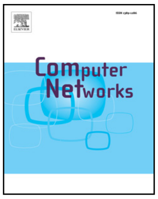
Computer Networks 112 (2017) 36–51

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet



# A high-performance Two-Phase Multipath scheme for data-center networks

Lyno Henrique G. Ferraz[a,c,1], Rafael Laufer[b], Diogo M.F. Mattos[a,c], Otto Carlos M.B. Duarte[a,*], Guy Pujolle[c]

[a] *Universidade Federal do Rio de Janeiro - GTA/POLI-COPPE/UFRJ, Rio de Janeiro, Brazil*
[b] *Bell Labs, Alcatel-Lucent, Holmdel, USA*
[c] *Laboratoire d'Informatique de Paris 6 - Sorbonne Universities, UPMC Univ Paris 06, Paris, France*

ARTICLE INFO

*Article history:*
Received 2 September 2015
Revised 3 May 2016
Accepted 21 September 2016
Available online 5 November 2016

*Keywords:*
Multipathing
Cloud
Data-center
SDN

ABSTRACT

Multipath forwarding has been recently proposed to improve utilization in data centers leveraged by its redundant network design. However, most multipath proposals require significant modifications to the tenants' network stack and therefore are only feasible in private clouds. In this paper, we propose the Two-Phase Multipath (TPM) forwarding scheme for public clouds. The proposal improves tenants' network throughput, whereas keeping unmodified network stack on tenants. Our 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 sets of paths, summarized into trees, during multipath configuration, and a local controller performs the multipath selection based on network usage. We analyze TPM for different workloads and topologies under several scenarios of usage database locations and update policies. The results show that our proposal yields up to 77% throughput gains over previously proposed approaches.

## 1. Introduction

In cloud computing, data centers share their infrastructure with several tenants having distinct application requirements [1]. This application diversity within data centers leads to multiple challenges for network design in terms of volume, predictability, and utilization. First, traffic between Top-of-Rack (ToR) switches is currently estimated to be 4x higher than incoming/outgoing traffic [2]. This high traffic volume requires specific network topologies for data centers in order to guarantee full bisection bandwidth and to provide fault tolerance [2–5]. Second, 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) [6], are usually 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) [7,8] to multipath congestion control [9]. These modifications are not an issue on private clouds, whose sole purpose is to provide services within a single domain. However, in infrastructure-as-a-service (IaaS) clouds, in which tenants rent virtual machines and have complete control of their network stacks [10], these solutions are not feasible. 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 data-centers [2,3,11–13]. Network switches supporting ECMP find multiple paths with the same cost and

[*] Corresponding author.
*E-mail addresses:* lyno@gta.ufrj.br (L.H.G. Ferraz), rafael.laufer@alcatel-lucent.com (R. Laufer), diogo@gta.ufrj.br (D.M.F. Mattos), otto@gta.ufrj.br (O.C.M.B. Duarte), Guy.Pujolle@lip6.fr (G. Pujolle).

[1] Grupo de Teleinformática e Automação - GTA Universidade Federal do Rio de Janeiro (UFRJ) P.O. Box: 68504 - ZIP Code 21945-972, Ilha do Fundão, Rio de Janeiro, RJ, Brasil phone: +55 21 3938–8635.

http://dx.doi.org/10.1016/j.comnet.2016.09.025
1389-1286/© 2016 Published by Elsevier B.V.

choose between them applying a hash function to 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 prevent 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 [14]. 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. Moreover, ECMP as well as VLB choose the next hop without taking into account the utilization of the links.

In this paper, we propose a Two-Phase Multipath (TPM) forwarding scheme that calculates multiple trees on the network, and keeps track of the usage of all trees. Our proposal is based on the 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.
- **Robustness to path and topology asymmetries:** 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.
- **Incremental deployment:** TPM can be deployed in only part of the data center, and work with other segments of the data center.

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 switches when the network is not yet operational. It creates several 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. To find these trees, the multipath configuration phase uses network topology information to reduce path lengths and increase link usage. In particular, we propose and formulate a genetic algorithm to find an optimal set of trees with disjoint links. Finding the optimized configuration of a datacenter network is a complex task and, for some datacenter topologies, it is an unstructured problem. Thus, it needs a back-end support for optimizing the VLAN tree configuration and to assure that all configured VLAN trees are correct [15]. The genetic algorithm formulation answers to these needs. Multipath selection is the online phase that chooses the best path for a new flow. The selection is based on path utilization in order to select the least used path.

The key idea of the proposal is to keep track of the usage of multiple disjoint paths on the network. We summarize a set of paths that shares links into a tree and, thus, the problem of calculating disjoint paths for all pairs of hosts is reduced to calculate disjoint trees that contain all hosts. We optimize the generation of trees, maximizing the diversity of links used by different trees through a genetic algorithm. The proposed genetic algorithm model achieves up to 100% of diversity between the calculated trees, i.e., the calculated trees do not share any link. Moreover, we develop a discrete-event simulator at flow-level granularity to model the data center to evaluate our proposed approaches for selecting a tree to forward a new flow. We test several scenarios inspired in realistic traffic workloads [16]. The results show that TPM always performs better than the traditional forwarding schemes with gains up to 77%.

The rest of the paper is structured as follows. Section 2 presents the architecture of TPM and our design choices. Section 3 describes the offline multipath configuration phase and Section 4 presents the online multipath selection phase. We simulate different scenarios and topologies and present the results in Section 5. We present the related work in Section 6 and conclusions in Section 7.

## 2. Architecture of the Two-Phase Multipath scheme

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' protocol stack. As previously explained, this is performed in two phases: The multipath configuration phase and the multipath selection phase.

TPM requires two types of devices to manage the entire network: a global logically centralized manager responsible for the multipath configuration phase, and local controllers responsible for the multipath selection phase. 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. The path computation is performed before the network becomes operational and also upon any topology change. To obtain the topology, the global manager may use either the Simple Network Management Protocol (SNMP) for topology discovery [17] or OpenFlow [18]. 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 path availability 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 a set of trees that share the lowest number of links, we propose a genetic algorithm in Section 3.

We assume that each physical machine in a rack has a virtual switch [19,20] 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 [10]. 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 Section 4. Once the path (and its corresponding VLAN) is selected, the local controller installs an OpenFlow rule on the virtual switch to handle future packets of this flow. Each subsequent packet then receives the assigned VLAN tag and does not require contacting the local controller again.

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 backwards 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.

## 3. Multipath configuration

The multipath configuration phase is responsible for calculating and installing the VLAN trees in the network devices. The challenge here is to provide enough path diversity for each pair of ToR switches to improve network utilization. However, it is not known *a priori* how many trees (and therefore paths) TPM 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 section, we propose a genetic algorithm to find the best tree set using two objective functions. Defining the size of a tree as the number of its nodes, then one objective function minimizes the tree size in order to avoid long paths; and the other objective function minimizes the link reutilization by the trees to create disjoint trees. The algorithm optimize both objective functions considering Pareto dominance group and number of individuals in the population. Considering both objective functions, the algorithm searches for the set of minimum trees that reaches the most of diverse paths on the network. 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 performed offline, and at each long-term modification of the network, such as the installation of new devices. Link failures do not trigger the calculation and installation of new trees; instead, they are detected by local controllers that blacklist the affected paths.

Genetic algorithms (GAs) are a stochastic optimal search technique inspired by the natural selection process during biological evolution. Its methodology consists of: (i) modeling each feasible solution as an individual, in which multiple individuals compose a population, i.e., a set of trees in our case; (ii) selecting an initial random population; (iii) calculating the phenotype of each individual, i.e. the evaluation of a solution; (iv) selecting parents according to their phenotype; (v) recombining the selected parents to form new children; (vi) mutating individuals to form new children; (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 [21]. The selection of parents and children forces the GA to prioritize the best solutions. The recombination, step (v), creates new children taking the qualities of the parents, and the mutation, step (vi), drastically changes the individuals to avoid local optima. Next, we describe each of these steps in detail.

(i) **Individual model:** In our application, each individual is a tree interconnecting all ToR switches[2]. 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 $I_k$. 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.

