06-生成树机制实验报告

陈彦帆 2018K8009918002

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。

（3）在 four_node_ring.py 基础上，添加两个端节点，把第 05 次实验的交换机转发代码与本实验代码结合，试着构建生成树之后进行转发表学习和数据包转发。

2、实验流程

（1）实现生成树运行机制:完成下列函数。

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

（2）在给定 4 节点拓扑下运行生成树机制:

编译(make)。

修改脚本（见 four_node_ring.py）并执行。

（3）在 7 节点拓扑下运行生成树机制:

构建 7 节点拓扑（见 seven_node_ring.py)。

执行脚本。

（4）将构建生成树与转发数据包相结合：

将 05 次实验的代码合并进本次实验。

修改以下函数，使其支持转发数据包：

void handle_packet(iface_info_t *iface, char *packet, int len)

void broadcast_packet(iface_info_t *iface, const char *packet, int len)

修改脚本 four_node_ring.py，添加 h1 节点（连接 b1），h2 节点（连接 b4），在运行 stp 程序后一段时间后让 h1 ping h2，并打印结果。

net.start()

```
sleep(30)
print(h1.cmd('ping -c 2 '+ h2.IP()))
...
```

执行脚本。

（5）完成思考题

3、设计与实现

（1）stp 程序运行机制和流程

机制：

每个节点包括若干端口，每个网段包括若干一跳可达的端口。把每个节点优先级最高的端口称为根端口，把每个网段优先级最高的端口称为指定端口。每个端口记录并维护它认为的本网段优先级最高的 config 包，指定端口每隔一段时间向本网段其他端口发送 config 包。

config 包记录的 config 信息包括：根节点 ID，本网段到根节点的开销，本网段的指定端口所在节点的 ID，本网段指定端口的 ID。

生成树机制收敛后，每个网段内所有端口存储的 config 包的信息都相同，每个节点认为的根节点都相同。只有根端口和指定端口可以转发数据包。

同网段两个端口 config 包的优先级比较如下：

根节点 ID 小的优先级高；若相同，到根节点开销小的优先级高；若相同，到根节点上一跳节点（即该网段的指定端口所在节点）ID 小的优先级高；若相同，到根节点上一跳端口（即该网段的指定端口）ID 小的优先级高。

其中，每个节点（到根节点）的开销等于该节点根端口 config 的开销加上根端口所在网段的路径开销，也等于该节点任一指定端口 config 的开销，等于该节点任一指定端口所在网段的 config 开销。

其中端口 config 比较的代码实现如下：

```
#define COMPARE_RETURN(x,y) if((x)<(y)) return 1; else if((x)>(y)) return -1
int compare_ports(stp_port_t *p1, stp_port_t *p2)
{
    COMPARE_RETURN(p1->designated_root,p2->designated_root);
    COMPARE_RETURN(p1->designated_cost,p2->designated_cost);
    COMPARE_RETURN(p1->designated_switch,p2->designated_switch);
    COMPARE_RETURN(p1->designated_port,p2->designated_port);
    return 0;
}
```

流程：

初始化后，每个节点的 stp 程序分成三个线程。线程 1 每隔一个 hello time，遍历节点的每个指定端口，向端口所处网段发送记录的 config 包。线程 2 监听该节点收到的包，对于每个端口收到的包，若其为 stp 协议 config 包，根据其内容更新本节点及其各个端口状态；否则，根据规则向本节点其它端口转发。线程 3 每隔一段时间扫描 mac-端口映射表 map，删除过期的条目。在访问 stp 结构或映射表 map 结构时，需要加锁实现互斥。

