Load Balanced Spanning Tree in Metro Ethernet Networks

Ghasem Mirjalily*
Department of Electrical and Computer Engineering, Yazd University, Yazd, Iran
mirjalily@yazd.ac.ir
Samira Samadi
Department of Electrical and Computer Engineering, Yazd University, Yazd, Iran
s.samadi@stu.yazd.ac.ir

Received: 28/Sep/2013 Accepted: 12/Apr/2014

Abstract
Spanning Tree Protocol (STP) is a link management standard that provides loop free paths in Ethernet networks. Deploying STP in metro area networks is inadequate because it does not meet the requirements of these networks. STP blocks redundant links, causing the risk of congestion close to the root. As a result, STP provides poor support for load balancing in metro Ethernet networks. A solution for this problem is using multi-criteria spanning tree by considering criterions related to load balancing over links and switches. In our previous work, an algorithm named Best Spanning Tree (BST) is proposed to find the best spanning tree in a metro Ethernet network. BST is based on the computation of total cost for each possible spanning tree; therefore, it is very time consuming especially when the network is large. In this paper, two heuristic algorithms named Load Balanced Spanning Tree (LBST) and Modified LBST (MLBST) will be proposed to find the near-optimal balanced spanning tree in metro Ethernet networks. The computational complexity of the proposed algorithms is much less than BST algorithm. Furthermore, simulation results show that the spanning tree obtained by proposed algorithms is the same or similar to the spanning tree obtained by BST algorithm.

Keywords: Metro Ethernet Network, Spanning Tree, Load Balancing, Shortest Path Selection.

1. Introduction

Ethernet technology is widely accepted in enterprise deployments and it is said that today more than ninety percent of data traffic is Ethernet encapsulated. Currently, the Ethernet technology is applicable in all levels from local area to metropolitan and wide area network environments [1].

In an Ethernet network, only one active path can exist between two nodes, because multiple active paths create loops in the network. Existence of the loops in the network topology confuses the forwarding and learning algorithms. Current Ethernet networks rely on IEEE STP [2] or IEEE Rapid Spanning Tree Protocol (RSTP) [3]. These are link management protocols that provide path redundancy while preventing undesirable loops in the network.

In both STP and RSTP, all of the traffic will be routed on the same spanning tree. Furthermore there isn't any mechanism for load balancing. These result in unbalanced load distribution and bottlenecks, especially close to the root and therefore, cause inefficient utilization of resources in metropolitan area networks [4].

Traffic engineering in metro Ethernet networks is a widely researched topic and some improvements have been proposed in the literature in order to solve this problem [5-8].

SmartBridge [5] is a bridged network architecture that addresses the problems associated with spanning trees in Ethernet networks. In SmartBridge architecture, packets will be forwarded along the shortest paths. Although shortest path provides low latency, it does not solve the problem of load balancing in the network. Furthermore, in this architecture, all of the bridges must be SmartBridge compliant.

K. Lui et. al [6] propose an approach named STAR (Spanning Tree Alternate Routing). STAR finds paths that are shorter than their corresponding tree paths; therefore it reduces latency between source and destination pairs. However, STAR is complex and it risks overloading of critical links.

Tree-Based Turn-Prohibition (TBTP) [7] is another approach to load balancing in Ethernet networks. TBTP constructs a spanning tree by blocking some pairs of links around nodes, such that all cycles in the network will be broken. However, TBTP does not consider the best spanning tree and switch load balancing.

Multiple Spanning Tree Protocol (MSTP) [8] is another related standard that is defined in IEEE 802.1s. MSTP improves the load balancing and failure recovery capabilities. However, maintaining multiple spanning trees adds a large complexity to network management and causes high control overheads.

In our previous works, some solutions to the problem of load balancing in metro Ethernet networks have been proposed. In [9], a novel algorithm is introduced to select the Best Spanning Tree (BST) for a given network topology based on the load balancing criterions. BST algorithm finds the best spanning tree by calculating the defined score for each possible tree and by selecting the spanning tree with highest score as the Best Spanning

* Corresponding Author

Tree. The defined score is a function of shortest path criterion and load balancing on links and switches. In summary, BST algorithm is an exhaustive search algorithm that finds the Best (Optimal) spanning tree. This is done by finding all spanning trees of the network, evaluating them based on the defined criteria and then selecting one with greatest score. In [10], a Multiple BST (MBST) approach is proposed that is the application of BST algorithm in a multiple spanning tree scheme. MBST considers all of the possible edge-disjoint spanning trees and all of the possible VLANs grouping and finds the best solution. Note that a set of spanning trees are edge-disjoint if they have not any common edges (links). Using edge-disjoint spanning trees enhances load balancing and resiliency, because in the case of a link failure, only one spanning tree will be failed and the failure has no impact on the spanning trees not including the failed link.

