From Spanning Trees to Meshed Trees

H. B. Acharya, John Hamilton, and Nirmala Shenoy
Rochester Institute of Technology

I. Introduction

Switching operations, at Layer 2 of the network protocol stack, are fundamental to network communications. The internal networks of data centers, as well as traditional Layer 2 networks – customer, service provider, and backbone provider networks [1], [2] – rely primarily on switching operations.

Over the last several decades, the performance and reliability demands on such networks has grown by orders of magnitude. Such demands have no doubt been supported through advances in robust and high-performance switch hardware, transmission and media technologies [3]. However, no matter how robust, network components still eventually fail, at which point they cause disruptions in network operation. At the time of failure, network *resilience* and *fast recovery* depend on having built-in redundancy in the network, and on fast, fault-tolerant Layer 2 protocols to take advantage of it.

A closer look reveals that there is a strong unmet need for such protocols. While existing protocols have been regularly updated [4], completely new protocols have not been introduced in many years. At the same time, the demands placed upon Layer-2 networks have grown enormously. This has led to workarounds, such as layer flattening and provisioning of massively redundant architectures.

Loop Avoidance

In this paper, we target one obvious bottleneck: the loop avoidance protocol in switched networks. Loop avoidance protocols are necessary to avoid *broadcast storms* in switched networks, which otherwise occur as follows.

- Switches deal with broadcast frames by forwarding them out through every available interface, other than the one at which they arrive.
- Loops in the network topology lead to the same frame being forwarded around the loop repeatedly. (At each switch the frame is forwarded out all outgoing ports, including the one taking it to the next switch in the loop. As forwarding is stateless – i.e. the switches do not remember whether this frame was seen before – they continue to forward it.)

In other words, a switched network should be loop-free, to avoid broadcast storms. However, it is hardly a good idea to build a physical network with such a topology (a tree), as it is highly non-resilient: the failure of any non-leaf node or edge is sufficient to disconnect a tree network. (More formally, a tree is 1-connected i.e., the minimum number of nodes we must remove to disconnect a tree is 1, and 1-edge connected, i.e. the minimum number of edges is 1 also.)

Loop-avoidance protocols allow the use of a more robust (i.e. multiply-connected) network, without the danger of broadcast storms. The standard approach, underlying the Spanning Tree Protocol, is to find a *logical spanning tree* in the network and to block all edges that do not belong to it. In case of failure, we need only reconfigure the tree logically rather than having to physically repair the network. However, the actual protocols as they exist now, are fast enough for customer-LAN networks, but *not* adequate for service and backbone provider networks (Metro Ethernet etc), or for the backplane in a Data Center.

Towards a New Protocol

We propose to attack the loop avoidance problem with a new protocol, the Meshed Tree Protocol (MTP). The protocol has essentially two major ideas:

- To prevent broadcast storms, it is sufficient to make sure that the paths used by *broadcast* frames do not have loops. It is *not* necessary to logically delete edges from the underlying network graph, as done in STP using blocked ports[1].
  We note that blocked edges may actually be the best path for unicast traffic, and STP prevents this from happening.
- A failure should not completely invalidate all knowledge gained about the existing network. It is wasteful to abandon *all* knowledge of existing links, just because one link or switch has failed.

II. Current Work: Algorithmic Foundation

The basis of solutions to construct loop-free paths in a switched network, is Dijkstra's algorithm.

Algorithm 1 Dijkstra's algorithm

```
procedure DIJKSTRA
    Start a tree, with one vertex (root) and cost = 0.
    repeat
        Pick each non-tree vertex $v$ one hop away from some vertex in tree, $u$.
        Find the vertex $v$ with smallest cost.
                    ▷ (Cost = cost at $u$ + cost of hop $u - v$).
        Add vertex $v$, and corresponding edge $u - v$, to tree.
    until all vertices are in tree.
end procedure
```

We note that the graph here corresponds to the switched network, vertices correspond to switches, and edges correspond

[1] In STP, the interface that connects a switch to its parent in the spanning tree is a *root port* and an interface connecting a switch to another, to which it is offering to be a parent, is a *designated port*. The other ports – notably, ports connecting a switch to possible but rejected parents – are *non-designated ports*; these are the blocked ports.

to links in the network. (Also, one edge connects exactly two nodes. In other words, our model does not allow shared media taps.)

