Traffic Engineering for Multiple Spanning Tree Protocol in Large Data Centers

HO Trong Viet, Yves Deville, Olivier Bonaventure, Pierre Francois
{trong.ho, yves.deville, olivier.bonaventure, pierre.francois}@uclouvain.be
ICTEAM, Université catholique de Louvain (UCL), Belgium

***Abstract*—The size of the capacity of data centers have been growing significantly during the last years. Most data centers rely on switched Ethernet networks. A drawback of the Ethernet technology is that it relies on the spanning tree protocol (or variants of it) to select the links that are used to forward packets inside the data center. In this paper we propose a Constrained-Based Local Search optimization scheme that is able to efficiently compute the optimum spanning tree in large data center networks. Our technique exploits the division of the data center network in VLANs. We evaluate its performance based on traffic matrices collected in data center networks and show good improvements compared to the standard spanning tree protocol with up to 16 VLANs.**

***Index Terms*—Traffic Engineering, Multiple Spanning Tree Protocol, Combinatorial Optimization, Data center traffic, Local search.**

I. Introduction

Data centers are now a key part of the Internet. Their number and their size are growing quickly. Some reports [18], [20], [21] indicate that there are data centers containing up to 10K servers and some speculate that data centers could contain 100K servers or more. Data centers are used for various purposes. Some data centers are mainly used to perform computation while others are mainly used to provide Internet services. Many data centers support various applications at the same time and each application runs on a set of virtual machines that are distributed on physical servers.

From a networking viewpoint, data centers heavily rely on switched Ethernet networks. Servers are typically attached by using one or more Gigabit Ethernet interfaces to top of the rack switches that are connected to aggregation switches by using one or more 10 Gbps Ethernet links. These aggregation switches are then connected by using one or more 10 Gbps Ethernet links to core switches that are attached to routers when Internet access is required. The switched Ethernet networks used in data centers are redundant to enable recovery in case of link or switch failure. However, currently deployed Ethernet switches can not directly use a mesh of links and need to use variants of the Spanning Tree protocol to restrict the Ethernet network to a tree rooted on, typically a core, switch. Several variants of the spanning tree protocol are used. The first standard, IEEE 802.1d [9], enables switches to compute a single spanning tree over a large network. The rapid spanning tree protocol [10] is an extension to 802.1d that enables switches to converge quickly to an alternate spanning tree in case of link failures. 802.1d is largely used in campus networks. However, its major drawback is that it disables the links that do not belong to the selected spanning tree. This is potentially a waste of resources since these links exist in the network. Fortunately, large Ethernet networks such as data center networks are logically divided in several Virtual Local Area Networks (VLANs). VLANs are mainly used to isolate one application or one data center customer from the others. The servers (or virtual machines) that belong to a given VLAN can only communicate with the other servers that belong to the same VLAN. A switched Ethernet network can contain up to roughly 4000 different VLANs. The Multiple Spanning Tree Protocol(MSTP) [11] is an extension to the Spanning Tree Protocol that allows switches to compute several spanning trees over a single physical topology. The utilization of multiple spanning tree enables a network operator to spread different VLANs over different spanning trees that use different physical links and switches. Networks that implement the MSTP protocol are typically able to use more links than networks using a single spanning tree. In practice however, it should be noted that most implementations of MSTP can only compute up to 16 different spanning trees, and map each VLAN onto one of these spanning trees.

Constraint Programming (CP) [14] and Constraint-Based Local Search (CBLS)[13] are well suited for solving complex combinatorial problems. COMET [13] is an object-oriented language with several innovative modeling and control abstractions for CP and CBLS. In COMET, some classical problems can be modeled in only about a dozen lines of code. We chose a local search approach implemented in COMET for solving this traffic engineering (TE) problem.

In this paper, we propose a Local Search Algorithm for MSTP (LSA4MSTP), which guides the MSTP to select a good set of spanning trees for a given set of traffic demand matrices. The remainder of this paper is organized as follows.

We first present the related works in section II. We define the problem formulation in section III. Next, we present our local search algorithm with the techniques for speeding up the search in section IV. Our evaluation is presented in section V with an analysis of the experimental results, and we conclude the paper in Section VI.

II. Related Works

There exist four main approaches to deal with the TE for MSTP. First, several MSTP optimization techniques such as [1], [2], [3] aimed to map a set of VLANs to a given number of spanning trees. Second, [4] has proposed a multi-objectives meta-heuristic ensuring the load balancing of the Metro Ethernet using MSTP by mapping a set of flows to a set of given spanning trees. Third, the construction algorithm [5] addresses the TE problem for the US network with 12 vertices and 17 links by building source-based multiple spanning trees (construct a spanning tree for each of the given source nodes). Last, [6], [7] advocated for solving the TE for MSTP by finding the set of spanning trees in the metro domain described by customer traffic demands and given network topology. In this paper, we follow the same approach as in [6], [7] but we aim to find solutions for large data center networks with hundreds of switches instead of small networks with dozen switches as in the state-of-the-art techniques.

Our previous work in [12] considered the traffic engineering problem for the STP 802.1d [9] in switched Ethernet networks. We proposed a local search algorithm using spanning tree neighborhood instead of link cost neighborhood. In this paper, we extend our local search proposed in [12]. We introduce new heuristics to cope with large data centers and many VLANs (spanning trees). In addition, the topologies and traffic demand matrices of current data centers mentioned in [18] are used for evaluating the performance of our algorithm.

