TÉCNICO LISBOA

Cost Analysis of Data Centers with High Availability

Sofia Alexandra Carlos Salgueiro

Thesis to obtain the Master of Science Degree in

Electrical and Computer Engineering

Supervisors: Prof. João Luís Costa Campos Gonçalves Sobrinho
Dr. Nuno Claudino Pereira Lopes

Examination Committee

Chairperson: Prof. Teresa Maria Sá Ferreira Vazão Vasques
Supervisor: Prof. João Luís Costa Campos Gonçalves Sobrinho
Member of the Committee: Prof. Luís Eduardo Teixeira Rodrigues

January 2021

Declaration

I declare that this document is an original work of my own authorship and that it fulfills all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa.

Acknowledgments

This thesis was made at Instituto de Telecomunicações and would not have been possible without the help of some very important people.

I want to start by thanking my supervisors, Prof. João Sobrinho, and Dr. Nuno Lopes, for their guidance, patience, and for pushing me to always improve my work. This thesis would not have been possible without their advice and willingness to help.

I also want to thank my family, my parents, my sister, my two grandmothers and three grandfathers. Thank you for always being there for me, for believing in me, for always supporting my academic choices and cheering me on.

I was lucky enough to meet some incredible people when I was still in school, especially Maria, Inês, and Daniela. Thank you for always supporting and encouraging me.

I want to thank the friends I made at Instituto Superior Técnico. The five years I spent at Instituto Superior Técnico were some of the most challenging in my life but they made them easier. Thank you for always being there to lend a hand and for uplifting me during the hardest times of this journey.

Last but not least, I would like to express my gratitude for my boyfriend Alex, thank you for supporting me and being by my side during each obstacle I overcame.

Abstract

With the growth of cloud computing, data centers are increasingly being used to host applications and data of external companies. As such, data center companies must guarantee that the services hosted in their data centers are available most of the time with minimum latency. Nonetheless, failures in the services offered by data centers lead to the loss of revenue of clients and consequent reimbursements from data center companies. We studied how a data center network with hundreds of thousands of servers can be built to minimize the impact of link failures and, at the same time, its cost.

We assumed data centers use folded Clos topologies and implement the Border Gateway Protocol. We identified the characteristics of the different switches used in data center networks and estimated their cost with the main goal of comparing the acquisition cost of switches of different topologies. We analyzed, through mathematical expressions and algorithmic computations, the impact of link failures in the connectivity between pairs of servers and between the servers and the Internet. We studied the cost also considering traffic and the congestion in links in the presence of link failures. To do this, we assumed traffic patterns of three applications: distributed applications with communications only internal to the data center, search engines and streaming services. Finally, we were able to provide answers on how to build the minimum cost network that provides the availability of services offered by data centers.

Keywords

Data center, availability, cost, network topology

Resumo

Com o crescimento da computação na *cloud*, os centros de dados são cada vez mais escolhidos para alojar as aplicações e dados de empresas externas. Dado isto, os centros de dados têm de garantir que os serviços alojados estão disponíveis a maior parte do tempo e com a mínima latência. No entanto, falhas nos serviços oferecidos por centros de dados levam à perda de rendimentos dos clientes e, consequentemente, a reembolsos por parte das empresas de centros de dados. Neste trabalho, estudámos como construir uma rede de centro de dados com centenas de milhares de servidores que minimize o impacto de falhas nas ligações e, simultaneamente, o custo.

Assumimos que os centros de dados usam topologias de Clos dobradas e implementam o *Border Gateway Protocol*. Identificámos as características dos diferentes comutadores utilizados em redes de centros de dados e estimámos o seu custo, com o objetivo de comparar custos de aquisição de comutadores de diferentes topologias. Analisámos, através de expressões matemáticas e computações algorítmicas, o impacto de falhas em ligações na conectividade entre pares de servidores e entre os servidores e a Internet. Estudámos o custo considerando também o tráfego e a congestão de ligações quando ocorrem falhas em ligações. Para isto, assumimos padrões de tráfego de três aplicações: aplicações distribuídas com comunicação interna ao centro de dados, motores de busca e serviços de *streaming*. Finalmente, foi possível retirar conclusões sobre como construir a rede de menor custo que cumpra os valores de disponibilidade de serviços de um centro de dados.

Palavras Chave

Centro de dados, disponibilidade, custo, topologia de rede

Contents

List of Figures

List of Tables

List of Algorithms

Listings

Acronyms

AS	Autonomous System
ASN	Autonomous System Number
ACL	Access Control List
BFS	Breadth-First Search
BGP	Border Gateway Protocol
CORE	Common Open Research Emulator
DRAM	Dynamic Random-Access Memory
eBGP	external BGP
ECMP	Equal-Cost Multi-Path
FIB	Forwarding Information Base
FRR	FRRouting
HPC	High-Performance Computing
iBGP	internal BGP
IP	Internet Protocol
MGEN	Multi-Generator
OSPF	Open Shortest Path First
RIB	Routing Information Base
RIP	Routing Information Protocol
SLA	Service-Level Agreement
SSH	Secure Shell
TCAM	Ternary Content-Addressable Memory
TCP	Transmission Control Protocol
ToR	Top of Rack
TTL	Time To Live
UDP	User Datagram Protocol

Chapter 1

Introduction

1.1 Motivation

Data centers comprise hundreds of thousands of servers interconnected by thousands of routers in a single network [4]. Although data centers may be used to house resources of the owning company, the data centers we focus on are used to lease resources, in the form of, for example, data storage, virtual machines and full servers, to external clients. This model is known as cloud computing [5] and it is used by private costumers and companies. Therefore, each data center may house a large number of different applications: machine learning models that can take days to train [6], websites of enterprises, online stores, search engines, streaming services, etc...

There are multiple advantages for companies to choose data centers to host their applications instead of having their own servers. Firstly, building a data center is an investment of millions of dollars [7] and most companies do not have this much money upfront so it is simpler to monthly, or annually, pay to have their resources on external data centers. Secondly, the responsibility of hardware management, which includes not only the server components but also the network, cooling equipment, and also real estate management, is transferred to data center companies. And finally, when the use of resources varies greatly, it is easier to scale up and down the resources when using data centers. For example, if an external company sees a big surge of traffic to its website due to a promotion, it can more easily scale the number of servers handling the requests when it uses a data center. There is also the model of "pay as you go" in which a client only pays when resources are effectively being used, such as when a machine learning model is being trained.

However, when there are failures in data centers, which may result in delays or unavailability, that is, resources are unreachable, clients that use data centers to host their resources lose revenue. For example, a study [8] found that, for Amazon, an increase in latency of 100 ms resulted in a loss revenue of 1%. Google also studied how the increasing loading times of mobile pages is associated with bouncing, which occurs when people abandon the site before it is fully loaded, and discovered that, for example, if the loading time of a page grows from 1 s to 5 s, the probability of bouncing increases 90% [9].

In order to protect themselves, clients that use data centers to host their services sign agreements, called Service-Level Agreements (SLAs), with the owning companies. SLAs define the reimbursements

made by data center companies as a function of loss of availability and extended delay. When availability falls below or delays rise above the thresholds defined in SLAs, data center companies are required to reimburse clients in, generally, a percentage of what the client paid for the service [10–12]. Since each service has a different definition of availability and delay, each client signs SLAs in function of the services it uses. For example, Microsoft Azure guarantees availability for virtual machines, defined by the percentage of time that these virtual machines can be reached from outside the data center [10]. However, it guarantees not only availability but also a maximum upper bound on processing latency for the database service Azure Cosmos DB. In this case, availability is defined by the percentage of successful Read Requests and the upper bound on processing latency is defined for each type of operation in the database [11].

Since delays and unavailability can cause a significant loss in revenue, SLAs have demanding characteristics. For example, Azure guarantees that, when there are two instances of a virtual machine in different data centers, the client can access at least one instance 99.99% of the time in a month. It should be noted that this value of minimum availability translates to a downtime of, at most, 4 minutes. If, however, there are failures that result in a bigger downtime, then they will reimburse 10% of what the client paid by the service [10]. And, if machines are reachable less than 95% of the time, that is, are unavailable for more than 36 hours in a month, the service is paid back in full. Therefore, when there are failures in data centers, not only do clients lose money but data center companies lose as well, in addition to being detrimental to their reputation and make them lose actual and prospective clients. For example, when failures cause an outage, the number of companies affected is so large that it causes a disruption on the Internet and raises the attention of the news.

With the growth of cloud computing, new data centers are being constructed [7], especially in metropolitan areas to minimize latency between the data center and clients, and companies have even started building multiple data centers in the same metropolitan region, which are all interconnected and only tens of kilometers apart [13]. Thus, the competitiveness between companies to deliver the most reliable services to clients at attractive prices is increasing. However, data centers have hundreds of thousands of routers and there are, consequently, acquisition expenses (Capex) that amount to hundreds of millions of dollars [7] distributed by servers, routers, cabling, infrastructure, power and cooling equipment, real estate, etc... In addition, there are also operational expenses (Opex), such as maintenance personnel and power usage. Therefore, how can we minimize the cost of data centers while, at the same time, minimize the impact of failures and consequent violation of SLAs?

In this work, we focus on minimizing the cost of data centers for availabilities usually specified in SLAs. To do this, we study the topology of the network, that is, how servers are connected to each other and to the endpoints external to the data center, and the routing protocol, which determines the paths used to connect endpoints and how the traffic is distributed across these paths.

In order to minimize the impact of link failures, we want to have many redundant paths between a pair of endpoints to choose from. Additionally, we also want low latency, that is, the amount of time it takes for the first packet sent from a server to arrive at the destination, as it has an impact on performance [14]. Therefore, we want a high number of shortest and disjoint paths. In this work, we analyze folded Clos topologies, which are implemented in many data centers [15, 16]. These topologies offer many shortest and disjoint paths while easing the layout of cable.

After defining the topology of the network, we still need to implement a routing protocol to elect the shortest paths between a pair of endpoints and, ideally, to distribute the traffic equally through all of the shortest paths available in order to increase bandwidth usage. Border Gateway Protocol (BGP) is a well-defined protocol, has been in use for decades and, consequently, is implemented in routers available in the market and there are people experienced in it, which allows saving money. Additionally, it is already implemented in data centers [16] and, consequently, we chose it as the routing protocol used in our work.

We study two different aspects that have an impact on the availability of a network: link failures and congestion. In the case of link failures, we assume independent failures. Since there is a multitude of applications running in a data center and, consequently, different types of availability, we analyze the connectivity between pairs of servers and between the servers and the Internet. The connection between servers can be of higher importance in, for example, distributed applications, while the connection to the Internet is more important when accessing virtual machines, web pages, and so on. By doing this, we are able to have an approximation of the availability in the presence of link failures.

We determine the percentage of pairs of servers that are able to connect with each other and the percentage of servers that can connect with the Internet in order to obtain an approximated value of average availability. It should be noted, however, that reimbursement in accordance to the availabilities defined in SLAs is not linear. For example, having an average availability of 99% because one machine has 100% of availability and another has 98% is not the same as having two machines with 99% of availability. If we defined a threshold at 99%, in the first case the data center would have to reimburse the client of the machine with 98% of availability but would not have to reimburse any clients in the second case, even though both networks have the same average availability.

In terms of congestion in the links of the network, we assume traffic patterns of three well-known applications: High-Performance Computing (HPC), characterized by distributed applications with traffic only between the servers of the data center; search engines, where we assumed 80% of the traffic is between servers internal to the data center and the remaining 20% between the servers and the Internet; and finally, a streaming service pattern, in which all the servers send 80% of the traffic to the Internet and the remaining 20% to other servers of the data center. We assume an equal distribution of traffic flows by every path.

1.2 Goals

Our main goal in this work is to give answers to a high-level problem, which is to determine the data center with the lowest acquisition cost (Capex) and/or operational cost (Opex), given the following inputs:

1. Number of target servers in a data center.
2. Cost of network equipment, which is composed of routers and cables.
3. Link failure model, represented by a probability of a link failure.
4. Availability target of the service provided by data centers, as specified in SLAs, which we represent by the probability of existence of a non-congestioned path between servers and between servers and external endpoints.

In this work, we assume that the network topology is a 3-tier folded Clos network and that BGP is the routing protocol used. Additionally, we provide the following options:

- Exclusive use of shortest paths or additional use of non-shortest paths.
- Exclusive use of simple links or use of parallel links.

In order to fulfill our main goal, we have the following partial goals:

1. Determine the cost of a network in terms of routers. By determining what characteristics, such as memory size and number of ports, each router should have in the network, we are able to estimate an acquisition cost of routers in a network.
2. Compute the cost of the best network in terms of number of servers and the capacity of each link by using a program that generates all topologies.
3. Determine mathematical expressions that provide the probability of connectivity between endpoints, that is, between internal servers or between the data center and the Internet, in terms of the probability of a link failure. We also aim to identify the critical links in the network, that is, the type of links that have a larger impact in the availability of the network.
4. Identify how to increase connectivity by determining the topology and routing protocol configuration that maximizes it, using both mathematical expressions and algorithmic computations.
5. Analyze the congestion of the network in the presence of link failures for assumed traffic patterns of well-known applications in data centers.

1.3 Outline

This document is structured as follows:

- In chapter 2, we present the state of the art in data center networks. We begin by describing the physical structure most commonly used, namely folded Clos topologies, and its characteristics. Then, we provide a brief explanation of BGP and its configuration in data center networks.
- In chapter 3, we analyze the cost of data center networks without taking into account link failures. We break down the acquisition and operational cost of routers in data center networks by identifying the cost and power used of the different routers in the network. We calculate the cost of routers of different topologies and show how the cost of a network varies with the number of servers and with different link and port capacities. We analyze how parallel links between routers can help diminish the overall cost of the network.
- In chapter 4, we present mathematical expressions for the probability of connectivity between endpoints in terms of the probability of link failures. Then, we study the cost of data center networks in terms of connectivity between servers, and between servers and the Internet, in the presence of link failures. We determine how the number of paths decreases with the probability of link failures and present expressions that allow an estimation of the number of entries in the routing tables of the different routers in the network.
- In chapter 5, we analyze the congestion in the network in the presence of link failures by applying traffic to different topologies. We assume traffic patterns of three well-known applications in data centers: HPC, with communication only internal to the data center, search engines, with queries and replies, and streaming services. We assume that traffic flows are equally distributed by every shortest path available between a pair of endpoints.
- In chapter 6, we detail an emulation of a small folded Clos topology and the configuration of BGP to verify some of our results.
- Finally, in chapter 7, we present the main results and proposals for future work.

Chapter 2

An overview of data center networks

We start with a brief overview of the most common network topology and routing protocol used in data centers.

2.1 Clos network topology

Most data centers receive requests external to the data center. However, the way these requests are processed differs greatly with the service provided. For example, search engines, such as Google Search [17] and Bing [18], are distributed applications that work in the following way: a server receives a request from a user in the Internet and then, due to the complexity of the task, distributes it by multiple servers. Afterwards, all of these servers send the relevant data to the first server so that it can, finally, reply to the user. We can conclude from this example that in distributed applications most of the traffic occurs inside the data center and, as a consequence, server-to-server connections should be highly fault tolerant so that, even if a path becomes unavailable because of a link failure, servers are able to connect through another path.

However, in streaming services, for example, the traffic flow is very different. With platforms such as Netflix, a single server can handle the request without having to share the load with other servers. In this case, most of the traffic goes from the data center to the Internet and, consequently, each server should have many redundant paths to reach the Internet.

In the case of cloud computing, data centers provide services to external companies and private clients and have to reimburse them, in accordance to the SLAs signed, if there are failures that cause unavailability, such as when services are unreachable, or an increase in latency. These SLAs have demanding characteristics. For example, Azure guarantees that, when there are two instances of a virtual machine, the client can access at least one of them 99.99% of the time in a month [10], which translates to an unavailability of, at most, 4 minutes. In this case, the reimbursement is of 10% of what the client paid for the service but, for lower values of availability, such as below 95%, the service is paid back in full.

Therefore, data center companies want to minimize the risk of unavailability. One way to do this is, as previously mentioned, to have many redundant paths. So far, Microsoft [19], Google [20], and Facebook [1] data centers are using extensions of Clos topologies. As we will see further on, Clos

topologies can be designed to provide a high number of shortest disjoint paths between pairs of servers and between the data center and external endpoints. Additionally, its modular and repetitive structure simplifies the physical implementation of the network.

Introduction to Clos networks

A three-stage Clos network can be seen in figure 2.1, with r input routers, m middle routers, and r output routers. Each input router has n inputs and m outputs and each output router has m inputs and n outputs. The middle router has one input per input router and one output per output router, for a total of r inputs and r outputs.

There are two important properties that, depending on the number of input, middle, and output routers, Clos networks may have, which are:

- A strictly non-blocking network means it is always possible to connect a free input port to a free output port, no matter the actual state of connections. A Clos network is strictly non-blocking if and only if $m \geq 2 \times n - 1$.
- A rearrangeable non-blocking network means it is always possible to connect a free input port to a free output port but there may be the need to "re-arrange", i.e. change the path, of current connections. This is the case for Clos networks if and only if $m \geq n$.

It should be noted, however, that these properties assume that the load-balancing in the network is perfect, that is, each traffic flow follows a different path, which may not actually occur.

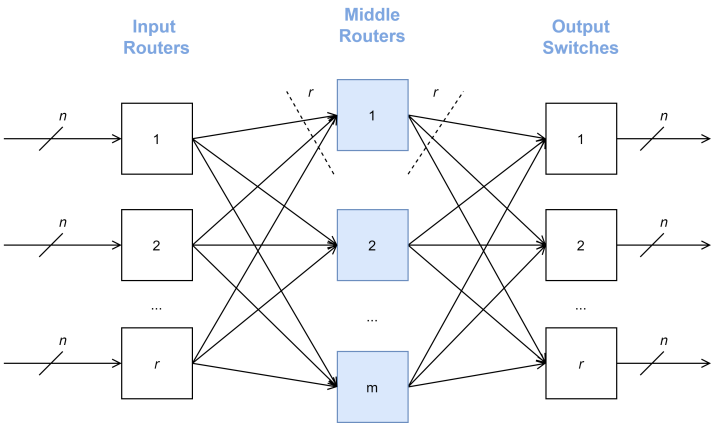


Figure 2.1: A three-stage Clos network.

Clos networks in data centers

In data centers, we want routers that send information not just in one direction but in both, which happens in fat-trees. A fat-tree, depicted in figure 2.2, is a tree where each node represents a set of routers

and, at every node, the number of downlinks is the same as the number of uplinks, and all links are bidirectional. Consequently, the number of ports used in a node doubles at every level from bottom to top. In order to obtain, from Clos topologies, a network equivalent to a fat-tree, it is possible to fold them, resulting in folded Clos topologies. In folded Clos topologies the input routers are also output routers and, consequently, with bidirectional links, the inputs are the same as the outputs.

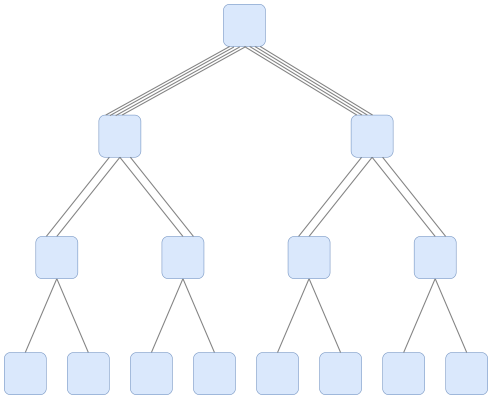
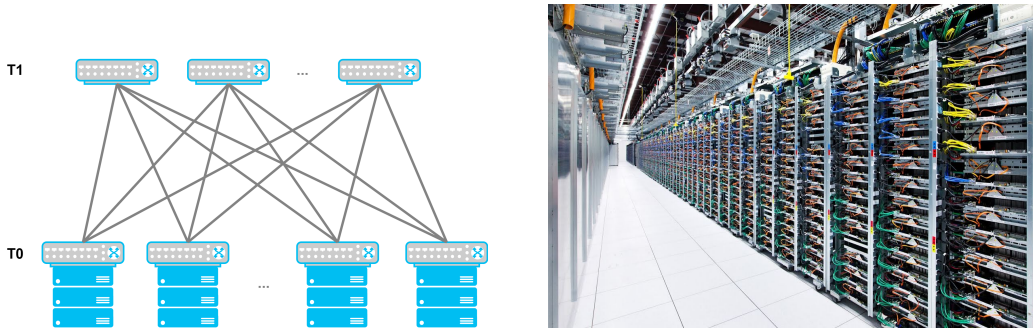


Figure 2.2: An example of a fat-tree.

In the simplest form of a folded Clos network, a 2-tier folded Clos network, shown in figure 2.3(a), the servers mounted inside a rack connect to a router called T_0. The T_0s of the network make the bottom-tier of this architecture and every one of them connects to every router in the adjacent top-tier, designated T_1s. T_0s are also called Top of Racks (ToRs) due to being placed at the top of rack cabinets, as depicted in figure 2.3(b). However, to minimize the average cable length connecting servers to the ToR, today they are more commonly placed in the middle of the cabinet.



(a) A 2-tier folded Clos network.

(b) Example of server racks in a Google data center. Taken from [21].

Figure 2.3: A 2-tier folded Clos network and server racks inside a Google data center.

This network provides redundancy, i.e. there are multiple paths between every pair of servers. For example, with T_{ij} representing a router in tier i and position j, if a server attached to T_{01} wants to

connect to a server in T_{04}, it has multiple paths to choose from: T_{01} - T_{11} - T_{04}, and T_{01} - T_{12} - T_{04}, etc... Therefore, if, for example, there is a single link failure, the server will still have an alternative path to reconnect.

In order to reduce costs, while keeping in mind that not all servers are communicating at the same time, some companies began using oversubscription in the ToRs. If every port of a router provides the same capacity, oversubscription occurs when the number of ports connecting to the lower tier (downlinks), which are in this case servers, is bigger than the number of ports connecting to the upper tier (uplinks), composed of T_1s, thus providing different values of bandwidth to the upper and lower tier. The oversubscription factor represents the relationship, at each router, between the bandwidth available to the lower tier and to the upper tier and it is common to all routers of the same tier:

$$oversubscription = \frac{\text{bandwidth available to the lower tier}}{\text{bandwidth available to the upper tier}}. \tag{2.1}$$

For example, a ToR that has an oversubscription of 5:1 with every link providing 40 Gbps translates to a maximum of 8 Gbps per server (40 Gbps / 5), if all servers are trying to communicate with servers of another rack. Equivalently, an oversubscription of 5:1 in ToRs means that each ToR connects to 5 servers for each connection to a T_1.

After introducing the definition of oversubscription, it is also important to present a way to evaluate the bandwidth in the whole network. A metric commonly used is the *bisection bandwidth*: when we divide the network in two halves, each one with the same number of servers, the bisection bandwidth is the minimum bandwidth available between these two halves [7, 22].

Returning to the analysis of the 2-tier folded Clos network, we can conclude that it is limited in terms of scalability. If the number of servers keeps increasing, there will come a time when routers will have no ports left to connect. For example, even if both T_0 and T_1 routers have 128 ports, without oversubscription, each T_0 can connect to $\frac{128}{2} = 64$ servers and to 64 T_1s. Then, since each T_1 has 128 ports, each T_1 can connect to 128 T_0s and the maximum number of servers supported is $\frac{128}{2} \times 128 = 8192$. Even if there is oversubscription, i.e., a T_0 connects to a greater number of servers than T_1s, the maximum number of servers supported will still not be big enough for data centers that need hundreds of thousands of servers.

In order to deal with the aforementioned problem, another tier was added to the network, leading to the 3-tier folded Clos network, as in figure 2.4. The routers from the bottom tier are, once again, called T_0s and the ones from the second and third tier are designated T_1s and T_2s, respectively.

In this network, the T_0s and T_1s are grouped in **clusters**, and each cluster is, consequently, a 2-tier folded Clos network in itself, and the T_2s provide the connection between clusters, allowing a bigger number of servers in the data center. The T_2s are grouped in *spine planes* and the first T_1 of each cluster connects to all the T_2s of the first spine plane, the second T_1 of each cluster connects to all the

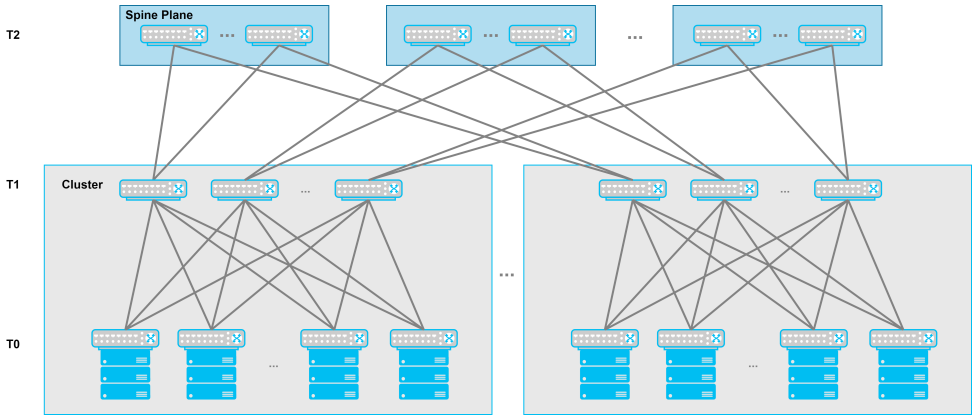


Figure 2.4: A 3-tier folded Clos network.

