UNIVERSITY OF CALIFORNIA, SAN DIEGO

Miswirings Diagnosis, Detection and Recovery in Data Centers

A thesis submitted in partial satisfaction of the
requirements for the degree
Master of Science

in

Computer Science

by

Seyedeh Pardis Miri

Committee in charge:

Professor Amin Vahat, Chair
Professor Keith Marzullo
Professor Joe Pasqualle

2010

The thesis of Seyedeh Pardis Miri is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

Chair

University of California, San Diego

2010

DEDICATION

First and foremost, I would like to dedicate this thesis to my parents, for their endless love, support and encouragement.

To two of my professors at UCSD department of Computer Science and Engineering I am indebted for my intellectual formation: Professor Amin Vahdat, my adviser; Professor Keith Murzullo, my co-advisor; Professor Vahdat and Professor Murzullo both believed in me and my work when I most needed this belief. Their excitement about my work, their tolerance of my wayward writing and submission habits, and above all, their willingness to contribute insightful comments to my long-gestated work, have all been instrumental in making this thesis a reality.

Words fail to express my indebtedness to both of them. If my professors have contributed to my intellectual formation, they, as my best friends, have helped me to develop as a human being. Indeed, their support in times of trouble, but also their unwavering belief in my potential as a researcher, has always been a motive for me to live up to their expectations.

EPIGRAPH

Life is pretty simple: You do some stuff.
Some works. You do more of what works.
If it works big, others quickly copy it.
Then you do something else.
The trick is the doing something else.
—Leonardo da Vinci

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGEMENTS

I would like to thank my colleagues at Data Center Networking group for sharing their knowledge and experience with me and for their help in my research. I would like to thank Radhika Niranjan Mysore, Andreas Pamboris, Nathan Farrington, Nelson Huang, Sivasankar Radhakrishnan, and Vikram Subramanya, individuals who have contributed to publication of "PortLand: A Scalable Fault-Tolerant Layer 2 Data Center Network Fabric". I am glad to be a part of a great and inspiring team that provides an important work in data center networking.

I would like to thank Meg Walraed-Sullivan, a friend and a mentor, who has been a driving force behind this research. It has been a great pleasure working with such a genius computer scientist.

I would also thank Alex Rasmussen, Reza Hemmati, Amirali Shayan, and Brian Kantor for their help and support and for being there for me whenever I needed them.

Last but not least, I would like to dedicate this thesis to all my friends, past and present. Indeed, it was my need to live up to their expectations of me that encouraged to put as much of myself into this work as humanly possible.

VITA

2007	B. S. in Computer Engineering, Amirkabir University of Technology, Tehran
2009	Graduate Teaching Assistant, University of California, San Diego
2010	M. S. in Computer Science, University of California, San Diego

PUBLICATIONS

Radhika Niranjan Mysore, Andreas Pamboris, Nathan Farrington, Nelson Huang, Pardis Miri, Sivasankar Radhakrishnan, Vikram Subramanya, and Amin Vahdat, “PortLand: A Scalable Fault-Tolerant Layer 2 Data Center Network Fabric," in *Proceedings of the ACM SIGCOMM Conference,* August 2009

Behnam Ghavami, SeyedehPardis Miri, Hossein Pedram. “Exploring an AES Cryptoprocessor Design Using a Secure Asynchronous Toolset”, IEEE EWDTS, Yerevan, September 7-10, 2007.

ABSTRACT OF THE THESIS

Miswirings Diagnosis, Detection and Recovery in Data Centers

by

Seyedeh Pardis Miri

Master of Science in Computer Science

University of California, San Diego, 2010

Professor Amin Vahat, Chair

As data volume increases, scalability of hardware can rapidly become a problem. Hence, it is easy to make a mistake while wiring up switches, particularly in large scale systems like a data center. As a data center scales up, bad wirings or miswirings generally results in diminishing performance in terms of wasting bandwidth, losing reachability, having long forwarding tables and latency. Hence, it is both important and beneficial to identify whether limited number of changes such as add/remove wires and/or switches could be made to improve data center performance.

Data center fabric, built in PortLand [4],is subject to many kinds of fault - some introduced by faulty components, some introduced by operator errors, and some introduced by misconfiguration. UNRP [9] was developed to allow the fabric to quickly establish itself and enable forwarding, but it has almost no diagnostic abilities. UNRP operates with no knowledge of the desired configuration nor does it evaluate the quality of the configuration it found. This is because UNRP is in the operational critical path, and has to be fast and human intervention-free.

In this thesis, we look at the complementary problem: evaluating the actual configuration of a data center fabric. Solving this problem allows the fabric engineers to

understand the behavior of the fabric and make informed decisions on how it should be reconfigured or repaired, trading off issues such as cost, disruption, and physical limitations. This thesis focuses on an important core problem in this space: given a known intended topology and the output of UNRP, (1) report on the utility of the fabric as compared to the utility of the intended fabric; (2) come up with a complete and accurate list of the faults in the fabric; (2) give a repair plan that assigns priorities to which faults should be repaired first in order to increase utility.

In order to evaluate UNDP, we show that injecting faults into a fat-tree topology will be detected and recovered, as long as failures occur within a certain boundary. However, when combination of faults is injected, it is not easy to neither detect them all nor cover them. In addition, we present an analysis of why some of these cases are undetectable. We also demonstrate a comparison between the currently wired topology and the optimal topology when the injected fault is the switch crashes. The comparison between the optimal topology and the current topology indicates how efficiently the currently wired topology could have used the available hardware.

Chapter 1

Introduction

1.1 Motivation

A data center is a centralized repository for data storage, management, and dissemination, comprising server and networking infrastructure, which generally include redundant or backup power supplies, air conditioning system, fire suppression, security devices, etc.

Since the anticipation of the growth and technological changes can be difficult, data centers should be able to respond to the accelerating growth in demand and changes in equipment, devices, and standards while remaining manageable and reliable.

As data demand increases, the need for developing a data center design that optimizes availability, scalability, and flexibility to enhance *performance*, while minimizing costs is significantly realized by organizations and companies. In order to scale up data centers, companies and organizations need to focus on several factors, mostly related to physical condition of the building (i.e. cooling system, power supply and backup, area) constraining the data center and the placement of the rack of servers inside of it.

Hence, monitoring power consumption, configuring adaptable electrical system, developing an energy efficient cooling system, anticipating the need to build or expand the data center are challenges that companies are faced with in order to maintain reasonable performance, as they scale up their data centers.

1.2 Problem Statement

The rapid increase in data volume requires system scalability provision. By system scalability, we mean deploying efficient and effective management strategies to have data availability and continual quality assurance of searching and indexing. From this perspective, the necessity for a high quality adequate, 'hardware", forming a data center, is becoming increasingly essential for successful and well-managed long-term data availability as well as storage quality assurance. An adequate number of end-hosts and a fairly large number of cables and servers are considered as hardware essentials for a data center.

As data volume increases, scalability of hardware can rapidly become a problem. Each end-host requires data acquisition, interchange and processing, at a desirable rate, which inevitably insists upon adequate bandwidth availability, enough RAM and disk storage. End-hosts hardware limitation, at some point, enforces scalability in terms of the number of end-host enhancement. As the number of end-hosts increases, network bandwidth availability has to be intensified, to avoid network bottleneck, particularly when running bandwidth-intensive applications. Since the amount of available network bandwidth is not foreseeable, the risk of network congestion and data loss becomes a significant management challenge. Accordingly, ensuring the availability of the minimum required bandwidth in the network based on the number of end-hosts is essential. Given that the minimum required bandwidth is application-specific and may change dynamically, oversubscription ratio of the network is what has to be determined.

Oversubscription is the fundamental measure of network's performance. Depending on the traffic characteristics of the applications running on network, the oversubscription ratio is defined. Too much oversubscription results in congestion and packet loss in network. However, very low over subscription ratio is not economically beneficial since more hardware in terms of servers and wire is required to provide more bandwidth. Hence, oversubscription is an optimized parameter for hardware cost and bandwidth requirement.

Recent publication [4] has proposed a network architecture that provides oversubscription ratio of 1 for bandwidth availability and is cost-efficient as compared with those supporting the same number of end-hosts. Fat-tree architecture (see. Figure 1) is a

cost-beneficial scalable architecture built with identical commodity switches and it provides full bisection bandwidth to the end-hosts. Aside from these three main properties, fat-tree architecture offers multiple paths with the same metric to any destination. This property allows deploying load balancing in case of network congestion in some of the paths, and fault tolerance by rerouting in case of losing some of the paths. Two other recent publications [4] and [7] have implemented load balancing, fault tolerance and flow scheduling in a fat-tree data center. In conclusion, fat-tree architecture is recognized as a suitable design for data center networking.

Even though a data center needs to be designed with a life expectancy of 10 to 15 years, most IT equipment must be refreshed after 2 to 5 years. Hence, the design needs to be able to adapt to the equipment relocation within the data center and to handle high frequency of changes anywhere between two and four complete IT equipment refreshes.[3]

Adaptation to such a high rate of changes to maintain the desirable performance requires accurate and precise design anticipation. Unfortunately, such a precise anticipation is not viable due to high trend of technology growth. Therefore, to maintain and maximize the desired performance of a data center, a cheaper alternative approach with less prediction is a valuable and meaningful attempt.

As a data center scales up, bad wiring or miswirings generally results in diminishing performance in terms of wasting bandwidth, losing reachability, having long forwarding tables and latency.

For this reason, it is both important and beneficial to identify whether limited number of changes such as add/remove wires and/or switches could be made to improve data center performance. Providing a protocol that diagnoses whether performance is diminished due to bad wiring or miswirings and that presents suggestions for performance enhancement is the main focus of this thesis.

1.3 Thesis Contributions

One of the primary contributions of this thesis is the proposal for transforming any arbitrary data center network topology into a three level multi-rooted tree with a

specified oversubscription ratio. The oversubscription ratio will be specified by the data center administrator. It is considered to be 1 in the default case.

Ideally fat-tree, shown in Figure 1.1, is a rational, cost-efficient and scalable model for a data center to be transformed into. Fat-tree, a specific three level multi-rooted tree with oversubscription ratio of 1, is a non-blocking data center network topology that offers a full bisection bandwidth. [4] However, we have left a degree of freedom for data center administration to specify the desirable over-subscription ratio.

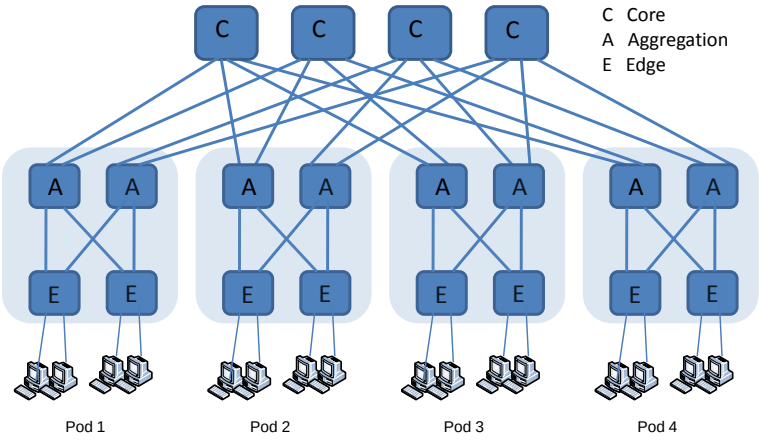

Figure 1.1: Sample fat tree topology

Definition of objective and quantitative measures for adding/removeing links and/or switches to determine what rewirings would move a current data center topology closer to a fat-tree-like topology is another part of this thesis contribution for gaining higher performance in a given data center topology. Based on optimized output results of the measurements, the diagnostic protocol generates a prioritized chain of change suggestions with the goal of improving reachability, bisection bandwidth and cross section bandwidth.

The main contributions of the thesis can be summarized as:

- Definition of objective and quantitative measures for transforming a current data center topology closer to a fat-tree like topology, considering the oversubscription ratio and connectivity parameters.
- Identification of optimized possible solutions, based on administrative feedback, to be presented as the final prioritized chain of performance improvement suggestions.