(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. Fig. 1 depicts the proposed tree creation procedure. The procedure starts with a random switch, which forms the first subtree, shown in Fig. 1(a). For ease of presentation, we drop the superscript index $k$ and represent the switch vector simply as $\mathbf{s} = \begin{bmatrix} s_1 \end{bmatrix}$. 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. Fig. 1(b)–(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 shown in Fig. 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 the switches are in different subtrees, the new switch connects to all of them and forms a single subtree, as shown in Fig. 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 shown in Fig. 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.,

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

If we define the individual objective functions

$$f_1(I) = -|I| \tag{2}$$

and

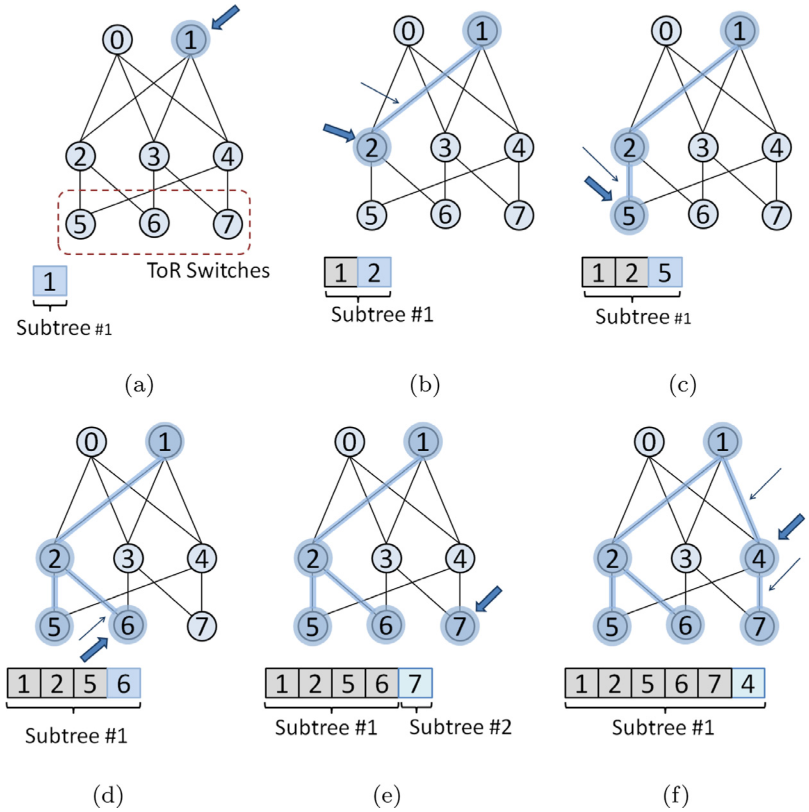
$$f_2(I) = \sum_{e \in I} \frac{1}{\sum_{I' \in P} x(e, I')}, \tag{3}$$

then, the individual phenotype 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 Eq. (2) 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 Eq. (3) provides the sum of the count of each edge $e \in I$, considering all individuals in the population $P$. It has a higher value if $I$ counts links that are not counted 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 $F(I_j) \succ F(I_k)$, if $f_i(I_j) \geq f_i(I_k)$, for $i \in \{1, 2\}$ and if $\exists i$ such that $f_i(I_j) > f_i(I_k)$, where $F(I_j)$ denotes Pareto evaluation of all objective functions for an individual $I_j$.

(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

[2] For the sake of simplicity, we name switch all network devices that forward packets on the data-center network. When we are referring to a specific type of switch, we always highlight it as core, aggregation or top-of-the-rack (ToR) switch.



(a) (b) (c)

(d) (e) (f)

**Fig. 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 **s** is shown 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 **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 **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.

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 shown in step (v).

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

$$p(I) = \frac{\tilde{p}(I)}{\sum_{I' \in P} \tilde{p}(I')}. \tag{4}$$

(v) **Recombination of parents:** Each pair of parents sampled in step (iv) must be recombined to pass their genotype 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 parents $I_a$ and $I_b$, respectively. The recombination procedure starts by picking a random position $r \in [1, \min(|I_a|, |I_b|)]$ of the parents' genotypes, and splitting them into a radical and a suffix. Fig. 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 descendants. 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. Fig. 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 shown in Fig. 2(d). Second, after processing the new suffix, ToR switches not connected to the tree are added in random order, as shown in Fig. 2(f).

(vi) **Mutation of descendants:** After the recombination of parents, each descendant may mutate and generate another individual. This occurs with probability $p_m = 0.10$. This parameter is selected based on the convergence analysis of our genetic algorithm [22] and it is a tradeoff between search space coverage and avoiding local minima. 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

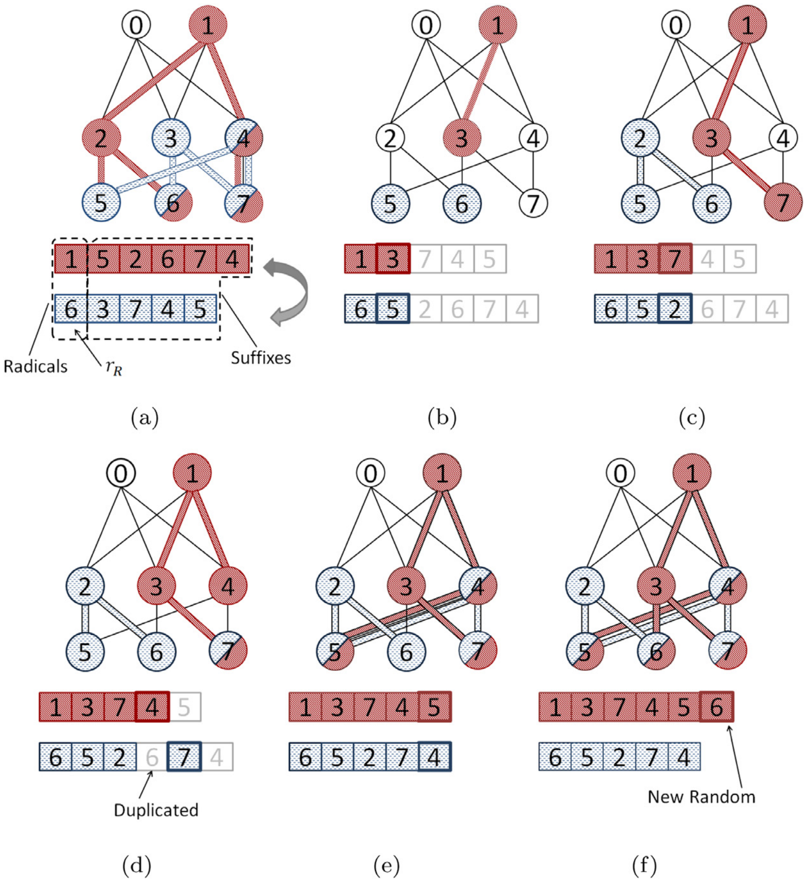


**Fig. 2.** Recombination of two parents to generate two descendants. The procedure is similar to the tree creation procedure, except that new switches are selected from the suffix of the other parent. (a) The two trees are shown with their respective genotypes. The random integer $r = 1$ is selected and separates the genotypes in radicals and suffixes, which will be exchanged. (b)–(f) The two descendants are generated in parallel. The dark shadowed nodes belong to the first descendant and the light shadowed nodes belong to second descendant.

switch that is not present in at the end is included in random order. We present the mutation procedure in Fig. 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 $\sigma$. 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

$$G(P) = \left| \bigcup_{i=1}^{n} I_i \right|. \tag{5}$$

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 $\sigma$, we use Rechenberg one-fifth success rule [23]. This rule introduces a self-adaptation strategy to genetic algorithms in order to not constrained in local minima. This mechanism changes the parameters of the genetic algorithm based on the feedback from the search. As we model our population size as a normal distribution, using the Rechenberg one-fifth success rule, we let the size of the population be self-adapted according to the feedback of the search. We observe a certain number of generations and compute the fraction $q$ of successful generations. Let $c \in [0.817, 1]$ be a constant, then $\sigma$ is updated as follows

$$\sigma = \begin{cases} \sigma / c, & q > 1/5 \\ \sigma \cdot c, & q < 1/5 \\ \sigma, & q = 1/5. \end{cases} \tag{6}$$

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 $\sigma$ 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.

