A TRILL-based multi-tenant data center network

Ahmed AMAMAOU, Kamel HADDADOU, Guy PUJOLLE

{*ahmed, kamel*}*@gandi.net, guy.pujolle@lip6.fr*

Abstract

IaaS Providers need next-generation virtualized data center networks, providing large-scale multi-tenancy services and meeting new requirements in terms of efficiency and availability. The main limitation of current network architectures mainly comes from inadequate design standards, such as those for Ethernet Bridging (802.1D) and Virtual LAN (802.1Q), which were not designed to handle large numbers of tenants and the resulting traffic volumes.

In this paper, we present a TRILL based multi-tenant network architecture for virtualized data center networks. Through the TRILL protocol, this architecture achieves the best of both layers (bridging and routing), thus building a large-scale layer-2 network while ensuring scalability, efficiency, fault-tolerance and simplified management. In addition, this architecture integrates the VNT (Virtual Network over TRILL) solution, enabling the creation of large numbers of logical networks within a virtualized data center. The complete implementation of our solution is currently deployed and running in the GANDI public cloud infrastructure.

Experimental evaluations show that VNT is able to support several hundreds of switches, thousands of physical nodes, and hundreds of thousands of tenants with only negligible overhead, all with very minimal configuration which may be completely automated. VNT additionally permits backward compatibility with both traditional layer-2 equipment as well as newer hardware supporting TRILL.

Keywords: layer 2 routing, Data center, Management, Private network

1. Introduction

The public cloud market should benefit from strong growth in 2013. This growth is estimated at more than 18% by the end of this year, according to Gartner[1]. Infrastructure as a Service (IaaS) is expected to be the most dynamic segment in this market: it has an expected growth of 43.7% in 2013 and will reach a total of \$9 billion. The enthusiasm for this market is driven by the multiple benefits to companies of reduced operating costs. IaaS solutions can help companies achieve greater operational effectiveness and performance. These solutions are designed to consolidate IT infrastructures and streamline business operations for optimal productivity. They also aim to reduce not only associated maintenance and management costs, but also costs for space, power, and cooling requirements associated with housing the infrastructure on premises. To achieve this, the cloud provider should deploy a modern Data Center (DC), or more precisely a Virtualized Data Center (VDC) capable of seamlessly incorporating cloud applications and new related services, with more agile management and simplified IT operations.

A Virtualized Data Center (VDC) is a data center where some of the hardware installations are virtualized. Concretely, a VDC is composed of several servers and storage systems, power conditioning equipment, cooling and humidity control systems, networking equipment (e.g. cables, routing/switching equipment), and lighting/physical security. Virtualization in the VDC relies on hypervisor technologies (e.g. KVM[2], Xen[3]), and mainly involves the physical machines (servers or hosts). Typically, this virtualization consists of dividing the physical server into multiple isolated virtual instances, each potentially having different sizes of system resources (CPU, memory size, disk space) and running different operating systems. Each tenant in a VDC owns a set of virtual machines (VM), interconnected through virtual links.

Today's VDC is confronted with an increasingly high demand for virtual resources (millions of virtual Endpoints) with unpredictable fluctuations over time. The VDC should not only guarantee the resources allocated to each tenant, but also provide tenants isolation of their virtual links from those of other tenants. Moreover, the traditional data center network architecture consisting of a layer 2 network, a layer 3 network, and/or multiple Ethernet LANs interconnected by IP routing has not been designed to handle a large number of VMs and the resulting traffic volume.

Indeed, to be adapted to a virtualized environment, the network architecture of a VDC faces some revolutionary challenges. These challenges create additional requirements for, especially: Any-to-any connectivity with full non-blocking fabric, dynamic scalability, fault tolerance, seamless VM mobility, and dynamic private networks, all with simplified management.

These new requirements raise many technical issues related mainly to traditional data center network design. The IETF Transparent Interconnection of Lots of Links (TRILL)[4][5] is an industry standard, specifically conceived for data center deployments. It basically meets all of these needs simultaneously. In fact, through its Rbridge[4] device that combines the advantages of Layer 2 (Bridges) and Layer 3 (Routers), the TRILL protocol enhances Layer 2 routing by introducing multi-path and multi-hop routing capabilities. This implies improved bandwidth, faster convergence, rich connectivity and better fault tolerance, all with little configuration. Moreover, with the use of Mac-in-Mac encapsulation, the TRILL protocol not only allows fast transparent VM migration, but also backward compatibility with existing infrastructures.

Notwithstanding these enhancements, in order to provide enough segments for scalable multi-tenant deployment, a VDC requires more VLANs than can be specified in 12 bits with the VLAN (IEEE 802.1Q) standard. It is therefore important to extend TRILL to encode a new, 24-bit segment identifier inside data frames. In addition, provisioning of the new tag and its integration in the TRILL data frames should be done in the End Host. This necessarily involves designing and implementing a TRILL Switch in End Hosts, since all current implementations of the TRILL Data plane are provided in hardware chips by the major network equipment manufacturers [6][7][8].

In this paper, we present Virtual Network over TRILL (VNT) which provides a large-scale Layer 2 network and easy support for multi-tenancy scalability. The VNT solution is a TRILL switch design and its implementation on End Hosts with the goal of meeting the above requirements. This also permits the creation of flexible and logical networks that are free from the usual network infrastructure limitations, such as the limit of 4096 LAN segments for IEEE 802.1Q. We accomplish this by introducing the VNI Tag, a 24 bit segment ID. This large VNI Tag is transported in the TRILL Header, and the additional addressing it provides allows the number of LAN segments to scale up to 16 million. We have a complete implementation of VNT, and we will provide here a set of experimental results to demonstrate the feasibility and scalability of VNT with only negligible overhead. This implementation does not impose any new requirements on the underlying switching software and hardware.

This paper is organized as follows: In Section 2 we discuss some background information, and the motivations for and goals of our work. Section 3 shows some related works. The design of the TRILL switch in the End Host and the interaction between its two planes (Control plane and Data plane) is presented in the Section 4. In Section 5, we present our VNT solution and the mechanics that allow building several virtual networks on a TRILL-based Layer 2 network. Section 6 will be dedicated to describing the implementation and our experimental evaluations to demonstrate Control plane effectiveness and Data plane robustness. Finally, Section 7 concludes the paper with a summary and discussion of future work.

2. Backgrounds

In this section, we present the new VDC requirements for providing an efficient and flexible IaaS service. To meet these needs, a VDC should build a large Layer 2 network. We will also introduce TRILL, which is an essential component in large-scale data center network construction.

2.1. VDC requirements

Unlike traditional data centers, a VDC faces major additional requirements that complicate implementation:

Virtual Machine mobility: VDCs need to be designed to support frequent and transparent Virtual Machine (VM) migration. This manual or dynamic migration allows improved resource utilization, execution of maintenance tasks, addressing specific geographic needs, or migrating machines for energy saving purposes. However, the migrated VM should not have to change either its IP addresses nor its security policies, as doing so may affect the services it runs.

Any-to-any connectivity with full non-blocking fabric: The used routing protocol should be able to establish one or multiple paths between two End Hosts along any of the available physical communication interfaces in the Data Center. With an unprecedented bandwidth requirement impinging on physical topology, the VDC should guarantee rich connectivity between each physical interface to increase the aggregated bandwidth and avoid link congestion, particularly between two End Hosts. The ultimate objective is to logically create a single switch, with non-blocking internal fabric that is able to scale up to support the requirements of a cloud-scale data center.

Fault tolerance: Several failures may affect processing at a data center. The VDC must be able to detect, diagnose and resolve faults. These failures may impact physical machines, but also switches or links. With live migration, a VM can resume its activity on another physical machine, providing the time necessary to resolve the failure of the physical machine. However, switch or link failures will break pre-existing TCP connections, which will cause disconnection for directly connected hosts. A non-blocking fabric that provides multiple traffic paths is the only way to tolerate such failures.

Dynamic scalability: VDCs should be designed to permit increasing or decreasing the number of virtual machines, links, and interfaces while the system is running. More precisely, a significant increase in the number of VMs necessarily entails an increase in IP and MAC addresses. In such a case, saturation of the limited memory set reserved to store the MAC address table in the switch can occur. This memory is commonly known as Content Addressable Memory (CAM). Depleting the MAC address table space causes significant quantities of incoming frames to be flooded to all ports of all switches. To avoid this, data center should be able to learn a large number of MAC addresses without necessarily adding new switches with larger capacity CAM.

Simplified management: VDCs must have an easy to manage, plug-and-play, transport layer that can support virtual machine migration with zero configuration. It must also be able to cope with changes to physical connections, dynamically changing numbers of logical connections, and changes to network size. All of this should not require administrators to perform heavy configuration tasks.

Private Network support: VDCs should be able to offer end users a self-service function that gives them the ability to build complex network topologies for their VMs. VDCs can use Private VLANs (PVLANs) to keep their customers isolated from each other, and to limit broadcast and unknown unicast traffic. This also restricts the traffic flow through their private ports. Additional ports can be added or removed from a PVLAN, based on users needs. Tag provisioning on ports should be more extensible in order to enable the customer to create elastic logical networks that span physical networks.

2.2. *Layer 2 network limitations*

The Ethernet-based network stands out as one of the most commonly used networking technologies by cloud operators. This popularity is mainly due to its operating simplicity and ease of configuration. In fact, in an Ethernet network, a MAC address is assigned to each host interface and Ethernet bridges will automatically learn this address and discover its location. Ethernet bridges connect multiple hosts and bridges to form a multi-hop network, and maintain the data forwarding table. When a frame is present on a port of a particular bridge, the learning process will automatically associate the port with the source MAC address. Then, the bridge forwards packets on the outgoing port following the learned destination MAC address by looking it up in the forwarding table. However, if the destination MAC address is not present in the forwarding table or is equal to ff:ff:ff:ff:ff:ff (i.e. a broadcast frame), the bridge broadcasts the packets on all ports except the receiving one, initiating a domain-wide flood. If some loops are present in the physical topology, this flooding can take the network down. In fact, in contrast to IP packets, the Ethernet frame does not have a TTL (Time To Live) field. Thus, a broadcast frame which enters a loop will be replicated and forwarded to other bridges, resulting in an infinite number of frames flowing through the data center network.