Spanning Tree Protocol

The Spanning Tree protocol makes an adjustment to Dijkstra's algorithm, by making the process of vertex selection *distributed*. Each vertex, i.e. switch, attempts to extend the tree.

Algorithm 2 Spanning Tree Protocol algorithm

```
procedure STP
    Start a tree, with one initial vertex (root) and cost = 0.
    repeat
        At all tree vertices $u$ in parallel.
        Pick each non-tree vertex $v$ one hop away from $u$.
        Find the vertex $v$ with smallest cost.
                    ▷ (Cost = cost at $u$ + cost of hop $u - v$).
        Add vertex $v$, and corresponding edge $u - v$ ,to tree.
    until all vertices are in tree.
end procedure
```

In other words, while Dijkstra's algorithm works to add the best vertex at each step, the Spanning Tree Algorithm adds vertices using only local decisions (each frontier vertex independently adds children)[2].

The other detail that STP adds to Dijkstra's algorithm, is that when choosing the lowest-cost path, ties are broken using node id. (If there are two possible parents with same cost path to root, the switch with lower ID is preferred as parent.)

TRILL and Shortest Path Bridging

Radia Perlman, the author of STP, in 2011 proposed a successor: TRILL (TRansparent Interconnection of Lots of Links) [5] protocol on RBridges (Router Bridges) [6].

- TRILL improves convergence, by computing spanning trees for the network rooted at each switch.
- Unicast frames follow *separate* optimal forwarding paths, computed between pairs of switches using the routing functionality of the IS-IS protocol.
- Inconsistencies and loops, caused by topology changes, are caught using hop counts.

TRILL operates as a 'shim' between Layers 2 and 3. A parallel effort, initiated under IEEE 802.1Q, introduces IS-IS link state routing into Layer 2 itself, as a component of the Shortest Path Bridging (SPB) protocol [7] .

The underlying path-finding algorithm, again, is Dijkstra's algorithm (used in IS-IS routing). TRILL makes a major improvement to STP, in that it allows for the pre-computation of backup trees. TRILL introduces a very powerful new idea: rather than waiting for a failure and starting from scratch, alternate paths to the root are constructed in advance. This allows for much faster recovery. However, we make two observations regarding TRILL.

- TRILL has not achieved good market penetration. One reason is simply licensing [8]; solutions are expensive. Another issue is that, being a cross-layer solution, it is heavy and slow. (It uses the Layer 3 protocol IS-IS in Layer 2 operations; this requires encapsulation of its messages [9].)
- TRILL is somewhat inflexible. It brings in a heavy cost in computing spanning trees rooted at every node, in order to provide unicast paths. Also, it takes an all-or-none approach to backup spanning trees: it either constructs a complete spanning tree (with the costs of keeping track of no looping, complete coverage etc.), or nothing at all.

III. The Meshed Tree Algorithm

The Meshed Tree Algorithm moves the focus from *spanning trees* to *paths* in the network.