To evaluate the algorithm performance, we use the diversity index, defined in the Eq. (5), normalized by the total number of links on the network $|E|$. The normalized diversity index is calculated between the trees, thus the closer the index is to 1, the more diverse are the links used on the calculated trees. Moreover, we also evaluate the Number of VLAN trees that identifies how many trees are selected into each generation. Fig. 4 shows our results of three topologies where we run our genetic algorithm. We identify

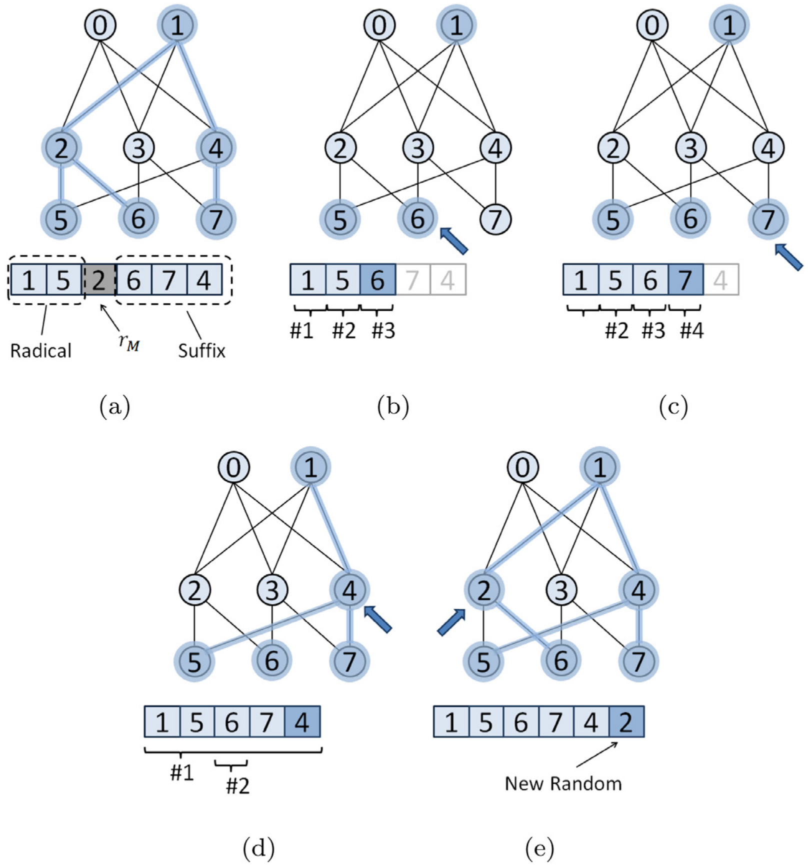


**Fig. 3.** Mutation of an individual to form a new individual. (a) The switch vector of the original descendant and the random integer $r$ indexing the switch to be removed. (b)–(e) The formation of the new mutated individual by adding the switches in the suffix one at a time. Although the mutated individual has the same switch set, the two trees are different due to the order of the switches in the vector.

two different populations, the population that is being evaluated in each generation, and the best population that the algorithm has already found. Running our algorithm over a Fattree topology, composed with 4-port switches, it converges into less than 25 generations, shown in Fig. 4(a). By its turns, when running the algorithm over a Cisco 3-Tiers topology, it requires up to 3000 generations to reach the optimal value for population entropy. Besides, when considering a randomized topology, in our experiments we considered a Jellyfish topology with 20 4-port switches, our algorithm reaches a sub-optimal solution after 500 generation. It is worth noting that the algorithm always finds sub-optimal solution in a short period of time, although after some generations, the solution is enhanced. Fig. 4 also highlights that the proposed genetic algorithm achieves the two objective function as it finds the minimal set of VLAN tree that generates the maximum diversity between all used links on the network.

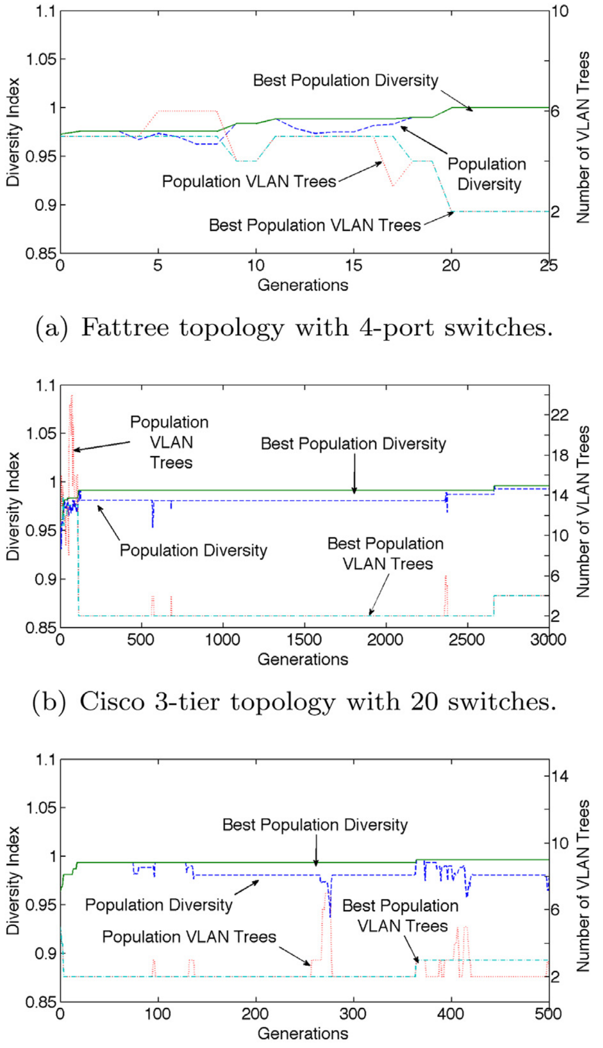
The fast convergence and the quality of the solution found by our genetic algorithm are mainly due to the proposed method of generating the trees and representing them as a 2-dimension vector, and also due to the usage of the Rechenberg rule to grow and to shrink the population size according to the success rate. The proposed tree generation provides a good initial solution to the genetic algorithm. Moreover, the initial solution has already a randomized pattern thanks to generation phase. As the mutation and recombination operators keep the same principles of the tree generation method, operating over the trees keeps a randomized link selection method. Therefore, it enables the genetic algorithm to explore a bigger search space in a few generations. The Rechenberg one-fifth rule helps to define the size of the search space for each interaction. As the best population is a subset of an actual population, the population size may grow, but the number of selected individuals for the best population is just the ones that contribute to enhancing the diversity index.

## 4. Multipath selection

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. 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 4.2) containing the active flows and their corresponding paths to select a path. Once this is done, the controller installs a rule, such as OpenFlow, 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 proper virtual machine.

Each local controller manages the flows originated at in a single rack to determine when a flow finishes, the local controller frequently queries the flow statistics of the virtual switches (cf. Section 4.3). To detect and cope with link and devices failures in paths, the local controller senses when a flow stops to transmit, which is considered a path failure in that tree and the flow is rescheduled to use a different path.

The proposed TPM scheme assumes two different control/management levels in the network. First, a centralized manager is responsible for calculating and installing all trees in the network devices. Second, each edge host has its own local controller



(a) Fattree topology with 4-port switches.

(b) Cisco 3-tier topology with 20 switches.

(c) Jellyfish topology with 20 4-port switches.

**Fig. 4.** Performance evaluation of our two objective functions genetic algorithm for three topologies. The `Population Diversity` is the diversity index of the current population, given by Eq (5), while `Best Population Diversity` is the same index for the best subset of the population that the algorithm has already calculated. `Population VLAN Trees` and `Best Population VLAN Trees` stand for the number of VLAN Trees used in each population. a) The proposed algorithm finds two VLAN trees that reach 100% of diversity between links in a Fattree topology. b) The algorithm finds the optimal solution, 100% of diversity between links, with 2 VLAN trees, in a Cisco 3-tier topology. c) The best solution for a Jellyfish topology is a near-optimal solution with 3 VLANs.

for selecting the least overloaded tree to forward each new flow. The global manager installs the trees and VLANs before the network becomes operational. The local controller then controls the virtual switch of its own host to select the best VLAN tree. This two-level approach for controlling and managing the network enables the local controller to always know *a priori* the end-to-end path of a packet, even though it does not control any switch other than its local virtual switch.

### 4.1. Selection heuristics

To select a path for each new flow, the local controller selects the path with the least used links (LUL) [1]. 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. After the selection, the link usage $u(e)$ for all links in the path are incremented. Similarly, when the flow finishes, all link costs of the path are decremented.

### 4.2. Database location

We consider two extreme cases for the network usage database location. 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 predict 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 communication 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.

