实验报告

生成树机制实验

一、实验内容

了解生成树的拓扑结构和生成树机制的基本原理，了解如何在网络中实现唯一的、基于优先级的生成树。具体实现基于 Config 消息的生成树算法，掌握处理 Config 消息的流程。在给定拓扑以及自己构造的更复杂拓扑下验证生成树算法功能。试着添加上次实验的交换机和转发表功能，实现在构建生成树之后进行转发表学习和数据包转发。最后，调研了解标准生成树协议中如何处理动态拓扑、如何在构建生成树过程中保持网络连通、如何快速构建生成树等原理。

二、实验流程

1. 根据生成树原理，实现生成树运行机制。
2. 利用给定的 4 节点拓扑，输出其最小生成树拓扑结构。
3. 构造 7 个节点的复杂拓扑，输出其最小生成树拓扑结构。
4. 增加主机节点，添加交换机功能，实现构建生成树之后进行转发表学习和数据包转发。

三、实验结果及分析

（一）实现生成树运行机制

1、比较 Config 优先级

在端口收到 Config 消息后，首先将其与本端口 Config 进行优先级比较。我们首先实现两个优先级比较函数，一是本地端口与收到的 Config 消息比较，注意收到的 Config 消息中的数据需要先转为本地字节序再进行比较；二是两个本地端口间进行比较。比较函数具体实现讲义已经讲的很清晰了，这里不再赘述。将函数封装如下，便于之后调用。

```
// return 1 when packet config has higher priority, return 0 when port config has higher priority
static int compare_packet_and_port(struct stp_config *config, stp_port_t *p)
```

```
// return 1 when p has higher priority, return 0 when q has higher priority
static int compare_port_and_port(stp_port_t *p, stp_port_t *q)
```

于是在接收到 Config 消息时，先进行比较。如果收到的 Config 优先级更高，就把本端口的 Config 替换为收到

的 Config 消息，之后再进行后续处理。如果本地 Config 优先级更高，不做更改，若该端口是指定端口，发送 Config 消息。

```
if (compare_packet_and_port(config, p)) {
    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);
```

```
else {
    if (stp_port_is_designated(p)) {
        stp_port_send_config(p);
    }
}
```

2、更新节点状态

首先遍历所有端口，尝试找到根端口。如果存在根端口，则该节点为非根节点，选择通过 root_port 连接到根节点，更新节点认定的根、到根节点的路径开销、根端口。

如果不存在根端口，则该节点为根节点。根端口为空，根节点指向自己，距离为 0。

最后需要判断节点性质是否发生变化，如果节点由根节点变为非根节点，停止 hello 定时器。

```
int is_root_before = stp_is_root_switch(stp);
int is_root_after;
stp_port_t *root_port = find_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;
}
is_root_after = stp_is_root_switch(stp);
if (is_root_before && !is_root_after)
    stp_stop_timer(&stp->hello_timer);
```

3、更新剩余端口的 Config

最后更新该节点的其他端口信息。对于非指定端口，如果其 Config 较网段内其他端口优先级更高，那么该端口成为指定端口。具体来说更新前该非指定端口的 Config 是网段内最高优先级 Config，如果节点 Config 信息继承到该端口后的优先级高于更新前，那么就把该端口更新为指定端口。

