Optimum Rooted Trees for Failover in Switched Networks

Peter Willis[1], Nirmala Shenoy[2]
Dept. of Information Sciences and Technologies,
Rochester Institute of Technology, Rochester, New York
[1]pjw7904@rit.edu, [2]nxsvks@rit.edu

***Abstract*— Loop-avoidance is essential in switched networks to avoid indefinite looping of broadcast traffic. Traditionally, spanning trees or Dijkstra trees are constructed to provide logical loop free forwarding paths on the physically meshed network to overcome this problem. Spanning tree based protocols construct a tree from a single root and incur high convergence latency on root switch failure. Dijkstra tree based protocols construct a tree from every switches and incur high operational overhead. Given the probability of simultaneous multiple switch failures is very low, we propose an approach that allows an administrator to predesignate an optimum number of roots. The advantage of this approach - the high convergence latency experienced with Spanning tree protocols and the high operational overhead and complexity of Dijkstra tree based protocols both are avoided. The new approach based on Meshed Tree Bridging provides a novel loop-avoidance system that reconverges networks quickly with very low downtime in the event of root switch failure. This is demonstrated on the Global Environment for Network Innovations (GENI) testbed against Rapid Spanning Tree Protocol – an industry accepted standard for loop-avoidance.**

Keywords— Ethernet Switching, Meshed Trees, Optimal Roots

I. INTRODUCTION

Networks implementing IEEE 802.3 Ethernet [1] and IEEE 802.1D [2] bridging require loop-avoidance protocols to forward broadcast traffic without duplication and indefinite forwarding of frames. To provide redundancy in the event of interface or link failures, meshed networks with physical loops are used in switched networks. This imposes the need for loop-avoidance protocols to stop broadcast traffic from being flooded indefinitely and causing a network crash.

To stop *frames from looping* in the network, loop-avoidance protocols have historically utilized spanning trees to create a logical broadcast forwarding path. The loop-free nature of a tree graph in which all nodes are visited once led to the development of the original Spanning Tree Protocol (STP) [2]. Paths which are not part of the tree are disabled by STP. Later Rapid Spanning Tree Protocol (RSTP)**Error! Reference source not found.** enhanced STP for faster convergence on topology changes and became the industry standard. Networks running RSTP elect a root and then construct the spanning tree from this root. Root election incurs high delays and result in high network downtime on root switch failure in RSTP. Hence protocols implementing link state routing was introduced (for loop avoidance in switched networks), where every switch is a root and a Dijkstra shortest path tree is constructed from every switch. Link state routing builds a database of all nodes and their connecting links to construct Disjktra trees to every switch. IEEE 802.1aq Shortest Path Bridging (SPB)[3] and the Internet Engineering Task Force's (IETF) TRILL (Transparent Interconnection of Lots of Links) on Routing Bridges [5] both adopt the Intermediate System to Intermediate System (IS-IS) link state routing protocol to implement this functionality. Building, maintaining, and pruning link state databases are computationally expensive and the protocols are much more complex than RSTP, and requires all IS-IS message to be encapsulated in SPB or TRILL headers.

In the event of the root switch failure, RSTP requires a new root to be elected and then a new spanning tree constructed from this root. This incurs very high delays. With IS-IS based protocols, on a switch failure, the associated link state changes has to be disseminated to all switches. Switches then wait for a database settling time before recomputing a Dijkstra tree to every other switch – and this happens at all switches. During this time, frame forwarding is not reliable. The overhead and computational complexity is very high – given that Dijkstra tree construction is by itself computation intensive[8].

Meshed Tree Bridging (MTB), a proposed standard for use in IEEE-compliant networks, provides broadcast frame forwarding paths without the need to disable or modify port roles in switches. MTB does not require network wide information dissemination. It uses a novel meshed tree algorithm (MTA), to pre-construct a number of paths between a root and a non-root switch. On link or interface failures the next path is ready to take over. Convergence latency is very low[7]. To address root switch failure, Meshed Trees was extended to Multi Meshed Trees (MMT) that allows construction of meshed trees from a number of roots. Too many roots would result in high operational overhead (example IS-IS based solutions). One root does not offer adequate redundancy (example RSTP). So, with MMT, we offer the option of deciding the number of roots based on the failure probability of switches and the uptime desired in the network – i.e. the optimal number of roots. MTB provisions redundancy in paths and roots with very low operational overhead – attributed to a simple yet novel numbering scheme via Virtual IDs (VID).

II. MESHED TREE BRIDGING

We first explain a one-root meshed tree construction with Meshed Tree Bridging Protocol (MTBP). The MTBP implementation starts with a root node selected manually to provide administrators the freedom of matching the design of the network and the root switch capacity; this also avoids root election delays. The root node is given a single digit Virtual Identifier (VID). A VID, structured as a collection of decimal-delimited integers, describes the path from the root to a given

Annex to ISBN 978-3-903176-28-7

interface on another switch. The numbers are a collection of the egress interface values at each switch in the path thus far. VIDs are disseminated through the advertisement issued by MTBP running in the switches. For example, in Fig. 1, S1 is given a VID of 1.1 from the root, which appends the egress interface number of 1 to its base VID '1'. Every VID received is then forwarded to all other neighbors with an appended egress port. These VIDs not only provide a record of each path taken to get to a given interface on a switch, but also provide a cost (hop count) for the path. The lowest-cost VID is declared the primary VID (PVID). The PVIDs and their ports of acquisition form the broadcast tree defining the non-looping paths. Through a substring check of VIDs received against what is already stored, loops can be detected to avoid such VIDs [7]