### 4.3. 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 in order to free network resources for new flows. The aforementioned heuristics determines when the flow finishes at the cost of the local controller constantly monitoring the flows. 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 end (NO-END). Although simplistic, this approach has some knowledge of the congestion, 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 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 use a fixed end timeout value for all flows.

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, which can be costly.

## 5. Simulations results

The simulations compare the well-known multipath schemes Spanning Tree Protocol (STP) and Equal Cost MultiPath (ECMP) with the proposed Two-Phase Multipath (TPM) forwarding scheme.

### 5.1. Simulation model

In the simulations, all servers send and receive flows through a ToR switch. Thus, we only consider ToR switches as source and destination, since the path from the ToR switch to the physical machine is unique. 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 workload model consists of two random variables, the flow size $X_s$ and the flow inter-arrival interval $X_t$. Larger flows require a longer time to be transmitted and, thus, they have a higher probability of sharing the link bandwidth with other flows. The flow size $X_s$ follows a lognormal distribution $\ln N(\mu_s, \sigma_s)$. We choose $\mu_s = 7$ and $\sigma_s = 2.8$, such that the cumulative density function (CDF) presents the following values $F(x) = \{\approx 0.5|x = 1000, \approx 0.95|x = 100,000\}$, according to empirical data center measures published by Benson et al. [16]. The second random variable characterizing the workload, the inter-arrival time $X_t$, directly affects the data center load. Smaller inter-arrival times increase the flow arrival rate and, consequently, increase the network link usage. The inter-arrival time (in microseconds) also follows a lognormal distribution $\ln N(\mu_t, \sigma_t)$ with $\sigma_t = 2$ and $\mu_t$ varying to decrease the workload. We chose different values for $\mu_t$ in order to have the median $t_a$ within the set {1, 2, 5, 10, 15, 20, 30}, in milliseconds. We use the median instead of the mean to avoid dependency on the standard deviation $\sigma_t$. The inter-arrival time forms a heavy tailed distribution, and the $t_a$ values are chosen to have a similar distribution to the empirical measures of Benson et al. [16]. As $t_a$ increases, the expected time between flow arrivals also increases and the load decreases.

We compare our proposed scheme with the Spanning Tree Protocol (STP) and with Equal Cost MultiPath (ECMP), because these forwarding schemes are widely adopted and do not change the tenants' network stack. Moreover, Ethernet networks, mainly for datacenters, and for MPLS relies on ECMP for multipath forwarding [24].

**Spanning Tree Protocol (STP):** the switches create a single tree to forward all traffic. Thus, there is a single path between each pair of ToR switches.

**Equal Cost MultiPath (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 [12]. We model ECMP as a uniform random variable to select paths and the costs of the paths are the hop count from source to destination.

### 5.2. Fattree 4 topology

We first run simulations using the Fattree topology with 4-port switches [25], which presents four different paths between any pair of pods, and two different paths between any pair of ToR switches in the same pod, as depicted in Fig. 5. The destinations of the flows are uniformly selected to evenly distribute the traffic across the data center.

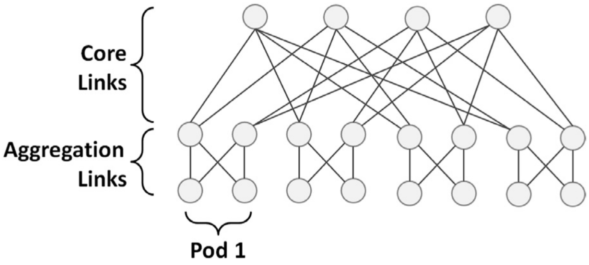


**Fig. 5.** Fattree topology with four-port switches used in our simulations. There are four different paths to any ToR switch in another pod, and two different paths to a switch in the same pod.

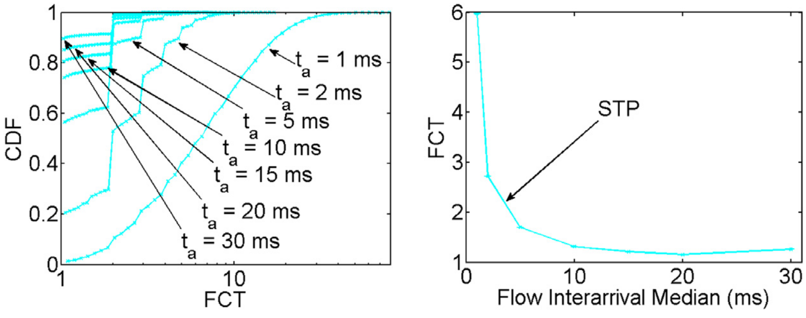
We present the results using the flow completion time (FCT), a metric that indicates the quality of the forwarding scheme of the data center network. For a particular forwarding scheme, the FCT is the flow duration when multiple flows are present normalized by the flow duration when there is no other 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. Fig. 6(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 shown in Fig. 6(b), which shows the expected FCT as a function of the inter-arrival time. The results reveal a low flow completion time because of link bottlenecks due to spanning tree algorithm. Nevertheless, Fig. 7(a) shows that the network still have forwarding resources that are not utilized or underutilized[3].

To understand the STP performance, we investigate the core and aggregation link usage. Fig. 7(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 greater, 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. 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. Moreover, the core links in the tree have active flows only for 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 flow completion time is also caused by the high the number of active flows sharing the same link. We see from Fig. 7(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. Fig. 7(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.

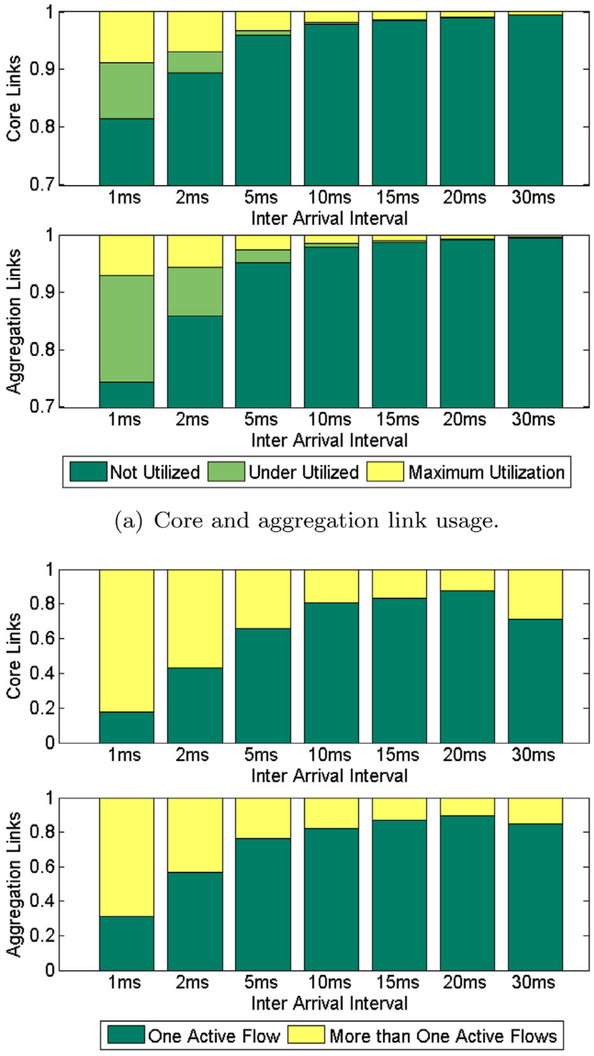
We now compare the performance of the same Fattree topology for STP, ECMP, and the proposed TPM using the least used links (LUL) heuristic for path selection and LOCAL database location. We also present results using a GLOBAL database location with entire knowledge of the network just as a performance baseline, but do

[3] A link that is not utilized carries no traffic during the simulation period; a link that is under-utilized is a link that does not use all of its capacity during the simulation.



(a) (b)

**Fig. 6.** 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 10× 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 in links.



(a) Core and aggregation link usage.

(b) Number of active flows per link.

**Fig. 7.** (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. At higher workloads, almost 80% of the utilized resources is overloaded, even when more than 70% of the total network resources is idle.

