Improving Load Balance of Ethernet Carrier Networks using IEEE 802.1S MSTP with Multiple Regions

Amaro de Sousa, Gil Soares

Institute of Telecommunications, Department of Electronics and Telecommunications, University of Aveiro, 3810-193 Aveiro, Portugal, Emails: asou@det.ua.pt, gsoares@av.it.pt

With IEEE 802.1S Multiple Spanning Tree Protocol, an Ethernet operator can define different network regions. A Common Spanning Tree (CST) is defined in such a way that a link failure inside a region does not affect the CST outside the region and additional Spanning Trees inside each region can be configured to achieve better load balance. We propose a procedure to determine the MSTP parameters configuration with multiple regions that optimize load balance and show its efficiency through computational results. We compare multiple region solutions with single region solutions, using a previous work on the single region case, and show that the multiple regions approach is better when traffic is mainly between switches belonging to the same region.

1. Introduction

The IEEE 802.1D STP [1] and IEEE 802.1Q VLAN Protocol [2] are two well established protocols for Ethernet networks. STP routes demands based on a set of active links spanning all switches without cycles, i.e., a Spanning Tree. It includes detection of network changes and Spanning Tree recalculation to recover full connectivity. 802.1Q enables the assignment of traffic demands of different clients to different VLANs in order to prevent packets from one client to reach other client ports. Recently, two new protocols were proposed to enhance the survivability and traffic engineering capabilities of IEEE 802 switching networks. One is the IEEE 802.1W Rapid Spanning Tree Protocol [3], an evolution of STP where port states and roles are redefined and a negotiation mechanism is used to accelerate the convergence to a new Spanning Tree when the network changes. The other is the IEEE 802.1S Multiple Spanning Tree Protocol (MSTP) [4]. With MSTP, a network operator can define different network regions. A Common Spanning Tree (CST) connecting all switches of all regions is set-up in such a way that a link failure inside a region does not affect the CST outside the region. It enables also additional Multiple Spanning Trees to be configured inside each region although limited to support only internal VLANs (VLANs whose ports are in the same region). MSTP does not state how regions should be defined, which Spanning Trees should be created and how VLANs should be assigned to Spanning Trees. There is a trade-off between considering a single region or adopting multiple regions. In terms of failure recovery, the multiple regions case is superior. However, the additional Spanning Trees can only support internal VLANs which limits the load balancing that can be obtained. Previously, we have addressed in [5] the single region case. Other authors [6-8] have addressed the dynamic scenario where VLANs are dynamically requested and Spanning Trees are dynamically set-up and tear-down. In [9], authors address the problem of how to divide the network into regions. Other works propose MSTP as a means to improve the network support of other important aspects like mobility [10] and quality of service [11].

Given (a) a network composed by a set of switches connected through point-to-point links, (b) a set of regions defined on the network and (c) a set of VLANs, each one defined by a set of traffic flows, we determine (i) the appropriate MSTP parameters implementing the CST and the additional Spanning Trees and (ii) the assignment of VLANs to Spanning Trees. The aim is to optimize the network load balance. For any desired Spanning Tree, it is always possible to determine a set of MSTP parameters that makes active its links. Therefore, we propose a procedure that first determines the set of Spanning Trees (together with the mapping of VLANs to Spanning Trees) and, then, determines the MSTP parameters.

2. Solving Procedure

Consider an Ethernet carrier network composed by switches connected through point-to-point links. The network is represented by the directed graph $G = (N,A)$ where N is the set of switches and A is the set of directions (i,j) of all links (E is the set of links $\{i,j\}$). The bandwidth capacity of link $\{i,j\}$ is $b\{i,j\}$.

Consider r regions defined on the network, each one identified with a positive number i between 1 and r. Each region is defined by the sub-graph $G_i = (N_i, A_i)$ where $N_i \subset N$ is the set of switches and $A_i \subset A$ the set of direction of links belonging to region i. The network supports a set of VLANs represented by set V. Each VLAN $v \in V$ is characterized by a set of traffic flows $T(v)$ and each traffic flow $t \in T(v)$ is characterized by its origin switch $o(t)$, destination switch $d(t)$ and bandwidth demand $b(t)$.