T_2s of the second spine plane, and so on. Thus, servers from a cluster can connect to servers of another cluster with paths of the form T_0 - T_1 - T_2 - T_1 - T_0.

Alternatively, we can separate T_2s by their position on the spine plane. In this case, the first T_2s of each spine plane are grouped in what is called a *spine set* and, consequently, there are as many spine sets as T_2s in a spine plane.

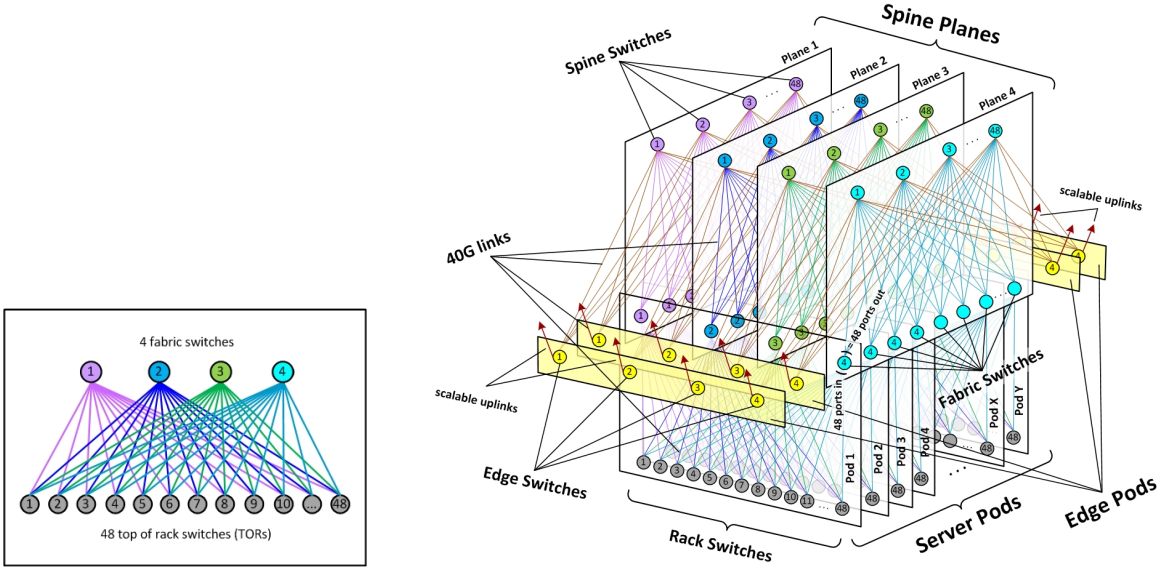
The additional tier of the folded Clos network continues to provide redundant paths between servers. A T_0 has many T_1s to choose from inside a cluster and each T_1 is also connected to many T_2s. However, as we will see in section 4.2, the number of disjoint paths, i.e. paths that do not share any link, is limited by the lower tier.

A 3-tier folded Clos network scales better than the 2-tier network. Treating clusters as a base unit, if we want more computing capacity, i.e. servers, we just need to add clusters and, if we need to increase the inter-cluster capacity, we add more routers in each spine plane. However, the maximum number of clusters and, consequently, the number of servers, continues to be limited by the number of ports of T_2s since we can only add as many clusters as the number of ports of a T_2.

Facebook and folded Clos networks

In 2014, Facebook presented the *data center fabric*, depicted in figure 2.5(b), that is very similar to a 3-tier folded Clos network, and uses *server pods*, figure 2.5(a), as clusters. In the figure 2.5(b), Facebook refers to T_0s or ToRs as *rack switches*, T_1s as *fabric switches* and T_2s as *spine switches*.

In this architecture, Facebook uses 48 T_0s and 4 T_1s per cluster. There is no oversubscription in the T_1s, so the bandwidth available to the upper tier is the same as the bandwidth available to the lower tier. Thus, since links connecting T_1s to T_0s have the same capacity, of 40 Gbps, as links connecting T_1s to T_2s, T_1s connect to the same number of T_0s and T_2s. Since each T_1 connects to 48 T_0s, then each T_1 connects to 48 T_2s and there are, consequently, 48 T_2s per spine plane. Finally, there is an additional



(a) A *server pod*.

(b) The full architecture.

Figure 2.5: A Facebook *data center fabric*. Taken from [1].

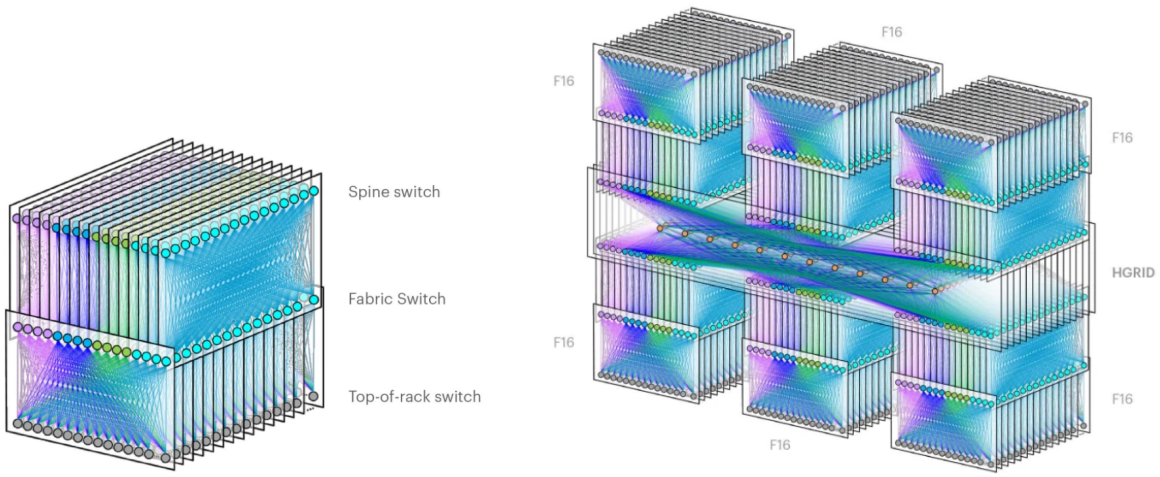
tier, composed of 4 *edge switches* per spine plane, that connects the data center to the Internet.

However, as time went by, the number of users and services of Facebook kept increasing and this architecture was no longer sufficient. Therefore, in 2019, Facebook presented an update of this network based on a new modular unit, the *F16* [2], depicted in figure 2.6(a).

As we can see, the *F16* is a 3-tier folded Clos network and resembles the *data center fabric* without the edge switches. The remaining main difference in both architectures is the bandwidth available to servers and that, instead of only 4 T_1s per cluster, there are now 16. They wanted to increase the bandwidth available to servers so that each ToR had 1.6 Tbps downlink capacity to servers and 1.6 Tbps to T_1s. Therefore, instead of having 4 T_1s per cluster and each T_0-T_1 link with a capacity of 400 Gbps, which corresponds to a total of $4 \times 400\ Gbps = 1.6\ Tbps$, they chose to use 16 T_1s instead, with links of 100 Gbps, which also results in $16 \times 100 = 1.6\ Tbps$.

Finally, in order to build a bigger data center and connect the *F16*s to the Internet, Facebook interconnects *F16*s with the *HGRID*, as shown in figure 2.6(b). The *HGRID* consists of planes of routers that connect the *F16*s to each other and to the Internet.

In this work, however, we will only analyze 3-tier Clos networks and, thus, assume that the edge routers that connect the data center to the Internet are "super nodes" connected to every T_2, as depicted in figure 2.7.



(a) An *F16*.

(b) The full architecture.

Figure 2.6: A Facebook data center based on a new modular unit, the *F16*. Taken from [2].

2.2 Border Gateway Protocol (BGP)

Routing protocols and why we chose BGP

After defining the physical structure of a data center network, we still need to make sure servers are able to communicate with each other and the Internet. Therefore, routers need to know where to send a packet so that it reaches its destination after receiving it from a server or another router.

Routing tables can be filled manually by telling routers the existing routes using static routing. However, in the presence of link failures, if a router can no longer ensure a path to a certain destination, other routers that rely on it will not be aware of this and will keep sending it packets, which will never reach the destination. Additionally, in networks that support hundreds of thousands of servers and virtual machines, and, consequently, use thousands of routers, static routing is more difficult to manage. If there is a change in the network, which can be as minimal as a new virtual machine, we do not want to manually alter the configuration of every router, we want routers to automatically learn about the change and alter their routing tables. To do this, we need to use a dynamic routing protocol.

Dynamic routing protocols are responsible for electing the best routes, given metrics specific to each protocol, and advertise these routes to the other routers in the network. Therefore, all routers automatically learn how to reach each network prefix and, each time the preferred route of a router changes, the other routers are informed of this change. This way, every alteration in the network is propagated to all routers. The dynamic routing protocol is also responsible for determining the number of redundant paths and, if possible, to distribute the traffic by the best paths available.

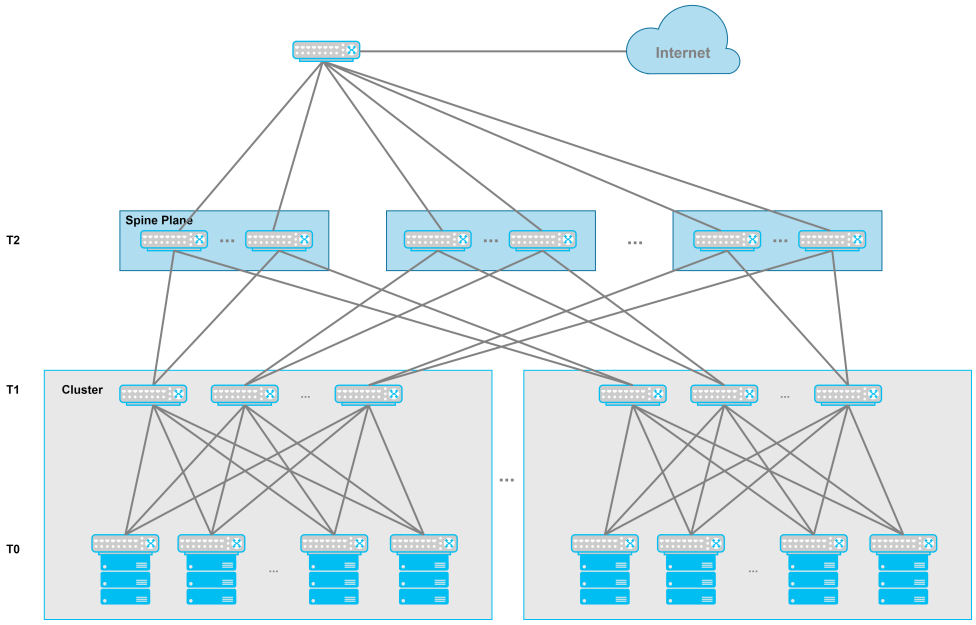


Figure 2.7: A folded 3-tier Clos network with a "super node" connecting the data center to the Internet.

Some of the most common dynamic routing protocols are Routing Information Protocol (RIP), Open Shortest Path First (OSPF) and BGP. In RIP, each router keeps information, for each destination, of the neighbor router closest to the destination and the distance of the path. Distance in RIP is measured in terms of hops, i.e., the number of routers the path goes through, and, every time the distance of a path changes, it needs to be propagated throughout the network. In OSPF, each router keeps information about every link, with its associated cost, in the network so it can build a graph and run an algorithm to find the shortest path to every destination. When the cost of a link changes, this information is broadcasted to the entire network so every router can update its table. In both OSPF and RIP, with richly connected networks such as folded Clos topologies, changes can lead to thousands of messages exchanged throughout the network which is something that we want to avoid. Furthermore, RIP does not support the use of multiple paths to reach a destination.

BGP, Border Gateway Protocol, is the routing protocol used in the Internet [23]. BGP has had many improvements and extensions since its first version and, currently, the fourth version, BGP4, is the one used in the Internet. BGP4, which we will refer to from now on as simply BGP, has many stable and robust implementations and, additionally, supports different extensions [16], such as *multi-path* [24, 25] which allows routers to register multiple paths to reach a destination. Additionally, it is implemented in most routers available in the market and there are people specialized in it. These advantages contributed to the first implementation of BGP in the data center by Microsoft [16]. Today, it is also chosen by many other companies that have data centers, such as Facebook [1, 16].

Google, however, created their own protocol, Firepath [20], before BGP was being used in data

centers. Firepath resembles a *link-state* routing protocol: each router collects the state of its interfaces and, every time it detects a change, sends this information to a master node that generates a link state database of the full network. The master node then propagates this information to every router. Finally, each router computes its routing table using the network topology, which was previously configured in every router.

Nonetheless, BGP is the more commonly used protocol in data centers [16] so we chose it as our routing protocol in this work. However, before we detail its implementation in data centers, we give a brief explanation of its functioning.

Brief overview of BGP

As previously mentioned, BGP is the routing protocol used in the Internet. The Internet is divided into smaller networks, Autonomous Systems (ASs), and each AS is assigned a unique number, the Autonomous System Number (ASN). BGP enables the connection between ASs and, consequently, makes the Internet a globally available network.

BGP connections are of two types: if they connect two routers in different ASs, the session established is external BGP (eBGP) and if they connect two routers of the same AS, the session established is internal BGP (iBGP). When a router learns a new route, it spreads it via iBGP sessions to the other routers of the AS and via eBGP sessions to routers outside the AS.

BGP is a *path vector protocol*, i.e., it keeps information about the whole path, in terms of ASs it passes through, as we will further see, and the path changes as the algorithm progresses. In BGP, destinations are prefixes that represent one or more subnets. If an AS has a connection to more than one subnet, it can also announce a prefix corresponding to an aggregation of prefixes [26].

When a router advertises a prefix, it must initialize the mandatory attributes of BGP in order to correctly implement it. A few of these attributes are [27]:

- **Origin -** indicates how the prefix was known, if through iBGP, eBGP or none of them, in the case of static routes.
- **AS-PATH -** keeps information about the whole path, i.e., list of ASNs that the advertisement has passed through. Every time a prefix is advertised, the router appends its ASN to the *AS-PATH*. The *AS-PATH* allows the detection of loops in routes because, if an AS receives a prefix and sees that its own ASN is already in the *AS-PATH*, it simply discards the route.
- **NEXT-HOP -** Internet Protocol (IP) address of the interface of the router to which packets should be sent in order to get them to the destination.

There are also other attributes that allow BGP to define preferences between routes or to provide

additional information about routes. For example, *LOCAL-PREF* is used to choose between routes to the same prefix.

Every route learned to a prefix is stored in the Routing Information Base (RIB). However, out of all the paths stored in the RIB, only the best ones are used to forward packets. These paths are stored in the Forwarding Information Base (FIB), which is responsible for the fast lookup of the *NEXT-HOP*s of each prefix learned. When a router learns a route to a new prefix, or a better route to a previously learned prefix, the older route, if it exists, is withdrawn from the FIB and the new route must be advertised. If the route was learned via an eBGP neighbor, the router advertises it to the other eBGP neighbors, via eBGP sessions, and to all the routers inside the AS via iBGP sessions. However, in the case of eBGP, before advertising the route, the router must append its ASN to the *AS-PATH* and set the *NEXT-HOP*, which is its own address by default. This way, routes to prefixes are propagated throughout the Internet until the algorithm converges.

If a router receives two route advertisements to the same prefix, it chooses the route as follows: if there is a route with a higher *LOCAL-PREF* value, this route is chosen; if both routes have the same, or none, *LOCAL-PREF* value, then the route with the lowest *AS-PATH* length is chosen; if both routes have the same *AS-PATH* length, the route with the closest *NEXT-HOP* router, as defined by the routing protocol implemented inside the AS, is chosen; if this is not enough to produce a single route, the router uses the BGP identifier, an IP address that identifies the BGP router, to select a route.

To exemplify a route advertisement of a prefix, a simple network is depicted in figure 2.8. This network has four ASs, each one with multiple routers. As can be seen, all routers inside an AS have iBGP sessions established between them and some routers have eBGP sessions established with routers from other ASs. For example, router R_4 from AS_2 has an eBGP session with router R_1 from AS_1.

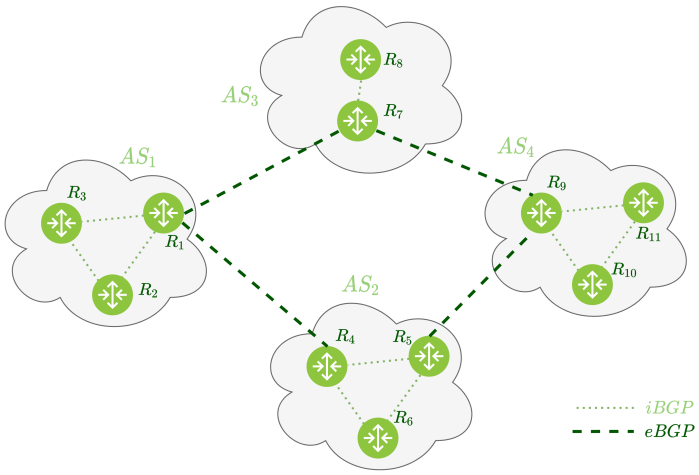


Figure 2.8: Network with four ASs and the BGP sessions established between and within them.

We now present a simplified version of the advertisement of a new prefix to the network to exemplify

Table 2.1: eBGP packets sent in the network of figure 2.8 when R_1 is advertising a route to all the other routers.

Advertisement	Origin	AS-PATH	NEXT-HOP
1	iBGP	AS_1	R_1
2	eBGP	$AS_1 AS_3$	R_7
3	eBGP	$AS_1 AS_2$	R_4
4	eBGP	$AS_1 AS_2 AS_4$	R_9

the working of BGP. If router R_1 wants to advertise the route for prefix $10.0.1.0/24$, for example, it will broadcast this route to its AS neighbors, AS_2 and AS_3, via the eBGP sessions established with routers R_4 and R_7, respectively. The advertisement sent, advertisement 1, has the attributes defined in table 2.1.

Then, routers R_4 and R_7 save the route for the prefix, previously unlearned, and append the ASN to the *AS-PATH* before propagating the route. They propagate it via iBGP to the routers inside their ASs and via eBGP to external neighbors, except R_1 because the route was learned from this router. In this case, R_4 only advertises the route via iBGP because it has no external neighbors other than R_1. Router R_9 learns the route from R_7, with advertisement 2 of table 2.1.

When router R_5 learns the route via iBGP, it advertises the route via eBGP to its external neighbor R_9 with advertisement 3 of table 2.1. Then, router R_9 receives the packet and determines that it already knows this route. Since there is no *LOCAL-PREF* attribute and the length of the *AS-PATH* is the same as the one of the route already stored, it decides which route to choose via the BGP identifier. In this example, supposing router R_4 has a BGP identifier preferable when compared to R_7, router R_9 decides to elect the new route, learned from R_4, as the best path. Finally, router R_9 advertises this route to router R_7 with advertisement 4 of table 2.1. When router R_8 receives this new advertisement, it detects it already knows a route for the prefix, so it makes a lexicographic comparison of the tuple (*LOCAL-PREF, AS-PATH, BGP identifier*) and keeps the previously saved route in the FIB and stores the new one in the RIB.

BGP in data centers

In data centers, only eBGP is used since, according to [16], it is simpler to understand and implement than iBGP and, additionally, it does not have the multi-path limitations that iBGP does. Thus, from now on we will refer to eBGP as BGP. In order to use BGP as the routing protocol in the data center, it has to be adapted to the network topology and the requirements of the network manager. In data centers, BGP configurations should make sure a pair of servers are always connected by the shortest path and, if possible, traffic should be distributed by all the shortest paths available.

Jayaraman et al. [19] provide, in their work, an example of a BGP adaptation to a folded Clos network of a Microsoft data center. Given the well-defined structure of a folded Clos network, the shortest paths

between servers are known and BGP was implemented to take advantage of this, specifically with the assignment of ASNs. An ASN is assigned to each router and ASNs used are private so the internal BGP information is not accidentally leaked to external networks. ASNs are assigned in the following way:

- every T_0 has its own ASN.
- all the T_1s inside a cluster have the same ASN.
- all the T_2s of the data center have the same ASN.

Figure 2.9 is an example of this numbering scheme for a network with 2 clusters and 4 T_0s per cluster.

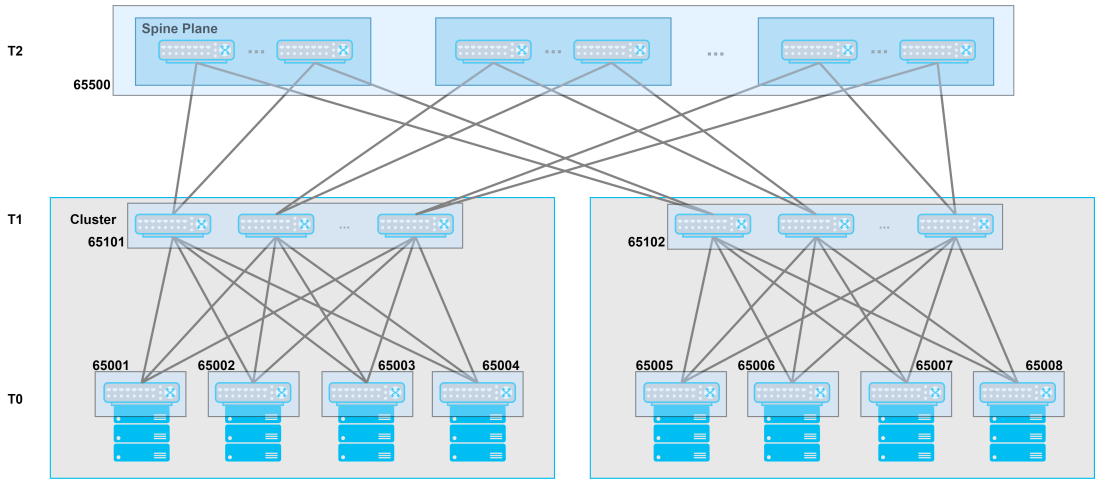


Figure 2.9: Assignment of ASNs to a folded Clos network.

This numbering scheme implies that, due to the loop detection in BGP through the *AS-PATH* attribute, each T_0 only keeps the shortest paths in both the RIB and the FIB. The shortest paths are of length 2 to other T_0s of the same cluster and of length 4 to T_0s of other clusters. When a T_0 advertises a route to another T_0, the T_1 that receives it will see that its ASN is already on the *AS-PATH* and discard the route. The same happens when a T_2 receives an advertisement of a route that has already passed through a T_2. This is advantageous since, given the elevated number of redundant paths in folded Clos topologies, if every path was advertised, BGP messages would go through the entire network but the elected paths would not, for the most part, change [16].

Given the ASN assignment in figure 2.9, an example of a path of length 2 between two T_0s is 65001 - 65101 - 65002, where 65001 and 65002 identify the first and second T_0s of the first cluster, respectively, and 65101 can be any of the T_1s of the first cluster. An example of a path of length 4 between two T_0s of different clusters is 65001 - 65101 - 65500 - 65102 - 65006, where 65101 is any of the T_1s of the first cluster, 65102 is any of the T_1s of the second cluster and 65500 is any of the T_2s,.

By using a BGP extension called *multi-path* [24, 25], it is possible to install many paths of equal cost in the FIB. In the case of the network in figure 2.9, the cost of each route is determined by the *AS-PATH* length and, consequently, routes with the same length have the same cost. Therefore, each router is able to store every shortest path available for each prefix. Then, multi-path uses Equal-Cost Multi-Path (ECMP) to distribute the traffic by the routes installed in the FIB, in order to achieve load-balancing in terms of traffic flows. This way, although a traffic flow is indivisible, different traffic flows can be distributed among different paths. For example, if there are no link failures, router 65001 can use every T_1 of its cluster to send packets to router 65002. This not only minimizes the load of each link used but also increases the bandwidth available between the servers. If, however, we want to limit the number of routes installed in the FIB, in order to diminish the memory needed, it is possible to be configured.

Chapter 3

Network cost estimation and analysis

We now analyze the cost of a data center network. We can divide the cost in two parts: acquisition, related to the phase of building the data center, and operational, the cost of running the data center.

3.1 Acquisition cost

The acquisition cost (Capex) is the cost needed to set up the data center so that it can begin to operate. It comprises the cost of the land, the building, servers, routers, cable links, cooling equipment and other auxiliary equipment.

In this study, we are only analyzing the cost of the network and, consequently, assume that the servers, cooling and auxiliary equipment have a fixed cost, proportional to the dimension of the data center. Thus, we differentiate the acquisition cost of each topology by the cost of the routers and cable links.

The different characteristics and costs of routers used in data centers

The routers used in data centers have characteristics such as:

- *Low latency* - latency in routers is the time that it takes for a router to start sending a packet after receiving it. In the case of data center routers, it should be in the order of nanoseconds.
- *High throughput* - data center routers have ports of 10, 40, or even 100 Gbps.
- *High forwarding capacity* - forwarding capacity represents the number of minimum sized Ethernet packets a router can switch per second. In data centers, it can go as high as billion packets per second (Bpps).
- *High number of ports* - in order to support hundreds of thousands of servers, routers used in data centers can have as many as 200 ports.