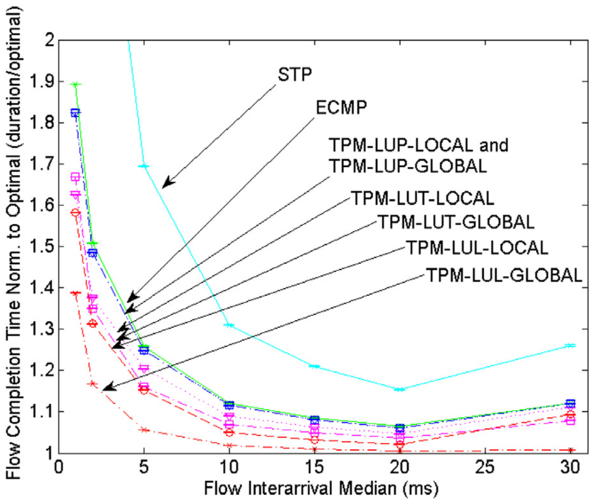


**Fig. 8.** Expected flow completion time under different workload scenarios ($t_a$ values) for the multipath selection heuristics for Spanning Tree Protocol (STP), Equal Cost MultiPath Protocol (ECMP), and the proposed Two-Phase Multipath (TPM) with different heuristics Least Used Tree (LUT), Least Used Path (LUP), and Least Used Links (LUL) with global (GLOBAL) and local (LOCAL) database locations for Fattree 4 ports.

not consider the communication nor the query/update overheads of this approach. In addition, we also present results using previously proposed path selection heuristics, namely, least used tree (LUT) and least used path (LUP) [1]. In 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 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.

Fig. 8 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 4.4× 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. It is worth noting that the proposed heuristics introduces a trade-off between knowledge about the flow behavior and the gains on the flow completion time. The more knowledge the local controller has about the network state, the greater is the TPM gain. Nevertheless, even when considering just the local knowledge, the proposed scheme achieves a reduction of up to 20% of the FCT over ECMP and 3.4× over STP. This result shows that the calculation of the trees in TPM installs substantially disjoint trees that are able to significantly reduce the FCT even without global knowledge.

Fig. 8 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 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 the performance.

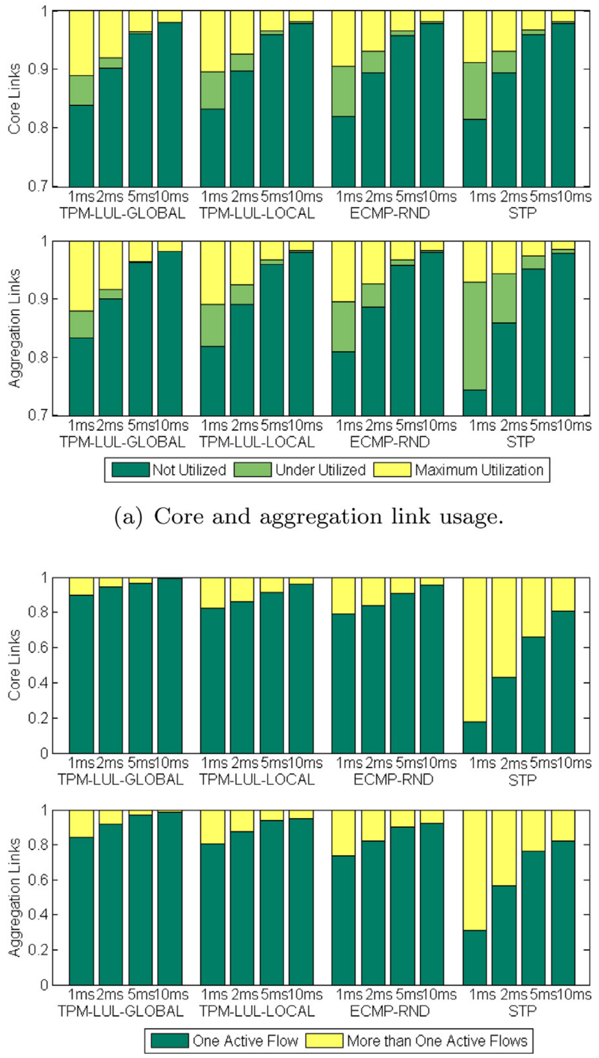
To further investigate the performance of the schemes, we again measure the core and aggregate link usage as well as the number of active flows per link. Fig. 9(a) shows the utilization of core and aggregation links for 1 ms ≤ $t_a$ ≤ 10 ms.

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. Fig. 9(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 creates 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. These results highlight the efficiency of the proposed TPM selection phase, because, even when ECMP and TPM calculate the same trees, the TPM selection phase reduces the underutilization of the network resources, Actually, the proposed TPM selection phase chooses the least congested tree instead of randomizing the choice because it knows the state of the network.

**Small and large flows relationship:**

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 10 MB 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 $\sigma_s$ in 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%. We keep $t_a$ fixed at 5 ms for different $\sigma_s$ values to provide the same load across experiments.

As $\sigma_s$ increases, there is more variance on the flow size and therefore we have more flows sharing the same links, which severely impacts FCT as shown by Fig. 10. The FCT can be up



(a) Core and aggregation link usage.

(b) Number of active flows per link.

**Fig. 9.** (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. For the same workloads, the proposed multipath scheme allocates for a shorter period the network resources, because flows are completed earlier as links are used up to 90% of the time by just one flow.

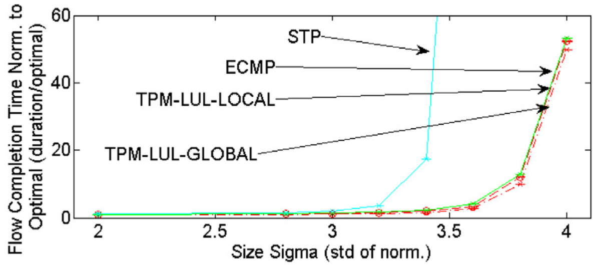
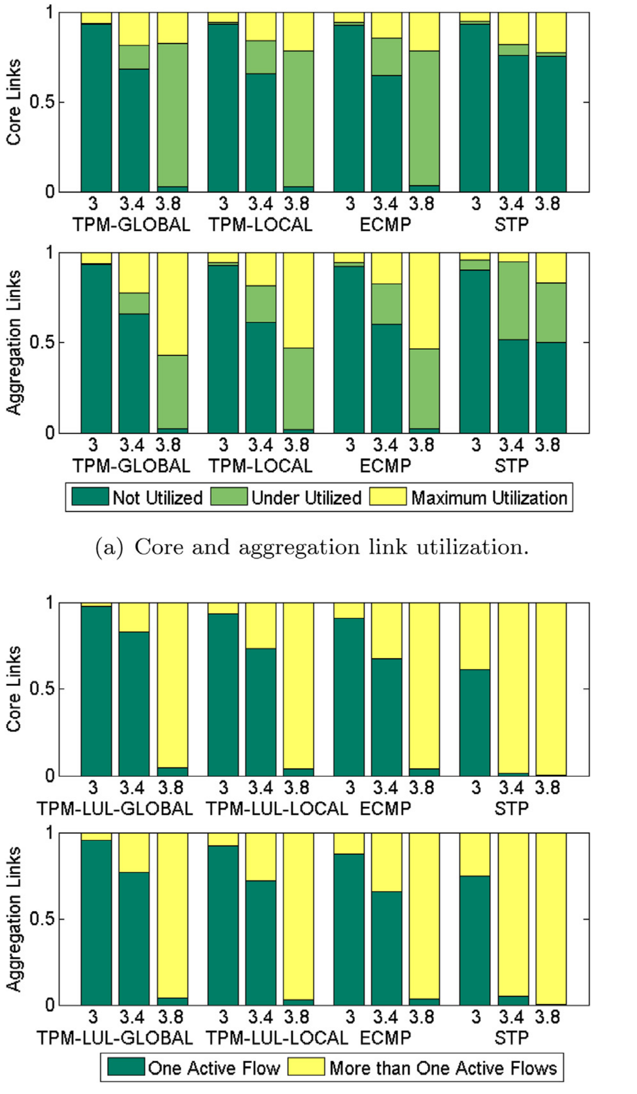


**Fig. 10.** Expected FCT varying the proportion of small and large flows by changing the standard deviation $\sigma_s$ of the lognormal distribution of flow sizes, with $\mu_s = 7$ and $t_a = 5$ ms.



(a) Core and aggregation link utilization.

(b) Number of active flows per link.

**Fig. 11.** Core and aggregation link utilization of $\sigma_s \in [2, 4]$ for ECMP and TPM for both GLOBAL and LOCAL database placements.