For a particular set of Spanning Trees and a particular mapping of VLANs to Spanning Trees, each traffic flow is routed through the path defined in the Spanning Tree that its VLAN was assigned to. Assume that $a(v)$ indicates the Spanning Tree instance assigned to VLAN v. Assume a binary parameter $s[(i,j), t]$ that is one if arc (i,j) is in the path of traffic flow $t \in T(v)$ defined by the Spanning Tree instance $a(v)$ assigned to its VLAN v. The load on arc (i,j) is the sum of the demands of all traffic flows that use it. Consider $l(i,j)$ the resulting load (in percentage):

$$l(i,j) = \frac{\sum_{v \in V} \sum_{t \in T(v)} \left(b(t) s[(i,j), t] \right)}{b\{i,j\}} \times 100\% \qquad (1)$$

We define the load array L of a particular solution, the array which is formed by all $l(i,j)$ values sorted in a non increasing order: first element of L is the highest load value; second element is the second highest load value, and so on… In the remaining of this paper, load array L_G is the set of non-increasing link loads on graph G and load array L_{Gi} is the set of non-increasing link loads on sub-graph G_i. Load arrays are used to compare the load balance between different solutions: for two given solutions 1 and 2 whose load arrays are L_1 e L_2, we consider solution 1 better than solution 2 if L_1 has a smaller value than L_2 in the first array position whose values of both arrays are different.

Consider the following additional notation. The set of links forming the CST is given by ϖ where the links inside region i are denoted by ϖ_i and the links outside regions are denoted by ϖ_0 ($\varpi = \varpi_0 \cup \varpi_1 \cup \ldots \cup \varpi_r$). Inside region i, besides the CST ϖ_i, there are n additional Spanning Trees, each one denoted by ϖ_{ij} where $j = 1 \ldots n$. The set of additional Spanning Trees inside region i is denoted by Ω_i ($\Omega_i = \varpi_{i1} \cup \ldots \cup \varpi_{in}$). Array Φ is a VLAN to Spanning Tree assignment array with index $v = 1 \ldots |V|$ where its v^{th} position is $a(v)$, the Spanning Tree assigned to VLAN v. Array Φ is decomposed in r arrays where Φ_i , $i = 1 \ldots r$, refers to the VLANs internal to region i (VLANs not internal to any region are assigned to CST ϖ). In the following description, $\underline{L_G}$ and $\underline{L_{Gi}}$ are the best load arrays obtained respectively in Step 1 and Step 2, $\underline{\varpi}$ is the set of CST links in the best solution, $\underline{\Omega_i}$ is the set of n additional Spanning Trees of region i in the best solution and $\underline{\Phi}$ is the VLAN to Spanning Tree assignment array of the best solution. The proposed procedure is composed of three steps (presented in the next box).

```
1:  Set all values of L_G to +∞                                   // Begin of Step 1
2:  while Number of Iterations < MaxMain do:
3:          ϖ ← GenerateCST(G)
4:          (L_G , ϖ) ← ImproveCST(ϖ)
5:          if L_G is better than L_G do:
6:              L_G ← L_G , ϖ ← ϖ                                 // End of Step 1
7:  for i = 1…r do:                                               // Begin of Step 2
8:          Set all values of L_Gi to +∞
9:          while Number of Iterations < MaxRegion do:
10:             (ϖ_i , Ω_i) ← GenerateMST(n+1 , G_i)
11:             (L_Gi , Φ_i) ← AssignVLANtoTrees(ϖ_i , Ω_i)
12:             if L_Gi is better than L_Gi do:
13:                 L_Gi ← L_Gi , Ω_i ← Ω_i , Φ_i ← Φ_i , ϖ_i ← ϖ_i   // End of Step 2
14: DetermineCSTParameters(ϖ , G)                                 // Begin of Step 3
15: for i = 1…r do:
16:     for j = 1…n do:
17:         DetermineSTParameters(ϖ_ij , G_i)                     // Begin of Step 3
```

