生成树机制实验报告

李昊宸

2017K8009929044

（一）生成树机制-处理 config 消息实现

一、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

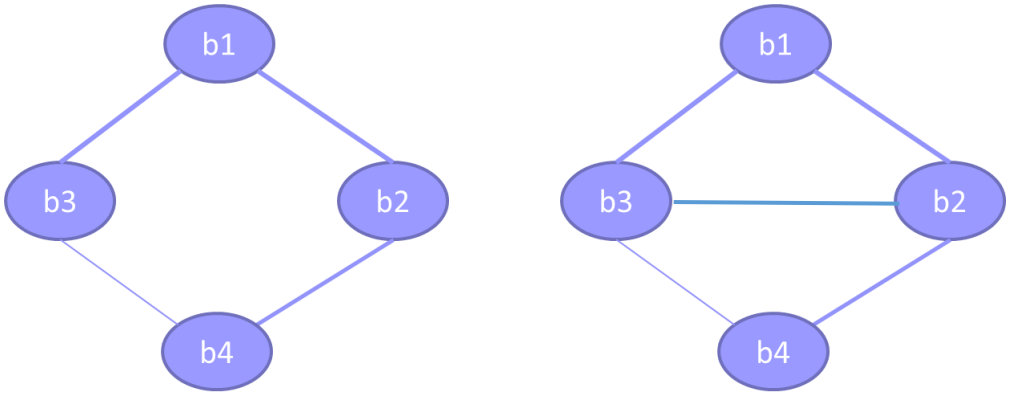
二、实验流程

1. 搭建实验环境

include：相关头文件
scripts：禁用 TCP Offloading、IPV6 功能，避免抓到无用包
main.c： Stp 的代码实现，编译后在交换机结点上运行
stp.c：所有的 STP 机制相关，本次实验要求实现函数：

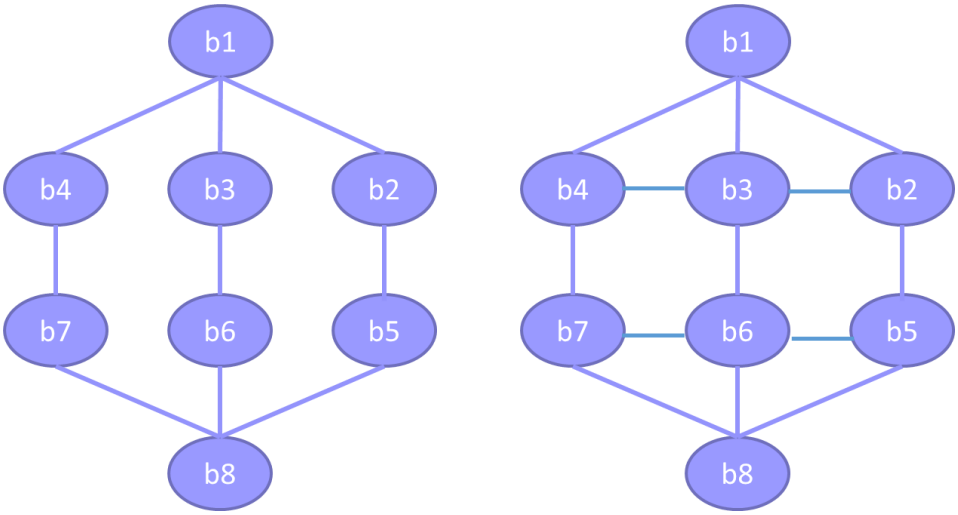
```
static void stp_handle_config_packet(stp_t *stp, stp_port_t *p,
struct stp_config *config)
```

packet.c：发包函数
stp_timer.c：定时器实现
dump_output.sh：汇总输出各节点状态信息
four_nodes_ring(four_nodes_ring _diamond).py: 实现如下左（右）图的四节点拓扑



图一 四节点网络拓扑

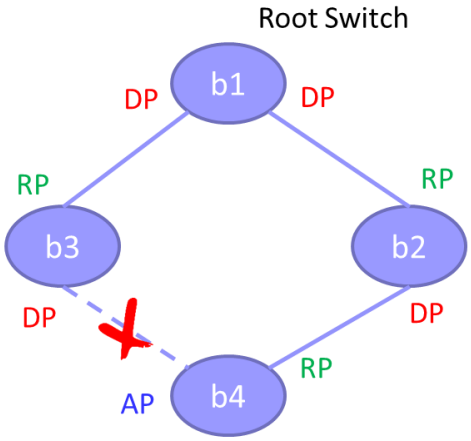
eight_node（eight_node_diamond）.py：实现如下左（右）图的八节点拓扑