1.4 Organization of the Thesis

The rest of this thesis is organized as follows. A background review of current data center network architectures is presented in Chapter 2. a discerption of centralized and a distributed architectures for Location Discovery Protocol (LDP) is provided in Chapter 3. Potential miswirings and new LDP protocol for transforming a current topology into a three level multi-rooted tree with specific over-subscription ratio are introduced in Chapter 4 and an analysis of the new LDP protocol is presented in this chapter as well. The design of diagnostic protocol and its optimization method to diagnose the wiring problems and to present the optimization chain of results are proposed in Chapter 5. A detail of experimental results, evaluation and the environmental set up are elaborated in chapter 6. The thesis is concluded in Chapter 7 where some possible directions for future research are also suggested.

1.5 Related Publications

- Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat, “A Scal- able, Commodity Data Center Network Architecture," in *Proceedings of the ACM SIGCOMM Conference,* August 2008
- Radhika Niranjan Mysore, Andreas Pamboris, Nathan Farrington, Nelson Huang, Pardis Miri, Sivasankar Radhakrishnan, Vikram Subramanya, and Amin Vahdat,

“PortLand: A Scalable Fault-Tolerant Layer 2 Data Center Network Fabric," in *Proceedings of the ACM SIGCOMM Conference,* August 2009

Chapter 2

Literature Review

2.1 Data center network Architectures

2.1.1 Topology

Although we focus on the mulit-rooted tree architecture, particularly fat-tree interconnection networks, there are several other notable topologies deployed for data center networking in the next chapters of this thesis.

BCube

Container-based modular data center is an efficient approach to deliver computing and storage service in a large scale. BCube, an inter-container service-centric network architecture, interconnects 1000 to 2000 services in a container and provides a high bandwidth support for specific traffic patterns such as one-to-one, one-to-several, and one-to-all. The containers have low cooling and manufacturing cost while a high degree of mobility and deployment. The containers provide a reasonable performance as the number of servers or switch failures inside the container increase.

BCube is a special case of generalized Hyper-Cube, however, it uses commodity mini-switches to reduce cabling complexity for server interconnection which is not employed in generalized Hyper-Cube.

BCube also provides a high bandwidth inside the containers. Sometimes the aggregation bandwidth among the containers can reach tera-bit per second; however, no

provision on bandwidth providence between containers is made.

Finally, BCube is a fault tolerant design and significantly accelerates representative bandwidth- intensive applications by performing load balancing methods.[5]

MDCubes

MDCube, is a high performance interconnection structure that scales BCube-based containers to mega-data centers by deploying a high-speed uplink such as fiber optic to interconnect the containers. MDCube provides a high network capacity and handles fault-tolerance and load balancing by putting routing intelligence into servers. like Portland, MDCube uses commodity servers instead of high-end ones to scale.

Even though MDCube offers a high aggregate bandwidth between containers, achieved by the MDCube topology design and a hierarchical routing algorithm used on top of it, when the number of containers scales to hundreds, the containers require long cables between them which practically become a barrier for scalability. Hence, the scalability barriers for MDCubes are expressed in terms of both cabling complexity and cost. Portland also suffers from the same problem.[10]

Dcell

Dcell is another instance of server-centric topology. DCell uses recursion to scale, in which a high-level DCell structure is constructed from many low-level DCells and at the same level DCells are fully connected with one another. The recursion structure of DCell results in double exponential scale up as the number of servers increases. Dcell provides high aggregate bandwidth capacity using commodity mini-switches to scale out; however, it introduces high wiring cost and complexity just like in Portland.[6]

Portland: Fat-tree

Recent publications [7] and [**?**] on data center networking have proposed a topology based on multi-rooted trees, so called fat-trees.

Portland leverages identical small commodity k-port switches, fully compatible with Ethernet and IP, into a Fat-tree architecture with three layers, labeled as edge, aggregation, and core (shown in Figure 1.1). The advantage of using commodity switches

is that it easily fits into the current Internet architecture (Ethernet, IP, TCP) and it reduces power and cost significantly as data center scales.

Fat-trees have two attractive properties. One of which is having multi possible/available shortest paths between end-hosts, an the other one is having non-blocking property when certain conditions are met in Fat-tree topology. However, non-blocking operation requires load balancing by careful scheduling of packets among all possible/available paths.

In order to perform routing/forwarding, load balancing and implement fault tolerance in fat-tree data center, Portland leaves the end hosts unmodified but uses openflow [2] that allows switch modification. Portland allows end-hosts flat Mac address to be rewritten by a hierarchial pseudo Mac address in switches which results in network scalability by avoiding broadcast in layer 2.

Portland also envisions a large scale plug-and-play network fabric for data centers which resulted in introducing Location Discovery Protocol (LDP) in this paper. LDP is discussed in detail in chapter 3. As a conclusion, Portland is in fact a DHCP in layer 2 that can be applied to any multi-rooted, multi-level tree.

2.1.2 Structure and definitions

Portland: Fat-tree

A three layer fat-tree built by using identical K-port switches has $K^2/2$ switches in each of edge and aggregation layers and $k^2/4$ switches in the core layer. Every $k/2$ switches in the edge and the aggregation layers form a "putative pod". A fat-tree has K putative pods. It has $K/2$ distinct 3 hop count paths from any hosts to any other host within a putative pod and $K/2$ distinct 5 hop count paths from any hosts to any other host, each within a different putative pod. In addition, there is exactly one 3 hop count path from any given core switch to any destination putative pod.

- Pod Definition

  Fat-tree segmentation can be done in several ways. We introduce three ways of doing it (see Figure 2.1)

1) LDP pods: these are the pods that the LDP protocol computes. Each aggregation switch is in exactly one LDP pod and edge switches can be in more than one LDP pod. Aggregation switches connected to the same set of edge switches are in the same LDP number and have the same LDP pod number.

2) Putative pods: a putative pod is a set of aggregation and edge switches such that there is a path between any two switches in the putative pod that does not traverse a core switch. A putative pod may contains one or more LDP pods in its entirety, and each edge and aggregation switch is in exactly one putative pod. In other words, LDP pods do not overlap.

3) Intensional pods: these are the pods that the network administrators had in mind when they wired up the network. Each aggregation and edge switch is in exactly one intensional pod. Since we do not know the intensional pods, the diagnostic protocol tries to identify/compute the best guess of the intensional pods.

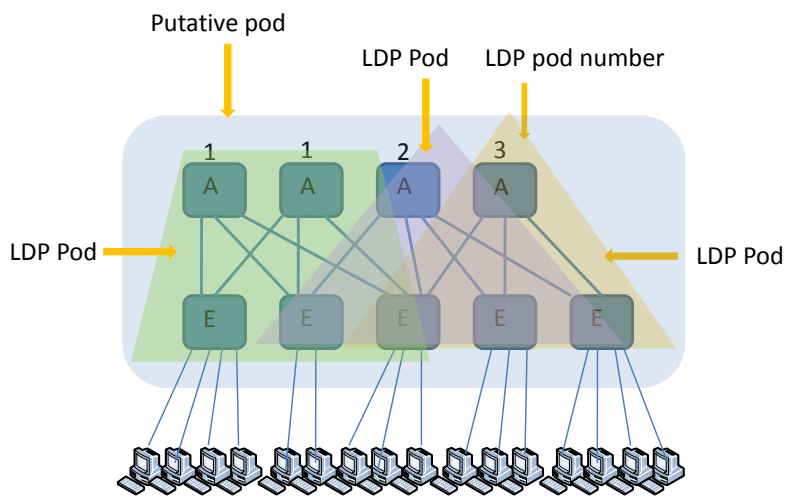

Figure 2.1: Sample fat tree topology with 3 LDP pods and a putative pod

- Oversubscription

  Oversubscription is the fundamental measure of the network performance. Oversubscription ratio is the access bandwidth divided by the amount of guaranteed bandwidth that a user has to the core of the design. An oversubscription of 1:1 indicates that all end hosts are able to communicate with each other at full bandwidth of their network interface (e.g. 1 Gb/s for commodity Ethernet servers).

  Oversubscription deal specifically with points in a network where bottlenecks occur. The impact of improper oversubscription ratios is congestion, which can result in packet loss. However, over-subscription is quite common in current data center designs since the cost for having a network without it could be far too great. Fat-tree however, is a network topology that provides 1:1 oversubscription ratio and scales without introducing exponential hardware cost.

  In a multi-rooted network topology, the oversubscription ratio (fan out) is the minimum of oversubscription ratios among putative pods. Oversubscription of each putative pod is the number of edge switches to the number of aggregation switches in that pod. One of the goals of diagnostic protocol is to compute each putative pod fan out and compare it against its intensional pod fan out. Knowing such result is helpful to estimate hardware requirement to ensure desirable oversubscription ratio. The default assumption for intensional pod fan out is 1:1 in case the information was not provided by network administrator.

- Reachability

  Fat-trees provide path redundancy which enhances reachability in two levels:inter-putative pod and intra-putative pod. Inter-putative reachability allows two end-hosts within the same putative pod to direct packets between each other by using all reachable paths. The maximum inter-putative reachability degree of a fat-tree is $K/2$. The intra-putative pod reachability allows any given core switch to reach all destination putative pod with the maximum intra-putative reachability degree of K. It should be noted that providing intra-putative pod reachability has a higher priority compared with inter-putative pod reachability. This is mostly due to the fact that each core has only a single path access to each putative pod. If a link

between a core and aggregation switch fails, assuming no miswiring is present in the network, the putative pod which the aggregation switch is belonging to becomes unreachable. Diagnostic protocol priority is to provide intra-putative pod reachability first, unless certain reachability policies are defined by network administration.

Chapter 3

Location Discovery Protocol: Related Work

Future data centers have millions of virtual end points so it is expected that those data center network architects would have plug-and-play deployment for switches. Portland introduced Location Discovery Protocol (LDP) that makes manual configuration of switches no longer necessary. Hence, switches discover their place in the network themselves which eliminates manual configuration on them.

In this chapter, two LDP protocols are presented: 1. LDP proposed in in Portland [7] which is a centralized fat-tree specific protocol; 2. LDP proposed in [8] which is a distributed protocol that addresses the general case of multi-rooted trees.

3.1 Centralized LDP

Portland LDP is a centralized protocol, i.e. switches interact with a centralized fabric manager to obtain their configuration information. Portland switches do not begin packet forwarding until their location is established. This is mainly because edge switches need to create appropriate Pseudo Mac addresses (PMACs) for hosts connected to their ports for routing/forwarding purpose.

PMAC is a 48-bit address with 4 tuples of pod, position, port and virtual machine id (vmid). The definition of pod in Portland paper is exactly the same as that for putative pod, explained in chapter 2. All edge and aggregation switches that are connected to

each other and that can be traversed not through a core switch, form a putative pod. Hence, putative pod number is shared among all edge and aggregation switches within a putative pod. The uniqueness of putative pod numbers has to be assured in the data center network. This is mostly because when a core switch forwards a packet, it will be able to correctly forward the packet to the destined pod.

Once a packet reaches an aggregation switch within the destined pod, it has to be forwarded to the right destined edge switch. This can be done based on the position value listed in 4-tuple destination PMAC address. The edge switches within a pod are assigned with unique position numbers so that aggregation switches will be able to correctly forward the packet to the destined edge switch.

Once a packet reaches an edge switch, it must be forwarded to the right port based on the port value, listed in PMAC address. Hence, the packet will be forwarded to the appropriate destination based on the virtual machine id numbers that can be reached through the destined port.

In conclusion, switches must adopt appropriate pod and position numbers so that forwarding correctness is assured. To ensure correctness of the pod and position numbers, switches must first learn which layer (edge, aggregation, core) they belong to. The first phase of LDP protocol in which switches learn their levels is called level assignment.

3.1.1 Level Assignment

During level assignment phase, switches constantly send keep alive messages, so-called LDM, to their neighbor switches. Since this LDP version is fat-tree specific, it can be concluded that if a switch is not connected to more than $k/2$ other switches, it is definitely an edge switch. A switch can further confirm its level by sending pings on all its ports knowing the fact that hosts will reply to such pings but will not transmit LDMs. However, switches will reply to the pings and will transmit LDMs as well. Getting back the replies from both hosts and switches, an edge switch can determine its upward and downward facing ports. Those ports that are connected to the host are considered as downward facing ports where as those connected to switches are known as upward facing ports.