Although, BST and MBST algorithms can find the best answer for small networks, but their complexity is too large in large-scale networks.

In [11], in order to reduce the complexity of BST algorithm, a simple approach that finds the sub-optimal spanning tree in small metro Ethernet networks is introduced. In [12], the algorithm is improved by introducing Shortest Path Selection criterion. The new algorithm is applicable in realistic large-scale metro Ethernet topologies. In this paper, that is an extended version of [12], we introduce Load Balanced Spanning Tree (LBST) algorithm. LBST is a simple algorithm that finds the load balanced spanning tree by using the same criterions used in BST algorithm. The computational complexity of LBST algorithm is much less than BST. LBST simplifies the process of finding spanning tree by using an iterative algorithm with zero initial values for link loads. Although setting zero is the simplest way for specifying the initial values, but it is very far from the real values and this can degrades the performance of the LBST algorithm. In order to improve the performance of the algorithm, a modified version of the algorithm called Modified LBST (MLBST) will be introduced. In MLBST, more accurate estimation of the initial values of link loads will be used.

In fact, LBST and its modified version MLBST find a near-optimal spanning tree for a given metro Ethernet network in a simple manner based on shortest path criterion and load balancing on links and switches. In this way, three major criterions are introduced: load balancing over links, load balancing on switches and shortest path selection. Also, three coefficients α, β and γ corresponding to above criterions are defined. This allows the network managers to weight the importance of each criterion based on their defined goal. These criterions and corresponding coefficients will be used to assign weights to the links and then algorithm finds the shortest path between each node pair by using Dijkstra's algorithm [13]. During this process, the link weights must be updated to reflect the effects of adding new traffic demands at each step.

Although the constructed spanning tree may be not the best, but it is minimum weight and is a good approximation to the BST result.

The rest of the paper is organized as follows: Section 2 introduces some definitions and notations. In section 3, the LBST algorithm is explained in detail. In Section 4, some numerical simulation results are presented. Section 5 proposes a modified version of LBST algorithm, and finally, some conclusions are drawn in section 6.

2. Definitions and Notations

In this paper, a metro Ethernet network is modeled by a graph with N nodes representing switches and a set of M links connecting nodes. Here, for simplicity assume symmetric links and symmetric traffic demands. The traffic demands between nodes are represented by a N-by-N matrix D, that its component $d_{i,j}$ $(i, j = 1, 2, \ldots N, i \neq j)$ represents the mean traffic rate between nodes i and j. Also $b_{i,j}$ $(i, j = 1, 2, \ldots N, i \neq j)$ represents the bandwidth of the link between nodes i and j and c_j $(i = 1, 2, \ldots N)$ represents the switching capacity of ith node.

The goal of this paper is finding the best spanning tree based on three defined criterions. These criterions are links load balancing (LLB), switches load balancing (SLB) and shortest path selection (SPS). Here, the shortest path between two nodes is defined as a path with maximum aggregated bandwidth and minimum hop counts.

In this work, three coefficients α, β and γ, are defined to indicate the importance of each criterion. Note that $0 \leq \alpha, \beta, \gamma \leq 1$ and $\alpha+\beta+\gamma=1$. For example, if the main criterion is LLB, assign $\alpha=1, \beta=\gamma=0$, but if the goal is to find the best tree based on LLB and SPS criterions but not SLB, assign $\alpha=0.5, \gamma= 0.5, \beta=0$.

