethernet Tag ID	4 octets	VLAN ID
MAC Address Length	1 octets	Length of host MAC
MAC Address	6 octets	Host MAC
IP Address Length	1 octets	Mask length of host IP
IP Address	0, 4 or 16 octets	Host IP
MPLS Label1	3 octets	L2 VNI
MPLS Label2	0 or 3 octets	L3 VNI

The Inclusive Multicast Ethernet Tag Route (i.e. RT-3) is used for the delivery of Broadcast, Unknown Unicast and Multicast (BUM) traffic across the remote VTEPs. Meaning that without a RT-3 advertisement, the ingress VTEP would not know how to deliver BUM traffic to the other VTEPs [10]. The RT-3 EVPN NLRI consists of a RD, Ethernet Tag, IP address length and the originating router's IP address, as shown in Table 3. The RD, Ethernet Tag and IP address length are similar as in the RT-2. The originating router's IP address is a new attribute that specifies the IP address of the originating VTEP (e.g. the loopback address) [23].

Table 3: EVPN RT-3 [10]

Field	Size	Description
RD	8 octets	RD of an EVI
Ethernet Tag ID	4 octets	VLAN ID
IP Address Length	1 octets	Mask length of host IP
Originating router's IP address	4 or 16 octets	Host IP

The RT-5 or IP Prefix Route advertisement, as shown in Table 4, consist of a RD, ESI, Ethernet Tag and have the same means as in the RT-2. Furthermore, it consists of an IP Prefix length and the IP prefix to be advertised. The RT-5 also contains the gateway IP address, which is only needed if there is no L3VNI, and therefore is often set to zero. The MPLS label is the L3VNI used [11].

The RT-5 was designed in order to meet the need for dynamic and efficient inter-subnet connectivity in NVOs [11]. RFC 8365 [9] continued on the requirements for EVPN as an NVO solution. One of these requirements is the extension of L2 connectivity among VMs belonging to a tenant, whether in the same data center or between different data centers. As mentioned in section 2.2 of the related work, RT-5 can be used to advertise host routes or a group of routes (i.e. aggregated) without MAC addresses. This brings some benefits to NVOs where certain inter-subnet forwarding scenarios are required [11]. An example of a RT-5 benefit is the mass withdrawal of routes, instead of invalidating each route individually, which is less efficient. Furthermore, as mentioned earlier these RT-5 advertisements could also be used to propagate routes between different data centers.

Table 4: EVPN RT-5 [11]

Field	**Size**	**Description**
RD	8 octets	RD of an EVI
ESI	10 octets	ESI of a connection to a peer
Ethernet Tag ID	4 octets	VLAN ID
IP Prefix Length	1 octets	Mask length of host IP
IP Prefix	4 or 16 octets	Host IP
Gateway IP address	4 or 16 octets	Underlay VTEP IP
MPLS Label	3 octets	L3VNI label

4 Methodology

Section 2.3 from the related work describes that inter-host communication is a difficult task in dynamic container networks and section 3.1 of the background describes that there exist 25 CNIs to target this task, each with a different focus. Since Makowski and Grosso [12] specify that multi-tenancy is a major feature for modern NVOs, our focus was to achieve multi-tenancy in a Kubernetes orchestrated container environment with the use of EVPN as an alternative CNI.

4.1 Requirements for multi-tenancy

In order to achieve multi-tenancy with the use of EVPN, the requirements that tenants have for multi-tenancy need to be specified. RFC 7364 [2] describes that the main expectations tenants have in a multi-tenancy data center is that a certain level of security and privacy is met in order to separate their resources from other tenants. An example of those expectations is traffic isolation and address space isolation, as mentioned earlier in section 2.1 of the related work, such that the traffic is isolated between tenants without restricting address allocation.

Other requirements are dynamic provisioning of network resources, MAC mobility, decoupling of logical and physical configuration, the scalability of the forwarding table, and optimal forwarding [2]. Dynamic provisioning of network resources, such as routing, is required in order to allow for quick changes in tenant systems and services. MAC mobility and the decoupling of logical and physical configuration is required for the movement of logical systems between physical systems in order to achieve higher utilization and allow for live migrations. Furthermore, virtualized environments have additional demands on the scalability of the forwarding tables in comparison to physical environments. Examples are location independence and the number of MAC addresses per server, requiring more forwarding table capacity than physical environments. Optimal forwarding requires the handling of east-west traffic within the data center, but also north-south traffic from the clients on the internet to the data center. Therefore, the best paths need to be chosen and network resources must be efficiently used [2].

4.2 Approach for EVPN as multi-tenancy solution

From these multi-tenancy requirements, we created an architecture. From the architecture, the design choices and there advantages were described, in order to answer our first subquestion ”What are the advantages of EVPN as a solution to implement multi-tenancy in a Kubernetes orchestrated container environment?”. Then the integration of EVPN in Kubernetes was tested on a virtual environment as a PoC. The virtual environment that was used is shown in Figure 5. Note that the servers communicate directly over L3, whereas in the earlier VXLAN as provided by Cilium, the servers used L2. This PoC is used to answer the second subquestion ”How can the recent improvements in FRR allow for EVPN to be integrated into the Kubernetes container orchestration platform?”.

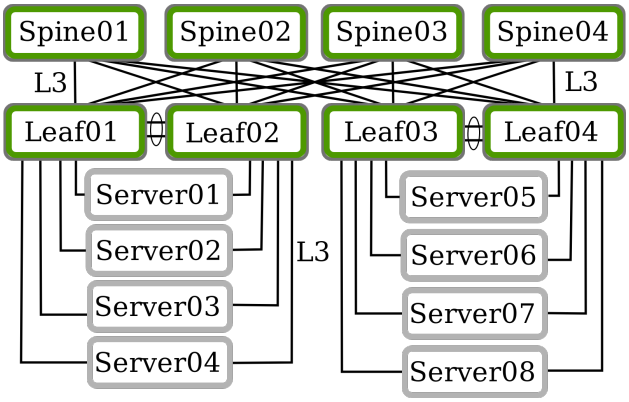

Figure 5: Virtual test environment

4.3 Experiments

To test whether our architecture was integrated as intended we performed two experiments. The first experiment is used to test the inter-host connectivity between containers of tenants on different nodes through the VXLAN overlay. From this experiment we could see if inter-host connectivity is possible with EVPN-VXLAN, and how the traffic flows over the network between the containers. The second experiment is used to test multi-tenancy, where multiple tenants share the node and therefore require isolation between tenants while maintaining connectivity between their containers on the different nodes.

4.3.1 Experiment 1: Inter-host connectivity and traffic flow

For the first experiment, we made use of two tenants that both have containers spread over four nodes (i.e. one container on each node). In this experiment a node only had one VRF, such that the overlay itself and the traffic flow could be thoroughly tested, to get an good understanding of EVPN-VXLAN and Kubernetes in the data center. Furthermore, it allows us to test the deployment of containers and the automatic advertisement of their IP address with FRR and EVPN.

4.3.2 Experiment 2: Multi-tenancy and isolation

The second experiment consisted of three tenants that have containers spread over three nodes (i.e. three tenants per node, nine containers in total), and therefore have multiple VRF tables on each node. The tenants must still have connectivity to their containers, but the connectivity must also be separated from other tenants to provide for hard multi-tenancy. Since, the host itself is shared and the tenants want their resources completely separated from other tenants. Therefore, the tenants must not be able to intercept any traffic of other tenants.

