Lab 1 STP Report

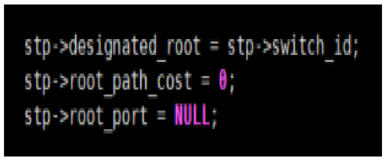
Siyuan Sheng 201828013229050

April 28th 2019

1 OVERVIEW

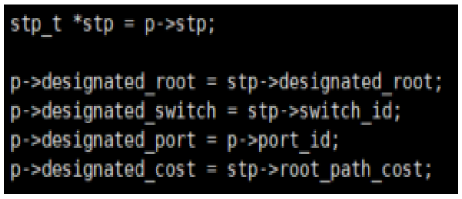
1.1 STP INIT

According to the function stp_init() in Figure 1, we set each switch as a root node first. The ports of each switch are initialized as designated ports (DPs) in Figure 2.



```
stp->designated_root = stp->switch_id;
stp->root_path_cost = 0;
stp->root_port = NULL;
```

Figure 1: Init switch as a root node



```
stp_t *stp = p->stp;

p->designated_root = stp->designated_root;
p->designated_switch = stp->switch_id;
p->designated_port = p->port_id;
p->designated_cost = stp->root_path_cost;
```

Figure 2: Init ports as DP

Besides DPs, there're root ports (RPs) and alternative ports (APs). Only DPs can send config packets through their physical links.

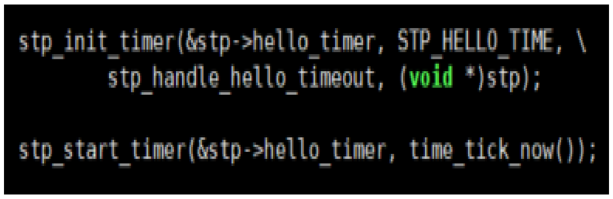
There're three situations where config sending will be triggered.

(1) For root nodes, they send config via all DPs every two seconds named hello time.

(2) For all nodes, if some port receives a config with higher priority and updates its status, it must broadcast the new config via all the DPs.

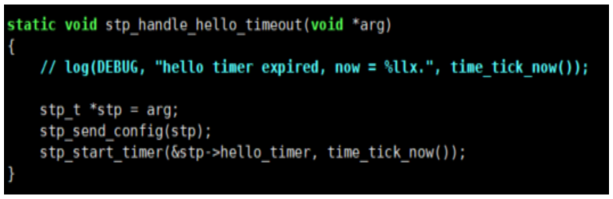
(3) For all nodes, if some port receives a config with lower priority and the port is a DP, then it only need to send its config through that port.

Figure 3 and Figure 4 are the initialization of the timer and its handler for the first situation.



```
stp_init_timer(&stp->hello_timer, STP_HELLO_TIME, \
        stp_handle_hello_timeout, (void *)stp);

stp_start_timer(&stp->hello_timer, time_tick_now());
```

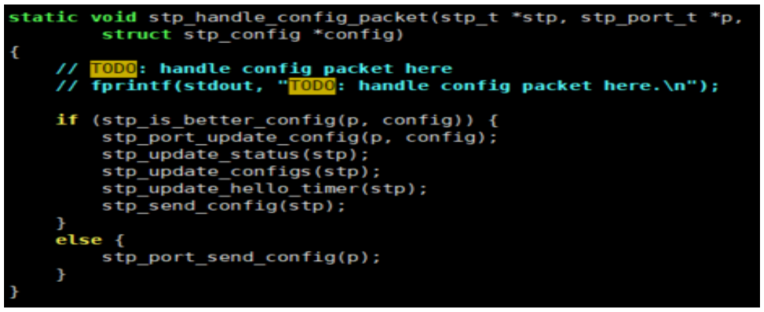
Figure 3: Init hello timer



```
static void stp_handle_hello_timeout(void *arg)
{
    // log(DEBUG, "hello timer expired, now = %llx.", time_tick_now());

    stp_t *stp = arg;
    stp_send_config(stp);
    stp_start_timer(&stp->hello_timer, time_tick_now());
}
```

