

## Lab 6 – Spanning Tree Protocol

RES 841

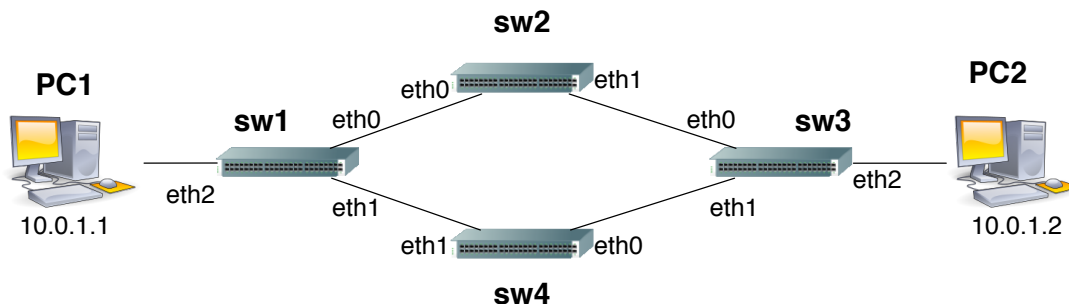
October 2013

### Introduction

In this lab, you will discover the *Spanning Tree Protocol*, which is specified in the IEEE 802.1d standard. This protocol is used between switches, that will sometimes be called *bridges* in this lab.

### 1 Uncontrolled broadcast

You will start to launch the virtualized environment using VMWare player. Refer to the first lab for instructions and **change directory to TP\_6a** before typing `lstart`. This will create the network represented on the figure below:



You can capture traffic on various network elements using the command:

```
tcpdump -i any -s 0 -w /hostlab/capture-XX.pcap &
```

where XX can be replaced by whatever identifier you like (for instance, you may use different identifiers on different machines to distinguish who captures what). The "&" character at the end of the command allows to execute the capture in background, leaving you the control of the command line. To get the control back on the backgrounded process (for example to stop it with Control-C), type the `fg` command in the terminal.

The topology represented above has some redundancy. In terms of graphs it has a cycle. We will first examine what happens when a broadcasted frame is sent on such a topology.

1. Capture the traffic using `tcpdump`, as explained above, on a switch.
2. You will send a **single broadcast message** from PC1 using the following command: commande suivante :

```
ping -b -c 1 10.0.1.255
```

The `-b` option indicates that ping shall query a broadcast address and the `-c 1` option indicates that a single message should be sent (ping usually sends a sequence of messages).

3. After a few seconds, stop TCPDump with a Control-C. TCPDump, as usual, will display statistics about the number of frames captured. What do you notice?

4. Return to the host machine ; stop the virtual machine with `lcrash`, then clean up temporary files with `lclean` (if you do not perform these tasks, your computer may rapidly become very slow).
5. Examine the capture file with Wireshark. You will notice two types of packets. The first type belongs to the ARP protocol, which will be examined in the next lab. The second type belongs to the ICMP protocol, and corresponds to the traffic generated by your `ping` command. You can use a filter to display only these packets, simply type `icmp` in the `filter` field and press `Apply`.
6. You can use the `IO Graphs` command in the `Statistics` menu to represent the throughput that these packets induce. Do not forget to filter again these packets to obtain the graph that effectively represents the traffic you are looking for. Do not hesitate to tune the axis scales (`tick interval` and `Unit`).
7. How do you interpret and explain the results?

## 2 Spanning Tree Protocol

As you noticed in the previous scenario, broadcasts can have a very important effect on the network. However, being able to build a network with cycles is very important for an administrator, as cycles mean redundancy. In addition to load balancing, when an equipment or a link fails, this redundancy becomes useful, as it allows the network to remain connected. That's why bridges communicate together to build on the top of the redundant network a virtual topology that does not contain any cycle by de-activating some links (indeed, these links remain active but do not accept any data frame). This topology will be used in normal operation, until a failure is detected, which triggers the reactivation of a backup link.

