生成树机制实验报告

热伊莱·图尔贡　2018K8009929030

一、 实验题目：

生成树机制实验

二、 实验内容：

1. 基于已有代码，实现生成树运行机制，对于给定拓扑(four_node_ring.py)，计算输出相应状态下的最小生成树拓扑

2. 自己构造一个不少于 7 个节点，冗余链路不少于 2 条的拓扑，节点和端口的命名规则可参考 four_node_ring.py，使用 stp 程序计算输出最小生成树拓扑

三、 实验流程：

以 four_node_ring.py 拓扑为例

1. 运行 four_node_ring.py 拓扑，4 个节点分别运行 stp 程序，将输出重定向到 b*-output.txt 文件，以 b1 为例：

```
b1# ./stp > b1-output.txt 2>&1
```

2. 等待一段时间(4 个节点大概 30 秒钟)后，执行如下命令：

```
(b?/root)# pkill -SIGTERM stp
```

该命令强制所有 stp 程序输出最终状态并退出

可以在 xterm 或 gnome-terminal 中执行该命令，需要 root 权限

3. 执行 dump_output.sh 脚本，输出个 4 个节点的状态

```
# ./dump_output.sh 4
```

四、 实验思路及结果：

1. 对于给定拓扑(four_node_ring.py)，计算输出相应状态下的最小生成树拓扑

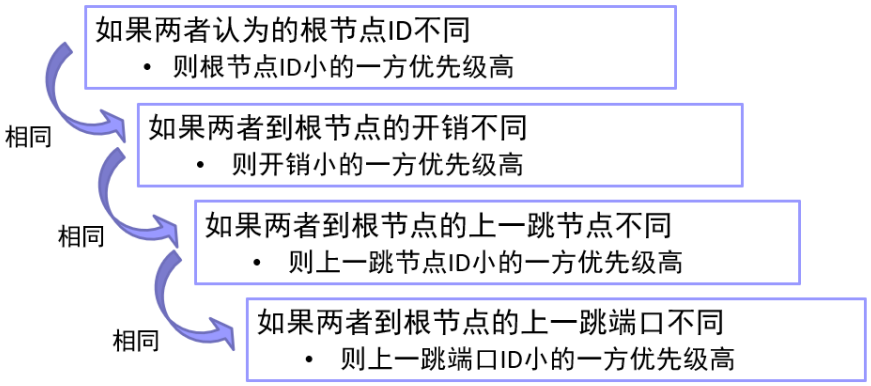
i. 实验思路

✧ 实现 stp 端口处理 config 消息，补全 stp.c 中的 stp_handle_config_packet()函数

```
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 (config_priority_cmp(p, config))
        {
                replace_port_config(p, config);
                update_stp_status(stp);
                update_port_config(stp);
                if (!stp_is_root_switch(stp))
                        stp_stop_timer(&stp->hello_timer);
                stp_send_config(stp);
        }
        else
                stp_port_send_config(p);
}
```

✧ 根据下图中的逻辑，收到 Config 消息后，将其与本端口 Config 进行优先级比较

(②) Config之间的优先级比较



- 什么时候进行Config优先级比较？
  - 端口收到的Config消息之后（端口Config与收到Config消息的比较）
  - 节点更新状态，从所有非指定端口中选取根端口时（端口间的比较）

```
// Config优先级比较
int config_priority_cmp(stp_port_t *p, struct stp_config *config)
{
        int priority = 0;
        if (p->designated_root != ntohll(config->root_id))
                priority = (p->designated_root > ntohll(config->root_id)) ? 1 : 0;
        else if (p->designated_cost != ntohl(config->root_path_cost))
                priority = (p->designated_cost > ntohl(config->root_path_cost)) ? 1 : 0;
        else if (p->designated_switch != ntohll(config->switch_id))
                priority = (p->designated_switch > ntohll(config->switch_id)) ? 1 : 0;
        else if (p->designated_port != ntohs(config->port_id))
                priority = (p->designated_port > ntohs(config->port_id)) ? 1 : 0;
        return priority;
}
```