（2）处理每个端口收到的 config 包：

经过仔细分析，本实现略微简化了讲义上的流程，结果是等价的。

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

参数 stp 为当前节点的结构，p 为当前端口的结构，config 为当前端口收到的 config 结构。

函数体伪代码如下：

```
if(config > p.config):                  // 比较方法见(1)小节
    p.config = config                   // 本端口为非指定端口
    //要选出非指定端口中优先级最高的端口作为根端口，只需和原有根端口比较
    if(stp.root == null or p.config > stp.root.config):
        stp.root_port = p
        stp.root_id = p.config.root_id
        stp.cost = p.config.cost + p.path_cost
        for(s in stp.port and s!=p):
        //无须显式区分 s 是否为指定端口，假设 s 为指定端口，设置 s 的临时 config t
        //如下，若临时 config 优先级更高，则替换掉它原来的 config
            configtype t
            t.root_id = p.root_id, t.cost = stp.cost, t.switch_id = stp.id,
t.port_id = s.id
            if(t>s.config):
                s.config = t
//如果 if 条件不成立，则当前端口状态不变，自然也不必更新当前节点以及其他端口的状态
```

在本函数中无须显式关闭或打开定时器，定时器线程会自己判断当前端口的状态来决定是否发送。

具体实现的 C 代码如下：

```
static void stp_handle_config_packet(stp_t *stp, stp_port_t *p,
        struct stp_config *config)
{
    stp_port_t tmp;
    stp_port_t *t = &tmp;
    t->designated_root = ntohll(config->root_id);
    t->designated_cost = ntohl(config->root_path_cost);
    t->designated_switch = ntohll(config->switch_id);
    t->designated_port = ntohs(config->port_id);
    if(compare_ports(t,p)>0) {
    // means that p is a non-designated_port
        log(DEBUG, "port %02d conf has changed to %04lx:%02d\n",p->port_id & 0xFF,t
->designated_switch& 0xFFFF,t->designated_port & 0xFF);
        p->designated_root = t->designated_root;
        p->designated_cost = t->designated_cost;
        p->designated_switch = t->designated_switch;
        p->designated_port = t->designated_port;

        if(stp->root_port==NULL || compare_ports(t,stp->root_port)>0){
            stp->root_port = p;
            stp->designated_root = t->designated_root;
            stp->root_path_cost = t->designated_cost += p->path_cost;
            t->designated_switch = stp->switch_id;
            stp_port_t *s;
            for(int i=0;i<stp->nports;i++){
                s = &stp->ports[i];
                t->designated_port = s->port_id;
                if(t!=p && compare_ports(t,s)>0){
                    log(DEBUG, "port %02d has changed to %04lx:%02d followed\n",s->
port_id & 0xFF,t->designated_switch& 0xFFFF,t->designated_port & 0xFF);
                    s->designated_root = t->designated_root;
                    s->designated_cost = t->designated_cost;
                    s->designated_switch = t->designated_switch;
                    s->designated_port = t->designated_port;
                }
            }
        }
    }
    // else the stp state stays the same
}
```

（3）将构建生成树与转发数据包相结合

称根端口或指定端口为非阻塞端口，其它端口为阻塞端口。

修改 handle_packet 函数：

若收到的包为 stp config 包，则执行 stp_port_handle_packet。

若收到的包不是 stp config 包：若当前端口为阻塞端口，什么也不做。

否则，从转发表 map 查找目的 mac 地址对应的端口，若找到且该端口为非阻塞端口，则发往相应端口，否则，向该节点所有其它的非阻塞端口广播该数据包。将发送方 mac 地址-当前端口插入转发表 map。

具体实现如下：

```
void handle_packet(iface_info_t *iface, char *packet, int len)
{
    struct ether_header *eh = (struct ether_header *)packet;
    stp_port_t *p = iface->port;
    pthread_mutex_lock(&p->stp->lock);

    if (memcmp(eh->ether_dhost, eth_stp_addr, sizeof(*eth_stp_addr))!=0) {
        if(port_is_valid(p)){
            log(DEBUG, "port %02d has recv\n",p->port_id & 0xFF);
            struct ether_header *eh = (struct ether_header *)packet;
            iface_info_t* dest = lookup_port(eh->ether_dhost);
            if(dest != NULL && port_is_valid(dest->port)){
                log(DEBUG, "send from port %02d\n",dest->port->port_id & 0xFF);
                iface_send_packet(dest, packet, len);
            }
            else {
                log(DEBUG, "broatcast.\n");
                broadcast_packet(iface, packet, len);
            }
            insert_mac_port(eh->ether_shost, iface);
        }
    }
    else {
        stp_port_handle_packet(p, packet, len);
    }
    pthread_mutex_unlock(&p->stp->lock);
    free(packet);
}
```