Figure 4: Hello timer handler

The last two situations are resolved in the function stp_handle_config_packet() in Figure5.



```
static void stp_handle_config_packet(stp_t *stp, stp_port_t *p,
        struct stp_config *config)
{
    // TODO: handle config packet here
    // fprintf(stdout, "TODO: handle config packet here.\n");

    if (stp_is_better_config(p, config)) {
        stp_port_update_config(p, config);
        stp_update_status(stp);
        stp_update_configs(stp);
        stp_update_hello_timer(stp);
        stp_send_config(stp);
    }
    else {
        stp_port_send_config(p);
    }
}
```

Figure 5: STP handle received config

Note that we don't judge if the port is a DP while receiving a lower priority config. Actually if a port is a RP or a AP, it's impossible to receive a lower

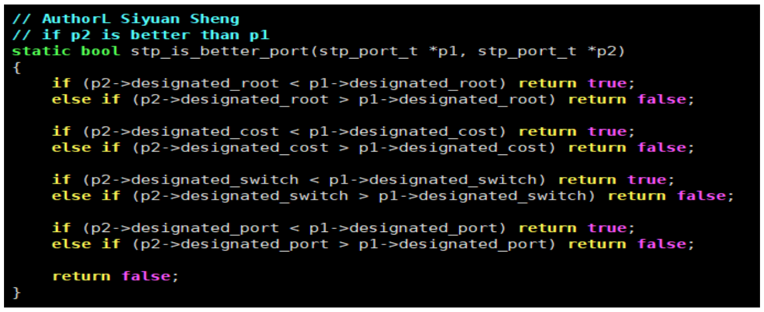

```
// AuthorL Siyuan Sheng
// if p2 is better than p1
static bool stp_is_better_port(stp_port_t *p1, stp_port_t *p2)
{
    if (p2->designated_root < p1->designated_root) return true;
    else if (p2->designated_root > p1->designated_root) return false;

    if (p2->designated_cost < p1->designated_cost) return true;
    else if (p2->designated_cost > p1->designated_cost) return false;

    if (p2->designated_switch < p1->designated_switch) return true;
    else if (p2->designated_switch > p1->designated_switch) return false;

    if (p2->designated_port < p1->designated_port) return true;
    else if (p2->designated_port > p1->designated_port) return false;

    return false;
}
```

Figure 6: Compare priority

priority config. More specifically, the priority of the received config is either identical or higher than its own config.

2 IMPLEMENTATION

Since the 1st or the 3rd situations is easy to fix, where we only need to send the config of the node via all DPs or a single specific DP.

They're just some portions of the entire process in the 2nd situation, so we take the 2nd situation as an example to show our STP handle procedure.

2.1 PRIORITY JUDGEMENT

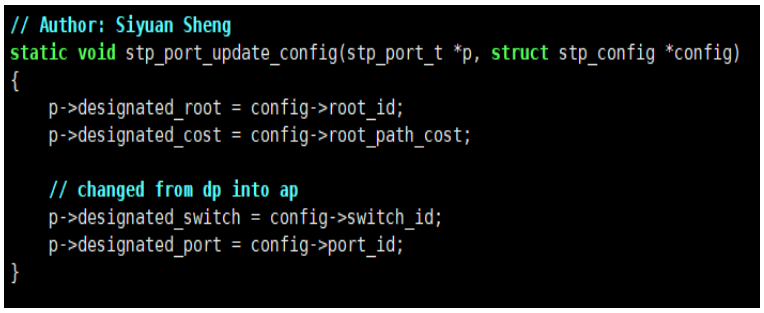
As shown in Figure 6, once we receive a config packet, we must compare its priority with that of the node itself.

These attributes are compared one by one, which are the root id, the root path cost, the switch id and the port id.