In this context, the Spanning Tree Protocol (STP) standard [9] ensures a loop-free topology, by creating a logical tree within a Layer 2 network and disabling links that are not part of the tree, leaving a single active path between any two network nodes. The STP algorithm converges when all links of the tree are either in the blocking state or in the forwarding state. Integration of the STP algorithm in the broadcast-based Ethernet architecture can provide plug-and-play service. When a new network element (a switch, a host, etc.) is connected to a data center network, all other elements are able to discover the new topology automatically and learn the host address and its location on the network. Furthermore, through flat addressing, Ethernet technology simplifies the processing of topological changes and virtual machine migration. This agility is realized without requiring the reassignment of IP addresses (unlike with layer-3 networks).

However, nowadays Ethernet technology is being confronted with revolutionary challenges to meet the requirements for cloud-scale data centers.

First, Ethernet still relies on packet flooding to locate End Hosts. With a large number of VMs that communicate with each other, this flooding technique causes a massive load on all forwarding tables. Such flooding results in a degradation of the overall network performance. Traditionally, the forwarding table is stored in Content Addressable Memory (CAM), which is

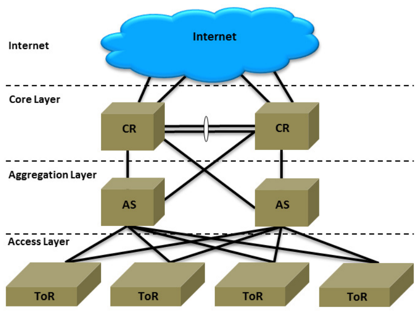


Figure 1: Conventional data center network

costly. Second, as indicated previously, STP builds a tree topology without loops, which is quite efficient for small networks, however, in a large network context, an STP solution suffers from several drawbacks. STP produces a single tree for all traffic and a single path for unicast and multicast traffic, which leads to the underutilization of network resources and results in a severe drop in network performance.

Finally, broadcasting or flooding mechanisms will also be used by bootstrap and broadcast protocols, such as Address Resolution Protocol (ARP) to resolve IP addresses and Dynamic Host Configuration Protocol (DHCP) to assign IP addresses. These broadcast frames will be dispatched to all hosts in a given domain, which might result not only in link congestion but also in bursts of resource consumption at End Hosts (e.g. a network with 1 million hosts generates more than 468,240 ARP packets per second or 239 Mbps of ARP traffic [10]).

To deal with the scaleability limits of Ethernet, conventional data center networks (Figure 1) are composed of multiple Ethernet LANs (Layer 2) interconnected by IP an routing layer (Layer 3). Hierarchically, conventional data center networks are built from a set of servers connected to ToR (Top of Rack) switches. ToR switches are connected to aggregation switches (AS) to ensure rack-to-rack interconnections. Access Switches (AS) are connected to core routers (CR) through access routers (AR) for traffic coming from and going to the Internet. However, in such network architecture, if VMs are migrated between LANs (Layer 2 areas), their IP addresses have to be reassigned. Moreover, it imposes the use of STP. Figure 2 shows a TRILL-based Network architecture defining a new-generation of virtualized data centers. In such an architecture, there is any-to-any connectivity with a fully non-blocking fabric, and each device has a direct connection to every other device.

2.3. *TRILL*

TRILL is an IETF industry standard protocol, implementing Layer 2 routing. Through new ”routing bridge” (RBridges) devices, TRILL runs a Layer 2 routing protocol, using the intermediate system to intermediate system (IS-IS) protocol [11] to identify the shortest paths between switches on a hop-by-hop approach. Each RBridge knows all the other RBridges and all connections between them, which enables calculation of the shortest paths to all other nodes through the shortest path

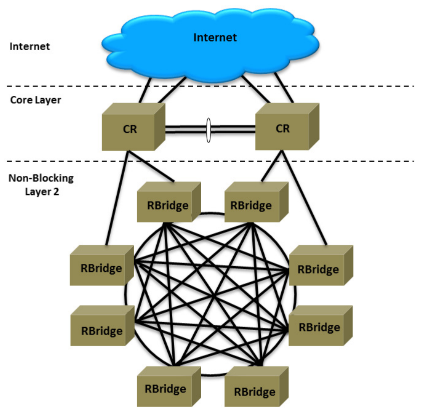


Figure 2: TRILL-based network for virtualized data center

tree algorithm. This provides enough information to RBridges to use optimal pairwise paths for unicast traffic. In addition, TRILL uses distributed trees and a designated Rbridge as the Distributed Tree root (DTroot) for forwarding traffic to multicast/broadcast groups or delivering frames to unknown destinations. TRILL meets the previously mentioned VDC needs. Indeed, TRILL builds loop-free multipath topologies without needing STP. This allows multi-destination frames to be efficiently distributed over the whole network. This results in improved bandwidth, better fault tolerance and provides a plug-and-play service. Through its Mac-in-Mac encapsulation, TRILL should help to face issues associated with forwarding table overflow. TRILL also allows for transparent VM migration. In fact it does not require changing the IP address of the VM while migrating between End Hosts.

3. Related works

There have been several studies focusing on scalable virtualized data center and/or logical virtual networks.

3.1. Scalable virtualized data center

Studies like [12][13][14][15] have already tried to achieve a scalable virtualized data center. For instance, Portland[12] and SPB[13] suggest a new transport layer by replacing or adding a new MAC header. Portland is based on multi rooted Fat Tree topology characteristics. This new approach consists of maintaining the current transport layer unchanged, and replacing the actual MAC address by a new pseudo MAC PMAC address. PMAC enables a hierarchical VM addressing scheme for Layer 2 routing. An edge switch connected to the End Host containing the VM converts MAC into PMAC. The position of this switch is either set manually by an administrator or automatically set through Location Discovery Protocol (LDP). In order to resolve ARP correctly and handle multicast traffic and fault tolerance, a fabric manager is added. This fabric manager consists of an independent process running on a specific machine. Even if Portland addresses some of the current data-center limitations by achieving provably loop free L2 routing, this solution remains valid only in multi-rooted fat-tree architecture, and cannot be applied on other architectures. In addition to this limitation, Portland is highly dependent on a central server (the fabric manager). This makes the network vulnerable to failure of or malicious attacks against this particular server. Unlike Portland, SPB adds an additional MAC header and achieves Layer 2 routing using IS-IS protocol. But SPB relies exclusively on the Control planes global view of the network to achieve loop avoidance and does not offer any backup solution such as a hop count in Data plane.

SEATTLE [14] and EtherProxy [15] have a different approach. Both of them provide techniques to limit broadcast traffic without requiring any modification to the transport layer. The SEATTLE network architecture converts ARP into a unicast ARP message using a distributed network layer hash table. Etherproxy is a piece of network equipment designed on the principle that much of the broadcast traffic in a given network does not need to reach its destination to get a coherent response. Etherproxy will catch any recognized broadcast traffic and reply on behalf of the destination, using cached responses. If it has no response in cache, it will act as the source and will inquire for a coherent answer from the destination. Both of these techniques reduce broadcast traffic, however, they increase unicast. Also SEATTLE depends on a directory service in order to save the proxy table.

3.2. Logical virtual networks

Other work in this area has focused on producing several virtual networks inside a virtualized data center. For instance, VXLAN[16] and DOVE[17] are based on encapsulating the L2 tenant frame within a UDP header, adding a 24-bit Virtual Network Identifier (VNI) tag to each tenant. When VXLAN frames are sent from one VM to another, the VTEP (a kind of gateway) adds an IP header containing the destination VTEP IP address. For unknown or multicast destinations each VNI has a multicast group allowing a frame to be flooded only onto the VTEPs that support this VNI.

Despite the efficiency gained in tenant isolation, this approach suffers from two major faults: Layer 2 in layer 3 encapsulation, and the need for a multicast group for each tenant. Using L2 in L3 encapsulation involves adding all L3 security problems to L2 switching [16]. This will impose a requirement for more stringent L3 security rules in order to avoid having a malicious entity send a compromised L2 frame to a tenant encapsulated in an apparently valid L3 frame. In addition, mapping a VNI per multicast group will eventually incur a high configuration overhead.

NetLord [20] is another L2 in L3 encapsulation approach. It does isolate tenant traffic, however the IP Layer is actually used to encode tenant information and not to route frames. The NETLord IP source address is used to encode a MACASID. MACASID is a MAC address space ID, used to support the use of multiple L2 address spaces. The destination IP address encodes the destination edge switch port and tenant ID.

While reducing overhead compared to VXLAN, NETLord, by using this encoding, faces two main challenges: First, it requires a manual static switching table between all switches in order to avoid any switching error. Second, it uses a special ARP message for the NLA (NetLord Agent) to learn the switch MAC address and port for each corresponding VM MAC. NVGRE[21] also uses an L2 in L3 encapsulation by using Mac-in-Gre. In this case the tenant ID, which is 24 bits long, is contained in the GRE header. As is the case for VXLAN and DOVE, this protocol will expose layer 2 to all layer 3 security problems.

Other approaches have been based on Software Defined Networking architecture where the control plane and the data plane are decoupled. The control plane functionalities are migrated to a logically centralized computing devices and forwarding instructions are sent to the data plane via specific protocols such as Openflow [18]. In this context some innovative solutions have been suggested by different research works. For instance Nicira has a specific solution which is Network Virtualization Platform (NVP) [19]. NVP uses an L2 in L3 encapsulation with a MAC over IP format which is different from the VXLAN one. This encapsulation consists in a point-to-point generic routing encapsulation (GRE) tunnels. Unlike non-SDN solutions, NVP does not need any multicast group as it does not need flooding to build forwarding tables. In fact, thanks to the SDN paradigm the MAC-to-IP mapping is directly provisioned in vswitch using Open vSwitch Database Management Protocol (OVSDB) [19]. It also eliminates all flooding comparing to VXLAN or others, non-SDN, L2 in L3 encapsulation protocols. It also eliminate the need of keeping state in the network. But, this also makes the NVP highly dependent of two main logically centralized controllers which are the Openflow controller and the OVSDB one.