Once a switch identifies its level, it updates the LDM messages it sends to its neighbor switches to reflect its level. A switch receiving an LDM message from an edge switch on one of its ports concludes that it must be an aggregation switch. It also concludes that the corresponding incoming port is a downward facing port. Once a switch receives a LDM message from aggregation switches, it identifies its level to be a core. After core switch updats its LDM message and informs neighbors, aggregation switches will set their upward ports.

Right after a switch learns its level and determines that its neighbors have learned their levels as well, pod and position number assignment phase begins for the corresponding switch.

3.1.2 Pod and Position Assignment

As soon as an edge switch learns its level and realizes that all LDM messages that it receives are from aggregation switches, it proposes a randomly chosen number range from 0 to $K/2-1$ to all aggregation switches from which it received LDM. If the proposal is verified by a majority of these aggregation switches, the edge switch adopts the position number and updates its LDM messages accordingly.

Since it is assumed that the underlying network is a fat-tree and no failure condition or miswiring is present, it can be determined that there exists an edge switch that adopts position 0. This edge switch requests a pod number from the fabric manager and then the fabric manager assigns a unique pod number (putative pod number) to all switches within the putative pod. The uniqueness of putative pod number can be assured since the fabric manager has the global knowledge of all putative pods within the network.

3.2 Distributed LDP

In LDP centralized, strong assumptions such as no miswirings or switch/host/fabirc manager failure condition are made to provide liveness and safety properties for the protocol. However, in reality failure avoidance cannot be guarantied. If for any reason fabric manager fails, the liveness property of LDP will no longer be held. Hence, recent

work [????andreas's thesis] has presented a distributed LDP protocol in which switches discover their place based on their position in the multi-rooted tree with no involvement of fabric manager. However, still some strong assumptions are made regarding disallowance of some failure condition.

3.2.1 LDP for general multi-rooted trees

In the distributed protocol presented is this chapter, level assignment phase is exactly the same as that presented in centralized LDP. The difference is mainly in pod and position assignment.

3.2.2 Pod and Position Assignment

In this version, pod and position assignments are done in a distributed manner. The notion of pod formation is that at most K edge and aggregation switches group themselves into a putative pods. As a result, an aggregation switch, so-called leader, must be chosen randomly to invite sufficient numbers of edge and aggregation switches to form a putative pod. The leader is an aggregation switch that selects a random pod number within the possible range of 0 to K and then sends a QUERY message to all edge neighbors. The QUERY message asks neighbor switches whether or not they have joined a pod yet. After waiting for a around trip time, the leader collects the REPLY messages from its edge switch neighbors. A neighbor edge switch which has not joined a pod already, replies back with positive interest and waits until it receives the JOIN message from the leader. The JOIN message also includes the position number that the receiver edge switch must adopt. Once an edge switch is joined, it learns the putative pod number selected by the leader and also its position within the putative pod. This eventually leads to the creation of a putative pod with at most K switches.

There are failure conditions that can violate liveness property of this protocol;e.g., the failure of the leader after receiving the REPLY message. If such a case happens, all edge switches that are not part of a pod yet will sit in the waiting state for ever. There is a full chapter in [8] that elaborates such corner case conditions.

In chapter 4, a more fault-tolerant and adaptive version of LDP protocol that

reacts quickly to failures without human interaction or violation of safety property will be introduced. This new version of LDP is not only fat-tree specific but also works for the general cases of multi-root trees.

Chapter 4

Miswirings and New Location Discovery Protocol Design

It is easy to make a mistake while wiring up switches, especially in large scale systems like a data center. Even if the intentional topology is known, switch failure, asynchronous host arrival, and errors in switch connection make the determination of desired topology difficult. Accordingly, it is beneficial to be able to automatically detect and fix the miswirings. The detection can be done based on the difference between the current topology and the intentional topology, assuming that the intensional topology is known.

Here we assume that the intentional topology is a 3-tiered multi-rooted tree that underlies many data center network fabrics. Switch failure can happen and maintain for unbounded time and arbitrary miswiring cables are able to interconnect switches in any way, regardless of the intended network topology.

4.1 Miswirings

In this section we categorize all possible miswiring cases of the fat-tree topology into four major groups. This is mainly because all types of faults that can be injected into a fat-tree lead to these four independant miswiring cases. In addition, each of these four types of miswirings can be identifed differently. For instance, group 1 miswirings are identified by UNRP. Group 2 miswirings are cross link pod cases which are identied

by min-cut algorithm. Group 3 are host misplacement that are identified by UNRP and mincut algorithm. Group 4 miswirings are multiple links between a core and aggregation switches within a single putative pod which can be identifed after all previous three types of miswirings are detected and fixed and by comparing core switches connectivity lists. When these four types of miswirings are detected and necessary links causing them are deactivated, then the network topology can be improved by links being added between appropriate switches that results in network performance improvement. It worth noting that all possible faults that can be injected to a fat-tree will be fully elaborated in chapter 5.

4.1.1 Group 1: UNRP miswiring

This group includes cases where there exists a cable between two edge switches, an edge and a core switch, two aggregation switches, two core switches. In addition, there are also cases in which a loop back cable connects a switch to itself and multiple cables connect any two switches. These cases of miscablings in a three level multi-rooted tree can be detected through running the new version of LDP. If changes to the switch's level assignment creates a link between two switches with the same tree level, or between an edge switch and a core switch, that link is marked as inactive with respect to the address assignment phase. However, keep alive messages are still received and processed over such links, so that a change to either neighbor's tree level is noticed, but switches ignore neighbors connected by such links in calculation of address assignments.

4.1.2 Group 2: Super pod formation miswiring

Super pod formation is a case where an edge switch in one putative pod is connected to an aggregation switch in another putative pod. This case of miswiring cannot be identified by running LDP. The diagnostic protocol, addressed in chapter 5, identifies such miswiring by processing the information received from all edge and aggregation switches. It is difficult to distinguish between such miswiring and a case where switches with smaller size (fewer ports) are used in either edge or aggregation level. They are

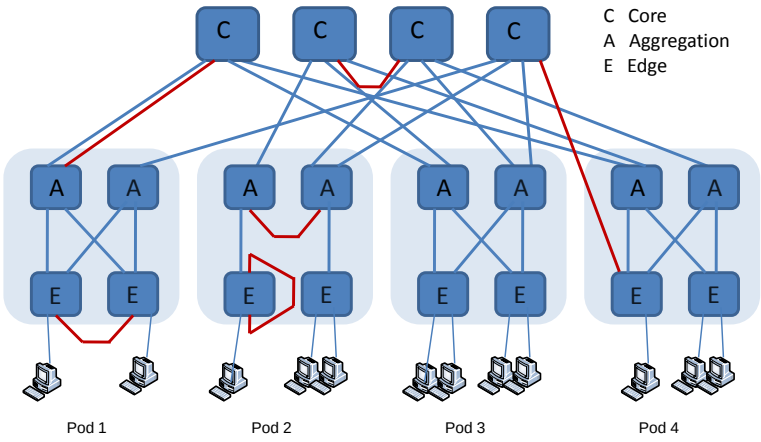

Figure 4.1: Group1: All 6 cases of group 1 miswirings

smaller compared to the rest of edge and aggregation switches used in a putative pod. For simplicity, we assume that switches have similar size so there is no need to deal with such a case.

4.1.3 Group 3: End-host misplacement

End-host misplacement is the connection of one or more end-hosts to an aggregation or a core switch. Such miswiring cannot easily be detected since the new version of LDP assumes that a switch connected to an end-host is an edge switch. This is a base case for running the level assignment phase of new LDP. Hence, the new LDP level assignment phase cannot determine whether or not an edge switch is potentially suitable to be an aggregation or a core switch. As shown in Figures 4.3and 4.4, it is significantly important to identify these two cases and treat them differently. This is mainly because the level assignment of the new LDP has several drawbacks such as losing available

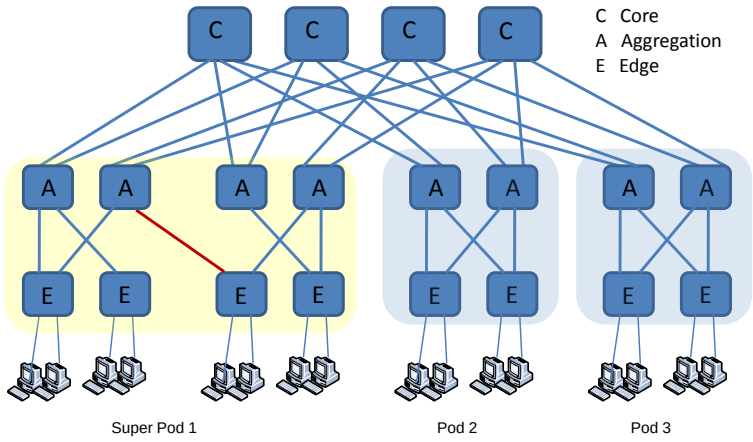

Figure 4.2: Group 2: Super pod formation miswiring

bandwidth and reachability in case 1 and having longer routing tables as well as losing reachability in case 2.

In case 1, as soon as switch x identifies an end-host, it stopes forwarding along the red links connecting x to three edge switches and one core switch. By doing this, switch x will become isolated. Hence, the end host connected to x will no longer be reachable. In addition, available bandwidth for end-hosts connected to three edge switches get reduced since packets, except keep-alive ones, are not allowed to be forwarded along these three red links.

In case 2, as soon as switch x identifies an end-host, it sets its level to be an edge. If that happens, the miswiring shown in the Figures 4.4 can be categorized as a case of super-pod formation miswiring in which an edge switch from one putative pod is connected to an aggregation switch in another putative pod. If the diagnostic protocol categorizes such miswiring as a group 2 type, all links connecting x to other putative pods must become deactivated. This results in reduction of reachability between pods.

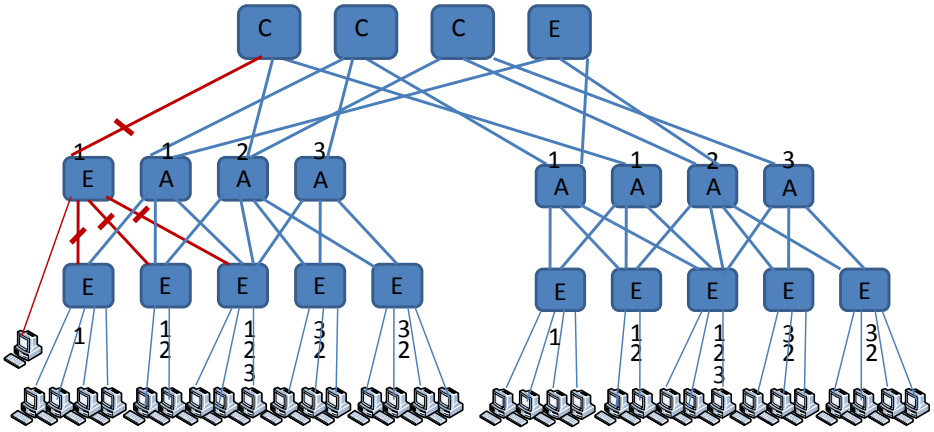

Figure 4.3: Case 1: a host connected to an aggregation switch

Moreover, running min-cut algorithm to diagnose group 2 miswirings causes this switch to become isolated. In order to avoid this, such miswiring should be resulted in end-host removal instead of switch isolation. The other downside of such miswiring is that it increases the number of addresses associated with edge switches which leads to longer routing tables and a change of switch forwarding.

Diagnostic protocol approach to solve such miswrings is to identify these two cases and disallow losing network connectivity and available bandwidth. In case 1, diagnostic protocol detects that configuring switch x as an edge switch results in isolation by losing all of its links. Likewise, in case 2, diagnostic protocol detects that configuring switch x as an edge switch creats to super-pod formation miswiring. Hence, the super-pod has to be split that causes switch x isolation. Once these two cases are identified, the misidentified edge switch x plays the role of both an edge and aggregation switch in case 1 and an edge, an aggregation and a core in case 2 until the end-hosts are removed from it.