50 times worse due to the long-lived flows. Fig. 11 shows both the utilization and the number of flows per aggregation and core link. 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 shown in Fig. 11. Although the relative increase of large to the small degrades the performance, TPM can still improve the performance when compared to ECMP.

### 5.3. Larger fattree topologies

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. Figs. 12 and 13 show the FCT achieved in both topologies. Considering the Fattree with 4-port switches, Fattree with 6-port and 8-port switches reduces the overall FCT. Although there are more servers (and ToR switches) in these larger topologies, there are also more available paths to transmit the additional workload. This difference is visible when comparing Figs. 12 and 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 and, as a consequence, LOCAL database approaches do not perform as well as a GLOBAL database. The trend is that ECMP has a similar behavior to LOCAL database approaches as the topology gets larger. This trend is explained by the fact that LOCAL database approaches only take into consideration the knowledge of the creation of flows in a single ToR swtich and, thus, ignores the reutilization of links in the core of the network. In turn, ECMP randomizes the path selection of at each switch on the network, avoiding flows of different ToR switches to deliberately share the same links on the core. This results from the local information being less relevant as the topology enlarges.

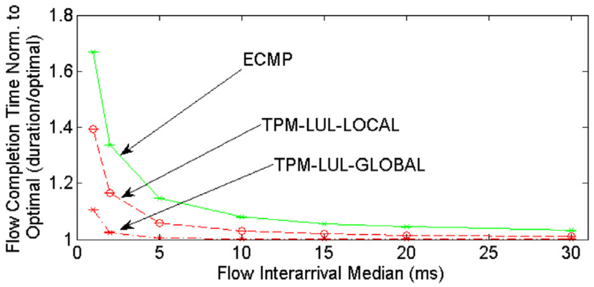


**Fig. 12.** Expected FCT for Fattree with 6-port switches.

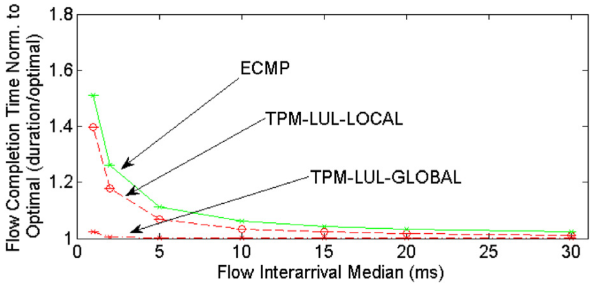


**Fig. 13.** Expected FCT for Fattree with 8-port switches.

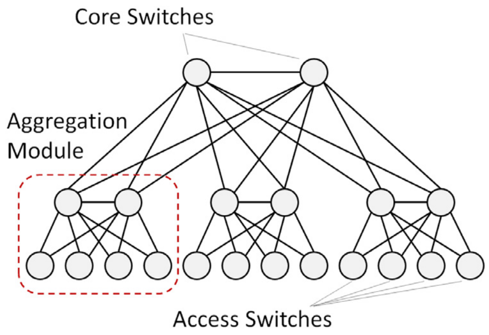


**Fig. 14.** The Cisco 3-tier topology, composed of the same 20 switches as in the previous Fattree 4 topology. The main goal of this topology is the vertical communication.

### 5.4. Cisco 3-tier topology

We also used Cisco 3-tier topology [26] having three aggregation modules, each with four access switches, as shown as Fig. 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

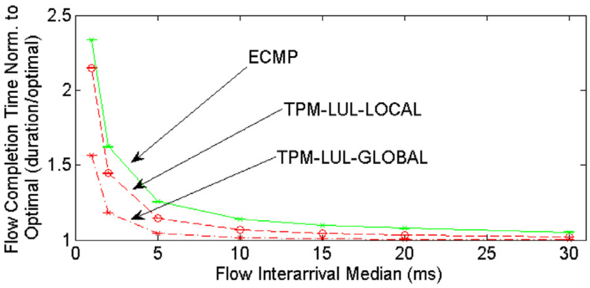


**Fig. 15.** Expected FCT for the Cisco 3-tier topology. Although under high horizontal communication the overall performance degrades when compared to Fattree 4, the topology benefits by using TPM forwarding scheme.

paths are mostly used for fault tolerance in the Cisco 3-tier topology, we allowed all links to be equally selected for forwarding.

Fig. 15 shows the expected FCT for the Cisco 3-tier topology. The performance of this topology is worse than the Fattree 4 topology 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 benefits significantly with TPM, achieving an FCT reduction of 33% under the highest workload when compared to ECMP. When compared to the Fattree 4 topology, the overall performance of the Cisco 3-tier topology degrades because the Cisco topology benefits the communication of hosts belonging to the same aggregation module. Nevertheless, when using TPM, all traffic always traverses one of the previously calculated trees. Sending traffic over a tree favors vertical communication instead of horizontal communication. As a result, TPM is more suitable for vertical communication topologies, such as Fattree, because it introduces more redundant links on the network core than on the aggregation layers. Therefore, for highly horizontal connected topologies, such as Cisco 3-tier, TPM may underutilize the links horizontally connecting aggregation switches as this links are not mainly selected to the calculated VLAN trees.

### *5.5. Jellyfish topology*

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. [27] 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. Fig. 16 shows the Jellyfish topology used in our simulations, which has eight 4-port ToR switches that use two of those ports to connect to servers.

Fig. 17 shows the expected FCT for the 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.

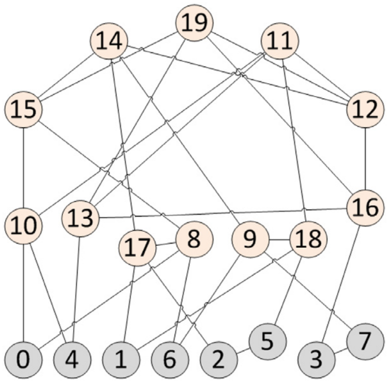


**Fig. 16.** Randomly generated Jellyfish topology with 20 4-port switches. This topology is similar to the Fattree 4 topology, thus 8 switches are ToR with two servers each.

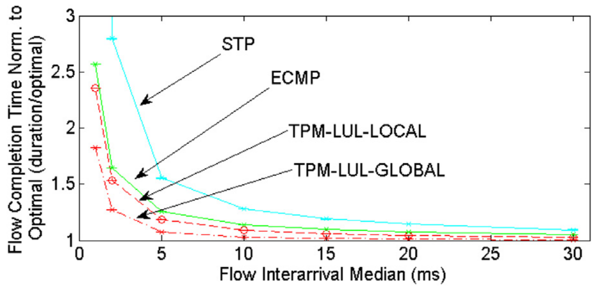


**Fig. 17.** Expected FCT for the Jellyfish topology. Connections between two ToR switches benefit communication between the two racks, but degrade communication with other ToR switches. Still, TPM improves FCT when compared to ECMP.

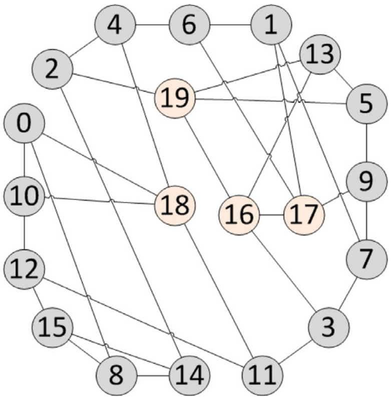


**Fig. 18.** Randomly generated Jellyfish topology with 20 4-port switches, out of which 16 are ToR switches. This topology also uses 20 switches, but allows servers to communicate with fewer hops, increasing the overall end-to-end throughput.

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. Fig. 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. As each ToR has half of the number of servers, we halved the inter-arrival time to fairly compare it to the previous Jellyfish topology.

Fig. 19 shows the expected FCT for the second Jellyfish topology with 16 ToR switches. As we can see, each ToR switch uses

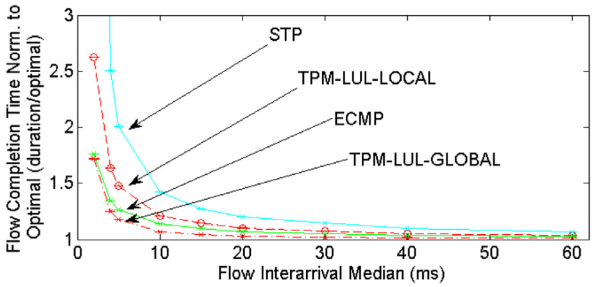


**Fig. 19.** Expected FCT for the Jellyfish topology with 16 ToR switches. The presence of more ToR ports for communication between other switches improves the overall performance.

