Fábio José Correia Júlio

Licenciado em Ciências da Engenharia Electrotécnica e de Computadores

A Layer 2 Multipath Fabric using a Centralized Controller

Dissertação para obtenção do Grau de Mestre em Engenharia Electrotécnica e de Computadores

Orientador : Pedro Amaral, Professor Auxiliar, FCT-UNL

Presidente: Prof. Doutor José Manuel Matos Ribeiro da Fonseca
Arguente: Prof. Doutor Paulo da Costa Luís da Fonseca Pinto
Vogais: Prof. Doutor Pedro Miguel Figueiredo Amaral

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Setembro, 2013

A Layer 2 Multipath Fabric using a Centralized Controller

To the greatest man I have ever met, my Grandfather.

Agradecimentos

Gostaria de agradecer a todos aqueles que de alguma forma participaram e estiveram presentes na minha vida académica. Todos eles, sem qualquer excepção, fizeram com que todas as horas de trabalho e divertimento tivessem mais valor e fossem recompensadas com sucessos e alegrias.

Obrigado a todos os Professores, Amigos, Colegas, Funcionários e Conhecidos porque sem eles esta jornada não teria sido a mesma coisa.

Resumo

Actualmente, a comunicação em redes *datacenters* é feita usando Ethernet. Estas redes são usadas para fazer a comunicação em serviços de *cloud computing* e como tal devem estar preparados para suportar mobilidade de *hosts* enquanto usam eficientemente a largura de banda disponível com o mínimo custo possível.

Algumas das principais características pretendidas em *datacenters* acabam por não ser satisfeitas usando os protocolos existentes na Ethernet. O desperdício de largura de banda disponível é causado pelo uso do *Spanning Tree Protocol* (STP) e a escalabilidade é afectada devido à aprendizagem de endereços MAC por inundação de pacotes.

Estes problemas podem ser resolvidos com o uso de soluções presentes na camada 3, nível de rede, mas tornam todo o processo mais complicado e aumentam também a dificuldade em relação à mobilidade de hosts. A grande maioria destas soluções utiliza ECMP (Equal Cost Multipath) no cálculo de caminhos.

A solução proposta neste documento passa por calcular caminhos usando plano de controlo implementado num controlador OpenFlow e usando politicas em vez de custos unitários.

O Openflow é um novo protocolo que permite testar novos protocolos em redes reais sem afectar o tráfego real. Isto é feito através de ligações SSL entre os *switches* e o controlador, onde irá estar disponível toda a informação necessária.

O algoritmo de cálculo de caminhos baseado em políticas atribui uma política diferente a cada ligação em vez de usar uma métrica unitária. Com isto é possível ter caminhos com número de ligações diferente mas com a mesma preferência. É também usado o mecanismo introduzido pelo *Provider Backbone Bridging* (PBB) para conseguir isolar o tráfego da rede e o tráfego dos *hosts* aumentando assim a escalabilidade.

Palavras Chave: Data Center, Routing, OpenFlow, Políticas, Tratamento de Falhas.

Abstract

Ethernet is the most used L2 protocol in modern datacenters networks. These networks serve many times like the underlying infrastructure for highly virtualised cloud computing services. To support such services the underlying network needs to be prepared to support host mobility and multi-tenant isolation for a high number of hosts while using the available bandwidth efficiently and maintaing the inherent costs low.

These important properties are not ensured by Ethernet protocols. The bandwidth is always wasted because the spanning tree protocol is used to calculate paths. Also, the scalability can be an issue because the MAC learning process is based in frame flooding. On layer 3 some of this problems can be solved, but layer 3 is harder to configure, poses difficulties in host mobility and is more expensive.

Recent efforts try to bring the advantages of layer 3 to layer 2. Most of them are based in some form of Equal-Cost Multipath (ECMP) to calculate paths on data center network. The solution proposed on this document uses a different approach.

Paths are calculated using a non-ECMP policy based control-plane that is implemented in an OpenFlow controller.

OpenFlow is a new protocol developed to help researchers test their new discovers on real networks without messing with the real traffic. To do that OpenFlow has to be supported by the network's switches. The communication between systems is done by SSL and all switches features are available to the controller.

The non-ECMP policy based algorithm is a different way to do routing. Instead of using unitary metrics on each link, one policy is chosen for each link. The use of policies opens the possibility to consider very different paths as having the same forwarding preference increasing the number of used paths. Our approach uses the recent Backbone Provider Bridging (PBB) standard that adds extra header information to the Ethernet frame and provides isolation between customer and network address space improving scalability.

Keywords: Data Center, Routing, OpenFlow, Policies, Fault Tolerance.

Acronyms

AMSTP *Alternative Multiple Spanning Tree Protocol*

ARP *Address Resolution Protocol*

B-VID *Backbone VLAN Identifier*

BCB *Backbone Core Bridges*

BEB *Backbone Edge Bridge*

BPDU *Bridge Protocol Data Unit*

C-Tags *Costumer Tags*

C.D.F. *Cumulative Distribution Function*

CIST *Common Instance Spanning Tree*

DHT *Distributed Hash Table*

ECMP *Equal-Cost Multipath*

I-SID *Service Identifier*

IEEE *Institute of Electrical and Electronics Engineers*

IETF *Internet Engineering Task Force*

IP *Internet Protocol*

IS-IS *Intermediate System to Intermediate System*

IT *Information Technology*

LDM *Location Discovery Messages*

LDP *Location Discovery Protocol*

MAC *Media Access Control*

MOOSE *Multi-level Origin-Organised Scalable Ethernet*

MSTP *Multiple Spanning Tree Protocol*

NAT *Network Address Translation*

ns-2 *Network Simulator 2*

ns-3 *Network Simulator 3*

OSPF *Open Shortest Path First*

PB *Provider Bridging*

PBB *Provider Backbone Bridging*

PMAC *Pseudo MAC addresses*

QoS *Quality of Service*

RBridges *Routing Bridges*

RCB *Root Controlled Bridging*

RSTP *Rapid Spanning Tree Protocol*

S-Tag *Service Provider Tag*

SEATTLE *Scalable Ethernet Architecture for Large Enterprises*

SPB *Shortest Path Bridging*

SPVID *Shortest Path Virtual LAN Identifier*

STP *Spanning Tree Protocol*

TA *Tree Advertisement*

ToR *Top of Rack*

TRILL *Transparent Interconnection of Lots of Links*

VCS *Virtual Cluster Switching*

VLAN *Virtual Local Area Network*

VM *Virtual Machine*

VNI *VXLAN Network Identifier*

VTEP *VXLAN End Point*

VXLAN *Virtual eXtensible Local Area Network*

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Ethernet is a layer 2 infrastructure of choice for most campus, service provider and data center networks. Much of the appeal of Ethernet comes from its "plug and play" nature and the fact that it is a commoditized technology with low cost. However, its growing use in larger networks with a high number of end host belonging to different tenants has made it necessary to improve its scalability and traffic isolation capabilities. One of the first standards introduced to Ethernet was 802.1Q. This standard defines VLANs, which is a way to separate traffic using a VLAN identifying tag. However this tag of 12 bits only allows the existence of 4094 different VLANs which limits the scale for service provider and data center networks. A subsequent standard (802.1ad) usually called Provider Bridging (PB) introduced a second provider tag, equal to the VLAN tag but used to separate traffic at the provider level. This created a second level of separation where several client VLANs can be transported in the same backbone VLAN. However provider bridging maintained the need for all switches in the network to learn host MAC addresses which is very limiting in terms of scale. To solve this the 802.1ah standard, called Provider Backbone Bridging (PBB), isolates provider traffic using frame encapsulation thus creating two address spaces, one for client host and the other for backbone switches. This separation provided by PBB makes Ethernet function in a simpler way, the hosts addresses do not have to be learned by every switch, since different addresses are used in the backbone. With the new header hosts information is protected when forwarded on providers network. Although these standards have increased Ethernets scalability, it still depends on network wide

controlled flooding of frames for host discovery. This introduces the need to have a way of preventing flooding loops and for this purpose the Spanning Tree algorithm (STP) is used in one of its several forms. However, STP has several disadvantages: It does not use all the network links installed and therefore wastes bandwidth; its paths are inefficient for paths that do not end in the root node; and it converges very slowly to a solution. In modern service provider and data center networks these limitations become critical. For example, the bisection bandwidth is of extreme importance in datacenter networks that support distributed cloud computing architectures and with the construction of one spanning tree not all installed links are usable, this off course wastes available bandwidth with an inherent cost. Other problem is the slow path computation. When a failure occurs, it is important that the network re-converges quickly, contrarily to what happens with STP protocols. Cloud computing service supported by data center networks rely on rapid share of information between multiple hosts and equal separation of traffic flows between available servers. To have this, data center networks need to support a high number of connected virtual and physical devices and automated virtual machine migration.

The use of layer 3 protocols provides solutions for some of these problems. In service provider networks, L2 services can be transported on top of IP/MPLS networks; at data center level, L3 designs using Clos topologies or "fat-trees" can use routing protocols to provide an equal cost multipath (ECMP) fabric with advantages in terms of bandwidth use and failure re-convergence time when compared with L2 STP based networks. However, using a L3 network close to the hosts has also some drawbacks: the native L2 domains are quite small and each individual domain must be managed separately with different IP address space; IP/MPLS networks are quite complex and more expensive to run and manage; configurations have to be updated when the design changes and ensuring service continuity when host location changes can be tricky (like a VM migration in a data center network).

For these reasons the focus of this work is on L2 networks although many of the described solutions could be adapted to a L3 fabric.

Several standards and research proposals have tried to overcome the limitations of STP and bring some of the advantages of L3 networks to L2 fabrics. Almost all of them use

some form of equal cost multipath routing control plane that is applied to L2 maintaining MAC based forwarding and a single L2 domain.

ECMP is a well known problem and known routing protocols can be used for the control plane, such as IS-IS. However, ECMP routing only allows the use of paths with the same cost and this cost is usually expressed by a metric value (e.g. hops). This can provide a good bisection bandwidth use in regular topologies like Clos networks that are commonly used in data centres, but can still limit the use of the path diversity of a less regular topology like the ones that can be found in service provider networks. Non-ECMP can bring added flexibility for engineering networks and exploring new topology designs that can provide high bandwidth use with smaller costs.

1.1 Motivation

In this work we propose a non-equal cost multipath L2 fabric. Paths are selected based on a policy value that grades their preference. Paths with equal values might be different in the number of hops, bandwidth, etc. thus providing non-ECMP. Paths values are calculated from link attributes that can be seen as policies characterizing the forwarding behaviour. In a previous work [ABP13] was developed a formal model based in routing algebras [GG08] that provides a way to prove the correct operation of multipath based in policies in distributed destination-based forwarding. In this work we use these results to design the path calculation mechanism to be applied in the fabric, and prove that correct operation occurs for several possible control plane implementation solutions. Our L2 fabric should be able to scale to thousands of MAC addresses, provide a mechanism for isolation between the network traffic of different tenants and be easy to maintain and operate. To accomplish that it supports PB and PBB as a means to isolate tenants and provide scale, and uses an OpenFlow [5] controller and OpenFlow controlled switches to implement the control plane and create the services between host facing switches in an automated way.

1.2 Objectives and Contributions

The main contributions of this work are the design of a L2 fabric that features policy based non- ECMP multipath and the implementation of a simulation prototype that uses OpenFlow. The prototype can serve to test the design and one of its components is an OpenFlow controller that can be ported to a real implementation. We provide a comparation of the performance of our policy based non-ECMP control plane versus a simple ECMP solution.

1.3 Structure of the Dissertation

This dissertation is split in 5 different chapters. The first one, Introduction, as its name indicates, introduces the readers on the dissertation theme and also names the proposed objectives to accomplish.

On the second chapter all the work done to solve equal goals is described (Related Work). There we revisit STP and its variants but also more recent standardized approaches like SPB (Shortest Path Bridging) and TRILL (Transparent Interconnection of Lots of Links). The academical side was also studied with important papers describing protocols like PortLand or SEATTLE (Scalable Ethernet Architecture for Large Enterprises).

The third chapter, Architecture Characteristics, has all details behind the decisions made on this architecture construction. It is explained how scalability in the network is guaranteed and the transparency of this method to the hosts. It is also explained how failures are treated and the algorithms used to compute paths at the beginning and when some of these failures occur.

The fourth chapter, Simulation Results, is focused on the experimental part of the work. It is explained with what tools the architecture was developed and the limitations inherent to them. All the graphics from executed tests were also represented. The tests made were focused on the number of different and disjoint paths and how the controller responded when failures occurred.

The fifth chapter, as the first, has its purpose indicated on its title, Conclusions. All lessons learned, goals achieved and the not so good parts are analysed on this chapter.

It is also referred some future work that can be continued from the one described in this document.

Chapter 2

Related Work

2.1 Introduction to VLANs and Encapsulation

The traditional way to isolate tenants (customers or customer networks linked to a data center or service provider network) that share a layer 2 Ethernet network is to use VLANs to abstract an isolated L2 network. Each VLAN is identified by a tag and the frame format is defined in the IEEE 802.1Q standard.

2.1.1 IEEE 802.1Q - Bridging

Bridging[Aca13a] allows communication between hardware devices, normally called bridges or switches. IEEE 802.1Q specifies the use of C-Tags (Costumer Tags) to identify the specific VLAN of a given frame.

The C-tag is a 2 bytes field composed by: 3 bits to define the priority of the frame; 1 bit to establish Quality of Service (QoS); and the remaining 12 bits to be used for the VLAN ID. This tag was added to IEEE 802.1Q frame format with another field that comes right before it. This field is composed by 2 bytes and defines the tagging protocol used. It is possible to view an illustration of C-Tag frame format in figure 2.1.

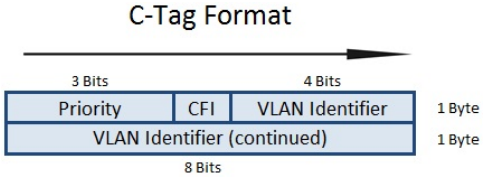


Figure 2.1: C-Tag Frame Format (adapted from [Aca13a])

This is used both in metro networks separating traffic between customers and in datacenter networks where it can be extended to virtualized settings if the hypervisor is able to encapsulate the VM traffic with the correspondent VLAN tag.
It is simple and works fine in terms of separating address space and traffic. It still suffers from some scalability problems. Although it can separate user address space all host MAC addresses are still exposed to all bridges and there is a maximum limit for the number of VLANs (4096), which limits the amount of different separated clients. IEEE 802.1ad solved this later problem by adding a second VLAN tag.

