Self-chosen project report

110550070 郭傑誠

Topic: Spanning tree protocol implementation

Spanning tree protocol:

When there's a loop between a switches network, it can cause broadcast storm due to repeatedly sending and duplicating packets, which can cause network congestion, spanning tree protocol can prevent this situation from happening.

First, STP select a switch as the root bridge, select order is, lowest bridge ID, if same then compared smaller mac address.

All switches in the network first assume itself is the root bridge, then they send hello BPDU to each other to announce it is the root bridge. Once a switch found out there exist a root bridge with lower bridge ID it'll change root bridge to it, and only send its hello BPDU if received.

Next, assigning the ports roles and ports states.

All ports on the root bridge will be designated port, and are put in forwarding state, which can send and receive packets.

Other switches in the topology must have a path to reach the root bridge, they'll choose the least-cost path, the port will be root port, and will be put in forwarding state.

The ports that are connected to the root ports will be assigned as designated ports, put in forwarding state.

The remaining ports will be blocked port, put in blocking state, which cannot send packet from it, and will drop any packets coming in.

STP deals with route failure:

Every switch other then root bridge will have a max age timer = 10*BPDU.

The root bridge will send hello BPDU every two seconds, and the other switches will forward the BPDU out their designated port.

If a switch max age timer expire it will know something was wrong and reevaluate its STP choices.

If a non-designated port is selected to become a designated port or root port it will transition from blocking state to forwarding state after a while (I simplified the steps between).

Code implementation:

a. STP function:

```
def send_bpdu(self):
    while self.root == self.switch_id:
        bpdu = "bpdu " + str(self.switch_id) + " " + str(self.switch_id)
        for neighbor in self.neighbors:
            self.socket.sendto(bpdu.encode(), ('localhost', 10000 + neighbor))

        time.sleep(2)
```

If switch itself is the root bridge, broadcast its BPDU to all the neighbors for every two seconds.

```
  if packet.startswith("bpdu"):
      root = int(packet.split(" ")[2])
      if root != self.root:
          if root < self.root:
              self.root = root
              print("set root to " + str(root))
              self.set = True
              for port in self.ports:
                  self.ports[port]["role"] = ""
                  self.ports[port]["stats"] = 0
      else:
          self.last_received = time.time()

      message = "bpdu " + str(self.switch_id) + " " + str(root)
      for neighbor in self.neighbors:
          if neighbor != root and neighbor != src:
              self.socket.sendto(message.encode(), ('localhost', 10000 + neighbor))
```

When a switch received BPDU, check if the BPDU is same as the current root bridge or not. If is not, check whether the received BPDU ID is lower or not, if lower update current root bridge.

If received BPDU is same as current root bridge, reset the max age timer.

Lastly, send the current root BPDU to its neighbors, except for the source port.

```
def set_port(self):
    while True:
        if self.set:
            if self.root == self.switch_id:
                for port in self.ports:
                    self.ports[port]["role"] = "designated_port"
                self.set = False
                continue
            else:
                if len(self.ports) == 1:
                    self.ports[0]["role"] = "root_port"
                    self.set = False
                    continue
                else:
                    _, p = self.shortest_path_root(None)
                    self.ports[p]["role"] = "root_port"
                    self.socket.sendto(f"dest_port {self.switch_id}".encode(),
                                       ('localhost', 10000 + self.ports[p]["dst"]))
                self.set = False
            time.sleep(2)
            for p in self.ports:
                if self.ports[p]["role"] == "":
                    self.ports[p]["role"] = "blocked_port"
                    self.ports[p]["stats"] = 1
                    self.socket.sendto(f"dest_port {self.switch_id}".encode(),
                                       ('localhost', 10000 + self.ports[p]["dst"]))
```

This part set the roles and states of all port.

Self.set will be true when, a new neighbor is added and root bridge has been changed to a different one.

First, if the switch is root bridge, all the port will be designated port, and are put in forwarding state.