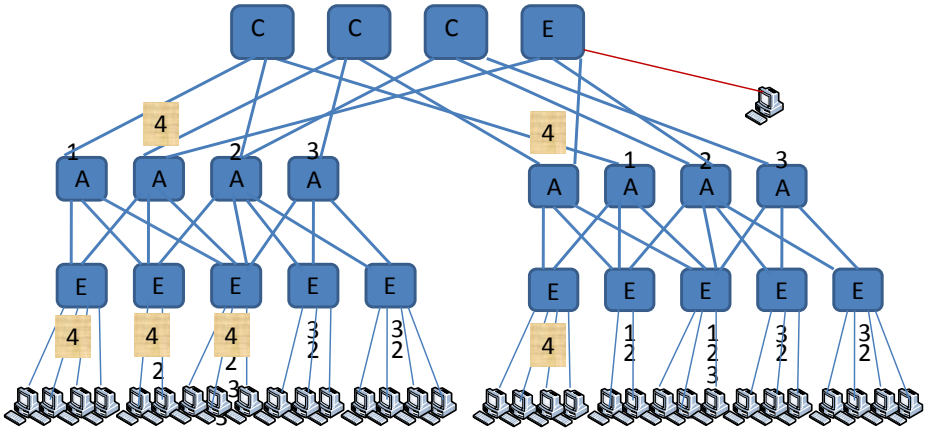

Figure 4.4: Case2: a host connected to a core switch

4.1.4 Group 4: Multiple link from a core switch to different aggregation switches within a putative pod

Such a scenario is considered as a miswiring case since having a multiple connection from a putative to a core switch neither increases the bandwidth nor enhances reachability. In fact, the main purpose of having a core switch is to augment reachability between putative pods. However, in a case where the desired network topology is a partial fat-tree, as shown in Figures 4.5, core switches have multiple connections to a pod but the connection pattern is similar in all core switches. For instance, both cores C1 and C2 are connected with two cables to each pod. If by any chance one the cables gets disconnected, then group 4 miswiring is produced.

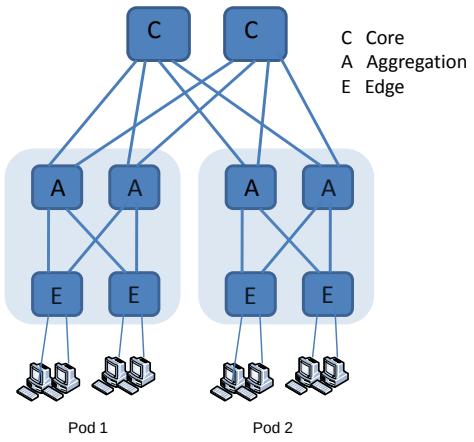

Figure 4.5: Partial fat-tree

4.2 Location Discovery Protocol: New Design

LDP transforms a given arbitrary network topology into a three level multi-rooted tree. However, If a switch is not connected to more than $k/2$ neighbor switches, it can be concluded that it is called an edge switch even though it is not connected to a host yet. Edge switch detection in such a way does not violate LDP liveness property as long as the underlying network is assured to be a fat-tree and no miswiring or failure is present. But in reality, such a strong assumption does not imply conformity with real case senarios such as having a multi-rooted tree instead of a fat-tree as the underlying network topology.

Therefore, a new desing of LDP, so-called Unified Naming and Routing Protocol (UNRP) with the goal of automating topology discovery and scaling forwarding for multi-rooted tree topologies that underlie many data center network fabrics is presented in this section. UNRP scales to a large number of switches and hosts, and is able to dy-

namically recover from arbitrary failures. In [9], UNRPŠs robustness and performance through model checking is shown and a proof of corectness plus an evaluation of a fully functional prototype is presented.

UNRP has two major phases: level assignment and pod assignment. In level assignment phase, one of the three levels of core, aggregation, and edge is assigned to a switch. In pod assignment phase, aggregation switches connected to the same set of edge switches receive the same LDP pod number. An edge switch, reachable through two different aggregation switches with two distinct LDP pod numbers, should be assigned with two different addresses.

4.2.1 Phase 1: Level Assignment

The level assignment phase allows each switch to discover its level within the hierarchy (edge, aggregation, or core). The level assignment phase starts once a switch detects a host connected to one of its ports. Then a wave of information from the lowest level of the tree which is edge to aggregation, and aggregation to cores will flow. This information is carried via keep alive-messages, so called Topology View Messages (TVMs), which are sent periodically from each switch to its neighbors for use in level assignment, address assignment, and the propagation of forwarding information. In addition to keep-alive messages sent periodically, each switch also sends ping requests to any neighbors not known as a switch. Hosts reply to pings but do not send TVMs, allowing switches to detect neighboring hosts. If hosts can be modified to support self-identification, the protocol becomes much simpler.

4.2.2 Phase 2: Pod Assignment

Once a switch knows its tree level, it participates in the address assignment phase of the UNRP. It is required to assign meaningful addresses to hosts through selection of appropriate coordinate numbers. The addresses consist of two elements: E-coordinate and A-coordinate. All aggregations connected to the same set of edges share the same number, so called LDP pod number. Aggregations with the same LDP pod number and the set of edges they are connected to form an LDP pod. As shown in Figures 4.6,

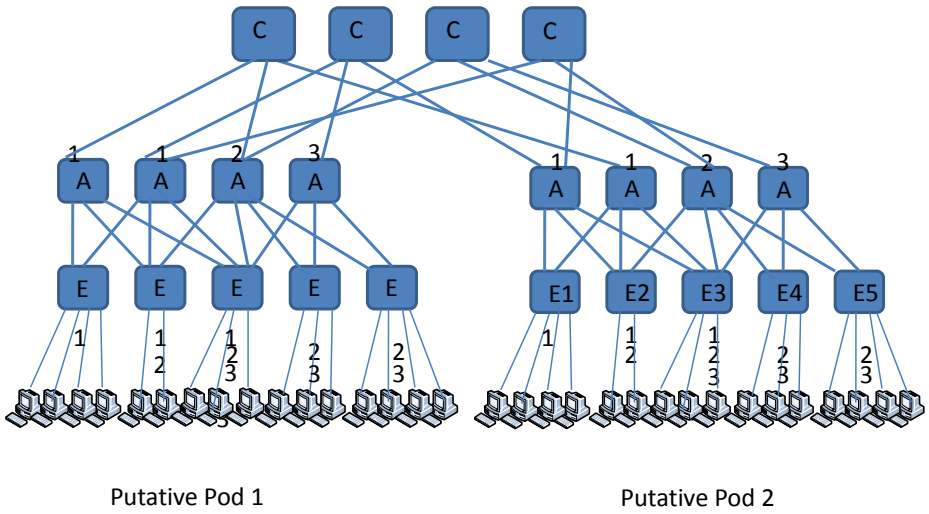

Figure 4.6: LDP Pods

switches (both edge and aggregations) with the same number are considered to be in the same LDP pod set. Edge switches use these numbers as their A-coordinate and aggregation switches use them as their LDP pod numbers. In addition, LDP relies on shared core switch parents to enforce the restriction that pairs of LDP pods do not select identical coordinate numbers.

Aggregation switches are also concatenated along paths from core switches to hosts to form host addresses. Aggregations select a core switch as their coordinates, while edges select an aggregation switch as their coordinates. Coordinate assignment relies on the concurrent operation of two tasks: Coordinate Restriction and Coordinate Combination. The former ensures unique addresses while the latter reduces required switch forwarding state.

In sum, set of aggregation switches connnected to the same set of edges share the same LDP pod number. However, edge switches in the same LDP pod should have a unique address to be reached with. The first part of this address is the LDP pod number

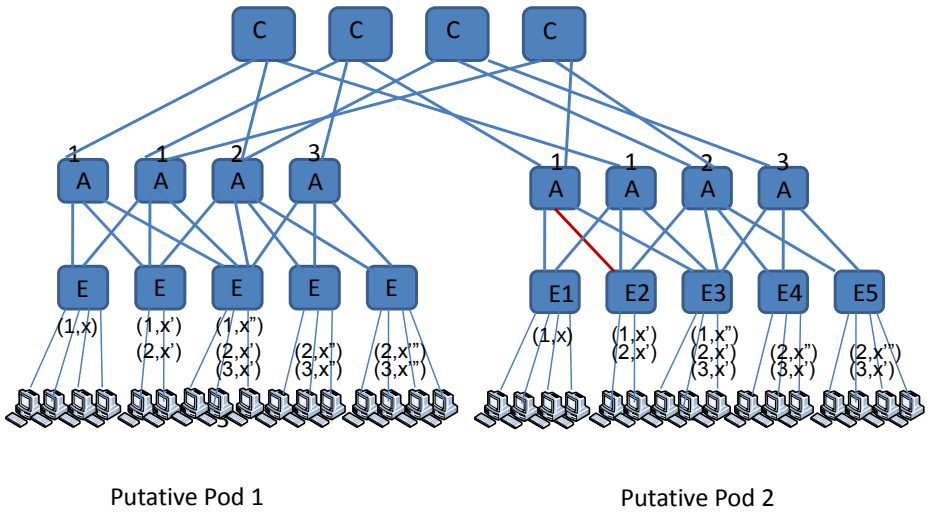

Figure 4.7: LDP Pods and edge switch address assignment

and the second part is a unique number that no two edge switches sharing the same LDP pod number should be assigned to. As shown in Figures 4.7, edge switches having the same LDP pod number 1 have addresses (1,x),(1,x') and (1,x") respectively, that x $\neq$ x' $\neq$ x". However, the edge switch in pod 1, with three addresses of (1,x"), (2,x') and (3,x') may or may not have the same second tuple. That means x" and x' may or may not be the same. This does not matter since the first tuple is the same.

Since UNRP is a live protocol and LDM messages are sent constantly and periodically, topology changes may affect aggregation and edge coordinates, thereby affecting host addresses. For instance, when the set of edge switches reachable by a particular aggregation changes, the aggregation may have to select a new pod number. Renumbering is the process for changing E and A-coordinates. Consider the example shown in Figures 4.7, where the highlighted link fails. At this point, affected switches must adjust their coordinates. With keep-alive messages, the aggregation switch detects no more keep-alive messages are received from E2. As a consequence, the aggregation switch

informs its edge set of its new connection status. Then it picks a new A-coordinate number that is acceptable by core parents switches. If the picked number has any conflict, it will choose another number and propose it again to core parent switches unil it gets accepted. Once the A-coodinate number gets accepted, edge switches connected to the cooresponding aggregation switch(s) will adjust their addresses acoordingly. The aggregation switch(s) are responsible to fix any conflicts that may occur when edge switches adjust their addresses.

It is important to note that the addition of a link can also cause renumbering. The effects of renumbering are primarily determined by whether the affected aggregation joins an existing or brand new pod.

Chapter 5

Network Diagnostic Protocol (NDP) Design

In this chapter, we present a diagnostic protocol for identifying the missing and miswired links in a multi-rooted tree topology, and for transforming it into a fat-tree, with respect to a given over subscription ratio. For simplicity, we assume that pod oversubscription ratio is one and that the switches used for building the data centers are all identical. The Network diagnostic protocol (NDP) provides cost estimate to build a pre-planed network topology with the current available hardware regardless of the current topology. NDP also presents suggestions about how many switches or wires are required and where they should be placed in order to achieve a fat-tree network topology. In addition, it provides an estimate about how far away a wired topology is from the desired/planned multi-rooted tree (if there exists one) and what is the cost of modification to improve it. NDP can help to ensure the accuracy of a newly wired up network topology for large systems like a 1000-host fat-tree data center.

5.1 Overview

In chapter 4, four types of miswirings (UNRP, super-pod formation, host misplacement, and multiple of links from a core switch to a pod) were introduced. These four groups set the basis for the heuristics of the NDP algorithm. However, it is useful to first consider a set of low-level faults. This set contains all the faults we consider,

and we can give upper bounds on the number of each type that can be detected. We can also give algorithm for detecting them. These bounds and algorithms motivates the heuristics we give later in this chapter.