2.1.2 IEEE 802.1ad - Provider Bridging (PB)

The IEEE 802.1ad, also known as Provider Bridging (PB) [Cis10], is an evolution of IEEE 802.1Q. Control and traffic frames are transported transparently using this method. One more VLAN tag, named S-Tag (Service Provider Tag), is added before the C-tag. This helps increasing the number of distinct VLANs available on the network. Although both tags are equal (only the name changes because the fields are similar) each one has a different role. Costumer and provider traffic are defined by C-tag and S-tag respectively [Aca13b].
An S-tag with a unique identifier is attached to all frames that go from a costumer to the service provider network . This tags are assigned and removed by service Provider Bridges (S-Bridge) at the ingress or egress points of the provider network. All frames sent from a certain Service Provider Bridge, will have the same S-tag.
Customers differentiate a frame using the C-tag. All VLAN's of a customer are then tunnelled through provider network using the S-tag.
This standard solves the number of VLANs scaling issue by adding extra space for tenant isolation. However in some networks this number can still be insufficient. A problem not solved is the MAC addresses exposure in the provider bridges.

2.1.3 IEEE 802.1ah - Provider Backbone Bridging (PBB)

Compared to PB, Provider Backbone Bridging (PBB) [Sys10] removes the exposure of backbone bridges to host MAC addresses providing much better scalability.

An additional header makes a strong demarcation between service provider and costumer networks. This new header contains the B-VID (Backbone VLAN Identifier), which is identical to the S-Tag in PB, and adds the I-SID (Service Identifier) to identify the service instance using a 24 bit field.

On an ingress BEB (Backbone Edge Bridge) two different components are present, the I-Component and the B-Component. The first associates the C-Tags with the I-SIDs and the second associates I-SIDs to the corresponding B-VIDs that will transport the traffic. On an egress BEB the inverse process is performed. PBB networks can transport traffic from either client networks containing a single VLAN tag (C-tag) or from PB networks containing a C-Tag and a S-Tag (figure 2.2 illustrates this second scenario). The initial frame (from costumer) is encapsulated by the new header (service provider) in the BEBs, making its transport "transparent" through the network. Then frames are forwarded by BCB (Backbone Core Bridges) to an egress BEB that can retrieve the primary frame. The BCBs no longer have to know the customers MAC addresses to forward frames. This improves the number of customers supported by the network, and core devices can be simpler and cheaper because the memory and processing capabilities are lesser. Only the BEBs need to know more than only the backbone addresses [Sys10].

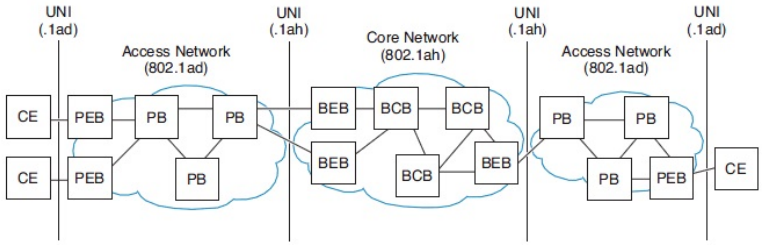


Figure 2.2: Provider Backbone Bridge Network (adapted from [Cis09])

IEEE 802.1ah interconnect many PB networks without modifying its information and the MAC address learning requirements are reduced using MAC-in-MAC encapsulation. Also, PBB ends the evolution, at this point, of the frame format as it is illustrated in the figure 2.3.

PBB solves some of the scaling issues of Ethernet and forms a good base for use in networks with a high number of hosts and different client networks. However it does not address the problems of using STP to control the flooding and learning process. In the next sections

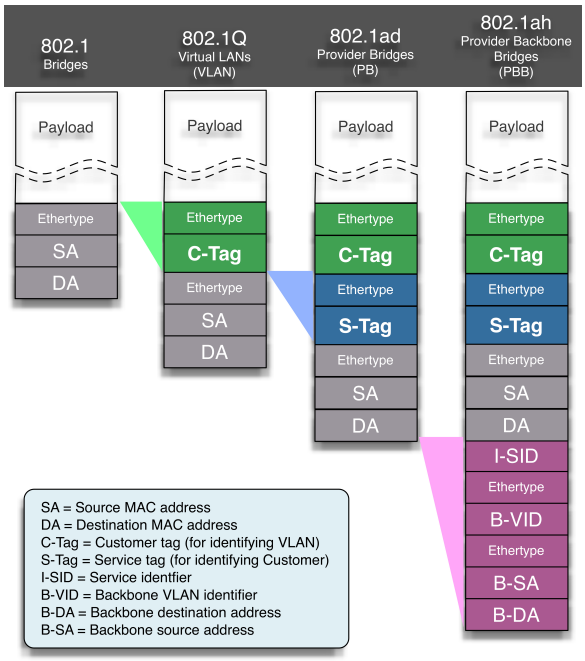


Figure 2.3: Evolution of Frame Format (adapted from [Sea05])

the several STP flavours and improvements are presented.

2.2 IEEE 802.1D - Spanning Tree Protocol (STP)

STP (standardized as IEEE 802.1D [IEE04]) was created to avoid loops in an Ethernet network topology. A single tree is calculated with only one path between any pair of nodes with all other possible paths being blocked [AB12].

The spanning tree is calculated using information learned and exchanged by all network nodes with the transmission of special messages called BPDUs (Bridge Protocol Data Units), containing, among other information, three fields:

- **Root ID -** ID of the root node when the message is sent;
- **Transmitting ID -** ID of the node that transmitted the message;
- **Cost -** Cost of the path from the transmitting node to the root node.

When the Cost field is 0 that means the transmitting and the root node are the same [Per00].

Beyond the configuration messages there are also Topology Change BPDUs. These messages are sent to the root every time a change is identified in the network topology. After

that, the root node has to inform all the nodes of this change by sending an equal message. The root node is the switch with the smallest ID. All available paths in the spanning tree from a device to the root switch have the minimum cost among all possible paths in the network. This cost represents the sum of the costs of the segments on the path. Ports that not belong to this minimum cost paths are blocked and frames received by them will be discarded.

Although this protocol is still in use: it cannot use all the possible paths decreasing the available bandwidth and wasting resources; the path can be suboptimal because the traffic always has to pass in the root node; and when a change occurs in the topology, all traffic exchange has to be suspended while the new spanning tree is not calculated.

To improve the protocol and solve these problems some solutions were designed, but the basis of STP was always kept.

2.2.1 IEEE 802.1w - Rapid Spanning Tree Protocol (RSTP)

The Rapid Spanning Tree Protocol (RSTP) was defined to solve the STP slow convergence problem [Woj03].

The STP has several different port states: Blocking - a port that can cause switching loops; Listening - a port waiting for BPDUs; Learning - a port learning MAC addresses and populating the forwarding tables; and Forwarding - a port receiving and sending frames. The Listening and Learning states are important to ensure that all information is spread on the network and the right ports are blocked. Only after that, the port state changes to Forwarding. Four improvements were made in RSTP to make this process faster:

- the number of port states changed from five to three, Discarding (this state does not learn addresses or forward traffic), Learning (learns addresses but does not forward traffic) and Forwarding (learns addresses and forwards traffic);
- two new port roles were added to substitute the Blocking role, Backup (it is the alternative port for the Designated port if it goes down) and Alternate (it is an alternative port to the Root port if it goes down);
- the BPDU has only one type, similar to STPs Configuration BPDU but with a

type field (to distinguish what version is used) and a flag field carrying additional information;

- the message exchanging was simplified, because now any node can send information when a topology change happens.

Besides all improvements, the initial converge time (when the network is established) is equal to STP. It only decreases when the tree has to be computed again. In this case a bridge uses a negotiation mechanism to assure that no loops are formed in the downstream bridges. This allows the change of the designated ports to the forwarding state without the listening state, making it possible to re-converge in the time that BPDUs take to reach all bridges. RSTP provides a better convergence time but still only one path is available for traffic. In traditional STP all VLANs in a network use the same tree and therefore blocked ports are not used for forwarding in any of the VLANs. MSTP addresses this issue allowing the definition of different spanning trees for different VLANs.

2.2.2 IEEE 802.1s - Multiple Spanning Tree Protocol (MSTP)

MSTP [Net13] is another evolution of STP that allows the distribution of VLANs in several spanning tree instances.

MSTP divides the network into regions. The regions have a configuration name and the mapping of all the spanning tree instances used in it. MST then maps VLANs to the available spanning tree instances. In figure 2.4 it is illustrated a simple MSTP topology divided by regions.

This approach can allow a better use of network links since blocked ports in one instance can be forwarding in other instances. However the number of possible instances is very low and its configuration very complex. If some PB and PBB procedures were used, maybe MSTP has a better performance.

2.3 Non-STP Standards

Two standards have emerged to address the STP limitations: TRILL [Eas10] and shortest path bridging [AB12]. They have slightly different approaches, but they both rely

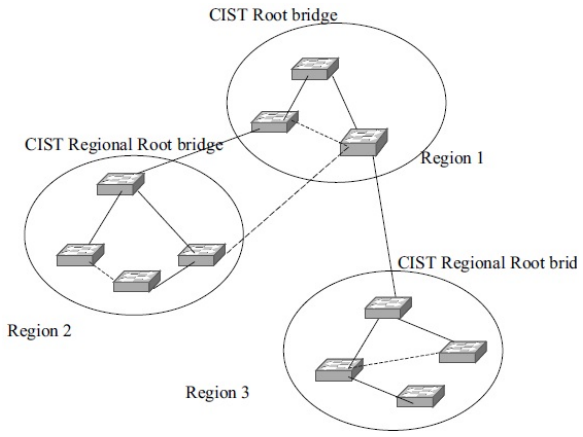


Figure 2.4: Example of MSTP regions (adapted from [IGA04])

on encapsulation (MAC-in-MAC) for scale and traffic separation and have an equal cost multipath (ECMP) approach based on a traditional shortest-path routing algorithm control plane using a link state algorithm to build the shortest bidirectional paths between bridges.

2.3.1 IEEE 802.1aq - Shortest Path Bridging (SPB)

The IEEE 802.1aq is a protocol that forwards frames through shortest paths between service end-points. It uses IS-IS [Ora04] (Intermediate System to Intermediate System) [AASB+10] for its control plane. The IS-IS for SPB has a little modification from the standard version. Additional information (VLAN identifiers) was introduced to reduce the exchange of control frames and to simplify the registration of services without using extra discovery mechanisms or further signalling.

SPB has two modes of operation: SPBV and SPBM. The SPBV is VLAN based so every shortest path tree is defined by a different SPVID (Shortest Path Virtual LAN Identifier) and the SPBM is based on the combination of MAC addresses and B-VIDs according to the standard definitions published in IEEE 802.1ah. The service identifier (I-SID) and the backbone VLAN (B-VID) are separated allowing services to scale independently of the topology. Traffic forwarding is performed as defined in the PBB standard [AB12]. The difference in shortest path bridging resides in the way B-VIDs forwarding tree are calculated. Instead of a spanning tree an ECMP approach is followed. The paths are discovered using dissimilar costs associated with links when paths are being computed.

All bridges have to use the same rules to choose the same paths and maintain the SPB congruency properties. At the moment, there can be 16 shortest path trees, achieved by manipulation of tie breaking between multiple equal cost trees.
Other important point in SPB is the loop avoidance. This is achieved by doing loop prevention and using mitigation techniques. The first method does not allow loops on the beginning of the process and the other exists to avoid damages when a loop is formed. Loop prevention depends on shortest path algorithms and the loop mitigation checks if a frame arrived on the correct port [AASB+10].
Beyond IS-IS, two other different control plane protocols for the SPB were proposed [FA09]:

- **Distance Vector SPB**;

- **Root Controlled Bridging (RCB)**;

The first solution proposed uses MSTP BPDUs to perform a distance vectoring algorithm. Every switch has its own spanning tree instance where it is the root switch. The BPDUs have all bridges of a path listed (this list is called the Path Vector) and the list is included in the end-to-end path cost, allowing different bridges to select the same path. The count-to-infinity problem, present in distance-vector protocols, when a Root Bridge failure happens is not a problem for SPB. The instance where the problem occurred became unused.
The second solution is an extension to IS-IS. The Link-state database is maintained but each bridge only computes its own shortest path tree. Using Tree Advertisement (TA) messages all bridges are informed. TA messages are only sent to bridges that belong to the tree and the unused ports are automatically blocked after it is received. This approach decreases the computational complexity compared to basic IS-IS but increases the number of control messages that have to be exchange by bridges.
In [FA09], the converging times of these three solutions was tested. The basic IS-IS is better in sparse topologies while the RCB is faster in mesh topologies because of the message exchange. MSTP based SPB is like RCB, better on mesh topologies than in sparse ones, because of its lack of alternative paths.

2.3.2 Transparent Interconnection of Lots of Links (TRILL)

TRILL is the other used solution on data centers and service provider networks. It provides Layer 2 forwarding with traffic encapsulated by a header with hop counting. The creator of STP, Radia Perlmam, proposed and the Internet Engineering Task Force (IETF) developed.

As in SPB, an ECMP algorithm and the IS-IS protocol were used for path computation and the control plane. However a different kind of devices, routing bridges (RBridges), and a special header are used.

RBridges are used to increase the throughput and capabilities of classic bridges, representing a new kind of hardware between classic bridges and routers. Classic bridges are also supported by TRILL.

The special header does not allow a granular control of traffic [DCD12]. As is illustrated in figure 2.5, TRILL data frames are encapsulated and addressed to the next hop RBridge before they are forwarded. They can be addressed for a known unicast path or to all RBridges multicast address. The VLAN identifier is not mandatory.

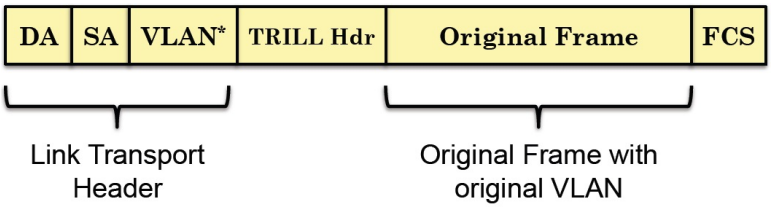


Figure 2.5: The TRILL Encapsulated Frame (adapted from [Eas10])

TRILL process is very similar to PBB but a different header is used. As illustrated in figure 2.6, TRILL Header has: two fields with RBridges names (Nickname); a field for protocol version (V); another one reserved (R); one that indicates if multi destination is allowed (M); plus one to know TRILL options length (OpLng); and the last to count until hop limit is achieved (Hop).

