生成树机制实验

学号： 2021K8009929010

姓名： 贾城昊

一、 实验题目：生成树机制实验

二、 实验任务：

了解生成树的拓扑结构和生成树机制的基本原理,了解如何在网络中实现唯一的、基于优先级的生成树。具体实现基于 Config 消息的生成树算法,掌握处理 Config 消息的流程。在给定拓扑以及自己构造的更复杂拓扑下验证生成树算法功能。最后,调研了解标准生成树协议中如何处理动态拓扑、如何在构建生成树过程中保持网络连通、如何快速构建生成树等原理。

三、 实验流程

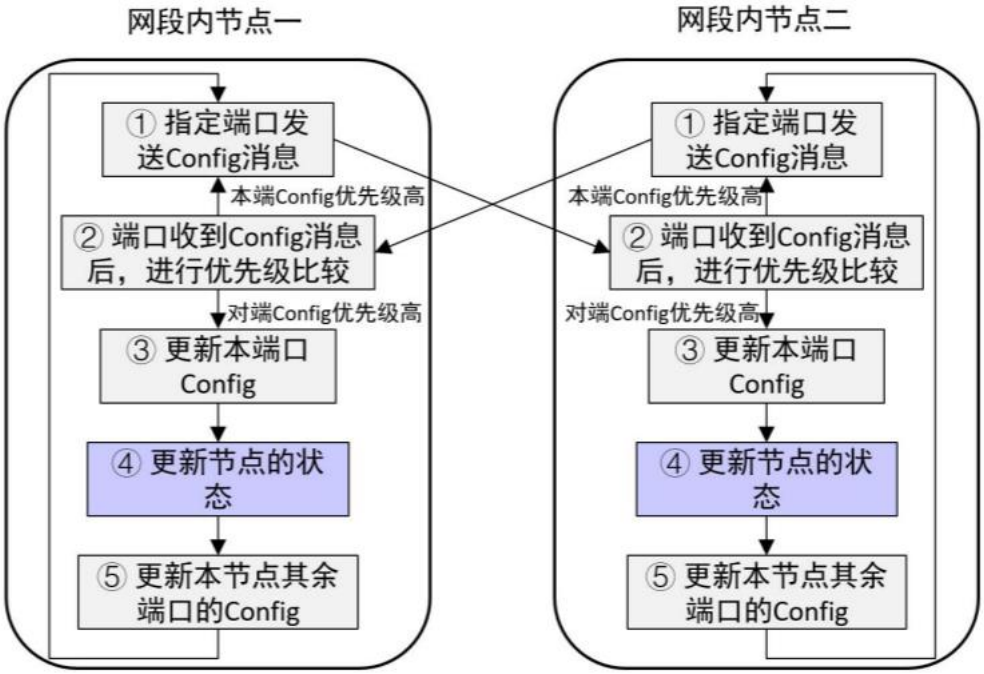
1. 基于已有代码，实现生成树运行机制，对于给定拓扑(four_node_ring.py)，计算输出相应状态下的最小生成树拓扑。
2. 自己构造一个不少于 7 个节点，冗余链路不少于 2 条的拓扑，节点和端口的命名规则可参考 four_node_ring. py，使用 stp 程序计算输出最小生成树拓扑。