In Step 1, we determine the CST set of links optimizing the resulting L_G array. Step 1 procedure runs *MaxMain* iterations (**while** cycle from line 2 to 6). On each iteration, it generates one Spanning Tree over the graph G (procedure *GenerateCST*); it then improves this Spanning Tree with respect to the link load array L_G (procedure *ImproveCST*) and, if the resulting load array L_G is better than the best load array $\underline{L_G}$ (line 5), it saves the present Spanning Tree as the best solution found so far (line 6). In step 2, we solve each region separately; we determine the set of links of CST inside the region and of all additional Spanning Trees optimizing the resulting L_{Gi} array. Step 2 procedure is executed one time for each region

(**for** cycle from line 7 to line 13). For each region, Step 2 procedure runs *MaxRegion* iterations (**while** cycle from line 9 to 13). On each iteration, it generates a set of n+1 Spanning Trees over graph G_i (procedure *GenerateMST*); it determines an assignment array Φ_i indicating the Spanning Tree assigned to each VLAN, together with the resulting load array L_{Gi} (procedure *AssignVLANtoTrees*) and if load array L_{Gi} is better than the best load array $\underline{L}_{Gi}$ (line 12), it saves the present set of Spanning Trees as the best solution found so far (step 13). In Step 3, we determine the appropriate MSTP protocol parameters. The procedure determines the MSTP parameters for the CST set of links over graph G (procedure *DetermineCSTParameters*) and for all additional Spanning Trees (line 16) of all regions (line 15) over the corresponding graph G_i (procedure *DetermineSTParameters*). In the remaining of this section, the elementary procedures defined on each step are separately described.

There are two procedures for the generation of Spanning Trees. The aim of procedure *GenerateCST*(G) in Step 1, is to randomly generate a set of links ϖ that can define a CST. In order to agree with MSTP, this set of links must be an in-region Spanning Tree, i.e., it should form a Spanning Tree on each sub-graph G_i (see example in Fig. 1). We generate a random spanning tree as follows. First, we assign all nodes with a different label. Then, we repeat $|N| - 1$ times the following operations: (i) select randomly one link among all links whose end-nodes have different labels and (ii) assign all nodes with the label of one end-node with the label of the other end-node. In order to guarantee the in-region property, procedure *GenerateCST*(G) first selects the links of each region and, then, proceed to the links outside all regions.

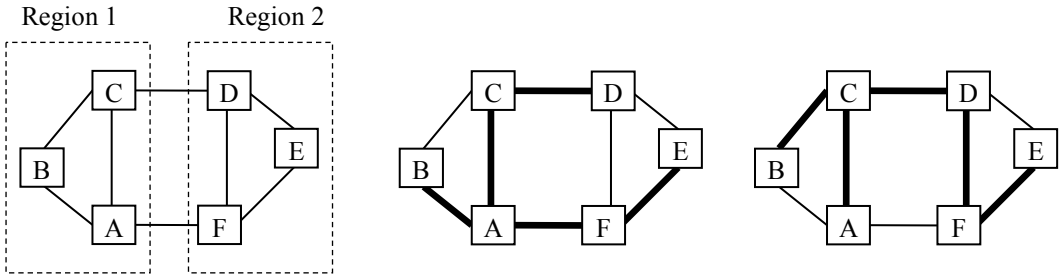

Fig. 1. In the middle, the thick links do not define a proper CST since they do not form a Spanning Tree inside Region 2. In the right, the thick links define a proper CST.

