Lab of Computer Network: stp **Fall 2024**

Report 5 — September 29

Lecturer: Wu Qinghua *Completed by: Zhang Jiawei*

5.1 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。

5.2 实验过程

5.2.1 总体流程

本次实验中，计算最小生成树拓扑需要实现的操作比较多，用流程图概括如下：

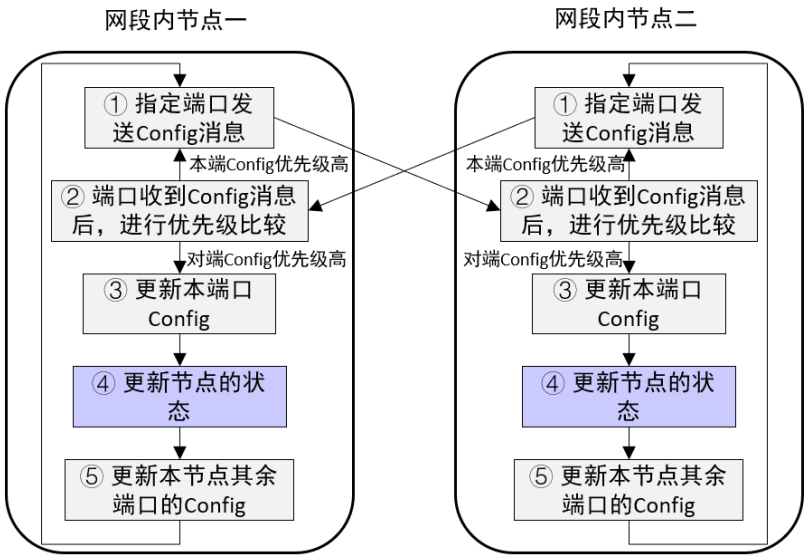


图 **5.1.** 总体流程

网段中的两个节点，通过收发 config 消息，并比较自身收端口 config 和对端发端口 config 的优先级，来决定自己的状态。如果收到的 config 优先级高，说明该网段应该通过对方端口连接根节点，需要进行以下操作：

1. 将本端口的 config 替换为收到的 config 消息 (③)，本端口为非指定端口；

2. 更新节点状态 (④);
3. 更新其余 (Other) 端口的 config(⑤);
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 (①)。

如果收到的 config 优先级低,则该网段应该通过本端口存储 config 对应的端口连接根节点,只需要发送 config 消息 (①) 告知对方优先级更高的 config。

5.2.2 比较 config 优先级

比较 config 优先级的操作如下图所示:

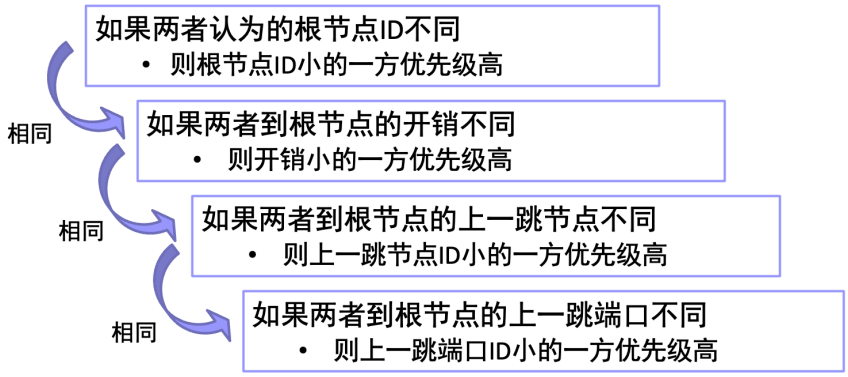


图 5.2. 比较 config 优先级

转化为 C 代码如下:

```
// return true if config of p is superior to the received config
static bool config_is_superior(stp_port_t *p, u64 designated_root, u32
    root_path_cost, u64 switch_id, u16 port_id)
{
  if (p->designated_root < designated_root)
    return true;
  else if (p->designated_root > designated_root)
    return false;
  else if (p->designated_cost < root_path_cost)
    return true;
  else if (p->designated_cost > root_path_cost)
    return false;
  else if (p->designated_switch < switch_id)
    return true;
  else if (p->designated_switch > switch_id)
```

```
    return false;
  else if (p->designated_port < port_id)
    return true;
  else
    return false;
}
```

