Implementing a Data Center Network

André Rosa
DI/FCT/NOVA University of Lisbon, Lisbon, Portugal
Student № 48043
af.rosa@campus.fct.unl

***Abstract*—The progress of Cloud Computing motivates the construction of increasingly larger data centers, which nowadays host hundreds of thousands of tenants, each with several virtual machines (VMs). To meet their needs, each tenant may leverage multiple VMs, that may communicate with each other. As such, data centers must be able to provide the illusion that a tenant's VMs are connected to the same private virtual Local Area Network (VLAN). Nonetheless, conceiving such networks can be rather challenging due to their strict requirements, whereas the responsibility of successfully ensuring them relies upon the data center's network architecture. However, the conventional data center network architectures are incapable to meet those requirements, and, as a result, new solutions started to emerge to address their limitations. In this regard, we report in this document some network technologies that can be leveraged upon to devise a concrete data center network, capable of ensuring the VLANs' requirements.**

***Index Terms*—VL2, ECMP, Open vSwitch, VXLAN, data center network**

I. Introduction

The growth of the Cloud Computing paradigm has led to the construction of ever-increasing larger data centers, currently comprising hundreds of thousands of servers interconnected through the underlying data center network (DCN). These servers host multiple virtual machines (VMs), which are leased to several customers - *tenants*. In order to quickly and automatically provide flexibility to their tenants, data centers must be *agile*, i.e., they must be able to, on-demand, dynamically allocate these VMs on any available server, regardless of their location within the data center. Additionally, VMs can be migrated from one server to another at any time, which should be done without disrupting the services those machines host.

To implement their services, each tenant might leverage several VMs, which usually need to communicate with each other. Hence, data centers must be able to provide the illusion that a tenant's VMs are connected to the same private Local Area Network (LAN) - a virtual LAN (VLAN). However, creating such networks can be quite demanding, being necessary to address several challenges. On the one hand, these VLANs must be *Topology Agnostic*, i.e., the VMs of a given VLAN should have the liberty to be spread across any server within the data center, without any constraints, and thus allowing the data center to be agile. On the other hand, it is required that the traffic of a given VLAN does not interfere with the traffic of other VLANs, as is the case with real physical networks - *Performance Isolation*. Furthermore, the data center's underlying network must not be a bottleneck in the maximum available bandwidth between any pair of servers, which should only be limited by their network interface cards - *Uniform High Capacity*. Finally, these VLANs must provide *Layer-2 Semantics*, i.e., they must provide the illusion to its VMs that they are connected to the same Ethernet switch. Therefore, any VM must be able to have assigned any IP address, which must be maintained in case the VM is migrated. Furthermore, Layer-2 network features, such as network-wide broadcast, should be available. The task of successfully ensuring these requirements relies upon the data center's network architecture.

Unfortunately, the conventional data center network architectures are unable to meet the previous requirements. These architectures have a tree-like topology, with high levels of oversubscription among the different levels, and with servers and their VMs being assigned IP addresses with topological semantics. Moreover, these architectures also resort to conventional Ethernet VLANs[1] between their servers to limit overheads. Consequently, this hinders the agility of the data center in allocating and migrating VMs, which fragments the server pool. Besides, these architectures are also unable to provide performance isolation as well as enough bandwidth between the servers, worsening the server pool's fragmentation. An implication of this fragmentation is the formation of hot-spots, network regions with high congestion and server usage, while there are available servers in other regions of the network. On top of that, the conventional architectures yield almost no redundant paths, further jeopardizing the data center's performance in case of network failures.

As a result, new solutions started to emerge to address the limitations of conventional data center network architectures. Taking this into account, we report in this document a set of network solutions that can be used to devise a concrete data center network capable of ensuring the VLANs' requirements.

The remainder of this document is structured as follows: Section II presents an overview of the VL2 data center network, highlighting its main features; Section III aggregates a summary of some network technologies, tools, and protocols, relevant in building data center networks; Section IV discusses how the previous solutions can be leveraged upon to implement a concrete data center network; and finally, Section V finishes this report with some final remarks.

This document was written in the context of the APRC course in replacement of the second midterm test.

[1] Not to be confused with VLANs of VMs.