The aim of procedure *GenerateMST*(n+1,G_i) in Step 2, is to generate for the region defined by G_i a set of n+1 Spanning Trees avoiding common links. The Spanning Trees to be generated are ϖ_i (the CST part internal to region i) and the n internal Spanning Trees $\Omega_i = \varpi_{i1} \cup \ldots \cup \varpi_{in}$. The reason for avoiding common links is twofold: it helps splitting the traffic flows among more links, which results in a better load balanced network, and it minimizes the impact of link failures inside the region since a link failure only affects the traffic flows assigned to the Spanning Trees that use the failed link. We avoid common links in a greedy way: we generate randomly the first Spanning Tree; then, we generate randomly the second Spanning Tree using as much as possible the links not used in the first one; etc…

The aim of procedure *ImproveCST*(**ϖ**) in Step 1 is, based on the current Spanning Tree ϖ, to determine a better Spanning Tree ϖ and to determine its load array L_G. This procedure is based on a local search technique with the following neighbor definition: a Spanning Tree ω' is a neighbor of a given Spanning Tree ω if it differs from ω only in a single link. $V(\omega)$ designates the set of all neighbors of ω with the in-region Spanning Tree property. In the following description, ω and ω' are auxiliary Spanning Trees and $\underline{L}_G$ is the best load array obtained at the end of the procedure.

Procedure *ImproveCST*(**ϖ**)

```
1:  Determine L_G assigning all VLANs to ϖ
2:  repeat
3:      ω ← ϖ
4:      for (ω' ∈ V(ω)) do:
5:          Determine L_G assigning all VLANs to ω'
6:          if (L_G is better than L_G) do:
7:              L_G ← L_G , ϖ ← ω'
8:  until (ϖ equal to ω)
9:  return(L_G , ϖ)
```

The aim of procedure *AssignVLANtoTrees*(**ϖ_i**,Ω_i) in Step 2 is to determine a VLAN to Spanning Tree assignment array Φ that minimizes the resulting load array L_{Gi}. Note that from the previously determined

CST in Step 1, it is possible to determine the flows of external VLANs that cross region *i* and, for each of them, their incoming region node (if their origin node is outside the region) and their outgoing region node (if their destination node is outside the region). Consider V_{int} the set of internal VLANs. This procedure first generates a random assignment array and, then, performs a local search algorithm with the following neighbor structure: an assignment array $\Phi' = a'(1\ldots|V_{int}|)$ is a neighbor of a given $\Phi = a(1\ldots|V_{int}|)$ if it differs from Φ only in a single element. In the following description, Φ' and Φ'' are two auxiliary VLAN to Spanning Trees assignment arrays.

Procedure *AssignVLANtoTrees*(ϖ_i , Ω_i)

1: **for** ($v = 1\ldots|V_{int}|$) **do:**
2: $a(v) \leftarrow$ random($0\ldots n$)
3: Determine $\underline{L}_{Gi}$ assigning external VLANs to 0 and internal VLANs v according to Φ
4: **repeat**
5: $\Phi' \leftarrow \Phi$
6: **for** ($v = 1\ldots|V_{int}|$) **do:**
7: **for** ($p = 0\ldots n$ **and** $p \neq a(v)$) **do:**
8: $\Phi'' \leftarrow \Phi'$
9: $a''(v) \leftarrow p$
10: Determine L_{Gi} assigning external VLANs to 0 and internal VLANs v according to Φ''
11: **if** (L_{Gi} is better than $\underline{L}_{Gi}$) **do:**
12: $\Phi \leftarrow \Phi''$, $\underline{L}_{Gi} \leftarrow L_{Gi}$
13: **until** (Φ equal to Φ')
14: **return**(L_{Gi} , Φ)

There are two procedures for the determination of MSTP parameters. Each $(k,l) \in A_i$ has as associated forwarding port on switch k towards switch l whose *PortCost* must be determined. Given a desired Spanning Tree ϖ_{ij} over graph G_i and a current set of *BridgeID* and *PortCost* values, procedure *DetermineSTParameters*(ϖ_{ij},G_i) updates these values in order to make active the links of ϖ_{ij}:

1. Keep all *BridgeID* values unchanged. The switch with lowest *BridgeID* is the Root Bridge.
2. The forwarding ports in the path defined by ϖ_{ij} from every switch to the Root Bridge are root ports. Keep the *PortCost* values of root ports unchanged.
3. For each switch k, determine its root path cost c_k as the sum of the *PortCost* values of all root ports in the path from k to the Root Bridge.
4. For all non root ports of all links $(k,l) \in A_i$, assign a *PortCost* value equal to $c_k - c_l + 1$ if this value is lower than its current *PortCost* value; if not, let the current *PortCost* value unchanged.

The aim of procedure *DetermineCSTParameters*(ϖ,G) is similar to the previous one but its implementation is more complex. MSTP [4] defines that (i) the switch with lowest *BridgeID* becomes the Root Bridge, (ii) the root path cost of a path from any switch to the Root Bridge (if it is not in the same region as the Root Bridge) considers only the *PortCost* values of the root ports belonging to links outside regions (e.g., the *PortCost* values of links inside regions are considered NULL), (iii) at each region, the Bridge with lower root path cost to the Root Bridge becomes the Regional Root Bridge and (iv) inside each region, the active links are the ones in the minimum cost paths from each switch to its Regional Root Bridge. The following procedure minimizes the number of parameters to be updated:

1. Keep all *BridgeID* values unchanged. The switch with lowest *BridgeID* is the Root Bridge.
2. The forwarding ports in the path defined by ϖ from every switch to the Root Bridge are root ports. Keep the *PortCost* values of root ports unchanged.
3. For each switch k determine its root path cost c_k as the sum of the *PortCost* values of the root ports not internal to any region in the path from k to the Root Bridge.
4. For all non root ports of links (k,l) not internal to any region, assign a *PortCost* value equal to $c_k - c_l + 1$ if this value is lower than its current value; if not, let the current *PortCost* value unchanged.
5. For each region $i = 1\ldots r$ do:
   5.1. The switch belonging to region i with the lowest root path cost is the Regional Root Bridge.
   5.2. For each switch k of region i, determine its regional root path cost α_k as the sum of the *PortCost* values of all root ports in the path from k to the Regional Root Bridge.
   5.3. For all non root ports of links (k,l) internal to region i, assign a *PortCost* value equal to $\alpha_k - \alpha_l + 1$ if this value is lower than its current *PortCost* value; if not, let the current *PortCost* value unchanged.

3. Computational Results

The proposed procedure was implemented in C and run in a PC, 2.0 GHz Pentium 4 processor, 512 MB of RAM. The case studies consider the network shown in Fig.2. Concerning traffic demands, we have considered 51 VLANs (each VLAN with 2 traffic flows) randomly selected between all access switch pairs and all access-gateway pairs. Demand values were randomly selected among 4 different values (10, 20 50 and 100 Mbps) considering three different case studies: the percentage of total demand internal to regions is 20% for case study A, 50% for case study B and 80% for case study C.