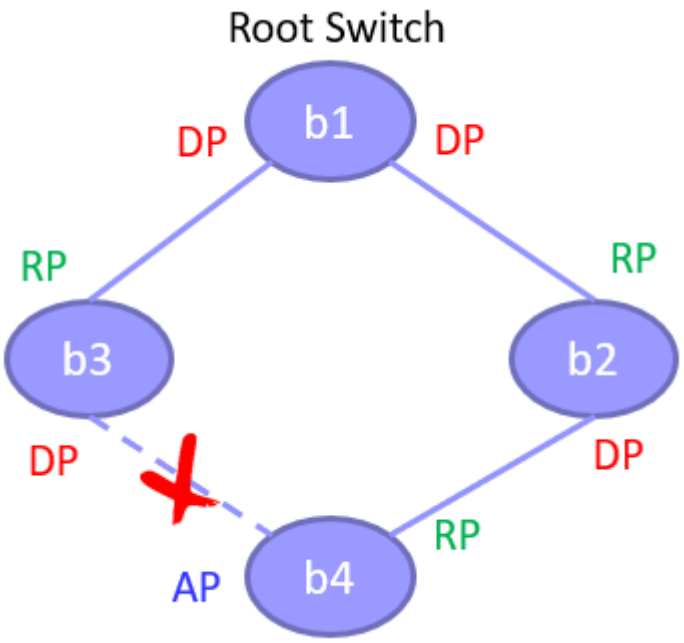
对于所有指定端口，需要更新其认为的根节点和路径开销。

最后，将更新后的 Config 从每个指定端口转发出去。

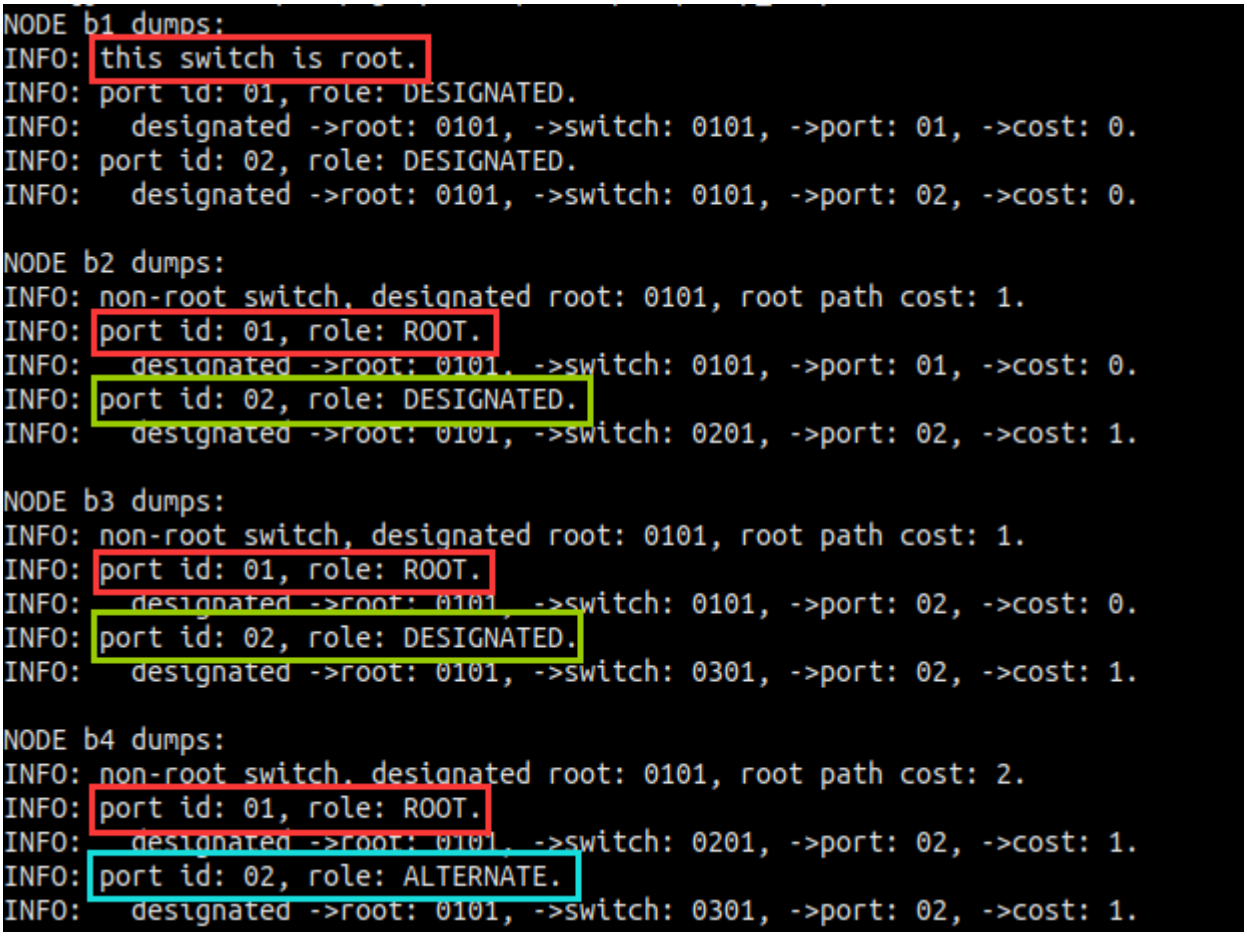
```
for (int i = 0; i < stp->nports; i++) {
    stp_port_t *q = &stp->ports[i];
    if (!stp_port_is_designated(q)) {
        stp_port_t *temp = (stp_port_t*)malloc(sizeof(stp_port_t));
        memset(temp, 0, sizeof(stp_port_t));
        temp->designated_root = stp->designated_root;
        temp->designated_cost = stp->root_path_cost;
        temp->designated_switch = q->designated_switch;
        temp->designated_port = q->designated_port;
        if (compare_port_and_port(temp, q)) {
            q->designated_switch = stp->switch_id;
            q->designated_port = q->port_id;
        }
    }
}
for (int i = 0; i < stp->nports; i++) {
    stp_port_t *q = &stp->ports[i];
    if (stp_port_is_designated(q)) {
        q->designated_root = stp->designated_root;
        q->designated_cost = stp->root_path_cost;
    }
}
stp_send_config(stp);
```