Next, other switches choose a port with least-cost path to be root port, and will be put in forwarding state.

```
def shortest_path_root(self, src):
    d = 999
    po = 0
    for port in self.ports:
        if self.ports[port]["dst"] == src:
            continue
        if self.ports[port]["dst"] == self.root:
            if self.ports[port]["cost"] < d:
                d = self.ports[port]["cost"]
                po = port
        else:
            while self.r is None:
                self.socket.sendto((f"root_dist {self.switch_id}").encode(),
                                   ('localhost', 10000 + self.ports[port]["dst"]))
                time.sleep(2)
                continue
            if self.r + self.ports[port]["cost"] < d:
                d = self.r + self.ports[port]["cost"]
                po = port
            self.r = None
    return d, po
```

```
elif packet.startswith("root_dist"):
    for p in self.ports:
            if self.ports[p]["role"] == "root_port":
                self.socket.sendto(("r_root_dist" + " " + str(self.switch_id) +
                                    " " + str(self.ports[p]["cost"])).encode(),
                                   ('localhost', 10000 + src))
                break
    else:
        distance, _ = self.shortest_path_root(src)
        self.socket.sendto(("r_root_dist " + str(self.switch_id) + " " +
                            str(distance)).encode(),
                           ('localhost', 10000 + src))

elif packet.startswith("r_root_dist"):
    self.r = int(packet.split(" ")[2])
```

Shortest_path_root function can find what path is shortest to the root. It is similar to Bellman Ford equation, but instead of recording and sending estimated distance vector until converge, it'll only asked other neighbor its cost to root only when needed to calculate.

Back to set_port function, after decided which port to be root port, switch will then send dest_port to connected port and inform the port role set to designated port.

Lastly, the remaining port with no role will be set to blocked port.

```
def timer(self):
    while True:
        if self.last_received is None:
            continue
        if self.root != self.switch_id:
            if time.time() - self.last_received > 10:
                print("timer expired")
                time.sleep(1)
                self.last_received = None
                if(len(self.ports) == 2):
                    for port in self.ports:
                        if self.ports[port]["role"] != "root_port":
                            self.ports[port]["role"] = "root_port"
                            self.ports[port]["stats"] = 0
                            self.socket.sendto(f"dest_port {self.switch_id}".encode(),
                                               ('localhost', 10000 + self.ports[port]["dst"]))
                        else:
                            self.ports[port]["role"] = ""
                            self.ports[port]["stats"] = ""
        time.sleep(1)
```

When switch detected timer expired, it'll changed the port state and wait for BPDU to come in and then set the port role correctly.

b. Switch set up and some functions:

```
def setup(self):
    self.socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    self.socket.bind(('localhost', self.p))
```

Switch is set up same as ospf homework, port is 10000+id.

```
elif command == "show_root":
    print(self.root)
elif command == "show_port":
    for port in self.ports:
        print(f"port {port}: {self.ports[port]}")
```

Show the current root bridge and all ports roles, stats, neighbor, cost.

```
if command == "add":
    neighbor = int(target.split(" ")[1])
    cost = int(target.split(" ")[2])
    self.neighbors[neighbor] = cost
    if self.ports is None:
        self.ports[0] = {"dst": neighbor, "cost": cost, "stats": 0}
    else:
        self.ports[len(self.ports)] = {"dst": neighbor, "cost": cost,
                                       "stats": 0, "role": ""}
    self.set = True
    for port in self.ports:
        self.ports[port]["role"] = ""
        self.ports[port]["stats"] = 0
```

Add a neighbor and the cost to it, the link should be double linked.

```
elif command == "send":
    dest = int(target.split(" ")[1])
    message = target.split(" ")[2]
    packet = f"p_send {self.switch_id} {dest} {message}"
    for port in self.ports:
        if self.ports[port]["dst"] == dest:
            if self.ports[port]["role"] == "blocked_port":
                continue
            self.socket.sendto(packet.encode(), ('localhost', 10000 + dest))
        else:
            if self.ports[port]["role"] != "blocked_port":
                self.socket.sendto(packet.encode(), ('localhost',
                                                     10000 + self.ports[port]["dst"]))
```

Send a message to other switch.

```
elif command == "rm":
    neighbor = int(target.split(" ")[1])
    if neighbor in self.neighbors:
        del self.neighbors[neighbor]
        for port in self.ports:
            if self.ports[port]["dst"] == neighbor:
                del self.ports[port]
                break
```

Remove a neighbor, to test if the timer expired correctly, this will only remove one side of two switches.

Experiment and result:

First, set the network as Fig. a, the result of root bridge and port state should be same as it.

