Project title:

An analysis on network virtualization protocols and technologies

by

Vida Shahrokhkhani

Mentor: Kanwal Cheema

Contents

1. Introduction

“Virtualization means that Applications can use a resource without any concern for where it resides, what the technical interface is, how it has been implemented, which platform it uses, and how much of it is available.” [1]

There are many motivations why we need to use virtualization for each section of our network. Sharing is one of the reasons for using virtualization. It means we can break up resources, capacities, servers and even speed. Second motivation is isolation using Virtual Private Networks (VPN) which can be used to protect from other tenants. In addition, we can aggregate and combine many resources e.g. storage. Finally, network management can be easier due to easier distribution, deployment and testing.
Virtualization brings IT companies many advantages including:

- Minimize hardware costs: Since we can run multiple virtual servers on one physical hardware.
- Easily move VMs to other data centers: Virtualization provides disaster recovery and easier hardware maintenance.
- Consolidate workloads: It refers to higher availability of physical resources when the number of user request is high and increasing device utilization.
- Lowering the operational expense: This is the result of automation which simplifies provisioning or administration of hardware and software.
- Flexibility and scalability: We can have multiple operating systems on one server at the same time.

We can implement virtualization in multiple areas of computing including: storage (by implementing virtual memories, virtual disks or RAID and cloud storage; computing (by using virtual desktop e.g. Thin client, virtual servers e.g. VMs and even virtual data centers e.g. cloud.
The third area, which this project is focusing on, is networking. Network virtualization allows tenants to form an overlay network in a multi-tenant network such that each tenant can control:
Layer of connection: Tenant network can be layer 2 while the provider is layer 3 and vice versa.
Addresses: MAC addresses and IP addresses.
Network Partitions: VLANs and Subnets.
Node Location: Move nodes freely
Network virtualization can be done in each and every level of our network. These levels are:
Network Interface Card (NIC), layer 2 Links, layer 2 Bridges, layer 2 Networks, layer 3 Links, layer 3 Routers, layer 3 Networks and Data Centers.
There are different protocols which are used to implement virtualization in each level. The table 1 shows some protocols or techniques which are used to virtualize entities of the network. These different protocols are using different approaches to do their tasks and some of them has higher performance compared to others.

Table 1: Network virtualization protocols and techniques

Entity	Protocols/techniques
NIC	VMQD, SR-IOV, VPP
Switch	VEB, VEPA
L2 Link	VLANs (aggregated by virtual port channels using protocols such as LACP)
L2 Network using L2	VLAN (interconnecting by techniques like PB or Q-in-Q, PBB or Mac in Mac, PBB-TE, Access-EPL, EVPL,EVP-Tree, EVPLAN
L2 Networking using L3	NVO3, VXLAN, NVGRE, STT
Router	VDCs, VRF
L3 Networking using L3	MPLS, GRE, PW, IPsec, LISP
Application	ADCs

One of the challenges in modern data centers is transferring virtual machines from one data center to another which are located in different network segments and subnets. So if we consider virtualization as creation of multiple virtual networks sharing a physical common infrastructure, we should be able to manage the following features:

1. Support for multiple VM mobility; means, VMs should be able to cross layer 3 boundaries without requiring manual reconfiguration of networks.
2. Ability to manage overlapping IP addresses between multiple tenants.
3. Support for multiple path forwarding within virtual networks.

One solution is using protocols like OTV[1], which abstract the logical network from the physical network. This is done using tunneling techniques and by encapsulating traffic inside IP packets so that traffic can cross layer 3 boundaries. Several protocols use this technique, such as VXLAN, NVGRE, STT which are proposed IETF standards. There are some differences among these protocols but in all of these instances, they use 24 bit identifier which allows them to have over 16 million possible networks. In each case, the end points belongs to a virtual network, regardless of its location in the underlying physical network.
Since in all mentioned protocols, virtual networks are independent of underlying network, network applications cannot be guaranteed the TOS[2] and this can reduce the performance. In addition unlike server virtualization, they cannot reserve resources. In server virtualization, VMs can be guaranteed I/O, memory, CPU and other resources they need. But using these protocols, virtual networks cannot reserve resources. Finally, there is no ability to centralize functions such as configuration management and policy management.
We can also achieve network virtualization using SDN[3], which enables programmatically efficient network configuration by segregating control plane from forwarding plane. One of the ways to do this is by manipulating the tables in SDN switches. So using SDN switches we can have all the advantages of network virtualizations and we can have the ability for applications to request

[1] Overlay Transport Virtualization
[2] Type of Service
[3] Software-defined Networking

services and resources and centralize our network configurations. In addition, with SDN approach to network virtualization, IT organizations also will have the ability to do more granular traffic routing and more intelligence towards using network infrastructure.
This project is going to analyze the differences in operation and performance of the protocols and techniques (including SDN and non-SDN), which are using in network virtualization.

2. Non-SDN solutions for network virtualization (Network Overlays)

Network virtualization protocols should address some issues. Since data centers services multiple tenants, they need to manage the overlap that may exist between address spaces of tenants across virtual networks. So the data center should be able to allow per tenant addressing which is separate from other tenants and the data center infrastructure.
In addition, the protocol should provide the ability of multipathing in both layer 2 and layer 3. This is necessary when for example two or more routes are exiting the data center or virtual network.
The other issue that the protocol should resolve is scaling which is important in both forwarding tables and network segment. One of the basic tool which is used for segmentation of broadcast domains is VLAN. But VLANs are design to provide 4096 isolated network segments which is not enough today.
Finally, the protocol should not be related to or dependent upon the limitations of other previously designed protocols which are not scalable for today's data centers.
By considering these design considerations, this section introduces protocols which are used for network virtualization and discusses their advantages and limitations.

Overlay networks can be classified into two categories:

- Network-based overlay networks
- Host-based overlay networks

2.1 Network based Overlay technologies

Network based protocols have some properties which are common between all of them. These properties are as follows:

- These protocols have distributed control and management, means almost all of the core devices should be part of the control mechanism of the protocol
- In almost all of these protocols the edge devices are responsible to perform encapsulation
- End points are *physical* switches or routers

2.1.1 IEEE 802.1ad or Q-in-Q (provider Bridging)

Since the size of VLAN tag specified in IEEE 802.1q is 12 bits (equal to 4096 VLANs), it cannot completely identify and segregate a large number of users in data center networks or expanding metropolitan Ethernet networks. Q-in-Q solved this problem by encapsulating the VLAN tag of a private network in the VLAN tag of a public network. In Q-in-Q tunneling, each private network add their own VLAN, which is called customer VLAN or C-VLAN. In addition to customer VLAN, as frame travels data center network, another 802.1q tag will be added to existing C-VLAN

which is called Service provider VLAN or S-VLAN. When the packet leaves service provider network, the S-VLAN 802.1q VLAN is removed (Figure 1).

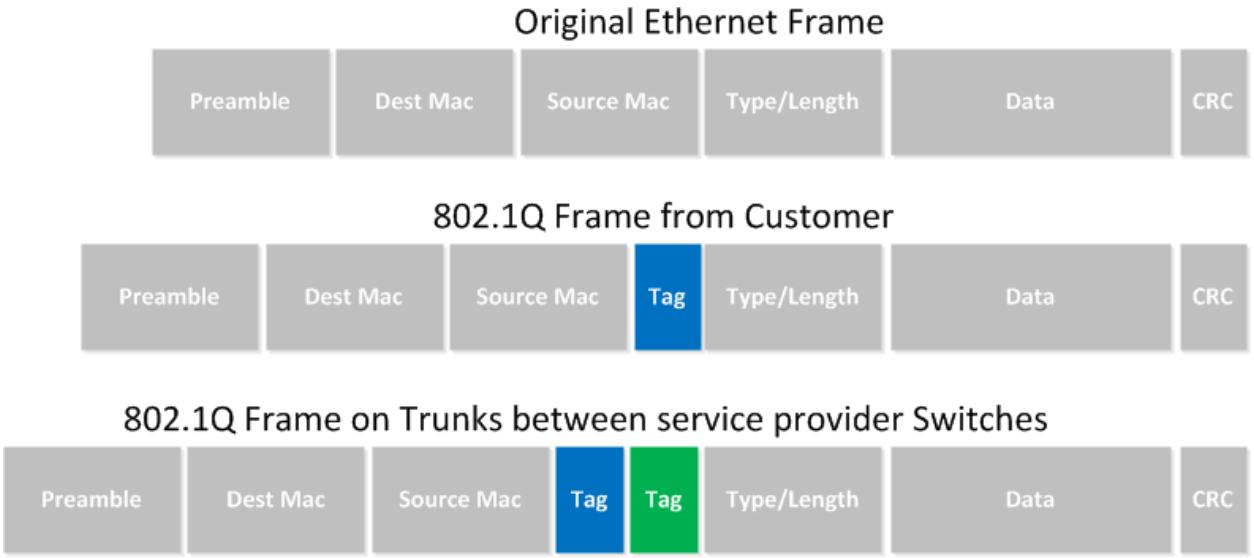


Figure 1 Q-in-Q frame format. (Image from: https://networklessons.com/switching/802-1q-tunneling-q-q-configuration-example/)

This protocol doesn't need any additional control protocols. The only protocols which might be useful is standard bridging protocols such as RSTP[1] and MST[2] to have a loop free network. In addition, customer devices are not aware of this function and the only devices which are aware of the process are the devices that performing encapsulation and de-encapsulation. So other devices in the core do not have the knowledge of C-TAG in frame [2].

Advantages of Q-in-Q:

- As mentioned, by using IEEE 802.1q standard, we can only have about 4k VLANs and so we can only have 4096 separated broadcast domains. This tunneling method provides 4k * 4k (nearly 16 M) VLAN IDs, which can meet the shortage of VLAN ID resources.
- Each customer can plan for its own VLAN and will not have conflict with other VLAN IDs from other customers and service provider.
- It can also provide simple VPN service.

Limitations of Q-in-Q:

- Using this method, MAC address of customers will not be hidden from service provider network, as only C-TAG is encapsulated inside S-TAG.
- All customer MAC addresses will be learnt by every switch in the network. This means that this tunneling method is not scaling from forwarding table point of view.

[1] Rapid Spanning Tree Protocol
[2] Multiple Spanning Tree

2.1.2 IEEE 802.1ah Mac-in-Mac (Provider Backbone Bridges)

The main issue with IEEE 802.1ad or Q-in-Q method is that since the forwarding in switching is based on source and destination MAC address, the backbone should be aware of customer's MAC address. As a result, the number of MAC addresses passing through backbone is too large for any bridge to store in its forwarding table. So, when forwarding table doesn't have more capacity to learn new MAC addresses, it broadcasts and floods unknown address frames. This causes unwanted traffic and security issues.

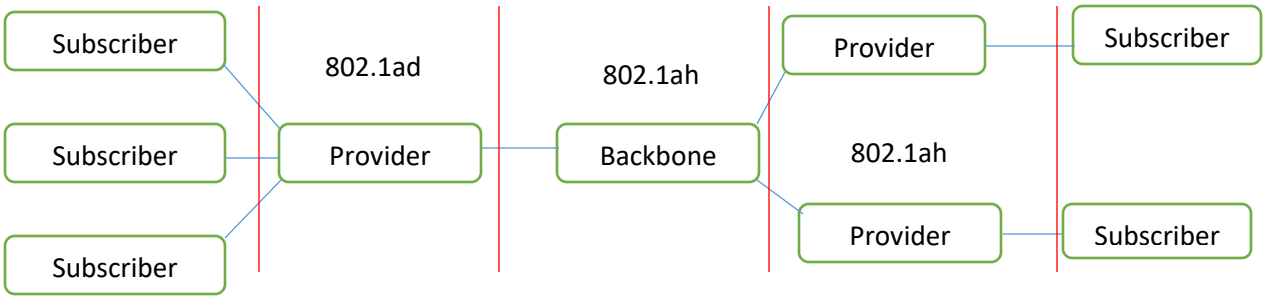


Figure 2 Deployment of IEEE 802.1ad and IEEE 802.1ah

To solve this problem, IEEE 802.1ah or MAC-in-MAC was introduced [3]. In this protocol, the customer traffic is encapsulated in provider's MAC address header, so that the customer MAC address will be hidden from backbone bridges.

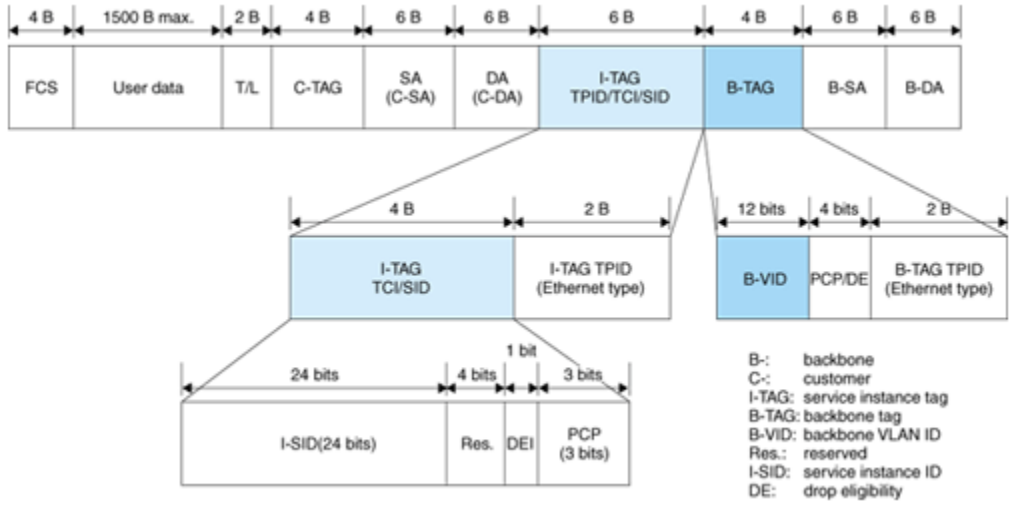


Figure 3 IEEE 802.1ah Provider Backbone Bridge Frame Format (image from: https://www.ntt-review.jp/archive/ntttechnical.php?contents=ntr200802gls.html)

There are some terminologies which are necessary to define [4]:

Backbone Edge Bridge (BEB): A backbone edge bridge which is located between customer and backbone network (Provider Backbone Bridge network or IEEE 802.1ah network) and is responsible for encapsulating customer frames before entering the backbone network and de-encapsulating the frames that enter the customer network.

B-Component: A bridging component that can be part of BEB and is responsible for bridging in backbone part of the network (deals with backbone VLAN or B-VLAN and backbone MAC address)

I-Component: A bridging component that can be part of the BEB and is responsible for bridging in customer network (deals with service VLAN or S-VLAN and customer MAC addresses).

Backbone Service Instance Tag (I-TAG): A tag which consists of 24 bits of I-SID, Priority Code Point, Drop Eligibility, and User Customer Address (1 bits used in CFM[1]). I-TAG doesn't use in core.

Backbone VLAN Tag (B-TAG): This tag includes backbone VLAN ID information. Each backbone VLAN (B-VLAN) can carry multiple services.

Backbone Service Instance Identifier (I-SID): Identifies the backbone service instance that the frame should be mapped to. It is a 24 bits field and indicates a specific flow. Service Instance can be:

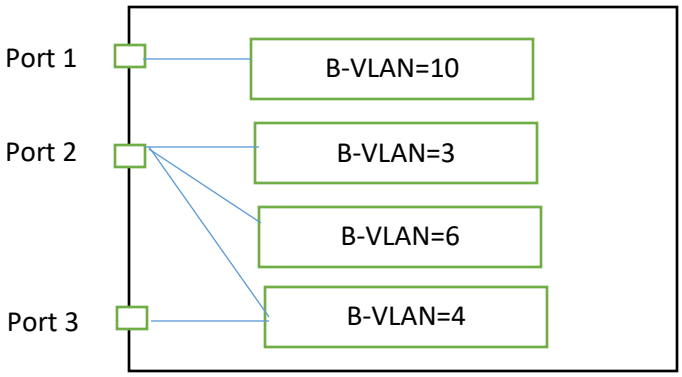
- All frames on a specific port
- All frames on a specific port with a specific service VLAN
- All frames on a specific port with a specific service VLAN and specific customer VLAN

Table 2 can be considered as an example for Service Instance ID and the mapped backbone VLAN (B-VLAN) for each flow, and figure 4 is a logical map, shows how each SID is mapped to a B-VLAN ID.

Table 2 Example of SID and mapped B-VLAN

SID	Source of the traffic	Mapped B-VLAN
10	Port 1	10
30	Port 2, S-VLAN=20	3
35	Port 2, S-VLAN=40	6
501	Port 2, S-VLAN=50, C-VLAN=200	4
505	Port 2, S-VLAN=60, C-VLAN=100	4

[1] Connectivity Fault Management



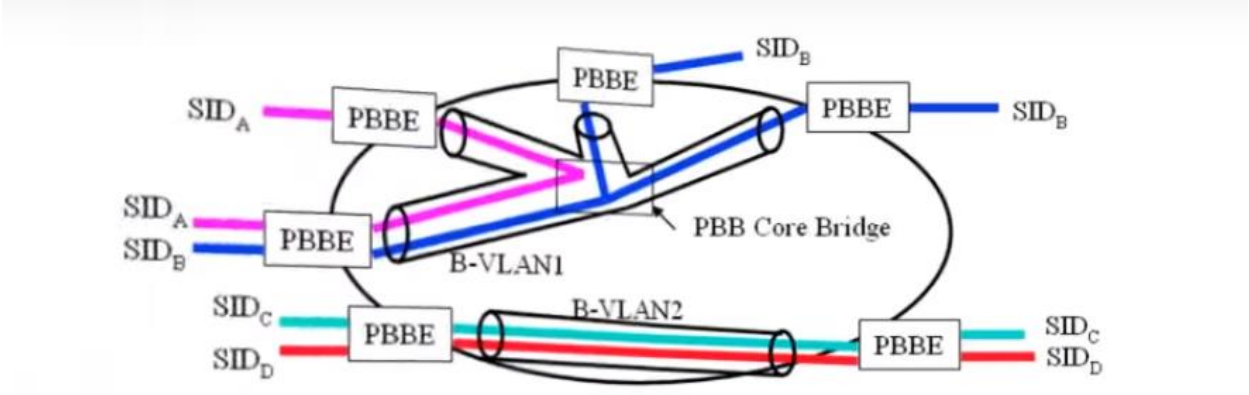


Figure 4 mapping each I-SID to B-VLAN

The IEEE 802.1ah uses MAC address tunneling encapsulation to tunnel customer frames across backbone network. In addition, B-VLANs are used to separate backbone network into multiple broadcast domains, and I-SID is used to map a given customer's MAC address frame to provider's service instance.

Advantages of Mac-in-MAC:

- It doesn't make any changes in switching process of the backbone network
- Provides a clear separation between customer and provider networks
- The only bridge that should learn customers MAC addresses in provider network is BEB, So it has some level of scalability
- PBB network can provide 2^{24} service instances
- Hide customers MAC address from provider network which brings us more security than Q-in-Q protocol

Limitations of Mac-in-Mac:

- It does not provide features such as multipathing, traffic-engineering and carrier-class resiliency[1] due to its dependency on traditional protocols such as spanning tree protocol for loop prevention

2.1.3 Cisco FabricPath

This protocol is Cisco proprietary and is mainly layer 2 routing protocol [5]. This protocol allows multipath networking at layer 2 and encapsulates Ethernet frame in its own header (FabricPath header).