II. The VL2 Data Center Network Architecture

VL2 [1], [2] is a scalable data center network architecture, designed to leverage conventional commodity hardware. As such, it avoids modifying any switch/router software and only requires a Layer-2.5 "shim" in the servers' network stack - the *VL2 agent*. VL2's design was influenced by traffic and fault measurements, taken from a large cloud service provider. These collected traffic measurements revealed that the majority of the traffic was internal to the data center, with highly volatile traffic patterns, few distinct flows per server, and very small flows, i.e., small duration and few kB. Furthermore, the results also revealed that data center networks are characterized by frequent failures. Taking these traffic characteristics into account as well as the limitations of conventional architectures, VL2 decided to scale out the network switches rather than scaling up each switch, to achieve better combined performance. In fact, VL2 operates akin to a very large Virtual Layer-2 switch. Next, we highlight and discuss the major key features of VL2's architecture.

A. Network Topology

VL2 resorts to a network of commodity switches with *folded Clos topology*. In this, the switches are divided into two disjoint groups, intermediate and aggregation switches, whose links form a complete bipartite graph, i.e., any intermediate switch is connected to all aggregation switches and vice-versa. The servers, which are grouped into racks, are connected to a switch - Top of Rack (ToR) switch. These ToR switches are then connected to two aggregation switches. As a result, this provides a vast amount of redundant routes between any servers. Due to this, the failure of an intermediate switch only reduces the total bandwidth by $1/n$, being n the number of intermediate switches, - *graceful degradation*. Another consequence of this topology is that it is highly scalable, with no oversubscription, being only necessary to add a few more switches.

B. Packet Forwarding

Network switches operate at the Layer-2 in the network model, which unfortunately presents several scalability difficulties, such as not taking advantage of multiple paths and slow convergence times due to the Spanning Tree Protocol (STP) or broadcast storms due to Address Resolution Protocol (ARP) requests. For this reason, in VL2 the network switches operate as IP Routers (Layer-3), leveraging the well known OSPF routing protocol [3]. However, in VL2, OSPF is only used to maintain the data center's network topology without disseminating any of the end host's (the VMs) addresses, which prevents the switches from repeatedly learning massive amounts of highly volatile information.

C. Addressing Scheme

To enable agility and overcome the server pool's fragmentation, the IP addresses of the VMs should not be bound to any switch or server. Therefore, the VMs addresses should remain unaltered despite its host server's location or server migrations. As such, VL2 dissociates the VM's addresses, *application-specific addresses (AAs)*, from their location in the network, *location specific addresses (LAs)*. Consequently, the AA's loose their topological semantics, functioning only as names. This, however, leads to the need of a mechanism to discover the current location of an AA, i.e., discover the LA currently associated with an AA. To address this, VL2 resorts to a directory system that stores the mappings from AAs to LAs. The LA associated with an AA can belong to a ToR switch, capable of decapsulating the packets, or a virtual switch running within a hypervisor. This directory system has a 2-tier architecture: a Replicated State Machine (RSM) and several caches. The RSM, materialized by a small number of servers, handles the reliable storage of mappings, providing strong consistency for the entries. On the other hand, the caches only provide available consistency to enable high throughput and fast look-ups. In effect, the directory system acts as a replacement of ARP, overcoming its scalability limitations while maintaining Layer-2 semantics. Additionally, the directory system can enforce access control policies between pairs of AAs, not answering or giving an error to look-ups that are not authorized.

In this regard, when a VM sends a packet to another VM on the same VLAN, the VL2 agent running on the sender's hypervisor intercepts that packet. Then, in case the VL2 agent does not know the LA of the destination, it invokes a lookup on the directory system, and finally tunnels the packet to that LA, i.e., it encapsulates the original packet in a new packet having as destination the LA. Upon a packet arriving at the outer LA destination, this destination decapsulates the packet and delivers or forwards it to the inner AA destination.