In the proposed algorithm, the link weight is a major component in finding the best spanning tree. Here, the link weight is defined as a linear function of three components: the utilization of the link, the average utilization of switches that the link is between them, and the inverse of normalized bandwidth of the link. Therefore, one can write the link weight between nodes i and j ($W_{i,j}$) as:

$$W_{i,j} = \alpha u_{l_{i,j}} + \beta u_{s_{i,j}} + \gamma u_{b_{i,j}} . \quad (1)$$

In above Equation,

$$u_{l_{i,j}} = \frac{l_{i,j}}{b_{i,j}} \quad (2)$$

is the utilization of the link between nodes i and j and $l_{i,j}$ is the traffic flowing through it. Also,

$$u_{s_{i,j}} = \frac{1}{2}\left(u_{s_i} + u_{s_j}\right), \quad (3)$$

is the average utilization of switches i and j, where:

$$u_{s_i} = \frac{s_i}{c_i} , u_{s_j} = \frac{s_j}{c_j} . \quad (4)$$

Here s_i and s_j are the traffics flowing through switches i and j respectively. Also,

$$u_{b_{i,j}} = \frac{b_{min}}{b_{i,j}} \quad (5)$$

is the inverse of normalized bandwidth of the link between nodes *i* and *j*, where b_{min} is bandwidth of the link with minimum bandwidth in the network. Note that the b_{min} must be selected over all active links of the network.

For a network graph with *N* nodes, each spanning tree has *N-1* links. For each spanning tree, the variance of link utilizations (σ_l^2) can be defined as [9]:

$$\sigma_l^2 = \frac{1}{N-1}\sum_{k=1}^{N-1}\left(\frac{l_k}{b_k} - \bar{l}\right)^2 , \quad (6)$$

where

$$\bar{l} = \frac{1}{N-1}\sum_{k=1}^{N-1}\frac{l_k}{b_k} \quad (7)$$

is the average of link utilizations, l_k is traffic load on kth link and b_k denotes the bandwidth of *k*th link. For each spanning tree, σ_l^2 indicates the degree of link load balancing, therefore in LLB criterion, the main goal is to find a spanning tree with minimum σ_l^2.

Similar to (6), for each spanning tree, the variance of switch utilizations (σ_s^2) can be defined as [9]:

$$\sigma_s^2 = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{s_i}{c_i} - \bar{s}\right)^2 , \quad (8)$$

where

$$\bar{s} = \frac{1}{N}\sum_{i=1}^{N}\frac{s_i}{c_i} \quad (9)$$

is the average of switch utilizations, s_i is traffic load cross *i*th switch and c_i denotes the switching capacity of *i*th switch. For each spanning tree, σ_s^2 indicates the degree of switch load balancing, therefore in SLB criterion, the goal is to find a spanning tree with minimum σ_s^2.

In SPS criterion, the goal is to find a spanning tree with maximum bandwidth links and minimum hop count paths. In this way, the parameter *L* is defined as:

$$L = \frac{\sum_{k=1}^{N-1} l_k}{\sum_{k=1}^{N-1} b_k} . \quad (10)$$

The goal is to select the spanning tree with minimum *L*. To clarify the definition of parameter *L*, note that $\sum_{k=1}^{N-1} l_k = \sum_{i=1}^{N}\sum_{j=1, j\neq i}^{N} d_{ij}h_{ij}$ indicates the aggregated load on links. Where h_{ij} is the number of hops from node *i* to node *j* and d_{ij} is the traffic demand between these two nodes that is a fixed known value. It is clear that if traffic demands pass on shortest (minimum hop count) paths, $\sum_{k=1}^{N-1} l_k$ will be minimum. On the other hand, $\sum_{k=1}^{N-1} b_k$ indicates the aggregated bandwidth of links. Therefore, minimizing *L* guarantees each traffic demand travels on shortest path with maximum bandwidth and minimum hop count.

The proposed algorithm also uses a parameter named Minimum Function (MF) to break loops. This parameter is defined for a given spanning tree as:

$$MF = \alpha\sigma_l^2 + \beta\sigma_s^2 + \gamma L . \quad (11)$$

In next section, the details of the new algorithm are explained.

3. LBST Algorithm

In previous section, some definitions that will be used here are described. This section describes the proposed algorithm named LBST in detail. This algorithm is simple and easy to implement in comparison to BST that is a computationally complex algorithm.

LBST algorithm first sorts node pairs based on their traffic demand and builds shortest path for each pair sequentially.

Here the highest-demand-first technique will be used because inserting lower traffic demand into the network can be done without loss of much performance [4]. After finding the best path for each node pair, the algorithm loads the traffic on this path and then updates the weights of links on the path. This process will be continued for other node pairs in descending order of their traffic demands.

