Spanning Tree Protocol through Software-Defined Networking

Theoretical Foundation

1. Introduction

The need to have data shared remotely lead to interconnected computers, thus creating computer networks. Local networks are private networks formed by computers interconnected by means of a bus (Ethernet) or connected between them forming a ring (Token Ring).
The predominant technology in local networks is Ethernet, in which it assumes a topology in bus form. In this model, the signal is transmitted on the bus, which is a physical medium such as a copper wire forming a single collision domain, thus making it for all computers. However, only the computer that has the network adapter with the physical address contained in the frame header (link PDU) will extract the datagram and deliver to the network layer. However, this structure becomes saturated with the increase of hosts and a very high network traffic, thus making access to the media highly disputed, generating many collisions.
With the evolution of networks and the need for more flexible and scalable topologies, equipment such as bridges and switches have been developed. These switching devices eliminate the single collision domain by isolating their ports, thereby forming a collision domain for each interface. So, the received frame will be processed by the equipment and forwarded to the port on which the destination host is, according to its switching table which is formed automatically and dynamically. Then, if a destination address is not included in the switching table, the switch passes the frame on all its interfaces, and for each received frame, the sender's physical address is stored in the switch table.
Switches have become very important for network performance, being responsible for the connection of the local network. In medium to large networks it is necessary to use many switches. In these cases it is possible to organize these equipments in different topologies, in order to have redundant paths and avoid failures.
However, forming redundant paths will form loops between switches, which are undesirable because they generate problems such as allocation of all links, and network locking. These problems occur due to the standard operation of the equipment and the network protocols, which send messages in a broadcast generating frame copy forwarding, passing from one switch to another in a loop. In this way, the frames will cycle through the switches, multiplying exponentially (Figure 1).

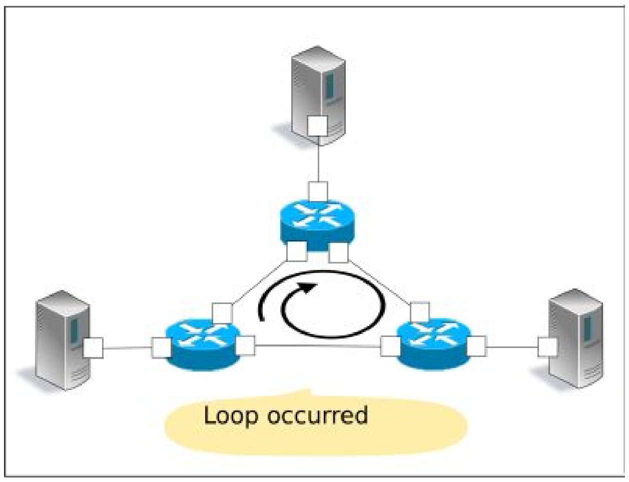


Figure 1

An example of broadcast frame transmission is verified in ARP messages. In a local network, this protocol is intended to identify the MAC address of the network interface of an equipment, that has a specific IP address. In order to a host to communicate with this specific equipment, it's required to discover the MAC address of whose IP address is already known by the host. Thus, the source host transmits an ARP broadcast message, asking what is the MAC address of the host that has the IP address in question.

To avoid the problem of redundant paths between switches, Spanning Tree (STP) protocols are used. STP act by controlling the switches, determining the ports to be disabled and thus avoiding loops (Figure 2). For this, the STP protocol sends BPDUs (Bridge PDUs) between the switches. So, according to Bridge ID, the root bridge will be chosen, determining a topology according to the ID of the others bridges and the cost of the links, so that they do not form any loop between them. Although the use of these protocols solve the problem of frames circulating indefinitely by the switches, to solve the problem they end up reducing the optimization of the physical resources.

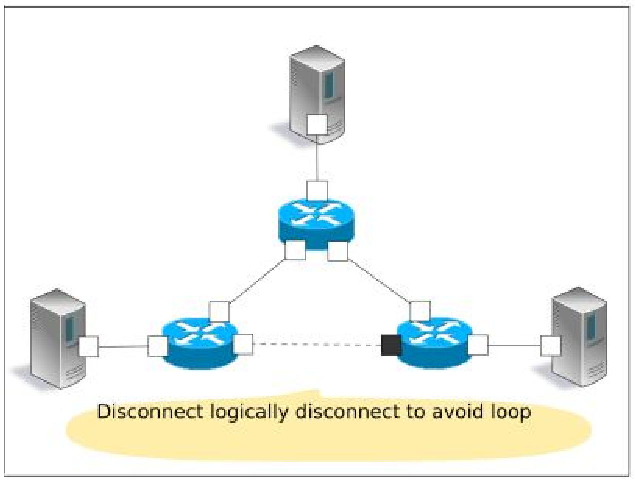


Figure 2 -Switch port's disabled for avoiding loops

An interesting approach to addressing these types of scenarios is Software-Defined Networking (SDN). With SDN, it is possible to create a controller for network equipment and program its entire operation. To do this, it abstracts the hardware layer and centrally controls the SDN control layer. The control layer interacts directly with the application and hardware layers (Figure 3). For this reason, you can create network environments in a customized way, specifically meeting the needs of the network administrator. With this possibility to program the operation of the network in the control layer, it is possible to create routing rules so that the data flow assumes a certain path, in order to carry out traffic balancing and avoid the problem of loops in redundant paths

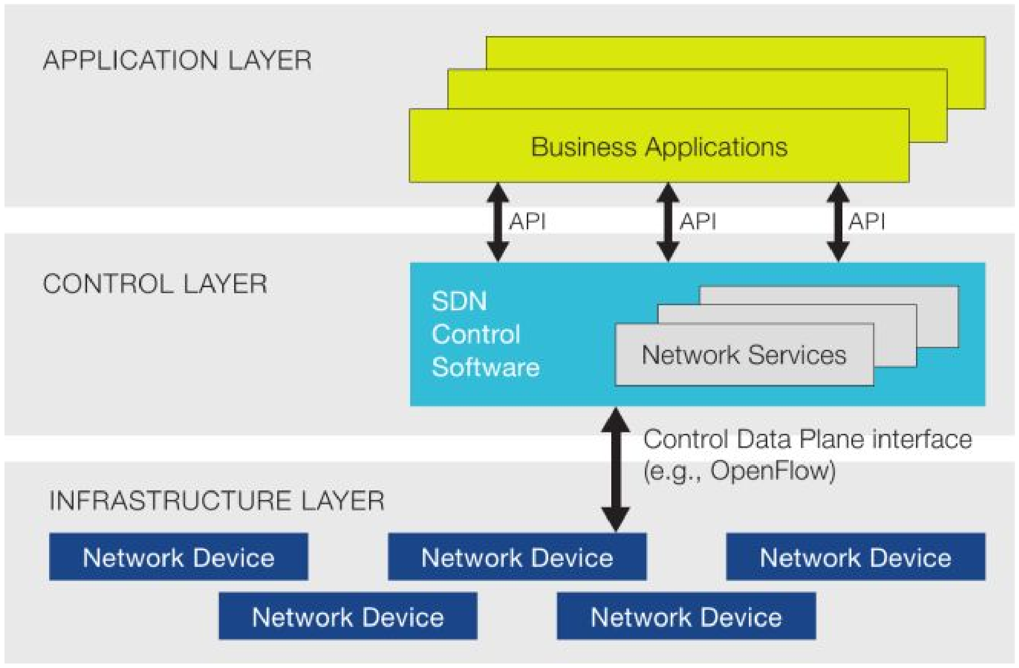


Figure 3 - SDN Arquitecture

A model for SDN is Openflow, which was proposed at Stanford University in the United States. Openflow uses information contained in received packet protocol headers to identify packet flows, and thus apply routing rules statically or dynamically defined by a controller. The controller can reside in other equipment, allowing the centralized control of the network. With this, the network administrator can define how certain packet flows must be routed through the network equipment. This model makes it possible, at one time, to perform traffic balancing and treat the problem with redundant links. To do this, it is necessary to identify the particular packet flows and assume different paths through the switches, in order to use all available resources and avoid cyclical routing.

