实验 6 · 生成树机制实验

吴嘉皓
2015K8009915007

一、实验内容

1. 补全 stp.c 中的 `stp_handle_config_packet(stp_t *stp, stp_port_t *p, struct stp_config *config)`函数，实现对给定拓扑的生成树机制；
2. 自己构造一个不少于 6 个节点，链路冗余度不小于 2 的拓扑，并使用 1 中完成的 stp 程序计算输出最小生成树拓扑；

二、实验流程

（一）代码目录

```
2015K8009915007_吴嘉皓_06.tar.gz
└── 06-stp
    ├── disable_ipv6.sh
    ├── disable_offloading.sh
    ├── dump_output_4.sh
    ├── dump_output_6.sh
    ├── four_node_ring.py
    ├── four_result.log
    ├── include
    │   ├── base.h
    │   ├── ether.h
    │   ├── hash.h
    │   ├── headers.h
    │   ├── list.h
    │   ├── log.h
    │   ├── packet.h
    │   ├── stp.h
    │   ├── stp_proto.h
    │   ├── stp_timer.h
    │   ├── types.h
    │   └── utils.h
    ├── main.c
    ├── Makefile
    ├── packet.c
    ├── six_node_ring.py
    ├── six_result.log
```

```
│   ├── stp.c
│   ├── stp-reference
│   └── stp_timer.c
├── README.txt
└── 实验 6-生成树机制-实验报告.pdf
```

（二）实验流程

1. 按序在 06-stp 目录下输入如下命令：

```
make
sudo python four_nodes_ring.py
mininet> xterm b1
```

然后，等待 30s 左右，在 xterm 的 b1 终端中运行 `pkill -SIGTERM stp` ；

接着在终端中运行，`./dump_output_4.sh` ，并查看比对结果。

2. 按序在 06-stp 目录下输入如下命令：

```
make
sudo python six_node_ring.py
mininet> xterm b1
```

然后，等待 30s 左右，在 xterm 的 b1 终端中运行 `pkill -SIGTERM stp` ；

接着在终端中运行，`./dump_output_6.sh` ，并查看比对结果。

三、实验结果

（一）四节点环路拓扑的文字输出结果

```
NODEb1 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, desinated 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, desinated 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, desinated 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.
```

（二）六节点环路拓扑的文字输出结果

```
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, desinated 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, desinated 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, desinated 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.
INFO: port id: 03, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0401, ->port: 03, ->cost: 2.

NODE b5 dumps:
INFO: non-root switch, desinated 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, desinated root: 0101, root path cost: 3.
INFO: port id: 01, role: ROOT.
INFO:   designated ->root: 0101, ->switch: 0401, ->port: 03, ->cost: 2.
INFO: port id: 02, role: ALTERNATE.
INFO:   designated ->root: 0101, ->switch: 0501, ->port: 02, ->cost: 2.
```

四、结果分析

（一）四节点拓扑结果分析

⇨ 四节点环路拓扑结构如下:

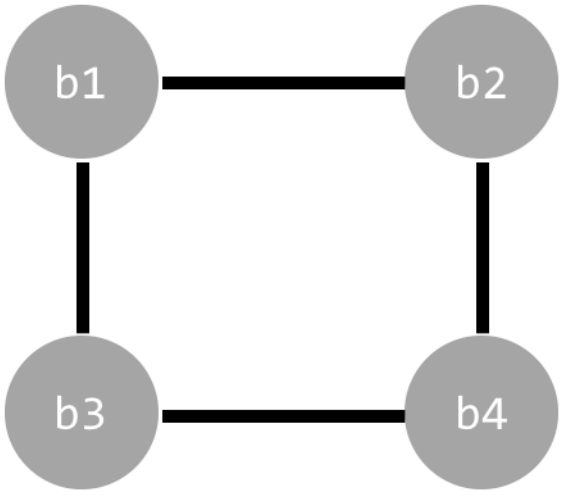

图 1 四节点的环路拓扑

⇨ 依据【三、实验结果】中 dump_output_4 的输出，整理生成树拓扑如下：

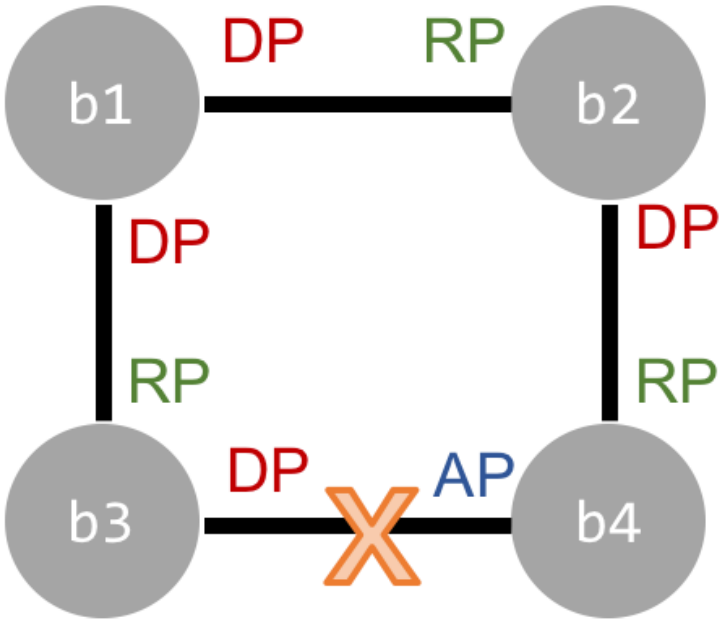

图 2 四节点环路的生成树拓扑

（二）六节点拓扑结果分析

⇨ 六节点环路拓扑结构如下：

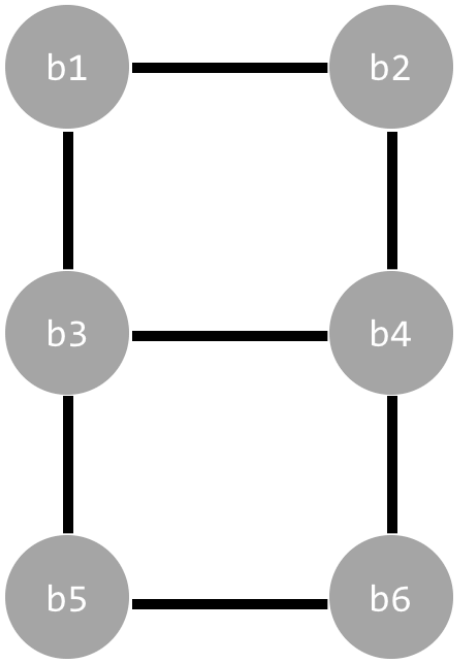

图 3 链路冗余度为 2 的六节点环路拓扑

⇨ 依据【三、实验结果】中 dump_output_4 的输出，整理生成树拓扑如下：

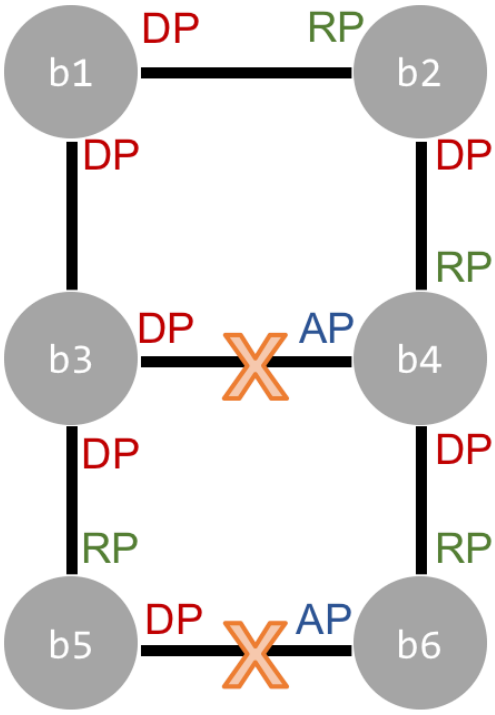

图 4 上述六节点环路的生成树拓扑

（二）代码实现分析

⇨ 主函数 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");
    // if the received config message is prior to port config
    if(stp_pm_priority_compare(config, p)){
        int is_root = stp_is_root_switch(stp);
        stp_replace_port_config(p, config);
        stp_update_switch_status(stp);
        stp_update_port_config(stp);
        is_root &= !stp_is_root_switch(stp);
        if(is_root)
            stp_stop_timer(&(stp->hello_timer));
        stp_send_config(stp);
    }
    else
        stp_port_send_config(p);
}
```

先比较 config 消息和本地端口的优先级（stp_pm_priority_compare()），

如果接收到的 config 消息优先级高，说明该网段通过对方端口连接根节点代价更小，则进行如下操作：

1) 将本端口的 Config 替换为收到的 Config 消息（stp_replace_port_config(p, config)）
2) 更新节点状态(stp_update_switch_status(stp))
3) 更新剩余端口的 Config(stp_update_port_config(stp))
4) 如果节点由根节点变为非根节点，停止 hello 定时器
5) 将更新后的 Config 从每个指定端口转发出去

否则，说明该网段通过本端口连接根节点代价更小，则进行以下操作：

1) 从该端口发送 Config 消息

【注意】判断节点由根结点变为非根节点时，需要注意，由于在 2）中更新节点状态时已经将该节点的 switch_id 改成了另外一个更优先的节点。因此，判断节点由根结点变为非根节点的条件为：

更新节点状态前是根结点，并且跟新节点状态之后，变成了非根节点。

⇨ stp_pm_priority_compare() && stp_pp_priority_compare()