The LBST algorithm is implemented in two different methods. In first version named LBST I, loops will be broken step by step. The steps of this algorithm are described as follows:

1. Assign the initial value of link weights using Equations (1)-(5). The initial spanning tree is a null graph. Note that in the first step, there is no load on links and switches, therefore $u_{l_{i,j}}$ is zero for all links, but about the switches, algorithm uses the demand matrix to find the initial load on switches. To do this, add the traffic demands that the *i*th switch is their origination and assign the result to calculate the initial value of u_{s_i} by using Equation (4).
2. Sort the node pairs based on the traffic demands in descending order and set *k=1*.
3. Select the *kth* node pair and find the shortest path between them.
4. Check whether or not the links and switches of this path have enough capacity. If yes then load the corresponding traffic demand on path. If not, select next shortest path, then go back to step 4.
5. Concatenate the discovered path to the spanning tree.
6. Update the link loads and switch loads by adding the corresponding traffic demand to the loads of the links and switches located on the path. Then, update the link weights according to the Equations (1)-(5).
7. Check whether is any loop in the constructed tree or not. If not go to step 8, else break the loop:
   a. For selected loop, determine the links which formed the loop.
   b. For each link check if this link is removed, do other links and switches in the loop have enough capacity to handle the excess traffic? If yes, then this link is a candidate, otherwise this link cannot be removed.

c. If there is not any candidate link, ignore this path and select next shortest path, then go back to step 4. Otherwise, by having the list of candidate links for removal, remove the link which contributes the lowest MF to the tree. In other words, after removing each link in the loop, you get a different tree. Now keep the tree with the lowest Minimum Function defined in Equation (11).
d. Update parameters and go back to step 7.

8. Set $k=k+1$ and go back to step 3. Continue this process for all node pairs.

Note that if during the process, some links or switches are fully used; in the rest, ignore them in selecting the best paths.

As the first issue, breaking loops step by step in LBST I is a time consuming process that decreases the speed of the algorithm considerably. One approach to speed up the algorithm is breaking all the loops at the end of the algorithm instead of breaking them step by step. The new approach is named LBST II algorithm. It is clear that the speed of the LBST II is much more in comparison to the LBST I. The steps of the LBST II algorithm are described as follows:

1. Assign the initial value of link weights using Equations (1)-(5) as described in LBST I algorithm. The initial spanning tree is a null graph.
2. Sort the node pairs based on the traffic demands in descending order and set $k=1$.
3. Select the *kth* node pair and find the shortest path between them.
4. Check whether or not the links and switches of this path have enough capacity. If yes then load the corresponding traffic demand on path. If not, select next shortest path, the go back to step 4.
5. Concatenate the discovered path to the spanning tree.
6. Update the link loads and switch loads by adding the corresponding traffic demand to the loads of the links and switches located on the path. Then, update the link weights according to the Equations (1)-(5).
7. Set $k=k+1$ and go back to step 3. Continue this process for all node pairs.
8. Check whether is any loop in the constructed tree or not. If not go to step 9, else break the loop:
   a. For selected loop, determine the links which formed the loop.
   b. For each link in the selected loop, check if this link is removed, do other links and switches in the loop have enough capacity to handle the excess traffic? If yes, then this link is a candidate, otherwise this link cannot be removed.
   c. By having the list of candidate links for removal, remove the link which contributes the lowest MF to the tree. In other words, after removing each link in the loop, you get a different tree. Now keep the tree with the lowest MF. If there is not any candidate link, ignore this sub-step and go to sub-step d. (Note that in very rare situations, the algorithm may be not able to break loops. In these rare cases, we must use LBST I algorithm to find the solution).
   d. Update parameters and go back to step 8.
9. End.

The proposed algorithms are trying to find the shortest possible paths between nodes and in the same time, they are trying to balance the utilization of link bandwidths and switching capacities.

4. Simulation Results

In this section, the performance of the proposed algorithms will be compared with BST algorithm proposed in [9]. The algorithms are implemented in MATLAB. The input parameters for a network with N nodes and M links are network graph, bandwidth vector B which elements are $b_{i,j}$ $(i=1,2,...,N)$, switching capacity vector C which elements are c_i $(i=1,2,...,N)$, traffic demands matrix D which elements are $d_{i,j}$ $(i=1,2,...,N, d_{i,i}=0)$, and α , β, γ $(0\le\alpha,\beta,\gamma\le1,\ \alpha+\beta+\gamma=1)$.