四、 实验结果与分析

（一）实现生成树运行机制

1. 总体逻辑

总的流程如下图的流程图所示：



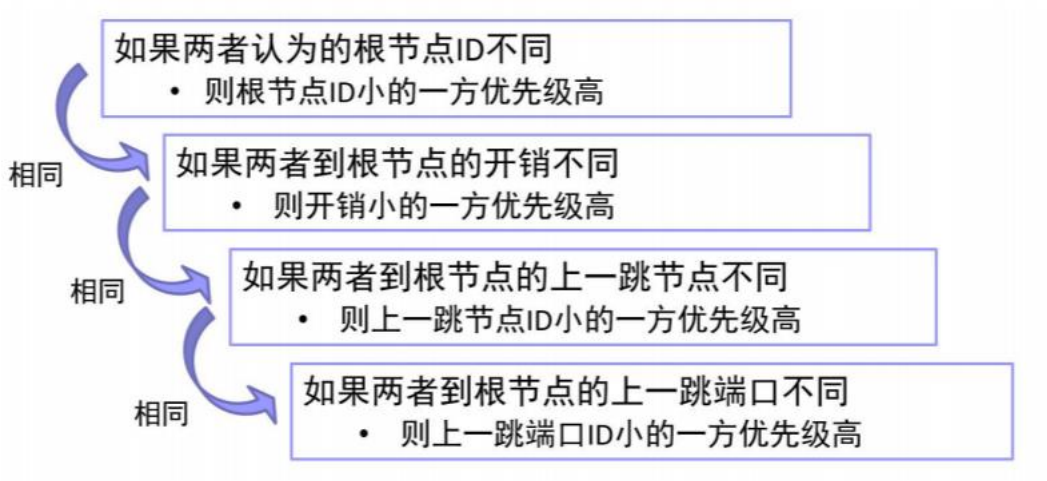
首先我们需要判断收到的 config 消息和本端口的 config 消息哪个优先级高，如果本端口优先级高,则说明该网段应该通过本端口存储 config 对应的端口连接根节点,在代码中体现就是不需要做额外的事情。如果本端口的 config 消息没有收到的 config 消息高，则需要重新选举根端口，并更新本节点的配置信息，然后更新本节点其它端口的配置。如果更新后自己不再是根节点则停止计时器。最后，将自己更新过后的配置信息通过所有的指定端口转发出去。

下面将进行具体的介绍。

2. 比较 config 的优先级

首先我们需要实现 config 之间的优先级的比较，这里有两种比较，第一种是比较两个 port 的 config 的优先级，第二个是比较收到的 config 和端口 config 的优先级，两者

比较逻辑大同小异，其逻辑均如下所示：



在本次实验中，本人针对两者的共同性，只写了一个函数进行比较，若是进行端口接收的 config 和端口的 config 比较的时候，只需要事先对接收端口的相应字段进行字节序转化即可，具体函数的代码如下：

```
static bool config_compare(stp_port_t *p,
                u64 designed_root, u32 root_path_cost,
                u64 switch_id, u16 port_id)
{
    if(p->designated_root < designed_root){
        return true;
    }
    else if(p->designated_root > designed_root){
        return false;
    }
    else{
        if(p->designated_cost < root_path_cost){
            return true;
        }
        else if(p->designated_cost > root_path_cost){
            return false;
        }
        else{
            if(p->designated_switch < switch_id){
                return true;
            }
            else if(p->designated_switch > switch_id){
                return false;
            }
            else{
                if(p->designated_port < port_id){
                    return true;
                }
                else if(p->designated_port > port_id){
                    return false;
                }
                else{
                    return false;
                }
            }
        }
    }
}
```

3. 更新端口的 congfig

于是在接收到 Config 消息时,先进行比较。如果收到的 Config 优先级更高,就把本端口的 Config 替换为收到的 Config 消息,之后再进行后续处理。如果本地 Config 优先级更高,不做更改,若该端口是指定端口,发送 Config 消息，如下所示:

```
if(config_superior){
    // p is the designated port, send config
    if (stp_port_is_designated(p)) {
        stp_port_send_config(p);
    }
}
else{
    //p's config is replaced by opposite config
    //p will not be designated port in this time
    p->designated_root = designed_root;
    p->designated_cost = root_path_cost;
    p->designated_switch = switch_id;
    p->designated_port = port_id;
```

4. 更新节点状态

首先遍历所有端口，尝试找到根端口。如果存在根端口，则该节点为非根节点，选择通过 root_port 连接到根节点，更新节点认定的根、到根节点的路径开销、根端口。如果不存在根端口,则该节点为根节点。根端口为空，根节点指向自己，距离为 0。