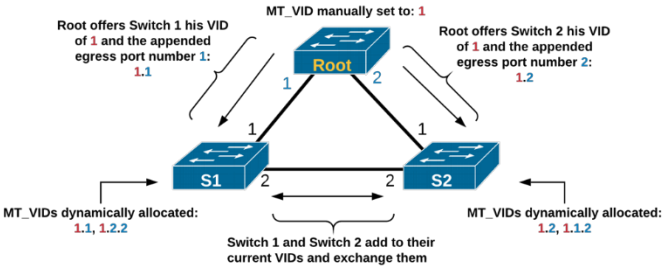

Fig. 1. Meshed Tree Construction with Virtual IDs

Multi Meshed Trees: In MTBP explained above what if the root switch fails? To avoid root election delays, we predesignate a secondary, and if needed a tertiary and a fourth root. And we pre-construct meshed trees from each of the predesignated roots. To cut down on the construction and maintenance of the multiple meshed trees but at the same time to provide adequate root redundancy it is necessary to decide on an optimal number of roots as explained earlier. MMT will construct meshed trees from the multiple pre-selected roots. In this demo we present a two-root MMT implementation.

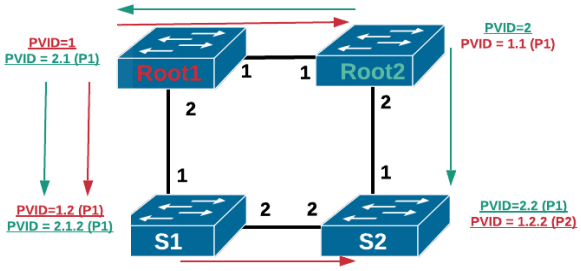

Fig. 2. Multi Meshed Trees with 2 Roots

Similar to the meshed tree construction from one root (Fig. 1), in Fig. 2 we show a 4-switch network, where another meshed tree is constructed from another Root2 (green tree), assigned a VID 2. (For simplicity we do not show meshing within each tree). The roots are assigned a preference. Under normal operations the meshed tree from Root1(red tree) is used; on its failure the meshed tree from Root2 takes over. MTBP implementation details using a single root is available in [7]. MTBP with 2 roots is implemented along the same lines. Included in this implementation is Root2's detection of Root1 failure and how the meshed tree from Root2 takes over broadcast frame forwarding. For the demo, we select a neighbor of Root1, as Root2. This will speed up the failover to the secondary meshed tree.

III. PROTOTYPE DEMONSTRATION

To understand impact of a root switch failure on networks running MTP and RSTP, an implementation will be executed on topologies designed on the GENI testbed [7]. Resulting data collected from the two implementations will be presented and compared. ***By looking at how the protocols respond to the root switch failure, one can observe the resiliency to root switch failures and the impact on broadcast traffic during the recovery.*** Using the GENI testbed, custom machines reserved from universities across the United States will be used for the demo. Three topologies, with 5, 10, and 17 switches (see Fig 3), were designed to study the protocol performance as the network size (diameter) and the connectivity increases. Clients connected to the switches (not shown in Figure) will generate broadcast traffic. We will present

- Convergence time on Root1 failure
- Control traffic generated for convergence to assess and explain complexity of the protocol recovery process
- Broadcast traffic lost, duplicated and delivered out of sequence during convergence

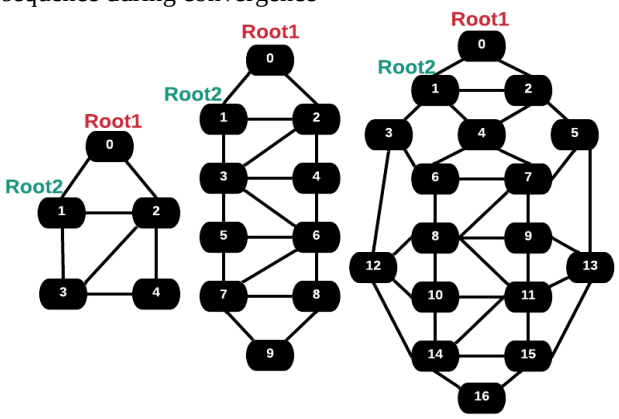

Fig. 3. Topologies for Demo (Clients are not shown)

REFERENCES

[1] "IEEE Standard for Ethernet," IEEE Std 802.3-2018 (Revision of IEEE Std 802.3-2015), 2018.

[2] "IEEE Standard for Local and metropolitan area networks: Media Access Control (MAC) Bridges," IEEE Std 802.1D-2004 (Revision of IEEE Std 802.1D-1998), 2004.

[3] IEEE 802.1w - Rapid Reconfiguration of Spanning Tree, supplement to ISO/IEC 15802-3:1998 (IEEE Std 802.1D-1998)

[4] 802.1aq-2012 - IEEE Standard for Local and metropolitan area networks--Media Access Control (MAC) Bridges and Virtual Bridged Local Area Networks--Amendment 20: Shortest Path Bridging , Retrieved 4th Nov 2014

[5] R. Perlman, D. Eastlake, D. Dutt, S. Gai, and A. Ghanwani, "Routing bridges (rbridges): Base protocol specification," RFC 6325, RFC Editor, July 2011.

[6] www.geni.net

[7] P. Willis and N. Shenoy, "A meshed tree protocol for loop avoidance in switched networks," in 2019 International Conference on Computing, Networking and Communications (ICNC), pp. 303–307, Feb 2019.

[8] http://mathworld.wolfram.com/DijkstrasAlgorithm.html, Retrieved 4th Nov 2014