生成树机制实验报告

张翔雨 2018K8009929035

一、实验题目：生成树机制实验

二、实验内容

- 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
- 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
- 在 `four_node_ring.py` 基础上，添加两个端节点，把第05次实验的交换机转发代码与本实验代码结合，试着构建生成树之后进行转发表学习和数据包转发

三、实验过程

1.实现stp端口处理config消息的流程

（1）收到Config消息后，将其与本端口Config进行优先级比较

该功能的实现被封装在 `stp.c` 中的 `config_packet_priority_cmp(stp_port_t *p,struct stp_config *config)` 函数中。

函数功能为端口Config与收到Config消息优先级的比较

函数的实现逻辑为：

- 先比较根节点ID，根节点ID小的一方优先级高。
- 再比较两者到根节点的开销，开销小的一方优先级高。
- 再比较两者到根节点的上一跳节点，上一跳节点ID小的一方优先级高。
- 最后比较两者到根节点的上一跳端口，上一跳端口ID小的一方优先级高。

```
int config_packet_priority_cmp(stp_port_t *p,struct stp_config *config)//return 1 for
config has higher priority
{
    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;
}
```

代码是比较简单的比较结构，需要注意的是这里端口与config消息之间的比较要进行网络、本地字节序转换。

（2）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

这里只需要在判断得到结果后将config消息从本端口发送出去即可。

```
static void stp_handle_config_packet(stp_t *stp, stp_port_t *p,
        struct stp_config *config)
{
    if(config_packet_priority_cmp(p,config))
    {
        `...
    }
    else
        stp_port_send_config(p);
}
```

（3）否则，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

函数的功能为将本端口的Config替换为收到的Config消息，本端口为非指定端口。

函数实现逻辑较为简单，只需简单的替换即可，同样需要注意这里应该进行网络本地字节序转换。

```
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);
}
```

（4）更新节点状态

该功能的实现被封装在 `stp.c` 中的 `update_stp_status(stp_t *stp)` 函数中。

函数主要功能是更新当前节点的状态。

函数实现逻辑是遍历所有端口，找到满足条件的根端口（该端口是非指定端口，该端口的优先级要高于所有剩余非指定端口），如果不存在根端口，则该节点为根节点；否则，选择通过root_port连接到根节点，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)//说明是第一次找到非指定端口
                cur_root_id = i;
            else
            {
                if(port_priority_cmp(&stp->ports[cur_root_id],&stp->ports[i]))//与之前找到
的优先级最高的端口作比较
                    cur_root_id = i;
            }
        }
        if((i == stp->nports - 1)&&(cur_root_id == -1))//到最后一个也没有找到非指定端口
            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;
    }
}
```

这里需要实现端口之间的优先级比较函数，由于不需要字节序转换，因此需要单独实现。该功能的实现被封装在 `stp.c` 中的 `port_priority_cmp(stp_port_t *p,stp_port_t *q)` 函数中。比较逻辑与端口和config消息之间的比较相同。

```
int port_priority_cmp(stp_port_t *p,stp_port_t *q)//return 1 for q has higher priority
{
    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;
}
```

（5）更新剩余端口的Config

该功能的实现被封装在 `stp.c` 中的 `update_port_config(stp_t *stp)` 函数中。

函数功能是节点更新自己的状态后更新其他端口的信息。

函数实现逻辑为如果一个端口为非指定端口，且其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。对于所有指定端口，更新其认为的根节点和路径开销。

```
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) && port_stp_all_priority_cmp(p,stp))
        {
            p->designated_switch = stp->switch_id;
            p->designated_port = p->port_id;
        }
```

```
    }
}
```

这里需要用到两个新的功能函数，均实现在`stp.c`中，`stp_port_is_root(stp_t *stp, stp_port_t *p)`判断对应端口是否为根端口，既不是DP也不是RP的端口才是非指定端口。`port_stp_all_priority_cmp(stp_port_t *p,stp_t *stp)`实现非指定端口与网段内其他端口优先级的比较，若优先级为最高，则返回1.