With these demanding characteristics, routers used in data centers are expensive and building an entire data center requires thousands of them. This leads to a very competitive market in which data center companies try to minimize the cost of acquiring routers and manufacturers compete between

themselves to provide the best prices. As a consequence, the price of these routers is not, for the most part, publicly available and it is difficult to find reliable sources that provide the price of a router from each manufacturer. Some companies, such as Facebook [1], even started building their own routers. In this study, we use the cost of the Juniper routers provided in [3] to approximate the prices.

The cost of the routers depends on many factors:

- **Memory size** - the Ternary Content-Addressable Memory (TCAM) is used to store the FIB and Access Control Lists (ACLs) [28, 29]. The FIB is responsible for the fast lookup of the *NEXT-HOPs* of each prefix learned and ACLs are rules that tell if packets from specific IP addresses or prefixes should be forwarded or discarded. The RIB, responsible for saving all paths learned, is stored in the Dynamic Random-Access Memory (DRAM) [30], which is hundreds of times cheaper than the TCAM [31].

- **Number of and capacity of each port** - routers have a set of physical ports with a specific maximum capacity. However, it is possible to have different port combinations for the same router.

- **Forwarding capacity**

- **Modularity -** there are two types of routers in terms of structure: fixed routers, as in figure 3.1(a), with a fixed number of ports, and modular routers, depicted in figure 3.1(b), which are composed of a chassis and multiple line cards. The chassis functions as a cabinet that also provides the backplane, the power source and the ventilation. The line cards are inserted in the chassis, resembling drawers, and each one has a set of ports and its own FIB table. The backplane is responsible for the management of the FIB in each line card, so that they all share the same information, and for the switching between line cards.

In order to understand which characteristic has a bigger influence in the cost of these routers, we provide, in table 3.1, the approximated prices of some routers, taken from [3], along with the number of supported FIB (IPv4) routes and forwarding capacity. In order to simplify the analysis, we compare only the TCAM, since it is hundreds of times more expensive than the DRAM, size of each router by comparing the number of FIB (IPv4) routes each router supports. We obtained the number of FIB routes and the forwarding capacity by analyzing each datasheet of the routers, except for the value of the forwarding capacity of the QFX10000-30C, which was not provided in the datasheet.

With the information presented in table 3.1, we can conclude that, since routers with different forwarding capacities and number of ports, but the same number of FIB (IPv4) routes have a similar cost, the memory is what more influences the cost.

Table 3.1: Number and capacity of ports, number of IPv4 routes, forwarding capacity and cost of some routers used in data centers. The number of IPv4 routes and forwarding capacity were taken from datasheets provided by Juniper and the costs are approximations of the ones presented in [3].

Routers	Ports (number x capacity (Gbps))	FIB (IPv4) routes	Forwarding capacity (Bpps)	Price (USD)
QFX5110-48S	48x10 + 4x100	128 000	1.32	20 000
QFX5110-32Q	32x40		1.44	
QFX5200-48Y	48x25 + 6x100		2.1	
QFX5200-32C	32x100		2.4	
QFX5210-64C	64x100	260 000	4.2	30 000
QFX10008 (chassis)	-	2 000 000	16	60 000
QFX10000-30C (line card)	30x100	2 000 000	–	90 000

Example of routers used in each tier and price per port

Routers in the two bottom tiers, T_0s and T_1s, have similar characteristics and are used in clusters, the base unit of folded Clos topologies. Therefore, these routers are not usually used to scale the network and are responsible for switching the traffic of a limited number of servers. Consequently, we are using fixed routers for T_0s and T_1s, i.e., routers with a fixed number of ports. Usually, a rack stores up to 40 servers [16], but Google also claims having a *"custom made server rack"* that supports 80 servers so we consider T_0s and T_1s with a maximum of 96 ports.

Due to the fact that routers in the two bottom tiers, T_0s and T_1s, have similar characteristics, we set the cost of these routers independently of the number of ports. For port speeds of 10 Gbps, we used the Juniper QFX5200-32C router, depicted in figure 3.1(a), which has 32 ports of 100 Gbps but can be configured as a router with 128 ports of 10 Gbps, to set the cost at $20 000. For T_0 and T_1 routers with 40 Gbps ports, we used the Juniper QFX5210-64C router. This router has 64 ports of 100 Gbps but it has a possible configuration of 96 ports of 50 Gbps so we used it to set the price of T_0 and T_1 routers with 40 Gbps ports at $30 000 each.

However, T_2 routers have different and more demanding characteristics when compared to T_0s and T_1s. Firstly, since T_2s connect the data center to the Internet, they need to know routes imported from all over the Internet and, consequently, need bigger FIBs. Additionally, since T_2s connect clusters to each other and the Internet, they have a faster backplane, when compared to T_0s and T_1s, in order to provide a higher forwarding capacity.

Due to these characteristics, for T_2s we used modular routers, the Juniper chassis QFX10008, that supports 8 line cards, and QFX10000-30C line cards. These line cards have 30 physical ports and two possible configurations that interest us, 96 ports of 10 Gbps or 30 ports of 40 Gbps.

(a) A fixed router, Juniper QFX5200-23C. Taken from [32].

(b) A modular router, Juniper QFX10008, with 8 line cards. Taken from [33].

Figure 3.1: Example of a fixed router and a modular router.

Therefore, each T_2 has the fixed cost of the chassis QFX10008, which is, approximately, \$60 000, and the variable cost of the number of line cards used, at a price of \$90 000 per line card. For example, for a T_2 router of 192x10 Gbps ports we only need the chassis and two line cards but, for a 192x40 Gbps ports we need, in addition to the chassis, $192/30 = 7$ line cards.

The prices of routers are summarized in table 3.2. As an example, we can compare the price of routers with 96 ports of 10 Gbps. In the case of T_0s and T_1s, the price is $\$20\ 000$ but, in the case of T_2s, it is $60\ 000 + 90\ 000 \times \frac{96}{96} = 150\ 000$. We can see that, in this case, T_2s have a cost more than 7 times higher than T_1s and T_0s with the same number of ports. This difference derives from the different characteristics required in both types of routers, as was previously discussed.

In figure 3.2, we present the cost per port of each type of router. We can conclude, by analyzing the figure, that by setting a fixed cost of T_0s and T_1s, the cost per port is inversely proportional to the number of ports. In the case of T_2s, the cost per port also diminishes with the number of ports. This is because the cost has a fixed part, corresponding to the chassis, and a variable cost corresponding to the number of line cards used, that is, the number of ports. Thus, T_2s with a higher number of ports distribute the cost of the chassis by more ports and, consequently, the cost per port decreases.

Table 3.2: Price of each router and each cable.

Throughput	Cost of each router ($)		Price per connection ($)		
	T_0s and T_1s	T_2s	Server to T_0	T_0 to T_1	T_1 to T_2
10 Gbps	20 000	$60\,000 + 90\,000 \times \left\lceil \frac{nrofports}{96} \right\rceil$	140	240	420
40 Gbps	30 000	$60\,000 + 90\,000 \times \left\lceil \frac{nrofports}{30} \right\rceil$	240	390	680

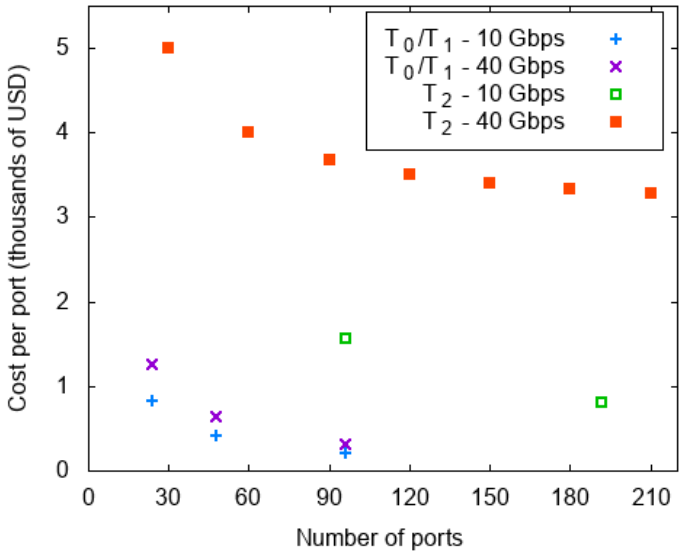
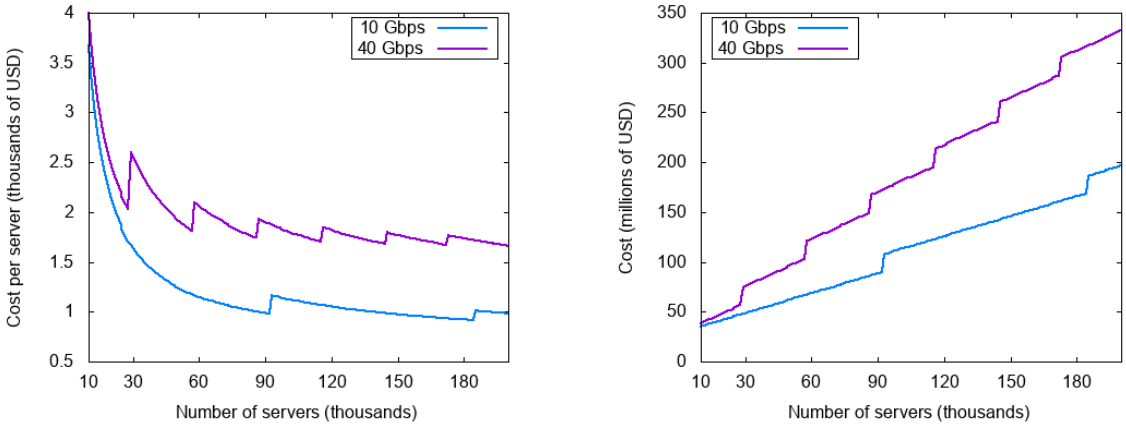


Figure 3.2: Cost per port of each type of router used in the network.

We can also derive that, although 40 Gbps routers have 4 times the capacity per port of 10 Gbps routers, the difference in the cost is not as high. In T_0s and T_1s, 40 Gbps routers cost up to 1.5 times more than 10 Gbps routers. In T_2s, routers of 40 Gbps can cost up to 3 times more than routers of 10 Gbps.

Comparison of the cost of routers of different network sizes

In order to analyze the cost of the networks, in terms of routers, with a different number of servers, we fixed a cluster of approximately a thousand servers. We used T_0s of 48 ports and an oversubscription of 5:1, which results in 40 servers per T_0 and 8 T_1s per cluster. We used T_1s of 48 ports and chose to have no oversubscription at the T_1s so they connect to 24 T_0s and 24 T_2s. Thus, we started with a single cluster and 24 T_2s and, using the same T_2s, we kept adding clusters to find the cost per server of each network. In figure 3.3(a), we can see that the cost per sever tends to diminish with the increase of the number of servers. It can be concluded that this tendency is due to the evolution of cost of the T_2s with the number of ports. We also present the total cost of routers of these networks in figure 3.3(b).



(a) Cost per server.

(b) Total cost.

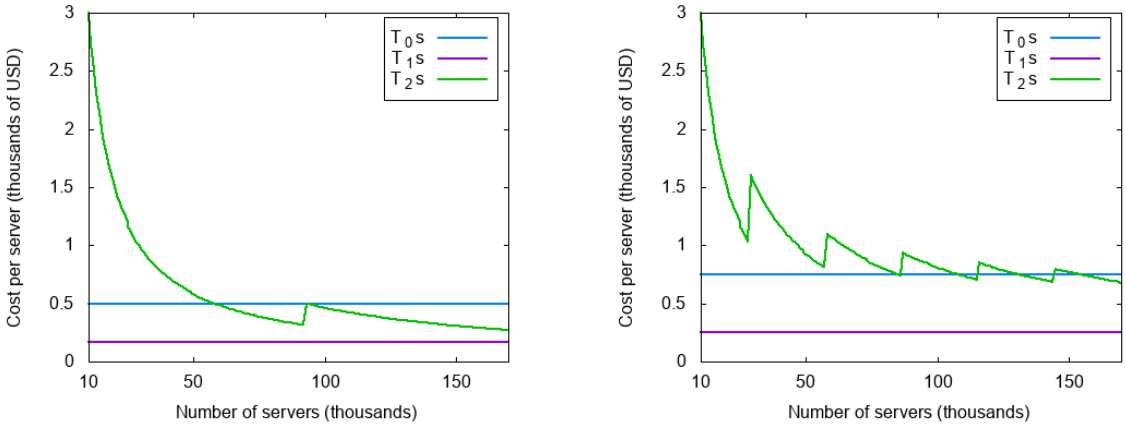
Figure 3.3: Cost of routers as a function of the number of clusters and, consequently, servers, of a network.

When we want to add more servers, which translates, in this case, to adding more clusters, we have two options, either there are available ports in T_2s that we can use, or we have to add a line card to each T_2 in order to accommodate the new cluster. Thus, when we add a line card to each T_2, the cost per server rises abruptly due to the additional cost of the line card. However, as we keep adding clusters to the free ports of the line card previously inserted, the cost of the T_2s gets distributed by a bigger number of servers, resulting in a lower cost per server. This can be seen in figure 3.3(a), in which the peaks represent the added cost of acquiring more line cards.

Additionally, it can be seen that, as the network grows, the difference between the cost per server diminishes and networks with 40 Gbps links are, at most, 2 times more expensive than networks with 10 Gbps links.

Using the same networks of figure 3.3(a), it is possible to analyze the cost per server of each type of router, T_0s, T_1s and T_2s, and, consequently, determine the importance of each one in the overall cost of the network. This is depicted in figure 3.4 for networks of 10 Gbps and 40 Gbps.

In figure 3.4, we can observe that, since the clusters are fixed, T_0s and T_1s have, naturally, a constant cost per server, and that the T_2s follow the tendency of the overall cost of the network represented in figure 3.3(a), which asserts our conclusions that the T_2s are responsible for this tendency. We also verify that, in small networks, the T_2s are responsible for most of the cost of the network and, as the network grows, the cost of T_0s gets more relevant and ends up surpassing the cost of T_2s, which can be seen more clearly in networks of 10 Gbps.



(a) Networks of 10 Gbps.

(b) Networks of 40 Gbps.

Figure 3.4: Cost per server of each type of routers as a function of the number of clusters and, consequently, servers.

Reducing the cost per server by using parallel links

Nonetheless, it is possible to reduce the number of T_2s and, consequently, the total cost of the network, while maintaining the number of links between routers. We can do this by using fewer T_2s and employing parallel links between the same pair of routers T_1s-T_2s while maintaining the total number of links between a T_1 and all T_2s. For example, if a T_1 has a single link to 24 T_2s, it may be possible to connect it to 12 T_2s and, instead of a single link per T_2, make use of two links in order to maintain the total number of links between a T_1 and T_2s. Since, without parallel links, each T_2 has one connection per cluster, as long as it is possible, either by using free ports or by adding line cards, to accommodate 2, 3, or more, connections per cluster, then we can use parallel links. By using parallel links, we eliminate the fixed cost of the chassis of the routers that will not be used and it may even be possible to reduce the total number of line cards used since we may be making use of available ports, and, consequently, the cost of the network will decrease.

In order to evaluate the savings allowed by using parallel links, we developed a program in python that generates all topologies. The program has as input arguments the number of servers that the network must support and the number of ports of each router.

In order to simplify the execution of the algorithm, we generate topologies using routers with a fixed number of ports. For T_0s and T_1s the number of ports we use are 24, 48 and 96. For T_2s we use 24, 48, 96 and 192 ports. We then generate all the different combinations possible of T_0s, T_1s and T_2s with these numbers of ports and run algorithm 3.1, which generates all possible topologies for each set of routers.

To generate a topology, we start with no oversubscription at the T_0s, which implies that each T_0

connects to as many servers as T_1s. Then, the number of servers that connect to T_0s allows us to calculate the total number of T_0s needed to support the number of servers in the network.

Afterwards, we, once again, begin with no oversubscription at the T_1s so half of the ports will be connected to T_0s and the other half to T_2s. We can now calculate the number of T_0s inside a cluster and, consequently, the number of clusters needed in the network.

Since each T_2 has at least one connection to one T_1 of each cluster, the number of ports of a T_2 has to be equal or greater than the number of clusters. If this is not possible in the topology being generated, the network is discarded. However, if T_2s still have unused ports, then it may be possible to increase parallel connections between a T_1-T_2 pair and decrease the number of spine sets, i.e. the number of T_2s in the topology, while maintaining the number of connections between T_1s and T_2s. We generate new topologies by incrementing parallel links as long as there are ports available.

We also generate new topologies by increasing the oversubscription in the T_1s, so that they connect to more T_0s than T_2s, thus increasing the size of clusters in terms of servers. We also change the oversubscription of T_0s and end up with all the topologies possible, given the set of routers used. It should be noted that, even though we are using a finite set of routers, we still obtain, for example, more than 50 000 topologies for networks with a hundred thousand servers.

Finally, when a topology is generated, we obtain the following network characteristics:

- Number of ports of each type of router in the topology
- Number of T_0s, T_1s, T_2s and clusters
- Oversubscription at the T_0s and T_1s
- Router acquisition cost
- Length of cable needed

We ran the algorithm for networks with 10 000, 50 000, 100 000 and 150 000 servers. Then, in figure 3.5, we compare the cost of the cheapest networks that make, and do not make, use of parallel links. It should be noted that all networks have the same level of resilience to link failures, calculated using expressions indicated in chapter 4, so that we can truthfully compare the cost per server.

From figure 3.5, it can be concluded that parallel links allow a larger reduction, in terms of percentage of cost, in smaller networks. This is due to the fact that, in smaller networks, the cost of a T_2 chassis has a bigger impact in the cost per server due to the fewer number of servers. Hence, networks with parallel links can cost up to 61% less in small networks of 10 Gbps links and up to 55% less in networks of 40 Gbps links. Nonetheless, even in the biggest networks, with approximately a hundred and fifty thousand of servers, the cost can be reduced up to 18% in 10 Gbps networks and 6% in 40 Gbps networks.

Algorithm 3.1: GenerateTopologies(nrServers)

```
begin
    for i ← NrPorts(T0)/2 to NrPorts(T0) do
        nrServersPerT0 ⟵ i
        nrT1sPerCluster ⟵ NrPorts(T0) − nrServersPerT0
        nrT0s ⟵ nrServers/nrServersPerT0
        for j ← NrPorts(T1)/2 to NrPorts(T1) do
            nrT0sPerCluster ⟵ j
            nrClusters ⟵ nrT0s/nrT0sPerCluster
            maxLinksT1T2 ⟵ NrPorts(T1) − nrT0sPerCluster
            if nrClusters > NrPorts(T2) then
                continue
            nrParallelLinksT1T2 ⟵ 1
            nrSpineSets ⟵ NrPorts(T1) − nrT0sPerCluster
            nrPortsAvailableT1 ⟵ maxLinksT1T2
            nrPortsAvailableT2 ⟵ NrPorts(T2) − nrClusters
            while nrPortsAvailableT1 ≥ 0 and nrPortsAvailableT2 ≥ 0 do
                PrintTopology(nrServersPerT0, nrT0sPerCluster, nrT1sPerCluster,
                  nrClusters, nrSpineSets, nrParallelLinks)
                nrParallelLinksT1T2 ⟵ nrParallelLinksT1T2 + 1
                nrSpineSets ⟵ maxLinksT1T2/nrParallelLinksT1T2
                nrPortsAvailableT2 ⟵ nrPortsAvailableT2 − nrClusters
                nrPortsAvailableT1 ⟵ maxLinksT1T2 − nrSpineSets × nrParallelLinksT1T2
```

Cost of cables

After determining the routers used, we still need to calculate how much it will cost to connect them, and the servers, using fiber optical cables.

Firstly, we need to know the distance between routers and between servers to T_0s. In data centers, the distance between each tier increases as we go further up the tiers. The length of cable needed to connect equipments in different tiers is, according to [3], the following:

- Server to T_0 - 5 m
- T_0 to T_1 - 20 m
- T_1 to T_2 - 50 m

These lengths can, however, differ with the topology. For example, using parallel links may simplify the cabling and shorten the distance between T_1s and T_2s.

We also consider two different throughputs, 10 Gbps and 40 Gbps, and the price will be different for each throughput. In the case of 40 Gbps links, we use approximate prices [3] of QSFP+ cables, which

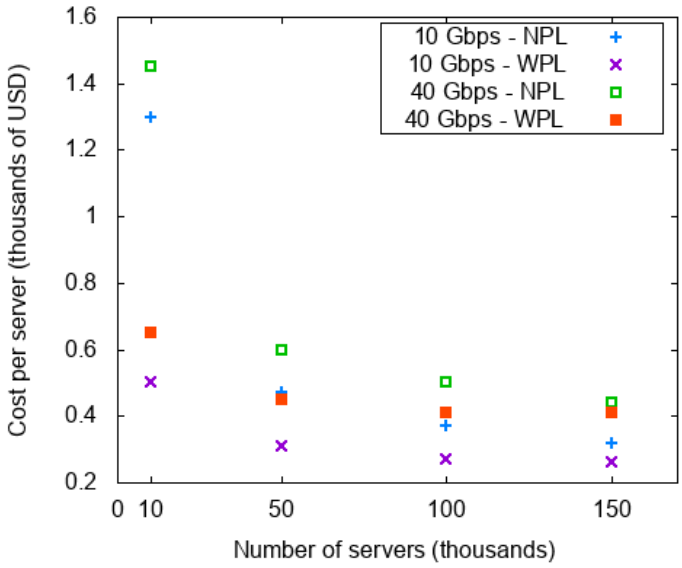


Figure 3.5: Cost per server of networks with and without parallel links. NPL - No parallel links. WPL - With parallel links.

are multimode and active optic cables of 40 Gbps. However, 10 Gbps links usually use SFP+ cables and the only price mentioned in [3] is of a 5 m length cable. Therefore, we will assume the price of SFP+ cables increases with the distance in the same proportion that QSFP+ cables do, in order to estimate the price of cable links with 20 m and 50 m.

The price per cable is shown in table 3.2 and, as can be concluded, the cost increases with the distance between equipments and also with the throughput.

3.2 Operational cost

After setting up a data center, we need to take into account the operational cost (Opex), i.e., the cost of keeping the data center running. This includes the power used by all the equipment and its maintenance.

Once again, we are analyzing only the operational cost of the network and consider the following costs as fixed: the cost needed to power anything but the network equipment and the maintenance of the data center.

Now, we need to define the cost needed to support each topology. Since the cost per power unit changes with the data center location, we provide the total power used as a measure of cost. To do this, we calculate the power utilized by the routers and assume that the cooling equipment needed for the network will use a proportional cost. Thus, even though the cooling power is not fixed, it is sufficient to calculate the power used by the routers to analyze and evaluate the different networks in terms of operational cost.

Determining the power consumed by the routers

To do this, we analyzed the routers previously used to define the acquisition cost. In each router, there are two power limits, the minimum power is used when the router is idle and the maximum power consumed occurs when the traffic load is maximum. In [3] it is said that the power used varies linearly with the number of ports in use and the traffic load. Thus, in our calculations we assumed a traffic load of 50% with half of the ports in use.

Similarly to what was done in the acquisition cost, we assume that both T_0s and T_1s consume the same, 389 W in the case of routers with 10 Gbps per port, and 688.5 W when using routers with 40 Gbps ports. Regarding the T_2s, similarly to what was done to calculate their cost, the total power is the sum of the power consumed by the chassis, 1483 W, and the energy consumption of each line card, 891 W.

The power consumed by the different routers is summarized in table 3.3.

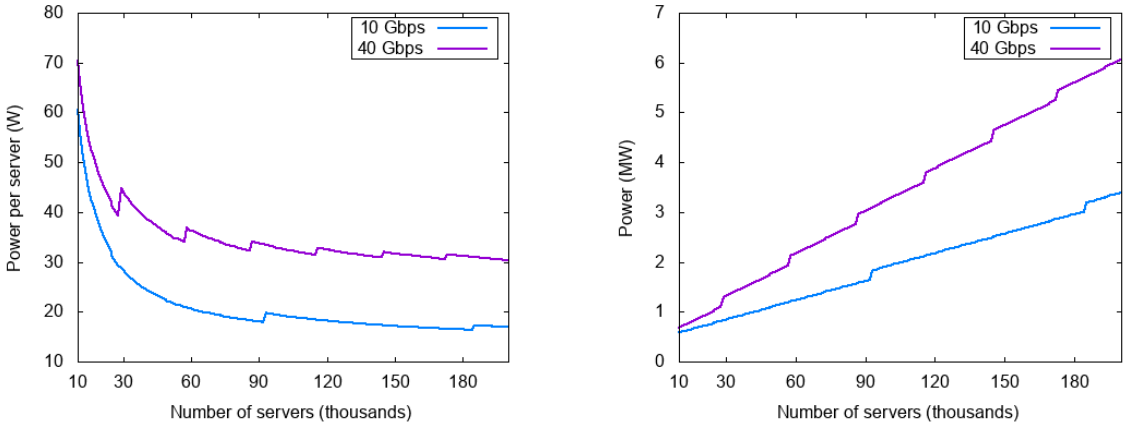
Table 3.3: Power consumed by each type of router.

