计算机网络实验 6 报告

学号　2017K8009929059
姓名　於修远

生成树机制实验

一、实验内容

- 基于已有代码，实现生成树机制。
- 对于给定的四节点拓扑，计算并输出生成树拓扑。
- 构造一个节点不少于 7 个，冗余链路不少于 2 条的拓扑，计算并输出生成树拓扑。

二、实验流程

（一）生成树机制实现

1、config 读取

如下图所示，在本次实验中专门设置了一个函数来实现 config 包中有效信息到端口 config 的复制过程。该过程实质上就是各端口生成 config 包的逆过程，需要注意的是网络字节序的转换。

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

2、端口优先级比较

由于端口优先级比较的过程需要多次执行，所以在也单独设置一个函数来实现。输入两个端口，按照根节点、路径开销、上一节点、上一端口的顺序一次比较，返回 1 当且仅当输入端口 p1 的优先级小于 p2。

```
static int port_cmp(stp_port_t *p1, stp_port_t *p2) {
    if (p1->designated_root != p2->designated_root) {
        return (p1->designated_root > p2->designated_root);
    } else if (p1->designated_cost != p2->designated_cost) {
        return (p1->designated_cost > p2->designated_cost);
    } else if (p1->designated_switch != p2->designated_switch) {
        return (p1->designated_switch > p2->designated_switch);
    } else {
        return (p1->designated_port >= p2->designated_port);
    }
}
```

3、config 消息处理

如下图所示，端口在接收到 config 数据包后，首先与本端口的 config 信息进行优先级比较，其中暂时开辟了一块空间来存储转换后的 config 信息，比较完成后释放。若本端口的 config 优先级更高，则直接向来源方发送自己的 config 消息。

```
stp_port_t *temp = (stp_port_t *) malloc(sizeof(stp_port_t));
config_to_port(config, temp);
int renew = port_cmp(p, temp);
free(temp);
if (!renew) stp_port_send_config(p);
else {...}
```

而若接收到的 config 消息优先级更高，意味着端口 config 需要进行更新。在更新前，首先需要进行一次自身是否为根节点的判断，并记录判断结果。此后，需要遍历节点的所有端口，遍历过程包含两项任务：一、找出所有结点中优先度最高的节点；二、找出所有非指定端口中优先度最高的节点。前者会在后续的非指定端口向指定端口改变操作中用到，后者（若有）即为根端口。二者判断逻辑相似，只是根端口比较时需要筛选出所有非指定端口，也正是因此根端口可能为空。

```
int is_root = stp_is_root_switch(stp);
config_to_port(config, p);
stp_port_t *root_port = NULL, *max_port = NULL;
for (int i = 0; i < stp->nports; i++) {
    if (!max_port || port_cmp(max_port, &stp->ports[i]))
        max_port = &stp->ports[i];
    if (!stp_port_is_designated(&stp->ports[i])) {
        if (!root_port || port_cmp(root_port, &stp->ports[i]))
            root_port = &stp->ports[i];
    }
}
stp->root_port = root_port;
```

若不存在根端口，意味着节点为根节点，设置 stp 的相应属性；否则，用根端口的 config 信息来更新节点状态。其中，路径开销应为 config 包中的路径开销加上根端口与上一跳节点间开销。并且，若当前节点更新后不是根节点，且更新前是根节点，就需要停止自发发送以自身为根节点的 config 包的 hello 定时器。

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

最后，更新所有指定端口的 config 信息后即可从这些端口发送新的 config 信息。更新前需要注意的是一个特例：若存在一个优先级高于网段内所有其他端口的非指定端口，则将其设为指定端口。具体实现上，只需判断先前得到的网段内优先级最高端口是否为指定端口，若否则修改为指定端口，从而减少遍历开销。需要注意的是，不能对新的根端口进行向指定端口的转化，否则会导致错误。

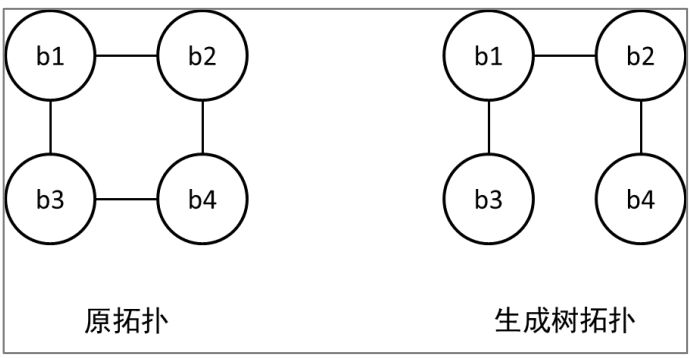
```
if ((max_port != root_port) && !stp_port_is_designated(max_port)) {
    max_port->designated_switch = stp->switch_id;
    max_port->designated_port = p->port_id;
}
for (int i = 0; i < stp->nports; i++) {
    if (stp_port_is_designated(&stp->ports[i])) {
        stp->ports[i].designated_root = stp->designated_root;
        stp->ports[i].designated_cost = stp->root_path_cost;
    }
}
stp_send_config(stp);
```