For simulation, a typical popular topology for metro Ethernet networks [14], is considered. A metro Ethernet network usually consists of a core part and several aggregation and access regions. The task of the core part is to forward the traffic load of the aggregation regions toward the edge nodes. The shape of the core is usually one or more rings formed by high speed switches and links. The aggregation part aggregates the traffic of access parts to several internal switches that are connected to the core rings. For aggregation part, usually topologies such as rings or dual homing structures are used. The access parts are usually tree shaped, because the cost of building the interconnections are high.

Figure 1 shows a typical metro Ethernet network. Here, the core consists of four switches with switching capacity of 8 Gbps interconnected to a ring formed by 2 Gbps Ethernet links. Also, two edge nodes with switching capacity of 8 Gbps are connected to core switches with 2 Gbps links and four aggregation nodes with switching capacity of 4 Gbps are connected to two core nodes using dual homing with 1 Gbps links. In this topology, there are eight access nodes with switching capacity of 1 Gbps are connected to aggregation switches with 1Gbps links.

In this typical network, for simplicity consider constant bit rate traffic demands between access nodes and edge nodes. These bidirectional traffic flows are shown in Table 1 where notation “Ac” stands for Access.

As you can see from Figure 1, the physical structure of the access part is tree. Therefore, only the core, aggregation and edge parts of the network are considered in the load balancing algorithm as shown in Figure 2. In this figure, the labels indicated on nodes are switch names; where, the notation “Ed”, “Co”, and “Ag” stands for Edge, Core and Aggregation, respectively.

First consider LLB criterion ($\alpha=1$, $\beta=\gamma=0$). The spanning trees selected by BST, LBST I and LBST II

algorithms are shown in Figures 3.a, 3.b and 3.c, respectively. In this case, the variance of link utilizations (σ_l^2) is 0.0133 for BST tree, 0.0230 for LBST I tree and 0.0179 for LBST II tree. The selected spanning tree by LBST I is ranked third best (3rd) tree by BST algorithm and selected spanning tree by LBST II is ranked second best (2nd) one.

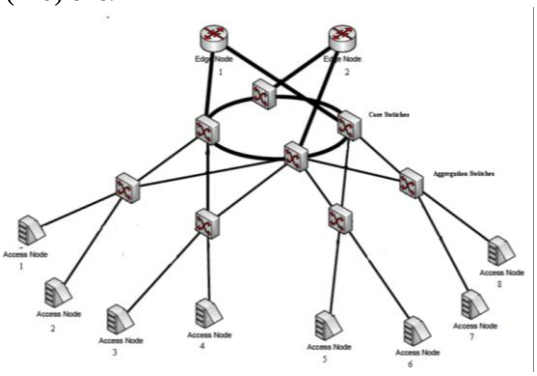


Fig. 1. A typical metro Ethernet network.

Table 1. Traffic demands (Mbps)

	Ac1	Ac2	Ac3	Ac4	Ac5	Ac6	Ac7	Ac8
Edge1	50	50	100	200	200	200	200	100
Edge2	100	100	300	300	200	100	100	200

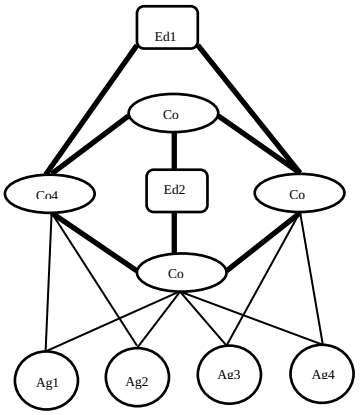
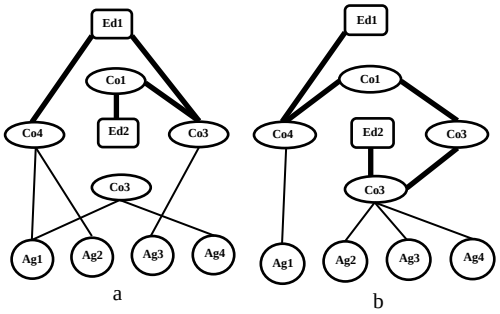


Fig. 2. Metro Ethernet network graph.