Throughput	Power consumed by router (W)	
	T_0s and T_1s	T_2s
10 Gbps	389	$1483 + 891 \times \left\lceil \frac{nrofports}{96} \right\rceil$
40 Gbps	688.5	$1483 + 891 \times \left\lceil \frac{nrofports}{30} \right\rceil$

Comparison of power per server of different network sizes

In order to compare the power consumed by networks of different sizes, that is, with a different number of servers, we executed the same process that resulted in figures 3.3(a) and 3.3(b). The power per server of the different networks created can be seen in figure 3.6(a) and the total power is depicted in 3.6(b).

Similar conclusions to the ones drawn from figure 3.3(a) can be derived from figure 3.6(a). When we add more clusters to the network, two different situations occur: if we add them to available ports then the overall power of the network can be distributed by more servers and, consequently, the power per server decreases, but, when we need to add more line cards in order to add clusters, the power by server suddenly increases due to the added power of the new line cards. Additionally, we can also conclude from the figure that networks with 40 Gbps links only use up to 2 times more power than 10 Gbps networks.



(a) Power per server.

(b) Total power.

Figure 3.6: Power of routers as a function of the number of clusters and, consequently, servers, of a network.

Reducing the power per server by using parallel links

We also compared the power consumed when using parallel links between T_1s and T_2s. In order to do this, we modified the program that generates all topologies so that it also outputs the power consumed by the routers in the network. Once again, we ran the algorithm for networks with 10 000, 50 000, 100 000 and 150 000 servers. Then, we obtained the lowest value of power used by networks with the same level of resilience to link failures and that make, and do not make, use of parallel links.

The results are depicted in figure 3.7 and, as with the acquisition cost of routers, we can see that using parallel links enables the saving of power. This is, once again, due to using fewer T_2s and, consequently, eliminating, at least, the power consumed by the chassis of the T_2s removed.

The savings go from, approximately, 15% to 58%, in the case of networks with links of 10 Gbps, and from 7% to 48% in networks with 40 Gbps links. Additionally, the relative savings are, once again, higher in the smaller networks due to the larger impact that the power consumed by the chassis has when distributed by fewer servers.

3.3 Conclusions

In this chapter, we identified the main characteristics of the routers that compose a network of a data center. Specifically, T_2 routers need more memory and forwarding capacity than T_0s and T_1s. In order to interconnect the clusters and connect them to external endpoints to the data center, T_2 routers need a faster backplane to provide higher forwarding capacities, and bigger RIBs and FIBs in order to install the routes to prefixes from all over the Internet.

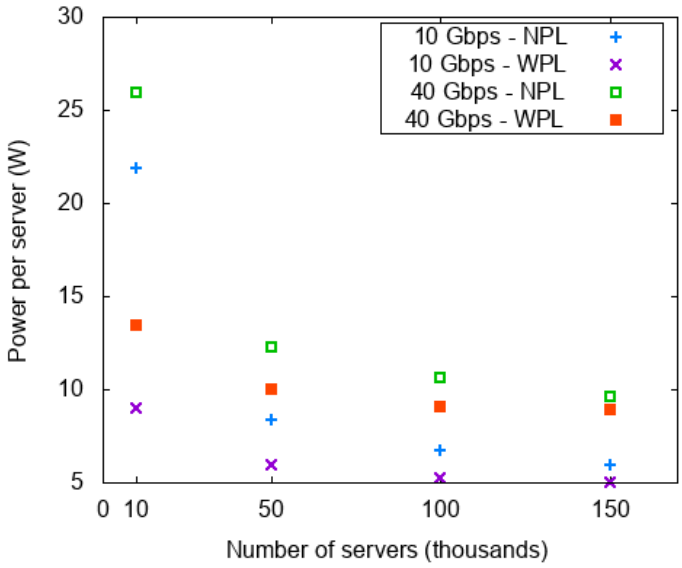


Figure 3.7: Power per server of networks with and without parallel links. NPL - No parallel links. WPL - With parallel links.

Then, we determined an estimate of the cost per port of routers and concluded that the cost diminishes with the increase of the number of ports.

We calculated the cost of routers of different topologies and concluded that the cost per server diminishes with the increase of the number of servers due to the modularity of T_2 routers. By adding clusters to the networks and, consequently, increasing the number of servers, the fixed cost of the chassis of T_2s is progressively distributed by a bigger number of servers, which leads to a reduction of the cost per server.

We determined that networks with 40 Gbps links cost up to 2 times more than 10 Gbps networks, even though the capacity of 40 Gbps networks is 4 times higher. Additionally, we saw that T_2s are responsible for higher percentages of the cost of routers of small networks, up to 50 000 servers in the case of 10 Gbps networks, but, as the number of servers in a network increases, T_0s surpass the cost of T_2s.

We estimated the power used by routers of different topologies and were able to determine that the power used diminishes with the increase of the number of servers due to, similarly to the cost, the modularity of T_2 routers. In the case of power usage, networks with 40 Gbps links consume up to 2 times more than networks with 10 Gbps links.

We determined that, while maintaining the number of connections between a T_1 and T_2s, using parallel links allows the reduction of up to 61% of the overall cost of the network and up to 58% of the power used.

Chapter 4

Connectivity Analysis

Data center companies have to reimburse clients if the data center availability falls below a certain threshold. For example, Azure has to reimburse 10% of the credits of clients when they have access to a virtual machine less than 99.99% of the time during a single month [34]. It should be noted that, in this case, 99.99% of availability, or uptime, in a month is approximately four minutes.

Consequently, one of the objectives of data centers is to maximize the redundancy of the network so that, when a link fails or a server unexpectedly crashes, the network can recover in the shortest time possible in order to minimize the risk of needing to refund clients.

Additionally, since there are different applications running in data centers, the concept of availability differs from application to application. For example, in distributed applications, such as the training of a machine learning model, the availability refers to connections between servers inside the data center. However, when clients are trying to access virtual machines in the data center, the concept of availability is applied to connections between the data center servers and the Internet. Therefore, we study the impact of link failures in three different types of connections: between servers of the same cluster, between servers of different clusters and between a server and the Internet.

In the networks we study, we do not consider redundancy between servers and T_0s so every server has only one connection to a T_0, independently of the network. Therefore, this allows us to focus solely on the connectivity from and to T_0s, since only T_0-T_1 and T_1-T_2 links have an impact in the resilience to link failures of each network.

4.1 Connectivity inside a cluster

As previously mentioned in 2.1, in data centers T_0s and T_1s only connect to other T_1s and T_0s, respectively, of the same cluster. A T_0 connects to every T_1 of the cluster and a T_1 connects to all T_0s of the same cluster, as depicted in figure 2.3(a).

If k is the number of routers T_0 inside a cluster and m is the number of T_1s inside a cluster, we can conclude that there are m paths, **pairwise disjoint** and all of **length 2**, connecting two T_0s **of the same cluster**.

Thus, to connect 2 T_0s of a cluster via a T_1, as in green in figure 4.2, both links connecting the T_1 to the T_0s must be available. Since there are m T_1s, it only takes one link failure per T_1, either to the

source or the destination, to disconnect 2 T_0s, which totals m failures. Additionally, if x is the probability of a link failure between a T_0 and a T_1 and assuming that each failure is independent, the probability of 2 T_0s connecting via a single T1 is

$$P(\text{connectivity between 2 } T_0\text{s via a single } T_1) = (1-x)^2. \tag{4.1}$$

Since there are m T_1s in each cluster, as long as there is one of the m paths available, we can connect 2 T_0s. Hence, the probability that all paths are unavailable and, consequently, a pair of T_0s cannot connect to each other, is given by

$$P(\text{no connectivity}) = (1-(1-x)^2)^m. \tag{4.2}$$

Finally, the probability that 2 T_0s can establish a connection is

$$P(\text{connectivity}) = 1 - P(\text{no connectivity}) = 1-(1-(1-x)^2)^m. \tag{4.3}$$

We are now able to approximate the availability that data centers define in SLAs by the probability of connectivity. In this case, we are studying connectivity between T_0s of a cluster, which is more important in applications that rely on connections between servers, such as distributed applications.

Since data centers aim to have maximum availability and need to refund their clients if it falls below a threshold as high as 99.99%, we examine how many link failures a network can support while 99.999%, 99.99% and 99.9%, that is, 5x9, 4x9, and 3x9, respectively, of the pairs of T_0s are able to connect with each other. Thus, if we equal equation 4.3 to the different values of availability, we get the maximum probabilities of failure, for different values of T_1s per cluster, m, as represented in figure 4.1.

Increasing the resilience to link failures using valleys

In the *fabric* design of Facebook [1], there are only 4 T_1s per cluster. Thus, if a link has a probability of failure higher than 2.85%, the connectivity between T_0s would fall below 99.999%. Their new design, with *F16s* [2], increased the number of T_1s per cluster from 4 to 16, which resulted in a higher support of link failures of 28.37%.

However, instead of upgrading existing networks, it is possible to use *valleys* as a work around to increase availability. Valleys are paths with an up-down-up-down shape that use more than a single T_1 to connect two T_0s, as can be seen in orange in figure 4.2.

It should be noted that, in order to do this, the BGP implementation described in chapter 2 needed to be modified because, since T_1s of a cluster share the same ASN, valley shaped paths were discarded when reaching the second T_1. Therefore, an additional BGP command was added in each T_1 so they can accept routes advertised by T_0s that already have the ASN of the T_1 in the path. The command inserted,

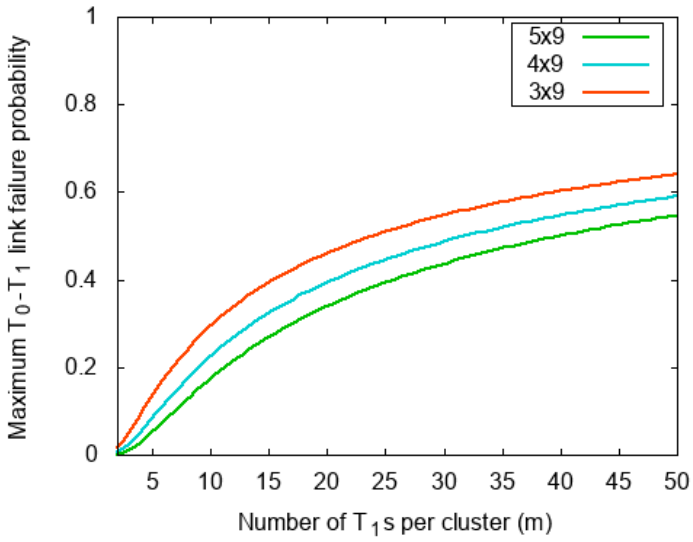


Figure 4.1: Maximum probabilities of a T_0-T_1 link failure that a network supports, for 99.999%, 99.99% and 99.9% of connectivity between T_0s, and for different values of T_1s per cluster, m.

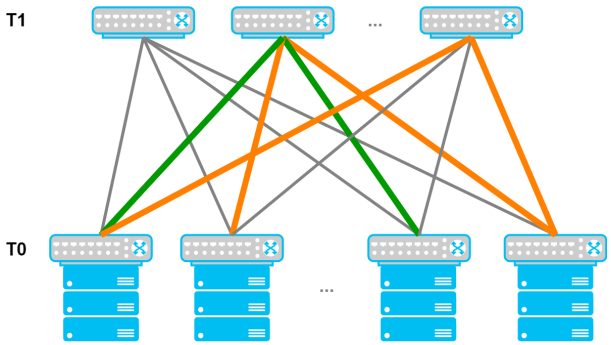


Figure 4.2: Example of a 2-tier folded Clos network with two different types of path that connect pairs of ToRs: in green, a path of length 2, and, in orange, a valley shaped path of length 4.

`neighbor <ip-address> allowas-in 1`, allows a single valley, so T_1s only accept and re-advertise routes where their ASN appears in the *AS-PATH*, at maximum, once.

It is also worth mentioning that, when two servers of different clusters communicate with each other, only valleys inside clusters are permitted, that is, a path is not allowed to go through T_2s twice. In figure 4.3, we can see two types of valley shaped paths between T_0s of different clusters: in green, an allowed path of length 8 and, in red, a path of length 6 that is not allowed due to the valley between T_1s and T_2s.

In order to understand the importance of valleys, we examine now how many link failures it takes for two T_0s **of the same cluster** to be unable to connect with each other when valleys are allowed.

If we can connect 2 T_0s through a single T_1, then there is a T_1 that has active links to both T_0s. Thus, if all T_1s only have a link to the T_0 source or the destination, they can no longer connect, unless we use valleys.

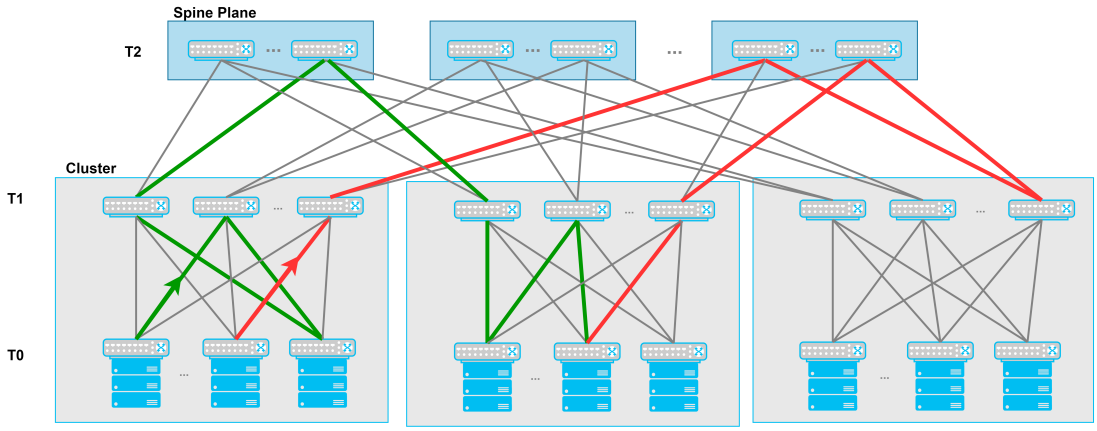


Figure 4.3: Example of a 3-tier folded Clos network with two different types of path that connect pairs of ToRs in different clusters: in green, a path of length 8, and, in red, a valley shaped path that is not allowed.

When we use valleys to connect 2 T_0s, we connect them through an intermediate T_0 that has active links to at least 2 T_1s, one that has an active link to the source and another one that has an active link to the destination.

Therefore, in order to disconnect 2 T_0s, even if the option of valleys is allowed, each intermediate T_0 must only be able to either connect to T_1s that can only connect to the source, or to T_1s that only have a connection to the destination. Thus, if a T_0 can only connect to T_1s that only have active links to the destination, then the T_0 is connected to the same T_1s that the destination is and, consequently, has the exact same link failures that the source does.

If the source is a and has x link failures, the destination is b with, consequently, $m - x$ failures, since there needs to be one failure per T_1, and, finally, every remaining T_0 has, at least, the same failures that a or b have, then **the minimum number of failures that completely disconnect 2 T_0s of the same cluster is:**

$$m + \min(x, m - x) \times (k - 2). \tag{4.4}$$

This expression can take values considerably higher than the m failures it takes to disconnect two T_0s without the use of valleys and, consequently, allowing valleys should reflect an increase in the maximum probability of link failures that support, for example, 99.999% of connectivity between pairs of T_0s.

In order to evaluate the impact of valleys in connectivity between T_0s, we developed a C++ program and implemented algorithm 4.1 to find the shortest paths between 2 nodes, which are, in this case, T_0s.

This program takes as input a given network, in order to create a graph that represents it, and whether or not valleys are allowed. Afterwards, we apply a probability of failure in the network links and select which type of links can fail, i.e., we can apply a probability of failure only in T_0-T_1 or T_1-T_2 links

or, alternatively, in both. This allows us to examine the impact of failures on the different tiers of the network.

Then, we run algorithm 4.1 for every ToR of the network. This algorithm starts at the source, s, and puts its neighbors, which are, in this case, the T_1s that it has a connection to, in a first in, first out queue. Since each link in the network has a unitary cost and we are visiting the neighbor nodes before moving to further depths of the graph, as in a Breadth-First Search (BFS), every time we put a node in the queue we determine the shortest path to it and which will never be shorter than the paths to nodes already in the queue. Consequently, we are able to use a first in, first out queue instead of a priority one. Thus, the algorithm progressively visits the T_1s and afterwards the T_0s.

When visiting a node, however, the algorithm must also see if the ASN of each neighbor of the node is already on the *AS-PATH*. If the ASN of the neighbor does not appear on the *AS-PATH*, then it is inserted in the queue. However, if the ASN is already on the *AS-PATH*, the algorithm checks if valleys are allowed or not. If valleys are not allowed, the neighbor is not inserted in the queue. If valleys are allowed and the node visited is a T_0, then the T_1 neighbor is inserted if its ASN appears only once on the *AS-PATH*. In the end, the program gives us the percentage of pairs of T_0s that can no longer establish a connection after applying link failures.

Algorithm 4.1: FindPath(s, valleys)

```
begin
    for each v do
        dist[v] ⟵ +∞
        visited[v] ⟵ false
        path[v] ⟵ ∅
    Q ⟵ s
    dist[s] ⟵ 0
    visited[s] ⟵ true
    while Q ≠ ∅ do
        select u of Q for which d[u] is smallest
        Q ⟵ Q − {u}
        for each adjacent node v of u do
            if visited[v] ≠ true then
                if ASNInPath(v, u) = 1 and (valleys ≠ true or IsT2(v) = true) or
                  ASNInPath(v, u) = 2 then
                    continue
                dist[v] ⟵ dist[u] + 1
                visited[v] ⟵ true
                path[v] ⟵ u
                Q ⟵ Q + {v}
```

In order to analyze the connectivity between T_0s of a cluster, with and without valleys, we chose a network composed of a single cluster of 48 T_0s and, for each value of m, i.e., T_1s per cluster, ran the algorithm for a given probability of failure of T_0-T_1 links. Then, we evaluated the average of connectivity between T_0s and kept running the algorithm, increasing the probability of failure, in order to find the maximum probability of failure that supports 5x9 of connectivity. We did this for a thousand samples for each probability, in order to increase the degree of confidence. The results are represented in figure 4.4.

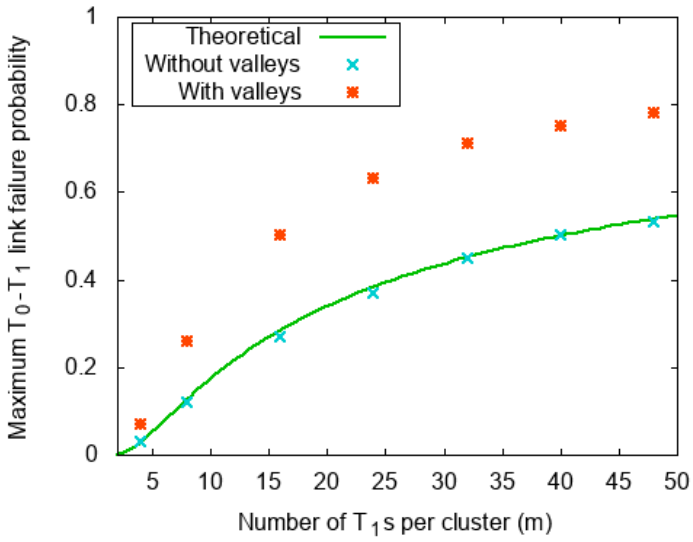


Figure 4.4: Comparison between the maximum probabilities of a T_0-T_1 link failure that a network supports, for 99.999% connectivity between T_0s, and for different values of T_1s per cluster, m, with and without the use of valleys.

As can be seen in figure 4.4, the **use of valleys increases the resilience to link failures of a network without needing to upgrade its equipment**. It is also especially relevant for low values of the number of T_1s per cluster. For example, for 8 T_1s per cluster, the maximum probability of a T_0-T_1 link failure increases from 12% to 26% by using valleys.

It could be possible to increase the resilience to link failures by making use of even longer paths and, consequently, allowing more valleys. However, a single valley already provides a significant number of paths and allowing more valleys would not be as advantageous. Additionally, using valleys also has some considerations that must be taken into account. Firstly, in the T_1s, the size of the RIB and the FIB grows, the latter only in the presence of link failures. The size of the RIB of T_0s and T_2s also increases with link failures, when valleys are being used. The increase in memory size when using valleys will be discussed in more detail in section 4.5. Secondly, valley paths are longer and use more routers than non-valley paths, which results in paths with higher latency. Finally, when using valleys we are no longer using only disjoint paths and, consequently, when distributing the traffic, the links in common of these paths may end up with a load higher than their capacity. In section 5.2, we test the option of traffic with

valleys and discuss it in more detail.

We also decided to test two different modes of link failures using the program, link failure probability and percentage of links down, which represent different failures in data centers. The probability of link failure represents failures over time and does not take into account the mean time to repair, that is, the time between the occurrence of a failure and its repair. Since mean time to repair can be directly related to the maintenance of the data center and differs from company to company, we opted to test the mode of percentage of links down. This mode represents an instant in time, that is, the percentage of links down that the network has simultaneously.

When applying link failure probability, we go through each link and select a random value that tell us if the link fails or not, according to the probability of failure. With percentage of links down we randomly select, from the pool of links, the links that will fail, until the number of links down totals the selected percentage. The values obtained theoretically for the maximum probability of T_0-T_1 link failure that supports 99.999% of connectivity and the values given by the algorithm for both modes of link failures can be compared in figure 4.5.

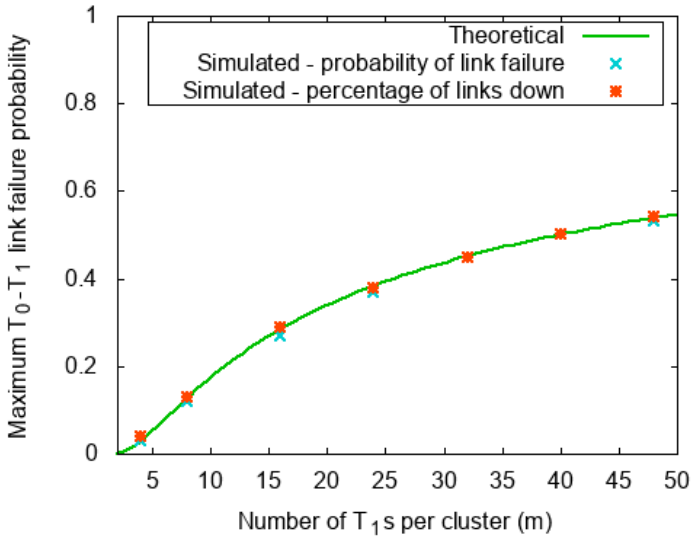


Figure 4.5: Comparison between the theoretical and simulated values of the maximum T_0-T_1 link failures that the network supports, for 99.999% of connectivity between T_0s, and different values of T_1s per cluster, m.

As can be seen, the different types of simulated values are similar. We can also conclude that, since we are evaluating large networks, the number of links is sufficiently high for there to be no relevant difference between the two modes of link failures.

Relationship between the resilience to T_0-T_1 link failures and the cost

From figure 4.1, it is possible to conclude that the maximum probability of a T_0-T_1 link failure increases more rapidly for a low number of T_1s per cluster, especially between 1 and 10, which translates, in the case of 5x9, from 0% to, approximately, 17% of link failure probability. Past these values, the growth rate of maximum T_0-T_1 link failure probabilities that support 99.999% of connectivity between T_0s of a cluster decreases. This leads to the conclusion that, in the beginning of the curve, that is, for low values of link failure probabilities, adding a T_1 to the network has a greater impact in the resilience to link failures than at the end of the curve. For example, going from 5 T_1s to 10 T_1s per cluster increases the maximum probability of T_0-T_1 link failures from 5% to 17%, an increase of 240%. However, going from 15 T_1s to 30 T_1s per cluster, which is also doubling the number of routers, only increases the maximum probability from 26% to 43%, which is an increase of 65%.

This has, naturally, an impact on the cost of the networks, in terms of maximum percentage of T_0-T_1 links down they can support. By using equation 4.3, we were able to modify the program that generates all topologies, mentioned in section 3.1, so that it also outputs the maximum probability of a T_0-T_1 link failure that a network supports for 99.999%, 99.99%, and 99.9% of connectivity between T_0s of a cluster. Thus, we were able to build the graph in figure 4.6, which presents the cheapest networks of 10 Gbps links that support 99.999% of connectivity, for different values of maximum T_0-T_1 link probabilities. It can be seen, for example, that for a maximum probability of link failure between 5% and 15%, the cost per server is similar. However, after a probability of link failure of 15%, the cost starts to increase.

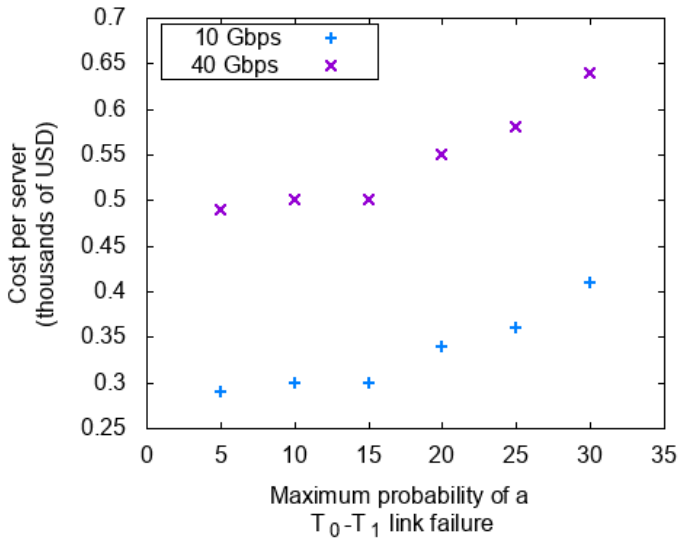


Figure 4.6: Cost per sever of networks of 10 Gbps links with different maximum probabilities of T_0-T_1 link failures that each network supports for 99.999% of connectivity between servers.