III. Problem Formulation

Our Ethernet network is modeled as an undirected graph $G = (N, E)$, where N is the set of nodes (switches) and E is the set of links between nodes. Each link $(i, j) \in E$ has a bandwidth denoted by $BW[i, j]$ (note that $BW[i, j]$ = $BW[j, i]$). When link bundles are used between switches, we consider each bundle as a single link having the bandwidth of the bundle. Let V=$\{V_1, V_2, ..., V_k\}$ be the set of k given VLANs in the network (each VLAN is a connected component of G represented by $V_r \subseteq N \ \forall r \in [1..k]$) and TD=$\{TD_1, TD_2, ..., TD_k\}$ be the set of k traffic demand matrices. $TD_r[i, j]$ (r=1, 2, ..., k; i, $j \in V_r$), represents the traffic that switch i sends to switch j. Let W=$\{W_1, W_2, ..., W_k\}$ be the set of link weight matrices for the k given VLANs. We call $MST(G, V, W)$ the set of k spanning trees $ST_1(G, V_1, W_1)$, ..., $ST_k(G, V_k, W_k)$ obtained by the Multiple Spanning Tree Protocol [11] on graph G, with set of VLANs V and set of link weight matrices W. The Ethernet switching problem is defined as follows: for all $TD_r[i, j] > 0$ (r=1, 2, ..., k; i, j $\in V_r$), distribute the traffic demand over unique path from i to j in $ST_r(G, V_r, W_r)$.

Assume $L[i, j]$ (i, j $\in N$) denotes the load (sum of traffic flow) on the link (i, j). For the computation of $L[i, j]$, the traffic flow is directed ($L[i, j] \neq L[j, i]$).

The utilization of a link (i, j) is the ratio between its load and its bandwidth:

$$U[i, j] = \frac{L[i,j]}{BW[i,j]} \text{ (i, j} \in N).$$

The link utilization is also directed. The link (i, j) is overloaded if its load is greater than its bandwidth ($U[i, j] > 1$ or $U[j, i] > 1$).

In this work, our goal is to find the *optimal* (best possible) set of link weights matrices W^* minimizing the maximal utilization:

$$U_{max} = max\{max(U[i, j], U[j, i]) \mid (i, j) \in E\}.$$

The formulation of this problem is the following:
Input: Graph $G = (N, E)$, set of k VLANs V, bandwidth matrix BW, set of k traffic demand matrices TD
Output: Set of link weights matrices W^* such that $MST(G, V, W^*)$ yields k spanning trees minimizing U_{max}

There are many possible QoS objectives for this traffic engineering problem: minimization of the maximal link utilization, minimization of the network delay, fault tolerance, etc. In this work, we start with the popular objective of minimizing the maximal utilization because this is the most important and the most visible objective. We will take into account the other objectives in our further work.

This problem is an expansion of the Spanning Tree Protocol Optimization in [12]. Its search space is exponential. It is too complex to be solved with exact methods even for reasonable size instances. In this paper, we use a local search algorithm for achieving a good approximation of this optimization problem.

IV. MSTP Optimization Using Local Search

The MSTP creates the spanning trees based on two parameters: the switch IDs and the link weights. The weight of a link is an integer number in $[1..2^{16} - 1]$. Suppose that we have k VLANs and each VLAN consists of all the switches in the network. Even if we do not consider the choices of root for each spanning tree, the size of the search space if we do a search on link weights is $(2^{16} - 1)^{k.m}$ (with m the number of links). In addition, it is difficult to control the change of link weights on the spanning tree. Another possibility is to search the space of spanning trees and at the end to generate the link weights from the obtained spanning trees. Of course the generated link weights are such that MSTP will provide the obtained spanning trees. This method reduces significantly the size of the search space to $\binom{m}{n-1}^k$ (with n the number of switches). This is the solution chosen in this paper.

Local Search (LS) is a powerful method for solving computational optimization problems such as the Vertex Cover, Traveling Salesman, or Boolean Satisfiability. The advantage of LS for these problems is its ability to find an intelligent path from a low quality solution to a high quality one in a huge search space. This can be done by iterating a heuristic of exploration to the neighborhood solutions [13]. So, we choose a local search approach on the spanning tree space for solving this problem. In this section, we first present an overview of our local search algorithm called LSA4MSTP (Local Search Algorithms for the Multiple Spanning Tree Protocol problem). Then, we describe the algorithms and the techniques allowing to speed up the search.

Algorithm 1 Pseudo-code for LSA4MSTP

```
1: $MST$ = getInitialMST($G, V$)
2: $MST^*$= $MST$
3: $U^*_{max}$ = getMaxUtilization($MST, BW, TD$)
4: while $time\_exec < time\_windows$ do
5:     $(s_{max}, t_{max})$ = getMaxCongestedLink($MST$)
6:     $selected\_vlan$=selectVLAN($MST, s_{max}, t_{max}$)
7:     $(s_0, t_0)$= getRemovedLink($MST, selected\_vlan,$
                                                    $s_{max}, t_{max}$)
8:     $(s_I, t_I)$ = getAddedLink($MST, selected\_vlan, s_0, t_0$)
9:     $MST$= replaceEdge($MST, selected\_vlan, s_0, t_0, s_I, t_I$)
10:    $U_{max}$ = getMaxUtilization($MST, BW, TD$)
11:    if $U_{max} < U^*_{max}$ then
12:        $U^*_{max} = U_{max}$
13:        $MST^* = MST$
14:    end if
15: end while
16: $W^*$ = generateLinkWeights($G, MST^*$)
```