这里的 `stp_port_t *p` 表示自身收端口，按照上图所示方法进行比较，如果自己的优先级更高，就返回 `true`，否则返回 `false`。

优先级比较完之后，需要根据结果进行相应的操作。如果自己的优先级更高，就从指定端口发出 config 消息：

```
u64 designated_root = ntohll(config->root_id);
u32 root_path_cost = ntohl(config->root_path_cost);
u64 switch_id = ntohll(config->switch_id);
u16 port_id = ntohs(config->port_id);
// if config of p is superior and p is designated, just send config
if (config_is_superior(p, designated_root, root_path_cost, switch_id, port_id) &&
    stp_port_is_designated(p))
     stp_port_send_config(p);
```

如果自己的优先级更低，所需的操作较多，在后面一一叙述。

5.2.3 更新本端口 config

这一部分比较简单，只需要将本端口的 config 的各个域更新为收到的 config 的各个域即可：

```
// 1. replace config of p with config of entry
p->designated_root = designated_root;
p->designated_switch = switch_id;
p->designated_port = port_id;
p->designated_cost = root_path_cost;
```

5.2.4 更新节点状态

首先遍历所有端口，满足如下条件的为根端口 (root_port)：

1. 该端口是非指定端口；
2. 该端口的优先级要高于所有其余非指定端口 (②)；

我先写了一个函数来寻找根端口：

```
// locate root port: the most superior non-designated port
```

```
static stp_port_t *locate_root_port(stp_t *stp)
{
  stp_port_t *root_port = NULL;
  stp_port_t *current_port;

  for (int i = 0; i < stp->nports; i++){
    current_port = &stp->ports[i];
    if (!stp_port_is_designated(current_port)){
      if(root_port){
        if(config_is_superior(current_port, root_port->designated_root,
            root_port->designated_cost, root_port->designated_switch,
            root_port->port_id))
          root_port = current_port;
      }
      else
        root_port = current_port;
    }
  }

  return root_port;
}
```

然后更新节点状态，选择通过 root_port 连接到根节点：

```
// 2. upgrade stp state
stp_port_t *root_port = locate_root_port(stp);

if (root_port){
  stp->root_port = root_port;
  stp->designated_root = root_port->designated_root;
  stp->root_path_cost = root_port->designated_cost + root_port->path_cost;
}
else{
  stp->root_port = NULL;
  stp->designated_root = stp->switch_id;
  stp->root_path_cost = 0;
}
```

这里 `stp->root_path_cost` 的更新是因为路径开销等于路径上全部链路开销之和。

5.2.5 更新其余端口 config

更新其余端口的 config 有多种情况：

1. 非指定端口 → 非指定端口(不需要处理);
2. 指定端口 → 指定端口(需要更新信息,对于所有指定端口,更新其认为的根节点和路径开销);
3. 指定端口 → 非指定端口(只有收到 config 时可能,已在之前的函数中处理);
4. 非指定端口 → 指定端口(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 (②),那么该端口成为指定端口)。

代码如下:

```
// 3. upgrade config of other ports
// for non-designated ports, if it's superior to all other ports, set it as
    designated
for (int i = 0; i < stp->nports; i++){
  stp_port_t* current_port = &stp->ports[i];
  if (root_port && !stp_port_is_designated(current_port) &&
      !config_is_superior(current_port, stp->designated_root, stp->root_path_cost,
      stp->switch_id, current_port->port_id)){
       current_port->designated_switch = stp->switch_id;
       current_port->designated_port = current_port->port_id;
  }
}

// for all designated ports, update designated root and cost
for (int i = 0; i < stp->nports; i++){
  stp_port_t* current_port = &stp->ports[i];
  if (stp_port_is_designated(current_port)){
    current_port->designated_root = stp->designated_root;
    current_port->designated_cost = stp->root_path_cost;
  }
}
```