4、结果与讨论

（1）在给定 4 节点拓扑下运行生成树机制：

运行前节点拓扑结构如图 1。运行后节点及端口的状态如图 2。

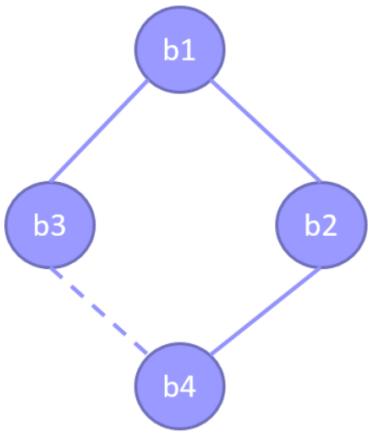

图 1

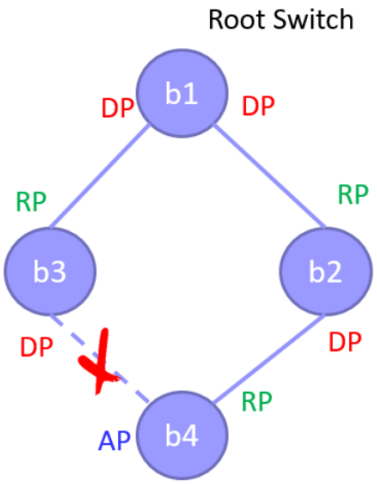

图 2

运行过程中程序的 INFO 打印结果如下，与图 2 相符，节点拓扑确实形成了一棵生成树。

```
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.
```