D. Traffic Engineering

Due to the highly volatile traffic patterns, traffic engineering optimization techniques are not suitable. As such, VL2 leverages instead on Valiant Load Balancing (VLB), which consists in randomly spreading the traffic across intermediate nodes. One of the benefits of VLB is that it guarantees a non-interfering (packet switched) network provided that: i) traffic is uniformly spread through the intermediate nodes and that ii) traffic patterns do not violate capacity constraints of the network links. The former, i), is achieved leveraging Equal-Cost Multi-Path (ECMP) routing, discussed in §III-A. Unfortunately, switches have a limited amount of multiple paths they can store per destination[2], which may be smaller than the amount of available paths in VL2. As such, VL2 divides the intermediate switches into groups, with a size of at most the number of supported paths, and assigns to these groups IP anycast addresses, being that any switch in the same group has the same address. Then, the packets are tunneled into one of these groups, resorting to IP-over-IP encapsulation, being that the specific intermediate switch of the group is chosen by ECMP. The group to which the packet is forwarded is randomly chosen although it has to

[2]Usually 4 or 8 different paths.

be the same for packets belonging to the same flow to avoid packet reorderings. Furthermore, some switches are not able to consult transport protocol's ports, which are used by ECMP to distinguish flows, when IP-over-IP encapsulation is used. For this reason, the source IP address of the outer IP packet is set as the hash of the inner IP address and inner port. These combinations ensures the distribution of the flows throughout all the available paths in the network. As such, a packet is, in fact, encapsulated two times by the VL2 agent. First, as mentioned before, the original packet destined to an AA is encapsulated into a new packet destined to the LA associated with that AA. Then, that new packet is further encapsulated, having as new destination an intermediate switch. Next, the packet is forwarded to that intermediate switch, which decapsulates the packet and forwards it to the LA associated with the destination. This destination then decapsulates the packet again, and finally forwards/delivers it to the destination VM. The latter, ii), is achieved through TCP's end-to-end congestion control mechanism, which strives to fairly divide the available bandwidth through all the active connections. Although ECMP and TCP do not fully ensure the previous two traffic assumptions, i) and ii), they are close to the optimum in practice. Furthermore, the *folded Clos topology* is remarkably adequate for VLB, providing bandwidth guarantees as long as the traffic obeys the hose model [4]. In this way, VL2 is able to ensure uniform high capacity and performance isolation.

E. Experimental Evaluation

In [1], it was performed an experimental evaluation on VL2's performance on a network composed of 80 servers and 10 commodity switches.

Regarding network failures, VL2 displayed *graceful degradation* with fast reconvergence.

Concerning the network's capacity, VL2 achieved 94% optimal network capacity, which corresponds to the fraction of total goodput[3] over the total interfaces' capacities, being the deviation from perfect performance mostly due to encapsulation headers, with some small contributions from TCP's control dynamics and retransmissions. Furthermore, VL2 also achieved 0.995 TCP fairness index, which corresponds to the ratio of allocated capacity each TCP flow has, being that a fairness ratio of 1 means that all flows obtain the same capacity. As result, 94% efficiency together with 0.995 fairness index confirms that VL2 is able to achieve Uniform High Capacity throughout all servers of the data center.

In respect to performance isolation, VL2 relies upon TCP's congestion control to ensure that each flow is limited to its fair share of the available bandwidth, making sure that each flow conforms to the hose model. As such, TCP must react rapidly enough to adjust the throughput. The results revealed that, indeed, TCP is able to enforce the hose model on flows greater than a few RTTs[4].

[3]Amount of useful information delivered by the network to a certain application per unit of time, excluding protocol overhead bits as well as retransmitted data packets.

[4]1-10ms in data centers

Finally, the directory system was shown to be robust to component failures, to exhibit incremental scalability, to provide high throughput and fast response time for look-ups, and to have update rates higher than the expected churn rate.

III. Relevant Network Technologies

In this section, it is presented a set of relevant tools, techniques, and protocols that may present valuable when designing a data center network.

A. Equal-Cost Multi-Path (ECMP) Routing

ECMP [5] is a family of algorithms for determining the next-hop while routing packets when several equal-cost paths are available to the destination. Therefore, this technique is able to provide significant increases in bandwidth by load-balancing the traffic across several paths. An important metric of ECMP algorithms is their *disruption*, defined as the ratio of flows that alter their routes. In this regard, some algorithms, such as *round-robin* (i.e., chose the least recently used neighbor as the next-hop), are disruptive regardless of any change. As such, these algorithms are not well suited in the presence of traffic of protocols that ensure ordered delivery, such as TCP. For this reason, we will only focus on ECMP algorithms which are disruptive only when there are changes in the set of equal-cost paths for a given destination. There are three main such ECMP algorithms: *Hash-Threshold*, *Modulo-N*, and *Highest Random Weight (HRW)*.