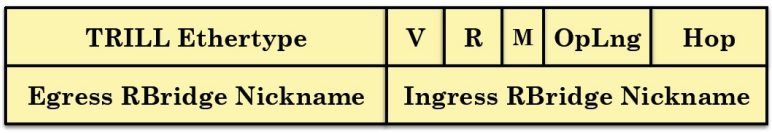


Figure 2.6: The TRILL Header (adapted from [Eas10])

Lookup/Forwarding is the big difference between TRILL and SPB. SPB uses MAC forwarding instead of a hop-by-hop forwarding model used in TRILL. MAC forwarding is simpler because it has no TTL decrement or frame check sequence recalculation or MAC swap. This is the main reason SPB is more "enterprise-friendly" than TRILL [Ava11].

However, ECMP and IS-IS are used on both approaches. On our work a non-ECMP algorithm and a centralized controller are used to build a different solution.

2.4 Research Works

Several research proposals also address this problem. On this chapter three different solutions are described.

2.4.1 Multi-level Origin-Organised Scalable Ethernet (MOOSE)

MOOSE [SMC09] is a protocol design for Ethernet to improve scalability using topology dependent hierarchical MAC addresses and a new protocol for the control plane.
New hierarchical MAC addresses are dynamically and automatically assigned to all hosts. They are assigned to frames when they arrive at the first switch (called home switch). Only the source address is modified because the destination address is expected to already be in the hierarchical form.

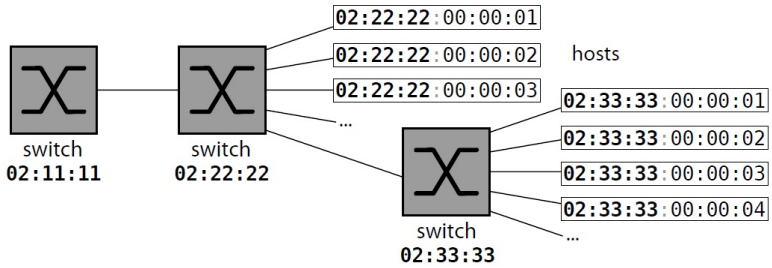


Figure 2.7: Assignment of MOOSE hierarchical addresses by switches (adapted from [SMC09])

A hierarchical address represents the switch and the host addresses combined. The three last bytes are the host identification and the first three bytes the switch where it is linked, as it illustrated in figure 2.7. The host addresses are already in MOOSE format.
MOOSE uses a variant of OSPF (Open Shortest Path First), instead of STP. To improve the performance multiple equal-cost routing paths are used by MOOSE.

Switch addresses are automatically chosen so conflicts can exist. If it happens on different networks, there is no problem but if it occurs on the same network problems can occur. The solution consists in sending a message to the conflicting switch telling that its address has to change. An ARP (Address Resolution Protocol) will be sent to all the hosts to update their caches when the switch address is changed.

Before switches can participate in the routing protocol, they have to be identified (using 802.1X - a standard for Port-based Network Access Control). These switches do not need to know all network MAC addresses, only the ones that belong to other switches. Only the *home switch* has to know what hosts are connected to it.

Finally, the mobility problem has to be resolved, MOOSE MAC addresses and the ARP cache entries of the hosts become totally wrong when a host location changes. The problem can be solved if: the frames that arrive at the old home switch are redirected to the host's new home switch until the ARP caches expire; or a broadcast "gratuitous ARP" is sent by the new home switch for immediate cache entries update.

MOOSE performance can not be commented because it was not tested. However the new hierarchical MAC is a different way of solving the switching problem. The MAC address re-mapping has a cost as well as the conflict resolution and host mobility solutions. These solutions might not scale.

2.4.2 PortLand

PortLand [NMPF+09] brings routing, forwarding and fault tolerance to Layer 2 environments using: Pseudo MAC addresses (giving a new way of addressing the network switches); ECMP to compute paths and perform load balancing; and a process called *fabric manager* (a centralized way to control the ARP protocol, resolve failures and deal with the multicast forwarding).

Portland is commanded by the *fabric manager*. It can be installed in a network host or in a controller outside the network using a protocol like OpenFlow (that will be explained in a later section) to communicate. It will receive network state updates so administrator configuration is not needed.

To reduce the difficulty to locate hosts positional Pseudo MAC addresses (PMAC) is used.

They are assigned to frames by ingress switches and the ARP protocol shares them with the rest of the hosts. In order to maintain this process unknown to hosts, the Actual MAC addresses (AMAC) conversion to the PMAC addresses is done by the egress switches. The PMAC addresses are composed by 48 bits of form *pod.position.port.vmid*. The *pod* is the pods number of the edge switch, the *position* is the position in the pod, the *port* is the switch local view of the port number the host is connected to and the *vmid* to identify different virtual machines.

These ingress and egress switches, plus the ones that are not linked to hosts, cannot forward or receive frames until their locations are established. A Location Discovery Protocol (LDP) is used to make all switches periodically send Location Discovery Messages (LDM) to all its ports with information about them. These messages contain the switch identifier, pods number, its position, tree level where it is located and if it is linked to an up/down port.

With the information received by LDMs and PMAC addresses, switches have all information needed to forward a certain frame. If the pod indicated in the frame is not the one where the switch is located the frame is sent to an upper level but if the pod is the same it is forwarded to its correspondent port. Other advantage is the centralized controller, *fabric manager*, use. It has information at one place from all network switches to improve faults handling.

However it uses an ECMP algorithm (as all research works) and, as MOOSE, use hierarchical addressing with MAC-Address rewriting that only works in multi rooted tree topologies since forwarding is based in the address hierarchy.

2.4.3 Scalable Ethernet Architecture for Large Enterprises (SEATTLE)

SEATTLE [KCR11] uses a link state algorithm to distribute switch level topology information and build shortest paths between switches for forwarding. It uses encapsulation and a distributed resolution mechanism to map hosts to switches. SEATTLE developers believe that it gives the best of two worlds, the scalability of IP plus the simplicity of Ethernet.

The link-state algorithm has the ability to: distinguish which links are attached to hosts

and to switches; compute shortest paths; and inform network switches when a change in the topology occurs.

To reduce lookup complexity and simplify certain aspects of network administration, a network level DHT (Distributed Hash Table) is used. With this approach, the maps are stored at switches to facilitate systems response to network failures. Also, switches and respective identifiers are mapped using a hash function.

In large networks that are susceptible to frequent topological changes, running a single network-wide link-state routing protocol can increase message exchange and decrease its reliability. It is possible to divide the network into regions with a multilevel one-hop DHT. This different configuration connects all regions to a backbone region composed by border switches from all regions. Thus, the overall lookup latency is reduced and the links failures/recovery are constrained to the region where they occur, eliminating the unnecessary sending of information frames.

The backbone region needs more powerful equipment than the others because it stores all hosts linked in the network unlike other regions where only their own hosts are kept. Instead of storing key-value pairs an optimization was proposed where backbone switches will relay them to its respective regions to know if they belong there.

The ARP protocol is replaced by an extension to the one-hop DHT directory service that allows enhancement of scalability. Instead of only returning the MAC address of a certain IP address, the ARP protocol returns the MAC address plus the switch location.

All mobility problems are resolved keeping the directory service up-to-date with three simple functions, insert, update and delete. Despite this, other hosts have to be informed of the alterations. A gratuitous ARP request is sent to hosts that need to update this information.

SEATTLE can be installed on equipment already in use. However it has some complexity and the implementation of the DHT has a scaling cost. Once again the control plane is a shortest path ECMP based solution.

Some of these research proposals use centralized controllers for path calculation and control plane management. One of the solutions for interfacing between centralized controllers and switches is OpenFlow [MAB+08], a protocol that can communicate with net-

work switches and control their flow tables. We use OpenFlow in this work and therefore present a small description of the protocol in the following section.

2.5 OpenFlow

OpenFlow is used to control by software the forwarding behaviour of switches.
The components of an OpenFlow based architecture are illustrated in figure 2.8, they are [Fou12]:

- **the Controller -** manages the switch by sending messages with flow entries or state requests;
- **the OpenFlow Protocol -** makes the communication between switch and controller possible;
- **the OpenFlow Switch -** does the frame lookup/forwarding and communication with the controller by a Secure Channel and using the OpenFlow Protocol;

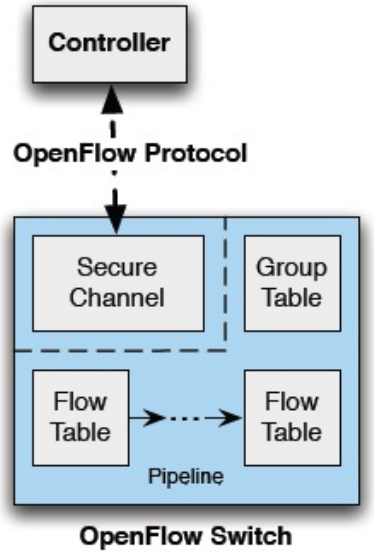


Figure 2.8: Main components of an OpenFlow Switch (adapted from [Fou12])

Flow tables reside inside OpenFlow Switches. Every flow entry is composed by three components: 12 fields with received information; a list of actions that say what to do with the matching frame; and a collection of statistics about this flow. Each flow table has a level of priority associated that decides which table is searched first.

When a frame arrives at an OpenFlow Switch, a cyclic process, illustrated in figure 2.9 starts. The frames header is extracted and it is matched against all flows in the tables. If a match occurs, the respective action is executed. Otherwise, the frame is sent to the controller using the Secure Channel and the OpenFlow Protocol and the switch waits until the controller informs what to do [Fou12]. There are many OpenFlow Controllers

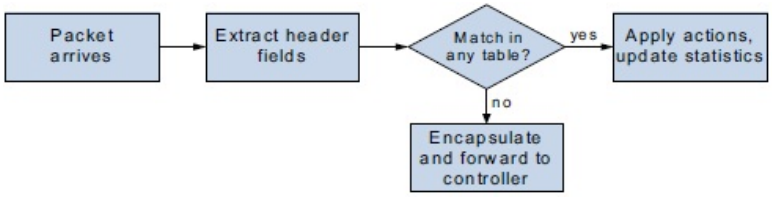


Figure 2.9: Handling of incoming frames in an OpenFlow Switch (adapted from [JOS+11])

available on the internet. Each one of them is written in a certain programming language and has its particular functions but common to all of them are the learning and discovering processes of common switches.

NOX [GKP+08] was the first controller developed. It is an Open-Source controller widely used that provides a platform for deeper software implementations in C++, it also has a version in Phyton called POX. NOX is not an OpenFlow exclusive (similar protocols can use it) and many elementar functions are already implemented. It proves complete visibility of the network topology and can operate in a prl-active (filling the flow tables before frame forwarding starts) or re-active mode (forwarding the frames to the controller).

Other well tested and Open-Source OpenFlow controller is Beacon [Eri]. It is programmed in Java and supports both event and threaded operations. Any needed modifications can be done without disconnecting the switches from the network. Also, its implementation becomes very easy and effective due to the parallelism capability of the programming language used.

Beacon is not the only controller who exploits parallelism and uses Java to do it. Maestro [Ng] uses a single-threaded programming model for application programmers. However it enables and manages parallelism as a service to. In other words, the most tedious and complicated job is already done. Such as NOX and Beacon, Maestro interfaces are provided to:

- introduce new customized control functions;
- maintain network state;
- composing control components;

One of the biggest problems in OpenFlow is its speed. In [JOS+11] a model was proposed to evaluate the forwarding speed and blocking probabilities in an OpenFlow network. All executed tests showed that the most important thing is the controller processing speed to deal with high number of unmatched flows being sent to the controller.

Chapter 3

Architecture Characteristics

We start by describing the control plane that supports the non-ECMP path selection based on policies. We then describe the architecture itself and finally present other implementation possibilities that also lead to a correct behaviour of the system.

3.1 Control Plane

On [Ama12][Cha13][ABP13] an algebraic routing model was used to study how the correct behaviour of multipath, policy based, routing can be achieved. Correct behaviour consists in reaching a stable set of paths in finite time, and having loop free forwarding when each of the nodes forwards frames per destination along those paths.
The first step is to define the set of policies to use. There is usually a separation between access nodes connected to hosts or other networks, and backbone areas that interconnect access nodes to transport traffic. This hierarchy can have several levels and nodes can be distributed amongst them in whatever way the operator wants. With this in mind we defined a very simple set S containing three types of values that can be applied to edges of the network graph and that express the policies to be used in finding and selecting paths:

- D_W - links in the downwards direction of the hierarchy;
- U_W - links in the upwards direction;
- S_L - links between nodes at the same level of the hierarchy.

According to the conditions referred on [Ama12] correct operation is
closely related with graph circuits. Graph circuits are paths where all switches are different except the first and the last and can exist in networks where an infinite number of paths have the minimal weight.
Correctness is assured if all circuits found in the graph are not labelled in one of the following manners:

1. No circuit contains only D_W links;
2. No circuit contains only U_W links;
3. No circuit contains a sequence of three or more consecutive S_L links.

The first two cases make no sense if we establish a hierarchy in the network graph. The third case is too restrictive on the use of S_L links. One way to circumvent the restriction (and allow paths with any number of consecutive S_L links) is to add a secondary metric to those links, enlarging the set of policies and defining a new composition operation - a new algebra.
The idea is to impose a preference order (the secondary label, L or R, with R being more preferred than L) over the links S_L. Instead of the same-level links being labelled with the S_L label in both directions, they are now labelled with different values in each direction using the secondary label. The $\otimes$ operation for this secondary algebra is represented on table 3.1.

$\bigotimes_T$	$\emptyset$	**L**	**R**
$\emptyset$	$\emptyset$	L	R
L	L	L	L
R	R	$\emptyset$	R

Table 3.1: $\bigotimes_T$ Operation

The idea behind policy routing is that two paths can be considered equal when their policy values are the same. As an example consider that the result of $D_W \bigotimes D_W$ is D_W. Then one path with two D_W links has the same forwarding preference than another with

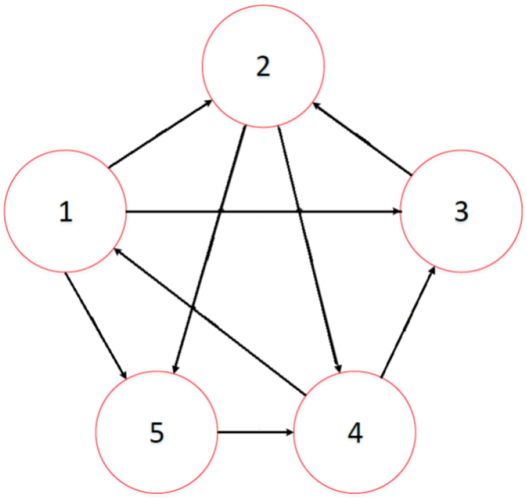


Figure 3.1: A simple Topology with five nodes

five D_W links. This means that the value of the path does not take into account the number of links, the bandwidth or any other metric but only the policies applied to each of its links. This opens the possibility to manipulate the way traffic flows in a network by setting the appropriate policies, manipulating the number of available paths between each pair of nodes. It also opens the possibility to have more paths between nodes than a simple ECMP solution

To compute those paths, table 3.2 is used with all possible results for each pair of policies. The goal is to maximize the number of equal preference paths while keeping the labelling restrictions in circuits as few as possible.

