COPPE UFRJ

Instituto Alberto Luiz Coimbra de Pós-Graduação e Pesquisa de Engenharia

A HIGH-PERFORMANCE TWO-PHASE MULTIPATH SCHEME FOR DATA-CENTER NETWORKS

Lyno Henrique Gonçalves Ferraz

Tese de Doutorado apresentada ao Programa de Pós-graduação em Engenharia Elétrica, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Doutor em Engenharia Elétrica.

Orientador: Otto Carlos Muniz Bandeira Duarte

Rio de Janeiro
Novembro de 2015

A HIGH-PERFORMANCE TWO-PHASE MULTIPATH SCHEME FOR DATA-CENTER NETWORKS

Lyno Henrique Gonçalves Ferraz

TESE SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE DOUTOR EM CIÊNCIAS EM ENGENHARIA ELÉTRICA.

Examinada por:

RIO DE JANEIRO, RJ – BRASIL
NOVEMBRO DE 2015

Ferraz, Lyno Henrique Gonçalves

A High-Performance Two-Phase Multipath Scheme for Data-Center Networks/Lyno Henrique Gonçalves Ferraz. – Rio de Janeiro: UFRJ/COPPE, 2015.

XVI, 63 p.: il.; 29, 7cm.

Orientador: Otto Carlos Muniz Bandeira Duarte

Tese (doutorado) – UFRJ/COPPE/Programa de Engenharia Elétrica, 2015.

Bibliography: p. 59 – 63.

1. Virtualização de Redes. 2. Redes Definidas por Software. 3. Computação em Nuvens. 4. Multicaminhos. 5. Centro de Dados. I. Duarte, Otto Carlos Muniz Bandeira. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia Elétrica. III. Título.

À minha família.

Agradecimentos

Agradeço aos meus pais e toda minha família pelo constante apoio e eterno carinho. Agradeço à minha namorada que está sempre ao meu lado, por todo seu carinho e paciência.

Ao professor Otto, meu orientador, e aos professores Antonio Marinho Pilla Barcellos, Artur Ziviani, Igor Monteiro Moraes, Miguel Elias Mitre Campista e Pedro Braconnot Velloso pela participação da banca examinadora. Agradeço também aos professores Guy e Luís pelos diversos conselhos durante todo o trabalho. Agradeço também ao Rafael pelos diversos conselhos e por ajudar a organizar o trabalho.

A todos os amigos do GTA, em especial Diogo, Martin e João, pelos conselhos e pela grande ajuda.

A todos os amigos que sempre estiveram do meu lado.

Aos funcionários do Programa de Engenharia Elétrica da COPPE/UFRJ, Daniele, Maurício e Rosa e pela presteza no atendimento na secretaria do Programa.

A todos que contribuíram direta ou indiretamente para a minha formação.

Por fim, agradeço a FINEP, CNPq, CAPES, FAPERJ e UOL pelo financiamento deste trabalho.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Doutor em Ciências (D.Sc.)

UM ESQUEMA DE ENCAMINHAMENTO MULTICAMINHOS DE DUAS FASES COM ALTO DESEMPENHO PARA REDES DE CENTROS DE DADOS

Lyno Henrique Gonçalves Ferraz

Novembro/2015

Orientador: Otto Carlos Muniz Bandeira Duarte

Programa: Engenharia Elétrica

Encaminhamento multicaminhos é usado em centros de dados para melhorar o desempenho de rede ao aproveitar as topologias redundantes. Entretanto, as propostas de encaminhamento multicaminhos existentes ou requerem modificações na pilha de protocolos de inquilinos de computação em nuvem, praticável somente em nuvem privativas, ou não distribuem adequadamente os fluxos na rede. Esta tese propõe um esquema de encaminhamento que não requer modificações na pilha de protocolos dos inquilinos e balanceia a carga de rede eficientemente. O proposto esquema de encaminhamento Multicaminhos de Duas Fases (*Two-Phase Multipath* – TPM) é composto de uma fase inteligente de configuração de multicaminhos descobre caminhos disjuntos ótimos e uma fase rápida de seleção de caminhos em tempo real que aumenta a vazão de rede. Um gerente logicamente centralizado utiliza algoritmos genéticos para criar e posteriormente instalar os caminhos na fase fora de tempo de execução, e controladores locais realizam a escolha de caminhos baseados na utilização de rede em tempo real. O esquema de encaminhamento Multicaminhos de Duas Fases (TPM) é analisado em cenários diferentes de localização de base de dados de utilização de rede, comparado com protocolos similares e apresenta o melhor desempenho, com uma redução do tempo para completar o fluxo 50% menor que *equal cost multipath* e 4.6x menor que *spanning tree protocol.*

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Doctor of Science (D.Sc.)

A HIGH-PERFORMANCE TWO-PHASE MULTIPATH SCHEME FOR DATA-CENTER NETWORKS

Lyno Henrique Gonçalves Ferraz

November/2015

Advisor: Otto Carlos Muniz Bandeira Duarte

Department: Electrical Engineering

Multipath forwarding has been recently proposed to improve network utilization in data centers by leveraging its redundant topological design. However, current multipath proposals either require modifications to the tenants' network stack, being feasible only in private clouds, or do not evenly distribute flows over the network. In this thesis, we propose a novel multipath forwarding scheme that does not require modifications to the tenants' network stack and efficiently balances the network load. Our Two-Phase Multipath (TPM) forwarding scheme is composed of a smart offline configuration phase that discover optimal disjoint paths and a fast online path selection phase that improves flow throughput at run time. A logically centralized manager uses a genetic algorithm to generate and install paths during the offline configuration, and a local controller performs the online multipath selection based on network usage. We analyze our Two-Phase Multipath (TPM) scheme for different workloads and topologies under several scenarios of usage database location, and show that it yields a flow completion time reduction of more than 50% over equal cost multipath and 4.6x over spanning tree protocol.

Contents

List of Figures

List of Tables

List of Abbreviations

ACK	Acknowledge, p. 5
ARP	Address Resolution Protocol, p. 5
CAGR	Compound Annual Growth Rate, p. 3
CONGA	CONGestion-Aware balancing mechanism, p. 5
COTS	Commercial Off-The-Shelf, p. 14
D3	Deadline-Driven Delivery, p. 5
DCTCP	Data-Center Transmission Control Protocol, p. 5
DOVE	Distributed Overlay Virtual Ethernet, p. 14
ECMP	Equal Cost MultiPath, p. 3
ECN	Explicit Congestion Notification, p. 5
FCT	Flow Completion Time, p. 3
GLOBAL	Single database for the data-center, p. 33
HULL	High-bandwidth Ultra-Low Latency, p. 5
IaaS	Infrastructure-as-a-Service, p. 3
LAN	Local Area Network, p. 3
LOCAL	Per rack database, p. 33
LUL	Least-used-links, p. 33
LUP	Least-used-path, p. 33
LUT	Least-used-tree, p. 33
MAC	Media Access Control, p. 14

MPTCP	MultiPath Transmission Control Protocol, p. 5
NO-END	Never decrement cost, p. 33
NVGRE	Network Virtualization using Generic Routing Encapsulation, p. 14
OS	Operational System, p. 5
PCP	Priority Code Point, p. 14
PM	Periodic monitoring to detect flow end, p. 33
RND-MIG	Small in random path, migrate Long using LUL, p. 33
SDN	Software-Defined Networking, p. 5
SFE	Scheduled fixed end when flow arrives, p. 33
SING-NIG	Small in single path, migrate long using LUL, p. 33
SNMP	Simple Network Management Protocol, p. 14
SPAIN	Smart Path Assignment In Networks, p. 5
STP	Spanning Tree Protocol, p. 3
TCP	Transmission Control Protocol, p. 5
TPM	Two-Phase Multipath, p. 3
TRILL	Transparent Interconnection of Lots of Links, p. 3
ToR	Top of Rack, p. 3
UDP	User Datagram Protocol, p. 14
VLAN	Virtual LAN, p. 3
VLB	Valiant Load Balancing, p. 3
VM	Virtual Machine, p. 5
VXLAN	Virtual eXtensible LAN, p. 14

Chapter 1

Introduction

In cloud computing, data centers share their infrastructure with several tenants having distinct application requirements [2, 3]. This application diversity within data centers leads to multiple challenges for network design in terms of volume, predictability, utilization, and security [4, 5]. The volume of traffic in data-center in growing, and comprises the majority of traffic. According to forecasts [1], while IP wide area network traffic will move 2 ZB in 2019, the data-center related traffic will move 10.4 ZB, as Figure 1.1 depicts. The data-center traffic Compound Annual Growth Rate (CAGR) is 25%, while the cloud data-center CAGR is 33% against 5% of traditional data-centers.

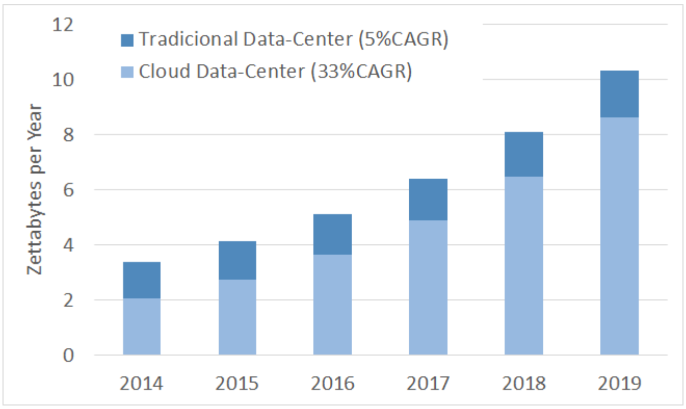


Figure 1.1: Traditional data-center and cloud data-center traffic growth forecast from 2014 to 2019. Overall, the data-center Compound Annual Growth Rate (CAGR) is 25%, while the cloud data-center CAGR is 33% resulting in 10.4 ZB in 2019. Adapted from Cisco Global Cloud Index, 2014-2019 [1].

First, traffic between Top-of-Rack (ToR) switches is currently estimated to be 4x higher than incoming/outgoing traffic [1, 6], as Figure 1.2 shows. This high traffic volume requires specific network topologies for data centers in order to guarantee full bisection bandwidth and to provide fault tolerance [6–9].

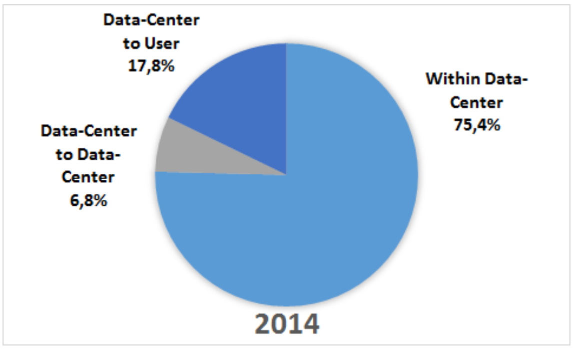


Figure 1.2: Division of data-center traffic. Internal data-center traffic is up to 4x higher than incoming/outgoing traffic. Adapted from Cisco Global Cloud Index, 2014-2019 [1].

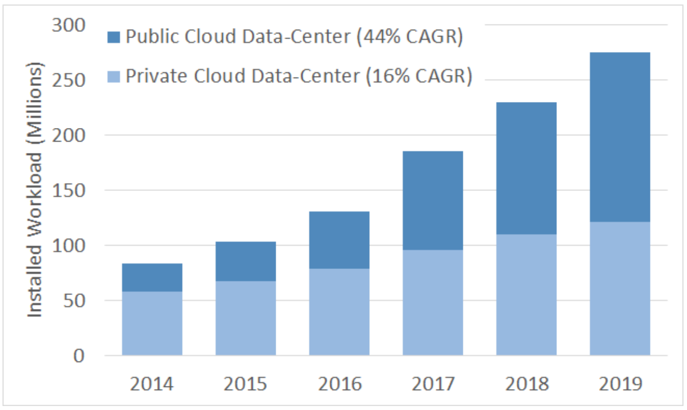


Figure 1.3: Public and private cloud workload forecast. In 2014, 30% of installed workloads are in public cloud data-center, but as it has a Compound Annual Growth Rate (CAGR) of 44%, it surpasses the private cloud workload in 2018. Adapted from Cisco Global Cloud Index, 2014-2019 [1].

Second, the composition data-center workload is migrating from private cloud data-center to public cloud, as Figure 1.3 shows. In 2014, 30% of installed workloads are in public cloud data-center, but it surpasses the private cloud workload in 2018. Hence, the cloud providers have less control of the traffic inside the date-center. Further, the random arrival and departure of virtual machines from multiple tenants result in an unpredictable traffic workload, making it hard to provide manual solutions for traffic management. Therefore, automated solutions that respond quickly to changes are required to efficiently allocate the network resources. Finally, in order to avoid forwarding loops, legacy network protocols, such as the Spanning Tree Protocol (STP) [10], can be employed to disable certain network links. This ensures that every pair of ToR switches communicates over a single path and that the network is loop-free; however, it also restricts the switches from taking advantage of the multiple available paths in data center topologies.

Whereas volume and predictability are inherent to the traffic nature of the application, network utilization can be significantly improved by multipath forwarding. The idea is to split traffic at flow-level granularity among different paths in order to fully utilize the available capacity. Although promising, most approaches rely on heavy modifications to the network stack of end hosts, ranging from explicit congestion notification (ECN) [11, 12] to multipath congestion control [13]. These modifications are not an issue on private clouds, whose sole purpose is to provide services within a single domain. However, in public infrastructure-as-a-service (IaaS) clouds, in which tenants rent virtual machines and have complete control of their network stacks [14], these solutions are not recommended since they interfere with tenants autonomy. Therefore, solutions that only enhance the network infrastructure while not touching the end hosts are required.

A well-known approach for deploying multipath forwarding without modifying the end host is Equal Cost MultiPath (ECMP), commonly adopted in datacenters [6, 7, 15, 16] as well as in the recent standard called Transparent Interconnection of Lots of Links (TRILL) [17]. Network switches supporting ECMP find multiple paths with the same cost and apply a hash function to certain fields of the packet header in order to find the next hop. ECMP is expected to evenly distribute the flows among the multiple paths and thus reduce network congestion. Nevertheless, since hash-based path selection does not keep track of path utilization, ECMP commonly causes load imbalances when long-lived flows are present on selected paths [18]. Similarly, in Valiant Load-Balancing (VLB), the flow source sends traffic to a random intermediate node which, in turn, forwards it to the destination. As ECMP, this also achieves uniform flow distribution on paths; however, due to the stateless selection of the intermediate node, VLB suffers from the same problems as ECMP.

With these issues in mind, we propose a Two-Phase Multipath (TPM) forwarding scheme with a number of key properties:

- **No modifications at end hosts:** TPM is an in-network load balancing scheme that does not require modifications to the end hosts. This is required in multitenant clouds where the provider does not have any access whatsoever to the tenants' network stack.

- **No modifications in hardware:** TPM increases the performance of the data center with no hardware modifications and avoids changes to the infrastructure fabric. In addition, it also requires only a handful of configurable features to keep the implementation cost low.

- **Robust to asymmetry and topology:** TPM handles path asymmetry due to link failures and topology design. It can also be deployed in arbitrary topologies and covers the entire spectrum of data center topologies.

- **Incrementally deployable:** TPM can be deployed in a part of the data center, and work with other segments of the data center seamlessly.

The proposed TPM multipath scheme separates the forwarding functionality into two distinct phases, namely, multipath configuration and multipath selection. Multipath configuration is the offline phase that computes the best possible paths and configures them in the switches. It creates several virtual local area network (VLAN) trees interconnecting all ToR switches, and therefore the path selection can be performed by simply tagging packets with the proper VLAN ID at the outgoing ToR switch. The VLANs trees reutilize a VLAN ID to several paths, which provides multiple paths with a reduced number of utilized VLANs ID to comply with the VLAN ID hard limit. To find these trees, the multipath configuration phase uses network topology information to reduce path lengths and link diversity. In particular, we propose and formulate a genetic algorithm to find an optimal set of trees with disjoint links. The multipath configuration phase remain active during the operation of the data-center, and reconfigures the trees in the switches when needed without shutting down the infrastructure. Multipath selection is the online phase that chooses the best path for a new flow. The selection is based on path utilization database in order to select the least-used path.

To evaluate TPM, we develop a discrete-event simulator at flow-level granularity to model the data center. We test scenarios inspired in realistic traffic workloads [19], and data-center topologies. We also used different configuration and parameters to evaluate the proposal performance with different database designs, which change the information stored, update rate, and thus the requirements. The results show that TPM always performs better than the traditional forwarding schemes. TPM successfully reduces the congestion and improves the maximum throughput time, with reductions in the Flow Completion Time (FCT) by more than 50% compared to uniform path selection schemes such as ECMP, and 4.6x compared to the single tree scheme STP.

The remainder of this thesis is structured in six chapters. We cover some related works in Chapter 2. Chapter 3 presents an overview of TPM and our design choices. Chapter 4 describes the offline multipath configuration phase and Chapter 5 presents the online multipath selection phase. We present the developed simulator and the obtained results in different scenarios and topologies in Chapter 6. Finally, we present the final remarks and discuss future work in Chapter 7.

Chapter 2

Related Work

Traditional data centers use dedicated servers to run specific applications, which result in inefficient resources utilization due to demand variations and consequently the unused resources [20]. The growth of cloud computing and virtualization technologies enables the aggregation of several services and applications in a single data center. The services and applications are virtual networks that share the data center resources [14]. The aggregation increases resource usage and reduces operation and maintenance costs. Besides, the virtualized cloud data centers host several tenants, each has its own applications and services with different requirements. Therefore, the diversity of applications sharing the infrastructure cause in a variety of traffic patterns in data center network [21].

The data center topology is built with the goal of providing high computation capacity with several servers interconnected. To connect the computational resources, the network topology employs several paths between servers [22]. Thus, an application is distributed across servers, which results in a intra data center traffic four times the outgoing traffic [6].

One of the technologies that allows the cloud computing is server virtualization. Server virtualization allows efficient data-center resource utilization, because several virtual machines share the infrastructure, thus avoiding idleness [14, 23, 24]. Nevertheless, cloud computing customers reported "noisy neighbors" problems regarding the other tenants sharing the resources, which result in unpredictable performance when collocated tenants try to grab resources. While the hypervisor have means to ensure the memory disk, and CPU performance to virtual machines, network performance isolation still an issue [25].

2.1 Virtual Machine Placement for Load-Balancing

One important feature provided by virtual machines is the migration, which transfers on virtual machine from one server to another. This feature allows data-center optimal resource utilization, leveraging the virtual machine CPU, memory and network demands with the available resources [23, 26, 27]. Besides, the infrastructure is prone to network failures that hamper the data-center, and even with few, the network performance degrades significantly by increasing paths size and forcing the path sharing [9]. Then, some proposals use the migration capability of cloud computing to ensure virtual machine communication and availability in failures events, while reducing the bandwidth utilization in data-centers [28].

The optimization of virtual machine (VM) placement ensures efficient resource utilization, while optimization in multipath routing ensures link efficiency usage. Nevertheless, managers often perform these optimizations separately, even though one might affect the other. Jiang *et al.* formulate and propose an algorithm to solve the joint routing and VM placement problem [26]. The joint problem formulation considers tenants or jobs, a set of VMs, to be assigned to the data-center and with a number of links and machines. The VMs of a job have a resource requirement and also communicate with each other with certain load. Thus, the optimization problem tries to allocate these VMs in the servers minimizing the network total load and the operating costs of the physical machine, while respecting the VM resource and traffic load requirement. As VMs constantly enter and leave the data-center, the authors propose an online solution to the problem by a Markov chain approximation, such that jobs interarrival interval and permanence in the system have exponential times resulting in a $M/M/\infty$ queue for the number of jobs in the system. As it is possible to get an approximate value to the objective function for a particular set of jobs, one can get the optimal approximated solution by traversing a time-reversible Markov chain. Therefore, the authors successfully achieve cost efficient routing and VM placement.