图二 八节点网络拓扑

生成树协议：Spanning Tree Protocol，一种用于在网络中检测环路并逻辑地阻塞冗余路径，以确保在任意两个节点之间只存在一条路径的技术。生成树机制通过禁止网络设备的相关端口，在有环路的网络中构造出一个总体开销最小的树状拓扑，使得网络在连通的前提下，避免广播风暴。

当网络中出现环路时，该协议可以采用生成树的算法从逻辑上断开其中一条连接，使其成为备份线路。当网络出现断路时，该协议会自动启动上述备份线路，确保网络正常工作。



图三 生成树拓扑

部分**术语**：

根节点：Root Switch，生成树网络的根。一个网络中只有一个根节点，网络中 ID 最小的交换机作为根节点，也即最终网络去环后的树状图的根。

描述对象：整个网络

根端口：Root Port，RP，网络中除了根节点之外，每一个结点有一个根端口。节点在网络中使用根端口连接到根节点，根端口是该节点到根节点路径开销最小的端口。

描述对象：某个节点

指定端口：Designated Port，DP，每一个网段（Segment，直连网络段，不需要经由交换机就能到达的部分，即一跳可达）有且只有一个指定端口。指定端口为该网段所有端口中到根节点开销最小的端口。一方面，构建生成树拓扑时，STP

消息通过指定端口发送到该网段内；另一方面，该网段通过指定端口连接其他网段。
描述对象：某个网段

其他端口：Alternate Port，AP，剩余的端口，不参与构建生成树拓扑，不转发任何消息。

配置消息：BPDU Config，节点之间通过交换 Config 消息获取路径及优先级等信息。每个端口独立生成自己的 Config 消息，包括：自己的节点 ID，发送端口 ID，自己认为的根节点 ID，以及到根节点的路径和开销。

发送方式：基于二层组播，目的 MAC 地址为 01-80-C2-00-00-00
这里简述一下几个典型的组播地址：
01-80-C2-00-00-00(STP 协议使用)
01-80-C2-00-00-01(MAC Control 的 PAUSE 帧使用)
01-80-C2-00-00-02(Slow Protocol: 802.3ah OAM/ LACP 协议使用)
01-00-5E-xx-xx-xx(IP 组播地址对应的二层组播地址)。
对于 01-80-C2-00-00-00 该地址，所有参与 STP 计算的交换机都会监听该地址，并对目的地址为该地址的数据包进行响应。

Config 消息由根节点以 Hello Time 的周期发出，消息老化时间为 Max Age
生成树协议的数据包，从上到下封装为：IEEE 802.3 Ethernet > Logical-Link Control > Spanning Tree Protocol
802.1D STP 的消息格式：

00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15	16 17 18 19 20 21 22 23	24 25 26 27 28 29 30 31
Proto ID (0)	**Version (0)**	**Msg Type**
Flags	**Root Switch ID (8 bytes)**	
Root Switch ID		
Root Switch ID	**Root Path Cost**	
Root Path Cost	**Switch ID (8 bytes)**	
Switch ID		
Switch ID	**Port ID**	**Msg Age**
Msg Age	**Max Age**	**Hello Time**
Hello Time	**Forward Delay**	

Proto ID: STP 协议标识，为 0
Version: STP 版本号，为 0
Msg Type: 标识是配置包(0x00)还是拓扑变动包(0x80)
Flags: 标志位，第 1 位标识拓扑变更，第 8 位标志拓扑变更确认
Root Switch ID: 该节点认为的根节点 ID，前 16 位为优先级，后 48 位为 MAC 地址
Root Path Cost: 从该节点该端口到根节点的开销
Switch ID: 发送该消息的节点 ID，定义方式同 Root Switch ID
Port ID: 发送该消息的端口 ID，前 8 位为优先级，后 8 位为编号

Msg Age: 该消息已存活时间，单位为 1/256 秒
Max Age: 消息最长允许存活时间，单位同上，默认 20 秒
Hello Time: 配置消息发送时间间隔，单位同上，默认 2 秒
Forward Delay: 不同状态间切换时延，单位同上，默认 15 秒