There are some terminologies in FabricPath protocol which are described here:

Classical Ethernet (CE): The regular Ethernet frame with features and protocols such as flooding, STP, etc.

Leaf Switches: These switches are the ones which are connected to both FabricPath core network and CE network.

Spine switches: These switches are FabricPath backbone switches with ports residing in FabricPath domain.

FabricPath core ports: links on leaf switches which are connected to FabricPath domain, or links that connect one spine switch to another.

Switch ID: Unique number identifying each FabricPath switch.

Cisco FabricPath links are point-to-point. Leaf switches are responsible for encapsulating the Ethernet frames coming from CE network and de-encapsulating the FabricPath frames coming from FabricPath core, before transferring them to destination CE domain. In FabricPath domain and between spine switches, IS-IS protocol is used for layer 2 routing. The main goal of using IS-IS is to compute SPT (Shortest Path Tree) between all FabricPath nodes. There are several advantages for using IS-IS in core network. One of them is that it uses its own layer 3 transport which is part of CLNS stack. So we do not need IP in the core of layer 2 topology. In addition, IS-IS is natively extensible because of existence of TLV field. This field is where several features such as IP route, metric value, MPLS traffic engineering are encoded inside and included in IS-IS updates. For FabricPath protocol, it is used to advertise the FabricPath switch ID, forwarding path tags and ultimately to exchange control plane information that is going to allow IS-IS to build shortest path tree for leaf and spine switches.
Finally, IS-IS provide the advantage of ECMP[2] meaning layer 2 load balancing without relying on protocols such as STP, Port Channel and virtual Port Channel.
Because FabricPath encapsulates the entire Ethernet frame with another header, all nodes in FabricPath network need to support the protocol to forward the frame throughout the rest of the network (figure 5).

[1] "carrier class" refers to a system, a hardware or software component that is extremely reliable, well tested and proven in its capabilities[http://en.wikipedia.org/wiki/Carrier_Class]
[2] Equal Cost Multi Path

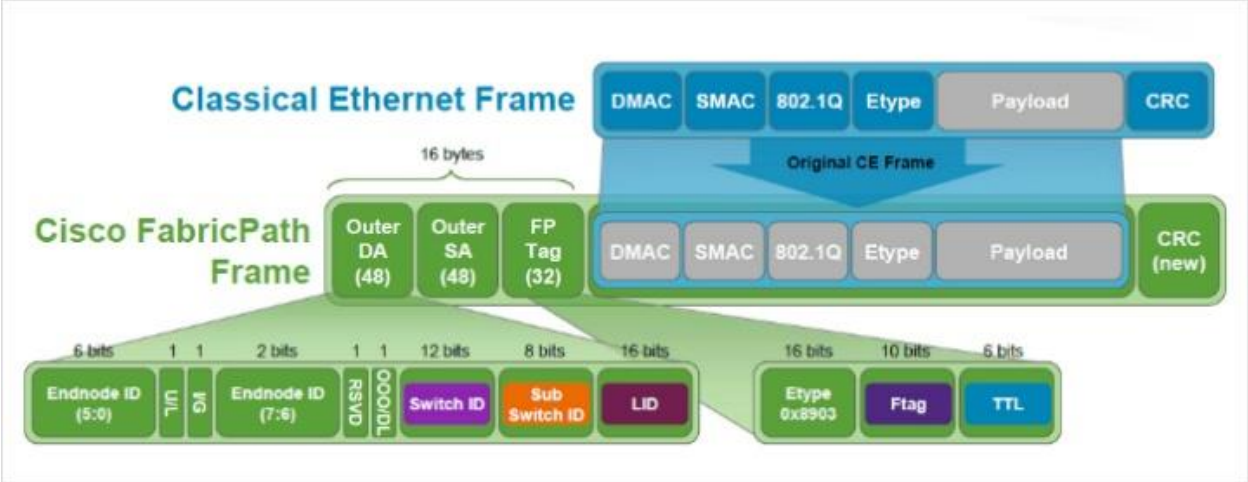


Figure 5 FabricPath header (image from: https://www.cisco.com/en/US/docs/switches/datacenter/sw/5_x/nx-os/fabricpath/configuration/guide/fp_switching.html)

According to figure 5, FabricPath header includes outer destination address and outer source address. ODA is the destination switch that we are trying to tunnel towards in FabricPath core and OSA is the switch ID of the source switch which is the leaf switch which encapsulates the CE frames. Each outer address consist of switch ID, sub switch ID (which is only changed if VPC[1]+ is used and otherwise is equal to zero), and Local ID (LID) which identifies the exact port which sourced the frame, or to which the frame is destined. LID removes the requirement for MAC learning on FabricPath core ports.

Ftag or Forwarding Tag is a unique 10 bit number identifying topology and/or distribution tree. For unicast packets, this field identifies which fabricPath IS-IS topology to use. For multi destination packets such as broadcast, multicast and unknown unicast, based on type of frame, different Ftag values are set. Based on Ftag values, the device understands the kind of forwarding and processing that needs to be done for each packet within FabricPath topology.

TTL field decremented at each switch hop and prevents frames looping indefinitely.

FabricPath protocol learns end host information through traditional MAC learning and Conversational MAC learning. The traditional MAC learning is the regular data-plane learning where each switch learns the source MAC of all received traffic and in the case of unknown destination, floods traffic to elicit response from the destination and then learns MAC address of destination from its response. Unlike this method, Conversational MAC address learning considers the bidirectional nature of communication flows, means an interface learns the source MAC address from a frame if that particular interface already has the destination MAC address in its MAC address table. Otherwise the source MAC will not be learned. This leads optimization of the MAC address table.

Advantages of FabricPath:

- Easy Configuration
- Multipathing feature (ECMP) using IS-IS
- High scalability

[1] Virtual Private Cloud

- Fast convergence
- Independency to traditional protocols such as STP

Limitations of FabricPath:

- Hardware support is limited since the protocol is Cisco proprietary and even some limited Cisco models support the protocol (Nexus 7000 F1 and F2 and Nexus 5500)
- FabricPath interfaces can only carry traffics that has been encapsulated using fabricPath protocol [6]

2.1.4 Transparent Interconnection of Lots of Links (TRILL)

This protocol is designed by IETF and is very similar to cisco FabricPath protocol, which is described in 2.1.3. It is implemented on devices called RBridges (Router Bridges), which apply network layer routing features to the link layer [7]. These devices are compatible with today's bridges and routers and same as routers, they terminate spanning tree protocol, bring optimal paths, have fast convergence and at the same time bring us a loop free network. On the other hand, they have some features which are considered as bridge features, such as transparency, plug and play (can be run with no or minimum configurations), support VLANs and can be used for data center bridging. RBridges find each other by exchanging TRILL Hello frames, which is one of the TRILL's control frames. These Hello packets have the multicast destination Mac address of "All-IS-IS-Rbridges" which are transparently forwarded and understandable by RBridges. There is two types of Hello frames: the default one is TRILL Hello and the other one is P2P Hello (only configured if exactly two RBridges exist on a link). TRILL Hellos are different from layer 3 IS-IS LAN Hellos since they are small, unpadded and support fragmentation. In order to use information exchanged in Hello frames, the RBridges located on same link elect a Designated RBridge for the link. Since it is important for RBridges to see each other's Hellos, Designated RBridge sends on lots of VLANs, tells others the single VLAN they should use to communicate.

RBridges use IS-IS flooding protocol, in order to make sure each one has the global link state database. The available information in link state database is comprehensive including not only connectivity and link cost, but also VLAN connectivity, multicast listeners, claimed nicknames (which is an identification for each Rbridge and announced in IS-IS LSP[1]s), ingress-to-egress option supported, etc.

The forwarding process from a source end node to destination end node using TRILL RBridges is as follows:

The first RBridge which picks up the frame (known unicast frame) encapsulates it to the last RBridge which can deliver it to its destination (figure 6). These frames are forwarded hop by hop by RBridge towards the egress RBridge. For multi destination frames such as broadcast, multicast and unknown destination unicast, forwarding takes place on a distribution tree selected by the ingress RBridge.

[1] Link State Packet

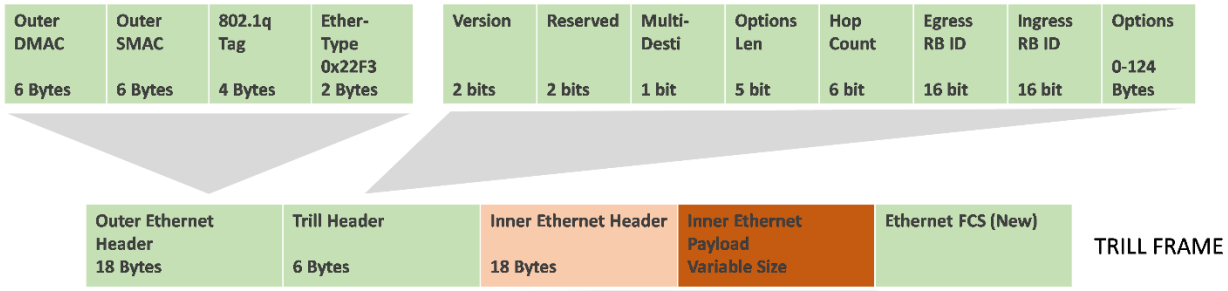


Figure 6 TRILL frame structure (image from: http://www.brocade.com/content/html/en/brocade-validated-design/brocade-vcs-fabric-ip-storage-bvd/GUID-C402D27C-09B4-429D-A63D-9FDE25246128.html)

As it is shown in figure 6, the first RBridge that picks up the frame encapsulates it with an outer Ethernet header. One reason to include this is so that if there is a bridge in the network, it can understand the frame and be able to forward it. In addition, having the outer Ethernet header is a way to specify "next hop" if there are multiple RBridge on a shared link. This header includes:

- outer destination MAC, which is the MAC address of the receiving RBridge,
- outer source MAC, which is transmitting RBridge,
- the 802.1q tag (Designated VLAN) which is only needed if RBridges are connected by a bridge LAN or carrier Ethernet and require a VLAN tag,
- And Finally the Ethernet type (0x22F3)

Right after the outer Ethernet header, we have the TRILL header which includes: 2 bits for version and 2 bits reserved, Multi destination bit, which will be equal to one if the frame is multicast or broadcast, option length which is the length of TRILL options, hop count which will be decremented per each RBridge hop and is a mechanism to prevent loops in the network and finally egress and ingress RBridge ID which is known as "nick name" according to TRILL RFC. The purpose of introducing nick name is to make TRILL header shorter. This address is dynamically acquired and is chosen randomly or can be configured. If two RBridges choose the same nick name, priority is used as tie breaker. Each RBridge announce its nick name via LSP. Including ingress RBridge nick name in TRILL header brings us some advantages such as the ability to create a mapping between source MAC address, which is part of original Ethernet frame, and the RBridge which makes the source reachable. In the case of having unknown destination or multicast, ingress RBridge address is useful for RPF[1] check and egress RBrdige address is used to select which "tree" to use to send multicast frames.
End host address information can be learned either via source MAC of encapsulated frame and source RBridge nickname or through a protocol named End-Station Address Information Distribution (ESADI), which is optional and mostly used to advertise end hosts that are more securely learned. The information distributed with this protocol is a table of local end hosts MAC

[1] Reverse Path Forwarding

addresses, which are known by the originating RBridge. ESADI is highly efficient transmission since information is tunneled through transit RBridges and encapsulated as if it was normal multicast data. This method is VLAN-scoped, means, only the RBridges that have end stations connected to the specific VLAN are paying attention to the frames.
TRILL advantages are as follows:

- Independency to traditional protocols such as STP
- Unicast forwarding tables at transit RBridges scale with the number of RBridges in the network and not the number of end stations. This is because transit RBridges do not need to learn end station addresses
- Provide multi pathing feature [8]
- Compatible with today's bridges

Limitations of TRILL:

- TRILL currently has no provision for expanding the LAN segment space beyond 4000 segments [8]
- TRILL uses different mechanism for forwarding multicast and unicast traffic, means for known unicast frames it uses the link state database to choose the optimal path whereas for unknown unicast and multicast, TRILL uses distribution trees by considering one RBridge as root. So if the state of MAC address changes from unknown to known, the packets might reach out of order. This make it more difficult to know the exact path which each flow travels.

2.1.5 IEEE 802.1 aq : Shortest-Path Bridging

Shortest Path Bridge (SPB) [9] is mainly defined to replace the use of Spanning Tree Protocol (and other protocols which can be considered as family of STP such as RSTP and MST), since STP does not use all links and blocks the alternate paths. This protocol is a layer 2 multi pathing protocol, same as cisco FabricPath and TRILL, means it enables us to have multiple paths with equal costs. So we can use arbitrary mesh topologies with high link utilization. This protocol is also efficient in handling broadcast and multicast traffic. In addition it uses extension to IS-IS to calculate Shortest Path Tree (SPT), similar to FabricPath and TRILL. The other noticeable aspect in design of SPB is that it supports two modes of operation and both can exist together in the same network. These modes are: Shortest Path Bridging-VLAN ID (SPBV) and Shortest Path Bridging-MAC address (SPBM). SPBV can be considered as a new control plane for Q-in-Q protocol (described in 2.1.1) and SPBM considered to be a new control plane for MAC-in-MAC (described in 2.1.2).
SPM is a deterministic protocol, since forwarding tables are populated locally to deterministically implement its part of the network forwarding behavior. As a result, offline tools are able to predict exactly where the traffic is going to go. This means SPB provides a light weight traffic engineering without involving CSPF and signaling, because of our ability to head end assignment of traffic to different shortest paths.
As mentioned, IS-IS is the link state protocol used to exchange information to calculate Shortest Path Tree (SPT) and operates as independent IS-IS instance, or within IS-IS/IP, supports multiple

topologies to allow multiple instances work efficiently. No new PDU is introduced in IS-IS to be used for SPB control plane. SPB uses the first TLV as part of Hello mechanism to inform other bridges about VIDs that it uses. The other TLVs (update TLV) informs about metric that it uses as SPB metric (beneficial if two different protocols for same functionality is used in a network). SPT algorithms are designed so that all the bridges in SPB network calculates the exact same shortest path trees. The main feature of shortest paths between each two bridges in SPB region is that they are symmetric means forward and reverse traffic uses the same path. In addition, unicast and multicast traffic between each two bridges takes the same path (congruence). By default, 16 Equal Cost Trees (ECTs) are used, in order to support load balancing. Then different VLANs are distributed over this 16 SPT sets. This control plane functionality is the same for both variants of SPB. Following is the description of each SPB mode:
Shortest Path Bridging VID- SPBV: SPBV is a flexible protocol that at the same time, can be used in networks with protocols such as IEEE 802.1q, IEEE 802.1ad (Q-in-Q) and IEEE 802.1ah (MAC-in-MAC). VID is VLAN Identifier and each VLAN that exits in network and handle by SPBV uses a SPT set. The assignment of each VLAN to each SPT can be done manually or automatically and the assignment map should be distributed to all the bridges in SPBV domain using an extension to IS-IS called ISIS-SPB. In fact, SPBV uses VLAN ID to encapsulate the frames in order to perform service delineation and load balancing. The encapsulation process is done at ingress, means the S-TAG or C-TAG of customer frame is mapped to the SPVID corresponding to the SPT that supports that VID. If customer frames do not have any 802.1q tag, the SPBV protocol adds a tag to the frames, at the entrance of SPBV domain. At the egress of SPBV network, the SPVID will be mapped back to its original VID or entirely removed (if the customer frames did not have any VLAN tag at ingress). SPBV mode of SPB differs from SPBM mode in that MAC addresses are learned on all bridges that exist on shortest path.

Shortest Path Bridging MAC – SPBM: SPBM mode uses 802.1ah or MAC-in-MAC for encapsulation in data plane, means customer frames are encapsulated in an Ethernet header which is MAC-in-MAC header. From operators point of view, the only things which needs to be done is to plug NNI's[1] together, group ports/C-VLAN and S-VLANs at UNI[2]'s that are needed to be bridged (we can have 2^{24} groups which is the number of services that can be defined in SPBM mode) and then assign an I-SID to each group. The process which takes place internally between bridges is that IS-IS reads box MAC address and forms NNI adjacencies and advertises the box MAC addresses. In addition, IS-IS reads the assignment of services to ports and then advertises that information along with topology information. At last each bridge computes its forwarding table that bridge service members. At data plane, when traffic arrives from customer at UNI, whether it is C-tagged, S-tagged or untagged will be encapsulated with B-SA (Bridge Source Address) of the ingress bridge, I-SID configured for group, and B-VID that is chosen for the route. The other field that should be set at ingress is B-DA (Bridge Destination Address) which should be found out using C-DA (Customer Destination Address). If the C-DA couldn't be found at ingress will be multicast.

[1] Network-to-Network Interface
[2] User-to-Network interface

The process of path calculation and filing the forwarding data base is done through following steps:

- One specific shortest path is calculated from each node to all the other nodes
- B-MAC (Bridge MAC) of other bridges in SPBM domain will be installed in forwarding table
- The two steps above will be repeated for 16 shortest paths (by default) and each results a different B-VID to be used and symmetry of each path will be assured (table 3 and 4)

Each shortest path will be calculated by a different SPT algorithms and these algorithms are defined in such a way that it is guaranteed that all bridges in the region calculate exactly the same set of trees. SPTs are bidirectional, means forwarding and reverse traffic travel the same path.

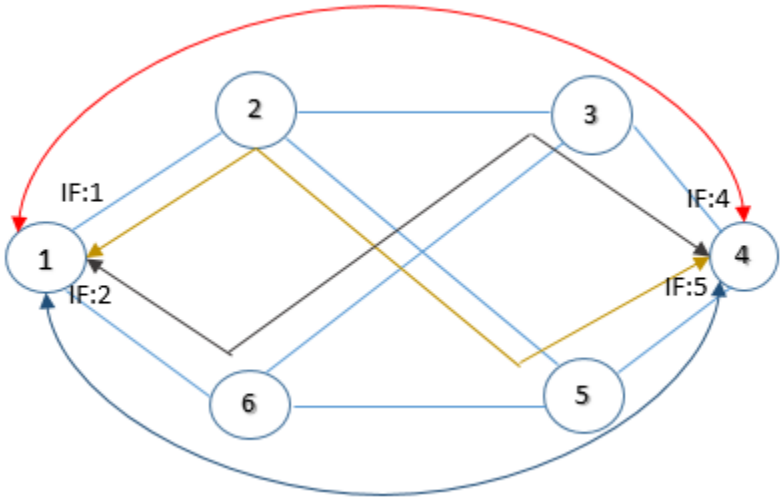


Figure 7: selected and usable Shortest Paths between two nodes in SPB

Bridge 1 FDB:

Table 3 Bridge 1 Forwarding Data Base

B-MAC	B-VID	Interface (IF)
:4	1	1
:4	2	2
:4	3	1
:4	4	2

Bridge 4 FDB:

Table 4 : Bridge 4 Forwarding Data Base

B-MAC	B-VID	Interface (IF)
:1	1	4
:1	2	5
:1	3	4
:1	4	5

Advantages of SBP:

- Good scaling without loops(using two mechanisms for loop prevention including loop suppression and loop avoidance)
- Uses multiple shortest paths
- Efficient broadcast / multicast and existence of replication points
- Great recovery and convergence time
- Independent to STP

Limitation of SPB:

- Same as other protocols discussed earlier, SPB tunnels layer 2 frames over layer 2 network. These protocols are locked into hardware and cannot be easily integrated into other solutions. We have other protocols such as OTV, VXLAN and NV-GRE which are not limited to the hardware and in fact, tunnel layer 2 frames over layer 3 network and solve some limitations of hardware.