4. TRILL switch design on end hosts

The TRILL protocol relies on frame encapsulation with an additional Layer 2 TRILL header in order to achieve Layer 2 routing. Such encapsulation avoids introducing additional security problems. Encapsulation, however, also implies that TRILL equipment, in addition to being connected to other TRILL equipment, will also be connected via a particular gateway to the standard Ethernet network. This gateway to the TRILL network can be chosen among several particular network inter-connexions. One of the most crucial questions before starting the RBridge design discussion is to decide this gateway location. The location of this gateway is essential since it will subsequently affect the whole encapsulation and decapsulation design.

4.1. TRILL gateway location discussion

There are three main potential locations for this gateway which are: in a silicon switch, on End Hosts in a software switch, and at the Endpoints network interface.

4.1.1. Silicon switch

Silicon switch may seem to be the best approach to deploy the RBridge solution. In fact, this is a natural first choice when thinking about such and adaptation. This is the location described on the RFC[4] (the original one, not the recent smart end nodes draft[22]). But rapidly, we realized that it is not such a nice pick. In fact switches are not supposed to copy frames in their internal memory. So using such an approach will add additional memory transactions which may critically reduce performance. In addition to this, deploying the gateway on the switch will add a huge number of MAC addresses in switch forwarding and correspondence tables. Finally, there is another problem to face. Silicon switches use embedded switching software (or hardware switching) and this makes them difficult (or impossible) to modify without full manufacturer support.

4.1.2. End host software switch

Another option would be to implement the gateway directly in the End Host, which means directly in physical server. Such an approach will have several advantages, such as:

- Separation of VM traffic that will be TRILL encapsulated from server traffic that will continue in standard Ethernet format.
- Avoiding encapsulating and decapsulating traffic between VMs on the same host.
- Allowing the number of MAC addresses handled by each server to be reasonable compared to those that a switch would have to handle. (see section 6.2.3.1)
- Lowering the overhead of frame encapsulation, considering that they are generated on the server (already copied on the memory).
- Code implementation and maintenance does not require any manufacturer support and can be easily upgraded with new features.

The only drawback of such an approach would be the additional overhead that End Hosts would have to support.

4.1.3. Endpoints network interface

Implementing encapsulation directly in the Endpoints and decapsulating at the End Host level would certainly produce the least overhead. In fact, adding a TRILL header while creating the frame will induce almost no additional overhead, but such a scheme would be too risky. As tenants have full access to their VM they would be able to modify encapsulation and this would produce an unacceptable security fault. Another major problem of such a solution would be the continuous communication between End Hosts and Endpoints in order to update routing information.

4.1.4. Discussion

Taking into account advantages and drawbacks of each potential gateway location, it is obvious that the best way to proceed would be to make End Hosts software switch act like a gateway RBridge. This choice will have the lowest cost in overhead while not creating any major security risks. In fact, when placing the RBridges at Endpoints level theirs configuration will be accessible to tenants which will enable them to change Rbridges routing. Finally such an approach unlike the silicon switch one would not require too much complex and manufacturer dependent development.

4.2. RBridge implementation design

Following the choosing of the TRILL gateway location, which we decided was End Hosts software switch we will describe the implementation. Implementation design will be explained in two parts, Control and Data plane. The Control plane will handle all decisional parts and forwarding table generation, while the Data plane will handle all encapsulation, decapsulation and forwarding tasks.

4.2.1. Control plane

The Control plane ties together all TRILL components responsible for building topology and making routing decisions based on link costs. The Control plane in TRILL is based on the IS-IS protocol. In order to route frames, it uses SPF (Short Path First) tree topology. The SPF tree is generated based on Link State PDU (LSP) messages. The Control plane therefore consists of LSP exchanges between RBridges, which are used to compute SPF. Based on [4], LSP transports information in a TLV (Type, Length, Value) form. The TRILL routing protocol is based on the link state protocol with an additional nickname exchange phase. This routing process runs as follow : The control process starts with a discovery phase. In this phase each RBridge port sends a hello message containing the supported MTU, port priority, and a list of known neighbors. Once an RBridge receives a hello, it will add the MAC address of the sender in its known neighbors list in its own hello. If an RBridge receives a hello containing its own MAC address (in known neighbors), it infers that there is two-way communication on this port and adjacency will be marked as up. Based on port priority, RBridges that are on the same LAN will elect a designated RBridge (DRB or DR) (equivalent to Designated Router (DIS)[11] in IS-IS). After exchanging hello messages, RBridges exchange LSP messages. These LSPs contain the RBridge nickname. Each RBridge sends its nickname and all its adjacenct RBridge nicknames over LSP. For RBridges that are in the same LAN, only the DRB will send these LSPs, on behalf of the others. Others RBridges will send LSPs only after a new adjacency is discovered on another port, or after an LSP time out. This ensures DRB data coherence. Taking into consideration LSPs and adjacency information, each RBridge will compute it own forwarding table and adjacency for each RBridge in the topology. This information will be sent to Data plane to allow selection of the next hop for each host based on forwarding table. The messages exchanged will consist of a list of all RBridges in the topology with the following information for each:

