Dynamic Node Clustering in Hierarchical Optical Data Center Network Architectures

by

Georgia Dimaki

B.S. Computer Science, Athens University of Economics and Business (2018)

Submitted to the Sloan School of Management in partial fulfillment of the requirements for the degree of

Master of Science in Operations Research

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2020

Signature of Author .....
Sloan School of Management
August 7, 2020

Certified by .....
Eytan Modiano
Professor, Department of Aeronautics and Astronautics
Thesis Supervisor

Accepted by .....
Patrick Jaillet
Dugald C. Jackson Professor, Electrical Engineering and Computer Science
CoDirector, Operations Research Center

Dynamic Node Clustering in Hierarchical Optical Data Center Network Architectures

by

Georgia Dimaki

Submitted to the Sloan School of Management
on August 7, 2020, in partial fulfillment of the
requirements for the degree of
Master of Science in Operations Research

Abstract

During the past decade an increasing trend in the Data Center Network's traffic has been observed. This traffic is characterized mostly by many small bursty flows (mice) that last for less than few milliseconds as well as a few heavier more persistent (elephant) flows between certain number of nodes. As a result many relatively under-utilized network links become momentarily hotspots with increased chance of packet loss. A potential solution could be given by Reconfigurable Optical Data Centers, due to higher traffic aggregation links and topology adaptation capabilities. An example is a novel two level hierarchical WDM-Based scalable Data Center Network architecture, RHODA, which is based on the interconnection of high speed equal sized clusters of Racks. We study the traffic based dynamic cluster membership reconfiguration of the Racks. Main goal is to maintain a near optimal network operation with respect to minimization of the inter cluster traffic, while emphasising better link utilization and network scalability. We present four algorithms, two deterministic greedy and two stochastic iterative, and discuss the tradeoffs of their use. Our results draw two main conclusion: 1) Stochastic iterative algorithms are more suitable for dynamic traffic based reconfiguration 2) Fast algorithmic deployments come at a price of reduced optimality.

Thesis Supervisor: Eytan Modiano
Title: Professor, Department of Aeronautics and Astronautics

Acknowledgments

I would like to thank everything and everyone that made this journey one of the most life changing experiences. First and foremost, my greatest thanks go to my advisor, Professor Eytan Modiano. Thank you for your patience and support, for untangling the thoughts in my brain, for helping me learn how to conduct research.

This material is based upon work supported by the National Science Foundation under Grant No. CNS-1617091. So, thank you NSF as well.

Also I would like to give my thanks to my past advisors, Stavros Toumpis and Vangelis Markakis for being so inspiring and for encouraging me to come to United States. To a mentor and very patient friend, Esmerald. And last but not least, to my first Computer Languages teacher, Makis Kapetis, without whom my career goals would not have been the ones I have today.

Finally, I want to give my warmest thank to my family and friends who always support and trust my decisions. To my grandfather who was the first one to be excited for my trip to Boston and to my grandmother. To my mother, who made sure to check on me, remind me of all important dates and update me with every crazy detail of what is happening around the world. To my father, always curious about my ideas. To my sister who was there in every crisis supporting me with unconditional love. To my uncle, aunt and cousin. To Dimitri, Evangelia, Andriani and Alex who were always there to remind me all my achievements the moments I thought I had none. To Evita and Stephania, who are with me in every spiritual journey I take. To Martin, who shared a wonderful quarantine/thesis writing with me. To my roommates, Basuhi, Martina and Julie, my first and dearest friends in Boston. And to my friends from the OR Center, Kayla, Vassilis, Galit and Sam, for our long discussions with beer and dumplings. Thank you all so much.

And to me. For changing. For learning. For not giving up.

THIS PAGE INTENTIONALLY LEFT BLANK

Contents

Chapter 1

Introduction

Data Centers are a big part of today's Internet, providing service to millions of users around the world. During the past decade an increasing trend has been observed in the Data Center Networks' traffic. This traffic is characterized mostly by small bursty flows (mice) that last for less than few milliseconds as well as heavier more persistent (elephant) flows between certain number of nodes. As a result of the bursty traffic we have the effect of momentary hotspot links causing congestion and packet losses, even though in general Data Center links are under-utilized.

Optical technology and WDM can help in building adaptable networks with high traffic aggregation within the optical links. Many proposed network architectures have already employed Optical components in order to circuit switch big traffic flows dynamically [72][34]. More generally, Optical Networks enable better data aggregation and network topology reconfigurability. In this sense we could benefit from hierarchical optical architectures that localize the elephant flows in high speed communication clusters, while supporting mice flows as inter cluster traffic in an aggregate manner. The flexibility of such networks to dynamically adapt in the changing traffic patterns has the potential of increasing the link utilization while protecting the network from momentary congestion.

RHODA [87], a novel two level hierarchical Wavelength Division Multiplexing - Based scalable Data Center Network architecture, is based on the interconnection and reconfiguration of high speed equal sized clusters of Racks. The success of RHODA,

or similar hierarchical architectures, highly depends on the reconfiguration of the clusters. In this work we study and evaluate algorithms for the dynamic cluster membership reconfiguration of the network nodes, i.e. network server switches in a data center. Main objective is to maintain a near optimal network operation with respect to minimization of the inter cluster traffic while emphasising on better link utilization and network scalability. We describe four algorithms, two deterministic greedy and two stochastic, and evaluate their performance both in the static and in the dynamic network models. We find that stochastic iterative optimization methods can lead to satisfying solutions and that there is a trade-off between time and optimality.

The present thesis is structured as follows. In Chapter 2 we explore the Data Center Networks, aiming to find the system requirements they impose on our algorithms. We emphasize specifically on Optical Data Center capabilities closing this chapter with a presentation of the optical architecture of interest, RHODA. Then, in Chapter 3 we focus on the problem of cluster membership assignment and reconfiguration. This problem belongs to the broader family of problems named Graph Partitioning, so this chapter is dedicated in realizing the complexity of problems of this class as well as the methods used to solve them. We conclude with a presentation of the model and the algorithms we use for the static clustering assignment. Chapter 4 continues with the dynamic model for traffic based reconfiguration of the clusters. The results and conclusions drawn from our methods are discussed in Chapter 5. Finally, Chapter **??** concludes this work and provides some ideas for future directions in this area.

Chapter 2

Data Center Networks

Data Centers are probably one of the biggest and most important parts of the Internet today [54]. It is an essential component of every company to be successful and serves millions of users around the globe. Their history ages three decades back, in 1991, when e-commerce was an emerging and promising business plan and the client-server model attracted more attention. Companies needed a non-stop online operation and the first goal of data centers was mostly to act as crossover backup with infrastructure to support the increasing needs of users.

Through the years, the requirements and structure for Data Centers matured. Today, a Data Center is defined as a large group of networked computer servers typically used by organizations for remote storage, processing and distribution of large amounts of data. Storage can be a service provided to users, as for example cloud storage, but more generally it is used for all the important user information the company owning the data center holds. Thus the data center needs to maintain and manage a data base. In terms of maintenance and robustness data bases operate in a distributed manner with multiple replicas, thus a big part of traffic within a data center is dedicated to data base fault tolerance and synchronization. Processing as well is performed in a distributed manner for better parallelization (for example in MapReduce framework). Very often it aims to solve computationally intensive large-scale problems, resulting in extensive data exchange within a data center. Finally, closely related to both of the previous tasks, data distribution and exchange happens

either internally to support the data center operations or it aims to update one of the million users that use the data center. Thus, the computational needs and distributed parallel nature of data centers require fast, robust and scalable internal networks.

The aim of this section is to describe those increasing needs in modern data centers that motivate this study. In section 2.1 we present the infrastructure and communication patterns of data centers the last two decades. In section 2.2 we present the most used network topology of data centers, hierarchical data centers. Section 2.3 introduces the optical technology which enables future optical data centers with various literature examples presented in section 2.4. This chapter concludes with the presentation of a novel hierarchical optical data center network architecture in section 2.5, for the operation of which this work is dedicated to.

2.1 Inside a Data Center

A data center is an extended communicative network of servers. Servers manage all the heavy computation that is taking place within the data center. For an effective distributed parallel computing there is need for a supportive network infrastructure that connects the servers and is fast, efficient and robust. Of course so many devices located together generate heat that needs to be managed properly for them to operate at their full potential. So, effectively the key components for a data center are the servers, network switches and communication devices, cables to make the connection happen and cooling system.

All these might seem trivial today, but they took at least a decade to mature into the data centers as we know them. Due to the large number of machines in a data center the complexity and the cost of the network is very high. To be maintainable and manageable a lot of optimization took place, to an extend that may not be immediately realized by a simple observer. The level of detail in a data center's design starts from the single devices up to even the location of the data center. So as easy as it will be described now, the data center infrastructure is one of the highlights of the networks best design.

Infrastructure

To start, the general structure of a data center is as follows. The data center is broken into clusters placed in different locations. Each cluster comprises of large number of servers and switches that are interconnected through an internal network infrastructure. To keep trouble shooting and cabling complexity at manageable levels, devices need to be organised. Thus placement of machines is very important. Typically 20-40 servers [54] are packed into Racks, a structure similar to a refrigerator with its own cooling mechanism and a central switch to inter connect the servers. The switch is called Top of the Rack Switch (ToR switch). Each ToR switch then connects to the cluster switches that form what we know as Data Center Network (DCN).

The DCN is a uniformly structured network of local devices. To maintain costs reasonable, some of the most common architectures for such a network focus on the use of cheap commercialized switches [2]. This of course requires many more devices and cabling. Cables are placed and packed together in an optimal way, so that their length, which costs in money and latency, is no more than needed, and troubleshooting is possible. The switches' interconnection infrastructure is the backbone of the communication and computation that takes place in a data center. We seek to provide architectures that are cost and power efficient with high throughput, low latency and fault tolerance. These are of great importance for this study.

Typically however when we discuss or research about data centers we look at a data center cluster in an abstract level. Since servers for example are always connecting to the network through their ToR switch, we can treat them as groups creating the abstraction of a network node which corresponds to one Rack. The network nodes then connect in the way the data center network architecture describes. Switches and other components are also part of the architecture's specification. In this way we can now focus on the network itself to find the best way to form and analyse it. The machines' placement and maintenance are not easy tasks but are typically discussed in an abstract level through the complexity metrics of a network.

Applications

So what is the direction of data center networks' future? This is basically drawn by the way the data center is used. Data centers started growing as simple enterprise networks dedicated to support different services. For this reason they present variation in traffic, caused by the different types of services they provide. However, as their operation started creating value to the owning company, things moved towards more competitive grounds and the initial variation based on the applications run started decaying. Many services became the norm of what a data center should provide and a lot of effort was put to optimize all the internal processing taking place. Nowadays data management methods are almost standardized, taking the lead to data center traffic shaping.

The most common services supported in modern data centers are web mail, web services, user authentication and storage. Others commercial ones include instant messaging, search, video streaming, scientific computing, data analysis etc. Each service to be provided has a major workflow with an effect on the traffic generated, in both the amount of traffic and the way it spreads across the network/computation nodes. What all have in common however is the data manipulation. There are a few very important operations required for the data center to be highly available to the end user and keep the data safe. These include system snapshotting, virtualization, data replication and deduplication - removal of duplicates - and virtual machine migration. Very briefly all these operations verify that the data are backed up and synchronized, when outdated deleted, and that neither the data or jobs processing them are tied on a single machine creating critical points of failure. Thus, there is some traffic shaping depending only to these essential operations, independent of the specific services provided.

Traffic patterns

The knowledge around specific data center traffic patterns however is limited. There are two reasons on why it is so. First, it is difficult to track and analyse the traffic

of many machines in different type of data centers. Thus in the existing literature, common practice it is to track some of the machines, using SNMP (Simple Network Management Protocol) polls [9][10], in strategic locations within the data center and from those to infer the network structure as well as key traffic characteristics [9]. However the picture is not complete, causing conflicting results and conclusions. Second, big vendors do not allow tracking their traffic for various reasons that may extend to personal data safety as well. There have been a few examples though of research originated in a specific company that gives insights. For example in [75] the traffic of Facebook clusters is analysed. The absence of the big picture suggests that for now we can focus on the characteristics the existing research has concluded on, realizing that the results might be outdated as data centers keep expanding.

There are a few traffic characteristics researchers agree on. Generally it seems that there are two types of flows traveling the network. A high percentage of the flows are small in size ($\leq$10KB) and last for a few milliseconds [9]. To these we refer to as *mice flows*. On the other hand, almost 10% of the flows appear to concentrate most of the bytes transmitted [9] and tend to last longer. These are likewise called *elephant flows*. Long-lived flows however are not always heavy, as indicated by [75], and load balancing is likely related factor. Not only that but also for many flows internal transmissions appear to be similarly bursty. In terms of interarrival time, flows arrive at microsecond scale and the average number of active flows per rack is less than 10000. What is noticeable is that even though certainly different applications generate different traffic patterns [9][75], according to [9] dominant application traffic patterns are not enough to understand traffic patterns within a data center.

Some other very insightful observations relate to link utilization. It seems that constantly there exist a small number (less than 25% in the core) of hot-spot links the set of whom might remain the same or randomly vary. In general however, both link and buffer utilization seems to follow diurnal patterns [9] [10] [75]. Also, in [32] is observed that in Rack level communication hot-spots tend to be stable in a hundreds of millisecond scale. Interestingly enough, losses do not correlate with high utilization [9]. This suggests that cause are momentary bursts fast enough not to

cause increase in the average link utilization. To accommodate the peaks of traffic load without failures, over-provision of network communication resources is typical in the commonly used data center network architectures [2] [32] [80] with emphasis on full bisection bandwidth, so that link capacity shortage is avoided. However, this is a worst case scenario approach which leads to even lower network utilization [34], which increases the costs (infrastructure acquiring, maintenance etc.) without necessarily be, as the aforementioned results suggest, more effective than traffic offloading [9] or buffer size tuning [75].

Conflicts in the results of traffic research relate mostly to the traffic Rack locality and packets interarrivals. In [9] it was observed that in big cloud data centers traffic was mostly Rack local. The reasoning behind this observation was related to possibly optimized job placement so that highly communicative components communicate faster without congesting the network. On the contrary this statement fades away under the results of [75]. We could thus say that traffic locality mostly relates to job and virtual machine placement and traffic offloading policies that may vary across different vendors. Regarding packet interarrivals, both [10] and [9] agree on an ON-OFF nature with ON duration characterized by heavy tailed distribution. This is not supported by [75], explained as per destination only phenomenon. Concluding it seems that the points of conflict are caused by algorithmic schemes for traffic engineering applied, which means they should be considered in a per data center manner.

Those traffic characteristics suggest that we should be cautious when it comes to data center deployments. Specifically, as argued in [9] apart from resource over-provisioning, centralized routing or network reconfiguration might also be in performance questioning. This is because as networks scale those algorithms need to maintain certain performance guarantees in order to be able to capture the small micro second scale of interarrival times of flows. Otherwise they might impose big delays, and overhead comparable to a typical flow size, making the solutions impractical. Promising results of [75] however, show that traffic engineering in rack flow aggregation scale has a potential to be effective, since aggregated flows present

more temporal stability, while per application engineering might be valuable as well. Possible approaches include hot-spot fine grained predictions or joint optimization of traffic offloading and routing/reconfiguration.

In another perspective, this elephant/mice flow co-existence could potentially benefit by two level hierarchical architectures able to dynamically clusters nodes (Rack) with high communication requirements (elephant flows) together, while satisfy the small communication demand in an aggregate manner. As we shall discuss later in this chapter, such ideas are the drivers of hybrid electrical/optical data center architecture designs. Since traffic changes through the time, flexible reconfiguration of node cluster membership in a timely manner is critical to get the full benefits from such a design. For highly clustered traffic, or in other words concentrated hot-spot traffic, such an approach would decrease the capacity demands of the network links and provide a better utilization of the network. Given an architecture were this is possible there are a lot of things to consider, such as what are practical reconfiguration algorithms or when should reconfiguration be performed. Subsequent chapters study those exact architectural and algorithmic enablers, their promises and limitations.

Summarizing, all the discussed ideas, special design and uniform characteristic of data center networks create opportunities for optimized network deployment almost analogous to those of supercomputer processors. Data center traffic characteristics are helpful towards these ends. When designing architectural capabilities without focusing on one specific type of Data Center, one should have in mind the different trends in the traffic presents in order to capture a variety of types. The solution proposed can be later tied down to a specific Data Center's needs, based on services provided or other policy concerns.

2.2 Hierarchical Data Center Network Architectures

Hierarchical design in networks has many advantages when it comes to large networks such as DCN. The inherited modularity and simplicity of such design makes the network easy to maintain, troubleshoot and most importantly scale. For this reason

the most successful and highly used designs follow a tree structure. Even more, complex designs often implement a modular pattern easy to replicate in order to maintain the similar useful principles.

The traditional DCN architectures are hierarchically structured in two or three layers typically, with enterprise equipment at the higher layers. This is called the Multi-tier architecture. In the three tier model the levels are top-down named as core, aggregation and access level. As their name defines, the access level provides connectivity to the end user, the aggregation level aggregates the traffic from the access layer and finally the core acts as the backbone of the network, helping to the traffic distribution.

This type of design often benefits from having downlink with smaller bandwidth than uplink so that traffic aggregation is achieved. Also, since most of the times not all users access the network simultaneously, ports tend to be oversubscribed, yielding an oversubscription ratio as part of a specific architecture's implementation specification. Broadly speaking oversubscription, as the word meaning implies, is when more entities subscribe to a resource than the resource can sustain per unit of time. In a network the oversubscription ratio can have many definitions depending on the point of reference. For example switch oversubscription ratio is the ingress ports (ports of incoming traffic) bandwidth over the overall switching bandwidth. As for the network oversubscription ratio, [2] defines it as "the ratio of the worst-case achievable aggregate bandwidth among the end hosts to the total bisection bandwidth of a particular communication topology." This translates into a metric of the bottleneck of the infrastructure when traffic demand is more than the network's capacity. For effective communication, it is important that balance between effective source utilization and potential network congestion is maintained.

Initially DCN were relatively small (a few hundreds of servers) and this type or design was useful to provide enough bandwidth for the end-to-end user communications. As the data center model started becoming more and more adopted at early 2000 the need for more servers and higher communication support was eventually realized. However, enterprise equipment is more expensive and power consuming, while, only

50% of the aggregate bandwidth may be actually provided [2] (due to oversubscription for example) which makes scaling the network inefficient. Not only that, but also the realization over the increasing computational needs of data centers [70] triggered the collective effort of the research community to create better designs. Some important considerations in the process were also agility, fault tolerance, maximum end-to-end aggregate bandwidth and latency.

Aimed to tackle these scalability issues, the Fat tree [2] architecture proposed in 2008 suggests the replacement of enterprise equipment used in the higher layers of the multi-tier model with commodity switches, typically used in the access layers. The key idea utilized over the network's topology is called fat-tree and it was originally proposed for hardware efficient supercomputing [57]. Fat tree is a complete binary tree with the special property that the links used in higher tree levels are "more fat". A way for this to be achieved is either with switches that support links of higher capacity in the direction of the core, or by just connecting the higher layers with more than one links. The later can be optimally realized by folding the non-blocking switching network architecture described by Charles Clos in 1953 [23]. For this reason, architectural structures of this form are also referred as CLOS networks. This specific property results into a network with full bisection bandwidth or in other words 1:1 oversubscription ratio. This over-provisioning is unavoidable when it comes to the unpredictable bursty nature of the traffic that we see in the data centers today. The final architecture at [2] is a multi-routed tree, so that the required capacity is achieved with only commodity switches. An important characteristic of this topology is that there are multiple paths between any pair of end-nodes, thus it is more fault-tolerant. Also the maximum number of hops between end-nodes is 2log(n), where log(n) is the tree height, while for k levels connecting N servers we need at least Nxk switch ports and wires [83]. A small topology instance provided in [2] is presented in Figure 2-1

To maintain fast communication, the authors of [2] propose a customized routing that requires specific device addressing and prefix/suffix lookup table techniques. An example of the addressing scheme is also present in Figure 2-1. This, along with high cabling complexity, ended up to be a barrier for the adoption of the proposed model as

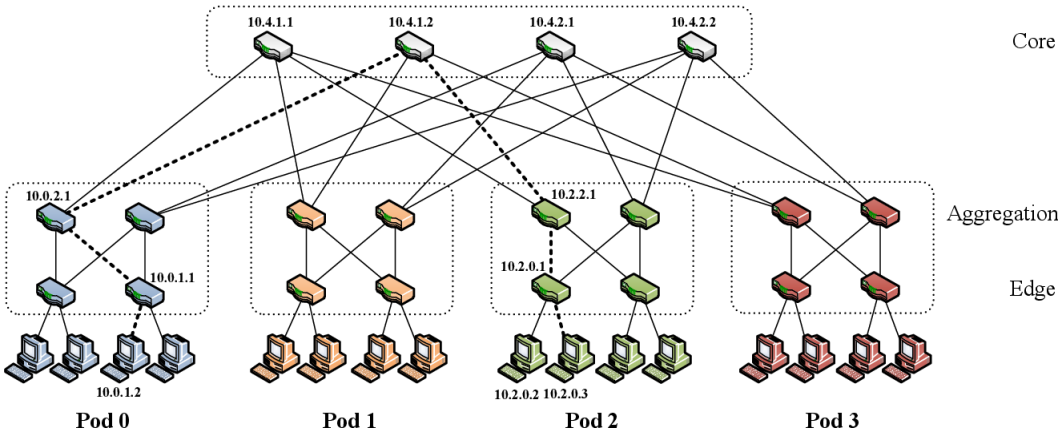

Figure 2-1: Fat-tree small example as provided in [2]

is, because it requires specific configuration for addressing purposes. However as architectural alternative became very popular over the following years, triggering other similar and improved architecture proposals [33] [45], as well as real life implementations. Among others, some of the famous examples of architectures employing CLOS structures in their design is the Facebook data center [32] and the Google data center [81]. Their fast widespread use proved once more the practicality of hierarchical network topologies.

2.3 Optical Technology and WDM

Optical technology has gained a lot of attention the last decade as promising for the upscaling of future data centers [82]. The increased server communication within current data center along with need for more user support, fault tolerance and network robustness, require bandwidth increase in the network links. Optical technologies, such as transceivers, fibers and switches combined with Wavelength Division Multiplexing (WDM) are able to support higher traffic rates and have low latency and better scalability. Specifically, using WDM and Optical Circuit Switching (OCS) enables a rate agnostic network able to adjust to future traffic needs without device change and with less cabling. Finally such networks are more power efficient and thus cost effective.

Despite the long-term cost savings, the initial capital for a network architecture

update are still prohibitive and not all of the optical fabric is characterized by an economies of scale [34]. For this reason there is slow process in the transition to fully optical architectures. On the bright side, there is room for planning ahead and improving optical technologies before the actual change, which, as the networks start getting to their limits, does not seem to be so far away. In this mindset, many papers have already tried to summarize the requirements for effective data center network optical update.

In [82] the authors, about a decade ago, explored the requirements for a transition to full optical large scale data centers based on existing optical technologies. These requirements include the use of OCS with less than 100μs switching time and less than 2dB insertion losses. Similarly in [59] the authors build some requirements around the multiplexing needed to scale up the bandwidth in optical networks, concluding that dense WDM will drive the future. Drawing emphasis on the switching time, critical for algorithm design decisions in data centers, in this section we discuss some of promising optical devices typically employed by modern optical data center architectures along with the benefits of WDM.

2.3.1 Optical fibers and light transmission

When light is directionally projected at an angle through an optical (silica) fiber, it gets reflected and it bounces from the one side of the fiber to the other until it eventually exits the fiber. This light is called a light beam. In this way we can use light as signal carrier and transmit data through optical fibers. This is a simplification of what optical networks are based on.

Optical fibers are an attractive transmission medium since they do not allow for electromagnetic or radio-frequency interference compared to cooper links. Not only that but they also require less maintenance. Optical fibers are made either with plastic or silica glass. There are two types of optical fibers, the single-mode and the multi-mode. Single-mode allows only one wavelength transmission, while multi-mode, due to bigger diameter, allows us to transmit many wavelengths over the same fiber simultaneously. As we will discuss later this enables better capacity provision

opportunities in the optical networks. For this reason their usage extends over the domain of optical networks.

Light communication in a network is enabled through optical transmitters and receivers. In a typical network today there exist both electronic and optical devices. Therefore there is need for the following functionalities: (a) Signal conversion from electrical to optical, (b) Transmission of optical signals, (c) Reception of optical signals and finally (d) Signal conversion from optical to electrical. In networks that employ WDM there exist also the need for (e) wavelength tuning / conversion of the wavelength of a signal, which can happen either at the end nodes so that they are able to transmit in different wavelengths or in intermediate nodes in order to change the wavelength of an incoming signal before it enters a new fiber. These WDM related operations generally dictate how signals are switched or multiplexed in optical fibers and we shall see how they operate in following sections. Details on how all the above are achieved in the physical layer are beyond the scope of this study. However, to realize the optical alternatives when building or analysing optical communications we would like to know their general performance and costs.

Optical transmitters are the first component of our interest. Their basic operation is the transmission of light signals. The signal can be transmitted in one specific wavelength, in which case the transmitter is fixed, or can change the wavelength of transmission through time, in which case the transmitter is called tunable. Finally, transmitters at the end nodes are also responsible for electrical to optical conversion.

The quality and commercialization of a transmitter is based on the way it achieves the aforementioned operations. Able to transmit light in optical communications are both Light-Emitting Diode (LED) as well as laser diode. In large scale optical networks though, laser takes the lead because, due to directionality of light and narrow spectral width, they tend to achieve higher bit rates. In current data centers the most used laser version is the Vertical Cavity Surface Emitting Laser (VCSEL) that has properties such as speed, power and better multi-mode fiber coupling. To be fitted to high bit rates for long distances, VCSEL keeps sending a continuous light that is being amplified and immediately modulated by a semiconductor device called electro-

absorption modulator (EAM) or by Mach-Zehnder Interferometer (MZI). A control module commercialized for this purpose, MZ-SOA, combines MZI with Semiconductor Optical Amplifier (SOA) and is of research interest by itself [71][24].

In the tunable transmitters the tuning happens with a similar MZ-SOA technology. Transmission rates can be up to 160Gbps and tuning time nanosecond and down to sub-nanosecond scale [84]. Important characteristic with key role in WDM is the range of wavelengths a transceiver is able to tune to. For Coarse WDM (CWDM) we have CWDM tunable lasers, which have better temperature control however they are more expensive due to low demand [34]. For Dense WDM (DWDM) we have similarly DWDM tunable lasers which tends to have slower tuning time though for "fine grained" tuning. All those choices are to be considered since they have potentially different tradeoffs between bandwidth enabled and tuning times.

Finally, the common electro-optical converter/transmitter is fabricated on a plug and play interface called Small Form-Factor Pluggable (SFP). SFP are available in a variety of transmitter specifications allowing different uses and transmission rates. Most famous one for data centers is the SFP+ achieving 10Gbps transmission rate. The fabrication of SFP has the receiver included, which is our next point of interest.

Communication is not complete if the optical signal is not received with an optical receiver. An optical receiver, analogously, receives the light beam and converts it into electrical signal. These devices are basically photodetectors and are much simpler. The light is converted into electricity using the photoelectric effect. When the receivers are combined with the transmitters on a device, like in SFP, the combination is called a transceiver. Photodetectors are made out of gallium arsenide whose processing is more expensive and hard to which is unfortunately more expensive to process and cannot benefit from economies of scale [34], thus the same is true for the optical transceivers as well.