2.1.6 Overlay Transparent Virtualization (OTV)

OTV [10] essentially is a tunneling technique that allows us to bridge layer 2 traffic over a layer 3 data center interconnect. This encapsulation is also called MAC-in-IP and designed to expand layer 2 domain between different sites. One of the most important use cases of this technology is Virtual Machine workload mobility. As an example, if we have virtual machines in one data center and number of virtual machines in another data center and we want to implement VMotion[1] between them, one of the requirements of this from design perspective is having layer 2 reachability from one data center to another. In this case we can use OTV to extend layer 2 domain over layer 3 data center interconnect.

OTV uses stateless tunnels for encapsulation process and in fact, encapsulates the whole Ethernet frame in IP/UDP header, so the core network will be transparent to the service provided by OTV. There are three areas that OTV tries to improve by its internal mechanisms.

1. Going from data plane learning to control plane learning
2. Going from pseudo wire tunneling to dynamic encapsulation
3. Going from complex multi homing to built-in multi homing

The first area is equivalent to MAC address routing. OTV uses IS-IS control protocol with some extensions in order to support MAC address reachability advertisement and end-node information, instead of traditional flood and learn mechanism. Since OTV mostly used over WAN or MAN networks for connecting multiple sites and data centers, layer 2 mechanisms for different purposes such as flooding leads to problems in the network.

[1] Available feature in VMware for virtual machine mobility.

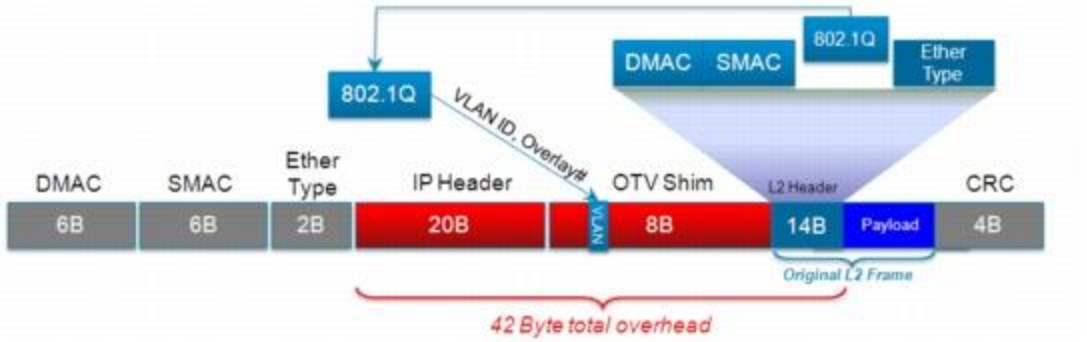


Figure 8: OTV header (image from: https://www.cisco.com/c/en/us/td/docs/solutions/Enterprise/Data_Center/DCI/whitepaper/DCI3_OTV_Intro/DCI_1.html)

As it is showed in figure 8, entire Ethernet frame is encapsulated in IP packet. Then we have OTV shim to the header, which is basically 802.1q header copied to OTV header to encode VLAN information.

There are some terminologies which are used in developing OTV protocol, and they are described as follows:

Edge device: edge device connects the site to LAN and WAN and is responsible for all OTV functions. In most of network designs, edge device is in aggregation layer with possibility of multiple edge devices on a single site.

Internal interfaces: interfaces on the edge device that only the internal site uses. These interfaces act as normal layer 2 interfaces and can be part of STP which may implemented in local site.

External interface: This interface is NNI and can be a router point to point interface on the edge device that connects the local site to the provider or core network.

Each edge device can discover other edge devices running OTV by using external interfaces to join multicast group in the core and as source address for OTV encapsulation.

If you consider figure 9 as an example for creating OTV control plane, the first step for forming adjacencies is that all edge devices will join a multicast group in the core as a *host*. Edge A sends Hello encapsulated in IP packet and send it to the core. Core network replicates the packet and forwards to other edge devices to unwrap it. Edge B and C encapsulate Hello response and send back to edge device A. When all the edge devices become adjacent, they can start to send reachability information to each other. To achieve this amount of scalability, OTV uses two multicast group for communication:

ASM/BIDIR group: this group is used to establish adjacencies and MAC address reachability information.

SSM group: It used to multicast data, generated per each site.

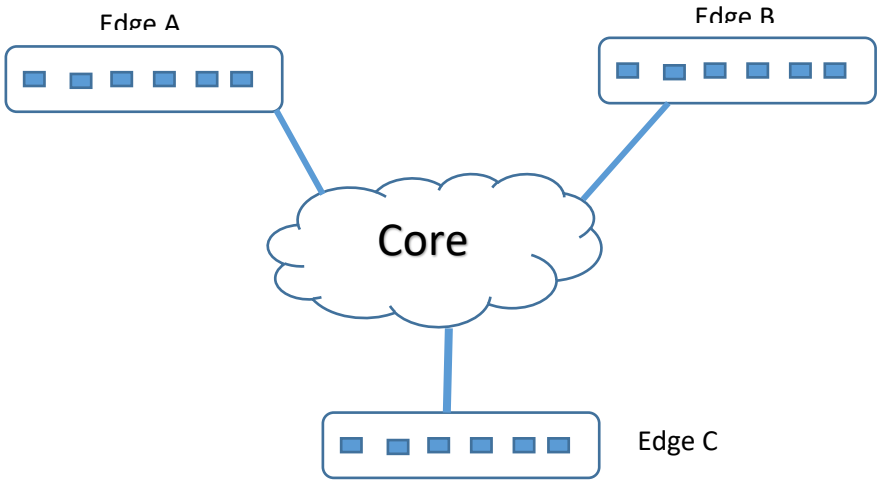


Figure 9 Topology of an OTV network

Another way to make proper adjacencies (in the case that multicast support would not be possible) is an OTV adjacency server to distribute a list of all peer edge devices in the overlay.
As mentioned before, OTV does not flood MAC addresses or STP BPDUs. The way each edge device learns about end-hosts MAC addresses is by advertising MAC address reachability among themselves, after they become adjacent with each other. So each time an edge device learns a new MAC address, OTV advertises it with its VLAN ID and next hop IP. As a result, each edge devices have their own MAC routing table. This process is a dynamic process and no configuration is needed.
There are always unknown unicast packets in the network, which OTV should handle it. On the internal interfaces, the normal layer 2 flooding can be used. On the other hand, the edge devices can send unknown unicast packets out onto the OTV grid. So OTV takes care of the unknown unicast packets by snooping the ARP reply, learn the MAC address and add it to the OTV MAC table. Edge devices act as proxy ARP for all the end hosts in the local sites to reduce ARP traffic on OTV grid.
For handling loop free replication of devices and multi-homing without STP, OTV uses a designated forwarding device per each site and for each VLAN. This is known as AED (Authoritated Edge Device). This AED is the only device that can forward broadcast or multicast traffic across the overlay.
Advantages of OTV:

- This protocol is *transport agnostic*, means any infrastructure which implements IPV4 can work with OTV, no matter it is MPLS or regular internet transport. The only requirement is that both sides can have ping of each other.
- Scaling enhancements by suppressing STP, ARP and unknown flooding. OTV has cashing mechanism which basically performs snooping inside layer 2 payload, looking for ARP frames, then cash whatever IP to MAC mapping and reply locally to whatever internal end host which is trying to resolve address.

Limitations of OTV

- Same as FabricPath (described in section 2.1.3), OTV is a cisco proprietary protocol and is only available on specific models
- In some large data centers, several points of delivery (PoD) connect over layer 3 network per data center. In this case, if the data center operator needs to extend layer 2 VLANs across multiple PoDs in a same data center and across other data centers in different regions, it can be challenging for OTV. The reason is OTV has some limitation in maximum number of sites that needs to be interconnected. This number is highly related to the platform or software release [11].

2.1.7 Locator/Identifier Separation Protocol (LISP)

LISP [12] is a layer 3 overlay scheme over layer 3 network, provides new semantics for IP addressing.

When a node moves from one network to another, the IP address of the node is no longer valid. So the node needs configuring new IP address, which is not a good decision when VMs are moving from one subnet to another or in the case of mobile devices.

LISP separate node address from location addresses so that when a node moves from one network to another (for example from one ISP to another), the node ID stays the same and location address will change. These addresses are defined as EID and RLOC. Routing locators (RLOC) describes the topology and location of the endpoints, which are attached to a network, and is used to forward traffic. End point Identifiers (EIDs) are used to address end points or hosts, separate from location of the node in the network. LISP uses encapsulation of IP packets in another IP header (figure 10), where the address in outer header will be locator (RLOC) and the address in inner header is EID.

By keeping EIDs fixed, even if a company wants to change its ISP from one service provider to another, the only thing that needs to change is the IP address of point to point link of the customer edge device to provider.

So using this method, companies do not have to worry about PA[1] addresses that the service providers are providing to the customers, since service providers only need to address the point to point link on customer edge side. As a result address space can be used more efficiently.

The other advantage of LISP is that it enables us to move a VM from one site to another and keep the exact IP address (EID) and only configure the locators that are associated with its EDI prefix. Generally locators will be changed in the case of: 1) changing service providers 2) roaming hand-sets 3) relocating infrastructure into a cloud.

[1] Provider Aggregatable address space: Address space that is typically assigned by ISPs to customer so that routing information for many customers can be aggregated once it leaves the provider's routing domain. If the customer leaves the ISP who assigned the IP address space, it will have to reconfigure all their hosts and routers if they continue to require globally unique address space.

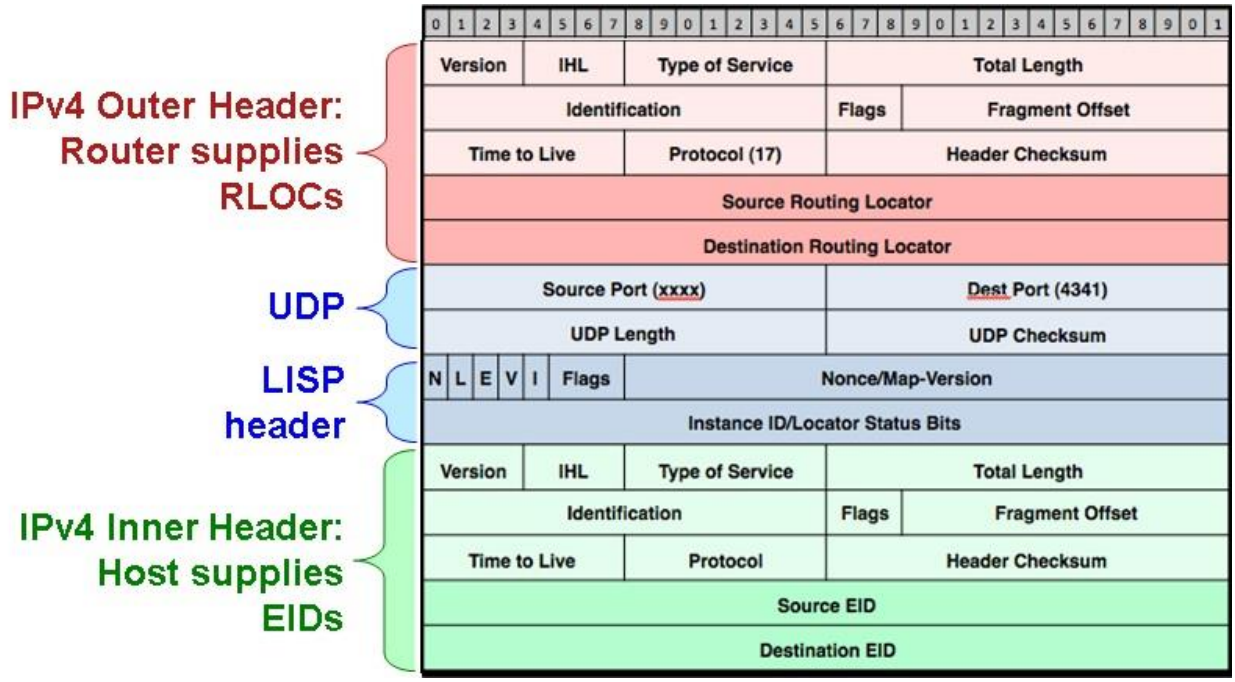


Figure 10 LISP packet format (image from: https://commons.wikimedia.org/wiki/File:Lisp-header.jpg)

The encapsulation process is done as follows: Consider figure 11 as an example. If node S wants to sends packet to node D, the host S builds the packets and put IP header by considering IP address of source (itself) and the destination (both are defined as EIDs). The packets then find their way to the edge. Then there should be a LISP router in edge to encapsulate the packets with another IP header. This device is called iTR (ingress Tunnel Router). In LISP, the packet is encapsulated using UDP header (figure 10). When the packet reaches iTR for encapsulation, iTR should search its map cache to find the corresponding locator for the destination IP address (destination EID). Map cache or mapping entry is a new data structure which is introduced by LISP and uses longest match look up to find the matching address of the locator that can deliver packet to its destination. If we assume that the entry for D (with IP address of 2.0.0.1) **has been already cached**, we will have a map entry on one of the iTRs as follows:

EID-prefix: 2.0.0.0/8
Locator-set: 12.0.0.2, priority:1 weight:50 (eTR1)
13.0.0.2, priority:1, weight:50 (eTR2)

Note that priority and weight are the only parameters that are used to make policies and these policies are determined by eTRs (egress Tunnel Router). ETRs are LISP routers that de-encapsulate the LISP packets and since each LISP router can be both iTR and eTR, they can be referred as xTRs. So priority 1 for both eTR1 and eTR2 at destination site means they both are active to be used as egress. When the priorities are the same for multiple locators, the Weight indicates how to balance unicast traffic between them. If all the weights are the same like what is shown in the example, the iTRs will decide how to load-split the traffic between locators or eTRs. In outer IP header, the source address will be the address of the iTR which is sending the

encapsulated packet to the core and the destination will be one of the eTRs (e.g. outer source: 11.0.0.1 and outer destination: 13.0.0.1). Packet in the core then will be sent on the shortest path and will be received by eTR to unwrap and deliver to destination D.

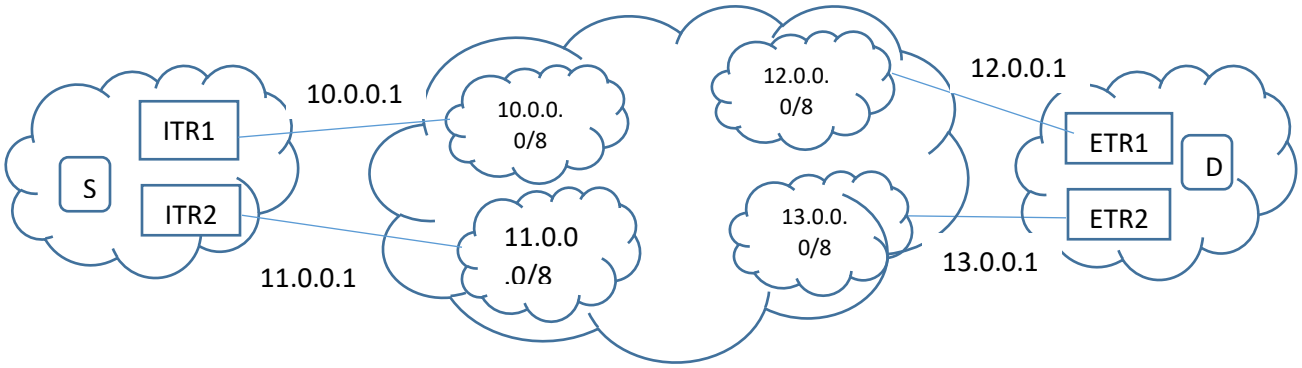


Figure 11 Example of LISP network

In the above example we assumed that the entry corresponding to the destination exists in the map cache. If the map cache for destination D missed, iTR2 builds a control packet called a map request for destination D (2.0.0.1) and so the destination address of the outer header will be IP address of D and the source is the IP address of iTR2. The information about each EID prefixes and their corresponding locators are pushed in devices called MS (Mapping Servers), which are connected together using an alternate logical topology of GRE tunnels, called ALT (Alternate LISP Topology). Each locator needs to access MS to get the map entries they want and this access can be done via MR (Map resolvers). So iTR2 which creates map request, sends the request to one of the MRs and will be responded by the map entry of the EID prefix it needs.

LISP advantages are as follow:

- LISP provides a mechanism to ensure that all virtual segments are isolated from each other. This is done by adding 24-bit instance ID field in LISP header, which allows more than 16 million virtual segments to be instantiated and is set by iTR.
- Improved routing scalability and efficient use of address space
- Easier mobility for devices (there is no need to change end node address as it moves)
- Provides the multi-homing feature without any BGP configuration for having active-active mode

Limitations of LISP are as follows:

- The protocol complexity is one of the drawbacks, since LISP is a different concept and users need to learn it and adopt themselves with how the technology is designed and should be implemented
- There is a delay in transmission, since the first packet towards a new destination needs destination look up, which takes time, although other next incoming packets will benefit from destination information cache

- Since LISP uses tunneling technique to transfer packets from iTR to eTR, it might experience some potential MTU issues (which is also a common issue in other overlay protocols) [13]

2.1.8 Virtual Private LAN Service (VPLS)

VPLS [14] enables service providers to offer their customers the operational cost of the Ethernet with the QOS facilities that MPLS provides for the users. VPLS is a Multipoint layer 2 VPN and allows the creation of pseudo-wires that emulate LAN segments for a set of customers and they have the ability to learn and forward Ethernet MAC addresses of corresponding sets of users. So we can describe VPLS as a multipoint VPN that allows multiple sites to be connected in a single bridged domain over a core network which is a MPLS[1] switching and is managing by the service provider. The MPLS network of the provider is transparent to customers and each customer uses Ethernet interface to connect to the WAN network (MPLS) which provides rapid and flexible provisioning. A VPLS capable network consist of three main components:

1) Customer Edge: The edge device located in customer's site and connects the customer's LAN to provider's MPLS network

2) Provider Edge: All the configuration related to the VPLS tunnels is done on PE devices. These devices are originating and terminating point of each tunnel and for each and every VPLS and in *most* of the implementations each PE needs to have a connection to all other PEs (full mesh topology). A tunnel between each pair of PEs are called LSP[2] and is created by exchanging UDP Hello packets for establishing a TCP connection.