To avoid this problem, protocols have been developed that alter the network topology logically avoiding redundant paths. Spanning Tree (STP, RSTP, MSTP) protocols are the standard protocols for addressing this problem.

2. Spanning Tree Protocol (STP, RSTP, MSTP)

The STP is a protocol specified by the IEEE 802.1d - 1998 standard. The protocol principle is to form a logical topology without loops. Therefore, it is necessary that all Bridges and Switches are enabled with the mechanism so that the protocol performs the communication between the equipments and defines a topology free of closed paths. For this, the STP protocol sends its configuration messages between the switches, which according to the switch ID chooses who will be the root switch . From this, a topology will be determined according to the ID of the others and the cost of the links, so that they do not form a loop between them. Although the use of these protocols solves the problem of frames being routed indefinitely by the switches, it ends up wasting physical resources forcing them to be disabled.

3. Block or Forward: The decision process

In STP, BPDU packets are exchanged between bridges to compare the bridge and port information, and decide whether or not frame transfer of each port is available (block or foward). This is achieved by the following procedure:

I. Selecting the root bridge

The Bridge ID is the switch identifier parameter that can be set by the administrator. It is composed of priority and MAC address. If the priority is not set, the MAC address will be the determinant.
The bridge having the smallest bridge ID is selected as the root bridge through BPDU packet exchange between bridges. After that, only the root bridge sends the original BPDU packet and other bridges transfer BPDU packets received from the root bridge.

Note: The bridge ID is calculated through a combination of the bridge priority set for each bridge and the MAC address of the specific port.

Bridge ID

Upper 2byte	Lower 6byte
Bridge priority	MAC address

II. Deciding the role of ports

Based on the cost of each port to reach the root bridge, decide the role of the ports (Figure 4).
There are three types of ports:

- *Root port*
    - The port having the smallest cost among bridges to reach the root bridge. This port receives BPDU packets from the root bridge.
- *Designated ports*
    - Ports at the side having the small cost to reach the root bridge of each link. These ports sends BPDU packets received from the root bridge. Root bridge ports are all designated ports.
- *Non designated ports*
    - Ports other than the root port and designated port. These ports suppress frame transfer.

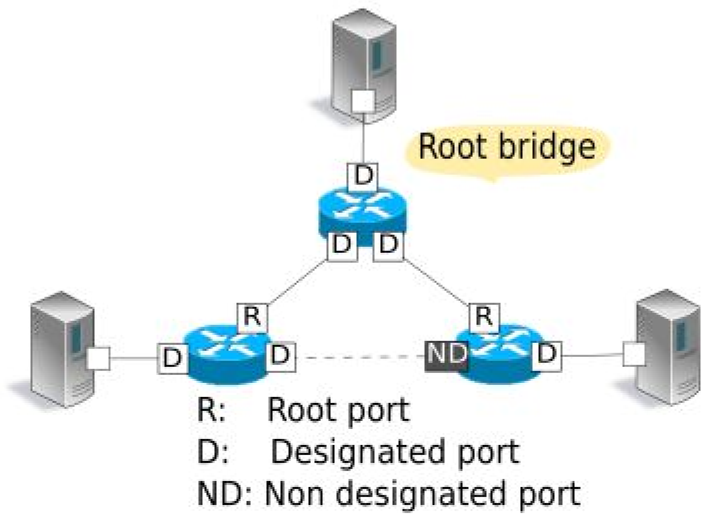
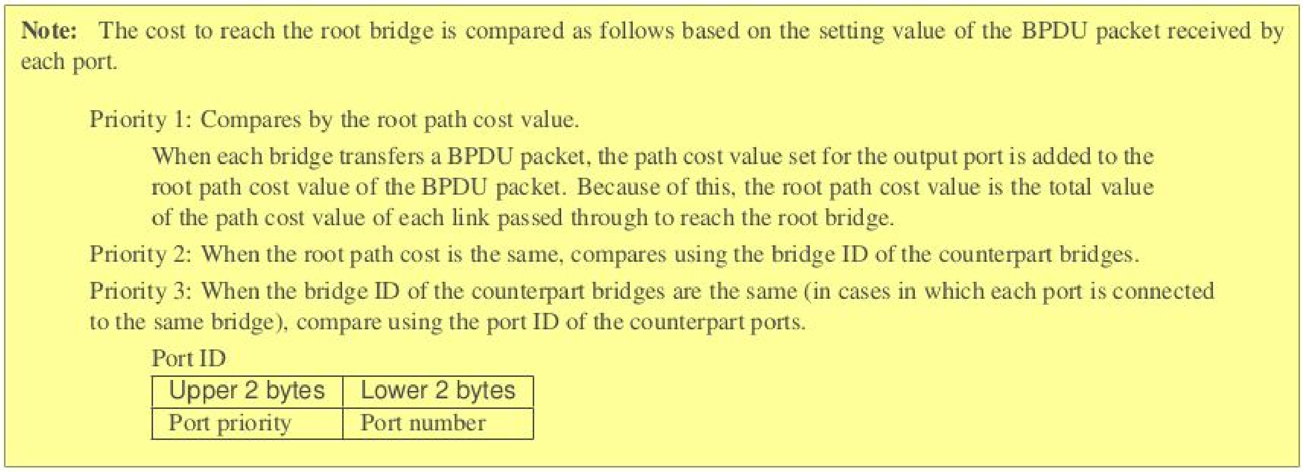


Figure 4 - Role of the ports


Note: The cost to reach the root bridge is compared as follows based on the setting value of the BPDU packet received by each port.

Priority 1: Compares by the root path cost value.

When each bridge transfers a BPDU packet, the path cost value set for the output port is added to the root path cost value of the BPDU packet. Because of this, the root path cost value is the total value of the path cost value of each link passed through to reach the root bridge.

Priority 2: When the root path cost is the same, compares using the bridge ID of the counterpart bridges.

Priority 3: When the bridge ID of the counterpart bridges are the same (in cases in which each port is connected to the same bridge), compare using the port ID of the counterpart ports.

Port ID

Upper 2 bytes	Lower 2 bytes
Port priority	Port number

In short, the steps for deciding the role of ports are as follows:

- The switch with the lowest ID becomes the Root;
- The port of each lower cost switch to Root Bridge will be determined as the root port (R);
- In each segment will be determined as designated ports (D) the lowest cost ports to reach the Root. If the cost is the same, the designated port will be the port of the switch that has the lowest ID;
- The designated ports and routers will be marked as routing ports, the other ports will be blocked ports (ND)

Each device will contain the information of its root bridge ID. The transmission rates of the links are also used as a way to prioritize the best paths.

III. Port state change

In STP 6 types of states are defined for the switches ports (Table 1 and Figure 5):

- ***Blocking***: This is the port that would form the loop if it were not locked. Do not forward frames, but can change its state to forwarding if any fails occurs elsewhere in the network;
- ***Listening***: processes BPDUs and waits for new information that may cause your state to go back to blocking. Does not feed the MAC table and does not forward frames
- ***Learning***: feeds the MAC table, but does not forward frames;
- ***Forwarding***: operates normally. Forwards and receives frames;
- ***Initializing***: only initializes the port, and can be kept active or deactivated by the administrator;
- ***Disabled***: does not forward frames and does not participate in the spanning tree configuration.