- Host Nickname
- Mac address of next hop to reach this particular node
- Selected Distributed Tree root (DTroot)
- RBridge adjacencies on selected Tree

```
                                +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                                | V | R |M|op-Length| Hop Count |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Egress RBridge Nickname      |    Ingress RBridge Nickname   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Option ......                                                 |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 3: Standard TRILL frame

4.2.2. Data plane

TRILL encapsulates Layer 2 frames with an additional Layer 2 header. This new header contains an outer MAC layer and a new TRILL header shown in figure 3. This TRILL header contains an Ingress and Egress nickname identifier. These nicknames are used for routing decisions. The header also contains a hop count to avoid routing loops, and a multicast bit to identify multicast TRILL frames. The TRILL RFC provides an extra option field along with an option length field. This option length is expressed in 4 bytes units. The Data plane handles all frames arriving at the End Host software switch. This plane provides routing decisions based on both the TRILL header and information on the Egress End Host, using data determined by the Control plane. There are three main routing decisions that can be made when a frame reaches an End Host software switch:

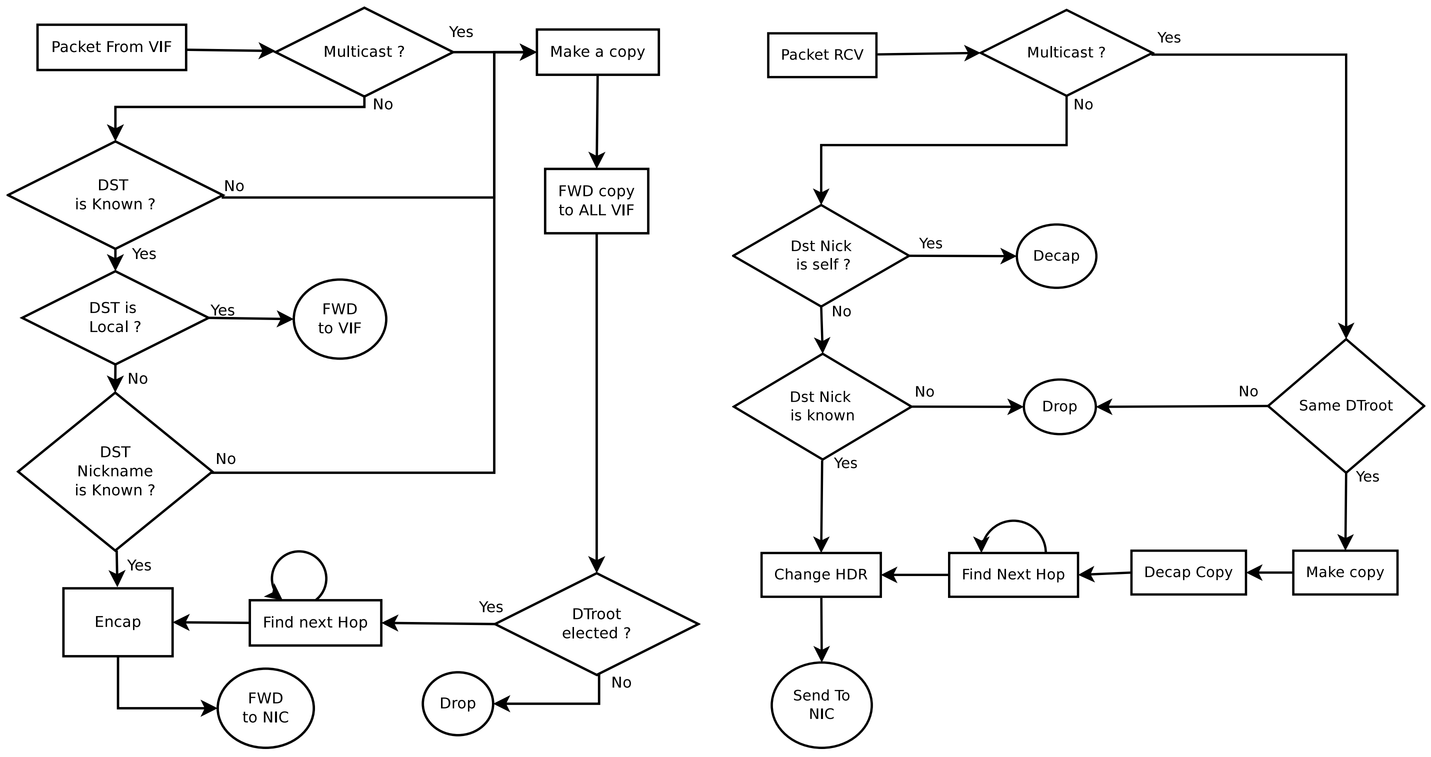
- Frame has a native Ethernet type provided by an Endpoint port: In this case it should be encapsulated and/or forwarded (locally forwarded with no encapsulation if destination is a local Endpoint).
- Frame has a native Ethernet type provided by local Network Interface Card (NIC): In this case it should be considered as a non TRILL frame, and forwarded to a non Endpoint port as a native Ethernet frame (as will be the case for all end-host-to-end-host traffic or End Host management frames).
- Frame has a TRILL Ethertype: In this case the frame is handled by the receiving function and routing decisions are made based on TRILL header information.

Details of frame handling decisions can be found on the corresponding flow chart on Figure 4.

4.2.2.1. Packets encapsulation.

A frame is handled by the encapsulating function if this frame corresponds to the standard Ethernet type, the source is a local Endpoint, and the destination is not a local Endpoint or is unknown. The first step is to determine whether the destination address is known or not. For this, a lookup in the forwarding table is performed.

- If an entry is found and this entry contains the destination End Host nickname, then the frame is encapsulated with a TRILL header. This additional header specifies localhost as Ingress nickname and destination End Host as the Egress nickname. The frame is then forwarded to the next hop to reach the Egress End Host, based on the Egress host information (as received from the Control plane).



(a) send (b) receive

Figure 4: Packet processing chart flow

- If no entry is found, the frame is duplicated. A first copy is flooded on all local Endpoint ports. A second one will be encapsulated with a particular TRILL header. This header specifies local-host nickname as Ingress nickname, the locally elected DTroot as the Egress. The Multicast bit in the TRILL header will be non zero. This second copy is then duplicated and sent to all the adjacencies in the Tree rooted on the locally chosen DTroot, using unicast to the destination MAC address.

4.2.2.2. TRILL reception handler.
Once a TRILL frame is received, there are two possibilities: the frame may be of type unicast or type multicast.

- If the frame is type unicast, then a check is made to determine whether its destination is the current End Host or not. If so, the frame is decapsulated. Once decapsulated there are three options: If the destination is a local Endpoint port, then the frame is forwarded. Otherwise, if the destination is a local non-Endpoint port, the frame is destroyed to avoid re-injecting a decapsulated frame in the TRILL network. Finally, if the destination is unknown, the frame is flooded on all Endpoint ports in local-host. If the frame is of type unicast, and the local host is not the Egress host, then, if the Egress host information exists in the local-host database, then the frame is forwarded to the next hop to reach the Egress and frame's hop count is decremented. If the local-host has no information about Egress host then the frame is dropped.
- If the frame is of type multicast, a check is made to determine if the DTroot (Egress nickname) is present on the locally supported DTroot list. If the DTroot is not the same or is unknown then the frame is discarded. If the check of the DTroot is successful, then the frame is duplicated and the hop count decremented. A first copy is forwarded to all local-host adjacencies on the Tree rooted at the DTroot, except to the one that has sent the original frame. The second copy is decapsulated and forwarded locally to all Endpoint ports.

5. VNT: Virtual Network over TRILL design

In this section we present our approach for abstracting tenant networks by enhancing the TRILL protocol. We will start by introducing our concept of network abstraction from the tenant network point of view, followed by the virtual routing process. We will conclude with a description of VNT component design.

5.1. A tenant's view of VNT

A tenant, due to the abstraction layer, will have a particular view of the network. Each one one his Endpoints will only receive traffic issued from other Endpoints he owns. His Endpoints will not receive messages from other tenant Endpoints. A tenant will have complete freedom of network configuration, will not be restricted to any particular IP/MAC addresses ranges, and will have the possibility to configure his Endpoint interfaces with VLAN tags to further segment his own network view.

5.2. Virtual routing

By introducing the VNI tag, we have created a network abstraction layer for each tenant. Thus for each tenant two virtual routing processes will be executed simultaneously. The first virtual routing process will be a Layer 2 routing scheme. It will take the VNI into consideration to virtually restrict tenant frame propagation only to End Hosts that contain his Endpoints (i.e. that share his VNI tag). This first virtual routing scheme uses a generated tree topology. This scheme will then deny frame forwarding on any vertex that neither supports that particular VNI tag nor are an intermediate vertex to vertex that supports it. With this first virtual routing process, each tenants traffic is isolated from other tenant traffic in a subnetwork representation. Within each tenant subnetwork representation, a second-virtual routing process based on the endpoints IP configuration will take place. This routing process will completely depend on tenant Endpoint L3 configuration, within each tenant network representation. This Layer 3 routing process will benefit from all standard Layer 3 routing techniques.

5.3. Required configuration information

5.3.1. Control plane

Concerning the Control plane configuration requirement, an RBridge requires little configuration, but some parameters can be specified in order to induce specific behaviors. As well as TRILL, VNT has designated RBridge (DRB) and Distributed Tree root (DTroot) elections. These two elections are based on RBridge priority. This priority can be specified to force a specific behavior or the default value can be used. When no priority is set, the MAC address is used as tie breaker. An RBridge nickname can also be specified, but if it is not, a unique nickname is automatically generated.

5.3.2. Data plane

For the Data plane, in order to tag frames with the correct VNI a new virtual network component is introduced, called a Virtual Switch (VS), shown in figure 5. A VS is dedicated to host all interfaces tagged with a particular VNI corresponding to a tenant ID. These VS are introduced to simplify flooding on interfaces with the same tag. Once a tenant requests a new VM (Endpoint), the cloud service provider will instantiate it. Then, based on a tenant ID, all Endpoint interfaces (virtual interfaces) will be attached to a VS corresponding to that ID. In this way each time a frame reaches the RBridge from a VS, the TRILL encapsulation process will take into consideration the VS VNI tag. We will not detail the VM creation process here, as it is not relevant for our proposal. Instantiation and interface tagging are also out of scope for this paper, as they are cloud provider dependent. Furthers details of tagging will be omitted. In all remaining parts of this paper an Endpoint will always be considered as VNI tagged. In conclusion, the unique mandatory parameter in a VNT RBridge is the VNI that is attached to all Endpoint interfaces (VS VNI TAG) on interface start up.

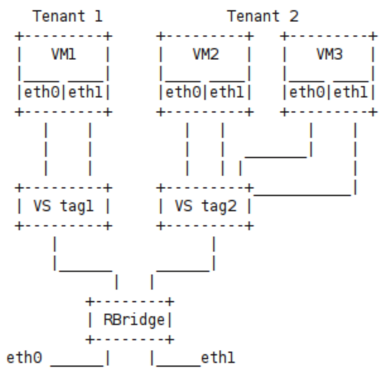


Figure 5: Virtual switch design

5.4. VNT components

VNT is mainly composed of two parts: Control plane and Data plane.

- The Control plane performs LSP (Link State PDU) message exchange based on IS-IS messages in order to build a topological representation. The Control plane task is performed by an enhanced Quagga [23] daemon. Quagga is open source routing software implementing all major routing protocols.
- The Data plane performs MAC in MAC encapsulation and decapsulation of frames to the original form based on the topology information provided by the Control plane. The Data plane task is performed by an RBridge Linux module.

5.4.1. Control plane design

The VNT Control plane is based on the TRILL Control plane, with some modifications in order to add VNI tag propagation and tree pruning based on the VNI tag. To achieve network abstraction with VNT, it is essential to carry out a VNI tag propagation at the Control plane level. This is done to advertise which VNIs can be handled by each End Host. An End Host handles a VNI if this VNI is locally supported, which means that this host has an Endpoint that belongs to that VNI. A VNI can also be forwarded, which means that while it is not supported locally, two neighbors connected to two different interfaces support it, even though they may be N hops away. In other words, the local RBridge may be needed to forward this VNI between two others. In order to compute which VNI are handled where, the control plane will periodically check locally supported VNI tags by querying the Data plane. Once the list of locally supported VNI is updated, a new sub-TLV message is generated in a new LSP. The new sub-TLV will contain the number of VNI tags, followed by a list of the handled VNI that are locally supported and the VNI that are forwarded but not necessarily locally supported. Naturally, in the first LSPs generated, the handled VNI will include only locally supported, until the computation of forwarded VNIs completes.

Once an RBridge receives an LSP containing a VNI sub-TLV message, it computes the handled VNI list for others RBridges received in the LSP using Algorithm 1.

```
                                +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                                | V | R |M|op-Length| Hop Count |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Egress RBridge Nickname      |    Ingress RBridge Nickname   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Crit.|  CHbH   |   NCHbH   |CRSV | NCRSV |   CItE    |  NCItE  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|             Reserved            |          Flow ID            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|APP|*|      Type    |#| Length  |            reserved           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     VNI                     | reserved        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

*: NC      #: MU
```

Figure 6: Standard VNT frame

A loop is performed on these LSPs, and once a non supported VNI is found it is added to the local VNI list with a ”discovered” state flag and the interface number from which the LSP containing this VNI was received. If a VNI is flagged as discovered twice by two LSPs received from two different interfaces and from two different End Hosts, the VNI flag is changed to forwarded state. Each time SPF trees are computed, the handled VNI list is reset to supported VNI only, in order to avoid false positives. For example, if one of the RBridges sends a no longer supported VNI to the local End Host, this reset will age it out. Finally, when sending nickname information for an End Host to the Data plane (see TRILL Control plane) the information for each End Host VNI is appended to the nickname information.

5.4.2. Data plane design

5.4.2.1. VNT frame.

A VNT frame is based on a standard TRILL frame, enhanced with a VNI option. As specified in drafts [24] and [25], when an option field is present the frame has to start with 32 mandatory bits specifying whether included options are critical or not. This 32-bit block is followed by another 32-bit block containing reserved bits and flow id. Flow id is used to distribute traffic over several paths. The flow id will indicate which path to choose when forwarding the traffic. Finally, the two drafts specify that any following option will be in TLV format. VNT uses this format to add another 64-bit block containing VNI. The final VNT frame will have the form described in Figure 6.

The first 32-bit block describing option criticality is composed as follows: The three first bits are critical summary bits which indicate the existence of critical options in option header. Remaining critical bits describe the critical type of each extension. This 32-bit block is followed by another containing 18 reserved bits and 14 bits for flow id. These two 32-bit blocks are specified by RFC drafts [25]. In order to add VNI on the TRILL header, another 64-bit block VNT extension in TLV format [24] [25] is added as follows: The two first bits (bit 0 and bit 1) describe the applicability using these values:

- 0 (00): critical hop by hop
- 1-2 (01 or 10): reserved values
- 3 (11): critical Ingress-to-Egress