$\bigotimes_S$	$\overline{1}$	D_W	S_L	U_W
$\overline{1}$	$\overline{1}$	D_W	S_L	U_W
D_W	D_W	D_W	$\overline{0}$	$\overline{0}$
S_L	S_L	S_L	S_L	$\overline{0}$
U_W	U_W	U_W	U_W	U_W

Table 3.2: $\bigotimes_S$ Operation

Using as an example the topology represented in figure 3.1, we will explain how the control plane calculates paths. All the links are directed and the link policies are defined

by the network administrator for each direction.

The links will have the policies indicated in the adjacency matrix, represented in 3.1. For correct operation, policies were chosen such that the circuit labelling leads to correct operation.

$$A = \begin{bmatrix} (\overline{1}, \emptyset) & (U_W, \emptyset) & (S_L, R) & (D_W, \emptyset) & (D_W, \emptyset) \\ (D_W, \emptyset) & (\overline{1}, \emptyset) & (D_W, \emptyset) & (D_W, \emptyset) & (D_W, \emptyset) \\ (S_L, L) & (U_W, \emptyset) & (\overline{1}, \emptyset) & (D_W, \emptyset) & (\overline{0}, \emptyset) \\ (U_W, \emptyset) & (U_W, \emptyset) & (U_W, \emptyset) & (\overline{1}, \emptyset) & (S_L, L) \\ (U_W, \emptyset) & (U_W, \emptyset) & (\overline{0}, \emptyset) & (S_L, R) & (\overline{1}, \emptyset) \end{bmatrix} \tag{3.1}$$

The equation 3.2 is used to compute, at a time, all the possible paths between a source and a destination.

$$a_{12}^{k+1} = (a_{11} \otimes a_{12}^{k}) \oplus (a_{12} \otimes a_{22}^{k}) \oplus ... \oplus (a_{1n} \otimes a_{n2}^{k}) \tag{3.2}$$

The result of the matrix iteration is the closure matrix A^* of the adjacency matrix or the result of the iteration a, each element is the result the operations present on tables $\bigotimes_T$, Path Composition, and $\bigotimes_S$, Path Selection. For example the value $a_{1,2}^k$ contains the minimal weights of paths with at most k edges.

The result contains the minimal paths weights but it is possible to obtain the paths that correspond to those weights in the course of the iteration. In the end we obtain two matrices, one with the minimal weights solution a and the other with the possible next hops to achieve the designated destination with those weights, 3.4. The value present in the matrix variable represents the number of iterations needed to calculate the final value, in this case, two.

$$A^2 = \begin{bmatrix} (\overline{1},\emptyset) & (U_W,\emptyset) & (S_L,R) & (D_W,\emptyset) & (D_W,\emptyset) \\ (D_W,\emptyset) & (\overline{1},\emptyset) & (D_W,\emptyset) & (D_W,\emptyset) & (D_W,\emptyset) \\ (S_L,L) & (U_W,\emptyset) & (\overline{1},\emptyset) & (S_L,L) & (D_W,\emptyset) \\ (U_W,\emptyset) & (U_W,\emptyset) & (U_W,\emptyset) & (\overline{1},\emptyset) & (S_L,L) \\ (U_W,\emptyset) & (U_W,\emptyset) & (U_W,\emptyset) & (S_L,R) & (\overline{1},\emptyset) \end{bmatrix} \tag{3.3}$$

The proper choice of labellings influences the amount of available paths between a pair of nodes. Using the example above, if the labelling of all links from node 3 to node 1 was U_W all paths from 3 to 1 will be equally preferred for forwarding, maximizing the number of possibilities. So the operator has the flexibility to opt between increasing or decreasing the amount of available paths, steering the traffic through a certain amount of paths for some destinations, etc.

$$\begin{bmatrix} - & 1 & 2 & 3 & 4 \\ 0 & - & 2 & \{0,3\} & \{0,2,4\} \\ 0 & 1 & - & 0 & 4 \\ \{0,1\} & \{0,1\} & \{0,1\} & - & 4 \\ \{0,1,2\} & \{0,1,2\} & \{0,1,2\} & 3 & - \end{bmatrix} \tag{3.4}$$

3.2 Architecture

The main purpose of our architecture is to encapsulate host frames into PBB frames and forward them through multipath backbone VLANs between border switches. An OpenFlow controller calculates the multiple paths between border switches belonging to the same backbone VLAN.

The end points of a L2 service of a given client or tenant are identified by the service identifier I-SID. This I-SID is mapped to a backbone VLAN indicator (B-VID) that will transport the frames. Border switches encapsulate the frames containing the client source and destination MAC addresses and the client VLAN ID (C-VLAN) in to a PBB frame

with destination and source MAC addresses of border switches, the I-SID and the corresponding B-VID.

Most part of information needed for path computing is defined by the network administrator. This information consists on: VLAN identifiers; what and how many switches are in a VLAN and which links are used and what policies do they have. Only the links between backbone switches have policies. Network switches will inform the controller with the rest of the information. That information consist in switches IDs, respective MAC addresses and connections available by them. In our architecture there are three major components: the OpenFlow controller; backbone switches; and border switches that can act as ingress or egress switches.

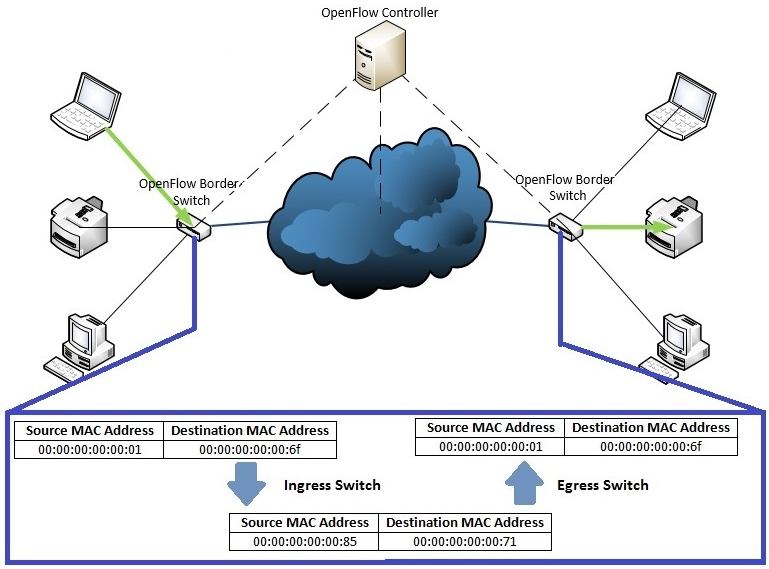


Figure 3.2: Overall view of the Implemented Architecture

3.2.1 Controller

It controls the host MAC learning process and the mapping (learning) between backbone MAC addresses and host MAC addresses in border switches. The MAC learning depends of what connections are available and also in what VLANs the switches are allowed to participate.

It calculates the multiple non-ECMP paths between B-VID endpoints and builds the flow tables for backbone switching between the backbone facing ports of border switches and backbone switches. A different calculation is performed using the non-ECMP policy based

algorithm for each B-VID. The flow tables will know the next hop to a certain destination, in other words, a switch does not know the entire path because frame forwarding is hop-by-hop destination based.

The flow tables are built for each B-VID with entries matched by destination MAC address and source and destination IP address for flow identification. The distribution of the flows by the several entries available for a given B-MAC address are made in a round robin logic. They can also have different priority levels defined.

Flow table entries can be sent to switches in two different times: when the network starts; or when a switch needs information. Both situations are valid, the difference is in the processing overhead. In the first situation the controller has to process all the possible flows on the network, even the ones that will be not used. This solution unnecessarily wastes time. The second approach only sends information to a switch when it asks for it. There is an initial latency for the first frames for a given destination but the flow table only contains active entries.

All entries sent by the controller to switches are related with the paths to reach a border switch. The backbone switches are never an endpoint so they only have to know how the border switches can be reached.

In the figure 3.3 a flowchart that explains how the controller informs the switches is represented.

3.2.2 Backbone Switch

Backbone switches forward frames based on the B-MAC of the destination border switch, the B-VID, and the source and destination IP addresses that identify a given flow. The controller calculates the multiple paths available between the border switches that belong to a given B-VID, it then stores the flow tables for that B-VID in the backbone switches. Upon reception of a frame the backbone switch looks for an entry for the destination address B-MAC in the B-VID forwarding table. More than one entry can exist due to multipath forwarding. The choice of which one to choose depends on the source and destination IP addresses that are used to identify a flow. In the first frame of a flow the switch asks the controller which port to use amongst the available multipath forwarding

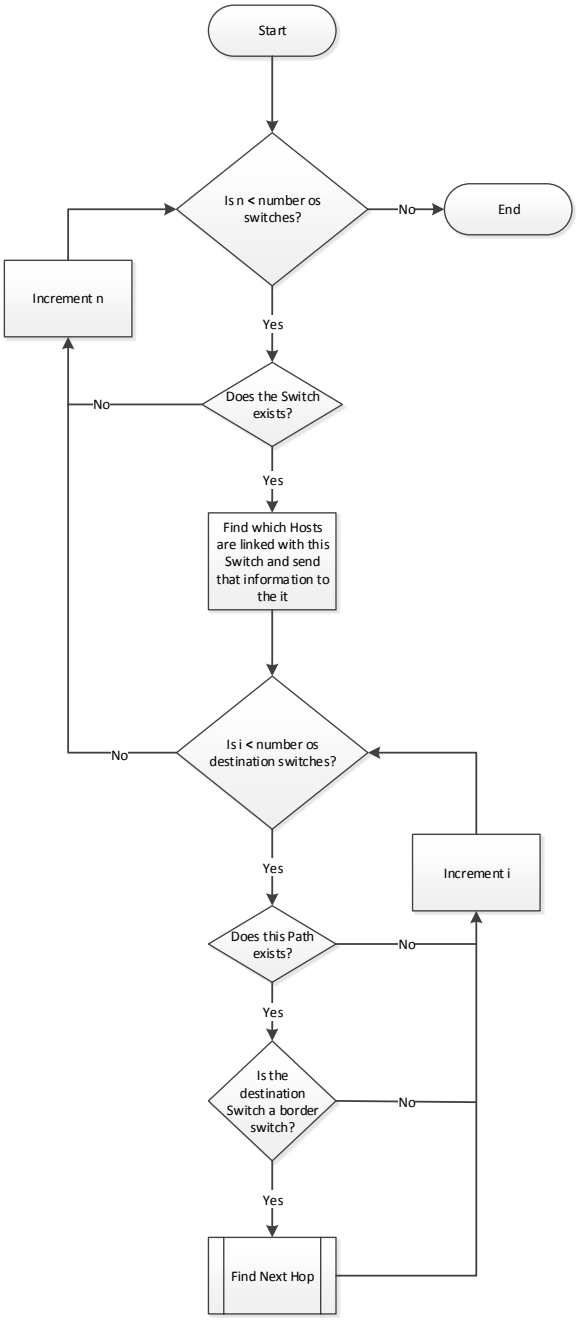


Figure 3.3: Flowchart illustrating how the controller fill all the switches tables at the beginning of the simulation

ports, the controller decides (distributing flows in a round robin policy) and downloads the information to the flow table. The reverse mapping between port and source destination IP addresses is also installed to assure that frames from the same flow follow the same path in the reverse direction. The switch will only perform another communication with the controller if one of two possible events happens: the path became unavailable and a new one has to be used for the same flow; or if the flow expires.

3.2.3 Ingress Switch

An ingress switch is a border switch that receives frames in host facing ports. The first time a host belonging to a given I-SID sends a frame the ingress border switch sends it to the controller that does the mapping between the switch input port and client MAC and downloads it into the switches flow table for future use when the border switch is acting as the egress switch. The exception is if the frame destination is directly linked to the switch in which case this process is not executed.

The switch then floods the frame to all of the border switches that have clients belonging to the same I-SID as the sending host through the corresponding B-VID and according to the multipath forwarding table for that B-VID. Some validations have to be done during this process, like: check if the path is available; if the port is "up"; and if the next switch belongs to this B-VID.

If the destination client MAC address/Backbone MAC address mapping had been previously learnt by the controller then instead of flooding the frame the border switch will send the frame only to the corresponding border switch MAC address. Once again the frame is forwarded according to the multipath forwarding table for the corresponding B-VID.

As was told before, the flows are separated between the possible multiple paths using the destination/source IP addresses. This separation uses an auxiliary table that stores the number of flows that are using a given forwarding path. Flows are then distributed to the forwarding paths that are less used in a round robin distribution. Until the flow expires the frames from a certain host to a defined destination are always forwarded by the same paths.

Ingress switch operation is illustrated in figure 3.4.