4.2 Connectivity between clusters

After studying clusters as a unit, we now need to study connectivity between T_0s of different clusters. As previously mentioned in 2.1, clusters are connected via spine planes, as in figure 2.4.

Since each T_0 connects to a T_1 and each T_1 connects to n T_2s, we can determine that there are $m \times n$ shortest paths, of **length 4**, connecting two T_0s of different clusters. However, only m of these paths are **pairwise disjoint**. Thus, if a link between a T_0 and a T_1 fails, there are $(m-1) \times n$ paths left between that T_0 and T_0s of other clusters. However, if a link between a T_1 and a T_2 fails, T_0s that connect to that T_1 still have $m \times n - 1$ paths to T_0s of other clusters.

Link failures between T_0s and T_1s

Dismissing the option of valleys, if we only consider link failures between T_0s and T_1s, we only need m failures to disconnect two T_0s of different clusters, given the symmetry of the network. Since the first T_1s of each cluster are exclusively connected to the first spine plane, if a T_0 has a link failure to the first T_1, then it cannot connect to T_0s of other clusters using the first T_1s of any cluster. This can raise the question if it would be preferable to have a T_1 connecting to more than a single spine plane. However, in our study, we do not consider this option and, therefore, the pattern of failures is exactly the same as between two T_0s of the same cluster except for the fact that, now, there must be a failure in a T_1 of each position, either in the cluster of the source or the destination.

If we assume that there can only be link failures between T_0s and T_1s and, consequently, links between T_1s and T_2s are perfect, then, given the symmetry of the network, the probability defined in equation 4.3 still stands. That is, the probability of two T_0s being able to connect with each other, whether or not they are in the same cluster, in the presence of T_0-T_1 link failures is given by equation4.3.

Since the pattern of failures that determines the probability of connectivity between a pair of T_0s in different clusters is the same as between a pair of T_0s in the same cluster, and we have already shown how valleys can increase the resilience to failures for this pattern of link failures, we will now study link failures between T_1s and T_2s.

Link failures between T_1s and T_2s

Now, if we assume that only links between T_1s and T_2s can fail, there need to be $m \times n$ link failures to disconnect two T_0s of different clusters.

As a result, since m, specific, link failures between T_0s and T_1s are able to disconnect a pair of T_0s but, in the case of links between T_1s and T_2s, there need to be $m \times n$ link failures, we can see that T_0-T_1 link failures have a bigger impact in the network, in terms of connectivity, than failures between T_1s and

T_2s. Thus, this is why we are only considering valleys between T_0s and T_1s but not between T_1s and T_2s. The BGP implementation allows this, as explained in the previous section.

We shall now try to determine the probability of two T_0s being able to connect with each other in the presence of T_1-T_2 link failures. We start by analyzing networks without parallel links between pairs of T_1s-T_2s

If we assume there can only be link failures between T_1s and T_2s and links between T_0s and T_1s are perfect, we can determine that, with y being the probability of a T_1-T_2 link failure, the probability of being unable to connect 2 T_0s of different clusters via a pair of T_1s in the same relative position of the cluster, considering there are n spine sets, is

$$P(\text{no connectivity via a specific pair of } T_1\text{s}) = (1 - (1 - y)^2)^n. \tag{4.5}$$

Thus, the probability of a pair of T_0s not being able to establish a connection through any T_1 is given by

$$P(\text{no connectivity}) = ((1 - (1 - y)^2)^n)^m = (1 - (1 - y)^2)^{m \times n}. \tag{4.6}$$

Finally, we can determine that the probability that a pair of T_0s can connect with each other is

$$P(\text{connectivity}) = 1 - P(\text{no connectivity}) = 1 - (1 - (1 - y)^2)^{m \times n}. \tag{4.7}$$

This probability is dependent not only on m, the number of T_1s per cluster, but also on n, the number of spine sets.

Once again, we want to analyze the impact of T_1-T_2 link failures in the connectivity between ToRs. In order to do this, we determined the maximum probability of link failures that still maintain 99.999% of connectivity between T_0s, using different values of m and n. Due to the fact that, in the topologies generated in 3, the maximum of T_1s per cluster is 48, as well as the maximum number of spine sets, we evaluated equation 4.7 with values of m and n up to 48. These results can be seen in figure 4.7.

From the figure, we can verify that the maximum probability of a T_1-T_2 link failure increases with the number of T_1s inside a cluster and with the number of spine sets. However, the difference between probabilities starts to decrease as m, or n, increases.

For example, with $m = 8$, for $n = 32$ and $n = 40$ we have a maximum probability of link failure of, approximately, 79% and 81%, respectively. This is a small difference when compared with the probabilities for $n = 8$ and $n = 16$, which are 59% and 70%. We can also do a similar analysis for higher values of m. For example, with $n = 8$, the maximum probabilities of link failure for $m = 32$ and $m = 48$ are 79% and 82%, respectively, which are similar values when compared to the difference between the probabilities of $m = 8$ and $m = 16$, 59% and 70%.

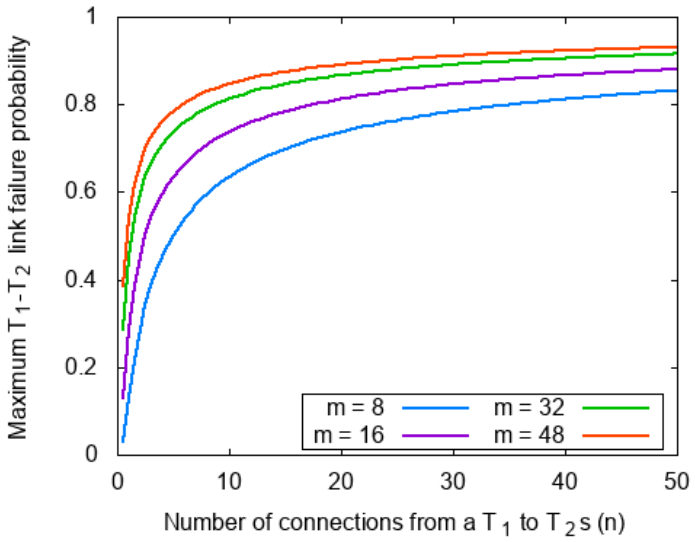


Figure 4.7: Maximum probabilities of a T_1-T_2 link failure that a network supports, for 99.999% connectivity between ToRs, and for different values of T_1s per cluster, m, and T_2s per spine plane, n.

We can also conclude that, even for small values of m and n, these probabilities of link failures are higher than the ones achieved with failures between T_0s and T_1s, shown previously in figure 4.5. This is due do the fact that, for the minimum number of spine sets, $n = 1$, the expression is exactly the same as 4.3. Therefore, as we increase the number of spine sets, the maximum probability of T_1-T_2 link failures increases as well and it will always be higher than the failure probabilities of T_0-T_1 links.

For example, with $m = 8$, and $n = 1$ the maximum probability of a link failure between a T_1 and a T_2 that maintains 99.999% of connectivity between T_0s is 12% but, for $n = 2$, this value increases to 28%. This validates the conclusion that links between T_0s and T_1s are critical in the network and why valleys are used to increase the resilience to failures in this set of links.

We also present, in figure 4.8, the different maximum probabilities of a T_1-T_2 link failure that support 99.999%, 99.99% and 99.9% of connectivity between T_0s, for $m = 8$ and different values of n.

Use of parallel links

As previously mentioned, it is cheaper to use parallel links between pairs of T_1-T_2s in order to use fewer T_2s. In this case, even if the number of connections that a T_1 has to T_2s is the same, the probability of connectivity differs when we use parallel links.

In the case of parallel links, a T_0 can connect to a T_0 of another cluster through two specific T_1s and a specific T_2 **as long as there is a single active connection** from each T_1 to the T_2. Therefore, if there are b parallel links between a T_1 and a T_2, there are n spine sets and x is the probability of a T_1-T_2 link failure, then the probability of two T_0s not being able to connect via a specific pair of T_1s is given by

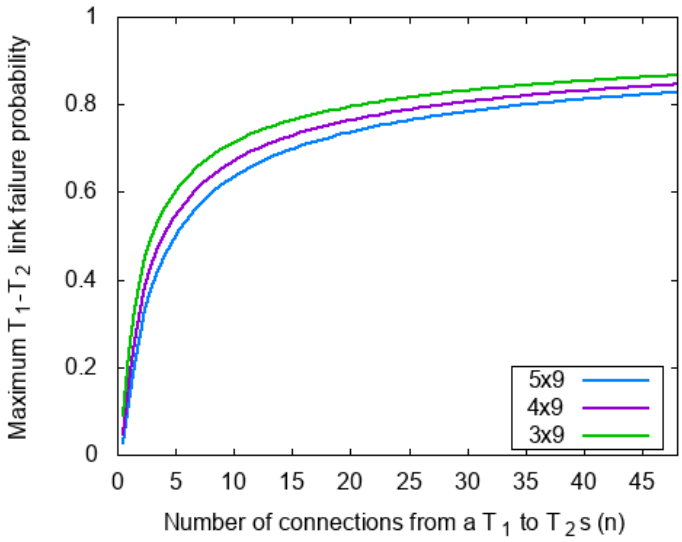


Figure 4.8: Maximum probabilities of a T_1-T_2 link failure that a network supports, for 99.999%, 99.99% and 99.9% of connectivity between T_0s, and for 8 T_1s per cluster, $m = 8$, and different values of T_2s per spine plane, n.

$$P(\text{no connectivity through a specific pair of } T_1\text{s}) = (1 - (1 - x^b)^2)^n. \tag{4.8}$$

Consequently, the probability of connectivity between a pair of T_0s, when using parallel links, is

$$P(\text{connectivity}) = 1 - (1 - (1 - x^b)^2)^{n \times m}. \tag{4.9}$$

Given both equations 4.7 and 4.9, it is possible to conclude that there is a higher resilience to link failures when using parallel links. We can verify this by comparing both expressions for the same number of T_1s per cluster and total connections from a T_1 to T_2s. This can be seen in figure 4.9, in which we represent the maximum probability of a T_1-T_2 link failure that still supports 99.999% of connectivity between T_0s, for networks with 8 T_1s per cluster. We include in the figure three networks, one that does not use parallel links and another two networks with two and three parallel links, respectively, between T_1s and T_2s.

However, it should be noted that we are only analyzing **link failures**. If we were considering router failures, for the same number of connections, a T_2 router failure would bring down more paths when using parallel links than when not using. Additionally, when routers need updates of their firmware, they need to be temporarily taken out of production and, consequently, updating a T_2 would also have a bigger impact in the network when using parallel links.

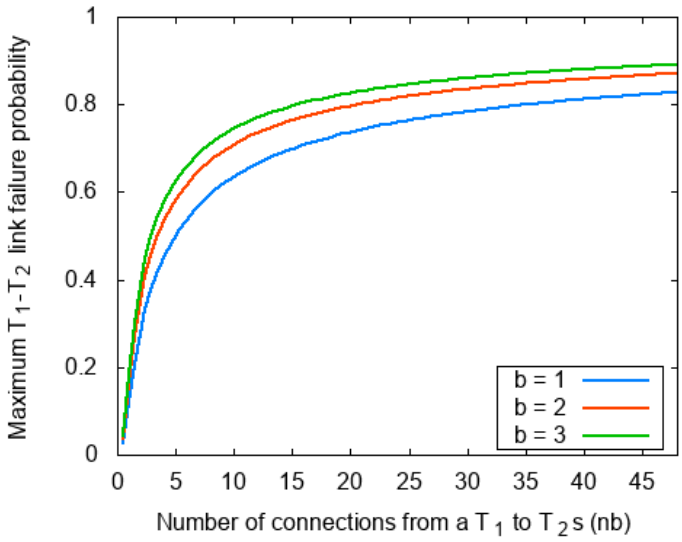


Figure 4.9: Maximum probabilities of a T_1-T_2 link failure that a network supports, for 99.999% connectivity between ToRs, and for different values of T_2s per spine plane, n, and number of parallel links from a T_1 to a T_2, b.

Relationship between the resilience to T_1-T_2 link failures and the cost

By observing figure 4.8, it is possible to conclude that the maximum probability of a T_1-T_2 link failure increases more rapidly for a low number of spine sets. When looking at the curve corresponding to $m = 8$, for example, we can see that from 1 to 5 spine sets, that is, from 0% to 49%, approximately, maximum probabilities of T_1-T_2 link failures, adding a spine set has a bigger impact on the maximum probability than for higher values of spine sets. For example, for 99.999% of connectivity between T_0s, and $m = 8$, going from 2 spine sets to 4 can increase the maximum probability of a T_1-T_2 link failure from 28% to 45%, a growth of 61%, as opposed to the increase from 49% to 63%, which corresponds to a growth of 29%, when we increase the 5 spine sets to 10.

Thus, as we increase the probabilities of link failures, it will be more and more expensive. We can see this in figure 4.10, which was obtained through a similar process used in figure 4.6. We modified algorithm 3.1 to also provide the maximum probability of a T_1-T_2 link failure that each network supports for 99.999%, 99.99%, and 99.9% of connectivity between T_0s. Then, we obtained the cheapest networks of 10 Gbps links that provide 99.999% of connectivity, for different values of maximum probabilities of T_1-T_2 link failures.

Comparison of the cost of networks with different availabilities

Finally, we can also compare the cost of supporting different levels of availability between servers, that is, in our case, the probability of a T_0 being able to connect with another T_0. Therefore, by using algorithm

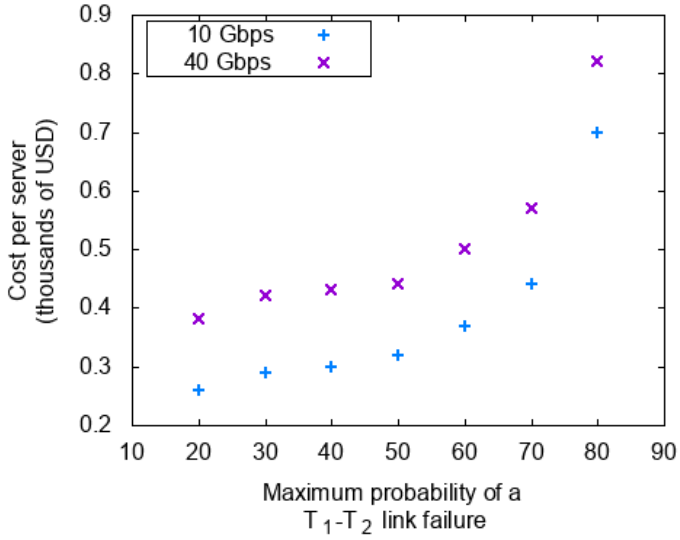


Figure 4.10: Cost per sever of networks of 10 Gbps links with different maximum probabilities of T_1-T_2 link failures that a network supports, for 99.999% of connectivity between servers.

3.1, and its further modifications, we are able to obtain the cheapest networks that support, for example, at least, 10% of the T_0-T_1 and 60% of the T_1-T_2 links down while achieving an availability of 5x9.

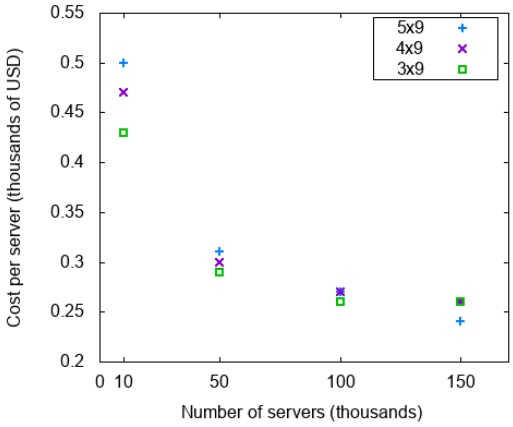
In figure 4.11, we analyze the price per server of networks with 3 levels of availability, 3x9, 4x9, and 5x9. Additionally, we compare two different sets of maximum probabilities of link failures, one with 10% of T_0-T_1 and 60% of T_1-T_2 link failures, depicted in figure 4.11(a), and another with 30% of T_0-T_1 and 80% of T_1-T_2 link failures, represented in figure 4.11(b).

It is possible to determine, by these figures, that the difference between the cost per server of the different availabilities diminishes with the increase of the number of servers.

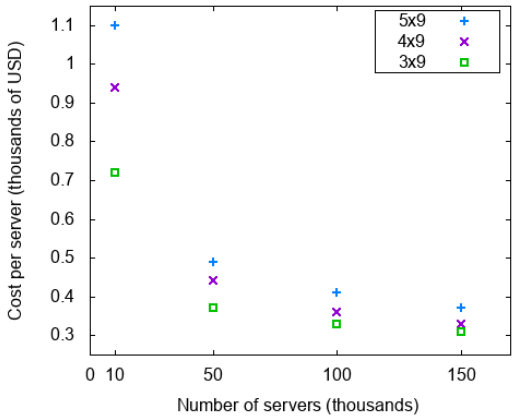
4.3 Connectivity to the Internet

Data center companies also want to maximize the availability of external access to the data center. In our topology, we consider, as shown previously in figure 2.7, that all T_2s are connected to a "super node" with perfect links, i.e. that does not suffer failures, which connects the data center to the Internet. Therefore, we only take into account paths from T_0s to T_2s to analyze the connection to the Internet.

With this topology, it is possible to determine that there are $m \times n$ shortest paths, of **length 2**, connecting a T_0 to a T_2 and, consequently, the Internet. However, only m of these paths are **disjoint**.



(a) Network that supports 10% of T_0-T_1 link failures and 60% of T_1-T_2 link failures.



(b) Network that supports 30% of T_0-T_1 link failures and 80% of T_1-T_2 link failures.

Figure 4.11: Cost per server of networks of 10 Gbps linkswith different resilience to failures, for different values of availabilities.

Link failures between T_0s and T_1s

In order to analyze the connectivity of T_0s to the Internet, in the presence of link failures, we first assume that T_1-T_2 links are perfect and evaluate the probability of connectivity in the presence of T_0-T_1 link failures.

A T_0 cannot connect to the Internet, in the presence of T_0-T_1 link failures, if all m links it has to T_1s fail. Thus, if x the probability of a T_0-T_1 link failing, then the probability that a T_0 can connect to the Internet is given by

$$P(\text{connectivity to the Internet}) = 1 - x^m. \tag{4.10}$$

Similarly to what was previously done, we equaled equation 4.10 to the different values of connectivity, that is 5x9, 4x9, and 3x9, in order to determine the maximum probability of a link failure between a T_0 and a T_1 that guarantees the different values of connectivity between a T_0 and the Internet. The results are presented in figure 4.12.

As can be seen, these values are higher than the ones achieved in previous sections. For example, for $m = 8$ and a connectivity of 5x9, the maximum probability of failure that supported connectivity between a pair of T_0s is, approximately, 12 % but, in this case, it is 23%.

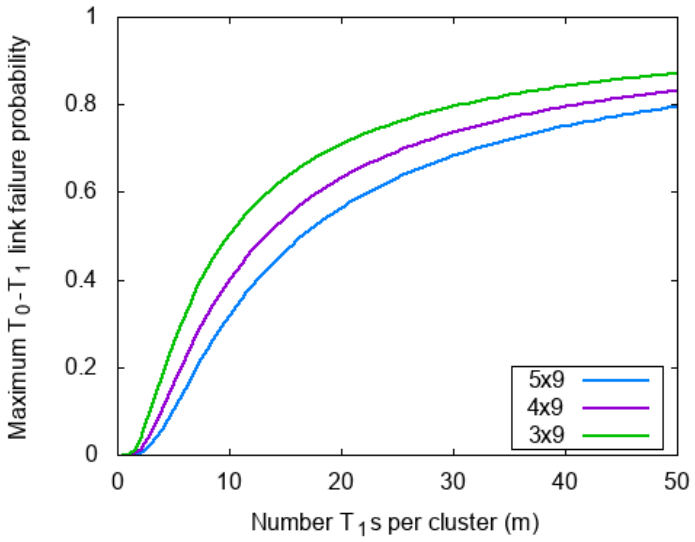


Figure 4.12: Maximum probabilities of a T_0-T_1 link failure that a network supports, for 99.999%, 99.99% and 99.9% of connectivity between T_0s and the Internet, and for different values of T_1s per cluster, m.

Link failures between T_1s and T_2s

Now, similarly to what was done in the previous section, we assume that T_0-T_1 links are perfect and that only T_1-T_2 links suffer failures. In this scenario, all $m \times n$ T_1-T_2 links must fail to disconnect a T_0 from the Internet. If y is the probability of a T_1-T_2 link failing, then the probability of a T_0 connecting to the Internet is

$$P(\text{connectivity to the Internet}) = 1 - y^{m \times n}. \tag{4.11}$$

Using the same values as in section 4.2, we calculate the highest probability of a T_1-T_2 link failure that still guarantees that a T_0 can connect to the Internet with a probability of 99.999%, depicted in figure 4.13.

As with link failures between T_0s and T_1s, the maximum probability of a T_1-T_2 link failure that still respects 99.999% connectivity between a ToR and the Internet is higher than between a pair of T_0s. For example, for $m = 8$ and $n = 2$, the maximum probability of failure for connectivity between T_0s is, approximately, 28% and, in this case, it is 48%.

We also present, in figure 4.14, the comparison of maximum probabilities of a T_1-T_2 link failure that support 99.999%, 99.99% and 99.9% of connectivity between T_0s and the Internet, for $m = 8$ and different values of n.

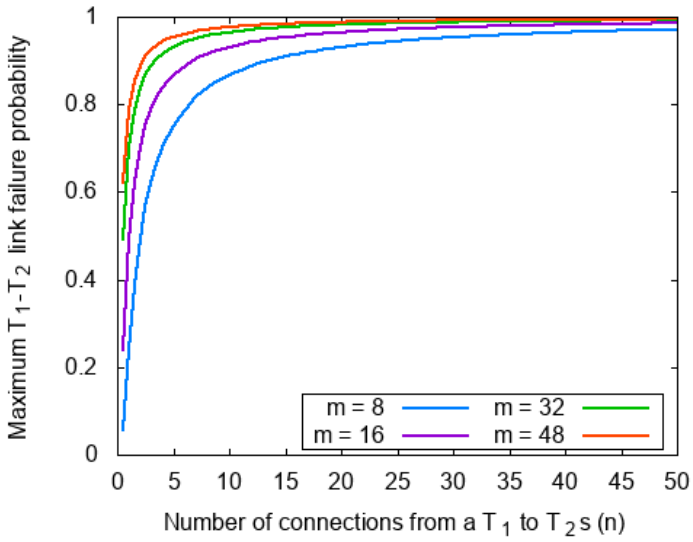


Figure 4.13: Maximum probability of a T_1-T_2 link failure that a network supports, for 99.999% connectivity between a ToR and the Internet, and for different values of T_1s per cluster, m, and T_2s per spine plane, n.

Use of valleys and parallel links

We will not be studying the use of valleys in this case because it would not achieve better results. This can be explained by the following: if we are only studying failures between T_0s and T_1s, then every T_1 has a connection to T_2s and, consequently, a T_0 only needs to have an active connection to a T_1. However, if a T_0 has a connection to a T_1, then it will not need to use valleys because that T_1 already makes the connection to the T_2s.

In this case, using parallel links will achieve the same results as not using due to the fact that we consider that links from T_2s to the "super node" are perfect. If we were considering that there were faulty parallel links between T_2s and the "super node" then, once again, we would see an increased resilience to link failures when using parallel links.

Comparison of the cost of networks with different availabilities

Now, we can compare the cost of supporting different levels of availability between the data center and the Internet. We modified, once again, algorithm 3.1 so that it also outputs the maximum probabilities of T_0-T_1 and T_1-T_2 link failures that each network supports in order to provide 99.999%, 99.99% and 99.9% of availability.

In figure 4.15, we analyze the price per server of networks with 3 levels of availability, 3x9, 4x9, and 5x9, for maximum probabilities of 30% of T_0-T_1 link failures and 80% of T_1-T_2 link failures.