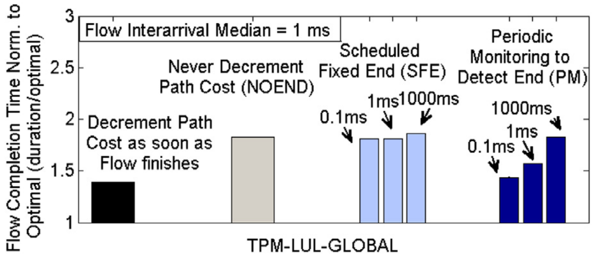


**Fig. 20.** Expected FCT for different flow tracking policies for TPM with a GLOBAL database in Fattree 4 topology.

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. Therefore, a less structured topology, such as Jellyfish, enhances the ability of growing the datacenter infrastructure on demand, but introduces a trade-off between multipath performance and flexibility. Because of the randomized and non-structured pattern of the Jellyfish topology, all structured multipath schemes, such as TPM, suffer from a lower performance gain when compared to ECMP that randomizes the path selection and is able to use shortest paths between ToR switches. Nevertheless, our results show that, even in the worst scenario for TPM, our proposal with GLOBAL database outperforms ECMP.

### 5.6. 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). Fig. 20 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.

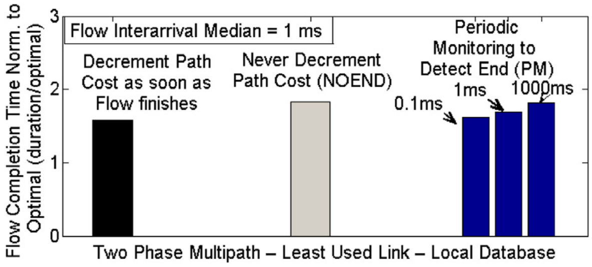


**Fig. 21.** Expected FCT for different flow tracking policies for TPM with a LOCAL database in Fattree 4 topology.

Fig. 21 presents the equivalent results using the proposed LOCAL database placement. The results show that a short monitoring period does not provide too much advantage under the LOCAL database placement. In particular, this occurs because the information is only local and a higher polling rate is not enough to improve performance. The results are similar to the NOEND policy.

The NOEND policy acts as a round-robin policy between all paths. Considering both GLOBAL and LOCAL database locations, we identify that the NOEND policy is the one that provides the best tradeoff between monitoring overload and networking performance. The scheduled flow end approach (SFE) depends on monitoring the network to profile the expected flow ending period. Indeed, the NOEND policy is the only policy that introduces no monitoring overload and reaches a FCT comparable to the most frequent monitoring policy (0.1 ms) in the LOCAL approach.

## 6. Related work

Nakibly et al. argue that splitting network flows over multiple paths is a complex problem that can be divided into two problems: Decomposition with Minimum forwarding Overhead (DMO) and Routing with Minimum forwarding Overhead (RMO) [24]. Nakibly et al. prove that both problems are NP-hard and, thus, they propose two heuristics for solving them. The DMO problem may be understood as a generalization of the selection phase of our Two-Phase Multipath scheme, because both DMO and the selection phase deal with selecting a disjoint path for each flow on the network. Besides, the RMO problem is comparable to the generation phase of our proposal. The key idea of RMO is to minimize the resource usage when configuring multiple paths on the network, as our tree generation approach does. Our genetic algorithm proposal deals with these two NP-hard problems when it calculates its multiple-objective cost function.

A few proposals randomly select one of the multiple paths in the data centers to distribute the network traffic [2,25,28]. Al-Fares et al. [25] 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. [2] 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. 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. Mudigonda et al. [28] introduces NetLord, a multi-tenant network architecture that encapsulates tenants' Layer-2 packets to provide full address-space virtualization. The forwarding mechanism uses a so-called smart path assignment in networks (SPAIN) [29] to distribute traffic among multiple paths. Servers run an online algorithm to test the connectivity of the destination and randomly select a path to send each 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. The 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.

Other multipath forwarding schemes were proposed in the context of software-defined networking (SDN) to manage and distribute data center traffic [30]. Al-Fares et al. [11] 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. [31] 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. Our approach distributes the path selection to the virtual switches at the physical machines to avoid such constraints.

Taking into consideration the traffic engineering on Software-Defined Networks, Luo et al. propose the Adaptive Dynamic Multi-Path Computation Framework (ADMPCF) [32]. The key idea is to develop a framework, for networks with centralized control that enables the network to rapidly respond to changes on the network topology and workload. The framework introduces datamining techniques to identify flow patterns and it also calculates online optimizations of multiple paths. The key challenges of this proposal are the large computing overload and storage constrains on the network controller. Our proposal avoids the overload on the network controller by using a pre-calculated set of forwarding trees instead of calculating them online. Moreover, the TPM proposal does not require a centralized controller online on the network. The selection phase is supposed to be performed by local controllers with limited constrains on computing and storage power.

Alizadeh et al. [14] propose a distributed global congestion-aware balancing (CONGA) 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 significant modifications to the data-center fabric, which could be costly. Rojas et al. propose All-Path, a routing paradigm that uses reactive approach to learn the paths in data-center, campus and enterprise networks [33]. 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 they store the port from which they received the first copy of the ARP request. The ARP reply follows the learnt path, allowing switches store the path to the destination. Thus, the approach 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 virtual switches at the physical machines.

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 physical machines.

## 7. Conclusion

The main goal of the proposed two-phase multipath (TPM) scheme is to improve the network performance of cloud computing data centers with no modification to the tenants' network stack requiring no modifications on the data-center infrastructure. We formulate the multipath forwarding problem as two different problems: generating disjoint multiple trees, and selecting the tree that contains the least used links to forward a new flow. We improve the network performance summarizing a set of paths on VLAN trees, taking advantage of the VLAN mechanism that is 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 install 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. Our formulation of the genetic algorithm uses an innovative representation of trees as a 2-dimenson vector, which enable a fast convergence of the algorithm. Our initial solution, before optimization, creates already randomized disjoint tree, which fasts the genetic algorithm into reaching the optimal solution. The fast 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. We demonstrate through simulations that the proposed TPM scheme has better performance than the conventional equal cost multipath (ECMP) scheme in high workload scenarios. The experiments show that our formulation of genetic algorithm is able to find a set of completely disjoint trees, 100% disjoint trees, in typical datacenter topology within a few iterations. The simulation results show that the TPM achieves up to 77% and 27% gains when compared to STP and ECMP, respectively.

## Acknowledgments

The authors would like to thank CNPq, CAPES, and FAPERJ for their financial support.

## References

[1] L.H.G. Ferraz, D.M.F. Mattos, O.C.M.B. Duarte, A two-phase multipathing scheme based on genetic algorithm for data center networking, IEEE - GLOBECOM 2014, 2014.

[2] A. Greenberg, J.R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D.A. Maltz, P. Patel, S. Sengupta, VL2: a scalable and flexible data center network, in: Proceedings of the ACM SIGCOMM 2009 conference on Data communication, in: SIGCOMM'09, ACM, 2009, pp. 51–62.

[3] R. Niranjan Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, A. Vahdat, Portland: A scalable fault-tolerant layer 2 data center network fabric, in: Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication, in: SIGCOMM '09, ACM, 2009, pp. 39–50.

[4] F. Yao, J. Wu, G. Venkataramani, S. Subramaniam, A comparative analysis of data center network architectures, in: IEEE ICC 2014, 2014, pp. 3106–3111.

[5] R.S. Couto, S. Secci, M.E.M. Campista, L.H.M.K. Costa, Latency versus survivability in geo-distributed data center design, IEEE - GLOBECOM 2014, 2014.

[6] R. Perlman, An algorithm for distributed computation of a spanningtree in an extended lan, in: ACM SIGCOMM Computer Communication Review, vol. 15, ACM, 1985, pp. 44–53.

[7] M. Alizadeh, A. Greenberg, D.A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, M. Sridharan, Data center TCP (DCTCP), in: Proceedings of ACM SIGCOMM, ACM, 2010, pp. 63–74.

[8] M. Alizadeh, A. Kabbani, T. Edsall, B. Prabhakar, A. Vahdat, M. Yasuda, 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, 2012. 19–19.