The design of network topologies and resource allocation mechanisms account for failures normally by providing multiple paths between servers and bandwidth reservation. Nevertheless, components may fail and shut down for maintenance, and network congestion may cause latency spikes, thus resulting in sections of the data-center becoming unavailable. Bodik *et al.* propose an optimization framework that at the same time it reduces the bandwidth utilization of cloud-based services, it also improves fault tolerance by spreading virtual machines across the data-center servers [28]. Given that both improving fault tolerance and improving bandwidth usage problems are NP-hard, the authors formulate a convex optimization problem and an algorithm to solve it. The optimization problem incentives virtual machine spreading across servers, and also they include a penalty term for machine reallocations that increase bandwidth usage. The algorithm that solves the optimization problem optimizes each objective at a time then combines the solution, and overall, it successfully increases the fault tolerance of services while reducing the bandwidth usage.

Machine virtualization is one of the features of cloud computing IaaS. The presence of virtual machines introduce new challenges on where allocate the virtual machines to ensure the resource requirements, but introduces new opportunities to improve the network usage, reduce congestion and load-balance the traffic. The virtual machine allocation mechanism are independent from the forwarding mechanism. Although our proposed scheme does not provide a virtual machine allocation mechanism built-in, we consider that a complete cloud computing data-center should include one. The virtual machine allocation mechanisms are compatible with our proposed forwarding scheme and may improve even more the overall performance.

2.2 Mice Flows and Elephant Flows Focused Network Improvements

The applications running in the data center are varied, each with distinct traffic patterns [19]. Most of the flows are small and short-lived, the so called mice flows. On the other hand, most of the bytes are transferred in big and long-lived flows, which compose a minority of total flows. The big and long-lived flows are the elephant flows. The mice and elephant flows have distinct requirements and behaviors, which the interaction between them causes packet losses and delays in requests fulfillment that deteriorate the applications performance.

The mice flows are latency-sensitive, because normally are generated by partition/aggregate applications, such as Map/Reduce applications used in web searches, social network content composition, and advertising selection [11]. In this type of application, the aggregator divides a request into sub-requests and delivers them to workers during the partition phase. After the computation of the sub-requests, the workers send the result to the aggregator, which in turn perform the aggregation phase. The flows of this kind of application have strict delay limits so the responses are timely presented to the users. The responses that violates the delay limits are discarded, which degrades the overall quality of the response to users. The main cause of the response delay increase is the flash congestion during the aggregation phase, which corresponds to a many-to-one communication. Hence, the prevention of the issue is mainly the prioritization of delay-sensitive flows and low link usage to avoid traffic congestion losses [29].

Alizadeh *et al.* propose a fine-grained data-center transmission control protocol (DCTCP) to overcome the linear increase and exponential decrease procedure of TCP [11]. In this approach, the switches monitor the port out buffers and when they are full, the switches interpret it as congestion. The switches mark the explicit congestion notification (ECN) field in packets headers when they detect the queue occupancy is greater than a threshold, which is small relatively to the queue size. When the ECN-marked packet arrives at the receiver, the receiver sends ACK packets with the ECN-Echo field enabled. Based on the ACKs received, the sender knows the fraction f_c of packets of the stream encountered "congested" queues. Then, once for every window of data, the sender calculates an estimate of the probability p_c of the next window of data encounter congested queues by $p_c \leftarrow \alpha f_c + (1-\alpha)p_c$, where α is the weight of new samples. Then, the congestion window at the sender is updated to $cwnd \leftarrow cwnd(1-p_c/2)$. This algorithm allows the sender react to as soon as the flow encounters congested queues, before the links are in fact congested. Besides, the congestion window size changes accordingly to the congestion of the network, avoiding the cut-in-half window policy of conventional transmission control protocol (TCP), thus allowing a fine-grained rate control.

DCTCP can effectively reduce the congestion by the fine-grained rate control. Nevertheless, DCTCP still suffers from flash congestion, which instantaneously floods the network with packets. Besides, DCTCP allows the queues to build up, increasing the total latency of the packets. This scenario is inappropriate for ultra-low latency application such as high-frequency trading, high-performance computing, and RAM-Cloud. In this kind of applications, machines interact with each other with operations involving parallel requests to thousands of servers at microseconds scale, and the operation is complete only when all requests are satisfied. Thus, any delay in individual request, degrade the target quality of service. The microsecond scale latency is achievable if queuing delays are reduced to zero, in a conventional large scale data-center. With this in mind, Alizadeh *et al.* extend DCTCP and propose a High-bandwidth Ultra-Low Latency (HULL) architecture to deliver near baseline latency and high throughput [12]. Low and predictable latency essentially requires congestion signaling before congestion occurs. This imposes a tradeoff between total bandwidth usage and latency by creating bandwidth headroom. The authors accomplish that by a phantom queue, which simulates a queue and mark ECN bits based on link utilization instead of queue occupancy. The phantom queue simulates a queue buildup in a configurable speed slower than the line speed. Thus, whenever a packet exits the link, the phantom queue updates a packet counter that allows the calculation of the transmission rate. When the calculated speed is above the configurable speed threshold, the switch marks the packets, which allows the transport protocol to control the transmission rate with fine-granularity while leaving the headroom to ensure the low latency.

Providing low latency is extremely important, but as several applications share the data-center network, not every application have their latency needs accomplished. Wilson *et al.* argue that as applications have different deadlines for their flows, it is possible to schedule in such way that the majority of flows meet their demands [30]. Wilson *et al.* propose Deadline-Driven Delivery (D3) control protocol, which assign flows different transmission rates to accomplish all deadlines as possible. In D3, the applications request a specific rate to fulfill their deadlines. The rate request is carried on a packet header sent to the destination, and the network devices send back the allocated rate on the acknowledgement packet. The applications then use the smallest rate allocated. The network devices use a greedy algorithm to allocate the requested rates to the applications, and as the flows and required rates are always changing, the applications periodically send the rate request. If there is not enough capacity for all request rates, it greedily allocates the rates, and gives a base rate for the remaining flows. If there is spare rate after allocating the requested rates, the router allocates the fair share to all flows. In this way, it reduces the required rates for the next request allowing even more deadlines to be accomplished.

The congestion in data-center networks may increase the flow completion time up to two orders of magnitude, which forms a long tail distribution for the flow of a partition-aggregate application. This long tail means that the partition-aggregate job has to wait for all flows, thus hampering the overall performance of the applications. Zats *et al.* propose a multilayer approach to reduce the long tails in flow completion time (DeTail) [29]. At the link layer, DeTail employs a lossless fabric with priority flow control standard capable Ethernet switches, in which switches send pause frames to previous hop when the ingress queue is occupied to stop the transmission of a given priority, quenching up to the sources. At the network layer, the switches load-balance the traffic using the least-congested equal-cost shortest paths. The switches use the egress queue occupancy as an indicator of congestion, because if the path is congested, the next hop sends pause frames increasing the egress queue occupancy. The multipath and pausing may result in out-of-order packet arrival, thus at the transport layer must be robust to packet reordering. Finally, at the application layer, applications set the corresponding priority of the flows in the socket interface. The applications should also be prepared to experience long quenching time in extreme congestion scenarios. The correlation between the elements of each layer is extremely important, and the whole system is capable of significantly reducing the long tails of the flow completion time.

The proposals in literature that cope with mice flows long delays not only require several infrastructure devices modifications, but also modify end servers applications and protocols to interact with the infrastructure devices, normally to reserve resources. Thus, these proposals are not suited to cloud data centers with multi tenants, since each tenant has own distinct applications and protocols. Besides, even if the protocols are standardized, the interaction between the applications and network devices present security risks due to isolation violations [4].

At the same time the packet losses due to flash congestion should be avoided, other flow requirements should also be fulfilled. Elephant flows transfer large amounts of data, then they demand high throughput but are flexible regarding delays. Hence, these flows should be organized to utilize the maximum available bandwidth of the data center. Besides, the elephant flows use all link capacity, which may cause congestions that affect mice flows. Various proposals use multiple paths provided by data center network topology to avail the links bandwidth capacity. One technique that uses multiple paths to increase overall throughput is the Equal Cost MultiPath (ECMP). The routing protocol calculates multiple minimal paths with the same cost and uses a hash function to spread flows in the different paths. Then, the flows are expected to be randomly and evenly distributed in the multiple paths. Link layer protocols already use ECMP such as Transparent Interconnection of Lots of Links [17] and IEEE 802.1aq Shortest Path Bridging [31]. Valiant Load Balancing [6] is similar to ECMP, but to select the flow path, the flow sending server randomly picks up an intermediate switch to forward the flow to.

The MultiPath TCP (MPTCP) [13] divides a TCP flow into several subflows and send each in a different path, so each subflow has its own congestion control. The MPTCP approach tries to send each subflow at maximum rate available in each path to use all available bandwidth for the flow. Since this approach changes the normal TCP operation, it requires modifications in the guest operational system (OS).

Some multipath forwarding schemes were proposed in the context of Software-Defined Networking (SDN) to manage and distribute data center traffic [32]. Al-Fares *et al.* [15] propose Hedera, a centralized flow scheduling system that uses an OpenFlow controller to gather information and manage switches. Hedera uses ECMP to distribute traffic among different paths and monitors their duration over time. Hedera detects the presence of long-lived flows and periodically runs a simulated annealing algorithm to distribute these flows into different paths to maximize transmission rates. In a similar approach, Curtis *et al.* [33] propose DevoFlow, which devolves the flow management to switches while the controller only keeps track of a few targeted flows. DevoFlow uses local probability distributions to select the next hop for each flow, and also can use centralized algorithms to reschedule flow paths as in Hedera. Nevertheless, the centralized algorithms are too slow to optimize the variable data-center traffic [29]. Our approach assigns the path selection functionality to the local controllers to avoid such constraints.

A few proposals randomly select one of the multiple paths in the data centers to distribute the network traffic [6, 22, 34]. Al-Fares *et al.* [22] design a communication architecture for the Clos fattree network topology. The solution has no end-host modification, but requires moderate modifications in switch forwarding functions. The authors propose an addressing scheme for switches and hosts and a two-level routing table, which splits traffic according to the destination host. Greenberg *et al.* [6] propose VL2, a network architecture that uses a Clos network topology to form a complete bipartite graph between core and aggregation switches. VL2 addressing scheme uses two separate classes of IP addresses, one for the infrastructure topology and another for the tenants' applications. The VL2 architecture uses a directory system to map the tenants' applications addresses to the infrastructure servers addresses, and the source server encapsulates the tenant's application packets with an IP header to the destination server. Besides, for packet forwarding, it uses Valiant Load Balancing (VLB) to distribute the traffic among the different paths. The source encapsulates packets with another IP header to a random intermediate switch and ECMP distributes the flows among the paths to this intermediate switch. To increase entropy of packet headers, the source addresses of the extra headers are the hash value of the inner packet header, which avoids the switches ECMP forwarding use the same path due to the multiple encapsulation.

Mudigonda *et al.* [34] introduces NetLord, a multi-tenant network architecture that encapsulates tenants' Layer-2 packets to provide full address-space virtualization. An agent in physical servers encapsulates and decapsulates packets of tenants' virtualized systems. As VL2, only the infrastructure server addresses are exposed in the network, thus it scales the number of virtual machines. The forwarding mechanism uses a so-called smart path assignment in networks (SPAIN) [35] to distribute traffic among multiple paths. The SPAIN proposal explores the path diversity of data center topologies without any changes in the common of the shelf switches. SPAIN uses an offline greedy algorithm to configure Virtual Local Area Network (VLAN) trees, so that each tree is built based on minimal path size and link reuse. The use of VLAN trees allows an easy multipath setup just by configuring different VLANs in the switches. Further, this approach requires minimal switch features, including VLAN-based Media Access Control (MAC) address learning and storage. SPAIN also runs an online algorithm in servers which test the connectivity of the destination and randomly selects a tree to send a flow. All aforementioned proposals rely on a random distribution of flows among the available paths, which performs poorly in the presence of long-lived (elephant) flows. Our proposed TPM scheme avoids that by employing a path selection heuristic based on link utilization, significantly reducing path selection collision and improving the overall performance.

Alizadeh *et al.* [18] propose a distributed global congestion-aware (CONGA) balancing mechanism. Each source ToR switch encapsulates the tenants' packets using a VXLAN header, and spine switches update a congestion metric field in this header. The destination ToR switch decapsulates the packets, forwards them to the corresponding tenant, and stores the congestion metric of the incoming path. The congestion metric is opportunistically piggybacked in the VXLAN header when the destination ToR switch sends packets back to the source ToR switch. Thus, ToR switches constantly receive the congestion metric for each path it sends traffic, and choose the path which minimizes the congestion metric. Conga utilizes an in-network approach, but it requires a new forwarding engine in switches, which could be costly.

Rojas *et al.* [36] propose All-Path, a routing paradigm that uses a reactive approach to learn the paths in data-center, campus, and enterprise networks. Based on this paradigm, they propose a protocol that learns low-latency paths on-demand based on broadcasted Address Resolution Protocol (ARP) messages from hosts. Switches broadcast ARP request messages and store the port from which the first copy is received. The ARP reply follows the reverse path, allowing switches to reach every destination. This approach thus frequently balances the flows among the low-latency paths. As Conga, All-Path also requires costly modifications on the switches. The proposed TPM scheme can be currently deployed in data centers by only modifying the software in virtual switches at the physical machines at the network edge.

Even though these works contribute to key aspects of multipath forwarding for cloud computing data centers, we claim that to maintain scalability, the network infrastructure should be oblivious to the multipath scheme. Hence, both encapsulation mechanism and mapping system are always required. For this purpose, we use VLAN trees, which enable multipath forwarding using conventional features of commercial off-the-shelf switches. The per-flow management is easily implemented using distributed SDN controllers to manage the virtual switches at the servers.

Our work proposes a multipathing scheme that generates optimized paths with genetic algorithm. The approach suits the cloud data center, so it does not require any tenant modification. The modifications proposed are in infrastructure provider and are maintained at a minimum level. The multipathing is accomplished in two phases: Multipath Configuration and Multipath Selection. In Multipath Configuration phase, the genetic algorithm calculates the multiple paths that can be used by the flows. Then, the multiple paths are configured in the network devices. The Multipath Configuration is an offline phase; it configures the devices when the network is not yet in operation or when there are planned or long term topology changes, such as installation of new devices, maintenance operations, and device failures. In the Multipath Selection phase, the flow path is chosen online. TPM selects paths for all data-center flows, thus both mice and elephant flows, although elephant flows benefit more because the scheme improve their overall transmission rate, their most needed resource. As there is more available bandwidth from thorough path selection, mice flows benefit indirectly by experiencing less contended paths. Besides, TPM employs tweaks to address mice flows specific needs.

Chapter 3

Architecture

The proposed Two-Phase Multipath (TPM) scheme explores the path diversity of the network to load balance flows using an in-network approach, without requiring any modification to the tenants' virtual machines (VMs) protocol stack. As previously explained, this is performed in two phases: The multipath configuration phase and the multipath selection phase.

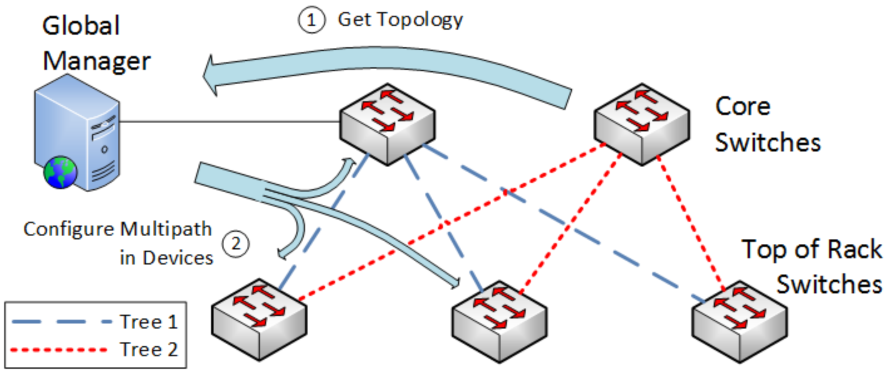


Figure 3.1: Two Phase Multipath Scheme Architecture: Multipath Configuration phase. The global manager 1) gets the topology to calculate multipaths and 2) configures the multipaths in network devices.

TPM requires two types of devices to manage the entire network: a logically centralized global manager responsible for the multipath configuration phase, and local controllers responsible for the multipath selection phase. Figures 3.1 and 3.2 depict the TPM architecture. In essence, the global manager collects network topology information, calculates the available paths, and sends them to the network devices to be used later during online path selection, as in Figure 3.1. The path computation is performed before the network becomes operational. When there is a topology change such as installation of new devices, partial deactivation due to maintenance, or long term failures, the global manager performs the path computation to install the new paths, without the need to deactivate the network. To obtain the topology periodically, the global manager may use either the Simple Network Management Protocol (SNMP) for topology discovery [37] or OpenFlow [38]. The VLAN for each tree can also be configured using either approach, which guarantees that most commercial off-the-shelf (COTS) devices are suitable to be used with TPM.

In order to exploit multiple paths without having to modify the network core, TPM uses VLANs (IEEE 802.1Q). Each VLAN uses a subset of aggregation/core switches to interconnect all ToR switches in a tree topology.

Instead of assigning a VLAN to each path, TPM uses a VLAN for each tree in order to aggregate multiple paths into a single VLAN ID, thus saving precious VLAN ID space (each VLAN ID has only 12 bits). Assuming a data center with n ToR switches, each tree contains $n(n-1)/2$ symmetric paths; our approach is then able to support up to $n(n-1)2^{11}$ different paths between every pair of ToR switches. This increases the number of potential paths by a factor of at least $n(n-1)/2$ when compared to the case of using a VLAN per path. In addition to increasing path availability, VLAN trees do not require a routing protocol, since there is only a single path between any pair of ToR switches in each VLAN.

The trees of each VLAN are not entirely disjoint, and thus each link may belong to multiple trees. During the multipath configuration phase, however, the trees are selected to be as disjoint as possible in order to ensure maximum path availability between any pair of ToR switches. To find these trees, we propose a genetic algorithm presented in Chapter 4.

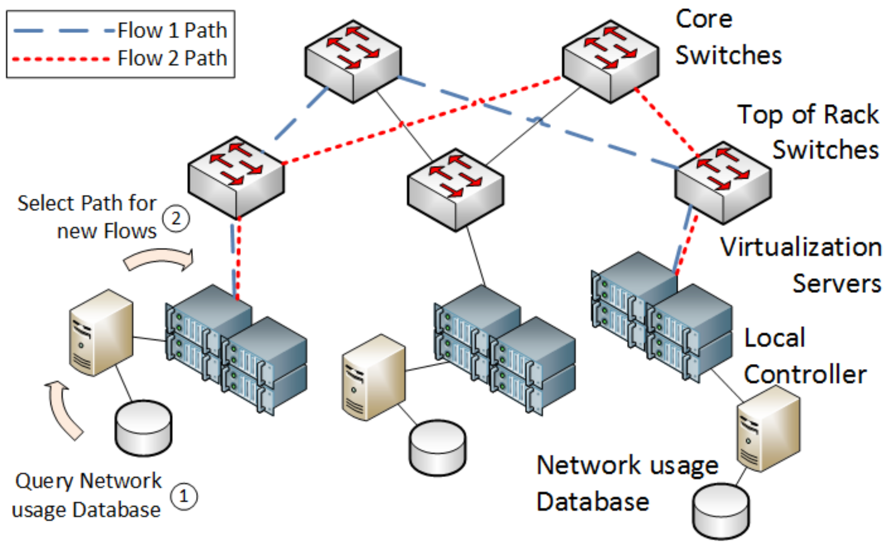


Figure 3.2: Two Phase Multipath Scheme Architecture: Multipath Selection phase. When a new flow arrives at the local controller, it 1) queries the network usage database and then 2) selects the best path for that flow.