When comparing figures 4.11(b) and 4.15, it can be observed that networks in figure 4.15 have, with

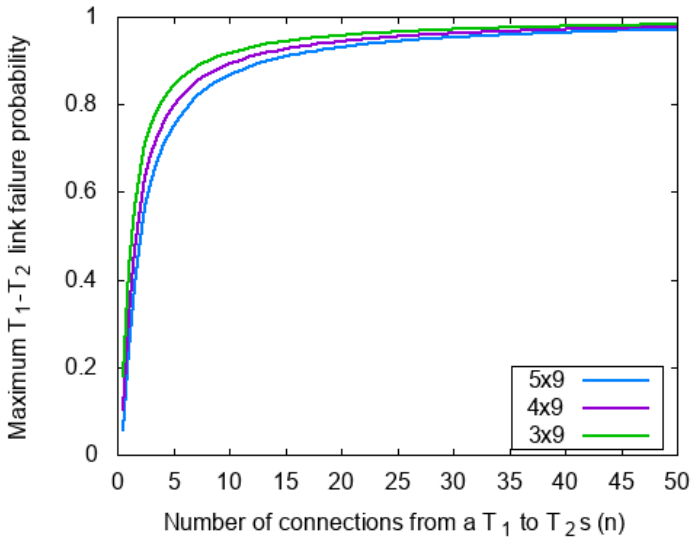


Figure 4.14: Maximum probabilities of a T_1-T_2 link failure that a network supports, for 99.999%, 99.99% and 99.9% of connectivity between T_0s and the Internet, and for 8 T_1s per spine plane, $m = 8$, and different values of T_2s per spine plane, n.

the exception of networks with 10 000 servers, a lower cost per server. This is due to the fact that, for the same number of T_1s per cluster and connections from a T_1 to T_2s, the value of the maximum probability of both T_0-T_1 and T_1-T_2 link failures is higher in the case of connecting the data center to the Internet than connecting pairs of T_0s.

4.4 Path diversity

Folded Clos topologies have many redundant paths between different nodes. In data centers, we want to take advantage of this characteristic and use different paths to reach the same destination. This way, we can distribute the traffic by many paths and, consequently, enable load-balancing in the network.

In BGP, there is an extension called multi-path that allows the installation of multiple routes **with equal cost** to the same prefix in the FIB, and, afterwards, makes use of ECMP to use, in simultaneous, these routes. In this case, the cost is defined by the length of the *AS-PATH*. Consequently, only the best paths with the same length are stored. Even if, for example, there is a single shortest path available and other paths with a higher *AS-PATH* length, the shortest path is the only one stored and used.

Additionally, ECMP allows us to configure how many redundant paths to the same destination are stored in the FIB. We want to make use of as many paths as possible but, due to the cost or the maximum size of the FIB, we may want to limit the number of paths stored.

However, when we limit the number of paths stored in the FIB, sometimes, given the fact that the paths are stored in chronological order, i.e., in the order that the routes are learned, most nodes end up

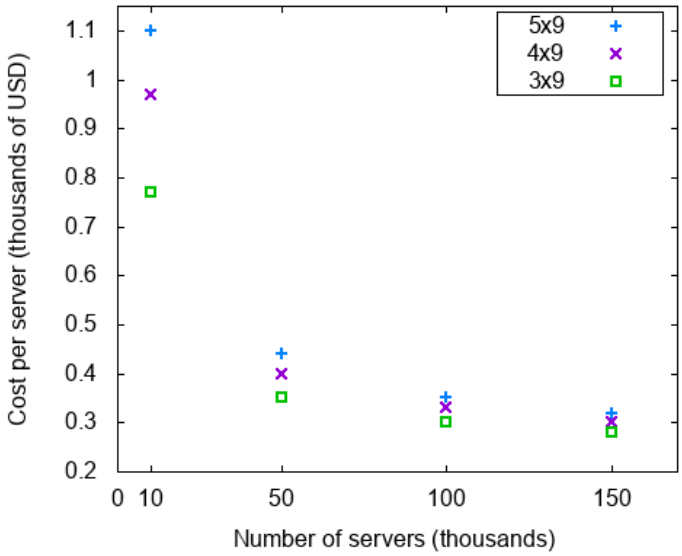


Figure 4.15: Cost per server of networks of 10 Gbps links that support 99.999% of connectivity, for maximum probabilities of 30% T_0-T_1 link failures and 80% of T_1-T_2 link failures.

saving the same routes. If, for example, in data centers, we want T_0s to distribute the traffic by only half of the available T_1s, each T_0 will save the first routes learned by the T_1s. But, when advertising a route, each T_1 will broadcast it to all T_0s at the same time. As a consequence, T_0s may save the routes learned from the same T_1s, which leads to some T_1s barely receiving traffic and others that receive most of the traffic from the T_0s. This causes a very unequal load-balancing in the network.

Nevertheless, in this work we assume perfect load-balancing. For each volume of traffic that a node receives for a given prefix, it evenly distributes it by all the paths stored in the FIB. It should be noted that, in reality, traffic flows are indivisible.

ECMP value in the presence of link failures

In the presence of link failures, some paths will no longer be available. In folded Clos topologies, given the fact that every connection originated in T_0s is made through T_1s, the number of redundant paths, i.e., the ECMP value, will be determined by how many T_1s can be used to reach the destination. As a consequence, we only study link failures between T_0s and T_1s, since it only takes a T_0-T_1 failure to decrease the number of redundant paths from that T_0 but, in the case of T_1-T_2 link failures, it takes all of the links between a T_1 and T_2s.

When valleys are not enabled, if we take the expression in equation 4.1 and multiply it by the number of T_1s in a single cluster, we end up with the value of ECMP, that is, number of paths stored in the FIB for the same prefix, for a probability x of a T_0-T_1 link failure:

$$(1-x)^2 \times m. \tag{4.12}$$

In figure 4.16, it is depicted the ECMP value in terms of the probability of a T_0-T_1 link failure, for different values of T_1s per cluster, m. We can observe in this figure how the ECMP value decreases as the probability of a T_0-T_1 link failure increases.

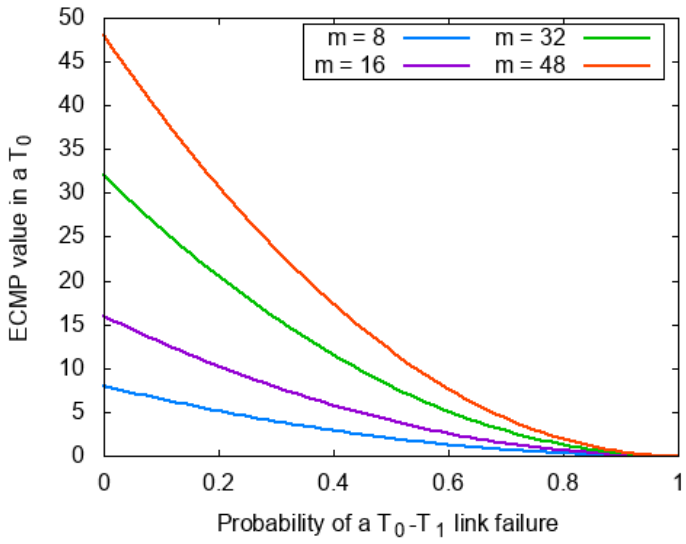


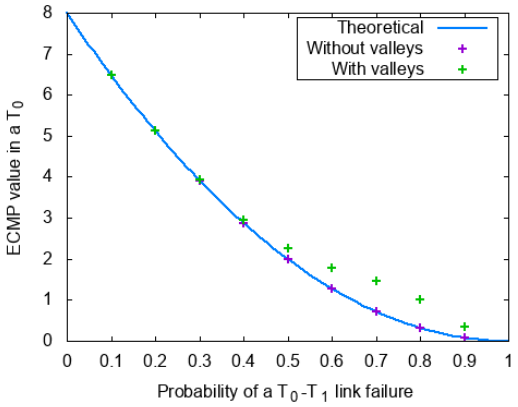
Figure 4.16: Value of ECMP of T_0s for different values of T_1s per cluster, m, and for different values of T_0-T_1 link failure probabilities.

Use of valleys

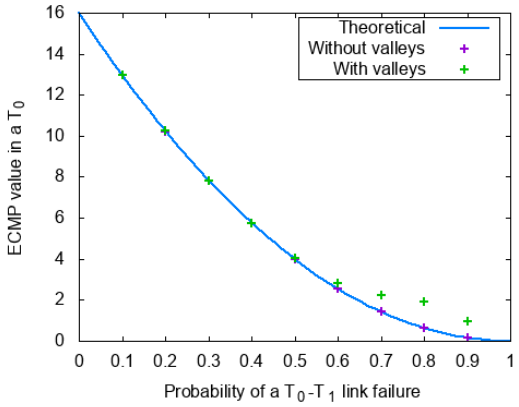
Nonetheless, similarly to what is done to increase connectivity, we can use valleys to augment the ECMP value in the presence of link failures. In figures 4.17(a) and 4.17(b), we can see the theoretical value for networks with 8 T_1s per cluster, $m = 8$, and 16 T_1s per cluster, $m = 16$, respectively, as depicted in figure 4.16, and the simulated values with and without valleys. The simulated values were obtained by running a slight modification of algorithm 4.1 so that it also determines the ECMP value.

The graphs in figure 4.17 allow us to conclude that, given the fact that valleys only start being used for high probabilities of a T_0-T_1 link failure, the ECMP value obtained with and without valleys is the same up until a 50% probability of link failure for networks with 8 T_1s per cluster, and 60% for networks with 16 T_1s per cluster.

However, as the probabilities of link failure increase, the use of valleys allows the use of more paths. For example, for $m = 16$, a probability of link failure of 80% results in an average of the ECMP value without valleys of 0.64, and, by using valleys, this value increases to 1.94.



(a) Network with 8 T_1s per cluster, $m = 8$.



(b) Network with 16 T_1s per cluster, $m = 16$.

Figure 4.17: Comparison between the use of valleys in the value of ECMP of T_0s, for different T_0-T_1 link failure probabilities.

Additionally, we can see that the ECMP value using valleys surpasses the value without valleys for lower link failure probabilities in networks with 8 T_1s per cluster, figure 4.17(a), than in networks with 16 T_1s per cluster, figure 4.17(b). From this we can conclude that valleys are more relevant in architectures with fewer T_1s per cluster. This is due to these architectures supporting less failures in T_0-T_1 links and, consequently, valleys start being used at lower probabilities as well.

Therefore, valleys not only allow more pairs of T_0s to connect with each other in the presence of link failures, but also allow a higher number of paths to be used.

However, since valleys only start being used at high probabilities of failures, one must conclude that, if BGP allowed, it would be preferable to use, in terms of connectivity, non-valley and valley paths simultaneously, that is, 2-hop and 4-hop paths. In this case, an approximation of the value that ECMP would have can be deduced from the fact that two T_0s would be able to connect through a T_1, or a pair of T_1s in the same relative position of both clusters, if both T_0s were in different clusters, if there was a **single active link from the source to a T_1 or from the destination to a T_1**. Therefore, if x is the probability of a T_0-T_1 link failure and there are m T_1s per cluster, then the ECMP value is given by

$$(1 - x) \times m. \tag{4.13}$$

This expression is represented, for networks with 8 T_1s per cluster, $m = 8$, in figure 4.18, along with the previous values obtained in figure 4.17(a), in order to compare the ECMP value. We can conclude that using valleys in simultaneous would be advantageous in terms of the number of paths available.

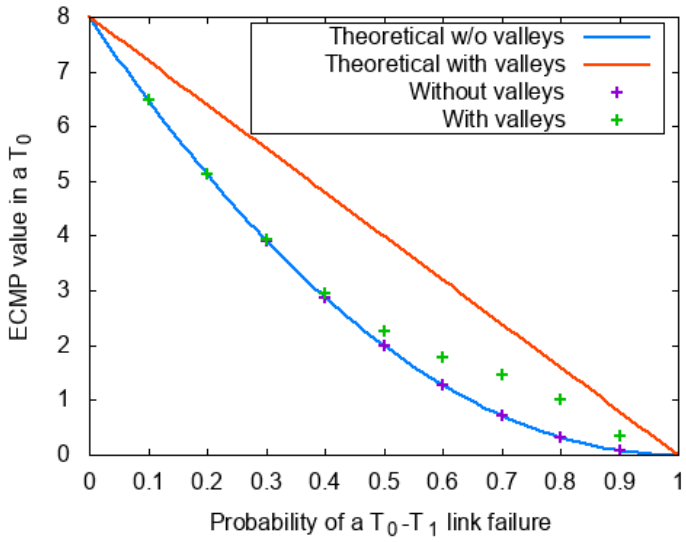


Figure 4.18: Values of ECMP with and without the use of valleys, and comparison with the simultaneous use of both types of paths, for different values of T_0-T_1 link failure probabilites and a network with 8 T_1s per cluster, $m = 8$.

4.5 Routing State

We now present the number of routes installed in the FIB, and in the RIB, for a complete analysis, without link failures and in terms of the different parameters of the network. We consider the following parameters: k T_0s per cluster, m T_1s per cluster, n spine sets and, finally, p clusters.

In this study, we consider that only T_0s are advertising a single aggregated route to its servers. Therefore, we consider that each router only stores routes to T_0s, which may not happen in every network, and we are also not considering the number of routes from the Internet that a T_2 stores.

RIB values

In the case of the RIB, it stores every path learned to a prefix, whether or not the router in question elected this route as the best one, and independently of the cost, which is, in this case, determined by the *AS-PATH* length.

Each T_0 saves a route to all of the remaining T_0s in the data center and, since it is connected to m T_1s that provide a path to the T_0s, it has m routes per T_0 in the data center.

In the case of T_1s, we need to differentiate the values when we do not use valleys and when we do and, additionally, when we use parallel links or not. In order to simplify, we consider that the parallel links form a single link, that is, even though there may be more than one physical link, it is registered as a single logic link.

Without valleys, each T_1 has one route to each T_0 of its cluster, that is, k routes, and, since it is connected to n T_2s, it has n routes per T_0 of other clusters. In the case of valleys, however, the T_1 has, in addition to the routes that do not use valleys, $k-1$ routes per T_0 of its cluster, since each T_0 of its cluster will advertise a route to other T_0s, and k routes per T_0 of other clusters.

Finally, in the case of T_2s, they only connect to a T_1 of each cluster and, consequently, have a single route per T_0 in the data center. However, as previously mentioned, T_2s also store routes from the Internet.

It should be noted that, since we are analyzing the number of routes in the RIB when there are no link failures, T_1s only advertise paths without valleys and, consequently, T_0s and T_2s will not learn about these paths until T_1s start using them. As such, the number of routes in the RIBs of T_0s and T_2s does not change with the use of valleys when there are no link failures.

The expressions that represent the number of RIB entries for each type of router can be found in table 4.1.

Table 4.1: Number of RIB entries, without failures.

T_0s	T_2s
$(k \times p - 1) \times m$	$k \times p$

Options	T_1s
No valleys	$k + k \times (p-1) \times n$
With a valley	$k + k \times (p-1) \times n + k \times (k-1) + k \times (p-1) \times k$

In order to analyze these expressions, we assume that we are dealing with a fixed number of servers, designated by N, in the data center.

For example, if we consider that T_0s are connected to l servers, we need N/l T_0s. Thus, even if we change the number of T_0s per cluster and, consequently, the number of clusters, the number of T_0s is the same and, thus, $k \times p$ is a constant.

Therefore, since each router has a number of entries directly proportional to $k \times p$, this value will not have an impact in the number of entries, for equal values of l. We can also conclude that, for bigger values of l, fewer T_0s are required and, consequently, the number of entries in the RIB diminishes in each router.

However, it should be noted that we are assuming an aggregation of prefixes in the T_0s and, if this is not the case, and every prefix that the T_0 connects to is advertised, then the value of l will not change the number of entries. Thus, we can see the advantages that prefix aggregations have.

Additionally, we can conclude that aggregation of prefixes at higher tiers could also lead to improvements in the number of entries installed in the RIB. For example, if each T_1 could advertise an aggregation of the prefixes advertised by the T_0s of its cluster, then the term $k \times p$, in all the tables of the

routers, would be substituted by only p. And, if each T_2 was able to aggregate the prefixes advertised by each T_1, then we could even substitute $k \times p$, in T_0s and T_2s, by only one route. However, in the case of failures, routers may not be able to aggregate and still need to advertise "individual" prefixes.

We can also conclude that the number of entries in T_0s is directly proportional to the number of T_1s per cluster and the number of entries in T_1s is directly proportional to the number of spine sets and, in the case of valleys, also to the number of T_0s.

FIB values

In the FIB, only the elected routes are installed. Therefore, if multi-path is not used, only a route per prefix is installed. However, in this work, we have to take multi-path into account and, consequently, routes with the same *AS-PATH* length will be stored. There is also an option, if needed, of limiting the number of equal-cost paths stored, which we will designate by ECMP in the expressions.

In the case of T_0s, they have m routes, that is, the number of T_1s per cluster, per T_0 in the data center. The number of routes installed is limited by this number or, alternatively, by the ECMP value set in the BGP implementation.

In T_1s, if there are no failures in the network, their FIBs will only store the paths that are not valleys. As a consequence, they save a route per T_0 of the cluster and the minimum between n, the number of spine sets, and the ECMP value defined, per each T_0 of other clusters. However, when we use valleys, if the T_1 is unable to connect directly to a T_0 of its cluster then, instead of storing a single route to it, it stores either up to $k-1$ routes that the other T_0s of the cluster advertise or the limit imposed by ECMP. And, if it does not have a shortest path through any T_2 to a T_0 of another cluster, then it will store, instead of the n shortest routes, up to n routes adversited by T_2s that provide paths with a valley on the cluster destination, in addition to up to k routes advertised by the T_0s of its cluster that provide paths with a valley on the cluster source, or the limit imposed by ECMP. Although we only present, in table 4.2, the values of the number of entries in the FIB when there are no failures, it is possible to understand how the table can grow when valleys are allowed and there are link failures.

Finally, T_2s store the same number of routes as in the RIB since they only have a route per T_0. Additionally, they store an elevated number of routes from the Internet.

The expressions that represent the number of FIB routes, without failures, for each type of router can be seen in table 4.2.

Table 4.2: Number of FIB entries, without failures.

T_0s	T_1s	T_2s
$(k \times p - 1) \times \min(m, ECMP)$	$k + (k \times p - 1) \times \min(n, ECMP)$	$k \times p$

By analyzing table 4.2, we end up with the same conclusions that were derived from table 4.1. Since

each router has a number of entries in the FIB directly proportional to $k \times (p-1)$, if only T_0s advertise prefixes, then clusters with more servers per T_0 and fewer T_0s allow the installation of less routes. Additionally, it would also be advantageous to have T_1s announce an aggregated prefix of the prefixes that each T_0 of its cluster advertises, in order to substitute the expression $k \times p$ by simply p. And, as previously concluded, having T_2s advertise an aggregated prefix of the advertised prefixes by T_1s would even substitute the expression $k \times p$ by a single route.

When a T_1 connects a T_0 of its cluster to a T_0 of a different cluster, and it needs to use a valley, it may have the option of using a valley inside its cluster or in the cluster of the destination. However, the probability of the T_1 choosing a valley in its cluster is not uniform and it is, instead, related to the storage of paths in the FIB. Since it will have, at most $k-1$ paths, installed in the FIB, that use a valley in the source cluster and n in the destination cluster, if x is the number of T_0s, in the cluster of the source, that the T_1 can use to reach the destination, and y is the number of T_2s that it can also use, then the probability of the T_1 choosing the cluster origin to accommodate the valley is given by

$$P(\text{valley in the source cluster}) = \frac{x}{x+y}.$$

This means that, if a T_1 only has a single failure to the T_0 origin, then

$$P(\text{valley in the source cluster}) = \frac{k-1}{k-1+n}. \tag{4.14}$$

4.6 Conclusions

In this chapter, we were able to determine mathematical expressions that provide the probability of connectivity between pairs of servers, and between servers and the Internet, in terms of link failures. By doing this, it was possible to determine approximate values of availability, usually defined in the SLAs of data centers. We also compared the cost of networks for different values of availability between servers, and between the data center and the Internet.

We concluded that links between T_0s and T_1s are critical in the network due to the number of disjoint paths connecting a T_0 to other T_0s, or the Internet, being limited by the number of T_1s that the T_0 connects to. Therefore, the link failures that have a bigger impact in the network are T_0-T_1 link failures.

We saw how using non-shortest paths, which we designated by valleys, can improve end-to-end connectivity. In the presence of link failures, valleys allow pairs of servers to connect with each other that otherwise could not. Additionally, valleys allow an increase in the number of redundant paths that a T_0 has, but only for a high percentage of link failures. We presented how, instead of an exclusive use of shortest or non-shortest paths, the simultaneous use of both paths can increase the number of redundant paths that a T_0 has stored in the FIB. However, the need to store more routes when using

valleys can have an impact in the cost of routers due to the increase of the memory needed.

We were able to determine mathematical expressions that, by assuming that each T_0 advertises a prefix, provide the number of routes in both the RIB and the FIB as a function of the routers in the topology. We provided an insight into how aggregation of prefixes can help reduce the memory size needed.

Chapter 5

Congestion Analysis

Data centers run different types of applications, which will determine the traffic pattern of each network. In this work, we want to examine how link failures affect the congestion of links in the different folded Clos topologies by studying the, assumed by us, traffic patterns of 3 well-known application models.

5.1 Traffic patterns

The traffic patterns that we are analyzing correspond to the following applications:

1. In **HPC**, the application is distributed by many servers that process the data and communicate with each other to obtain results. Therefore, the traffic occurs inside the data center, that is, between servers. Distributed applications can, however, have different patterns of communication between servers [35]: one to all, all to one, all to all and one to one. Additionally, applications may take advantage of the location of the servers in the data center, that is, spatial locality. Consequently, we assume flows are preferably intra-rack, followed by intra-cluster and, if more servers are needed, by inter-cluster. In this study, we analyze all to all communications between servers of the same cluster and between all the servers of the data center.

2. **Search engines**, which we are treating as a concrete example of an HPC application with additional communication from and to the Internet, such as Google search [17], have specific servers dedicated to accepting requests from the Internet, distribute each request by other servers, assemble the results of the computations and reply to the user. In this case, we assume most of the traffic occurs inside the data center and the rest between the data center and the Internet. As an HPC application, search engines make use of spatial locality and, consequently, we only consider traffic intra-cluster between internal servers. Thus, a single ToR per cluster is elected to receive and distribute the requests by the remaining servers of the cluster. Additionally, we are assuming that servers dedicated to receiving requests from the Internet have 80% of their share of the traffic between themselves and the other servers and the remaining 20% between themselves and the Internet. The remaining servers reply solely to the server that distributes requests with the same amount of traffic received by him, that is, 80% of the total volume divided by the number of remaining servers.

3. **Streaming services** are assumed to mostly having servers dedicated to serving clients from the Internet. Servers receive requests from the Internet and start sending large amounts of data, obtained from the storage resources, to the clients. Therefore, we assume most traffic occurs between the servers and the Internet. In our study, we consider that all servers are serving requests from clients, that is, are sending data to the Internet, and have occasional communications between themselves and the other servers of the cluster. Thus, for each server, we consider 80% of the share of the traffic is between the server and the Internet and the remaining 20% to servers of the same cluster.

In our work, two important aspects should be taken into account. Firstly, as with link failures, since we do not assume that a server has more than one connection to a T_0, we are only measuring traffic to and from T_0s, that is, we do not study the traffic between servers and the ToRs. Secondly, we assume that each traffic flow is perfectly distributed by the ECMP and, consequently, load-balancing is perfect, that is, each traffic flow is equally distributed by all paths at each router. In practice, however, traffic flows are indivisible.

In order to analyze the different traffic patterns, we started with the program, previously mentioned in section 4.1, which builds a graph representing the network given as input. However, algorithm 4.1, described as well in section 4.1, which discovers the existence of a path from one node to others, is not sufficient to distribute traffic since, in this case, we need to know not only if there is a path but also **every shortest path between nodes**.

Therefore, we slightly modified algorithm 4.1 into algorithm 5.1. This algorithm discovers all the shortest paths between a source node, s, and the other nodes in the network. Each time we took a node u out of the queue, we check if the path to its neighbors through him has the same distance as the shortest path found to the neighbors. Therefore, if, for a neighbor v, the distance is the same, we can register node u as one of the parents of node v, that is, node v has a path to the source through node u.

After running the algorithm, we still need to distribute the traffic. Given the fact that we only store the parents of every node, we need to use recursion to go through the path and add the flows of traffic to each link, as presented in algorithm 5.2.

Finally, we want to determine the maximum probability of link failures that each network supports. We consider that a network supports a given probability of failure if, for half, or less, the samples we take, there are no links with a sum of traffic flows higher than their capacity, that is, no link is supposed to deliver, for example, 12 Gbps of traffic if it only has a capacity of 10 Gbps. To do this, we took 100 samples, for each traffic pattern and different volumes of traffic, until we found a probability of link failure that presented more than half of the samples with overcapacitated links.

Similarly to the connectivity analysis, we divided the link failures in T_0s-T_1s and T_1s-T_2s. However, since paths from T_1s to T_2s share a T_0-T_1 link, then failures in T_0s-T_1s are the critical ones, as seen in

Algorithm 5.1: FindShortestPaths(s, valleys)

```
begin
    for each v do
        dist[v] ⟵ +∞
        visited[v] ⟵ false
        path[v] ⟵ ∅
        parents[v] ⟵ ∅
        nrPaths[v] ⟵ 0

    Q ⟵ s
    dist[s] ⟵ 0
    visited[s] ⟵ true
    while Q ≠ ∅ do
        select u of Q for which d[u] is smallest
        Q ⟵ Q − {u}
        for each adjacent node v of u do
            if visited[v] ≠ true then
                if ASNInPath(v, u) = 1 and (valleys ≠ true or IsT2(v) = true) or
                  ASNInPath(v, u) = 2 then
                    continue
                dist[v] ⟵ dist[u] + 1
                visited[v] ⟵ true
                path[v] ⟵ u
                Q ⟵ Q + {v}
            if (dist[u] + 1) = dist[v] then
                nrPaths[v] ⟵ nrPaths[v] + 1
                parents[v] ⟵ parents[v] + {u}
```

the connectivity analysis and, as a result, we applied no oversubscription in T_1s. Therefore, our analysis focuses on T_0-T_1 link failures, with the exception of the impact of using parallel links, explained in detail in section 5.3.