Table 1 - STP - Port States

STP Modes	**Receive BPDUs**	**Sends BPDUs**	**Learns MAC Address**	**Forward Data Packets**
Disabled				
Blocking	🟩			
Listening	🟩	🟩		
Learning	🟩	🟩	🟩	
Forwarding	🟩	🟩	🟩	🟩

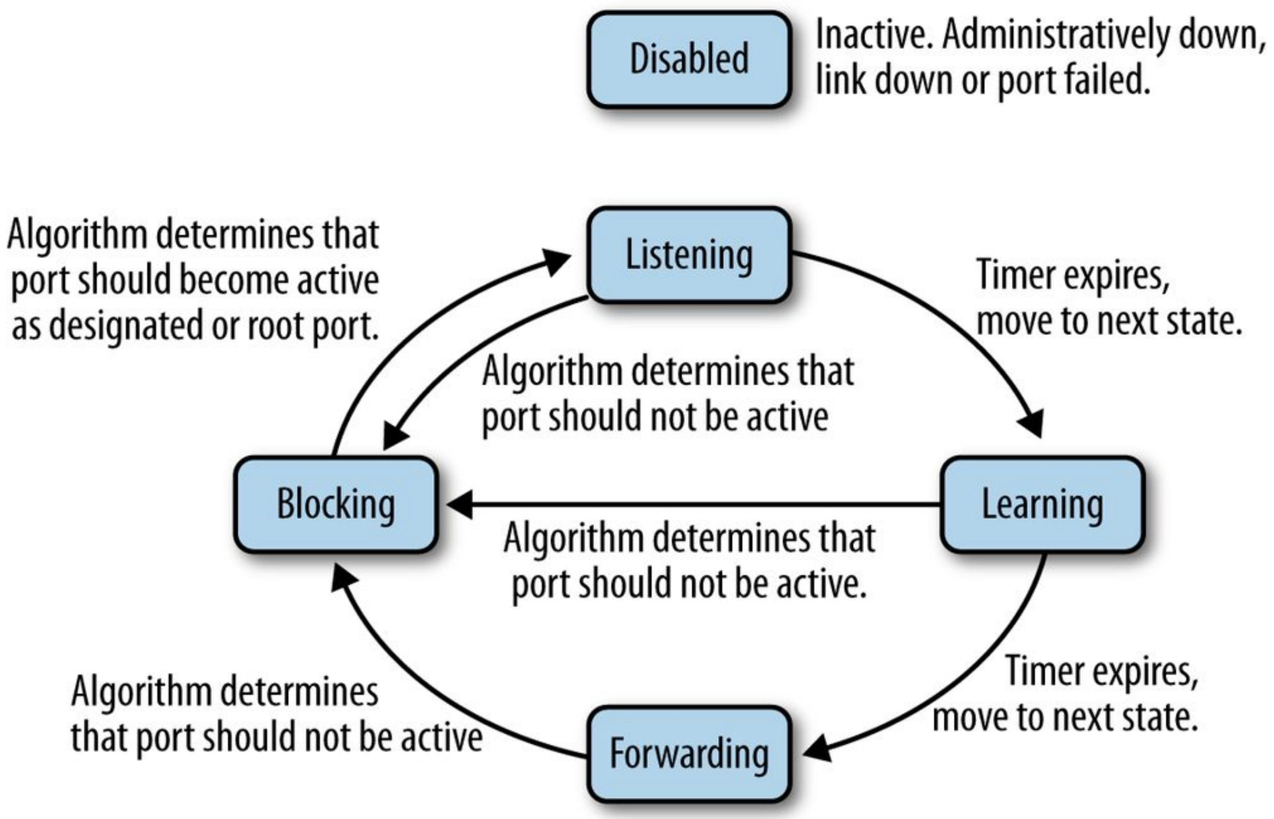

Disabled
Inactive. Administratively down, link down or port failed.
Listening
Algorithm determines that port should become active as designated or root port.
Timer expires, move to next state.
Algorithm determines that port should not be active
Blocking
Learning
Algorithm determines that port should not be active.
Timer expires, move to next state.
Algorithm determines that port should not be active
Forwarding

Figura 5 - STP port states

4. Openflow

Openflow is a protocol that enables the implementation of SDN in different routers and switches. It acts on both the control layer and the physical layer of the switch (datapath), making the communication interface between them. Defines message streams according with rules defined through some parameters such as the header fields of the packets, such as MAC address, IP address, TCP / UDP ports. For these flows are defined actions like forward, drop, rewrite, or "send to the controller".

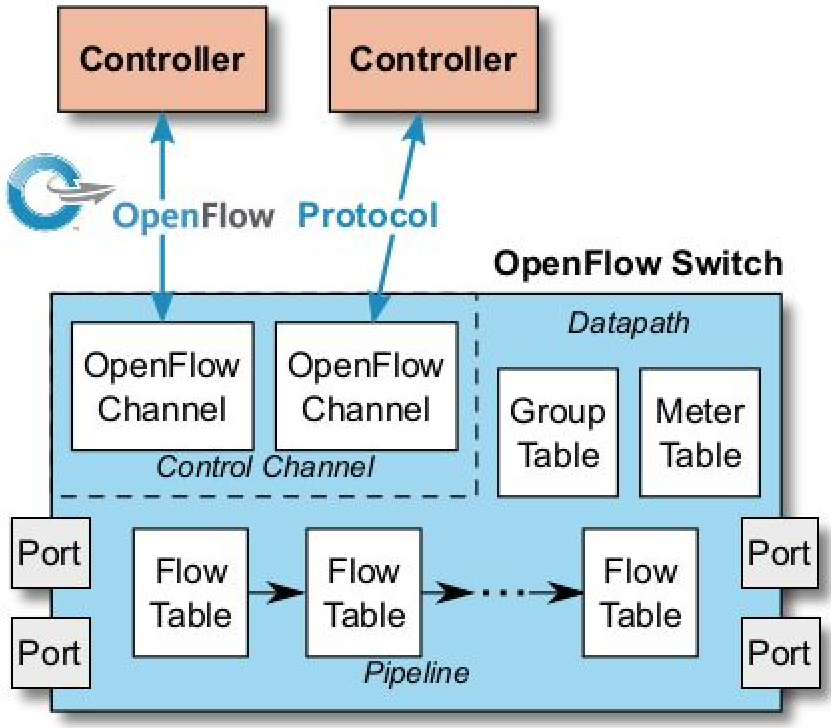


Figure 6 - Main components of a OpenFlow Switch

Demonstration of spanning tree using Ryu STP Controller

1. **Executing the Ryu Application**

In this demo using the Spanning Tree controller app from in Ryu, will be created a network with multiple paths between hosts. If a path is broken, the controller tries and updates the path, if possible.
We will also use MiniNAM, a utility that provides real-time animation of network created.

2. **Installing the required software (for a Linux Mint 19.1 - Cinnamon)**
    a. Virtual machine image (OVF Format, 64-bit, Mininet 2.2.2) avaliable at https://github.com/mininet/openflow-tutorial/wiki/Installing-Required-Software
    The file include virtualization software, a SSH-capable terminal, an X server.
    b. Configure the network adapter, in the VM configurations , as "*host-only*"
    c. Initialize and login the VM
    d. Check the VM's IP address. It will be necessaire later for SSH sessions:
    ```
    mininet@mininet-vm:~$ ifconfig
    ```
    In this demo, the IP address obtained was 192.168.56.101.

3. **Install visualization tool: MiniNAM**
    a. Install requirements:
    ```
    mininet@mininet-vm:~$ sudo apt-get install git
    python-imaging python-imaging-tk
    ```
    b. Clone MiniNAN repository
    ```
    $ git clone https://github.com/uccmisl/MiniNAM.git
    ```

4. **Open SSH sessions in the host machine**
    a. Open 3 SSH connections to be use with the:
        - Ryu STP controller app
        - Mininet/MiniNAM experimental environment
        - IDE for python programming
    b. The SSH connections to mininet VM are done with:
    ```
    user@host:~$ ssh mininet@192.168.56.101 -X
    ```

5. **Executing the Ryu STP Application**
    a. To see the behavior where routing will not work due to loop, you can run a simple switch:
    ```
    mininet@mininet-vm:~$ sudo ryu-manager ./simple_switch_13.py
    ```
    b. In this demo we will only simply start the following controller:
    ```
    mininet@mininet-vm:~$ sudo ryu-manager ./simple_switch_stp_13.py
    ```

6. **Configuring the Experimental Environment**
    a. Start MiniNAN. executing the program in the VM environment:
    ```
    mininet@mininet-vm:~$ sudo python MiniNAM.py --custom
    spanning_tree.py --topo mytopo --controller remote
    ```
    b. A topology is created in which a loop exists between switches s1, s2, and s3.