3) Core MPLS network: MPLS uses virtual paths to link different locations together. MPLS is a switching technique which has layer 3 awareness but instead of using addresses in a routing table, it forwards traffic based on path labels, which are identifying paths in the network and not the endpoints. These labels are created per each prefix and uses routing table to build the forwarding table based on labels (LFIB). One of the features which makes MPLS interesting to use for VPN services such as VPLS is its capability to stack multiple tags, enabling overlay services to be used transparently from the transport network. By using MPLS as the core forwarding protocol, specific applications can be prioritize using QoS, so that most important information gets the most bandwidth.
MPLS is only used to interconnect the PEs. So it does not really participate in VPN functionality and all the VPN intelligence resides in PEs.
For encapsulation, VPLS uses MPLS labels to identify the virtual customer segment or VPLS instances. The outer label is used for normal MPLS switching functionality. For label distribution, signaling and discovery, VPLS can use two methods: One of them is using BGP and the other is using LDP[3]. If BGP is used to establish the tunnels, the inner label will be assigned by PE as part of a label block and if LDP is used as control plane method, the inner label is virtual circuit ID

[1] Multi-Protocol Layer Switching
[2] Label Switch Path
[3] Label Distribution Protocol

which is assigned by LDP when it first establishes tunnels between PEs that are participating in the VPLS.

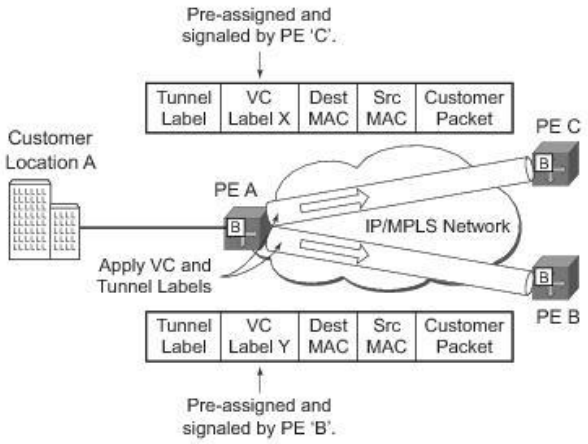


Figure 12 VPLS header (image from: https://infoproducts.alcatel-lucent.com/html/0_add-h-f/93-0076-10-01/7750_SR_OS_Services_Guide/services_con_vpls.html)

VPLS can be implemented in two modes:

Tagged mode: In this mode all LAN segments of a customer that are connected to PE belong to the same broadcast domain. It means the customer payloads which are carried over a MPLS core will be stayed unchanged and even if the customer frame had an IEEE 802.1q VLAN tag, it will be ignored.
Untagged mode: In this mode customers LAN segments can have several broadcast domains by assuming that at least one VLAN tag is present. So whenever a PE receives a frame, it removes the VLAN tag and map it to the inner VPLS label. The destination PE or the termination point of the tunnel reads the VLAN ID so that if the frame is broadcast, it will be forwarded only to CEs that are in the same VLAN. In addition, the egress PE will append the VLAN tag back to the frame for forwarding, and removes the MPLS label tags.
For MAC address learning, each PE has the responsibility to learn which remote PE has the connection to each given MAC address learning.

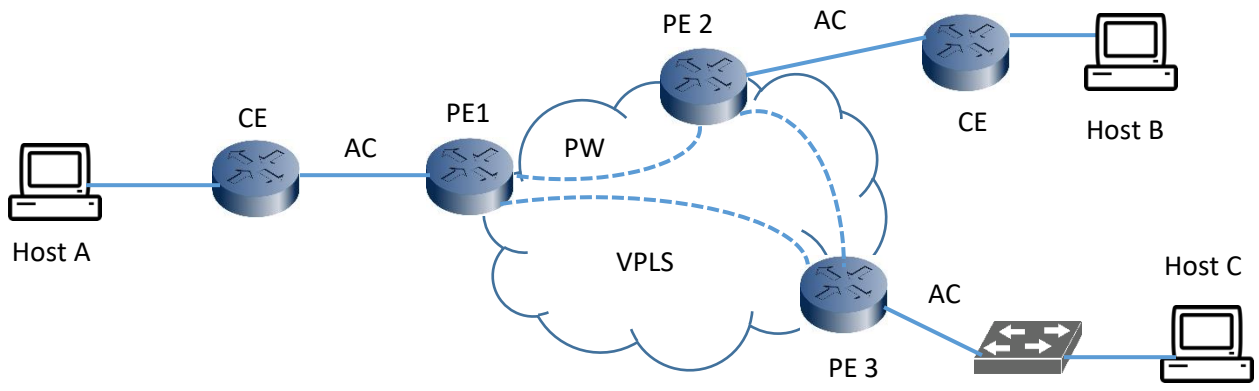


Figure 13 Example of VPLS network

For describing Unicast forwarding, consider figure 13 as an example. Suppose that MAC address of Host A is A, MAC address of Host B is B and MAC address of Host C is C and they all belong the same customer network (network C). AC (Attachment Circuit) represents the connection between the customer and a service provider network, which can be a physical port or a virtual port. Suppose that host A sends a frame which is destined to MAC address C. If PE1 does not know the location of MAC address C, it floods the packet to all of its ports except the port which it received the packet from (same as what a bridge would do). This means that packet will be flooded to pseudo wires towards PE2 and PE3. Since pseudo wire label (VC label) for each PE peers are unique per VPLS, PE2 and PE3 know that the packet belong to customer network C's VPLS. Each PE has a forwarding table corresponding to customer C network and when they receive a packet from associated pseudo wire, they perform destination MAC address lookup in their forwarding table. If PE3 does not know the location of MAC C, it floods the frame on its local ports to CE. But PE3 will not flood the frame to another PEs. The reason is "split horizon" rule, which indicated that a PE should not forward traffic from one pseudo wire to another in the same VPLS mesh. In addition, each PE which receives a frame from a remote PE, if it does not have the source of the frame in its forwarding table, it will create an entry with association between source MAC address of the frame and its respective pseudo wire. So if PE3 does not have an entry for MAC A, it will create one and associate it with the pseudo wire to PE1. Similarly, PE2 forwards the frame on the port facing the router of CE.
VPLS architecture has some limitations:

- Scaling is one of the limitation in flat VPLS model. This limitation is due to the requirement of having full mesh LSP tunnels between all PE peers per each VPLS.
- Packet replication requirement is another limitation which can adversely affect large network deployments
- As number of sites increases, each PE should learn more MAC addresses. MAC address does not have hierarchical structure as IP address has and so forwarding tables do not scale well

There is another VPLS model which is called H-VPLS (Hierarchical VPLS) which addresses the limitations of flat VPLS. In H-VPLS, there are two types of routers: u-PE and n-PE. U-PE routers are the routers which are facing the users and n-PE routers are the ones facing provider network. Using H-VPLS instead of flat one gives us the benefit of less signaling in MPLS core and less packet replication on n-PE routers. In addition, the hierarchical model solves the problem of high rate growth of MAC addresses learned by PEs by hiding the addresses of edge VPLS with the address of the edge PE that is connected to the main VPLS network. If figure 14 is considered as an example of an H-VPLS network, the network consist of 2 hierarchical level: One is VPLS core pseudo wires (can be considered as hub) and the other is access pseudo wires (the spoke). Each customer site is connected to a u-PE device. The u-PE device has one pseudo wire to n-PE devices. N-PE devices are attached in a full mesh VPLS network. Each spoke pseudo wire terminates on a VSI[1] on u-PE and n-PE. U-PE devices support layer 2 switching and act as a bridge, means they can do learning MACs and replicate the broadcast and unknown packets to all of its ports including

[1] Virtual Switch Instance

the local ports and spoke pseudo wire. So after the learning process of all the MAC addresses of the customer devices are done, u-PE can decide whether it should send the packet locally or switch it onto the pseudo wire and send it to n-PEs. U-PEs can be dual homed to prevent loss of connectivity in the case of n-PE failure or pseudo wire connectivity. N-PEs, on the other hand, support both the bridging functionalities and routing and MPLS encapsulation.

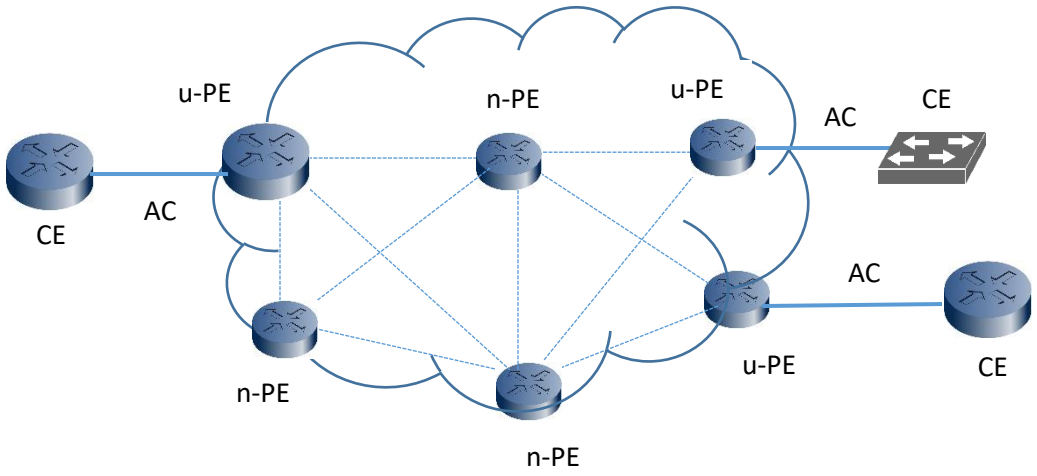


Figure 14 example of an H-VPLS network

2.2 Host based Overlay technologies

Host based protocols are trying to address some problems which remained unresolved in most of the network based overlay protocols. These problems include the ability to provide mobility in workload in the network, simplification and automation of workload provisioning and scalability in multitenancy. Host-based protocols have some properties which are common in most of them. These properties are include:

- Unlike network based overlay technologies, host based technologies have centralized control and point of management
- The elements that perform encapsulation and forwarding can be servers or virtual end points

2.2.1 Virtual Extensible LAN (VXLAN)

VXLAN [15] is a layer 2 overlay protocol over a layer 3 network. It uses an IP/UDP encapsulation. This mechanism makes it appear to hosts or virtual machines of a particular tenant, separated by layer 3 devices and subnets, as if they have a direct layer 2 connection to one another. There are different tunneling and overlay protocols that make this happen but VXLAN has different advantages that makes it a popular technology to use in data center networks. Using VXLAN, the provider or core network does not need to know about the services that VXLAN provides. So almost the only tunneling requirement for underlying network is that there should be IP connectivity between leafs or the devices which connect the customer sites (including its application servers) to provider core network. In addition, a 24 bit VXLAN segment ID called VNI

is included in the encapsulation which provides up to 16 million LAN segments. This can be considered as a benefit compared to traditional 802.1q protocol which only supports 4096 tags. It also allows multi-pathing since the core network is a layer 3 network and uses layer 3 ECMP. The other advantage of VXLAN is that it allows virtual machine workload mobility because it keeps layer 2 adjacency requirements.

Figure 15 shows the encapsulated VXLAN frame. VXLAN tunnels are stateless and the encapsulation process is done by an entity called VTEP[1]. When a frame reaches a VTEP which can be located either on a physical switch or within the hypervisor virtual switch in a server virtualization deployment, it first map the VLAN ID of the packet to a VXLAN ID or VNI. Then it looks for remote VTEP which is associated to the destination MAC address of the packet in order to encapsulate the packet towards it. So every VTEP has a mapping table with associated MAC addresses, VXLAN ID and remote VTEP. These mapping tables can get these mappings by manual configuration, control plane source learning (push mode) or central directory lookup (pull mode). The most popular way of learning these information is flood and learn mechanism. In this mechanism VXLAN establishes point to multipoint tunnels to all VTEPs on the same LAN segment. So unknown destination packets can be handled by encapsulating broadcast messages (same as normal ARP broadcast) from each host or virtual machine into multicast messages to get across core network to VTEP subscribers which are a member of the designated multicast group. Consider figure 16 as an example of VXLAN network.

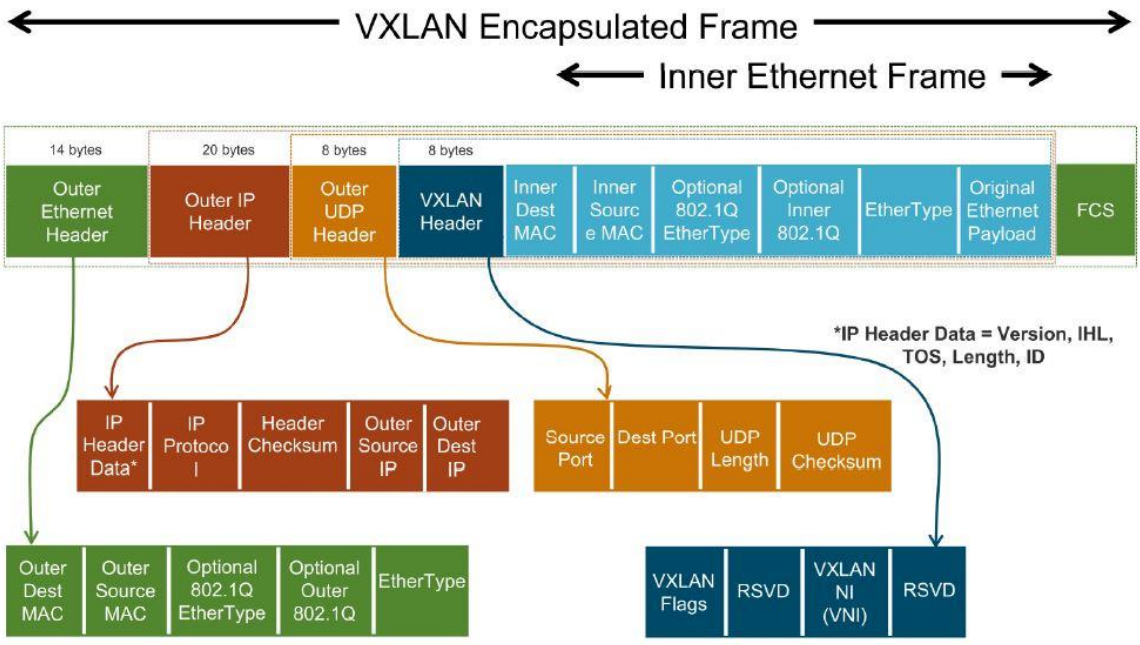


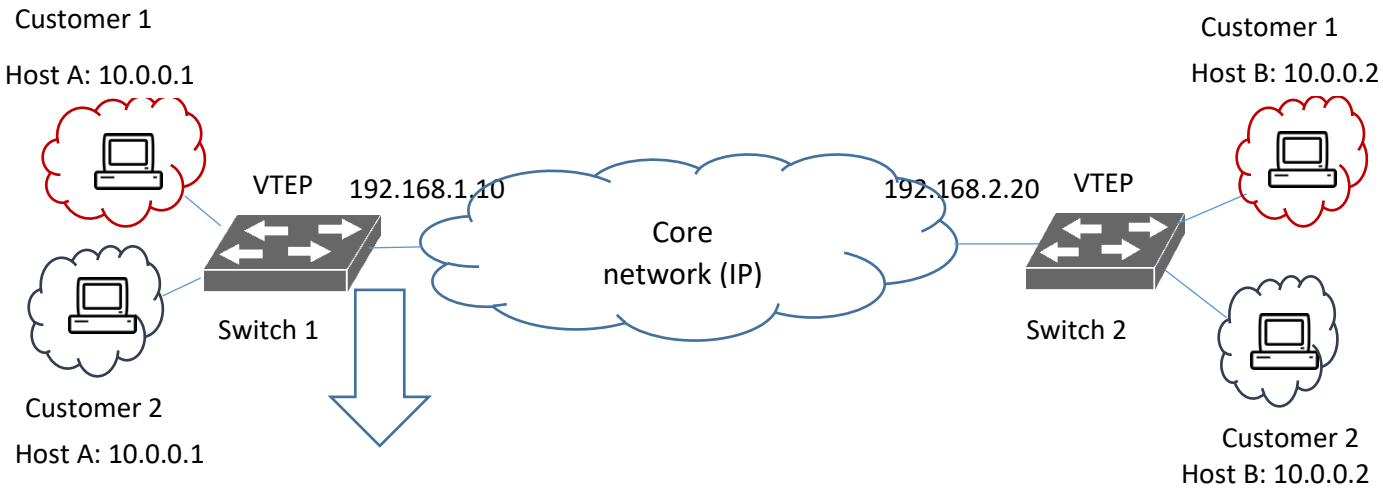
Figure 15 VXLAN frame format. (Image from http://chansblog.com/tag/vxlan-tunnel-endpoint/)

If host A wants to send packet to host B, by considering both hosts part of customer 1 network, the packet first will be received by VTEP on switch 1. It looks up the VLAN to VXLA ID mapping

[1] Virtual Tunnel Endpoint

table and finds the corresponding VXLAN ID. Then by having VXLAN ID and destination MAC address, it finds the IP address of switch 2, which is associated with host B. So it encapsulates the packet with new headers as follows:

Destination IP address: 192.168.2.20
Source IP address: 192.168.1.10
UDP destination: 4789 (VXLAN port)
UDP source: dynamic (based of hash of the inner packet)
VXLAN ID (VNI): 100
VTEPs use VXLAN ID or VNI to distinguish between tenants or customers. So in the example network shown in figure 16, VTEP on switch 1 first find the VXLAN ID of the customer site according to its VLAN tag, so if customer 1 VLAN is 50, according to VLAN to VXLAN ID map, the VXLAN will be 100. Then it finds the remote VTEP for encapsulation, according to this VXLAN ID. When packet received by VTEP on switch 2, it knows VXLAN ID of 100 is reserved for customer 1.



VLAN	VXLAN ID
50	100
20	200

MAC	VXLAN ID	Remote VTEP
:B	100	192.168.2.20
:B	200	192.168.2.20

Figure 16 Example of VXLAN network

So destination VTEP, de-encapsulates the outer header, leaving the original header, which indicates a packet from source host A to destination host B and so it forwards the packet to host B.

Limitation of VXLAN:

- By using VXLAN, another 50 bytes overhead should be added to each packet. So even small packets such as Telnet/SSH need to wrap by VXLAN header before being sent to the other host or Virtual Machine. This can be considered as an overhead.

In summary, VXLAN is an overlay technology which is widely used in data centers. It provides high level of scalability because of using 24 bits VXLAN ID or VNI, allowing layer 2 adjacency over layer 3 and multitenancy.

2.2.2 Network Virtualization Using Generic Routing Encapsulation (NVGRE)

This protocol creates a virtual layer 2 topology on top of the physical layer 3 network (IP network). NVGRE [16] solve the exact problems that VXLAN solved but it uses different technology for it. The NVGRE specifications was proposed by Microsoft, Intel, HP and Dell. NVGRE uses GRE[1] tunnel for encapsulation. This protocol can be used to encapsulate different types of network protocols inside virtual point to point links over IP underlying network.

The terminologies which is used in describing NVGRE are as follows:

Customer Address (CA): This is the IP address which is set on customer's host or virtual machine. This IP is only visible by the Virtual Machine itself and the applications which are running on it.

Network Virtualization Edge (NVE): This is the device which encapsulates or de-encapsulates customer packets. These end points are similar to VTEP in VXLAN environment and they are also perform layer 2 functionalities. These devices are also used to apply isolation policies based on virtual segment ID (VSID) and can exist on a network device or physical server.