```
int port_stp_all_priority_cmp(stp_port_t *p,stp_t *stp)
{
    for (int i = 0; i < stp->nports; i++)
    {
        stp_port_t *port = &stp->ports[i];
        if(!stp_port_is_designated(port) && port_priority_cmp(p,port))
            return 0;
    }
    return 1;
}
static bool stp_port_is_root(stp_t *stp, stp_port_t *p)
{
    return stp->root_port && (p->port_id == stp->root_port->port_id);
}
```

（6）总体流程整合

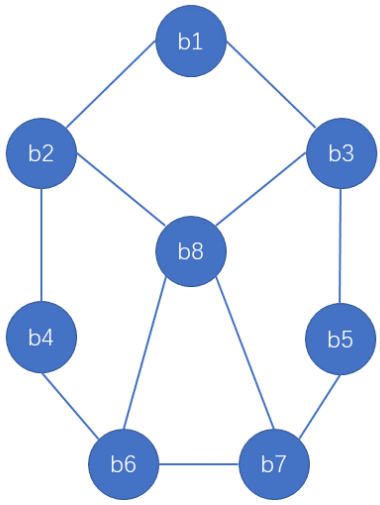
将上文实现的函数按顺序组织完成对config消息的处理。

```
static void stp_handle_config_packet(stp_t *stp, stp_port_t *p,struct stp_config *config)
{
    if(config_packet_priority_cmp(p,config))
    {
        replace_port_config(p,config);
        update_stp_status(stp);
        update_port_config(stp);
        if(!stp_is_root_switch(stp))//如果节点由根节点变为非根节点，停止hello定时器
            stp_stop_timer(&stp->hello_timer);
        stp_send_config(stp);
    }
    else
        stp_port_send_config(p);
}
```

2.构建更加复杂的环形拓扑，计算最小生成树

修改脚本，构造8个节点，4条冗余边的环形拓扑，运行程序，构造拓扑部分脚本代码如下。

```
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)
```



原始八节点环形拓扑

3.将交换机实验代码与本次实验结合，实现转发表学习和数据包转发

首先在脚本中添加两个主机节点，部分脚本代码如下：

```
class RingTopo(Topo):
    def build(self):
        b1 = self.addHost('b1')
        b2 = self.addHost('b2')
        b3 = self.addHost('b3')
        b4 = self.addHost('b4')
        h1 = self.addHost('h1')
        h2 = self.addHost('h2')

        self.addLink(b1, b2)
        self.addLink(b1, b3)
        self.addLink(b2, b4)
        self.addLink(b3, b4)
        self.addLink(h1, b1)
        self.addLink(h2, b4)
if __name__ == '__main__':
    ...
    h1, h2= net.get('h1', 'h2')
    h1.cmd('ifconfig h1-eth0 10.0.0.1/8')
    h2.cmd('ifconfig h2-eth0 10.0.0.2/8')
    for h in [ h1, h2 ]:
        h.cmd('./scripts/disable_offloading.sh')
        h.cmd('./scripts/disable_ipv6.sh')
    ...
```

将交换机相关文件复制到本次实验目录下，首先在main函数中添加转发表初始化操作，然后在对应位置仿照上次实验添加交换机数据包转发的操作，需要注意的是，为了避免形成广播风暴，我们需要禁止 `Alternate Port` 收发数据包，事实上就是将AP对应的冗余边真正从网络中删掉，具体实现为，处理数据包之前判断自身是否为AP，若是则直接释放数据包，不进行处理，同时在 `device_internal.c`

中的 `iface_send_packet` 函数添加判断，如果要发送的端口为AP，则不进行本次发送，具体代码如下：