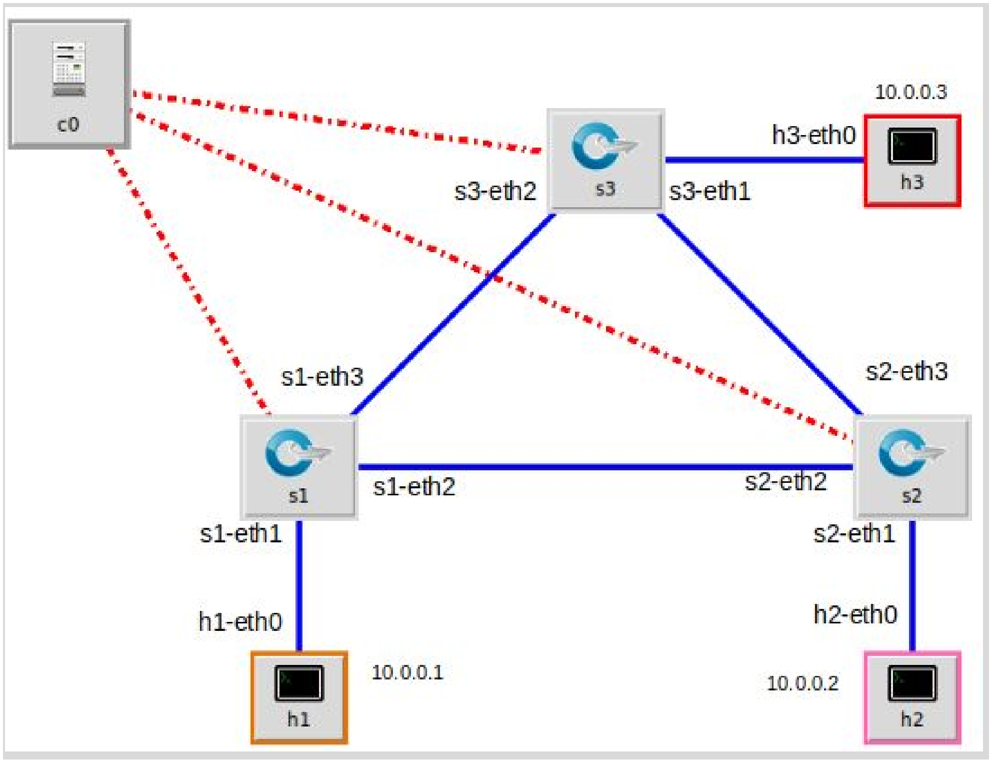


Figure 7 - Topology created in which a loop exists

A list of current links is as follows:

```
mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s2-eth1
h3 h3-eth0:s3-eth1
s1 lo:  s1-eth1:h1-eth0 s1-eth2:s2-eth2 s1-eth3:s3-eth2
s2 lo:  s2-eth1:h2-eth0 s2-eth2:s1-eth2 s2-eth3:s3-eth1
s3 lo:  s3-eth1:h3-eth0 s3-eth2:s2-eth3 s3-eth3:s1-eth3
c0
```

7. Setting the Openflow version of switches

a. Set the switches to OF13 by running following as sudo:

```
sudo  ovs-vsctl set Bridge s1 protocols=OpenFlow13
sudo  ovs-vsctl set Bridge s2 protocols=OpenFlow13
sudo  ovs-vsctl set Bridge s3 protocols=OpenFlow13
```

8. Calculating the STP upon OpenFlow Switch Starts

When connection between each OpenFlow switch and the controller is completed, exchange of BPDU packets starts and root bridge selection, port role setting, and port state change takes place.

```
[STP][INFO] dpid=0000000000000001: Join as stp bridge.
[STP][INFO] dpid=0000000000000001: [port=1] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000001: [port=2] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000001: [port=3] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000003: Join as stp bridge.
[STP][INFO] dpid=0000000000000003: [port=1] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000003: [port=2] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000003: [port=3] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000001: [port=3] Receive superior BPDU.
[STP][INFO] dpid=0000000000000001: [port=1] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000001: [port=2] DESIGNATED_PORT     / BLOCK
```

```
[STP][INFO] dpid=0000000000000001: [port=3] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000001: Root bridge.
[STP][INFO] dpid=0000000000000001: [port=1] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000001: [port=2] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000001: [port=3] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000003: [port=3] Receive superior BPDU.
[STP][INFO] dpid=0000000000000003: [port=1] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000003: [port=2] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000003: [port=3] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000003: Non root bridge.
[STP][INFO] dpid=0000000000000003: [port=1] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000003: [port=2] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000003: [port=3] ROOT_PORT           / LISTEN
[STP][INFO] dpid=0000000000000001: [port=1] DESIGNATED_PORT     / LEARN
[STP][INFO] dpid=0000000000000001: [port=2] DESIGNATED_PORT     / LEARN
[STP][INFO] dpid=0000000000000001: [port=3] DESIGNATED_PORT     / LEARN
[STP][INFO] dpid=0000000000000003: [port=1] DESIGNATED_PORT     / LEARN
[STP][INFO] dpid=0000000000000003: [port=2] DESIGNATED_PORT     / LEARN
[STP][INFO] dpid=0000000000000003: [port=3] ROOT_PORT           / LEARN
[STP][INFO] dpid=0000000000000001: [port=1] DESIGNATED_PORT     / FORWARD
[STP][INFO] dpid=0000000000000001: [port=2] DESIGNATED_PORT     / FORWARD
[STP][INFO] dpid=0000000000000001: [port=3] DESIGNATED_PORT     / FORWARD
[STP][INFO] dpid=0000000000000003: [port=1] DESIGNATED_PORT     / FORWARD
[STP][INFO] dpid=0000000000000003: [port=2] DESIGNATED_PORT     / FORWARD
[STP][INFO] dpid=0000000000000003: [port=3] ROOT_PORT           / FORWARD
[STP][INFO] dpid=0000000000000002: Join as stp bridge.
[STP][INFO] dpid=0000000000000002: [port=1] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000002: [port=2] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000002: [port=3] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000001: [port=2] Receive superior BPDU.
[STP][INFO] dpid=0000000000000001: [port=1] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000001: [port=2] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000001: [port=3] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000001: Root bridge.
[STP][INFO] dpid=0000000000000001: [port=1] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000001: [port=2] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000001: [port=3] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000003: [port=2] Receive superior BPDU.
[STP][INFO] dpid=0000000000000003: [port=1] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000003: [port=2] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000003: [port=3] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000003: Non root bridge.
[STP][INFO] dpid=0000000000000003: [port=1] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000003: [port=2] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000003: [port=3] ROOT_PORT           / LISTEN
[STP][INFO] dpid=0000000000000002: [port=2] Receive superior BPDU.
[STP][INFO] dpid=0000000000000002: [port=1] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000002: [port=2] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000002: [port=3] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000002: Non root bridge.
[STP][INFO] dpid=0000000000000002: [port=1] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000002: [port=2] ROOT_PORT           / LISTEN
[STP][INFO] dpid=0000000000000002: [port=3] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000002: [port=3] Receive superior BPDU.
[STP][INFO] dpid=0000000000000002: [port=1] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000002: [port=2] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000002: [port=3] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000002: Non root bridge.
[STP][INFO] dpid=0000000000000002: [port=1] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000002: [port=2] ROOT_PORT           / LISTEN
[STP][INFO] dpid=0000000000000002: [port=3] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000003: [port=2] Receive superior BPDU.
[STP][INFO] dpid=0000000000000003: [port=1] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000003: [port=2] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000003: [port=3] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000003: Non root bridge.
[STP][INFO] dpid=0000000000000003: [port=1] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000003: [port=2] NON_DESIGNATED_PORT / LISTEN
[STP][INFO] dpid=0000000000000003: [port=3] ROOT_PORT           / LISTEN
[STP][INFO] dpid=0000000000000001: [port=1] DESIGNATED_PORT     / LEARN
```

```
[STP][INFO] dpid=0000000000000001: [port=2] DESIGNATED_PORT     / LEARN
[STP][INFO] dpid=0000000000000001: [port=3] DESIGNATED_PORT     / LEARN
[STP][INFO] dpid=0000000000000002: [port=1] DESIGNATED_PORT     / LEARN
[STP][INFO] dpid=0000000000000002: [port=2] ROOT_PORT           / LEARN
[STP][INFO] dpid=0000000000000002: [port=3] DESIGNATED_PORT     / LEARN
[STP][INFO] dpid=0000000000000003: [port=1] DESIGNATED_PORT     / LEARN
[STP][INFO] dpid=0000000000000003: [port=2] NON_DESIGNATED_PORT / LEARN
[STP][INFO] dpid=0000000000000003: [port=3] ROOT_PORT           / LEARN
[STP][INFO] dpid=0000000000000001: [port=1] DESIGNATED_PORT     / FORWARD
[STP][INFO] dpid=0000000000000001: [port=2] DESIGNATED_PORT     / FORWARD
[STP][INFO] dpid=0000000000000001: [port=3] DESIGNATED_PORT     / FORWARD
[STP][INFO] dpid=0000000000000002: [port=1] DESIGNATED_PORT     / FORWARD
[STP][INFO] dpid=0000000000000002: [port=2] ROOT_PORT           / FORWARD
[STP][INFO] dpid=0000000000000002: [port=3] DESIGNATED_PORT     / FORWARD
[STP][INFO] dpid=0000000000000003: [port=1] DESIGNATED_PORT     / FORWARD
[STP][INFO] dpid=0000000000000003: [port=3] ROOT_PORT           / FORWARD
[STP][INFO] dpid=0000000000000003: [port=2] NON_DESIGNATED_PORT / BLOCK
```