生成树协议原理：

数据结构：

```
typedef struct stp stp_t;
struct stp {
    u64 switch_id;              // 节点 ID

    u64 designated_root;        // 节点自己认为的根节点，指定根节点
    int root_path_cost;         // 从自己到根节点的路径开销
    stp_port_t *root_port;      // 根端口，起初该指针为空

    long long int last_tick;    // switch timers

    stp_timer_t hello_timer;    // hello timer

    // ports
    int nports;                 // 端口数
    stp_port_t ports[STP_MAX_PORTS];  // 节点的各个端口

    pthread_mutex_t lock;
    pthread_t timer_thread;
};

struct stp_port {
    stp_t *stp;                 // 指向该端口所在节点

    int port_id;                // 端口 ID
    char *port_name;
    iface_info_t *iface;

    int path_cost;              // 通过该端口所在网段的开销，本实验
                                   中为 1

    u64 designated_root;        // 端口自己认为的根节点，指定根节点

    u64 designated_switch;      // 端口所在网段到根节点的上一跳节点
                                   的 ID，也就是该端口所在网段指定端
                                   口所在的节点 ID
    int designated_port;        // 端口所在网段的指定端口
    int designated_cost;        // 本端口所在网段到根节点的路径开销
```

};

选择根节点：

1.初始化时，**所有节点都认为自己是根节点：**

1.1 节点将**自己认定的根节点**修改为**自己的节点 ID**

1.2 遍历每个端口，设置每个端口为**指定端口**：端口将自己认定的根节点设置为自己所在节点,自己认定到根节点的路径开销设置为 0，自己所在网段到根节点的上一跳节点的 ID 设置为自己所在节点，自己所在网段的指定端口设置为自己

2.算法开始后，当节点**认为自己是根节点**时，周期性的（通过 hello 定时器）**主动向外发送 Config 信息**，直到该节点不再认为自己是根节点为止。如果收到的 Config 信息中的根节点 ID 比自己认为的根节点 ID **小**，就将自己认为的根节点**更新**为消息中的根节点，并**转发**该 Config 消息。一直迭代下去，直到所有节点认为的根节点是同一节点。

选择端口状态：

除了根节点，每个节点都有一个根端口，根端口在该节点的所有端口中到根节点的开销最小。每个网段在自己所有的端口中选择一个到根节点开销最小的端口作为指定端口。剩余的端口都称为其他端口。

1.某节点的某端口收到 Config 消息后，将其与**本端口的 Config 进行优先级比较：**

1.1 如果收到的 Config 优先级**高**（收到的 Config 消息只可能从该端口所在的网段的其他端口发送而来），说明该网段**应该通过对方端口连接根节点：**

1.1.1 首先将**本端口的 Config 按照收到的 Config 消息更新**（修改该端口自己认为的根节点、自己认为的到根节点的路径代价、自己认为的所在网段的上一跳交换机，自己认为的所在网段的指定端口）

1.1.2 随后更新节点状态：**遍历所有非指定端口**，找到**优先级最高的非指定端口**（并认定该端口为根端口，因为该端口优先级最高，意味着是距离根节点最近的端口）：

注：该步的意义在于，接收 config 的端口信息更新后，可能会取代节点原来的根端口，成为该节点新的根端口

1.1.2.1 如果**根端口不存在**（即不存在非指定端口），说明节点本身就是根节点，就修改节点的根端口为空，节点认为的根节点为节点自身 ID，节点认为的到根节点的路径代价为 0。

1.1.2.2 如果**根端口存在**，节点就选择通过根端口连接到根节点（此时有可能**接收 Config 的端口优先级提高，取代**了该节点原来的根端口），更新节点状态：节点**根端口**修改为**刚刚查找到的根端口，**节点自己认定的根节点修改为根端口自己认定的根节点，节点自己认为到根节点的路径开销修改为**根端口所在网段到根节点的路径开销**与**通过端口所在网段的路径开销**之和

1.1.3 节点状态更新完后，**更新剩余端口的 Config：**

1.1.3.1 对于所有**指定端口**，更新该端口认为的根节点为节点认为的根节点，更新该端口所在网段到根节点的路径开销为节点自己到

根节点的路径开销

1.1.3.2 对于**非指定端口**，如果该端口的 Config **较该端口所在网段内其他端口优先级更高**，那么**修改该端口为所在网段的指定端口**：更新该端口认为的根节点为节点认为的根节点，更新该端口所在网段到根节点的路径开销为节点自己到根节点的路径开销，更新该端口所在网段到根节点的上一跳节点 ID 为当前节点，更新该端口所在网段的指定端口 ID 为该端口