Algorithm 3 The Meshed Tree Algorithm

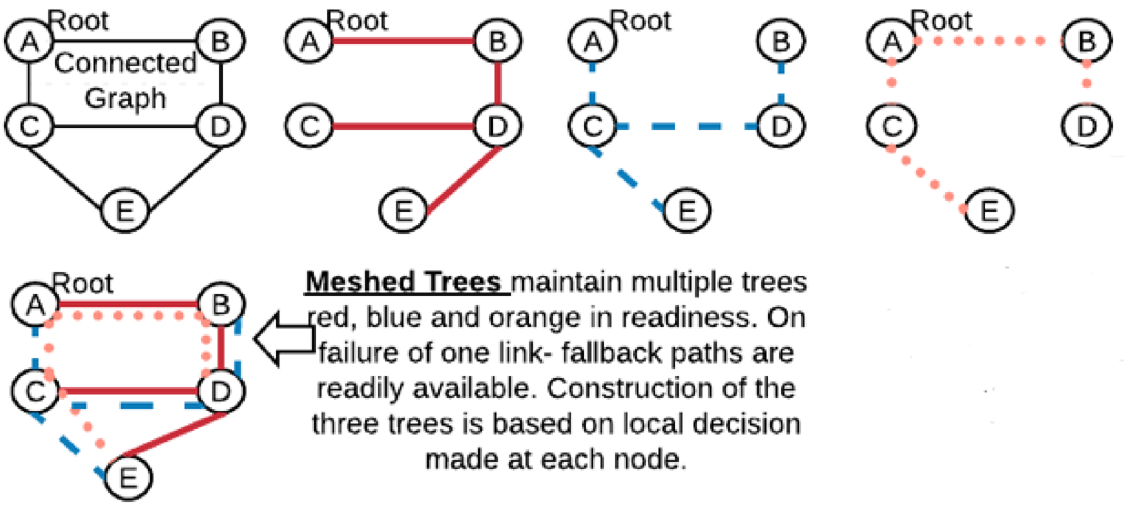
```
procedure MTA
    Start an empty path, at one vertex (root) with cost = 0.
    repeat
        At all vertices $u$ in parallel.
        Pick each vertex $v$ one hop away from $u$.
        Find the vertex $v$ with smallest cost.
                    ▷ (Cost = cost at $u$ + cost of hop $u - v$).
        Offer $v$ all known paths from $u$, appending $v$ to each
    path and adding the cost of the hop $u - v$.
        Accept all new paths received at $u$ where $u$ does not
    itself appear on the path. (Loop prevention)
        Keep the best path known at $u$ as its "chosen" path,
    and send a message ("I am your child") to the vertex which
    advertised it.
    until no vertex $u$ receives any new paths
end procedure
```

To clarify: the "best" path, as we mention above, is the best according to the metric of goodness used, such as hop count.

The structure created by these paths is called a *meshed tree*, as unlike a spanning tree it allows *multiple* paths from the root to any given vertex. A meshed tree has the following properties.

1) Each vertex has multiple possible paths to the root. (In fact, for the naive version of the algorithm as posted above, it finally obtains *all* paths to the root.)
2) However, at a given point in time, only one path is *chosen*. Note that eventually *every* vertex has a chosen path.
3) All paths are loop-free, so there is no danger of broadcast storms. (In particular, this property holds for the chosen path at a vertex.)

[2] To see the difference, consider the graph (itself a tree):
A – B – C, A – D.
A – D costs 100 and the other two links cost 1.
Dijkstra's algorithm is centralized, and is able to add the links of the spanning tree in increasing order of weight:
A – B, B – C, A – D.
STP is distributed, and could easily add them in a different order. If A adds a second child before B gets started, links are added in the order:
A – B, A – D, B – C.



Meshed Trees maintain multiple trees red, blue and orange in readiness. On failure of one link- fallback paths are readily available. Construction of the three trees is based on local decision made at each node.

Fig. 1: Meshed Trees can contain multiple Spanning Trees. []

As Fig 1 illustrates, it is simple to see that a meshed tree is not a spanning tree (except when the underlying graph is a tree!) However, we have an important conjecture.

Conjecture 1. The union of *chosen paths* at the vertices constitute a spanning tree of the network.

This property holds under all cases in our tests, and we aim to develop a formal proof in our future work. We now present a few other important properties of the Meshed Tree Protocol, based on this algorithm, and discuss its practical implementation.

IV. Properties of the Meshed Tree Construction

The Meshed Tree Algorithm, presented in the previous section, has the important properties of *convergence* and *completeness*. More formally:

Theorem 1. After a finite number of steps, every vertex in the graph has a stable chosen path.

Proof:

The graph is connected and finite (being the topology of a switched network). In other words, the number of hops from the root to any vertex u is finite.

Therefore, in a finite number of steps, at least one path advertisement eventually arrives at u. As a vertex accepts all path advertisements, u now has at least one path.

When a vertex has one or more paths, by definition, one of these is its chosen path. Hence, after a finite number of steps, u (and by generality, every vertex in the network) has *at least one* chosen path.

Next, we consider the fact that a vertex can update its chosen path (for example, when it receives a better path).

The graph is finite, hence there are a finite number of paths from a vertex (again, say u) to the root.

Further, every such path is of finite length. (This is because a path is a string with a finite alphabet – the names of the vertices – and is loop-free, i.e. no repetition.) In other words, if we consider any path P, it takes only a finite number of steps for P to be propagated down along all the vertices in the path, and reach u.