A. Algorithm description

Algorithm 1 is an extension of our algorithm LSA4STP in [12]. It provides the pseudo-code of our local search algorithm. We aim to find a good solution in the spanning tree search space by moving from one solution to a neighbor solution. At each iteration, we try to replace one edge in one of the k spanning trees to reduce U_{max}. The steps of LSA4MSTP, according to the pseudo-code, are:

- Line 1: The method $getInitialMST(G,\ V)$ returns an initial solution (k spanning trees) for the local search algorithm. We simply simulate the MSTP to compute this initial solution. In this computation, we use the link weights for each VLAN as the 802.1d [9] cost by default.
- Line 2 and Line 3: We store the initial solution obtained with 802.1s as the best solution at the start of the search. The method $getMaxUtilization(MST, BW, TD)$ returns U_{max} after computing the utilization of each links in MST.
- Line 4: We use the time window as the termination criteria. The choice of time windows depends on the test size (number of nodes and number of links).
- Line 5-7: At each search iteration, we first try to find the most congested link (s_{max}, t_{max}) (Line 5). Then, we select a VLAN $selected\_vlan$ that contains (s_{max}, t_{max}) (Line 6). An edge (s_0, t_0) in $selected\_vlan$ will be chosen to be removed (Line 7). We describe this task in section IV.A.1.
- Line 8: Section IV.A.2 describes the choice of the replacing edge (s_I, t_I) in the method $getAddedLink(MST, selected\_vlan, s_0, t_0)$. A new spanning tree for $selected\_vlan$ is created by replacing (s_0, t_0) with (s_I, t_I).
- Line 9: An update of the link loads is performed when the edge (s_0, t_0) is replaced by (s_I, t_I). The speeding up technique for computing the link loads is depicted in section IV.B.2.
- Line 10-14: If the new U_{max} is less than the best known U^*_{max}, we store this solution as the best one.
- Line 16: the method $generateLinkWeights$ (G, MST^*) generates k link weight matrices so that 802.1s protocol [11] produces exactly the k spanning trees in MST^*. This link weight generation is described in section IV.A.3.

The search process from line 5 to line 14 is iterated until the execution time reaches the time window (Line 4).

1) Removing an edge: In each search iteration, we try to relieve the most congested link (U_{max}) of its load by replacing an edge in the spanning tree containing it. To determine which edge from which VLAN to be replaced, LSA4MSTP extends from the heuristic in [12]. Let (s_{max}, t_{max}) be the most congested oriented link, the key decision of this heuristic is to select one of the VLANs containing (s_{max}, t_{max}) to do the replacement. In the method $selectVLAN(MST, s_{max}, t_{max})$ (Line 6 - Algorithm 1), we create a set SV of VLANs that contains (s_{max}, t_{max}). We assign to each VLAN in SV a probability to be selected based on its load on (s_{max}, t_{max}). The probability for a $vlan \in SV$ to be $selected\_vlan$ is:

$pr[vlan] = \frac{L_{vlan}[s_{max}, t_{max}]}{\sum_{i=1}^{k} L_i[s_{max}, t_{max}]}$. Obviously, $\sum_{i \in SV} pr[i] = 1$.

This strategy can find a balance between greedy search and unexplored neighborhood. From $selected\_vlan$, we can assume that the congestion is caused by the traffic coming from the subtree of the spanning tree $ST_{selected\_vlan}$ dominated by s_{max}. Next, the method $getRemovedLink(MST, selected\_vlan, s_{max}, t_{max})$ (Line 7- Algorithm1) uses the heuristic described in [12] to determine an edge (s_0, t_0) to be removed from the set of edges containing (s_{max}, t_{max}) and all the edges belonging to the subtree dominated by s_{max}. The edges closer to the root have a higher probability to be removed.

2) Adding an edge: After having removed (s_0, t_0) from $ST_{selected\_vlan}$, we obtain two separate trees that must be reconnected with a new edge. Our objective is to have more bandwidth and a less congested solution. We consider two criteria for choosing an edge to be added to form the new spanning tree. First, we call SA the set of all the edges that join the two separate trees. Second, we consider h edges having the highest remaining bandwidth from SA. Next, we compute the resulting U_{max} when adding each of these h edges. The edge (s_I, t_I) offering the minimal value of U_{max} is selected to be added into $ST_{selected\_vlan}$.

This heuristic differs from [12] as an edge can belong to many spanning trees. Therefore, we can consider only the highest *remaining* bandwidth edges instead of the highest bandwidth edges.

We also use tabu search [15] - a heuristic preventing the search from visiting the same points in the search space. But in this problem, we insert only the added edge in each search step into the tabu list. We do not tabu the max congested edge and removed edge as in [12] because in MSTP, an edge can be used by different spanning trees.

3) Link Weight Generation: From the k spanning trees in MST^* obtained by LSA4MSTP, we generate the k link weight matrices of W^*. For each VLAN $V_i \in V$, we assign a unit cost to all the links in the spanning tree ST_i and by assigning a weight of n_i (number of nodes of V_i) to all the other links in V_i. After this assignment, we can see that the weight of the longest possible path between a pair of nodes in spanning tree ST_i is $n-1$ (passes $n-1$ edges) while the cost of the shortest path between any pair of nodes with out using of spanning tree edges is n (passes one edge). Consequently, the 802.1s protocol [11] will produce exactly the k spanning trees in MST^*.

