stp-tree-generator

Alexander Schloegl, Matr Nr. 1315020

August 29, 2016

Chapter 1

Introduction

Why would you use STP? Why did we make the tool? How does it work? (Why passive?)

Chapter 2

Background

What other tools are out there? How does STP work? Pcap JSON very brief

Chapter 3

STP Tree Generator

What it do?

Chapter 4

Software Switch Testing Utility

Why did we use this? Technologies used? Capabilities? Limitations?

Chapter 5

Testing

5.1 Setup

The original paper discussing STP [1] is openly available. However, it does not dictate exact implementation details. Therefore, in order to reach an accurate enough understanding of it multiple tests were performed. To minimize external influences these tests needed to be done in a network disconnected from any external network. Additionally, to be able to check the results for correctness, we kept the network small and simple. The layout of the test setup can be seen in Figure 5.1. While a very basic network, it is sufficient for the tests described below, which will guarantee the required capabilities of the developed tool. The root in figure 5.1 was made root by manually assigning it a higher priority than the other bridges (note that *higher* in this case means *smaller*). Bridges A, B and C had their priorities left to the default value. Nodes A, B and C were running *Wireshark* and the developed tool. Additionally using *Wireshark* allowed us to check the actual packages involved in the testing process to monitor progress and note situations not handled correctly by the tool. The server for our tool was run on Node A. Test results were checked for their correctness by checking the report provided by the tool.

5.2 Tests

5.2.1 Usage Test

The Usage test proves the general capability of the tool. It is designed to simulate the connection of nodes to an established network. Using a simple setup like shown in Figure 5.1 allows us to collect information about the whole network. If the spanning tree were more than 2 layers deep and we were using less nodes than bridges in the network some information about the topology would be unobtainable without additional information.

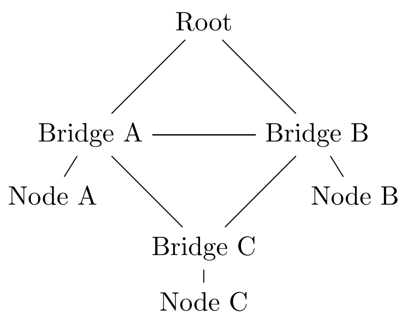


Figure 5.1: Test setup

Performing the test

1. The layout shown in Figure 5.1 was established and all devices were started.

2. No tool was started yet, identification during the tree establishment is covered by the Tree Establishment Test (5.2.2)

3. We waited for the bridges to establish a stable spanning tree, checking the progress by observing the STP packets for the TC flag.

4. After the tree had stabilized we started the server on node A and the tools on all the nodes.

5. We waited for the tools to send their data to the server, which takes one $helloTime$ plus the latency between the nodes and the server. Default value for the $helloTime$ is 2 seconds.

6. When all nodes had sent their data to the server we created a report to check it against the expected result.

Expected Result

With the limited size of the test network all bridges can and must be identified correctly for this test to be counted as successful.

5.2.2 Tree Establishment Test

To test whether the tool can handle bridges being added to the network during runtime, we start the tool on all nodes before the establishment of the spanning tree, and check for correct identification afterwards.

Performing the test

1. The layout shown in Figure 5.1 was established and all devices were started.
2. Before enabling STP on all the bridges we started the server and tools.
3. After enabling STP on the bridges we waited for the tree to be established, again checking the progress via the TC flag.
4. Lastly we checked the result for correctness.

Expected Result

Again all the bridges can and must be correctly identified.

5.2.3 Bridge Removal Test

The tool must also be able to handle bridges dropping from a network. This test makes sure that capability is given.

Performing the test

1. We started all the nodes and tools and waited for the spanning tree to be established and correctly identified (see the sections on the Usage Test 5.2.1 and the Tree Establishment Test 5.2.2).
2. After the tree was constructed we unplugged Node B and waited for the tree to stabilize.
3. Finally we checked the output of the report for correct identification of the smaller tree.

Expected Result

The new path through which Node C is connected to the root must be correctly identified.

5.2.4 Slow Dynamic Change Test

Changes in the network topology must not be a problem for the tool. By succesfully testing for additions and removals from the network (sections 5.2.1 and 5.2.3) one could assume that changes can be handled as well, but caution demands that we test specifically for changes in the topology. Node C is connected to the root either via node A or node B. The node with the smaller MAC address (as they have the same priority) will be the preferred hop on the path to the root. Therefore the logical topology will look like either one of the figures in Figure 5.2. In our case bridge C was logically connected to bridge A as that was the possible connection to the root with the smaller MAC address.

Performing the test

1. We set up all the nodes and waited for a stable tree to be generated by the bridges like before (sections 5.2.1, 5.2.2 and 5.2.3).
2. We disconnected bridge A from the root, leaving us with a 2 bridge subtree.
3. The connection between bridges A and B was also severed, as well as the connection between bridge C and B.
4. We waited for the spanning tree to stabilize after the removal.
5. Bridges A and C were connected to bridge B yielding the physical topology shown in Figure 5.3.
6. After the tree had stabilized we checked the report output for correctness.

Expected Result

The tool must correctly identify the new topology.

5.2.5 Fast Dynamic Change Test

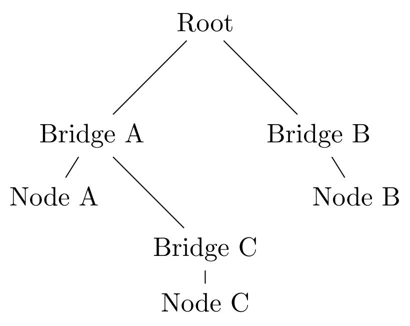
Robustness to changes in the topology are a must for the developed tool. It would however also be nice if the tool were robust enough to handle topology changes without a spanning tree stabilization in between. The logical topology is the same as for the Slow Dynamic Change Test 5.2.4

Performing the test

1. We set up all the nodes and waited for a stable spanning tree.
2. After we removed node A from the network we immediately plugged nodes A and C into B.
3. We waited $2 * helloTime$ to make sure the changes were propagated to the server before plugging node C into the root.
4. Finally we waited for the tree to stabilize and checked the output for correctness.

Expected Result

Whether the output was correct or not gave us information about the capabilities of the tool. Due to its distributed nature it is also difficult to debug and this gave us additional insights into the execution behaviour.



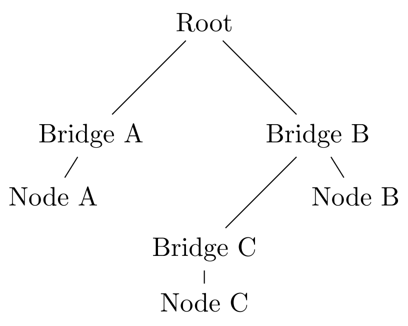


Figure 5.2: The 2 possible paths to the root from node *C*

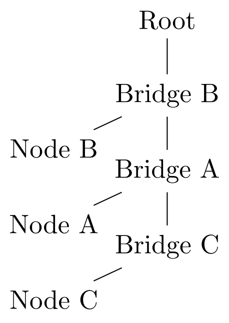


Figure 5.3: The physical network after performing the steps described in the Slow Dynamic Change Test 5.2.4

Chapter 6

Conclusion

What did we learn? What can be expanded? - stp-t-g - s-s

Bibliography

[1] Radia Perlman. An algorithm for distributed computation of a spanningtree in an extended lan. *SIGCOMM Comput. Commun. Rev.*, 15(4):44–53, September 1985.