Hence, in a finite time, every possible path from u to the root is seen at u. The best path (i.e. the shortest) is its chosen path, and as no better path will be advertised, it has reached a stable state. □

We have therefore proven that executing the Meshed Tree Algorithm on a finite connected graph is guaranteed to converge to a working state. (In practical terms, every switch in the network has a stable path to the root.)

However, two concerns remain.

1) As demonstrated above, every vertex eventually settles on a chosen path. However, *we do not have a guarantee that these paths will actually be the paths used when a vertex forwards messages to the root.*

   We explain with an example. Consider if a vertex u is parent to a vertex v, i.e. v accepted a path P from u and made it its chosen path.

   If P is the best path available to v, clearly its corresponding subpath P' in u must not be worse than any other path available to u. (P' is P minus the last vertex, which is v.) Otherwise, the better path when extended to v, would be superior to P, and would be the chosen path at v – a contradiction.

   However, there remains the chance of a tie. Both u and v may have multiple best paths, and chosen path is chosen independently by each vertex. So in the case there were two equally good paths P' and Q' at u, which were then advertised to v as P and Q, it is possible u chose Q' as its chosen path with v chose P.

   This is not a problem with reaching the root, as u will still forward the frame, only along a different path than v expects. However, the property of *consistency* – that nodes are in agreement about the ranking of paths, so the above scenario does not happen – is necessary if we are to prove Conjecture 1. Strengthening the algorithm, so we can guarantee consistency, is our immediate priority for future work.

2) While the algorithm converges to a working state in

finite time, we have not yet spoken of its space or time complexity. In fact, the naive version we consider is not practical, as the number of possible paths in a graph is very large and it is not feasible to store so many paths at a vertex. (Counting the paths between two vertices in a graph is a $\#P$-complete problem [10].)

For a practical implementation, we must impose a limit on the number of paths stored at a vertex, i.e. switch. We discuss the implementation of the Meshed Tree Protocol, a practical use of the Meshed Tree Algorithm, in the following section.

V. The Meshed Tree Protocol.

The Meshed Tree Protocol makes one change and one implementation decision, with respect to the Meshed Tree Algorithm.

- The change is that each vertex (i.e. network switch) stores only a finite number of the best known paths to the root. In our tests, we store three paths at each vertex – sufficient to have one chosen path and two back-ups.

  As we now have a choice, this raises the question of which paths are the best to store. At the moment we simply store the best (shortest) 3 paths, but in future this could be changed so a vertex stores paths that have as little overlap as possible. (This ensures the least chance that one upstream vertex or edge will invalidate *all* the stored paths if it fails.)
- The implementation decision, is to build and propagate paths using local coordinates, i.e. the outgoing interface numbers at switches.

  A path is implemented as what we name a VID. The VID is a string of numbers, beginning with the Root switch identifier (which we usually set to 1) followed by, in order, the number of the outgoing interface on each switch in the path. In other words, each switch a VID was forwarded by (on its journey to the present position) appended the outgoing interface number to the string; this stores the entire sequence of local forwarding decisions, from the root up to the present vertex, in the VID – and hence the path it followed.
  - There is a one-to-one and onto mapping between VIDs and paths in the network. (In case two switches are connected by multiple edges, there could be multiple VIDs that refer to the same path; this is not allowed in our network model.)
  - Note that we make use of the outgoing rather than the incoming interface number. This is necessary to ensure that VIDs are always unique.
  - VIDs offer *in-built loop detection*. If a switch S ever receives a VID V that is a superstring of a VID U which it already has, this immediately indicates that V has travelled in a loop. (It first reached S following the path U, then looped around and came back to S following the remainder of the path V.)

    Avoiding loops in the Meshed Tree Protocol is as simple as discarding all such VIDs.