1.1.4 之后，如果节点**由根节点变为非根节点**，就**停止** hello 计时器

1.1.5 最后，每个指定端口将自己的 Config 信息从端口自身发送出去

1.2 如果收到的 Config 优先级比接收端口的**优先级低**，说明接收端口为其所在网段的**指定端口**，只**将该端口本身的 Config 消息从端口自身发送出去**

2. 当 STP 收敛后，**每个网段内所有端口的配置都相同**。需要注意的是，本次实验中，不考虑拓扑变动下的生成树重构（标准 STP 中，当一个节点感知到链路/端口变化后，通过发送 TCN（拓扑变动提醒）数据包告知根节点，根节点确认后再重新构建生成树），也没有考虑如何与 MAC 学习共存，也没有考虑快速构建生成树

附：优先级：

1. 如果两者认为的根节点 ID 不同，则根节点 ID 小的一方优先级高
2. 若相同，如果两者到根节点的路径开销不同，则开销小的一方优先级高
3. 若相同，如果两者到根节点的上一跳节点 ID 不同，则上一跳节点 ID 小的一方优先级高
4. 若相同，如果两者到根节点的上一跳端口 ID 不同，则上一跳端口 ID 小的一方优先级高

注：以上详细信息记录在个人 CSDN 博客下：
https://blog.csdn.net/Therock_of_lty/article/details/106027152

2. 启动脚本

1) 四节点环形拓扑 STP 测试

```
make all
sudo python four_nodes_ring.py / sudo python four_nodes_ring_diamonds.py
mininet> xterm b1 b2 b3 b4
b1# ./stp > b1-output.txt 2>&1
b2# ./stp > b2-output.txt 2>&1
b3# ./stp > b3-output.txt 2>&1
b4# ./stp > b4-output.txt 2>&1
//注：以上四行可以在 py 脚本中添加下行命令替代：
      node.cmd('./stp > %s-output.txt 2>&1 &' % name)
//等待一段时间后，在另一个新终端中：
sudo pkill -SIGTERM stp
./dump_output.sh 4
```

```
//在原终端中：
mininet> quit
```

2)八节点拓扑 STP 测试

```
make all
sudo python eight_nodes.py / sudo python eight_nodes_diamonds.py
mininet> xterm b1 b2 b3 b4 b5 b6 b7 b8
b1# ./stp > b1-output.txt 2>&1
b2# ./stp > b2-output.txt 2>&1
b3# ./stp > b3-output.txt 2>&1
b4# ./stp > b4-output.txt 2>&1
b5# ./stp > b5-output.txt 2>&1
b6# ./stp > b6-output.txt 2>&1
b7# ./stp > b7-output.txt 2>&1
b8# ./stp > b8-output.txt 2>&1
//注：以上八行可以在 py 脚本中添加下行命令替代：
      node.cmd('./stp > %s-output.txt 2>&1 &' % name)
//等待一段时间后，在另一个新终端中：
sudo pkill -SIGTERM stp
./dump_output.sh 8
//在原终端中：
mininet> quit
```

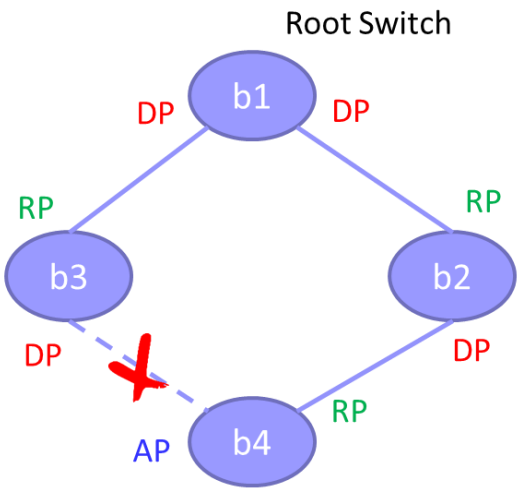
三、实验结果及分析

1. 实验结果

```
moto@CN-VirtualBox:/mnt/shared/06-stp/06-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.
```