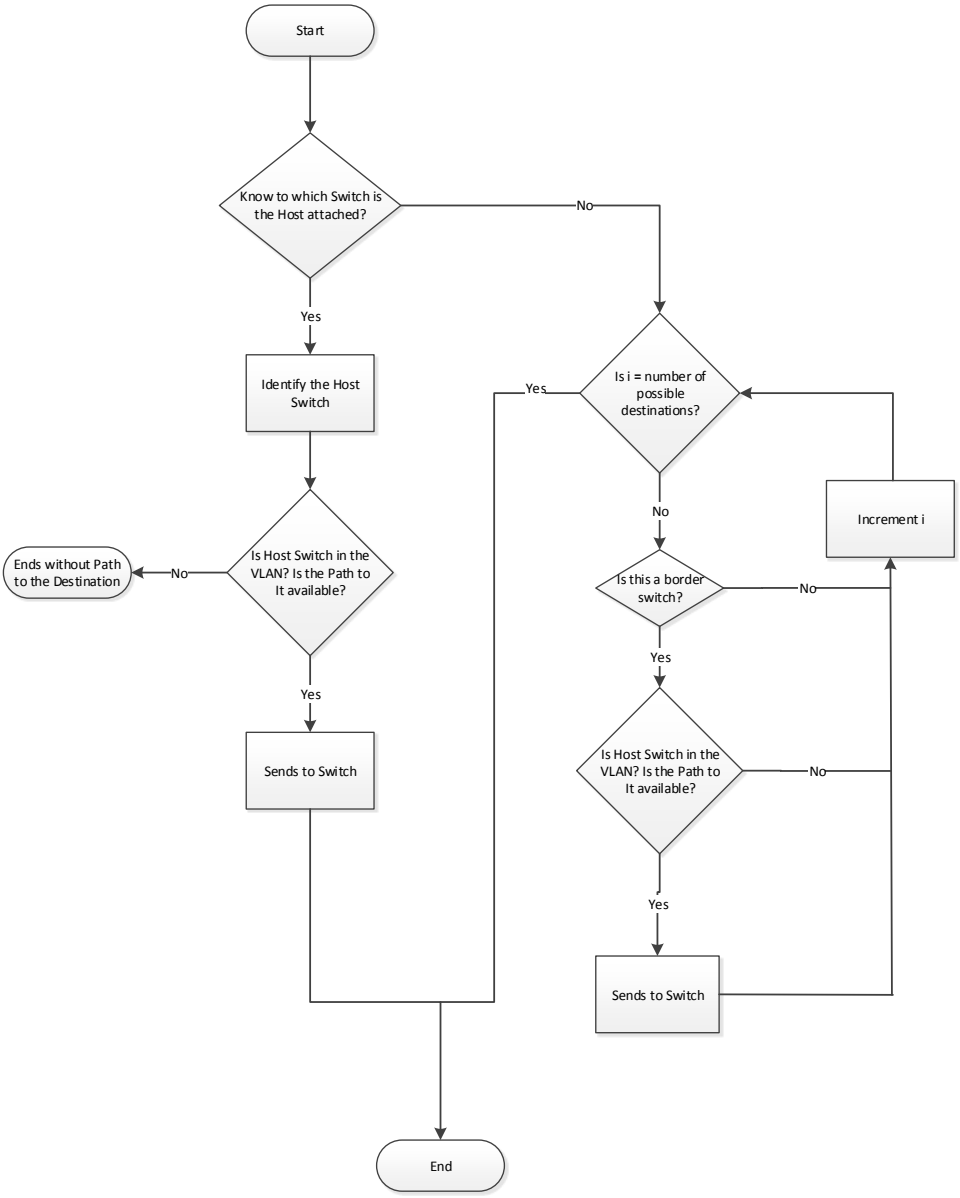


Figure 3.4: Flowchart illustrating how the frame is process when it reaches the Ingress Switch

3.2.4 Egress Switch

At the egress switch when the first frame for a given host is received it is sent to the controller where the encapsulation is removed and the client source MAC Address/Backbone source MAC address mapping is downloaded to the switch flow table. These addresses are saved to simplify frame forwarding when the switch is acting as the ingress switch. If this is not the first time that a source host sends a frame, there is no need to perform this mapping.

If the switch has previously learned the port/MAC address mapping for the destination host address then it forwards the frame to that port. If not, the switch floods the frame to all ports belonging to the corresponding I-SID and C-VLAN. Forwarding to the client side does not use multipath as it behaves like regular Ethernet being based solely on the destination MAC. It is possible that destination host is not connected with the egress switch. If this was the case the controller will inform the switch that the frame has to be dropped.

Figure 3.5 illustrates the operation of an Egress switch.

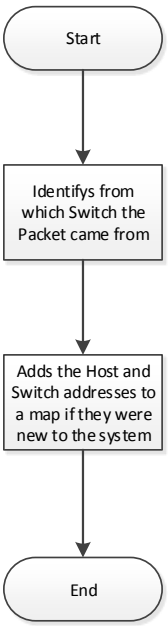


Figure 3.5: Flowchart illustrating how the frame is process when it reaches the Egress Switch

3.3 Failure Handling

When a switch detects a failure affecting a connection of a given port it sends that information to the controller. The controller checks if the switch detecting the failure stills reaches all border switches through paths with the same policy (to assure that changing paths does not cause forwarding loops).

If that is the case it simply moves the flows that were being forwarded in the failed path to one of the alternatives with the same preference, sets a timer and if the failure is resolved before the timer expires no further action is needed.

If alternatives of the same preference are no longer available or the failure is not solved before the timer expires, the controller updates the network graph and recalculates the paths updating the forwarding tables. Current active flows remain distributed to the same paths as before the failure, except the ones that lost reachability or have to be changed to less preferred paths. New flows are distributed using the regular distribution between the paths available in the current network map.

With this scheme many transient failures result in no action by the controller apart from re-directing the flow to an alternative path, which is a big improvement over spanning tree. Even when the paths need to be recalculated, the cost is similar to the re-convergence of a shortest path routing algorithm.

This procedure is illustrated on the flowchart of figure 3.6.

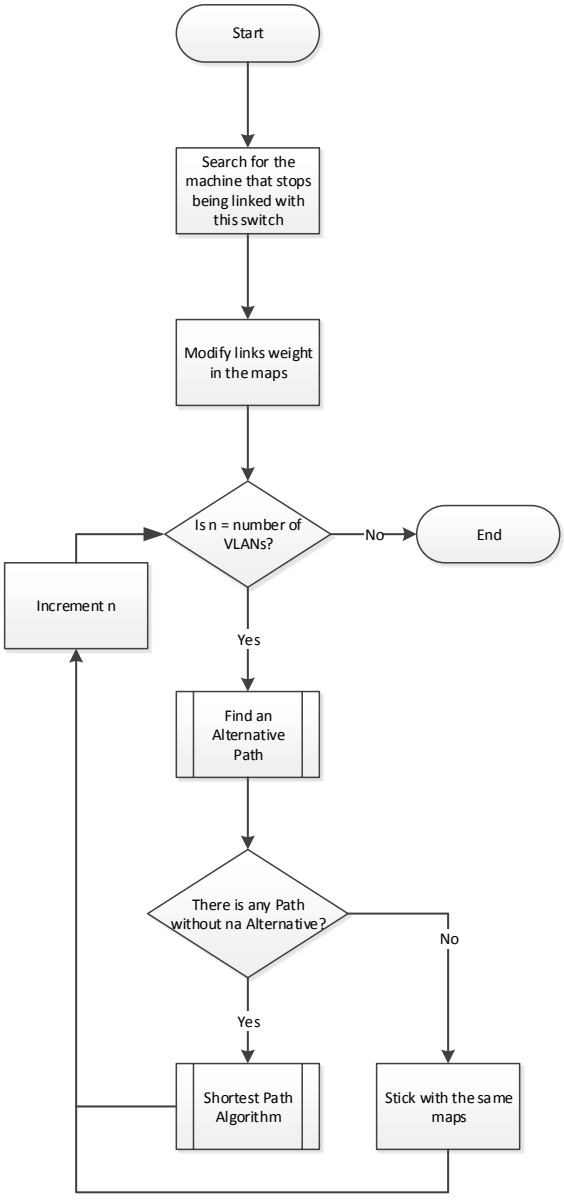


Figure 3.6: Flowchart illustrating how the frame is process when a Failure Occurs

Chapter 4

Simulation Results

4.1 Software Used

In this section we describe the tools used to simulate and test the designed architecture, and discuss their limitations.

We implemented the architecture using an OpenFlow switch simulation module built in the ns3 [bib] simulator.

4.1.1 ns-3 Network Simulator

Ns-3[bib] is a discrete event network simulator where all models and core programs are implemented using C++ language. Ns-3 is an "evolution" of ns-2 and it tries to simplify ns-2 using only C++ instead of C++ and OTcl.

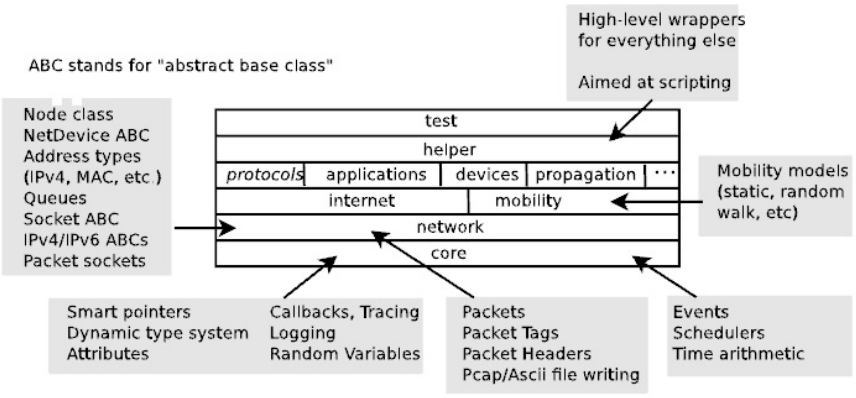


Figure 4.1: Software organization of ns-3 (adapted from ns3 Manual)

ns-3 source code is organized as described in figure 4.1. The Physical Layer in ns-3

modules is not well defined. The upper layers, Layer 2 and 3, are the ones where all developers focused their attention because it is where this tool can have a bigger impact. Other programs, like Matlab, are used when the improvements are focused on the Physical Layer.

This work used the OpenFlow module, framed in the ns-3s network layer, provided and developed by Blake Hurd. It is divided in three parts (like most of ns-3 modules): a Helper; an Interface; and a Net-Device. The Helper is used to easily configure all the properties of a certain module and initialize and retrieve an object with an Interface and a Net-Device associated to it. These later parts implement the actual functional code.

The OpenFlow Controller functions are implemented on the Interface side using many basic functions already available for basic controllers. The other part is implemented on the Net-Device. C++ programming uses a well-known concept, heritage. With that it is possible to use functionalities already implemented on other modules and classes. On the OpenFlow Net-Device, the functionalities implemented in a general Net-Device module written by ns-3 developers, are used. This module represents a ns-3 switch.

The last important part provided by ns-3 is the OpenFlow protocol itself. All features and messages available in version 0.8.9 are present on the OpenFlow module.

4.1.2 Software Limitations

Several limitations along this work development were faced. The OpenFlow version implemented in ns-3 was the major limitation.

Nowadays, OpenFlow is in version 1.3.1. The available module on ns-3 represents version 0.8.9. This is enough information to know that this work can be improved when an updated version is available. For example, the header encapsulation we performed is already implemented on version 1.3.1. of OpenFlow header. It is only necessary to fill the header fields with the appropriate values.

Another problem occurred in the implementation was the multipath use. The priority field, on the frames header, do the flow distribution between different forwarding tables using the priority values. Flows with less preference are never used even when the flows with higher preference are unavailable.

Another limitation is the lack of traffic generation capabilities. It is not possible to test the bandwidth capacity. The links bandwidths are only used for frames travel time simulation. When a frame is using a link, even with more available bandwidth, other frame can not use it until the first frame is delivered.

The last, but not the least, limitation is the lack of installation/configuration information of the OpenFlow module and ns-3. Setting up ns-3 is not trivial and can take several hours to have all modules working perfectly. OpenFlow modules lack information and many features are not implemented.

4.2 Executed Tests

The executed tests and the topologies used, are explained in the next subsections. We then present and discuss the results.

4.2.1 Topologies Tested

The developed architecture was tested in five different topologies: a Fat-Tree topology; a Fat-Tree topology with links between same level pods; a Clos network topology; a Hierarchical topology with same level links and a hierarchical topology used on SPB tests.

4.2.1.1 Fat-Tree Topology

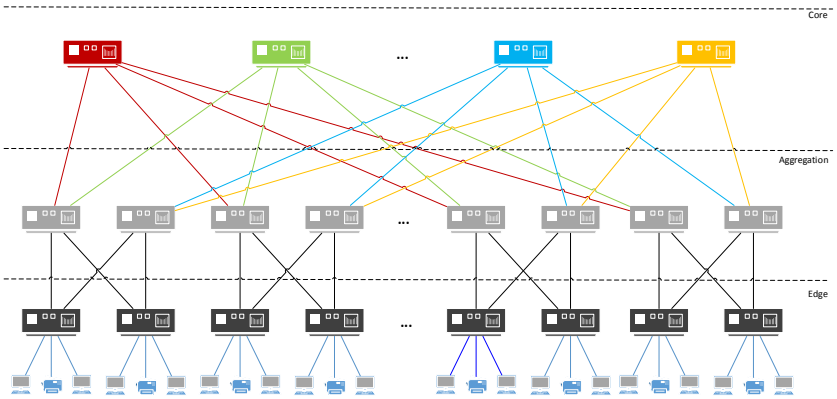


Figure 4.2: Fat-Tree Topology

The Fat-Tree topology is one of the most popular types of topology in use nowadays.

Its hierarchical way of distributing switches on different levels as it is possible to see in figure 4.2, is the reason of its success and popularity. A topology with this characteristic spreads its switches into three different kinds of levels, Edge, Aggregation and Core.
The Edge Level (black switches) is where hosts and switches are linked. To allow scalability, reliability and machine migration, normally these switches have a different implementation comparing to switches present on other levels.
The Aggregation Level (grey switches) do the traffic aggregation. In a normal Fat-Tree this level has the same number of switches as the Edge Level but with more redundant connections, to increase the alternatives to reach the Core Level. A group of linked Aggregation and Edge switches is called a Pod.
Finally the Core Level (sorted colours switches) is the one with less switches but with more redundancy. This layer communicates with elements outside the network and connects the different pods.
When datacenter developers use a Fat-Tree topology on their architecture, they try to spread traffic in the network using as much bandwidth as it is possible. To spread traffic the paths preference has to be the same, in other words, the number of hops has to be equal. Also, the number of paths with equal preference has to be equal to the number of ending hosts on the Edge Level, one path per host.
By introducing same level links the path diversity can increase when using our policy based control plane. Using the normal Fat-Tree topology, same level links between pods on the Aggregation level were introduced to test our architecture performance.

4.2.1.2 Clos Network Topology

In a Clos Network topology there are no pods but only different levels. Figure 4.3 illustrates this type of topology. All switches from a level are connected to the ones on the upper and lower levels. This kind of solution has an immediate advantage, there are as many equal alternative equal shortest paths available as there are different connections.
This is a useful characteristic for networks that need a reliable and robust topology.
However the cost of connecting all switches together is a disadvantage.
The available ports number is another inherent problem to the use of so much links, this

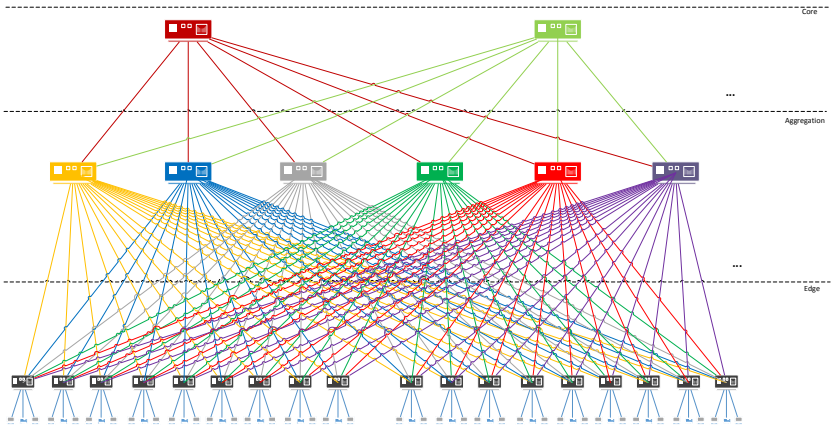


Figure 4.3: Clos Network Topology

a problem especially in the Aggregation Level.

