生成树机制实验

学号：2016K8009908007
姓名：薛峰

一、实验内容

1、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

①将接收的和本端口的 config 进行优先级比较：两者认为的根节点 ID、两者到根节点的开销、两者到根节点的上一跳节点 ID、两者到根节点的上一跳端口 ID；

②如果收到的 Config 优先级高，则将本端口的 Config 替换为收到的 Config 消息，本端口确定为非指定端口；更新节点状态，更新剩余端口的 Config；如果节点由根节点变为非根节点，停止 hello 定时器；将更新后的 Config 从每个指定端口转发出去；

③如果收到的 Config 优先级低，则继续向外发送本端口的 config。

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

该函数根据实验内容中的生成树算法得到。首先将接收的和本端口的 config 进行优先级比较，若收到的 config 优先级高，则将该端口的 config 替换为接受的 config，之后更新结点状态，停止 hello 定时器，发送更新后的 config；若本端口优先级高，则继续发送本端口的 config。

```
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_cmp_port_conf(p, config) == -1 ) { // config's priority is higher than p's
    bool is_root = stp_is_root_switch(stp);

    undesigne_port(p, config);

    stp_update(stp, config);

    if(is_root)
      stp_stop_timer(&stp->hello_timer);

    stp_send_config(stp);
  }
  else // this port is designated port
    stp_port_send_config(p);
}
```

2、int config_cmp_port_conf(stp_port_t *p, struct stp_config *config)函数

该函数的作用为比较对应接收到的 config 和本端口的 config 的优先级关系。依次比较两者认为的根节点 ID、两者到根节点的开销、两者到根节点的上一跳节点 ID、两者到根节点的上一跳端口 ID。若接收到的 config 优先级高则返回-1，相等则返回 0，本端口的 config 优先级高则返回 1。

```
// compare the priority
// *p > *config, return 1; *p = *config, return 0; *p < *config, return -1;
int config_cmp_port_conf(stp_port_t *p, struct stp_config *config) {

  if( p->designated_root !=  ntohll(config->root_id) )
    return ( p->designated_root > ntohll(config->root_id) )? -1:1;
  else if( p->designated_cost !=  ntohl(config->root_path_cost) )
    return ( p->designated_cost > ntohl(config->root_path_cost) )? -1:1;
  else if( get_switch_id(p->designated_switch) != get_switch_id(ntohll(config->switch_id)) )
    return ( get_switch_id(p->designated_switch) > get_switch_id(ntohll(config->switch_id)) )? -1:1;
  else if( get_port_id(p->designated_port) != get_port_id(ntohs(config->port_id)) )
    return ( get_port_id(p->designated_port) > get_port_id(ntohs(config->port_id)) )? -1:1;
  else
    return 0;
}
```