（2）在 7 节点拓扑下运行生成树机制

运行前节点拓扑结构如图 3。运行后节点及端口的状态如图 4。

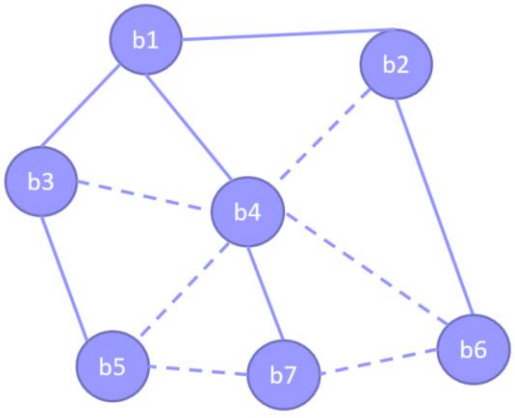

图 3

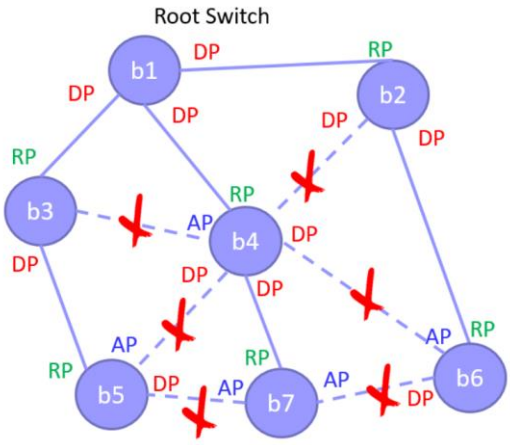

图 4

运行过程中程序的 INFO 打印结果如下，与图 4 相符，节点拓扑确实形成了一棵生成树。

```
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.
INFO: port id: 03, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0201, ->port: 03, ->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: 0201, ->port: 02, ->cost: 1.
INFO: port id: 03, role: ALTERNATE.
INFO:   designated ->root: 0101, ->switch: 0301, ->port: 02, ->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: 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: 0401, ->port: 04, ->cost: 1.
INFO: port id: 03, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0501, ->port: 03, ->cost: 2.

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: 0201, ->port: 03, ->cost: 1.
INFO: port id: 02, role: ALTERNATE.
INFO:   designated ->root: 0101, ->switch: 0401, ->port: 05, ->cost: 1.
INFO: port id: 03, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0601, ->port: 03, ->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: 0401, ->port: 06, ->cost: 1.
INFO: port id: 02, role: ALTERNATE.
INFO:   designated ->root: 0101, ->switch: 0501, ->port: 03, ->cost: 2.
INFO: port id: 03, role: ALTERNATE.
INFO:   designated ->root: 0101, ->switch: 0601, ->port: 03, ->cost: 2.
```

（3）将构建生成树与转发数据包相结合：

在 4 节点拓扑的基础上加入两个 host 节点，如图 5。运行生成树后，生成的拓扑关系如图 6。

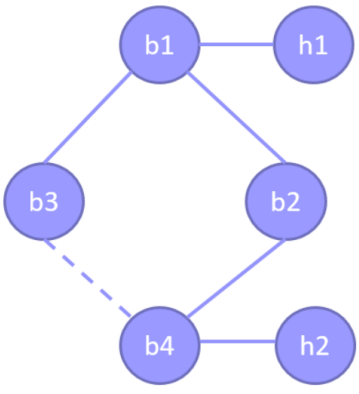

图 5

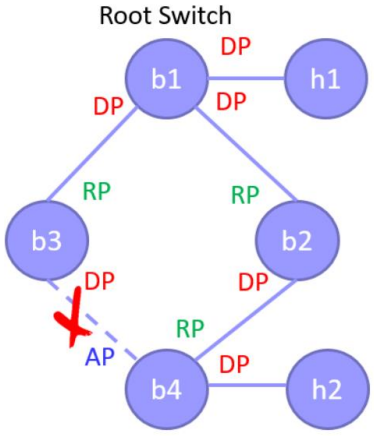

图 6

ping 命令的执行结果如下。显示 h1 和 h2 连通。

```
PING 10.0.0.6 (10.0.0.6) 56(84) bytes of data.
64 bytes from 10.0.0.6: icmp_seq=1 ttl=64 time=0.956 ms
64 bytes from 10.0.0.6: icmp_seq=2 ttl=64 time=0.600 ms

--- 10.0.0.6 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1000ms
rtt min/avg/max/mdev = 0.600/0.778/0.956/0.178 ms
```

打印 stp 程序的转发信息，可以追溯每个端口转发的流程，可以看出没有形成广播风暴。篇幅关系，以下仅给出 b1 节点的打印内容。可以看出 h1 ping 了 2 次，但 b1 只进行了一次广播，之后的发送为定向发送，学习成功。

```
NODE b1 dumps:
DEBUG: find the following interfaces:  b1-eth0 b1-eth1 b1-eth2.
DEBUG: port 03 has recv
DEBUG: broatcast.
DEBUG: bsend from port 01
DEBUG: bsend from port 02
DEBUG: port 01 has recv
DEBUG: send from port 03
DEBUG: port 03 has recv
DEBUG: send from port 01
DEBUG: port 01 has recv
DEBUG: send from port 03
DEBUG: port 03 has recv
DEBUG: send from port 01
DEBUG: port 01 has recv
DEBUG: send from port 03
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.
```

5、思考题

（1）调研说明标准生成树协议中，如何处理网络拓扑变动的情况：当节点加入时？当节点离开时？

STP 协议包格式称为 BPDU（Bridge Protocol Data Unit）。当生成树选举完成后，根节点继续周期性发送 config BPDU 信息，非指定端口会不断的侦听对端发送的 BPDU，如果超过一定时间内没有收到 config BPDU 信息，非指定端口认为网络发生变化，网络将会重新进行收敛计算。