Traffic patterns comparison

As previously mentioned, we studied HPC patterns with all to all communication between the T_0s of a single cluster, intra-cluster, and between every pair of T_0s in the data center, inter-cluster. The comparison of the maximum probability of a T_0-T_1 link failure that a network, with 48 T_0s per cluster, 16 T_1s per cluster, and 27 clusters, supports for these two modes is represented in figure 5.1. From this figure we can conclude that the difference between the results is minimal, with a maximum of 2%, and this is due to the fact that, in both cases, the amount of traffic transversing the network is the same. Therefore, this allows us to largely reduce the computation time by only analyzing communication intra-cluster.

Algorithm 5.2: RebuildPaths(v, traffic)

begin

  if $parents[v] = \emptyset$ **then**

    return

  $distributedTraffic \longleftarrow traffic/nrParents[v]$

  for each parent p of v **do**

    $addTrafficToLink(v, p, distributedTraffic,)$

    $RebuildPaths(p, distributedTraffic)$

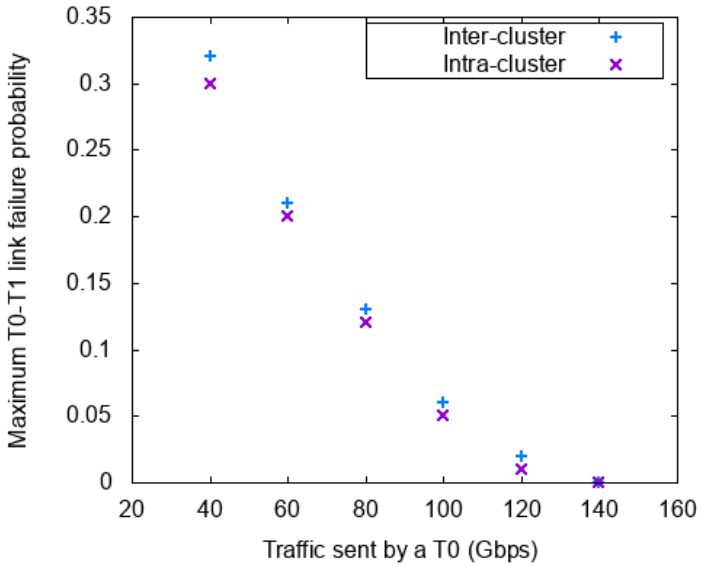
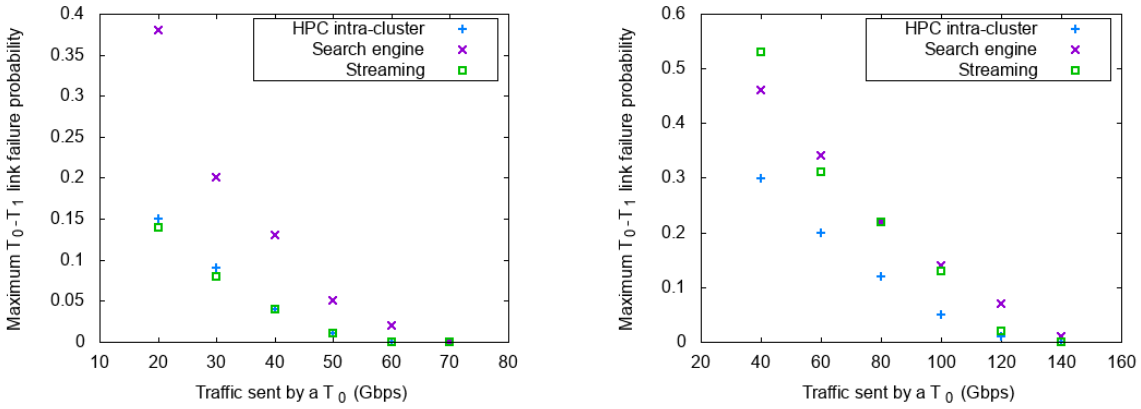


Figure 5.1: Comparison between HPC all to all communication intra-cluster and inter-cluster of the maximum probabilities of a T_0-T_1 link failure that a network supports, as a function of the traffic sent by each T_0.

We compare, in figure 5.2, the maximum probability of link failure for each traffic pattern, for two networks with 10 Gbps of capacity links,: network **A** in figure 5.2(a), and network **B** in figure 5.2(b). Both networks have the same number of servers, approximately a hundred thousand, and the same number of T_0s per cluster, but have T_0s with a different number of ports. On network A, each T_0 connects to 40 servers and 8 T_1s, and on network B, each T_0 connects to 80 servers and 16 T_1s. Consequently, since both networks share the number of T_0s per cluster, network A has double the number of clusters of network B.

In figure 5.2, we present, in the axis x, the traffic sent by every T_0 in the network, in the case of the HPC and streaming patterns. In the case of the search engine, we present the value of traffic sent by the T_0 that receives requests from users in the Internet, distributes these requests and then replies to users.

Since each network has a different number of T_1s, then the axis in figures 5.2(a) and 5.2(b) differ. However, if we divide the traffic sent by a T_0 by the number of servers in each T_0, we obtain the value per server and are able to compare the probabilities in terms of traffic per server. For example, 40 Gbps sent by a T_0 in network A corresponds to, approximately, 1 Gbps per server, and 80 Gbps sent by a T_0 in network B corresponds to the same value per server, when divided by the 80 servers connected to each T_0.



(a) Network A, with 8 T_1s per cluster.

(b) Network B, with 16 T_1s per cluster.

Figure 5.2: Comparison between the three different patterns of communication of the maximum probabilities of a T_0-T_1 link failure that two networks, A and B, support, as a function of the traffic sent by a T_0.

We can conclude that network B supports higher probabilities of link failure than network A due to the higher number of T_1s per cluster. For example, for HPC, network A only supports a probability of link failure of 4% for a traffic volume of 40 Gbps, and network B supports 12%, which is a value 3 times higher, for 80 Gbps of traffic.

We can see in both networks that, in general, the HPC traffic pattern is the one that supports less failures and the search engine the one that supports more. Additionally, the streaming traffic pattern is the pattern that suffers more variation with the number of T_1s per cluster. Although, in network A, the streaming pattern supports similar failures to HPC, in network B it supports more failures than this pattern and has similar values to the search engine pattern.

Additionally, we decided to compare the values obtained with network B to the values obtained with network C, which has exactly the same number of servers per T_0 and T_1s per cluster, but has half the number of T_0s per cluster and, consequently, double the number of clusters. To summarize, network C has 80 servers per T_0, 24 T_0s per cluster, instead of 48, 16 T_1s per cluster, 24 spine sets, since there is no oversubscription at the T_0s, and 53 clusters, instead of the 27 of network B. The results are shown in figure 5.3.

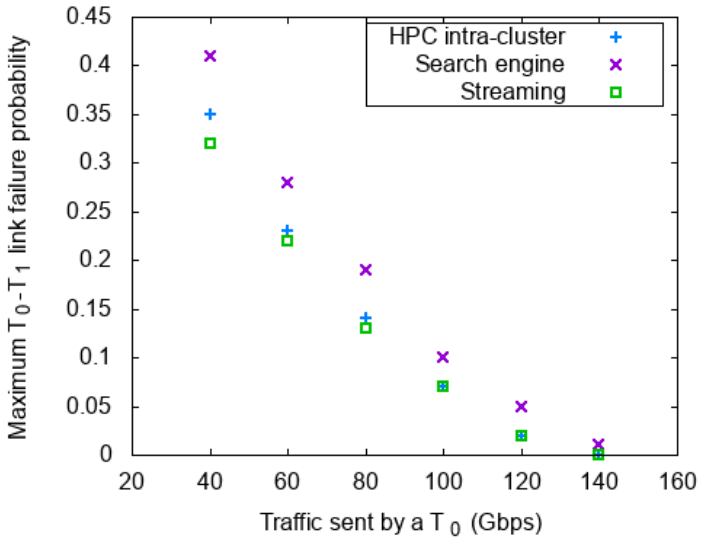


Figure 5.3: Comparison between the three different patterns of communication of the maximum probabilities of a T_0-T_1 link failure that network C, with 16 T_1s per cluster, supports, as a function of the traffic sent by each T_0.

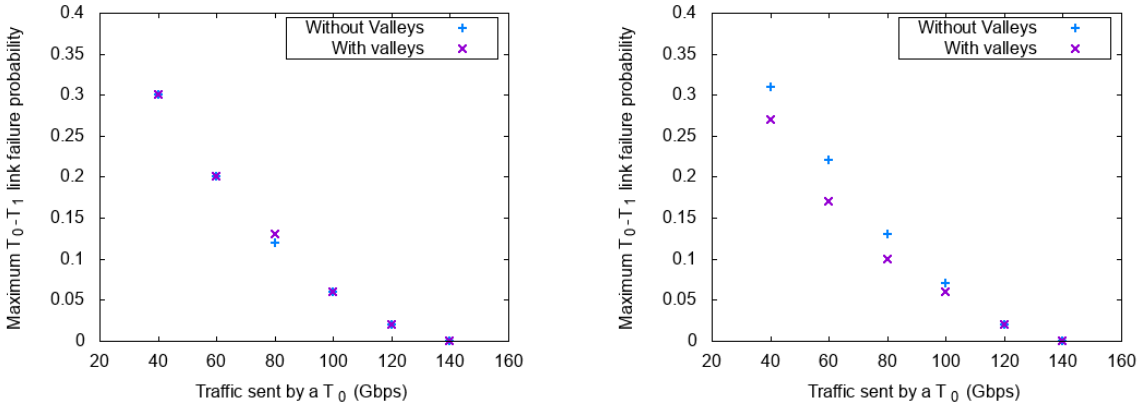
From figure 5.3, it is possible to determine that this network presents slightly better results for the HPC pattern. For example, for a volume of traffic of 60 Gbps, network B supports 20% of link failures and network C supports 23%. However, for the other two traffic patterns, the network has worse results, especially in the case of the streaming pattern. For example, for the streaming traffic pattern, while network B supports 22% of failures for a volume of traffic of 80 Gbps sent by each T_0, network C only supports 13%. As such, we can conclude that, for the same number of servers, spine sets, and T_1s per cluster, networks with more clusters and fewer T_0s per cluster achieve better results for HPC intra-cluster communication patterns and worse results for the search engine and streaming patterns.

It should also be noted that, while we are not applying oversubscription in the T_1s, in the case of the HPC intra-cluster and the streaming patterns, since the volume of traffic inter-cluster, and between the data center and the Internet, is small, compared to the remaining traffic, then oversubscription in the T_1s should be applied. Applying oversubscription in the T_1s allows a reduction in the number of T_2s in the network, since T_1s connect to more T_0s than T_1s, and, consequently, the cost of the network decreases.

5.2 Non-shortest path routing

We studied, for identical networks and traffic patterns, the difference in the maximum T_0-T_1 link failure probabilities when using valleys. We can observe the results for the HPC intra-cluster communication pattern in figure 5.4(a), and for the streaming pattern in figure 5.4(b).

It can be concluded, from figure 5.4, that valleys do not improve the maximum probability of a link



(a) HPC intra-cluster.

(b) Streaming.

Figure 5.4: Comparison between the use of valleys, for the HPC intra-cluster and streaming patterns, of the maximum probabilities of a T_1-T_2 link failure that a network supports, as a function of the traffic sent by each T_0.

failure and can, on the contrary, make the network more susceptible to overcapacitated links, as can be seen in figure 5.4(b). This is due to the fact that valleys only start being used when the probability of failure is very high and all of the shortest paths become unavailable, that is, a T_0 will begin to have only three, two, and eventually a single shortest path available to other T_0s before valleys are used. When valleys start being used, even though there may be multiple paths available, the number of disjoint paths is very small, as seen in the path diversity analysis. Therefore, if a T_0 is sending a large volume of traffic, as the percentage of links down increases, there will come a time when that T_0 may have multiple paths, but the number of disjoint paths is so mall that links that share multiple paths will be overcapacitated.

Thus, with the BGP implementation as it is, valleys are mostly advantageous when the amount of traffic being sent is very small, so that it can provide connectivity between T_0s that otherwise would not be able to connect. Therefore, to take a greater advantage of valleys, it would be useful if the routing protocol could be modified in order to use, simultaneously, shortest and non-shortest paths, that is, paths with and without valleys, since the average ECMP value would be higher, as seen in 4.18.

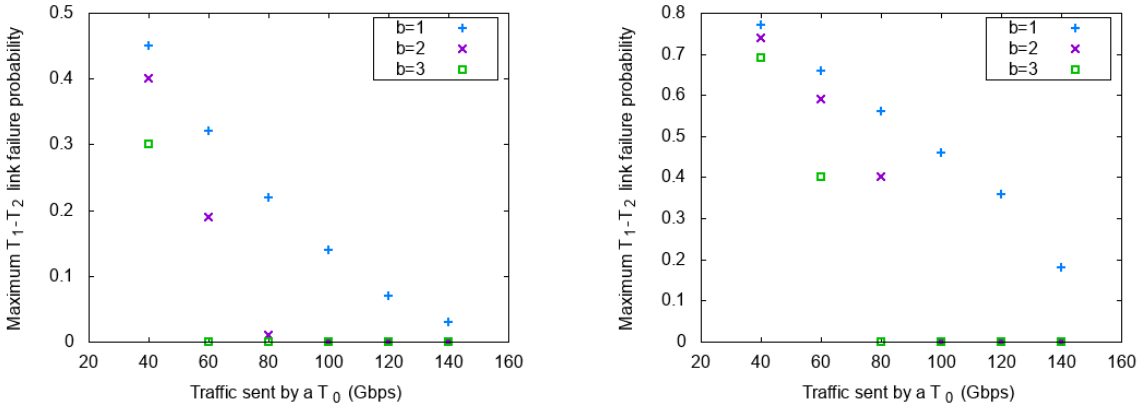
5.3 Parallel links

In this work, we consider the option of using parallel links between the same pair of T_1s-T_2s. For example, a T_1 can connect to 24 T_2s using a single link to each one or, alternatively, to 12 T_2s using two different links to each T_2. This way, the number of links from a T_1 to T_2s is the same. However, as we will further see, even though, in terms of connectivity, the use of parallel links increases the probability of two routers being able to connect with each other in the presence of T_1-T_2 link failures, it has the

opposite impact in the distribution of traffic due to BGP being unable to advertise the number of parallel paths.

As can be seen in figure 5.5, for a network with 16 T_1s and no oversubscription at the T_1s, for the HPC inter-cluster pattern, in figure 5.5(a), and the streaming pattern, in figure 5.5(b), the use of parallel links greatly reduces the maximum T_1-T_2 link failure probability that a network supports.

For example, we can see that on both the HPC inter-cluster and streaming patterns, using parallel links leads to overcapacitated links for any probability of link failure for traffic volumes higher than 100 Gbps. But, on the contrary, when using a single link between pairs of T_1s-T_2s, the network supports, for a volume of 100 Gbps of traffic, approximately, 14% and 36% of link failures for HPC inter-cluster and streaming communication patterns, respectively.



(a) HPC inter-cluster.

(b) Streaming.

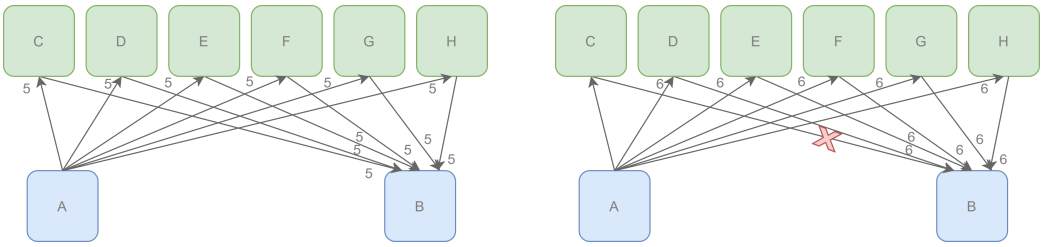
Figure 5.5: Comparison between the use of parallel links between a T_1 and a T_2, given by parameter b, for the HPC inter-cluster and streaming communication patterns, of the maximum probabilities of a T_1-T_2 link failure that a network supports, as a function of the traffic sent by each T_0.

To further understand why this happens and what can be done to minimize this disadvantage of parallel links, we now give a simple example. In figure 5.6(a), we have a network with 8 routers, each link has a capacity of 8 Gbps and A is trying to send 30 Gbps of traffic to router B. Routers A and B have each a link to routers C, D, E, F, G, and H, similarly to what happens in folded Clos topologies, if we consider that A and B are T_1s in different clusters and the remaining routers are T_2s. In the figure, we only represent the flow of traffic, although links between routers are bidirectional.

Now, A can use any of the 6 available paths, each one through a different router, as advertised by BGP. However, when making use of ECMP, it can distribute the traffic by the 6 paths available. Thus, router A sends $30/6 = 5$ Gbps of traffic through routers C, D, E, F, G, and H, enabling load-balancing in the network.

If, however, the link between C and B fails, as represented in figure 5.6(b), BGP messages are

exchanged in the network and A learns it no longer has a path through C. Therefore, to send 30 Gbps to router B, it has to distribute the traffic through the 5 remaining available paths, resulting in $30/5 = 6$ Gbps per path. In this case, the load-balancing continues to be perfect and none of the links are overcapacitated, that is, have a traffic load above their capacity.

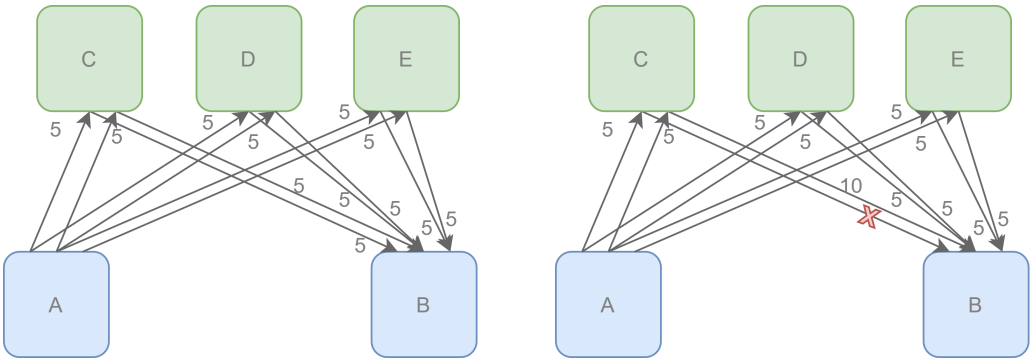


(a) Network without failures. Router A sends 5 Gbps through each path.

(b) Network with a link failure between routers C and B. Router A sends 6 Gbps through each available path.

Figure 5.6: Example of a network with 8 routers and the flow of traffic from router A to router B.

When using parallel links we can, however, use only 3 routers in the upper tier and achieve the same results that we did in figure 5.6(a) when there are no link failures. This is represented in figure 5.7(a), where router A has a path through routers C, D, and E, and, consequently sends 30 Gbps to router B by distributing the traffic through the 6 links that connect router A to routers C, D, and E.



(a) Network without failures. Router A sends 5 Gbps through each link connecting it to routers C, D, and E.

(b) Network with a link failure between routers C and B. Router A sends 5 Gbps through each link connecting to routers C,D and E. However, router C sends 10 Gbps through its only available link to router D.

Figure 5.7: Example of a network with 5 routers and two parallel links between pairs of routers in different tiers. The flow of traffic represented is from router A to router B.

However, when a link from C to B fails, which is the equivalent of what happened previously in figure 5.6(b), the resulting flows are not equal. While in figure 5.6(b), the link failure determines that A has

one less path available to C, this does not happen in figure 5.7(b) because BGP only informs routers of the availability of a path and does not include information about the number of parallel paths. Therefore, router C will not advertise to router A that it has one less path to router B because it continues to have at least one path to it. Router A will then keep making use of the 6 links that have a path to router B due to having paths through any router. However, the traffic that arrives in router C is, afterwards, only routed through the single remaining link that connects it to B. Thus, router A will send 5 Gbps of traffic through each link connecting it to routers C, D, and E, but, in router C, the link that connects C to B will accommodate both flows of 5 Gbps and, in total, carry 10 Gbps to router B.

This situation can, as it happens in this case, lead to links with traffic loads higher than their capacity. Given the fact that, in both networks, links have a capacity of 8 Gbps then, in figure 5.7(b), the remaining link between routers C and B has a flow of 10 Gbps and, consequently, some of the packets arriving at router C will be discarded.

To try and mitigate this issue, there is the option of aggregating parallel links in a single virtual link and, when the number of unavailable physical links reaches a previously defined value, the whole virtual link is taken down. This way, the whole path goes from fully, or almost fully available, in terms of capacity, to no longer available. This can avoid what happens in figure 5.7(b) because, if we had set the threshold at the unavailability of a single parallel link, connection from router C to B would be shutdown and router A would no longer send any traffic through router C, since it no longer had a path to router B through it. This situation is depicted in figure 5.8 and, as can be seen, each link represented carries $30/4 = 7.5$ Gbps of traffic, which is lower than the capacity of each link.

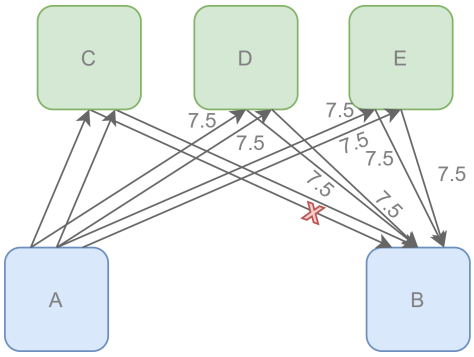


Figure 5.8: Example of a network with 5 routers and two parallel links between pairs of routers in different tiers. The flow of traffic represented is from router A to router B. In this example, given that there is a failure from router C to router B, the path from router C to router B is shut down and the traffic flows through routers D and E.

However, by shutting down whole connections between routers, we are not making use of paths that are still available, even if at lower capacity. One solution that could solve this would be for BGP to include the capacity of the path when advertising it. This already exists, using an extended community, as defined, for example, in [36], but it is static and, consequently, would require a software to detect

failures and automatically implement the correct bandwidth value for each path. Additionally, ECMP has a uniform hash so, in order to distribute more traffic through a certain path, the only solution presented so far is to have repeated paths in the FIB, in proportion to the bandwidth of each path. Thus, if, for example, path U has 10 Gbps of capacity and path V only has 5 Gbps, then, there would be two routes with path U installed in the FIB and only one with path V. This would increase the size and, consequently, the cost of the FIB.

5.4 Conclusions

In this chaper, we assumed traffic patterns of three different applications: HPC, search engines and streaming services. By simulating the use of traffic in different networks, we were able to analyze congestion in links of different topologies, in the presence of link failures.

We concluded that, from the three applications, HPC tolerates the lowest percentage of link failures and search engines the highest. We determined that increasing the number of T_1s per cluster allows a network to support higher percentages of link failures and saw that, for the same number of servers, T_1s per cluster, and spine sets, networks with a smaller number of T_0s per cluster and more clusters achieve better results for the HPC pattern, and worse results for the search engine and streaming patterns. We also determined that oversubscription in the T_1s should be applied in the HPC intra-cluster and search engine patterns, in order to reduce the overall cost of the network.

We studied non-shortest path routing and concluded that allowing valleys does not change the maximum probability of T_0-T_1 link failures for the HPC pattern but leads to worse results for the streaming pattern. This is due to the fact that valleys only start being used when the percentage of link failures is very high and, with valleys, even though many paths are used, the number of disjoint paths is very small, which leads to links common to all paths to have a higher traffic load than their capacity.

We analyzed congestion in networks with and without the use of parallel links and were able to determine that the actual configuration of BGP leads to overcapacitated links when using parallel links, even if the probability of T_1-T_2 link failures is very small. This happens because BGP does not advertise the number of parallel paths.

Chapter 6

Emulation of networks

In order to validate some of our results, we emulated small folded Clos networks using Common Open Research Emulator (CORE) 6.1.0 [37] and FRRouting (FRR) 7.4-dev [38]. Using this software, we were able to configure BGP in the networks and verify that, with the previously disclosed assignment of ASNs, BGP is able to provide a connection between each pair of routers only through the shortest paths. We also confirmed that BGP multi-path allows the installation of multiple shortest paths in the FIB and that ECMP is able to load-balance the traffic by these paths. We verified that, by applying link failures in the network, paths in the shape of valleys were discovered and used when shorter paths were unavailable.

We now present, in detail, how we configured the networks used.

6.1 Initial configuration

An example of a network used is depicted in figure 6.1, with 2 clusters, each one with 4 T_0s and 3 T_1s, and spine planes composed of a single T_2 each.