As a result, each por eventually becomes FORWARD state or BLOCK state:

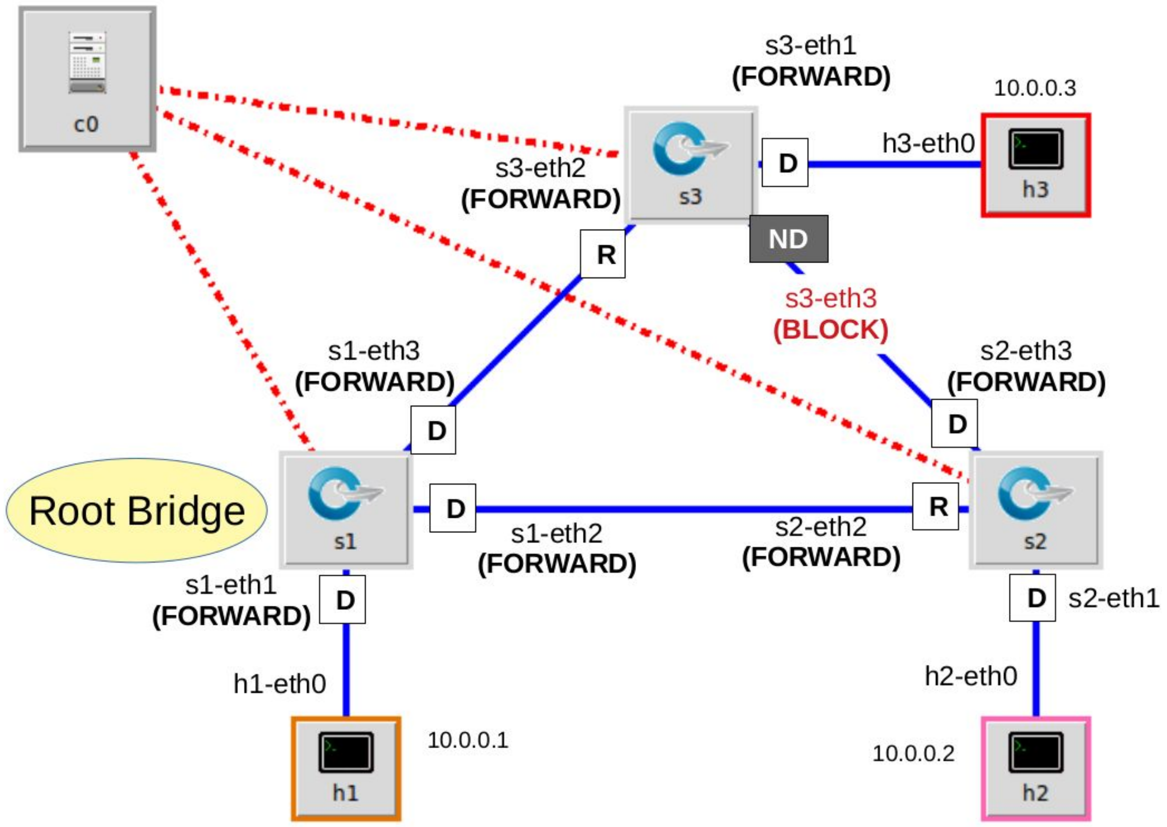


Figure 8 - Network state after the STP algorithm calculations completion

9. Verify that the packets are not looped.

Next, in order to confirm that packets are not looped, execute ping from host 1 to host 2. Before executing the ping command, execute the tcpdump command.

- Node: s1:

  ```
  sudo tcpdump -i s1-eth2 arp
  ```

- Node: s2:

  ```
  sudo tcpdump -i s2-eth2 arp
  ```

- Node: s3:

  ```
  sudo tcpdump -i s3-eth2 arp
  ```

On the console where the topology configuration script is executed, execute the following commands to issue a ping from host 1 to host 2.

```
mininet> h1 ping -c11 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=18.3 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=25.0 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=3.02 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=31.0 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=17.7 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=10.6 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=0.369 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=22.8 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=9.88 ms
64 bytes from 10.0.0.2: icmp_seq=10 ttl=64 time=16.2 ms
64 bytes from 10.0.0.2: icmp_seq=11 ttl=64 time=4.62 ms

--- 10.0.0.2 ping statistics ---
11 packets transmitted, 11 received, 0% packet loss, time 10015ms
rtt min/avg/max/mdev = 0.369/14.530/31.028/9.281 ms
```

As a result of tcpdump output, you can confirm that ARP is not looped.