We assume that each physical machine in a rack has a virtual switch to share the network device between the virtual machines, such as Open vSwitch [39]. The virtual switches of a rack are connected to the same local controller. During packet forwarding, the virtual switch inserts a VLAN tag into each outgoing packet and also removes the VLAN tag from each incoming packet. Upon arrival of a new outgoing flow, the virtual switch contacts the controller to select an available path for it. The controller then queries a database with network usage information to determine the least congested path for the new flow, as explained later in Chapter 5. Once the path (the corresponding VLAN) is selected, the local controller installs an OpenFlow VLAN tagging rule on the virtual switch to handle future packets of this flow, as in Figure 3.2. Each subsequent packet then receives the assigned VLAN tag and does not require contacting the local controller again.

3.1 Two Phase Multipath Scheme Improvements

TPM scheme allows the optimal path choice to avoid congestion and increase network performance. Nevertheless, the scheme supports tweaks that improve even more the performance, which do not require neither modifications on the virtual machines nor modifications on the infrastructure hardware.

3.1.1 Prioritization of Latency-sensitive Small Flows

There are basically two types of flow in the data center: the bandwidth-hungry large flows and the latency-sensitive small flows. The large flows do not have strict latency requirements, thus the multipath load-balancing provides good performance to transmit the flows. On the other hand, although latency-sensitive small flows do not use much bandwidth, they have strict latency requirements. The violation of the deadlines of these flows results in poor application performance, hence they should be prioritized over the large flows.

Flow prioritization is a common feature of most commercial switches, and the IEEE 802.1Q Priority Code Point (PCP) bits to classify the priority packets in a special queue and send them before the other packets. Additionally, switches may use two more queues with minor priority which they send traffic with a weighted round robin scheduling discipline. Therefore, the global manager activates the flow prioritization based on 802.1Q PCP bits only to prioritize the marked flows. The local controller already sets the virtual switch of the servers to add the 802.1Q header and set the VLAN tags. Thus, the local controller also sets the virtual switches to mark the small flows PCP bits. As the local controller has no knowledge of which are the latency-sensitive small flows a priori, it sets the PCP bits in all new flows, and unsets the PCP bits of the flows that remain active after some time. Alternatively, as the virtual switches use Open Flow, they can use the conventional flow header fields to detect the small flows.

3.1.2 Fine-grained Load-balancing

The main goal of load balancing is to distribute the transmitted data between all available paths, so sending each packet to a different path would be the optimal solution. Nevertheless, each packet experiences different delays and arrive at the destination in different instants, causing transport layer protocols such as Transmission Control Protocol (TCP) to interpret the out-of-order packets as sign of congestion and reduce transmission rates accordingly [40]. Therefore, flows should use a single path to avoid packet reordering or all transport layer of the data center should be modified to resist to packet reordering.

In this scenario, to increase the granularity of the load-balancing the flows should be as small as possible. With that in mind, we use the flowlet concept, which states that full flows are essentially a set of bursts of packets, the flowlets, in such way that there is a gap between each flowlet of a flow [40]. If the gap between flowlets is large enough, each flowlet can use a different path and the packets are received in-order.

As the local controller uses OpenFlow to control the virtual switches of servers, every rule is set with a parameter idle timeout. The idle timeout parameter makes the virtual switch to expire and remove the rule when it does not receive any packets of the corresponding flow. Thus, by setting the idle timeout of flows rules in virtual switches ensures different path selection for each flowlet. Indeed, the parameter value assignment is important, because too small values may result in out-of-order packets and overload the local controller, and too large values may ignore flowlets.

3.1.3 Tenant Isolation and Scalability

Cloud computing data centers host a huge amount of tenants, which have several different requirements. One of the most important requirements is the isolation between each tenant. The traffic of a tenant should be totally oblivious to other tenants. Besides, the isolation mechanism should support a large number of tenants, given the scale of cloud computing data centers. The presence of a large number of virtual machines, can force the infrastructure fabric to learn all MAC addresses of the virtual machines. To address this concern, many proposals create an overlay to isolate the traffic of tenants with VLAN tags [41], User Datagram Protocol (UDP) encapsulation such as Distributed Overlay Virtual Ethernet (DOVE) [42] and Virtual Extensible LAN (VXLAN) [43], or tunneling with Network Virtualization using Generic Routing Encapsulation (NVGRE) [44]. The encapsulation allows the separation of each tenant traffic, avoid sending packets to another tenant' virtual machines. Besides, the encapsulation hides the virtual machines Media Access Control (MAC) addresses to the infrastructure, which has to learn only the physical machine MAC address. Except from VLAN tagging which is used to identify the multiple paths, all of these proposals are compatible with TPM scheme.

3.1.4 Incremental Deployability

Since the VLAN tag insertion/removal is performed at the edge by virtual switches, the ToR, aggregation, and core switches are oblivious to the existence of TPM and only forward packets based on their respective VLAN tag and destination MAC address. This design choice allows TPM to be backward compatible with existing data center infrastructure and reduce its adoption costs. Additionally, TPM can also be incrementally deployed. All devices run the default STP protocol to create a single untagged tree to ensure the interconnection of all ToR switches, and therefore of all physical machines. In order to send packets to devices that do not support TPM, the controller instructs the virtual switches to not insert a VLAN tag into these packets.

Next, in Chapter 4, we describe the offline multipath configuration phase and the proposed genetic algorithm, and in Chapter 5, we present the online multipath selection phase proposed heuristics.

Chapter 4

Offline Multipath Configuration Phase

The multipath configuration phase is responsible for calculating and installing the VLAN trees in the network devices. Each tree interconnects all Top of Rack (ToR) switches, thus all servers, ideally each tree with different paths between the ToR switches. The challenge here is to provide enough path diversity for each pair of ToR switches to improve network utilization. However, it is not clear how many trees (and different paths) our Two-Phase Multipath (TPM) scheme should use in total to achieve this. In addition, it is important that the chosen trees be as disjoint as possible to provide multiple independent paths for each pair of ToR switches.

In this chapter we model the multiple disjoint trees problem, to find the best tree set with two objective functions. One objective function minimizes the size of the tree, to avoid long paths; and the other minimizes the link reutilization by the trees to create disjoint trees and disperse paths. We then propose a genetic algorithm to find the best tree set using two objective functions. One objective function minimizes the tree size in order to avoid long paths; and the other minimizes the link reutilization by the trees to create disjoint trees. The algorithm optimizes both objective functions considering Pareto dominance group and number of individuals in the population. The Pareto dominance ensures that the algorithm does not favor one objective function or the other, because it considers that two individuals are equivalent if each is better than the one in one of the objective functions. To calculate the trees, the global manager first acquires the network topology, runs the proposed genetic algorithm, and then installs the VLAN trees in the network devices. These operations are first performed offline, and at each long-term modification of the network, such as the installation of new devices. Nevertheless, once in operation, the network does not need to shut down to install the new or modified VLAN trees. Link failures do not trigger the calculation and installation of new trees; instead, they are detected by local controllers that simply blacklist the affected paths. The paths are periodically probed to verify their availability to remove them from the blacklist.

We first formulate the multiple disjoint trees problems and then we propose a genetic algorithm to solve the problem as optimal as possible.

4.1 Mathematical Formulation of Multiple Disjoint Steiner Trees Problem

We derive the model from the Steiner Tree Problem, which aims at finding the minimum weight tree that interconnects a set of terminal nodes. Given the undirected network graph $G = (V, E)$, an edge cost c_e defined on $e \in E$, we want to interconnect $T \subseteq V$ terminal nodes via multiple disjoint trees $I \subseteq E$ that form a set P, such that $I_k \in P, k \in [1 \dots n]$, whose nodes of I are in $S \subseteq V$. The cardinality of the tree set P is the maximum number of trees that is possible to configure in the network, thus the maximum number of VLANs ($n = |P| \leq 2^{12}$). We fix $|P| = 2^{12}$ and let the number of valid trees ($I_k \neq \emptyset, k \in [1 \dots n]$) be an objective of the formulation determined by the variable x_e^I to indicate whether tree I contains edge e, i.e.,

$$x_e^I = \begin{cases} 1, & \text{if } e \in I \\ 0, & \text{otherwise.} \end{cases} \tag{4.1}$$

We also define the function z_i^S to indicate whether node i is in tree S i.e.,

$$z_i^S = \begin{cases} 1, & \text{if } i \in S \\ 0, & \text{otherwise.} \end{cases} \tag{4.2}$$

Note that if the tree node set is not empty ($S \neq \emptyset$), then $S \supseteq T$ and $z_i^S = 1, i \in T$[1]. Our objective is minimize tree sizes and create disjoint trees, thus we define two objective functions of the edge pertinence matrix $\mathbf{x}_{|P| \times |E|}$, $F_1(\mathbf{x})$ and $F_2(\mathbf{x})$ as

$$F_1(\mathbf{x}) = \sum_{I \in P} \sum_{e \in I} c_e x_e^I \tag{4.3}$$

and

$$F_2(\mathbf{x}) = \sum_{I \in P} \sum_{e \in I} \frac{x_e^I}{\sum_{I' \in P} x_e^{I'}}. \tag{4.4}$$

Equation 4.3 is the sum of all tree edge costs. The tree edge costs should be minimal.

[1]This definition is slightly different than the traditional node variable Steiner Tree Problem formulation, which defines also a node cost w_i and variable z_i on $i \in V \setminus T$, which indicates the node pertinence in the tree. The formulation assumes that as terminal nodes are already in the tree, they are not considered.

Note that minimizing Equation 4.3 also reduces the number of valid trees ($I \neq \emptyset, I \in P$). The $\frac{x_e^I}{\sum_{I' \in P} x_e^{I'}}$ part of Equation 4.4 represents the utilization fraction of edge $e \in E$ by all trees. Each tree should use different and least used edges to maximize the fraction. Equation 4.4 expresses the sum of all trees and edges utilization, and should be maximized. Note that Equation 4.4 maximizes when the number of tree with different edges, increasing the number of valid trees ($I \neq \emptyset, I \in P$).

We define $E(X) = e = (a, b) \in E : a, b \in X \subseteq V$ the edge set with both end nodes in X. We next formulate the problem based on the node variable Steiner Tree Problem subtour elimination.

$$\min F_1(\mathbf{x}) - \alpha F_2(\mathbf{x}) \tag{4.5}$$

s.t.

$$\sum_{e \in E} x_e^{I_k} = \sum_{i \in S_k} z_i^{S_k} - 1 \qquad S_k \neq \emptyset, k \in [1 \dots n] \tag{4.6a}$$

$$\sum_{e \in E(X)} x_e^{I_k} \leq \sum_{i \in X} z_i^{S_k} - 1 \qquad X \subseteq V, X \cap S_k \neq \emptyset, k \in [1 \dots n] \tag{4.6b}$$

$$\sum_{e \in E(X)} x_e^{I_k} \leq \sum_{i \in X \setminus v} z_i^{S_k} \qquad v \in X \subseteq S_k, k \in [1 \dots n] \tag{4.6c}$$

$$\sum_{I \in P} \sum_{e \in E} x_e^{I_k} > 0 \tag{4.6d}$$

$$x_e^{I_k} \geq 0 \qquad \text{integer}, e \in E, k \in [1 \dots n] \tag{4.6e}$$

$$z_i^{S_k} \geq 0 \qquad \text{integer}, i \in V, k \in [1 \dots n] \tag{4.6f}$$

The Objective Function 4.5 is composed by the two Objective Function 4.3 and 4.4. As we want to maximize 4.4, we multiply it by -1. Parameter α indicates the importance of Objective Function 4.4 when compared to 4.3.

The restriction 4.6a limits the tree edge number, and restrictions 4.6b and 4.6c ensures the trees are connected and there are no loops. The restriction 4.6d guarantees that there is at least one tree. Restrictions 4.6e and 4.6f define the variables.

We now show that the multiple disjoint Steiner trees problem is NP-hard. For the proof, we consider a restricted version of the problem where the tree set size is one $|P| = 1$ and the second Objective Function $F_2(\mathbf{x})$ is not considered with $\alpha = 0$. We show that the restricted problem is NP-hard, thus the generalized version is also NP-hard.

Definition 1. *Given a graph $G = (V, E)$ and a set of edge costs c_e, a tree $|A| = |N| - 1, A \subseteq E, N \subseteq V$, we define the* tree cost *as the sum of the costs of its edges.*

Problem 1. Steiner Tree: *Given a graph $G = (V, E)$, a set of terminal nodes $T \subseteq V$, and a set of edge costs $c_e, e \in E$, find a tree with minimum* tree cost *that* *spans all terminal nodes T.*

Theorem 1. *Problem 1 is NP-hard. Proof Outline.* The Problem 1, The minimum Steiner tree Problem is known as NP-hard [45]. There are several definitions for the Steiner Tree problem including planar graphs, bipartite graphs and grid graphs, and it results in NP-hard. There several linear programming formulations, and the subtour elimination formulation is the same as described in this section setting $\alpha = 0$.

The Definition 1 can be easily extended to a set of trees.

Definition 2. *Given a set of trees $P = I_1 \dots I_n$, the* tree set cost *is the sum of each tree cost.*

As the tree set cost is the sum of all trees cost, the fewer members produce smaller, thus better, tree set cost. Other goal is to have the multiple trees share the fewest edges as possible, thus we define the edge utilization:

Definition 3. *The* edge utilization *is the inverse of number of times an edge is used by all trees.*

Thus, the fewer the edge is used, the higher the edge utilization. Next, we define the tree edges utilization:

Definition 4. *The* tree edges utilization *is the sum of the edges utilization of all its edges.*

The tree edges utilization represents how disjoint are the edges of a tree. Considering a set of trees:

Definition 5. *The* tree set edges utilization *is the sum of all tree edges utilization.*

As the tree set edges utilization sums the edge utilization for all trees, it results in the total number of edges that the tree set uses. With all definitions, we can now evolve from the reduced Steiner tree problem to a more general.

Problem 2. Multiple Disjoint Steiner Trees: *Given a graph $G = (V, E)$, a set of terminal nodes $T \subseteq V$, and a set of edge costs $c_e, e \in E$, find a minimum cost tree set that spans all terminal nodes T, while having the higher tree set edges utilization as possible.*

The multiple disjoint Steiner trees problem tries not only to find the best way to connect the terminal nodes in a single tree, but also tries to find the fewest number of trees using the maximum different edges. The number of trees is part of the problem, and at the same time that the higher number of trees possibly uses more edges, it also unnecessarily increases the costs. Next, we restrict the number of trees to two, and try to find the best pair of trees that interconnects the terminal nodes.

Problem 3. Two Disjoint Steiner Trees: *Given a graph $G = (V, E)$, a set of terminal nodes $T \subseteq V$, and a set of edge costs $c_e, e \in E$, find two minimum cost trees that spans span all terminal nodes T, while having the higher tree set edges* *utilization as possible.*

Theorem 2. *Problem 3 is NP-hard. Proof.* This problem is a generalized version of Problem 1, which we introduce another non-linear term in the objective of Problem 1. The Problem 3 is easily reduced to Problem 1 by reducing the number of trees to one and setting $\alpha = 0$. As we prove Problem 1 is NP-hard by Theorem 1, the Two Disjoint Steiner Trees problem is NP-hard as well.

Finally, by induction we can conclude that:

Theorem 3. *The Problem 2, the Multiple Disjoint Steiner Trees problem is NP-hard. Proof.* This problem is a generalized version of Problem 3, which we prove is NP-hard by Theorem 2. Thus, the multiple disjoint Steiner trees problem is NP-hard as well. Thus, we should design algorithms and heuristics to solve the multiple disjoint trees problem.

4.2 Proposed Genetic Algorithm for Tree Creation

Genetic algorithms are a stochastic optimal search technique inspired by the natural selection process during biological evolution [46, 47]. The GA is known to present good solutions to large complex and uncomprehended search spaces, when the knowledge of the problem is insufficient to shorten the search space, when there is no available mathematical analysis and traditional methods fail. The GA also works well with restriction without the need to use extra objective functions to model the restrictions, and GA has several good approaches to integrate several objective functions. As the calculus trees set is performed with no strict time requirements in large interval periodic events, GA is an excellent choice to solve the multiple disjoint trees problem.

The GA methodology consists of (i) modeling each feasible solution (i.e., a tree in our case) as an individual, (ii) selecting an initial random population, (iii) calculating the phenotype of each individual, (iv) selecting parents according to their phenotype, (v) recombining the selected parents to form new children, (vi) mutating individuals to form new ones, (vii) selecting the best children to survive to the next generation, and (viii) go to step (iii) and repeat until a good enough solution is found [48]. The selection of parents and children forces the genetic algorithm to prioritize the best solutions. The recombination in step (v) creates new children taking the qualities of the parents, and the mutation in step (vi) drastically changes the individuals to avoid local optima. Next, we describe each of these steps in detail and how they apply to our case.

(i) Individual model: In our application, each individual is a tree interconnecting all ToR switches. More specifically, given the network graph $G = (V, E)$, where V is the set of switches of the network and E is the set of edges, we define the population as the set $P = \{I_1, I_2, \ldots, I_n\}$, where each individual $I_k \subseteq E$ is a subset of edges interconnecting all ToR switches in a tree topology. We define the genotype of an individual I_k as an ordered vector $\mathbf{s}_k = \begin{bmatrix} s_1^k & s_2^k & \cdots & s_m^k \end{bmatrix}$ containing the switches that belong to the tree in I_k, with m the size of the tree. The order of the switch vector $\mathbf{s}_k$ is important because it is used to find the tree that $\mathbf{s}_k$ represents, as showed next. If the superscript index k can be understood from context, we drop it and represent the switch vector simply as $\mathbf{s}_k = \begin{bmatrix} s_1 & s_2 & \cdots & s_m \end{bmatrix}$ for ease of presentation.

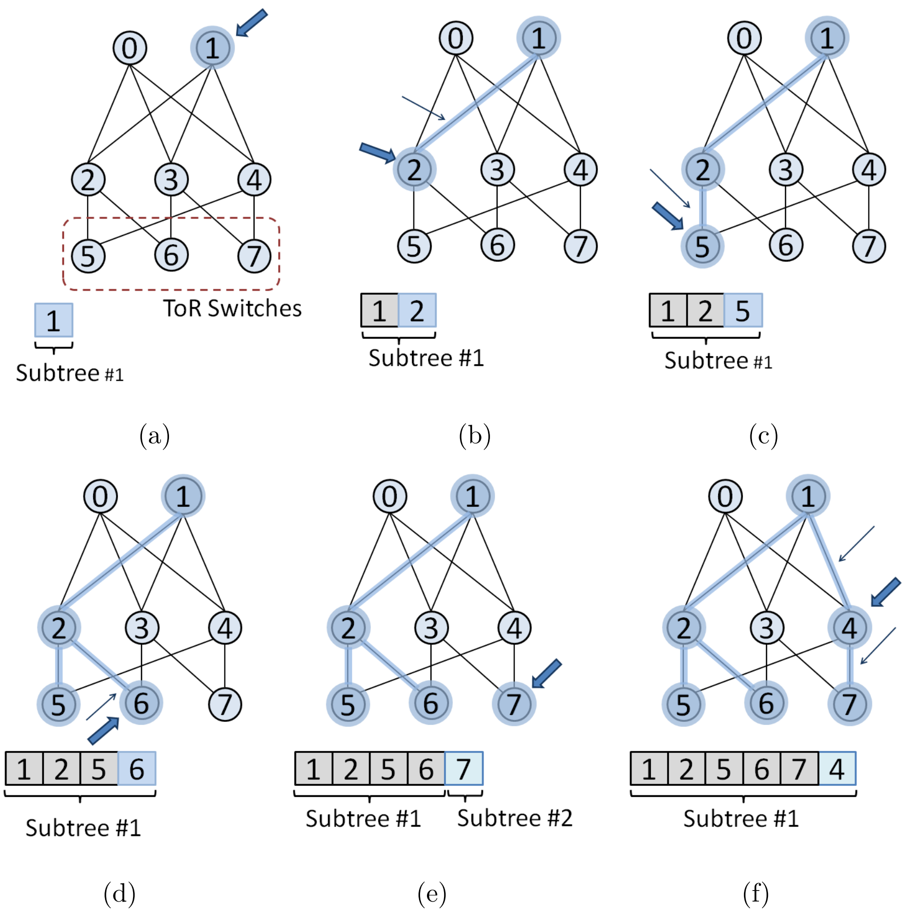


Figure 4.1: The execution of the proposed tree creation procedure to connect all ToR switches. Highlighted nodes and edges are part of the tree. The switch vector $\mathbf{s}$ is showed below the graph. (a) The procedure first picks a random switch forming the first subtree, i.e., $\mathbf{s} = [s_1]$. (b)–(d) A new random switch is selected and, if it has a link to any switch in $\mathbf{s}$, then this link and switch are added to the tree, resulting in $\mathbf{s} = [s_1\ s_2\ s_5\ s_6]$. (e) If the selected switch has no link to other switches in $\mathbf{s}$, it forms a new subtree, but it is still added to the switch vector. (f) If the selected switch has links with switches in different subtrees, it connects to all of them to form a larger tree, resulting in $\mathbf{s} = [s_1\ s_2\ s_5\ s_6\ s_7\ s_4]$. The procedure in (b)–(f) is repeated until all ToR switches are connected in a single tree.