B. Speeding up the Search

1) Root Selection: Symmetry breaking [16] is a well-known technique in Constraint Satisfaction Problems (CSPs) to speed up the search. In [12], we showed that the root determination does not change solution. By fixing a unique root for each VLAN, we can eliminate all the symmetries in this problem. The search space is reduced from $(n.\binom{m}{n-1})^k$ to $\binom{m}{n-1}^k$ (with n the number of nodes, m the number of links and k VLANs). The choice of root can however influence the choice of the neighborhood solution in each iteration and thus the balance of the trees. Network operators normally configure the switches with the highest capacity (ports x bandwidth) as the root of their spanning trees. In our algorithm, we a priori select the node with maximal sum of associated link capacities (bandwidth) as the root.

2) Incremental Link Load Computation: Link load computation is a computationally expensive task at iteration, especially when the size of networks is large. In [12], we proved that for each replacement of an edge (s_0, t_0) by another edge (s_I, t_I), the load changes only on the links on the cycle C created by adding (s_I, t_I) into the spanning tree. This substabtially reduces the computation cost.

In our algorithm, spanning trees are represented using the LS(Graph & Tree) framework [17]. With incremental data structures (auto-update after each change of the tree), all queries on spanning tree mentioned above can be performed in time $O(1)$ and the update action is performed in $O(n_v)$ where n_v is the number of vertices of the VLAN.

V. Experiments and evaluation

In this section, we first present two topologies coming from the private enterprise and cloud data centers which are studied in [18]. Second, we described the method for generating these topologies, the traffic demand matrices and the VLANs for our tests. Next, we analyze the obtained results and evaluate the performance of our local search algorithm LSA4MSTP.

A. Data Center topologies

The extensive studies by Benson et al. [18] showed that there are three main classes of data centers, namely university campus data centers, private enterprise data centers and cloud data centers. The statistics were collected from 10 data centers in US and South America. In this work, we only consider the large data centers of private enterprises and clouds containing a few thousand to more than 10K servers. We choose to not consider university campus data centers because their size is often too small, containing only a few decades of switches. As depicted in Fig 1, the private enterprise data center uses a canonical 2 or 3-Tier Cisco architecture [19] while the cloud data center uses the 3-Tier textbook data center architecture in [22].

3-Tier Cisco architecture (Fig 1a) consists of core, aggregation and edge (or access) tier. At the highest level, core tier contains switches connecting the data center to extranet, WAN or Internet. The aggregation tier consists of switches connecting to many edge tier uplinks, and aggregating flows going in and out of the data center. Core and aggregation switches are usually equipped with 10 Gbps interfaces [19]. At the lowest level, edge tier consists of the racks. Each rack contains 20-80 servers interconnected by a Top of Rack switch (ToR). Each ToR switch has usually a small number (4-8) of 10 Gbps uplinks and servers are attached to their ToR switch through 1 Gbps links [19]. The 2-Tier Cisco architecture is used in small data centers in which the core tier and aggregation tier are merged into one tier.

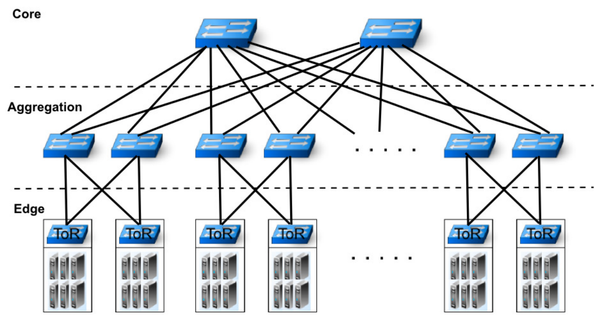
Cloud data center architecture described in [22] (Fig 1b) is an improvement of the canonical 3-Tier Cisco architecture. In this topology, the core tier is replaced by an intermedia tier to improve the performance of the aggregation layer. A large number of 10GigE ports of each aggregation switch is used that can provide a huge aggregate capacity. The links of the intermediate and aggregation switches form a complete bipartite graph [22]. Suppose each aggregation switch uses k ports of 10 GigE, $k/2$ of these ports will connect to $k/2$ switches in the intermedia tier. The remaining $k/2$ ports of each aggregation switch are used for connecting to the ToRs in the edge tier. As in 3-Tier Cisco architecture, core and aggregation switches are equipped with 10 Gbps interfaces and ToRs have 4-8 interfaces of 10 Gbps and a large number of 1Gbps links.

In our experiments, the number of servers per rack is fixed to 20. Thus, each private enterprise data center with 4K servers consists of 242 switches (200 ToRs + 40 aggregation switches + 2 core switches) and each cloud data center with 10K servers contains 564 switches (500 ToRs + 32 aggregation switches + 32 intermedia switches).

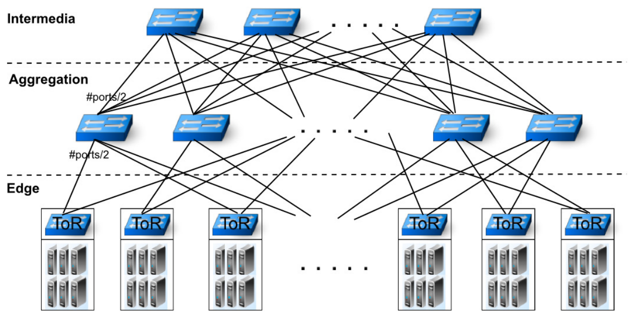
B. Traffic demand matrices and VLANs generation

To obtain data sets that are representative of data centers, we analyzed the SNMP data from [18] on a private enterprise (PR) data center (53 switches). These data allowed us to synthesize the traffic flow in the network by 10 minutes, 1 hour, 1 day and 1 week worth of data. Unfortunately, there is no information related to VLANs composition and on traffic demand between each pair of switches.