4.2.1.3 Hierarchical Topology with Same Level Links

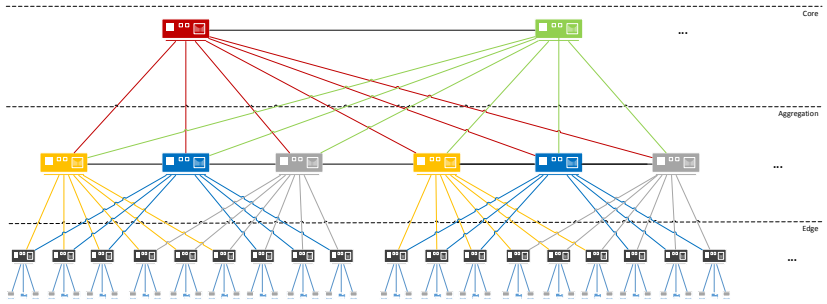


Figure 4.4: Hierarchical Topology with Same Level Links (adapted from [Ama12] and [Cha13])

This topology is a fat-tree like design with the addition of links between the switches of the same level in the aggregation and core levels and is depicted in figure 4.4.

Aggregation and Edge switches are distributed into pods, as in a Fat-Tree approach. However this pods distribution is different. A pod is constituted by three Aggregation Level switches and nine Edge Level switches. Each Aggregation switch is linked to six different Edge switches making each Edge switch linked with two different Aggregation switches.

In the Core Level there is also a division where larger pods are formed. In the original formula presented in [Ama12] and [Cha13] Core switches are all linked and divided into

two different pods to establish connection with the Aggregation Level. On this approach it was chosen to only have three links between Core level switches.

This kind of topologies spread traffic on lower levels.

4.2.1.4 Topology used on IEEE 802.1aq Tests

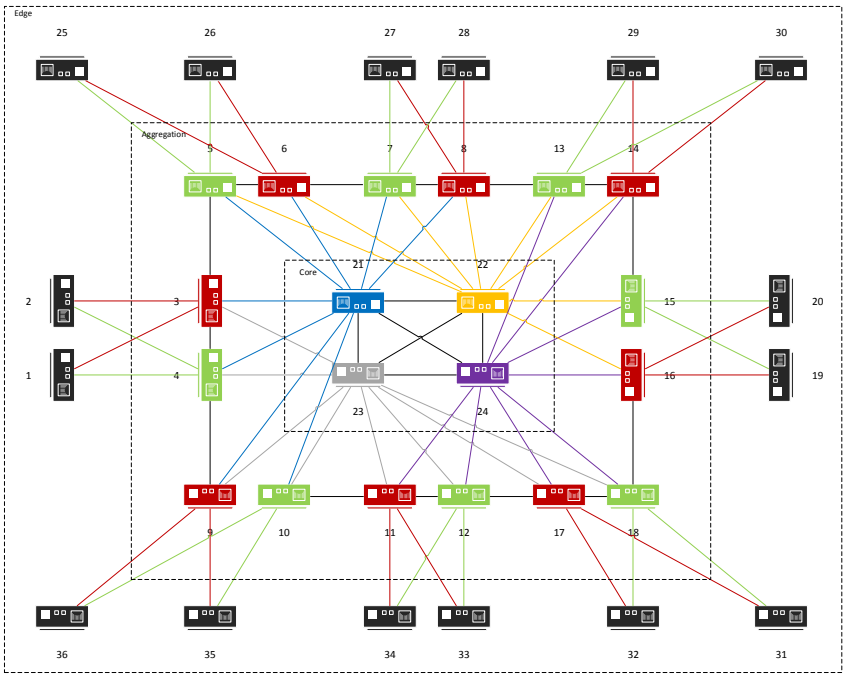


Figure 4.5: Topology used on 802.1aq (adapted from [AASB+10])

To compare our architecture to the SPB standard we performed experiments using the same topology used on IEEE 802.1aq paper [AASB+10]. The topology is illustrated in figure 4.5 and it is very similar to the last one but with differences on the connections between different level switches.

Links between Edge (black switches) and Aggregation (red and green switches) Levels use the same logic as the Fat-Tree. Each Edge switch has two links connecting it to two different Aggregation switches creating a pod.

In the Aggregation Level all switches are directly linked to their nearest same level switches except between switches 15-16 and 9-10. The Core is constituted by four switches, all connected, and linked to eight different Aggregation switches. Each switch on the Aggregation Level has to be connected with two different switches on the Edge Level.

The similarities with the topology used in [AASB+10], allows us to directly compare the

amount of paths obtained with our approach with the results of an ECMP solution like shortest path bridging.

4.2.2 Number of Paths

The first set of experiments measures the diversity of paths in the network (number of paths with at least a different link or node between a source-destination pair). The number of disjoint paths (paths that do not share a single link from its source to its destination) is also calculated.

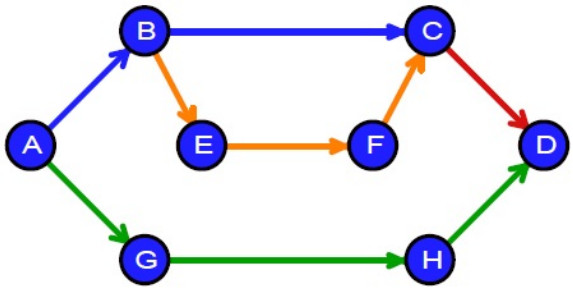


Figure 4.6: Disjoint paths (adapted from [Cha13])

The big importance of having a great number of disjoint paths is the networks reliability improvement. When a link goes down, if 2 disjoint paths are available is certain that an alternative way exists. If only one disjoint path it is available even when lots of different paths can be used, it is possible that the connection is lost. Taking an example from [Cha13], it is possible to see in the network represented on figure 4.6 that two disjoint paths and three different paths exist between node A and node D. If the link connecting C to D goes down, two different paths became unavailable but as another disjoint path (A-G-H-D) exists the communication between switches is still possible.

Path counting is done using the results of the path calculation algorithm. For each source destination pair, there will be a list of possible choices for the next hop. A simple algorithm saves all hops that form a path to the intended destination. Figure 4.7 illustrates how this works.

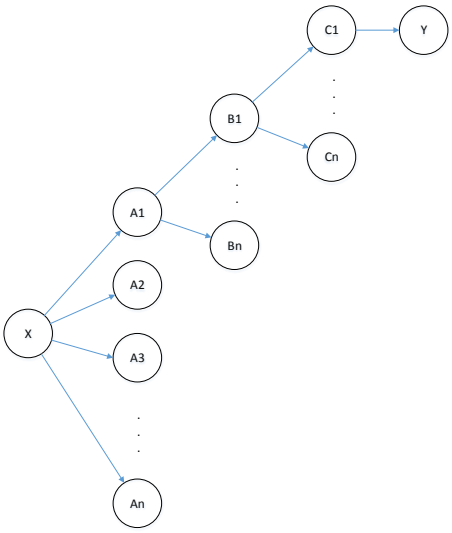


Figure 4.7: Path Calculation Process

4.2.3 Fault Tolerance

The test of fault tolerance consists in measuring how the number of available paths degrades with the accumulated occurrence of link failures.

The failures are produced in a random fashion in any link, except the ones connecting hosts to edge switches.

When a switch identifies a failure it informs the controller where it happened. When a message is received by the controller it decides between computing a new path or using an available valid solution. This process is demonstrated in figure 4.8. The number of different and disjoint paths is saved after each failure. With this it is possible to compare what is the evolution in the number of available paths.

The number of provoked failures corresponds to 50/55% of the total number of existing links.

4.3 Performance Results

The results discussed in the next subsections compare the performance of ECMP and non-ECMP policy based routing algorithms.

In 4.3.1 the number of available different and disjoint paths are compared in terms of the accumulated distribution of those paths per source-destination pair. The best curve for

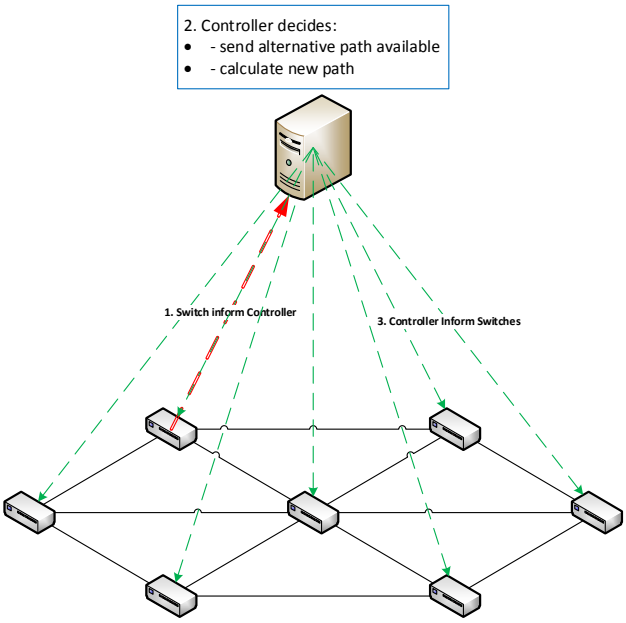


Figure 4.8: Fault Tolerance Process

this representation, is the one that stays near 0 most of the time and only goes to one when is reaching the end. If this happens it means that almost all source destination pairs have high path diversity.

The other experiment, 4.3.2, compares the evolution of different and disjoint paths while the number of faults is accumulating. In this case, the best solution is represented by a line with no slope. This indicates that the number of paths is not affected with the accumulation of faults.

On 4.3.1, all the results are only obtained using the paths between Edge switches and Core and Edge switches.

In 4.3.2 the results are for a single pair of nodes for direct comparison with a similar experiment in the 802.1aq paper.

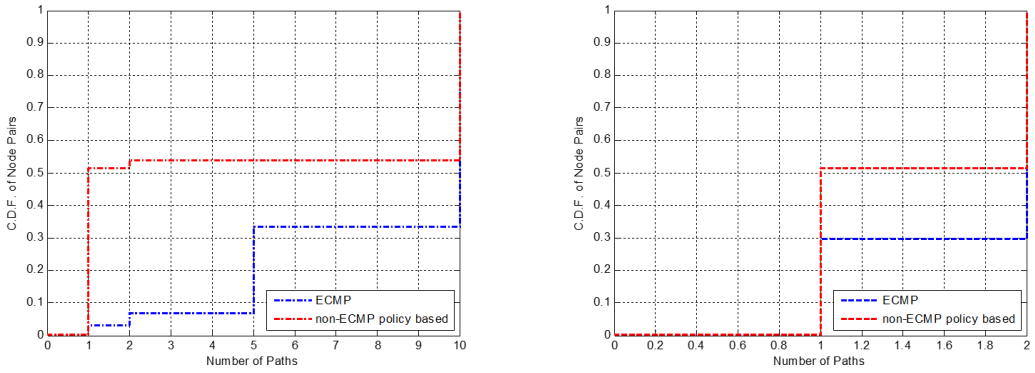
Our control plane uses policies to calculate paths while the ECMP control plane uses a simple metric with all links having equal cost.

In the policy-based control plane we applied the U_W/D_W policies to the links between switches in different hierarchy levels.

In the links between switches at the same level the most intuitive policy to apply would

be the S_L policy. However, this implies that the S_L paths between same-level switches become preferred even if multiple U_W paths are available. If multipath diversity is the goal then if we apply the U_W/D_W policy instead of S_L for the same-level switches then paths through the same-level switches have the same policy and preference than the ones going to higher-level switches and we are able to use all of them. In our simulations after measuring the resulting paths in both situations we opted to label same-level links with U_W/D_W to have the configuration that yields more usable paths. Note that this kind of flexibility is not possible for cost based ECMP solutions because we either just use the paths via same-level links (they have a smaller cost), or we increase the cost of those links and only use the lower cost paths through higher levels.

4.3.1 Number of Paths Results



(a) Comparison of Different Paths Evolution

(b) Comparison of Disjoint Paths Evolution

Figure 4.9: Comparing ECMP and non-ECMP policy based algorithms in a Fat-Tree Topology

In figure 4.9 the results are for the evolution of different and disjoint paths in a Fat-Tree topology.

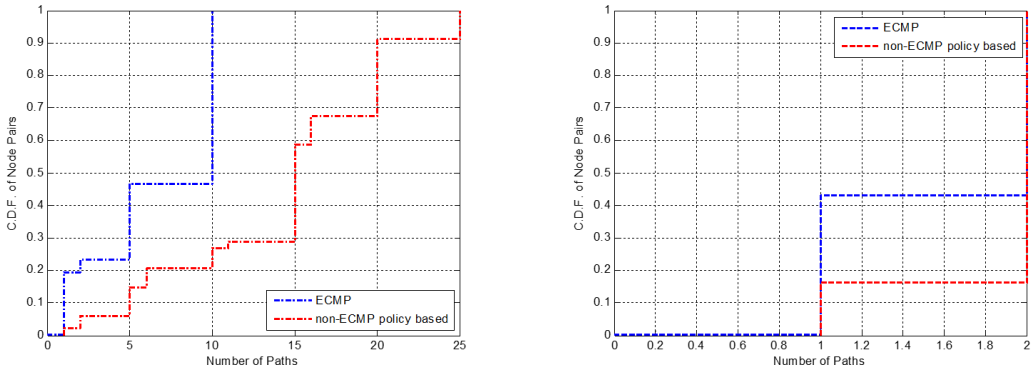
The ECMP has a better performance than non-ECMP policy based. On both 4.9a and 4.9b the ECMP representation is more alike with what is defined as best result (a line near 0 most of the time and only at the end near 1). However both approaches still have a very equal behaviour. This happens due to Fat-Tree topology architecture.