（二）4 节点拓扑的最小生成树

给定的 4 节点拓扑结构如下图所示：



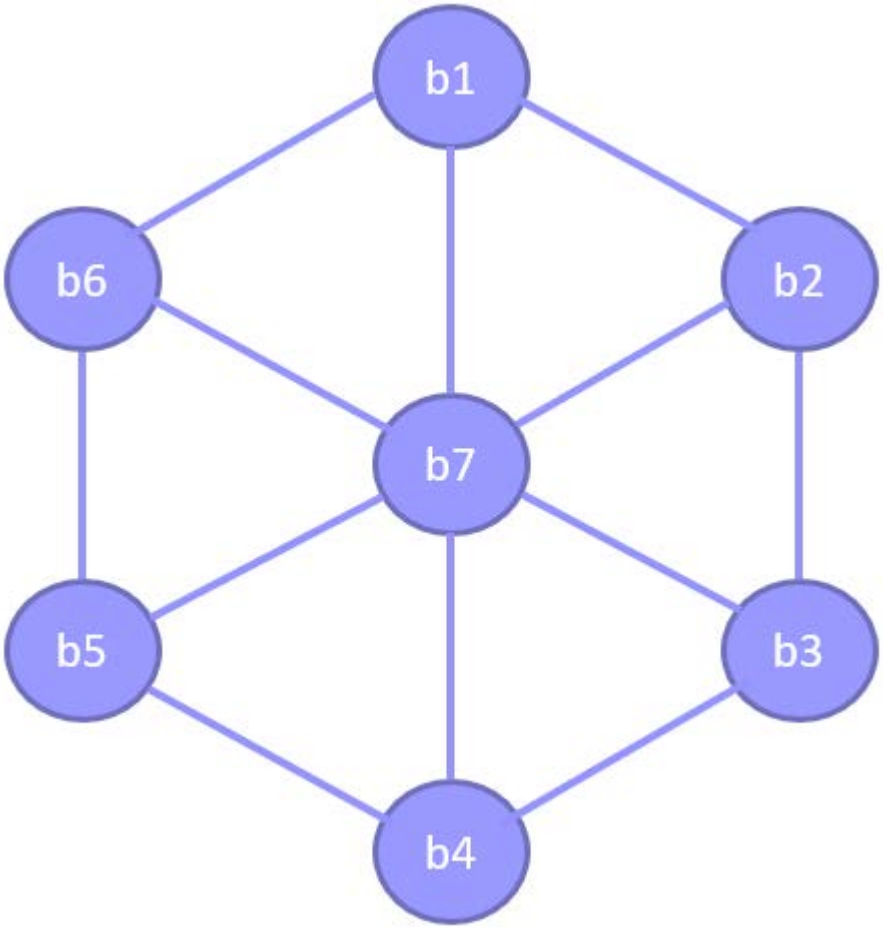
使用 stp 程序计算输出最小生成树拓扑，结果如下：



可以看到 b1 节点为根节点，其两个端口都为指定端口。b4 节点的 2 端口为 AP 端口，不参与构建生成树拓扑。其他节点和端口也都符合预期的拓扑结构。由此可以看出该生成树算法功能正确。