```
void handle_packet(iface_info_t *iface, char *packet, int len)
{
    struct ether_header *eh = (struct ether_header *)packet;
    if (memcmp(eh->ether_dhost, eth_stp_addr, sizeof(*eth_stp_addr))) {//收到非stp包
        iface_info_t *dst_iface = lookup_port(eh->ether_dhost);
        char* kind1 = stp_port_state(iface->port);//得到当前端口的状态
        if(strcmp(kind1,"ALTERNATE")){
            if(dst_iface)
                iface_send_packet(dst_iface,packet,len);
            else
                broadcast_packet(iface,packet,len);
```

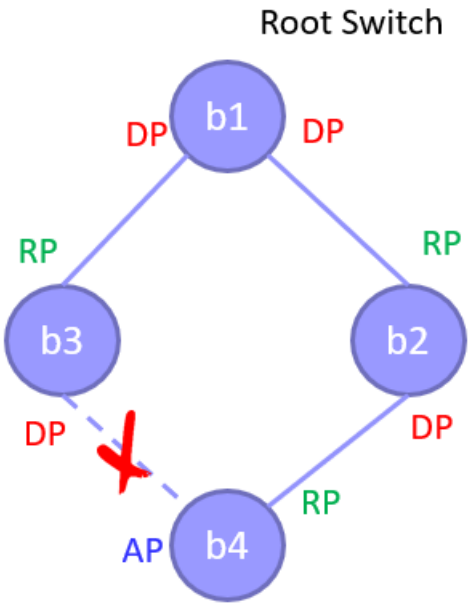
```
            insert_mac_port(eh->ether_shost,iface);
        }
        free(packet);
        return ;
    }
    stp_port_handle_packet(iface->port, packet, len);
    free(packet);
}

void iface_send_packet(iface_info_t *iface, const char *packet, int len)
{
    char* kind = stp_port_state(iface->port);
    if(!strcmp(kind,"ALTERNATE"))//判断要发给AP则直接返回
        return ;
    ...
}
```

四、实验结果

1.生成4个节点环形拓扑的最小生成树

运行四个节点的拓扑脚本，将生成树结果打印如下，与预期结果相符。

```
samuel@ubuntu:/mnt/hgfs/network-labs/Lab5/06-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.
```

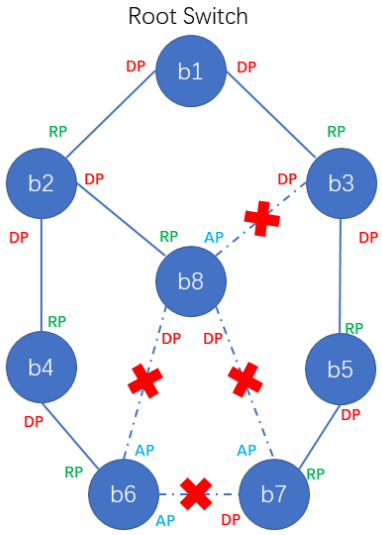


4个节点期望的最小生成树

2.生成8个节点环形拓扑的最小生成树

运行脚本后打印结果如下，与期望最小生成树相符：

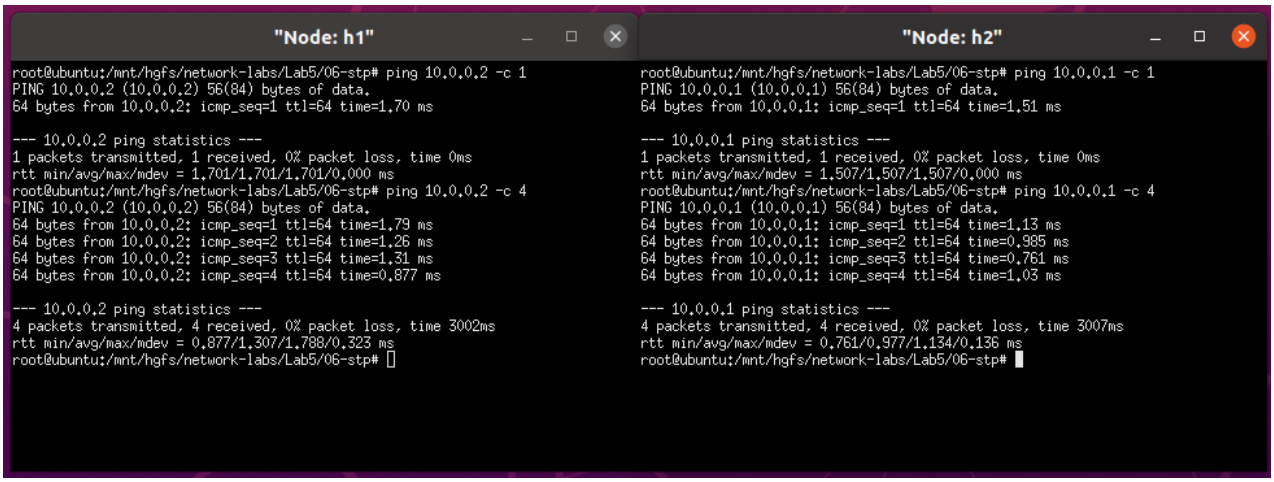
```
samuel@ubuntu:/mnt/hgfs/network-labs/Lab5/06-stp$ ./dump_output.sh 8
NODE b1 dumps:
INFO: this switch is root.
INFO: port id: 01, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0101, ->port: 01, ->cost: 0.
INFO: port id: 02, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0101, ->port: 02, ->cost: 0.