The *Spanning Tree Protocol* is the protocol that is in charge of identifying which links shall remain active and which links should be deactivated. Its purpose is to extract, from the topology graph a single, non-ambiguous, cycle-free sub-graph, i.e a tree. This tree shall cover and connect all the network elements; that's why it is called a *spanning tree*. The protocol operates in a fully distributed manner, i.e. no central server decides of the tree but bridges communicate together to form this tree.

Let us re-launch the previous experiment when the *Spanning Tree Protocol* is active between the different bridges. Make sure that the temporary files are cleaned up (`lclean` and you may remove the capture file too) before starting again the lab (`lstart`).

The bridges implemented here are not real switches. They are (virtual) machines, running Linux and operating under the *bridge* mode. In other words, they are configured to act as bridges between their network interfaces, which allows a relatively easy configuration of the bridges behavior using the `brctl` command, for which you can read the manual page (`man brctl`).

1. To activate the *Spanning Tree Protocol*, type following command on *all* bridges (sw1 to sw4) :

```
brctl stp br0 on
```

This command activates the spanning tree protocol on the bridge interface `br0` (it is possible to define several bridges per machine, for instance to group interfaces in two groups of interfaces).

2. Once spanning tree protocol is active, retry the previous experiment (Sending a broadcast `ping` command). What do you notice?
3. Using `tcpdump`, you can have an indication on which interfaces have been selected. Launch `TCPDump` to display packets instead of storing them into a file, (removing the `-w` option and the following path) and displaying only the `icmp` packets on one or all the interfaces. For example, to restrict the capture to the first interface, `eth0`, type:

```
tcpdump -i eth0 icmp
```

Try, with these commands, to identify and list the active and inactive interfaces in the network. You will have to manipulate the interfaces, each bridge having one, two or three interfaces (`eth0`, `eth1` et `eth2`). Specify any as interface name will allow you to capture the traffic on all interfaces simultaneously.

4. You can check the result by displaying, on the different bridges, with the following command:

```
brctl showstp br0
```

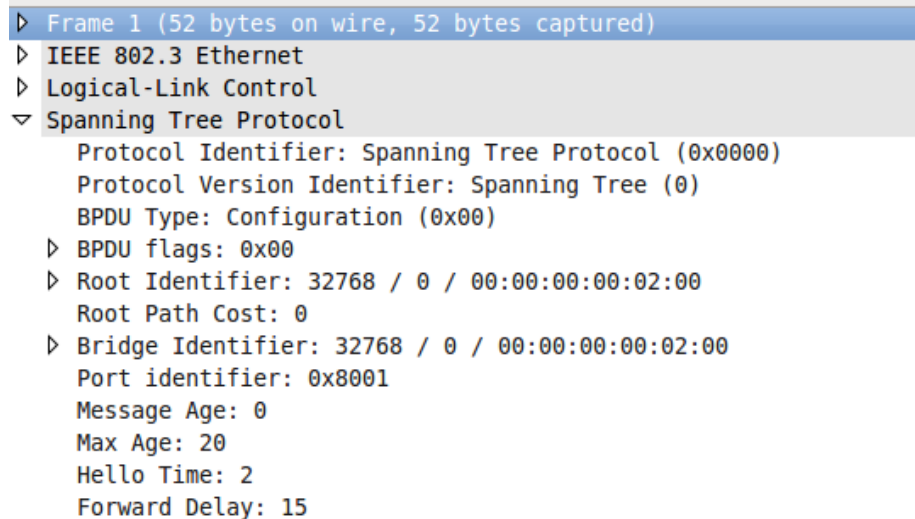
This command displays a few information on each interface (`ethXX`). The information *Root Port*, for each bridge identifies the interface by which the *root* bridge is reachable. Be careful, this interface identifier number is shifted compared to its classical identifier (`eth0` corresponds to interface number 1). From this information, you can draw the tree by looking, on each bridge, its root port.

You will notice that every interface state may be either *forwarding* (meaning active), or *blocking* (meaning inactive). On a single link, the interface at one end may be active while the other may be inactive. This is due to the fact that the bridges take their decision individually. However, the traffic is blocked as soon as one of the interfaces is in blocking state.

