网络实验报告

雷正宇 2016K8009909005

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。

实验流程

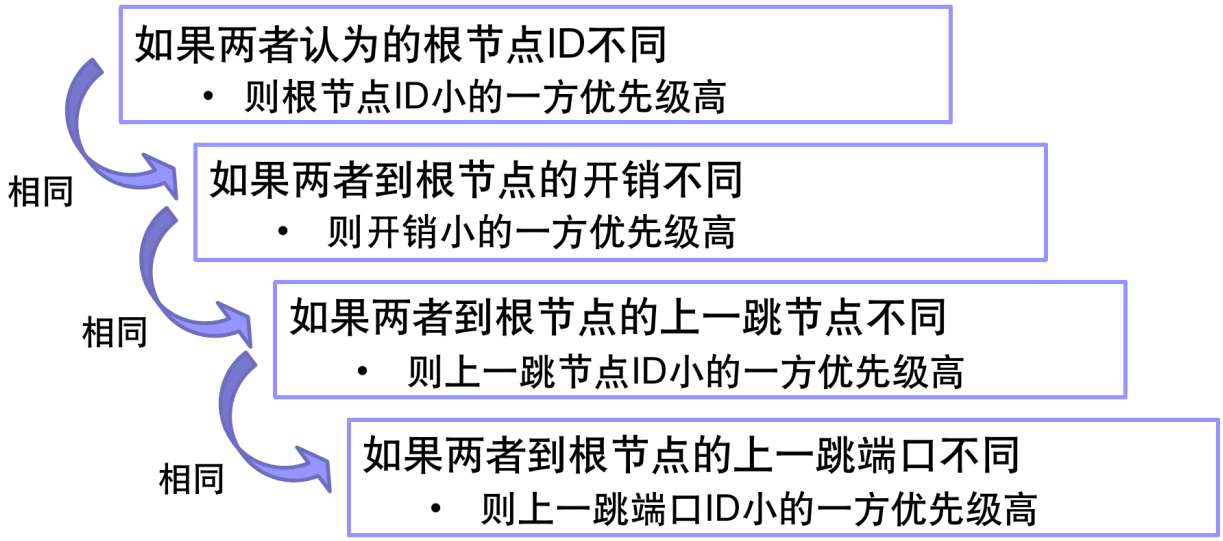
以下实现都在 stp.c 文件中，所有函数都限定为 static，因为

编写优先级逻辑

实验中需要多次比较 port 和 config 的优先级、port 和 port 的优先级。port 和 port 的优先级比较函数如下：

```
static int port_cmp_priority(stp_port_t *a, stp_port_t *b)
{
  // return 1 if a is priorier, -1 if b is priorier
  return a->designated_root > b->designated_root ? -1 :
        a->designated_root < b->designated_root ? 1 :
        a->designated_cost > b->designated_cost ? -1 :
        a->designated_cost < b->designated_cost ? 1 :
        a->designated_switch > b->designated_switch ? -1 :
        a->designated_switch < b->designated_switch ? 1 :
        a->designated_port > b->designated_port ? -1 :
        a->designated_port < b->designated_port ? 1 :
        0;
}
```

比较逻辑为课件中的该部分：



如果需要比较 config 和 port 的优先级，则需要先抽取 config 中用于比较的几项属性，一种比较优雅的实现方式是使用 C99 的临时变量：

```
static int port_config_cmp_priority(stp_port_t *a, struct stp_config
*config)
{
  return port_cmp_priority(a, &((stp_port_t) {
    .designated_root = ntohll(config->root_id),
    .designated_cost = ntohl(config->root_path_cost),
    .designated_switch = ntohll(config->switch_id),
    .designated_port = ntohs(config->port_id)
  }));
}
```

然而，考虑到该实验中还需要用 config 对 port 进行修改，实现如下：

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

复用这个过程，port 和 config 的比较可以写成更简洁的形式：

```c
static int port_config_cmp_priority(stp_port_t *a, struct stp_config *b)
{
  stp_port_t tmp;
  modify_port_by_config(&tmp, b);
  return port_cmp_priority(a, &tmp);
}
```

寻找根节点过程

```c
static stp_port_t *find_root_port(stp_t *stp)
{
  stp_port_t *root_port = NULL;
  for (int i = 0; i < stp->nports; i++) {
    if (!stp_port_is_designated(&stp->ports[i])) {
      root_port = &stp->ports[i];
      break;
    }
  }
  for (int i = 0; i < stp->nports; i++) {
    if (port_cmp_priority(&stp->ports[i], root_port) == 1) {
      root_port = &stp->ports[i];
    }
  }
  return root_port;
}
```