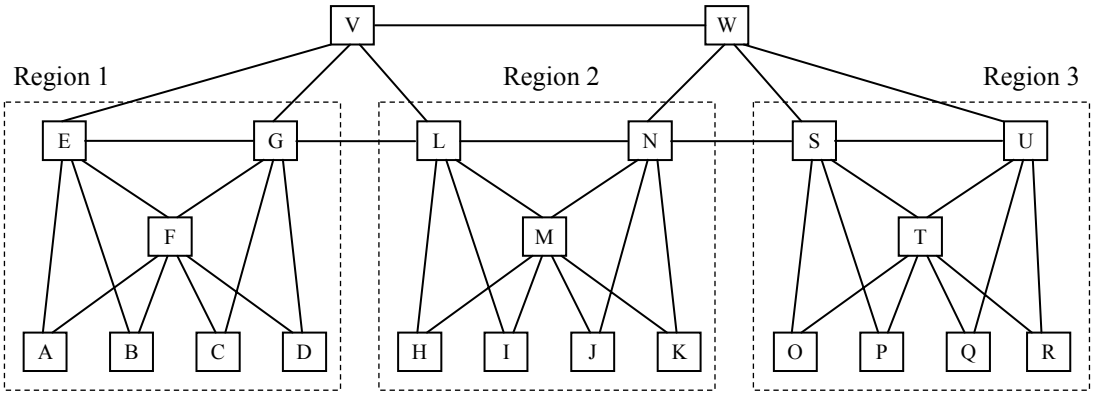

Fig. 2. An Ethernet network with 23 nodes, 42 links and 3 regions (all links have a capacity of 1 Gbps). Switches A to D, H to K and O to R are access switches where customer equipment is connected to and switches V and W are gateway switches that connect the Ethernet network to other core networks.

We have solved all three case studies considering a number *n* of additional Spanning Trees inside each region equal to 1 and 2. In all cases, the $n = 2$ case did not find a solution better than the best solution for the $n = 1$ case. Table 1 shows the obtained results. In case study A, most of the worst load values are on links outside the regions (as we can see in the $n = 0$ line) and since multiple Spanning Trees can only improve the load balance inside regions, there is only a minor improvement in using one additional Spanning Tree. On the other end, the additional Spanning Tree provides a significant improvement in case study C (worst link loads are decreased from 41% to 29%) since all worst load values in the single CST best solution correspond to links inside regions. Case study B is in the middle of the two previous results: the additional Spanning Tree enabled a small decrease of the worst link loads from 55% to 51%.

All cases were solved with both *MaxMain* and *MaxRegion* set to 10000. The procedure Step 1 part took at most 160 seconds with the best solutions always found within the first 10 iterations for all cases (below 0.1 seconds). The procedure Step 2 part took at most 2 seconds with the best solutions always found within the first 1000 iterations for all cases (below 0.2 seconds). These results show that the proposed procedure is very efficient and able to obtain solutions within short computing times.

Table 1. Highest 10 values of load array L_G for each case study; $n = 0$ corresponds to the best solution at the end of Step 1, i.e., the best CST configuration with no addtitional Spanning Trees; $n = 1$ corresponds to the best solution found with 1 additional Spanning Tree on each region. The values marked with * correspond to region internal links.

		Index of first 10 positions of Load Array (%)									
Case Study	*n*	1	2	3	4	5	6	7	8	9	10
A	0	76	76	70*	70*	68	68	58	58	46*	46*
	1	76	76	68	68	59*	59*	58	58	40*	40*
B	0	55*	55*	51	51	49	49	46*	46*	43	43
	1	51	51	49	49	43	43	36*	36*	23*	23*
C	0	41*	41*	39*	39*	34*	34*	32*	32*	32*	32*
	1	29	29	22*	22*	22*	22*	21*	21*	20*	20*

In order to compare both approaches, we have used the algorithm proposed in [5] to obtain the best solutions for the three case studies assuming the whole network as a single region (Table 2). Consider first the case study C. In this case, most of the traffic is internal to regions and the solution for the 3 regions approach shown in Table 1 (with a worst load value of 29%) is only slightly worst than the best solution with the single region approach shown in Table 2 (with a worst load value of 21%). In this case, the 3 region approach is a good solution since it can achieve a load balance almost as good as the optimal single region solution and it minimizes link failure impact on the network. In the A and B case studies,

the single region approach obtains significantly better load balanced solutions (worst link values decrease from 76% to 31% in case study A and from 51% to 23% in case study B). Note that in case study A, the 3 regions approach is even worst than the IEEE 802.1D STP solution (worst load is 70%) and this is because the STP case does not require a set of active links with the in-region Spanning Tree property. In these cases, there is a trade-off between load balance and link failure impact.

Table 2. Highest 10 values of load array L_G for each case study considering the whole network as a single region; the $n = 1$ case corresponds to the standard IEEE 802.1D STP protocol.