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.
```



期望的8节点拓扑最小生成树

3.实现转发表学习和数据包转发

在脚本中添加两个主机节点，启动脚本，等待30s后在h1向h2发送数据包，测试连通性。



可以看到连通性良好。

五、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？

当节点加入时，需要进行TCN消息处理，简要说明如下

- 如果非根桥上发生拓扑变化，向根桥发送配置TCN信息，通告根桥拓扑已改变。
- 上联的非根桥从指定端口收到配置TCN信息后，会向发送者回复TCA flag位置位的配置TCN信息包，同时继续向根桥发送配置信息。
- 如果跟桥上发生拓扑变化或根桥收到配置TCN信息包后，向发送者回复TCA Flag位置位的配置TCN信息包，同时向所有指定端口发送TC Flag位置位的配置TCN信息包。
- 非根节点根据配置TCN信息生成新的拓扑。

当节点离开时，需要区分是否为根节点，如果不是根节点，则与节点加入时相同处理，如果是根节点离开，其他非根节点在等待最长max age时间（即一个设定的老化时间）后，才能发现根桥失效，进而进行新的根桥选举、根端口和指定端口的确定。

2.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

在初始情况下，端口处于listening状态，STP为避免临时环路，等待足够长的时间，即确保配置消息能同步发送至所有节点时，才会进入转发。STP采用的方式是被动等待计时器超时的方式，首先等待15s后，端口会进入learning状态，此时端口不进行转发，但会学习构建Mac转发表，再等待15s后，进入fowrding状态，开始转发。

3.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（RSTP）在STP基础上新增加了2种端口角色：、Backup端口和边缘端口。通过端口角色的增补，简化了生成树协议的理解及部署。其快速收敛机制为采用了P/A协商机制，用来回确认机制和同步变量机制，无需依靠计时器来保障无环。可以让交换机的互联接口快速进入转发模式。

P/A协商的过程：

- SW1向SW2发送p置位的BPDU包。
- 同步变量（阻塞除边缘端口以外的其他端口，防止出现环路）。
- SW2向SW1发送A置位的BPDU包。
- SW1收到A置位的BPDU包后，端口立即进入Forwarding状态。（一般都是秒级）

七、实验总结

本次实验相较于之前的实验有了比较明显的难度提升，但主体代码部分有课件的详细解释写起来还是比较顺畅的，只是需要合理的封装各种功能的函数才能让代码更加清晰。同时与前面实验的结合部分也非常考验对实验内容的理解，本次实验的思考题也比较有深度，加深了我对STP协议的理解。