VSID: This is a 24 bit segment ID which uniquely identifies a virtual layer 2 broadcast domain. Similar to VXLAN, this address provides 16 million segments to identify each tenant network uniquely. As a result, NVGRE solves the current VLAN limitation for providers and data centers with high number of tenants. NVGRE uses the GRE header to include VSID information in each packet.

Provider Address (PA): This is IP address of underlying physical network. In NVGRE, there is mapping policy which maps PAs with associated customer address VM.

[1] Generic Routing Encapsulation

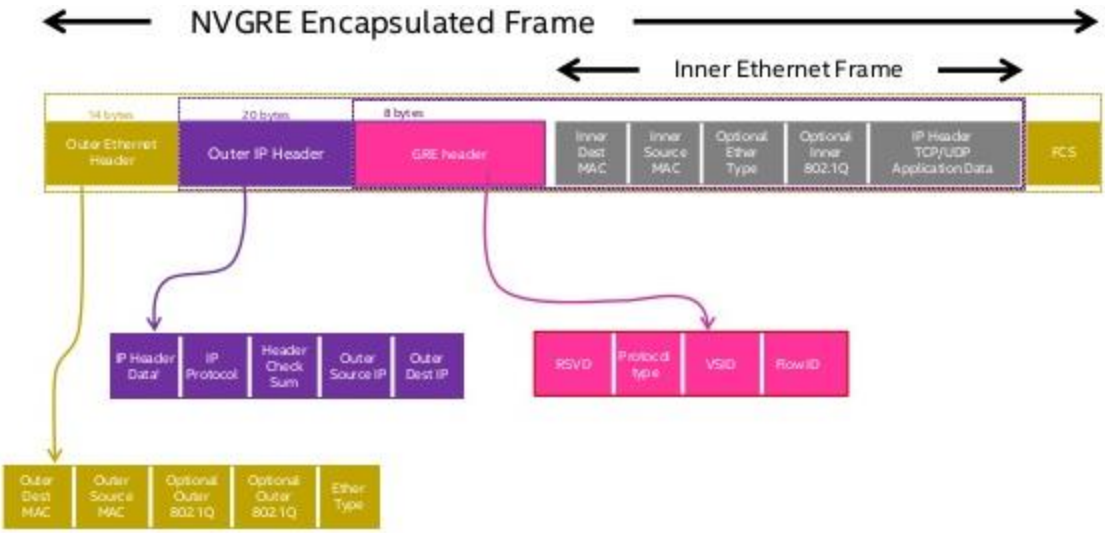


Figure 17 NVGRE frame format (image from: https://www.slideshare.net/LarryCover/virtualizing-the-network-to-enable-a-software-defined-infrastructure-sdi)

When a host in a customer network sends a frame to another host in the same network (in the same broadcast domain), the NVE or NVGRE end points, which are the ingress/egress points between virtual and physical network, are responsible for encapsulating and de-encapsulating the frame. To encapsulate an Ethernet frame, they need to set the following parameters for outer header:

The outer Ethernet header: As it is shown in figure 17, outer Ethernet header includes destination and source MAC address and 802.1q tag. The source Ethernet address is the MAC address of ingress NVE. The destination address is the MAC address of the next hop device towards the egress NVE or destination NVE. The outer 802.1q VLAN tag is optional and can be used by provider for its physical network to have broadcast scalability.

The outer IP header: Both source and destination IP addresses of the outer IP header are PAs. So the source IP is the IP address of ingress NVE (the device which encapsulates the packet) and the destination IP is the egress NVE, which receives the packet, de-encapsulates it and deliver it to the destination host. To encapsulate an Ethernet frame, NVE needs to know the associated egress NVE which can deliver frame to the destination host; and the related location information including VSID. Each NVE can obtain these information by management plane, data plane learning or combination of control plane distribution.

GRE header: In GRE header there are two fields that the NVE should set specifically for NVGRE protocol. One of them is VSID to identify which virtual layer 2 network the customer belongs to, and the flow ID. Flow ID is an 8 bits value and requires additional hardware capability on NVEs. This field is mainly used in multi pathing deployment and routers and switches can add a flow based entropy by parsing this header and use it with VSID field together. This flow ID should not be changed by the transit devices and if the provider decides not to use this feature, NVEs should set this field to all zeros.

The inner header is what the customer sends, so the addresses are CAs. The only thing that NVEs should do about the inner header is that the encapsulating NVE should remove any existing 802.1q

tag prior the encapsulation. So if the egress NVE (de-encapsulating device) receives a packet that its inner Ethernet frame contains an 802.1q tag, the egress NVE will drop it. The egress NVE, which de-encapsulates the packet, should also be a NVGRE-aware device to perform required processing to parse GRE to get access to tenant specification information.

The positive thing about NVGRE is that many existing hardware support the protocol but unlike with UDP, wrapping layer 2 packets within a GRE layer is not something that regular devices such as firewalls and load balancers can do. So they need to act as a gateway and remove GRE header to inspect inside the packets.

2.2.3 Stateless Transport Tunneling (STT)

STT [17] is proposed by Nicira and is implemented from software centric view rather than other overlay protocols that have been discussed are deployed from a network centric view. This protocol uses Ethernet over *TCP-like* over IP tunnel. To have more security, IPSec also can be used instead of IP. The STT tunnel end points are generally inside the end systems like vSwitches. The main differentiator of STT from other overlay protocols, such as VXLAN, NVGRE and TRILL is that it is designed for handling large 64KB data blocks. So it is not optimized for transferring small packets, but for relatively large storage blocks. Most of other overlay protocols, such as VXLAN, use UDP and they don't allow fragmentation. But with STT, the fragmentation is allowed because of large data chunks.

As mentioned, STT is an overlay encapsulation schema over layer 3 networks which uses a TCP-like header inside the IP header. TCP-like means STT uses identical header to TCP and even the protocol number is same as TCP (number 6). But on the other hand, most of the protocol features are different from TCP. As an example, STT tunnel is stateless and there is no 3 way handshake, connection, windows, retransmission of lost or destroyed packets and congestion state. As a result, the protocol acts more like UDP. The main reason of using TCP is that most NICs have hardware features that implement TCP functionalities such as segmentation and reassembly. So using hardware implemented TCP will be faster than UDP. These hardware implemented functionalities for TCP include:

- LSO (Large Send Offload): Each host gives a large chunk of data to NIC to send it to the network. NIC makes MSS size segments, adds checksum, TCP, IP and MAC headers to each segment of data.
- LRO (Large Receive Offload): On receiver side, if the NIC has the LRO feature, it attempts to reassemble multiple TCP segments and pass larger pieces to the host. So that the host does the final reassembly with fewer per packet operation.

STT uses these features to allow frame fragmentation with appropriate TCP, IP and MAC headers and also reassembly of these segments on the receiver side. The result is having large data size and so less overhead per payload byte.

Same as VXLAN and NVGRE, broadcast, multicast and unknown destination packets are handled by IP multicast. So it is possible to establish point to multipoint STT tunnels by setting an IP multicast as the destination address of the tunnel. Note that the physical underlying network should support control mechanisms and IP multicast group.

Similar to overlay protocols discussed earlier, STT has a virtual network Identifier, which is used to deliver the packets to correct virtual network segment. This is called context Identification and includes 64 bits tunnel end point identifier which provides a large space to address different service models.

STT has done some optimizations including alignment which is adding 2 bytes of padding to Ethernet frames to make their size multiple of 32 bits. The other optimization is that same as VXLAN, the source port is a hash of the inner header which can be used to have ECMP by letting each flow to take a different path. The result will be having all packets of a flow taking one path, so all packets will deliver in order.

The STT frame format is shown in figure 18 and 19. The flag field includes two important fields. One of them is checksum verified and it sets to 1 means checksum covers entire payload and is valid and the other is checksum partial and if it is 1 it means checksum only includes TCP/IP header.

L4 offset: This field offset from start of the STT header to the start of encapsulated layer 4 or TCP/IP header and help to quickly locate where the payload begins.

STT header includes a 16 bit MSS field. So the Maximum Segment Size is 64KB.

As mentioned STT uses TCP header. The fields of TCP headers are shown in figure 19. Destination port of STT, which is recently assigned by IANA is 7471. Source address is a random hash on ports and addresses of the TCP headers of the flow in virtual segment.

Sequence number and Ack number are two fields which are reused by STT for purposes other than what they are designed for in original TCP. Sequence number which is a 32 bits field is divided into two 16 bits field. The first 16 bits determines the length of the STT frame and the second 16 bits uses as offset of the current fragmentation within larger STT frame.

Ack number is reused as packet ID for fragmentation purpose. This number should be same in all segments of a payload. This field is used for reassembly in receiver side.

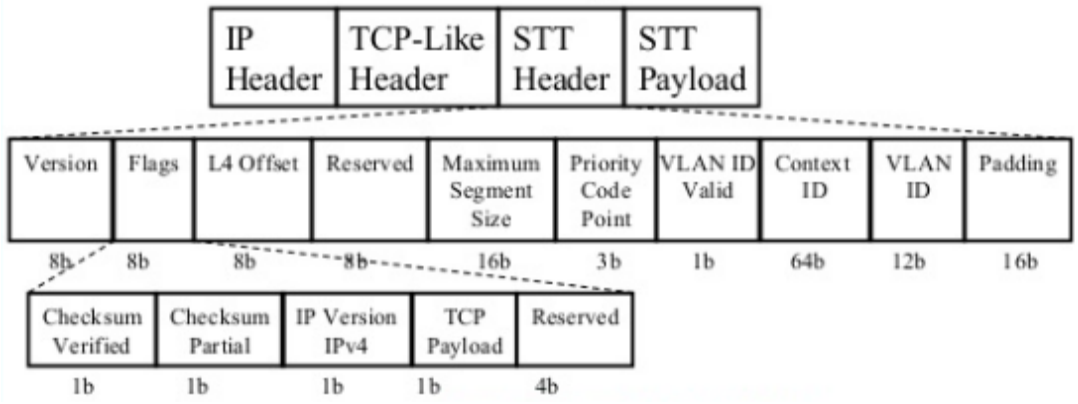


Figure 18 STT header format. (Image from https://www.slideshare.net/rjain51/m-13dmt)

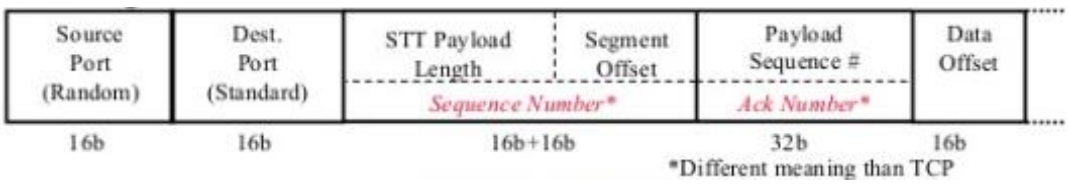


Figure 19 TCP-Like header format for STT (Image from https://www.slideshare.net/rjain51/m-13dmt)

Overall, STT has some advantages over other host based overlay protocols. It solves the problem of efficient transport of large chunks of data. In addition by using offload engine which has been implemented in most of today's NICs, it significantly reduces the workload of CPU on hosts, especially in high bandwidth systems.

3. SDN solution for network virtualization

3.1 SDN technology overview

The main goal of SDN is making an open and programmable network. If an organization requires a specific type of network behavior, it can develop or install an application to do what needs to be done. These applications may be for common networking functions such as traffic engineering and security, or for functionalities that have not been thought of today but might be introduced in the future.

SDN uses a model which is relatively similar to the operating system model but instead of operating system, the middle layer would be the Network Operating System (NOS), as it is shown in figure 20. This is commonly called SDN controller.

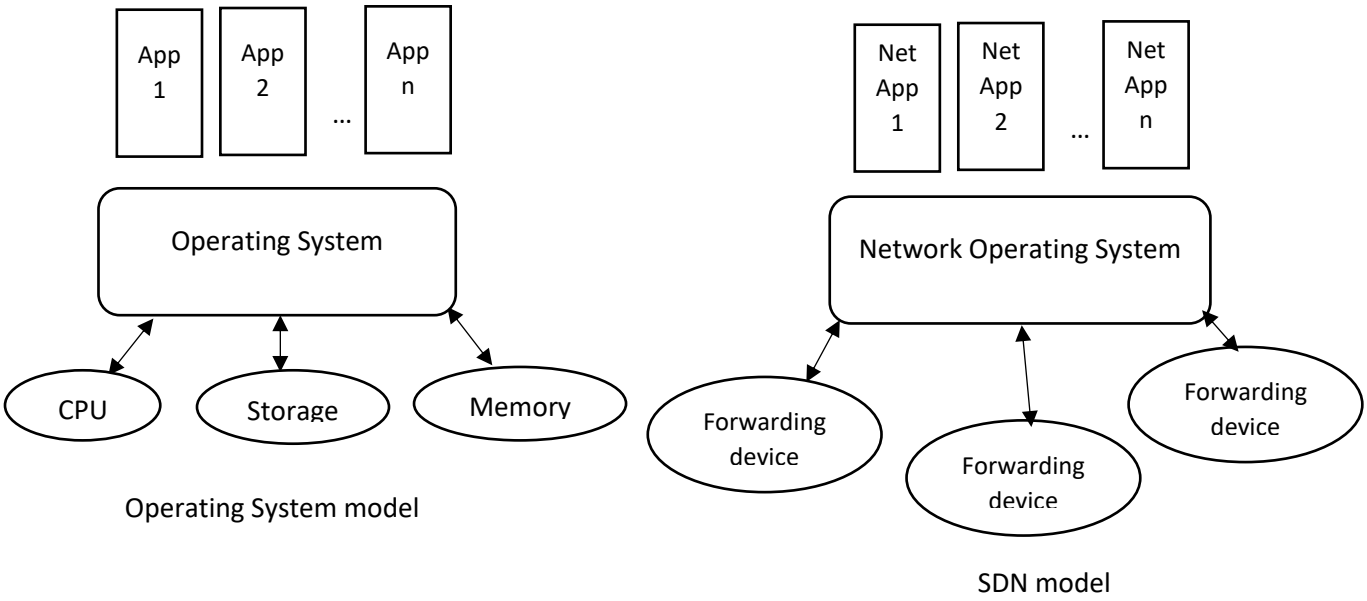


Figure 20 SDN model comparing to operating system model

The NOS typically has core services to aid it in its job of interfacing with network nodes and for providing a programmable interface to network applications. Below this layer, instead of hardware which is used for various purposes such as processing, storing data and memory in operating system model (figure 20), there are networking devices. Forwarding devices receive packets, take action on those packets and update counters. Types of actions that are expected from forwarding devices include simply dropping the packet, modifying packet header and sending packets out of a single port or multiple ports. The instructions for how to handle packets are received from SDN controller. Same as operating system model, we have applications at the top of the model, which we can call them network applications [18].

Packet flow according to this model is as follows:

When a packet arrives at a SDN controlled forwarding device, the forwarding device will parse the packet headers and it either knows what to do with the packet or it queries NOS, if it doesn't have any instruction for handling the packet. The network applications determine what action needs to be taken on each flow of packets and push this information down to the forwarding devices. So, the SDN controller acts as a translator. The forwarding device will then take the

assigned action on the packet, whether that's modifying header, dropping it, tunneling the packet somewhere and so on. The forwarding device also cache instructions, so that it doesn't require checking with the SDN controller for the future packets belonging to the same flow. The same process continues device by device until the packet leaves the SDN network to a destination in a different SDN region or into a traditional non SDN network. The instructions stored in forwarding device per packet flow will last until flow entries on forwarding device expire due to timers running out.

To sum up, there is a logically centralized network operating system which is called SDN controller. The controller has the global view of all network forwarding devices and delivers packet forwarding instructions to them. The controller can then create an abstraction or simplified view of the network for the network applications, which use this information to make key decisions about how to implement network policies.

3.2 SDN network architecture

SDN architecture can be considered as a composition of different layers, as shown in figure 21. In this architecture, some layers are always present in SDN deployment, such as the southbound API, NOS, northbound API, and network applications. There is additional layer in this architecture that may be present only in some special deployments. This layer is hypervisor or language based virtualization. The following section introduces each layer, by focusing on hypervisor layer, which is the main layer that enables network virtualization.

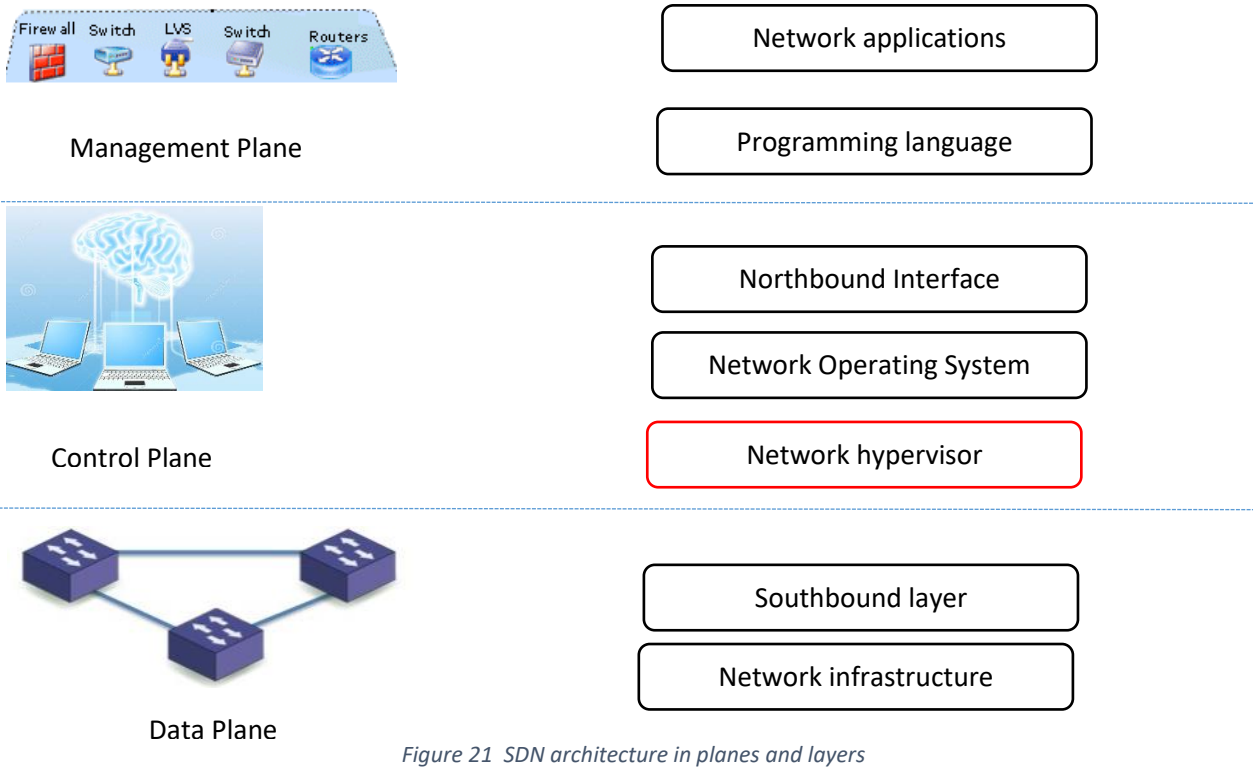


Figure 21 SDN architecture in planes and layers

3.2.1 Data Plane

This layer mainly includes networking devices, which are used for forwarding data efficiently. Forwarding devices can be hardware switches that support a programmable interface, such as open Flow, or software switches such as Open vSwitch[1]. One of the main difference between traditional networks and SDN, is that in SDN, control functionalities and data plane are decoupled. It means control functionality is removed from network devices, so that they act as simple forwarding devices and just perform a set of elementary operations. These forwarding devices have a set of instruction sets (e.g. flow table), which are used to take action on incoming packets. These instructions can result in actions such as forwarding packets to a specific port, dropping packets, forwarding to the controller, re-write some parameters in header, etc. These instructions are defined by south bound interfaces and installed in forwarding devices by SDN controller, which implements the southbound protocols. Southbound interface is part of southbound API and defines the communication protocol between forwarding device and control plane. Types of information that needs to be communicated includes packet handling instructions, alerts of packet arrivals on network devices, notification of status changes (for example a link may go up or down) and providing statistical information (e.g. flow counters). All of these happens on southbound interface. Some examples of south bound interface are Open Flow, which is the most widespread design of data plane devices, OVSDB, NETCONF and SNMP.