✧ 如果收到的 Config 优先级高，则该网段应该通过对方端口连接根节点，首先应将本端口的 Config 替换为收到的 Config 消息，本端口为非指定端口。如果收到的 Config 优先级低，说明该网段应该通过本端口连接根节点，需要在判断得到结果后将 config 消息从本端口发送出去。

```
void replace_port_config(stp_port_t *p, struct stp_config *config)
{
        p->designated_root = ntohll(config->root_id);
        p->designated_cost = ntohl(config->root_path_cost);
        p->designated_switch = ntohll(config->switch_id);
        p->designated_port = ntohs(config->port_id);
}
```

✧ 更新节点状态

```
//更新节点状态
void update_stp_status(stp_t *stp)
{
        int find = 1;
        int cur_root_id = -1;
        for (int i = 0; i < stp->nports; i++)
        {
                if (!stp_port_is_designated(&stp->ports[i]))
                {
                        if (cur_root_id == -1)
                                cur_root_id = i;
                        else
                        {
                                if (port_priority_cmp(&stp->ports[cur_root_id], &stp->ports[i]))
                                        cur_root_id = i;
                        }
                }
                if ((i == stp->nports - 1) && (cur_root_id == -1))
                        find = 0;
        }
        if (find == 0)
        {
                stp->root_port = NULL;
                stp->designated_root = stp->switch_id;
                stp->root_path_cost = 0;
        }
        else
        {
                stp->root_port = &stp->ports[cur_root_id];
                stp->designated_root = stp->root_port->designated_root;
                stp->root_path_cost = stp->root_port->designated_cost + stp->root_port->path_cost;
        }
}
```

✧ 端口优先级比较

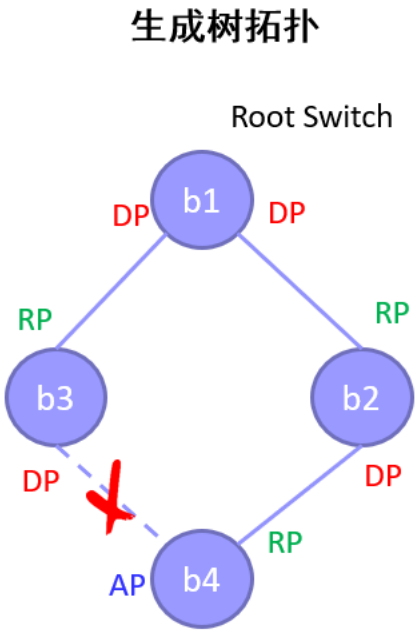
```
//端口之间的优先级
int port_priority_cmp(stp_port_t *p, stp_port_t *q)
{
        int priority = 0;
        if (p->designated_root != q->designated_root)
                priority = (p->designated_root > q->designated_root) ? 1 : 0;
        else if (p->designated_cost != q->designated_cost)
                priority = (p->designated_cost > q->designated_cost) ? 1 : 0;
        else if (p->designated_switch != q->designated_switch)
                priority = (p->designated_switch > q->designated_switch) ? 1 : 0;
        else if (p->designated_port != q->designated_port)
                priority = (p->designated_port > q->designated_port) ? 1 : 0;

        return priority;
}
```

✧ 更新剩余端口的 Config

```c
//更新剩余端口的Config
void update_port_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))
                {
                        p->designated_root = stp->designated_root;
                        p->designated_cost = stp->root_path_cost;
                }
                else if (!stp_port_is_root(stp, p) && DP_all_priority_cmp(p, stp))
                {
                        p->designated_switch = stp->switch_id;
                        p->designated_port = p->port_id;
                }
        }
}
```

ii. 实验结果

```
rayllam@rayllam-VirtualBox:~/CN/04-stp$ ./dump_output.sh 4
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.
```