图四 四节点环型拓扑 STP 测试结果

将以上信息表示成图：



图五 四节点环型拓扑 STP 拓扑图

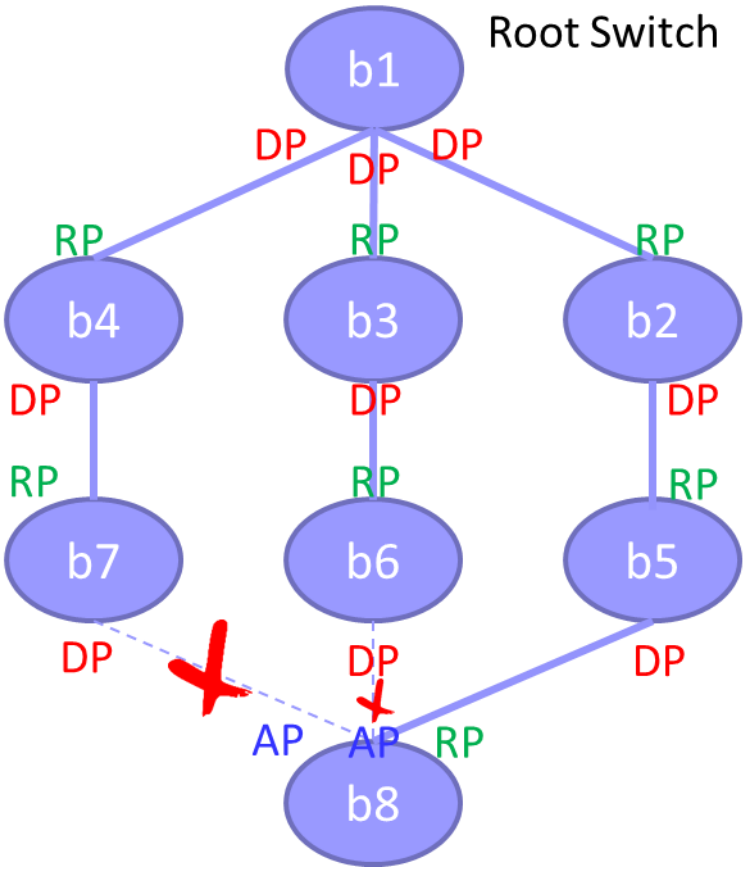
对于四节点环路，STP 形成的拓扑满足我们生成优先级最高的生成树要求。

```
moto@CN-VirtualBox:/mnt/shared/06-stp/06-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.
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.

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: 1.
INFO: port id: 01, role: ROOT.
INFO:   designated ->root: 0101, ->switch: 0101, ->port: 03, ->cost: 0.
INFO: port id: 02, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0401, ->port: 02, ->cost: 1.
```

```
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: 0401, ->port: 02, ->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: 2.
INFO: port id: 01, role: ROOT.
INFO:   designated ->root: 0101, ->switch: 0301, ->port: 02, ->cost: 1.
INFO: port id: 02, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0601, ->port: 02, ->cost: 2.

NODE b7 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: 0701, ->port: 02, ->cost: 2.

NODE b8 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: 0601, ->port: 02, ->cost: 2.
INFO: port id: 03, role: ALTERNATE.
INFO:   designated ->root: 0101, ->switch: 0701, ->port: 02, ->cost: 2.
```

图五 八节点环型拓扑 STP 测试结果

将以上信息表示成图：



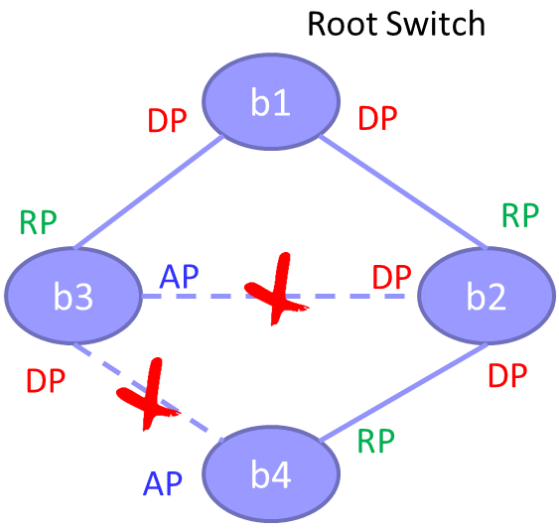
图六 八节点环型拓扑 STP 拓扑图

对于八节点环路，STP 形成的拓扑满足我们生成优先级最高的生成树要求，尤其是对于 b8 而言根端口的选择，需要选择连接节点 ID 较小的节点。

```
moto@CN-VirtualBox:/mnt/shared/06-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.
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: ALTERNATE.
INFO:   designated ->root: 0101, ->switch: 0201, ->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: 03, ->cost: 1.
INFO: port id: 02, role: ALTERNATE.
INFO:   designated ->root: 0101, ->switch: 0301, ->port: 03, ->cost: 1.
```