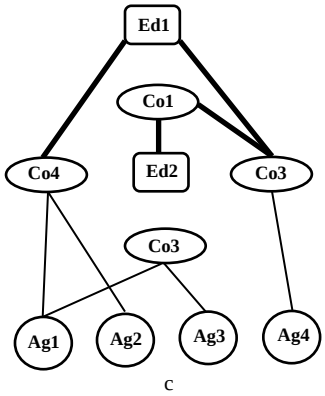


Fig. 3. Spanning tree selected based on LLB criterion by: a) BST algorithm b) LBST I algorithm c) LBST II algorithm.

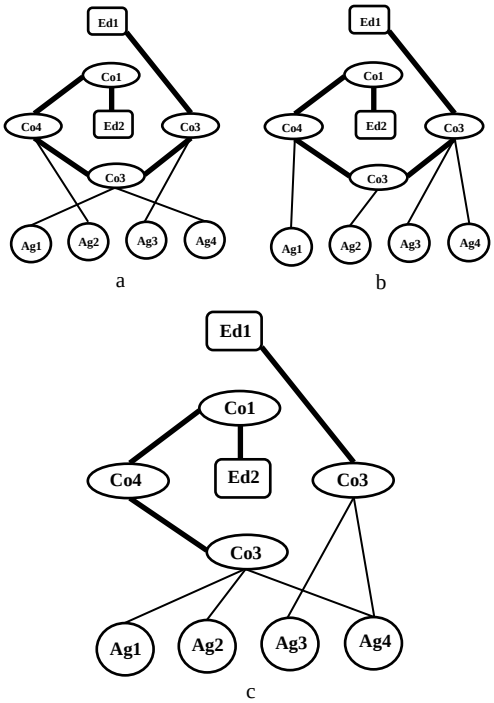


Fig. 4. Spanning tree selected based on SLB criterion by: a) BST algorithm b) LBST I algorithm c) LBST II algorithm.

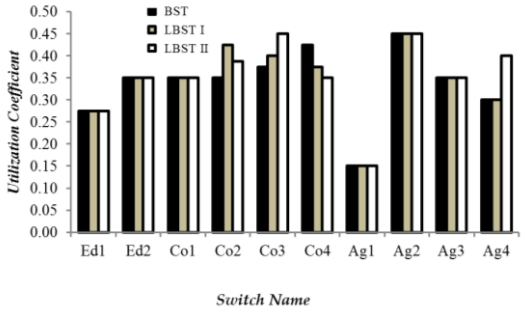


Fig. 5. Utilization coefficient of switches for SLB criterion.

Now consider SLB criterion (α=0, β=1, γ=0). The spanning trees selected by BST, LBST I and LBST II algorithms are shown in Figures 4.a, 4.b and 4.c respectively. In this case, the variance of switch utilizations (σ_s^2) is 0.0063 for BST tree, 0.0066 for LBST I tree and 0.0069 for LBST II tree. The selected spanning tree by LBST I is ranked third best (3rd) tree by BST algorithm and selected spanning tree by LBST II is ranked fourth best (4th) one.

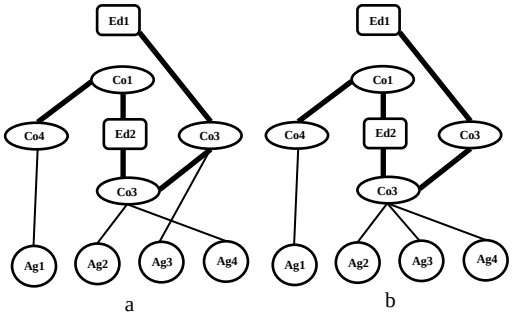


Fig. 6. Spanning tree selected based on SPS criterion by: a) BST algorithm b) LBST I and LBST II algorithms.

The utilization coefficients of switches for SLB criterion are shown in Figure 5. This Figure shows that the load balancing obtained by using LBST I and LBST II algorithms is very close to the results obtained by using BST algorithm.