		Index of first 10 positions of Load Array (%)									
Case Study	*n*	1	2	3	4	5	6	7	8	9	10
A	1	70	70	68	68	59	59	58	58	52	52
	2	31	31	31	31	30	30	30	30	30	30
	3	31	31	31	31	30	30	26	26	25	25
B	1	55	55	51	51	46	46	43	43	40	40
	2	23	23	23	23	21	21	21	21	21	21
	3	23	23	23	23	20	20	20	20	20	20
C	1	41	41	39	39	34	34	32	32	32	32
	2	21	21	20	20	20	20	19	19	18	18
	3	21	21	20	20	20	20	19	19	17	17

As a final conclusion, the results shown in Table 2 show that near optimal solutions were obtained with 2 additional Spanning Trees and negligible gains were obtained with 3 additional Spanning Trees. Remember also that in the previous section 1 additional Spanning Tree was enough to obtain the best load balance. These observations show that it is possible to optimize load balance with a small number of Spanning Trees, thus, not penalizing significantly the switches processing overhead.

4. Conclusions

In this paper, we have addressed the problem of how to use the IEEE 802.1S MSTP to improve load balance in Ethernet carrier networks. We have addressed the multiple region case and we have proposed a procedure to determine the MSTP parameters configuration that optimizes network load balancing. The computational results show that this procedure is very efficient and able to obtain solutions within short computing times. We have compared how good the multiple region case is in terms of load balancing. We have showed that (i) the multiple regions approach is the best solution when traffic is mainly between switches belonging to the same region (minimizes link failure impact while achieving good load balance) but (ii) both approaches represent a trade-off between load balance and link failure impact when traffic is more uniformly distributed.

5. References

1. IEEE 802.1D, “Media Access Control (MAC) Bridges” (1998)
2. IEEE 802.1Q, “Virtual Bridged Local Area Networks” (1998)
3. IEEE 802.1W, “Part 3: Media Access Control (MAC) Bridges – Amendment 2: Rapid Reconfiguration” (2001)
4. IEEE Standard 802.1S, “Virtual Bridged Local Area Networks – Amendment 3: Multiple Spanning Trees (2002)
5. de Sousa, A.: Improving Load Balance and Resilience of Ethernet Carrier Networks with IEEE 802.1S Multiple Spanning Tree Protocol. 5th Int. Conference on Networking (ICN’06), Mauritius Islands (2006)
6. Ali, M., Chiruvolu, G., Ge, A.: Traffic Engineering in Metro Ethernet. IEEE Network Vol. 19 No. 2 (2005) 10–17
7. Kolarov, A., Sengupta, B., Iwata, A.: Design of Multiple Reverse Spanning Trees in Next Generation of Ethernet-VPNs. IEEE GLOBECOM’04, Dallas, USA Vol. 3 (2004) 1390–1395
8. Sharma, S., Gopalan, K., Nanda, S., Chiueh, T.: Viking: A Multi-Spanning-Tree Ethernet Architecture for Metropolitan Area and Cluster Networks. IEEE INFOCOM’04, Hong Kong, Vol. 4 (2004) 2283–2294
9. Padmaraj, M., Nair, S., Marchetti, M., Chiruvolu, G., Ali, M.: Traffic Engineering in Enterprise Ethernet with Multiple Spanning Tree Regions. Proc. of System Communications (ICW’05), Montreal, Canada (2005) 261–266
10. Ishizu, K., Kuroda, M., Kamura, K.: SSTP: an 802.1s Extension to Support Scalable Spanning Tree for Mobile Metropolitan Area Network. IEEE GLOBECOM’04, Dallas, USA Vol. 3 (2004) 1500–1504
11. Lim, Y., Yu, H., Das, S., Lee, S.-S., Gerla, M.: QoS-aware multiple spanning tree mechanism over a bridged LAN environment. IEEE GLOBECOM’03, San Francisco, USA Vol. 6 (2003) 3068–3072