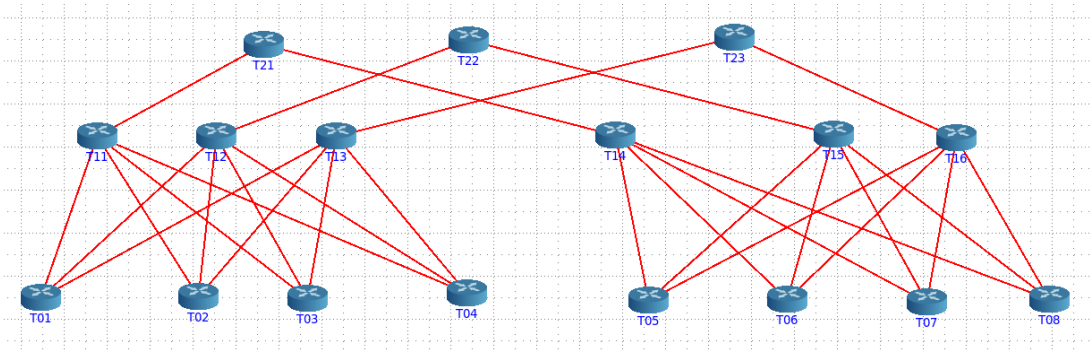


Figure 6.1: Example of a 3-tier folded Clos network, emulated using CORE.

It is possible to add routers, add links between them and configure them using the graphical interface of CORE, as shown in figure 6.1, or, alternatively, by using an XML file. In this case, we started by trying out the graphical interface but ended up implementing a program that, by inserting the number of T_0s and T_1s per cluster, and the number of clusters and spine planes as arguments, builds the XML file with the proper BGP configuration. Then, it is only needed to load the file in CORE.

Before configuring BGP, however, we needed to activate the following services in each router:

- **IPForward**, which allows the routers to, essentially, work as routers and, consequently, when they receive a packet meant for an IP address that it is not its own, it forwards the packet in accordance to the information stored in the FIB.
- **Secure Shell (SSH)**, which enabled us to start an SSH connection from the host to each router and then use the virtual shell, *vtysh*, to apply any configurations needed or to access the RIB and FIB during the emulation.
- **FRRBGP**, a daemon responsible for the implementation of BGP.
- **FRRZebra**, which, as designed by FRR, implements *zebra*, a middleman daemon that FRRBGP communicates with. This daemon is responsible for talking to the dataplane and it is mainly important when multiple protocols are implemented since it coordinates routing decisions but, since it is part of the default configuration of FRR, we decided to keep it.

6.2 BGP configuration

After adding routers to the network, we need to configure the interfaces that interconnect them and implement our routing protocol, BGP. The configuration can be done in two ways: by using the virtual shell in the router, during the emulation, or by editing the file ”usr/local/etc/frr/frr.conf”.

In listing 6.1, we present the interface and BGP configuration of router T01.

Listing 6.1: BGP configuration of router T01.

```
1  interface eth0
2   ip address 10.0.1.2/31
3  !
4  interface eth1
5   ip address 10.0.1.4/31
6  !
7  interface eth2
8   ip address 10.0.1.6/31
9  !
10 interface loopback
11  ip address 10.0.1.0/24
12 !
13 !
14 router bgp 1                          ! set ASN of 1
15  bgp router-id 10.0.1.0               ! set BGP identifier
```

```
16  bgp bestpath as-path multipath-relax
17  neighbor 10.0.1.3 remote-as 100        ! peer with T11
18  neighbor 10.0.1.5 remote-as 100        ! peer with T12
19  neighbor 10.0.1.7 remote-as 100        ! peer with T13
20  !
21  address-family ipv4 unicast
22   network 10.0.1.0/24                   ! advertise prefix 10.0.1.0/24
23  exit-address-family
```

In the first 8 lines, we define the interfaces, eth0, eth1, and eth2, that connect to routers T11, T12 and T13, respectively. By issuing command `interface <name>` we can choose an interface, such as eth0, which is the first Ethernet interface of the router, and then, by following with the command `ip address`, we assign an IP address to the interface. In this case, eth0 has the IP address $10.0.1.2$ and, as we will further see, it is connected to the interface $10.0.1.3$ of router T11. We also define a loopback interface that identifies the device in the network.

Then, from line 14, we have the BGP configuration, using commands defined in [39]. In line 14, by issuing command `router bgp`, we assign an ASN to the router, which is, in this case, 1. Then, in line 15, with the command `bgp router-id`, we assign a BGP router identifier to the router so that it can establish peering sessions with other BGP peers. The identifier is, usually, the first IPv4 address of the loopback interface and, consequently, we defined the BGP identifier as $10.0.1.0$.

On line 16, since multi-path is enabled, by using the command `bgp bestpath as-path multipath-relax`, we are allowing BGP to consider that paths with the same *AS-PATH* length, but with a different AS sequence, have equal cost. This is important due to the use of valleys. Otherwise, since all of the T_1s of a cluster have the same ASN and so do the T_2s in the data center, all of the *AS-PATH*s of shortest paths would be the same. However, by using valleys, since each T_0 has its own ASN, then the *AS-PATH*s of valley paths are different and we need to use this command to allow storing them, in simultaneous, in the FIB.

On lines 17 to 19, we define the peer neighbors. Following `neighbor` we insert the IP address of the interface of the peer that the router is connected to. We also input the ASN of the peer, which appears after `remote-as`. In this case, each peer is a T_1 with ASN 100 and each command shows the IP address of the interface connected to routers T11, T12 and T12, in lines 17, 18 and 19, respectively.

Finally, on line 22 we define the prefix that we want to advertise, $10.0.1.0/24$, using the command `network`. When the `network` command is issued, the network is advertised to every peer of the router. In this case, however, since the command `network 10.0.1.0/24` is between commands `address-family ipv4 unicast` and `exit-addres-family`, it is specified that we want to advertise this route only to all of our neighbors with whom we exchange IPv4 unicast routes. For this family of routers in particular, it is not necessary to specify which neighbors belong to it since all of them are added by default and,

consequently, every router trades IPv4 unicast routes with peers.

Then, in listing 6.2, we present the interface and BGP configuration of router T11.

Listing 6.2: BGP configuration of router T11.

```
1  interface eth0
2   ip address 10.0.1.3/31
3  !
4  interface eth1
5   ip address 10.0.2.3/31
6  !
7  interface eth2
8   ip address 10.0.3.3/31
9  !
10 interface eth3
11  ip address 10.0.4.3/31
12 !
13 interface eth4
14  ip address 10.1.1.2/31
15 !
16 router bgp 100                          ! set ASN of 100
17  bgp router-id 10.0.1.3                 ! set BGP identifier
18  bgp bestpath as-path multipath-relax
19  neighbor 10.0.1.2 remote-as 1          ! peer with T01
20  neighbor 10.0.2.2 remote-as 2          ! peer with T02
21  neighbor 10.0.3.2 remote-as 3          ! peer with T03
22  neighbor 10.0.4.2 remote-as 4          ! peer with T04
23  neighbor 10.1.1.3 remote-as 5000       ! peer with T21
24  !
25  address-family ipv4 unicast
26   neighbor 10.0.1.2 allowas-in 1        ! accept valleys from T01
27   neighbor 10.0.2.2 allowas-in 1        ! accept valleys from T02
28   neighbor 10.0.3.2 allowas-in 1        ! accept valleys from T03
29   neighbor 10.0.4.2 allowas-in 1        ! accept valleys from T04
30  exit-address-family
```

Having previously explained the purpose of each command, we now briefly summarize configuration of router T11. We can see that, similarly to router T01, the interfaces are firstly configured and that interface eth0, with an IP address of $10.0.1.3$, is the one that connects with router T01, as defined in line

17 of listing 6.1. Then, interfaces eth1, eth2, eth3 and eth4 connect to routers T02, T03, T04 and T21, respectively. In this case, since we are not advertising a network from router T11, we did not setup a loopback interface.

It can also be observed that this router has an ASN of 100 and, since there is no loopback interface, its BGP identifier is the IP address of interface eth0: $10.0.1.3$. Additionally, the router connects to routers T01, T02, T03, T04, and T22, with ASNs 1, 2, 3, 4, and 5000, respectively.

Finally, the main difference between T01 and T11, in terms of configuration, is that T11 does not advertise its own route but only re-advertises routes that receives from its peers. Additionally, the command that allows valley paths, `neighbor <ip-address> allowas-in <nr>` is added here, in the T_1s, as can be seen in lines 26-30. In these lines, it is configured that the router T11 accepts routes from each neighbor, even if they have, at most once, the ASN of T11 in the *AS-PATH*.

Finally, we present the interface and BGP configuration of router T22 in listing 6.3.

Listing 6.3: BGP configuration of router T21.

```
1  interface eth0
2   ip address 10.1.1.3/31
3  !
4  interface eth1
5   ip address 10.1.4.3/31
6  !
7  router bgp 5000                          ! set ASN of 5000
8   bgp router-id 10.1.1.3                  ! set BGP identifier
9   bgp bestpath as-path multipath-relax
10  neighbor 10.1.1.2 remote-as 100         ! peer with T11
11  neighbor 10.1.4.2 remote-as 101         ! peer with T14
```

It is worth noticing that, since we did not want to limit the number of equal-cost paths to a prefix installed in the FIB, and the default was 64, we did not use the command `maximum-paths <nr>`.

6.3 Simulating link failures

In order to test how the network responded to link failures, we had to simulate them. The most efficient way of doing this that we found, since we could not disconnect links during the emulation, was to alter the BGP and interface configuration while the emulation was running, and send the signal to shutdown connections to peers.

To do this, whenever we wanted to drop a connection between peers, we issued the commands

`neighbor <ip-address-> shutdown` and `interface <ethi> shutdown` in the virtual shell of the router.

6.4 Adding traffic flows

In order to apply traffic between the T_0s, we used Multi-Generator (MGEN) [40]. By using SSH and MGEN, we were able to start flows of traffic from each router using the command

```
1     mgen event "<start-time> ON <flow-id> UDP DST <destination-ip>/<destination-
          port> <traffic-pattern> TTL 30".
```

To stop the flow we simply needed to execute the command

```
1     mgen event "<start-time> OFF <flow-id>".
```

We could also begin a Transmission Control Protocol (TCP) flow, instead of User Datagram Protocol (UDP), after starting a listening port on the router destination with the command

```
1     mgen event "listen TCP <port>"
```

It should also be noted that we set the option of Time To Live (TTL) to 30 since, even though they specify the default value is 255, we noticed flows would stop after just 3 hops. However, after setting TTL to 30, the traffic flows behaved as expected.

Chapter 7

Conclusions

We now present the main results of our work and suggest how it can be further improved.

7.1 Results

In this thesis, we studied in detail the networks in data centers, analyzed their cost, in terms of routers, and their resilience to link failures, given the different architectures we can build using folded Clos topologies.

We studied the cost of the routers in the different tiers of a 3-tier folded Clos topology, T_0s, T_1s and T_2s. We identified that T_2 routers need more memory and forwarding capacity than T_0s and T_1s. T_2 routers need a faster backplane to provide higher forwarding capacities, in order to interconnect the clusters and connect them to external endpoints to the data center, and bigger RIBs and FIBs in order to install the routes to prefixes from all over the Internet. We concluded that the cost per server, as well as the power per server, tends to diminish with the increase in the number of servers. Additionally, we found that T_2s have a bigger percentage in the acquisition cost of smaller networks than in bigger networks. We showed that networks with 40 Gbps links have a cost and power usage up to 2 times higher than networks with links of 10 Gbps, even though the capacity is 4 times higher.

A detailed analysis of connectivity in the presence of link failures in terms of the different parameters of a topology, namely the number of T_1s per cluster and the number of connections from a T_1 to T_2s, was presented. We studied T_0-T_1 and T_1-T_2 links separately and found that the crucial links in the network are the ones between T_0s and T_1s since all of the paths from a T_0 to other T_0s, or to endpoints external to the data center, share these links. Therefore, the number of disjoint paths that a T_0 has is limited by the number of T_1s in its cluster.

We determined mathematical expressions that provide the probability of connectivity between pairs of servers, and between servers and the Internet, in terms of link failures. Therefore, we were able to obtain approximate values of availability, usually defined in the SLAs of data centers, and compare the cost of networks for different values of availability between servers, and between the data center and the Internet.

We showed how the number of paths that a T_0 has to other endpoints diminishes in the presence of T_0-T_1 link failures and presented mathematical expressions that provide the number of entries needed

in the RIB and the FIB of each router, assuming that each T_0 advertises a single aggregated prefix to its servers.

Three different traffic patterns that can be present in data centers were assumed and analyzed: HPC intra-cluster and inter-cluster, search engines and streaming services. We showed that increasing the number of T_1s per cluster increases the resilience to link failures in the network. We were able to conclude that, for HPC intra-cluster and search engine traffic patterns, most of the traffic will be intra-cluster and, consequently, oversubscription at the T_1s should be applied. By applying oversubscription at the T_1s, they connect to more T_0s than T_2s so we can reduce the number of T_2s used, which leads to a reduction of the overall cost of the network.

We studied non-shortest path routing and concluded that using non-shortest paths, which we designated by valleys, improves end-to-end connectivity in the presence of link failures. We saw how the average of the number of redundant paths that a T_0 has to other endpoints increases by using valleys, but only for high probabilities of link failures. We concluded that the simultaneous use of shortest and non-shortest paths, instead of exclusively using shortest or non-shortest paths, can further increase the number of redundant paths, in the presence of link failures. However, the use of valleys requires more memory in routers, which can have an impact in the cost of the network, and it can lead to networks with overcapacitated links for lower values of T_0-T_1 link failure probabilities than when only shortest path routing is used.

We presented the option of parallel links and concluded that, while maintaining the number of connections between a T_1 and T_2s, parallel links allow a reduction of the overall cost of the network. By using fewer T_2s and using parallel links between T_1s and T_2s, it is possible, as it was seen, to reduce up to 61% of the overall cost of the network. We also saw an improvement in connectivity, in the presence of T_1-T_2 link failures, when parallel links are used, but we did not consider router failures. If we consider router failures, a T_2 failure has a bigger impact on a network with parallel links due to more paths becoming unavailable. Additionally, since BGP does not advertise the number of parallel paths, the use of parallel links can lead to overcapacitated links, even for small values of T_1-T_2 link failure probabilities. Therefore, measures such as aggregating physical links into a single virtual link should be applied in order to avoid congestion.

Finally, we emulated small networks, by using CORE and configured BGP using FRR, which enabled us to verify some of our results.

7.2 Future Work

For future work, we think it would be interesting to study the following:

- **Widen this study to different extensions of Clos topologies**. Different connections between the

tiers should be explored, such as, for example, a T_1 connecting to more than a single spine plane. Additionally, as in the HGRID of Facebook, there may be an additional tier of routers connecting 3-tier folded Clos networks as a unit.

- **Study in more detail the connection of the data center to the Internet**. In this work, we considered that every T_2 connected to the same "super node" that made the connection to the Internet but, other than an upper tier, the routers connecting the data center to the Internet can be, for example, structured as an additional cluster.

- **Add failures in routers.** Our analysis does not take into account the probability of a failure in a router. This can be important since, when a router fails, every link connected to it is unusable and, for example, this may have an impact in the use of parallel links between the pairs of T_1s-T_2s.

- **Analyze the simultaneous use of shortest and non-shortest paths.** Non-shortest paths, which we designate, in this case, by valleys, are advantageous to connect two T_0s that otherwise could not but, however, they only start being used when the number of link failures is elevated. Therefore, the option of using, simultaneously, valley and non-valley paths should be studied to determine the advantages it could provide, especially in the distribution of traffic.

- **Envision how to distribute traffic proportionately to the bandwidth of each path.** Using parallel links is extremely important to reduce the overall cost of the network and, additionally, can improve the probability of connectivity between T_0s in the presence of link failures. However, as was previously seen, it makes the network highly susceptible to overcapacitated links when there are failures due to BGP not advertising the number of parallel paths. And, even if it advertised the bandwidth of each path, the FIB would largely increase in size. Therefore, it would be useful to find a way to distribute traffic proportionately to the bandwidth of each path without these constraints.

- **Study aggregation of routes**. The aggregation of routes allows the reduction of the size of the FIBs and RIBs and, consequently, could be important in reducing the cost of the many routers used.

Bibliography

[1] Andreyev, A., "Introducing data center fabric, the next-generation Facebook data center network," Facebook, Tech. Rep., 2014. [Online]. Available: https://engineering.fb.com/production-engineering/introducing-data-center-fabric-the-next-generation-facebook-data-center-network/#

[2] Andreyev, A. and Wang, X. and Eckert, A., "Reinventing Facebook's data center network," Facebook, Tech. Rep., 2019. [Online]. Available: https://engineering.fb.com/data-center-engineering/f16-minipack/

[3] R. R. Reyes, S. Sultana, V. V. Pai, and T. Bauschert, "Analysis and evaluation of capex and opex in intra-data centre network architectures," in *2019 IEEE Latin-American Conference on Communications (LATINCOM)*, 2019, pp. 1–6.

[4] X. Wang, J.-X. Fan, C.-K. Lin, J.-Y. Zhou, and Z. Liu, "BCDC: a high-performance, server-centric data center network," *Journal of Computer Science and Technology*, vol. 33, no. 2, pp. 400–416, Mar 2018. [Online]. Available: https://doi.org/10.1007/s11390-018-1826-3

[5] P. Mell, T. Grance *et al.*, "The nist definition of cloud computing," 2011.

[6] K. Hazelwood, S. Bird, D. Brooks, S. Chintala, U. Diril, D. Dzhulgakov, M. Fawzy, B. Jia, Y. Jia, A. Kalro, J. Law, K. Lee, J. Lu, P. Noordhuis, M. Smelyanskiy, L. Xiong, and X. Wang, "Applied Machine Learning at Facebook: A Datacenter Infrastructure Perspective," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2018, pp. 620–629.

[7] L. A. Barroso, J. Clidaras, and U. Hölzle, *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines, Second Edition*. Morgan & Claypool Publishers, 2013. [Online]. Available: http://dx.doi.org/10.2200/S00516ED2V01Y201306CAC024

[8] R. Kohavi, A. Deng, B. Frasca, T. Walker, Y. Xu, and N. Pohlmann, "Online controlled experiments at large scale," in *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 1168–1176. [Online]. Available: https://doi.org/10.1145/2487575.2488217

[9] [Online; accessed December 13, 2020]. [Online]. Available: https://www.thinkwithgoogle.com/marketing-strategies/app-and-mobile/mobile-page-speed-new-industry-benchmarks/

[10] “SLA for Virtual Machines,” [Online; accessed November 30, 2020]. [Online]. Available: https://azure.microsoft.com/en-us/support/legal/sla/virtual-machines/v1_9/

[11] “SLA for Azure Cosmos DB,” [Online; accessed December 13, 2020]. [Online]. Available: https://azure.microsoft.com/en-gb/support/legal/sla/cosmos-db/v1_3/

[12] [Online; accessed December 26, 2020]. [Online]. Available: https://cloud.google.com/compute/sla

[13] V. Dukic, G. Khanna, C. Gkantsidis, T. Karagiannis, F. Parmigiani, A. Singla, M. Filer, J. L. Cox, A. Ptasznik, N. Harland, W. Saunders, and C. Belady, “Beyond the mega-data center: Networking multi-data center regions,” in *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication*, ser. SIGCOMM ’20. New York, NY, USA: Association for Computing Machinery, 2020, p. 765–781. [Online]. Available: https://doi.org/10.1145/3387514.3406220

[14] D. A. Popescu, N. Zilberman, and A. Moore, “Characterizing the impact of network latency on cloud-based applications’ performance,” 11 2017.

[15] M. Al-Fares, A. Loukissas, and A. Vahdat, “A scalable, commodity data center network architecture,” in *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*, ser. SIGCOMM ’08. New York, NY, USA: Association for Computing Machinery, 2008, p. 63–74. [Online]. Available: https://doi.org/10.1145/1402958.1402967

[16] D. G. Dutt, *BGP in the datacenter*. O’Reilly Media, 2017. [Online]. Available: https://www.oreilly.com/library/view/bgp-in-the/9781491983416/

[17] L. A. Barroso, J. Dean, and U. Holzle, “Web search for a planet: The Google cluster architecture,” *IEEE Micro*, vol. 23, no. 2, pp. 22–28, 2003.

[18] H. Hu, Y. Wang, L. Yang, P. Komlev, L. Huang, X. S. Chen, J. Huang, Y. Wu, M. Merchant, and A. Sacheti, “Web-Scale Responsive Visual Search at Bing,” in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery ; Data Mining*, ser. KDD ’18. New York, NY, USA: Association for Computing Machinery, 2018, p. 359–367. [Online]. Available: https://doi.org/10.1145/3219819.3219843

[19] K. Jayaraman, N. Bjørner, J. Padhye, A. Agrawal, A. Bhargava, P.-A. C. Bissonnette, S. Foster, A. Helwer, M. Kasten, I. Lee, A. Namdhari, H. Niaz, A. Parkhi, H. Pinnamraju, A. Power, N. M. Raje, and P. Sharma, “Validating datacenters at scale,” in *Proceedings of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM ’19. New York, NY, USA: ACM, 2019, pp. 200–213. [Online]. Available: http://doi.acm.org/10.1145/3341302.3342094

[20] A. Singh, J. Ong, A. Agarwal, G. Anderson, A. Armistead, R. Bannon, S. Boving, G. Desai, B. Felderman, P. Germano, A. Kanagala, H. Liu, J. Provost, J. Simmons, E. Tanda, J. Wanderer, U. Hölzle, S. Stuart, and A. Vahdat, "Jupiter Rising: A Decade of Clos Topologies and Centralized Control in Google's Datacenter Network," *Commun. ACM*, vol. 59, no. 9, p. 88–97, Aug. 2016. [Online]. Available: https://doi.org/10.1145/2975159

[21] "Google's data center photo gallery," [Online; accessed November 6, 2020]. [Online]. Available: https://www.google.com/about/datacenters/gallery

[22] J. Arjona Aroca and A. Fernández Anta, "Bisection (band)width of product networks with application to data centers," in *Theory and Applications of Models of Computation*, M. Agrawal, S. B. Cooper, and A. Li, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 461–472.

[23] X. Meng, Z. Xu, B. Zhang, G. Huston, S. Lu, and L. Zhang, "IPv4 Address Allocation and the BGP Routing Table Evolution," *SIGCOMM Comput. Commun. Rev.*, vol. 35, no. 1, p. 71–80, Jan. 2005. [Online]. Available: https://doi.org/10.1145/1052812.1052827

[24] "BGP Best Path Selection Algorithm," [Online; accessed November 19, 2020]. [Online]. Available: https://www.cisco.com/c/en/us/support/docs/ip/border-gateway-protocol-bgp/13753-25.html

[25] "Examples: Configuring BGP Multipath," [Online; accessed November 19, 2020]. [Online]. Available: https://www.juniper.net/documentation/en_US/junos/topics/topic-map/bgp-multipath.html

[26] J. F. Kurose and K. W. Ross, *Computer Networking: A Top-Down Approach (6th Edition)*, 6th ed. Pearson, 2012.

[27] J. W. Stewart, III, *BGP4: Inter-Domain Routing in the Internet*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1998.

[28] L. Luo, G. Xie, S. Uhlig, L. Mathy, K. Salamatian, and Y. Xie, "Towards TCAM-Based Scalable Virtual Routers," in *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 73–84. [Online]. Available: https://doi.org/10.1145/2413176.2413186

[29] H. Hwang, S. Ata, K. Yamamoto, K. Inoue, and M. Murata, "A new TCAM architecture for managing ACL in routers," *IEICE Transactions*, vol. 93-B, pp. 3004–3012, 11 2010.

[30] D. Andersen, H. Balakrishnan, N. Feamster, T. Koponen, D. Moon, and S. Shenker, "Accountable Internet Protocol (AIP)," vol. 38, 10 2008, pp. 339–350.

[31] Q. Guo, X. Guo, Y. Bai, and E. İpek, "A resistive TCAM accelerator for data-intensive computing," in *2011 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2011, pp. 339–350.

[32] “QFX5200 Ethernet Switches,” [Online; accessed November 20, 2020]. [Online]. Available: https://www.juniper.net/us/en/products-services/switching/qfx-series/qfx5200/

[33] “QFX10008 and QFX10016 Ethernet Switches,” [Online; accessed November 20, 2020]. [Online]. Available: https://www.juniper.net/us/en/products-services/switching/qfx-series/qfx10000/

[34] O. Alipourfard, J. Gao, J. Koenig, C. Harshaw, A. Vahdat, and M. Yu, “Risk based planning of network changes in evolving data centers,” in *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, ser. SOSP ’19. New York, NY, USA: Association for Computing Machinery, 2019, p. 414–429. [Online]. Available: https://doi.org/10.1145/3341301.3359664

[35] N. Liu, A. Haider, D. Jin, and X.-H. Sun, “Modeling and Simulation of Extreme-Scale Fat-Tree Networks for HPC Systems and Data Centers,” *ACM Transactions on Modeling and Computer Simulation*, vol. 27, pp. 1–23, 07 2017.

[36] “BGP Link Bandwidth,” [Online; accessed November 29, 2020]. [Online]. Available: https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/iproute_bgp/configuration/xe-16/irg-xe-16-book/bgp-link-bandwidth.html

[37] [Online; accessed November 19, 2020]. [Online]. Available: https://www.nrl.navy.mil/itd/ncs/products/core

[38] [Online; accessed November 19, 2020]. [Online]. Available: https://frrouting.org

[39] [Online; accessed December 03, 2020]. [Online]. Available: http://docs.frrouting.org/en/latest/bgp.html

[40] [Online; accessed December 03, 2020]. [Online]. Available: https://www.nrl.navy.mil/itd/ncs/products/mgen