2.2 PORT CONFIG UPDATE

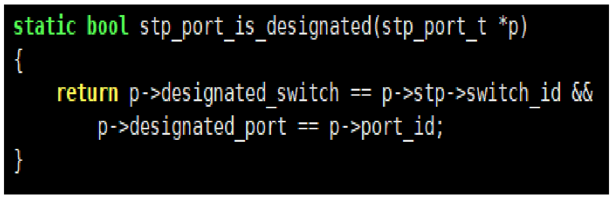
Since we take the 2nd situation as our example, the priority of incoming ocnfig is higher than the original one of the node itself. So we should udpate the config of the specific port which received the better config.

As Figure 7 shows, we copy the information of incoming config into that port directly. According to Figure 8, we can distinguish whether a port is a DP through its switch id and port id. So after updating the ports' config, its switch id and port id changes, thus it has been converted into a AP automatically.



```c
// Author: Siyuan Sheng
static void stp_port_update_config(stp_port_t *p, struct stp_config *config)
{
    p->designated_root = config->root_id;
    p->designated_cost = config->root_path_cost;

    // changed from dp into ap
    p->designated_switch = config->switch_id;
    p->designated_port = config->port_id;
}
```

Figure 7: Update port config



```c
static bool stp_port_is_designated(stp_port_t *p)
{
    return p->designated_switch == p->stp->switch_id &&
        p->designated_port == p->port_id;
}
```

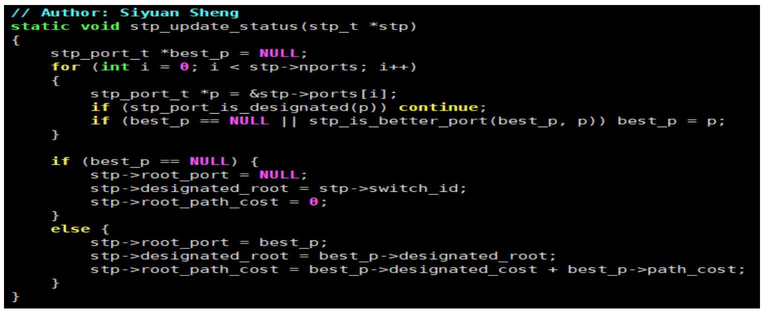
Figure 8: Judge DP

2.3 NODE STATUS UPDATE

After updating the ports' config, we need to update the status of the entire node as Figure 9.

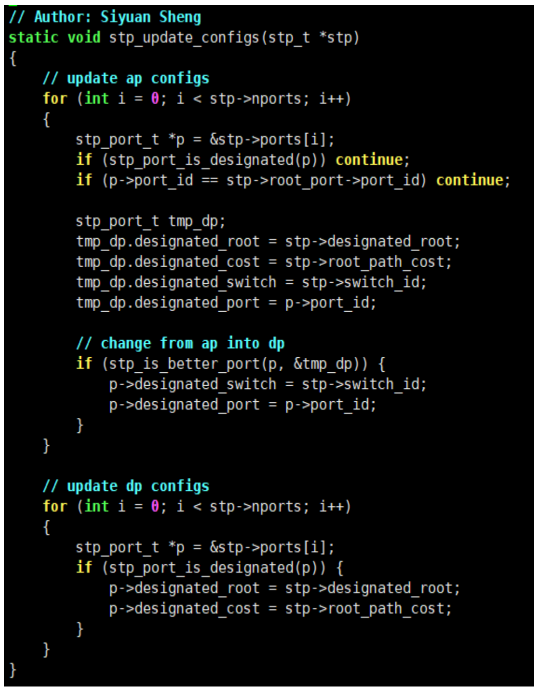
First we try to select a port with the highest priority among all APs as the root port (RP), which indicates the root path of the node in the spanning tree.