NDP's main goal is to identify and handle these sets of faults and provide useful suggestions to improve the currently wired network topology. Therefore the goal of NDP is to fulfil the following two tasks:

1. Detecting potential faults that may occur in a multi-rooted network topology to recover the intended topology.

2. Providing helpful suggestions on how to improve current network topology wirings to improve network performance.

Before presenting more detailed description about these two major tasks and the possible combination of faults that can possible occur in a network, a better understanding of the following two concepts is required.

1. A description of the desired/planned network topology in terms of several factors: intentional pods size, number of pods, and core interconnection cabling matrix (reachability matrix).

2. A way to evaluate a topology based on two network evaluation matrices: oversubscription ratio and connectivity.

These two concepts are elaborated in detail in the following two sections.

5.1.1 Description of a desired/planned multi-rooted network topology

In this section, we introduce three parameters required to map a three-level multi-rooted tree into a simple data structure are introduced. A multi-rooted tree topology can be described with following parameters.

1. P: Number of intensional pods.

2. For each intentional pod, three tuples of:

(a) A.down: Number of downward ports of an aggregation switch.

(b) A.up: Number of upward ports of an aggregation switch

(c) E.down: Number of downward ports of an edge switch that are connected to end hosts.

3. Reachability[i][j] (i,j in [1..p]). This matrix gives the connectivity between pods via independent switches. Reachability [i][j] = Reachability [j][i], Reachability [i][i] = 0. The sum of elements in each row or column of the Reachability matrix is equal to the sum of upward ports of aggregation switches belonging to that intentional pod.)

There are several assumptions associated with this description.

1. All aggregation switches within a pod have the same number of downward ports.

2. All edge switches within an intentional pod have the same number of upward ports.

3. The number of aggregation switches within each intentional pod (|A|) is equal to the number of upward ports of an edge switch belonging to that pod (E.up).

4. The number of edge switches within each intentional pod (|E|) is equal to the number of downward ports of an aggregation switch belonging to that pod (A.down).

5. The cabling connectivity between |A| aggregation and |E| edge switches within an intentional pod resembles a full bipartite graph with |A| nodes in one set and |E| nodes in the other.

6. The tree uses the fewest number of core switches as necessary to satisfy the reachability matrix. For instance, in Figure 5.1 both a and b can be mapped to the reachability matrix below. However, only a is a valid mapping since there has only one core switch.

$$\begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

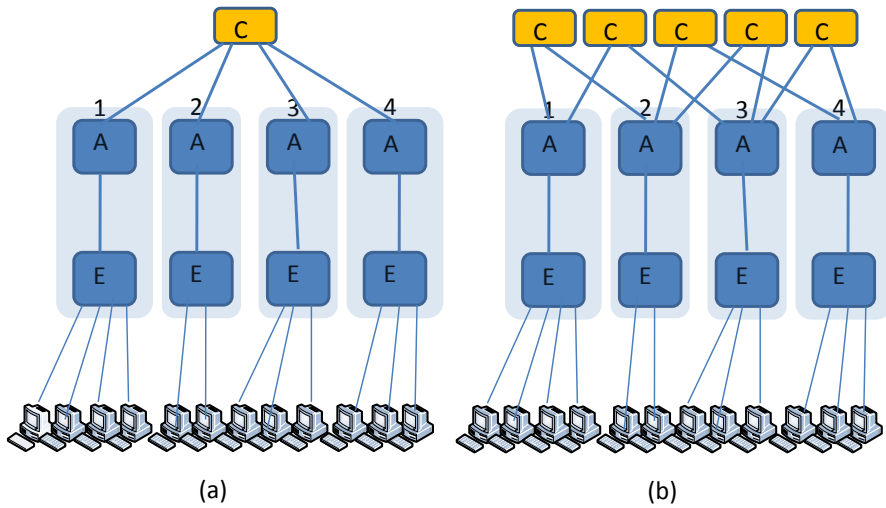

Figure 5.1: (a)Valid Reachability matrix mapping (b)Invalid Reachability matrix mapping

7. When mapping the reachability matrix to an actual three-level multi-rooted topology, the policy is to use as many disjoint aggregation switches as possible when all paths are going to the same pod, doing so allows better load distribution. As shown in Figure 5.2, both a and b are the mappings of the below matrix. The aggregation switch connected to the three cores in b can easily become a bottleneck. However, in a, the red links are distributed among two aggregation switches.

$$\begin{pmatrix} 0 & 3 \\ 3 & 0 \end{pmatrix}$$

In addition, we make there observation:

1. The ratio of upward ports to downward ports of an edge switch varies between $\frac{E.up}{1}$ to $\frac{E.up}{K-E.up}$ where K is the size of the edge switch.

2. The number of end-hosts in a pod depends on the edge switches size and number of their upward ports. Hence, we can claim that it is bounded by sum of size of

each edge switch in a pod minus number of its upward-ports for all edges within a pod. It should be noted that the number of upward ports used in all edge switches within a pod is the same. Hence, the larger an edge switch size is, the more it may get oversubscribed.

3. The minimum elements in Reachability[i][j] matrix where $i \neq j$ is the total number of core switches that connect all pods to each other.

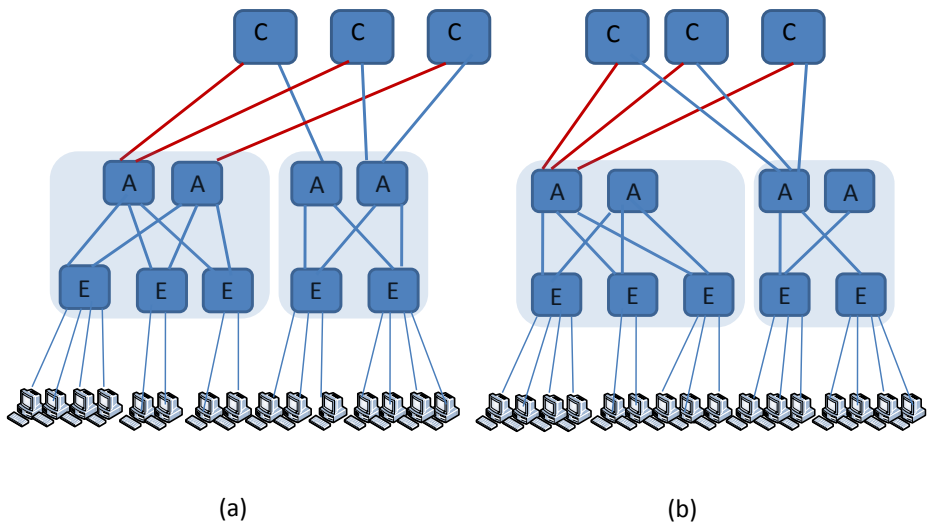

Figure 5.2: (a)Valid Reachability matrix mapping (b)Invalid Reachability matrix mapping

In summary, a network topology can be translated into a network description with a simple structure. Having the network description available, network topology performance can be measured in terms of oversubscription ratio and reachability. In the next section, these two metrics for performance measurement of a network description will be introduced.

We now look at metrics for the topology returned by UNRP. They are similar to the ones we just gave for ideal network.

5.1.2 Performance Measures

- Reachability:

  Reachability is a $P \times P$ symmetric matrix that indicates the level of connectivity between the P tentative pods in a multi-rooted topology. For simplicity, this matrix only stores the number of connections between pods but does not maintain by which aggregation switches this connectivity is provided. As shown in Figure 5.2, both a and b can be mapped to the same reachability matrix. In addition, the elements on the main diagonal of this matrix are all zero since having multiple connections from a pod to itself is counted as a miswiring. (Such a (group 1) miswirings can be detected by running UNRP).

  In general, data center designers prefer uniformity in the reachability matrix, so one way to detect bad wiring is by locating significant variation in the matrix. For instance, having one column or a row of the matrix filled with zeros indicates the existence of an isolated pod in the network. Therefore, it can be stated that either the aggregation switches in the pod are down or the cables are problematic.

  It is difficult to know what the connectivity matrix should be ahead of time. This is mostly because the communication pattern between end-hosts can be unpredictable. Hence, interpod communication cannot be anticipated in advance, precisely. Lacking more detailed infomation, network designers try to provide connection uniformity among pods. Often, network designers try to build a fat tree or a multi-rooted tree similar to a fat-tree (so called fat-like tree) architecture. However, it would be valuable to have feedback information from the network designer with respect to what the connectivity matrix is.

- Oversubscription:

  Oversubscription ratio is a fundamental measurement of a network performance. Basically, it is the access bandwidth divided by the amount of guaranteed bandwidth. To measure performance of a three level multi-rooted tree, several oversubscription ratios are defined.

  1. The intra-pod oversubscription ratio, which is the number of internal links of a pod divided by the number of hosts. The number of internal links of

a putative pod is equal to the number of downward ports of an aggregation switch times the number of upward ports of an edge switch in a pod.

2. The inter-pod oversubscription ratio between all pods, which is the minimum, off diagonal element of the reachability matrix times number of the pods divided by the total number of end hosts in the network topology. The minimum, off diagonal element of the reachability matrix indicates number of core switches that provide connectivity between all pods.
3. The inter-pod oversubscription ratio between pod i and pod j, which is reachability[i][j] divided by the sum of end-hosts connected to pod i and j.
4. The edge switch oversubscription ratio, which is the number of upward ports of a switch divided by the number of end-hosts connected to that particular switch.

Once we have the reachability matrix, the interpod oversubscription ratio can be calculated easily. To illustrate, the interpod oversubscription ratio between pods 1, 2 and 3, given the reachability matrix below, is 3 divided by the average number of end hosts connected to these three pods. This is because, 3 is the minimum number among 3, 5, and 7 which are the connectivity between pod 1 and 2, pod 1 and 3 and pod 2 and 3, respectively.

$$\begin{pmatrix} 0 & 3 & 5 & 3 \\ 3 & 0 & 7 & 2 \\ 5 & 7 & 0 & 3 \\ 3 & 2 & 3 & 0 \end{pmatrix}$$

All of these metrics are valuable tools for network performance measurement. In the next section, we explain how each of these metrics issued to provide optimal network topology that could be built with the current available hardware.

5.2 NDP: Fault Detection and Recovery

In this section, all possible faults that can occur in a network are categorized into 7 major groups. These 7 fault models can happen in isolation or in combination

with on another in a network topology. In this section, the upper bound for each fault occurrence, in isolation, is determined. Basically, if the number of fault occurrence exceeds a defined upper threshold, then the exact failures may not be enumerable. As we show, when when combination of these faults can occurs, the exact failures may also not be enumerable. These 7 categories of faults are as follows:

5.2.1 Fault Models

1. Missing links: if the only failures are missing links, then the intentional fat-tree architecture can be recovered - that is, we can detect the missing links reliably - as long as there are fewer than $K/4$ missing links per pod. This is because when the number of missing links in a pod can go beyond $K/4$ or more, an intentional pod may become partitioned. If that happens, figuring out which switches are part of an intentional pod can become impossible. It is true that by randomly selecting small partitioned pods to merge, a full fat tree can be recovered. Hence, since the switches were are physical hardware located in racks, there is no guarantee that the switches chosen to be merged have minimum physical distance to each other.

   This type of fault can be identified while running UNRP. The more the number of LDP pods in a putative pod, the more missing links faults are in the network topology. Assuming that the network is only suffering from missing links fault, it can be concluded that if no fewer than $K/4$ missing links exists in each putative pod, the original fat-tree topology can be retrieved by adding links.

2. Swapped links: Let (a_1, b_1) and (a_2, b_2) be links where a_i is the switch of higher level. There are two ways these links can be swapped: to (a_1, a_2) and (b_1, b_2) or to (a_1, b_2) and (a_2, b_1). In most cases, such swaps will result in parallel edge or peer edges, and so will fall into group 1 or group 4 miswirings. The exception is when (a_1, b_1) is between an aggregation switch and edge switch in one pod, and (a_2, b_2) is between an aggregation switch and edge switch in another pod. In this case, swapping to (a_1, b_2) and (a_2, b_1) results in creating a super pod. Given a set P' of intentional pods and s such swaps between pairs of aggregation-edge links in two of pods of P', the swapping can

create a cut of $2s$ in any super pod in P'. A min-cut algorithm can be used to detect such swapped links as long as $2s$ is the minimum cut. This holds as long as $s < K/4$. For example, Figure 5.3 shows two pods for K = 6, first with s = 1 and then with s = 2. Figure 5.3 *a* shows two swaps (s=2) which results in a cut of 4 separating the intentional pods is not a minimum cut. Howeve, Figure 5.3 *b* shows one swap and the resulting cut of 2 which is a minimum cut.