In 4.2 the results illustrate the limited number of possible connections in this type of

topology. If a switch wants to communicate with another one in a different pod, there are not many alternatives. This is good when it is mandatory to delimit a hierarchical topology but on the other hand decreases the number of alternative paths. Also, increases the probability of having isolated switches when link failures start appearing.

Observing the red line on 4.9a is possible to show that approximately 52% of node pairs have one path, 2% have 2 different paths and the remaining 46% have 10 different paths. Analysing the blue line more switches have more diversity of available paths on ECMP because approximately 30% have 5 different paths and more than 65% have 10 different paths. On 4.9b the difference is lower but ECMP continues having the best results. On non-ECMP policy based 48% pairs of nodes have 2 disjoint paths and on ECMP 70% have the same number of disjoint paths.

Due to ECMP best results, connections between pods on the Aggregation layer were introduced. The results of this topology are represented on figure 4.10.



(a) Comparison of Different Paths Evolution

(b) Comparison of Disjoint Paths Evolution

Figure 4.10: Comparing ECMP and non-ECMP policy based algorithms in a Fat-Tree Topology with links between pods

In 4.10 the results are the opposite comparing to what was observed in 4.9. The addition of new links between nearest pods on the Aggregation layer turns to be better for policy routing.

The ECMP has a worse representation than the non-ECMP policy based. In first, more than 45% of switches have less than 10 different paths available comparing with the 22% identified in non-ECMP policy based. With this limitation, it is possible that many paths

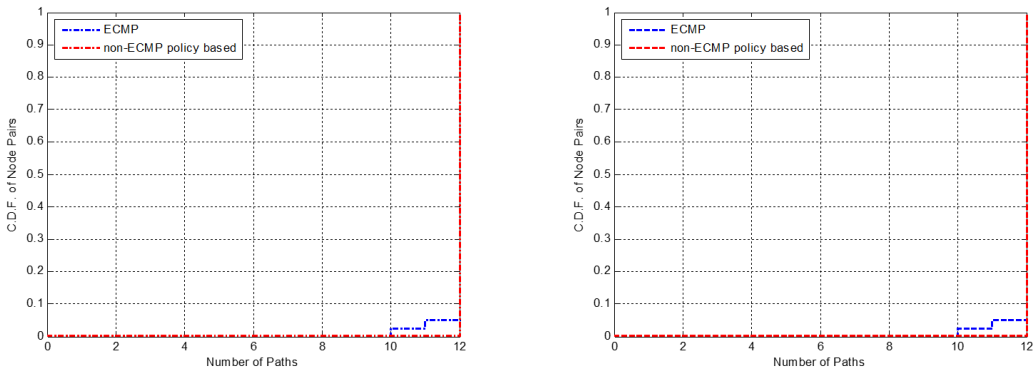
have to be recalculated in a shorter period of time if failures start to occur.

In ECMP the maximum number of different paths available is 10 and 54% of switches pairs have it. In non-ECMP the same number of different paths are available for almost 80% and the maximum number of different paths that a switch can have is 25. In other words, more than double comparing to ECMP.

Another characteristic that is not in favour of ECMP is the curve progression. Observing the non-ECMP progression curve, it is approximately what is identified as a good curve. In opposite, ECMP curve reaches the 100% in a very short period meaning that it has less alternative paths.

Finally, on 4.10b the results indicate that non-ECMP is a better approach for this architecture. 84% of the topology switches have 2 disjoint paths, comparing with the 55% existing in ECMP.

The short number of disjoint paths is shared by both algorithms but this is a characteristic of this topology. On the next topology, it will be demonstrated what is possible to achieve with much more available links.



(a) Comparison of Different Paths Evolution (b) Comparison of Disjoint Paths Evolution

Figure 4.11: Comparing ECMP and non-ECMP policy based algorithms in a Clos Network Topology

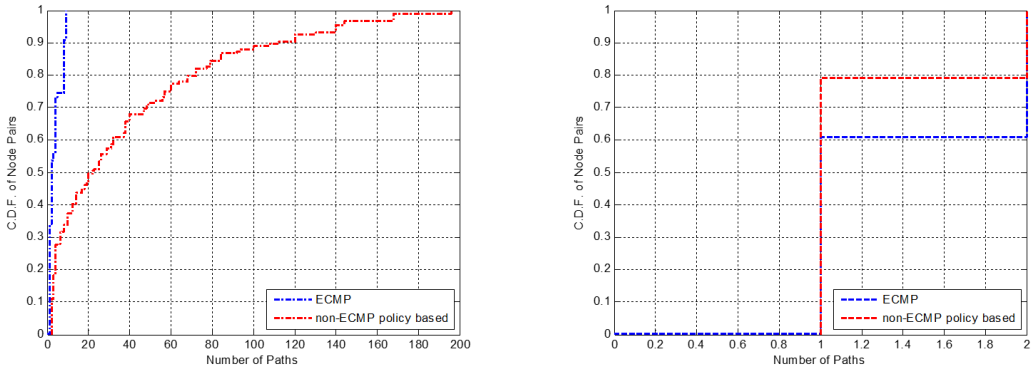
On a Clos Network Topology, due to its huge number of connections, it is expected that the number of different and disjoint paths is also big.

This previous analysis was confirmed by figure 4.11 and the first observation has to be related with the "perfect" curve represented in both graphics. It indicates that on all

switches in this topology the maximum number of different and disjoint paths possible is obtained. Only a little difference is observed on ECMP.

The absence of connections between switches on the same layer happens to limit the algorithms calculations. The problem is simplified and the decisions pass to switches present on Aggregation layer because they are all connected. The number of different and disjoint paths is almost the same and equal to 12. This is the number of existing Aggregation level switches.

This topology, with ECMP or non-ECMP policy based, represents the best architecture in terms of reliability. A problem inherent to it, is the big number of available connections representing also a huge portion of resources that needs to be allocated and can be wasted.



(a) Comparison of Different Paths Evolution

(b) Comparison of Disjoint Paths Evolution

Figure 4.12: Comparing ECMP and non-ECMP policy based algorithms in a Topology withdrawal from [Ama12] and [Cha13]

Returning to more organized topologies with links between same level switches, the topology presented in [Ama12] and [Cha13] was tested.

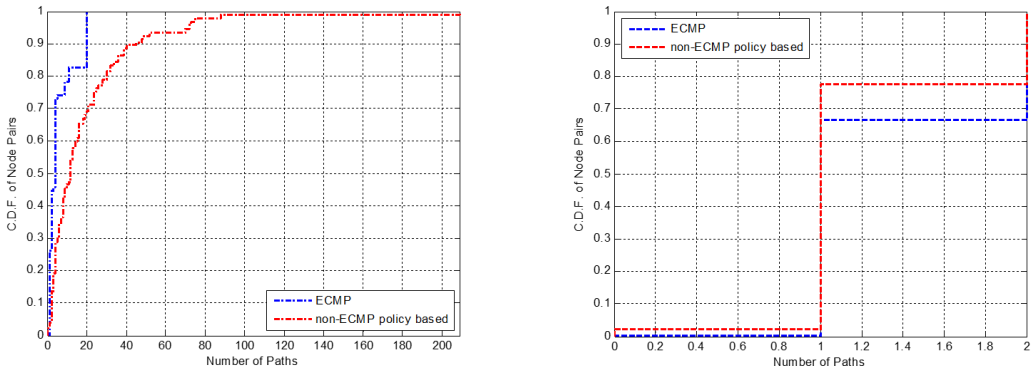
It was used to test the non-ECMP policy based algorithm so it is expected that the results are better than ECMP.

Computed the results it is very clear that in terms of different paths, the ECMP loses. In figure 4.12a is represented the maximum number of different paths available in non-ECMP and it is more than one hundred times bigger than ECMP, more precisely 209 for non-ECMP and 10 for ECMP. Also, although it is not possible to observe on the graphic,

less than 10% of node pairs have 10 different paths available on ECMP. Comparing with non-ECMP the difference is again huge because approximately 60% of pairs have at least that number of paths.

On the other side, in figure 4.12b it is represented the Cumulative Distribution Function (C.D.F.) representations for the disjoint paths and in this case a bigger number of switches have 2 disjoint paths available in ECMP (almost 40% comparing with approximately 20% in non-ECMP policy based).

This means that apart from the non-ECMP having more possibilities than ECMP, most of them share a very important link that when it goes down makes all the different paths unavailable.



(a) Comparison of Different Paths Evolution

(b) Comparison of Disjoint Paths Evolution

Figure 4.13: Comparing ECMP and non-ECMP policy based algorithms in a Topology retrieved from [AASB+10]

The last topology is the one presented in [AASB+10] and in this case there will be an extra topic to compare with the obtained results.

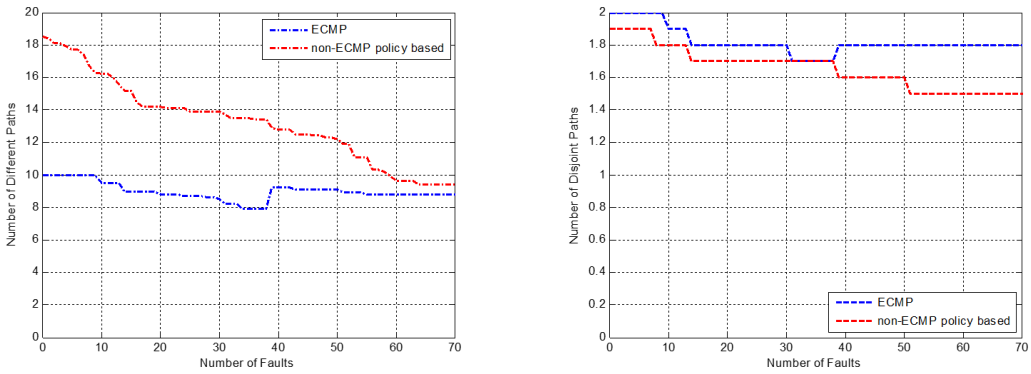
Both figures 4.13 and 4.12 have very similar representations. Therefore, the conclusions for each one are the same because the topologies (presented in sections 4.2.1.4 and 4.2.1.3 respectively) are similar. The differences are only in the number of connections between layers and between switches on the same layer.

At this point, with this last topology no new conclusions were achieved but one comparison can be done between SPB and the architecture described on this document. The results can be compared between the same path that is referred in [AASB+10] (from 34 to 28).

In [AASB+10] it is written that between these two switches there are 16 different paths available. After testing the architecture with non-ECMP policy based 209 different paths were calculated. This number can appear a little "out of bound" and exaggerated but the links policies were chosen to maximize the number of paths between this two switches. The other ones also have more paths than in SPB but not on the same scale of this.
These results indicate that with the proper choice of policies a much higher path diversity can be achieved when compared to traditional ECMP approaches when same level links are used on topologies.

4.3.2 Fault Tolerance Results

Before starting this experiments it was expected that topologies with less number of disjoint paths had a bigger difficulty to maintain their routes and when the controller has to calculate new routes, ECMP will be faster (and in certain situations more flexible) than the non-ECMP policy based.
To have more reliable results, for each topology we isolated the set of nodes pairs where both nodes are at the edge and have at least two completely disjoint paths between them (i.e. they do not share any link or node). From this set we selected the pair with the least number of different paths to perform the measurement. The reason is that pairs with totally disjoint paths are more likely to resist a large number of failures without losing reachability. Also, each experience was executed 10 times and the link failures were random.
In 4.14 the results are for the fault tolerance tests on a Fat-Tree topology. Looking to figure 4.14a results, it is possible to see that at the beginning the non-ECMP policy based has more different paths then ECMP (20 comparing with ECMP 10). However when the number of faults starts to increase its paths diversity decreases while ECMP stays nearly the same (both algorithms end with approximately 9 different paths available). The same difference continues on figure 4.14b results, where ECMP has always more disjoint paths than non-ECMP.
A simple explanation for this behaviour is the flexibility that ECMP can have. If the shortest paths available disappears, when the ECMP computes its new path the number



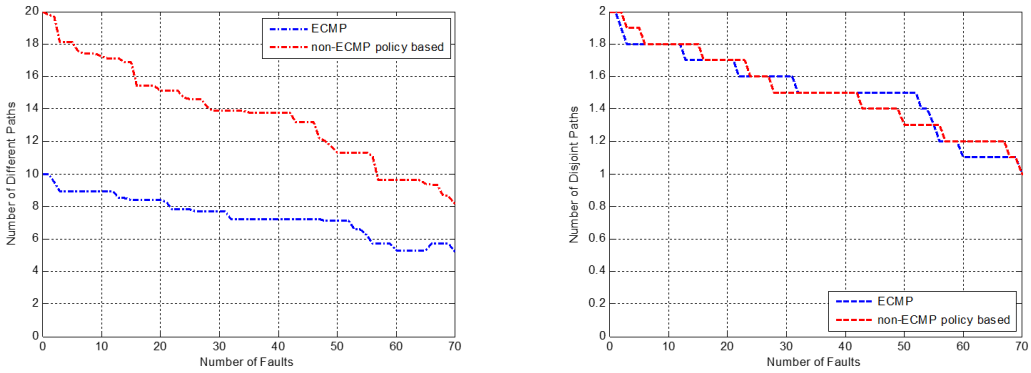
(a) Comparison of Different Paths Evolution (b) Comparison of Disjoint Paths Evolution

Figure 4.14: Comparing ECMP and non-ECMP policy based algorithms in a Fat-Tree Topology

of alternatives can increase. This happens because the shortest cost inherent to its new path can be bigger enough to have more paths with equal cost.

In non-ECMP the unavailability of its first set of shortest paths can cause great limitations to the next calculus because some policies block available paths to ensure no loops are available. If it was possible to change link policies "on-line" this problem would be solved and the flexibility of non-ECMP could be observed.

The results obtained on figure 4.15, when same level links were introduced in Fat-Tree topology, are similar.



(a) Comparison of Different Paths Evolution (b) Comparison of Disjoint Paths Evolution

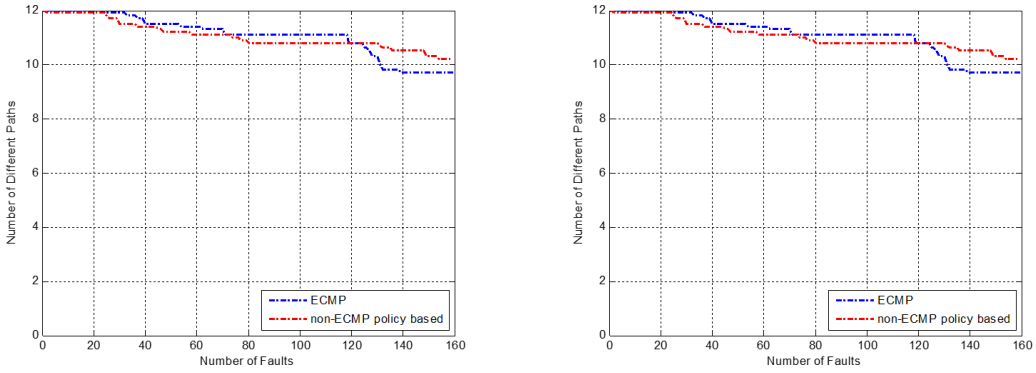
Figure 4.15: Comparing ECMP and non-ECMP policy based algorithms in a Fat-Tree Topology with links between pods

The results in figure 4.15a showed that the initial difference is sharp (20 for non-ECMP and 10 for ECMP) but at the end they are separated only by 2 paths.