In last scenario, consider SPS criterion (α=β=0, γ=1). The selected spanning trees are shown in Figure 6. As a numerical comparison, the trees selected by LBST I and LBST II algorithms are the same and are ranked third best (3rd) tree by BST algorithm with *L* equal to 0.471 while this value is 0.464 for the best spanning tree selected by BST. These values are very close to each other.

From above simulation results, the following statements can be concluded:

Although LBST algorithms are not the best but their results are the same or similar to the results obtained by using BST algorithm.

The computational complexity of LBST algorithms is much less than BST algorithm. As a roughly comparison, the typically run time of BST algorithm for a network with tens of switches and links on a new high speed computer is tens of minutes, while the run time of our new approaches for the same network is only several seconds.

LBST II algorithm breaks all of the loops in the last step, while the LBST I algorithm breaks loops step by step. Therefore, LBST II is much faster than LBST I. Furthermore, simulation results show that the output of LBST II is the same or similar to the output of LBST I algorithm.

5. Modified LBST Algorithm

As described in section 3, the initial values of link weights in the first step of the LBST algorithms are assigned using Equations (1)-(5). In the beginning, there is no load on links, therefore in simulations done in previous section, the initial value of $u_{l_{i,j}}$ was set to zero for all links. About the switches, algorithm uses the demand matrix to find the initial load on switches. To do this, add the traffic demands that the *i*th switch is their origination and assign the result to calculate the initial value of u_{s_i} by using Equation (4). Although this is a simple method for specifying the initial values, but it is far from the real values. In this section, we want to study the effects of initial values on the performance of the LBST algorithm. In this way, a different method for calculating the initial values of link loads and switch loads is introduced. This modification enables the algorithms to estimate the initial link weights with more accuracy. For future references, name the new algorithm, Modified LBST (MLBST).

Note that MLBST is useful for LLB and SLB criterions. For SPS criterion, the output of LBST and MLBST algorithms are the same.

In the following, the new algorithm is described, and then by driving some simulations, the effectiveness of using accurate initial values on the performance of the algorithm is showed.

As mentioned in previous section, the output of LBST II is the same or similar to the output of LBST I algorithm, but its computational complexity is less. For this reason, in the rest, only the LBST II algorithm is considered.

The MLBST algorithm, first sets the initial link loads to zero and then runs the LBST II algorithm described before once without loop breaking (by ignoring the 8th step). After that, the traffic loads on links and switches are known. By using this information, algorithm obtains the link weights using Equations (1)-(5). Now, using these new initial values, it runs the LBST II algorithm again without loop breaking. This process can be repeated several times. In the last run, the MLBST algorithm uses LBST II algorithm exactly as described before (without ignoring the 8th step).

For performance evaluation of MLBST, the simulation scenarios described in previous section are run again by using MLBST. The spanning tree obtained based on LLB criterion with one run for obtaining initial values is ranked second best (2nd) tree by BST algorithm with σ_l^2 equal to 0.0179, while the spanning tree obtained by using two runs for obtaining initial values is the Best Spanning Tree that is shown in Figure 3.a.

By repeating the simulation for SLB criterion, the obtained tree for MLBST with just two runs for obtaining initial values, is the Best Spanning Tree.

The above results show that MLBST algorithm can find the best spanning tree by repeating the algorithm steps several times.

6. Conclusions

In this paper, first a new simple approach named LBST algorithm is introduced for finding the best load balanced spanning tree in metro Ethernet networks. LBST is an iterative algorithm that finds the spanning tree based on load balancing on links and switches. The criterions used are the same as criterions used in our previous algorithm named BST. BST algorithm is an exhaustive search algorithm that finds the Best Spanning Tree. Although, BST algorithm can find the best answer but its complexity is too large in large-scale networks.

Simulation results showed that the output of LBST algorithm is close to the output of BST algorithm while its computational complexity is much less than it.

In the first step of the LBST algorithm, the initial value of link loads must be assigned. In LBST, for simplicity, these initial values were set to zero for all links. For performance improvement, the Modified version of LBST algorithm named MLBST is introduced. MLBST calculates the initial values of link loads and switch loads in a more accurate way. Simulation results showed that the MLBST algorithm can find the best spanning tree by repeating the algorithm steps several times. In summary, the following conclusions can be derived from simulation results:

1. The computational complexity of the proposed algorithms is much less than BST.
2. Although LBST algorithms are not the best but their results are the same or similar to the results obtained by using BST algorithm.
3. The performance of LBST II is very close to the performance of LBST I algorithm, but its computational complexity is much less.
4. MLBST can find the best spanning tree for LLB and SLB criterions by repeating the algorithm steps several times. For this reasons, in practice we prefer to use MLBST for finding the best spanning tree in metro Ethernet networks.