1) Traffic demand matrices: In spite of the fact that the VLAN and traffic demand information is inaccessible, the SNMP data from [18] is worthy to infer traffic demand



(a) Private enterprise data center



(b) Cloud data center

Fig. 1: Large data center topologies

matrices. The SNMP data captures the amount of traffic on each link at a precise time. Therefore, we were able to retrieve the total traffic entering or departing from each switch over various time intervals. We use the *simple gravity model* [23] relying on proportionality relationships to build the traffic demand matrix. This method is developed for large-scale IP networks but we assume that it is also appropriate for the data center networks. The simple gravity model is defined as follows:

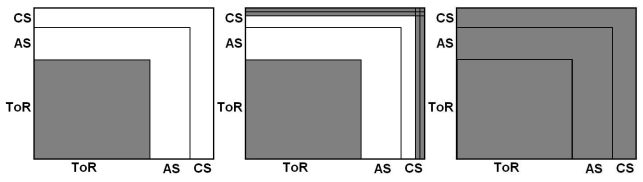
$$TD[i,j] = TI(i,*)\frac{TO(*,j)}{\sum_k TI(*,k)}.$$

where $TD[i,j]$ is the traffic amount switch i sends to switch j. $TI(i,*)$ represents the total traffic entering at switch i. $TO(*,j)$ denotes the total traffic departing from switch j. And $\sum_k TI(*,k)$ is the total amount of traffic departing of every switch.

In these data centers, there is always a switch receiving a large amount of traffic (20-40% total traffic). There are about ten other switches receiving from 2 to 18% total traffic. For the remaining switches, the traffic amount is less than 2%. When we look at each line of the traffic demand matrices, each switch has about ten ”big clients” with a demand from 2 to 30% of its total traffic volume.

For each time sample, we thus obtain a demand matrix. For each demand matrix, we define $SumTD = \sum TD[i,j]$, denoting the total amount of traffic demand. We compute the ratio of traffic volume of each switch to total traffic: $\%TD\_SW[i] = \sum_j TD[i,j]/SumTD$ and the ratio of each traffic demand element to total traffic demand of each switch: $\%TD[i,j] = TD[i,j]/TD\_SW[i]$.

The obtained demand matrices will be used to generate the traffic demand matrices of our VLANs. The traffic demand of each of our VLAN will thus be considered as the demand of a small private enterprise data center. For the number of racks (ToRs) in each VLAN, we consider 40 ToRs for each VLAN in cloud topologies (with 564 nodes) and 20 ToRs for each VLAN in private topologies (with 242 nodes). Each VLAN will then also contain the minimum of aggregation and core/intermediate switches in order to cover the ToRs of the VLAN. In our experiments, we consider three types of traffic



(a) Internal TM (b) Internet TM (c) Uniform TM

Fig. 2: Traffic demand matrix types

demand matrices for the VLANs.

- **Internal TM**: We here assume that all the traffic stays within the VLAN and consists of discussions accross the k racks (ToRs) of the VLAN. The demand matrix of the VLAN is thus composed of zeros, except between these k ToRs (Figure 2a). The traffic demand between the ToRs of the VLAN is based on the obtained demand matrices on a private enterprise data center (PR) presented above. We first choose a target $SumTD$. Then, using $\%TD\_SW[i]$ and $\%TD[i,j]$, we derive a demand matrix as follows. The k ToRs of the VLAN are randomly assigned to k different nodes of PR (the nodes of PR with the smallest $\%TD\_SW[i]$ are not considered). Then, given a ToR of the VLAN associated to node i of PR, its traffic demand with the other k ToRs of the VLAN will be a random permutation of the top k values of $\%TD[i,j]$. These values are then randomly assigned.
- **Internet TM**: This case extends the previous case by considering traffic outside network, consisting of traffic accross VLANs and traffic from/to Internet. The traffic entering/leaving each VLAN is centralized at one or two core switches (with an average of 1.5) for private enterprise networks, and at two core switches for cloud data centers. The traffic demand matrix will thus have (one or) two other non zero lines and columns (as described in Figure 2b), associated to these core switches. We will assume that 20% of $SumTD$ is interconnection

traffic, and 80% of $SumTD$ stays within the VLAN. The traffic demand within the VLAN is obtained as described above. The interconnection traffic is uniformly distributed between all the switches.

- **Uniform TM**: This type of traffic will be used as a reference for the experiments, with a uniform distribution of the traffic demand between every pairs of switches in the VLAN (see Figure 2c). The values in the matrix are varied in a small interval. The value is chosen to achieving the targeted $SumTD$.

2) VLANs generation: We rely on two approaches for generating VLANs. In the first case, each VLAN is generated geographically by grouping a set of neighboring racks that are interconnected by the ToRs and a number of aggregation and core (or intermedia) switches (i.e. the racks of servers in the same or neighboring buildings). In the second case, we assume that the racks are assigned randomly to the different VLANs depending on their increasing need. For this reason, each VLAN can contain a set of arbitrary racks.

In our experiments, for each data center topology, we generated 16 VLANs for both geographic and random case. The 16 VLANs are generated by combining the four time samples (4 VLANs generated using each of the time sample). We also ensured that the 16 VLANs cover all the switches.

In order to analyse the influence of the number of VLANs on the performance of our algorithm, we merged the 16 VLANs, 2 by 2, in order to obtain a new test with 8 VLANs, with an equivalent total traffic. Two VLANs can be merged if they have at least one common switch (for the geographic case, this common switch must be ToR). We repeated this process to obtain tests with 4, 2 and a unique VLAN (containing all the switches in the network).