### 3 Examining the spanning tree creation

Terminate the machines with the `lcrash` command, clean up the temporary files and change directory to `TP_6b` (`cd ../TP_6b`). Launch the machines again (`lstart`); a new network starts up, using the same topology as in the previous exercise, except that the spanning tree protocol is directly active and that the traffic is automatically captured. The capture starts 1 minute after each bridge startup (to allow each bridge to fully start up) and to last 30 packets. Consequently, you will find, after the lab has started a certain number of files in the host home named `capture-stp-swX.pcap`, where `X` is the bridge number, that you can examine with Wireshark.

By looking at the capture files, you can see the information relative to the *Spanning Tree* in the "Spanning Tree Protocol" section, as showed in the screen capture below:



```

  ▸ Frame 1 (52 bytes on wire, 52 bytes captured)
  ▸ IEEE 802.3 Ethernet
  ▸ Logical-Link Control
  ▾ Spanning Tree Protocol
    Protocol Identifier: Spanning Tree Protocol (0x0000)
    Protocol Version Identifier: Spanning Tree (0)
    BPDU Type: Configuration (0x00)
  ▸ BPDU flags: 0x00
  ▸ Root Identifier: 32768 / 0 / 00:00:00:00:02:00
    Root Path Cost: 0
  ▸ Bridge Identifier: 32768 / 0 / 00:00:00:00:02:00
    Port identifier: 0x8001
    Message Age: 0
    Max Age: 20
    Hello Time: 2
    Forward Delay: 15
  
```

The protocol works in two phases. In the first phase, *Configuration* messages are exchanged to elect the root bridge, i.e. the reference bridge. In the next phase, every topology change will induce a rebuilding of the tree.

#### 3.1 Root identification

In the root election phase, bridges exchange *Configuration* messages in which they indicate to their neighbors which bridge they consider to be the root. Each bridge regularly emits similar information to all its neighbors. When receiving such a message, each bridge compares the announced bridge identifier with the one of the bridge it currently considers to be the root, looking for the smallest identifier.

A bridge identifier, that we can see in the `Root Identifier` field of the `Configuration` packets, is formed by two fields:

**A priority** over 16 bits, that is by default equal to 32768 (80:00 in hexadecimal), and allowing a range between 0 and 65535.

**The MAC address** of the first interface of the bridge. MAC Addresses are, in this lab, defined such that the fifth field is the bridge number and that the last one is the interface identifier. (ex : 00:00:00:00:03:02 identifies interface `sw3` on the bridge `sw3`).

The identifier is therefore based on the MAC address, but the priority field allows a system administrator to tune the protocol in order to influence the tree creation. You can display each bridge identifier (hexadecimal) by typing `brctl show` in a terminal.

If the received announcement contains a lower ID than the current root, the bridge updates its choice, stops sending `Configuration` messages of its own and only retransmits `Configuration` messages that announce this root (or a better one). Once this phase is complete, only messages that are issued by the true root (lowest ID bridge) are retransmitted in the network.

1. Typing `brctl showstp br0` on the bridges, identify the root of the tree (*designated root*). Is this root the identical for all bridges? How is this unicity guaranteed?
2. By examining each capture file, determine how the root information propagates. Examine the root ID that each bridge selects at first. How many messages are required for this process to converge?

You can now observe the effect of a change in a bridge priority:

1. Stop the *Spanning Tree* on each bridge, typing `brctl stp br0 off`, otherwise you may not manage to capture the initialization phase.
2. Change, then, each bridge priority using the command `brctl setbridgeprio br0 XXX`, replacing XXX by a number between 0 and 65535. The lowest value corresponds to the highest priority.
3. Start the capture on each bridge with `tcpdump` and relaunch the spanning tree on each bridge, typing `brctl stp br0 on`
4. Look, using `brctl showstp br0`, changes in the root and active interfaces.