(ii) Initial population: In order to start the genetic algorithm, we need an initial population P composed of a few trees. Our general strategy to form a tree is starting from a random switch and adding one additional switch at a time. Figure 4.1 depicts the proposed tree creation procedure. The procedure starts with a random switch, which forms the first subtree, showed in Figure 4.1(a). Then, we select another switch at random and, if it has a link to any of the already picked switches, that link is added to the tree and the switch is added to the genotype. Figures 4.1(b)–4.1(d) show this process, after which we have $\mathbf{s} = \begin{bmatrix} s_1 & s_2 & s_5 & s_6 \end{bmatrix}$. If the selected switch has no link to any of the switches in $\mathbf{s}$, it forms a new subtree, but it is still added to the genotype, as showed in Figure 4.1(e) with $\mathbf{s} = \begin{bmatrix} s_1 & s_2 & s_5 & s_6 & s_7 \end{bmatrix}$. If the chosen switch has links to more than one switch in $\mathbf{s}$, then there are two possible cases. First, if they are in different subtrees, the new switch connects to all of them and forms a single subtree, as showed in Figure 4.1(f). Second, if the new switch has links to more than one switch in the same subtree, it connects to the first switch that appears in $\mathbf{s}$. This is showed in Figure 4.1(f); s_4 connects to s_1 instead of s_5 because it is the first switch in $\mathbf{s}$ to which s_4 has a link. This procedure is repeated until all ToR switches are connected in a single tree.

(iii) The individual phenotype: While the genotype $\mathbf{s}$ of individual I provides a unique representation of its tree, its phenotype provides a way to quantitatively compare two individuals. Our goal is to provide a higher phenotype value to an individual I with smaller tree sizes and higher link diversity. With this goal in mind, we first define the function $x(e, I)$ to indicate whether individual I contains edge e, i.e.,

$$x(e, I) = \begin{cases} 1, & \text{if } e \in I \\ 0, & \text{otherwise.} \end{cases} \tag{4.7}$$

If we define the individual objective functions

$$f_1(I) = -|I| \tag{4.8}$$

and

$$f_2(I) = \sum_{e \in I} \frac{1}{\sum_{I' \in P} x(e, I')}, \tag{4.9}$$

then, the phenotype of an individual I is defined as the two-dimensional vector $\mathbf{f}(I) = \begin{bmatrix} f_1(I) & f_2(I) \end{bmatrix}$. Function $f_1(I)$ in Equation 4.8 provides the negative number of edges in the tree of individual I, and it has a higher value for smaller tree sizes. Function $f_2(I)$ in Equation 4.9 provides the sum of the utilizations of each edge e by I, considering all individuals in the population P. It has a higher value if I uses links that are not used by other individuals in P.

As both characteristics are important, we consider the Pareto dominance to compare individuals. Thus, an individual I_j is better than other individual I_k, noted as $\mathbf{f}(I_j) > \mathbf{f}(I_k)$, if $f_i(I_j) \geq f_i(I_k)$, for $i \in \{1, 2\}$ and if $\exists i$ such that $f_i(I_j) > f_i(I_k)$.

(iv) Selection of parents: In order to create the next generation, the first step is to select certain individuals to be parents. Each individual in the population P is then assigned a selection probability based on its phenotype value, such that individuals with a higher phenotype have a higher selection probability. Each pair of individuals sampled from P becomes the parents of two new descendants. The number of sampled pairs is therefore $|P|/2$. The same individual can be sampled multiple times and participate in different pairs. When all parents are chosen, the genetic algorithm begins the recombination of parents, as showed in step (v).

In order to calculate the selection probability of each individual, we first find the best individual I_b in our population P, i.e., $I_b = \arg\max_{I \in P} F(I)$. Then, for each individual $I \in P$, we compute its Euclidean distance to I_b as $d(I) = \|\mathbf{f}(I) - \mathbf{f}(I_b)\|$. We define the worst individual $I_w = \arg\max_{I \in P} d(I)$ as the individual that is the furthest away from I_b. Since we want nodes closer to I_b to have a higher chance of being sampled, we define the non-normalized selection probability as $\tilde{p}(I) = (1 + \epsilon)d(I_w) - d(I)$, where $\epsilon > 0$ is a small constant to ensure that $\tilde{p}(I_w) > 0$. Finally, we use $\tilde{p}(I)$ to obtain the normalized selection probability $p(I)$ as

$$p(I) = \frac{\tilde{p}(I)}{\sum_{I' \in P} \tilde{p}(I')}. \tag{4.10}$$

(v) Recombination of parents: Each pair of parents sampled in step (iv) must be recombined to pass their genotype on to two descendants. Let $\mathbf{s}_a = \begin{bmatrix} a_1 & a_2 & \cdots & a_m \end{bmatrix}$ and $\mathbf{s}_b = \begin{bmatrix} b_1 & b_2 & \cdots & b_n \end{bmatrix}$ be the genotypes of two parents, I_a and I_b, respectively. The recombination procedure starts by selecting a random position $r \in [1, \min(|I_a|, |I_b|)]$ of the parents' genotypes, and splitting them into both a radical and a suffix. Figure 4.2(a) shows this case for $r = 1$. The suffixes of the parents are then exchanged, forming two new individuals I_u and I_v, with genotypes $\mathbf{s}_u = \begin{bmatrix} a_1 & \cdots & a_r & b_{r+1} & \cdots & b_n \end{bmatrix}$ and $\mathbf{s}_v = \begin{bmatrix} b_1 & \cdots & b_r & a_{r+1} & \cdots & a_m \end{bmatrix}$. These genotypes, however, do not necessarily form a tree and thus we run a procedure to ensure the coherence of the descendents. The procedure is similar to the tree creation procedure in step (ii), but the new switches are selected from the suffix of the other parent. Figure 4.2 shows a step-by-step example of this recombination of two parents to generate two descendants. There are only two differences to the previous tree creating procedure. First, if a switch in the new suffix is already in the tree, then it is ignored, as showed in Figure 4.2(d). Second, after processing the new suffix, the ToR switches that are still not connected to the tree are added in random order, as showed in Figure 4.2(f).

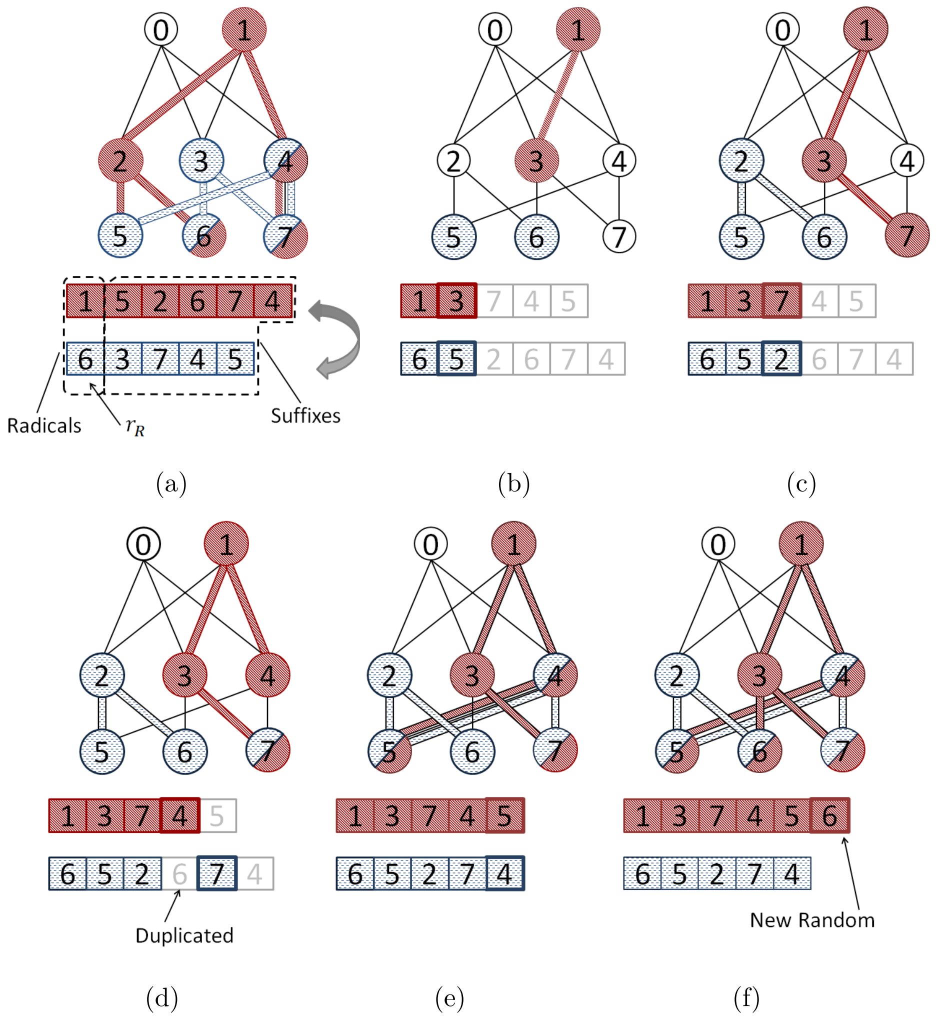


Figure 4.2: Recombination of two parents to generate two descendants. The procedure is similar to the tree creation procedure, except that new switches are selected from the suffix of the other parent. (a) The two trees are showed with their respective genotypes. The random integer $r = 1$ is selected and separates the genotypes in radicals and suffixes, which will be exchanged. (b)–(f) The two descendants are generated in parallel. The dark red nodes belong to the first descendant and the light blue nodes belong to second descendant.

(vi) Mutation of descendants: After the recombination of parents, each descendant may mutate and generate another individual. This occurs with a probability $p_m = 0.10$. The mutation procedure uses the genotype of a descendant I_y to generate a mutated individual I_z, such that the genetic algorithm increases the search space and avoids premature convergence. Similar to the recombination, the mutation procedure samples a random integer $r \in [1, |I_y|]$ as an index on the descendant's genotype $\mathbf{s}_y = \begin{bmatrix} s_1 & s_2 & \cdots & s_m \end{bmatrix}$. Then, the switch at that position is removed and the genotype is divided in a radical and a suffix. The resulting genotype is $\mathbf{s}_y = \begin{bmatrix} s_1 & s_2 & \cdots & s_{r-1} & s_{r+1} & \cdots & s_m \end{bmatrix}$. As in the recombination procedure, the switches in the suffix are added one at time to form the new mutated individual coherently, and any ToR switch that is not present in at the end is included in random

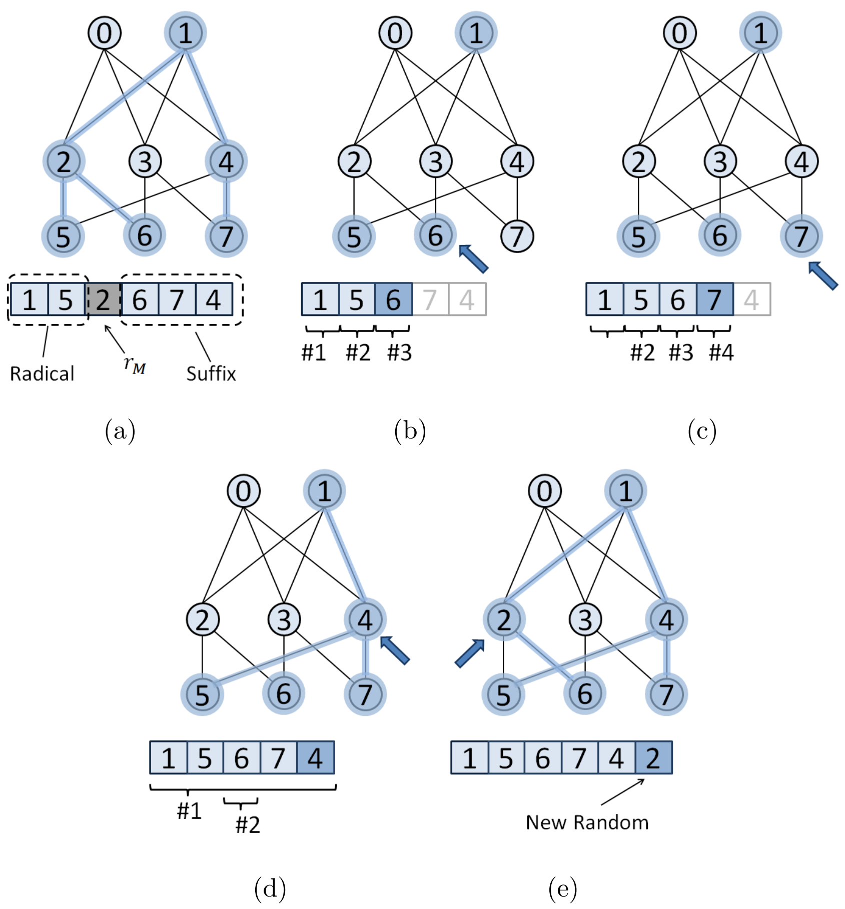


Figure 4.3: Mutation of an individual to form a new individual. (a) The switch vector of the original descendant and the random integer r indexing the switch to be removed. (b)–(e) The formation of the new mutated individual by adding the switches in the suffix one at a time. Although the mutated individual has the same switch set, the two genotypes and resulting trees are different due to the order of the switches in the vector.

order. We present the mutation procedure in Figure 4.3.

(vii) Survivor selection and population size mutation: After the aforementioned operations, we have a set of the original individuals, the descendants, and the mutated individuals. It is then required to select which of these individuals will survive to the next generation. First, we remove all duplicate individuals to ensure that each individual is a different tree. Next, we must determine the size of the new population, since the optimal size is unknown beforehand. We sample the number of individuals of the new population P' from a normal distribution $N(|P|, \sigma)$ centered in the current population size $|P|$ and having standard deviation σ. The result is rounded to the nearest integer. Once the new population size is determined, then the individuals are ordered according to their phenotype and the individuals with the highest phenotype are selected to survive to the next generation P'.

After the new generation P' is created, we compare it to P to determine if the new generation is better. Recalling that each individual $I \subseteq E$ is a subset of edges, we define a function $G(P)$ to quantify a population P as

$$G(P) = \left| \bigcup_{i=1}^{n} I_i \right| . \tag{4.11}$$

The value of $G(P)$ is the number of links used by all individuals in P and thus it serves as a diversity index for the population. In essence, populations that use more links are considered better than those that use fewer links. If $G(P') > G(P)$, then we classify P' as a successful generation of P.

To update σ, we use Rechenberg one-fifth success rule [46]. We observe a certain number of generations and compute the fraction q of successful generations. If $c \in [0.817, 1]$ is a constant, then σ is updated as follows

$$\sigma = \begin{cases} \sigma/c, & q > 1/5 \\ \sigma \cdot c, & q < 1/5 \\ \sigma, & q = 1/5. \end{cases} \tag{4.12}$$

The idea behind Rechenberg one-fifth success rule is that, if q is too large, we may be approaching a local minimum and therefore increasing σ is beneficial to increase the search space in the population size. Likewise, if q is too low, the search space may be too large and we must narrow it down.

In the offline multipath configuration phase, the genetic algorithm calculates and configures the multiple paths as VLAN trees. After the multipath configuration phase, the forwarding scheme selects paths for the flows in the multipath selection phase. This selection of paths is an online and fast procedure. The next chapter presents heuristics for the online selection of paths.

Chapter 5

Online Multipath Selection Phase

The multipath selection phase occurs online whenever a new flow departs from a virtual switch. In this case, the virtual switch contacts its local controller to assign a path (and therefore a VLAN) to the new flow. The entire data-center has a number of local controllers, each itself controls a set of virtual switches. The number of local controllers and virtual switches each one controls depends on the total number of flow arrivals, and we consider as standard case a local controller per rack.

All possible VLAN trees are computed and installed during the multipath configuration phase, and are available to each controller. In order to compute the path, the local controller accesses a database (cf. Section 5.2) containing the active flows and their corresponding paths to select a path. Once this is done, the controller installs an OpenFlow rule on the virtual switch in order to tag each outgoing packet of this flow with the assigned VLAN ID. Likewise, another rule is installed to untag each incoming packet of this flow before forwarding them on to the corresponding virtual machine.

Each local controller manages the flows originated from or directed to a single rack. In order to offload the amount of controlled flows, the local controller can keep track of only the large flows, and as soon as it detects them, migrate to a better path (cf. Section 5.3). To determine when a flow finishes, the local controller frequently queries the flow statistics of the virtual switches (cf. Section 5.4). To detect and cope with link and devices failures in paths, the local controller senses when a flow stops transmitting, which is considered a path failure in that tree and the flow is rescheduled to use a different path. Periodically, the controller sends probes in the failed path and reactivates it once it is available again.

5.1 Selection Heuristics

To select a path for each new flow, the local controller selects the path with the Least-Used-Links (LUL). In order to keep track of link usage, for each link $e \in E$, the database stores the link rate $r(e)$ and the number of flows $u(e)$ concurrently using each link. The link cost is then computed as the ratio $u(e)/r(e)$, such that a link with a higher number of flows and lower rate has a higher cost. The path cost is then computed as the maximum link cost along the path, and the path with the lower cost is selected for a new flow. In case of a tie, the controller chooses the path uniformly. After the selection, the link usage $u(e)$ for all links in the path are incremented. Similarly, when a flow finishes, all link costs of the path are decremented.

We also tested other selection heuristics that relax some database requirements, but still rely on network usage to determine the path. The Least-Used-Tree (LUT), we track the number of flows using each VLAN tree and use this as the cost of the tree. Every time a path of this tree is selected for a new flow, the tree cost is incremented. Similarly, when a flow finishes, the tree cost is decremented. In Least-Used-Path (LUP), the number of flows is tracked on a per-path basis instead of a per-tree basis. Different than the proposed LUL heuristic, both LUT and LUP cannot be applied when links have multiple bit rates because these heuristics only track the number of flows using the resource (a tree for LUT and a path for LUP). Therefore, in order to provide a fair comparison, we use in our simulations the same rate for all links of the data center.

5.2 Database Placement

We consider two extreme cases for the network usage database placement. First, we consider a single GLOBAL database that is accessed by all local controllers. In addition, we also consider each local controller having its own LOCAL database to store the link usage of the paths used by its flows. These two cases (i.e., centralized and distributed) are in opposite sides of the spectrum and should be enough to tell the performance of any hybrid solution, if required.