5 Architecture of EVPN for multi-tenancy

In order to allow for EVPN as a multi-tenancy solution, based on the multi-tenancy requirements (section 4.1), the design depicted in Figure 6 was created. The design choices to meet those requirements and the advantages are specified below.

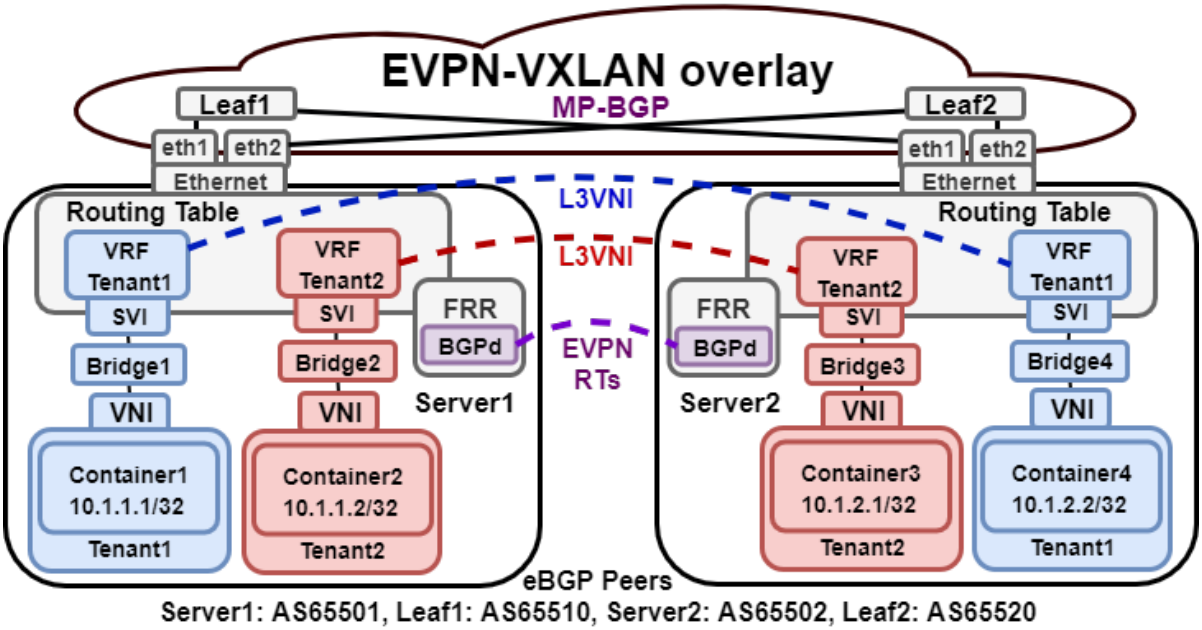

Figure 6: EVPN architecture for multi-tenancy

5.1 Traffic isolation and address space isolation

To comply with the main requirements, traffic isolation and address space isolation, we made use of an IP-VRF (i.e. L3VNI) and a virtual bridge for each tenant. The IP-VRF provides L3 traffic isolation by creating a separate virtual routing table per tenant. This VRF table only consists of the routes to the containers belonging to the tenant, such that the tenant can only reach its own containers. Beside the L3 isolation, a virtual bridge for each tenant was used to prevent a malicious tenant from sniffing L2 traffic and potentially advertising the anycast Switched Virtual Interface (SVI) MAC address. The anycast SVI acts as the default gateway, which might be spoofed in order for a malicious tenant to become a Man-In-The-Middle (MITM). The use of a separate VRF table per tenant also provides address space isolation. Since the VRF only contains the tenant specific IP addresses, a different tenant can reuse the same IP addressing. However, Kubernetes requires every pod to have an unique IP address, and therefore currently does not reuse addresses. Despite Kubernetes not reusing addresses, there might be use for it in the future.

5.2 Dynamic provisioning

The dynamic provisioning of network resources is provided through the use of FRR and EVPN RT-5. Once a container IP address is imported in the corresponding VRF table, which will be described in detail in section 5.6, FRR advertises the IP prefix through an EVPN RT-5 to the same EVPN VRF instance on different nodes. Tenants on different nodes can then learn the IP prefix of the container and add it to its corresponding tenant VRF table. In our design we decided to advertise containers as a host route with EVPN RT-5, since RT-5 advertisements allow for mass withdrawal, container connectivity between different data centers, and the propagation of aggregated routes. However, the effect of the mass withdrawal and the aggregation of routes depends on the number of containers per node.

5.3 MAC mobility

MAC mobility is not provided by RT-5 advertisements. However, Kubernetes does not rely on MAC mobility in order to move containers through the nodes as was done with VMs. Kubernetes rather destroys the container and creates a new container on another node. Therefore, MAC mobility is not applicable.

If one still desires to allow for MAC mobility, then the RT-2 is needed. The RT-2 allows the node (i.e. VTEP) to directly advertise the locally learned MAC addresses to the other nodes (as mentioned in section 3.3), instead of having to relearn the MAC/IP address and the corresponding VTEP through the data plane. Furthermore, an anycast SVI is used to prevent the container from relearning the IP address of the default gateway. The SVI does this by sharing a virtual IP address that is used by the tenant across different nodes.

The disadvantage of control plane MAC learning, as done by RT-2, is that remote VTEPs that have no interest in the RT-2 advertisement still get the advertisement and have to discard it. VTEPs can have no interest in the RT-2 advertisement if it does not host the VNI that is advertised. As a result, unnecessary traffic is sent to the VTEPs that are not interested in the advertisement. The reason that all VTEPs receive these advertisements is that otherwise the spine switches need to keep track of the VTEPs that are interested in these advertisements [5]. Beside the unnecessary computation to provide this tracking, it also scales better when the spines do not have to track VNIs. Therefore, the default implementation of EVPN is not to keep track of the VTEP VNIs [5].

To overcome this disadvantage, the RT-3 advertisement could be used. The RT-3 allows for BUM traffic, which makes it possible to replicate unicast traffic (i.e. ingress replication) to all relevant remote VTEPs [23]. Therefore, only the VTEPs that host the VNI get the advertisement.

5.4 The scalability of the forwarding table sizes

The scalability of the forwarding table sizes is provided by splitting the forwarding table into smaller forwarding tables per tenant with the use of VRF. Furthermore, L3 networks are tunneled over a L3 network, instead of L2 networks tunneled over a L3 network. Meaning the L2 networks are not stretched and remain small. Kubernetes can manage a total of 300,000 containers spread over a maximum of 5,000 nodes [14]. Assuming a larger tenant has 5,000 containers, e.g. two container on each node, then the forwarding table of that tenant contains 4,998 routes (not counting the containers on the node itself). However, the two host routes for the containers can be aggregated to RT-5 Prefix routes for each node, reducing the number of routes to 2,499. Furthermore, the other tenants forwarding tables can remain small, whereas with other CNIs that use network policies, all the nodes share the forwarding table. As a result the forwarding table of those CNIs can consist of thousands of routes and thousands of network policies are required to prevent the communication between tenants.

5.5 Equal-Cost Multi-Path routing

The requirement for optimal forwarding for east-west traffic is provided through the use of Equal-Cost Multi-Path routing (ECMP) in EVPN. ECMP is specified as a requirement of EVPN in section 4.2 of RFC 7209 [24]. ECMP allows for the use of same cost paths to a destination and can provide for load balancing between those equal cost paths [24]. Especially in multi-homed scenarios as in our virtual test environment, where the node is connected to two leaf switches, the leverage of ECMP for load balancing purposes can increase the efficient use of network resources.