Using Fig. 2 we demonstrate the construction of meshed trees in a switched network. Fig. 2A shows the physical

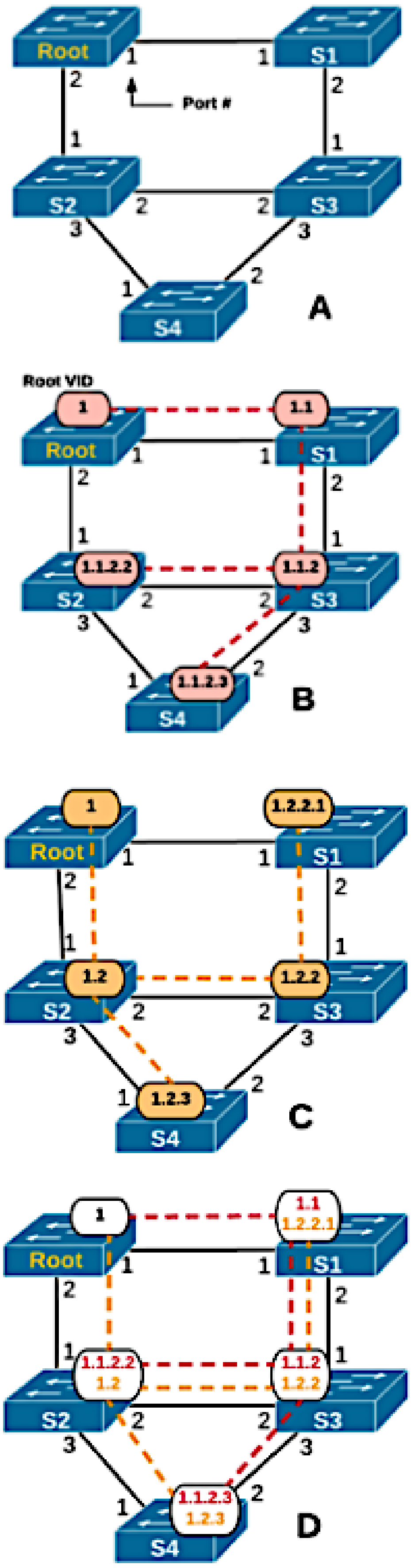


Fig. 2: The Meshed Tree Protocol.

meshed network with 5 switches. Switch port numbers are noted beside links connecting the switches. The root switch is assigned a Virtual Identifier (VID) =1 (in Fig. 2B).

Multiple non-looping paths (or branches) are propagated through the graph as follows, using VIDs.

Fig. 2B shows one part of the meshed tree: a logical tree, starting at the Root, and traversing the network via switch S1. (VIDs defining this branch are shown in pink boxes.)

1) The Root has a VID 1.
2) What is the VID of S1? We start with the VID of its parent, the Root (i.e. ‘1’). To this, we append the number of the outgoing port on the parent where S1 is connected, (also ‘1’). Thus, S1 has a VID of 11.
3) S3 has a VID 112, as it is connected on port 2 of S1.
4) Proceeding as above, we can construct an entire spanning tree defined by VIDs 1, 11, 112, 1122, 1123.

Similarly, in Fig 2C, we show another part of the meshed tree, from the Root via switch S2, defined by VIDs 1, 12, 122, 123, 1221.

Fig. 2D shows both branches, (pink and orange) co-existing. In this example, each switch has two paths to the Root. (In this particular case, each switch is part of two spanning trees.) The VIDs stored at the switches differentiate and identify the multiple tree paths.

- Meshed Tree Protocol (MTP) allows a tree link to be discovered with only a pair of messages between adjacent switches. An upstream switch offers a VID on its port, appending the outgoing port number to its own VID. The downstream switch, if it decides to join the tree branch, accepts. Thus the process of tree construction is simple, and requires very little computation.
- We further note that MTP allows for considerable flexibility in its design. For instance, we can vary how a switch decides which paths to join, (e.g. using hop count or link costs, or disjoint paths as discussed above); the number of VIDs a switch stores; which VID the switch chooses for forwarding, and so on.

VI. Concluding Remarks

In this paper, we have laid out an introduction to a new loop-avoidance protocol for switched networks, called the Meshed Tree Protocol (MTP). Our aim is to use this protocol, both as a fast provider of loop-free broadcast trees (to replace Spanning Tree Protocol) in customer, service provider, and backbone provider LAN/WAN, and as a fabric protocol in Data Center Networks.

The Meshed Tree Procotol is flexible and low-overhead; it maintains multiple paths in the network, and allows them to self-organize into trees as necessary. However, before we can recommend it as a standard replacement for Spanning Tree Protocol, some work remains to be done.