（三）7 节点复杂拓扑的最小生成树

构造一个含有 7 个节点的复杂拓扑结构，如下图所示：



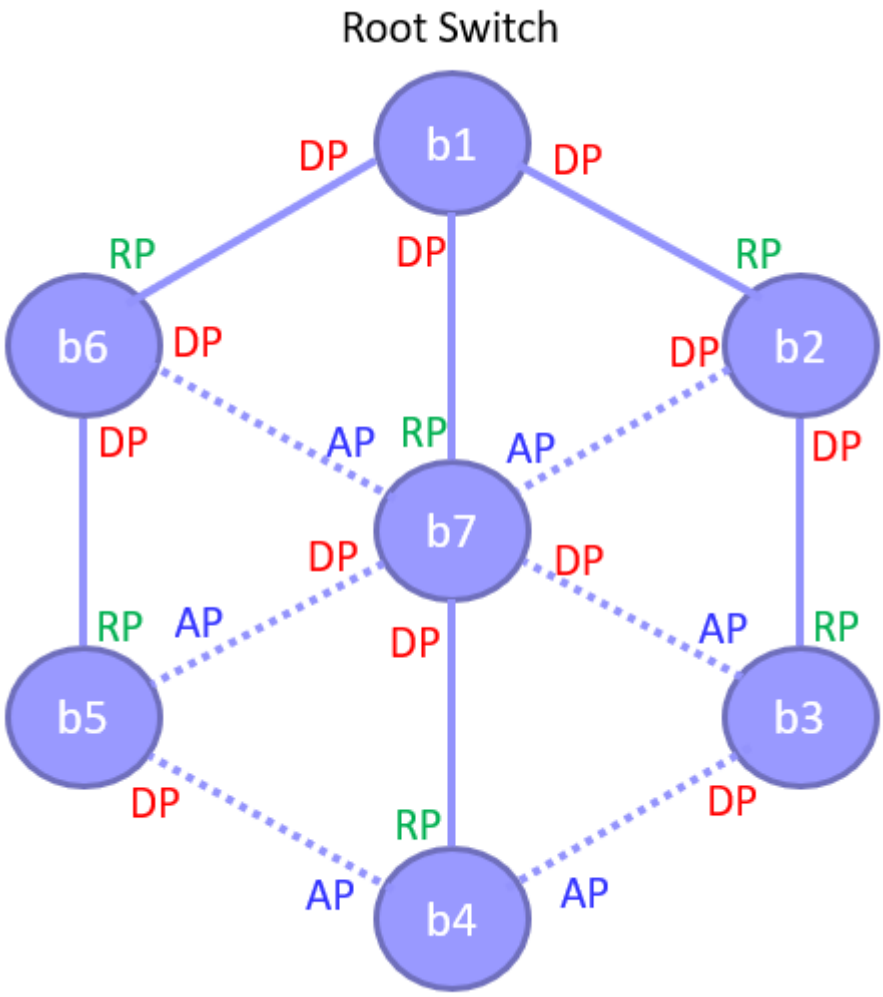
使用 stp 程序计算输出最小生成树拓扑，结果如下：

```
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: 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: 0301, ->port: 02, ->cost: 2.
INFO: port id: 03, role: ALTERNATE.
INFO:   designated ->root: 0101, ->switch: 0701, ->port: 03, ->cost: 1.

NODE b4 dumps:
INFO: non-root switch, designated root: 0101, root path cost: 2.
INFO: port id: 01, role: ALTERNATE.
INFO:   designated ->root: 0101, ->switch: 0301, ->port: 02, ->cost: 2.
INFO: port id: 02, role: ALTERNATE.
INFO:   designated ->root: 0101, ->switch: 0501, ->port: 01, ->cost: 2.
INFO: port id: 03, role: ROOT.
INFO:   designated ->root: 0101, ->switch: 0701, ->port: 04, ->cost: 1.
```

```
NODE b5 dumps:
INFO: non-root switch, designated root: 0101, root path cost: 2.
INFO: port id: 01, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0501, ->port: 01, ->cost: 2.
INFO: port id: 02, role: ROOT.
INFO:   designated ->root: 0101, ->switch: 0602, ->port: 02, ->cost: 1.
INFO: port id: 03, role: ALTERNATE.
INFO:   designated ->root: 0101, ->switch: 0701, ->port: 05, ->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: 02, ->cost: 0.
INFO: port id: 02, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0602, ->port: 02, ->cost: 1.
INFO: port id: 03, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0602, ->port: 03, ->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: 03, ->cost: 0.
INFO: port id: 02, role: ALTERNATE.
INFO:   designated ->root: 0101, ->switch: 0201, ->port: 03, ->cost: 1.
INFO: port id: 03, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0701, ->port: 03, ->cost: 1.
INFO: port id: 04, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0701, ->port: 04, ->cost: 1.
INFO: port id: 05, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0701, ->port: 05, ->cost: 1.
INFO: port id: 06, role: ALTERNATE.
INFO:   designated ->root: 0101, ->switch: 0602, ->port: 03, ->cost: 1.
```