The north-south traffic can be achieved by mapping the tenant VRFs to a VRF-Lite or global routing table on the border leaf switches (as shown in Figure 7) that are connected to edge routers and provide external connectivity. These border leaf switches can use route leaking to "leak" routes from the tenant VRFs to the global routing table of the border leafs in order to advertise those routes to the outside world (as shown in Figure 6). Routing leaking becomes possible with the use of VRFs, since a VRF can leak routes to the global routing table or another VRF [25].

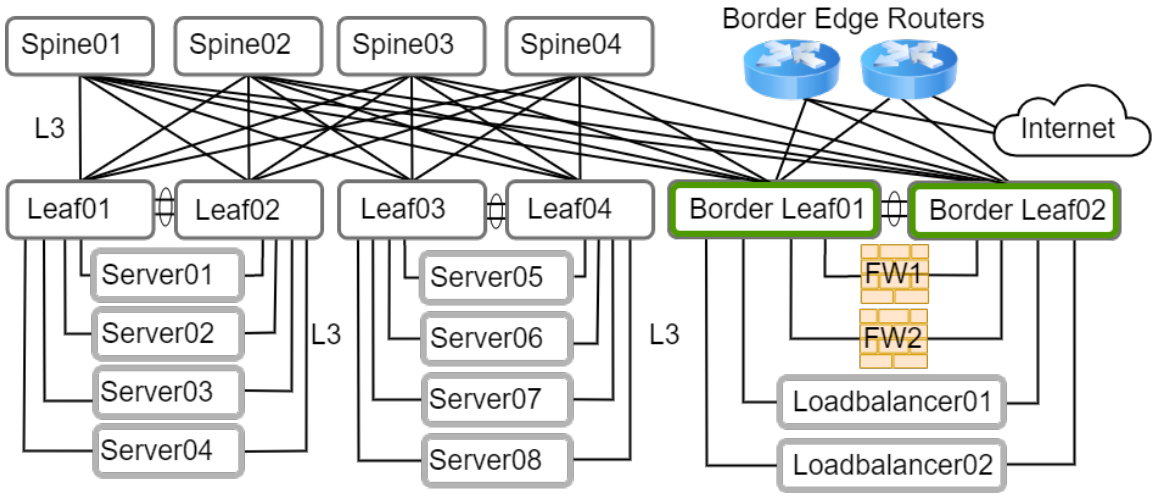

Figure 7: North-south traffic through the Border Leafs and Border Edge Routers.

In addition to ECMP, EVPN also allows for multihoming. Data planes such as VXLAN currently do not provide for multihoming. As mentioned earlier in section 3.3, EVPN allows for Ethernet Segment (ES) multihoming. An ES is a device or network that is connected to one or more links [9]. All-active multihoming allows for flow-based load balancing in order to fully utilize all the links. However, multihoming can also cause loops. To avoid loops, EVPN makes use of split-horizon, such that packets leaving one multihomed link are not sent back to one of the other multihomed links. An MPLS label is used by EVPN to support split-horizon filtering. This label contains the ESI. When an NVE is attempting to forward a multi-destination frame, it checks the label and if the ESI corresponds to the ESI of the outgoing interface, it will not be forwarded to that interface. Furthermore, EVPN selects a Designated Forwarder (DF) so that only one link is allowed to send BUM traffic. This prevents a multihomed device or network from delivering duplicate BUM traffic into the network. The election of a DF can be done for each VNI, which allows for load sharing [9].

5.6 Integration of EVPN in Kubernetes

For the integration of EVPN in Kubernetes as CNI, the IP addresses of the containers had to be added to the VRF table of the corresponding tenant. Then these IP addresses need to be advertised to other tenant VRF tables on different nodes. In order to get the container IP addresses of a tenant and advertise them to the VRF tables of that tenant on different nodes, we used the process as shown in Figure 8.

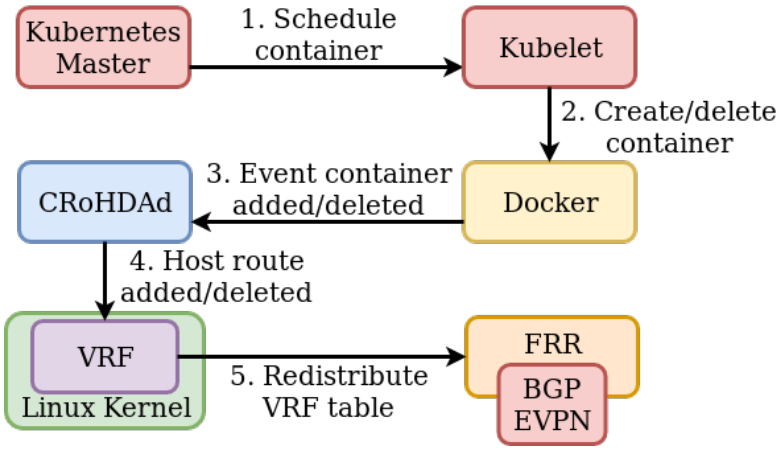

Figure 8: Integration of EVPN in Kubernetes flowchart

When the Kubernetes Master schedules a container to the Kubelet agent on the node, the Kubelet agent creates the container with Docker. The creation or deletion of a container triggers an event, which the Cumulus Routing on the Host Docker Advertisement daemon (CRoHDAd) listens to. The original CRoHDAd checks for the IP address of the container and adds or deletes the host route to a default routing table. We adjusted the CRoHDAd (see Github [26]), such that based on the labels provided in the Kubernetes deployment of the container, CRoHDAd could add or delete the host route to the corresponding VRF table. When the host routes are added to the tenant VRF, they get imported to FRR and advertised by BGP-EVPN.

6 Results

From the created architecture, the experiments were conducted (see section 4.3). In this chapter we present the results of the conducted experiments.

6.1 Experiment 1: Inter-host connectivity and traffic flow

For the first experiment we used Server2 and Server6 as Tenant1 and Server4 and Server8 as Tenant2 (depicted in Figure 9). Note that the last octet of the IP address of Container1 on Server2 is higher than the other containers. This is because Server2 was used a lot for the container creation to test the adjusted CRoHDAd. Since Kubernetes uses flexible IP address allocation, the IP address increased by 1 each time the container was recreated. Kube-dns can be used to provide a mapping between a container IP address and the naming of the container.