There are other key components in the optical communications, not as critical implementation choices though in terms of cost or power consumption. For example many times the signal needs to be amplified, either before it gets modulated or when it needs to travel higher distances. This is what the optical amplifiers take care of.

Another one, which we will also refer to later, is the Reconfigurable Optical Add-Drop Multiplexer (ROADM), which is able to add and drop incoming wavelengths before the multiplexing on an optical fiber. Finally, optical circulator is the device that enables bidirectional communication on optical fibers.

2.3.2 Optical switching

Optical switching is the switching of light beams by optical means and can be performed in packet level, circuit level or mix of both, called burst switching. Typically, Electronic Packet Switching (EPS) has been a paradigm of fast communication in large scale networks, while use of circuits has been critical for time sensitive applications. When we move to the optical domain, Optical Packet Switching (OPS) can be either all optical or partly electronic with use of optical-electronic conversion. When conversion is used, as in the electronic domain, processing in all intermediate nodes is required, in which case we care about the number of hops in routing. A key difference with EPS is the lack of buffers and the different processing. Label processing instead of the typical header processing has been more useful in this case [13]. Compared to EPS, OPS has not been that successful in (optical) networks design. Optical circuit switching on the other hand requires the establishment of a light path between source end destination with wavelength (which acts as the communication channel) routing. Finally Optical Burst Switching (OBS) [89] is a faster alternative of Optical Circuit Switching. Specifically, packets get statistically multiplexed into bursts and , instead of establishing a connection between source and destination before transmission, the destination just gets informed of the incoming burst and the burst gets transmitted. No connection established acknowledgement is needed.

Typically, data center networks have a uniform design and management. Since they belong to one legal entity, devices are located in the same facility and the communication protocols used are unified. Thus, all critical network decisions can be planned and optimized under the specific architectural and usage paradigm the network implements. In this setting it is obvious that uncertainty of traffic or algorithms in use can be eliminated as much as possible. Thus, circuit or burst switching fits

more to this type of networks, that can then effectively experience the benefits of higher throughput and less packet processing. This is important for optical networks that seek to operate in their full potential communication rates. OCS is actually described as data rate agnostic, referring to the fact that we can transmit at the peak rate of the electronic devices and transceivers used (since no other processing is needed in intermediate nodes as in the packet transmission), which means that to increase the capacity we just only need to improve those electronic bottlenecks without changing the way the network operates. This is extremely important for any future up-scaling process. Historically, OCS has been used in some of the most well known hybrid data center network architectures [34][83][72] to deliver the large data flows over a few Optical Circuit Switches.

Optical circuit switching in optical networks can be performed with use of Optical Circuit Switch (OCS) and Wavelength Selective Switch (WSS) that employ Micro-Electro-Mechanical Systems (MEMS), or with use of passive Arrayed Waveguide Grating Router (AWGR). What is especially interesting about these components is that they do not need extra transceivers to operate like electronic switches. This means that the high cost optical transceivers are only needed at the ToR switch, and given that we can add transceivers on demand without changing the other optical elements in a network, the cost is manageable. The different alternatives though of optical switches have their own strengths and weaknesses that need to be considered before network provision.

Optical Switching methods

MEMS use arrays of reflective surfaces (mirrors) to redirect light beams from the input ports to the output ports. There are two types of MEMS technologies, 2D digital or 3D mechanical/analog. The 2 dimensional one is basically a cross bar switch of $\#ports^2$ mirrors, where initially flat mirrors get activated by being placed in 45 angle. Because of the two realized states per mirror, 2D MEMS are also called digital. When no mirror is activated on a port, then the light passes through, a functionality used for adding/dropping wavelengths. Depending on the number of ports and the material

used for the mirror switches, it can have switching times up to 10 ms [25] and as low as 10s of μs [40]. This technology, even though experiencing fast switching times, suffers from light path losses, which effectively means that the more the ports the higher the losses. To provide a perspective, a 16x16 switch has total insertion losses (including the path losses) over 3dB, which is over 1dB increased compared to a 8x8 switch. This makes higher port counts impractical.

To overcome losses and provide a higher port count solution, the second generation of MEMS had a 3 dimensional design. In 3D MEMS there are two mirror arrays facing each other with $\#ports$ mirrors each (total $2\#ports$ mirrors). Light from an input port is directed on a mirror on the first array, reflected on a mirror on the second array until finally is reflected from there to the output port. With continuously adjustable angle mirrors - analog model - and precise alignment the path from input to output becomes way smaller, allowing for higher port counts (> 1000) while maintaining the insertion losses as low as 3dB [25][51]. However, due to precise analog control that they require for the reconfiguration, they typically have slower switching times, at the scale of 10s of ms. For faster switching the mirrors would have to be smaller but then the losses are increased.

Both of those methods are based on light reflection on the mirrors. A dual idea is to use light diffraction instead, changing the amplitude, polarization or the phase of the signal. In this scenario several small elements could be used to diffract the light. A way to use MEMS for light phase change is described in [11] and is promising as a high port microsecond switching element. Other implementations include also Liquid Crystals on Silicon (LCoS), with which we can modulate phase, amplitude and polarization of incident light beam using many pixels per channel. The pixels are coated with aluminum on the silicon backplane and are reflective.A central unit is controlling the voltage on the pixels can adjust their solid state. The switching time, which highly depends on this adjustment, is relatively slow (up to 100ms). A new material, polymer stabilized blue phase liquid crystal (PSBPLC), is more promising for phase modulation, having a sub-millisecond switching latency. Finally, grating and Semiconductor Optical Amplifier (SOA) have been also used for switching, presenting

nanosecond reconfiguration [74].

Wavelength Selective Switching

Wavelength Selective Switch (WSS), which can be implemented with any of the switching methods described, is a 1xN switch that splits the wavelengths in its input into any N subsets and transmits them to N outputs. Any "leftover" wavelengths are just dropped of the switch. This makes the device suitable for ROADM as well, called also in this case eROADM. WSS can be built with either 3D or 2D MEMS, with the later providing switch speeding orders of magnitude smaller than the former [72]. There also exist commercialized implementations of WSS based on LCoS [8] or SOA [74]. These devices are particularly useful in architectures where a Rack connects to N other Racks, since they allow adjusting the link capacity to each of the N neighbors on demand.

Arrayed Waveguide Grating

Another type of optical technology, Arrayed Waveguide Grating (AWG) achieves contention resolution in the wavelength domain. Each input receives the combined signal of k wavelengths, which is then transmitted to k different length waveguides (fibers). Due to this difference in length the input signals are phase-shifted enough by each waveguide so that each signal is constructively recombined only at one of the output ports. AWGs can be constructed with different materials, among which are silicon (Si) and silica $(SiO)_2$ [56], whose use is more standardized thus the commercialization of such is easier and so is economies of scale [34].

AWGs can be used in the forward direction as demultiplexers, in the opposite dirrection as multiplexers and have been a useful component for the creation of tunable transmitters and receivers, and most importantly for our discussion, optical circuit switches called Arrayed Waveguide Grating Router (AWGR). An MxM AWGR is constructed with M AWGs, allowing the use of M different wavelengths for data transmission. The routing of wavelengths is performed in cyclic manner which allows multiple inputs to reach simultaneously the same output by using different wave-

lengths. In this way AWGRs can realize a complete topology graph. These routers are described as passive devices with zero power consumption that provide loss-less optical connectivity. The scalability of AWGRs is small due to limited number of ports they can support, so to scale up a network, combination of many AWGRs is required. However, to operate, they need to be combined with optical transceivers, who are expensive and power consuming. Thus when extensively used in network architecture design, there are trade-offs between the benefits of their passive nature compared to the increased need for transceivers and wavelength planning. Currently in the market there are 32x32 AWGRs already available [30], while more complex 128x128, 256x256 [90], 270x270 [78] and 512x512 [21] demonstrations also exist in literature .

2.3.3 Wavelength Division Multiplexing

Multiplexing in communication networks refers to the combination of signals into one to be transmitted over a shared medium. Typical ways of multiplexing signals is through time, frequency or code division. In the optical domain the information carrier is infrared light as an electromagnetic wave, while optical fibers are the transmission medium. Wavelength Division Multiplexing (WDM) is the multiplexing of optical carrier signals (light) of different wavelength over the same optical fiber. This technique can be realized as the optical alternative of frequency division multiplexing.

Optical fibers have two low-loss wavebands of widths 100 nm and 150 nm respectively [64]. The granularity in which we are able to distinguish between different wavelengths in this spectrum defines how many wavelengths can be multiplexed within a single fiber. This technically depends on how stable optical tunable transceivers are. If for example a transceiver varies the wavelength it is supposed to tune to by 0.5nm, then 1nm wavelength spacing is needed to distinguish between the different wavelengths. Tunable transceiver components discussed in 2.3.1 are able to realize over 128 wavelengths. The more the channels available the higher bandwidth available per fiber. Depending on the number of multiplexed wavelengths WDM is referred as Coarse WDM (CWDM) or Dense WDM (DWDM) with the naming convention

inherited to the transceiver categorization. Finally the capacity that a wavelength channel provides depends on the peak electronic processing speed and transceiver transmission rates achievable.

This last remark indicates that we might not be able in the near future to increase those channel capacities. However, through multiple channels multiplexing using WDM we can aggregating many flows together. Intuitively this means that we might not increase capacity per channel but we can significantly increase the number of supported connections. Thus, with demonstrations of 40Gpbs [66] optical transceivers or even 100Gbps achievable data rates over a single wavelength [22], networks that leverage WDM can comprise of millions of servers and support a scale over terabits per second bandwidth (for example with a 128 wavelengths and 40Gps per wavelength can support over 5Tbps)

The great benefit of WDM is closely related to the optical transmission characteristics and really depends on the enabling optical components. For example WDM can be used to set up any virtual topology over connected physical topologies if properly combined with transceivers and ROADMs [63]. Specifically, given the traffic pattern we can create an optimal virtual network which is going to stay fixed until we tune the transceivers to create a different virtual topology. This can be very useful traffic control mechanism for routing purposes if for instance we want to implement load balancing [67]. The applications thus are far more than just traffic multiplexing. Also, WDM is more practical compared code and time division multiplexing, since to synchronization is required [64].

When it comes to routing in WDM networks we have two alternatives, single-hop and multi-hop [65]. In single-hop configurations we require that the communicating nodes are connected through the same wavelengths. This means that when a node a wants to transmit to a node b, the transmitter of a and the receiver of b need to be tuned to the same wavelength. For this setting to successfully support the network's traffic a lot of coordination is required between the nodes and the transceivers used must be able to tune fast. However the transmission between a pair of nodes avoids any intermediate processing and it stays fast even if the traffic travels a big path on

the physical topology.

On the other hand, multi-hop is the hop by hop stitching of an optical communication path. Therefore as long as neighbor nodes are tuned to the same wavelength, the wavelengths along the path need not to be the same and the routing is executed hop by hop. In this case, optical links can stay fixed for a period of time so no fast tuning is needed. Instead, we need to have fast per hop processing to enjoy a high-speed network and prefer to build minimum average hop length (similarly minimum max flow) topologies. When traffic is relatively uniform, structured topologies that form a regular graph have been successful in meeting both of these goals since they can have controlled average path length and simplify routing. When the traffic is skewed and arbitrary, the regular graph solutions might be susceptible of congestion, thus offloading solutions or topology optimization and reconfiguration might be needed. In all cases, buffering at the intermediate nodes is not so attractive since it requires optical-electronic-optical conversion and is going to slow down the optical network's speed.

Summing up, if we wish to design fast, high aggregation networks that are power and cost-efficient in the long run, optical networks are an inevitable choice. Especially WDM enables most of the benefits we can get from optical networks, as for example higher data rates due to traffic aggregation. When passive devices are combined with reconfigurable switches and tunable transceivers that enable WDM then indeed networks are able to operate in their full potential. However, optical devices can present many trade-offs that we need to acknowledge when designing a new optical network architecture. Our discussion continues with a study of Optical Network architectures to unravel such critical design choices.

2.4 Optics in Data Centers

In the previous section we briefly discussed about the increasing need for optical components in DCNs. This has been understood already the past decades resulting in many proposals about this transition. To fully transition into an optical DCN though,

huge percentage of the infrastructure already existing would need to be replaced. Of course this is not backward compatible neither economically feasible yet, due to high costs of optical devices compared to their electronic alternatives. For this reason emerging architectural designs that consider optical devices are either full optical or hybrid in an effort to balance trade-offs.

Hybrid Architectures

Hybrid architectures aim to combine benefits of both the well established electronic packet switching and optical circuit switching alternatives. First of all, typically they require less optical components, making a transition easier due to less network changes and smaller costs. Technically speaking, with addition of few transceivers and few Optical Circuit Switches we can increase the bisection bandwidth of the network without altering the electronic parts and incurring only small increase in cost compared to better electronic alternatives, as for example better Ethernet cables [83]. What is more, this approach allows the coexistence of both EPS and OCS, something attractive in the networks history as the ATM protocol proves [68]. Data centers, that most of the time tend to be underutilized [9], using a hybrid architecture, can achieve good network performance without network over-provisioning for full bisection bandwidth, which would inherit higher complexity and costs [34]. It has also been noticed that full bisection bandwidth is less needed at packet granularity [83].

Trying to make the most of it, the proposed architectures often focus on two aspects, the network traffic demands calculation and the choice of the right flows to circuit switch. As a general observation, since optical switching is slower than electronic switching it mostly fits to large long-lived (elephant) flows [83].Two examples of hybrid architecture are Helios [34] and c-Through [83].

Helios, is based on the idea that it is better to circuit switch only bigger flows while keep packet switching the small bursty traffic. The goal is to achieve similar performance with a full bisection bandwidth network but with less cost (proved 3 times less), complexity (proved 6 times less based on cabling) and power consumption (proved 9 times less). For this to be achieved optical circuit switches based on MEMS

are used along with topology reconfiguration. The full pipeline of Helios contains the flow estimation and optimal circuits calculation, that can take over 100 ms, and finally the network reconfiguration which is again around 100ms. These time results, which tend to be increased with the network scaling, show that indeed the algorithms and the switching time in reconfigurable designs can generate several trade-offs in the network performance.

The spirit of c-Through is not very far from that of Helios. Basic differences between the two is the flow size calculation and the traffic dispatch. Helios' flow calculation algorithm is adopted from [3] and uses the flow counters immediately available at the switches. This algorithm as well as the optimal circuits algorithm run on a network server. On the other hand c-Through moves all the complexity of demand estimation and traffic demultiplexing to the end nodes. In this way there is no need for customization of the switching components (to retrieve counters and communicate with a server for example, as Helios would require). For the flow estimation the data are queued an a TCP socket buffer. This allows both the byte calculation as well as head of the line blocking avoidance. Suggested alternatives also contain the explicit statement of demand requirements from the applications.

The authors of the two papers disagree on whether or not end hosts should be part of the network control. Specifically the authors of Helios find the host buffering increasing latency, while the authors of c-Through believe it is essential for better system scaling. The later is also supported by Helios results on scaling, which state that as the network grows the time it takes to access all the flow counts for flow calculation estimation increases as well. Both sides though believe that the circuit switching matches more to large time-insensitive long-lasting traffic and that hybrid architectures can improve the network with small additional cost.

Optical only Architectures

Designing networks from scratch, typically creates opportunity for optimization of every aspect of the new design. When conversions of signal within a network can be avoided we can have both decrease in latency and in power consumption. In this

perspective a network operating completely in the optical domain is preferred.

Contrary to the hybrid architectures, the construction of a full optical architecture can be also combined with a new different perspective about data centers. Mainly, optical components can be so fast that we can treat the full architecture as a big, scalable and fast switch, a process mentioned also as network flattening [84][85]. For example Petabit [84] views an optical CLOS constructed from AWGR switches as a huge input-queued switch, due to flow transmissions that do not require any inter rack buffering of packets. Some of the promising features of this design is the reduced latency, power consumption and cabling complexity. Notice that passive optical devices are very attractive choice for fast cost-effective solutions. Another showcase, [85] presents a switch comprised by a unique AWGR. Approaches that treat the network as a switch generally tend to describe in detail all the routing and forwarding algorithms, while they care about fabrication of architecture parts on a single unit/component. This is a promising view of the optical networks however misses a lot of flexibility due to lack of abstraction.

This perspective started being revised few years later. Several proposals started realizing that optical networks have a more flexible and adjustable nature. First of all, components like WSS and OCS help us build architectures that can dynamically alter the physical topology of the network. Not only that, but also WDM allows flexible dynamic capacity provisioning. In other words we can change the link capacity dynamically or structure a new virtual topology over the physical one. This is how network reconfiguration ideas got born.

Once this opportunity was realized, many research groups got dedicated to design reconfigurable architectures and evaluate all the different alternatives. The first idea of both topology and capacity reconfiguration was presented in [18], which we will discuss more later. Briefly though, it is a fully reconfigurable architecture where MEMS takes the place of a central switch. The first big debate in the literature was whether and to what extend MEMS were actually useful, since their per port cost is relatively high. Other components, like AWGRs for example, seem to be way more efficient in terms of power consumption and cost. First paper to oppose the idea

of optical switches was Wavecube [19]. Using exactly the aforementioned reasoning, [19] suggests interconnection of ToR switchs immediately and without intermediate switch layers, in order to form flat, n-dimensional cube network graph. Only WSS are used for link capacity allocation flexibility, which makes the architecture dynamically reconfigurable, at least partly. Similarly following a flat architecture design, OPSquare [88] uses AWGRs to interconnect the ToR switchs instead and it does not support capacity allocation flexibility. However, the number of interconnected ToR switchs scales as the square of the optical packet switches port count (OPSquare) and the latency is less than $2\mu s$.

Even though described as flat, the structure of OPSqure entails a two level hierarchy and cluster construction that make it more flexible when it comes to routing. Actually, the hierarchical design can realize better scalability. The architecture called PODCA [86] is another successful example of hierarchical networks based on AWGRs. According to the authors it is able to support 2 million servers with latency as low as 9μs and 100%throughput, which is impressive. A drawback however is that the topology is static as well as the link capacity, which meas that it should better be combined with load balancing techniques. An interesting part for one to notice is that even though AWGRs are passive, for an effective optical network we will need increasingly many transceivers as also [84] suggests. This means that the costs are not as low as we would wish.

Soon it became clear that in order to enjoy faster networks a combination of those approaches need to be done. Specifically, as hybrid models suggest, we can definitely benefit by dynamically "following" the traffic. At the same time topology reconfiguration and capacity adaptation enable these network dynamics. Finally, for scalability and lower latency purposes, the use of some hierarchy in the designed architecture becomes important. We seek to study a combination of all those ideas. First we will describe a flat architecture of similar goals. In the next section, the architecture of interest is described.

Reconfigurable (all) Optical Architecture

One of the first fully reconfigurable attempts in this area is OSA [18], an optical switching architecture. OSA, depicted in Figure 2-2, leverages OCS and WDM to dynamically adjust the network topology to traffic demands. Connecting each ToR switch in k ports of a MEMS switch is able to realize all k-regular graph topologies between the ToR switchs, with key optical component a WSS per ToR switch that through WDM enables capacity variation of the network links. Through the flexibility of the optical paths capacity it is able to adjust better to skewed traffic patterns or hot-spot traffic with relative to k number of hot-spots, without the need of over-provisioning. Main difference with hybrid or other all optical solutions is that it still packet switches the traffic instead of transmitting flows. Finally it is shown that this fully reconfigurable alternative has higher bisection bandwidth compared to hybrid alternatives.

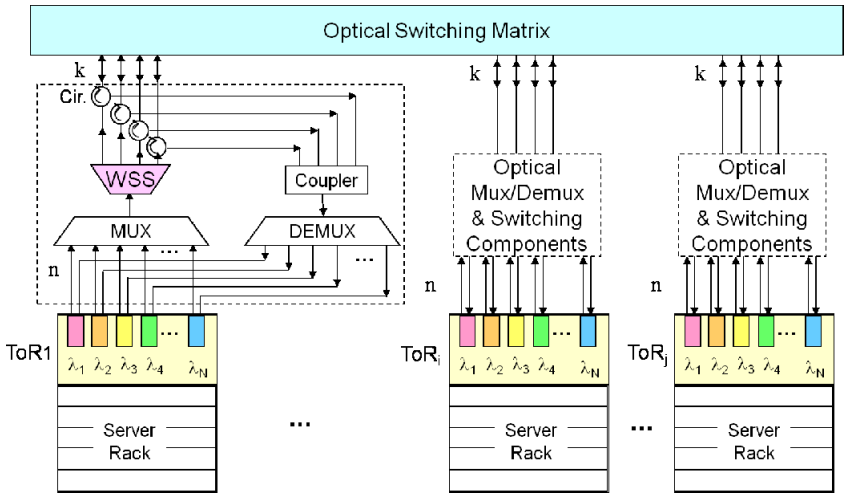

Figure 2-2: The OSA as presented in [18]

There are two limitations in this design. The first one is the scalability of the model. In the design the basic topology configuration is done by a central MEMS switch. This means that the size of the network is limited by the maximum number of available ports on the switch. Notice that as a network grows not only a higher port count is required but also the realizable k-regular graphs need to be of higher degree k for the model to operate optimally, which means that the network size might

not be able to linearly grow with the port count of the switch. The second one is the switching time. We already have discussed that MEMS take a considerable time to reconfigure compared to the mice flows lifetime. In the suggested implementation the running time is 10s of milliseconds scale, while the running time of the reconfiguration algorithms alone is almost 300ms. The reconfiguration seems to mostly affect mice flows and addressing the issue, the authors discuss potential solutions like immediate ToR switch to ToR switch connectivity for mice flow transmissions or a fixed preexisting path that does not change. These alternatives however are closer to the idea of the hybrid architectures. Notice that in the hybrid case the electronic part is still functional during reconfiguration, which means that no traffic disruption is needed. Since goal of this work is to prove the potential of the full reconfigurable designs, both the scaling and faster algorithmic alternatives for the reconfiguration of the network are left for future research.

Even though research has already taken the action towards creation of more flexible, traffic aware, adjustable networks, not many past works have realized the extend to which reconfigurability is leading or not the future path of data centers. OSA is the preamble of an architecture with central role to this work. RHODA, a Reconfigurable Hierarchical Optical Data center Architecture, with the ability to alter the network clusters (PoDs), is described in the next section (2.5).

2.5 Rhoda: A Hierarchical WDM-based Scalable Data Center Network Architecture

A very novel all optical hierarchical architecture was proposed last year [87]. It is an optical architecture that, as many, realizes the benefits of a hierarchical tree type structures that have been employed successfully by big data centers for years, while offering a new optical dynamically adjustable alternative. Aggregation has been found both reasonable and valuable for bursty traffic, especially in optical data centers where switching times are higher [34]. Thus, we can benefit by forming clusters of Racks

in two ways: big flows are transmitted in low latency high capacity clusters while the small bursty traffic is gets aggregated and then transmitted in the inter-cluster network. Of course, for such an architecture to be effective we need a way to place high communication nodes in the same or close by clusters. One way to achieve this is through job scheduling or virtual machine migration. What RHODA proposes instead is a reconfigurable design able to dynamically adjust to any type of changing traffic. The structure of RHODA is summarized in Figure 2-3

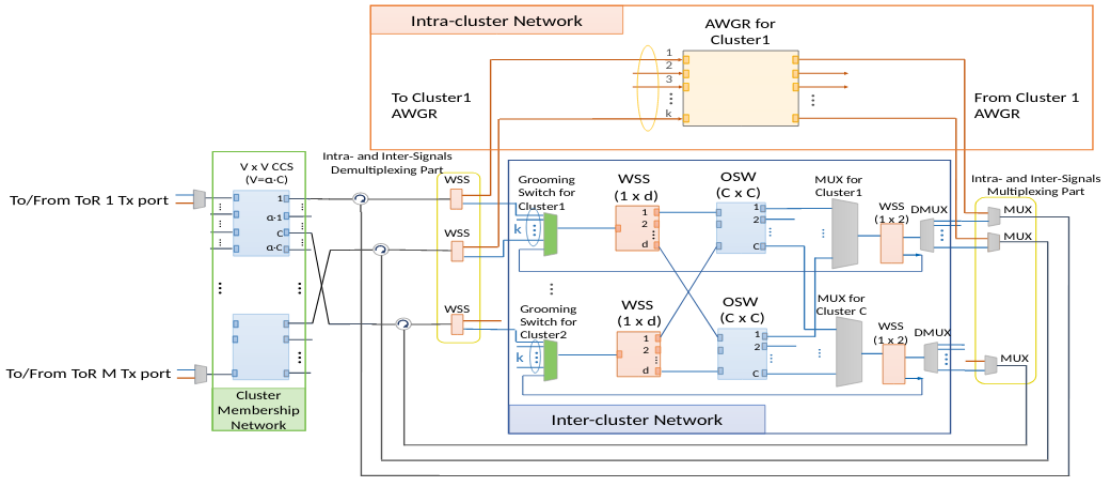

Figure 2-3: RHODA as found in [87]

RHODA has two levels. In the first level Racks form clusters of size k each. The second level is responsible for the clusters' interconnection. The design of those two levels consist of five parts, the intra cluster network, that describes the connectivity of the nodes within a cluster, the inter cluster network, that describes the clusters' connectivity, the cluster membership network that manages the cluster membership of the Racks and finally two parts for demultiplexing and multiplexing the optical signals to be transmitted. All parts are carefully designed so that the architecture can be easily extended and scalable. Main components are AWGRs to form the high communication clusters, MEMS to offer topology reconfiguration and finally WSS for flexible link capacity. We shall then describe the parts in more detail.

Initially, T wavelengths from each ToR switch get aggregated and connected to the cluster membership network (CMN). The CMN consists of Optical MEMS switches

that describe the cluster membership of the ToRs. There is one input for each ToR and the outputs are able to realize a permutation of the inputs. Then each output is fixed to forward the traffic to a specific cluster. To alter the cluster membership we just need to reconfigure the switches and obtain a new permutation of the ToR inputs at the outputs.

The traffic then moves to the demultiplexing stage. There a set of WSS, each dedicated to the traffic coming from a specific CMN output. The WSS splits the incoming wavelenghts in two sets, the inter cluster traffic set and the intra cluster traffic set. The two sets get then forwarded to their respective part.

Each cluster network receives exactly k of the aforementioned intra cluster traffic sets. An AWGR is responsible for supporting the intra cluster connectivity, typically able to realize a complete graph of the connected nodes. We can assume for now that this is the case.

The traffic that gets forwarded to the inter cluster network gets connected to the first substage, a grooming switch, which basically restructures the transmitted flows so that proper aggregation is performed. There are three more substages. Briefly, the first part performs capacity allocation to the inter cluster links, while the second is responsible for setting up a d-regular inter cluster network topology. The third is the aggregation substage that collects and aggregates all the traffic for a specific cluster.

Finally the flows need to be sent to their destinations. This is achieved in the following way. First the traffic for each cluster is decomposed to traffic for each rack. Then those flows get combined in the multiplexing stage with the respective flows from the AWGR output. The aggregated flows then moves to the circulators that make sure to deliver the traffic back, through the cluster membership network , to the respective Rack.

Intuitively, the inter cluster network is nothing else but a scalable OSA design. Generally there are many details regarding the scaling factor of the system that depend both to the number of switches used as well as the number of trasceivers. These details are omitted here but can be found in [87].