3.2.2 Control Plane

The most important part of control plane is Network Operating System, which is commonly referred as SDN-controller. Controllers typically have some core services, which implement the main part of their functionalities. These services include:

- Topology service: This service determines how network devices are connected to one another and builds the topology graph of the network. This topology graph can be built by instructing each switch to send LLDP[2], which allows nodes in the network to advertise to other nodes their capabilities and neighbors [19].
- Inventory Service: This service is used to track all SDN enable devices and record basic information about them. This information can be the type and version of southbound interface that each SDN enable device use and the capabilities that support.
- Statistic service: This service can be employed for reading counter information, such as traffic counters, from forwarding devices and flow tables.
- Host tracker: A host tracker service discovers where IP addresses and MAC addresses of hosts are located on the network. This is done usually by intercepting packets in the network.

[1] Open vSwitch is an open source production designed to automate network operations through programmatic extension. It also supports standard management interfaces and protocols such as NetFlow, CLI, LACP, 802.1ag. It mainly designed to support distribution across multiple physical servers. (reference : openvswitch.org)

[2] Link Layer Discovery Protocol

Commonly an SDN controller is considered as a *logically centralized* entity. This is important because it means that controller shouldn't be physically centralized, since it results a single point of failure for the entire network. As a result we need to make sure that SDN controller is highly available and scalable. The common way to implement high availability (HA) for controller is using clustering or teaming. This method is based on having a cluster of systems that can load balance the workloads, instead of depending on a single system to do that. This method also provides scalability, since multiple systems can handle requests.
The SDN controller provides application interfaces, so that network applications can be implemented and deployed using these interfaces. The SDN controller should provide a simplified abstraction of the underlying network infrastructure. These interfaces can be plugins collocated with controllers and use a programming API based language such as JAVA.
Northbound interface, is another part of the control plane which is considered as an application interface and is commonly a RESTful interface which allows the use of standard http calls (such as GET,PUT,DELETE,PATHCH) directed towards controller, to control network operations and network device behaviors and gather information, collected by core services. Unlike southbound interface, northbound interface hasn't been standardized yet.
The other important component of control layer, which we are mainly focusing on is network hypervisor layer. The network hypervisor provides abstraction, simplification and APIs which is needed for creating complex SDN network services. Particularly, the network hypervisor can enable interconnection of different SDN providers together under a single interface or abstraction layer. The result of this simplification is that applications can establish end to end flows of packets without dealing with differences between SDN providers [30].
Network hypervisor has the main role to implement overlay SDN. This is a networking approach where the software or hardware based edge of the network is programmed to manage tunnels between hypervisors. In fact, control plane provides a logical overlay that uses the infrastructure of the network or the underlay as a transport network. This model of SDN usually follows a proactive method to install the overlay tunnels, which terminate at hypervisors or switches. Network hypervisor layer will be discussed in more detail in section 3.4.

3.2.3 Management Plane

This plane is the set of network applications that uses the API based programming language and functions that are provided by the northbound interface of control plane, to implement network control and operation logic of the network devices. These network applications include routing, firewall, load balancer and other functionalities and services that also exist in traditional networks. It defines the policies which are finally translated to southbound instructions that define the behavior of forwarding devices.
To sum up, *network policies are determined in the management plane, control plane enforces it and data plane executes it by forwarding data according to the policies.*

3.3 Comparing traditional networks with SDN based networks

Generally, traditional networks are made up of networking devices that have both data plane and control plane, contained in a single physical system. The data plane is responsible for handling

incoming packets based on information that stored in their forwarding tables. These tables are different based on protocols and type of forwarding (routing, switching, labeling) that each node uses to forward each packet or flow of packets. For example, each node can have FIB, LFIB and MAC address table. The instructions, which are executed in data plane, are originated in control plane. In addition, each network device in a traditional network uses control plane in order to communicate with other network nodes in the network by running distributed protocols such as MPLS, BGP, IS-IS , etc. So, the control plane is the brain of each node, which handles complexity and defines network policies. By considering mentioned features as the characteristics of a traditional, non-SDN network, traditional network nodes are proprietary locked boxes that have both data plane and control plane connected to each other and there is no way to direct access into the data plane to define new behaviors that hasn't been programmed in the node before. So, in order to implement network policies, network operators should think in terms of what is already available in the control plane. Network operators often configure network devices via command line interface (or rarely through Graphical User Interface) *individually*. As a result, network operators are not only dealing with single point of configuration, but rather should configure tens or even hundreds of networking devices in a data center that each might have different commands and ways of configuration according to their vendors. Each of these nodes have control planes, which must communicate with each other using distributed protocols, which is complex and can lead to errors and mismatches.

In SDN network, there is a logically centralized controller, with a global view of the whole network. The result of this centralization is reduction of complexities of handling distributed protocols and configuring policies on large number of network nodes. In addition, the separation of control and data plane changes the nature of networking devices, including routers and switches, to simple forwarders; plus control logic implemented in control plane simplifies policy enforcement, reconfiguration and evolution.

3.4 Network hypervisor layer and Overlay SDN technologies

In network virtualization, different protocols and technologies try to hide the underlying details of provider's network from each tenant. This task is accomplished by network hypervisor layer in SDN based networks. It hides the details of SDN provider's underlying network from the applications. It also can provide a sets of APIs that makes creation and initialization of network topologies and applications simpler, so that SDN applications do not have to build the whole network from the scratch [18]. Using network hypervisor, each tenant can have end-to-end networks which are programmable and can cross multiple SDN providers without having to deal with details of each one.

In order to provide virtualization to a network, the protocol should bring the following characteristics to the network:

1) The underlying network should support arbitrary network topologies and addressing space and methods. This means different workloads need different network topologies and functionalities. In addition, address space should be changeable and the addressing scheme shouldn't be fixed.

2) Each tenant should be able to change the configuration of their network, just like what they can do with virtual servers (Virtual Machines).

3) Virtual machine migration should automatically trigger the migration of the corresponding virtual network ports.
Different protocols are trying to address these issues in different ways, all by leveraging the advances of SDN. Their methods can be categorized in two types of approaches [18]: slicing the network, Commercial multi-tenant network hypervisors

3.4.1 Slicing the Network

In this method, a hypervisor establishes multiple virtual SDN networks (vSDN) and each vSDN corresponds to a "slice" of the whole network. The main requirement for slicing the network is to classify packets into flows, which is mostly done using logical separators within the packet header. As a result, a controller may consider packet ID as the identifier of each slice. Some southbound protocols such as Open Flow also support using MPLS labels or VLAN as the identifier.
FlowVisor [20] is one of the early technologies, used to virtualize an SDN. It slices the communication between data plane and control plane, and sits between them, in order to provide a way for a single data plane to be controlled by multiple control planes, each may belong to a separate tenant. FlowVisor acts as a proxy controller and can be used to add a level of network virtualization to the Open Flow networks and allows multiple controllers to control overlapping sets of forwarding devices simultaneously. It also can be considered as an Open Flow controller, which plays the role of a transparent proxy between forwarding devices on one side and multiple open Flow controllers on the other side, as shown in figure 22.

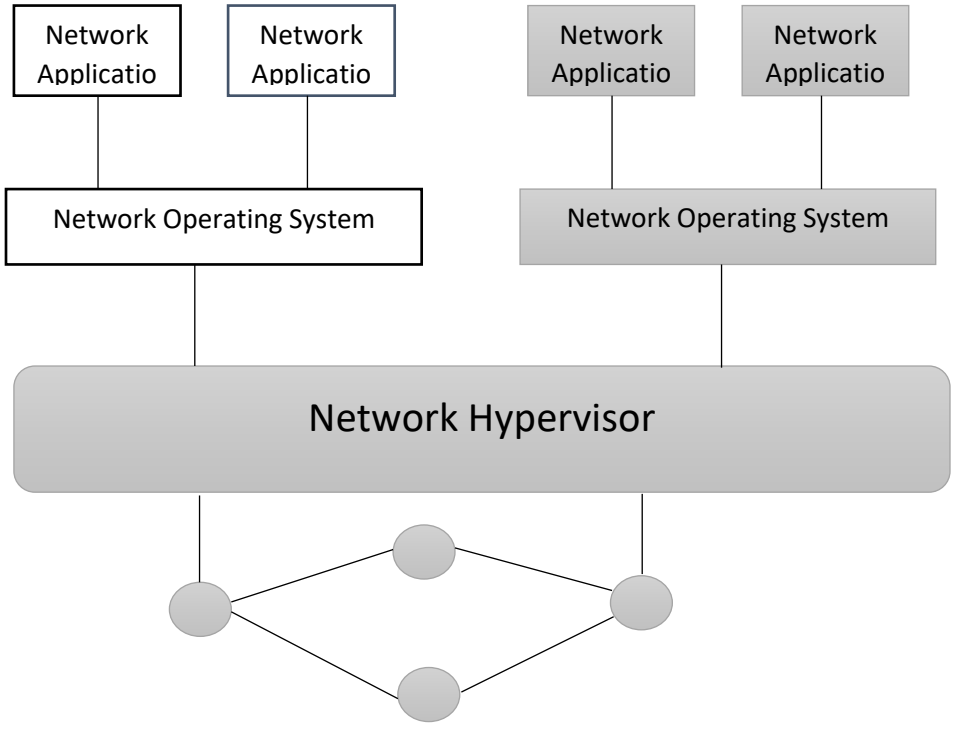


Figure 22 FlowVisor as a network slicer

FlowVisor ensures that each controller only access the switches and resources allocated to it. In addition, it introduces the term “flow space” for the set of packets that each slice controls. FlowVisor allocates each tenant its own flow space and makes sure that the flow spaces of tenants do not overlap. It also divides bandwidth and flow table resources on each switch and assigns those parts to each controller. FlowVisor slices a physical network into abstracted units of bandwidth, topology, forwarding table and device CPUs. Slicing of these resources is done as follows:

- Topology: each slice has its own view of the network, including network devices (nodes) and their interconnections.
- Bandwidth: Each slice has its own segment on each link, which prevent other slices to be affected in the case of one slice’s failure.
- Network device CPU: Each device has its own fraction of CPU on each forwarding device to use.
- Forwarding tables: Each slice has its own share of forwarding rules, which is finite, due to the fact that network devices support finite number of entries as forwarding rules.

With FlowVisor, each tenant (vSDN) runs their own slice of the network. To illustrate how slicing takes place by FlowVisor, imagine Tenant A as a customer that needs to deploy a network on an SDN-based environment. Tenant A begins by requesting a network slice from provider. The request includes the requirements of topology, bandwidth and the set of traffic which is defined by a set of flow (flow space). As a result, tenant A has its own controller, which it can define control logic that specifies how packets are forwarded and rewritten in its own space. So if the tenant wants to create a load balancer to distribute http traffic over multiple web servers, the slice that should be requested for this purpose should include the topology, that contains the web servers, and its flow space, that includes all flows with port 80. The tenant has the control plane which it can control load balancing logic and if the tenant wants to present its service to other users (be a provider itself), each user will be considered as a subset of tenant A’s flow space.
The policies which can be used to slice the SDN network can be different. The way these policies can be controlled is using another intermediate layer called Flow space Slicing Policy (FSP) [22]. FSP adds three alternatives for slicing: domain-wide slicing, switch-wide slicing and port-wide slicing.

- Domain-wide slicing: Each slice is identified with a unique logical identifier per domain, such as VLAN-ID
- Switch-wide slicing: Each slice is identified using multiple separators, which also include identification of forwarding devices that it spans, such as <VLAN-ID, switch-ID>
- Port-wide slicing: Each slice is identified using specific port of forwarding elements, such as <VLAN-ID, switch-ID, switch port-ID>

According to the chosen slicing policy, FSP engine translates the requests into multiple isolated flow spaces. These three methods have different performance and requirements mainly in terms of acceptance ratio, required hypervisor memory, required flow table size, and additional latency added by the hypervisor. The port-wide slicing method is more efficient in terms of tenant request acceptance ratio, means it can accept the most virtual network demands, while requiring the most resources.

FlowVisor lacks some of the basic network management interfaces. As an example, it doesn't have command line interface or web based administration console. Instead, users can make changes to the configuration using configuration file update. In addition, since it sits as an additional component between SDN controllers and forwarding layer, it adds latency overhead to the tenant control operations [20].

Finally, FlowVisor has a vulnerability called flow space problem [21]. In FlowVisor, the physical network's addressing space is sliced by flow space structure that indicates which value of each header fields belong to a given network. This prevents hypervisor from offering the entire Open Flow header field's space to the tenants. OpenVirteX [22] is a protocol which address this problem and provides each tenant with the full header space. Same as FlowVisor, OpenVirteX acts as a proxy between the Network Operating System (control layer) and the forwarding devices. But it also provides vSDNs through topology, address and control function virtualization. OpenVirteX does this by allowing each tenant to use their own NOS to control the network resources corresponding to their virtual network. As a result, unlike FlowVisor that simply slices the whole flow space between different tenants, OpenViteX provides a virtualized network with topology and full header space specific to each tenant. These features are needed in a multitenant environment and the way OpenVirteX achieve this is by placing edge switches at the border of underlying SDN network. The edge switches are responsible for mapping the virtually assigned IP and MAC addresses, which are used by hosts of each tenant (vSDN), to the addresses which are to be used within the physical network (as showed in figure 23). So tenants are able to manage their address space without depending on the underlying SDN network addressing scheme and the hypervisor ensures that the correct mapping has been stored in the edge switches. Since the OpenVirteX knows all the mapping of virtual to physical network, instead of physical SDN switches, it is this entity that responds to LLDP controller messages.

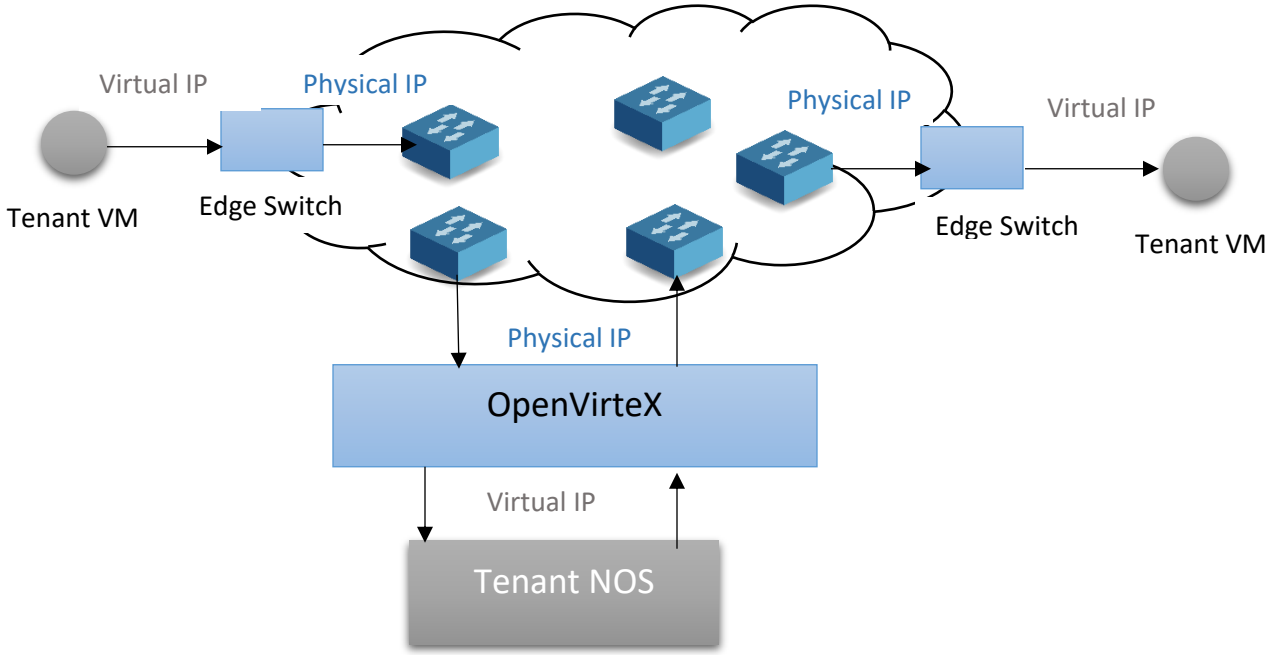


Figure 23 Address space Virtualization process in OpenVirteX

AutoSlice [24] is another proposal for virtualizing SDN-based underlying network. The main purpose of AutoSlice is to automate and deploy SDN slices that are built on the top of physical shared physical shared infrastructure, so that providers can sell or even resell vSDNs to tenants while minimizing the operator intervention. In addition, unlike other discussed methods such as FlowVisor, which has logical centralized hypervisor, AutoSlice tries to improve scalability of the hypervisor design by distributing the hypervisor workload. The process of assigning vSDN to each tenant begins with tenant's vSDN request, which includes a set of network's nodes and links with certain characteristics, such as device capacity and link bandwidth. The main feature of AutoSlice is that the hypervisor is distributed, in order to simplify handling flow table control messages received by multiple tenants. The components of AutoSlice are a management module (MM) and multiple distributed Controller Proxies (CPX) for handling control load. Each CPX is dedicated to an SDN domain, which is a segment of physical network (Figure 24).

When MM receives a request, it maps the vSDN topology to the available resources in each SDN domain and assigns these virtual resources to the proxies (CPX). Each CPX stores the resource assignment and translate the messages that are exchanged between tenant's network controller (vSDN controller) and actual forwarding devices. So all control communication between tenant's controller and forwarding plane is redirected by the CPX that is responsible for the domain.

The main challenge of AutoSlice design is that a large number of logical flow tables must be mapped onto a single Open Flow switch. CPX has the responsibility of isolating virtual flow tables and also making sure that all packet processing are applied in correct sequence; especially when a connected group of virtual nodes is mapped to the same switch (this is done by using loopback interfaces). Since the size of flow table for switches are limited, they may not be scalable enough to use. As a result, AutoSlice deploys Auxiliary Software Data paths (ASD), running on software switches like open vSwitch, which are deployed on commodity servers. In order to differentiate flow table entries of vSDNs on switches, AutoSlice uses an identifier, defined using VLAN tagging, named Virtual Flow table Identifier (VTID).

