Solution Review: Spanning Tree Protocol

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

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[2] < BPDU2[2]):
    return 1
  elif(BPDU1[0] == BPDU2[0] and BPDU1[1] == BPDU2[1] and BPDU1[2] == BPDU2[2] and BPDU1[3] <
    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 port number 0
    self.port_list = port_list # port_list[0] is the port with port number 0
    self.config_BPDU = [bridge_ID, 0, bridge_ID, None] # Root ID, Cost, Transmitting Bridge
    self.receive_queue = {}

  def initialize_recv_queue(self, bridges_dict):
    for b in bridges_dict:
      self.receive_queue[b] = []

  def set_bridge(self, bridge_ID, num_ports, mac_addresses):
    self.bridge_ID = bridge_ID
    self.num_ports = num_ports
    self.port_list = port_list

  def get_port(self, bridge_number, bridges_dict):
    for i in range(len(self.port_list)):
      if(bridge_number in self.port_list[i].get_reachable_bridge_ID(bridges_dict, self.bridg
        return i

  def find_best_BPDUs_received(self, bridges_dict):
    # Select best BPDU at each bridge
    best_BPDUs_recvd = {} # Bridge Number : BPDU
    best_bpdu = 0
    for sending_brg_id in self.receive_queue:
      if(len(self.receive_queue[sending_brg_id]) > 0):
        best_bpdu = self.receive_queue[sending_brg_id][0]
        for bpdu in self.receive_queue[sending_brg_id]:
          if is_better(bpdu, best_bpdu):
            best_bpdu = bpdu
        best_BPDUs_recvd[sending_brg_id] = best_bpdu
    self.initialize_recv_queue(bridges_dict)
    return best_BPDUs_recvd

  def update_ports(self, bridges_dict, best_BPDUs_recvd):
    for sending_brg_id in best_BPDUs_recvd:
      if sending_brg_id is not self.config_BPDU[0]:
        if self.port_list[self.get_port(sending_brg_id, bridges_dict)].port_type != 2:
          pn = self.get_port(sending_brg_id, bridges_dict)
          if(is_better(best_BPDUs_recvd[sending_brg_id], self.get_config_BPDU_at_port(pn))):
            self.port_list[self.get_port(sending_brg_id, bridges_dict)].port_type = 0
          else:
            self.port_list[self.get_port(sending_brg_id, bridges_dict)].port_type = 1

  def elect_root(self, bridges_dict, best_BPDUs_recvd):
    for sending_brg_id in best_BPDUs_recvd:
      if(best_BPDUs_recvd[sending_brg_id][0] < self.config_BPDU[0]):
        # New root bridge
        self.config_BPDU[0] = best_BPDUs_recvd[sending_brg_id][0]
        self.config_BPDU[1] = best_BPDUs_recvd[sending_brg_id][1] + 1
        self.config_BPDU[2] = self.bridge_ID
        self.config_BPDU[3] = self.get_port(sending_brg_id, bridges_dict)
        if(self.get_root_port_id() is not None):
          self.port_list[self.get_root_port_id()].port_type = 1
        self.port_list[self.get_port(sending_brg_id, bridges_dict)].port_type = 2

  def send_BPDUs(self, bridges_dict):
    # Find all neighbors
    for p in self.port_list:
      if(p.get_port_state() > 0): # If sending port
        # Send to all bridges on that segment
        for b in p.get_reachable_bridge_ID(bridges_dict, self.bridge_ID):
          pn = self.get_port(b, bridges_dict)
          bridges_dict[b].receive_queue[self.bridge_ID].append(self.get_config_BPDU_at_port(
    return(bridges_dict)
  def receive_BPDUs(self, bridges_dict):
    # Update the bridge's state according to the best BPDUs received

    # Find best BPDU received from each bridge
    best_BPDUs_recvd = self.find_best_BPDUs_received(bridges_dict)

    # Compare BPDUs with those received at non-root ports
    self.update_ports(bridges_dict, best_BPDUs_recvd)

    # Find root bridge
    self.elect_root(bridges_dict, best_BPDUs_recvd)

    # Update dictionary and return
    bridges_dict[self.bridge_ID] = self
    return bridges_dict

  def get_root_port_id(self):
    for p in range(len(self.port_list)):
      if self.port_list[p].port_type == 2:
        return p
    return None

  def get_config_BPDU_at_port(self, port_number):
    BPDU_at_port = self.config_BPDU
    BPDU_at_port[3] = self.port_list[port_number].mac_address
    return(BPDU_at_port)

  def print_bridge(self):
    print("~~~~~~Bridge ID: " + str(self.bridge_ID) + " Root ID: " + str(self.config_BPDU[0]
    print("BPDU:")
    print(self.config_BPDU)

    print("MAC address | Port Type | Segment Number")
    for port in self.port_list:
      port.print_port()
```

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!