In *Hash-Threshold*, for each packet, a key is computed as the hash of the fields that identify each flow. Furthermore, the N next-hops are assigned equal-sized regions, being this size equal to the ratio between the size of the key space and N. Then, the next-hop is determined as the result of dividing the key by the size of each region. This computation only requires the hash computation and 2 divisions, being extremely fast ($O(1)$ time complexity). The disruption is bounded within the interval $]1/4; 1/2]$, being the lowest value achieved when new regions are inserted in the center of the key space, i.e., new next-hops are inserted in the middle of the next-hops set.

Modulo-N, in turn, is often considered as a "simpler" version of *Hash-Threshold* (with similar performance), with the next-hop computed as $key \bmod N$. This is the most disruptive of the algorithms, with $(N-1)/N$ disruption.

HRW is also similar to *Hash-Threshold* yet with non-fixed size regions. In this algorithm, the next-hop is the one with the highest weight. This weight is computed by seeding a pseudo-random number generator with the packet fields that identify each flow and the next-hop. This algorithm has minimal disruption, $1/N$, however, the computation is much more expensive, with $O(N)$ time complexity.

A comparison between the three algorithms can be found in Table I. Overall, since *Modulo-N* is more disruptive and has similar performance to *Hash-Threshold*, the latter is better. In case state can be stored for each flow, the results of *HRW* can be cached, transforming the $O(N)$ complexity into $O(1)$, and thus becoming the better choice. However, if no state can be kept per flow, then *Hash-Threshold* is the best.

Algorithm	Disruption	Time Complexity
Hash-Threshold	$]1/4; 1/2]$	$O(1)$
Modulo-N	$(N-1)/N$	$O(1)$
Highest Random Weight (HRW)	$1/N$	$O(N)$

TABLE I
COMPARISON BETWEEN THE MAIN ECMP ALGORITHMS.

B. Open vSwitch (OVS)

OVS [6] is an open-source virtual switch, designed for bridging traffic on hypervisor platforms between the hosted VMs and the physical network. Its architecture consists of two main components: *ovs daemon* and *datapath module*.

The *ovs daemon* is a user-space module that implements a complete virtual multi-layer SDN[5] switch, managing all the forwarding decisions of incoming flows. These decisions are encoded by *datapaths*, materialized by a flow table containing mappings from packet header fields and metadata to sets of *actions*. These actions are mostly forwarding the packet to a specific *vport* (virtual port abstracting a physical or VM ports/interfaces or a tunnel), perform packet modifications, or simply drop the packet. One way to define these *datapaths* and *actions* is through OpenFlow [7], an SDN standard.

The *datapath module* is a kernel-space module that handles the reception of packets through *vports* and performs the specified *actions* for that packet. For this, the *datapath module* leverages two caches, *megaflow cache* and *microflow cache*, to verify, in the kernel-space, if it knows how to handle the received packet. If there is a match in these caches, then the *datapath module* simply performs the associated *actions*. However, if no match is found, the packet is delivered to the *ovs daemon*, which determines the *actions* to perform and dispatch the packet, along with the *actions*, back to the *datapath module*. In this latter case, the *ovs daemon* can also instruct the *datapath module* to cache the decisions to be used on future packets.

The *microflow cache* is a first-level cache implemented with a simple hash table, where each entry exactly matches with all the packet header fields and maps to an entry in the *megaflow cache*. It is extremely fast, however, it can incur in several misses due to it having so fine-grained entries, matching at most a single connection. To address this, the *megaflow cache*, a second-level cache that supports generic matching, being therefore slower, is consulted to attempt to find a match. If such a match is found, then a new entry is inserted into the *microflow cache*, otherwise, the packet is delivered to the *ovs daemon*. Upon receiving an answer from the daemon, a new *megaflow cache* entry is inserted as well as a *microflow cache* entry for that specific packet.