根据以上信息，画出生成树拓扑结构为。



经过分析，该结构符合预期，满足生成树要求。由此进一步证明我们的生成树算法功能正确。

（四）结合数据包转发实验

1、添加交换机和转发表

将上一实验中的交换机和转发表功能添加到本实验中，对于非 Config 的数据包，进行转发处理。

处理数据包的逻辑改为下图，对于非 stp 配置包进行转发。首先检查收到数据包的端口，如果不是指定端口也不是根端口，那么直接丢弃该包；否则进行下一步处理。如果在转发表中查到目的地址，检查目的端口，如果是指定端口或者根端口，转发该数据包。如果没有查到目的地址，广播该数据包。广播函数也需要进行一些更改，只向指定端口和根端口发送广播。

```c
if (memcmp(eh->ether_dhost, eth_stp_addr, sizeof(*eth_stp_addr))) {
    // TODO: forward this packet, if this lab has merged with 05-switch.
    if (stp_port_is_designated(iface->port) || stp_port_is_root(iface->port)) {
        //log(DEBUG, "the dst mac address is " ETHER_STRING ".\n", ETHER_FMT(eh->ether_dhost));
        // lookup dest
        iface_info_t *dest_iface = lookup_port(eh->ether_dhost);
        if (dest_iface) {
            if (stp_port_is_designated(dest_iface->port) || stp_port_is_root(dest_iface->port))
                iface_send_packet(dest_iface, packet, len);
        }
        else {
            broadcast_packet(iface, packet, len);
        }
        // update source
        insert_mac_port(eh->ether_shost, iface);
    }

    free(packet);
    return;
}
```

```c
void broadcast_packet(iface_info_t *iface, const char *packet, int len)
{
    iface_info_t *ifc = NULL;
    list_for_each_entry(ifc, &instance->iface_list, list) {
        if (ifc != iface && (stp_port_is_designated(ifc->port) || stp_port_is_root(ifc->port))) {
            iface_send_packet(ifc, packet, len);
        }
    }
}
```

2、测试网络连通

在给定的 4 节点拓扑中，增加 2 个主机节点。h1 连接 b3，h2 连接 b4。在主机 h1 上 ping 主机 h2。

"Node: h1"

```
root@joker-linux:/mnt/hgfs/share/06-stp-merge# ping -c 4 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=2.76 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.602 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=1.06 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=1.64 ms

--- 10.0.0.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3013ms
rtt min/avg/max/mdev = 0.602/1.515/2.758/0.806 ms
```

如上图所示，显示 ping 成功。

四、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？

（1）当网络拓扑变动时，将会使用另一种生成树协议报文。我们先回顾一下 STP 报文，其中有两个成员：BPDU Type 与 Flags。BPDU Type 标记协议报文的类型，配置 BPDU 类型为 0x00，TCN BPDU 类型为 0x80。我们实验中用到的为配置 BPDU，它的作用为就是传递 Config 信息，建立生成树。而 TCN BPDU 则用于处理拓扑变动。Flags 由 8 位组成，最低位为 TC 标志位，最高位为 TCA 标志位，其他 6 位保留。