5.2.6 停止 hello 定时器

如果节点由根节点变为非根节点,需要停止 hello 定时器:

```
// get before_root before receiving config
int before_root = stp_is_root_switch(stp);

// 4. if root port is changed, stop hello timer
if (before_root && !stp_is_root_switch(stp))
  stp_stop_timer(&stp->hello_timer);
```

5.2.7 转发 config

最后，将更新后的 config 从每个指定端口转发出去：

```
// 5. send config from all designated ports
stp_send_config(stp);
```

5.3 实验结果

5.3.1 使用 four_node_ring.py

使用 four_node_ring.py 拓扑，运行 stp 程序，可以得到如下结果：

```
❯ ./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.
```

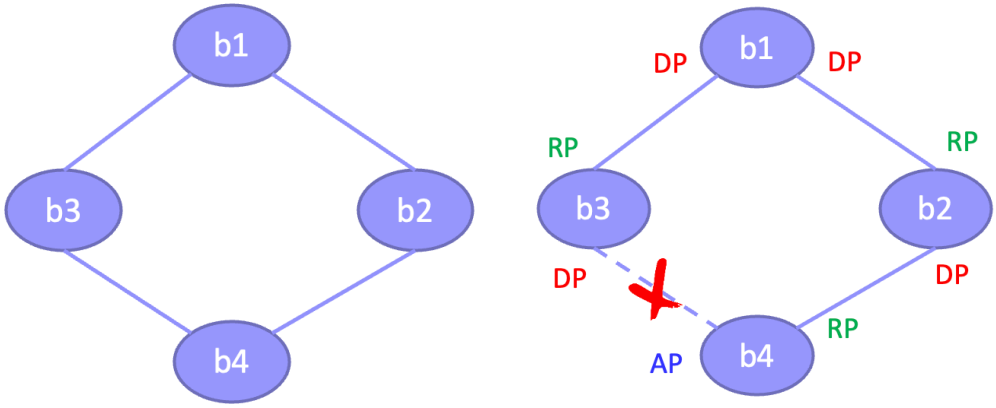
图 **5.3.** four_node_ring.py 拓扑

四节点环路拓扑如下：

```
class RingTopo(Topo):
def build(self):
```

```
    b1 = self.addHost('b1')
    b2 = self.addHost('b2')
    b3 = self.addHost('b3')
    b4 = self.addHost('b4')

    self.addLink(b1, b2)
    self.addLink(b1, b3)
    self.addLink(b2, b4)
    self.addLink(b3, b4)
```