图七 四节点环型拓扑 STP 测试结果

将以上信息表示成图：



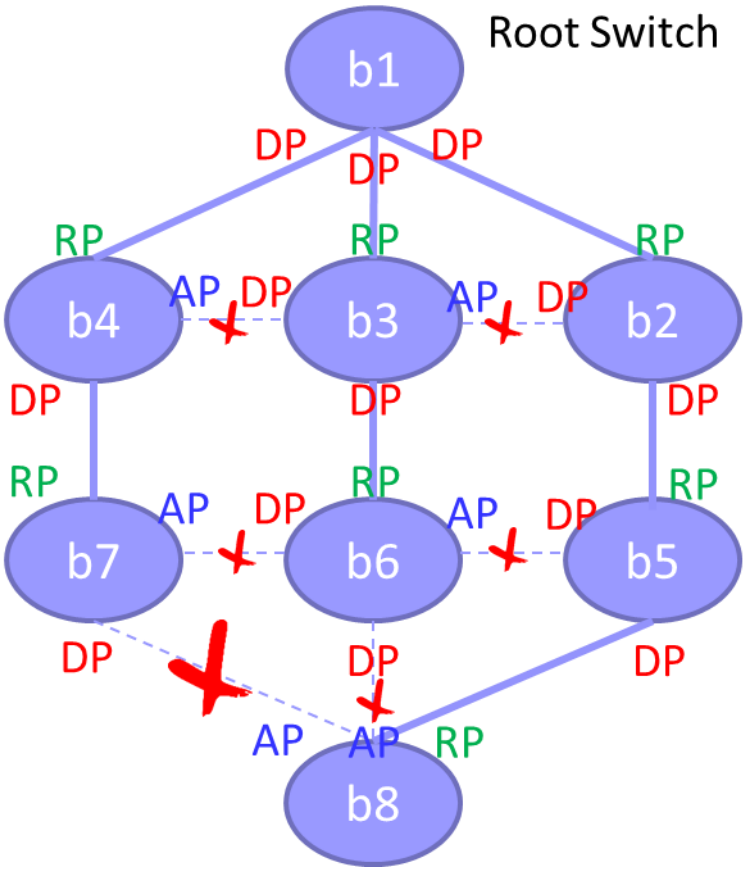
图八 四节点环型拓扑 STP 拓扑图

对于四节点环路，STP 形成的拓扑满足我们生成优先级最高的生成树要求。特别注意的是，对于 b2 和 b3 间的网段，两个端口到根节点路径开销相同，指定端口选择节点 ID 较小的端口。

```
moto@CN-VirtualBox:/mnt/shared/06-stp$ sudo  pkill -SIGTERM stp
moto@CN-VirtualBox:/mnt/shared/06-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.
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: 1.
INFO: port id: 01, role: ROOT.
INFO:   designated ->root: 0101, ->switch: 0101, ->port: 02, ->cost: 0.
INFO: port id: 02, role: ALTERNATE.
INFO:   designated ->root: 0101, ->switch: 0201, ->port: 02, ->cost: 1.
INFO: port id: 03, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0301, ->port: 03, ->cost: 1.
INFO: port id: 04, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0301, ->port: 04, ->cost: 1.

NODE b4 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: 0301, ->port: 03, ->cost: 1.
INFO: port id: 03, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0401, ->port: 03, ->cost: 1.

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: 0401, ->port: 03, ->cost: 1.
INFO: port id: 02, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0501, ->port: 02, ->cost: 2.
INFO: port id: 03, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0501, ->port: 03, ->cost: 2.

NODE b6 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: 04, ->cost: 1.
INFO: port id: 02, role: ALTERNATE.
INFO:   designated ->root: 0101, ->switch: 0501, ->port: 02, ->cost: 2.
INFO: port id: 03, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0601, ->port: 03, ->cost: 2.
INFO: port id: 04, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0601, ->port: 04, ->cost: 2.

NODE b7 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: 0601, ->port: 03, ->cost: 2.
INFO: port id: 03, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0701, ->port: 03, ->cost: 2.

NODE b8 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: 03, ->cost: 2.
INFO: port id: 02, role: ALTERNATE.
INFO:   designated ->root: 0101, ->switch: 0601, ->port: 04, ->cost: 2.
INFO: port id: 03, role: ALTERNATE.
INFO:   designated ->root: 0101, ->switch: 0701, ->port: 03, ->cost: 2.
```