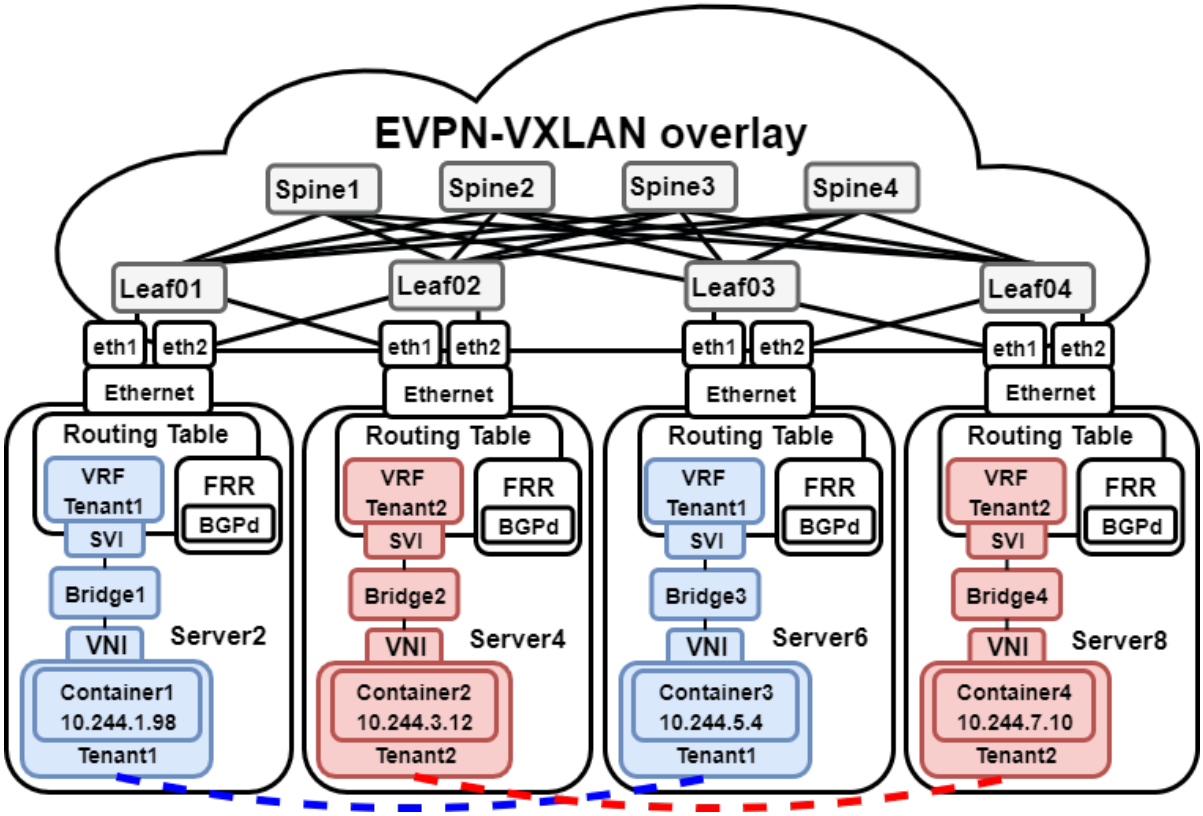

Figure 9: Experiment 1 - Inter-host connectivity between containers of the same tenant as if they were on the same host.

6.1.1 Experiment 1: Inter-host Connectivity

From the created architecture and the adjusted CRoHDAd, it was possible to create connectivity between Container1 and Container3 of Tenant1 through the use of VRF. Container2 and Container4 of Tenant2 also had connectivity through a different VRF table. Since each tenant had its own VRF table there was no cross connectivity between the containers of the different tenants (e.g. Container1 of Tenant1 to Container2 of Tenant2). A detailed overview of the results can be found on Github [27]. However, the global results of experiment1 are shown in the listings below. Starting with the overview of the containers, as shown in Listing 1.

```
root@server01:/home/cumulus# kubectl get pods -o wide
NAME             READY   STATUS    AGE         IP          NODE
my-deployment1   1/1     Running   2d9h    10.244.3.12   server04
my-deployment2   1/1     Running   2d9h    10.244.7.10   server08
my-deployment3   1/1     Running   2d5h    10.244.1.98   server02
my-deployment4   1/1     Running   2d5h    10.244.5.4    server06
```

Listing 1: Container overview

In Listing 2, the VRF tables on Server2, Server4, Server6 and Server8 are shown. Note that Server2 and Server6 are in VRF Tenant1 and Server4 and Server8 are in VRF Tenant2. The dev cni0 is the virtual bridge used. In experiment 1 only one virtual bridge was used, since each host only hosts one tenant.

```
Server2, VRF Tenant1:
root@server02:/home/cumulus# ip route sh vrf tenant1
10.244.1.98 dev cni0 scope link
10.244.5.4 via 10.250.250.16 dev dummysvi404001 proto bgp

Server4, VRF Tenant2:
root@server04:/home/cumulus# ip route sh vrf tenant2
10.244.3.12 dev cni0 scope link
10.244.7.10 via 10.250.250.18 dev dummysvi404002 proto bgp

Server6, VRF Tenant1:
root@server06:/home/cumulus# ip route sh vrf tenant1
10.244.1.98 via 10.250.250.11 dev dummysvi404001 proto bgp
10.244.5.4 dev cni0 scope link

Server8, VRF Tenant2:
root@server08:/home/cumulus# ip route sh vrf tenant2
10.244.3.12 via 10.250.250.14 dev dummysvi404002 proto bgp
10.244.7.10 dev cni0 scope link
```

Listing 2: VRF tables

The inter-host connectivity between the same VRF tables (e.g. Container1 of Tenant1 to Container3 of Tenant1) is shown in listing 3.

```
Container1 on Server2 towards Container3 on Server6:
root@server02:/home/cumulus# docker exec -it 5016fbcfc7cb sh
/ # ip address
3: eth0@if43: mtu 1450 qdisc noqueue state UP
    link/ether 9e:13:1f:86:ac:fe brd ff:ff:ff:ff:ff:ff
    inet 10.244.1.98/24 scope global eth0

/ # ping 10.244.5.4
PING 10.244.5.4 (10.244.5.4): 56 data bytes
64 bytes from 10.244.5.4: seq=2 ttl=62 time=2.033 ms
--- 10.244.5.4 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 2.033/2.128/2.273 ms

Container2 on Server4 towards Container4 on Server8:
root@server04:/home/cumulus# docker exec -it 129846c9053d sh
/ # ip address
3: eth0@if47: mtu 1450 qdisc noqueue state UP
    link/ether d6:fa:6a:5b:70:7f brd ff:ff:ff:ff:ff:ff
    inet 10.244.3.12/24 scope global eth0

/ # ping 10.244.7.10
PING 10.244.7.10 (10.244.7.10): 56 data bytes
64 bytes from 10.244.7.10: seq=1 ttl=62 time=1.593 ms
--- 10.244.7.10 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 1.593/1.609/1.625 ms
```

Listing 3: Connectivity between the same VRF

Listing 4 showed that no cross connectivity between Container3 on Server6 of Tenant1 towards Container2 or Container4 of Tenant2 was possible.

Container3 of Tenant1 towards Container2 and Container4 of Tenant2:

```
root@server06:/home/cumulus# docker exec -it f2c3b88744f3 sh
/ # ip address
3: eth0@if43: mtu 1450 qdisc noqueue state UP
    link/ether 5a:b4:0d:7d:66:2b brd ff:ff:ff:ff:ff:ff
    inet 10.244.5.4/24 scope global eth0

/ # ping 10.244.3.12
PING 10.244.3.12 (10.244.3.12): 56 data bytes
--- 10.244.3.12 ping statistics ---
3 packets transmitted, 0 packets received, 100% packet loss

/ # ping 10.244.7.10
PING 10.244.7.10 (10.244.7.10): 56 data bytes
--- 10.244.7.10 ping statistics ---
3 packets transmitted, 0 packets received, 100% packet loss
```

Listing 4: No cross connectivity between different tenants

6.1.2 Experiment 1: Traffic flow

Beside the connectivity between the containers of each tenant, we also looked into the traffic flow between Container4 on Server8 and Container2 on Server4, which are both of Tenant2. The results of the traffic flow are shown in Figure 10.