AutoSlice enables node and link migration. This migration happens in the case of link failure or vSDN topology changes and AutoSlice migrates the flow table and affected network traffic between switches in the correct update sequence. On the other hand, AutoSlice studies [24] do not provide any evolution and prototype implementation for the technology.

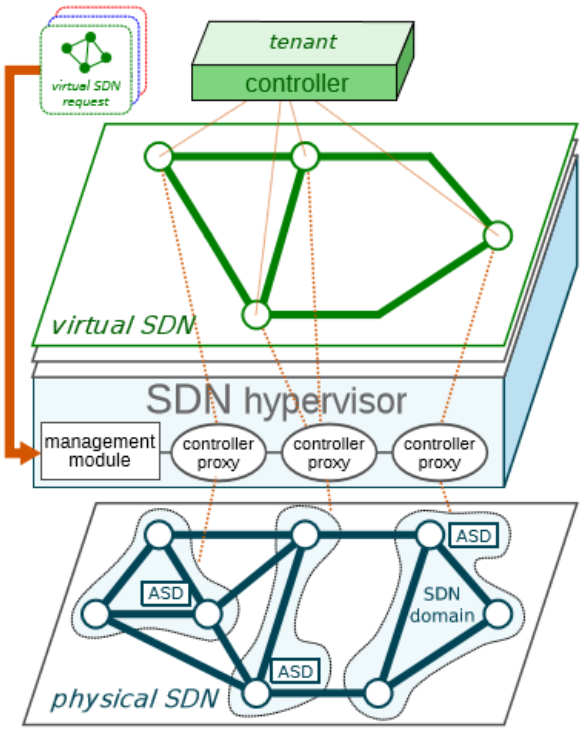


Figure 24 AutoSlice architecture [24]

FlowN [25] is another method for SDN-based network virtualization, which is based on a slightly different concept. FlowN is based on abstraction concept, which is different from virtualization. In fact, virtualization is a way that makes sharing of the physical infrastructure possible, whereas the abstraction specifies how the tenants use the network. FlowN defines a virtual layer and an abstraction layer, in a multi-tenant environment, which enable each tenant to be independent and control the entire address space, control all paths in its network and have an arbitrary topology.
For virtual layer, each tenant should be able to customize its own topology, as well as SDN controller. Each virtual topology consist of virtual nodes and virtual links. Virtual nodes can be servers or SDN-based switches (which as mentioned before, are limited by the number of flow table entries). Virtual links connect virtual nodes together and have a capacity constraint attached to them, which is related to required bandwidth and latency. Addressing of virtual interfaces is governed by each tenant independent from physical network.
The other important element that has been considered in FlowN is providing tenants with ability to customize the SDN controller. This means that the tenant can choose between already developed controller and the controller which is coded completely by tenant for having full control over their network. So even If the tenant chooses the latter option, it can use FlowN with the minimal changes to the controller code, just for interacting with the network to go through the virtualization layer.
Since tenants should manage their own address space and are not aware of physical IP addresses, FlowN uses encapsulation, in order to map the virtual addresses (managing by each tenant) into physical address space, transparently to the host and controller applications. In this encapsulation, instead of encapsulating each packet at the edge of the network with a unique header for each tenant, FlowN assigns a virtual address space to each physical link and as a packet enters the

network, each switch performs label swapping within the header. The result of this approach is that each packet received by a switch can be categorized based on: 1) physical interface, 2) label in the encapsulation header and 3) fields that are specified by tenant's application.
For bandwidth isolation, FlowN depends on Open Flow capabilities.
The process of assigning a virtual topology to a tenant starts with passing the specification to a topology parser. The parser uses the provided specifications to make a view of the virtual network; and finally, virtual resources will be mapped to available physical resources using a special algorithm called embedder.
In addition, FlowN uses an additional data base component, in order to provide a consistent mapping. Each virtual topology is uniquely identified by some key, and consists of nodes, interfaces and links. As shown in Figure 25, each virtual node is mapped to a physical node, each virtual link becomes a path, which is a collection of physical links, in addition to a hop counter for ordering physical links. To improve the scalability, FlowN uses master-slave database principle, means, the state of master database is replicated among multiple slave databases. The result is that FlowN can be distributed among multiple physical servers and each physical server can be equipped with a database and a controller.
Apart from using database for mapping look up, FlowN has another feature which makes it different from other discussed SDN-based virtualization methods, such as FlowVisor. Unlike thoses methods, which used separate controllers or controller proxies to communicate with tenants' applications, FlowN extends the capabilities of the controller platform to respond to multiple tenants' applications calls. So when FlowN receives a packet from a SDN switch, it performs a lookup to determine which tenant the packet belongs to. FlowN then *calls the appropriate function* in the tenant's controller application. In fact, FlowN isolates different hypervisor instances (from process handling perspective) by assigning a processing thread to each tenant (virtual topology). This is why FlowN is analogous to a container based virtualization, which is a light weight virtualization approach, compared to a full virtualized technology. This results lower resource consumption and being able to isolate controller logic.

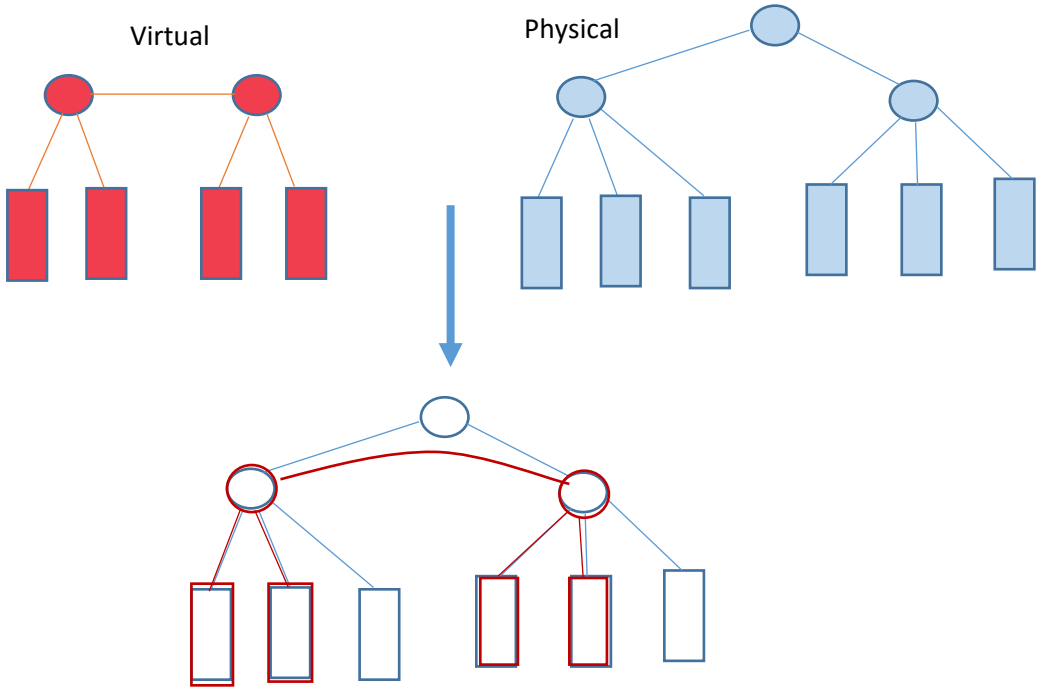


Figure 25 database driven mapping in FlowN

All the protocols mentioned before are focused on slicing underlying network into disjoint parts, in order to enable tenants to control their vSDN using their own control entities. As a result, previous hypervisors only restrict each controller to a specific slice of network traffic. Although this is necessary for multi-tenant environment, they do not bring the possibility of multiple applications to process the same traffic.

CoVisor (Compositional Hypervisor) was designed with different sets of goals. CoVisor [26] is a hypervisor that allows the deployment of multiple control applications, implemented with different programming languages and operating on different controller platforms to exist in a single SDN-based network. CoVisor also has an abstraction mechanism, in order to provide a vSDN controller with only necessary topology information.

CoVisor brings together the following three hypervisor features and builds a single system by combining all of these features:

1) Assembly of multiple controllers: CoVisor is independent of any programming languages, libraries or controller platforms, used to build client applications. The reason is CoVisor intercepts Open Flow messages from application controllers and change them in a way that makes them match composition policies (or rules in a simpler concept) defined by network administrator.

2) Defining abstract topologies: The provider of physical underlying network should be able to have control over what each controller can see from the underlying topology. So CoVisor allows providing a custom virtual topology to each controller and each application. This is an important feature because for instance, a firewall application does not need the detail of underlying topology to decide whether each packet should be dropped or forwarded. On the other hand, another

application like router needs to know the details of underlying topology to perform routing efficiently. In order to make this happen, the hypervisor can create *one big virtual switch* for each role and assign it to the related application and then compile rules that have been deployed for the virtual networks to the physical network.

3) Protecting against misbehaving controllers: In addition to restricting the view of network for each controller, it is important to have control on how a controller can process packets. This means by having different third party controllers, provider should be able to limit the access in the case of having malicious or misbehaving controller.

As mentioned different applications need different view of network, and also different levels of accessing and changing in the network. For instance, a firewall controller most of the time does not need to be allowed to modify packets; and this is hypervisor's responsibility to make sure of it.

In CoVisor, the policies which are prioritized Open Flow rules from multiple network applications running on different vSDN controllers, are compiled to a single composed policy to sit on a flow table. To implement this, CoVisor performs two compilation processes: In first phase, it assembles the policies of each controller, which are implemented in their own virtual network, into a composed rule for the whole virtual network. Second phase is to compile the rule or policy of the whole virtual network, into a policy for the hypervisor network.

Using policy composition, CoVisor allows the administrator to compile data plane policies in the following manners:

Parallel (+): The parallel composition of two policies P1 + P2, logically copies each corresponding packet, applying P1 to one copy and P2 to another copy.

Sequential (>>): The sequential composition P1>>P2 operates by applying policy P2 to each of the packets output by applying policy P2 to the incoming flow of packets.

Override (⊳): Each controller x, provides CoVisor with a member policy, determining how x wants the network to process the packet. As an example, x ⊳ P means the incoming packets first match with the policy specified by x, and if policy x does not determine how to handle incoming packets, then P is used as default.

In order to have a better understanding of how CoVisor works, let's consider figure 26 as an example. The first thing that should be considered is CoVisor takes the configuration from the administrator of the network, as it is shown in left side of figure 26. The physical topology represents an enterprise network which made up of an Ethernet portion (shown by blue) connected by a gateway (labeled as S) to the core network (shown by red). There is a physical-virtual topology mapping, which abstracts forwarding device S to three virtual switches: E, G and I. So, CoVisor allows administrator to create virtual topologies on top of the physical network (which is done by CoVisor's API). Each of the five network applications (MAC Learner, Firewall, Gateway, Monitor and IP Router) shown at the top of the picture is different SDN programs running on different controllers and each controller provides Open Flow rules for its own virtual network below it, without knowing that the virtual topology does not physically exist. CoVisor listens to the Open Flow rules output by all of the controllers and simplifies them using policy composition and compile them into a single policy for the enterprise physical network.

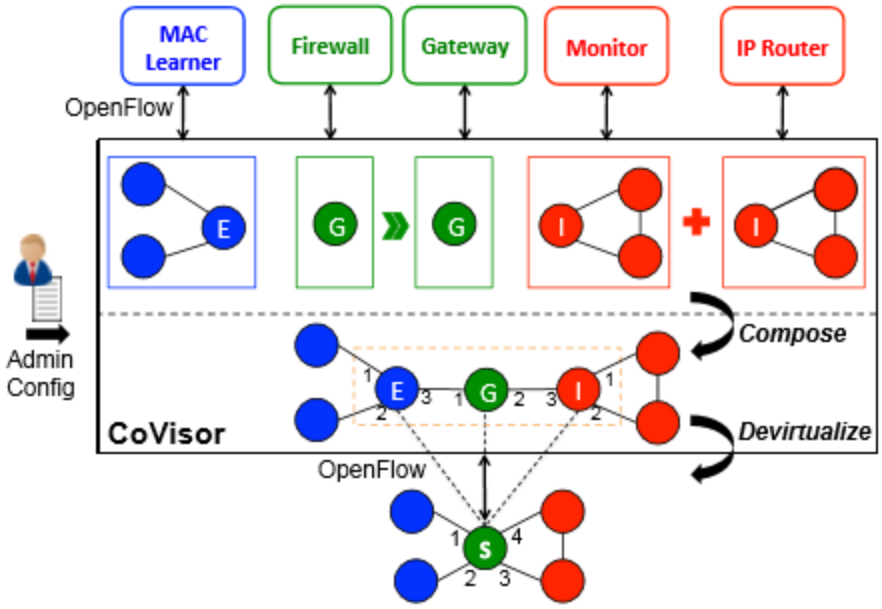


Figure 26 Example of a network that uses CoVisor as the hypervisor [26]

3.4.2 Commercial Network hypervisors for multi-tenant SDN-based networks

The protocols and method that have been discussed in section 3.4.1, cannot address all the challenges in multi-tenant networks. For instance, it is important for tenants to be able to migrate their whole network to a service provider (cloud provider) without the need to change the configuration of their enterprise network. Although discussed technologies try to address the requirements of a multi-tenant data center, the migration strategies that they represent have mostly failed to meet both tenant and service provider requirements at the same time [18]. In a multi-tenant environment, the hypervisor should abstracts the underlying network for each tenant, in addition to give them the access to control abstractions and manage their own vSDN, isolated from other tenants. So an SDN-based network needs a complete network virtualization solution that allows the creation of independent virtual networks for large scale multi-tenant environments. NVP (Network Virtualization Platform) [27] is one of those solutions, which is a commercial solution, since it has been deployed in various *production environment* over the past few years. NVP main objective is abstraction of data center network resources to be managed by tenants. The approach of NVP to network virtualization is that the provider should be able to replicate the arbitrary network environment that a particular application or tenant is expecting and allows to move the tenant's network into the cloud. NVP does not depend on the underlying physical network of provider, means it is just going to expect to see any standard layer 3 network (a set of servers with IP addresses connected to each other), and the virtualization will happen in software outside of that. The service (tenant's virtual network) and the physical network are brought together with network hypervisor layer. In fact, the intuition of NVP architecture is analogous to virtualization of servers, where we can take an arbitrary machine and run it on unmodified hardware, without the knowledge of Virtual Machine about where it is running. This is the virtualization layer that decouples virtual and physical environment. So what NVP is doing is

building a network virtualization layer that separates an arbitrary network service from an arbitrary physical network.

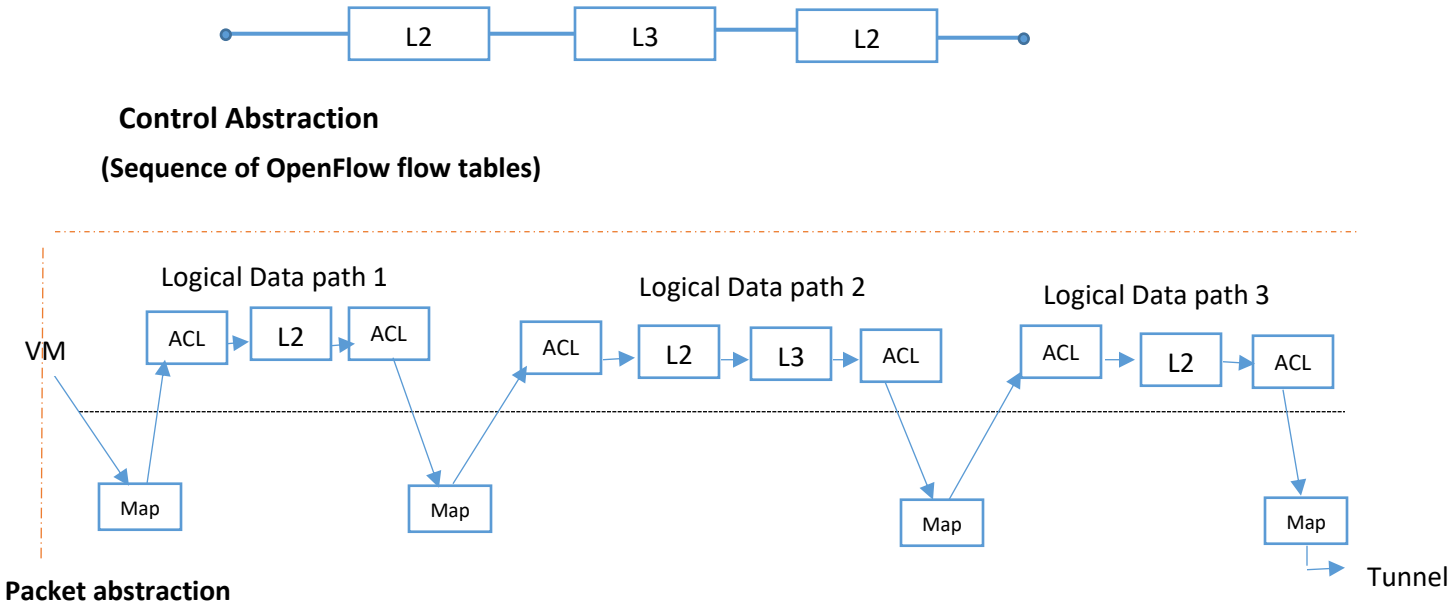


Figure 27 Example of NVP architecture (made up of a layer 2 switch connected to a layer 3 router which is connected to a layer 2 switch)

Consider a network consist of simple connection of a layer 2 switch to a layer 3 router connected to another layer 2 switch (as shown in figure 27). In NVP, this is modeled as a sequence of data path elements that represents the forwarding devices. Each one of these data path elements is an Open Flow forwarding table. That means the table will match on certain signature of packet headers and take certain resulting actions, such as modifying certain fields, forwarding or dropping the packet. So the idea of NVP is that *the switching and routing gear can be modeled with the right sequence of the right Open Flow tables and rules that administrator setup*. According to figure 27, there is a virtual machine (VM) that sends packets through what it thinks is its NIC. Each packet goes to the network virtualization layer that realizes this particular tenant VM is connected to the corresponding virtual network. The first hit to the tenant's virtual network is the layer 2 switch, represented by its logical data path. So the packet is going to make its way through each of the tables. It's going to be checked by the inbound Access Control List table (ACL) and then will be handed off to the layer 2 forwarding table, the outbound ACL and finally sent out of the first virtual network element, which is the layer 2 switch. The network hypervisor then realizes that network element is now connected to a logical layer 3 router. The packet again proceeds through the hops and finally will be mapped by hypervisor into the last vSwitch and then it reaches to the end of logical data paths. Now at this point packet has traveled through the software data plane and NVP has determined what the output virtual port of this particular packet is in the virtual network and which virtual machine it is destined for. So the packet will be tunneled to the final destination across the underlying layer 3 fabric. In fact, NVP provides two interfaces. There is a **packet abstraction** where virtual machines are able to inject traffic into the virtual network. The other interface is a **control abstraction** where the tenant is able to define entire virtual pipeline,

which is the sequence of Open Flow tables. This is the interface, which is given to tenants, so that the tenants can program their virtual network.