3、void undesigne_port(stp_port_t *p, struct stp_config *config)函数

进行 config 的替换。

```
void undesigne_port(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、static void stp_update(stp_t *stp, struct stp_config *config)函数

该函数的作用为更新结点状态。一、首先便利该节点的所有端口找到根端口，其满足的条件为：该端口是非指定端口且该端口的优先级要高于所有剩余非指定端口。因此我们首先找到第一个非指定端口，随后便利端口，找到优先级更高的非指定端口后便替换之。其中比较端口间优先级的函数同 config_cmp_port_conf 函数类似。二、随后更新结点状态，如果没找到跟端口，则该结点是根节点，否则选择通过 root_port 连接到根节点。三、最后更新各个端口的 config，如果一个端口为非指定端口，且其网段通过本节点到根节点的开销比通过对端节点的开销小，那么该端口成为指定端口，并且对于所有指定端口，更新其认为的根节点和路径开销。

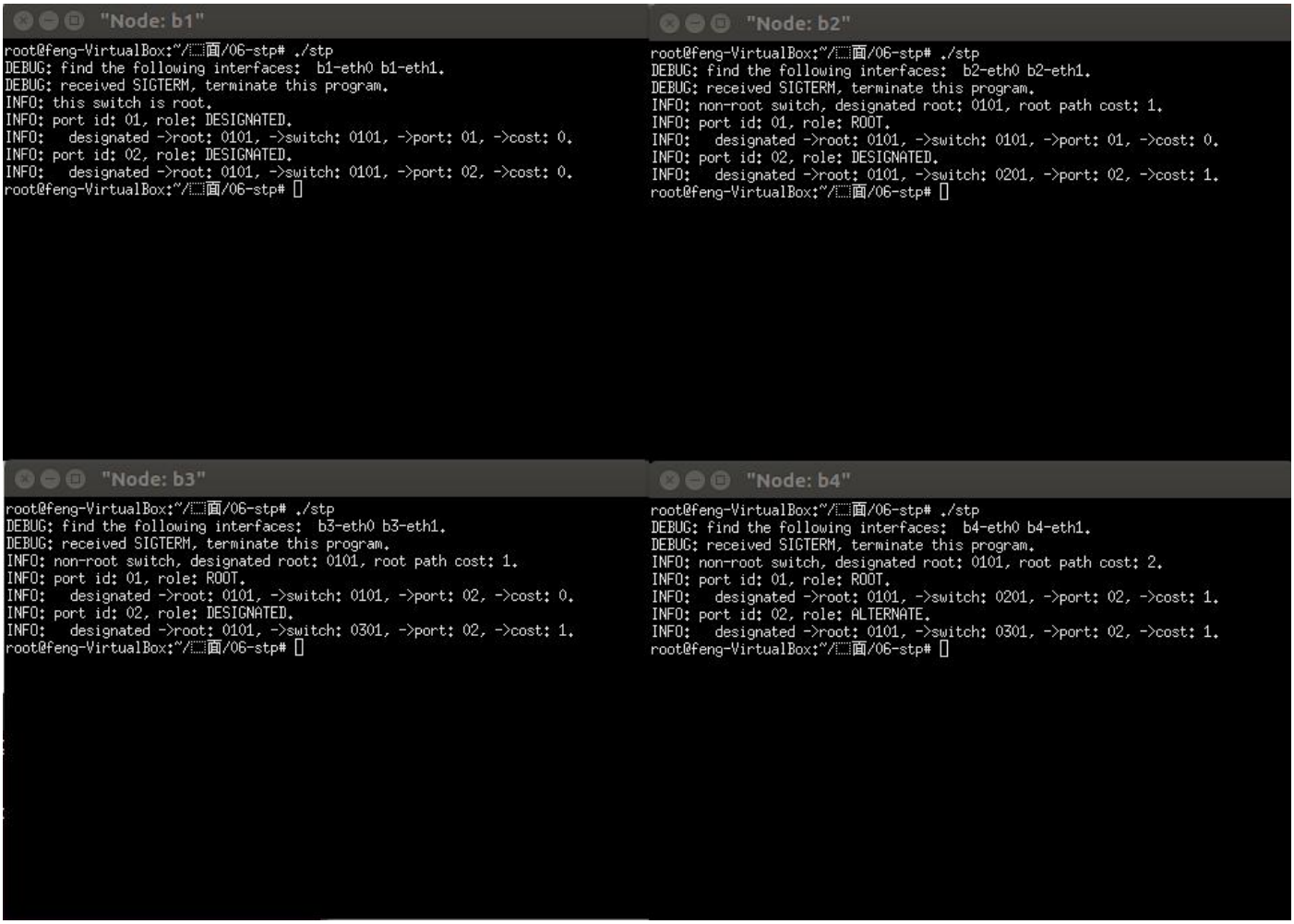
```
// update the node's state and each port's config
static void stp_update(stp_t *stp, struct stp_config *config) {

  int i;
  stp_port_t *root_port = NULL;

  // to find the first undesignated node
  for (i = 0; i < stp->nports; i++){
    if ( !stp_port_is_designated(&stp->ports[i]) ) {
      root_port = &stp->ports[i];
      break;
    }
  }

  // to find the root node
  for (i++ ; i < stp->nports ; i++) {
    if( root_port!=NULL && !stp_port_is_designated(&stp->ports[i])
        && config_cmp_port_port(root_port, &stp->ports[i]) == -1 )
      root_port = &stp->ports[i];
  }

  // update node's state
  if (root_port == NULL) { // this node is root node
    stp->root_port = NULL;
    stp->designated_root = stp->switch_id;
    stp->root_path_cost  = 0;
  }
  else {
    stp->root_port = root_port;
    stp->designated_root = root_port->designated_root;
    stp->root_path_cost  = root_port->designated_cost + root_port->path_cost;
  }
```