The GLOBAL database stores information of all active flows in the network; therefore, local controllers have accurate knowledge of the network congestion. However, this requires that all local controllers frequently query and update the database. If the traffic workload is high, the database would have to answer queries and update entries at a high rate, which could thwart the task or would require an elastic data store to keep up with the query/update rate. In addition, all local controllers must communicate with the central database, which may have a high overhead depending the query/update frequency. On the other hand, if the database is LOCAL and co-located with the local controller, then all communications remain local. Nevertheless, the database does not have global knowledge of the network usage and may assume that a path is free when in fact it is not due to the limited visibility.

5.3 Long-Lived Flows Migration

We also considered two approaches, in which new arriving flows use a path and we migrate the long-lived flows to new paths. These approaches relax the requirement of storing the network usage for all flows in the network usage database. Instead, the local controller periodically queries the virtual switches for the flows statistics. and The first approach creates a priority path for all small flows. When they are considered long-lived, we migrate them to different paths according to least-used-links heuristic (SING-MIG). Since the majority of the flows are small, the local controller has to keep track of only a small percentage of the flows, the long-lived ones. This approach reduces the amount of computation of the controller, and at the same time provides a specific priority path for the small flows.

In another approach, we consider that small flows are uniformly distributed in all available paths, and we migrate the long-lived flows to different paths according to least-used-links heuristic (RND-MIG). This approach assumes that the amount of small flows is too large to share a single path. Thus, the small flows follow uniformly allocated paths to balance the load. Both SING-MIG and RND-MIG have a configurable parameter the long-lived flow timer, the minimum time to consider a flow as long-lived. The long-lived flow timer is extremely important, because it determines how much time the long-lived flow occupies the small flows path, and at the same time it indirectly dictates the load at the controller to calculate new flows. To set the long-lived flow timer with OpenFlow, we could use the hard timeout parameter that expires rules, thus forcing the redefinitions of the rule. We use the opportunity to calculate and install a rule defining a new path. If we consider priority queues at the switches, the new rule could also mark the flow as low-priority to ensure the low latency performance to small flows.

5.4 Flow Tracking Policy

As link costs are used for the path selection, the cost information should be up-to-date to prevent avoidable collisions. Hence, in addition to updating the costs when a flow starts, it is also important to update them when a flow finishes. The aforementioned heuristics determines when the flow finishes at the cost of the local controller constantly monitoring the flows at the virtual switches. To loosen this requirement, we propose a few alternative policies to update the link costs when a flow ends.

In the first policy, the link costs are updated immediately when the flow ends (DEC-END). This approach can be implemented using notifications from the virtual switches to their local controllers. We consider this approach as a guideline.

The second policy simply does not update the costs when the flows finish (NO-END). Although simplistic, this approach has some knowledge of the network utilization, because it accumulates the information selection of paths over time.

The third policy schedules a timeout when the flow starts, regardless of the actual flow duration (scheduled fixed end – SFE). This approach sets a duration for each flow and decrements its cost after this interval regardless of the actual flow duration. Therefore, it does not require tracking of each of the existing flows. Nevertheless, it must estimate the duration of the flows *a priori*. In our simulations, we test different fixed end timeout values.

Finally, the last policy periodically monitors the virtual switches in servers to get the number of active flows (periodic monitoring – PM). This approach requires the local controller to constantly monitor the virtual switches to have up-to-date knowledge of the network utilization.

This chapter presented selection heuristics that are analyzed in the next chapter. The selection heuristics consider a GLOBAL or LOCAL database to keep track and select the least used links LUL, paths LUP, or trees LUT. We also relaxed the selection of the new paths to migrate only the long-lived flows SING-MIG and RND-MIG, and the detection of when a flow ends being either inexistent NO-END, scheduled SFE, or detected by monitoring PM. Next, we simulate the heuristics and present the results.

Chapter 6

Simulator and Results

In this chapter we present the simulator and the simulation results of the proposed Two-Phase Multipath (TPM) forwarding scheme compared with Spanning Tree Protocol (STP) and Equal Cost MultiPath (ECMP).

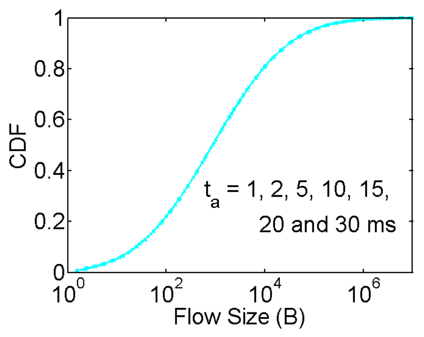
6.1 Simulator

6.1.1 Flow Simulator

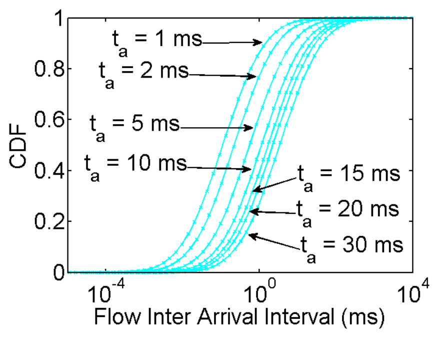
We developed a discrete event simulator that models data transmission as a flow, which allows simulations on higher abstraction level when compared to packet simulators such as NS3[1], which allows simulation of higher transmission rate and larger topologies. Our flow simulator organizes the events in a queue ordered by the time of the event. The simulator picks the next event from the queue, updates the simulation time and executes it. The simulation ends when the queue of events is empty or the simulation picks an event, which the time is greater than a limit.

In the simulations, all servers send and receive flows through a Top of Rack (ToR) switch. Thus, we only consider ToR switches as source and destination, since there is a single path from the virtual machines to the nearest ToR switch. The model of the flow considers the source ToR switch, the destination ToR switch, the flow size, the current transmission rate, and the transmitted bytes. We compute the flow transmission rate at a given time as the fair share of the most contended traversed link using a max-min fairness algorithm, which is an optimistic flow model, i.e., it assumes flows immediately increase/decrease their rate due to the arrival/departure of other flows in the path. The results are presented with 95% confidence interval. The total bytes field transferred by a flow includes the TCP/IP and Ethernet header bytes. All packets have the maximum size, 1500 bytes, allowed by Ethernet, except the last packet, which sends the remaining flow bytes.

[1]http://www.nsnam.org/



(a) CDF of the flow sizes.



(b) CDF of the flow inter-arrival intervals.

Figure 6.1: Workload model of the flow simulator. The mean of natural logarithm of the inter-arrival interval μ represents the workload scenario, the smaller the μ, the shorter the mean inter-arrival interval and the greater the load.

6.1.2 Workload Model

The workload model consists of two main parameters, the flow size and the flow inter-arrival interval. As the flow becomes larger, the time to transmit is longer and it has a greater probability that the flow shares the link bandwidth with other flows. The flow sizes follow a lognormal distribution ($X_s = e^{\mu_s + \sigma_s Z}$) with Z a standard normal variable. We choose the mean and standard deviation of the natural logarithm of the flow sizes $\mu_s = 7$ and $\sigma_s = 2.8$, such that the Cumulative Density Function (CDF) has the following values $F(x) = \{\approx 0.5 | x = 1000, \approx 0.95 | x = 100000\}$, according to empirical measures presented by Benson *et al.* [19]. The flow sizes CDF is presented in Figure 6.8(a). The second parameter of the workload, the inter-arrival interval, is used to test the data center with under different loads. Smaller inter-arrival intervals increase the creation rate of new flows and, consequently, increase the probability of more flows sharing links bandwidth. The inter-arrival intervals (in microseconds) also follow a lognormal distribution ($X_t = e^{\mu_t + \sigma_t Z} \cdot 10^{-6}$) with parameters mean and standard deviation of the variable natural logarithm $\sigma_t = 2$ and μ_t. We configured μ_t based on a chosen inter-arrival time median $t_a = e^{\mu_t}$, varying it to decrease the workload $t_a = \{1, 2, 5, 10, 15, 20, 30\}$ ms. The t_a values are chosen to model the inter-arrival intervals similar to the empirical measures presented by Benson *et al.* [19]. Figure 6.8(b) presents the inter-arrival intervals CDF for each workload scenario. As the t_a parameter increases the rate of new flows and the load decrease.

We compare our proposed scheme with two forwarding mechanisms that use conventional features of the network devices: the Spanning Tree Protocol (STP), and with Equal Cost MultiPath (ECMP) forwarding. The Valiant Load Balancing (VLB) selects paths at random with a uniform distribution, thus it achieves performance comparable to ECMP.

STP: the switches create a single tree to forward all traffic. Thus, there is a single path between each pair of ToR switches.

ECMP: link state routing protocols identify the multiple paths with the same costs between each pair of ToR switches. At each hop, a hash function is applied on certain fields of the packet header to uniformly distribute the flows over the paths, and to ensure that all packets of a flow traverse the same path [17]. We model ECMP as a uniform random variable to select paths.

6.2 Simulation Results

We used the Fattree, Cisco three-tier and Jellyfish topologies, and we also varied the size of the Fattree topology. These topologies represent both structured and random designs that provide multiple paths, which forwarding mechanisms avail to improve the network performance. We also tested other traffic load pattern to investigate the relation between small and large flows. Finally, we simulated the heuristics with relaxed constrains. Table 6.1 summarizes the notation used in this chapter to show the names of the proposals.

Table 6.1: Multipath schemes name abbreviations.

Multipath Configuration Proposals	
STP	Spanning Tree Protocol
ECMP	Equal Cost MultiPath
TPM	Two-Phase Multipath
Path Selection Heuristics for TPM Selection	
LUL	Least-Used-Links
LUT	Least-Used-Tree
LUP	Least-Used-Path
SING-MIG	Small in single path, migrate long-lived using LUL
RND-MIG	Small in random path, migrate long-lived using LUL
Database Location for TPM Selection	
GLOBAL	Single database for the data-center
LOCAL	Per rack database
Flow End Policy for TPM Selection	
DEC-END	Decrement cost whenever flow ends
NP-END	Never decrement cost
SFE	Scheduled fixed end when flow arrives
PM	Periodic monitoring to detect flow end

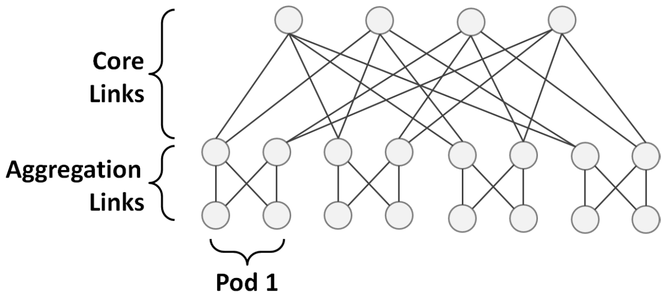
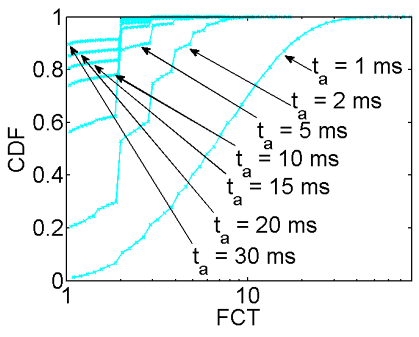
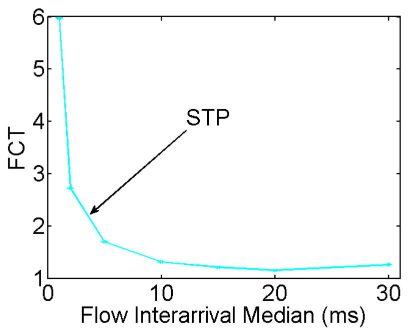


Figure 6.2: Fattree topology with 4-port switches used in our simulations. There are four different paths to any ToR switch in another pod, and two different paths to a switch in the same pod.



(a)



(b)

Figure 6.3: Flow completion time (FCT) of STP. (a) The FCT CDF as a function of inter-arrival time median t_a. For $t_a = 1$ ms, 30% of flows last more than 10x the line data-rate time transfer. (b) The expected FCT value of STP decreases when t_a increases because congestion is less likely to occur.

6.2.1 Fattree 4 Topology

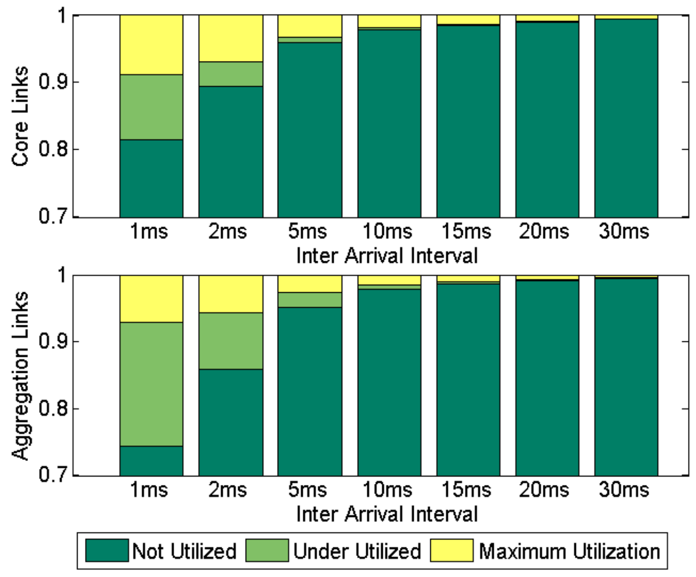
We first run simulations using the Fattree topology with 4-port switches [22], which presents two different paths to a switch in the same pod (an aggregation switch away from the ToR), and four different paths to the ToR switches in any other pods, as depicted in Figure 6.2. The destinations of the flows are uniformly selected to evenly distribute the traffic across the data center.

We use flow completion time (FCT) as our metric to indicate the quality of the multipath forwarding scheme. This metric indicates the quality of the forwarding scheme despite the topology and the network capacity. For each scheme, the FCT is the flow duration when multiple flows are present normalized by the flow duration when there is no other concurrent flow in the data center, and thus transmitted at line speed. A good forwarding scheme offers the maximum bandwidth for the flows and FCT approximates one. For mice flows, small FCTs represent that the flows are quickly transmitted, thereby meeting the deadline. Small FCTs for elephant flows means that these flows achieve high transmission rates which result in improved link usage efficiency, leaving freer paths for other flows. On the other hand, high FCT indicates that flows are contending for bandwidth, causing lost deadlines and inefficient link usage. Figure 6.3(a) show the FCT CDF for STP. For high inter-arrival times (e.g., $t_a > 15$ ms), more than 80% of flows have the minimum FCT of one, and a small percentage of flows has an FCT larger than one. Nevertheless, by decreasing the inter-arrival time ($t_a \leq 10$ ms), a higher percentage of the flows has an FCT larger than 2. In the heavier workload scenario ($t_a = 1$ ms), very few flows have an FCT of 1, and around 30% of flows have an FCT larger than 10. This trend is showed in Figure 6.3(b), which shows the expected FCT as a function of the inter-arrival time.

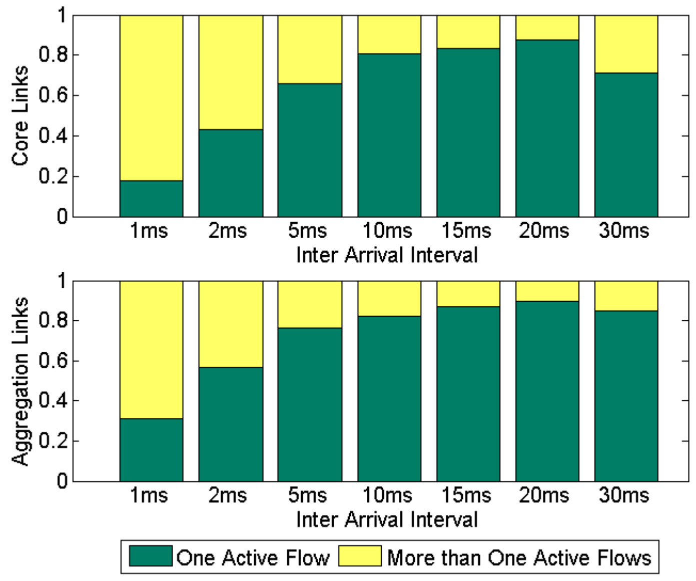
To understand the STP performance, we investigate the core and aggregation link usage. Figure 6.4(a) shows the fraction of time that links are not utilized, are underutilized due to a bottleneck in another link along the path, and are fully utilized at line rate. As the workload increases (t_a decreases), the fraction of time that links are utilized is higher, but flows cannot exploit the capacity of most links. All flows whose destination is in another pod share the same core links, and thus we expect that the core links are more heavily used. However, we observe that most of the time core links are not utilized. As STP uses a single tree to forward traffic, all links not belonging to this tree are free. The core links in the tree also have active flows only part of the time; however, the bandwidth sharing is uneven and flows cannot use all available bandwidth.

In addition to low link usage, the high expected FCT is also caused by the high number of active flows sharing the same link. We see from Figure 6.4(a) that links have no active flows most of the time, but as the workload increases (t_a decreases), several active flows share the link bandwidth, reducing the expected FCT. Figure 6.4(b) shows the fraction of time that links are used by a single flow or by more than one flow. For each link, we only consider the time that it has at least one active flow. We see that both core and aggregation links have more than one flow for more than 20% of the time when $t_a = 10$ ms and this is even worse with $t_a < 10$ ms, with a direct impact on FCT. However, in $t_a = 30$ ms, the percentage of more than one active flows is higher than smaller t_as. The generated data with random distribution of the flow sizes and inter-arrival interval resulted in some very large flows concurrently sharing the links, which impacted several other flows. This behavior is also seen with other forwarding schemes.

We now compare the performance of the same Fattree topology for STP, ECMP, and the proposed TPM using the Least-Used-Links (LUL) heuristic for path selection and LOCAL database placement. We also present results using a GLOBAL database placement with entire knowledge of the network just as a performance baseline, but do not consider the communication nor the query/update overheads of this



(a) Core and aggregation link usage.



(b) Number of active flows per link.

Figure 6.4: (a) Link usage at core and aggregation links. STP wastes significant network resources by using only a single tree to connect all ToR switches. (b) Number of active flows per link, considering that at least one flow is active. At higher workloads, multiple flows share the same link and reduce the available per-flow rate.