1) Our most important challenge is to prove that, both after initial construction and after re-construction, the paths defined by MTP actually produce a serviceable broadcast tree. (In other words, we must add constraints to the protocol to guarantee *consistency*, as described in Section IV.)
2) The next course of action is to benchmark an implementation of MTP versus the standard Rapid Spanning Tree Protocol, and if possible also TRILL, to determine performance (uptime, convergence time upon failure) and cost (computation as well as communication). We are currently developing a testbed on GENI for this study.
3) Finally, we intend to develop and demonstrate MTP in a Data Center Network, where the multiple paths of different spanning trees can be put to good use at the same time.

References

[1] I. . W. Group *et al.*, “Local and metropolitan area networks-virtual bridged local area networks,” *IEEE Std 802.1 Q-1998*, 1999.

[2] “Ieee standard for local and metropolitan area networks–bridges and bridged networks,” *IEEE Std 802.1Q-2014 (Revision of IEEE Std 802.1Q-2011)*, pp. 1–1832, Dec 2014.

[3] “Media access control (mac) bridges,” http://profesores.elo.utfsm.cl/ agv/elo309/doc/802.1D-1998.pdfMedia Access Control (MAC) Bridges, 1998.

[4] W. Wodjek, “Rapid spanning tree protocol: A new solution from old technology,” http://picmg.opensystemsmedia.com/PDFs/PerfTech.Mar03.pdf, 2014.

[5] R. P. J. Touch, “Transparent interconnection of lots of links (trill): Problem and applicability statement,” http://www.ietf.org/internet-drafts/draft-ietf-trill-prob-05.txt, September 2008.

[6] R. Perlman, “Rbridges: transparent routing,” in *IEEE INFOCOM 2004*, vol. 2. IEEE, 2004, pp. 1211–1218.

[7] P. Ashwood-Smith, “Shortest path bridging ieee 802.1 aq overview,” *Huawei. Retrieved*, vol. 11, 2012.

[8] T. Hollingsworth, 2013. [Online]. Available: https://www.networkcomputing.com/networking/trills-hidden-cost

[9] P. Ashwood-Smith, “Is-is extensions supporting ieee 802.1aq shortest path bridging.”

[10] B. Roberts and D. Kroese, “Estimating the number of s-t paths in a graph,” *J. Graph Algorithms Appl.*, vol. 11, pp. 195–214, 01 2007.

[11] P. M. Fernandez, “Circuit switching in the internet,” Ph.D. dissertation, Citeseer, 2003.

[12] F. Kobuszewski, “Network world 2018,” https://www.networkworld.com/category/network-switch/ , year = 2018.

[13] R. Mailheau, “Trends in network switch technology,” https://planetechusa.com/blog/predictions-2017-network-switch-trends/ , 2017.

[14] M. Seaman, “A multiple vlan registration protocol,” www.ieee802.org/1/files/public/docs2004/MVRP-Introduction-030.pdf, 2004.

[15] “Rapid reconfiguration of spanning tree,” http://www.ieee802.org/1/pages/802.1w.html, 1998.

[16] “Ieee standards for local and metropolitan area networks - amendment to 802.1q virtual bridged local area networks: Multiple spanning trees,” *IEEE Std 802.1s-2002 (Amendment to IEEE Std 802.1Q, 1998 Edition)*, pp. 1–211, Dec 2002.

[17] G. D. D. R. Perlman, D. Eastlake and A. G. Gai, “Rbridges: Base protocol specification,” https://tools.ietf.org/html/rfc6325, 2011.

[18] D. G. D. S. G. A. G. Radia Perlman, Donald Eastlake, “Rbridges: Base protocol specification (rfc 5556),” http://www.ietf.org/internet-drafts/draft-ietf-trill-rbridge-protocol-11.txt, January 2009.

[19] D. Eastlake, “Rbridges: Trill header options,” http://www.ietf.org/internet-drafts/draft-eastlake-trill-rbridge-options-01.txt.

[20] “Ieee standard for local and metropolitan area networks–media access control (mac) bridges and virtual bridged local area networks–amendment 20: Shortest path bridging,” *IEEE Std 802.1aq-2012 (Amendment to IEEE Std 802.1Q-2011)*, pp. 1–340, June 2012.

[21] D. Allan and N. Bragg, *802.1 aq shortest path bridging design and evolution: The architect’s perspective*. John Wiley & Sons, 2012.