- Node: s1:

```
mininet> h1 ping -c11 h2
mininet@mininet-vm:~$ sudo tcpdump -i s1-eth2 arp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on s1-eth2, link-type EN10MB (Ethernet), capture size 262144 bytes
17:08:31.903268 ARP, Request who-has 10.0.0.2 tell 10.0.0.1, length 28
17:08:31.910437 ARP, Reply 10.0.0.2 is-at 3a:e8:46:c0:87:cc (oui Unknown), length 28
17:08:36.930603 ARP, Request who-has 10.0.0.1 tell 10.0.0.2, length 28
17:08:36.930723 ARP, Reply 10.0.0.1 is-at a2:8d:2e:9d:94:a3 (oui Unknown), length 28
```

- Node: s2:

```
ininet@mininet-vm:~$ sudo tcpdump -i s2-eth2 arp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on s2-eth2, link-type EN10MB (Ethernet), capture size 262144 bytes
17:08:31.903274 ARP, Request who-has 10.0.0.2 tell 10.0.0.1, length 28
17:08:31.910431 ARP, Reply 10.0.0.2 is-at 3a:e8:46:c0:87:cc (oui Unknown), length
28
17:08:36.930593 ARP, Request who-has 10.0.0.1 tell 10.0.0.2, length 28
17:08:36.930726 ARP, Reply 10.0.0.1 is-at a2:8d:2e:9d:94:a3 (oui Unknown), length
28
```

- Node: s3:

```
mininet@mininet-vm:~$ sudo tcpdump -i s3-eth2 arp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on s3-eth2, link-type EN10MB (Ethernet), capture size 262144 bytes
17:08:31.906015 ARP, Request who-has 10.0.0.2 tell 10.0.0.1, length 28
```

10. Re-Calculation When a Failure is Detected

Next, let’s check re-calculation operation of STP in case of link down. In the state in which STP calculation has been completed after each OpenFlow switch starts, execute the following commands to make the port down.

- Node: s2:

`ifconfig s2-eth2 down`

Link down is detected and recalculation of STP is executed.

```
[STP][INFO] dpid=0000000000000001: [port=2] Link down.
```

```
[STP][INFO] dpid=0000000000000001: [port=2] DESIGNATED_PORT     / DISABLE
[STP][INFO] dpid=0000000000000002: [port=2] Link down.
[STP][INFO] dpid=0000000000000002: [port=2] DESIGNATED_PORT     / DISABLE
[STP][INFO] dpid=0000000000000002: [port=1] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000002: [port=3] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000002: Root bridge.
[STP][INFO] dpid=0000000000000002: [port=1] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000002: [port=3] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000002: [port=1] DESIGNATED_PORT     / LEARN
[STP][INFO] dpid=0000000000000002: [port=3] DESIGNATED_PORT     / LEARN
[STP][INFO] dpid=0000000000000003: [port=2] Wait BPDU timer is exceeded.
[STP][INFO] dpid=0000000000000003: [port=1] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000003: [port=2] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000003: [port=3] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000003: Root bridge.
[STP][INFO] dpid=0000000000000003: [port=1] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000003: [port=2] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000003: [port=3] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000002: [port=3] Receive superior BPDU.
[STP][INFO] dpid=0000000000000002: [port=1] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000002: [port=3] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000002: Root bridge.
[STP][INFO] dpid=0000000000000002: [port=1] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000002: [port=3] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000003: [port=2] Receive superior BPDU.
[STP][INFO] dpid=0000000000000003: [port=1] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000003: [port=2] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000003: [port=3] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000003: Non root bridge.
[STP][INFO] dpid=0000000000000003: [port=1] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000003: [port=2] ROOT_PORT           / LISTEN
[STP][INFO] dpid=0000000000000003: [port=3] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000003: [port=3] Receive superior BPDU.
[STP][INFO] dpid=0000000000000003: [port=1] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000003: [port=2] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000003: [port=3] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000003: Non root bridge.
[STP][INFO] dpid=0000000000000003: [port=1] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000003: [port=2] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000003: [port=3] ROOT_PORT           / LISTEN
[STP][INFO] dpid=0000000000000002: [port=3] Receive superior BPDU.
[STP][INFO] dpid=0000000000000002: [port=1] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000002: [port=3] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000002: Non root bridge.
[STP][INFO] dpid=0000000000000002: [port=1] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000002: [port=3] ROOT_PORT           / LISTEN
[STP][INFO] dpid=0000000000000003: [port=1] DESIGNATED_PORT     / LEARN
[STP][INFO] dpid=0000000000000003: [port=2] DESIGNATED_PORT     / LEARN
[STP][INFO] dpid=0000000000000003: [port=3] ROOT_PORT           / LEARN
[STP][INFO] dpid=0000000000000002: [port=1] DESIGNATED_PORT     / LEARN
[STP][INFO] dpid=0000000000000002: [port=3] ROOT_PORT           / LEARN
[STP][INFO] dpid=0000000000000003: [port=1] DESIGNATED_PORT     / FORWARD
[STP][INFO] dpid=0000000000000003: [port=2] DESIGNATED_PORT     / FORWARD
[STP][INFO] dpid=0000000000000003: [port=3] ROOT_PORT           / FORWARD
[STP][INFO] dpid=0000000000000002: [port=1] DESIGNATED_PORT     / FORWARD
[STP][INFO] dpid=0000000000000002: [port=3] ROOT_PORT           / FORWARD
```

You can confirm that the s3-eth2 port, which was in BLOCK state, became FORWARD state and frame transfer became available again.

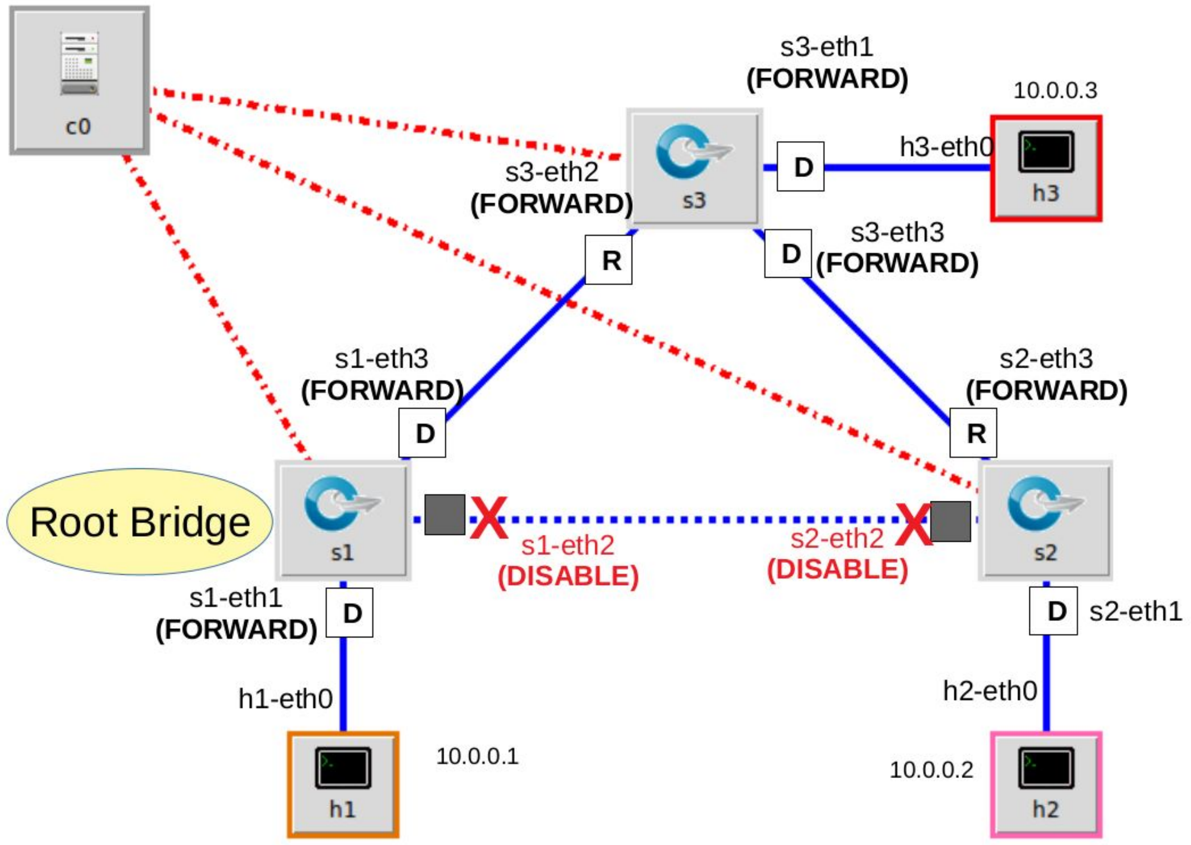


Figure 9 - Re-calculated network state after when a failure is detected

11. Recalculation of STP At Failure Recovery

Next, check operation of recalculation of STP when link down is recovered. To start the port execute the following commands while the link is down.