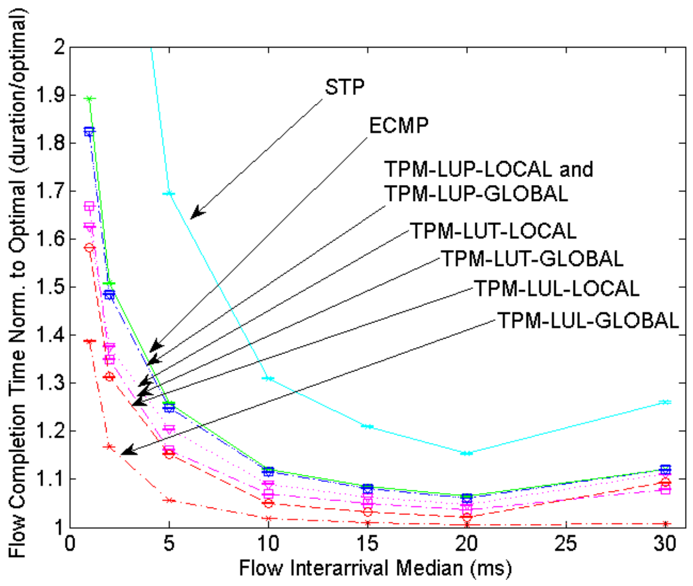
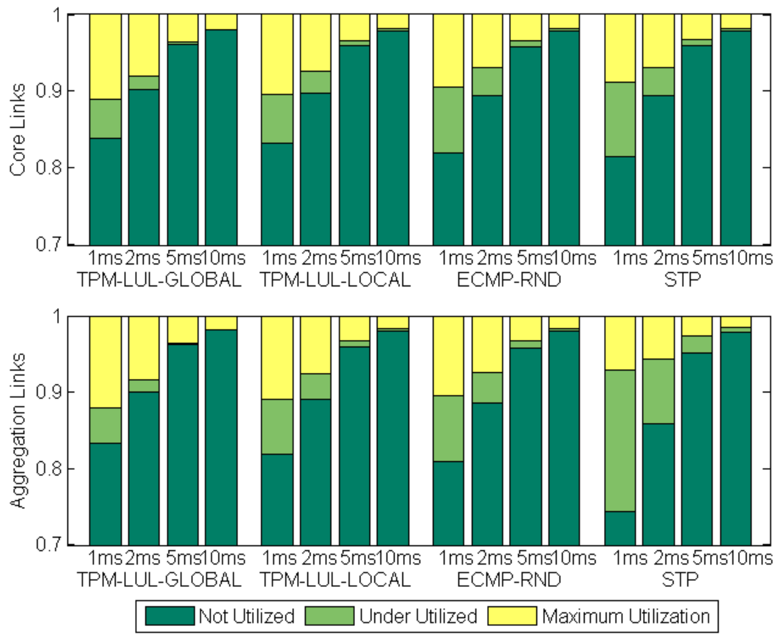


Figure 6.5: Expected FCT under different workload scenarios (t_a values) for the multipath selection heuristics for STP, ECMP, and TPM using LUT, LUP, and LUL as path selection heuristics and both GLOBAL and LOCAL database locations for Fattree 4 ports.

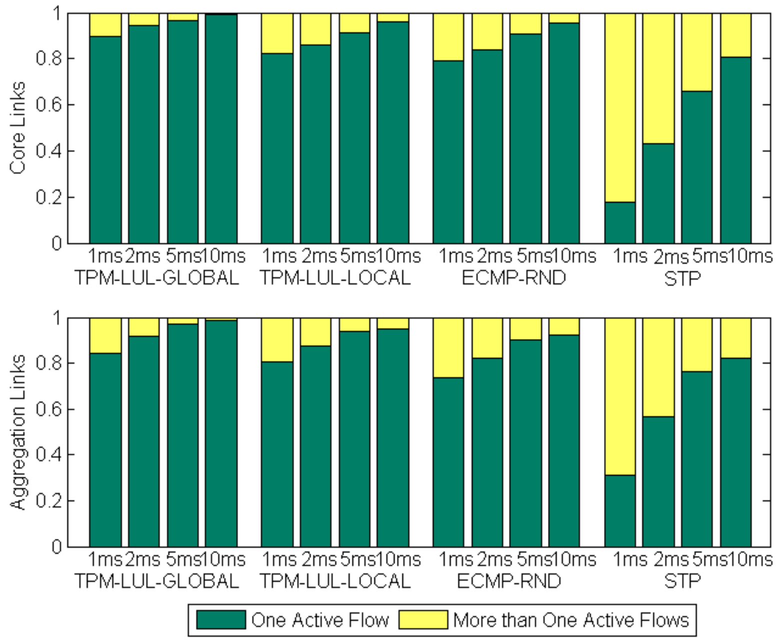
approach. In addition, we also present results using Least-Used-Tree (LUT) and Least-Used-Path (LUP) [2].

Figure 6.5 shows the expected FCT for each of the aforementioned techniques. TPM clearly outperforms STP and ECMP by a significant margin, especially when the data center is overloaded. In particular, when t_a is 5 ms, TPM-LOCAL reduces approximately 11% of the optimal flow completion time when compared to ECMP and approximately 54% when compared to STP. If the data center has an even higher load at $t_a = 1$ ms, then TPM-LOCAL roughly reduces 31% over ECMP and 4.4x over STP. If a GLOBAL database is used instead, then these gains are even more pronounced. At $t_a = 5$ ms, TPM-GLOBAL reduces 20% of the optimal flow completion time over ECMP and more than 64% over STP. In the highest workload scenario of $t_a = 1$ ms, TPM-GLOBAL reduces approximately 50% over ECMP and 4.6x over STP.

Figure 6.5 also presents results for different path selection heuristics and database locations. We see that LUP is an optimistic approach since it assumes that all paths are disjoint. However, as this is not the case in data centers, LUP suffers from selecting paths with already congested links. In contrast, LUT is a pessimistic approach, because it assumes that all flows in a tree share the same links. LUT achieves reasonable performance, reducing FCT up to 22% of the optimal flow completion time compared to ECMP. Our LUL heuristic, however, presents the most fine-grained knowledge of link usage and it has therefore the best performance. With regard to



(a) Core and aggregation link usage.



(b) Number of active flows per link.

Figure 6.6: (a) Link usage at core and aggregation links for STP, ECMP and TPM for both GLOBAL and LOCAL database placements. (b) Number of active flows per link, considering that at least one flow is active.

database location, both LUT and LUL benefit from the GLOBAL knowledge, since the LOCAL database is unaware of the true path usage. However, this is not the case for LUP, because paths originated by a particular ToR switch are not shared with other switches. Therefore, path usage information is contained in the local database and a global database does not improve much the performance.

To further investigate the performance of the schemes, we again measure the core and aggregate link usage as well as the number of active flows per link. Figure 6.6(a) shows the core and aggregation link utilization for $1 \leq t_a \leq 10$ ms. Clearly, STP presents the worse performance because it shares a single tree for all flows. Additionally, most links are often underutilized, reducing even more the performance. STP core links are highly utilized with more than 25% in the highest workload scenario and become bottlenecks. In contrast, ECMP and TPM use multiple paths for each pair of ToR switches. As a result, the bottleneck becomes the aggregation links because fattree topologies offer less disjoint paths to ToR switches in the same pod. Nevertheless, TPM with both database locations have less underutilized links than ECMP, which results in better transmission performance. Figure 6.6(b) shows the number of flows per link, when links are used by one or more flows. The number of active flows in STP is high due to the limited path diversity, reducing its performance. ECMP randomizes path selection and does not take network usage information into account, resulting in path collision even though there are other unloaded paths available. Similar to ECMP, TPM uses different paths to forward flows by creating trees that result in the same paths as ECMP in a Fattree 4 topology. Nevertheless, as TPM has the network utilization information available during path selection, its performance is higher even with only a LOCAL database. In this case, TPM reduces link underutilization as well as the number of flows sharing the same path. Overall, TPM path selection result in best performance due to the higher maximum utilization of links at line speed and lower under utilization rate, which also result in more flows not contending for bandwidth.

6.2.2 Small and Large Flows Proportion in Fattree 4

The workload used in the simulations so far uses a fixed relationship between small and large flows provided by the parameters $\mu_s = 7$ and $\sigma_s = 2.8$. Considering flows with less than 100 kB as small and flows with more than 10MB as large, the percentage of small and large flows in this workload are 94.86% and 0.0002%, respectively. To analyze the impact of the percentage of small and large flows, we vary the standard deviation σ_s of the flow size within the interval $[2, 4]$, which varies the percentage of small flows from 98.86% to 87.17% and the percentage of large flows from 0.0002% to 1.09%. As Figure 6.7 shows, the relationship between the

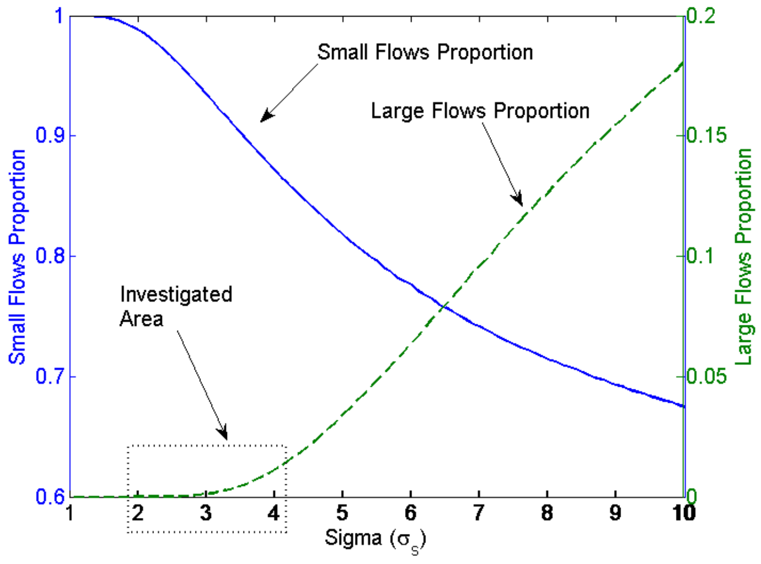
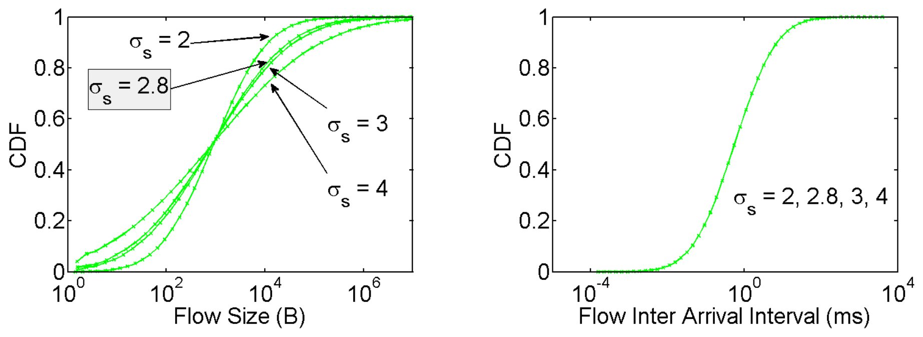


Figure 6.7: Percentage of small flows (less than 100kB) and large flows (more than 10MB) varying the σ_s of the flow size lognormal distribution with $\mu_s = 7$.



(a) CDF of the flow sizes.

(b) CDF of the flow inter-arrival intervals.

Figure 6.8: Workload model for the flows size lognormal σ_s variation. Besides the sigma variation simulation, the $\sigma_s = 2.8$ for other simulations.

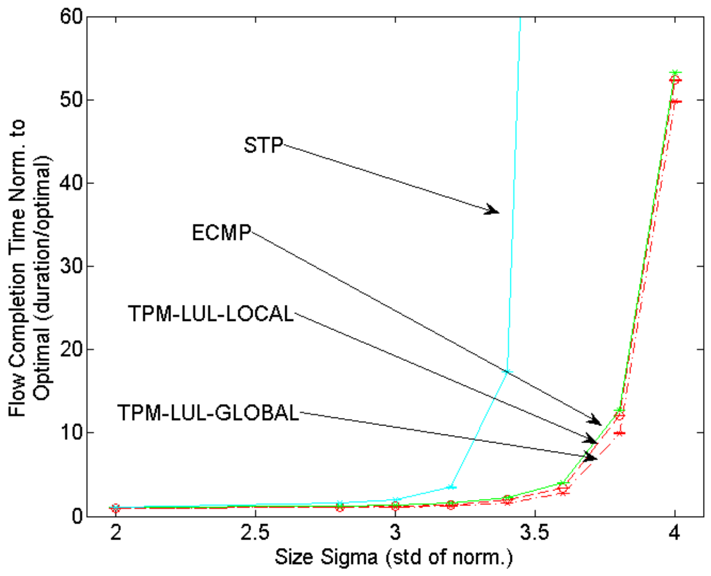
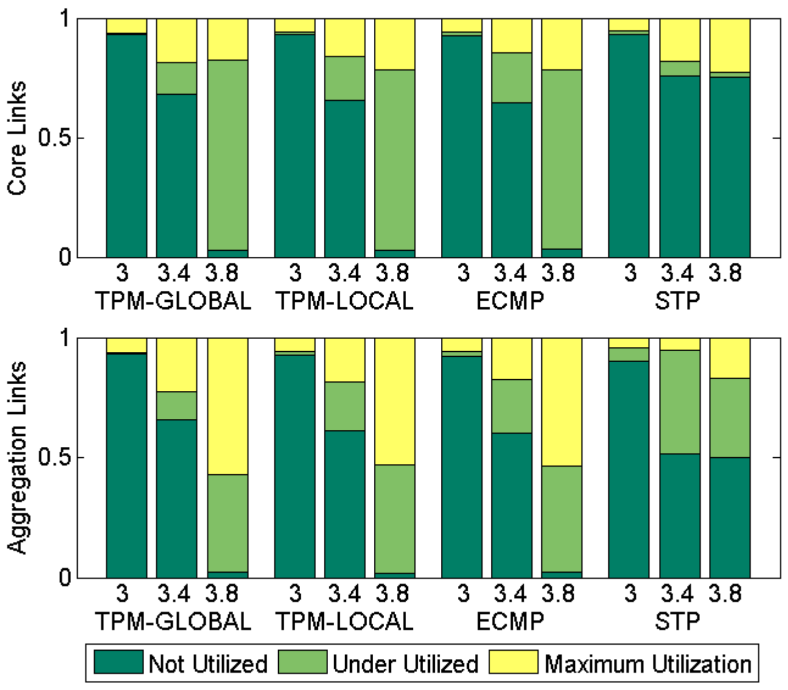


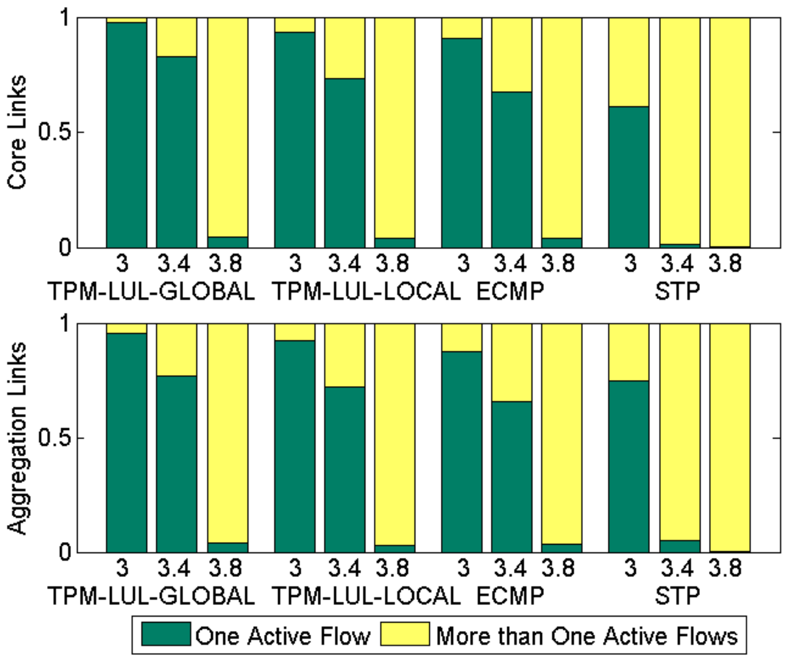
Figure 6.9: Expected FCT varying the proportion of small and large flows by changing the standard deviation σ_s of the lognormal distribution of flow sizes, with $\mu_s = 7$ and $t_a = 5$ ms.

large and small flows relationship changes considerably by varying σ_s. Figure 6.8 shows the workload for the chosen parameters. Although the variation of σ_s is in a small interval, it has huge impacts in the performance. We keep t_a fixed at 5 ms for different σ_s values to provide the same load across experiments.

As σ_s increases, there is more variance on the flow size and consequently more large flows. The presence of more large flows results in more flows sharing the same links, which severely impacts FCT, as showed in Figure 6.9. The FCT can be up 50 times worse due to the long-lived flows. Figure 6.10 shows both the utilization and the number of flows per aggregation and core link. The tree used by STP has half of the aggregation links and a fourth of the core links. Hence, increasing σ_s, the core links utilization approximates 25% and the aggregation links 50%, which are all links available for STP. Furthermore, the links used by STP forwarding scheme, often are shared by several flows. On the other hand, ECMP and TPM use all links of the topology, thus increasing σ_s causes the forwarding schemes to use more of the links, which enhances the performance. When $\sigma_s = 4$, approximately 1% of the flows are large, but this drastically changes the workload traffic. In particular, links are much more overloaded and used by several flows most of the time, as showed in Figure 6.10. Although the relative increase of large to the small degrades the performance, TPM can still improve the performance when compared to ECMP. The main reason is that unlike ECMP that blindly distributes the flows into paths, TPM detect which path more overloaded and avoid them.



(a) Core and aggregation link utilization.



(b) Number of active flows per link.

Figure 6.10: Core and aggregation link utilization of $\sigma_s \in [2, 4]$ for ECMP and TPM for both GLOBAL and LOCAL database placements.

6.2.3 Larger Fattree Topologies

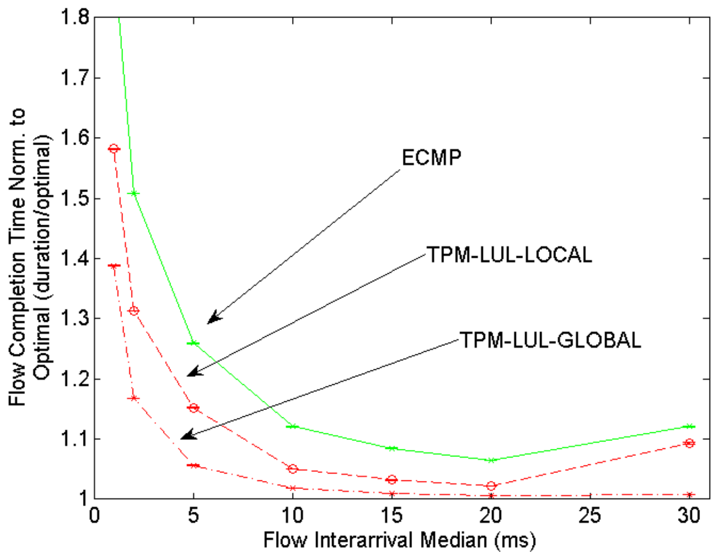


Figure 6.11: Expected FCT for Fattree with 4-port switches.

To analyze the performance of larger Fattree topologies, we also provide results using Fattree topologies using 6- and 8-port switches. The Fattree 6 topology is composed of 6 pods, 9 core switches, and 54 servers, while Fattree 8 is composed of 8 pods, 16 core switches and 128 servers.

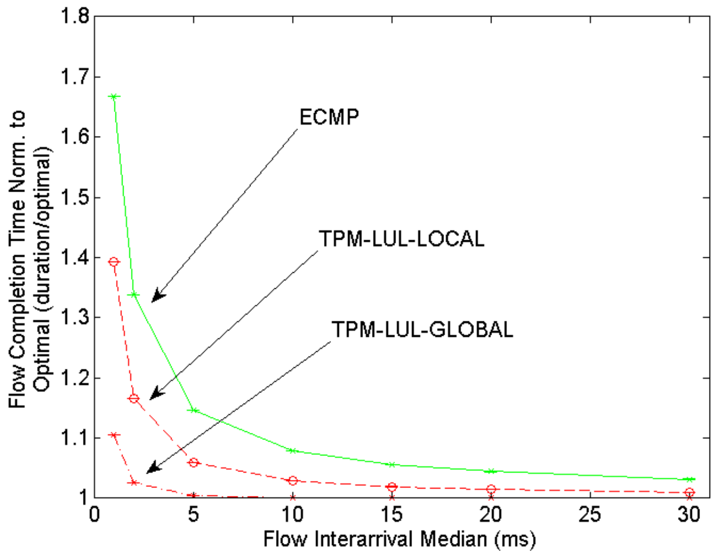


Figure 6.12: Expected FCT for Fattree with 6-port switches.