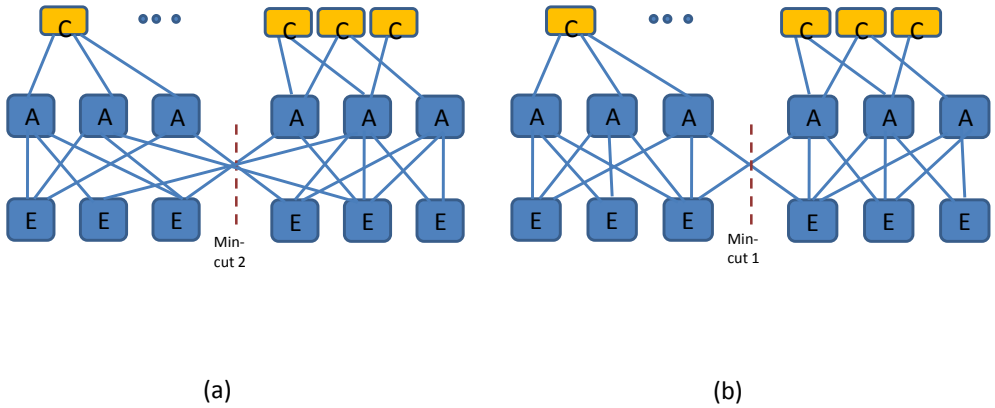

Figure 5.3: Case a is a pod with s = 2 and case b is a pod with s =1.

3. Host connected to an aggregation switch: If the only failures are hosts connected to aggregation switches, then we can detect all of the failures as long as there are fewer than $K/2$ aggregation switches with such hosts per pod. This is because, UNRP needs to distinguish between seemingly adjacent switches appear to be edges; UNRP resolves this by deactivating links incident at the aggregation switch.

4. Hosts connected to a core switch: If the only failures are host(s) connected to core

switches, we can detect all of the failures as long as there are $K^2/4$ core switches with such hosts per pod. There are $K^2/4$ core switches at most in a fat tree and this is fewer than total number of edge switches which is $K^2/2$. Therefore, even if all core switches turn into an edge still we would be able to detect the fault. We can distinguish between real edge switches and cores connected to host(s) by observing the fact that such core switches form super pods. In addition, these core switches provide only one links between each two pods formed a super pod. Therefore, running the min-cut algorithm can identify such miswirings. Since NDP deactivates links identified crossing the min-cut in the topology model, running min-cut results in such switches isolation - that is, all of the links connecting such a switch to other switches become deactivated.

5. Crashed edge switches: If the only failures are crashed edge switches, then all of the faults can be detected and the fat-tree can be reconstructed as long as each pod has fewer than $K/2$ crashed edge switches. This is because, at least one edge switch must exist within a pod to provide full connectivity to all aggregations within a pod. Otherwise, a pod will end up having an isolated switch. Thus, a correct aggregation switch cannot be associated into its correct pod.

6. Crashed aggregation switches: If the only failures are crashed aggregation switches, all of these failures can be detected and the fat-tree can be reconstructed, as long as each pod has fewer than $K/2$ crashed aggregation switches. This is because each pod must have at least one aggregation switch so that the existence of three levels of edge, aggregation, and core is not violated. Hence if $K/2-1$ aggregation switches are crashed and only one aggregation switch is left in a pod, the remaining aggregation switch guarantees connectivity to all edges in a pod since the assumption is this is the only failure can occur in the network.

7. Crashed core switches: If the only failures are crashed cores, then they can be detected and the fat-tree architecture can be reconstruct as long as no more than $K^2/4$ core switches are crashed. That means, if all of the core switches are crashed, we still can detect such a fault.

We use these 7 fault low level fault detection algorithms to motivate the heuristics underlying NDP. This section is the basis for the next section which introduces NDP algorithm and the heuristics to handle such 7 faults.

5.2.2 Fault Detection and Recovery

In this section, we give the 3 phases of NDP that results in fault identification and recovery. During these 3 phases of NDP all 4 types of miswirings, introduced in chapter 4, are detected and heuristic algorithms are used to fix them. Noted that NDP does not either fix the miswiring cases or add links in the actual topology. However, NDP maintains a model of a topology that it updates and it returns list of prioritized suggestions. These 3 phases are the following:

1. phase 1: Run the UNRP, described in chapter 4.3, on a given wired network topology. Fix group 1 miswirings and identify group 3, case *a* miswirings. Obtain putative pods sizes and deficiency degree of each putative pod.

2. phase 2: Identify group 2 miswirings by comparing ideal topology description with the current number of pods and pod sizes. In case of group 2 miswirings, running min-cut to detect how to create apart as super. Then, having repaired group 2 miswirings, if a switch has become isolated then there was a case of group 3 case *b* miswiring.

3. Phase 3: Identify and recover group 4 miswrings by removing multiple links connecting a core switch to a putative pod by comparing core switch connectivity lists.

Phase 1:

UNRP detects group 1 failures as well as group 3, case *a* failure. In this section, we describe what is the information that is required to be obtained from UNRP before running NDP.

1. Switch identifier (switch id): a unique 48-bit identifier for each switch, e.g., the lowest MAC address of all local ports.

2. Putative pod number (pod): an 8-bit number shared by all switches in the same putative pod. Switches in different pods will have different pod numbers. This value is never set for core switches.

3. LDP pod number (group id): a unique 48-bit number shared by all switches in the same LDP pod. Switches in different pods will have different pod number at all times which assists in eventually ensuring uniqueness of pod numbers assigned to each pod. This value is never set for core switches.

4. Tree level (level): 0, 1, or 2 depending on whether the switch is an edge, aggregation, or core switch.

5. Port number, up/down and free/deactivated (dir): a switch-local view of the port number along which the LDM is forwarded. Up/down is an additional bit which indicates whether the port is facing downward or upward in the multi-rooted tree. Free/deactivated indicated if the port is free or connected to a link which became deactivated due to fault detections by UNRP run.

6. Number of K-port switches available in the topology. The assumption is that all switches are similar.

7. aggregation with a misplaced host, list of uncrashed isolated edge switches that all their links were deactivated while UNRP run.

What we compute from the information obtained above is the following:

1. Switch deficiency degree: A switch is called d-deficient if the switch is missing no more than d links as compared to a full bipartite graph.

2. List of aggregation switches switches connected to each core switch expressed in terms <switch id, putative pod number, deficiency degree>

3. List of switches belonging to each LDP pod, expressed in terms of <switch id, Tree level, putative pod number>.

4. List of putative pods sizes, a list of tuples in form of <putative pod number, pod size>

While UNRP is running on a given network topology, all miswirings classified as group 1 and group 3, case *a* can be identified. To elaborate, running UNRP results in edge switch isolation which indicates group 3, case *a* miswiring. Having the list of uncrashed edge switches that became isolated due to link deactivation allows us to detect the host misplacement fault (a host connected to an aggregation switch).

The end host(s) connected to uncrashed isolated edge switches can no longer communicate with the rest of the network. Hence, disabling the end hosts and reactivating the links at least provides higher bandwidth for the rest of the network to communicate. This is what NDP does. NDP deactivates the misplaced host(s) in the topology model that it maintains to fix such miswiring case.

However, a more efficient approach is to improvement UNRP to allow these isolated edge switches to act both as an edge and aggregation switches until the misplaced hosts are removed. So that, the links connected to such a switch do not get deactivated or removed. Noted that this is not the focus of this thesis.

For all those miswirings that lead to deactivation of a link due to group 1 miswirings detection, NDP assumes that the link can be removed and the corresponding pair of ports can later on be treated as free ports.

Phase 2:

Phase 2 focus is to identify and fix group 2 miswirigns (super pod formation). Since it is assumed that switches have the same size, no more than K switches can be grouped in a putative pod. Hence, if a putative pod with size larger than K is identified, it must be a super pod. NDP compares the putative pod sizes with value of K. If the putative pod size is larger than K, NDP selects the corresponding pod as a super pod. Splitting the super pod requires finding nontrivial partitions of the super pod into two or more parts such that the sum of the weights of the edges connecting the two parts, is minimum.

The best known max-flow min-cut algorithm is Ford Fulkerson algorithm, which reveals a minimum s-t-cut in a graph. Hence, s and t are two vertices that are the source and the sink in the flow problem and are to be separated by the cut.

We cannot use Ford Fulkerson algorithm for two reasons. First, we have no

source nor destination node; having these is equivalent to knowing the intentional pod. Second, Ford Fulkerson only computes a single min-cut; we need to enumerate the min cuts.

The first problem can be solved by using Karger's min-cut algorithm introduced in [1]. This is a heuristic algorithm that finds the minimum cut with the probability of larger than Ω $(1/\log n)$ for an n node graph. By repeatedly performing this algorithm, this probability can exceed. Running the min cut repeatedly c.log^{2}n times, guarantees that the algorithm outputs the minimum cut with the probability equal or larger than 1 - $1/n^2$ which is a very large accuracy percentage.

The second problem can be solved by splitting the graph after the first min-cut is found and finding the the min-cut that is larger than K.

However, before running the heuristic algorithm, it is beneficial to identify super-pod cases where even cut would not provide beneficial cut suggestions. Our approach is to identify the largest LDP pods in the super pod and remove them. We then check whether the rest of remaining switches are partitioned appropriately. For instance, suppose K is equal to 6 and the putative pod has 13 switches. This must be made up of at least 3 intentional pods since each intentional pod has 6 switches. However, removing the first largest LDP pod, and then remove the remaining largest LDP in the remaining set resulted in more than one partition. If so, we conclude that there are more than two cuts required to split the super pod. In addition, running min-cut would not reveal two cuts to split the super pod into exactly three parts.

The analogy behind this approach is as follows. The largest LDP pod has the lowest deficiency degree. Accordingly, the switches in the largest LDP pod have stronger connection with each other as compared with the rest of the switches. Therefore, min-cut algorithm will never provide a cut that splits the largest LDP pod. Hence, all the nodes in the largest LDP pod need to be in one set. Since a cut splits a putative pod into two sets, if there exist no path between switches not listed in the largest LDP pod set, then they cannot be grouped into one set. Thus, if the largest LDP pod is removed and the remaining switches are not reachable through a single path, then running the min-cut algorithm will not reveal an appropriate cut to split the super-pod.

When min-cut algorithm cannot reveal a desirable min-cut, the policy is to split

Algorithm 1

```
phase 1:
Input:(S[n] List of switch id of uncrashed isolated edge switches returned by UNRP)
for i = 1 to n
    deactivate ports connected to end hosts
    reactivate ports connected to switches
phase 2:
Input:(List of putative pod sizes returned by UNRP P_i.size, switch size K, P_i.switch List
of switches in each putative pod)
for i = 1 to n
    if(P_i.size] > K)
        cut ← ⌈ P_i.size / K ⌉ - 1
Pod_n : List of switches in putative pod n
for i = 1 to n
    Pod_n_copy ← Pod_n
for i = 1 to cut
//remove Largest LDP pod size from Pod_n
    partition_i ← Largest LDP pod in Pod_n
    Pod_n ← Pod_n - partition_i
    identify the Largest LDP pod in the Pod_n
if (P[n] == a single connected component)
    huristic_min-cut(Pod_n_copy, cut)

for each set of switches in partition_i
        assign a new pod number
        update Core[n] List of putative pods each core switch is connected to
        compute aggregation switches deficiency degree
huristic_min-cut(Pod_n_copy, cut)
phase 3:
Input:(Core[n] List of putative pods each core switch is connected to)
```

the super-pod into the sets resulted from removing the largest LDP pods. For instance, if a super-pod was formed and the largest LDP pod removal resulted in two independent

sets of switches, then we need to split the super pod into three sets instead of two.