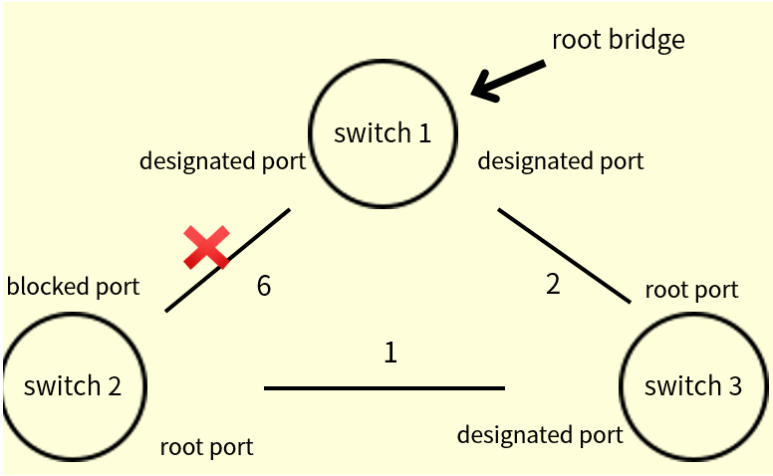

Fig. a

We can see Fig. b, Fig. c, Fig. d, respectively represent swtich 1, 2, 3, snd the results is same as Fig. a.

Next, we test if link between switch 1 and 2 is correctly disabled or not by sending a message from 1 to 2.

The message “hi” correctly send from switch 1 to 3 to 2.

```
PS C:\Users\OWNER\Desktop\nscap\hw\project> pytho
n3 stp.py 1
add 2 6
add 3 2
show_root
1
show_port
port 0: {'dst': 2, 'cost': 6, 'stats': 0, 'role':
 'designated_port'}
port 1: {'dst': 3, 'cost': 2, 'stats': 0, 'role':
 'designated_port'}
send 2 hi
```

Fig. b switch 1

```
PS C:\Users\OWNER\Desktop\nscap\hw\project> pytho
n3 stp.py 2
add 1 6
set root to 1
add 3 1
show_root
1
show_port
port 0: {'dst': 1, 'cost': 6, 'stats': 1, 'role':
 'blocked_port'}
port 1: {'dst': 3, 'cost': 1, 'stats': 0, 'role':
 'root_port'}
message from 3: hi
```

Fig. c switch 2

```
PS C:\Users\OWNER\Desktop\nscap\hw\project> pyth
on3 stp.py 3
add 1 2
set root to 1
add 2 1
show_root
1
show_port
port 0: {'dst': 1, 'cost': 2, 'stats': 0, 'role'
: 'root_port'}
port 1: {'dst': 2, 'cost': 1, 'stats': 0, 'role'
: 'designated_port'}
fowarding message to 2
```

Fig. d switch 3

Removed the link between switch 1 and switch 3, only enter command “rm” in switch 1 to test if ports role changed like Fig. e.

Fig. f remove neighbor3. Fig. g, Fig. h, results is correct.

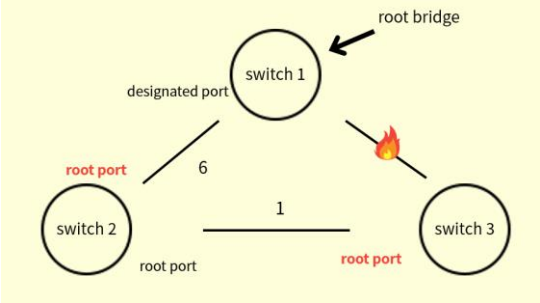

Fig. e

```
PS C:\Users\OWNER\Desktop\nscap\hw\project> pytho
n3 stp.py 1
add 2 6
add 3 2
rm 3
```

Fig. f

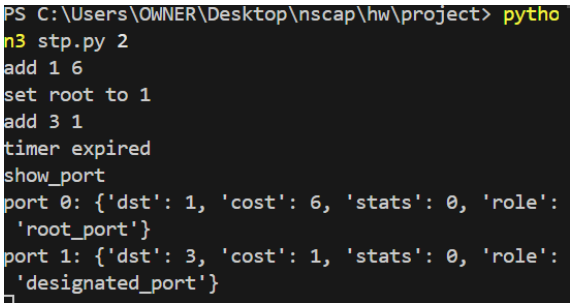

Fig. g

Fig. h

Reflection:

At first, I really can't come up the topic of what to do. After a while of thinking, I started to look at professor's power points and what had taught before. Then I thought Spanning tree Protocol was a really good topic, and started to study more about it. I had learned a lot during the process of researching and has a sense of achievement after failing a few time and finally finish my project.