Figures 6.12 and 6.13 show the FCT achieved in both topologies. Compared with Fattree 4 shown in Figure 6.11, both Fattree 6 and Fattree 8 reduce the overall FCT. Although there are more servers and ToR switches in these larger topologies,

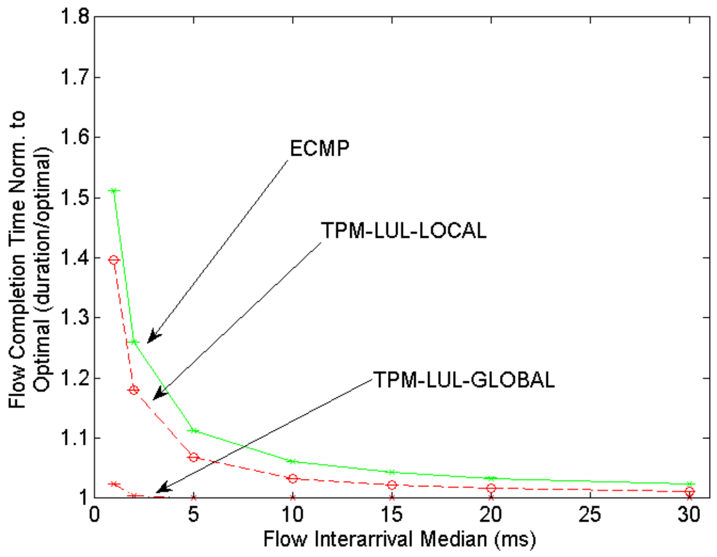


Figure 6.13: Expected FCT for Fattree with 8-port switches.

there are also more available paths to transmit the additional workload. This difference is visible when comparing Figures 6.11, 6.12 and 6.13. Additionally, the extra available paths benefits TPM even more, because it has more options to balance the traffic. However, in larger topologies, the local information becomes less relevant as it is limited to a smaller proportion of the network and, as a consequence, LOCAL database approaches do not perform as well as a GLOBAL database.

6.2.4 Cisco 3-Tier Topology

We also used Cisco 3-tier topology [49], the Cisco standard topology for three layer data-centers. The topology used has three aggregation modules, each with four access switches, as showed in Figure 6.14. All network devices operate only on the link layer and, thus, TPM is able to install the required VLANs. The topology is composed of 12 ToR, 6 aggregation, and 2 core switches, totaling the same 20 switches as in the Fattree 4 topology. Although the redundant paths are mostly used for fault tolerance in the Cisco 3-tier topology, we allowed all links to be equally selected for forwarding to provide a fair comparison.

Figure 6.15 shows the expected FCT for the Cisco 3-tier topology. The performance of this topology is worse than Fattree 4 because it has less disjoint paths, even though it has the same number of switches and less ToR switches generating traffic. The 3-tier topology main goal is the vertical communication, and under high horizontal communication, the overall performance degrades. Nevertheless, the Cisco 3-tier topology significantly benefits from TPM, achieving an FCT reduction of 33% under the highest workload compared to ECMP.

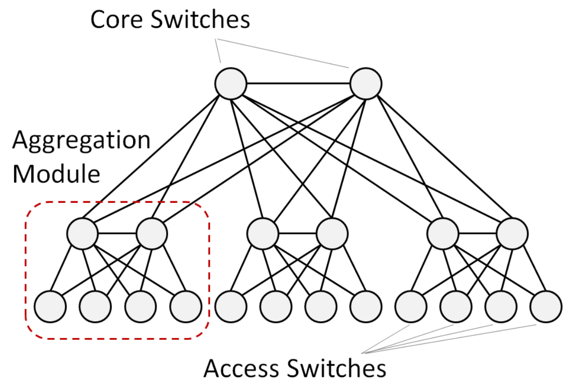


Figure 6.14: The Cisco 3-tier topology, composed of the same 20 switches as in the previous Fattree 4 topology. The main goal of this topology is the vertical communication.

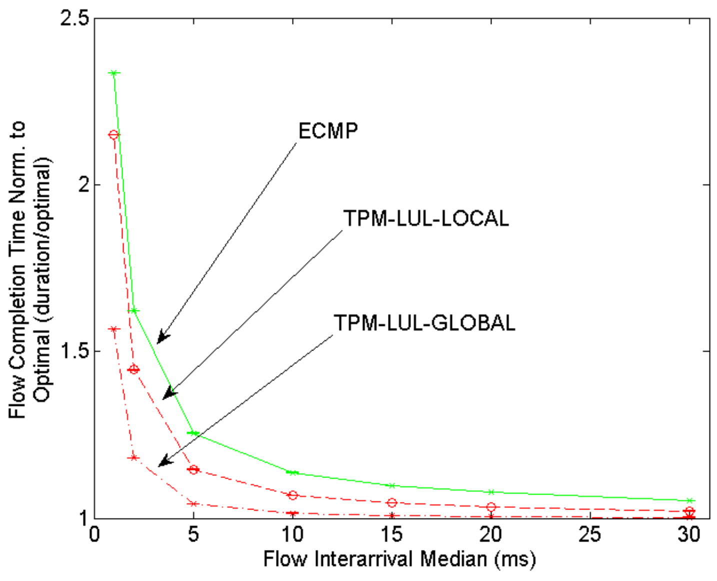
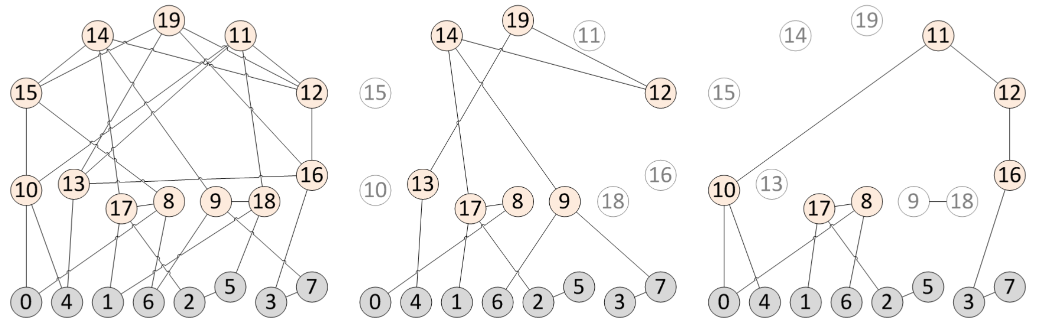


Figure 6.15: Expected FCT for the Cisco 3-tier topology. Although under high horizontal communication the overall performance degrades when compared to Fattree 4, the Cisco 3-tier topology significantly benefits from TPM.

6.2.5 Jellyfish Topology



(a) Jellyfish topology used on simulations. (b) Acquired VLAN 1 using genetic algorithm. (c) Acquired VLAN 2 using genetic algorithm.

Figure 6.16: Random generated 8-rack Jellyfish topology used on simulations and two examples of VLANs acquired by the proposed genetic algorithm. The jellyfish topology has 20 4-port switches whose 8 are Top of Rack with two servers each.

One of the data-center design problems is managing the growth of the infrastructure to support an increase in demand. Structured data-center designs do not allow the addition of few devices to supply the increasing demand; instead, the addition of an entire overprovisioned new module with several devices is required. Singla *et al.* [50] proposed the Jellyfish network topology to allow incremental infrastructure expansions in data centers. Jellyfish is a degree-bounded random regular graph interconnecting the ToR switches. Jellyfish uses a simple iterative procedure to create a sufficiently uniform random regular graph, solving efficiently a complex graph theory problem. The network is initially assumed to have no links. At each step, a pair of switches with free ports is selected and interconnected, repeating this procedure until no further links can be added. If in the end there is still a switch s_i with more than one free ports, then a random existing link (s_j, s_k) is removed, and two links (s_i, s_j) and (s_i, s_k) are created instead. Figure 6.16 shows the 8-rack Jellyfish topology used in our simulations, which has eight 4-port ToR switches that use two of those ports to connect to servers. Figures 6.16(b) and 6.16(c) shows two examples of the VLANs generated by the proposed genetic algorithm of Chapter 4.

Figure 6.17 shows the expected FCT for a 8-rack Jellyfish topology. The FCT values of all forwarding schemes are higher than both Cisco 3-tier and Fattree 4 topology. In particular, this occurs because the generated Jellyfish topology has the majority of ToR switches connected to two core switches, but some ToR switches are directly connected to other ToR switches. Although this benefits the communication between the two racks, it degrades the communication with other racks. In spite of this problem, performance still improves by using TPM, achieving a 29% of FCT reduction compared to ECMP.

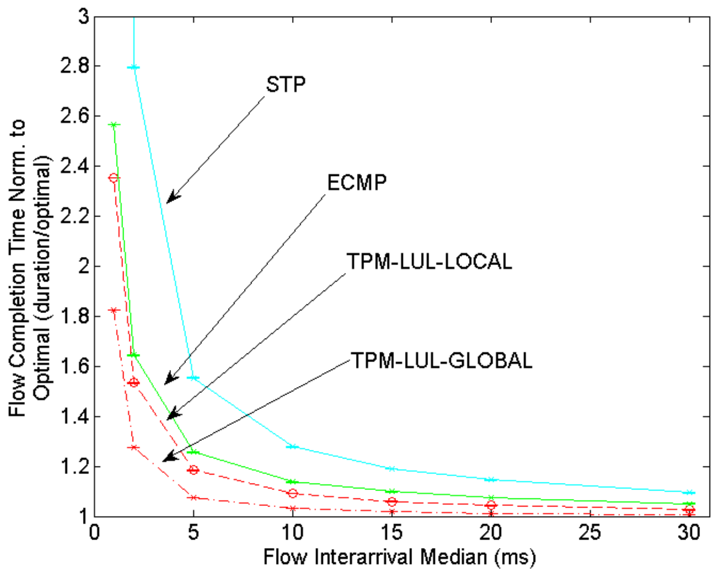


Figure 6.17: Expected FCT for the Jellyfish topology. Connections between two ToR switches benefit communication between the two racks, but degrade communication with other ToR switches. Still, TPM improves FCT when compared to ECMP.

We also used another configuration for the 20 4-port switch Jellyfish topology to investigate how the increase in number of links affects network performance. Figure 6.18 shows a second Jellyfish topology, which has 16 ToR switches using one port to connect to a server and the other three ports to connect to the network infrastructure. Figures 6.16(b) and 6.16(c) shows two examples of the VLANs generated by the proposed genetic algorithm of Chapter 4. As each ToR has half of the number of servers, we halved the inter-arrival time to fairly compare it to the 8-rack Jellyfish topology.

Figure 6.19 shows the expected FCT for the 16-rack Jellyfish topology. As we can see, each ToR switch uses one more port to connect to the other network devices, which creates the possibility of shorter paths. As ECMP always use the shortest paths, the presence of direct paths between ToR switches increases ECMP performance. Additionally, as TPM-GLOBAL has knowledge of all network usage, it chooses the best path, which in this scenario presented slightly better results than ECMP. In contrast, the LOCAL database has little knowledge of the network conditions, and since the several paths use the outbound paths of other ToR switches, it highly impacts the overall performance. Thus, the 16-rack Jellyfish topology, which connects servers within fewer hops, improves the overall performance.

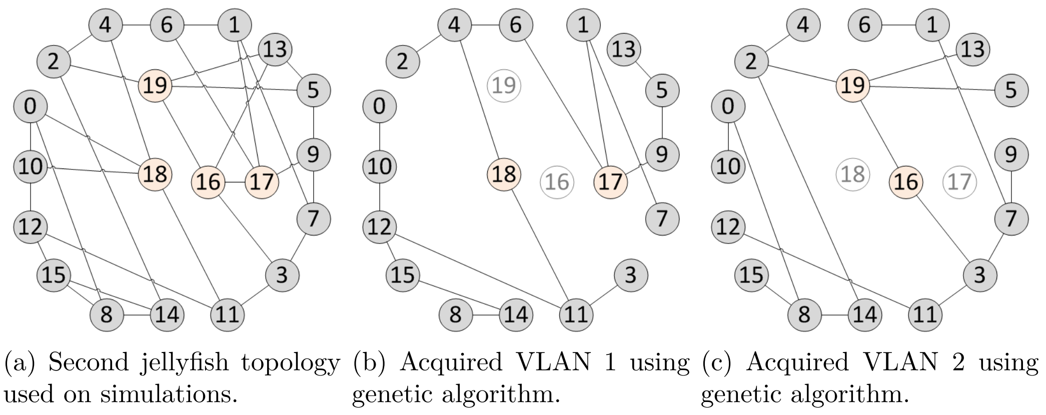

(a) Second jellyfish topology used on simulations.
(b) Acquired VLAN 1 using genetic algorithm.
(c) Acquired VLAN 2 using genetic algorithm.

Figure 6.18: Randomly generated Jellyfish topology with 20 4-port switches, out of which 16 are ToR switches. This topology also uses 20 switches, but allows servers to communicate with fewer hops, increasing the overall end-to-end throughput.

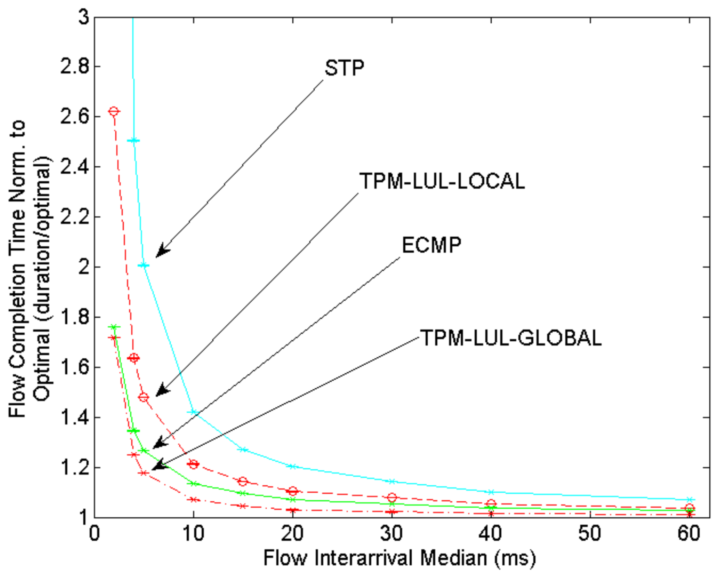


Figure 6.19: Expected FCT for the 16-rack Jellyfish topology. The presence of more ToR ports for communication between other switches improves the overall performance.

6.2.6 Long-Lived Flows Migration

Keeping track of all flows in a data-center may lead to huge processing and storage. Instead, the forwarding engine may keep track of fewer flows, mainly the long-lived. We simulated to approaches for migrating the flows: a single path for small flows and migrate long-lived flows to different paths according to LUL (SING-MIG); and random paths for small flows and migrate long-lived flows to paths according to LUL (RND-MIG). The main difference between both approaches is that in SING-MIG all flows use the same path when they first arrive, and they are detected as long-lived, they migrate to different paths. While in RND-MIG, the small flows are uniformly distributed in all available paths, which are shared with the long-lived flows. We configured t_l the timer to consider the flows as long-lived as 1 ms and 80 ms. 80 ms at line rate 1 Gb/s results in approximately 10 MB, above of which we consider large flows. A 1 ms transmission at line rate results in approximately 135 kB, which is 35 kB higher than what we consider small flows. We considered GLOBAL database for these simulations, and Fattree 4 topology.

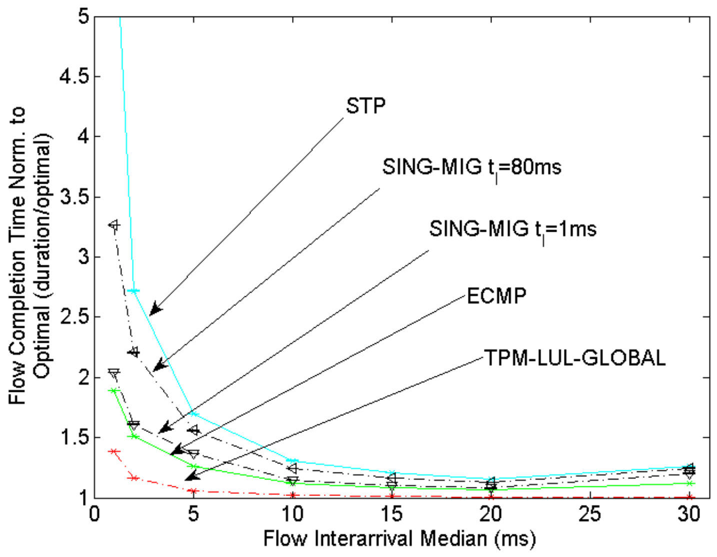


Figure 6.20: FCT of the migration of long-lived flows. SING-MIG approach uses a single path for small flows and migrate long-lived flows to different paths according to LUL.

Figure 6.20 presents the results for (SING-MIG approach. As we can see, even though the flows sharing the single path are short, there is a large number of them which severely impacts the performance. With $t_l = 80$ ms, the performance of SING-MIG is almost as bad as STP. This happens because a great majority of the flows share the same, exactly as in STP. Considering the STP as the worst scenario, using the timer at 80ms, up to 99.5% of the flows share the same tree. It would

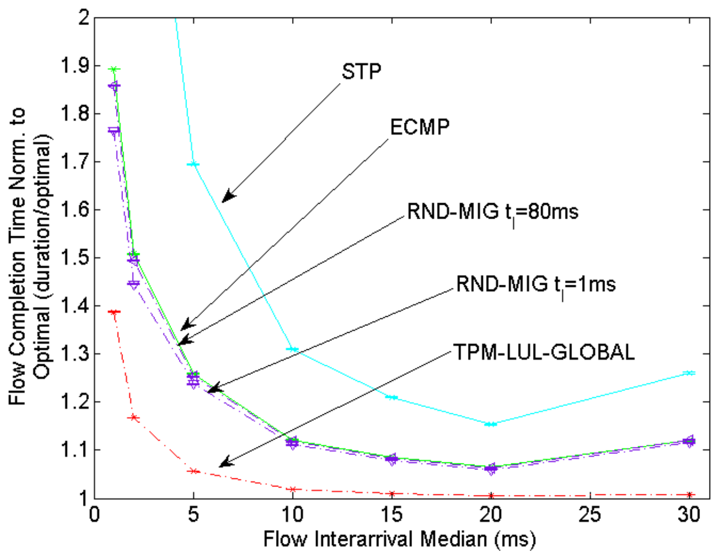


Figure 6.21: FCT of the migration of long-lived flows. RND-MIG approach chooses a random path for small flows and migrate long-lived flows according to LUL.

seem reasonable to use the 80ms t_l timer to consider large flows, but leaving the flows in the same path shared by several others for the time would increase the transmission times for all flows in that path. Thus, it also increases the fraction of flows that violate the t_l timer. Therefore, it is advisable to use smaller timers to begin migrating flows. The 1ms timer in the worst case corresponds to up to 85% of the flows. Although a little more performant than 80ms timer, with $t_l = 1$ ms the results are still lower than ECMP.

The simulation results for RND-MIG approach to migrate the long-lived flows are presented in Figure 6.21. Randomizing the path that the flow uses improves the performance when compared tom SING-MIG, and is still better than ECMP. In high workload scenarios with $t_a = 1$ ms the FCT of RND-MIG is 2% and 7% smaller than ECMP for $t_l = 80$ ms and $t_l = 1$ ms respectivelly. With $t_l = 80$ ms, approximately 99.5% of the flows remain in their random allocated path until it ends. This behavior is also seen in ECMP, which explains the close performance. At the same time, this also means that the local controller tracks only 0.5% of the flows, considerably reducing the overhead in the local controllers. When we use $t_l = 1$ ms, up to 85% of the flows remain unbalanced by the local controller, but it also means the local controller tracks 15% of the flows. As the results, show, we see a clear compromise between the fraction of tracked flows and the performance of the forwarding scheme.