C. Experiments

The different test sets are summarized in Table I. For each topology type (private enterprise and cloud), we generated 10 topologies. For each topology, we combined two VLAN distributions (Geographic and Random) with three traffic matrices (Internal TM, Internet TM and Uniform TM). For each of these 12 combinaisons, we generated 5 tests (16, 8, 4, 2 and 1 VLAN). We thus have 600 tests in our data sets. These data sets are available online in [24]. The time window for running LSA4MSTP for Private Enterprise and Cloud is 15 minutes.

D. Evaluation

As many traffic engineering studies in [4], [8], [12], we consider the improvement of the maximal utilization U_{max} as the criterion to evaluate the performance of LSA4MSTP. We compare two different computations of U_{max}. The first one is obtained with the solution of LSA4MSTP and second is obtained with the default weights by the STP 802.1s standard. With 802.1s, one spanning tree is computed for each VLAN based on the least cost paths from every switch in VLAN to an elected root switch (the switch with min ID - normally this is one of the core switches). Because the network consists of all 10 Gbps uplinks with the default cost of 2 [9], so the least cost path strategy of 802.1s seems to be limited.

We measure the improvement in a test by the ratio between $U_{max}[LSA4MSTP]$ and $U_{max}[802.1s]$:

$$\%Improve = \frac{U_{max}[LSA4MSTP]*100}{U_{max}[802.1s]}$$

Figure 3 presents the U_{max} values for Cloud data centers. LSA4MSTP always gives the best results for 16 VLANs, with $\%Improve$ around 50% (about half of the U_{max} value given by 802.1s). For 8 VLANs, this improvement is about 60% in both geographic and random case. With 4, 2 and even 1 VLAN, LSA4MSTP also reduces U_{max} to about 70%-80% in almost all the combinations. These results clearly show that our LSA4MSTP algorithm provide better results than 802.1s.

We describe in Table II the $\%Improve$ results for Private Enterprise data centers. We also observe that LSA4MSTP is more efficient when the number of VLANs is large. For 16 VLANs with the original traffic matrices, LSA4MSTP gives best performance with the uniform traffic matrices where there is no zero-demand for every pair of source-destination. The improvement is less important with the sparser traffic matrices as the internal VLAN matrices. In summary, our LSA4MSTP algorithm always provides better results than 802.1s. Moreover, the number of VLANs clearly further improves the quality of the solution produced by LSA4MSTP.

TABLE II: Results for Private Enterprise data centers: $\%Improve$ (in percent)

Combination	1 VL	2 VLs	4 VLs	8 VLs	16 VLs
Geographic/Internal TM	85.47	83.20	70.42	54.30	52.11
Geographic/Internet TM	87.67	83.04	73.67	52.43	51.26
Geographic/Uniform TM	88.97	83.05	80.51	62.13	42.46
Random/Internal TM	84.15	83.33	69.91	61.92	57.60
Random/Internet TM	83.36	75.63	70.25	58.54	52.19
Random/Uniform TM	88.03	80.77	78.17	62.85	42.02

TABLE III: $\%\Delta Links$ between LSA4MSTP and 802.1s (in percent)

Topo	1 VLAN	2 VLANs	4 VLANs	8 VLANs	16 VLANs
PR	0	2.59	7.26	10.46	12.54
Cloud	0	0.71	2.05	2.79	3.41

Figure 4 shows the distribution of link utilization of the solution provided by LSA4MSTP and by 802.1s on the private enterprise topology (Internal TM/Geographic). With 16 VLANs, the 802.1s solution uses only 822 links for the packet switching while LSA4MSTP uses 1008 links (we consider both directions of a link). This distribution shows that the high values of U_{max} in the 802.1s solution are concentrated on few links. The LSA4MSTP solution, by using more links, is able to reduce U_{max} from 0.66 to 0.37. For 8 VLANs, the most congested links are only concentrated on very few links (less than 1%) in the solutions given by 802.1s. The link

TABLE I: Data generation for LSA4MSTP

Topo	Topo. Type	Num. Switches.	Num. Servers	Traffic Matrices	VLAN distribution
Private Enterprise (PR)	3-Tier Cisco	242	4,000	Internal TM/Internet TM/Uniform TM	Geographic/Random
Cloud	VL2	564	10,000	Internal TM/Internet TM/Uniform TM	Geographic/Random