```
// compare the priority of configures between port and config message
// arg: struct stp_config msg && stp_port_t port
// ret: 1 if config message is prior to port config
//      0 if port config is prior to config message
static int stp_pm_priority_compare(struct stp_config *msg, stp_port_t *port){
    if(ntohll(msg->root_id) != port->designated_root)
        return (ntohll(msg->root_id) < port->designated_root);
    else if(ntohl(msg->root_path_cost) != (port->designated_cost+port->path_cost))
        return (ntohl(msg->root_path_cost) < (port->designated_cost+port->path_cost));
    else if(ntohll(msg->switch_id) != port->designated_switch)
        return (ntohll(msg->switch_id) < port->designated_switch);
    else
        return (ntohs(msg->port_id) < port->designated_port);
}
```

```
// compare the priority of configures between port and port
// arg: stp_port_t left_port && stp_port_t right_port
// ret: 1 if the left is prior to the right
//      0 if the right is prior to the left
static int stp_pp_priority_compare(stp_port_t *lp, stp_port_t *rp){
    if(lp->designated_root != rp->designated_root)
        return (lp->designated_root < rp->designated_root);
    else if(lp->designated_cost != rp->designated_cost)
        return (lp->designated_cost < rp->designated_cost);
    else if(lp->designated_switch != rp->designated_switch)
        return (lp->designated_switch < rp->designated_switch);
    else
        return (lp->designated_port < rp->designated_port);
}
```

依次比较认为的根结点 ID、到根结点的代价、上一跳的节点以及上一跳的端口 ID，来得到优先级。（如上图代码所示）比较时注意网络字节序的转换。

⇨ stp_replace_port_config()

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

将本地端口的信息更新为接收到的 config 消息。

【注意】因为 config 消息是网络传输过来的包，因此整型数需要将网络字节序转化为本地字节序才能正确使用。

⇨ stp_update_switch_status()

```
static void stp_update_switch_status(stp_t *stp){
    // try to find the root port
    bzero(p_ptrs, sizeof(stp_port_t*) * STP_MAX_PORTS);

    int rank = 0;
    stp_port_t * p = NULL;
    for (int i = 0; i < stp->nports; i++){
        p = &(stp->ports[i]);
        if (!stp_port_is_designated(p)){
            p_ptrs[rank++] = p;
        }
    }
    stp_port_t * root_port = find_dominated_port(rank);
    // if the root port is not found
    // set the switch as root switch
    if(!root_port){
        stp->designated_root = stp->switch_id;
        stp->root_path_cost = 0;
        return ;
    }
    // otherwise, update the status of the current switch
    stp->root_port = root_port;
    stp->designated_root = root_port->designated_root;
    stp->root_path_cost  = root_port->designated_cost + root_port->path_cost;
    return ;
}
```

1）寻找根端口

a） 找到所有非指定端口；

b） 对所有非指定端口进行优先级排序，取优先级最高的作为根端口；(find_dominated_port(int rank))

2）如果没有根端口，则该节点为根结点；

3）否则，（找到了根端口）设置节点的根端口、指定端口、路径代价。

⇨ find_dominated_port()

```
static stp_port_t* find_dominated_port(int rank){
    stp_port_t * tmp = NULL;
    if(rank == 0)
        return NULL;
    else{
        for(int i = 0; i < rank-1; i++){
            for(int j = 0; j < rank-1-i; j++){
                if(stp_pp_priority_compare(p_ptrs[j], p_ptrs[j+1])){
                    tmp = p_ptrs[j];
                    p_ptrs[j] = p_ptrs[j+1];
                    p_ptrs[j+1] = tmp;
                }
            }
        }
        return p_ptrs[rank-1];
    }
}
```

对所有的非指定端口进行比较，返回指向优先级最高的端口的指针。

如果没有非指定端口，则返回空指针。

⇨ stp_update_port_config()

```
static void stp_update_port_config(stp_t *stp){
    stp_port_t * p = NULL;
    for(int i = 0; i < stp->nports; i++){
        p = &(stp->ports[i]);
        // if the port is the designated port
        if(stp_port_is_designated(p)){
            p->designated_root = stp->designated_root;
            p->designated_cost = stp->root_path_cost;
        }
        else{
            if(stp_ap_to_dp(p)){
                // change the alternate port as designated port
                p->designated_port   = p->port_id;
                p->designated_switch = stp->switch_id;
                p->designated_cost   = stp->root_path_cost;
                p->designated_root   = stp->designated_root;
            }
        }
    }
}
```

扫描所有端口，更新其信息：

1）如果是指定端口，则只需要更新其指定端口信息和路径代价信息；

2）如果是非指定端口，且其网段通过本节点到根节点的代价比通过对端节点的代价小，那么该端口成为指定端口，并更新该端口的所有信息；