2. 自己构造一个不少于 7 个节点，冗余链路不少于 2 条的拓扑，使用 stp 程序计算输出最小生成树拓扑

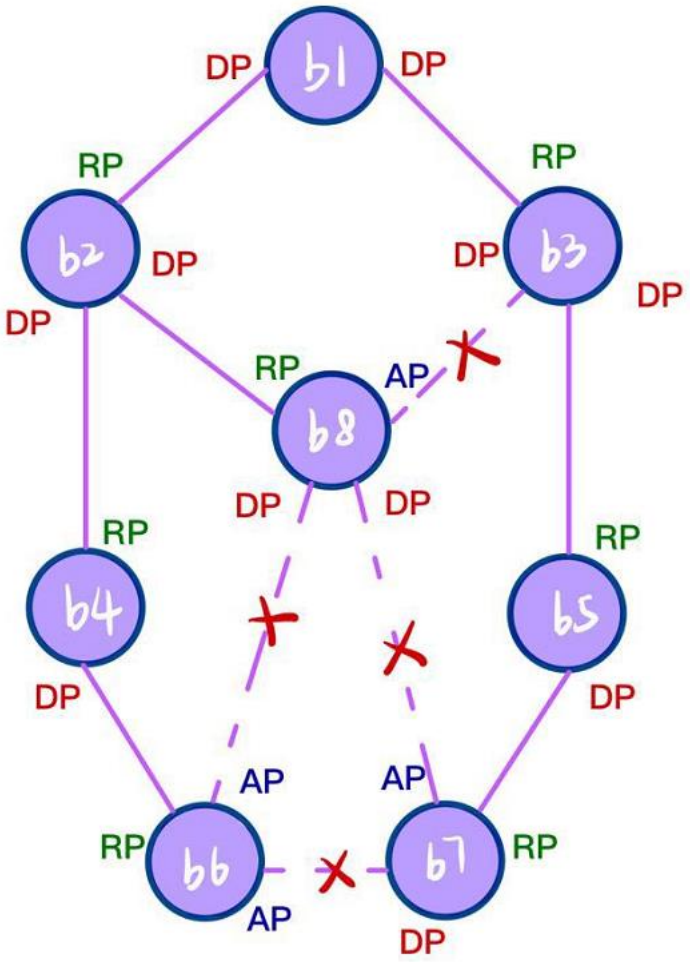
i. 构造 8 个节点，4 条冗余边的环形拓扑

```
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')
        b8 = self.addHost('b8')

        self.addLink(b1, b2)
        self.addLink(b1, b3)
        self.addLink(b2, b4)
        self.addLink(b5, b3)
        self.addLink(b2, b8)
        self.addLink(b3, b8)
        self.addLink(b8, b7)
        self.addLink(b8, b6)
        self.addLink(b5, b7)
        self.addLink(b6, b7)
        self.addLink(b6, b4)
```

ii. 实验结果

```
rayilam@rayilam-VirtualBox:~/CN/04-stp$ ./dump_output.sh 8
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.
INFO: port id: 03, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0201, ->port: 03, ->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.
INFO: port id: 03, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0301, ->port: 03, ->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: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0401, ->port: 02, ->cost: 2.
```

```
NODE b5 dumps:
INFO: non-root switch, designated root: 0101, root path cost: 2.
INFO: port id: 01, role: ROOT.
INFO:   designated ->root: 0101, ->switch: 0301, ->port: 03, ->cost: 1.
INFO: port id: 02, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0501, ->port: 02, ->cost: 2.

NODE b6 dumps:
INFO: non-root switch, designated root: 0101, root path cost: 3.
INFO: port id: 01, role: ROOT.
INFO:   designated ->root: 0101, ->switch: 0401, ->port: 02, ->cost: 2.
INFO: port id: 02, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0603, ->port: 02, ->cost: 3.
INFO: port id: 03, role: ALTERNATE.
INFO:   designated ->root: 0101, ->switch: 0801, ->port: 03, ->cost: 2.

NODE b7 dumps:
INFO: non-root switch, designated root: 0101, root path cost: 3.
INFO: port id: 01, role: ROOT.
INFO:   designated ->root: 0101, ->switch: 0501, ->port: 02, ->cost: 2.
INFO: port id: 02, role: ALTERNATE.
INFO:   designated ->root: 0101, ->switch: 0603, ->port: 02, ->cost: 3.
INFO: port id: 03, role: ALTERNATE.
INFO:   designated ->root: 0101, ->switch: 0801, ->port: 04, ->cost: 2.

NODE b8 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: 03, ->cost: 1.
INFO: port id: 02, role: ALTERNATE.
INFO:   designated ->root: 0101, ->switch: 0301, ->port: 02, ->cost: 1.
INFO: port id: 03, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0801, ->port: 03, ->cost: 2.
INFO: port id: 04, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0801, ->port: 04, ->cost: 2.
```