根端口需要满足两个条件：第一是该端口为非指定端口，第二为该端口的优先级要高于其它非指定端口。具体我们的实现细节是，遍历所有端口找到所有非指定端口，并选择其中优先级最高的一个。

寻找根端口具体的代码逻辑如下图所示：

```c
//find root port
//1.it is a non-designated port
//2.it the most superior non-designated port
//search cost O(n)
static stp_port_t *find_root_port(stp_t *stp)
{
    stp_port_t *root_port = NULL;
    stp_port_t *port_entry;

    for (int i = 0; i < stp->nports; i++) {
        port_entry = &stp->ports[i];

        if (!stp_port_is_designated(port_entry)) {
            if(root_port){
                if(config_compare(port_entry, root_port->designated_root, root_port->designated_cost,
                        root_port->designated_switch, root_port->port_id)){
                    root_port = port_entry;
                }
            }
            else{
                root_port = port_entry;
            }
        }
    }

    return root_port;
}
```

更新节点状态的具体代码逻辑如下所示：

```c
//update stp state
stp_port_t *root_port = find_root_port(stp);

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

5. 更新剩余端口的 config

该部分只用处理非指定端口变为指定端口和指定端口变为指定端口的情况。

对于非指定端口，如果其 Config 较网段内其他端口优先级更高，那么该端口成为指定端口。具体来说更新前该非指定端口的 Config 是网段内最高优先级 Config，如果节点 Config 信息继承到该端口后的优先级高于更新前，那么就把该端口更新为指定端口，如下所示：

```
//find all non-designated -> designated
for (int i = 0; i < stp->nports; i++) {
    stp_port_t* port_entry = &stp->ports[i];
    if (!stp_port_is_designated(port_entry)) {
        if(root_port){
            if(!config_compare(port_entry, stp->designated_root,
                    stp->root_path_cost, stp->switch_id, port_entry->port_id)){
                port_entry->designated_switch = stp->switch_id;
                port_entry->designated_port = port_entry->port_id;
            }
        }
    }
}
```

对于所有指定端口，需要更新其认为的根节点和路径开销，如下所示：

```
//update designed ports'config of designated_root & designated_cost
for (int i = 0; i < stp->nports; i++) {
    stp_port_t* port_entry = &stp->ports[i];
    if (stp_port_is_designated(port_entry)) {
        port_entry->designated_root = stp->designated_root;
        port_entry->designated_cost = stp->root_path_cost;
    }
}
```

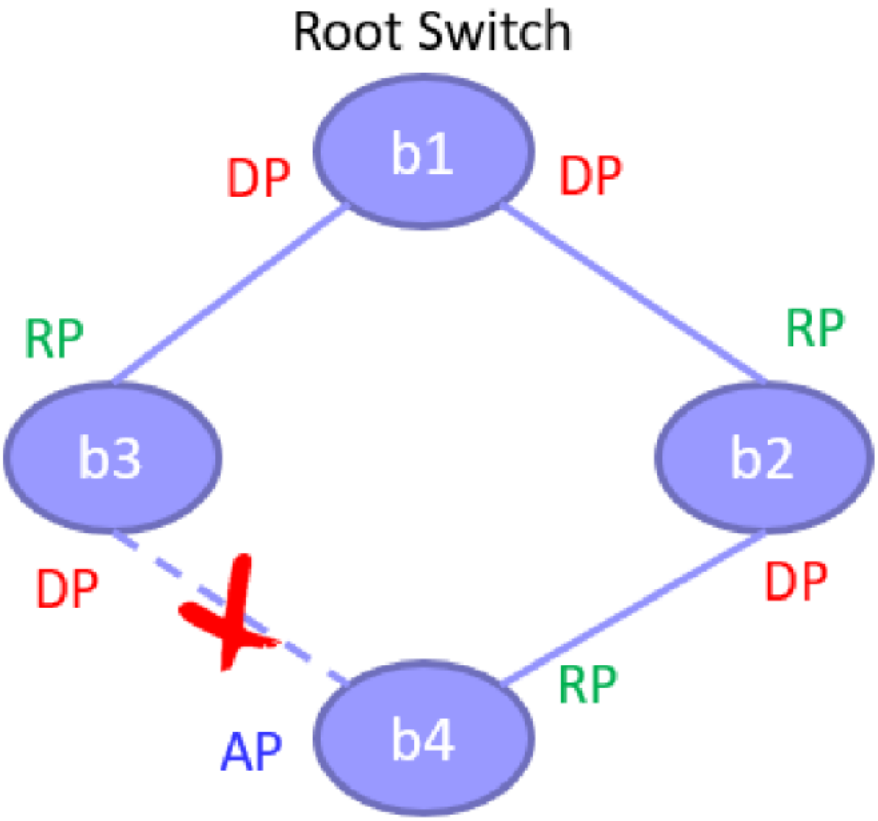
6. 停止 hello 计时器与转发 config

最后需要判断节点性质是否发生变化，如果节点由根节点变为非根节点，停止 hello 定时器，并将更新后的 Config 从每个指定端口转发出去，如下所示：