当节点删除时，某些非指定端口没有收到 config BPDU 信息，将产生 TCN BPDU，当新节点加入时也会产生 TCN BPDU。TCN BPDU 会被发往根节点。当根节点接收到 TCN BPDU 后（发现网路发生变化），使用 TC BPDU 通告整个网络，让网络中的所有交换机重新进行网络收敛。

（2）调研说明标准生成树协议是如何在构建生成树过程中保持网络连通的。

用不同的状态来标记每个端口，不同状态下允许不同的功能（Blocking, Listening, Learning, Forwarding 等）

Blocking:不会学习 MAC 地址，不能转发数据帧

Listening：不会学习 MAC 地址，不能转发数据帧（过渡状态）

Learning：学习 MAC 地址，不能转发数据帧

Forwarding：学习 MAC 地址，转发数据帧

Disabled：该端口被管理员关闭

选举开始时，端口处于 Blocking 状态。被选择成根端口或指定端口先进入 Listening，再经过 Learning 状态，最后进入 Forwarding 状态，其他端口(没有被选举为 RP 或 DP)继续停留在 blocking 状态。

选举开始时，每个端口认为自己是指定端口。而指定端口最终会进入 Forwarding 状态。选举过程中，虽然指定端口或根端口可能发生变化，但这些端口保持连通，会逐渐进入 Forwarding 状态，即使此时生成树尚未收敛，仍然可能转发数据报。同时，非指定端口或根端口会逐渐进入 Blocking 状态，不会转发数据报，避免在环路下转发，产生广播风暴。

（3）实验中的生成树机制效率较低，调研说明快速生成树机制的原理

RSTP 只有三种端口状态：Discarding、Learning 和 Forwarding，增加了 2 种端口角色：Backup 端口和边缘端口。RSTP 的协商过程和 STP 相同，改进之处在于节点状态改变到 Forwarding 更快。

RSTP 中收敛时间的优化：

①P/A 机制：可以让交换机的 RP 和 DP 的互联接口快速进入转发状态。

②直连故障：AP 口变为 RP 并快速进入转发状态，不需要 30s 延时。

③次优场景：AP 口收到次优的 RST BPDU 包后会马上变为 DP 口，并向该端口发送最优的 RST BPDU 包。

④非直连链路故障：连续丢失 3 个 RST BPDU 包，端口角色就需切换，最长时间为 6s。RSTP 中 TC 置位的 RST BPDU 包所有桥设备都可以发送，连续发送 4s（TC while 时间）。

其中,P/A 机制即 Proposal/Agreement 机制。目的是使一个指定端口尽快进入 Forwarding 状态。其过程的完成根据以下几个端口变量：

①Proposing。当一个指定端口处于 Discarding 或 Learning 状态的时候，该变量置位。并向下游交换传递 Proposal 位被置位的 BPDU。

②Proposed。当下游交设备端口收到对端的指定端口发来的携带 Proposal 的 BPDU 的时候。该变量置位。该变量指示上游网段的指定端口希望进入 Forwarding 状态。

③sync。当 Proposed 被设置以后，收到 Proposal 置位信息的根端口会依次为自己的其他端口置位 sync 变量。如果端口是非边缘的指定端口则会进入 Discarding 状态。

④synced。当其它端口完成转到 Discarding 后，会设置自己的 synced 变量（Alternate、Backup 和边缘端口会马上设置该变量）。根端口监视其他端口的 synced，当所有其他端口的 synced 全被设置，根端口会设置自己的 synced，然后传回 BPDU，其中 Agreement 位被置位。

⑤agreed。当指定端口接收到一个 BPDU 时，如果该 BPDU 中的 Agreement 位被置位且端口角色定义是“根端口”，该变量被设置。Agreed 变量一旦被置位，指定端口马上转入 Forwarding 状态。

RSTP 比较明确区分了端口状态与端口角色，收敛时更多的是依赖于端口角色的切换。STP 端口状态的切换必须被动等待时间的超时。而 RSTP 端口状态的切换却是一种主动的协商。STP 中的非根网桥只能被动中继 BPDU。而 RSTP 中的非根网桥对 BPDU 的中继具有一定的主动性。