For the network to optimally adjust to the traffic we would have to perform a joint

cluster membership assignment and routing and wavelength assignment optimization (an ILP for further insights is provided in Appendix B). However this is NP-hard problem, thus we decompose the problem and seek suboptimal solutions. Specifically, instead of solving the joint problem we can start by defining an optimal based on the traffic clustering of the racks, then generate an optimal intra cluster routing solution and finally solve the inter cluster routing problem on the aggregated clusters' traffic.

In this work we care about the first part: How can we form an optimal Rack clustering, and even more, how can we dynamically change it in an optimal way so that it follows the fast changing traffic patterns? When reconfiguration should be performed and what are the limitations of dynamic clustering? The next chapters investigate this problem in detail.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 3

Traffic Based node clustering

An important part of the RHODA Architecture that makes it particularly attractive is its ability to alter the clustering of Racks. This enables traffic based dynamic reconfiguration which would allow traffic localization contrasting methods such as dynamic job scheduling, virtual machine placement etc. Compared to other approaches that dynamically schedule flows and naturally realize pairs of communicating nodes,as for example [72], RHODA takes a step further to be able to dynamically realize groups of communicating nodes, giving a more stable over time alternative for distributed job efficient execution. We seek to model and study the cluster reconfiguration possibility of this architecture, which would potentially provide us with an architecture fully self adjustable to traffic demands. Treating server Racks as the smallest building block before traffic enters the optical domain, we create the *network node* abstraction, which represents a Rack of around 40 servers. We define the traffic based node clustering for the RHODA architecture as follows:

Definition 1 (Traffic based node clustering) *Consider a set of network nodes V of size N and an $N \times N$ traffic matrix D, where d_{uv} is the rate of the traffic flowing from node u to node v, with $u, v \in V$. Find a clustering of the nodes in sets of fixed size r, such that the aggregated traffic rate exchanged between clusters - inter cluster traffic - is minimized.*

The choice of this definition follows from the architecture itself. AWGRs have

fixed size k and provide up to all to all connectivity between the nodes they connect. Therefore we seek to find equally sized groups of nodes. Given that, there are certain points of interest. Since generally k is way smaller compared to the number of nodes N, the aggregated traffic that clusters exchange - inter cluster traffic - is expected to be way more than the traffic exchanged between nodes in the same cluster - intra cluster traffic - and also travel more network links. Specifically, for each cluster we can have up to k^2 simultaneous directed flows internally, and up to $2k(N-k)$ flows to and from other clusters. Therefore, for the traffic to be sustainable, we aim to aggregate the smaller flows of traffic and deliver them through the inter cluster network, while localize larger flows by placing them in clusters. This makes the goal of our optimization apparent. For lower chance of congestion in the inter cluster network, we could also further require low inter cluster link utilization. Similarly balancing the total amount of intra cluster traffic between the clusters, we can ensure that no cluster is overloaded. These are all valid choices, however, the later two highly depend on the size of the clusters, the traffic characteristics, the amount of wavelengths available etc. We find the first goal, i.e. inter-cluster link utilization, more universal and primarily focus there, however alternatives for the other cases will be also addressed.

Based on the requirements for the success of reconfigurable optical data center architectures, this should be done in a timely manner to actually benefit the data center operation. Ideally, our solution will perform millisecond scale changes that are able to track the fast changing traffic with the minimum network disruption. We make two important observations. First, the problem we seek to solve belongs to a big family of problems known with the name Graph Partitioning problems. These types of problems are graph problems that arise many times in areas that deal with network forming structures, such as communication networks, computing systems, biological networks etc., as well as in situations that clustering (partitioning) is required, like VLSI partitioning, image segmentation and many other applications. This reveals a big area of matured research in which we now search for a good practical approach. Second, the case of data centers specifically, the clustering of the network nodes that

we describe has similar goals with job scheduling, virtual machine placement and migration and all the algorithms that aim to localize computation in data centers. Therefore, the performance of our system should be comparable to these methods or present some potential to act as complementary solution.

In this chapter we explore more in detail the nature of this problem. With the basic requirements of the data center world in mind, we aim to define our goals and expectations. Section 3.1 walk us through the problem of Graph Partitioning, shedding light to the complexity, optimization limitations, while section 3.2 presents the algorithm design alternatives available. Finally, in section **??** we introduce the static node clustering model formulation and describe our basic algorithmic approaches.

3.1 Graph Partitioning

Graph partitioning describes a general class of problems that, given a graph G with set of vertices V, partition V into mutually exclusive groups of nodes, called parts, clusters or communities. Depending on the objective of the partitioning as well as the number and size of partitions, graph partitioning can be realized in many variations. It has been a famous technique for simplification of graphs analysis and has many applications in the areas of networks, parallel computing, biology, image processing and many others. Some of the most well known are scientific computing, VLSI circuit design [5], task scheduling in multiprocessor computers [15] and more recently cluster detection in big social and biological networks. What is more, in many cases, problems benefit from graph partitioning even though they do not impose any natural network structure, as for example clustering of data or image processing. In the area of communication networks specifically, minimum network bi-sectioning and all min cut related problems, which also relate to bisection capacity discussed in Chapter 2, have been of great interest for years. With a central role in this study, graph partitioning along with its many variations and their complexity is the focus of this section.

Definition 2 (Graph Partitioning) *Consider a graph $G = (V, E)$, where V is the finite set of vertices and E is the set of edges and some $B \in \Re$. A partitioning P*

of G is a collection of subsets $V_1, V_2, ..., V_k$ of the vertices such that $\bigcup_{i=1}^{k} V_i = V$ and $V_i \cap V_j = \emptyset,\ \forall i, j \in \{1, 2, ...k\},\ i \neq j$. It is called r-bounded if $\forall i \in \{1, 2, ...k\},\ |V_i| \leq r$ and has cost $C(P)$ equal to the number of edges joining nodes of different parts V_i. The ***graph partitioning problem*** *is to decide whether there exist an r-bounded partition P of G such that $C(P) \leq B$.*

Following this definition, a more general description of the problem provided in [44] is:

Definition 3 (General Graph Partitioning) *Consider a graph $G = (V, E)$, where V is the finite set of vertices and E is the set of edges and some $B \in \Re$. Assume now weights $w(\nu) \in Z^+$ for each $\nu \in V$ edge and $l(e) \in Z^+$for each $e \in E$. A partitioning P of G is a collection of subsets $V_1, V_2, ..., V_k$ of the vertices such that $\bigcup_{i=1}^{k} V_i = V$ and $V_i \cap V_j = \emptyset,\ \forall i, j \in \{1, 2, ...k\},\ i \neq j$. The* ***general graph partitioning problem*** *is to decide whether there exist a partition P of G such that $\sum_{\nu \in V_i} w(\nu) \leq r, \quad \forall i \in \{1, 2, ...k\}$ and, if $E' \subseteq E$ is the set of edges with endpoints in two different parts V_i, then $\sum_{e \in E'} l(e) \leq B$.*

Before we continue, to make our following discussions easier, we can call the edges that connect nodes of different partitions *cut-set*. Typically this term is used only when we partition our vertices into two sets of nodes, the cut. In the general case we will have more than two partitions, but the term can still serve our intuition.

Definition 4 (Cut-set) *Consider a graph $G = (V, E)$ and a partition $P = \{V_1, V_2, .., V_k\}$ of the vertex set V. We call generalized* ***cut-set*** *the set $S := \{e = (u, v), e \in E : u \in V_i, v \in V_j, i \neq j\}$. Suppose weights $l(e) \in Z^+$ on every edge $e \in E$. The cost of the cut-set is $cost(S) = \sum_{e \in S} l(e)$.*

The general graph partitioning definition can act as a template to generate many different variations of the graph partitioning problem. Here we define it as a decision problem that the complexity discussion mandates. However the problem can be realized as the minimization of the cut-set cost under structural conditions, such as

the partitions size inequalities. More abstractly, graph partitioning aims to form groups V_i so that nodes within a group are more similar and well connected to each other than to nodes of other groups. So, with central part of the definition being the nodes partition, various objectives and constraints can achieve similar goals. For the purposes of our study we mainly consider problems with unit weighted nodes and positively weighted edges that aim to minimize the cut-set cost.

Many times the resulting partitions may be unbalanced, especially when the bound r is very large, which can be undesirable in some applications. For this reason we may also form some constraints to require some balance in our solutions. Since, balance is intuitively a way of asking the clusters to be somehow similar to one another, it can take many forms. Some examples of balance enforcing constraints, with most typical the first one, are:

- Equally sized parts: $|V_i| = N/k = r,\ \forall i \in \{1, 2, .., k\}$
- Equally weighted parts (vertices): $\sum_{u \in V_i} w(u) = b, \forall i \in \{1, 2, .., k\}$
- Equally weighted parts (edges): $\sum_{e=(u,v):u,v \in V_i} l(e) = b, \forall i \in \{1, 2, .., k\}$

Notice that we might also create more relaxed formulations using inequalities or less tight bounds etc. Depending on the application requirements we instantiate our graph partitioning problem accordingly.

Complexity of the Graph Partitioning

Some instances of graph partitioning have been very famous problems by themselves and their complexity varies. As observed in [47] if r is 2 then the problem becomes the Maximum (weighted) Matching problem, for which an algorithm of running time $O(|V|^2|E|)$ was described by Edmond (blossom algorithm) in [29]. Similarly, for r equal to n-1, we force a partition (at least 1 node has to be separated in a different partition), however we allow the partitions to be as flexible as if we did not have any constraint on their size. Therefore this instantiates a Minimum Cut problem, which can be solved using a network flow formulation as shown by Ford and Fulkerson in

[41], with a time complexity $O(|E|f)$, f being the maximum flow in the graph. Both algorithms are polynomially achievable.

The aforementioned problems can be thought as special cases of the general formulation. Contrary to those however, as proved in [47], the Graph Partitioning problem is NP-Complete. Summarizing the proof here, we can basically reduce the 3-satisfability problem (3SAT) to the Graph Partitioning. Suppose a graph with 3r vertices, where r is the number of the clauses, which has an r-clique if and only if the clauses are satisfiable. This is the graph Karp suggested to reduce clique to 3SAT, in which each clause's literals are represented by nodes and each two non-contradicting literals from different clauses are connected with an edge. We can now add a (r-2)-clique for each of the clauses and connect each node of the clique to all the node-literals of the respective clause. Using this new graph we can then prove that the Graph partitioning problem with $B = |E| - (r+1)\binom{r}{2} + r$ has a solution if and only if the set of clauses is satisfiable, meaning that an r-clique exists, which is actually going to form a part by itself in the optimal partition. Therefore, using the Graph partitioning we can solve the satisfability problem by transforming it into this special type of graph. Since 3SAT is NP-Complete though, we cannot have a polynomial time algorithm for it, thus the Graph Partitioning must be NP-Complete as well. As in the perspective of [44], the problem is NP-Complete for fixed $K \geq 3$ even if all weights are 1.

Minimum Bisection

Due to its combinatorial nature and big search space, many of the other problems in this family have been proved hard as well. Among those, in the scope of our study, is also the Minimum Bisection Problem as described in [52]:

Definition 5 (Minimum Bisection) *Consider a graph G = (V, E), where V is the finite set of vertices with $|V| = n$ and E is the set of edges. For simplicity let n be even. Assume also weights $l(e) \in Z^+$ for each $e \in E$. For a subset S of the vertices let $\overline{S} = V \setminus S$. We define as cut$(S, \overline{S})$ the set of edges with one endpoint in S and one*

in $\overline{S}$. *The* ***minimum bisection problem*** *is to decide whether there exist a set* S *such that* $|S| = |\overline{S}| = \frac{n}{2}$ *and* $\sum_{e \in cut(S,\overline{S})} l(e) \leq B$.

The term bisection comes from the equal size requirement of the problem. As one can see this is a graph partitioning with k = 2 partitions and r = n/2, equivalently n/k. This is a nice way to actually force all the partitions to have equal size. In [43], Garey, Johnson and Stockmeyer prove in that min bisection can be reduced to the Max Cut problem. Max Cut is the maximization version of graph partitioning with $r = n - 1$, i.e. Min Cut, and it is one of the first problems to been shown NP-Complete. Thus Min Bisection is NP-Complete as well. However, for special graphs there have been positive results. For example grid graphs without holes can be bisected in $O(n^4)$ time [38], while in trees time reduces to $O(n^3)$ [61]. What is particularly interesting is that in certain cases the problem gets hard to approximate as well. Even though an $O(\log n)$-approximation exists [73], in the general case there is no PTAS for the problem [50]. The only found PTAS is for planar graphs [26]. Also, no constant approximation can be achieved in polynomial time [36]. This size restriction on the partitions make the problem noticeably harder.

β-balanced cut

On another perspective, if we only focus on the cut, i.e. 2 partition nature of this problem, we ca view it as a special case of a cut problem formulation named $\beta-$**balanced cut**. This problem, letting everything else the same, relaxes the set size constraint by asking for $\max\{|S|, |\overline{S}|\} \leq \beta n$, where $\frac{1}{2} < \beta < 1$ [52]. Sometimes the bound is also denoted as $(1 + \epsilon)\frac{n}{2}$. Solutions of this form are also called near-balanced and the goal is to have as small ϵ as possible. The special case for $\epsilon = \frac{1}{3}$, or $\beta = \frac{2}{3}$ is called **edge-separator problem** and it has been a great tool for graph partitioning algorithm design. In[55] an $O(\gamma/\epsilon) = O(\sqrt{\log n}/\epsilon)$ approximation ratio is shown for the near-balanced cuts. It is based on γ-approximation of the sparsest (also called quotient) cut, the min cut problem with objective to minimize the ratio of the cut cost to the number paths that cross the path ($\frac{cost(S,\overline{S})}{|S||\overline{S}|}$).

Due to the hardness of minimum bisection in the general case, there has been an effort to approximate the optimum using its connection to the $\beta-$balanced cut problem. For instance, [35] approximate the minimum bisection by combining small graph parts obtained by a divide and conquer graph decomposition. The later requires cutting the graph into two parts and recursively proceed to the obtained subgraphs until the small parts are created. Of course, for the recursion depth to be small and for the recombination easier it is better to obtain balanced subgraphs (not necessarily equal) in every step, so this is where the $\beta-$balanced cut comes into play. The algorithm is a bit more complicated than what was just described. Specifically the cuts need to be done in such way that the cost of the cut in every recombination is as small as possible. Another way to think about this is that the heavier edges should be "hidden" in every combination step. This can be achieved using some cost amortization schemes and performing what the authors call "amortized cuts". Effectively, for this process an approximation for the min quotient-cut problem (or sparsest-cut uniform weight) is used as a black box resulting an overall $O(\gamma \log n)$ approximation factor for the minimum bisection, where γ is the approximation factor of the quotient-cut and n is the number of nodes in the graph. For the latter, best approximation ratio for general graphs is $O(\sqrt{\log n})$, thus giving an $O(\log^{1.5} n)$-approximation for the minimum bisection. For graphs excluding a fixed minor[1] (which includes planar graphs) $\gamma = O(1)$ thus we have a special case approximation $O(\log n)$. This is not the only example. Other approaches, like the one in [55] aim to find $\beta-$balanced cuts with cost guaranteed to be small relative to that of a minimum bisection. This is called *bicriteria approximation* or *pseudo approximation*, since not they are approximations with respect another parameter, in this case β.

Apart from the approximation efforts, $\beta-$balanced cut methods can be useful when we prefer somewhat balanced cuts, but we are not strict about the size of the partitioning. The idea that can be used in this case and that all those approximations employ is to allow unbalanced cuts but penalize them through an engineered

[1] An undirected graph H is called a minor of the graph G if H can be formed from G by deleting edges and vertices and by contracting edges.

objective. The amortized cuts we described were build in this perspective. In this way the problem can maintain the nice min cut complexity, prioritizing balanced partitions more. Some examples of objectives found in [62] are a) Expansion: $\frac{E(S,\overline{S})}{\frac{|S|}{n}}$ or $\frac{E(S,\overline{S})}{\min\{|S|,|\overline{S}|\}}$ (a.k.a. ratio-cut) b) Sparsity: $\frac{E(S,\overline{S})}{|S||\overline{S}|}$ (a.k.a. approximate-expansion) c) Conductance: $\frac{E(S,\overline{S})}{\frac{Vol(S)}{n}}$ or $\frac{E(S,\overline{S})}{\min\{Vol(S),Vol(\overline{S})\}}$ (a.k.a. normalized cut) d) Normalized-cut: $\frac{E(S,\overline{S})}{Vol(S)Vol(\overline{S})}$ (a.k.a. approximate-conductance), where $E(S,\overline{S})$ is the cost of the cut and $Vol(S) = \sum_{u \in S} degree(u)$. Being based on ratios, cuts obtained under such objectives can also be found in the literature under the name ratio-cut. These can be easily generalized for the case of weighted edges or in case of many partitions as well.

Balanced Graph Partitioning

The immediate extension of min cut and minimum bisection problems is for $k > 2$ partitions. In the case of cuts, the problem is called *k-way cut* and it has no restriction on the size of the partitions. Technically, it is exactly the Graph Partitioning problem, which as we know is NP-Complete. However, when k is fixed or predetermined there exist algorithm that runs in $n^{(1+o(1))k}$ time [58]. When the partitions are equally sized, we care about a problem called *k-balanced graph partitioning*. As we already know, the term balance can have many interpretations, some stricter than others, but the definition of the problem in the most well known form is as follows:

Definition 6 (k-Balanced Graph Partitioning) *Consider a graph G = (V, E), where V is the finite set of vertices with $|V| = n$ and E is the set of edges. Assume also weights $l(e) \in Z^+$ for each $e \in E$. Assume also any partitioning of G, P. The* ***k-balanced graph partitioning problem*** *is to decide whether there exist a partition P of size k ($i \in \{1, 2, ...k\}$) such that $|V_i| \leq \lceil \frac{n}{k} \rceil$ and if $E' \subseteq E$ is the set of edges with endpoints in two different parts V_i, then $\sum_{e \in E'} l(e) \leq B$.*

Complexity of Balanced Graph Partitioning

As other problems in the family, this generalization to k sets has many applications [36]. The types of applications that fit the most to this variation are those that

have the partition size as a requirement, as for example VLSI circuit design or job placement in identical machines [53]. In the general case of arbitrary graphs it has been proven hard to approximate k-balanced partitioning within any finite factor [6] unless P=NP. This is can be shown by transforming the NP-Complete problem *3-partition* into a graph in which we search for the minimum k-balanced partitioning. Specifically, in the 3-Partition problem, given a set of 3k numbers and a threshold S, we aim to decide whether we can place them into triplets so that each one adds up to S. Converting now each number into a graph clique and using the k-partitioning on this disconnected graph, if the answer to the 3-Partitioning decision problem is YES then the k-partitioning problem would have k partitions of size S each and cost of the cut edges equal to 0. On the other hand if the answer is NO then the cost becomes positive, since we would have to separate at least one of the nodes of one clique. An approximation algorithm with finite approximation factor should be able to differentiate between these two cases (since $\alpha * OPT = \alpha * 0 = 0$). Thus, a polynomial time algorithm would be able to solve the 3-partitioning, which contradicts its NP-Completeness nature. Actually, the existing approximations known as for today have time exponential in k [6]. However, the graph employed in the mentioned reduction is not connected, and the proof really relies on that (through the cost that equals 0) which suggests that the hardness may not apply in problems treating connected graphs or some specific type of graphs in general [36].

To get more insight into the approximation complexity some works focused on graphs structures that arise more in practice like trees [37][36] and grids [36]. Unfortunately though the results suggest that in the case of unweighted graphs there is an approximation barrier even in those simpler cases. In the case of trees without weights, the work of [37], using a reduction from the 3-partitioning problem, shows that with arbitrary degree on the nodes it is NP-hard to approximate the cut cost within a factor n^c for any constant $c < 1$, even for fixed diameter at most 4, and that this result is tight. The reduction uses specifically structured star "gadgets" in the same way the general reduction we described earlier uses cliques. To generate a connected tree graph, all tree roots connect to the root of the first tree, so the poof

strongly relies on the fact that the degree of the tree is unbounded. On the other hand to prove the results for fixed degree trees a construction of arbitrary tree diameter (longest path in the tree) is utilized. In other words, when we deal with fixed degree and diameter trees we should be careful on claiming NP-hardness. Finally, for fixed small values of k, a polynomial time minimum bisection algorithm for trees described in [61] can be extended to solve in relatively slower time the problem of balanced partitioning on trees, with the time depending exponentially on k. When it comes to grids, similarly, through a reduction to 3-partitioning using grid gadgets, the factor remains n^c but for constant $c < 1/2$ as shown in [36]. The result is similarly tight on the c parameter. The same work contains also a general framework and approach to reductions of this kind, standardizing the 3-partition reduction as an approach. In conclusion, even if we could transform or decompose a graph in multiple trees/grid, we are still not guaranteed to be able to find an approximation in polynomial time.

On the approximation of Balanced Graph Partitioning

As in the problem of minimum bisection, the approximation hardness led to development of techniques referred to as bicriteria, resource-augmentation, or pseudo approximation, that relax the strict balance constraint. The second term most accurately entails the intuition behind those methods. Specifically, we can relax the strict equality constraint by augmenting the size of the partitions by some factor ν and allowing some of the clusters to be oversized, compared to what we would wish, reaching this upper size limit $\nu(n/k)$, while other ones to be smaller. The reasoning behind the use of this relaxed problem definition is that the more flexible is the size of the clusters, the closer we get to the k-cut problem and thus the easier is to find a solution. This is the analog of $\beta-$balanced cut for $k \geq 2$ partitions. Now, by evaluating the approximation to this problem using the actual optimal solution, which requires equally sized clusters, we impose two optimality criteria, the actual strict one and the artificial relaxed (in terms of size) one. The relaxed problem is called the (k, ν)**-balanced partition** [6] or (k, ν)-way separators [31]. In the literature people also define $\nu = 1 + \epsilon$ to distinguish the exact case, in which $\nu = 1$. Using this problem

as a tool, one can move forward and derive an unbalanced solution, which can then be balanced through some post process algorithm if needed (in some cases only the approximation of the cost suffices).

$(k, 2)-$balanced partition

Some initial efforts targeted solutions for $\epsilon = 1$. In [79], a first idea of recursively bisecting a graph yielded an approximation factor $O(\log n \log k)$, later shown to improve to $O(\log n \sqrt{\log k})$. Around the same time, using spreading metric techniques [31] achieved an approximation factor $O(\log n)$. Spreading metric is an assignment of lengths $0 \leq d(e) \leq 1$ on every edge. It can be realized as a relaxed 0-1 indicator of whether an edge belong to the partition cut or not. Very briefly, using then an lp formulation and an ellipsoidal method solution over it, the authors were able to get the specified results, which is independent of k. Finally, in [53] the authors utilize l_2^2 metrics in the spreading metric technique to provide better spreading and rounding to 0-1. Their approach led in a factor $O(\sqrt{\log n \log k})$. Their results also suggest that, in contrast to the results of [31], the approximation factor dependence on k is actually necessary. This latest claim is based mostly to the semidefinite relaxation used to solve the optimization model and related to the integrality gap induced, so probably could not be assumed as general enough. Interesting to mention is that in those approaches the size of the graph, meaning number of vertices and edges, most of the times dominates the number of partitions parameter in terms of time complexity.

$(k, 1+\epsilon)-$balanced partition

Soon, it was realized that for $\epsilon = 1$ the solutions are not so practical. To allow for limited unbalance it is better to aim for near-balanced solutions instead, in other words $\epsilon < 1$. The approximation though seems to get significantly worse, with the first result in this effort being an $O(log^{1.5} n/\epsilon^2)$-approximation given by [6]. The approach taken is based on the observation that all algorithms for $\epsilon = 1$ cut the graph into small pieces that place together into partitions, so for good balance the pieces should be relatively small and the repacking phase - which can be seen as a scheduling

problem - should schedule those pieces to partitions of small size. The algorithm proposed recursively partitions the graph into small partitions by finding an a cut where each set has size at least $f(\epsilon)|V|$ given a fixed $\epsilon > 0$. Then merges partitions using dynamic programming to obtain the k partitions. The running time of the algorithm ends up being polynomial in n and k, but exponential in $1/\epsilon$. With similarly polynomial but exponential dependent on ϵ time and with higher degree n polynomial, [37] later provided an $O(\log n)$-approximation which is based on decomposition of general graphs into trees. Even if the approximation time is still affected by ϵ, the approximation ratio is not, closing the open question of their predecessor work [6]. Notice that this is only an approximation of the cut cost and that a balanced partitioned is not provided at the end of the algorithm. Concluding, the closer we move to balance the harder, in terms of time, the problem gets.

Special cases: trees, grids and planar graphs

Approximation hardness and bi-criteria approximation results also extended to the special graph cases of interest as well. In [36], apart from the NP-complete proof for grids and trees we discussed before, it is also shown that it is hard to approximate the cut cost on grids and trees without weight using bi-criteria. Specifically, there is no fully polynomial algorithm on trees that for any $\epsilon > 0$ computes a partitioning in which each set has size at most $(1+\epsilon)\lceil n/k \rceil$ and the approximation ratio is n^c/ϵ^d , for any constants c and d where c <1. The same is true for grids with $c < 1/2$. In this statement, d is an algorithm parameter for proof purposes so, since there are no restriction on its value, we can assume it as 1 and the dependence on ϵ is still obvious. Furthermore, the running time increases exponentially with ϵ, in other words as we move closer to the exact approximation. On a brighter side, for the case of weighted edge trees, [37] shows that given a fixed $\epsilon > 0$ there is a polynomial time algorithm for the $(k, 1+\epsilon)$-balanced partitioning of cost at most equal to that of the perfectly balanced case. The time complexity is $O(n^4(k/\epsilon)^{1+3\lceil \frac{1}{\epsilon} log(\frac{1}{\epsilon})\rceil})$, which clearly still depends on ϵ, but this is why we need it fixed. Some other results on

graphs with excluded minors (including planar graphs) there are $\widetilde{O}(n^3)$[2] time $O(1)$-approximations for the case of $\epsilon = 1$. Slightly worse approximation ratio $O(\log k)$ but better running time - $\widetilde{O}(n^{1.5})$ can be shown for grids without holes [37]. A good observation made though by the authors of [36] is that one should rather focus on fixed parameter tractability of randomized techniques to achieve better, in terms of time complexity, results.

As you might have realized by now, all the definitions of problems in the graph partitioning family zoom in or zoom out within a specific problem definition. We can abstractly picture the complexity of these problems if we realize our parameters and constraints. A graph partitioning in 2 parts is intuitively easier than in k parts when the number of nodes is arbitrary, meaning when we try to give performance guarantees for the general case. For specific number of nodes $|V| = N$ though, the complexity can be thought as concave, since the problem is easier for k=2 or k=N. When it comes now to balance concerns, the more restricted the case, the harder the problem. Only exemption is for k=N/2 equally sized parts, where the problem becomes the Maximum Weighted Matching.

This overview did not exhaust the list of all graph partitioning problems. All the aforementioned variations are the most useful and relevant in our case. It is very educative to realize that, even under the same lens, a graph partitioning problem may receive many different characterizations based on the different perspectives its solution follows. And since there are many other fields in which these problems apply, it is worth noting that the literature is very broad and rich with results that one should explore in higher detail. In Appendix A we provide a summary of the time complexity and approximation factor results in the literature relevant to this study. This should give a good starting point to any reader eager to look more into graph partitioning.

Returning back to our node clustering problem one might have already figured that it is just an M-balanced graph partitioning over a complete directed graph. Typically, graph partitioning is defined on undirected graphs, as we implicitly assumed.