[9] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, M. Handley, Improving datacenter performance and robustness with multipath TCP, in: Proceedings of ACM SIGCOMM, ACM, 2011, pp. 266–277.

[10] D.M.F. Mattos, O.C.M.B. Duarte, Xenflow: seamless migration primitive and quality of service for virtual networks, in: IEEE Global Communications Conference - GLOBECOM, 2014.

[11] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, A. Vahdat, Hedera: dynamic flow scheduling for data center networks, in: Proceedings of the 7th USENIX NSDI Conference on NSDI, USENIX Association, 2010. 19–19.

[12] M. Zhang, A. Ghanwani, V. Manral, A. Banerjee, Transparent interconnection of lots of links (TRILL): clarifications, corrections, and updates, RFC 7180 (Standards Track), 2014.

[13] IEEE802.1aq, Standard for local and metropolitan area networks: Virtual bridges and virtual bridged local area networks - amendment 9: shortest path bridging, 2012.

[14] M. Alizadeh, T. Edsall, S. Dharmapurikar, R. Vaidyanathan, K. Chu, A. Fingerhut, V.T. Lam, F. Matus, R. Pan, N. Yadav, G. Varghese, CONGA: distributed congestion-aware load balancing for datacenters, in: Proceedings of ACM SIGCOMM, 2014, pp. 503–514.

[15] T. Ye, H.T. Kaur, S. Kalyanaraman, M. Yuksel, Large-scale network parameter configuration using an on-line simulation framework, IEEE/ACM Trans. Netw. 16 (4) (2008) 777–790.

[16] T. Benson, A. Akella, D.A. Maltz, Network traffic characteristics of data centers in the wild, in: Proceedings of the ACM SIGCOMM IMC, 2010, pp. 267–280.

[17] S. Pandey, M.-J. Choi, Y.J. Won, J. Won-Ki Hong, SNMP-Based enterprise IP network topology discovery, Int. J. Netw. Manage. 21 (3) (2011) 169–184.

[18] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, J. Turner, Openflow: enabling innovation in campus networks, SIGCOMM Comput. Commun. Rev. 38 (2) (2008) 69–74.

[19] I.M. Moraes, D.M. Mattos, L.H.G. Ferraz, M.E.M. Campista, M.G. Rubinstein, L.H.M. Costa, M.D. de Amorim, P.B. Velloso, O.C.M. Duarte, G. Pujolle, FITS: a flexible virtual network testbed architecture, Comput. Netw. 63 (0) (2014) 221–237. Special issue on Future Internet Testbeds - Part {II}.

[20] B. Pfaff, J. Pettit, T. Koponen, K. Amidon, M. Casado, S. Shenker, Extending networking into the virtualization layer, Proc. HotNets (2009).

[21] D.W. Coit, A.E. Smith, Solving the redundancy allocation problem using a combined neural network/genetic algorithm approach, Comput. Oper. Res. 23 (6) (1996) 515–526.

[22] G. Rudolph, Convergence analysis of canonical genetic algorithms, IEEE Trans. Neural Netw. 5 (1) (1994) 96–101.

[23] B. Doerr, C. Doerr, Optimal parameter choices through self-adjustment: applying the 1/5-th rule in discrete settings, GECCO'15, 2015.

[24] G. Nakibly, R. Cohen, L. Katzir, Optimizing data plane resources for multipath flows, IEEE/ACM Trans. Netw. 23 (1) (2015) 138–147.

[25] M. Al-Fares, A. Loukissas, A. Vahdat, A scalable, commodity data center network architecture, in: Proceedings of ACM SIGCOMM, ACM, 2008, pp. 63–74.

[26] A. Headquarters, Cisco data center infrastructure 2.5 design guide, Cisco Validated Design I, Cisco Systems, Inc, 2007.

[27] A. Singla, C.-Y. Hong, L. Popa, P.B. Godfrey, Jellyfish: networking data centers randomly, in: Proceedings of the 9th USENIX NSDI Conference on NSDI, in: NSDI'12, 2012.

[28] J. Mudigonda, P. Yalagandula, J. Mogul, B. Stiekes, Y. Pouffary, NetLord: a scalable multi-tenant network architecture for virtualized datacenters, in: Proceedings of the ACM SIGCOMM 2011 Conference, in: SIGCOMM '11, ACM, 2011, pp. 62–73.

[29] J. Mudigonda, P. Yalagandula, M. Al-Fares, J.C. Mogul, SPAIN: COTS data-center ethernet for multipathing over arbitrary topologies, in: Proceedings of the 7th USENIX NSDI Conference on NSDI, USENIX Association, 2010.

[30] A. Hakiri, A. Gokhale, P. Berthou, D.C. Schmidt, T. Gayraud, Software-defined networking: challenges and research opportunities for future internet, Comput. Netw. 75, Part A (2014) 453–471.

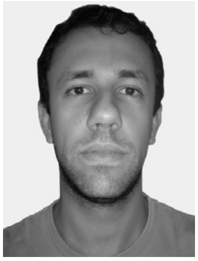
[31] A.R. Curtis, J.C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, S. Banerjee, DevoFlow: scaling flow management for high-performance networks, in: Proceedings of the ACM SIGCOMM 2011 Conference, ACM, 2011, pp. 254–265.

[32] M. Luo, Y. Zeng, J. Li, W. Chou, An adaptive multi-path computation framework for centrally controlled networks, Comput. Netw. 83 (2015) 30–44.

[33] E. Rojas, G. Ibaẽz, J.M. Gimenez-Guzman, J.A. Carral, A. Garcia-Martinez, I. Martinez-Yelmo, J.M. Arco, All-path bridging: path exploration protocols for data center and campus networks, Comput. Netw. 79 (2015) 120–132.

**Lyno Henrique G. Ferraz** is a D. Sc. in the Electrical Engineering Pro- gram from Federal University of Rio de Janeiro (Rio de Janeiro, RJ, Brazil). He received his B.Sc. and M.Sc. degrees in Electronical Engineering from the Federal University of Rio de (Rio de Janeiro, RJ, Brazil) in 2010 and 2011 respectively. His current research interests include network virtualiza- tion, big data and cloud computing.

**Rafael Laufer** received the B.Sc. and M.Sc. degrees in electrical engineering from Universidade Federal do Rio de Janeiro (UFRJ) in 2003 and 2005, respectively, and the Ph.D. degree in computer science from the University of California, Los Angeles (UCLA) in 2011. He is currently a Member of Technical Staff at Bell Laboratories, Alcatel–Lucent in Holmdel, NJ. During his graduate studies, he worked at Bell Labs, Cisco, Technicolor, and EPFL. Dr. Laufer was the recipient of the Marconi Society's Young Scholar Award in 2008 in recognition of outstanding academic achievement. He is a member of ACM and IEEE.



**Diogo M. F. Mattos** is currently a candidate to a degree of D. Sc. in Electrical Engineering from Universidade Federal do Rio de Janeiro, Rio de Janeiro, Brazil. He obtained a Master's degree in Electrical Engineering from Universidade Federal do Rio de Janeiro, in 2012. He received a Computer and Information Engineer degree from the same university, in 2010 with a GPA of 9.3 over 10. He is currently working on his PhD Thesis on the LIP6 (Laboratoire d'Informatique de Paris 6) at Université Pierre et Marie Curie, Paris, France.

**Guy Pujolle** is currently a Professor at the Pierre et Marie Curie University (Paris 6), a member of the Institut Universitaire de France. He is an editor for International Journal of Network Management, WINET, Telecommunication Systems and Editor-In-Chief of the indexed Journal "Annals of Telecommunications". He was an editor for Computer Networks, Operations Research, Editor-In-Chief of Networking and Information Systems Journal, Ad Hoc Journal and several other journals. Guy Pujolle is a pioneer in high-speed networking having led the development of the first Gb/s network to be tested in 1980.

**Otto Carlos M. B. Duarte** received the Eletronic Engineer degree and the M. Sc. degree in Electrical Engineering from Universidade Federal do Rio de Janeiro, Brazil, in 1976 and 1981, respectively, and the Dr. Ing. degree from ENST/Paris, France, in 1985. Since 1978 he is Professor at Universidade Federal do Rio de Janeiro. In 1992/1993 he has worked at Paris 6 University, in 1995, at International Com- puter Science Institute (ICSI) associated to the University of California at Berkeley, and in 2014 at University of California at Berkeley, He worked several times as invited professor at Paris 6 University.