Then if RP exists, we update the nodes' config based on the RPs' config. Otherwise if we don't find RP which means the node is a root switch, we should update the nodes' config as a root node, just like what we do in the init phase.



```c
// Author: Siyuan Sheng
static void stp_update_status(stp_t *stp)
{
    stp_port_t *best_p = NULL;
    for (int i = 0; i < stp->nports; i++)
    {
        stp_port_t *p = &stp->ports[i];
        if (stp_port_is_designated(p)) continue;
        if (best_p == NULL || stp_is_better_port(best_p, p)) best_p = p;
    }

    if (best_p == NULL) {
        stp->root_port = NULL;
        stp->designated_root = stp->switch_id;
        stp->root_path_cost = 0;
    }
    else {
        stp->root_port = best_p;
        stp->designated_root = best_p->designated_root;
        stp->root_path_cost = best_p->designated_cost + best_p->path_cost;
    }
}
```

Figure 9: Update node status



```
// Author: Siyuan Sheng
static void stp_update_configs(stp_t *stp)
{
    // update ap configs
    for (int i = 0; i < stp->nports; i++)
    {
        stp_port_t *p = &stp->ports[i];
        if (stp_port_is_designated(p)) continue;
        if (p->port_id == stp->root_port->port_id) continue;

        stp_port_t tmp_dp;
        tmp_dp.designated_root = stp->designated_root;
        tmp_dp.designated_cost = stp->root_path_cost;
        tmp_dp.designated_switch = stp->switch_id;
        tmp_dp.designated_port = p->port_id;

        // change from ap into dp
        if (stp_is_better_port(p, &tmp_dp)) {
            p->designated_switch = stp->switch_id;
            p->designated_port = p->port_id;
        }
    }

    // update dp configs
    for (int i = 0; i < stp->nports; i++)
    {
        stp_port_t *p = &stp->ports[i];
        if (stp_port_is_designated(p)) {
            p->designated_root = stp->designated_root;
            p->designated_cost = stp->root_path_cost;
        }
    }
}
```

Figure 10: Update other ports' configs

2.4 OTHER PORTS CONFIG UPDATE

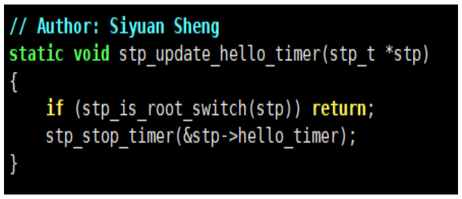
We have udpated the specific ports' config which receives the packet with the incoming config. Now we must udpate configs of all the other ports including both APs and DPs.

As Figure 10, we traverse all the APs one by one to see if we need to convert anyone into DP. Anyway, we choose some APs and reset their switch ids and port ids as DPs. The details have been demonstrated in the codes of Figure 10, here let's just skip it.

Finally, we update the configs of all DPs including the new converted ones.

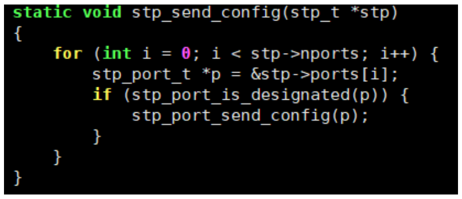
2.5 OTHER PORTS CONFIG UPDATE

This step is quite simple in Figure 11. We just reset the hello timer or stop it according to whether the node is still a root switch or not.



```
// Author: Siyuan Sheng
static void stp_update_hello_timer(stp_t *stp)
{
    if (stp_is_root_switch(stp)) return;
    stp_stop_timer(&stp->hello_timer);
}
```

Figure 11: Update hello timer



```
static void stp_send_config(stp_t *stp)
{
    for (int i = 0; i < stp->nports; i++) {
        stp_port_t *p = &stp->ports[i];
        if (stp_port_is_designated(p)) {
            stp_port_send_config(p);
        }
    }
}
```

Figure 12: Send config

2.6 SEND CONFIG

At last, we need to send configs through all the DPs of the node as Figure 12. The 1st situation where the hello timer is trigger, just need to call this step directly.

In the function stp_send_config(), it calls stp_port_send_config() for each DP.

In Figure 13, we package the config of the port into a STP packet whose format is ether_header + llc_header + stp_config (stp_header + other fields).