（2）当有端口断开或新网桥加入时，拓扑发生了改变，就会使用到 TCN BPDU 报文，目的是让 STP 能快速的收敛。拓扑改变时候 STP 处理步骤为：

1. 从该发生变化的交换机的根端口发送 TCN BPDU 报文，报文中 flags 的 TC 位置为 1。

2. 上游交换机收到 TCN BPDU 报文，会将下一个配置 BPDU 报文中的 TCA 置为 1，发送给下游交换机。

3. 重复 1, 2 步骤，直到根节点收到 TCN BPDU 报文，并将下一个配置 BPDU 报文中的 TCA 置为 1，发送给下游所有的交换机。各节点收到 TCA 置为 1 的配置 BPDU 报文后，会将 MAC 地址老化时间缩短为 15 秒。

这样就能使拓扑改变后，STP 能快速重新收敛。

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

标准生成树协议赋予端口 5 种状态。

状态名称	状态描述
禁用（Disable）	不能收发 BPDU，也不能收发数据帧。
阻塞（Blocking）	不能发送 BPDU，但是会持续侦听 BPDU。不能收发数据帧。
侦听（Listening）	可以收发 BPDU，但不能收发数据帧，也不能进行 MAC 地址学习。
学习（Learning）	会侦听业务数据帧，但不能转发数据帧，并可以学习 MAC。
转发（Forwarding）	正常收发数据帧，也会进行 BPDU 处理，只有根端口或指定端口才能进行转发态。

当交换机的一个端口被激活后，该端口会从禁用状态自动进入阻塞状态。阻塞状态的端口如果被选举为根端口或者指定端口，那么它将从阻塞状态进入侦听状态。

端口将在侦听状态停留 15s（转发延迟时间），期间收发 BPDU，让 STP 完成整个网络的计算。

在 15s 后，生成树构建完成，如果该端口还是根端口或指定端口，就会进入学习状态。这时将侦听数据帧，根据 MAC 地址学习转发表。

学习状态也将停留 15s，之后端口进入转发状态。此时正常收发数据帧，也处理 BPDU 帧。

综上，标准生成树协议是通过赋予端口不同状态，让其在不同状态下分别处理构建生成树和转发数据包功能，由此保证构建生成树过程中网络连通。

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

STP 的收敛速度较慢，因此之后出现了 RSTP 快速生成树协议。RSTP 中将禁用（Disabled）、阻塞（Blocking）、侦听（Listening）这 3 个状态合并为丢弃（Discarding）状态。

RSTP 端口会在 Discarding 状态完成角色的确定，当端口确定为根端口或指定端口后，经过 Forward Delay 时间会进入 Learning 状态，比 STP 就少一个 Forward Delay 时间。不是根端口或指定端口就会维持 Discarding 状态，减少了状态转换。

RSTP 还加入了 P/A 快速收敛机制。P/A 机制是指定端口可以通过与对端网桥进行一次握手，即可快速直接从 Discarding 转到 Forwarding 状态。

当指定端口处于 Discarding 和 Learning 状态时，所发送的 BPDU 中的 Proposal 位将被置为 1。收到 Proposal 置位的 RST BPDU 报文后，交换机会判断接收端口是否为根端口，如果是根端口，会进行同步过程。

完成同步过程后，根端口进入转发状态并从根端口回发 Agreement 置为 1 的 RST BPDU 报文，内容复制收到的 Proposal 置为 1 的 RST BPDU 报文，唯一不同的就是 Flags 字段的 Agreement 也置为 1。收到 Agreement 置 1 的 BPDU 报文后，该接口会立即进入转发状态。

另外，RSTP 对拓扑改变处理也进行了优化，交换机收到 TCA 置位的 BPDU 后不需要将 MAC 地址老化时间设置为 Forward Delay，而是直接清除端口 MAC 地址，重新进行学习，从而实现更快的收敛。

五、实验总结

通过本次实验，我对生成树的拓扑结构和生成树机制的基本原理有了一定的了解。首先，我学到了如何在网络中实现唯一的、基于优先级的生成树。然后，我掌握到处理 Config 消息的流程。之后，我具体实现了生成树算法，并通过给定的 4 节点拓扑和自己构造的 7 节点复杂拓扑进行测试，验证了算法的正确性。然后还尝试了添加上次实验的交换机和转发表功能，在构建生成树之后进行转发表学习和数据包转发。最后，通过思考题，我对标准生成树协议进行了更深入的调研。了解到现实中生成树算法如何处理实时的拓扑变动、如何在构造生成树的同时进行交换机转发、以及如何快速构建生成树。这让我对生成树机制有了更多的认识。