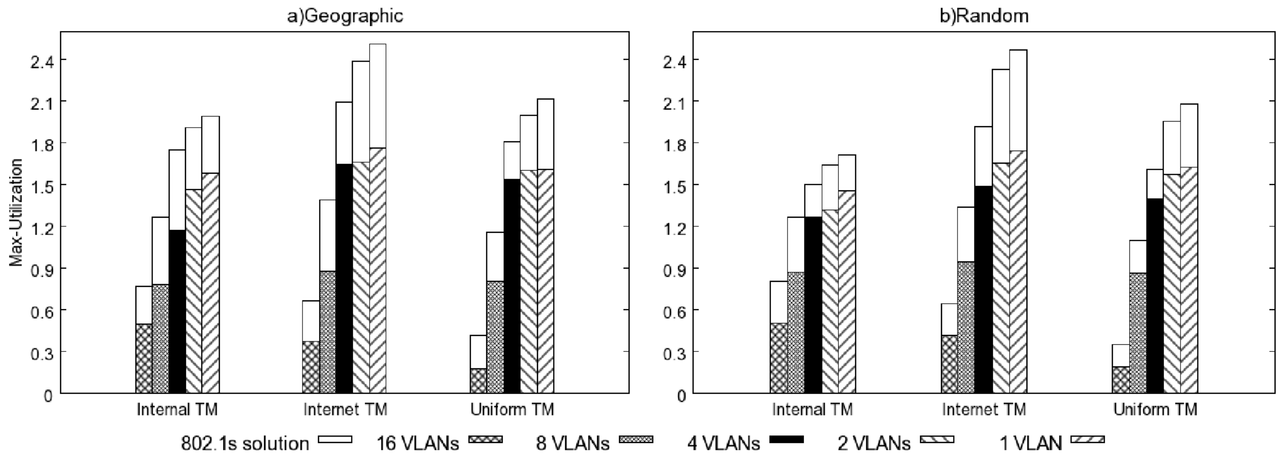


Fig. 3: LSA4MSTP for Cloud data centers

utilization of the other links are similar in the two solutions. This analysis can also be made on the other combinations with 4 and 2 VLANs where the congestion in the solutions obtained with 802.1s is centralized in about 2 or 3 bottleneck links.

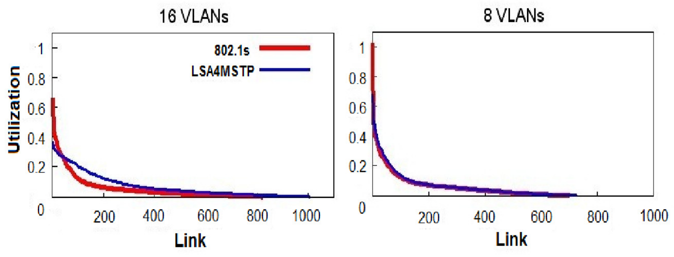


Fig. 4: Link utilization distribution

We now analyze the influence of the number of VLANs on the number of used links. Let $\#Links[LSA4MSTP]$ and $\#Links[802.1s]$ denote the number of links given by the solution of LSA4MSTP and 802.1s. We compute the percentage of links on the total number of available links $\#Links$ in the network that LSA4MSTP uses more than 802.1s:

$$\%\Delta Links = \frac{(\#Links[LSA4MSTP] - \#Links[802.1s]) * 100}{\#Links}$$

We present in Table III the value of $\%\Delta Links$ for each topology type. In the solutions given by LSA4MSTP, the spanning trees use more links than the ones obtained with 802.1s for all the combinations with more than 1 VLAN where $\#Links[LSA4MSTP]$ and $\#Links[802.1s]$ are fixed to $n-1$ (with n the number of switches in the data center). We can thus disjoin the VLAN spanning trees on the most congested links. The value of $\%\Delta Links$ increases naturally with the number of VLANs. This justifies why the best U_{max} results are obtained with 16 VLANs.

We further refine our analysis by presenting in Table IV the average number of links connecting Intermedia-Aggregation (Int-AS) and Aggregation-Edge (AS-ToR) for Cloud data centers. For both 802.1s and LSA4MSTP solutions, the number of used links Int-AS is very limited (always less than 100 links) comparing to the available links on this level (1024 links). Contrarily, the number of used links AS-ToR is growing quickly with the number of VLANs (up to 71% of available links on this level). Obviously, for the Cloud data centers there are 500 ToRs and only 32 Aggregation switches + 32 Intermedia switches. In addition, 80% of the total traffic amount is used for the traffic across racks. Our LSA4MSTP algorithm always uses more links than 802.1s. It is interesting to notice that LSA4MSTP can reduce 50% of U_{max} with only 63 more links for 16 VLANs.

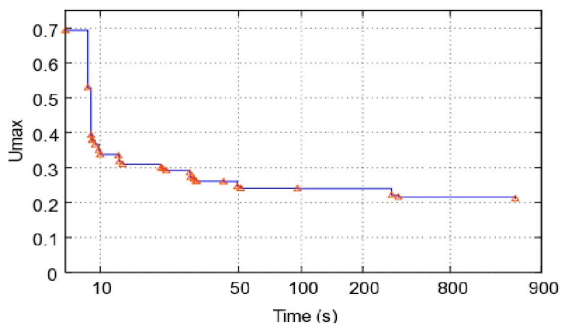


Fig. 5: Improvement of U_{max} over execution time

TABLE IV: Number of used links across tiers for Cloud data centers (LSA stands for LSA4MSTP)

		1 VLAN		2 VLANs		4 VLANs		8 VLANs		16 VLANs	
Links	**Available**	**802.1s**	**LSA**	**802.1s**	**LSA**	**802.1s**	**LSA**	**802.1s**	**LSA**	**802.1s**	**LSA**
Int-AS	1024	63	63	66	69	73	78	75	81	91	98
AS-ToR	1000	500	500	521	532	536	573	598	648	656	712
Total	2024	563	563	587	601	609	651	673	729	747	810

We finally describe in Figure 5 the improvement of the LSA4MSTP solution over execution time, for a test of Cloud with 16 VLANs/Uniform TM/Geographic. As expected, LSA4MSTP reduces about 50% U_{max} of 802.1s (from 0.69 to 0.33) after only 10s. We can state that most of improved solutions were found in the first 98s. In our experiments, the solution found by LSA4MSTP in the first 5 minutes is often very close to the best solution.

With the obtained results in our previous work in [12] with Grid, Cube, Expanded Tree, Fat Tree and PortLand, our local search algorithms give good performance with large instances of network topology.