图九 八节点环型拓扑 STP 测试结果

将以上信息表示成图：



图十 八节点环型拓扑 STP 拓扑图

对于八节点环路，STP 形成的拓扑满足我们生成优先级最高的生成树要求，一方面是对于 b8 而言根端口的选择，需要选择连接节点 ID 较小的节点；另一方面对于横向的各个网段，两个端口到根节点路径开销相同时，选择节点 ID 小的一个端口作为指定端口。

2. 实验分析

本次实验实现的具体代码见第二部分实验代码详解。
实现生成树协议的主函数：

```
int main(int argc, const char **argv)
{
    if (getuid() && geteuid()) {
        printf("Permission denied, should be superuser!\n");
        exit(1);
    }
    ustack_init();
    ustack_run();
    return 0;
```

```
}
void ustack_init()
{
    instance = safe_malloc(sizeof(ustack_t));

    bzero(instance, sizeof(ustack_t));
    init_list_head(&instance->iface_list);

    init_all_ifaces();

    stp_init(&instance->iface_list);
}
```

ustack_init 与之前几次实验比较类似，先创建实例，然后初始化实例链表和所有端口，随后调用本次实验的核心部分 stp_init 函数。

```
void stp_init(struct list_head *iface_list)
{
    stp = malloc(sizeof(*stp));

    // set switch ID
    u64 mac_addr = 0;
    iface_info_t *iface = list_entry(iface_list->next, iface_info_t,
 list);
    for (int i = 0; i < sizeof(iface->mac); i++) {
        mac_addr <<= 8;
        mac_addr += iface->mac[i];
    }
    stp->switch_id = mac_addr | ((u64) STP_BRIDGE_PRIORITY << 48);

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

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

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

    stp->nports = 0;
    list_for_each_entry(iface, iface_list, list) {
        stp_port_t *p = &stp->ports[stp->nports];

        p->stp = stp;
        p->port_id = (STP_PORT_PRIORITY << 8) | (stp->nports + 1);
```

```
            p->port_name = strdup(iface->name);
            p->iface = iface;
            p->path_cost = 1;
            stp_port_init(p);

            // store stp port in iface for efficient access
            iface->port = p;

            stp->nports += 1;
        }

        pthread_mutex_init(&stp->lock, NULL);
        pthread_create(&stp->timer_thread, NULL, stp_timer_routine, NULL
    );
        signal(SIGTERM, stp_handle_signal);
    }
```

对照代码来看，首先 malloc 一个 stp 节点空间，然后 list_entry 找到 instance->list 中的第二个节点（该节点记录了该 stp 节点自身的 MAC 信息，之所以寻找 iface_list->next，是因为 iface_list 实际上是&instance->iface_list，而该链表的第一个节点是空节点，第二个节点开始才储存信息（详情见 find_available_ifaces 函数））将记录的 MAC 地址和左移 48 位的 STP_BRIDGE_PRIORITY（32768，默认优先级）拼接后写入到 stp->switch_id 中。

随后**初始化**：

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

之后初始化 hello 计时器。

随后遍历所有的 iface，为每一个 iface 建立一个 stp_port：

```
    p->stp = stp;
     p->port_id = (STP_PORT_PRIORITY << 8) | (stp->nports + 1);
     p->port_name = strdup(iface->name);
     p->iface = iface;
     p->path_cost = 1;
```

初始化每个端口：

```
static void stp_port_init(stp_port_t *p)
{
    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;
}
```

将初始化后的端口保存到 iface 对应的 port 字段。最后，创建 stp 表项锁，启动 hello 线程，增加一个信号量 signal(SIGTERM, stp_handle_signal)用于使用例外终止 stp 程序。

（二）实验代码详解

一、stp_handle_config_packet

```
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");
    int priority = 0;
    //compare priority between configs
    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 &0xffffffffffff)> (ntohll(con
fig->switch_id) &0xffffffffffff))? 1:0;
    else if(p->designated_port != ntohs(config->port_id) )
        priority = ((p->designated_port &0xff)> (ntohs(config->port_id)
 &0xff))? 1:0;
```

首先将端口收到的 config 信息与端口自身的 config 信息进行优先级比较。

```
    if (priority == 1) {//Lastest received config's priority is higher

        // Replace config for this port
        // This case proves that this port is designated_port
        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);
```