(a) 拓扑图 (b) stp 结果

图 **5.4.** four_node_ring.py 拓扑

5.3.2 自定义拓扑

我自定义了一个七节点完全图拓扑 seven_node_complete.py，运行 stp 程序，可以得到如下结果：

```
❯ ./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.
INFO: port id: 04, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0101, ->port: 04, ->cost: 0.
INFO: port id: 05, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0101, ->port: 05, ->cost: 0.
INFO: port id: 06, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0101, ->port: 06, ->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.
INFO: port id: 04, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0201, ->port: 04, ->cost: 1.
INFO: port id: 05, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0201, ->port: 05, ->cost: 1.
INFO: port id: 06, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0201, ->port: 06, ->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.
INFO: port id: 05, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0301, ->port: 05, ->cost: 1.
INFO: port id: 06, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0301, ->port: 06, ->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: 0201, ->port: 03, ->cost: 1.
INFO: port id: 03, role: ALTERNATE.
INFO:   designated ->root: 0101, ->switch: 0301, ->port: 03, ->cost: 1.
INFO: port id: 04, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0401, ->port: 04, ->cost: 1.
INFO: port id: 05, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0401, ->port: 05, ->cost: 1.
INFO: port id: 06, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0401, ->port: 06, ->cost: 1.
```

```
NODE b5 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: 04, ->cost: 0.
INFO: port id: 02, role: ALTERNATE.
INFO:   designated ->root: 0101, ->switch: 0201, ->port: 04, ->cost: 1.
INFO: port id: 03, role: ALTERNATE.
INFO:   designated ->root: 0101, ->switch: 0301, ->port: 04, ->cost: 1.
INFO: port id: 04, role: ALTERNATE.
INFO:   designated ->root: 0101, ->switch: 0401, ->port: 04, ->cost: 1.
INFO: port id: 05, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0501, ->port: 05, ->cost: 1.
INFO: port id: 06, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0501, ->port: 06, ->cost: 1.

NODE b6 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: 05, ->cost: 0.
INFO: port id: 02, role: ALTERNATE.
INFO:   designated ->root: 0101, ->switch: 0201, ->port: 05, ->cost: 1.
INFO: port id: 03, role: ALTERNATE.
INFO:   designated ->root: 0101, ->switch: 0301, ->port: 05, ->cost: 1.
INFO: port id: 04, role: ALTERNATE.
INFO:   designated ->root: 0101, ->switch: 0401, ->port: 05, ->cost: 1.
INFO: port id: 05, role: ALTERNATE.
INFO:   designated ->root: 0101, ->switch: 0501, ->port: 05, ->cost: 1.
INFO: port id: 06, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0601, ->port: 06, ->cost: 1.
```

```
NODE b7 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: 06, ->cost: 0.
INFO: port id: 02, role: ALTERNATE.
INFO:   designated ->root: 0101, ->switch: 0201, ->port: 06, ->cost: 1.
INFO: port id: 03, role: ALTERNATE.
INFO:   designated ->root: 0101, ->switch: 0301, ->port: 06, ->cost: 1.
INFO: port id: 04, role: ALTERNATE.
INFO:   designated ->root: 0101, ->switch: 0401, ->port: 06, ->cost: 1.
INFO: port id: 05, role: ALTERNATE.
INFO:   designated ->root: 0101, ->switch: 0501, ->port: 06, ->cost: 1.
INFO: port id: 06, role: ALTERNATE.
INFO:   designated ->root: 0101, ->switch: 0601, ->port: 06, ->cost: 1.
```

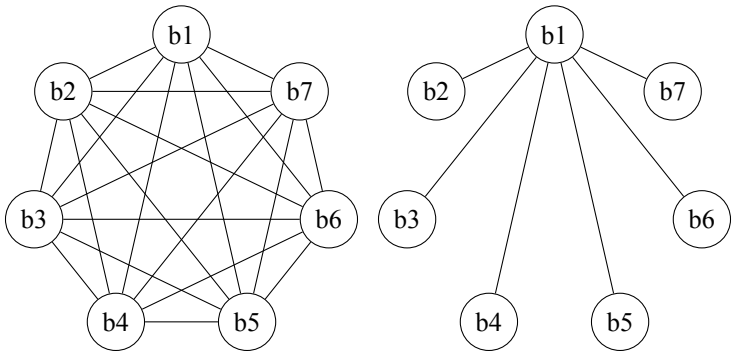
图 5.5. 七节点完全图拓扑

七节点完全图拓扑如下：

```python
class CompleteTopo(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(b1, b5)
    self.addLink(b1, b6)
    self.addLink(b1, b7)
    # code omitted
    self.addLink(b5, b6)
    self.addLink(b5, b7)
    self.addLink(b6, b7)
```



(a) 七节点完全图拓扑　　(b) stp 结果

图 5.6. 七节点完全图拓扑

可以看出算法正确地计算出了最小生成树拓扑。

5.4 实验总结

本次实验实现了生成树运行机制，对于给定拓扑，计算输出相应状态下的最小生成树拓扑。实验中，我实现了比较 config 优先级、更新本端口 config、更新节点状态、更新其余端口 config、停止 hello 定时器、转发 config 等操作，最终使用 four_node_ring.py 和自定义的七节点完全图拓扑进行了测试，得到了正确的结果。这让我更加深入地理解了生成树算法的运行机制。