- Node: s2:

```
ifconfig s2-eth2 up
```

Link recovery is detected and STP re-calculation is executed.

```
[STP][INFO] dpid=0000000000000002: [port=2] Link up.
[STP][INFO] dpid=0000000000000002: [port=2] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000001: [port=2] Link up.
[STP][INFO] dpid=0000000000000001: [port=2] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000001: [port=2] Receive superior BPDU.
[STP][INFO] dpid=0000000000000001: [port=1] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000001: [port=2] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000001: [port=3] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000001: Root bridge.
[STP][INFO] dpid=0000000000000001: [port=1] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000001: [port=2] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000001: [port=3] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000002: [port=2] Receive superior BPDU.
[STP][INFO] dpid=0000000000000002: [port=1] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000002: [port=2] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000002: [port=3] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000002: Non root bridge.
[STP][INFO] dpid=0000000000000002: [port=1] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000002: [port=2] ROOT_PORT           / LISTEN
[STP][INFO] dpid=0000000000000002: [port=3] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000003: [port=2] Receive superior BPDU.
[STP][INFO] dpid=0000000000000003: [port=1] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000003: [port=2] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000003: [port=3] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=0000000000000003: Non root bridge.
[STP][INFO] dpid=0000000000000003: [port=1] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=0000000000000003: [port=2] NON_DESIGNATED_PORT / LISTEN
[STP][INFO] dpid=0000000000000003: [port=3] ROOT_PORT           / LISTEN
[STP][INFO] dpid=0000000000000001: [port=1] DESIGNATED_PORT     / LEARN
```

```
[STP][INFO] dpid=0000000000000001: [port=2] DESIGNATED_PORT     / LEARN
[STP][INFO] dpid=0000000000000001: [port=3] DESIGNATED_PORT     / LEARN
[STP][INFO] dpid=0000000000000002: [port=1] DESIGNATED_PORT     / LEARN
[STP][INFO] dpid=0000000000000002: [port=2] ROOT_PORT           / LEARN
[STP][INFO] dpid=0000000000000002: [port=3] DESIGNATED_PORT     / LEARN
[STP][INFO] dpid=0000000000000003: [port=1] DESIGNATED_PORT     / LEARN
[STP][INFO] dpid=0000000000000003: [port=2] NON_DESIGNATED_PORT / LEARN
[STP][INFO] dpid=0000000000000003: [port=3] ROOT_PORT           / LEARN
[STP][INFO] dpid=0000000000000001: [port=1] DESIGNATED_PORT     / FORWARD
[STP][INFO] dpid=0000000000000001: [port=2] DESIGNATED_PORT     / FORWARD
[STP][INFO] dpid=0000000000000001: [port=3] DESIGNATED_PORT     / FORWARD
[STP][INFO] dpid=0000000000000002: [port=1] DESIGNATED_PORT     / FORWARD
[STP][INFO] dpid=0000000000000002: [port=2] ROOT_PORT           / FORWARD
[STP][INFO] dpid=0000000000000002: [port=3] DESIGNATED_PORT     / FORWARD
[STP][INFO] dpid=0000000000000003: [port=1] DESIGNATED_PORT     / FORWARD
[STP][INFO] dpid=0000000000000003: [port=2] NON_DESIGNATED_PORT / BLOCK
[STP][INFO] dpid=0000000000000003: [port=3] ROOT_PORT           / FORWARD
```

You can confirm that the tree structure becomes the same as that in effect when the application starts and frame transfer becomes available again.