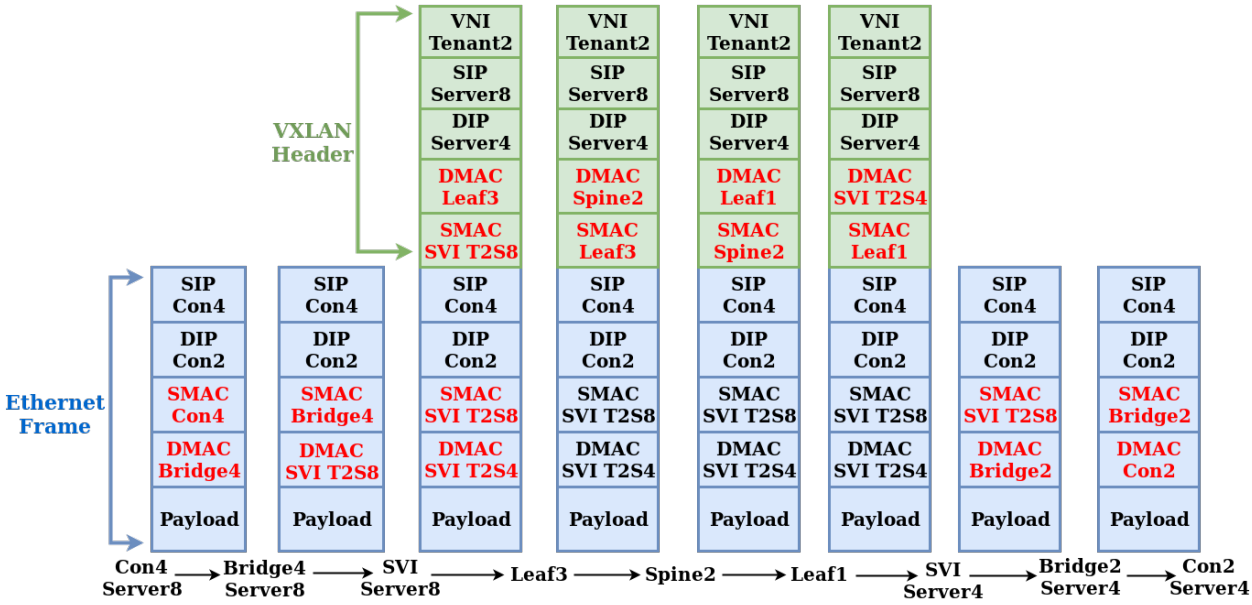

Figure 10: Experiment 1 - Traffic flow from Container4 on Server8 of Tenant2 towards Container2 on Server4 of Tenant2.

From this Traffic flow we could see that only the Source MAC (SMAC) and Destination MAC (DMAC) change throughout the traffic flow. The SMAC and DMAC change in the ethernet frame, until the traffic is encapsulated through the underlay with VXLAN. When the traffic is encapsulated with a VXLAN header from the Tenant2 SVI on Server8 (i.e. SVI T2S8) towards the Tenant2 SVI on Server4 (i.e. SVI T2S4), the SMAC and DMAC of the VXLAN header change, while the ethernet frame SMAC and DMAC remain untouched. When the Tenant2 SVI on Server4 obtains the encapsulated ethernet frame, it decapsulates it and forwards the ethernet frame to the bridge on Server4. In this traffic flow Leaf3, Spine2 and Leaf1 were chosen for the path through the underlay. However, other paths might also be chosen. For the SVIs on Server8 and Server4 it appears if they are directly connected. Also note that the SMAC and DMAC at Tenant2 SVI on Server8 change in both the ethernet frame as in the VXLAN header. This is because it first changes the ethernet frame and then encapsulate it with VXLAN.

6.2 Experiment 2: Multi-tenancy and isolation

In experiment 2 the multi-tenancy and isolation were tested. For this experiment Server3, Server5, and Server7 were used. As shown in Figure 11 each server was shared by three tenants, that had their own bridge, SVI and VRF.

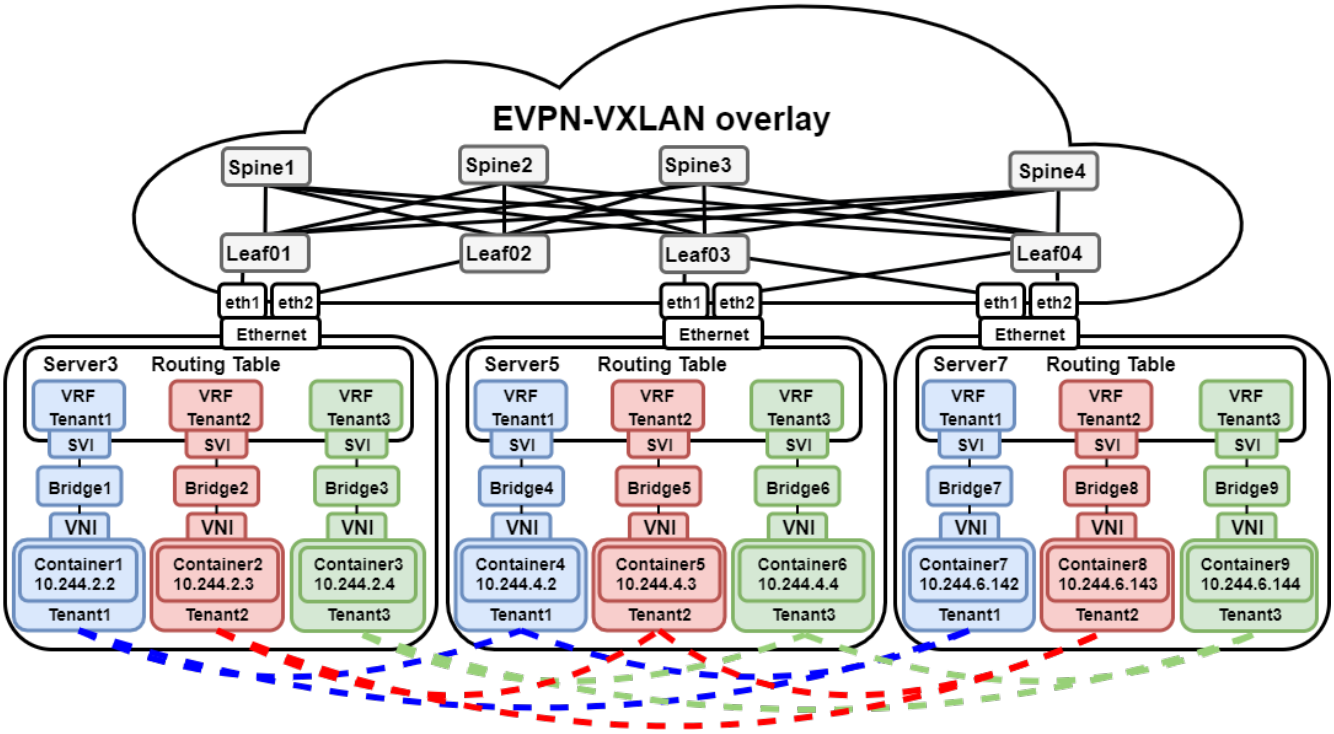

Figure 11: Experiment 2 - Multi-tenancy and isolation between tenants on the same nodes, while providing inter-host connectivity.