```
while TRUE do
    /* it is unnecessary to compute VNI for DTroot as it
       has to support all VNI                                  */
    if LocalHost ≠ Dtroot then
        CreateList(HandledVni);
        /* step1:  add locally supported VNI                   */
        foreach Vni in SupportedVni do
            NewElement .Vni ← Vni;
            NewElement .VniType ← supported;
            NewElement .SeenIface ← 0;
            AddToList (HandledVni, NewElement);
        end
        /* step2:  compute other known Hosts VNI               */
        foreach host in KnownHost do
            if host ∈ PseudoHost and PseudoHost .DR = LocalHost then
                /* DR has to support all VNI that are contained
                   on a pseudo Host                            */
                foreach Vni in host .AdvertisedVni do
                    NewElement .Vni ← Vni;
                    NewElement .VniType ← supported;
                    NewElement .SeenIface ← host .SeenIface;
                    AddToList (HandledVni, NewElement);
                end
            end
            else
                foreach Vni in host .AdvertisedVni do
                    foreach element in HandledVni do
                        if Vni = element .Vni then
                            if element .VniType = discovered and element
                            .SeenIface ≠ host .SeenIface then
                                element .VniType ← forwarded; ;
                            end
                        else
                            NewElement .Vni ← Vni;
                            NewElement .VniType ← discovered;
                            NewElement .SeenIface ← host .SeenIface;
                            AddToList (HandledVni, NewElement);
                        end
                    end
                end
            end
        end
        /* step3:  create advertised VNI list and remove
           unnecessary VNI                                     */
        CreateList(AdvertisedVni);
        foreach element in HandledVni do
        if element .VniType = discovered then
        | RemoveFromList(HandledVni, element);
        end
        else
        | AddToList (AdvertisedVni,element .Vni);
        end
        ;
        Send(AdvertisedVni);
    end
end
```

Algorithm 1: Computing forwarded VNI

In order to ensure backward compatibility with standard TRILL RBridges, these two bits have a value equal to 3 (11) (critical Ingress-to-Egress). The third bit (bit 2) is set to 1 if the extension is non-critical. For the VNT extension, this field is set to zero, because in the VNT definition, all End Hosts have to take into consideration the added VNT extension (the VNI) to ensure proper network isolation. The seven following bits describe the extension type. For the VNT extension type, this is set to 0x3D (0111101). Bit 10 determines extension mutability. For security reasons, the VNT extension has to be immutable. This bit is therefore set to zero. Bits from 11 to 15 are dedicated to field length. Option length in the TRILL header is expressed in a 4-byte block (32-bit block). In the VNT case the length is equal to two. In order to keep the VNI aligned on the same 32-bit block, the following 16 bits are reserved for future use. The following 24 bits are used to store the network identifier, and finally the last 8 bits are reserved for future use. Although the TRILL option field allow to use more than 24 bits to store the network identifier, the choice was made to use this specific value for two main reasons. Firstly, a 24 bits identifier will allow an easy mapping when interconnecting VNT network with VXLAN, 802.1AD or any other pre-existing logical network technologie. Secondly, the TRILL protocol support up to 65535 Rbridge using 24 bits segments will allows deploying more than 512 endpoint per RBridge (2 endpoints per each VNI). However, it is also possible to extend this identifier to 32 bits using the 8 remaining reserved bits in the same 32bits block.

5.4.2.2. MTU discussion.

VNT frames are Layer 2 frame, then they can not be fragmented. Thus it is important to avoid situation where two entities support conflicting MTU. The TRILL protocol already provides a negotiation phase for MTU, such a negotiation will synchronise MTU values in all End Hosts to the least supported one. End Hosts MTU synchronisation is not enough to avoid frame overflow. In fact, as VNT provides a Layer 2 encapsulation within Layer 2 frame, it is fundamental to provide an MTU difference between End Host and Endpoint that is at least equal to VNT overhead.

5.4.2.3. Encapsulation details.

When a frame is sent from a VM, the frame reaches an RBridge via the VS to which that VM interface is attached, and is then encapsulated based on the VS VNI.

- When destination nickname is known, the frame is encapsulated with the current nickname as Ingress and the destination nickname as Egress, and an option containing the VS VNI. For the outer MAC header the destination address is the MAC address of the next hop to the destination, and the source address is the MAC address of the current End Host.
- When the inner destination nickname is unknown, the Egress nickname is the DTroot.

Further details on forwarding steps can be found below.

5.4.2.4. A frame journey.

In this section, the frame journey from source to destination will be traced, based on the frame type, multicast or unicast. Once a VM transmits a frame, this frame is transmitted to the RBridge. When the frame reaches the RBridge, as it came from a TRILL flagged port, frame will be forwarded as follows:

5.4.2.4.1. Multicast frames

When a multicast frame comes from a TRILL flagged port, the VNI is checked. As it is a multicast frame, the frame is copied. A first copy is flooded on the local VS that handles the specific VNI, except to the source port. The second copy is sent to the encapsulation process. Such frames have this particular header:

- Multicast field will be set to one.
- VNI option field will be filled with the source port VNI.
- Hop count will be set to the default.
- Ingress RBridge will be defined as local RBridge.
- Egress RBridge will be the DTroot.

DTroot will be found in the nickname database as transferred from the Quagga daemon. After adding TRILL and Ethernet headers, a new frame will be "multi-forwarded". Multi-forwarding is based on each End Host neighbor list. The generated VNT frame with multicast flag will be forwarded to all neighbors on the distributed tree rooted at Egress RBridge. The frame is forwarded to all neighbors that support that particular VNI, and to the DTroot which is supports all VNIs. This is done by forwarding multiple unicast frames with different outer Ethernet destination addresses (multi-forwarding). When a frame with a multicast flag is received by another End Host, before multi-forwarding it, the End Host makes a local copy. The local copy is then decapsulated and flooded in the local VS that handles the indicated VNI. Once a frame is decapsulated its source address and Ingress nickname are entered in the forwarding database. After the hop count is decremented, the original frame will be multi-forwarded to all neighbors supporting/forwarding the VNI except to the End Host that sent the original frame. With such a technique, multicast frames are delivered exclusively to End Hosts that contain VMs that belong to the same VNI.

5.4.2.4.2. Unicast frames

When a unicast frame reaches an RBridge from a TRILL flagged port, a lookup in the forwarding database is done. Three possible scenarios exist:

- Address is known and local; in such a case if the destination belongs to the same VS, the frame is forwarded by the VS to destination. Otherwise the frame is discarded, since the destination no longer exists.
- Address is unknown: This generally happens when two VMs stop communicating for a long period, which leads

to the expiration of the forwarding database entry. In this case, the frame is forwarded as a multi-cast frame.

- Address is known and corresponding nickname is found. In such a case, frame is encapsulated with the following header:
  - Multicast field will be set to zero.
  - VNI option field will be filled with the source port VNI.
  - Hop count will be set to default.
  - Ingress RBridge will be defined as the local one.
  - Egress RBridge will be the one found in forwarding database.

  After encapsulation, the frame is forwarded based on the End Host database entry that specifies for each Egress End Host the next outer Ethernet MAC address needed to reach it. Upon reaching the destination, the frame is decapsulated and sent to the VS that supports the specific VNI. The VS transfers the decapsulated frame to the correct port. In the case where no port is found, the frame is dropped. This may happen when the destination VM was recently destroyed, or its MAC address was changed. To avoid any security issues the decision was made to destroy the frame in this case.

5.4.3. Control/Data plane communication

