Programming Challenge: Spanning Tree Protocol

WE'LL COVER THE FOLLOWING

Problem Statement

In this challenge, you will implement a simplified version of the spanning tree protocol. For this challenge, you should assume that a switch and a bridge are the same things. The only practical difference is that bridges have few ports, whereas switches have many many ports.

Spanning Tree Protocol

We saw a gist of this protocol in the last lesson. Let's build upon that.

Configuration Bridge Protocol Data Units

The spanning tree is created by the exchange of messages between the bridges. These messages are called **Configuration Bridge Protocol Data Units**, or **Configuration BPDUs**. We'll refer to them as **BPDUs**. The format of a BPDU for this challenge is a list of integers as follows: `[Root ID, Cost, Transmitting Bridge ID, Transmitting Port ID]`

The following table explains what each field of a BPDU means.

Field	Explanation of field
Root ID	ID of the bridge currently believed/known to be root
Transmitting Bridge ID	ID of the bridge transmitting this BPDU
Cost	Cost of the least cost path from the transmitting bridge to the currently known root
Transmitting Port ID	ID of the port out of which BPDU is transmitted

Comparing Bridge Protocol Data Units

Some BPDUs are better than others. Here's how to determine which is better.

- A BPDU with a **lower root bridge ID** is **better** than a BPDU with a higher root bridge ID.
- If the root bridge ID of both BPDUs is the same, then a BPDU with a **lower cost** is **better** than a BPDU with a higher cost.
- If the cost and bridge ID of both BPDUs is the same, then a BPDU with a **lower transmitting bridge ID** is **better** than a BPDU with a higher transmitting bridge ID.
- If the cost, the bridge ID, transmitting bridge ID of both BPDUs is the same, then a BPDU with a **lower transmitting port ID** is **better** than a BPDU with a higher transmitting port ID.

How It Works

As described above, the protocol aims to build a tree in which one bridge is determined to be the root. Bridges can only forward data frames **towards** the root bridge or away from the root bridge. In this way, cycles are avoided.

Each bridge is assigned a unique bridge ID and the root bridge is the one with the smallest bridge ID.

1. When a bridge first boots up, it believes itself to be the root, and hence its BPDU looks like `[its bridge ID, 0, its bridge ID, Port ID]`.
2. It multicasts this BPDU to all of its neighbors. The neighbors of this bridge are all the bridges on the LAN segment to which it is connected via any of its ports.
3. It also receives BPDUs from all of its neighbors.
4. It then processes these BPDUS to determine a couple of things:
    1. The root bridge. It declares a port to be `root` if the root bridge is accessible from this port.
    2. Which ports it should declare `blocking`. It declares a port `blocking` if it receives a better BPDU than the one it would have sent on that port. The ports are forwarding by default.
5. Finally, the bridge sends all of this newly learned information to all bridges accessible via its `forwarding` or `root` ports.

Coding Challenge

We've given you some starter code. Some helper methods have been created that you might find useful and others are declared but left empty. The main task is to fill in the functions `send_BPDUs()` and `receive_BPDUs()`. Good luck!

main.py

topology_reader.py

ports.py

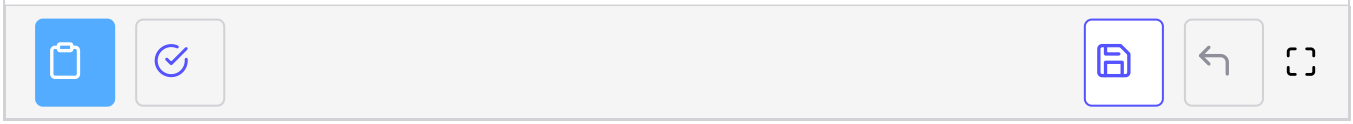
simulator.py

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.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
    # Write your code here
    return best_BPDUs_recvd

  def update_ports(self, bridges_dict, best_BPDUs_recvd):
    # Write your code here
    pass

  def elect_root(self, bridges_dict, best_BPDUs_recvd):
    # Write your code here
    pass

  def send_BPDUs(self, bridges_dict):
    # Write your code here
    return(bridges_dict)

  def receive_BPDUs(self, bridges_dict):
    # These functions are here to give you some structure
    # Feel free to ignore them and implement your own
    # 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()
```



Coming up next, we'll look at the solution to the spanning tree protocol programming challenge!