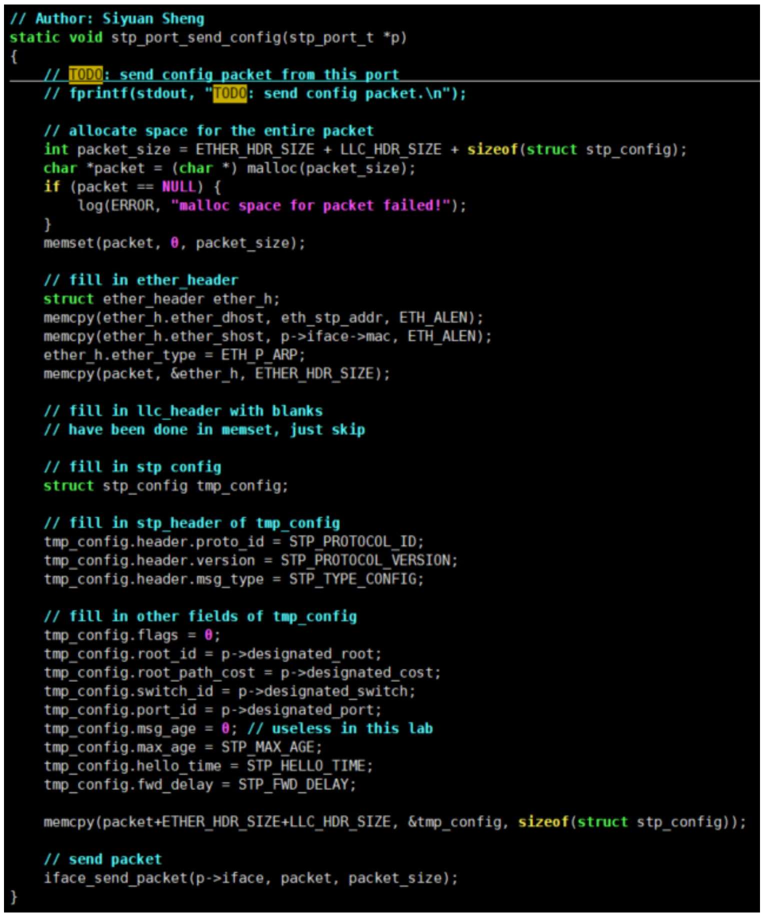
By the way, the ether_header is actually a redundant structure since we get the same information from the frame header. And the llc_header and some fields of stp_config aren't used in our simplified STP implementation.

3 EXPERIMENTS

3.1 FOUR-NODE TOPOLOGY

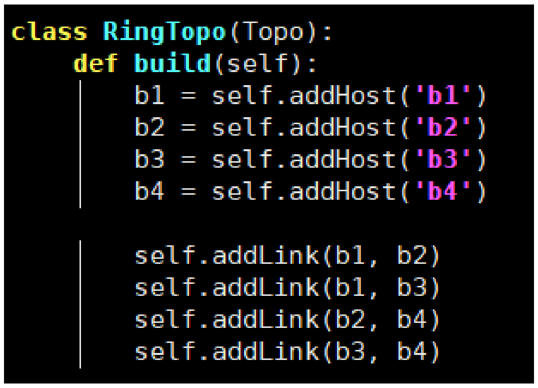
As in Figure 14, it's our four-node topology. Figure 16(a) shows our own output of this topology. Figure 16(b) shows reference output of this topology.

If you want to take a look at these outputs in detail, just open the directory under my project ./outputs/four.txt and ./outputs/four-reference.txt.