```
//only root switch can always send config
if (is_root_before && !stp_is_root_switch(stp)){
    stp_stop_timer(&stp->hello_timer);
}

//send update config to other stps' port
stp_send_config(stp);
```

（二）4 节点拓扑的最小生成树

给定的 4 节点拓扑结构如下图所示：



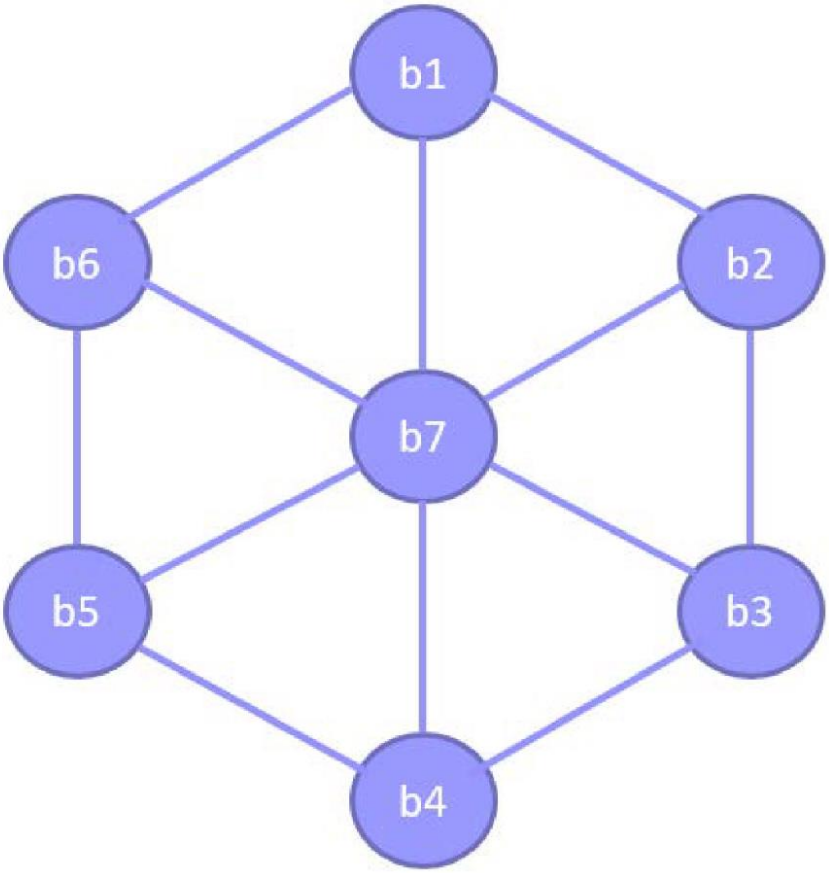
使用 stp 程序计算输出最小生成树拓扑，实验结果如下：