VI. Conclusion and Future work

The goal of this work is to give a new approach for the traffic engineering problem for metropolitan networks where Multiple Spanning Tree Protocol 802.1s is deployed. To cope with large data centers with many VLANs, our algorithm for single switched Ethernet network has been extended with new adapting heuristics. We considered the current modern topologies for large data centers containing up to 10K servers in our experiments. The SNMP data of a private enterprise in US has also been studied to create the traffic demand matrices that are representative of data centers for our tests. With regard to the load balancing aspect, our results show much improvement in the use of network available bandwidth. The solutions obtained with our algorithm could reduce up to 50% the maximal link utilization compared with the solution obtained by 802.1s for the data centers with 16 VLANs.

Our further work is to extend our scheme to take into account the delay and the fault tolerant aspect. We hope that with this extended algorithm, data centers can become more flexible and efficient in case of link or switch failures not only for speeding up the slow convergence time but also for achieving a high level of QoS.

Acknowledgements

We would like to thank T. Benson et al. for the precious data set. Ho Trong Viet is supported by the FRIA (Fonds pour la formation à la Recherche dans l'Industrie et dans l'Agriculture, Belgium). Pierre Francois is supported by the FRNS (Fonds National de la Recherche Scientifique, Belgium). This work was partly funded by CHANGE, a research project funded by the European Commission in its Seventh Framework program (Project 257422).

References

[1] A. Meddeb, "Multiple Spanning Tree Generation and Mapping Algorithms for Carrier Class Ethernet", in *Proceedings of the IEEE GLOBECOM Conference*, 2006.

[2] Xiaoming He, Mingying Zhu, and Qingxin Chu, "Traffic Engineering for Metro Ethernet Based on Multiple Spanning Trees", in *Proceedings of the International Conference ICNICONSMCL*, 2006.

[3] Y. Lim, H. Yu, S. Das, S. S. Lee, M. Gerla, "QoS-aware Multiple Spanning Tree Mechanism over a Bridged LAN Environment", in *Proceedings of the IEEE GLOBECOM Conference*, 2003

[4] D. Santos, et al., "Traffic Engineering of Multiple Spanning Tree Routing Networks: the Load Balancing case", in *Proceedings of the 5th Euro-NGI conference on Next Generation Internet networks*, 2009.

[5] W. Chen, D. Jin, L. Zeng, "Design of Multiple Spanning Trees for Traffic Engineering in Metro Ethernet", in *Proceedings of the International Conference on Communication Technology ICCT*, 2006.

[6] G. Mirjalily, F. A. Sigari, R. Saadat, "Best Multiple Spanning Tree in Metro Ethernet Networks", in *Proceedings of the Second International Conference on Computer and Electrical Engineering*, 2009.

[7] M. Padmaraj , et al., "Metro Ethernet traffic engineering based on optimal multiple spanning trees", in *Wireless and Optical Communications Networks*, 2005.

[8] Bernard Fortz, and Mikkel Thorup, "Internet Traffic Engineering By Optimizing OSPF Weights," in *IEEE INFOCOM*, 2000.

[9] IEEE Standard 802.1D, "Information technology-Telecommunications and information exchange between systems-Local and metropolitan area networks-Common specifications-Part 3: Media Access Control (MAC) Bridges", 1998.

[10] IEEE Standard 802.1W, "Rapid spanning tree configuration", 2001.

[11] IEEE Standard 802.1S, "Virtual Bridged Local Area Networks - Amendment 3: Multiple Spanning Trees", 2002

[12] T. V. Ho, O. Bonaventure, Y. Deville, Q. D. Pham, and P. Francois, "Using Local Search for Traffic Engineering in Switched Ethernet Networks", in *Proceedings of the* 22^{th} *ITC Conference*, Amsterdam, The Netherlands, 2010.

[13] P. V. Hentenryck, and L. Michel, "Constraint-based Local Search," *MIT Press*, 2005.

[14] Krysztof R. Apt, "Principles of Constraint Programming", *Cambridge University Press*, 2003.

[15] Glover, F., M. Laguna, "Tabu Search," *Kluwer Acadenic Publishers*, Norwell, MA, 1997.

[16] I. Gent, and B. Smith, "Symmetry breaking in constraint programming," in *Proc. ECAI00*, 2000.

[17] Q. D. Pham, Y. Deville, and P. V. Hentenryck, "LS(graph & tree): a local search framework for constraint optimization on graphs and trees," in *Proceedings of the 2009 ACM Symposium on Applied Computing (SAC)*, 2009.

[18] T. Benson, Aditya Akella, and David A. Maltz, "Network Traffic Characteristics of Data Centers in the Wild," in *Proceedings of the Internet Measurement Conference (IMC)*, Melbourne, Australia, Nov 2010.

[19] Cisco Systems, "Cisco data center infrastructure 2.5 design guide", http://www.cisco.com/univercd/cc/td/doc/solution/dcidg21.pdf

[20] T. Benson, A. Anand, A. Akella, and M. Zhang, "Understanding Data Center Traffic Characteristics, " in *Proceedings of Sigcomm Workshop: Research on Enterprise Networks*, 2009.

[21] S. Kandula et al., "The Nature of Data Center Traffic: Measurements and Analysis, " in *the Internet Measurement Conference*, 2009.

[22] Albert Greenberg, et al., "VL2: A Scalable and Flexible Data Center Network, " in *Proceedings of SIGCOMM* 2009.

[23] Y. Zhang, M. Roughan, N. Duffield, A. Greenberg, "Fast Accurate Computation of Large-Scale IP. Traffic Matrices from Link Loads, " in *ACM SIGMETRICS*, 2003.

[24] Ho et al., "Traffic Engineering for Multiple Spanning Tree Protocol in Large Data Centers", http://becool.info.ucl.ac.be/page/traffic-engineering-multiple-spanning-tree-protocol-large-data-centers.