C. Virtual eXtensible Local Area Network (VXLAN)

VXLAN [8] is a protocol for overlaying virtualized Layer-2 networks over Layer-3 networks, i.e., to build *tunnels*. Each VLAN is designated as a *VXLAN segment*, which is identified by a 42-bit number, called *VXLAN Network Identifier (VNI)*, and has associated an IP multicast address. Tunnel endpoints, called *VXLAN Tunnel End Point (VTEP)*, can be located either within hypervisors or a physical switch that supports VXLAN. When a VM delivers an Ethernet frame to the hypervisor, the *VTEP* verifies if the destination MAC address belongs to the same *VXLAN segment*. Additionally, the *VTEP* also verifies if it knows the IP address of the remote *VTEP* associated to the destination MAC address, in the case this MAC address is a unicast address; or if it knows the IP multicast address associated with the *VXLAN segment*, in case the destination MAC is the broadcast address[6]. If both verifications pass, then it is appended a VXLAN header to the frame, containing the VNI of the VXLAN segment the VM belongs to, and the frame is further encapsulated within a UDP packet. This packet has as destination either the remote *VTEP*'s address or the IP multicast address. Then this packet is forwarded towards its destination, either by IP unicast or IP multicast, depending on the destination address. Upon arrival at a remote *VTEP*, if the packet is valid, it is decapsulated and delivered to the destination VM. Moreover, the remote *VTEP* registers the mapping from the inner source MAC address to the source *VTEP* IP address. Since VMs broadcast ARP frames when they do not know the MAC address associated with the destination IP address to which they pretend to send a packet to, this enables *VTEPs* to discover all the associations between MAC addresses and *VTEP* IP addresses need to communicate. As such, it is only needed to exist a way for *VTEPs* to discover the IP multicast address associated with their *VXLAN segment*. One way to achieve this is through a directory system, similar to VL2. Likewise, the mappings from MAC address to *VTEP* IP addresses could also be delivered through such a system instead of leveraging source learning, despite source learning allowing reducing the amount of information that this system has to store.

IV. SOLUTION

This section contains a detailed description of how the previously presented technologies (§II and §III) could be leveraged to implement a real data center network.

A. Network Requirements

The required solution is expected to be capable of handling hundreds of thousands of tenants, each with possibly more than one VM to which can be assigned any IP prefix (public or private). A tenant's set of VMs can be scattered through the data center, running in several physical servers, and these VMs may migrate to a different server at any time. Each VM views the set of the other VMs of the same tenant as being directly connected in a private LAN, and can, therefore, leverage conventional network protocols (e.g. ARP, DHCP).

The hypervisor, that runs in each server, has an OVS for each tenant that has a VM being hosted (in that hypervisor).

[5]Software Defined Network

[6]ff:ff:ff:ff:ff:ff

Each one of these OVSes has a (virtual) uplink that connects it to the server's physical Network Interface Card (NIC), with the IP address of the OVS belonging to the server's IP prefix.

Every rack has 30 servers directly connected to a ToR switch, all belonging to the same IP subnet, i.e., sharing the same IP prefix. All the ToR switches are further connected to a physical network, with a *folded Clos topology*, like the one in VL2 (§II-A). This network is composed of several switches that support VLANs, STP, and IP routing based on OSPF with ECMP.

Furthermore, it is available a directory system maintaining the mappings from each tenant to the set of IP addresses of the OVSes it owns

Communications to the exterior of the data center are beyond the scope of this report.

B. IP Routing

Like in VL2, all the network switches operate as IP Routers (Layer-3), leveraging the well known OSPF routing protocol. For purposes of exposure, IPv4 is used, however, IPv6 can also be used similarly, if desired. Since data center switches do not directly forward traffic to the Internet, they can have any public or private address. To forward traffic from the VMs to the Internet, tunnels to gateways can be established.

The prefix 10.0.0.0/24 is reserved to obtain addresses of data center management servers, such as those that implement the directory service. The *intermediate switches* receive addresses in the range of prefixes 10.0.1.0/24 - 10.0.100.0/24 (25 500 addresses). The *aggregation switches* receive addresses in the range 10.0.101.0/24 - 10.0.255.0/24 (39 525 addresses). The *ToR switches* receive prefixes of the form X.Y.0.0/16, where X is between 1 and 255 (excluding 10) and Y is between 0 and 255, and has the address X.Y.0.1. As such, there can be a total of 64 770 *ToR switches* and hence a maximum of 1 943 100 total servers. Each server is assigned a 24 length prefix from the prefix of the *ToR switch* it is connected to, and the first address of that prefix is assigned to the server. Furthermore, the OVSes running on a server receive an address from that prefix as well. For instance, a *ToR switch* has the prefix 20.12.0.0/16, one of its servers has the prefix 20.12.1.0/24 and the IP address 20.12.1.1/32, and supports 254 different OVSes with addresses from 20.12.1.2/32 to 20.12.1.254/32.