6.2.7 Flow Tracking Policy

The path selection for a flow increases the cost of the links along this path. When a flow finishes, it is required to reduce the link costs such that future flows can properly use the available network resources. We analyze the performance of four policies to decrease the link cost: (i) immediately when each flow finishes (no special name), (ii) never (NOEND), (iii) prescheduling a fixed timer for the flows (SFE), and (iv) by periodically monitoring to sense the active flows (PM). We present the simulation results for TPM with both a GLOBAL and LOCAL database for the highest workload ($t_a = 1$ ms).

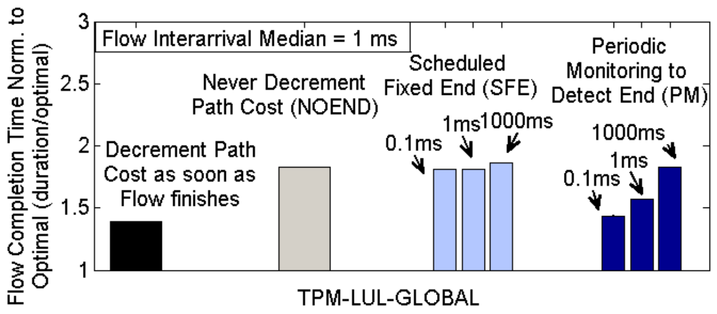


Figure 6.22: Expected FCT for different flow tracking policies for TPM with a GLOBAL database in Fattree 4 topology.

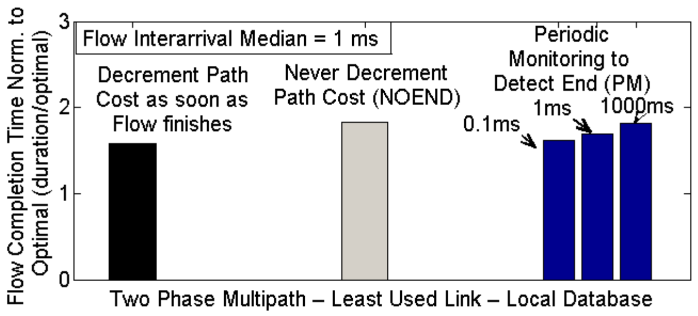


Figure 6.23: Expected FCT for different flow tracking policies for TPM with a LOCAL database in Fattree 4 topology.

Figure 6.22 shows the results of a GLOBAL database. Policy NOEND does not know when flows end and thus it balances the traffic by prioritizing paths whose links have been less selected. Policy SFE avoids active monitoring, but the correct tuning of the flow duration is crucial for performance. In the simulations, the flows durations vary from few microseconds to several seconds with an expected duration of roughly 1 ms. Thus, the scheduled flow end for 1 ms presents the best performance, but it is more complex and presents a performance comparable to never decrementing the costs NOEND. The PM policy actively monitors the virtual switches, and can accurately estimate the link usage. As expected, the more frequent the monitoring rate, the better the performance of the selection heuristic.

Figure 6.23 presents the equivalent results using the proposed LOCAL database placement. The results show that a short monitoring period improve the performance longer monitoring periods, although the difference in absolute terms are not notorious, since the performance of LOCAL database placement itself is comparable to never decrementing the costs NOEND. In particular, this occurs because the information is only local and a higher polling rate is not enough to improve performance. Nevertheless, the results are still better than ECMP.

Chapter 7

Conclusion

Cloud computing provides high processing and storage capacity by interconnecting several servers for tenants to share. Thus, it enables efficient resource usage and also reduces the management and operational costs, which encourage the adoption of public clouds to host tenants. Nevertheless, it leads to new challenges, because tenants that share data-centers infrastructure have a myriad of applications with distinct purposes and requirements. The integration of such tenants and applications creates high workloads with highly variable traffic patterns. Additionally, the cloud computing infrastructure provider has to ensure the isolation and security between tenants and between tenants and the infrastructure as well. Another cloud computing data-center feature network topology is the path mulplicity between servers, which is often used to increase the failure tolerance. Multipath forwarding has also been employed to distribute the traffic and enhance the data center network performance. The conventional forwarding schemes such as Spanning Tree Protocol (STP) does not avail the available paths and deactivate several to avoid loops. Other multipath forwarding schemes such as Equal Cost MultiPath (ECMP) and Valiant Load Balancing (VLB) avoid this by randomizing the paths the flows take. Nonetheless, even if the regular distribution mechanisms operate to choose the flows paths, the path usage is imbalanced due to the flows shape and size. Hence, we need forwarding mechanisms that load balance the traffic based on network conditions.

This thesis main goal is to present the Two-Phase Multipath (TPM) forwarding scheme to improve the network performance of public clouds without modification to the tenants' network stack and to the data-center infrastructure. We improve the network performance using VLANs, which are commonly available in commercial off-the-shelf switches. TPM divides the forwarding into two phases: An offline multipath configuration phase that calculates available multipaths and installs them in the virtual switches; and a fast online multipath selection phase to distribute flows among the available paths. The multipath configuration is based on a genetic algorithm proposed to find disjoint VLAN trees connecting all ToR switches, and the online multipath selection phase uses heuristics based on network usage to select the path for a new flow. The path selection heuristic may use either a local or a global database to keep track of link usage and different heuristics to detect when flows finish.

We formulated the disjoint trees problem and proved NP-hard and we proposed a genetic algorithm to solve the problem. The genetic algorithm calculates the tree set with two objective functions: Minimize the size of the trees; and minimize the link reutilization by the trees. The genetic algorithm also calculates the number of trees is best suitable the given topology. The utilization of such algorithm does not restrict the topology that can be used with the scheme, and also allow that topology changes trigger recalculations. Thus, the genetic algorithm finds an optimal disjoint tree set for arbitrary topology adapting to the network topology conditions.

The second and online phase uses the network usage information to decide the paths of the upcoming flows. The design allows the several selection heuristics, both pessimistic and optimistic, but we see that the most precise least-used-links is likely the best. Also important is the location of the database. One single database for all flows in the data-center may be unfeasible for large data-center that have large amount of traffic, but it reasonable to smaller data-centers. Likewise, the local database potentially loses information, but even so it improves the performance over random selection schemes.

We demonstrate through simulations that the proposed TPM scheme has better performance than the conventional ECMP and STP schemes in high-workload scenarios. We simulated Fattree with 4-port switches topology, but also with the lager Fattree 6-port switches and 8-port switches topologies. We also simulated the forwarding scheme on cisco 3-tier topology and the random jellyfish topology. Finally, we simulated capabilities TPM such as the when consider the flows to start load balancing them and how detec the end of a flow. The results show that the TPM achieves more than 30% and 4x gains in Flow Completion Time when compared to ECMP and STP, respectively, using only local databases to store information about the network utilization. Therefore, we achieve a high-performance multipathing scheme for cloud computing data-centers without requiring any changes in tenants.

7.1 Future Work

The adoption of a two phase scheme allows each phase mechanisms evolve separately. The genetic algorithm finds an optimal tree set minimizing both tree size and link reutilization. Still, there is opportunity to test other optimization methods to obtain these results. We could use simulated annealing heuristic or even a model and optimize with greedy algorithms. We could, thus, model and compare the optimization methods to conclude the scenario each is best suited for. Meanwhile, the multipath selection mechanism is also modular, and we could test new selection heuristics. The presence of a database of network usage allows the selection mechanism acquire knowledge about the communication pattern of the workload. This knowledge could be used to predict the workload and schedule the flows accordingly. Besides, the information could be used in conjunction virtual machine (VM) migration mechanisms to place peer VMs in the same rack to unload the data-center network.

The scheme would also profit from a testbed implementation. One of the design choices of the Two-Phase Multipath (TPM) forwarding scheme is requiring only commercial off-the-shelf switches feature. Provided that the proposal is readily implementable, building a testbed and testing in real scenarios would bring a series of benefits. The testbed can analyze aspects such as real requirements and limitation of the proposal. One of the benefits is testing under real traffic data. Although the workload model approximates the traffic measurements, real application data can lead to different behavior, which potentially may benefit an information-based path selection mechanism. The testbed also contributes to measurement such as the number of arriving flows, database queries, flow pattern and size. We could gather information to tune forwarding scheme parameters regarding timer to consider the flow as large and migrate according to the best path and the monitoring interval to track flow end. Ultimately, these results can drive design choices of how many servers a controller can handle, and how many controllers one database can handle.

Bibliography

[1] CISCO. "Cisco Global Cloud Index: Forecast and Methodology, 2014-2019", 2015. Americas Headquarters.

[2] FERRAZ, L. H. G., MATTOS, D. M. F., DUARTE, O. C. M. B. "A Two-Phase Multipathing Scheme based on Genetic Algorithm for Data Center Networking". In: *Global Communications Conference (GLOBECOM), 2014 IEEE*, pp. 2270–2275, December 2014.

[3] FERRAZ, L. H. G., MATTOS, D. M. F., DUARTE, O. C. M. B. "Um Esquema de Multicaminhos com Algoritmos Genéticos para Redes de Centro de Dados". In: *SBRC 2014*, Florianópolis, SC, May 2014.

[4] OLIVEIRA, R. R., MARCON, D. S., BAYS, L. R., et al. "Opportunistic resilience embedding (ORE): Toward cost-efficient resilient virtual networks", *Computer Networks*, v. 89, pp. 59–77, October 2015.

[5] LAS-CASAS, P. H., GUEDES, D., ALMEIDA, J. M., et al. "SpaDeS: Detecting spammers at the source network", *Computer Networks*, v. 57, n. 2, pp. 526–539, February 2013.

[6] GREENBERG, A., HAMILTON, J. R., JAIN, N., et al. "VL2: A Scalable and Flexible Data Center Network", v. 54, n. 3, pp. 95–104, March 2011.

[7] NIRANJAN MYSORE, R., PAMBORIS, A., FARRINGTON, N., et al. "PortLand: A Scalable Fault-tolerant Layer 2 Data Center Network Fabric". In: *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication*, SIGCOMM '09, pp. 39–50. ACM, 2009.

[8] YAO, F., WU, J., VENKATARAMANI, G., et al. "A comparative analysis of data center network architectures". In: *IEEE ICC 2014*, pp. 3106–3111, June 2014.

[9] COUTO, R. S., SECCI, S., CAMPISTA, M. E. M., et al. "Latency Versus Survivability in Geo-Distributed Data Center Design". In: *Global Communi-*

cations Conference (GLOBECOM), 2014 IEEE, pp. 1102–1107, December 2014.

[10] PERLMAN, R. "An algorithm for distributed computation of a spanningtree in an extended LAN". In: *ACM SIGCOMM Computer Communication Review*, v. 15, pp. 44–53. ACM, 1985.

[11] ALIZADEH, M., GREENBERG, A., MALTZ, D. A., et al. "Data Center TCP (DCTCP)". In: *Proceedings of ACM SIGCOMM*, pp. 63–74. ACM, 2010.

[12] ALIZADEH, M., KABBANI, A., EDSALL, T., et al. "Less is More: Trading a Little Bandwidth for Ultra-low Latency in the Data Center". In: *Proceedings of the 9th USENIX NSDI Conference on NSDI*, pp. 19–19, 2012.

[13] RAICIU, C., BARRE, S., PLUNTKE, C., et al. "Improving Datacenter Performance and Robustness with Multipath TCP". In: *Proceedings of ACM SIGCOMM*, pp. 266–277. ACM, 2011.

[14] MATTOS, D. M. F., DUARTE, O. C. M. B. "XenFlow: Seamless Migration Primitive and Quality of Service for Virtual Networks". In: *Global Communications Conference (GLOBECOM), 2014 IEEE*, pp. 2326–2331, December 2014.

[15] AL-FARES, M., RADHAKRISHNAN, S., RAGHAVAN, B., et al. "Hedera: Dynamic flow scheduling for data center networks". In: *Proceedings of the 7th USENIX NSDI Conference on NSDI*, pp. 19–19. USENIX Association, 2010.

[16] IEEE. "Standard for Local and Metropolitan Area Networks: Virtual Bridges and Virtual Bridged Local Area Networks - Amendment 9: Shortest Path Bridging". March 2012. IEEE802.1aq.

[17] ZHANG, M., GHANWANI, A., MANRAL, V., et al. "Transparent Interconnection of Lots of Links (TRILL): Clarifications, Corrections, and Updates". RFC 7180 (Standards Track), May 2014.

[18] ALIZADEH, M., EDSALL, T., DHARMAPURIKAR, S., et al. "CONGA: Distributed Congestion-aware Load Balancing for Datacenters". In: *Proceedings of ACM SIGCOMM*, pp. 503–514, 2014.

[19] BENSON, T., AKELLA, A., MALTZ, D. A. "Network traffic characteristics of data centers in the wild". In: *Proceedings of the ACM SIGCOMM IMC*, pp. 267–280, 2010.

[20] BELABED, D., SECCI, S., PUJOLLE, G., et al. "Impact of Ethernet Multipath Routing on Data Center Network Consolidations". In: *Proc. of the 4th Int. Workshop on Data Center Performance (DCPerf'14), ICDCS*. IEEE, June 2014.

[21] BARI, M., BOUTABA, R., ESTEVES, R., et al. "Data Center Network Virtualization: A Survey", *Comm. Surveys Tutorials, IEEE*, v. 15, n. 2, pp. 909–928, 2013.

[22] AL-FARES, M., LOUKISSAS, A., VAHDAT, A. "A scalable, commodity data center network architecture". In: *Proceedings of ACM SIGCOMM*, pp. 63–74. ACM, 2008.

[23] MATTOS, D. M. F., FERRAZ, L. H. G., DUARTE, O. C. M. B. "Virtual Machine Migration". In: da Fonseca, N. L. S., Boutaba, R. (Eds.), *Cloud Services, Networking and Management*, Wiley-IEEE Press, Hoboken, EUA, April 2015.

[24] CARVALHO, H., FERNANDES, N., DUARTE, O. "Um Controlador Robusto de Acordos de Nível de Serviço para Redes Virtuais Baseado em Lógica Nebulosa". In: *SBRC 2011*, Campo Grande, MS, April 2011.

[25] JEYAKUMAR, V., ALIZADEH, M., MAZIERES, D., et al. "EyeQ: Practical network performance isolation for the multi-tenant cloud". In: *Proceedings of the 4th USENIX conference on Hot Topics in Cloud Ccomputing*. USENIX Association, 2012.

[26] JIANG, J., LAN, T., HA, S., et al. "Joint VM placement and routing for data center traffic engineering". In: *INFOCOM, 2012 Proceedings IEEE*, pp. 2876 –2880, 2012.

[27] DIAS, D. S., COSTA, L. H. M. "Online traffic-aware virtual machine placement in data center networks". In: *Global Information Infrastructure and Networking Symposium (GIIS), 2012*, pp. 1–8. IEEE, 2012.

[28] BODÍK, P., MENACHE, I., CHOWDHURY, M., et al. "Surviving failures in bandwidth-constrained datacenters". In: *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*, SIGCOMM '12, pp. 431–442. ACM, 2012.

[29] ZATS, D., DAS, T., MOHAN, P., et al. "DeTail: Reducing the Flow Completion Time Tail in Datacenter Networks". In: *Proceedings of the ACM*

SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, SIGCOMM'12, pp. 139–150. ACM, 2012.

[30] WILSON, C., BALLANI, H., KARAGIANNIS, T., et al. "Better Never than Late: Meeting Deadlines in Datacenter Networks". In: *Proceedings of the ACM SIGCOMM 2011 conference*, SIGCOMM '11, pp. 50–61. ACM, 2011.

[31] ALLAN, D., ASHWOOD-SMITH, P., BRAGG, N., et al. "Shortest Path Bridging: Efficient Control of Larger Ethernet Networks", *Communications Magazine, IEEE*, v. 48, n. 10, pp. 128–135, 2010.

[32] HAKIRI, A., GOKHALE, A., BERTHOU, P., et al. "Software-Defined Networking: Challenges and research opportunities for Future Internet", *Computer Networks*, v. 75, Part A, pp. 453–471, 2014.

[33] CURTIS, A. R., MOGUL, J. C., TOURRILHES, J., et al. "DevoFlow: Scaling Flow Management for High-performance Networks". In: *Proceedings of the ACM SIGCOMM 2011 Conference*, pp. 254–265. ACM, 2011.

[34] MUDIGONDA, J., YALAGANDULA, P., MOGUL, J., et al. "NetLord: A Scalable Multi-tenant Network Architecture for Virtualized Datacenters". In: *Proceedings of the ACM SIGCOMM 2011 Conference*, SIGCOMM '11, pp. 62–73. ACM, 2011.

[35] MUDIGONDA, J., YALAGANDULA, P., AL-FARES, M., et al. "SPAIN: COTS data-center Ethernet for Multipathing over Arbitrary Topologies". In: *Proceedings of the 7th USENIX NSDI Conference on NSDI*. USENIX Association, 2010.

[36] ROJAS, E., IBAÑEZ, G., GIMENEZ-GUZMAN, J. M., et al. "All-Path bridging: Path exploration protocols for data center and campus networks", *Computer Networks*, v. 79, pp. 120 – 132, 2015.

[37] PANDEY, S., CHOI, M.-J., WON, Y. J., et al. "SNMP-based enterprise IP network topology discovery", *International Journal of Network Management*, v. 21, n. 3, pp. 169–184, 2011.

[38] MCKEOWN, N., ANDERSON, T., BALAKRISHNAN, H., et al. "OpenFlow: Enabling Innovation in Campus Networks", *SIGCOMM Comput. Commun. Rev.*, v. 38, n. 2, pp. 69–74, March 2008.

[39] PFAFF, B., PETTIT, J., KOPONEN, T., et al. "Extending networking into the virtualization layer", *Proc. HotNets*, October 2009.

[40] KANDULA, S., KATABI, D., SINHA, S., et al. "Dynamic Load Balancing Without Packet Reordering", *SIGCOMM Comput. Commun. Rev.*, v. 37, n. 2, pp. 51–62, March 2007.

[41] MORAES, I. M., MATTOS, D. M., FERRAZ, L. H. G., et al. "FITS: A flexible virtual network testbed architecture", *Computer Networks Special issue on Future Internet Testbeds - Part {II}*, v. 63, n. 0, pp. 221 – 237, 2014.

[42] BARABASH, K., COHEN, R., HADAS, D., et al. "A case for overlays in DCN virtualization". In: *Proceedings of the 3rd Workshop on Data Center - Converged and Virtual Ethernet Switching*, DC-CaVES '11, pp. 30–37. ITCP, 2011.

[43] BOUTROS, S., SAJASSI, A., SALAM, S., et al. "VXLAN DCI Using EVPN". February 2013.

[44] SRIDHARAN, M., GREENBERG, A., VENKATARAMIAH, N., et al. "NVGRE: Network virtualization using generic routing encapsulation". September 2011.

[45] GAREY, M. R., JOHNSON, D. S. "Computers and intractability: a guide to the theory of NP-completeness. 1979", *San Francisco, LA: Freeman*, 1979.

[46] DOERR, B., DOERR, C. "Optimal Parameter Choices Through Self-Adjustment: Applying the 1/5-th Rule in Discrete Settings". GECCO'15, July 2015.

[47] CARDOSO, L. P., MATTOS, D. M. F., FERRAZ, L. H. G., et al. "An Efficient Energy-Aware Mechanism for Virtual Machine Migration". In: *2015 IEEE Global Information Infrastructure and Networking Symposium (GIIS)*, October 2015.

[48] COIT, D. W., SMITH, A. E. "Solving the redundancy allocation problem using a combined neural network/genetic algorithm approach", *Computers & Operations Research*, v. 23, n. 6, pp. 515 – 526, 1996.

[49] CISCO. "Cisco Data Center Infrastructure 2.5 Design Guide", 2007. Headquarters, Americas.

[50] SINGLA, A., HONG, C.-Y., POPA, L., et al. "Jellyfish: Networking Data Centers Randomly". In: *Proceedings of the 9th USENIX NSDI Conference on NSDI*, NSDI'12, 2012.