[2] $\widetilde{O}$ or soft-O is shorthand for $O(n \log^k n)$, also found as quasi-linear time, for some k

Extensions in directed graphs tend to be more complicated [31] giving almost twice as worse approximation factors. Not only that but it seems that all those hardness results should make us feel rather helpless. Still, many of those problems are unavoidable and need a solution, such as this one, and people, aware or not of the hardness of graph partitioning, have developed various methods to approach the issue. The complexity understanding is just making us thoughtful about what approach fits best as well as what our expectations should be. In the next section we dive into those methods that will walk as through the options we have to solve the node clustering problem in a timely manner.

3.2 On solving the Graph Partitioning

In the previous section we discussed and realized the complexity of the graph partitioning problem. In many cases, especially as the partitions size gets more restraint, the problem gets hard even to approximate. The hardness can be seen in slow approximation time as well as bad approximation ratios that depend on the input. Despite that, due to the high applicability of problems of this family, there have been many algorithms to try and tackle them in a realistic manner. Some of the ideas were already discussed in Section 3.1. For example graph decomposition or ILP and semidefinite relaxations with use of spreading metrics are some of the methods utilized in the goal to give a better approximation to balanced graph partitioning. Apart from those, there are many other algorithmic approaches that we can take to optimize a system. In this section we will discuss those possible alternatives along with advantages and disadvantages they impose given the complexity of graph partitioning.

Before we start the discussion though, there are some important observations about the problem's nature. The graph contains nodes, edges and weights. The question is how to partition the nodes into the partitions so that the cut cost is minimized and the partitions maintain a certain structure. The weights might be real or integer, so as the cut cost. However, the edges and the nodes are always integer. Actually, most of the times, we have to somehow order the nodes so that

they are recognizable by the algorithm, even though there is no natural ordering. At the same time we might model the edges as a tuple of nodes or by a unique edge number. Finally we need to be able to distinguish between the partitions created. One natural way is to treat them as unordered sets of nodes. Many times though, for modeling purposes we can artificially give a number to the partitions and then have the partition's number assigned to a node indicating its "membership". All these little modeling details have the potential to dramatically change the viewpoint of an algorithm. Despite that, it is obvious that we are dealing with a discrete value problem, since a partition can be realized by the artificial orderings we used during the modeling phase. We shall later see when and how modeling details might affect an algorithm's performance.

On solving the graph partitioning, the methods typically follow at least one of four basic frameworks. The *constructive framework* contains all the methods that use the graph to construct a partitioning using some "building" rule. It can be also defined as the set of deterministic heuristic algorithms. Similarly but with a slightly shifted focus, the *transformation framework* is based on the reduction of the partitioning into another problem that we know how to solve. By solving the derived problem we obtain a solution able to be transformed into a feasible (optimal or near-optimal) partitioning. For example solving the min-cut using the max-flow problem is a problem transformation. Technically, any formulation of the problem can be thought as a transformation of the problem's general description into a specific perspective, so almost all methods could belong in this category, although we will emphasize on the least obvious transformations. From these methods we obtain a partitioning only at the end of the running process. On the contrary, methods that follow the *iterative framework* always maintain a feasible partitioning that is valid only for the current moment. Methods in this framework realize the optimization as a search of the optimal solution in the space of feasible solutions. Their main goal is to perform this search in a way that fast leads to an optimal solution. Finally, since partitioning is a hard problem, there is the *relaxation framework* that basically focus on obtaining approximate solutions. The idea here is that by relaxing either

the integrality or the size constraints of the partition we can solve the problem more efficiently, obtain an optimal solution, and then construct from this solution a feasible graph partitioning. The hope is that this partitioning is not going to differ a lot from the optimal. Methods in this framework contain the bi-criteria approximation, LP and semidefinite programming relaxations etc. The frameworks are nothing else but different ways of thinking of how a solutions should be obtained. Almost all of them will be found in the literature of any hard but well studied problem. However, the specific methods that give solutions to graph partitioning problems are accordingly customized and define a unique set of approaches.

The approaches themselves can be categorized based on the type of the algorithm they use. We will try to give a good overview of most of them, but there are three big categories of focus, bicriteria approximation, spectral partitioning and move-based (iterative) approaches. As we already saw, *bicriteria approximation*, or more generally the approximation of a graph partitioning problem using the simplest problems in this family, is a common approach, especially useful when we try to derive some approximation ratio. Methods that have been used to provide bicriteria approximations contain also graph decomposition either in terms of simpler standard graphs (for example trees) or in node chunks (subgraphs). Another natural and very common approach is to optimize using mathematical programming formulations. There are a few standard ways of how to define formulations for graph problems, as for example using the notion of a flow or following an edge driven approach. Typically, those formulations are mixed integer and they contain linear constraints, which enables some standard optimization techniques. Also, we can very easily create and solve relaxed ways of the problem that derive from the mathematical programming formulations. One of those methods is the *spreading metrics* formulation. Now, using mathematical programming and linear algebra we can also give a geometric representation to our graph of interest and enable another huge way of thinking. Specifically the partitioning can then be translated into vector or point partitioning in an n-dimensional space and performed faster by dimension reduction. This way *spectral partitioning* was born and extensively studied, mostly by linear algebra practitioners. Under completely dif-

ferent type of lens, move-based approaches, such as *iterative optimization* and *markov chain modeling* have been also used, especially in real-time scenarios. These big three categories have different characteristics that make them useful in different scenarios. Since our study requires fast algorithms, we mostly focus on spectral partitioning as one of the fastest useful approximation methods and finally for moved-based methods that can easily be applied in our setting. Fast deterministic heuristics to construct a clustering are considered in section **??** In what follows we present interesting methodologies and results of algorithm design for graph partitioning that mostly relate to k-balanced graph partitioning.

3.2.1 Spectral partitioning and geometric based algorithms

Geometric algorithms try to find a geometric representation of a problem, and then use the properties of this geometry to give an alternative way of thinking on how to construct solutions to the problem. For example, in the case of graph partitioning, if we find a good way to represent all the graph nodes on a line so that parts/clusters are revealed, then we can just find the segments of this line that describe the different clusters. More generally, transforming our data into an entity of the d-dimensional world, we can use characteristics of this world to partition our nodes. Some of the famous approaches contain the *linear ordering, multidimensional representations of points,* and finally *vector space graph embedding.* All of them consist of two parts mostly, the transformation of the data and the clustering of the transformed data.

Spectral techniques come to solve the transformation part. Spectrum of a matrix is called the set of its eigenvalues. So these transformation techniques are using the spectrum, i.e. the eigenvalues to transform the graph. In order to see this, consider representing a graph through its adjacency matrix, A. This can be a simple 0-1 edge indicator matrix or a 0-w_{ij} matrix that also represents the weights of this graph. Assume know that we represents the parts of a partition using 0-1 vertices. Then we can describe the graph partitioning as a quadratic problem. For example the two segments of the graph on the min cut problem can be represented by a 0-1 vector x, where label 0 is used for the one cluster and label 1 for the other. Then min cut is the

minimization of the quantity $x^T Ax = \sum a_{ij}(x_i - x_j)^2$, which is a quadratic function. Since though each entry of the vector corresponds to a node, if we relax the integrality of x and at the same time force a constraint $x^T x = 1$, saying that at least one value is non zero, then this is similar to the minimization of weighted squared distances of the nodes if placed on an 1-dimensional space, a line. Then we can heuristically use this ordering to approximate the min cut. The connection with the eigenvalues though was made in [46]. Using the fact that, since A is symmetric square matrix the linear equation $Av = \lambda v$ finds the eigenvalues and eigenvectors of A, we can show that the best not trivial ordering is provided by the second eigenvector of A's Laplacian matrix. The same work also presents the Laplacian as A's transformation to positive definite matrix, something useful in quadratic optimization attempts. This gave birth to the Spectral partitioning and Spectral clustering algorithms.

Algorithms using the linear ordering started being developed after [7] introduced the spectral bisection. However, this is a heuristic approach with no approximation guarantees. Actually, similar heuristics got considered that use k eigenvectors to solve the k-cut, i.e. k-way partitioning. The idea was that their values could be used for the node clustering. Specifically, in [16] the authors, after showing that the sum of the smaller k eignvalues is a lower bound to the minimum value of a k-way partitioning, they represent network nodes by the rows of the nxk matrix formed by the k eigenvectors and then use directional cosine similarity to partition the nodes. In analogous manner [4] uses the Euclidean distance instead. Non method is very promising, due to imposed assumptions in the process as well as poor quality partitions.

A more mature approach was later given by embeddings into vector spaces. The intuition is described in [12] and basically says that instead of approximating the solution by the eigenvectors as, we can provide a linear combination of those as a solution, and specifically find the one that is closer to be feasible. Then our search reduces to the search of this linear combination in the subspace spanned by the eigenvectors. A way to probe the space is presented in [42] with time $O(N^{d-1})$. Another approach in [17], realizes that the rows of the part indicator vectors matrix

represent the part assignment of each node and then shows that to find the optimal k-way partitioning then we would have to consider all N eigenvectors and perform vector partitioning. This is a more precise statement about how can we really use the spectral techniques as well as the hardness they impose.

Generally, however, this framework became well-known through applications like image segmentation or data points clustering. In cases like that it is able to unravel clusters with peculiar shape that other methods struggle to find. The approach used more often is that data are represented by a graph with nodes corresponding to the different observations and edges being weighted by a similarity metric. Then eigenvalue decomposition is applied on the Laplacian[3] matrix of the graph and the k eigenvectors corresponding to the smallest eigenvalues form the columns of a matrix U. Afterwards, the rows of U are places into the k-dimensional space and a clustering algorithm is used to derive k clusters. Each row represents a data point and so a final partitioning is immediately obtained [69][60]. For the partitioning k-means is often considered while algorithms that minimize distortion are suggested[69]. Typically, the hardest part is the choice of a good similarity matrix. We will return to this in the next section.

Spectral clustering is a relatively fast approach in the scale we care for. Its complexity is dominated by the eigenvalue decomposition which is roughly $O(N^3)$. There are some drawbacks still. First of all it is still an approximation and not an exact method. It performs very well on data that are partitioned but on non partitioned data the performance is more arbitrary. Also, it is a k-cut approach, which means that as is it will not result in balanced partition. This part can be dealt with through post-processing or other type of clustering algorithms.

[3]The Laplacian of the matrix can be formulated in many ways. A simple one is A-D where A is the weighted adjacency matrix and D is the diagonal matrix of the sums of each row of A. A detailed presentation of Laplacians and Spectral clustering techniques is provided in [60]

3.2.2 Move based algorithms

In a move-based approach the whole idea is to imagine the search space as a graph on which we take a walk to find the best solution. The graph nodes (states) are feasible solutions and the edges indicate other solutions/states we are allowed to move from our current point. As we walk on the graph we move from a feasible solution to another. In every step we evaluate our current position and compare it to our walk so far. This perspective is different from other approaches we discussed, in that, instead of composing a solution we believe as good, we actually view the feasible solution space as a flatten structure. We just need to find a good way to perform our walk, in a treasure hunt manner.

A way to algorithmically achieve this, is by using the current state (current solution) to construct a new state (next solution). Notice that since we construct the solutions we are guaranteed to only see feasible ones. So one important aspect of this approach is the construction itself. In the viewpoint we described before, solutions succeeding each other are connected with an edge. Thus the construction, or in other words the alternation of the current solution to derive the next, can be thought as a neighbor definition that causes a neighborhood of solutions. The other important aspect is the actual way we perform the walk given that we are in a certain neighborhood, since a neighbor definition might indicate many potential neighbors for us to "visit". To summarize, a move-based approach is a specification of which solutions are neighbors and how to prioritize neighbor visiting from a given solution/state. The first part is basically tied on our problem and modeling, so this is our first concern under our discussion about graph partitioning specifically. The second part is a characteristic of the moved-based algorithm which helps us define different types of algorithms. This is to be formally discussed a couple of paragraphs later, but to find an effective neighbor definition we should get a sense of how this neighborhood may be used in our walks.

The search space walk can be alternatively realized as an iterative function evaluation, since in every step we evaluate our objective function. From Analysis, we

know how to optimize over continuous value - continuous functions through their solution continuity. In a way, a neighborhood small enough around an x contained in the function's domain guarantees that the function values for this neighborhood are close to each other (see for example Lipschitz continuity). Using this property on a continuous domain during the optimization phase, we are able, given a x, to find such a small neighborhood and define the direction of our next step. For example Bayesian Optimization is one of the move-based approaches that follows such logic. Even if a function has a surprising form or extends to many dimensions, this structure is essentially part of its nature, thus can be appropriately exploited. On the other hand, when we deal with discrete domain functions, where the domain does not have any natural structure, to be able to follow similar techniques as in the aforementioned case we need to assume a structure. Typically, the smoother the function to be optimized the better. Thus, to create a smooth effect the neighborhood should contain solutions that are not too different from each other. Therefore our optimization highly depends our creativity, intuition and understanding of the problem as well as on the problem data themselves, the discrete domain.

Graph partitioning is a discrete domain problem with a numeric, real or discrete, value function. As we also noticed in the start of this section this domain highly depends on the modeling we choose. Generally a solution is a network partition, which can be conceived as a collection of sets of nodes. The number of sets that can belong to the partition as well their size depend on the type of partitioning problem. For instance, the minimum bisection problem asks for two parts of equal size. All the possible partitions that satisfy the number and size constraints are feasible solutions of the problem and valid states for a moved-based algorithm. Assume a unique representation of a partition and suppose we are trying to derive from it another feasible partition. The smallest change that we can perform is a "jump" of a node from a cluster to another [39]. However, in cases were the partitions need to satisfy some size constraints, a "jump" has the potential to result into an infeasible solution. This can be avoided when we have bounds on the size as in (k, ν)-balanced partitioning, but not in the of strict (equality) size constraints. Thus, the next smallest change

that logically follows the "jump" is a node "swap", i.e. the exchange (two successive "jumps") of the clustering assignment of two nodes [49]. Extending this thinking this can be generalized to n-cycle exchange. The first two cases however are sufficient to describe any other partition alternation, so they have been found as the most appropriate for moved-based approaches when we realize a partition as a collection of node sets.

Deterministic Improvement

Using this as our basic building block we can now focus more on the moved-based tactics. Intuitively, when we optimize over a function we have been used to look for the highest change step towards an optimal or improving direction. Some of the algorithms mimic this approach aiming to improve the objective function value in every step. These algorithms are called *iterative improvement algorithms*. Of course, if we only keep taking the best step that improves our current objective value then we can very easily fall into a local optima. To avoid this, while maintaining the intuitive simplicity of a greedy choice, we can instead take a step towards the state with the best value, even if this does not improve over the current state.

Following this logic, an algorithm proposed in [49] for the minimum bisection problem performs the optimization in rounds. In every round, using the abstraction of a "swap", it performs the swap with the best value and it locks the swap pair. Nodes that are locked cannot participate in any other swap until the next round. It continues until all the nodes are locked. At this point the best solution encountered during this round is chosen as the start of the next round. The process keeps performing rounds in this manner until a round does not improve over the previous one. Overall, the running time is $O(n^3)$ time per round which can be reduced to $O(n^2 \log n)$ when using sorted list of gains, where gain is how much better a solution is from the current state. Finally the number of rounds is 2-4 and mentioned to be not strongly related to the number of nodes [49],although tested on small problem instances, while the most improvement happens during the first few rounds.

As one can notice, evaluating all the available pairs per step is rather wasteful

idea, since only some of them will have the potential to improve, and it gets even worse when for example all pairs have the same gain. Based on this observations as well as on the fact that using more elaborate data structures in the implementation can boost the algorithm's performance, [28] improved later the time of the algorithm to $O(\max\{|E|\log n, |E|d_{max}\})$, where d_{max} is the maximum node degree. On the contrary, [39], under a context of VLSI circuits and netlists, improved the time complexity to $O(|E|)$ by using the idea of node "jumps" apart from optimized data structures. Intermediate solutions however are not always bisections and the final partition is allowed to slightly violate the size criteria.

Similar, but focused to avoid cycling is the tabu search. Tabu search, instead of performing rounds and locking moved pairs, is a continuous best move search that locks only the most recently swapped nodes. The pairs can be unlocked later if some constraints, defined by the specific implementation, are met. In this way the search is more controlled, hill climbing is enabled and cycles avoided.

Stochastic Improvement

The algorithms described already are deterministic in the sense that every run is going to result in the same walk over the solution space. Another way to deal with local minima is by randomizing the process. This means that every time we choose our next step, instead of greedily picking one, we pick based on some probability distribution over the neighbor solutions. These are called stochastic optimization algorithms. The most easy to perceive is the case of uniform probability, where we uniformly pick the next solution and we move if it improves the objective else we pick another one. When our space is just the full search space this is called pure random search and the search has no structure or focus resulting in slow running time and almost no performance guarantees [92]. A variant of it called pure adaptive search [91], tries to focus search by searching only into the reduced space of solutions that improve the objective function, but this is something almost only theoretically achieved. However, when we are using steps like the "swaps" not all states in the search space are reachable from our current state, only those "a swap away", thus

the search is less fuzzy.

There are two important observations of this method in comparison to the deterministic approach. First, since we will visit the first randomly picked improving state, we will only evaluate the objective function until we find the first improvement solution. This reduces the time needed in the first rounds of the search, as we get though closer to the local optima the rounds have similar running times as the deterministic approach and maybe a bit more due to the randomized search part. Of course, to avoid losing time by randomly swapping the same pair of nodes many times, a locking mechanism needs to be adopted as well. Secondly since we do not favor solutions that improve more the objective function, we have more time for exploration of the solution space. The forced improvement however, almost always guarantees the optima we will find is local.

Ideally, we need a way to move away from local optima, which when we talk about minimization problems is also referred as "hill climbing" process. One approach is to run in parallel many instances of any of the randomized techniques. Similarly, working with improvement approaches, we can run many instances but with different, randomly picked starting points, a method called multi-start. However, the stochastic walk can naturally avoid locality, if we move to worse solutions with non-zero probability. This idea is behind what is called simulated annealing. Simulated annealing, trying to mimic the physical annealing of a materials to their steady state, is a technique to allow random search in the space in the start of the algorithm, called exploration, and slowly to focus this search in a specific neighborhood, called exploitation. Specifically, when we come across worse solutions we flip a coin that specifies whether we will move with probability $e^{\Delta/T}$, where Δ is the difference of the value of the current solution minus the value of the current solution and T is a parameter that we dynamically change to guide the optimization. If we do not move we stay at the same state. Notice that Δ is a negative number and that as T decreases the probability of moving decreases. Therefore, by controlling the decrease of T we can balance between exploration and exploitation phases, thus control the rate of convergence. Using Markov chain theory and Gibbs-Boltzmann statistics properties it can

be shown that in infinite time the algorithm will converge to the global optimum [5]. Of course this does not sound practical, but since it has more potential to find the optimal it received a lot of attention. In the context of minimum bisection, a study in [1] suggests that good starting points, geometric in the number of steps decrease of T and search space relaxation (relaxation of the parts' size constraints) can be valuable, but multi-start of other algorithms with random solutions might be better.

The Markov chain model, which proved valuable for the simulated annealing analysis, became an paradigm by itself. The idea is simple. Picture all the feasible solution search space as a big Markov chain. If this Markov chain is irreducible (thus properly designed) then there is a steady state probability distribution that describes the percentage of time we spend in each state during our walk. We can then construct the Markov chain in such a way to either get average case guarantees or be led to the optimal solutions. During the construction phase the designer needs to decide on the neighborhood operations as well as the transition probabilities carefully. Tools like that can be more useful for analysis purposes or application on real time systems. They are not particularly better in terms of time complexity that other stochastic techniques.

A completely different perspective in the stochastic search space is given by genetic algorithms. In those, we have many active solutions at the same time, the population, that can reproduce with probability related to their quality. There are two basic operations that genetic algorithms define, the crossover and the mutation. Those define how the solutions of the upcoming population are to be produced and they are very elaborate in order to comply with solution feasibility and guided randomization. A way to realize them is like algorithms that allow multiple simultaneous swaps, and, if properly designed the probability of a swap can depend on "its quality" [76][14]. This approach is less practical and understandable because it cannot easily localize the search, but has been very successful as a fast starting points generation method to be used in multi-start.

Finally, what is nice for the stochastic algorithms is that they can use heuristics in a more effective way than deterministic step algorithms since there is always a

chance to randomly move to a better place in the search space closer to the optimal solution. This creates a balance between a huge unknown search space and a fast local heuristic. Specifically, an idea that lies between deterministic improvement and stochastic optimization is to enable moves to a small set of solutions with higher potential to improve over the current state, and then stochastically pick one among them. To be able to decide however which ones are seemingly better, a heuristic can be used to define this small solution set. This means that the solutions picked will be assumed good and that the running time is reduced to the running time of the heuristic, therefore potentially decreased overall.

Memory vs Memoryless

Another way to categorize moved-based algorithms is based on whether they need to remember past solutions or not. The greedy algorithms typically try to take the best step forward, therefore there is no need to remember past states since the final state is guaranteed to have better objective value from all the other states. The same is true for randomized techniques that ask for improvement only state visits. However, when we stochastically walk in the search space and allow worse solutions to be visited as well, then we need to remember the best solution so far. Some of the algorithms described before need an overall memory and only know what is best on termination [91]. Others, as for example [49][39][28], have a short term memory that expands to the time of a "round". What is better is always relevant to the application. If we deal with a real-time environment where changes happen on the spot, then we care more for the long-term expected performance (minimize the average over time cut-set for example) and the "optimal", local or not, might never be visited. In this case greedy or short memory approaches are probably better. We will get into more detail in the next chapter when we will talk about dynamical systems.

Benefits and Implications

In practice moved-based approaches have been very successful for many reasons. First and foremost this framework is rather simple and intuitive, which translates into

fast, understandable and modular implementations. It is very natural to try and iteratively improve over a given solution and by decomposing the "solution transition" from the "solution space search" parts of the moved based approaches, one can very easily make the implementation modular. This means that the way of transition (for example swap) can be fixed and tested towards any of the different searching techniques explained. Benefits can also be found in the small changes factor, since in many cases can be performed really fast, without significantly altering the rest of the solution structure which many times is desirable in real-time implementations. One example is the node clustering reconfiguration the current work conveys. In those cases the time to decide on the small change to be performed becomes the main point of interest, meaning that with some convergence or average case guarantees the total execution time reduces to the step execution time which can be orders of magnitude faster. Finally, the optimization can be totally oblivious of the objective function over which we want to optimize. This is especially useful when we deal with hard to optimize - because for example their form is unknown - but easy to evaluate objective functions, since moved-based approaches can be treated as a black box. Of course, specific objectives and domains can be exploited to boost performance and people should be encouraged to do so, but this is not mandatory in the general case. All these contribute to the high applicability these methods are experiencing.

Unfortunately, as we have seen, graph partitioning is a hard problem, especially when we deal with many (k) equal sized parts. The search space gets exponentially bigger with the size of the instance, while the optimal solutions' fraction can vary depending on the weights. So in other words, we can realize two problems of the move approaches. When the weights variation is big, then the fraction of optimal solutions is small an the chance to find them decreases leading to slower convergence in general. Not only that but even the upper bound of the convergence time is big (small convergence rate) [91], which agrees with the poor approximation rates seen in the previous section. As a result we have the effects of large running times, poor quality solutions and hardness to provide performance guarantees. On the other hand, where the weights variation is small, then almost all partitions will be close to the

optimal solution. However, this is not immediately realized by the algorithms, which means the running time might still be large, even if it is obvious that we can stop the process at any feasible partition with high chance of it being good. Apart from those two important observations, even though the random approaches have better chances to give good solutions they can lead to problems that might seem bizarre to a typical user. Namely, on the same set of data the solution is different every time we run the algorithm. Also, some of the algorithms, to get better chances of landing in a good solution, might temporarily take bad decisions like performing a swap that worsens the solution. This in some real-time implementations might be from undesirable to unacceptable (if for example a network crash is on stake). These should be carefully considered in order to get the full benefits moved-based can provide.

3.3 Static Model

Let's get back to our problem. As we realized earlier in our discussion, the minimization of the inter cluster traffic communication is closely related to the balanced graph partitioning with main difference the directionality of the graph links. We might also wish to perform some load balancing, in order to avoid congesting clusters or aggregation links (inter cluster communication links). In this case, the problem will still belong to the greater graph partitioning problem class, but the objective slightly differs from the basic templates discussed in previous sections. This does not affect the algorithmic choices available so much, but more their performance.

Therefore, our decision making can be summarized by three distinct choices. First of all, which is the optimization objective. Second, how are we going to deal with the directionality of the links of the graph we seek to partition. Lastly, which algorithmic approach serves better our optimization goals, naming fast execution time and solution with certain quality. Unfortunately, the performance of the algorithms changes based on our first two choices. So, the evaluation is not straight forward process if we want to consider all the alternatives.

We will try to keep it simple. To create a perspective towards the problem and the

effect of modeling in our optimization effort we will initially focus on the minimization of inter-cluster traffic. Later we shall revise the validity of our choice. To start, we will model and solve the static instance. In other words, we will pretend the time is frozen and that we optimize our system once. After we realize problem and algorithm specific limitations as well as appropriate assumptions based on our application field, the data center network, we will go back to the real, dynamic setting, where time is an important consideration.

There is an important disclaimer in our optimization process. We acknowledge that the problem is relatively hard, so we do not seek optimal nor even near optimal solutions. On the contrary we aim to give solutions good enough for the smooth operation of our network. These might be solutions that keep the link utilization in certain levels, can be achieved with very few changes of the current clustering etc and that generally provide some sort of guarantee overall. This does not mean that we are not optimizing, but that we optimize with realistic expectations.

3.3.1 Problem formulation

Suppose our network consists of N ToR switchs, the network nodes, and the cluster size is d. We assume that we always have enough clusters for the given N, specifically $M = \lceil N/d \rceil$ clusters. Let $D \in \Re^{N \times N}$ be the traffic rate matrix with D_{ij} the traffic rate from node i to node j. Let also $X \in \Re^{N \times M}$, with $x_{im} = \mathbb{1}\{$ node i belongs to cluster m$\}$, the matrix of the cluster membership assignments of the network nodes. Then we can define the problem of minimum inter cluster traffic clustering as follows:

$$
\begin{aligned}
\min \quad & \sum_{m=1}^{M} \sum_{i,j=1}^{N} D_{ij} x_{im} (1 - x_{jm}) \\
\text{s.t.} \quad & \sum_{m=1}^{M} x_{im} = 1 \quad \forall i \in [N] \\
& \sum_{i=1}^{N} x_{im} = d \quad \forall m \in [M] \\
& x_{im} \in \{1, 0\}
\end{aligned}
\tag{3.1}
$$

where the notation [K] is used for the set $\{1, 2, .., K\}$.