```
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.
```

可以看到 b1 节点为根节点,其两个端口都为指定端口。b4 节点的 2 端口为 AP 端口，不参与构建生成树拓扑其他节点和端口也都符合预期的拓扑结构。由此可以看出该生成树算法功能正确。

（三）7 节点拓扑的最小生成树

构造一个含有 7 个节点的复杂拓扑结构，如下图所示：

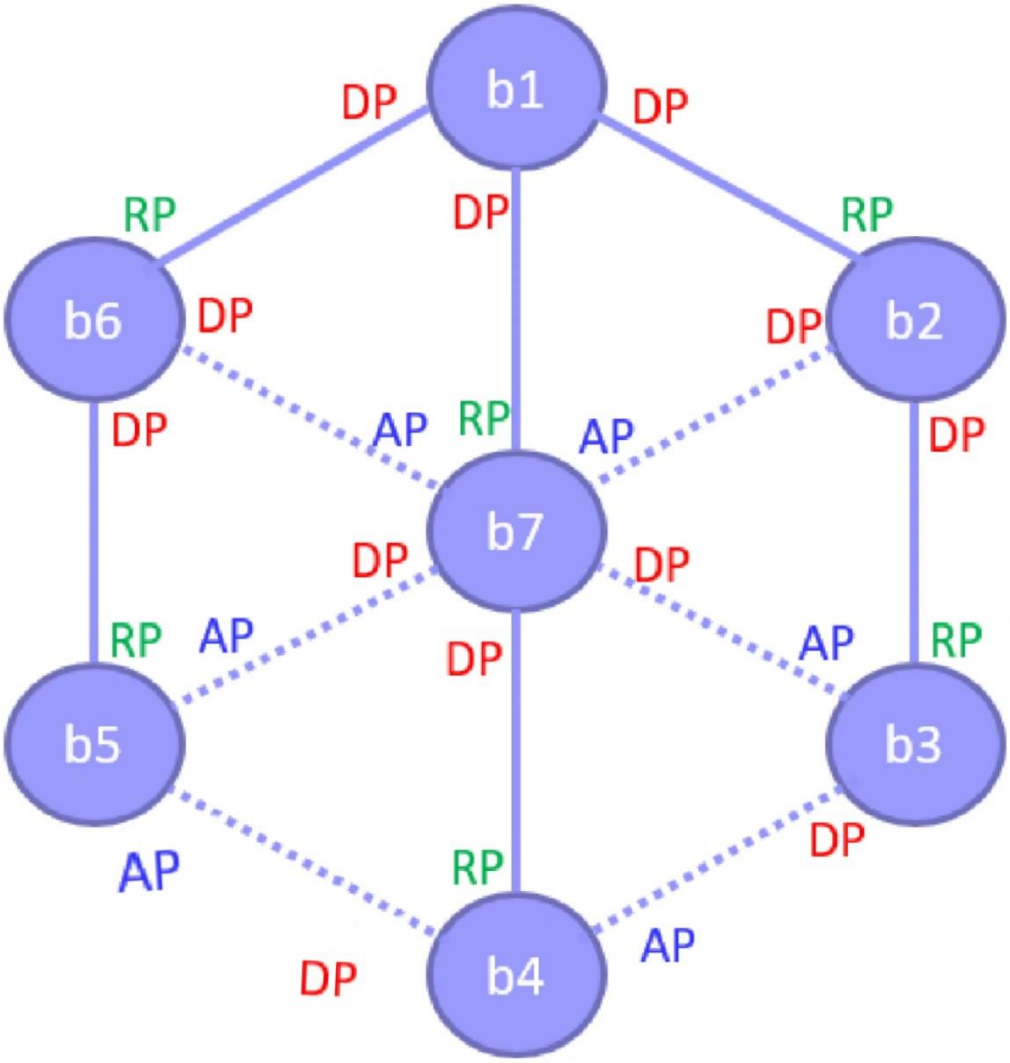


使用 stp 程序计算输出最小生成树拓扑，结果如下：

```
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.
INFO: port id: 03, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0101, ->port: 03, ->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: 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: 0301, ->port: 02, ->cost: 2.
INFO: port id: 03, role: ALTERNATE.
INFO:   designated ->root: 0101, ->switch: 0701, ->port: 03, ->cost: 1.