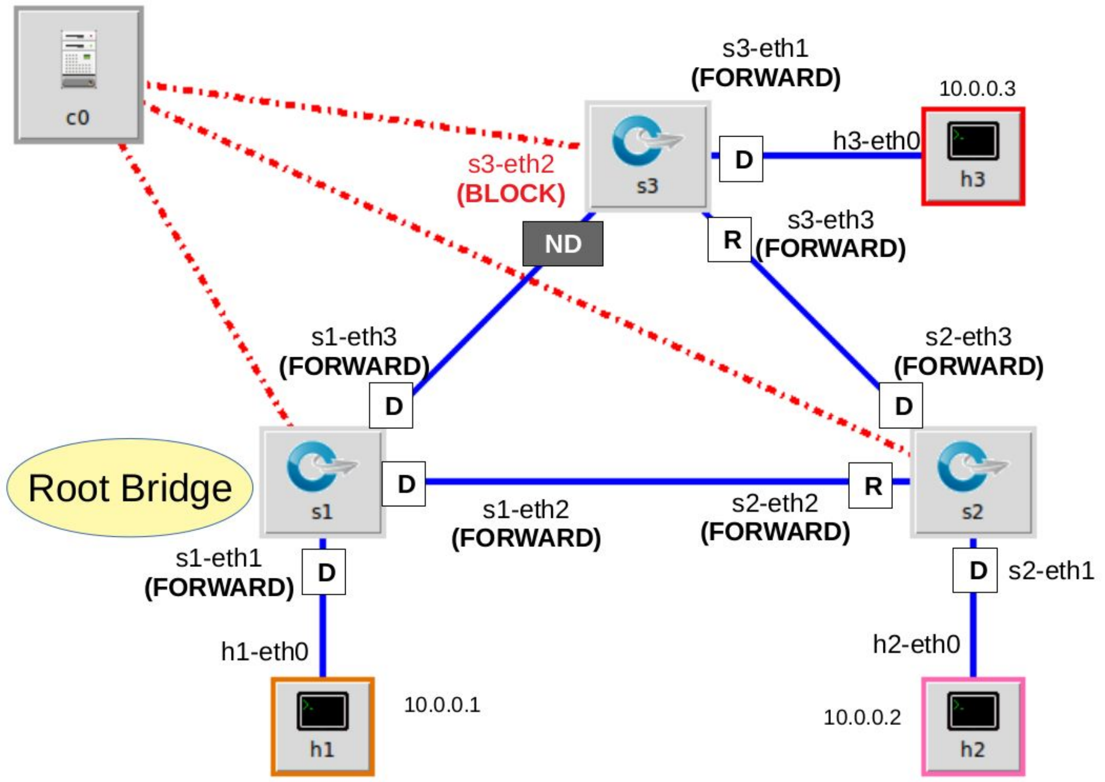


Figure 10 - Re-calculated network state after at failure recovery

Using Ryu to Implement Spanning Tree

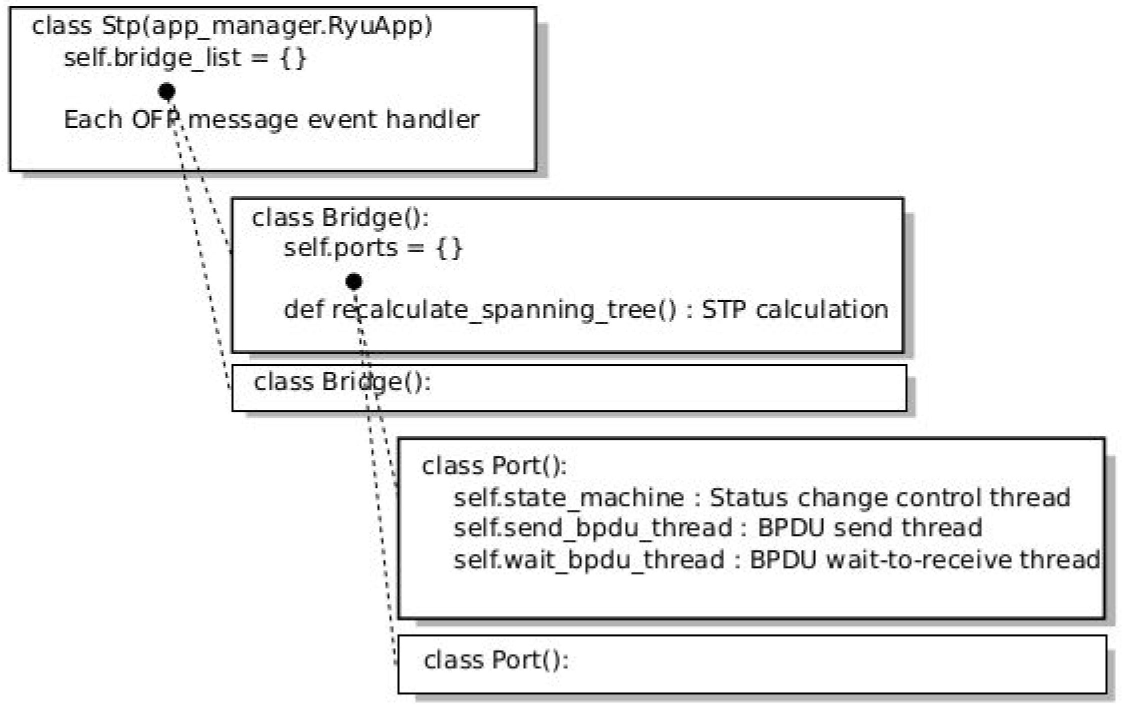
Next, let's take a look at the source code of spanning tree implemented using Ryu. The spanning tree source code is in the Ryu's source tree.

- ryu/lib/**stplib.py**

  stplib.py is a library that provides spanning tree functions such as BPDU packet exchange and management of rules, and the status of each port.
- ryu/app/**simple_switch_stp_13.py**

  The *simple_switch_stp_13.py* is an application program in which the spanning tree function is added to the switching hub application using the spanning tree library.

1. **Implementing the Library**
   Library Overview


class Stp(app_manager.RyuApp)
self.bridge_list = {}
Each OFP message event handler
class Bridge():
self.ports = {}
def recalculate_spanning_tree() : STP calculation
class Bridge():
class Port():
self.state_machine : Status change control thread
self.send_bpdu_thread : BPDU send thread
self.wait_bpdu_thread : BPDU wait-to-receive thread
class Port():

When the *STP library* (Stp class instance) detects connection of an OpenFlow switch to the controller, a *Bridge class* instance and *Port class* instance are generated.
After each class instance is generated and started, three types of events are monitored:

- Notification of the OpenFlow message reception from the Stp class instance
- STP calculation of the Bridge class instance (loot bridge selection and selection of the role of each port)
- Status change of the port of the Port class instance and send/receive of BPDU packets work together to achieve the spanning tree function.

Configurations settings item

The STP library provides the bridge port config setting IF using the `Stp.set_config()` method.
The following items can be set:

- bridge

Item	Explanation	Default value
`priority`	Bridge priority	0x8000
`sys_ext_id`	Sets VLAN-ID (*the current STP library does not support VLAN)	0
`max_age`	Timer value to wait to receive BPDU packets	20[sec]
`hello_time`	Send intervals of BPDU packets	2 [sec]
`fwd_delay`	Period that each port stays in LISTEN or LEARN status	15[sec]

- port

Item	Explanation	Default value
`priority`	Port priority	0x80
`path_cost`	Link cost value	Auto setting based on the link speed
`enable`	Port enable/disable setting	True

Sending BDPU Packet

Receiving BDPU Packets

Detecting Failures

STP Calculation

Port Status Change

2. Implementing the Application

Setting “_CONTEXTS”

A Ryu application that inherits ryu.base.app_manager.RyuApp starts other applications using separate threads by setting other Ryu applications in the “`_CONTEXTS`” dictionary. Here, the Stp class of the STP library is set in “`_CONTEXTS`” in the name of ” stplib”.

```
from ryu.lib import stplib
# ...
class SimpleSwitch13(simple_switch_13.SimpleSwitch13):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]
    _CONTEXTS = {'stplib': stplib.Stp}
```

Applications set in “_CONTEXTS” can acquire instances from the kwargs of the `__init__() method`.

```
    def __init__(self, *args, **kwargs):
        super(SimpleSwitch13, self).__init__(*args, **kwargs)
        self.mac_to_port = {}
        self.stp = kwargs['stplib']
        ...
```

Setting Configuration

Use the `set_config()` method of the STP library to set configuration. Here, the following values are set as a sample.

OpenFlow switch	Item	Setting
dpid=0000000000000001	bridge.priority	0x8000
dpid=0000000000000002	bridge.priority	0x9000
dpid=0000000000000003	bridge.priority	0xa000

Using these settings, the bridge ID of the dpid=0000000000000001 OpenFlow switch is always the smallest value and is selected as the root bridge.

```
    def __init__(self, *args, **kwargs):
        super(SimpleSwitch13, self).__init__(*args, **kwargs)
        self.mac_to_port = {}
        self.stp = kwargs['stplib']

        # Sample of stplib config.
        #  please refer to stplib.Stp.set_config() for details.
        config = {dpid_lib.str_to_dpid('0000000000000001'):
                     {'bridge': {'priority': 0x8000}},
                  dpid_lib.str_to_dpid('0000000000000002'):
                  {'bridge': {'priority': 0x9000}},
                  dpid_lib.str_to_dpid('0000000000000003'):
                  {'bridge': {'priority': 0xa000}}}
        self.stp.set_config(config)
```

STP Event Processing

In this sections we see how is prepared the event handler that receive events notified by the STP library.

By using the `stplib.EventPacketIn` event defined in the STP library, it is possible to receive packets other than BPDU packets

```
    @set_ev_cls(stplib.EventPacketIn, MAIN_DISPATCHER)
    def _packet_in_handler(self, ev):
        msg = ev.msg
        datapath = msg.datapath
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser
        in_port = msg.match['in_port']
```

The change notification event (`stplib.EventTopologyChange`) of the network topology is received and the learned MAC address and registered flow entry are initialized.

```
    def delete_flow(self, datapath):
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

        for dst in self.mac_to_port[datapath.id].keys():
            match = parser.OFPMatch(eth_dst=dst)
            mod = parser.OFPFlowMod(
                datapath, command=ofproto.OFPFC_DELETE,
                out_port=ofproto.OFPP_ANY, out_group=ofproto.OFPG_ANY,
                priority=1, match=match)
            datapath.send_msg(mod)
```

```
    @set_ev_cls(stplib.EventTopologyChange, MAIN_DISPATCHER)
    def _topology_change_handler(self, ev):
        dp = ev.dp
        dpid_str = dpid_lib.dpid_to_str(dp.id)
        msg = 'Receive topology change event. Flush MAC table.'
        self.logger.debug("[dpid=%s] %s", dpid_str, msg)

        if dp.id in self.mac_to_port:
            self.delete_flow(dp)
            del self.mac_to_port[dp.id]
```

The change notification event (`stplib.EventPortStateChange`) of the port status is received and the debug log of the port status is output.

```
    @set_ev_cls(stplib.EventPortStateChange, MAIN_DISPATCHER)
    def _port_state_change_handler(self, ev):
        dpid_str = dpid_lib.dpid_to_str(ev.dp.id)
        of_state = {stplib.PORT_STATE_DISABLE: 'DISABLE',
                    stplib.PORT_STATE_BLOCK: 'BLOCK',
                    stplib.PORT_STATE_LISTEN: 'LISTEN',
                    stplib.PORT_STATE_LEARN: 'LEARN',
                    stplib.PORT_STATE_FORWARD: 'FORWARD'}
        self.logger.debug("[dpid=%s][port=%d] state=%s",
                          dpid_str, ev.port_no, of_state[ev.port_state])
```