Focusing on the objective function, we can rewrite it as follows:

$$\sum_{m=1}^{M}\sum_{i,j=1}^{N} D_{ij}x_{im}(1-x_{jm}) = \sum_{m=1}^{M}\sum_{i,j=1}^{N} D_{ij}x_{im} - \sum_{m=1}^{M}\sum_{i,j=1}^{N} D_{ij}x_{im}x_{jm} =$$
$$\sum_{i,j=1}^{N} D_{ij}\sum_{m=1}^{M} x_{im} - \sum_{m=1}^{M}\sum_{i,j=1}^{N} D_{ij}x_{im}x_{jm} = \sum_{i,j=1}^{N} D_{ij} - \sum_{m=1}^{M}\sum_{i,j=1}^{N} D_{ij}x_{im}x_{jm} \quad (3.2)$$

where the last part follows, since for any valid membership assignment $\sum_{m=1}^{M} x_{im} = 1$. The above reformulation describes the inter cluster traffic as the remainder we get by subtracting the intra cluster traffic from the total traffic flowing in the network. Therefore, under the same constraints (which basically define a valid cluster membership assignment) we can claim that the minimization of the inter cluster traffic is equivalent to the maximization of the intra cluster traffic. This is intuitive, since the total traffic rate per moment is constant and so as we minimize the one type of traffic we maximize the other. We can thus rewrite the problem as:

$$\begin{aligned} \max \quad & tr(x^T D x) \\ \text{s.t.} \quad & \sum_{m=1}^{M} x_{im} = 1 \quad \forall i \in [N] \\ & \sum_{i=1}^{N} x_{im} = d \quad \forall m \in [M] \\ & x_{im} \in \{1, 0\} \end{aligned} \quad (3.3)$$

where $tr(A) = \sum_i a_{ii}$ is the trace of the matrix. Let us denote with x_n the rows of X, which are the one-hot key vector of membership assignment of node n, and with X_m the columns of X that are the cluster indicator vector. The objective function of formulation 3.3 is derived as follows:

$$\sum_{m=1}^{M}\sum_{i,j=1}^{N} D_{ij}x_{im}x_{jm} = \sum_{m=1}^{M}\sum_{i,j=1,j<i}^{N} (D_{ij}+D_{ji})x_{im}x_{jm} = \frac{1}{2}\sum_{m=1}^{M} x_m^T(D+D^T)x_m$$

$$= \frac{1}{2}tr(x^T(D+D^T)x) = \frac{1}{2}[tr(x^T D x) + tr(x^T D^T x)] = tr(x^T D x)$$

since $tr(A) = tr(A^T)$ and $(x^T D x)^T = x^T D^T x$.

This formulation indicates a Quadratic Program. We can see this if we define the vector $w^T = [X_1^T \ X_2^T \ \dots \ X_M^T]$ and the diagonal matrix $Q = \begin{bmatrix} \text{D} & & 0 \\ & \ddots & \\ 0 & & \text{D} \end{bmatrix}$. Then we have that

$$tr(x^T Dx) = X_1^T DX_1 + .. + X_M^T DX_M = w^T Qw$$

Not only the form is quadratic but it also has 0-1 decision variables. Problems like that arise often when we deal with combinatorial optimization, thus there exists a special category for them, 0-1 Quadratic Programming problems.***

Let's get some insights in our intuitively derived formulation. First of all there multiple choices of decision variables that can lead to different formulations. For example we might prefer, as we will see later, to model whether two nodes belong in the same cluster or not. However, the membership vectors immediately help us realize the actual clustering, which in our case was a natural decision, since we need the solution to perform the reconfiguration. Secondly, Quadratic Programming in general is hard, and being the intuitive way to describe the problem validates once more its hardness. Notice that for solvable instances we need definite matrices in our data, something that we cannot guarantee in an uncertain changing environment. Third our latest transformations indicate that dealing with matrix $D + D^T$ is equivalent to solving for matrix D. The reason this is important observation is because $D + D^T$ is a symmetric matrix, thus can enjoy many nice properties, but also because the directionality of the links does not affect the optimality of a solution. Specifically, we can solve the equivalent problem where the matrix $L = D + D^T$ represents the mutual traffic exchanged between nodes and the node communication can be represented by an undirectional graph. Finally, the communication graph that we use for our optimization is not the same with the network topology graph and it always depicts a complete graph with non-negative edge weights, the traffic rates.

The number of feasible solutions of the problem, our search space, is huge. Specifically, consider the M clusters we want to fill with network nodes. We have $\binom{N}{d}$ choices for the first cluster, $\binom{N-d}{d}$ choices for the second cluster etc. However, notice that

the cluster ordering is artificial. This means that when we enumerate our solutions (network configurations) in the way described, a unique clustering of the nodes has been enumerated for all of the $M!$ possible ways we have to assign the individual sets of the clustering in the numbered clusters. Therefore the number of distinct objective function values is at most:

$$\frac{\binom{N}{d}\binom{N-d}{d}\dots\binom{d}{d}}{M!} = \frac{1}{M!}\frac{N!}{(N-d)!d!}\frac{(N-d)!}{(N-2d)!d!}\dots\frac{2d!}{d!d!}\frac{d!}{0!d!} = \frac{1}{M!}\frac{N!}{d!^M} = \frac{(Md)!}{M!d!^M} \quad (3.4)$$

with each one of them, including the optimal, correspond to $M!$ different and equivalent network configurations. Not only the actual search space is big but also our modeling enlarges it by a factor of at least $M!$. This cannot really be avoided in the case of mathematical formulations, else we would have to enumerate all the possible $\binom{N}{d}$ sets along with rules regarding which M combinations of them form a valid clusterings. Notice that due to the formulation, the search space of the optimization can be broken in equivalent areas of similar structure between which our mathematical programs are not able to differentiate. This means that same optimization steps might be followed when the algorithm reaches equivalent feasible solutions and since there is no way for the algorithm to realize and terminate the process faster, even when all unique objective values have been seen (thus the optimal as well).

Integer Linear Programming

Transformation of the problem to Linear instead of Quadratic is not particularly useful. The "hardness" of the objective function is resolved through introduction of increasingly many constraints. Specifically the respective Integer Linear Program is the following:

$$
\begin{aligned}
\max \quad & \sum_{m=1}^{M} \sum_{i,j=1}^{N} D_{ij} y_{ijm} & & \\
\text{s.t.} \quad & \sum_{m=1}^{M} x_{im} = 1 & \forall i \in [N] & \\
& \sum_{i=1}^{N} x_{im} = d & \forall m \in [M] & \\
& 2y_{ijm} \leq x_{im} + x_{jm} & \forall i, j \in [N]\ \forall m \in [M] & \\
& x_{im}, y_{ijm} \in \{1, 0\} & &
\end{aligned}
\tag{3.5}
$$

where $y_{ijm} = \mathbb{1}\{$ node i and node j (both) belong to cluster m$\}$ and the third constraint assigns value to this variable based on the clustering indicated by X. Therefore, to make the problem linear, we needed to add $N \times N \times M$ 0-1 variables and introduce the same as many constraints. The model in total has $N \times M + N \times N \times M$ decision variables and $N + M + N \times N \times M$ constraints.

To avoid modeling the clusters explicitly and reduce the number of variables, but with increased number of constraints, we can use an indicator variable to model whether two nodes are in the same cluster or not. We then need only constrains that define the formation of clusters. The model derived is the following:

$$
\begin{aligned}
\max \quad & \sum_{i,j=1}^{N} D_{ij} y_{ij} & & & \\
\text{s.t.} \quad & y_{ij} = y_{ji} & \forall i, j \in [N] & \text{(symmetry)} & \\
& y_{ab} + y_{bc} \geq 2y_{ac} & \forall a, b, c \in [N] & \text{(transience)} & \\
& \textstyle\sum_{j=1}^{N} y_{ij} = d & \forall i \in [N] & \text{(cluster size)} & \\
& y_{ij} \in \{1, 0\} & & &
\end{aligned}
\tag{3.6}
$$

This problem has N^2 variables and $N + N + N^3$ constraints. So we have to accept the tradeoffs the different formulations have. However the running time of both is exponential to the size of the model, thus not fast enough for our case.

3.3.2 Methods

Spectral Clustering

Despite being impractical when it comes to solve the model, the mathematical formulations can serve analysis purposes and more importantly initiated Spectral Clustering. As we discussed earlier in this chapter, Spectral clustering aims to give an approximate partition using graph transformation to an d-dimensional vector space. Luckily, our structure is a graph. There are two points to be mad though. First of all, we have a directional graph. Even though the weights are appropriate to serve as "similarity" of our nodes, we still need to decide how we should make the graph undirectional. We choose to simply use the mutual traffic exchanged between any pair of nodes, which equals the value $D_{ij} + D_{ji}$. Secondly, the goal of the spectral clustering is to partition the nodes of a similarity matrix into groups minimizing at the same time the total weight of the cut links connecting the groups[4]. There is no guarantee that our clusters are going to be equally sized. Therefore we have to decide on any possible post processing heuristic. Notice that in such a case some clusters are going to be oversized and others undersized compared to our ideal size d. Our simple greedy heuristic is described in Function 1

Function 1: Balance_Clusters

input : D, c, d

output: Node cluster membership assignment

1. Find all oversized ($>$ d) and all the undersized ($<$ d) clusters;
2. From each oversized cluster take out the least communicative nodes until you are left only with d nodes;
3. Assign each least communicative node to the undersized cluster it has the most communication with;
4. Repeat until all the clusters have size $\leq$ d;
5. **return** new clustering;

[4]Many implementations aim to solve the ratio cut problem

However, due to its ability to fast recognise the clusters of a clustered traffic, it can be used for generation of starting points in what we will discuss right after, iterative improvement methods. Specifically, if the clustering derived consists of equal clusters then we can be sure that this is the optimal and stop the process right away. If this is not the case, then we can either heuristically re-balance the clusters, or by using some node "jump" / migration method. In the good case, clusters are going to be almost equal and only few post processing iterations are going to be needed for the re-balancing. Else, existence of very big clusters means that there is no inherent structure in the data that can be used for clustering, thus any random clustering of the nodes suffices as a solution (so we might as well avoid the clustering).

Iterative methods

More promising for optimization in huge feasible solution spaces are the iterative methods also called moved-based methods that were described extensively in section 3.2.2. They are characterized by a neighborhood operator and an iterative search algorithm. Here will focus on the equivalent problem of **Intra-Cluster traffic Maximization**, although our algorithms can be adjusted for the minimization problem as well. We describe three different algorithms, one greedy and two stochastic about the iterative search in the feasible clustering space. Since the size of our clusters represents the number of ports on an AWGR device, which is fixed, we decided that the appropriate smallest neighborhood operator is a "swap", i.e. the exchange of the clustering assignment of two nodes that belong to different clusters. In a sense, if we wanted to perform node "jumps" from a cluster to the other we would either need extra space, which translates to wasted AWGR ports, for incoming nodes to every cluster, or more time before node exchange is performed. Since the actual performance does not get significantly better, we simplify the problem by setting the swap as the neighborhood operator. Given the cluster membership indicator vector formulation described above, a swap is formally defined as follows:

Definition 7 (Node swap) *Assume a graph $G = (V, E)$ of N nodes that form M clusters of size $N/M = d$ each. Let X, represent the $N \times M$ indicator matrix of node*

cluster membership assignment where $x_{uc} = \mathbb{1}\{$*the node* u *belongs to cluster* $c\}$*. Suppose two nodes* $u, v \in V$ *that belong to two different clusters* C_i *and* C_j*, respectively, which translates to* $x_{ui} = 1$ *and* $x_{vj} = 1$*. We call a* **swap** *of a pair of nodes* (u, v) *the exchange of the clustering membership assignment of these two nodes. This can be represented by a new cluster membership assignment matrix* $\overline{X}$ *for which:*

- $\overline{x}_{nm} = 1 - x_{nm} \quad (n, m) \in \{(v, j), (v, i), (u, j), (u, i)\}$
- $\overline{x}_{mn} = x_{nm} \quad$ *otherwise*

Then, for X *and* $\overline{X}$ *we have that* $\quad \|X - \overline{X}\| = 4$

Notice that when a swap is performed 4 exactly elements are changed on the cluster membership indicator matrix. Specifically, for two nodes the indicator of their assigned cluster turns from 1 to 0, while the indicator of the cluster they move to changes from 0 to 1. In the same way one can define the *jumps* or an *n-cycle* of chained node jumps, where the last node joins the cluster that the first node left, leading to equal sized clusters clustering. In the first case, the resulting indicator vector is going to differ by a value 2 (two values where altered in the matrix) from the starting indicator vector an and *is not* going to represent a valid clustering. In the second case both vectors are valid and they differ by a value $2n$, where n is size of the cycle.

The last two operations are considered impractical for our case. Due to the size restriction, jumps are only valuable when performed in pairs or chains that lead to valid clusterings (a.k.a. swaps or n-cycles). Thus, we can either perform them virtually until a valid clustering is achieved, which might take arbitrarily long time, or skip them and perform only swaps (even under the assumption that our algorithms make decisions based on optimal node jumps we can execute them in pairs). Assuming that the performance is not getting significantly better, we make the second choice. In similar way, one can show that an n-cycle can be replaced by multiple swaps and that if a cycle is optimal for some n, then any deterministic improvement algorithm will iteratively perform all swaps comprising the cycle (otherwise no cycle of size n

is optimal). Swaps seem to be an appropriate choice to maintain changes simple, intuitive and fast.

From the definition of swaps and under the same network assumptions, we can define the neighbor operation as follows:

Definition 8 (Neighbors) *Two valid cluster membership assignment matrices, X and $\overline{X}$ are called* **neighbors** *if and only if $\|X - \overline{X}\| = 4$, in other words, if the one can be constructed from the other using only one (node pair) swap.*

For what follows, let us call the cluster membership assignment matrix a *state* in our search space. Each state then represents a feasible clustering (configuration) and is connected to other states through the neighbor operation. This means that each state has exactly $\frac{N(N-d)}{2}$ neighbors, as many as the possible swaps. Considering now equation 3.4, there are $\frac{(Md)!}{d!^M}$ states and each one of them has $M!$ equivalent states, which can be reached only if a sequence of swaps results in the exchange of two full clusters (for example cluster 1 gets renamed to cluster 2 and vice versa). When we are using algorithms that strictly improve the objective function, since equivalent states have the same objective value, we will not visit any equivalent state. This becomes a possibility only when we deal with stochastic iterative or (non-strictly) improving methods, in which case we have to decide whether how this affects the method's performance.

The states and the neighbor operation are the two building blocks of our iterative models and the only thing left is the actual method we will use to traverse our space of feasible cluster configurations. The methods define a) how we choose which neighbor to visit next given our current state b) when our walk terminates and finally c) what is the clustering solution. We describe our three iterative methods next.

Iterative improvement via Best Swap

The first and simplest method is to move to the neighbor which decreases the value of our objective function the most. This is a greedy choice. Then we move from state

to state until we cannot improve our objective anymore. The process is summarized in *Algorithm 2* below.

Algorithm 2: Best Swap

input : N, M, d, D

output: Node cluster membership assignment

```
1.  c[a] ← Initialize_Clustering(N,M,d);
2.  while True do
3.      swaps ← Generate_possible_swaps(c);
4.      best_swap ← first element of swaps;
5.      best_value ← Objective_Change(D, c, best_swap);
6.      foreach swap s in swaps do
7.          change ← Objective_Change(D, c, s);
8.          if change > best_value then
9.              best_value ← change;
10.             best_swap ← s;
            end
        end
11.     if best_value > 0 then
12.         Perform_swap(c, s);
        else
13.         return c;
        end
    end
```

[a] This can be a cluster membership indicator matrix or any other structure useful to represent a clustering.

This method is not expected to perform well and the reasons are simple and obvious. To find the best swap we need to evaluate all possible swaps from our current state. This requires calculating the objective value change for $O(N^2)$ swaps. What is more, because of the greedy approach, we will most likely end up in a local optima.

Despite those issues, this algorithm sets some ground toward the understanding of the iterative approaches. There are some important points to make. First of all, we have many choices on how to initialize the clustering our algorithm starts with. It can be a random clustering, a result of another clustering algorithm, or a specific network configuration at the moment. And this initialization can affect our proximity to an optimal solution. Actually, a way to increase the chances of finding the global optimum is to run many algorithm instances in parallel for different starting points. Motivated by this observation, the function *Initialize_ Clustering* in line 1 of the algorithm creates some space for exploration of those choices.

A second observation has to do with the swap quality evaluation. In other words, how much we can improve our objective value by performing a swap s. One way to find out would be to evaluate the function before and after the swap and then compute the difference. This would take $O(MN^2)$ time. However, using the current state and the swapping pair information we can immediately calculate the difference of the two objectives way faster and without the need to calculate any of them. To see this, consider the changes that happen in the intra cluster traffic because of a swap (i, j). The traffic is going to increase by the amount of traffic i exchanges with nodes at its destination cluster but decreased by the amount of traffic it used to exchange with its origin cluster. The same for node j. So now we just have to add and subtract O(d) values only.The calculation is described in *Function 3*.

Function 3: Objective_Change

input : D, c, s

output: Node cluster membership assignment

1. (i,j) ← s;
2. u ← cluster of i in c;
3. v ← cluster of j in c;
4. $\Delta \leftarrow \sum_{w \in v} (D_{wi} + D_{iw}) - \sum_{w \in u} (D_{ui} + D_{iu}) + \sum_{w \in u} (D_{wj} + D_{jw}) - \sum_{w \in v} (D_{wj} + D_{jw}) - 2(D_{ij} + D_{ji})$;
5. **return** Δ;

Iterative improvement via Random Swaps

A way to avoid both of our previous issues is to test randomly the swaps discarding from your list non improving swaps, until one improving one is found. The idea is the same, improve the objective in every step. The execution though results in a more exploratory search of the state space. This increases the chances of ending up in a global optimum under similar principles and with decrease in the average running time. Notice that in worst case we will have only one improving state which is going to be sampled last. Also, to terminate, the algorithm must have already evaluated all possible swaps realizing that there is no improving one. Generally as we proceed in our exploration we have less and less improving states and the time increases.

If we wish to keep the search tractable we can stop the process if K swaps are tested and no one leads to better solution. The bigger the K, the slower the algorithm, but we increase our chances to actually find the (local) optimum. Of course, K has to somehow depend on the network's size. We can pick a K heuristically, using tuning etc. We make the following remark that might give to the reader some insight:

Remark 1 *If the improving state is only one (a worst case scenario) then we will find it after testing on average half of the available swaps.*

Proof

Suppose that in a state we have x improving swaps. The expected time until one is sampled is:

$$Es = \frac{2x}{N(N-d)} + \sum_{i=1}^{\frac{N(N-d)}{2}-x} i * \prod_{k=0}^{i-1}(1 - \frac{x}{\frac{N(N-d)}{2} - k}) * \frac{x}{\frac{N(N-d)}{2}} - i$$

Now, let $x = 1$ and $a = \frac{N(N-d)}{2}$. We have:

$$\frac{1}{a} + \sum_{i=1}^{a-1}(i+1) * \prod_{k=0}^{i-1}(1 - \frac{1}{a-k}) * \frac{1}{a-i} = \frac{1}{a} + \sum_{i=1}^{a-1}(i+1) * \prod_{k=0}^{i-1}(\frac{a-(k+1)}{a-k}) * \frac{1}{a-i}$$
$$= \frac{1}{a} + \sum_{i=1}^{a-1}(i+1) * \frac{1}{a} = \frac{1}{a} + \frac{1}{a}[\frac{a(a+1)}{2} - 1] = \frac{a+1}{2} = \frac{\frac{N(N-d)}{2}+1}{2} \quad (3.7)$$

We will not focus so much on that since in the dynamic setting we will not have the time to actually perform the full optimization, but rather a few swaps. Therefore we care more about performing well during the first iterations of the algorithm, for which we expect to sample more often improving states. K can be as large as application/environment related time constraints allow. The detailed description of our algorithm follows.

Algorithm 4: Random Swaps

input : N, M, d, D, T

output: Node cluster membership assignment

```
1.  c ← Initialize_Clustering(N,M,d);
2.  swaps ← Generate_possible_swaps(c);
3.  tested_swaps ← 0;
4.  while tested_swaps < K and time < T do
5.      pick a swap s in random;
6.      if Objective_Change(D,c,s) > 0 then
7.          Perform_swap(c, s);
8.          swaps ← Generate_possible_swaps(c);
9.          tested_swaps ← 0;
        else
10.         remove s from swaps;
11.         tested_swaps ← tested_swaps + 1 ;
        end
    end
```

Stochastic iterative improvement via Best k candidates

The previous algorithm achieves exploration in a relatively faster way, however looses a lot of time to find an improvement resulting swap. The idea behind the *Best k candidates* method is that we can pick K swaps that present a potential of improving the objective value of the current state and stochastically perform one of them. This

is controlled in a way since we take more drastic steps in realizing what might be a good swap, but it remains random enough through the stochastic step. There are two things to notice here. First, we can only employ heuristics in the decision of the K candidates. Second, some of the candidates might be states that decrease the objective function instead of increasing it. In order to maintain a balance between exploration and exploitation we decide to allow swaps that do not improve to be selected, but with lower probability of them being selected. This idea follows more the paradigm of other stochastic approaches like simulated annealing, hesitant adaptive search etc. Compared to the former, our selection of K candidate swaps creates the "boundary" between exploration and exploitation. Larger K values allow more exploration. The process terminates when no candidate improves the objective. *Algorithm 5* presents the k best candidates method.

Algorithm 5: Best k candidates

input : N, M, d, D, T

output: Node cluster membership assignment

```
1.  c ← Initialize_clustering(N,M,d);
2.  candidates ← Generate_k_best_candidates(c);
3.  repeat
4.      foreach candidate s do
5.          probability of improvement p_s ← e^(Objective_Change(c,s) / Objective_Value(c));
        end
6.      p ← p/sum(p);
7.      create candidates probabilities p of improvement;
8.      Pick a candidate s at random using p;
9.      Perform_swap(c, s);
10.     candidates ← Generate_k_best_candidates(c);
    until time > T or no candidate is improving or a (unique) clustering has
     been visited twice;
11. return best clustering so far;
```

First important part to discuss is the swap generation heuristic. Our rational goes

as follows: If a node is better of by moved to a new cluster Y, then to allow such a move we have to consider another node from cluster Y to swap it with. A node from a cluster U has a motive to move to a new cluster Y if the ratio $\frac{\text{Traffic exchanged with cluster Y}}{\text{Traffic exchanged with cluster U}}$ is greater than 1. Our approach is totally greedy. We calculate all the $N \times M$ ratios and we pick K node-cluster pairs (excluding pairs of nodes paired their current cluster) that provide the highest such ratios, irrespective of whether the ratios are greater than 1. Then for each one of them we decide which node of the destination cluster has greater ratio towards the origin cluster. In this way we have greedily decided on K swap pairs, our candidates. This is summarized in Function 6.

Function 6: Generate_k_best_candidates

input : c

output: K candidate swaps

1. ratios $\leftarrow$ new $N \times M$ matrix;
2. **foreach** *node* **i** *and cluster* **u**, $i \notin u$ **do**
3. $\quad \text{ratio}_{i,u} \leftarrow \frac{\text{Traffic exchanged between i and nodes of u}}{\text{Traffic exchanged between i and nodes in } c_i, \quad i \in c_i}$;

   end
4. nodes $\leftarrow$ k node-cluster pairs (i, u) with the highest ratio_{iu};
5. candidates $\leftarrow \{\}$;
6. **foreach** (i, u) *in nodes* **do**
7. $\quad v \leftarrow$ cluster of node i;
8. $\quad \text{j} \leftarrow \arg\max_{l \in u}\{\text{ratio}_{lv}\}$;
9. $\quad$ Add pair of nodes (i, j) in candidates;

   end
10. **return** candidates;

Next thing in our discussion, is the probability of improvement. It is constructed so that a) the probabilities of all candidates sum up to 1 b) it is based on the percentage of improvement of the objective value so that the exponents used remain tractable and finally c) the bigger the improvement the larger the probability. This follows similar principles with other stochastic optimization methods with main difference the consideration of all k prospective states in the calculation. This is reasonable

since we decide based on all of them instead of just one.

Now, overall this algorithm differs in nature from the previous ones. Because it allows for non improving swaps to be performed, we cannot guarantee that the final state is the optimal one among the ones seen. Therefore we need to have memory of the best so far state. Also, if we wish to perform changes in our network on the fly then we need to follow a unique strategy that fits our goals. We can either wait for the full optimization to be over before we reconfigure, perform reconfiguration when we find a new improving state (in which case though we might have to alter many links in our topology at once causing more network disruption) or explore ideas like Markov chain approximation described in [20], that prove quality of performance over time even though momentary decisions might not be ideal. What is more important though is that this process might loop between states, while it can also visit states that belong to the same equivalent class, something that had been avoided before. In the second case actually, the full process runs over an equivalent part of the search space that was visited before, which means that if we do not have a proper way to avoid that, the algorithm might even loop forever. This is obviously waste of time, since with some memory the algorithm could at least change the K candidates set in order to have more choices and chances to get out of the looping. We decide to simply keep track of visited states and when a state gets visited more than two times then we stop the process. (Again this is a heuristic based on the fact that we care more about the first few iterations and we could technically terminate the process based on other factors.) To account for the equivalent classes we observe the following:

Remark 2 *Given a specific (feasible) clustering* **c** *of the numbered network nodes, where clusters are identified by unique numbers, no matter the numbering of the clusters, we can uniquely represent it by re-labelling the clusters based on ascending order of their smallest elements (nodes). For any cluster membership assignment indicator matrix X, we can use Function 7 to transform it to this unique representation.*

Function 7: Get_Unique_Representation

input : X, N, M

output: (new) Node cluster membership assignment

1. $\overline{X} \leftarrow$ new NxM matrix ;
2. marked_nodes $\leftarrow$ new list of size M ;
3. clusters_seen $\leftarrow \{\}$;
4. Convert X in a vector C where C_i the cluster of node i ;
5. $i \leftarrow 1$;
6. **while** $|clusters\_seen| < M$ **do**
7. | **if** $C_i \notin clusters\_seen$ **then**
8. | | Add **i** in marked_nodes and C_i in clusters_seen;
9. | | $i \leftarrow i + 1$;
   | **end**

   end
10. $u \leftarrow 1$;
11. **foreach** *node* **i** *in marked_nodes* **do**
12. | Assign all nodes of cluster C_i to cluster u in $\overline{X}$;
13. | $u \leftarrow u + 1$;

    end
14. **return** $\overline{X}$;

To understand this function, notice that by processing the nodes in ascending order we can immediately know the smallest node in each cluster. Specifically, every time a node belongs to a cluster not seen before, it is the smallest node of this cluster and we mark it. Once we have seen at least one node of all M different clusters, which happens after processing $N - d$ elements in the worst case (the last d nodes are clustered together) and after M elements in the best case (if the first M nodes belong to the M different clusters), we are ready to relabel the clusters. Reading the marked nodes in order, we change the cluster membership assignment of all the nodes in their cluster to the next available cluster, starting from cluster 1. For example if node 5 is the first that belongs to cluster 6 and we have already had 3 other clusters seen,

then the new ordering the set of nodes that includes node 5 should be the cluster 4. Using appropriate data structures this can get as fast as $O(N)$, dominated mostly by the initial smallest node marking process[5].

Therefore, we can use this to test whether classes that we visit are equivalent or not. However, this adds extra computation operations to our algorithm. We can either calculate it for only some states that we visit, randomly, or just skip it completely, maintaining all the states that we visit neglecting equivalents, but reducing the allowed amount of times we can revisit a state. For large networks, we choose to follow the later suggestion.

Finally, we should mention that again K can be fixed, tuned or even network size related. Since we need to evaluate the improvement of all candidates and then probabilistically pick one of them, we decide to keep it small and fixed throughout the algorithm. To keep it analogous to the network size, we decide that a valid choice is to pick at least M (number of clusters) candidates so that all clusters might have the chance (not guaranteed or mandated by the algorithm) to contribute a node to a swap.