The Data and Control planes need to communicate periodically. In fact, the Control plane needs to maintain a fresh VNI list in order to achieve distributed tree pruning on VNI. Also, as the Control plane computes the SPF tree and elects the DT-root, this information needs to be sent back to the Data plane in order to update the End Host database. An update can be triggered either by the Control or the Data plane: by the Control plane when a topology change occurs (new host arrival, host failure) and, by the Data plane when the list of locally supported VNIs changes. Such VNI changes happen when a new VM creation triggers a new VS creation, stemming from the fact that no other Endpoint belonging to that tenant is currently present on the End Host.

6. Experimental analysis

6.1. TRILL and VNT implementation

In this section, we will present the details of our implementation of the VNT. In order to implement VNT we have made modifications to the three components shown in figure 7. These components are: Quagga daemon, linux bridge module, and linux bridge-utils which is a configuration interface to linux Bridge.

6.1.1. Control plane: Quagga

The Control plane consists of a Quagga daemon (Quagga-0.99.21), that was modified by adding hundreds of lines of code, based on a previous unfinished Open Solaris[26] initiative. The Open Solaris implementation no longer exists, and is only mentioned in some email archives [27]. We ported this previous initiative to the recent Quagga version. We also have refactored this version in order to achieve correct nickname exchange. Finally we have modified the topology building process to accomodate multipathing and VNI propagation.

6.1.2. Data plane : Linux Bridge module

The Control plane was incorporated inside the pre-existing Linux bridge. The frame handler was modified in order to determine which frames to encapsulate, which to decapsulate and which to forward. Encapsulation, decapsulation and forwarding functions were also added to the pre-existing Bridge implementation. Finally, a Netlink interface has been implemented in order to send the VNI information to the Control plane and to receive all topology information from it. We have based our work on linux 3.4.x Linux Bridge, adding more than 3000 lines of code. As of this writing, our implementation is based on version 3.4.44 after being ported from several previous 3.4.X versions.

6.1.3. VNI configuration interface: bridge-utils

A final part of the development was for dynamic VNI management. To facilitate this, a few modifications have been made to bridge-utils in order to add new Ioctl commands. These commands allow attaching (or detaching) virtual interfaces to (from) a particular VS, and to show interfaces connected to a particular VS. For this development we have used the latest bridge-utils version 1.6.

6.2. GANDI VNT deployment

The VNT implementation [28] software is deployed on more than 160 servers (End Hosts) in the GANDI public cloud infrastructure, and is used by several tenants. Currently, we are considering adding RBridges states monitoring protocoles. A partial open source code containing the TRILL part is already available to scientific community at [29]

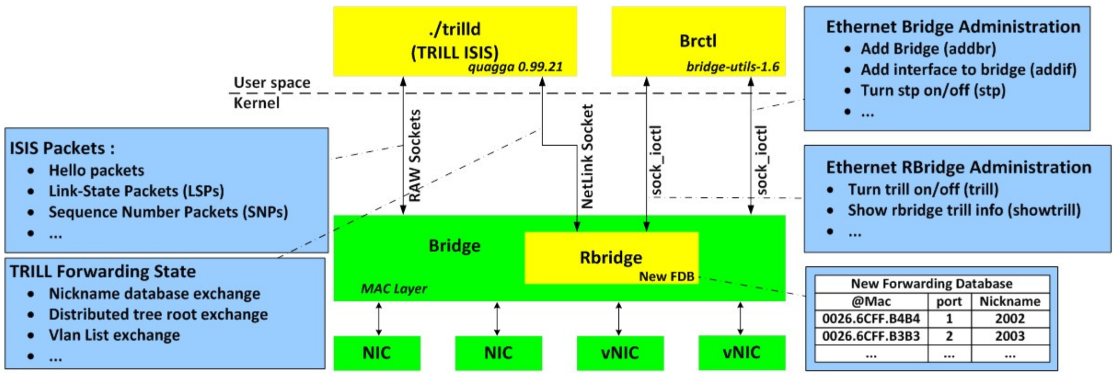
6.3. Testbed and experimental evaluation

6.3.1. Testbed

We have built our testbed using ten servers, running under xen 4.2.1 with a modified linux 3.4.44 as DOM0 and two different server types:

- DELL R310 with an Intel Xeon L3426 @ 1.87GHz 08 core CPU and 12 GB of Ram, each one having two 10Gb interfaces and two 1Gb interfaces.
- DELL C6100 with an Intel Xeon L5640 @ 2.27GHz 12 core CPU and 48 GB of Ram, each one having two 10Gb interfaces and two 1Gb interfaces.

We also have used 2 different switches:



@Mac	port	Nickname
0026.6CFF.B4B4	1	2002
0026.6CFF.B3B3	2	2003
...	...	...

Figure 7: Implementation design

- 1 Cisco nexus 5548 with 48 10 Gb ports.
- 1 commodity 1Gb switch

Considering this equipment we have built two different topologies (TRILL topologies):

- a fat tree topology with two levels: with two servers at the first level connected to each other in the second level. At each level one server is connected to 4 others via the nexus 5548 (2 different VLAN are used for each level 1 server connection).
- another topology where all ten servers are directly connected to the nexus 5548 switch.

To start the TRILL performance evaluation, we first tested Control plane efficiency and robustness. The second step was to evaluate Data plane performance. The third was quantifying the total overhead. Finally a brief description about backward compatibility is given.

6.3.2. Control plane evaluation

In order to evaluate our Control plane robustness, the convergence time for TRILL and VNT was evaluated. Convergence time needs to be defined, however. Convergence time is the time elapsed between a new End Host arrival and the maximum time after which all other End Hosts on the topology recognize it as an adjacent End Host. To evaluate this convergence time, a minor modification in the Control plane process was introduced. This modification consists of adding an uptime in each LSP. In this way, each RBridge can save the startup time for each one of the other RBridges in its local time reference. Such an approach will avoid possibly misleading time synchronization issues between End Hosts. By using this method, it is clear that we have neglected propagation time between End Hosts, but such an approximation should not impact the final result, due to the huge difference (one order of magnitude) in the timescale between propagation delay and convergence time. Each RBridge X has the same behavior: this RBridge will log all other RBridge information. Each time the Control plane in RBridge X transmits to its own Data plane information for RBridge Y, it will log the uptime and the topology transmitted by Y (adjacency number). When parsing the log for X, we consider Y as converged from the X point of view when X logs the first correct adjacency number for Y. To have a huge number of interconnected RBridge, we have emulated RBridges inside virtual machines. Such an approach will allow us to deploy as many RBridges as we want. Moreover, this allows a more flexible management process. Such an emulation does not affect the results due to the fact that we are testing only Control plane. For test purposes at random intervals a new RBridge was connected to the current topology; The convergence is computed in all RBridges.

6.3.2.1. Convergence without VNI.

For the convergence time, all Rbridges can be divided into three different entities: the newly attached End Host, the DRB and all other End Hosts already belonging to the topology. This test was performed using the default LSP refresh time of 30 seconds this LSP time is suggested by [11]. The topology recomputation is triggered by an LSP refresh with the same period.The first LSP generation happens 60 seconds after the beginning of the process. As expected, tests have shown (Figure 8) that convergence time is independent of the number of nodes in the topology. In fact, for the DRB the convergence time is about 30 seconds, which is the time of the emission of the first LSP by the newly arrived node. Newly arrived node converges after 60 seconds which is the time of the first topology computation. For all the other nodes, convergence time will be between 45 and 60 seconds which is due to asynchronous processing in the DRB, the newly arrived node, and the current (measured) one. So, a new arrival is detected at the worst in (2*LSP generation time) this time is relatively reasonable for an initialisation phase.

6.3.2.2. Convergence with VNI.

When using VNI, convergence conditions change. An Rbridge is considered as converged if not only his nickname is identified by all other Rbridges but also his VNI list. For VNI convergence shown in figure 9, as for the preceding case, three particular groups can be distinguished. First, the convergence for the DRB is between 60 and 120 seconds. Second, the convergence for the newly associated RBridge is 60 seconds which

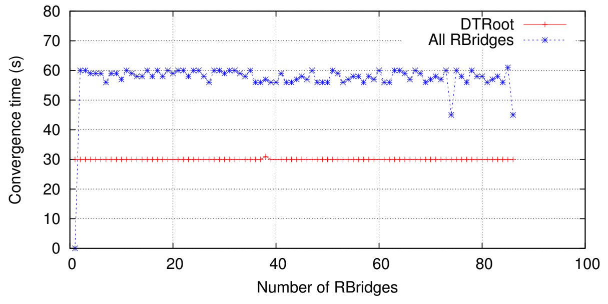


Figure 8: Convergence without VNI

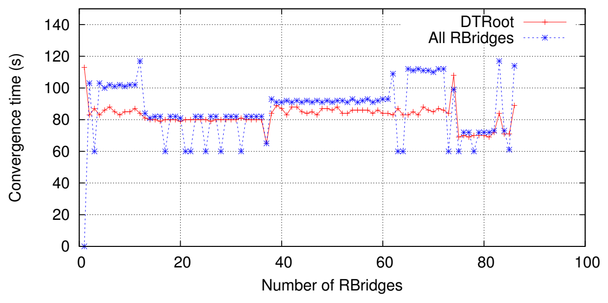
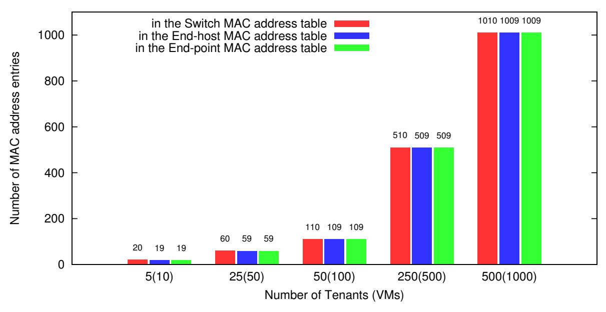
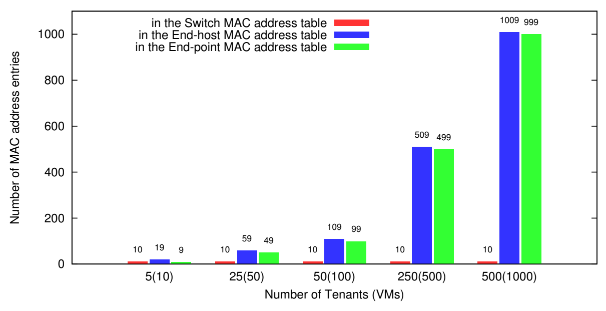


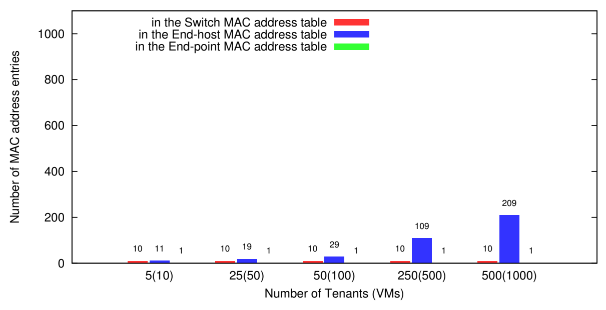
Figure 9: Convergence with VNI



(a) Standard layer 2



(b) TRILL



(c) VNT 2 vm per VNI

Figure 10: MAC address entry number

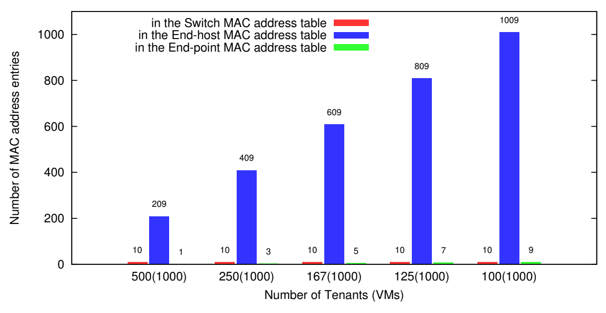


Figure 11: VNT evolution depending on number of VM per VNI

corresponds to the topology computation period. For all other nodes the convergence is between 60 and 120 seconds.

This is due to the fact that the newly associated node does not compute forwarding VNI upon startup, but while computing the topology. So, if the newly associated RBridge is not forwarding any VNI, the first correct LSP will be sent after 30 seconds uptime, but if it forwards any VNI, the first valid LSP will be sent after 60 or 90 seconds uptime depending on which asynchronous process happens first (topology computing or LSP sending). This time is also relatively reasonable regarding the fact that Endpoints are in an initialisation phase and not in communication phase.