Comparing disjoint paths (on figure 4.15b results) both algorithms have a similar evolution but with a much more sharp decrease comparing to regular Fat-Tree topology. On this case both algorithms end with only 1 disjoint path available and the ECMP has less performance than before.

However, a good quality observed on this topology, and also in normal Fat-Tree, is the non-isolation of switches. This result is explained using the luck factor because if the first faults introduced are on links that connect the Ingress switch to the Aggregation layer, automatically all different paths disappear and the switch becomes isolated.

Another observation is that the connection between both switches being tested has only 1 disjoint path after all the faults introduced but this only one represents minus 1 path than at the beginning.



(a) Comparison of Different Paths Evolution

(b) Comparison of Disjoint Paths Evolution

Figure 4.16: Comparing ECMP and non-ECMP policy based algorithms in a Clos Network Topology

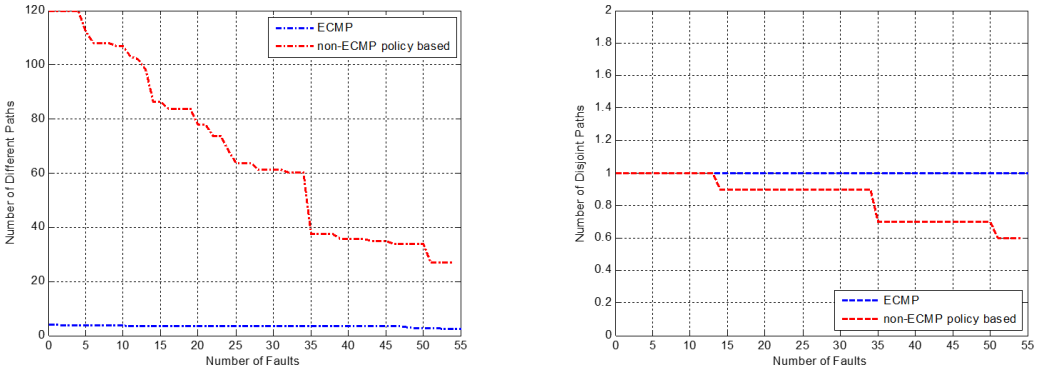
The Clos Network topology is the one with better results, as was expected. Its great variety of connections makes it very resistant to faults.

In the experiences to calculate the number of paths, the results for both algorithms are the same. In this experience they are not the same, because the faults introduced are random, but they are very similar.

The topology structure helps to avoid the faults introduced along the simulation and is

the cause for both graphs being equal (all the different paths are also disjoint paths and because of that figure 4.16a and figure 4.16b results are equal).

For this specific test (and why not for all of the others when money is not the problem) this is the best approach for every network that reliability as by its main goal.



(a) Comparison of Different Paths Evolution

(b) Comparison of Disjoint Paths Evolution

Figure 4.17: Comparing ECMP and non-ECMP policy based algorithms in a Topology withdrawal from [Ama12] and [Cha13]

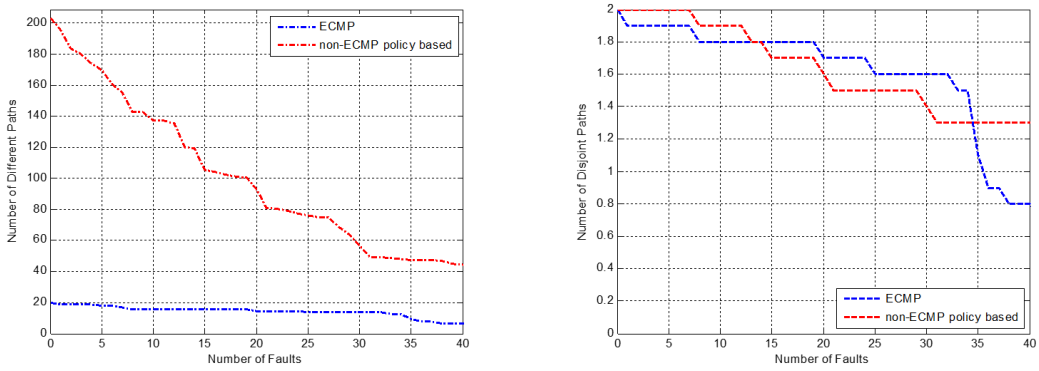
Continuing to more common topologies, on 4.18 it is illustrated the evolution of different paths while the number of faults is increasing in a topology with same level links on Aggregation and Core Layers.

As was tested before, the non-ECMP has more different paths than ECMP but in terms of disjoint paths they are similar. With that in mind and observing figure 4.18a it is easy to conclude that one algorithm is more reliable than the other.

Is normal that non-ECMP initial number of different paths, 120, is always decreasing because with policies it is possible to send information using most part of available connections. This is an advantage but it can also be a disadvantage because it is always modifying information routes, unlike ECMP, that despite having a lesser number of initial different paths, 4, it can maintain its number.

Looking at the results of figure 4.18b results the last observation becomes clearer. The number of disjoint paths is only 1 but for ECMP it does not modify unlike for non-ECMP where it decreases. Because the representation is the average of what can happen it is easy to deduce that after the tenth fault in one experience the most important link in

this path becomes unavailable. With that one switch turns into an isolated hardware. So, despite not being very well represented in the results of figure 4.18, in some tests the path becomes unavailable and one of the switches becomes isolated.



(a) Comparison of Different Paths Evolution

(b) Comparison of Disjoint Paths Evolution

Figure 4.18: Comparing ECMP and non-ECMP policy based algorithms in a Topology retrieved from [AASB+10]

The last tested topology, like in the number of paths test, is the comparison with SPB because all other observations can be done with the last topology.

The different paths results, on figure 4.18a, does not have much interest for this comparison because non-ECMP will always have more paths available than ECMP: in this case, 209 and 20 respectively. However the results for the evolution of disjoint paths, on figure 4.18b, can be more interesting.

Both approaches have a similar evolution trying to compensate the faults increase but in the experiments executed only one of them let one switch become isolated. As was explained before, this ending has a random factor and with that it can be observed in both cases with the same probability but it only happens with ECMP.

With the fault tolerance test it was not conclusive which approach is the best but it was demonstrated how many alternatives each one of them still have after 50/55% of the total number of existing links become unavailable. In this field the non-ECMP policy based algorithm has a bigger number of solutions.

Another factor available for comments in the fault tolerance test is the time spent by both approaches to compute the paths when a failure occurs. These times were not re-

ferred along with the tests comments because in every experiment ECMP was always faster than the non-ECMP. This fact has a simple and easy explanation and it is inherent to non-ECMP algebra calculations that became much more complex than ECMP matrix operations with links weights.

Chapter 5

Conclusions

We used a formal method based on routing algebras to design a control plane for an L2 Ethernet fabric suited for datacenter and metro networks. Instead of producing multipath forwarding between endpoints using a traditional ECMP control plane, like most of the literature, we use a control plane that calculates policy-based multiple paths between the endpoints of the fabric. Our method also handles network failures in an efficient way.

This approach opens the possibility to flexibly define the amount of paths that can exist between two switches via the application of policy. It also opens the possibility of non-ECMP with a great increase in the usage of the network links.

We presented a design that maintains standard Provider Backbone Bridging Ethernet frame formats using I-SID to identify services and B-VIDs to transport them. The policy-based multipath control plane is used to calculate the multiple paths between the endpoints of each B-VID. Flows are distributed amongst paths using IP header information assuring that traffic flows traverse the same path in each direction.

We presented a possible implementation design with a central OpenFlow controller and OpenFlow controllable switches. The characteristics of our control plane assures correct forwarding behaviour with independent switch decisions, this opens space for a number of possible implementations like distributed controllers for each switch or re-using the Provider Backbone learning control plane in switches that support it. Results indicate that path diversity can be more effectively used in traditional network topologies used in cloud supporting datacenters and metro networks. Nevertheless, the control plane is very

comprehensive in terms of topologies and can be used in all types.

The results obtained on chapter 4 prove that out approach, in terms of different paths, has a huge difference to ECMP. This fact helps on improving bandwidth use and decreasing traffic congestion. The reliability is very similar on both cases due to the low number of disjoint paths.

5.1 Future Work

All tests performed with this architecture were executed in a simulation environment. It will be a great challenge, as future work, to test it on a real environment where specific situations and network behaviours can be observed.

Also, as future work, it would be interesting to study how the flexibility introduced by the use of policies can be leveraged to design network topologies that can deliver a high bisection bandwidth with cheaper switches and fewer links than current traditional topologies.

Bibliography

[AASB+10] David Allan, Peter Ashwood-Smith, Nigel Bragg, János Farkas, Don Fedyk, Michel Ouellete, Mick Seaman, and Paul Unbehagen. Shortest path bridging: efficient control of larger ethernet networks. *Communications Magazine, IEEE*, 48(10):128–135, 2010.

[AB12] David Allan and Nigel Bragg. *802.1aq Shortest Path Bridging Design and Evolution: The Architect's Perspective*, chapter 1. John Wiley & Sons, 2012.

[ABP13] P. Amaral, L. Bernardo, and P. Pinto. Multipath policy routing using destination based hop-by-hop forwarding. *21th IEEE Internacional Conference on Network Protocols (ICNP)*, 2013.

[Aca13a] Ethernet Academy. Carrier ethernet services over transport technologies - bridging. http://www.carrierethernetstudyguide.org/MEF%20SG/pages/2transport/studyguide_2-1-1-1.html, 2013.

[Aca13b] Ethernet Academy. Carrier ethernet services over transport technologies - provider bridging. http://www.carrierethernetstudyguide.org/MEF%20SG/pages/2transport/studyguide_2-1-1-2.html, 2013.

[Ama12] Pedro Miguel Figueiredo Amaral. Multipath inter-domain policy routing. 2012.

[Ava11] Avaya. Compare and contrast spb and trill. Technical report, Avaya, 2011.

[bib] Ns-3, discrete-event network simulator. http://www.nsnam.org/.

[Cha13] João Pedro Gonçalves Chalaça. Multipath policy routing in packet switched networks. 2013.

[Cis09] Cisco. Ieee 802.1ah on provider backbone bridges, 2009.

[Cis10] Cisco. Ieee 802.1ad support on provider bridges, 2010.

[DCD12] Carolyn J Sher Decusatis, Aparico Carranza, and Casimer M Decusatis. Communication within clouds: open standards and proprietary protocols for data center networking. *Communications Magazine, IEEE*, 50(9):26–33, 2012.

[Eas10] D. Eastlake. Rbridges and the trill protocol, 2010.

[Eri] D. Erickson. What is beacon? https://openflow.stanford.edu/display/Beacon/Home.

[FA09] János Farkas and Zoltán Arató. Performance analysis of shortest path bridging control protocols. In *Global Telecommunications Conference, 2009. GLOBECOM 2009. IEEE*, pages 1–6. IEEE, 2009.

[Fou12] OpenFlow Networking Foundation. Openflow switch specification version 1.3.0, 2012.

[GG08] Timothy G Griffin and Alexander JT Gurney. Increasing bisemigroups and algebraic routing. In *Relations and Kleene algebra in computer science*, pages 123–137. Springer, 2008.

[GKP+08] Natasha Gude, Teemu Koponen, Justin Pettit, Ben Pfaff, Martín Casado, Nick McKeown, and Scott Shenker. Nox: towards an operating system for networks. *ACM SIGCOMM Computer Communication Review*, 38(3):105–110, 2008.

[IEE04] IEEE. Ieee standard for local metropolitan area networks - media access control (mac) brdges, 2004.

[IGA04] Guillermo Ibanez, Alberto Garcia, and Arturo Azcorra. Alternative multiple spanning tree protocol (amstp) for optical ethernet backbones. In *Local*

Computer Networks, 2004. 29th Annual IEEE International Conference on, pages 744–751. IEEE, 2004.

[JOS+11] Michael Jarschel, Simon Oechsner, Daniel Schlosser, Rastin Pries, Sebastian Goll, and Phuoc Tran-Gia. Modeling and performance evaluation of an openflow architecture. In *Proceedings of the 23rd International Teletraffic Congress*, pages 1–7. ITCP, 2011.

[KCR11] Changhoon Kim, Matthew Caesar, and Jennifer Rexford. Seattle: A scalable ethernet architecture for large enterprises. *ACM Transactions on Computer Systems (TOCS)*, 29(1):1, 2011.

[MAB+08] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, 2008.

[Net13] Juniper Networks. Multiple spanning tree protocol. http://www.juniper.net/techpubs/en_US/junos/topics/concept/mx-series-multiple-stp.html, 2013.

[Ng] Eugene Ng. Maestro: A system for scalable openflow control.

[NMPF+09] Radhika Niranjan Mysore, Andreas Pamboris, Nathan Farrington, Nelson Huang, Pardis Miri, Sivasankar Radhakrishnan, Vikram Subramanya, and Amin Vahdat. Portland: a scalable fault-tolerant layer 2 data center network fabric. In *ACM SIGCOMM Computer Communication Review*, volume 39, pages 39–50. ACM, 2009.

[Ora04] D. Oran. Ieee standard for local metropolitan area networks - media access control (mac) bridges, 2004.

[Per00] Radia Perlman. *Interconnections - Bridges, Routers, Switches, and Internetworking Protocols*, chapter 3.3, 3.4. Pearson Education India, 2000.

[Sea05] Mick Seaman. Shortest path bridging. *Available online: ieee802.org/1/files/public/docs2005/new-seaman-shortestpath-par-0405-02.htm*, 2005.

[SMC09] Malcolm Scott, Andrew Moore, and Jon Crowcroft. Addressing the scalability of ethernet with moose. In *Proc. DC CAVES Workshop*, 2009.

[Sys10] Brocade Communication Systems. Leveraging the benefits of provider backbone bridges, 2010.

[Woj03] Wald Wojdak. Rapid spanning tree protocol: A new solution from an old technology. *Reprinted from CompactPCI Systems*, 2003.