MinMax inter traffic link flow

We have already seen three iterative methods that aim to maximize the intra cluster traffic (thus minimize the inter cluster traffic). This is definitely a good goal, however we lack some guarantees in the process due to randomization, trapped to local minima etc. Specifically, without random point multi-starting, parallelization or time related consideration (long-term average performance guarantees) these methods are not very effective, yet more practical than others due to the small, naturally succeeding, changes. For this reason, we consider a new optimization objective, the minimization of the maximum amount of traffic exchanged between any pair of clusters. This in general is not the same with the inter-cluster traffic minimization, nevertheless is a load balancing technique good for achieving better traffic aggregation as well as re-

[5]In Function 7 as provided, the marking takes $O(NxM)$ time, but using extra memory we can maintain an N size vector containing in position i the label of the cluster of node i.

ducing network link utilization. With this. we attempt to give network performance guarantees, while maintaining the benefits the iterative methods offer.

Considering our previous methods, the adaptations are small. We will just need to maintain a matrix that contains all the $M \times M$ aggregate traffic amounts and update it through time, based on the swap performed. We also need to keep track of the maximum traffic value in the matrix. Using random swaps technique or allowing non-zero probabilities only to improving states when using best candidates approach, in every step we aim to reduce the maximum traffic value. Observe that if the current maximum is delivered from a cluster u to a cluster v, then if our next swap contains at least one node from either cluster u or v (a swap of one node from u and one from v is allowed) we are guaranteed to reduce the current maximum.

The simplest implementation to follow is the same with the random swaps approach, apart that now, we have less choices of valid swaps ($d(N-d)$). This can be faster, but performs more computations for the book-keeping required.

Deterministic methods

We also explore two deterministic methods. Since there is no fast way to find an optimal clustering right away, we employ greedy heuristics. The first method has as objective the maximization of the intra cluster traffic and the other one a load balancing across clusters approach. For them to be useful two points need to be achieved. First of all, since our optimization methods are relatively slow, by using deterministic heuristics we wish to have a rapidly available solution. Of course if this solution presents some good properties in for example the average case, then the respective algorithm is actually a good candidate. Finally, if fast enough, they might provide good starting points for our iterative algorithms.

Nonetheless, deterministic approaches are not ideal for our case because of the nature of our problem. The reason is that they do not account for the current state of the network, which can lead to a lot of changes during the reconfiguration process. For example there is a possibility that a set of nodes that belong to the same cluster, get relocated to a new cluster, thus get disrupted. This could have been avoided if the

algorithms considered the current state. Also, to work around that, the algorithms would be further more complicated, making the use of iterative approaches unavoidable. Despite that, both of our heuristics are intuitive and worth our time to find out about their performance.

Greedy Nearest Neighbor

This is one of the simplest intuitive approaches towards the cluster's formation. Imagine we decide to form the clusters one by one. We start with the first cluster. A way to fill it is by picking a node, the leader, and ask this node to pick d-1 nodes it prefers to be paired with. The node is going to pick the neighbors with whom it exchanges a lot of information, its nearest neighbors. We then move to the next cluster and so on. So, obviously, the quality of the clusters depends on the cluster's leader. And this is the main idea of our algorithm.

Randomly picking M leaders could be a choice, but then the quality of the clusters might vary and the method is not deterministic anymore. To realize good leaders we can look at the traffic matrix. Ideally, leaders are going to be somehow critical nodes that, if misplaced, the clustering quality degrades substantially. In a way, they are node for which it is obvious that we could have done better by assigning at least them right. For example nodes that communicate only with few other nodes compared to the general case, even if the traffic they exchange is little, might be potential leaders. This can be revealed by the variance of the traffic they exchange (mutual traffic) with other nodes. High variance implies that they exchange lot of traffic with some nodes and very little with others, so it is better to put them together with the group they prefer. On the other hand, nodes with high variance, even if they exchange huge amounts of traffic, probably they send similar amount of it in all their communication links, and, since they cannot be placed in a cluster with all the network nodes, they feel indifferent about which their neighbors are going to be. Note, that we use the mutual traffic, since it takes into account both directions of communication. The algorithm is described in Algorithm 8

Algorithm 8: Greedy Nearest Neighbors

input : N, M, d, D

output: Node cluster membership assignment

1. Mutual $\leftarrow D + D^T$;
2. **for** *every cluster* **u** *in* $[M]$ **do**
3. Compute variance for each line of Mutual;
4. leader $\leftarrow$ node (line) with highest variance;
5. $c_u \leftarrow$ leader$\cup\{$d-1 nodes i with highest Mutual$_{leader,i}\}$;
6. Remove from Mutual traffic that corresponds to nodes in c_u ;

end

This requires $O(N^2)$, which is the time to calculate the variance. Keep in mind that in every step we remove the nodes that got clustered together, reducing the computation needed in the next step.

Greedy pairing

Basic motive behind this algorithm is to greedily assign pairs with high mutual traffic into the same cluster and at the same time maintain similar load balance the intra cluster traffic within each cluster. This was an idea mentioned in the first prototypes of RHODA [87]. Briefly the idea is to calculate the mutual traffic between any pair of nodes, sort the pairs based on this value in descending order and finally assign them into clusters in a cyclic manner. Of course this cannot achieve optimality since we greedily place pair of nodes in clusters based only in the traffic exchanged between them, without assessing the traffic they exchange with nodes in the cluster of placement for example. The details of this process are described in *Algorithm 9*

Algorithm 9: Greedy pairing

input : N, M, d, D

output: Node cluster membership assignment

```
1.  nodes ← {};
2.  Mutual ← D + D^T;
3.  current cluster c ← -1;
4.  Sort Mutual in descending order;
5.  while |nodes| ≠ N or all Mutual processed do
6.      (i,j) ← next largest mutual traffic pair;
7.      if i and j assigned then
8.          Skip the pair;
9.      else if i and j unassigned then
            //in cyclic manner                                          ;
10.         c ← next cluster with space for a pair;
11.         if No c has space then
12.             Skip the pair;
            else
13.             Assign i and j to c;
14.             nodes ← nodes ∪ {i, j};
            end
15.     else if either i or j assigned then
16.         u ← cluster of the assigned node;
17.         if there is space in u then
18.             Assign the other node in u;
            else
19.             Skip the pair;
            end
    end
```

Due to the underlying sorting of the mutual traffic, and since mutual traffic is of order or $O(N^2)$, this algorithm is at least of order $O(N^2 \log N)$

3.3.3 Network scalability extensions

One of the key features of this architecture is that it is easily scalable. Specifically, having a tradeoff α between scalability and flexibility, we can split our cluster membership network into multiple small cluster membership switches, varying factor α from d (case of one big switch) to 1 (case of multiple small switches - the minimum value for α). However, as α decreases the network becomes less and less flexible. For example, suppose that Racks 1-8 are connected to the same cluster membership switch and $\alpha = 2$. Then clearly, nodes 1-3 (and any other subset of three nodes within those eight) will never be able to be in the same cluster, since there are only two nodes for each cluster assigned from this switch. Therefore, our algorithms need to address such a design.

Fortunately, there is a straight forward extension to all the aforementioned methods so that they are aware of this architectural design. Generally speaking this just reduces the size of our topologies search space by imposing some extra constraints. On the implementation side the way we treat these constraints is by limiting the choices the algorithm takes. For example when performing a swap, we cannot swap two nodes that are connected to different switches, since to perform the swap we would have $\alpha + 1$ nodes in the switch for the new cluster of each swapping node and $\alpha - 1$ nodes in their previous cluster. This violates the architectural design. So what we are left with are only swaps within the same switch, meaning that once a node is picked to be swapped, we can only pick its swap pair from nodes within the same switch.

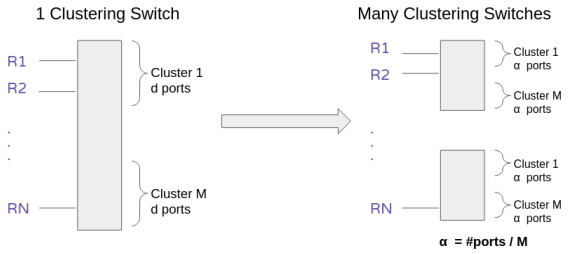

Figure 3-1: Clustering Network's Scalability

Clearly there are tradeoff that this scalability flexibility imposes, one of them

being that despite we are restricting the set of our available topologies, we also have faster algorithmic performance, thus we gain in optimality.

3.4 Time Model

In the dynamic system setting the basic thing that changes is the traffic matrix. We now have a matrix of traffic demands defined by D(t) where t is the changing time. Of course, given that the traffic arrivals are of a millisecond scale, this matrix rapidly changes. Notice that we are observing aggregated traffic at the level of a rack so the aggregated traffic demands are going to change slower. Still though, we do not have the time to optimize fully and then reconfigure. Remember that the reconfiguration alone takes approximately 200 ms, thus the added time of our algorithm should not be very long.

Let's for a moment observe our system. We start with a random reconfiguration and we let the traffic unfold. Occasionally we run our optimization of the new traffic demands and we reconfigure our cluster membership network switches. Every time, we reach a configuration that serves our network demands slightly better than the previous one, and we go on like that. Notice that a satisfying configuration at time t-1 can be a starting point for our next optimization attempt. There are some key considerations to take into account. First of all, we would like our network to be relatively stable. This means that we should try and make changes as small as we can and that also we should be slightly hesitant before placing nodes in the same cluster, so that we can verify that our change is not an artifact of momentary bursts and is more likely to be stable for the near future. This contradicts our second consideration, that is that the fast changing traffic requires actions taken as fast as possible so that our network avoids congestion, momentary losses and keeps the latency low as the optical technology promises. Therefore, we should find a way to balance the running time and optimality trade-offs most of the algorithms face.

We claim that the stochastic improvement approaches described in the previous section are a good fit to provide this balance. The reason is the simple neighborhood

operation they entail, the swaps. Swaps are small changes that provide a smooth an natural advance between clustering assignments that prevents unneeded network disruption. The execution of those algorithms alone take into consideration the previous clustering assignments, thus we do not have to optimize from scratch. Finally, the time of the algorithm reduces to the time it takes to find an improving swap.

To be able to use swapping techniques effectively we need to make sure that impulsive swaps are avoided. There are two types of impulsive swaps. The one is an artifact of the network restarts, namely, moments in time that we "forget" about previous cluster membership assignments and start optimizing given a completely random clustering of the nodes. In this case, as we will discuss more in the next chapter, the iterative algorithms rapidly move nodes around in an effort to reveal a relatively stable cluster membership assignment of the network nodes. When network memory resets, we should aim to optimize for a little longer, before any reconfiguration is performed to avoid optical link instability. The second case impulsive swaps might be observed is when we swap due to momentary bursts. Notice that our overall approach to the traffic burstiness is not to accommodate bursty traffic using the clusters, but to give node clusterings that decrease the effect of such burstiness. Therefore, we would like to make sure that a pair of communicating nodes with high mutual traffic at the moment, is going to exchange significant amount of traffic in the near future as well. This can be achieved if the traffic matrix, instead of just informing us about the incoming amount of traffic, maintains a certain type of memory of the previous moments in time. This could be achieved if the traffic matrix is a description of pairwise directional running averages of the traffic. The reason we do not suggest traffic rates immediately, is that running averages can be adjusted to "remember" more the latest few moments in time, favouring the more stably communicating nodes. Nodes that have a continuous stable communication are going to have higher running average values, which means that the algorithms will tend to placed closer. This summarizes our observations on why and how swaps should be used.

The framework we propose could be summarized in the following points:

1. Assume a traffic matrix in terms of running averages.

2. Choose an improvement algorithm based on swaps as the main optimization algorithm.

3. Given the *current* clustering of the network nodes, use the optimization algorithm to perform swaps until a better network state (at least one improving swap) is reached or time x has elapsed.

4. Adjust the time elapsed x to account for either impulsive swaps or optimal network configuration. The latest relates to the fact that when the network operates under a near optimal solution we can stop our optimization process temporarily and resume once a major network change is in effect.

A possibility to consider instead or along with the running average based traffic matrix is also traffic forecasting. Despite previous efforts that observe that it is hard o predict the fast changing traffic in server based resolution, the diurnal patterns present in data centers, as for example [75] reports, might enable some higher resolution forecasting. This could be a rack traffic forecasting or even more abstractly a specific traffic type or network operation expectation. There has been some literature already and we encourage the reader to explore it more. For now it is not among our main concerns, but it is a possible future path.

As we learned from our discussion in the first chapter around the Data Center deployments, there are definitely a few more things to note here. There have been multiple efforts to construct data centers that respond to network changes in real time. We primarily focused on clustering algorithms, however, effective network operation also relies on virtual machine placement and migration, load balancing and network resilience techniques etc. We would be way too optimistic to believe that clustering alone is going to be the ultimate solution to all those problems the aforementioned techniques tackle. For this reason, we believe that for our clustering methods to be effective a controller aware of the actions of all optimization units (for example the unit that optimized the machine migration policies etc) is needed. This does

not necessarily imply joint optimization, but rather a better synchronization of the system so that optimization of one aspect of the network does not lead to unexpected performance degradation of another. Not only that, but also this can improve and simplify the optimization goals of the various such entities. Consider as an example a virtual machine placement and migration aware implementation of RHODA. RHODA has the scalability factor α that not only suggest a topology reconfiguration flexibility - scalability of the network, but also can be a barrier to the maximum network size. On the other hand a very large network instance would be harder to manage and optimize in a ms timescale. At the same time, virtual machine migration and placement on a dynamically changing clustering topology might cause many impulsive and uninformed swaps. What we could do instead is to maintain and interconnect based on the inter cluster specification of RHODA, small cluster membership network blocks. Nodes in each block can only be clustered with nodes in the same block. These clustering blocks now can be small enough for the clustering optimization to performed fast but at the same time act as an independent, large capacity network machine. The optimal VM placement or migration can then be simplified by placing and migrating machines among only these blocks. The simplification comes from the fact that the blocks have higher capacity and faster operation time compared to the VMs requirements. Then each block is responsible for maintaining its own optimal performance without any interference with the VM allocation optimization process. This example just illustrates the possibilities a more overall aware network operation has to offer.

A final remark to be made has to do with our choice of a centralized optimization approach. As it has been observed many times, in centralized approaches it is harder to inform a controller in real time of all changes. On the other hand though, decentralized approaches tend to require a lot of signaling that generally occupies the network links with extra traffic, possible bursty. A solution though that aims to decrease the traffic burstiness effects would not make sense to be entirely distributed for example. Deployments like the one presented in the example above can also partly provide some "decentralization". We believe that similar choices, that do not depend

on signaling are more preferable and should be further explored.

This summarizes the perspective and ideas we adopt under the dynamic modeling assumptions. In Chapter 4 we present some simulations' results that support our claims and complete the study of the dynamic cluster membership assignment as adopted by the RHODA architecture.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 4

Simulation and Results

To evaluate the performance of the algorithms, ideally, for any traffic matrix, we would compare the final clustering to the actual optimal clustering. However, since the problem is hard, we cannot actually know the optimal solution for any traffic matrix in the scale of interest. For a valid comparison, we need to run the algorithms on properly constructed traffic patterns for which we know the optimal solution.

There are three points we want to test our algorithms over. The first is the case where the traffic pattern is naturally structured into equally sized parts. In this case we would expect our algorithms to recognise this fast and obtain the clusters. Similarly, the second case is the trivial case where all clusterings are optimal (or the majority of them). It would be ideal in this case the algorithms to stop as fast as possible without cycling over similar solutions. Finally, we seek to learn how the performance of the algorithms degrades as the structure changes from equally sized perfect clusters to totally unstructured random traffic. Of course performance on real life data would complete the evaluation of our system, but this is a harder task we will discuss later.

In this section we will perform two types of comparisons. In the first we seek to compare the performance of our two major iterative algorithms, the Random Swaps (*Algorithm 4*) and the Best Candidates (*Algorithm 2*). Both of those algorithms are a good fit to the dynamic system setting for the various reasons we closed Chapter 3 with. However, which one is better, if any, has to do with three key points: millisecond

improvement performance, iteration time and finally dynamic adaptability time. The latest explores the performance from a given, relatively good, starting point after a small traffic matrix perturbation. This is important since it simulates the expected real time performance.

Our second comparison aims to discover the potential of the deterministic methods over the iterative ones. The huge search space of the iterative approaches might be a barrier to the methods' scalability. In this case we explore the possibility of deterministic algorithms to be preferred just for scalability reasons. Notice that a few swaps after any deterministic algorithm have the potential to improve the clustering quality. However, when using deterministic methods we miss clustering assignment through time continuity, that might result in substantial network disruption.

In what follows we present the synthetic data matrices used in our simulations, the comparison of the iterative methods and the overall comparison of the methods proposed in this work. Through out this section we adopt the following network structure assumptions:

1. The cluster membership network consists of one large switch.
2. Unless discussing scalability, the typical number of Racks is 1024 each one supporting approximately 40 servers resulting in a 40960 servers network.
3. The clusters have always size fixed to 32, as the amount of ports of commercialized AWGRs.
4. Inter cluster traffic based comparison is analogous to intra cluster traffic based comparison due to the equivalence of the optimization problems.

Finally, our algorithms and simulations are implemented in Python 3 and are available in author's GitHub repository.

4.1 Traffic patterns used

To be able to address the performance of the algorithms in response to different traffic patterns, we define four different types of traffic. For each one of them we generate

25 random traffic matrix instances. Every structured instance generation follows by a random shuffling of the nodes, so the underline structure does not remain obvious.

Unit out clustered traffic

All the out of cluster traffic flows from the nodes get value 1. Then in blocks of (dxd) diagonally, we assign values between 7-10 uniformly at random. The optimal inter cluster traffic is obviously the sum of all values that equal to 1 in the matrix, or $N(N-d)$

Two level clustered traffic

Initially, all the out of cluster traffic flows from the nodes get values between 1 and 3 uniformly at random. Then in blocks of (2dx2d) diagonally, we assign the value 6. After that, blocks of (dxd) diagonally have a value between 7-10 assigned uniformly at random between. This traffic is naturally clustered in clusters of size 2d. We artificially impose a dxd clustering as well so that we can calculate the optimum. The optimal inter cluster traffic is obviously the sum of all values < 7 in the matrix with average value $6 * N * d + N(N-2d) * 2$.

Uniform traffic

In this traffic pattern all the traffic flows get values between 1 and 10 uniformly at random. The optimal inter cluster traffic has average value $\geq N(N-d)5$.

Finally all diagonal elements (traffic sent from node i to node i) get value 0.

4.2 Iterative methods comparison

Since we deal with two randomized algorithms we are faced with the following challenge. On the exact same data the algorithms can have different performance and results. To tackle this, focusing on 1 randomly generated traffic matrix from each traffic pattern class, we evaluate each one of the algorithms on a few runs. Each algorithm execution starts with a random clustering and goes on until the algorithmic

stopping criteria are met. At this point we do not impose any time constraint, we just observe the algorithms as they converge. In order to keep the comparison fair, we use the same random clustering starting points for both of the algorithms.

The first thing we will look at is the convergence of the algorithms. This is to test the hypothesis that at the start of the search the percentage of objective value improvement is higher and or faster. For all traffic types we repeat the two algorithms for 5 different random starting points. We then present the cumulative intra cluster traffic improvement. In the case of Unit out clustered traffic and Two level clustered traffic, every time we improve over the previous objective value, we keep track of the percentage of optimal we have achieved so far (cumulative percentage), which can be calculated as $\frac{\text{current value}}{\text{optimal value}}$. On the other hand, for Uniform traffic, since we do not know the optimal solution, we do the same but using the formula $\frac{\text{current value}}{\text{Maximum / final intra traffic value}}$. Basically, we just keep track of the cumulative percentage of improvement compared to the final value, thus how fast the algorithms achieve the best state they can (therefore the 100% in this case is always achieved but does not reflect how close we get to the optimal). We then plot the percentage over time in ms scale, with each different line representing each different starting point. Notice the following. The iteration step of the Random Swaps algorithm can be arbitrarily long (search swaps until an improving one is found - on average N(N-d)/4 worst case) while the iteration step of Best Candidates algorithm is almost fixed but might not always lead to improvement. For fair comparison we make the assumption that reconfiguration is performed only when an improving clustering is found. For this reason when it comes to the Best Candidates algorithm, we keep track of the improving states and the time it takes to be found. This might give the impression of less iterations of the algorithm, but this happens because we do not account for iterations that do not lead to improvement. Figure 4-1 presents the full simulation results in s scale and figure 4-2 the first 1 s (1000ms) of the simulation.

The results suggest that the overall performance of the algorithms, under the assumption that we only reconfigure for improving swaps, is similar. As we can see, the running time of the Random Swaps algorithm is larger, an indication that we

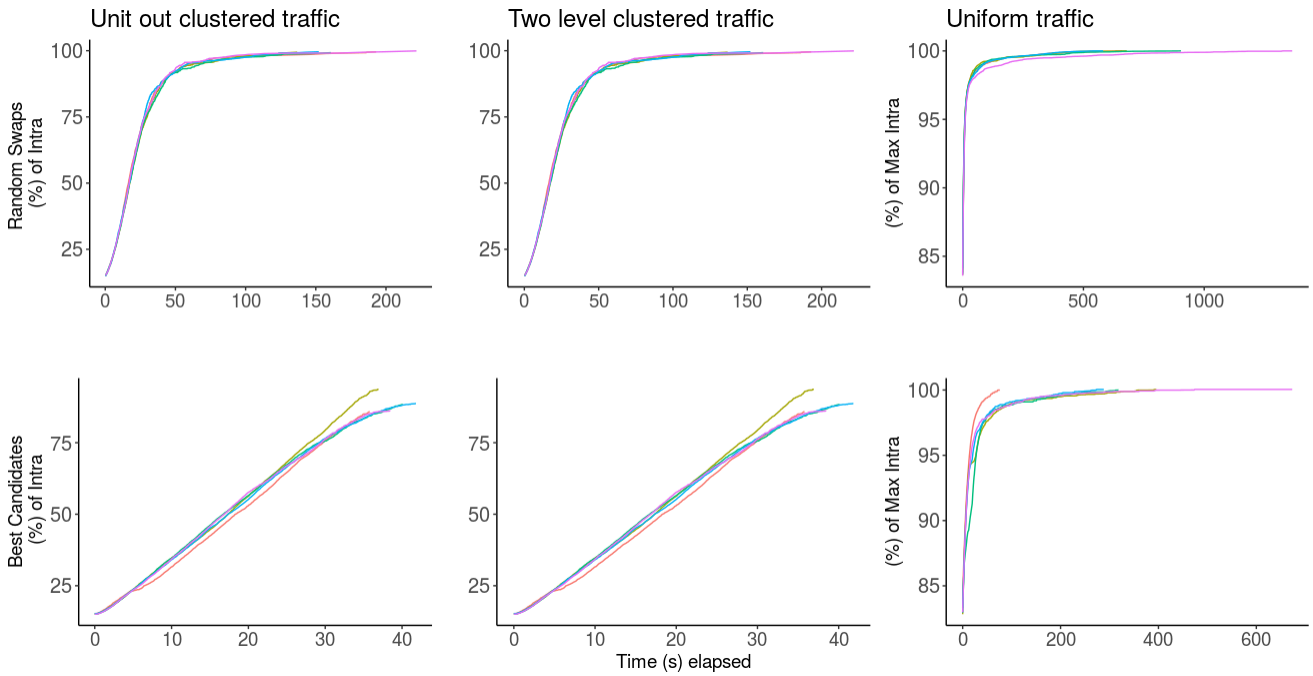

Figure 4-1: Change of intra cluster traffic achieved through the execution time of the algorithms for three different traffic types. Random swaps perform better but with substantially slower convergence rate.

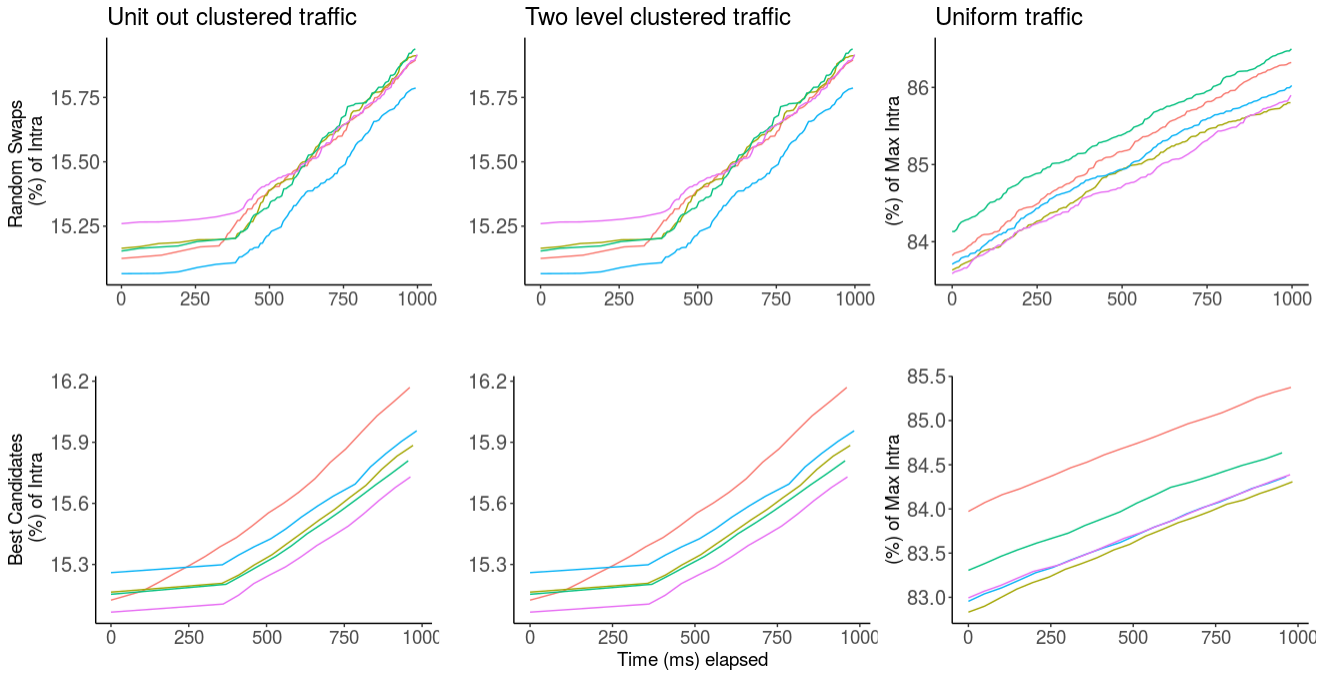

Figure 4-2: Change of intra cluster traffic achieved within the first second of the algorithms for three different traffic types. Results suggest similar performance guarantees and slow rate of improvement for both algorithms. The improvement is indicated by the slop of the curves.

may need stricter stopping rule. This is however the reason that the final clustering is optimal on the contrary to the Best Candidates that achieve around 80% of the optimal. What is important to notice though, is that most of the improvement happens within the first 40-50 seconds of the algorithm, while later the search gets harder. This should create a warning in the following sense: as we get closer to the optimal, the improving states are less, thus the improving swap search time increases a lot. This suggests that when we are our network operates closer to the optimal then the running time of the algorithm can potentially increase, even if we would like to perform only a few swaps. This is mostly a warning regarding the realization we get over these methods, however we should probably not worry about it. When our network operates close to optimal then we would possibly like to reconfigure less times and only if substantial changes happen. What we could therefore do is to pause our reconfiguration planning for a while and return back to it when the traffic matrix has more changes we need to incorporate. On the other hand, when we are under-performing in terms of clustering, the algorithms tend to require a very rapid reconfiguration including multiple unneeded swaps that aim to help to the node redistribution. In this case, we can perform the swaps virtually, and reconfigure when more nodes seem have received a stable cluster membership assignment.