From the conducted experiment it was possible to create a VRF for each tenant on the same node that contained only the routes from that tenant (i.e VRF Tenant1 contains the routes of Container1, Container4 and Container7). The global routing table only contained the underlay IP addresses to reach the other nodes, and inter-host connectivity was provided between containers of the same tenant. Furthermore, there was no connectivity possible between containers of different nodes, and no traffic could be intercepted by containers on different bridges that belonged to a different tenant. This was tested by running a tcpdump on the virtual bridges belonging to different tenants during connectivity tests. As a result there was inter-host connectivity between the tenant its containers, while complete network separation was provided between different tenants. The global results of experiment 2 are shown in the listings below. The detailed results can be found on Github [28].

Listing 5 shows an overview of the containers and the corresponding node.

```
root@server01:/home/cumulus# kubectl get pods -o wide
NAME          READY   STATUS    AGE        IP           NODE
exp2-c1-vrf1  1/1     Running   58m    10.244.2.2     server03
exp2-c2-vrf2  1/1     Running   56m    10.244.2.3     server03
exp2-c3-vrf3  1/1     Running   55m    10.244.2.4     server03
exp2-c4-vrf1  1/1     Running   41m    10.244.4.2     server05
exp2-c5-vrf2  1/1     Running   39m    10.244.4.3     server05
exp2-c6-vrf3  1/1     Running   39m    10.244.4.4     server05
exp2-c7-vrf1  1/1     Running   13m    10.244.6.142   server07
exp2-c8-vrf2  1/1     Running   12m    10.244.6.143   server07
exp2-c9-vrf3  1/1     Running   11m    10.244.6.144   server07
```

Listing 5: Container overview

This overview showed there were nine containers, three on each node. All the three containers on a node were separated by using a different VRF table and virtual bridge. However, the containers are allowed to communicate to different containers on the other nodes, as long as they are in the same VRF table. E.g Container1 on Server3 is allowed to communicate with Container4 on Server5 and Container7 on Server7, which all belong to the VRF of Tenant1. The VRF tables on each node are shown in Listing 6.

Server3, VRF Tenant1:

```
root@server03:/home/cumulus# ip route sh vrf tenant1
10.244.2.2 dev cni0 scope link
10.244.4.2 via 10.250.250.15 dev dummysvi404001 proto bgp
10.244.6.142 via 10.250.250.17 dev dummysvi404001 proto bgp
```

Server3, VRF Tenant2:

```
root@server03:/home/cumulus# ip route sh vrf tenant2
10.244.2.3 dev cni1 scope link
10.244.4.3 via 10.250.250.15 dev dummysvi404002 proto bgp
10.244.6.143 via 10.250.250.17 dev dummysvi404002 proto bgp
```

Server3, VRF Tenant3:

```
root@server03:/home/cumulus# ip route sh vrf tenant3
10.244.2.4 dev cni2 scope link
10.244.4.4 via 10.250.250.15 dev dummysvi404003 proto bgp
10.244.6.144 via 10.250.250.17 dev dummysvi404003 proto bgp
```

Server5, VRF Tenant1:

```
root@server05:/home/cumulus# ip route sh vrf tenant1
10.244.2.2 via 10.250.250.12 dev dummysvi404001 proto bgp
10.244.4.2 dev cni0 scope link
10.244.6.142 via 10.250.250.17 dev dummysvi404001 proto bgp
```

```
Server5, VRF Tenant2:
root@server05:/home/cumulus# ip route sh vrf tenant2
10.244.2.3 via 10.250.250.12 dev dummysvi404002 proto bgp
10.244.4.3 dev cni1 scope link
10.244.6.143 via 10.250.250.17 dev dummysvi404002 proto bgp

Server5, VRF Tenant3:
root@server05:/home/cumulus# ip route sh vrf tenant3
10.244.2.4 via 10.250.250.12 dev dummysvi404003 proto bgp
10.244.4.4 dev cni2 scope link
10.244.6.144 via 10.250.250.17 dev dummysvi404003 proto bgp

Server7, VRF Tenant1:
root@server07:/home/cumulus# ip route sh vrf tenant1
10.244.2.2 via 10.250.250.12 dev dummysvi404001 proto bgp
10.244.4.2 via 10.250.250.15 dev dummysvi404001 proto bgp
10.244.6.142 dev cni0 scope link

Server7, VRF Tenant2:
root@server07:/home/cumulus# ip route sh vrf tenant2
10.244.2.3 via 10.250.250.12 dev dummysvi404002 proto bgp
10.244.4.3 via 10.250.250.15 dev dummysvi404002 proto bgp
10.244.6.143 dev cni1 scope link

Server7, VRF Tenant3:
root@server07:/home/cumulus# ip route sh vrf tenant3
10.244.2.4 via 10.250.250.12 dev dummysvi404003 proto bgp
10.244.4.4 via 10.250.250.15 dev dummysvi404003 proto bgp
10.244.6.144 dev cni2 scope link
```

Listing 6: Container overview

From this configuration inter-host connectivity between containers in the same VRF table was possible, as shown in Listing 7.

```
Container1 on Server3 towards Container3 on Server5:
root@server03:/home/cumulus# docker exec -it 44c5ed7a397c sh
/ # ip a
3: eth0@if43: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1450
    inet 10.244.2.2/24 scope global eth0

/ # ping 10.244.4.2
PING 10.244.4.2 (10.244.4.2): 56 data bytes
--- 10.244.4.2 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 2.582/4.094/5.607 ms
```

Container1 on Server3 towards Container7 on Server7:

```
/ # ip a
3: eth0@if43: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1450
    inet 10.244.2.2/24 scope global eth0

/ # ping 10.244.6.142
PING 10.244.6.142 (10.244.6.142): 56 data bytes
--- 10.244.6.142 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 1.936/3.688/6.886 ms
```

Container2 on Server3 towards Container5 on Server5:

```
root@server03:/home/cumulus# docker exec -it ffc82320dce8 sh
3: eth0@if44: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1450
    inet 10.244.2.3/24 scope global eth0

/ # ping 10.244.4.3
PING 10.244.4.3 (10.244.4.3): 56 data bytes
--- 10.244.4.3 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 2.380/3.856/6.285 ms
```

Container2 on Server3 towards Container8 on Server7:

```
root@server03:/home/cumulus# docker exec -it ffc82320dce8 sh
3: eth0@if44: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1450
    inet 10.244.2.3/24 scope global eth0
       valid_lft forever preferred_lft forever

/ # ping 10.244.6.143
PING 10.244.6.143 (10.244.6.143): 56 data bytes
--- 10.244.6.143 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 1.818/2.150/2.424 ms
```

Container3 on Server3 towards Container6 on Server5:

```
root@server03:/home/cumulus# docker exec -it c810d0212cfc sh
/ # ip a
3: eth0@if45: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1450
    inet 10.244.2.4/24 scope global eth0
/ # ping 10.244.4.4
PING 10.244.4.4 (10.244.4.4): 56 data bytes
--- 10.244.4.4 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 2.057/2.845/4.016 ms
```

```
Container3 on Server3 towards Container9 on Server7:
root@server03:/home/cumulus# docker exec -it c810d0212cfc sh
/ # ip a
3: eth0@if45: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1450
    inet 10.244.2.4/24 scope global eth0

/ # ping 10.244.6.144
PING 10.244.6.144 (10.244.6.144): 56 data bytes
--- 10.244.6.144 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 2.011/2.305/2.882 ms
```