6.3.3. Data plane evaluation

Data plane evaluation is composed of three parts. The first evaluates the impact of VNT on reducing MAC address entries in ToR, End Hosts, and Endpoints. The second one focuses on throughput measurements. Finally, the third one is dedicated to migration schemes.

6.3.3.1. Impact on MAC address table.

In this section, we compare the impact on the MAC address table size when using standard Layer 2 protocol, TRILL, and VNT. For this experiment, to assure the same conditions for the three cases, no MAC filtering rules will be used. In other words, for the standard Layer 2 case, all multicast traffic that reaches End Hosts is forwarded to Endpoints.

For this experiment we have used the 10 servers we described, starting different Endpoints on each of them. Once an Endpoint (VM) boots, it will start pinging (ICMP Ping) a non configured IP address in order to generate a continuous ARP request broadcast.

For Endpoint deployment a deterministic method is used, each time a new Endpoint is added to the least loaded End Host in rotation. Tenant Endpoints are deployed in order, tenant by tenant, which means that if each tenant has 3 Endpoints, the

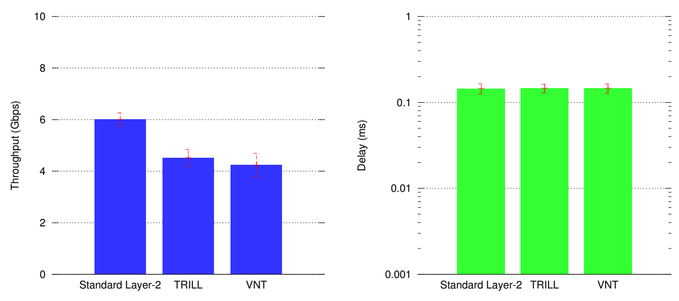


Figure 12: Basic performance tests

first tenant Endpoints will go respectively to End Host 1, 2, and 3. Second tenant Endpoints will be deployed to End Host 4, 5, 6 and so on.

Experiments have shown that the results presented in figure 10 and 11 are topology independent. The first experiment in Standard Layer 2 context has shown that when using no filtering rules at the switch, End Hosts and Endpoints will have an almost identical MAC address table size. Such a result is normal, considering that all transmitted frames are ARP requests. When using TRILL, switches will only learn server MAC addresses because Endpoints MAC addresses are hidden by the additional MAC header. Using the TRILL header will also impact Endpoint MAC address tables in a less significant way. In fact, at the Endpoint level, End Hosts MAC addresses will no longer be learned, due to the fact that at the End Host level only decapsulated frames can be forwarded to Endpoints.

Last but not least, the VNT case is highly dependent on the total number of tenants in the data center. It will also depend on Endpoint distribution among End Hosts. With our deterministic deployment method, we ensure that for tenants that have less than 10 Endpoints, all their Endpoints will be in different End Hosts. For the VNT case we have tested several scenarios with different numbers of Endpoints per tenant. These experiments have shown that at Endpoint level only MAC addresses of other Endpoints belonging to the same tenant are learned. At the End Hosts level, learned MAC addresses are those of other End Hosts, and those of all Endpoints belonging to locally supported VNI. Finally, at the switch level the only learned MAC addresses are those of the End Hosts. These MAC address table results prove the efficiency of our per tenant network abstraction. In fact, even when each entity in the network is broadcasting an ARP request, each Endpoint receives only ARP requests emitted from other Endpoints belonging to same tenant.

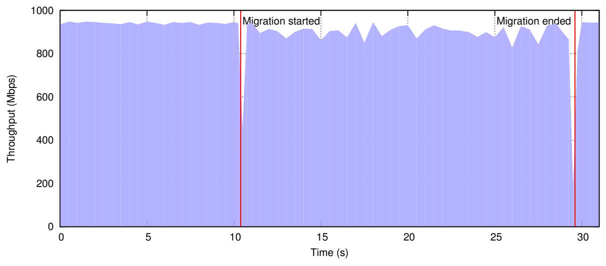
6.3.3.2. System Throughput and multipathing.

In order to evaluate Data plane efficiency it is important to evaluate the bandwidth that can be handled by a TRILL and a VNT End Host. For this test we used both netperf and iperf to send TCP flows, and computed an average value in each case. It is important to emphasize the fact that the bandwidth given are for Endpoint-to-Endpoint traffic. These bandwidth results therefore suffer from a decrease due to Xen virtualization overhead[30]. Figure 12 shows that in optimal conditions, Endpoint-to-Endpoint bandwidth in the standard Layer 2 case is about 6Gbps, while for TRILL it is about 4.5Gbps, and finally for VNT it is about 4.2Gbps. Figure 12 also shows that neither TRILL nor VNI impose any additional per packet latency, so this bandwidth decrease is a reflection of the maximum packets per second that can be handled. As described in [30], this bandwidth cap is a result of a limitation on memory transactions. This is confirmed by [31] that clearly states that the classic linux SKB mechanism is outdated, and that this mechanism consumes far more memory transactions per packet than are actually needed. Taking into consideration this linux network limitation and the additional memory transactions needed to encapsulate and decapsulate each frame, the resulting bandwidth fit our expectations.

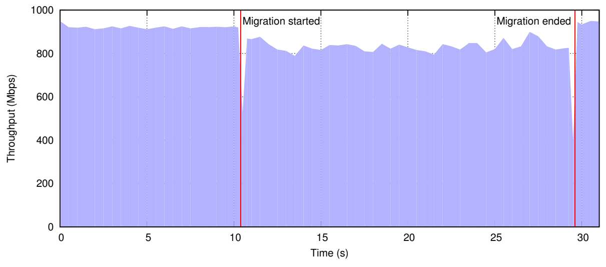
For multipathing validation we set up a special test. We used a fat tree topology where two Level 2 servers were connected with an additional 1Gb link to add an extra path. Flow ID assignment is out of scope for this paper, but for test purposes and in order to assign it in a deterministic way, Flow ID was set equal to the VLAN value. This Flow ID assignment method was chosen to guarantee a low jitter. Two Endpoints were instantiated with the same VNI, in each one two VLANs were created, and in each VLAN a 1Gb iperf flow was sent. Thus, each flow is forced to go on a different path. In the following, Path1 refers to the path using 1Gb link, Path2 to the 10Gb link, Flow1 and Flow2 represent flows on Path1 and Path2, respectively. We proceeded to perform two failure tests: for the first failure test a link on Path1 has been disabled. In this case an average service interruption of 29 seconds occurs after which Flow1 resumes on Path2. In the second failure test, a link is not disabled, but forwarding entries that route frames on Path1 are deleted from the Forwarding table. In the second test no interruption of service occurs, and Flow1 switches instantaneously to Path2. This second test has shown that interruption of services is due to the asynchronous communication between Control and Data plane. With these tests we validated that VNT still benefits from TRILL multipathing and fault tolerance assuming that Control/Data plane communication is reworked to provide instant failure notification.

6.3.3.3. Migration.

Migration and especially transparent live migration is one of the major benefits of virtual networking. Thus, ensuring that TRILL and particularly VNT do not cause any additional interruption of service is the main objective of this test. For this purpose, we have set up a testbed that does not have any significant interruption time when run in the Standard Layer 2 case. The test scenario consists of three End Hosts connected on the same 1Gb switch. We have on End Host 1, a virtual machine (VM1) with an iperf client that is measuring TCP bandwidth with an iperf server running on VM2, placed on End Host 2. Then, we migrate VM2 to End Host 3 while iperf client is still running and reporting bandwidth each 0.5 second. Test results are shown in figure 13. As expected there is no significant interruption of service. During migration time there is a decrease in bandwidth due to image transfer, and at the end of the migration there is also a decrease, probably due to a small amount



(a) Standard layer 2



(b) VNT

Figure 13: TCP flow behaviour during migration

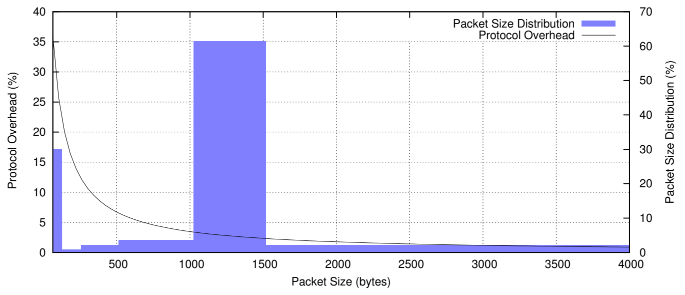


Figure 14: Protocol overhead

of packet loss. There is slight dip in the overall throughput in VNT case comparing to standard layer 2 one, this difference was expected. In fact , the throughputs are mesured Endpoint to Endpoint so the VNT overhead will reduce the total resulting bandwidth, but there is no significant change in comparison to the standard Layer 2 case, which has equivalent behavior.

This test clearly shows that VNT does not add any additional interruption time during live migration.

6.3.4. Total overhead

Overhead can be classified into two categories: first, the protocol overhead, which is the overhead caused by the additional VNT header. The second overhead is the system overhead which is caused by additional memory transactions needed to encapsulate / decapsulate VNT frames. The protocol overhead is insignificant for large frames. Indeed, with only 36 bytes of additional header, for packets of size 1500 and 9000 bytes, the overhead is respectively 2.34% and 0.39%. For small packets of size 64 bytes, however, this overhead reaches 36%, which means that VNT is better suited to traffic with large frames. An analysis of frame size within tenants privates networks in GANDI public cloud infrastructure have given the packet distribution shown in figure 14. This analysis which involves more than 400G packets, shows that more than 63% of the traffic will have an overhead below 3%.

For the system overhead, as already seen in the bandwidth section, encapsulation and decapsulation will produce some additional memory transactions that will reduce total bandwidth capacity, as described by [30].