如果新收到的 config 优先级高，就修改端口的 config

```
        int root_num = 0;
        int find_root = 1;
        // To find a root_port.
        for (int i = 0; i < stp->nports; i++) {
            //If one port is possiblely a root port, it is non-
```

```
designated
            if (!stp_port_is_designated(&(stp->ports[i]))) {
                root_num = i;
                break;
            }
            // Not found root_port.
            if (i == stp->nports - 1) find_root = 0;
        }
```

找到第一个非指定节点，以便接下来进行根节点的查找

```
        //After above, we find the smallest non-designated port
        for (int i = root_num + 1; i < stp->nports; i++) {
            if (stp_port_is_designated(&(stp->ports[i]))) continue;
            int priority = 1;
            if(stp->ports[i].designated_root !=  stp->ports[root_num].d
esignated_root )
                priority = ( stp->ports[i].designated_root > stp->ports
[root_num].designated_root )? 0:1;
            else if( stp->ports[i].designated_cost !=  stp->ports[root_
num].designated_cost )
                priority = ( stp->ports[i].designated_cost > stp->ports
[root_num].designated_cost )? 0:1;
            else if (stp->ports[i].designated_switch != stp->ports[root
_num].designated_switch )
                //fitst bit of switch is 1, so if compare directly, ans
wer will be opposite
                priority = ((stp->ports[i].designated_switch &0xffffff
fffff) > (stp->ports[root_num].designated_switch &0xffffffffffff))? 0:1;
            else if(stp->ports[i].designated_port != stp->ports[root_nu
m].designated_port )
                priority = ((p->designated_port &0xff)> (stp->ports[roo
t_num].designated_port &0xff))? 0:1;

            if (priority) root_num = i;
        }
```

各端口比较优先级，找到 config 信息优先级最高的端口

```
        //After above, we find the root_port(if exists)
        if (!find_root) {
            // This is root node.
            stp->root_port = NULL;
            stp->designated_root = stp->switch_id;
```

```
            stp->root_path_cost = 0;
        } else {
            stp->root_port = &(stp->ports[root_num]);
            stp->designated_root = stp->root_port->designated_root;
            stp->root_path_cost = stp->root_port->designated_cost + stp
->root_port->path_cost;
        }
```

如果没找到任何的非指定节点，说明本节点为根节点，进行节点设置
如果找到了优先级最高的节点，那么认为它为根节点，修改节点配置

```
        // Replace config for this node's port
        for (int i = 0; i < stp->nports; i++) {
            if (stp_port_is_designated(&(stp->ports[i]))) {
                stp->ports[i].designated_root = stp->designated_root;
                stp->ports[i].designated_cost = stp->root_path_cost;
            }
            //If one net's non-
designated port's designated_cost is higher than this port
            //'s node's root_path_cost, that net should change its desi
gnated port to this node's port
            else if ((stp->root_path_cost < stp->ports[i].designated_co
st) || (stp->root_path_cost==stp->ports[i].designated_cost)&
(stp->switch_id < stp->ports[i].designated_switch) ||
(stp->root_path_cost==stp->ports[i].designated_cost)&
(stp->switch_id==stp->ports[i].designated_switch)&
(stp->ports[i].port_id==stp->ports[i].designated_port)
) {        //this judge can be added like this:
            //

                stp->ports[i].designated_switch = stp->switch_id;
                stp->ports[i].designated_port = stp->ports[i].port_id;
                stp->ports[i].designated_root = stp->designated_root;
                stp->ports[i].designated_cost = stp->root_path_cost;
            }
        }
```

遍历所有节点，如果一个端口是指定端口，更新其认为的根节点和路径开销
如果是非指定端口，并且其 config 较网段内其他端口更高，那么修改该端口 config 使其成为指定端口，更改其认为的指定端口和节点，更新其认为的根节点和路径开销

```
        //If node lose its root character
```

```
        if (!stp_is_root_switch(stp))
            stp_stop_timer(&(stp->hello_timer));
        //Send updated config message from every desginated port
        for (int i = 0; i < stp->nports; i++)
            if (stp_port_is_designated(&(stp->ports[i])))
                stp_port_send_config(&(stp->ports[i]));
    } else {
        stp_port_send_config(p);
    }
}
```

最后，如果节点从根节点变成普通节点，就停止 hello 计时器，从每个指定端口发送 config。

如果起初收到的 config 优先级没有本端口高，那么说明本端口是指定端口，从该端口发送 config 消息。