### 3.2 Selection of the interfaces states

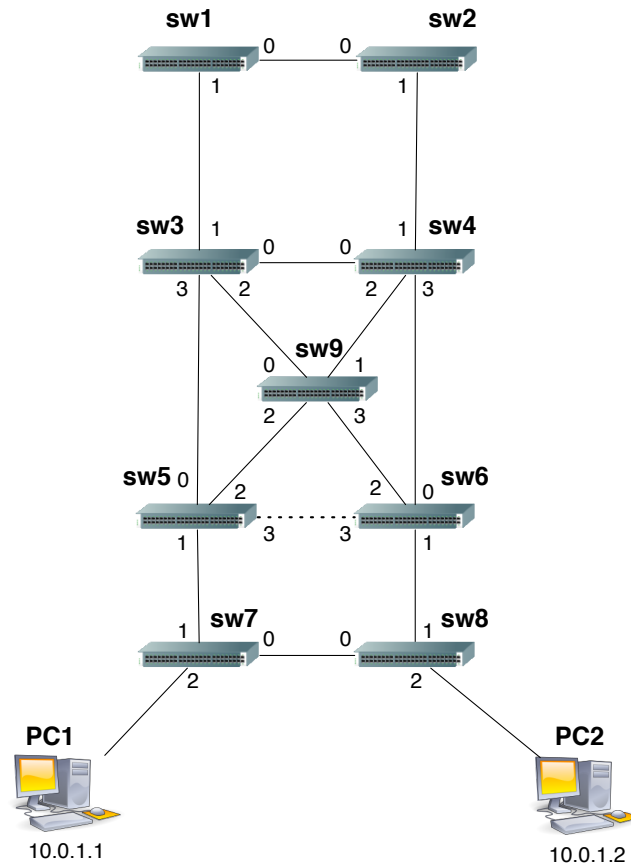
Once the root is identified, the tree still needs to be built by selecting active and blocking interfaces. Based on the `Configuration` message, each bridge will select an interface as its *root port*, i.e. as the interface that brings it closer to the root tree. This interface is the one that offers the best path towards the root using:

- the *cost* of the path. Each bridge defines, for each of its port, a cost, which is a numerical value that you can display using the `brctl showstp br0` command. For each interface, a *path cost* field displays the associated cost. It is possible to tune this value, using the `brctl setpathcost br0 ethX YY` command, where `ethX` is the interface identifier and `YY` is the cost. The protocol will select, as the *root port*, the one that has the minimal cumulative cost towards the root.
- When cost values are equal for two paths, the bridge *identifier* of the parent bridge becomes the discriminating criterion, taking into account the priority.
- If these values are still equal (when two links are connected redundantly to the same bridge for instance), the local interface number can be used as the final discrimination criterion.

1. play with the interface costs to define a tree in which the link between `sw1` and `sw2` is blocked.

## 4 Protocol efficiency

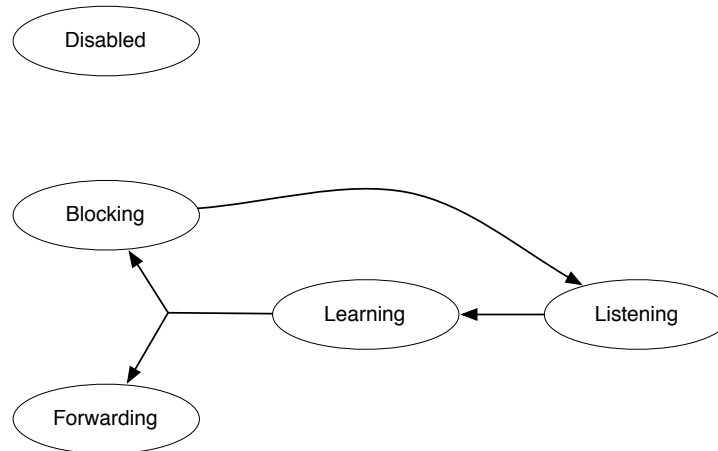
Stop the virtual network, clean up the temporary files, change directory to TP\_6c and restart the new virtual network with `lstart`. The new network is now represented on the figure below. The dotted line represents an inactive link that will not be taken into account by the protocol; numbers represent the interface identifiers.