An important aspect of both methods is the time per iteration. For the case of Best Candidates, we assume as iteration the time to get the next improving clustering. As it is obvious already from the figures, the initial iterations happen very rapidly, while the later ones get increasingly slower. We care both for the initial and for the later iterations for the reasons we realized above. To compare the algorithms we plot for each of those traffic types the box-plots of the time per iteration, in μs scale, for only the first random starting point. This should not change significantly from one starting point to another. Figure 4-3 summarizes our findings.

The time does not depend so much on the traffic type as on the algorithm. Specifically, Best Candidates takes almost twice the amount of time per iteration because a) non improving steps are allowed, but, more importantly, b) due to the fixed amount of objective value evaluations. The difference it is mostly caused by the increasing

number of initial rapid iterations of the random swaps, and this can be verified by the amount of points out of the box. These present the increasing amount of time an iteration takes when we get closer to the optimum. There is no clear preference in this case, because random swaps are faster but can also cause instabilities due to higher iteration time variation.

Over all, at least in the time scale of interest, the two algorithms seem to be fairly similar. A disclaimer in the above results is that the machine used for the simulations is relatively old. Thus, it is expected that running time should be times of order smaller in an enterprise server with a parallelized implementation.

Finally, the scalability of both algorithms is presented in Figure 4-4. Since we care about the time it takes to perform a few algorithm iterations, we again present the box-plot of the time per iteration and for different network sizes. We observe that Random Swaps have always faster iterations in the start, which get larger as the algorithms proceeds. On the other hand, Best Candidates iteration time gets more affected by the network size due to more objective function evaluations necessary (the amount of candidates depends on the network size). Both perform relatively well if we consider a few iterations before every reconfiguration of the network, but there are some drawbacks to consider. Specifically the variation in iteration time is way smaller in the case of Best Candidates algorithm, but comes with a cost of higher mean (time per iteration). To conclude, both methods seem to fit our goals.

4.3 All methods comparison

Heuristic deterministic approaches are a fast scalable alternative to our optimization goals. The resulting clustering is not expected to be optimal, but it can be available fast in larger scale. Their execution time though is comparable if not slower that a few iterations of the iterative methods we consider. Therefore we seek to realize whether there is any case that a deterministic algorithm would be preferred over the iterative ones. Our main hypothesis is that if any of the deterministic algorithms is able to provide good clusterings on average and has competitive execution time,

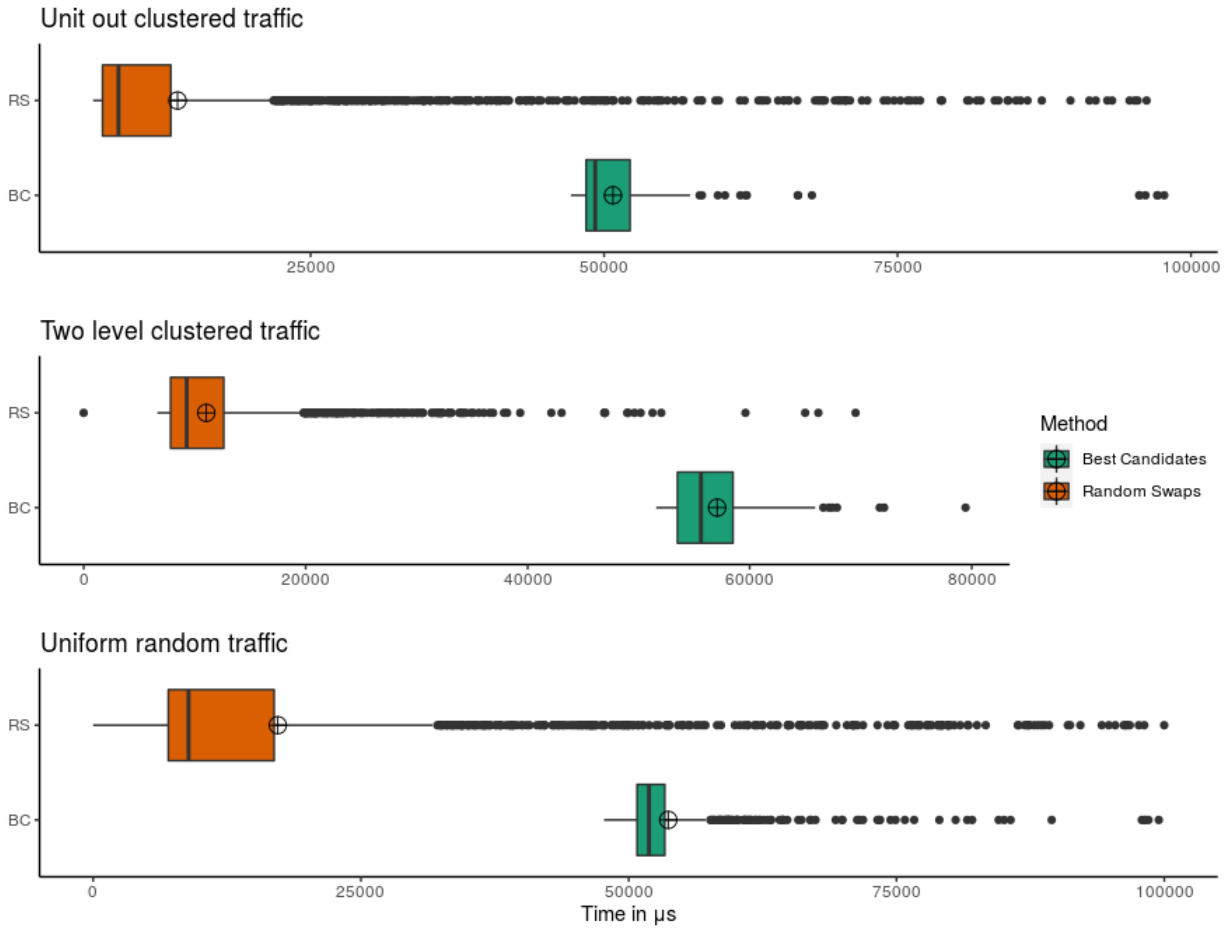

Figure 4-3: The iteration time boxplots. For clearer value resolution we use μs scale.

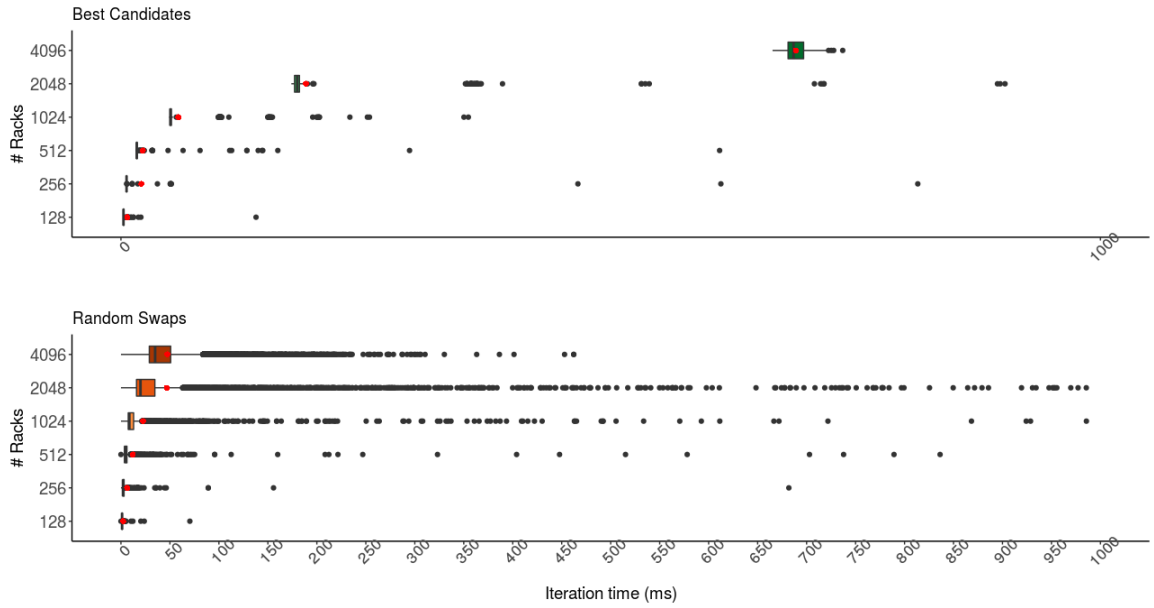

Figure 4-4: The iteration time boxplots for different network sizes. Traffic used for the evaluation is the Unit out clustered traffic. For clearer value resolution we use μs scale.

then we can skip our optimization efforts and focus on when reconfiguration should happen instead.

For a fair comparison, we evaluate the performance of our deterministic algorithms, Greedy Pairing (*Algorithm 9*), Greedy Nearest Neighbor (*Algorithm 8*), and Spectral Clustering with re-balancing, on twenty five (25) randomly generated instanced of each traffic type and for size network sizes of 128, 256, 512, 1024, 2048 and 4096 Racks respectively. Then we compare their average % of optimality and average execution time, with average taken over the 25 different instances.

The average execution time is presented on Figure 4-5 and Figure 4-6 (due to huge difference in scale). Notice that there is no notion of iteration here, so each algorithm execution counts as one iteration. The barplots present the results for increasing network size from the left to the right and from top to bottom. For the case of 1024 nodes the average running time is close to a few iterations of our iterative algorithms for the case of the Nearest Neighbors, while it gets substantially larger in the Greedy Pairing case. Generally, greedy pairing appears to take longer when network size is smaller and vice versa. Since the Nearest Neighbor implementation leverages the

numpy library matrix implementation for some of the critical calculations, we suspect that the difference is due to the optimized implementation. Finally, Spectral Clustering takes substantially longer time than both of them. Notice that unfortunately, on the contrary to the iterative methods, the execution of the algorithms cannot be broken into small reconfiguration operations (like the swaps), at least the way we have defined them, and even if they can, the partial changes are not guaranteed to be optimal.

This brings us to our second consideration, the percentage of optimal intra cluster traffic achieved - using the formula presented before - for the three traffic types we know the optimal for (all but the uniform random traffic - label 3). This will complete our realization towards whether or not the running time can be tolerated. Figure 4-7 summarizes our findings. The best for the case of clustered traffic is Specral Clustering, however is slow, so not ideal. The good news is that the Nearest Neighbor algorithm manages to well adapt to clustered traffic. As the traffic though gets more complicated (even if there is some underlying clustering) the performance of the algorithm decreases. As for the Greedy pairing we have to remind that it is more a load balancing approach rather than a minimization of inter cluster traffic. Interestingly enough, when the traffic is dense, it performs similar to the Nearest Neighbor algorithm. This probably implies that almost all clusterings have objective value close to the optimal, due to the density of the traffic.

To close our discussion regarding the performance of the algorithms, in case of uniform or even the exact same traffic amounts, none of the above algorithms is able to rapidly realize that all the clusterings are optimal and stop. The deterministic ones are going to perform their deterministic steps, while the iterative algorithms they will keep looping until all states are visited or a stopping condition is met, unless we define a specific stopping rule just for this case or require strict objective value improvement for the continuation of the algorithm. However there is no obvious way to know when we are faced with such traffic.

We believe that swaps are a better approach for our case, especially because of the dynamic network reconfigurability. As we veryfied in the previous comparison,

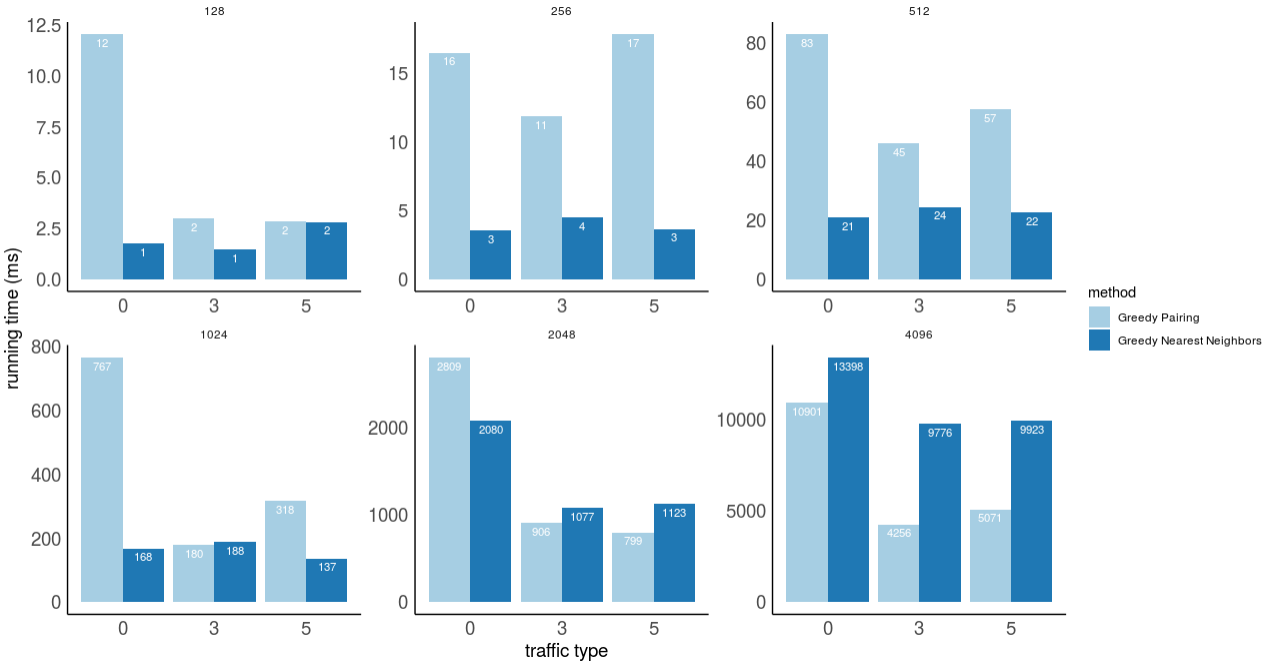

Figure 4-5: A barplot of the average execution time in (ms) of the two deterministic algorithms for different network sizes and traffic types. The traffic type labels are 0: Unit out clustered traffic, 3:Uniform traffic, 5: Two level clustered traffic. For all network sizes we assume cluster size d=32 Racks.

the first iterations take less than 700ms on average for all the network sizes up to 4096 nodes, on the contrary to the deterministic algorithms, that for larger networks take a few seconds to run.

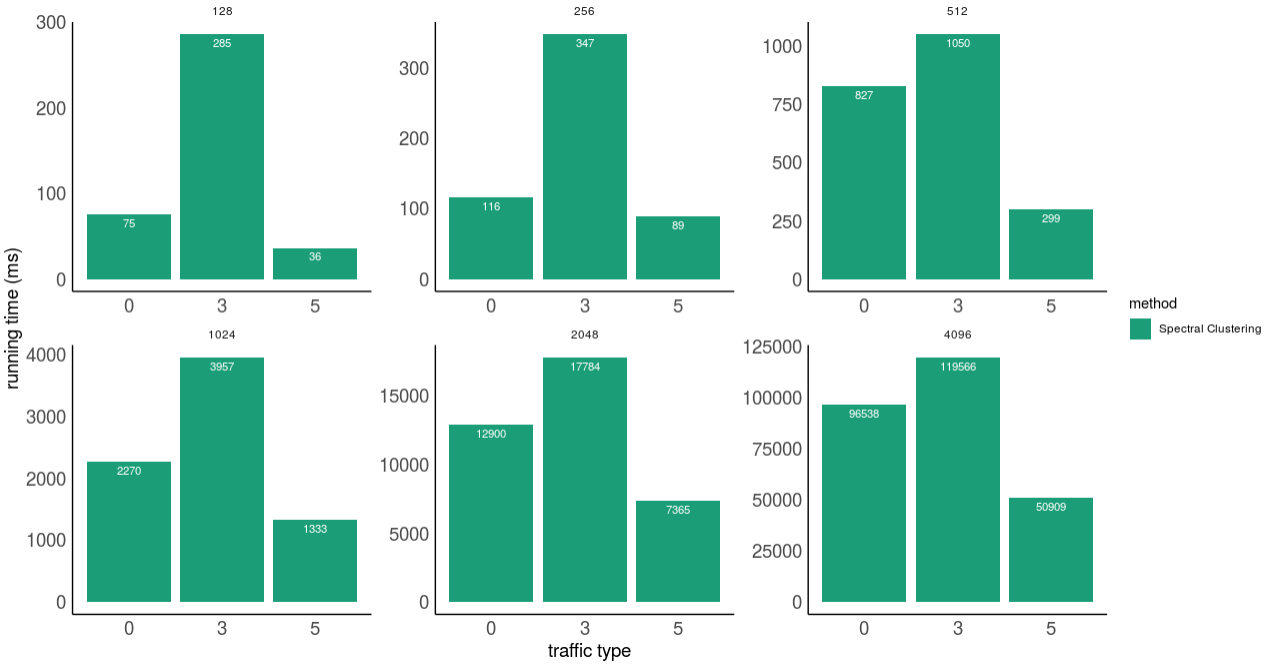

Figure 4-6: A barplot of the average execution time in (ms) of Spectral Clustering. The traffic type labels are 0: Unit out clustered traffic, 3:Uniform traffic, 5: Two level clustered traffic. For all network sizes we assume cluster size d=32 Racks.

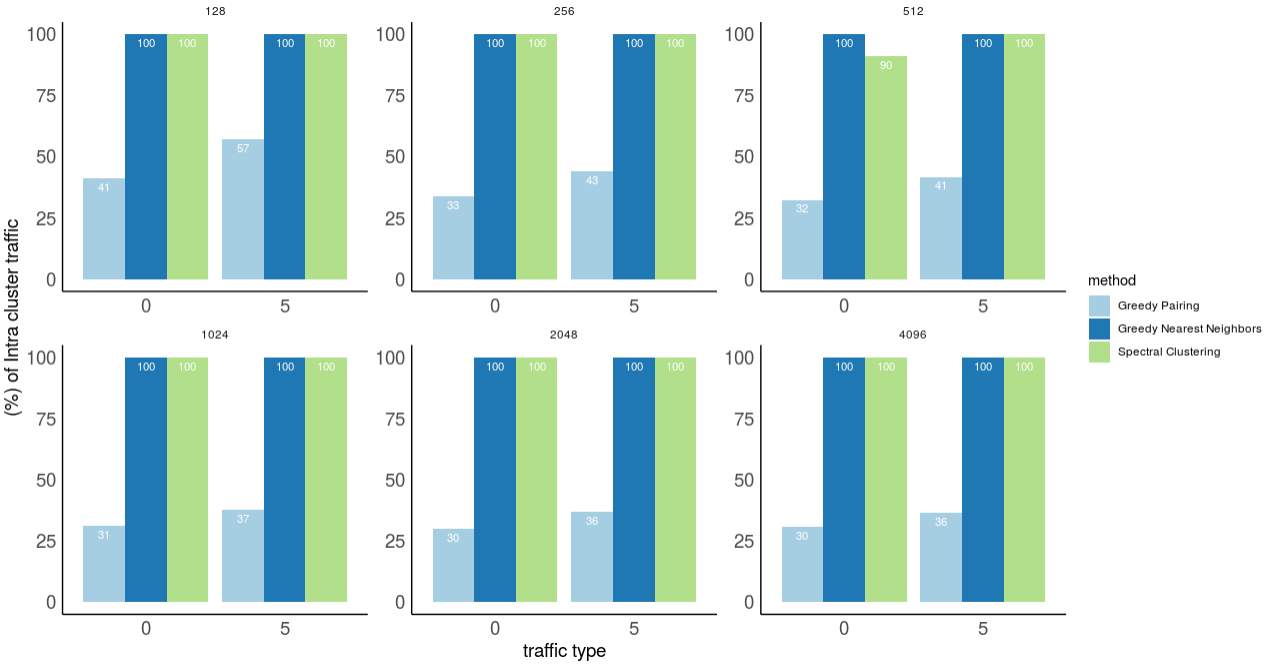

Figure 4-7: A barplot of the average percentage of optimal intra cluster traffic achieved by each algorithm for different network sizes and traffic types. The traffic type labels are 0: Unit out clustered traffic, 3:Uniform traffic, 5: Two level clustered traffic. For all network sizes we assume cluster size d=32 Racks.

Chapter 5

Conclusion

Networks have evolved through the years and are expected to be used in many new and creative ways in the foreseeable future. Some special entities that are part of the widest communication network we know today are called Data Center Networks and are charged with the storage, management and distribution of huge amounts of data that belong to users around the world. In their most recent version of interest, Data Center Networks are comprised of optical devices, that along with use of Wavelength Division Multiplexing present a promising path for low latency, high aggregation and low power consumption networking. Among the proposed Optical Data Center Network Architectures, RHODA, a two level hierarchical optical architecture, is based on the reconfiguration of MEMS switches to support dynamic node clustering in high speed and high aggregation clusters.

In this work we study the dynamic traffic based node clustering of RHODA. We first recognizing that the the underlying problem belongs to the Graph Partitioning Problems family and is an NP-hard combinatorial optimization problem. Then we propose and evaluate simple algorithms to support fast and scalable node clustering of the RHODA network nodes. In summary, we present two deterministic algorithms and three iterative ones, two of which are stochastic. We find that the stochastic iterative improvement approaches, based on simple pairwise node swaps, meet our goals the most. Concluding, fast algorithmic approaches come at a cost of reduced optimality.

THIS PAGE INTENTIONALLY LEFT BLANK

Appendix A

Graph Partitioning Summary

Table A.1: Graph Partitioning Summary Literature

Problem			Time Complexity	Approx. Ratio	Method	Ref
Min Cut			$O(\|E\|f_{max})$	1	Max flow - min cut theorem	[41]
			$O(\|V\|^{2a}\log^2\|V\|)$	a	Randomized / Edge contraction	[48]
Max Matching			$O(\|V\|^2\|E\|)$	1	Improvement via augmenting paths	[29]
			$O(\frac{\|E\|}{\epsilon}\log\frac{1}{\epsilon})$	1-ϵ	LP + dynamic relaxation of the feasibility and complementary slackness conditions	[27]
Min Bisection		trees	$O(\|V\|^3)$	1	Dynamical programming	[61]
		grids	$O(\|V\|^4)$	1	Dynamical programming	[38]
		general	$O(\|E\|\log\|V\|)$	$O(\log\|V\|)$	Multicommodity flows + decomposition tree	[73]
			O(poly)	$O(\gamma\log\|V\|) = O(\log^2\|V\|)$	Decomposition tree using amortized cuts + labeling using dynamic programming + recombine nodes to form bisection using the labeling	[35]
β-balanced Cut			$O(poly)$	$O(\log\|V\|/\epsilon)$	Multicommodity flows + sparsest cut	[55]
k-way Cut			$\|V\|^{(1+o(1))k}$	1	Multigraph creation + color coding +tree packing	[58]
$(k,1+\epsilon)-$balanced Partition	$\epsilon = 1$	general	O(poly)	$O(\log n\sqrt{\log k})$	Recursive partitioning the graph with approximate separator	[79]
			$O(\|E\|^2(\rho'-\rho)$[a]	$O(\log n\log\log n)\setminus O(\log n)$	Spreading metric + ellipsoid method	[31]
			O(poly)[b]+ $O(\|V\|k)$[c]	$O(\sqrt{\log\|V\|\log k})$	l_2^2 metrics for spreading metric	[53]
		planar[d]	$\widetilde{O}(\|V\|^3)$	O(1)	Graph decomposition	[37]
		grids	$\widetilde{O}(n^{1.5})$	$O(\log k)$	Graph decomposition using separators + repacking	[37]
	$\epsilon < 1$	general	$\max\{O(n(k/\epsilon)^{O((1/\epsilon)\log(1/\epsilon))})\}$, $O(\max\{\log n,\log\frac{1}{\epsilon}\}(n\frac{1}{\epsilon^2}+n^{4.5})\}$	$O(log^{1.5}\|V\|/\epsilon^2)$	Graph decomposition + repacking to form the parts	[6]
			$O(poly)$[e]	$O(\log\|V\|)$	Graph Decomposition + solving on the decomposition tree	[37]
		weighted trees	$O(\|V\|^4(k/\epsilon)^{1+3\lceil\frac{1}{\epsilon}log(\frac{1}{\epsilon})\rceil})$	≤ 1	Decomposition using dynamical programming + Repacking	[37]
	$\epsilon = 0$		Exponential on k	1	Decompose graph into trees + min bisection on the trees	[61]
				$\frac{k-1}{k}\|V\|$		[77]

[a] spreading metric relaxation computation complexity
[b] separation oracle in polynomially many constraints
[c] rounding complexity
[d] and generally graphs with excluded minors
[e] Time to find a family of decomposition trees

Appendix B

ILP for full RHODA reconfiguration

Assumptions

Multipath: A traffic flow can be transmitted through different lightpaths. In case two or more flows share lightpath time division multiplexing is applied.

Multihop: When a source s does not have the destination d as neighbor, the flow will use lightpaths with more than one hop.

Traffic aggregation: The traffic for nodes outside of a cluster is aggregated and treated as the "traffic of that cluster".

Bidirectionality: Although the links of the inter cluster network are bidirectional, they act as two unidirectional links, in the sense that they can have different capacities (number of assigned wavelengths

High Capacity Intra-Cluster Network: Nodes within a cluster form a complete graph and every node can communicate with every other node through high capacity links (no capacity constraints)

Wavelength Assignment: There are always enough wavelengths to support our network and the ones used in the intra cluster traffic do not interfere with those for inter cluster traffic. Also, given the number of wavelengths per link, there is always a way to assign wavelengths on the links (graph coloring on multiple edge graphs)

Switches: There is only one switch for the cluster membership network and only one for the (inter-)cluster network.