更新节点过程

```c
static void stp_update(stp_t *stp)
{
  stp_port_t *root = find_root_port(stp);
  if (root) {
    stp->designated_root = root->designated_root;
    stp->root_port = root;
    stp->root_path_cost = root->designated_cost + root->path_cost;
  } else {
    stp->designated_root = stp->switch_id;
    stp->root_path_cost = 0;
  }
}
```

更新端口过程

```c
static void stp_port_config_update(stp_t *stp, stp_port_t *p)
{
  for (int i = 0; i < stp->nports; i++) {
```

```
    stp_port_t *port = &stp->ports[i];
    if (port == p)
      continue;
    if (stp_port_is_designated(port)) {
      port->designated_cost = stp->root_path_cost;
      port->designated_root = stp->designated_root;
    } else if (stp->designated_root < port->designated_root ||
                (stp->designated_root == port->designated_root &&
                stp->root_path_cost < port->designated_cost)) {
      port->designated_switch = stp->switch_id;
      port->designated_port = port->port_id;
    }
  }
}
```

这里需要注意的一点是在课件中

- 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：
  - p->designated_switch = stp->switch_id
  - p->designated_port = p->port_id

红字部分想表达的意思可能是这样的判断逻辑：

```
stp->designated_root < port->designated_root ||
                (stp->designated_root == port->designated_root &&
                stp->root_path_cost < port->designated_cost)
```

但容易让人误解（不过大多数情况下不会造成问题）。

完整处理配置消息的过程

```
static void stp_handle_config_packet(stp_t *stp, stp_port_t *p,
    struct stp_config *config)
{
  if (port_config_cmp_priority(p, config) == 1) {
    stp_port_send_config(p);
  } else {
    modify_port_by_config(p, config);
    stp_update(stp);
    stp_port_config_update(stp, p);
    if (!stp_is_root_switch(stp)) {
      stp_stop_timer(&stp->hello_timer);
    }
    stp_send_config(stp);
```

```
  }
}
```

有了以上各个子过程，总的处理过程逻辑就很清晰了：收到配置消息后，先将它与本端口 config 的优先级进行比较，如果收到的配置消息优先级低，说明本端口不需要根据这个 config 进行改动，直接发送自己的 config 消息即可。否则，先根据收到的 config 修改自己的端口，再更新节点状态和剩余节点，如果节点变为非根节点，停止 timer 定时器，最后将更新后的 config 从每个指定端口发送出去。

自己编写一个不少于7个节点的网络拓扑

我编写的拓扑图形如下：

```
       b1
    /  |  \
b2     b4 -- b3
|   /  |     |
b5     b6 -- b7
```

代码如下：

```python
#!/usr/bin/python

from mininet.topo import Topo
from mininet.net import Mininet
from mininet.cli import CLI

def clearIP(n):
    for iface in n.intfList():
        n.cmd('ifconfig %s 0.0.0.0' % (iface))

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')

        self.addLink(b1, b2)
        self.addLink(b1, b3)
        self.addLink(b1, b4)
        self.addLink(b2, b5)
        self.addLink(b3, b4)
        self.addLink(b3, b7)
        self.addLink(b4, b5)
        self.addLink(b4, b6)
```

```
        self.addLink(b6, b7)

if __name__ == '__main__':
    topo = RingTopo()
    net = Mininet(topo = topo, controller = None)

    for idx in range(7):
        name = 'b' + str(idx+1)
        node = net.get(name)
        clearIP(node)
        node.cmd('./disable_offloading.sh')
        node.cmd('./disable_ipv6.sh')

        # set mac address for each interface
        for port in range(len(node.intfList())):
            intf = '%s-eth%d' % (name, port)
            mac = '00:00:00:00:0%d:0%d' % (idx+1, port+1)

            node.setMAC(mac, intf = intf)

        # node.cmd('./stp-reference > %s-output.txt 2>&1 &' % name)
        node.cmd('./stp > %s-output.txt 2>&1 &' % name)

    net.start()
    CLI(net)
    net.stop()
```

实验结果和分析

4节点环的生成树结果：

```
zhengyu@ubuntu:~/Desktop/课件/网络实验/实验5/06-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.
```

自制的7节点拓扑的生成树结果：

```
zhengyu@ubuntu:~/Desktop/课件/网络实验/实验5/06-stp$ ./dump_output.sh 7
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.
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: 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: 02, ->cost: 1.
INFO: port id: 03, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0401, ->port: 03, ->cost: 1.
INFO: port id: 04, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0401, ->port: 04, ->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: 0201, ->port: 02, ->cost: 1.
INFO: port id: 02, role: ALTERNATE.
INFO:   designated ->root: 0101, ->switch: 0401, ->port: 03, ->cost: 1.

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: 0401, ->port: 04, ->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: 0301, ->port: 03, ->cost: 1.
INFO: port id: 02, role: ALTERNATE.
INFO:   designated ->root: 0101, ->switch: 0601, ->port: 02, ->cost: 2.
```