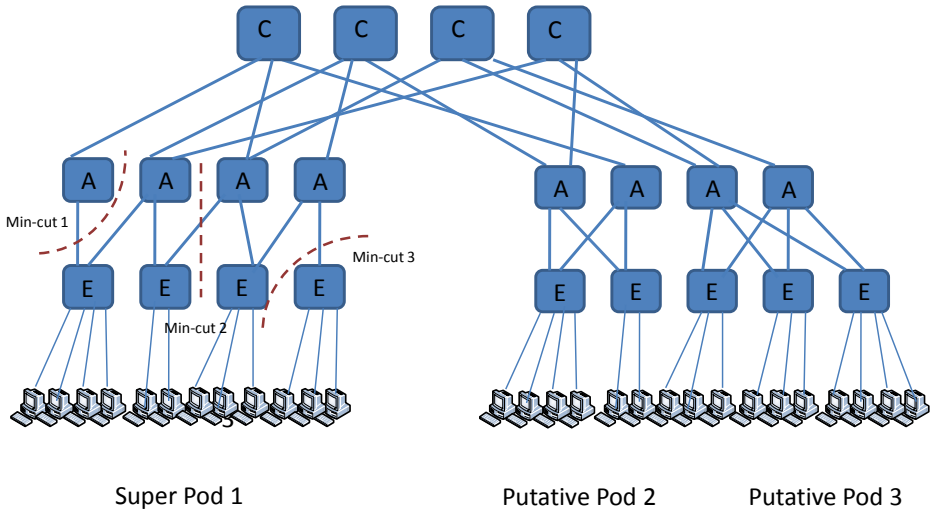

Figure 5.4: Three min-cuts can be identified but the appropriate min cut is min-cut2 which splits the network into two balanced sets.

While the min-cut algorithm is running, if a switch becomes isolated then case b of group 3 miswirings is detected. This is because a core switch provides connection between aggregation switches of each pod. If a core switch turns into an edge switch it causes super pod formation. In order to split a super pod, we run min cut algorithm and min cut algorithm deactivates all links from such a core switch to aggregation switches which leads to switch isolation.

There is also an additional way to identify group 3, case b miswirings: the number of aggregation switches through which this switch is connected is larger than $K/2+1$. This happens because each putative pod is allowed to hold at most $K/2$ aggregation switches. However, if the number of aggregation switches are less or equal to $K/2$ then the min-cut algorithm must catch the miswiring case.

In order to fix such miswring, NDP deactivates the end host(s) connected to the

isolated switch, reactivates the links connected to the corresponding switch, and sets the switch level as core. In addition NDP treats the ports connected to the remaining deactivated links as free ports.

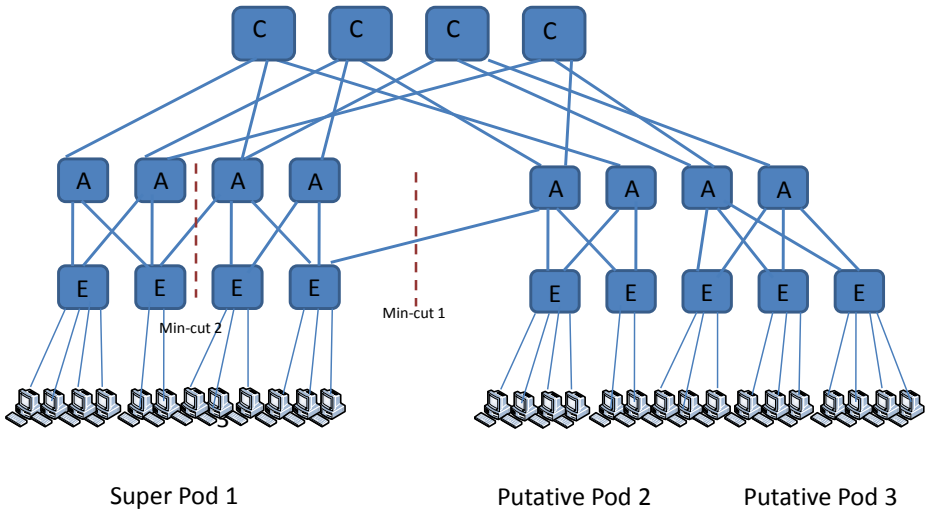

Figure 5.5: Both min-cuts that are identified are appropriate ones.

Phase 3:

After splitting super pods, a new putative pod number must be assigned to each newly formed putative pods. This information is required for accurate routing and forwarding purposes at core switches. NDP does that by checking the list of putative pod numbers in use obtained from UNRP results, to avoid conflicts in putative pod number assignment.

Once each putative pod received its unique putative pod number, core switches update their connectivity lists. More than one connection from a core switch to a pod indicates group 4 miswiring under the assumption that the intentional topology is a fat-

tree. NDP identifies such miswiring and deactivate additional connections to the same pod in its topology model.

We can observe that NDP also can identify group 4 miswiring when the intentional topology is a partial fat-tree. In a partial fat-tree the core switches connectivity lists follow a pattern. That means, all core switches have certain number of connections (could be more than one) to each pod. Assuming that the majority of core switches follow the same pattern, group 4 miswiring for partial fat trees can be identified. However, current NDP is written based on the assumption that the intentional topology is a fat tree.

In sum, after removing possible group 4 miswirings, the core switches connection list gets updated. Any link causing multiple connection to the same pod will become deactivated. Then the core switches will be sorted based on their connections in a decreasing order. This sorted list is used in phase 5 to improve the current topology model.

5.3 NDP: Network Improvement

During the 3 phases of NDP, all types of miswirings are detected and the topology model maintained by NDP is updated with recovery changes. In this section the focus is to improve the performance of the network considering the two metrics oversubscription and reachability. In phase 4, NDP tries to improve network performance by adding links but not switches in appropriate places in the network to reconstruct the intentional fat tree.

The reason why switches are not added is because it is possible that in phase 2, a super pod is partitioned into parts more than what the value of putative pod size divided by K show. Therefore the total number of putative pods in the topology may exceed K. In that case, the extra pods must become deactivated and their switches must be added to pods with size smaller than K. Since this problem involves merging pods and is very difficult to solve, we decided to improve the topology by adding links only.

In addition, NDP computes the optimal network topology based on the current available hardware regardless of current topology in phase 5. This information gives an

intuition about how efficiently the current available hardware could have been wired to provide optimal performance.

Phase 4: Improving Current Network Topology

The goal of this phase is to add links to the network to improve both the bandwidth and the connectivity of the current topology. Adding links between the first two layers (edge and aggregation) results in bandwidth and reachability improvement within a certain pod while adding links between the second and third layers (aggregation and core) improves interpod reachability. The analogy to improve both at the same time is to have at least one 3 level single-rooted tree in the network shown in Figure 5.6. Having such a tree ensures reachability between all end-hosts and a fair amount of bandwidth between them to communicate. Hence, by improving the network topology we mean to add more of such a tree into the topology.

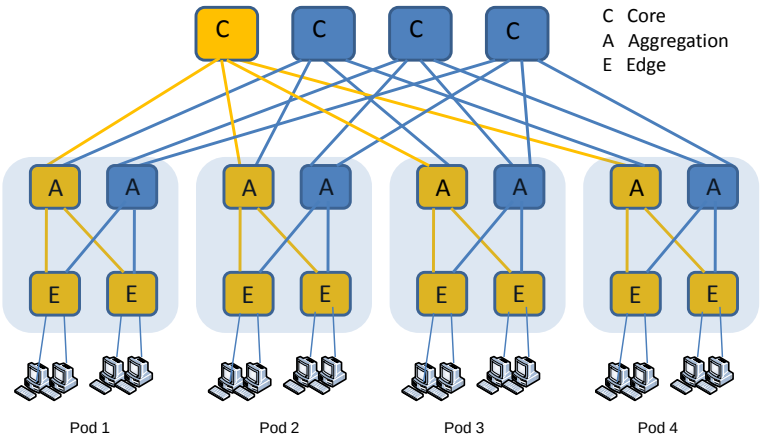

Figure 5.6: The yellow single-rooted tree provides reachability for all end-hosts in the topology.

The output of the Algorithm 4 is a list of core switches, through which the yellow single-rooted tree, shown in Figure 5.6, can be built. In every round of the algorithm, some number of links are added into the network topology. The algorithm terminates once no more such a tree shown in Figure 5.6 can be found.

NDP can be improved by obtaining information about desirable connectivity pattern describing where to add links provided by network administrators. However, NDP currently is implemented assuming that no default preference or setting is given.

NDP also can be improved to obtain the budget available and the cost associated with link addition so that the network administrators could decide on which set of changes to impose on the network. Since we did not have enough information about what link costs expected to be provided by network administrators, we did not implement this part.

Phase 5: Computing Optimal Network Topology

In order to build a K pod fat-tree, $5/4K^2$ switches, each with at least K ports, are required. Here we check whether it is feasible to build a fat-tree with F switches by running the code below. Once the largest value of K for building a fat-tree is determined,

Algorithm 2

```
    //Receive core switches list and the list of aggregation switches each of cores is connected to
    Input(List Core[n])
    Sort Core[n] in a deceasing manner
 4: minimum-deficiency ← ∞
    sum ← 0
    for i = 1 to n
        sum ← Core[i]
 8:     for j = 1 to Core[i]
            sum ← sum + aggregation-deficiency-degree[j]
            if( sum ≤ minimum-deficiency)
            minimum-deficiency ← sum)
12: return minimum-deficiency,i
```

Algorithm 3 terminates. Algorithm 4 checks if there are core switches left with free

ports to be used for additional pods formation. If so, then the number of the pods will be incremented.

Algorithm 3

```
//Receive switch size and number of which available
Input(SwitchSize S,SwitchFrequency F)
    A ← √(5.F/4)
    A ← Min(A, S)
//Return Largest number of ports that should be in use
return A
```

Algorithm 4

```
//Receive largest number of ports in use, switch size and number of which available
Input(SwitchSize F,SwitchSize S, A)
//List free ports of core switches
Pod ← S
Core_free_ports ← S - A
F ← F - 5/4A^2
for i = 1 to Core_free_ports
    if F ≥ S
        pod ++.
        F ← F - S
return pod
```

5.4 Summary of NDP:

1. Faults detection and Recovery:

    - In phase 1, the output of UNRP protocol which is a multi-rooted tree in which all group 1 miswirings being resolved is provided as the input of the diagnostic protocol. If an edge switch becomes isolated due to link deactivations during LDP run, the case must be reported as the host misplacement.

NDP fixes this case by deactivating the misplaced host(s) and reactivating the links connected to the corresponding switch in its topology model.

- In phase 2, size of each pod is checked to be no larger than K. If any pod size is larger than K, super pod has been formed and the pod has to be split into two or more smaller pods. Running the heuristic min cut algorithm computes the appropriate cut or cuts. If the proportion of super pod to K indicates that only c cut is required to split the super pod but the removal of the largest LDP pods results in more than c independent sets of switches then definitely the min-cut algorithm cannot return all desirable cuts. In this case, the policy is to split the super-pod with (the number of the remaining independent sets + the number of largest removed LDP pods) cuts.

  After resolving super-pod formation, the host misplacement classified as group 3 miswiring case b (where at least one host is connected to a core) can be identified because running the min-cut algorithm results in edge switch isolation. The policy to resolve the case is to reactive the links connected to the isolated switch until the hosts are removed. Meanwhile, the corresponding switch runs the code sets of edge, aggregation and core switches.

- In phase 3, core switches update their connectivity lists and deactivate any link that results in multiple connections to the same core switch. Therefore, NDP detects and fixes goup 4 miswirings (more than one connection from a core switch to a putative pod) in this phase. Right after this phase, all bad wires are deactivated and all deactivated links are marked to be treated as free ports.

2. Providing helpful suggestions to improve the faultless topology

- In phase 4, links are added to the topology in a fashion that a single-rooted three-level tree consists of one core and K aggregation switches, providing reachability between all end-hosts, is added in every round. In another words, in every round of the algorithm, a minimum set of links required to build the single-rooted tree that provides full connection is added to the topology. The policy to pick a link from a core to an aggregation is to select

an aggregation with the smallest deficiency degree and a free port.