（二）四节点生成树构造

构造四节点拓扑需要用到课程提供的` four_node_ring.py `脚本，在脚本中直接令四个节点分别执行编译得到的 stp 程序，并将输出重定向至` b*-output.txt `文件。等待约 30s 后使用` pkill -SIGTERM stp `命令中止所有节点的 stp 程序，并打印节点及内部各端口状态。最后执行` dump_output.sh `脚本，整合并输出所有四个节点的状态至` four_result.txt `文件，如下图所示。

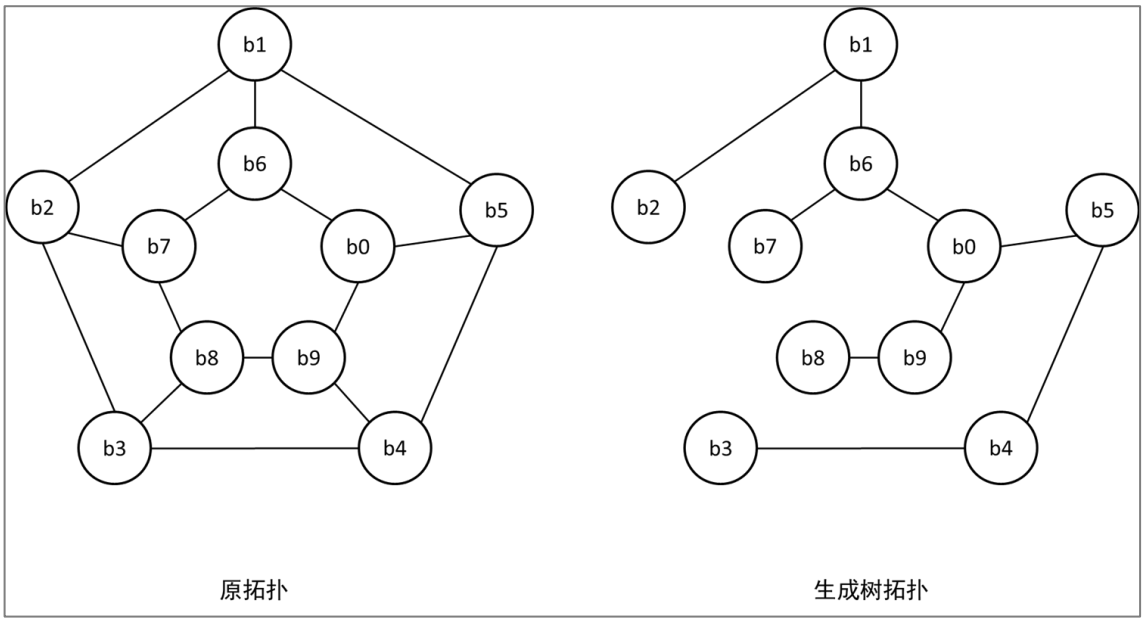
```
NODE b1 dumps:
INFO: this switch is root.
INFO: port id: 01, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0101, ->port: 01, ->cost: 0.
INFO: port id: 02, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0101, ->port: 02, ->cost: 0.

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.
```

对应的链路拓扑和生成树拓扑分别如下左右图所示，符合设计要求。



（三）十节点生成树构造

构造十节点拓扑需要的` ten_node_ring.py `脚本根据` four_node_ring.py `修改得到，改动仅限于节点输目的增加和链路的改动，故不赘述。在脚本中直接令十个节点（b0-b9）分别执行编译得到的 stp 程序，并将输出重定向至` b*-output.txt `文件。等待一段时间后使用` pkill -SIGTERM stp `命令中止所有节点的 stp 程序，并打印节点及内部各端口状态。最后执行` dump_output.sh `脚本，整合并输出所有节点的状态至` ten_result.txt `文件。具体拓扑构建及脚本输出内容可见随代码压缩在内的文档，这里限于篇幅略去。对应的链路拓扑和生成树拓扑分别如下左右图所示，共有 6 条冗余链路，符合设计要求。



三、结果分析

综合上述实验设计和实验结果，可以发现生成树机制运行正常。生成树算法的重点在于节点内部端口更新到节点更新再到端口更新的分支逻辑，应当充分考虑各种情况，否则容易引起错误。例如在考虑将非指定端口更新为指定端口的逻辑时，一度忽略了根端口的情况，而根端口往往会满足对应的更新条件，从而造成最终生成的拓扑中没有一个节点拥有根节点。

此外，由于没有考虑拓扑变动下的生成树重构等问题，虽然能正确构建生成树拓扑模型，但部分执行效率可能存在缺陷。