Solution Review: Spanning Tree Protocol

In this lesson, we'll look at a solution to the spanning tree protocol programming assignment.

We'll cover the following

Solution

main.py

topology_reader.py

ports.py

simulator.py

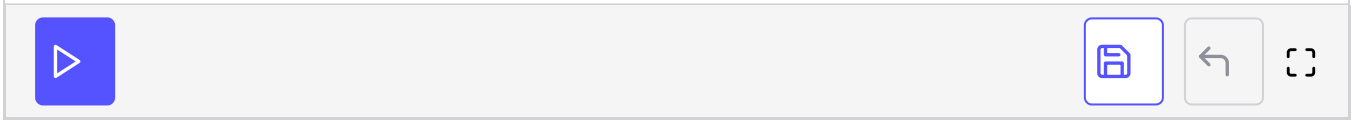
bridge.py

```
from ports import ports

def is_better(BPDU1, BPDU2):
  # If root is greater than BPDU1 is better
  if(BPDU1[0] < BPDU2[0]):
    return 1
  elif(BPDU1[0] == BPDU2[0] and BPDU1[1] < BPDU2[1]):
    return 1
  elif(BPDU1[0] == BPDU2[0] and BPDU1[1] == BPDU2[1] and BPDU1[
    return 1
  elif(BPDU1[0] == BPDU2[0] and BPDU1[1] == BPDU2[1] and BPDU1[
    return 1
  else:
    return 0

class bridge:
  def __init__(self, bridge_ID, port_list):
    self.bridge_ID = bridge_ID
    self.port_list = port_list # port_list[0] is the port with
```

```
20      self.config_BPDU = [bridge_ID, 0, bridge_ID, None] # Root I
21      self.receive_queue = {}
22
23    def initialize_recv_queue(self, bridges_dict):
24      for b in bridges_dict:
25        self.receive_queue[b] = []
26
27    def set_bridge(self, bridge_ID, num_ports, mac_addresses):
28      self.bridge_ID = bridge_ID
29      self.num_ports = num_ports
30      self.port_list = port_list
31
```



Explanation

`send_BPDUs()`

This function called on every bridge in the topology in a round-robin fashion. This function takes the bridges dictionary as input, makes some updates, and returns it.

It first iterates over all of **non-blocking** ports to find its neighbors. The neighbors are found by calling the function `get_reachable_bridge_ID()` on each port. This function returns a list of bridge IDs that are reachable from that port. Each of those bridges is sent the bridge's current BPDU by appending them to that bridge's `receive_queue`.

`receive_BPDUs()`

This function called on every bridge in the topology in a round-robin fashion right after the `send_BPDUs()` function is called. It consists of a number of function calls. Let's discuss each.

1. `find_best_BPDUs_received(bridges_dict)`: this function returns the best BPDUs received on each port. Since each port of a bridge could have received multiple BPDUs, the best ones out of those need to be retrieved first. A dictionary where the key is the bridge number and the received BPDU is the value is generated and returned.

2. `update_ports()`: this function iterates over the best BPDUs and updates the ports to be blocking if the one received is better than the one it would have sent and to forwarding otherwise. It uses the helper function

`is_better()` to do this. `is_better()` is fairly straightforward: it takes two BPDUs as input and returns 1 if the first one is better and 0 if it's not.

3. `elect_root`: lastly, the root is elected. The best BPDUs are iterated over and if any one of them's root ID is smaller than the one that the bridge currently believes to be the root, it's updated to the new values and the port from which this BPDU was received is set to be the new root. Furthermore, if an old root port existed, it is changed to `forwarding`.

Lastly, the dictionary of bridges is updated with `self` and returned.

In the next lesson, we'll study virtual LANs!