- In phase 5, based on the available number of K-port switches, the number of ports that should be in use for building the largest fat-tree is computed. This phase gives an intuition to network administrators on how efficiently the available hardware could have been used.

Chapter 6

Experimental Results and Evaluation

UNRP is a live protocol that runs in the background. We can assume that eventually network topology changes stabilize so do the UNRP results. The UNRP is written in Mace, a language for building distributed systems. In [9], Mace modelchecker is used to verify that the UNRP implementation matches specification, which in turn produces the expected results, i.e. the addresses are assigned correctly and forwarding operates accurately in the presence of network failures. Therefore, the results given here are based on the accuracy of the results presented by UNRP.

Once the network is stable, the output results of UNRP is fed into NDP. The procedure for the evaluation of the NDP is shown in Diagram 6.1. For a given value of K, a $K^3/4$ host fat-tree is generated and then the faults are injected into the fat tree topology. Running UNRP and then the NDP produce a list of failures being detected. This list may or may not exactly match the list of faults being injected initially. We show that if the 7 cases of faults are injected independently, within their defined boundaries, they all can be detected. Figure 6.2 indicates such a claim.

However, when the combination of these 7 fault cases are injected into a fat-tree (Figure **??**), it may or may not be possible to retrieve the original fat-tree. In addition, we present an analysis of why some of these cases are undetectable. We also demonstrate a comparison between the currently wired topology and the optimal topology when the injected fault is the switch crashes. The comparison between the optimal topology and the current topology indicates how efficiently the currently wired topology could have used the available hardware.

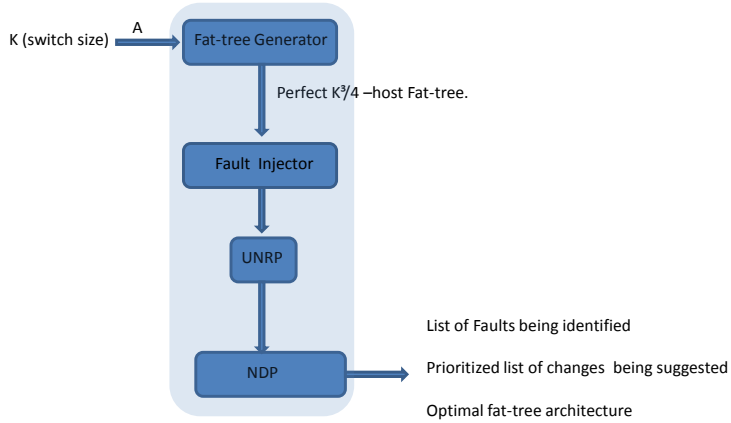

Figure 6.1: Diagram of protocol evaluation and results generation

Given a value of K, a fat-tree topology is generated as follows: $K^3/4$ hosts and $5/4K^2$ switches, of which $K^2/4$ are initially intended as cores, $K^2/2$ 2 as aggregations, and $K^2/4$ as edges. The values of K for which the results are generated is 16.

In order to inject the faults into a fat-tree, the following fault injector functions are used.

1. x_link_disabler: selects and disables x random links. The upper boundary for x is less than $K/4$ in each pod.

2. x_type_switch_disabler: selects and disables x random switches from a type: core, aggregation or edge. When the type is not specified, the switches can be selected from any type. The upper boundary for x highly depends on the specified type. If the switch type is core, the upper bound is less than $K^2/4$. If the switch type is aggregation or an edge, the upper bound is less than $K/4$.

   Since it is assumed that the number of edge and aggregation switches within a pod

are equal, if randomly selected switch to be deactivated is an edge (aggregation) in a pod, the second randomly selected switch must be an aggregation (edge) within the same pod. Therefore, even number of switches needs to be disabled in a pod.

3. x_host_misplacer_type: selects x random switches from the specified type and connects an end-host to the selected switches. If any of the these switches does not have a free port, a randomly selected link connected to the switch gets disconnected to provide a free port. The free port will then be connected to an end-host. In addition, the value of x in x_link_disabler fault injector will be incremented by one since a link to another switch was required to be removed before being replaced by an end-host. The upper bound for the host misplacement (connection to either an aggregation or a core switch) is less than $K/4$. When the type is not specified, a host can be connected to a randomly selected edge or an aggregation. In that case, the upper bound for x is the summation of the upper bounds for each case when they are injected in isolation.

4. x_link_swapper: for x times, randomly selects two pairs of cables connecting any two switches and swaps them.

5. x_super-pod_former: selects a random edge switch in one pod and a random aggregation switch in another and connects them by a cable. If any of these switches did not have a free port, a randomly selected link would be broken and used for super-pod formation fault. In addition, for any links being broken to free a port, x value of x_disabler fault increments by one. The reason why this fault is separated from the link_swapper fault is that we would like to form a super-pod when injecting the faults.

All possible 7 fault models, mentioned in chapter 5, can be injected via one of these fault injectors functions, within their upper bound being defined.

In order to evaluate the NDP, a tunable number of faults are injected into the fat-tree built with K port switches. The results shown in Figure 6.2 indicate that the NDP does a perfect job detecting all type of faults injected in the network. Inter-pod oversubscription ratio, evaluated in Figure **??**, is only presented for the cases where the network suffers from switch failures. This is because inter-pod oversubscription ratio

Fault Type	Upper Boundary	Number of the Runs	% of the Faults	% of Detection		
				Avg	Max	Min
Missing links	less than K/4 per pod	25	%20	%100	%100	%100
			%50	%100	%100	%100
			%70	%100	%100	%100
Swapped Links	less than K/4 per pod	25	%20	%100	%100	%100
			%50	%100	%100	%100
			%70	%100	%100	%100
Host connected to an aggregation	less than K/4 per pod	25	%20	%100	%100	%100
			%50	%100	%100	%100
			%70	%100	%100	%100
Host connected to an edge	less than K/4 per pod	25	%20	%100	%100	%100
			%50	%100	%100	%100
			%70	%100	%100	%100
Crashed edge	less than K/4 per pod	25	%20	%100	%100	%100
			%50	%100	%100	%100
			%70	%100	%100	%100
Crashed aggregation	less than K/4 per pod	25	%20	%100	%100	%100
			%50	%100	%100	%100
			%70	%100	%100	%100
Crashed Core	less than $K^2/4$ per pod	25	%20	%100	%100	%100
			%50	%100	%100	%100
			%70	%100	%100	%100

Figure 6.2: Fault Detection Evaluation for K=16

highly depends on the portion of faults injected through thee switch_deactivator_type fault injector. In fact, when the number of switches within each pod decreases, the number of core switches that provide connectivity between all pods decreases; thereby lowering the inter-pod oversubscription ratio. However, when the combination of switch deactivation and super-pod formation failures is injected into the network, it may results in the enhancement of the inter-pod oversubscription ratio. This occurs due to the fact that the switch failures reduce the number of the pods but the super-pod former increases the number of the switches within each pod; thus producing higher inter-pod oversubscription ratio. The results indicates that when the size of the pods are shrinked, the number of the edge switches upward ports becomes less than K/2. Therefore, the intrapod oversubscription ratio may be decreased as well.

In addition, the number of rounds, in which wires are added to the topology to improve network connectivity, enhances as the portion of the faults of link-deactivator

elevates. The results also indicate that, at some point, when the number of misplaced end hosts goes beyond a certain threshold (Figure **??**), the identification of such miswiring becomes impossible. This is due to switches cannot be classified as either cores or aggregations anymore, and the majority of switches becomes edges switches.

Even though the results of the NDP are expressed in terms of prioritized chain of changes required for network improvement, we did not evaluate the difference between the initial fat-tree and the improved fat-tree oversubscription ratios. It is apparent that the improved topology provides higher oversubscription ratio but what the focus of this chapter is to evaluate how well the NDP detects faults.

In sum, we have assessed how the NDP detects faults, how the heuristics meet their bounds, and how well the NDP detects the combined faults.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

Introducing a protocol that diagnoses whether the performance is diminished due to bad wiring or miswirings and that presents suggestions for the enhancement of performance, given a certain budget limit, is the main focus of this thesis. It demonstrates how to identify all possible types of miswirings by first running a new version of Location Discovery Protocol (UNRP) and then running the network diagnostic protocol (NDP) on a given network topology.

UNRP intends to transform a given network topology into a 3-tier multi-rooted tree, which is what most data centers look like nowadays. The network description gained through running UNRP is treated as an input for NDP. This thesis also shows how the diagnostic protocol compares the input network description with what it can compute itself, given the information about available hardware and desirable administrative constraints. The comparison between these two descriptions provides an insight regarding how far the current topology is from what could have ideally been wired up. In addition, NDP provides suggestions for improving the current topology, given a budget limit.

In this thesis, it is illustrated that oversubscription ratio and reachability metrics, end-hosts communication pattern, available hardware in terms of switches and cables, and available budget for network topology improvement influence the diagnostic protocol results. In addition, it is shown how the diagnostic protocol generates results when

no information about desirable oversubscription ratio, reachability and end-host communication pattern is not available.

The expected experimental results generated by NDP expresses the claim for its correctness. The experiment is by running UNRP and NDP on a given set of network topologies injected with various faults in terms of miswirings. The results indicate that for any given isolated fault, the list of faults being detected matches the list of faults being injected initially as long as faults are bounded. NDP also generates a prioritized list of required changes by trading off between main two factors of reachability and oversubscription.

In conclusion, the diagnostic protocol holds promise for data center performance enhancement which results in improving overall scalability.

7.2 Future Works

Bandwidth-sensitive applications and the growth of technology trends inquire use of commodity servers and the deployment of visualizations. Scalability faces space, location and resource constraints and inquires network elements modularity, auto-configuration (plug and play) and energy preservation. Cost for failure resilience is also another issue. Such inquiries bring about the need for an application specific diagnostic protocol with the ability to provide detailed information about exact required hardware and topology connectivity constraints. The diagnostic protocol introduced in this thesis is a preliminary work on this topic. However, the need to deploy diagnostic protocols for large scale, cost-efficient fault-tolerant, and green data centers that makes them intelligent and plug and play is highly sensible and desirable compared to the past.

Bibliography

[1] Min Cut. http://valis.cs.uiuc.edu/~sariel/teach/notes/rand_alg/lec/02_mincut.pdf.

[2] OpenFlow. http://openflowswitch.org/.

[3] Power Energy Efficientcy of Data Centers. http://sun.com/aboutsun/environment/docs/powering_energy_efficientdc.pdf/.

[4] M. Al-Fares, A. Loukissas, and A. Vahdat. A Scalable, Commodity Data Center Network Architecture. In *SIGCOMM '08: Proceedings of the ACM SIGCOMM 2008 conference on Data communication*, pages 63–74, New York, NY, USA, 2008. ACM.

[5] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu. BCube: A High Performance, Server-Centric Network Architecture for Modular Data Centers. In *SIGCOMM '09: Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, 2009.

[6] C. Guo, H. Wu, K. T. Shiy, Y. Zhang, and S. Luz. DCell: A Scalable and Fault-Tolerant Network Structure for Data Centers. In *SIGCOMM '08: Proceedings of the ACM SIGCOMM 2008 conference on Data communication*, pages 75–86, New York, NY, USA. ACM.

[7] R. N. Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subram, and A. Vahdat. PortLand: A Scalable Fault-Tolerant Layer 2 Data Center Network Fabric. In *SIGCOMM '09: Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, pages 39–50, Barcelona, Spain, 2009. ACM.

[8] A. Pamboris, 2009. LDP: Location Discovery Protocol for Data Center Networks.

[9] M. Walraed-Sullivan, R. N. Mysore, P. Miri, K. Marzullo, and A. Vahdat. Automated, Scalable Multi-Path Routing and Forwarding for the Data Center. In *SIGCOMM '10: Proceedings of the ACM SIGCOMM 2010 conference on Data communication*.

[10] H. Wu, G. Lu, D. Li, C. Guo, and Y. Zhang. MDCube: A High Performance Network Structure for Modular Data Center Interconnection. In *SIGCOMM '09: Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, 2009.