Listing 7: Inter-host connectivity

Beside the inter-host connectivity between containers of the same VRF, no cross connectivity between containers in different VRF tables was possible (depicted in Listing 8).

```
Container9 (Tenant3) towards Container1 (Tenant1) on Server3:
root@server07:/home/cumulus# docker exec -it ee9e54605d60 sh
/ # ip a
3: eth0@if47: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1450
    inet 10.244.6.144/24 scope global eth0

/ # ping 10.244.2.2
PING 10.244.2.2 (10.244.2.2): 56 data bytes
--- 10.244.2.2 ping statistics ---
3 packets transmitted, 0 packets received, 100% packet loss

Container9 (Tenant3) towards Container2 (Tenant2) on Server3:
root@server07:/home/cumulus# docker exec -it ee9e54605d60 sh
/ # ip a
3: eth0@if47: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1450
    inet 10.244.6.144/24 scope global eth0

/ # ping 10.244.2.3
PING 10.244.2.3 (10.244.2.3): 56 data bytes
--- 10.244.2.3 ping statistics ---
3 packets transmitted, 0 packets received, 100% packet loss
```

```
Container9 (Tenant3) towards Container4 (Tenant1) on Server5:
root@server07:/home/cumulus# docker exec -it ee9e54605d60 sh
/ # ip a
3: eth0@if47: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1450
    inet 10.244.6.144/24 scope global eth0

/ # ping 10.244.4.2
PING 10.244.4.2 (10.244.4.2): 56 data bytes
--- 10.244.4.2 ping statistics ---
3 packets transmitted, 0 packets received, 100% packet loss

Container9 (Tenant3) towards Container5 (Tenant2) on Server5:
root@server07:/home/cumulus# docker exec -it ee9e54605d60 sh
/ # ip a
3: eth0@if47: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1450
    inet 10.244.6.144/24 scope global eth0

/ # ping 10.244.4.3
PING 10.244.4.3 (10.244.4.3): 56 data bytes
--- 10.244.4.3 ping statistics ---
3 packets transmitted, 0 packets received, 100% packet loss

Container9 (Tenant3) towards Container7 (Tenant1) on Server7:
root@server07:/home/cumulus# docker exec -it ee9e54605d60 sh
/ # ip a
3: eth0@if47: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1450
    inet 10.244.6.144/24 scope global eth0

/ # ping 10.244.6.142
PING 10.244.6.142 (10.244.6.142): 56 data bytes
--- 10.244.6.142 ping statistics ---
3 packets transmitted, 0 packets received, 100% packet loss

Container9 (Tenant3) towards Container8 (Tenant2) on Server7:
root@server07:/home/cumulus# docker exec -it ee9e54605d60 sh
/ # ip a
3: eth0@if47: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1450
    inet 10.244.6.144/24 scope global eth0

/ # ping 10.244.6.143
PING 10.244.6.143 (10.244.6.143): 56 data bytes
--- 10.244.6.143 ping statistics ---
3 packets transmitted, 0 packets received, 100% packet loss
```

Listing 8: Container overview

7 Discussion

From the first experiment it was shown that EVPN could be integrated into Kubernetes with the use of CRoHDAd. CRoHDAd had been adjusted, such that it listens to events and could add or delete the corresponding IP address of the container to a tenant VRF. CRoHDAd could do so, by injecting or removing the route in FRR. FRR was used in order to allow for EVPN on the host. However, the adjusted CRoHDAd needs more development effort, since it currently is not stable enough to be used in a production environment.

The second experiment focused on inter-host connectivity and hard multi-tenancy. From this experiment it was shown that inter-host connectivity and hard multi-tenancy could be offered without the use of network policies. This experiment continued on the first experiment, but used a different VRF table for each tenant. However, the use of VRF only separates L3 traffic. Therefore, a different virtual bridge was used for each tenant that separates the L2 traffic of each tenant in order to allow for hard multi-tenancy. Furthermore, one could also directly connect the container to the host without the use of a virtual bridge for each tenant. Nevertheless, when containers are tightly coupled, a direct L2 connection might be preferred.

Moreover, it is unclear how many VRFs will affect the host, since it is unclear how many VRFs a host can handle. Therefore, EVPN with VRFs and virtual bridges to provide tenant isolation in Kubernetes should be thoroughly tested in a larger environment.

In addition, EVPN was only tested in a Kubernetes orchestrated containerized environment. Therefore, the results may vary in different orchestrated containerized environment, such as Docker Swarm. For example it might be that there is a use case for address space isolation in Docker Swarm.

Furthermore, the results from the conducted experiments are based on the design choices made in the created architecture, as mentioned in section 5. Thus, different design choices might lead to different architectures that also provides traffic isolation without the use for network policies in EVPN.

8 Conclusion

In this research, the inter-host connectivity and multi-tenancy of EVPN without the need for network policies was investigated using a virtual test environment. The focus of this research was based on network separation between tenants on the same node, such that smaller tenants could share a node with different tenants to provide higher resource utilization and thus reduce costs. In order to evaluate EVPN as a multi-tenancy solution, we investigated the following research question:

"How can EVPN provide for multi-tenancy in a Kubernetes containerized environment?"

To answer this main question, we first had to answer the first subquestion "What are the advantages of EVPN as a solution to implement multi-tenancy in a Kubernetes orchestrated container environment?". From this subquestion it was found that the most important advantage of EVPN is that it does not use network policies to provide for hard multi-tenancy, instead we made use of VRF and a separate virtual bridge per tenant. The advantage of not having to use network policies is that filtering of network policies has an impact on the network performance and the scalability in a large orchestrated data center environment. However, whether EVPN has a better network performance than Cilium or Calico has not been tested. Another advantage of EVPN is that tenants could use the same IP addresses. However, Kubernetes is not reusing IP addresses, since it currently wants every container to have an unique IP address. Other advantages of EVPN are dynamic provisioning of container IP addresses to tenant VRFs, the possibility to allow for MAC mobility, aggregating routes in order to keep small forwarding tables, and ECMP routing and multihoming to efficiently use networking resources.

After answering the first subquestion, the second subquestion "How can the recent improvements in FRR allow for EVPN to be integrated into the Kubernetes container orchestration platform?" had to be answered. During our research it was found that this could be done with the use of CRoHDAd. CRoHDAd was adjusted in order to listen to the event of the creation or deletion of a container and the corresponding IP address. These events occur when the Kubernetes Master deploys or deletes containers through the Kubelet agent, which makes use of Docker to create or delete the container. When CRoHDAd notices such an event, the container IP address could then be added or removed from the VRF table of a tenant through FRR. This was possible, since EVPN could run directly on the host with FRR.

To answer the main research question on how EVPN can provide for hard multi-tenancy in a Kubernetes containerized environment, two experiments were conducted. The first experiment was used to test the process of adding or removing the container IP to the VRF table, and the second experiment was used to test whether traffic was completely separated from other tenants. These experiments were based on the created architecture and showed that it was possible to provide for inter-host connectivity and hard multi-tenancy with EVPN through the use of a separate VRF table and virtual bridge per tenant.

9 Future work

In this research, we did not test the performance of the created architecture in comparison with the current CNIs that provide multi-tenancy such as Calico or Cilium. Like Cilium, our architecture made use of VXLAN as NVO. VXLAN creates an overhead of 50-54 bytes to the frame, which can have an impact on the network performance if the underlaying hardware does not support VXLAN offloading. When the underlaying hardware does not support VXLAN offloading, the MTU size needs to be considered. Otherwise fragmentation can occur, that puts extra load on the CPU. However, in contrast with Calico and Cilium, EVPN routing decisions can be made without having to check network policies, which might lead to a performance increase. In order to broaden this research, a performance comparison of EVPN with different multi-tenancy CNIs that also provide multi-tenancy could be conducted.