1. Based on your understanding of the algorithm, what should be the spanning tree on this network (all links costs are identical)?
2. Verify your answer by looking at the root ports on every bridge.
3. Capturing the traffic with `tcpdump` at different network locations, identify the path followed by a `ping` from PC1 (10.0.1.1) to PC2 (10.0.0.2) ?
4. Does this path seem efficient to you? What are, in your opinion, the problems that can appear with the *Spanning Tree*?
5. What would be best place to localize the root of the tree? Influence the protocol, by tuning the bridge priorities so that this root is effectively chosen.
6. Can you reach a good solution without changing the root bridge, but only playing with the interfaces costs?

## 5 Topology changes

The *Spanning Tree* protocol is always running, it monitors network links and sends `Topology Change` messages when it seems necessary to update the tree. Bridges then re-execute the classical procedure. Besides, when a link does not seem to work anymore, interfaces see their state evolve (active, blocking, etc.) according to the state machine below:



Depending on the state and results of the protocol, each interface of a bridge may be in five distinct states. In each of these states a different policy is applied regarding data and spanning tree control frames (these control frames such as the `Topology change` message, are called `BPDU` below).

**forwarding** : the port is fully active ; it receives and emits any type of frame.

**blocking** : The port does not emit or accept any data frame. It has been disabled by the spanning tree protocol. It does not send any `BPDU` but receives incoming `BPDU` in order to make sure that the link is still active, but does not look into these `BPDU` (it does not take into account the packet contents). If no `BPDU` is received during a certain time (around 20 seconds), the bridge considers that an error happened on the interface and passes in *listening* state.

**listening** : The interface only receives `BPDU`, but the content of these frames is taken into account. A new tree is being calculated and this state manages this transient phase. It generally lasts 15 seconds before the interface passes in *learning* state. No data frame is emitted or received in this state.

**learning** : the bridge accepts to receive data frames on the interface but does not emit data frames. Only `BPDU` are emitted, received and considered. The bridge participates to the tree but is still cautious about data frames. It remains in this state during 15 seconds unless the interface is de-activated by the protocol, reaching the *blocking* state.

**disabled** : The interface is explicitly disabled by the administrator.

You will notice, if you capture some of the network traffic, that any change in the topology (see below) generates a certain number of topology update messages. You have all the commands to understand the protocol behavior; you will combine `tcpdump`, `wireshark` et `brctl` to visualize the update process when you deactivate or activate a link. You may slow down the protocol reactions and operation by playing with the following timers:

**Hello Time** : is the interval between two *Hello* packets emissions, which are sent by bridges to notify their neighbors of their existence. By default this interval is equal to 2 seconds. and can be changed with the command `brctl sethello br0 XX` where `XX` is a number of seconds.

**Max Age** : is the timeout that defines how much time an interface should remain in *blocking* state before passing to *listening* when no BPDU is received. It defines the reaction time of the protocol to the errors and can be tuned with `brctl setmaxage br0 XX` where `XX` is a number of seconds.

**Forward Delay** : is the learning time, by default set to 15 seconds. It defines the time passed in the intermediate states (*Listening* and *Learning*) before passing to the next step. It represents the level of compromise between reactivity and caution and is tuned with `brctl setfd br0 XX` where `XX` is a number of seconds.

1. You can activate a link by adding the corresponding interface to the set of links handled by the bridge. For instance, between bridges 5 and 6 on the previous figure, le dotted line is an inactive link that you can activate by typing on `sw5` and `sw6` `ifconfig eth3 up`. You then need to include it to both machines bridges by typing `brctl addif br0 eth3`.
2. To deactivate a link, just deactivate the corresponding interface : `ifconfig ethX down`

Try to activate and deactivate links and observe the consequent traffic. Note the `Topology Update` which are followed by `Configuration` messages. Detail how the protocol reacts to your changes.