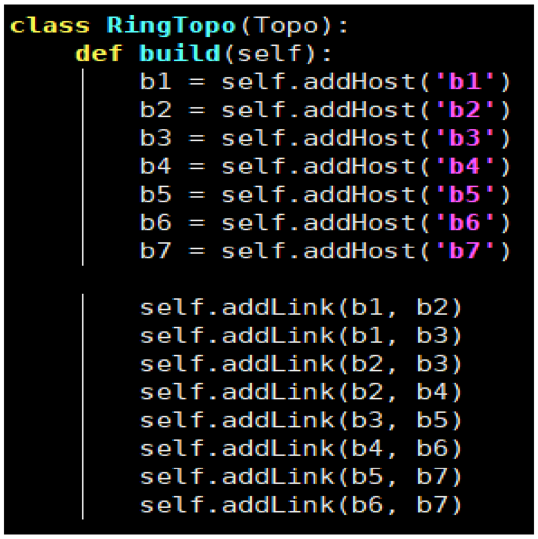
```c
// Author: Siyuan Sheng
static void stp_port_send_config(stp_port_t *p)
{
    // TODO: send config packet from this port
    // fprintf(stdout, "TODO: send config packet.\n");

    // allocate space for the entire packet
    int packet_size = ETHER_HDR_SIZE + LLC_HDR_SIZE + sizeof(struct stp_config);
    char *packet = (char *) malloc(packet_size);
    if (packet == NULL) {
        log(ERROR, "malloc space for packet failed!");
    }
    memset(packet, 0, packet_size);

    // fill in ether_header
    struct ether_header ether_h;
    memcpy(ether_h.ether_dhost, eth_stp_addr, ETH_ALEN);
    memcpy(ether_h.ether_shost, p->iface->mac, ETH_ALEN);
    ether_h.ether_type = ETH_P_ARP;
    memcpy(packet, &ether_h, ETHER_HDR_SIZE);

    // fill in llc_header with blanks
    // have been done in memset, just skip

    // fill in stp config
    struct stp_config tmp_config;

    // fill in stp_header of tmp_config
    tmp_config.header.proto_id = STP_PROTOCOL_ID;
    tmp_config.header.version = STP_PROTOCOL_VERSION;
    tmp_config.header.msg_type = STP_TYPE_CONFIG;

    // fill in other fields of tmp_config
    tmp_config.flags = 0;
    tmp_config.root_id = p->designated_root;
    tmp_config.root_path_cost = p->designated_cost;
    tmp_config.switch_id = p->designated_switch;
    tmp_config.port_id = p->designated_port;
    tmp_config.msg_age = 0; // useless in this lab
    tmp_config.max_age = STP_MAX_AGE;
    tmp_config.hello_time = STP_HELLO_TIME;
    tmp_config.fwd_delay = STP_FWD_DELAY;

    memcpy(packet+ETHER_HDR_SIZE+LLC_HDR_SIZE, &tmp_config, sizeof(struct stp_config));

    // send packet
    iface_send_packet(p->iface, packet, packet_size);
}
```

Figure 13: Port send config



```python
class RingTopo(Topo):
    def build(self):
        b1 = self.addHost('b1')
        b2 = self.addHost('b2')
        b3 = self.addHost('b3')
        b4 = self.addHost('b4')

        self.addLink(b1, b2)
        self.addLink(b1, b3)
        self.addLink(b2, b4)
        self.addLink(b3, b4)
```

Figure 14: Four node topology



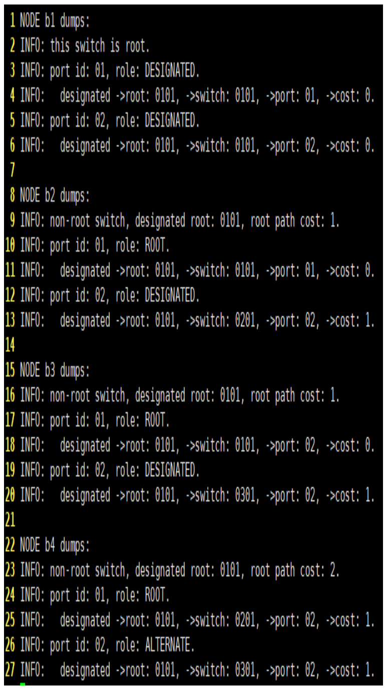
```python
class RingTopo(Topo):
    def build(self):
        b1 = self.addHost('b1')
        b2 = self.addHost('b2')
        b3 = self.addHost('b3')
        b4 = self.addHost('b4')
        b5 = self.addHost('b5')
        b6 = self.addHost('b6')
        b7 = self.addHost('b7')

        self.addLink(b1, b2)
        self.addLink(b1, b3)
        self.addLink(b2, b3)
        self.addLink(b2, b4)
        self.addLink(b3, b5)
        self.addLink(b4, b6)
        self.addLink(b5, b7)
        self.addLink(b6, b7)
```