As mentioned earlier, Kubernetes does not reuse addresses. If there is a use-case for address reuse (e.g. when multiple Kubernetes Masters are used or when Kubernetes recognizes and supports VRFs), the use of VRFs to provide address space isolation could be thoroughly tested in a PoC.

In addition, CRoHDAd was used to inject the host routes to the tenant VRF table in order to advertise the routes of the tenant VRF across the nodes. CRoHDAd is currently not available as a fully automated EVPN CNI plugin. Therefore, CRoHDAd could be further developed to be added as a fully automated EVPN CNI plugin, such that EVPN can directly be used as a CNI in Kubernetes without having to intervene.

10 Acknowledgements

We would like to show our gratitude to Attilla de Groot, system engineer at Cumulus Networks, for providing us the opportunity to explore EVPN and Kubernetes, but also for the virtual test environment to conduct our PoC and the support during the writing of this paper.

References

[1] T. Nadeau, J. Drake, B. Schlisser, Y. Rekhter, R. Shekhar, N. Bitar, A. Isaac, J. Uttaro, and W. Hendrix, "A Control Plane for Network Virtualized Overlays," tech. rep., IETF, 2012. https://tools.ietf.org/html/draft-drake-nvo3-evpn-control-plane-00.

[2] T. Narten, Ed., E. Gray, Ed., D. Black, L. Fang, L. Kreeger and M. Napierala, "Problem Statement: Overlays for Network Virtualization." http://www.hjp.at/doc/rfc/rfc7364.html, 2014. Accessed on August 23rd 2019.

[3] Google, "Cluster multi-tenancy." https://cloud.google.com/kubernetes-engine/docs/concepts/multitenancy-overview, 2019. Accessed on September 9th 2019.

[4] K. Goltsman, "Advanced Network Rules Configuration in Kubernetes with Cilium." https://supergiant.io/blog/advanced-network-rules-configuration-in-kubernetes-with-cilium/, 2019. Accessed on September 4th 2019.

[5] S. Talvinder, J. Varun, and B. G. Satish, "VXLAN and EVPN for data center network transformation," in *2017 8th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, IEEE, 2017.

[6] E. F. Naranjo and G. D. Salazar Ch, "Underlay and overlay networks: The approach to solve addressing and segmentation problems in the new networking era: VXLAN encapsulation with Cisco and open source networks," in *2017 IEEE Second Ecuador Technical Chapters Meeting (ETCM)*, pp. 1–6, Oct 2017.

[7] K. A. Noghani, A. Kassler, and P. S. Gopannan, "EVPN/SDN Assisted Live VM Migration between Geo-Distributed Data Centers," in *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*, pp. 105–113, June 2018.

[8] Y. Park, H. Yang, and Y. Kim, "Performance analysis of cni (container networking interface) based container network," in *2018 International Conference on Information and Communication Technology Convergence (ICTC)*, pp. 248–250, Oct 2018.

[9] A. Sajassi, J. Drake, N. Bitar, R. Shekhar, J. Uttaro, and W. Henderickx, "A Network Virtualization Overlay Solution Using Ethernet VPN (EVPN)," tech. rep., IETF, 2018. https://tools.ietf.org/html/rfc8365.

[10] A. Sajassi, R. Aggarwal, N. Bitar, A. Isaac, J. Uttaro, J. Drake and W. Henderickx, "BGP MPLS-Based Ethernet VPN." https://tools.ietf.org/html/rfc7432#section-7, 2015. Accessed on October 4th 2019.

[11] E. J. Rabadan, W. Henderickx, J. Drake, W. Lin, and A. Sajassi, "IP Prefix Advertisement in EVPN," tech. rep., IETF, 2018. https://tools.ietf.org/html/draft-ietf-bess-evpn-prefix-advertisement-11.

[12] L. Makowski and P. Grosso, "Evaluation of virtualization and traffic filtering methods for container networks," *Future Generation Computer Systems*, vol. 93, pp. 345 – 357, 2019.

[13] D. Bernstein, "Containers and cloud: From lxc to docker to kubernetes," *IEEE Cloud Computing*, vol. 1, no. 3, pp. 81,84, 2014.

[14] N. van Noort, "Secure data sharing in container networks." http://www.scriptiesonline.uba.uva.nl/document/673670, 2019.

[15] A. Gerrard, "What Is Kubernetes? An Introduction to the Wildly Popular Container Orchestration Platform." https://blog.newrelic.com/engineering/what-is-kubernetes/, 2019. Accessed on September 2nd 2019.

[16] Kubernetes, "Cluster Networking." https://kubernetes.io/docs/concepts/cluster-administration/networking/, 2019. Accessed on September 6th 2019.

[17] M. Laurent, "Kubernetes networks solutions comparison." https://www.objectif-libre.com/en/blog/2018/07/05/k8s-network-solutions-comparison/, 2018. Accessed on September 10th 2019.

[18] L. Briggs, "Kubernetes Networking with Calico." https://www.tigera.io/blog/kubernetes-networking-with-calico/, 2019. Accessed on September 5th 2019.

[19] Cisco Ltd, "Multi-tenancy." https://www.cisco.com/c/en/us/td/docs/switches/datacenter/pf/configuration/guide/b-pf-configuration/Multi-Tenancy.pdf, n.d. Accessed on September 25th 2019.

[20] M. Mahalingam, D. Dutt, K. Duda, P. Agarwal, L. Kreeger, T. Sridhar, M. Bursell, and C. Wright, "Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks." https://tools.ietf.org/html/rfc7348, 2014. Accessed on September 10th 2019.

[21] D. G. Dutt, *EVPN in the Data Center*. O'Reilly Media, Inc., 2018.

[22] S. Plug and L. Engels, "Using evpn to minimize arp traffic in an ixp environment." https://www.os3.nl/_media/2013-2014/courses/rp2/p31_report.pdf, 2014.

[23] A. Sajassi, R. Aggarwal, N. Bitar, A. Isaac, J. Uttaro, J. Drake and W. Henderickx, "BGP MPLS-Based Ethernet VPN." https://tools.ietf.org/html/rfc7432#section-11, 2015. Accessed on October 4th 2019.

[24] A. Sajassi, R. Aggarwal, J. Uttaro, N. Bitar, W. Henderickx and A. Isaac, "Requirements for Ethernet VPN (EVPN)," tech. rep., IETF, 2014. https://tools.ietf.org/html/rfc7209.

[25] A. Isaac, "Building blocks in evpn for multi-service fabrics." https://pc.nanog.org/static/published/meetings/NANOG75/1903/20190219_Isaac_Building_Blocks_In_v1.pdf, 2019.

[26] F. J. M. Potter, "Adjusted Cumulus Routing on the Host Docker Advertisement daemon for EVPN." https://github.com/F-Potter/CRoHDAd_EVPN, 2019.

[27] F. J. M. Potter, "Experiment 1: Connectivity and traffic flow." https://github.com/F-Potter/CRoHDAd_EVPN/blob/master/experiments/experiment1%20-%20inter-host%20connectivity%20and%20traffic%20flow.md, 2019.

[28] F. J. M. Potter, "Experiment 2: Multi-tenancy and traffic isolation." https://github.com/F-Potter/CRoHDAd_EVPN/blob/master/experiments/experiment2%20-%20multi-tenancy%20and%20traffic%20isolation.md, 2019.