```
  // update port's config
  for (i = 0; i != stp->nports; i++ ) {
    if(!stp_port_is_designated(&stp->ports[i])
    && stp->root_path_cost < stp->ports[i].designated_cost) {
      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;
    }
    else 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;
    }
  }
}
```

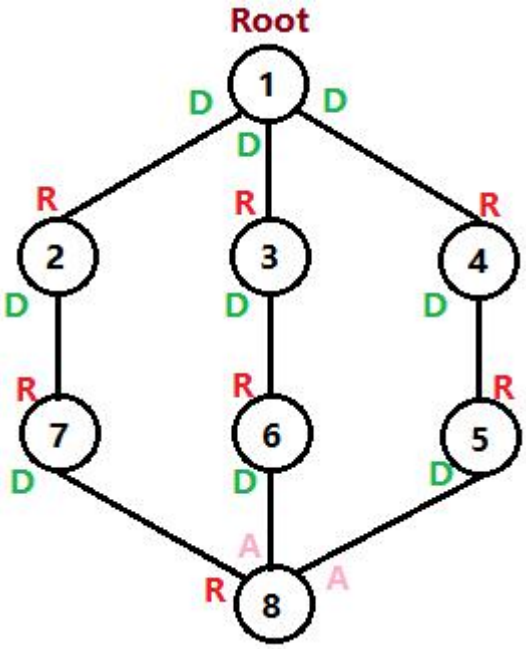
三、实验结果及分析

1、实验结果

four_node_ring



8 个结点的拓扑



"Node: b1"

```
root@feng-VirtualBox:~/Lab/P06/06-stp# ./stp
DEBUG: find the following interfaces:  b1-eth0 b1-eth1 b1-eth2.
DEBUG: received SIGTERM, terminate this program.
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.
root@feng-VirtualBox:~/Lab/P06/06-stp#
```

"Node: b2"

```
root@feng-VirtualBox:~/Lab/P06/06-stp# ./stp
DEBUG: find the following interfaces:  b2-eth0 b2-eth1.
DEBUG: received SIGTERM, terminate this program.
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.
root@feng-VirtualBox:~/Lab/P06/06-stp#
```

"Node: b3"

```
root@feng-VirtualBox:~/Lab/P06/06-stp# ./stp
DEBUG: find the following interfaces:  b3-eth0 b3-eth1.
DEBUG: received SIGTERM, terminate this program.
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.
root@feng-VirtualBox:~/Lab/P06/06-stp#
```

"Node: b4"

```
root@feng-VirtualBox:~/Lab/P06/06-stp# ./stp
DEBUG: find the following interfaces:  b4-eth0 b4-eth1.
DEBUG: received SIGTERM, terminate this program.
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.
root@feng-VirtualBox:~/Lab/P06/06-stp#
```

"Node: b5"

```
root@feng-VirtualBox:~/Lab/P06/06-stp# ./stp
DEBUG: find the following interfaces:  b5-eth0 b5-eth1.
DEBUG: received SIGTERM, terminate this program.
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.
root@feng-VirtualBox:~/Lab/P06/06-stp#
```

"Node: b6"

```
root@feng-VirtualBox:~/Lab/P06/06-stp# ./stp
DEBUG: find the following interfaces:  b6-eth0 b6-eth1.
DEBUG: received SIGTERM, terminate this program.
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.
root@feng-VirtualBox:~/Lab/P06/06-stp#
```

"Node: b7"

```
root@feng-VirtualBox:~/Lab/P06/06-stp# ./stp
DEBUG: find the following interfaces:  b7-eth0 b7-eth1.

DEBUG: received SIGTERM, terminate this program.
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.
root@feng-VirtualBox:~/Lab/P06/06-stp#
root@feng-VirtualBox:~/Lab/P06/06-stp#
```

"Node: b8"

```
root@feng-VirtualBox:~/Lab/P06/06-stp# ./stp
DEBUG: find the following interfaces:  b8-eth2 b8-eth1 b8-eth0.
DEBUG: received SIGTERM, terminate this program.
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.
root@feng-VirtualBox:~/Lab/P06/06-stp#
```

2、结果分析

现象：对于两个两个拓扑结构，stp 生成的结果都正确，与 stp-reference 生成的完全一致。

结论：stp 程序正确，并且生成树算法确实可以通过禁用相关端口，在有环路的网络中构建一个开销最小的树状拓扑，从而避免广播风暴。