Figure 15: Seven node topology

3.2 SEVEN-NODE TOPOLOGY

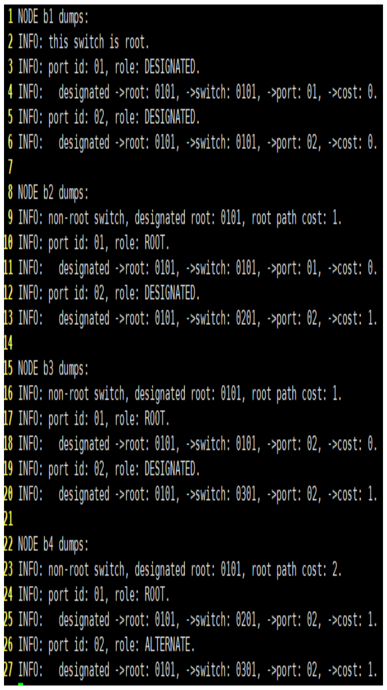
As in Figure 15, it's our seven-node topology. We won't show the figures of our own output and the reference output because of the space limit.

If you want to take a look at these outputs in detail, just open the directory under my project ./outputs/seven.txt and ./outputs/seven-reference.txt.



```
NODE b1 dumps:
INFO: this switch is root.
INFO: port id: 01, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0101, ->port: 01, ->cost: 0.
INFO: port id: 02, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0101, ->port: 02, ->cost: 0.

NODE b2 dumps:
INFO: non-root switch, designated root: 0101, root path cost: 1.
INFO: port id: 01, role: ROOT.
INFO:   designated ->root: 0101, ->switch: 0101, ->port: 01, ->cost: 0.
INFO: port id: 02, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0201, ->port: 02, ->cost: 1.

NODE b3 dumps:
INFO: non-root switch, designated root: 0101, root path cost: 1.
INFO: port id: 01, role: ROOT.
INFO:   designated ->root: 0101, ->switch: 0101, ->port: 02, ->cost: 0.
INFO: port id: 02, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0301, ->port: 02, ->cost: 1.

NODE b4 dumps:
INFO: non-root switch, designated root: 0101, root path cost: 2.
INFO: port id: 01, role: ROOT.
INFO:   designated ->root: 0101, ->switch: 0201, ->port: 02, ->cost: 1.
INFO: port id: 02, role: ALTERNATE.
INFO:   designated ->root: 0101, ->switch: 0301, ->port: 02, ->cost: 1.
```

(a) Our own output



```
NODE b1 dumps:
INFO: this switch is root.
INFO: port id: 01, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0101, ->port: 01, ->cost: 0.
INFO: port id: 02, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0101, ->port: 02, ->cost: 0.

NODE b2 dumps:
INFO: non-root switch, designated root: 0101, root path cost: 1.
INFO: port id: 01, role: ROOT.
INFO:   designated ->root: 0101, ->switch: 0101, ->port: 01, ->cost: 0.
INFO: port id: 02, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0201, ->port: 02, ->cost: 1.

NODE b3 dumps:
INFO: non-root switch, designated root: 0101, root path cost: 1.
INFO: port id: 01, role: ROOT.
INFO:   designated ->root: 0101, ->switch: 0101, ->port: 02, ->cost: 0.
INFO: port id: 02, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0301, ->port: 02, ->cost: 1.

NODE b4 dumps:
INFO: non-root switch, designated root: 0101, root path cost: 2.
INFO: port id: 01, role: ROOT.
INFO:   designated ->root: 0101, ->switch: 0201, ->port: 02, ->cost: 1.
INFO: port id: 02, role: ALTERNATE.
INFO:   designated ->root: 0101, ->switch: 0301, ->port: 02, ->cost: 1.
```

(b) Reference output

Figure 16: Compare outputs of four-node topology