NODE b4 dumps:
INFO: non-root switch, designated root: 0101, root path cost: 2.
INFO: port id: 01, role: ALTERNATE.
INFO:   designated ->root: 0101, ->switch: 0301, ->port: 02, ->cost: 2.
INFO: port id: 02, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0401, ->port: 02, ->cost: 2.
INFO: port id: 03, role: ROOT.
INFO:   designated ->root: 0101, ->switch: 0701, ->port: 04, ->cost: 1.
```

```
NODE b5 dumps:
INFO: non-root switch, designated root: 0101, root path cost: 2.
INFO: port id: 01, role: ALTERNATE.
INFO:   designated ->root: 0101, ->switch: 0401, ->port: 02, ->cost: 2.
INFO: port id: 02, role: ROOT.
INFO:   designated ->root: 0101, ->switch: 0602, ->port: 02, ->cost: 1.
INFO: port id: 03, role: ALTERNATE.
INFO:   designated ->root: 0101, ->switch: 0701, ->port: 05, ->cost: 1.

NODE b6 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: 0602, ->port: 02, ->cost: 1.
INFO: port id: 03, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0602, ->port: 03, ->cost: 1.

NODE b7 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: 03, ->cost: 0.
INFO: port id: 02, role: ALTERNATE.
INFO:   designated ->root: 0101, ->switch: 0201, ->port: 03, ->cost: 1.
INFO: port id: 03, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0701, ->port: 03, ->cost: 1.
INFO: port id: 04, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0701, ->port: 04, ->cost: 1.
INFO: port id: 05, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0701, ->port: 05, ->cost: 1.
INFO: port id: 06, role: ALTERNATE.
INFO:   designated ->root: 0101, ->switch: 0602, ->port: 03, ->cost: 1.
```

根据以上信息，画出生成树拓扑结构如下所示：



经过分析,该实验结果满足生成树要求。可见我们的生成树算法功能是正确的。

五、 思考题

1. 网络中的节点是动态的，如何在有节点加入和离开时，依然能够构建生成树拓扑？

   1) 当节点加入或者离开拓扑结构时候，生成树协议会自动重新计算生成树，更新节点状态和端口的配置信息(包括重新选举根节点，更新根端口，更新其它端口的配置等)。这是因为生成树的拓扑结构可能会受到节点加入或离开的影响，但在这之前需要通知所有网