To reduce the size of the routing tables of *ToR* and *aggregation switches*, these tables could be manipulated to contain only entries for downlinks[7] and a "generic small prefix" that matched with every uplink, e.g. the default route 0.0.0.0/0, having as next-hop every uplink (they have all the same cost). Thus, every packet not destined to a downlink would match with that entry and thus be correctly forwarded.

C. Load Balancing

Since IP-over-IP decapsulation is not a feature supported by the *intermediate switches*, the usage of ECMP can lead to only a small subset of the multi-paths being used, seriously jeopardizing the data center's performance. For this reason, the links between *aggregation switches* and *intermediate switches* are equally and randomly distributed into several groups, with size equal to the maximum number of multi-paths supported by ECMP (usually 4 or 8). Each group is then assigned a conventional Ethernet VLAN, being the ports of the switches within each group configured as access ports of the correspondent VLAN. Therefore, those links only carry traffic of a single VLAN and consequently, each VLAN has as many equal-paths (between *aggregation switches* and *intermediate switches*) as supported by ECMP. Then, there would be an OSPF instance running per VLAN, with ECMP enabled, to take advantage of those multiple paths. Additionally, the *ToR switches* must be configured such that their ports are all trunk ports. Furthermore, it is also necessary each hypervisor tagging (and untagging on reception) the frames of its hosted VMs with a VLAN tag, through the OVS that handles that frame. The VLAN tag should be computed in such a way that all the frames of the same flow end up in the same VLAN. As such, an ECMP algorithm (see §III-A) could be used to compute this VLAN tag, instead of the next-hop to be chosen, in exactly the same way. As a consequence, ECMP is leveraged two times: one at the OVS, for computing the VLAN tag, and the other at the switches to forward traffic through multiple paths within each VLAN. Hash-Threshold would be enough, however, HRW could also be used.

D. Tenant VLAN Overlays

Every tenant has associated a number that identifies its private VLAN. This number is the VNI (from VXLAN - §III-C). As previously mentioned, every hypervisor has an OVS for each tenant that has a VM running on it. Each one of these OVSes use VXLAN to establish tunnels between the several VMs (of the same VLAN), to exchange traffic between them. Every one of these OVSes is accompanied by a module, similar to the VL2 agent, which we named *OVS operator (OO)*. This module is responsible for configuring the OVS, with datapaths and the corresponding actions, whenever changes are needed. For instance, when it is learned a new mapping from a VM's MAC address to the IP address of the OVS associated with that VM, the OVS is configured to redirected every frame destined to that MAC address through a tunnel terminating on that IP address. Another example could be to configure the OVS to attach a specific VLAN tag (§IV-C) to every outgoing frame that carries traffic of a given flow.

In this way, whenever a VM delivers a frame to the hypervisor, the latter delegates the frame to the OVS associated with that VM. Then, the OVS finds the matching datapath for that frame and performs the corresponding actions. Usually, this corresponds to simply forwarding the frame through a tunnel (using VXLAN) with a specific VLAN tag appended (in the outer frame). For this, it is required that the OO had already previously configured the OVS with such datapath, which is discussed in more detail in §IV-F.

[7]Downlinks are links to servers, in case of *ToR* switches, and links to *ToR* switches, in case of *aggregation switches*.

E. VM Provision and Migration

A tenant is free to assign any IP address, public or private, to its VMs. This can be done by statically assigning addresses to the VMs or it can be done dynamically leveraging DHCP. The advantage of using DHCP is that new VMs could automatically be provisioned and started without the need for manual configuration. This is rather useful to handle spikes in client requests, for instance. In the case DHCP is used, it is necessary that tenants setup a DHCP server within their VLAN to manage the allocation of addresses according to their will. The infrastructure will not attempt in any way to assign addresses to VMs on the case such a server is not available.

When a new VM is provisioned or is migrated to another server, the hypervisor chosen to host that VM creates a new OVS and its OO companion, to manage the traffic in and out of that VM, unless such OVS already exists. This OVS is then assigned an IP address by the hypervisor, and the directory system is updated (by the OO) to include this address in the set of OVS IP addresses associated with the tenant. Furthermore, in case of a VM migration, the IP address of the old OVS associated with that VM should be removed from the directory system, and the new OO of that VM should send a notification to every other OO of the same tenant, to update their mappings from the VM's MAC address to the new OVS' IP address.