五、 调研思考题：

1. 调研说明标准生成树协议中，如何处理网络拓扑变动的情况：当节点加入时？当节点离开时？

在交换网络中，交换机依赖 MAC 地址表转发数据帧。在节点加入或离开拓扑结构时，都将先重新进行 STP 计算，更新各节点状态与端口的配置信息，产生新的树形拓扑结构。 TCN BPDU 用于处理拓扑变动。Flags 由 8 位组成，最低位为 TC 标志位，最高位为 TCA 标志位，其他 6 位保留。

在网络拓扑发生变化后，有端口转为转发状态的下游设备会不间

断地向上游设备发送 TCN BPDU 报文；上游交换机收到 TCN BPDU 报文，会将下一个配置 BPDU 报文中的 TCA 置为 1，发送给下游交换机。其它端口也有可能收到 TCN BPDU 报文，但不会处理；上游设备复制一份 TCN BPDU 报文，向根桥方向发送。

重复步骤以上，直到根桥收到 TCN BPDU 报文。

根桥收到 TCN BPDU 后，会将下一个配置 BPDU 报文中的 TCA 置为 1，发送给下游所有的交换机。

根桥在之后的 max age+forwarding delay 时间内，将发送 BPDU 中的 TC 置位的报文，收到该配置 BDPU 的网桥，会将自身 MAC 地址老化时间缩短为 forwarding delay。

1. 调研说明标准生成树协议是如何在构建生成树过程中保持网络连通的

   提示：用不同的状态来标记每个端口，不同状态下允许不同的功能（Blocking，Listening，Learning，Forwarding 等）

在构建生成树的过程中，STP 会将部分冗余链路强制转化为阻塞状态，其余链路处于转发状态。当处于转发状态的链路不可用时，STP 可以重新配置网络，恢复部分冗余链路的转发状态来确保网络的连通性。

2. 实验中的生成树机制效率较低，调研说明快速生成树

机制的原理

快速生成树协议RSTP(Rapid Spanning Tree Protocol)在STP基础上实现了快速收敛，并增加了边缘端口的概念及保护功能。快速生成树协议，在网络拓扑发生改变的时候，能够显著的提升生成树重新计算的速度。RSTP定义了两种端口角色：替代端口、备份端口。还定义了三种端口状态：丢弃状态（Discarding)、学习状态（Learning)、转发状态（Forwarding)。

✧ 配置BPDU的处理发生变化

RSTP 在拓扑稳定后，无论非根桥设备是否接收到桥传来的配置BPDU报文，非根桥设备仍然按照Hello Timer规定的时间间隔发送配置BPDU，该行为完全由每台设备自主进行。

✧ 更短的BPDU超时计时

如果一个端口连续3个Hello Time时间内没有收到上游设备发送过来的配置BPDU,那么该设备认为与此邻居之间的协商失败。而不像STP那样需要先等待一个Max age。

✧ 处理次优BPDU

当一个端口收到上游的指定桥发来的RST BPDU报文时，该端口会将自身储存的RST BPDU与收到的RST BPDU进行比较。如果该端口储存的RST BPDU的优先级高于收到的RST BPDU，那么该端口会直接丢弃收到的RST BPDU,立即回应自身储存的RST BPDU。当上游设备收到下游设备回应的RST BPDU后，上游设备会根据收到的 RST BPDU 报文中相应的字段立即更新自己储存的 RST

BPDU。由此，RSTP 处理次等 BPDU 报文不再依赖于任何定时器通过超时解决拓扑收敛，从而加快了拓扑收敛。

✧ 快速收敛

Proposal/Agreement 机制

在 RSTP 中，当一个端口被选举成为指定端口之后，此端口会先进入 DIscarding 状态，再通过 Proposal/Agreement 机制快速进入 Forward 状态。这种机制必须在点到点全双工链路上使用。[1]

[1] https://blog.csdn.net/Summnus_l/article/details/82817383