6.3.5. Backward compatibility

As already mentioned, VNT is both Standard Layer 2 and TRILL backwards compatible. Standard Layer 2 backwards compatibility is obvious, and in fact, due to MAC-IN-MAC encapsulation, since any Standard Layer 2 switch processes only the outer MAC header it would be able to deliver frame to the correct VNT capable End Host. Concerning TRILL, VNT was designed in order to ensure both Control plane and Data plane backward compatibility.

6.3.5.1. Control plane backward compatibility.

The VNT Control plane is based on the TRILL Control plane as described in [4], adding Algorithm 1 for VNI processing. When connecting a TRILL RBridge to a VNT RBridge, the VNT RBridge will send its own supported VNI but such a TLV will not be recognized by TRILL RBridge, and will not cause any change in its behavior. On the other hand, for VNT RBridges it will recognize the TRILL RBridge as such, since it does not include any VNI TLV in its LSP. The VNT RBridge will simply consider the TRILL RBridge as supporting all existing VNI.

6.3.5.2. Data plane backward compatibility.

As mentioned in the VNT frame section, VNT frames are based on RFC[4] and RFC proposal [24] [25], and add a standard option field to the TRILL frame. Any RBridge will be able to process a VNT frame successfully, even if it does not process the VNI option field. The VNI field is marked as critical Ingress-to-Egress in order to assure network isolation. Such critical bit combination means that only the Ingress and Egress RBridges have to be VNT capable. All intermediate RBridges can ignore said option field. In this way, VNT capable RBridges and TRILL RBridges can be interconnected, assuming that only VNT capable ones are used as Ingress and Egress RBridges.

7. Conclusion

In this paper, we have designed, implemented and evaluated VNT, a Virtual Network solution built on a TRILL-based virtualized data center network.
Firstly, we designed and integrated the RBridge device on an End Host. The solution is divided into two levels: the Control and Data planes. Each RBridges Control plane will be responsible for topological information exchange with its neighbors, and for calculating the topology trees for unicast and multicast forwarding at lowest cost. The Data plane is in charge of encapsulation, decapsulation and process of forwarding the TRILL frames. Secondly, we designed VNT, a solution for creating

flexible and elastic logical networks that are free from the usual network infrastructure limitations. We accomplished this by introducing a new VNI Tag, a 24-bit segment ID transported in the TRILL frame, thus providing a large-scale LAN segment number, to up to 16 million. This large VNI is used both by the Control plane, for building the RBridges map, but also by the Data plane for frame forwarding in accordance with the VNI provisioned on the End Host.

Finally, we implemented and evaluated our proposal. This implementation, which follows the TRILL standard recommendations, is currently deployed and running in the GANDI public cloud infrastructure [28]. Our experimental results showed that a VNT Control plane simplifies topological change management and provides plug-and-play service with little configuration and fast convergence. Meanwhile, the VNT Data plane provides for frequent and transparent VM migration without change of IP address, reduces the number of Mac address entries in switch forwarding tables, increases availability, and ensures fast recovery after failures through multipathing. Moreover, VNT ensures backward compatibility with existing layer 2 equipment as well as with newer hardware supporting TRILL, thus facilitating deployability in the the current VDC infrastructure.

Upgrading the VNT frame handling mechanism from actual SKB to Netmap [31] will be a subject of our future work, and will certainly improve bandwidth performance by decreasing the memory transactions consumed by frame handling.

References

[1] http://www.gartner.com/newsroom/id/2352816, accessed 30-Oct-2013

[2] Kivity, A., Kamay, Y., Laor, D., Lublin, U., & Liguori, A. (2007, July). kvm: the Linux virtual machine monitor. In Proceedings of the Linux Symposium (Vol. 1, pp. 225-230).

[3] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T.,Ho, A.,& Warfield, A. (2003). Xen and the art of virtualization. ACM SIGOPS Operating Systems Review, 37(5), 164-177.

[4] Perlman, R., Ghanwani, A., Eastlake, D., Dutt, D., & Gai, S. (2011). Routing Bridges (RBridges): Base Protocol Specification.

[5] Perlman, R. (2004, March). Rbridges: transparent routing.In INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies (Vol. 2, pp. 1211-1218). IEEE.

[6] http://juniper.net/QFabric, accessed 30-Oct-2013

[7] Scaling Data Centers with FabricPath and the Cisco FabricPath Switching System, 2010.

[8] http://www.brocade.com/downloads/documents/technical_briefs/vcs-technical-archi tecture-tb.pdf, accessed 30-Oct-2013

[9] Perlman, R. (1985, September). An algorithm for distributed computation of a spanning tree in an extended LAN. In ACM SIGCOMM Computer Communication Review (Vol. 15, No. 4, pp. 44-53). ACM.

[10] A. Myers, E. Ng, and H. Zhang, Rethinking the service model : Scaling ethernet to a million nodes, in Proc. ACM SIGCOMM Workshop on Hot Topics in Networking, Citeseer, 2004.

[11] Information Technology-Telecommunications and Information Exchange Between Systems-Intermediate System to Intermediate System Intra-Domain Routeing Information Exchange Protocol for Use in Conjunction with the Protocol for Providing the Connectionless-Mode Network Service (ISO 8475), ISO/IEC 10589, 1992.

[12] Niranjan Mysore, R., Pamboris, A., Farrington, N., Huang, N., Miri, P., Radhakrishnan, S., ... & Vahdat, A. (2009, August). PortLand: a scalable fault-tolerant layer 2 data center network fabric. In ACM SIGCOMM Computer Communication Review (Vol. 39, No. 4, pp. 39-50). ACM.

[13] Seaman,M(2005) Shortest path bridging.Available online: http://ieee802.org/1/files/public/docs2005/new-seaman-shortest-path-0305-01.pdf, accessed 30-Oct-2013

[14] Kim, C., Caesar, M., & Rexford, J. (2008, August). Floodless in seattle: a scalable ethernet architecture for large enterprises. In ACM SIGCOMM Computer Communication Review (Vol. 38, No. 4, pp. 3-14). ACM.

[15] Elmeleegy, K., & Cox, A. L. (2009, April). Etherproxy: scaling Ethernet by suppressing broadcast traffic. In INFOCOM 2009, IEEE (pp. 1584-1592). IEEE.

[16] Mahalingam, M., Dutt, D., Duda, K., Agarwal, P., Kreeger, L., Sridhar, T., & Wright, C. (2012). VXLAN: A framework for overlaying virtualized layer 2 networks over layer 3 networks. draftmahalingam-dutt-dcops-vxlan-01. txt.

[17] Barabash, K., Cohen, R., Hadas, D., Jain, V., Recio, R., & Rochwerger, B. (2011, September). A case for overlays in dcn virtualization. In Proceedings of the 3rd Workshop on Data Center-Converged and Virtual Ethernet Switching(pp. 30-37). ITCP.

[18] McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., ... & Turner, J. (2008). OpenFlow: enabling innovation in campus networks. ACM SIGCOMM Computer Communication Review, 38(2), 69-74.

[19] B. Pfaff and B. Davie, The Open vSwitch Database Management Protocol, Internet draft, draft-pfaff-ovsdb-proto-00, Nicira Inc., accessed 30-Oct-2013

[20] Jayaram Mudigonda, Praveen Yalagandula, Jeff Mogul, Bryan Stiekes, and Yanick Pouffary. 2011. NetLord: a scalable multi-tenant network architecture for virtualized datacenters. In Proceedings of the ACM SIGCOMM 2011 conference (SIGCOMM '11). ACM, New York, NY, USA, 62-73. DOI=10.1145/2018436.2018444 http://doi.acm.org/10.1145/2018436.2018444

[21] Sridharan, M., Pearson, M., Ganga, I., Lin, G., Thaler, P., Tumuluri, C., ... & Wang, Y. S. (2013). NVGRE: Network virtualization using generic routing encapsulation.

[22] Eastlake, D., & Krupakaran, K. V. (2013). TRILL Smart Endnodes draft-perlman-trill-smart-endnodes-01.

[23] K. Ishiguro, "Quagga Software Routing Suite." Online. Available: http://www.nongnu.org/quagga/ , accessed 30-Oct-2013

[24] Eastlake, D., Yizhou, L., Bestler, C., Manral, V., & Ghanwani, A. (2012). TRILL: Header Extension.

[25] Eastlake, D., Bestler, C., Manral, V., & Ghanwani, A. (2012). RBridges: Further TRILL Header Extensions.

[26] OpenSolaris is now officially dead. RIP. Online. Available: http://blogs.everycity.co.uk/alasdair/2010/08/opensolaris-is-now-officially-dead-rip/, accessed 30-Oct-2013

[27] http://web.archiveorange.com/archive/v/xWzZygnUNYVHHzVuQXFH , accessed 30-Oct-2013

[28] http://www.gandibar.net/public/Virtual_Network_over_TRILL.pdf, accessed 30-Oct-2013

[29] https://github.com/Gandi/ktrill/tree/dev_v3.11, accessed 30-Oct-2013

[30] M. Bourguiba, K. Haddadou, and G. Pujolle, "Packet Aggregation Based Network I/O Virtualization for Cloud Computing", Elsevier Computer Communication (ComCom), Elsevier Science, ISSN: 0140-3664, Volume 35, No. 3, pages 309-319 (2012).

[31] Rizzo, L. (2012, June). netmap: a novel framework for fast packet I/O. InProceedings of the 2012 USENIX conference on Annual Technical Conference(pp. 9-9). USENIX Association.