Fixed Parameters

W = number of wavelengths available in each node for inter cluster traffic communication

N = the number of nodes (ToRs)

M = the number of cluster (N/d)

d = number of nodes per cluster

k = the degree of a cluster node (number of neighbors in clusters' network)

C = traffic capacity per wavelength

D = traffic rate demand matrix where D_{ij} the amount of traffic from node i to node j

Variables

$c_{iu} = \mathbb{1}\{$node i is a member of cluster m$\}$

$a_{uv} = \mathbb{1}\{$cluster u is neighbor with cluster j, $(u, v) \in \mathcal{E}\}$

f_{sd}^{uv} = amount of traffic from cluster s to cluster d through link (u, v) in the clusters graph ≥ 0

nc_{ij}^{sd} = amount of traffic from node i in cluster s to node j in cluster d ≥ 0

l_{ij} = amount of traffic from node i to node j ≥ 0

L_{uv} = amount of traffic from cluster u to cluster v ≥ 0

w_{uv} = number of wavelengths on (inter cluster) link (u, v) $\in \{1, 2, .., W\}$

Constraints

Constraint 1 *General network constraints*

- There is neither edge nor flow from a node to itself.
  $a_{uu} = 0 \qquad f_{sd}^{uu} = 0, \qquad \forall\ u, s, d \in \mathcal{M}$

Constraint 2 *Topology constraints*

- In the inter cluster network topology of RHODA every node u has in_degree and out_degree at maximum k
  $\sum_v a_{uv} \leq k \qquad \forall\ u \in \mathcal{M}$
  $\sum_v a_{vu} \leq k \qquad \forall\ u \in \mathcal{M}$

Constraint 3 *Wavelength assignment constraints*

(**Note**: optical links in free space - use of MEMs - are bidirectional but act as two unidirectional links, meaning that we can apply different capacity - number of wavelengths - in each direction)

- Each cluster node has exactly W wavelengths available to send and receive traffic using the inter cluster network.
  $\sum_{v=1}^{M} w_{uv} \leq W \qquad \sum_{v=1}^{M} w_{vu} \leq W \qquad \forall u \in \mathcal{M}$
- Wavelengths can be assigned only to existing links.
  $w_{uv} \leq W a_{uv} \qquad \forall u, v \in \mathcal{M}$

Constraint 4 *Capacity constraint*

- The total number of flows going through each link cannot exceed its capacity.
  $\sum_{s=1}^{M} \sum_{d=1}^{M} f_{uv}^{sd} \leq C w_{uv} \qquad \forall u, v \in \mathcal{M}, u \neq v$

Constraint 5 *Traffic accommodation constraints*

- The amount of traffic of flows sent from cluster s to cluster d as well as the amount of traffic of flows received by cluster d with origin cluster s equals the traffic accommodated from cluster s to cluster d.
  $\sum_{v} f_{sv}^{sd} = L_{sd} \qquad \sum_{v} f_{vd}^{sd} = L_{sd} \qquad \forall\ s, d \in \mathcal{M},\ s \neq d$
- The traffic accommodated between any pair of nodes should not exceed the traffic demand between these two nodes.
  $l_{ij} \leq D_{ij} \qquad \forall\ i, j \in \mathcal{V},\ i \neq j$

Constraint 6 *Flow conservation constraints: The amount of flow that enters a node should equal to the amount of flow that exits that node.*

- A node u should not receive traffic originated from it.
  $\sum_{v=1}^{M} f_{ud}^{vu} = 0 \qquad \forall\ u, d \in \mathcal{M}$

- A node u should not forward traffic destined to it.

  $\sum_{v=1}^{M} f_{su}^{uv} = 0 \qquad \forall\ u, d \in \mathcal{M}$

- When node u receives transient traffic (neither originated or destined to it) it forwards it.

  $\sum_{v=1}^{M} f_{sd}^{vu} = \sum_{v=1}^{M} f_{sd}^{uv} \qquad \forall\ u, s, d \in \mathcal{M}, u \neq s, d$

Constraint 7 *Cluster membership constraints*

- One node can only belong to one cluster:
  $\sum_{v=1}^{M} c_{uv} = 1 \qquad \forall\ u \in \mathcal{V}$

- One cluster contains d nodes:
  $\sum_{u=1}^{N} c_{uv} = d \qquad \forall\ v \in \mathcal{M}$

Constraint 8 *Traffic aggregation constraints*

- There is no traffic from a cluster to itself:

  $nc_{ij}^{vv} = 0 \qquad \forall\ s, d \in \mathcal{V}, v \in \mathcal{M}$

- A node s only sends flow to the cluster aggregation point if it is destined for a node d out of its cluster:

  - Node s should belong in cluster u:

    $nc_{sd}^{uv} <= W * C * c_{s,u} \qquad \forall\ s, d \in \mathcal{V}, u, v \in \mathcal{M}$

  - Node d should belong in cluster v:

    $nc_{sd}^{uv} <= W * C * c_{d,v} \qquad \forall\ s, d \in \mathcal{V}, u, v \in \mathcal{M}$

  - The traffic from s to d through the inter cluster network equals the traffic accommodated from s to d:

    $nc_{sd}^{uv} <= l_{sd} \qquad \forall\ s, d \in \mathcal{V}, u, v \in \mathcal{M}$

- The outgoing traffic from cluster u to cluster v equals the sum of the traffic from the nodes in u to the nodes in v:

  $L_{uv} = \sum_{s=1}^{N} \sum_{d=1}^{N} nc_{sd}^{uv} \qquad \forall\ u, v \in \mathcal{M}$

List of Figures

Special Terms

(inter-)cluster network

A network with clusters as nodes. 115

bandwidth

The transmission capacity of an electronic communications device. 18, 19, 20, 21, 23, 28, 29

bisection bandwidth

The capacity of the links of the minimun cut over all possible network bisections. It is important metric of a network architecture since it is frames a prospective network bottleneck . 16, 18, 19, 31, 35

commodity switches

A cheap hardware switch with small amount of buffer memory and filter rules. 19

downlink

A link that transfers traffic to the end nodes.. 18

economies of scale

a proportionate saving in costs gained by an increased level of production. 21, 23

insertion losses

The loss of signal when traveling in a component and out of the component. Measured in decibels (dB).. 21

k-regular graph

A graph where each vertex has k number of neighbors. 35

light path

Path between two optical network over which light passes through unmodified . 24, 26

light beam

The directional projection of light energy radiating from a light source. 21, 23, 24, 25, 26

multiplexing

Combination of many signals into one signal over a shared medium. 21, 24, 28, 29

photodetector

Sensors of light or other electromagnetic radiation. 23

photoelectric effect

the emission of electrons when electromagnetic radiation, such as light, hits a material. 23

Rack

A cabinet that packs together 20-40 servers and a switch to interconnect them. Used for organization, security and heat control purposes. 13, 17, 36, 37, 92

switching time

The time taken by a switch to go from a state to the other. 21

traffic flow

Data that need to be sent from a source node (s) to a destination node (d) and follow the same path. (The total amount of data transmitted from s to d might form more than one flows). 115

transceiver

a device that can both transmit and receive communications, in particular a combined radio transmitter and receiver. 23, 25, 29

uplink

A link that transfers traffic from the end nodes.. 18

wavelength

The spatial period of a periodic wave / the distance over which the wave's shape repeats. 22, 27

wavelength spacing

Difference between adjacent wavelength channels. 28

THIS PAGE INTENTIONALLY LEFT BLANK

Acronyms

AWG Arrayed Waveguide Grating. 27

AWGR Arrayed Waveguide Grating Router. 25, 27, 28, 33, 34, 37, 38

CWDM Coarse WDM. 23, 28

DCN Data Center Network. 13, 17, 18, 30

DWDM Dense WDM. 23, 28

EAM electro-absorption modulator. 22, 23

EPS Electronic Packet Switching. 24, 31

LCoS Liquid Crystals on Silicon. 26, 27

LED Light-Emitting Diode. 22

MEMS Micro-Electro-Mechanical Systems. 25, 26, 27, 31, 33, 35, 36, 37

MZI Mach-Zehnder Interferometer. 23

OBS Optical Burst Switching. 24

OCS Optical Circuit Switching. 20, 21, 25, 31, 33, 35

OCS Optical Circuit Switch. 25

OPS Optical Packet Switching. 24

PoD Point of Delivery. 36

PSBPLC polymer stabilized blue phase liquid crystal. 26

ROADM Reconfigurable Optical Add-Drop Multiplexer. 24, 27, 29

SFP Small Form-Factor Pluggable. 23

SOA Semiconductor Optical Amplifier. 23, 26, 27

ToR switch Top of the Rack Switch. 13, 25, 34, 35, 36, 37, 70

VCSEL Vertical Cavity Surface Emitting Laser. 22

WDM Wavelength Division Multiplexing. 20, 21, 22, 28, 29, 30, 33, 35

WSS Wavelength Selective Switch. 25, 27, 33, 34, 35, 37, 38

Bibliography

[1] Optimization by simulated annealing: An experimental evaluation; part i, graph partitioning. *Oper. Res.*, 37(6):865–892, December 1989.

[2] Mohammad Al-fares, Alexander Loukissas, and Amin Vahdat. A scalable, commodity data center network architecture. In *In SIGCOMM '08: Proceedings of the ACM SIGCOMM 2008 conference on Data communication*, pages 63–74. ACM, 2008.

[3] Mohammad Al-Fares, Sivasankar Radhakrishnan, Barath Raghavan, Nelson Huang, and Amin Vahdat. Hedera: Dynamic flow scheduling for data center networks. In *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*, NSDI'10, page 19, USA, 2010. USENIX Association.

[4] C. J. Alpert and A. B. Kahng. Geometric embeddings for faster and better multi-way netlist partitioning. In *30th ACM/IEEE Design Automation Conference*, pages 743–748, 1993.

[5] Charles J Alpert and Andrew B Kahng. Recent directions in netlist partitioning: a survey. *Integration*, 19(1):1 – 81, 1995.

[6] Konstantin Andreev and Harald Racke. Balanced graph partitioning. *Theor. Comp. Sys.*, 39(6):929–939, November 2006.

[7] E. R. Barnes. An algorithm for partitioning the nodes of a graph. In *1981 20th IEEE Conference on Decision and Control including the Symposium on Adaptive Processes*, pages 303–304, 1981.

[8] G. Baxter, S. Frisken, D. Abakoumov, Hao Zhou, I. Clarke, A. Bartos, and S. Poole. Highly programmable wavelength selective switch based on liquid crystal on silicon switching elements. In *2006 Optical Fiber Communication Conference and the National Fiber Optic Engineers Conference*, pages 3 pp.–, 2006.

[9] Theophilus Benson, Aditya Akella, and David A. Maltz. Network traffic characteristics of data centers in the wild. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, IMC '10, page 267–280, New York, NY, USA, 2010. Association for Computing Machinery.

[10] Theophilus Benson, Ashok Anand, Aditya Akella, and Ming Zhang. Understanding data center traffic characteristics. *SIGCOMM Comput. Commun. Rev.*, 40(1):92–99, January 2010.

[11] Pierre-Alexandre Blanche, Lloyd LaComb, Youmin Wang, and Ming Wu. Diffraction-based optical switching with mems. *Applied Sciences*, 7(4):411, Apr 2017.

[12] J. P. Blanks. Near-optimal placement using a quadratic objective function. In *22nd ACM/IEEE Design Automation Conference*, pages 609–615, 1985.

[13] D. J. Blumenthal, P. R. Prucnal, and J. R. Sauer. Photonic packet switches: architectures and experimental implementations. *Proceedings of the IEEE*, 82(11):1650–1667, 1994.

[14] Thang Nguyen Bui and Byung Ro Moon. A fast and stable hybrid genetic algorithm for the ratio-cut partitioning problem on hypergraphs. In *Proceedings of the 31st Annual Design Automation Conference*, DAC '94, page 664–669, New York, NY, USA, 1994. Association for Computing Machinery.

[15] Aydin Buluç, Henning Meyerhenke, Ilya Safro, Peter Sanders, and Christian Schulz. Recent advances in graph partitioning. In *Algorithm Engineering*, 2016.

[16] P. K. Chan, M. D. F. Schlag, and J. Y. Zien. Spectral k-way ratio-cut partitioning and clustering. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 13(9):1088–1096, 1994.

[17] S. Y. Charles J. Alpert. Spectral partitioning: The more eigenvectors, the better. In *32nd Design Automation Conference*, pages 195–200, 1995.

[18] K. Chen, A. Singla, A. Singh, K. Ramachandran, L. Xu, Y. Zhang, X. Wen, and Y. Chen. Osa: An optical switching architecture for data center networks with unprecedented flexibility. *IEEE/ACM Transactions on Networking*, 22(2):498–511, 2014.

[19] K. Chen, X. Wen, X. Ma, Y. Chen, Y. Xia, C. Hu, and Q. Dong. Wavecube: A scalable, fault-tolerant, high-performance optical data center architecture. In *2015 IEEE Conference on Computer Communications (INFOCOM)*, pages 1903–1911, 2015.

[20] M. Chen, S. C. Liew, Z. Shao, and C. Kai. Markov approximation for combinatorial network optimization. In *2010 Proceedings IEEE INFOCOM*, pages 1–9, 2010.

[21] S. Cheung, T. Su, K. Okamoto, and S. J. B. Yoo. Ultra-compact silicon photonic 512 512 25 ghz arrayed waveguide grating router. *IEEE Journal of Selected Topics in Quantum Electronics*, 20(4):310–316, 2014.

[22] Andrea Chiuchiarelli, Rohan Gandhi, Sandro M. Rossi, Luís H.H. Carvalho, Francesco Caggioni, Júlio C.R.F. Oliveira, and Jacklyn D. Reis. Single wavelength 100g real-time transmission for high-speed data center communications. In *Optical Fiber Communication Conference*, page W4I.2. Optical Society of America, 2017.

[23] C. Clos. A study of non-blocking switching networks. *The Bell System Technical Journal*, 32(2):406–424, 1953.

[24] B. Dagens. Mz-soa interferometers for optical signal processing: from device to sytems. In *Optical Amplifiers and Their Applications*, page OTuD1. Optical Society of America, 2002.

[25] P. De Dobbelaere, K. Falta, S. Gloeckner, and S. Patra. Digital mems for optical switching. *IEEE Communications Magazine*, 40(3):88–95, 2002.

[26] J. Díaz, M. J. Serna, and J. Torán. Parallel approximation schemes for problems on planar graphs. *Acta Inf.*, 33(4):387–408, June 1996.

[27] Ran Duan and Seth Pettie. Linear-time approximation for maximum weight matching. *J. ACM*, 61(1), January 2014.

[28] Shantanu Dutt. New faster kernighan-lin-type graph-partitioning algorithms. In *Proceedings of the 1993 IEEE/ACM International Conference on Computer-Aided Design*, ICCAD '93, page 370–377, Washington, DC, USA, 1993. IEEE Computer Society Press.

[29] Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965.

[30] Enablence.com. *NxN AWG MULTIPLEXERS AND DEMULTIPLEXERS ROUTER MODULE (APRTE)- Datasheet*, 2010 (accessed June 3, 2020).

[31] Guy Even, Joseph (Seffi) Naor, Satish Rao, and Baruch Schieber. Fast approximate graph partitioning algorithms. In *Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '97, page 639–648, USA, 1997. Society for Industrial and Applied Mathematics.

[32] N. Farrington and A. Andreyev. Facebook's data center network architecture. In *2013 Optical Interconnects Conference*, pages 49–50, 2013.

[33] N. Farrington, E. Rubow, and A. Vahdat. Data center switch architecture in the age of merchant silicon. In *2009 17th IEEE Symposium on High Performance Interconnects*, pages 93–102, 2009.

[34] Nathan Farrington, George Porter, Sivasankar Radhakrishnan, Hamid Hajabdolali Bazzaz, Vikram Subramanya, Yeshaiahu Fainman, George Papen, and Amin Vahdat. Helios: A hybrid electrical/optical switch architecture for modular data centers. *SIGCOMM Comput. Commun. Rev.*, 40(4):339–350, August 2010.

[35] U. Feige and R. Krauthgamer. A polylogarithmic approximation of the minimum bisection. In *Proceedings 41st Annual Symposium on Foundations of Computer Science*, pages 105–115, 2000.

[36] Andreas Emil Feldmann. Fast balanced partitioning is hard even on grids and trees. In Branislav Rovan, Vladimiro Sassone, and Peter Widmayer, editors, *Mathematical Foundations of Computer Science 2012*, pages 372–382, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[37] Andreas Emil Feldmann and Luca Foschini. Balanced partitions of trees and applications. *Algorithmica*, 71(2):354–376, February 2015.

[38] Andreas Emil Feldmann and Peter Widmayer. An o(n4) time algorithm to compute the bisection width of solid grid graphs. In *Proceedings of the 19th European Conference on Algorithms*, ESA'11, page 143–154, Berlin, Heidelberg, 2011. Springer-Verlag.

[39] C. M. Fiduccia and R. M. Mattheyses. A linear-time heuristic for improving network partitions. In *Papers on Twenty-Five Years of Electronic Design Automation*, 25 years of DAC, page 241–247, New York, NY, USA, 1988. Association for Computing Machinery.

[40] J. E. Ford, V. A. Aksyuk, D. J. Bishop, and J. A. Walker. Wavelength add-drop switching using tilting micromirrors. *Journal of Lightwave Technology*, 17(5):904–911, 1999.

[41] L. R. Ford and D. R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956.

[42] Jonathan Frankle and Richard M. Karp. Circuit placements and cost bounds by eigenvector decomposition. In *Proceedings of the IEEE International Conference on Computer-Aided Design (ICCAD-86), A Conference for the EE CAD Professional*, pages 414–417, 1986.

[43] M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some simplified np-complete problems. In *Proceedings of the Sixth Annual ACM Symposium on Theory of Computing*, STOC '74, page 47–63, New York, NY, USA, 1974. Association for Computing Machinery.

[44] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman Co., USA, 1990.

[45] Albert Greenberg, James R. Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A. Maltz, Parveen Patel, and Sudipta Sengupta. Vl2: A scalable and flexible data center network. In *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication*, SIGCOMM '09, page 51–62, New York, NY, USA, 2009. Association for Computing Machinery.

[46] Kenneth M. Hall. An r-dimensional quadratic placement algorithm. *Management Science*, 17(3):219–229, 1970.

[47] L[aurent] Hyafil and R[onald] L. Rivest. Graph partitioning and constructing optimal decision trees are polynomial complete problems. Technical Report Rapport de Recherche no. 33, IRIA – Laboratoire de Recherche en Informatique et Automatique, October 1973.

[48] David R. Karger and Clifford Stein. A new approach to the minimum cut problem. *J. ACM*, 43(4):601–640, July 1996.

[49] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, 49(2):291–307, 1970.

[50] Subhash Khot. Ruling out ptas for graph min-bisection, dense k-subgraph, and bipartite clique. *SIAM Journal on Computing*, 36(4):1025–1071, December 2006.

[51] J. Kim, C. J. Nuzman, B. Kumar, D. F. Lieuwen, J. S. Kraus, A. Weiss, C. P. Lichtenwalner, A. R. Papazian, R. E. Frahm, N. R. Basavanhally, D. A. Ramsey, V. A. Aksyuk, F. Pardo, M. E. Simon, V. Lifton, H. B. Chan, M. Haueis, A. Gasparyan, H. R. Shea, S. Arney, C. A. Bolle, P. R. Kolodner, R. Ryf, D. T. Neilson, and J. V. Gates. 1100 x 1100 port mems-based optical crossconnect with 4-db maximum loss. *IEEE Photonics Technology Letters*, 15(11):1537–1539, 2003.

[52] Robert Krauthgamer. *Minimum Bisection*, pages 1294–1297. Springer New York, New York, NY, 2016.

[53] Robert Krauthgamer, Joseph (Seffi) Naor, and Roy Schwartz. Partitioning graphs into balanced components. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '09, page 942–949, USA, 2009. Society for Industrial and Applied Mathematics.

[54] C. F. Lam, H. Liu, B. Koley, X. Zhao, V. Kamalov, and V. Gill. Fiber optic communication technologies: What's needed for datacenter network operations. *IEEE Communications Magazine*, 48(7):32–39, 2010.

[55] Tom Leighton and Satish Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *J. ACM*, 46(6):787–832, November 1999.

[56] X.J.M. Leijtens, B. Kuhlow, and M.K. Smit. *Arrayed waveguide gratings*, pages 125–187. Springer, Germany, 2006.

[57] C. E. Leiserson. Fat-trees: Universal networks for hardware-efficient supercomputing. *IEEE Transactions on Computers*, C-34(10):892–901, 1985.

[58] Jason Li. Faster minimum k-cut of a simple graph, 2019.

[59] H. Liu, C. F. Lam, and C. Johnson. Scaling optical interconnects in datacenter networks opportunities and challenges for wdm. In *2010 18th IEEE Symposium on High Performance Interconnects*, pages 113–116, 2010.

[60] Ulrike Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4):395–416, December 2007.

[61] Robert M. MacGregor. *On Partitioning a Graph: A Theoretical and Empirical Study*. PhD thesis, EECS Department, University of California, Berkeley, Mar 1978.

[62] Michael Mahoney. Overview of graph partitioning. https://www.stat.berkeley.edu/~mmahoney/s15-stat260-cs294/Lectures/lecture05-05feb15.pdf. Accessed: June 2020.

[63] Eytan Modiano and Aradhana Narula-Tam. Mechanisms for procviding optical bypass in wdm-based networks. *Optical Networks*, 1:11–18, 2000.

[64] B. Mukherjee. Wdm-based local lightwave networks. i. single-hop systems. *IEEE Network*, 6(3):12–27, 1992.

[65] B. Mukherjee. Wdm-based local lightwave networks. ii. multihop systems. *IEEE Network*, 6(4):20–32, 1992.

[66] Adithyaram Narasimha, Behnam Analui, Erwin Balmater, Aaron Clark, Thomas Gal, Drew Guckenberger, Steve Gutierrez, Mark Harrison, Ryan Ingram, Roger Koumans, Daniel Kucharski, Kosal Leap, Yi Liang, Attila Mekis, Sina Mirsaidi, Mark Peterson, Tan Pham, Thierry Pinguet, David Rines, Vikram Sadagopan, Thomas J. Sleboda, Dan Song, Yanxin Wang, Brian Welch, Jeremy Witzens, Sherif Abdalla, Steffen Gloeckner, and Peter De Dobbelaere. A 40-gb/s qsfp optoelectronic transceiver in a 0.13μm cmos silicon-on-insulator technology. In *Optical Fiber Communication Conference/National Fiber Optic Engineers Conference*, page OMK7. Optical Society of America, 2008.

[67] A. Narula-Tam and E. Modiano. Dynamic load balancing in wdm packet networks with and without wavelength constraints. *IEEE Journal on Selected Areas in Communications*, 18(10):1972–1979, 2000.

[68] P. Newman, G. Minshall, and T. L. Lyon. Ip switching-atm under ip. *IEEE/ACM Transactions on Networking*, 6(2):117–129, 1998.

[69] Andrew Y. Ng, Michael I. Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. In *ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS*, pages 849–856. MIT Press, 2001.

[70] Radhika Niranjan Mysore, Andreas Pamboris, Nathan Farrington, Nelson Huang, Pardis Miri, Sivasankar Radhakrishnan, Vikram Subramanya, and Amin Vahdat. Portland: A scalable fault-tolerant layer 2 data center network fabric. *SIGCOMM Comput. Commun. Rev.*, 39(4):39–50, August 2009.

[71] E. A. Patent, J. J. G. M. van der Tol, Y. S. Oei, M. K. Smit, M. L. Nielsen, J. Mork, and J. J. M. Binsma. Integrated soa-mzi for pattern-effect-free amplification. *Electronics Letters*, 41(9):549–551, 2005.

[72] George Porter, Richard Strong, Nathan Farrington, Alex Forencich, Pang-Chen Sun, Tajana Rosing, Yeshaiahu Fainman, George Papen, and Amin Vahdat. Integrating microsecond circuit switching into the data center. In *In Proc. ACM SIGCOMM*, volume 43, pages 447–458, 08 2013.

[73] Harald Räcke. Optimal hierarchical decompositions for congestion minimization in networks. In *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing*, STOC '08, page 255–264, New York, NY, USA, 2008. Association for Computing Machinery.

[74] A. Rohit, A. Albores-Mejia, N. Calabretta, X. J. M. Leijtens, D. J. Robbins, M. K. Smit, and K. A. Williams. Fast remotely reconfigurable wavelength selective switch. In *2011 Optical Fiber Communication Conference and Exposition and the National Fiber Optic Engineers Conference*, pages 1–3, 2011.

[75] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C. Snoeren. Inside the social network's (datacenter) network. *SIGCOMM Comput. Commun. Rev.*, 45(4):123–137, August 2015.

[76] Y. G. Saab and V. B. Rao. Fast effective heuristics for the graph bisectioning problem. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 9(1):91–98, 1990.

[77] H. Saran and V. V. Vazirani. Finding k-cuts within twice the optimal. In *[1991] Proceedings 32nd Annual Symposium of Foundations of Computer Science*, pages 743–751, 1991.

[78] K. Sato, H. Hasegawa, T. Niwa, and T. Watanabe. A large-scale wavelength routing optical switch for data center networks. *IEEE Communications Magazine*, 51(9):46–52, 2013.

[79] Horst D. Simon and Shang-Hua Teng. How good is recursive bisection? *SIAM J. Sci. Comput.*, 18(5):1436–1445, September 1997.

[80] Arjun Singh, Joon Ong, Amit Agarwal, Glen Anderson, Ashby Armistead, Roy Bannon, Seb Boving, Gaurav Desai, Bob Felderman, Paulie Germano, Anand Kanagala, Jeff Provost, Jason Simmons, Eiichi Tanda, Jim Wanderer, Urs Hölzle, Stephen Stuart, and Amin Vahdat. Jupiter rising: A decade of clos topologies and centralized control in google's datacenter network. *SIGCOMM Comput. Commun. Rev.*, 45(4):183–197, August 2015.

[81] Arjun Singh, Joon Ong, Amit Agarwal, Glen Anderson, Ashby Armistead, Roy Bannon, Seb Boving, Gaurav Desai, Bob Felderman, Paulie Germano, Anand Kanagala, Jeff Provost, Jason Simmons, Eiichi Tanda, Jim Wanderer, Urs Hölzle,

Stephen Stuart, and Amin Vahdat. Jupiter rising: A decade of clos topologies and centralized control in google's datacenter network. In *Sigcomm '15*, 2015.

[82] A. Vahdat, H. Liu, Xiaoxue Zhao, and C. Johnson. The emerging optical data center. In *2011 Optical Fiber Communication Conference and Exposition and the National Fiber Optic Engineers Conference*, pages 1–3, 2011.

[83] Guohui Wang, David G. Andersen, Michael Kaminsky, Konstantina Papagiannaki, T.S. Eugene Ng, Michael Kozuch, and Michael Ryan. C-through: Part-time optics in data centers. *SIGCOMM Comput. Commun. Rev.*, 40(4):327–338, August 2010.

[84] Kang Xi, Yu-Hsiang Kao, and H. Jonathan Chao. *A Petabit Bufferless Optical Switch for Data Center Networks*, pages 135–154. Springer New York, New York, NY, 2013.

[85] Xiaohui Ye, P. Mejia, Yawei Yin, R. Proietti, S. J. B. Yoo, and V. Akella. Dos - a scalable optical switch for datacenters. In *2010 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, pages 1–12, 2010.

[86] M. Xu, C. Liu, and S. Subramaniam. Podca: A passive optical data center architecture. In *2016 IEEE International Conference on Communications (ICC)*, pages 1–6, 2016.

[87] Maotong Xu, Jelena Diakonikolas, Eytan Modiano, and Suresh Subramaniam. A hierarchical wdm-based scalable data center network architecture. *CoRR*, abs/1901.06450, 2019.

[88] F. Yan, W. Miao, O. Raz, and N. Calabretta. Opsquare: A flat dcn architecture based on flow-controlled optical packet switches. *IEEE/OSA Journal of Optical Communications and Networking*, 9(4):291–303, 2017.

[89] Yang Chen, Chunming Qiao, and Xiang Yu. Optical burst switching: a new area in optical networking research. *IEEE Network*, 18(3):16–23, 2004.

[90] S. J. B. Yoo, Fei Xue, Y. Bansal, J. Taylor, Zhong Pan, Jing Cao, Minyong Jeon, T. Nady, G. Goncher, K. Boyer, K. Okamoto, S. Kamei, and V. Akella. High-performance optical-label switching packet routers and smart edge routers for the next-generation internet. *IEEE Journal on Selected Areas in Communications*, 21(7):1041–1051, 2003.

[91] Zelda Zabinsky. *Stochastic Adaptive Search for Global Optimization*, volume 72. 01 2003.

[92] Zelda Zabinsky, Graham Wood, Mike Steel, and William Baritompa. Pure adaptive search for finite global optimization. *Math. Program.*, 69:443–448, 07 1995.