桥拓扑结构发生了变化。

2) 当网络拓扑变动时，将会使用另一种生成树协议报文。在 STP 报文中，有两个成员：BPDU Type 与 Flags。BPDU Type 标记协议报文的类型，配置 BPDU 类型为 0x00，TCN BPDU 类型为 0x80，我们实验中用到的为配置 BPDU，它的作用为就是传递 Config 信息，建立生成树。而 TCN BPDU 则用于处理拓扑变动 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。

3) 当有端口断开或新网桥加入时,拓扑发生了改变,就会使用到 TCN BPDU 报文，目的是让 STP 能快速的收敛。拓扑改变时候 STP 处理步骤为:

I. 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从根端口向上游设备发送 TCN BPDU 报文，报文中 flags 的 TC 位置为 1。

II. 上游设备收到下游设备发来的 TCN BPDU 报文后,只有指定端口处理 TCN BPDU 报文。其它端口也有可能收到 TCN BPDU 报文，但不会处理。

III. 上游设备会将下一个配置 BPDU 报文中的 TCA 置为 1，然后发送给下游设备，告知下游设备停止发送 TCN BPDU 报文。

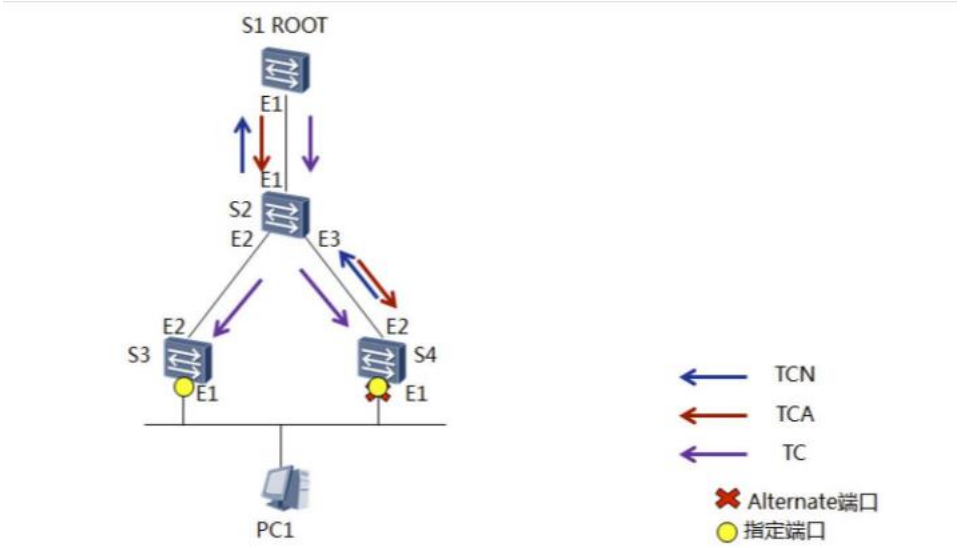
IV. 上游设备复制一份 TCN BPDU 报文，向根桥方向发送。

V. 重复步骤 1、2、3、4，直到根节点收到 TCN BPDU 报文。

VI. 根节点收到 TCN BPDU 报文后，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。作为对收到的 TCN 的确认；还会将该配置 BPDU 报文中的 Flags 的 TC 位置 1,用于通知所有网桥拓扑发生了变化。

VII. 根节点在之后的 max (age + forwardingdelay)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 forwardingdelay。

这样就能使拓扑改变后，STP 能快速重新收敛。上述处理流程也可见下图：



2. 网络中的节点还需要进行数据转发，如何设计生成树运行机制，保证与交换机数据包转发兼容？

标准生成树协议赋予端口 5 种状态

状态名称	状态描述
禁用(Disable)	不能收发 BPDU，也不能收发数据帧
阻塞（Blocking）	不能发送 BPDU，但是会持续侦听 BPDU，不能收发数据帧
侦听（Listening）	可以收发 BPDU，但不能收发数据帧，也不能进行 MAC 地址学习
学习(Learning)	会侦听业务数据帧，但不能转发数据帧，可以进行 MAC 地址学习
转发(Forwarding)	正常收发数据帧，也会进行 BPDU 处理

当交换机的一个端口被激活后，该端口会从禁用状态自动进入阻塞状态。阻塞状态的

端口如果被选举为根端口或者指定端口,那么它将从阻塞状态进入侦听状态。端口将在侦听状态停留 15s ,期间收发 BPDU,让 STP 完成整个网络的计算。在 15s 后,生成树构建完成,如果该端口还是根端口或指定端口,就会进入学习状态。这时将侦听数据帧,根据 MAC 地址学习转发表。学习状态也将停留 15s,之后端口进入转发状态。此时正常收发数据帧,也处理 BPDU 帧。

综上,标准生成树协议是通过赋予端口不同状态,让其在不同状态下分别处理构建生成树和转发数据包功能，由此保证生成树运行机制与交换机数据包转发兼容。

六、 实验总结

通过本次实验,我对生成树的拓扑结构和生成树机制的基本原理有了一定的了解。我学到了如何在网络中实现唯一的、基于优先级的生成树，掌握了处理 Config 消息的流程。在本次实验中,本人实现了生成树算法,并通过给定的 4 节点拓扑和自己构造的 7 节点复杂拓扑进行测试,验证了算法的正确性，并通过通过思考题,对标准生成树协议进行了更深入的调研，了解到了现实中生成树算法如何处理实时的拓扑变动以及如何设计生成树运行机制，保证与交换机数据包转发兼容，这让我对生成树机制有了更多的认识。