F. Address Resolution

As previously mentioned, a directory system is available, which maintains the mappings from each tenant to the set of IP addresses of the OVSes associated with those VMs. With this directory system, whenever a VM sends a frame destined to the broadcast address[8], the OVS multicasts the frame through every tunnel to other VMs of the same tenant. For this, it is required that the OO had configured its OVS with the IP addresses of every other OVS, in order to establish the tunnels. Thus, if that information is not already available to the OVS, the OO performs a look-up on the directory system to obtain the addresses. Therefore, ARP requests and DHCP discovery and requests work very similar to a real network, being these frames broadcast within the VLAN. Through these broadcasts, the OO discovers the mappings from the VMs MAC addresses to the IP addresses of the OVS connected to them (similar to VXLAN).

A simple directory system is enough to guarantee the proper functioning of the tenants' VLANS. However, the performance can be optimized using a more complex directory system. There is, therefore, a trade-off between a simple directory system, like the one presented, that stores few information per tenant yet incurs a lot of broadcasts within the tenants' VLANs (VXLAN-like), and a more complex directory system, that stores more information though it avoids such broadcasts (VL2-like). Thus, in addition to the mappings from a tenant to the set of the IP addresses of its OVSes, the directory system could also store, for each tenant, the mappings from a VM's MAC address to the IP address of the OVS to which that VM is connected to, and another mapping from a VM's IP address to its MAC address. Therefore, whenever a VM executes DHCP, a new registration in the directory system is issued by the OO associated with that VM. In case there is no DHCP, when a OO issues a look-up to the directory system for the first time, this registration can be performed. As a result, the broadcast of ARP requests can be replaced by unicast look-ups on the directory system and the broadcast of DHCP discoveries and requests can also be replaced by unicast/multicast tunneling to only VMs running a DHCP server, which would have to be registered and identified as a DHCP server in the directory system. DHCP could also be replaced directly by the directory system, which would assign IP addresses and immediately register the mappings.

[8] ff:ff:ff:ff:ff:ff

V. Conclusion

In this report, we addressed the challenges intrinsic in designing and implementing a real data center network. These networks must be able to provide the illusion that a tenant's VMs are connected to the same private virtual Local Area Network (VLAN). Nonetheless, conceiving such networks can be quite challenging due to their rigorous requirements, whereas the responsibility of successfully ensuring them relies upon the data center's network architecture. Thereby, we started by exposing a literature review on some technologies, tools, and protocols, that can be utilized to conceive a concrete data center network. Finally, we finished this report by providing a detailed description of how the reviewed technologies could be used to devise such network while ensuring their strict requirements.

References

[1] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "Vl2: A scalable and flexible data center network," *Commun. ACM*, vol. 54, no. 3, p. 95–104, Mar. 2011. [Online]. Available: https://doi.org/10.1145/1897852.1897877

[2] ——, "Vl2: A scalable and flexible data center network," *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 4, p. 51–62, Aug. 2009. [Online]. Available: https://doi.org/10.1145/1594977.1592576

[3] J. Moy, "OSPF Version 2," RFC 2328, Apr. 1998. [Online]. Available: https://rfc-editor.org/rfc/rfc2328.txt

[4] N. G. Duffield, P. Goyal, A. Greenberg, P. Mishra, K. K. Ramakrishnan, and J. E. van der Merive, "A flexible model for resource management in virtual private networks," *SIGCOMM Comput. Commun. Rev.*, vol. 29, no. 4, p. 95–108, Aug. 1999. [Online]. Available: https://doi.org/10.1145/316194.316209

[5] C. Hopps, "Analysis of an Equal-Cost Multi-Path Algorithm," RFC 2992, Nov. 2000. [Online]. Available: https://rfc-editor.org/rfc/rfc2992.txt

[6] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, and M. Casado, "The design and implementation of open vswitch," in *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. Oakland, CA: USENIX Association, May 2015, pp. 117–130. [Online]. Available: https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/pfaff

[7] "OpenFlow," http://www.opennetworking.org/sdn-resources/onf-specifications/openflow, accessed: 05-06-2020.

[8] 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," RFC 7348, Aug. 2014. [Online]. Available: https://rfc-editor.org/rfc/rfc7348.txt