NVP sees the underlying physical network as a set of hosts and servers with IP addresses connected to the fabric. The servers are running on a particular tenant's virtual machine. Each tenant uses API at the network hypervisor controllers to define its service or virtual network. The controller then will instantiates the underlying software switching data plane (virtual switch or Open vSwitch), which is at virtualization plane of each of these servers on which the tenant's VM resides. Notice that an instance of tenant's virtual network is present at each one of those servers (figure 28), where tenant's VM resides. So when the VM injects packets into the virtual network, the packet will go through the local copy of virtual network at the server and will be processed through the pipeline, till NVP knows the destination, and then it gets tunneled, using the tunnels that we set up between each of tenant's VMs to the final destination.

There is some performance challenges that NVP should address them. At the controller side, there are large amount of states to compute. The reason is every hosts at the tenant's VM have a full virtual network state and a full instance of the virtual network, so that they can locally perform the virtual network processing and decide what the ultimate destination of the packet is. Also NVP sets up tunnels, connecting each one of the tenant's VM. So if a tenant has n VMs, each one of them should connect to n-1 other VMs (O (n^2) for tunnel computation). The solution is *automated incremental state computation with nlogn declarative language*. This means rather than computing total state from scratch at all times, there is going to be automatic incremental changes, using a declarative language developed by NVP (nlogn) to specify the data paths and automatically make incremental adaptions. In addition, the controller does not compute the low level forwarding table of every instances of the virtual network. Instead it computes a higher level of representation of virtual network, called universal flows. Universal flows is an instance that is good enough for any hypervisor, hosting the corresponding tenant's VM. Then it is sent out to the physical controllers, which more locally compute the universal flow abstraction, and translate it down to the exact Open Flow forwarding table entries at that particular host.

There are also performance challenges at each host. The data flow pipeline is processing entirely in software (Open vSwitch) and it can include many forwarding elements. This can be a slow procedure. The proposed solution is that NVP sends first packet of each flow through the full pipeline to perform computation. But subsequent packets of the same flow that should be handled in the same way can be cached in the kernel of the Operating System. This procedure is more efficient than complex matching operation that performs in Open Flow; because once the result that should apply to that particular packet become known, it can be cached with an exact match on a packet header in certain fields to produce the result.

The other challenge is with tunneling method. Recall that TCP segmentation Offload (TSO) allows the Operating System to offload the task of splitting up the data stream into smaller chunks and some of the check summing, related to TCP, to the NIC hardware. Using tunneling techniques such as GRE or VXLAN, results additional IP header for each packet. As a result, the NIC doesn't see TCP outer header where it expects. The solution is using STT (discussed in section 2.2.3) as the tunneling technique, which adds fake outer TCP header, so that NIC can perform TSO operation.

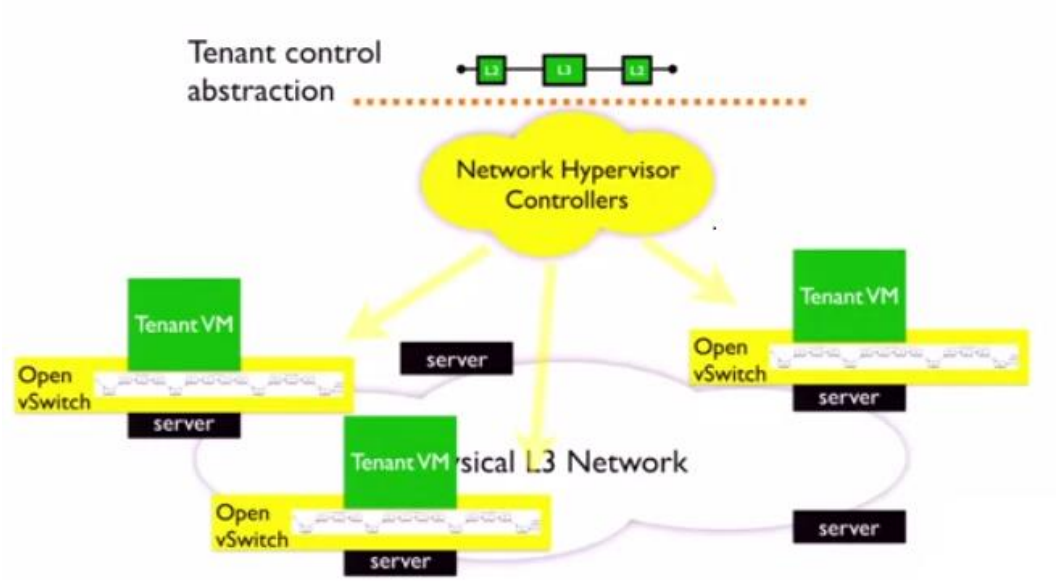


Figure 28 NVP implementation on underlying network

Another commercial solution for using SDN towards network virtualization is SDN VE [28], proposed by IBM. One of the main differences between this solution and previously discussed ones is that SDN VE is based on OpenDayLight, which offers a wider range of southbound APIs and/or protocol plug-ins. This protocol is an overlay based solution, which provides a complete implementation framework for network virtualization. SDN VE is similar to NVP in some aspects. One of the most important similarities is using host-based (also called server-centric) overlay technology for achieving network abstraction to enable application-level network services in a multi-tenant environment. SDN VE can be implemented on a network with sets of servers, connected in an IP based physical underlay. So like NVP, SDN only needs an IP based physical infrastructure. SDN VE is a multi-hypervisor approach that made up of four software components to provide host based network virtualization (figure 29):

1) SDN VE Virtual switch: This is the software that resides in hypervisor. This is the start and end point of each virtual network; and provides layer 2 and layer 3 network virtualization services, using a UDP overlay, which is mostly VXLAN. The virtual switch also provides control plane functionalities, such as virtual machine address auto discovery and VM migration, in addition to the policy configuration.
2) Connectivity Service: This is used to distribute VM addresses to vSwitches that are participating in an SDN VE virtual network.
3) Management console: This component enables administrator of the network to configure control policies and distribute the policies to vSwitches. This software component resides on a server and also can provide high availability. It can be used in active-standby mode, by enabling one instance to run in active mode and the other runs in standby mode.

4) VLAN and IP based gateway: This is SDN VE solution to establish interoperability with networks, which are external to SDN VE network. This gateway provides VLAN based gateway for layer 2 networks, and IP based gateway for layer 3 networks. This component also supports failover, in the case of an outage.

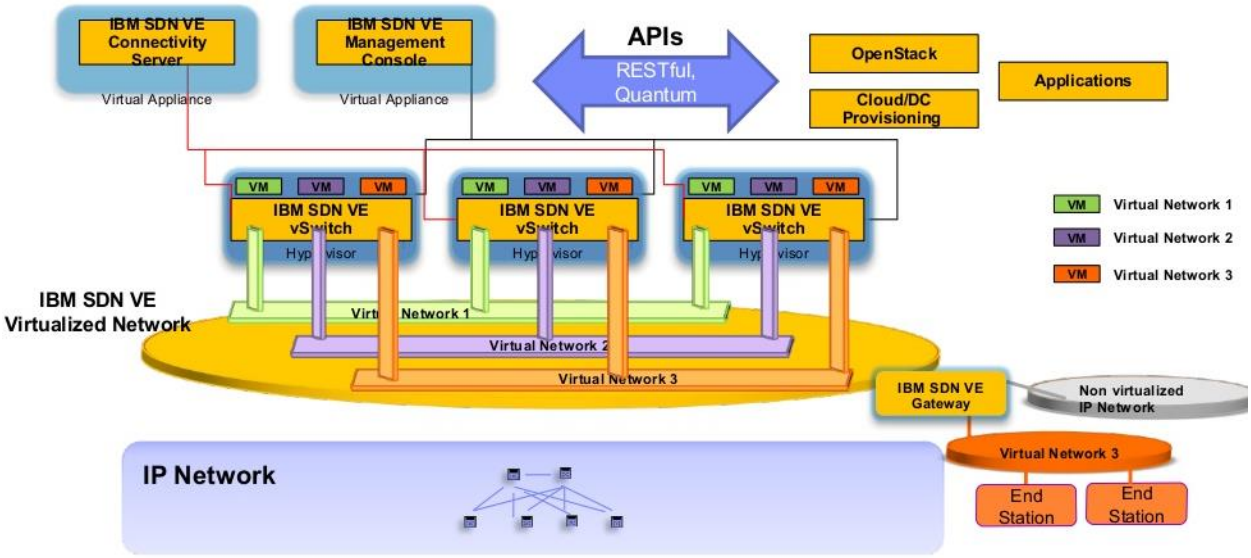


Figure 29 SDN VE software components (image from: https://www.slideshare.net/bladenet/ibm-sdn-ve-9-apr-2013-slideshare)

SDN VE adds a distinct header to packets, which are sent by VMs. So it works with ordinary IP packets, and underlying forwarding devices can operate normally without any changes. So unlike using traditional networks, which are limited to 4096 virtual networks (VLAN) and require end-to-end VLAN configuration on at least some of the physical devices, SDN VE can support up to 16 million virtual networks and each one can be managed using API. In addition, SDN VE provides simplicity in creating virtual networks by reusing existing IP addresses and preventing the address of each VM to get exposed to the physical network [29]. In fact, SDN VE addresses the challenge of overlapping virtual network addresses in data centers by tying the IP address to application licensing.

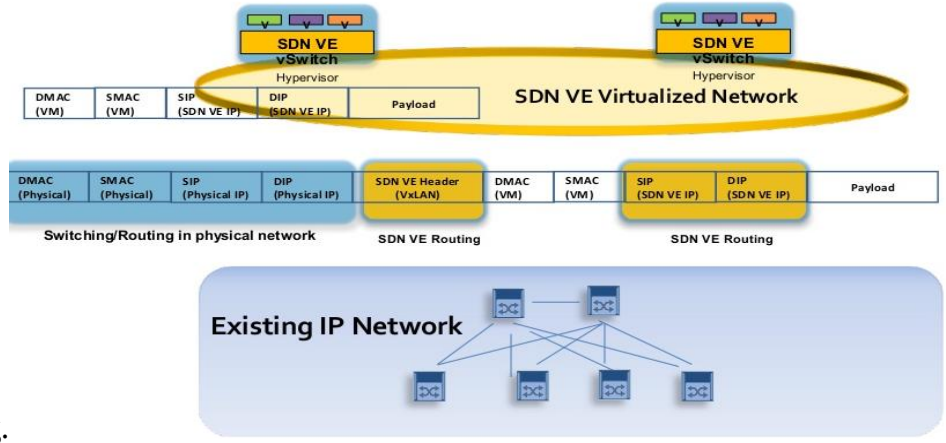


Figure 30 SDN VE packet structure (image from: https://www.slideshare.net/bladenet/ibm-sdn-ve-9-apr-2013-slideshare)

All the SDN-based network virtualization protocols follow the same goals:

- They all try to decouple virtual network topology from physical network topology
- They bring us the ability to create flexible networks that meet different requirements and variety of criteria
- They provide programmatic interface to dynamically create and deploy virtual networks
- From security perspective, they all make sure that there is enough isolation and separation between tenant's virtual networks

4. Comparing Overlay protocols and SDN-based approach, used to virtualize the network infrastructure

There are different competing approaches among protocols, come from overlay family. Although they all try to address the issues in multi-tenant environment and support dynamic movement of virtual machines, there are some criticisms that are applied to these types of protocols:
In section 2, eleven protocols from the overlay family have been discussed. Eight of these protocols are categorized as network-based overlays and three of them considered as host-based overlays. In all of these protocols, some sort of encapsulation or tunneling is used. This means each packet is tagged, which wraps around the message, before it is delivered to its destination. The process of encapsulation and de-encapsulation needs additional computing and processing. So it may cause scalability issues and adds higher level of complexity to the network. In addition, adding more layers of processing creates performance overhead.
SDN provides a new approach networking and also creating virtual networks. In SDN Open Flow is standard, which is an advantage, but how the controller implements network virtualization is not standardized. SDN makes network virtualization simpler and more flexible, because of the following reasons:

- Logically centralized controllers learn about multiple paths through the network between each source and destination, and can create multiple forwarding entries. So network switches can use multiple paths to deliver packets and traffic
- ECMP creation is relatively trivial task for a central controller, by having a general view of the network
- Having a centralized network view makes it possible to engineer end to end paths for each tenant traffic to follow, based on policies defined by a network engineer, such as latency and hop count
- Each application has the ability to request services and resources

A summary of comparison between overlay protocols and SDN-based approach is represented in table 5.

Table 5 Comparing overlay networks and SDN-based networks

Overlay networks	SDN-based approaches
Different competing approaches (discussed in section 1)	Open Flow is a standard protocol, implementing controller vary
Different degrees of maturity and completeness	Vendor specific maturity and completeness
Traditional approach	New approach
Modest current deployment	Very modest current deployment
Potential for added complexity	Potential to reduce complexity
Significant vendor support	Significant vendor support

5. Conclusion

There is significant need for network virtualization due to the multi-tenant environments in data centers and cloud providers. The approach that networks should move towards should be flexible, secure and cost effective. Different overlay methods for network virtualization have been discussed in this analysis. They are all a traditional approach to address multi-tenant network environment. SDN has the potential to fundamentally change how networking is implemented. It can provide network virtualization and other functionalities in less complex and more flexible way, but the price that should be paid is the fundamental change in how the network is designed, deployed and configured for service providers.

References

[1] Van der Lans, R. (2012). "*Data Virtualization for Business Intelligence Systems: Revolutionizing Data Integration for Data Warehouses"*. San Francisco. Morgan Kaufmann Publishers

[2] IEEE 802.1ad-2005, "*IEEE Standard for Local and Metropolitan Area Networks---Virtual Bridged Local Area Networks---Amendment 4: Provider Bridges"*, http://standards.ieee.org/findstds/standard/802.1ad-2005.html

[3] IEEE 802.1ah-2008, "*IEEE Standard for Local and metropolitan area networks -- Virtual Bridged Local Area Networks Amendment 7: Provider Backbone Bridges"*, https://standards.ieee.org/findstds/standard/802.1ah-2008.htmlb

[4] IEEE 802.1ah, "*Provider Backbone Bridging"*, https://infoproducts.alcatel-lucent.com/html/0_add-h-f/93-0076-10-01/7750_SR_OS_Services_Guide/services_PBB.html

[5] Cisco FabricPath Design Guide: Using FabricPath with an Aggregation and Access Topology, https://www.cisco.com/c/en/us/products/collateral/switches/nexus-5000-series-switches/guide_c07-690079.html

[6] Guidelines and Limitations for FabricPath Forwarding, https://www.cisco.com/en/US/docs/switches/datacenter/sw/5_x/nx-os/fabricpath/configuration/guide/fp_forwarding.html#wp18048811

[7] Eastlake, D., Banerjee, A., Dutt, D. "*Transparent Interconnection of Lots of Links (TRILL) Use of IS-IS"*. IETF RFC 6326 (2011)

[8] Touch, J., Perlman, R. "*Transparent interconnection of lots of links (TRILL): Problem and applicability statement*". IETF RFC 5556 (2009)

[9] IEEE 802.1aq-2012, *"Standard for Local and Metropolitan Area Networks: Virtual Bridges and Virtual Bridged Local Area Networks - Amendment 9: Shortest Path Bridging"*, https://standards.ieee.org/findstds/standard/802.1aq-2012.html

[10] Cisco Overlay Transport Virtualization Technology Introduction and Deployment Considerations, https://www.cisco.com/c/en/us/td/docs/solutions/Enterprise/Data_Center/DCI/whitepaper/DCI3_OTV_Intro/DCI_1.html

[11] Al-shawi, M. (2015), *"CCDE Study Guide"*, Cisco Press

[12] Farinacci, D., Fuller, V. *"The Locator/ID Separation Protocol (LISP)"*. IETF RFC 6830 (2006)

[13] Kampichler, W., Lindner, M., Haindl, B. *"LISP: The Novel Approach to the Future of Internet Architecture"*, DASC, 2013.

[14] Lasserre, M., Kompella, V., *"Virtual Private LAN Service (VPLS) Using Label Distribution Protocol (LDP) Signaling"*, IETF RFC 4762 (2007)

[15] Mahalingam, M., Dutt, D., Duda, K., *"Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks"*, IETF RFC 7348 (2014)

[16] Garg, P., Wang, Y., *"NVGRE: Network Virtualization Using Generic Routing Encapsulation"*, IETF RFC 7637 (2015)

[17] Davie, B., Gross, J., *"A Stateless Transport Tunneling Protocol for Network Virtualization (STT)"*, IETF draft-davie-stt-01 (2012)

[18] Kreutz, D., Ramos, F., Ver, P., Rothenberg, C., *"Software-Defined Networking: A Comprehensive Survey"*, Proceedings of the IEEE 103 (1): 63 (2015)

[19] Pakzad, F et al., "*Efficient topology discovery in OpenFlow-based software defined networks"*, Comput. Commun., vol. 77, pp. 52-61, Mar. 2016.

[20] R. Sherwood et al., *''Can the production network be the testbed?''*, in Proc. 9th USENIX Conf. Oper. Syst. Design Implement., 2010, pp. 1–6.

[21] Victor T. Costa, Luís Henrique, M. K. Costa, *"Vulnerabilities and solutions for isolation in FlowVisor-based virtual network environments"*, Journal of Internet Services and Applications, vol. 6, no. 1, 2015.

[22] Blenk, A., Basta, A., Reisslein, M., Kellerer, W., "*Survey on network virtualization hypervisors for software defined networking"*, IEEE Commun. Surveys Tuts., vol. 18, no. 1, pp. 655-685, 1st Quart 2016.

[23] Al-Shabibi, A., et al., "*OpenVirteX: A Network Hypervisor"*, 2014. [Online]. Available: http://ovx.onlab.us/wp-content/uploads/ 2014/04/ovx-ons14. pdf.

[24] Bozakov, Z., Papadimitriou, P., *"AutoSlice: Automated and scalable slicing for software-defined networks"*, in Proc. ACM Conf. CoNEXT Student Workshop, 2012, pp. 3–4.

[25] Drutskoy, D. A., *''Software-defined network virtualization with FlowN''*, Ph.D. dissertation, Dept. Comput. Sci., Princeton Univ., Princeton, NJ, USA, Jun. 2012.

[26] Jin, X., Gossels, J., Rexford, J., Walker, D., *"CoVisor: A compositional hypervisor for software-defined networks"*, Proc. USENIX Symp. NSDI, pp. 87-101, 2015.

[27] Koponen, T., et al., *''Network virtualization in multi-tenant datacenters''*, in Proc. 11th USENIX Symp. Netw. Syst. Design Implement., Apr. 2014, pp. 203–216.

[28] Racherla, S., et al., "*Implementing IBM Software Defined Network for Virtual environments"*, Durham, NC, USA: IBM RedBooks, May 2014.

[29] IBM Software Defined Network for Virtual Environments, IBM, 2013, Available: https://www.research.ibm.com/haifa/dept/stt/papers/QCW03028USEN.PDF

[30] Huang, S., Griffioen, J., Kenneth, L., *"Calvert. Network Hypervisors: Enhancing SDN Infras-tructure"*, Computer Communications on, pp. 87-96, 2014.