The proposed approaches can be used offline in the design process of a new metro Ethernet network or online during the operation of the network. In online mode, a central node (for example an edge node) is responsible for collecting the traffic and topology information and calculating the best spanning trees.

Note that even though our new algorithms have a good performance in the considered scenarios, other popular realistic traffic models such as multimedia traffic in more realistic metro Ethernet topologies can be considered in future works.

Acknowledgment

This work is supported by Iranian Telecommunication Research Center (ITRC).

References

[1] M. Huynh, and P. Mohapatra, "Metropolitan Ethernet network: A move from LAN to MAN," *Computer Networks*, Elsevier, pp. 4867-4894, 2007.

[2] CISCO SYSTEM, "Understanding spanning tree protocol," CISCO white paper, www.cisco.com, 1997.

[3] CISCO SYSTEM, "Understanding rapid spanning tree protocol," CISCO white paper, www.cisco.com, 2006.

[4] P.M.V. Nair, "Quality of service in metro Ethernet," *Ph.D. thesis*, Southern Methodist University, 2006.

[5] T. Rodeheffer, C. Thekkat, and D. Anderson,"Smartbridge: A scalable bridge architecture", *ACM Computer Communication Review*, Vol. 30, No. 4, pp. 205-218, 2000.

[6] K. Lui, W.C. Lee, and K. Nahrstedt, "STAR: a transparent spanning tree bridge protocol with alternate routing", *ACM Computer Communication Review* , Vol. 32, No. 3, pp. 33-46, 2002.

[7] F. De Pellegrini, D. Starobinski, M. G. Karpovsky, and L. B. Levitin, "Scalable, distributed cycle-breaking algorithms for gigabit Ethernet backbones", *Optical Networking*, Vol. 5, No. 2, pp. 122-144, 2006.

[8] IEEE 802.1s, *Standards for local and metropolitan area networks*, 2002.

[9] G. Mirjalily, H. Karimi, and S. Rajai, "Load balancing in metro Ethernet networks by selecting the best spanning tree", *Journal of Information Science and Engineering*, Vol. 27, No. 5, pp. 1747–1759, 2011.

[10] G. Mirjalily, F. Akhavan Sigari, and R. Saadat, "Best multiple spanning tree in metro Ethernet networks", Proc. of International Conference on Computer and Electrical Engineering, Dubai, UAE, pp. 117-121, 2009.

[11] S. Samadi, G. Mirjalily, and S.M.T. Almodarresi, "A simple algorithm to find the proper spanning tree in metro Ethernet networks", Proc. of International Conference on Software Technology and Engineering, KLL, Malaysia, pp. 185-189, 2011.

[12] S. Samadi, G. Mirjalily, "Load balanced spanning tree in metro Ethernet networks", Proc. of Iranian Conference on Electrical Engineering, Mashhad, Iran, 2013.

[13] Y. W. Bang, and C. Kun-Mao, "Spanning trees and optimization problems", Chapman & Hall, 2004.

[14] A. Kern, "Traffic-driven optimization of resilient QoS-aware metro Ethernet networks", *Ph.D. Thesis*, Budapest University of Technology and Economics, 2007.

Ghasem Mirjalily received his Ph.D. degree in Telecommunication Engineering from Tarbiat Modarres University, Iran in 2000. He has been a visiting researcher at the Communications Research Laboratory, McMaster University, Canada in 1998. Since 2000, he has been with Yazd University, Iran, where he is an associate professor. Dr. Mirjalily is senior member of IEEE. His current research interests include Traffic Engineering, Metro Ethernet Networks, and wireless Networks.

Samira Samadi received the B.S. degree in Electrical Engineering (Electronics) from Isfahan University of Technology (IUT) in 2006 with honors and M.S. degree in Electrical Engineering (Communication Systems) from Yazd University in 2011. Now, she is working on industrial Ethernet networks. Her interest is working on Data Communication Networks, especially on Traffic Engineering and Metro Ethernet Networks.