STP 实验报告

一. 实验任务

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.

二. 实验过程

1. 总逻辑

总的流程如下图的流程图所示：

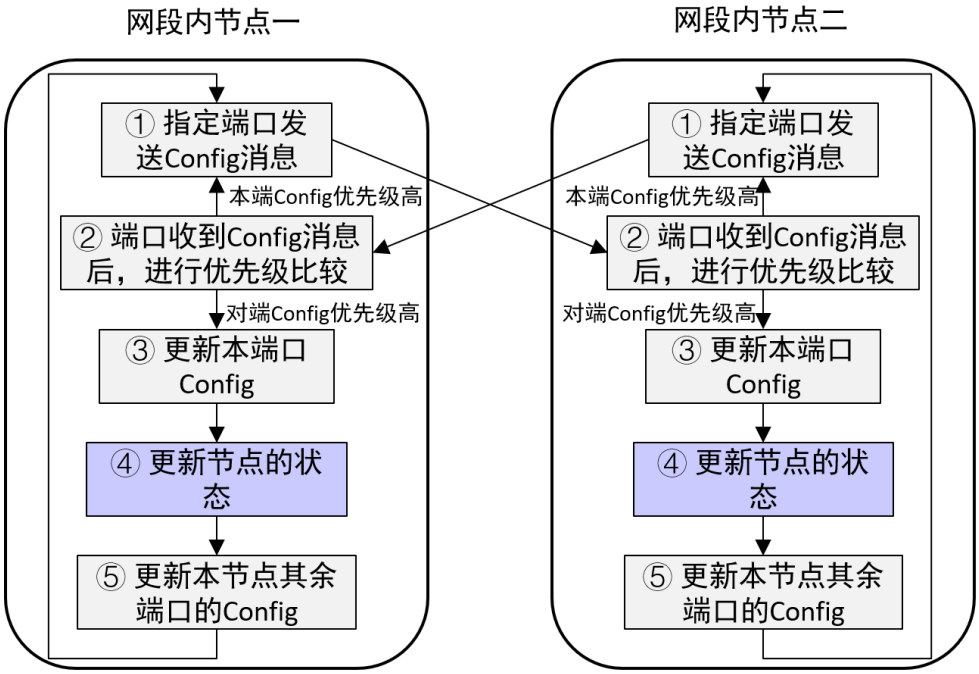


图 1：stp 算法的流程图

首先我们需要判断收到的 config 消息和本端口的 config 消息哪个优先级高，如果本端口优先级高，则说明该网段应该通过本端口存储 config 对应的端口连接根节点，在代码中体现就是不需要做额外的事情。如果本端口的 config 消息没有收到的 config 消息高，则需要更新本端口和本端口节点所对应的状态；并且停止计时器，将自己更新过后的状态转发出去，具体的代码如下图所示：

```c
/* Handle config packtes. */
static void stp_handle_config_packet(stp_t *stp, stp_port_t *p,
        struct stp_config *config)
{
    // fprintf(stdout, "TODO: handle config packet here.\n");
    if (compare_recv_priority(config, p)) {
        update_port_config(p, config);
        update_node_status(stp);
        update_other_ports_config(stp);
        if (!stp_is_root_switch(stp)) {
            stp_stop_timer(&stp -> hello_timer);
            stp_send_config(stp);
        }
    }

}
```

图 2：总的处理逻辑

之后我们会一个一个函数的进行说明。

2. Config 之间比较优先级

首先我们需要比较 config 之间的优先级，这里有两种比较，第一种是比较两个 port 的 config 的优先级，第二个是比较收到的 config 和端口 config 的优先级，两者比较逻辑大同小异，都遵循下图的原则：

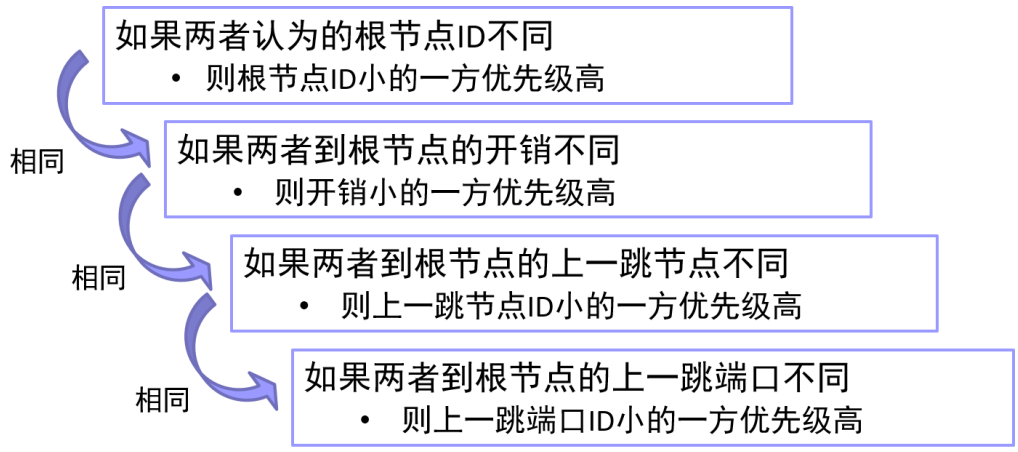


图 3：config 之间的比较逻辑

具体到代码来说即下图：

```c
/* Helper function. To compare 2 ports' priority. */
static int compare_ports_priority(stp_port_t *p1, stp_port_t *p2) {
    if (p1 -> designated_root != p2 -> designated_root) {
        if (p1 -> designated_root < p2 -> designated_root) {
            return 0;
        } else {
            return 1;
        }
    } else if (p1 -> designated_cost != p2 -> designated_cost) {
        if (p1 -> designated_cost < p2 -> designated_cost) {
            return 0;
        } else {
            return 1;
        }
    } else if (p1 -> designated_switch != p2 -> designated_switch) {
        if (p1 -> designated_switch < p2 -> designated_switch) {
            return 0;
        } else {
            return 1;
        }
    } else if (p1 -> designated_port != p2 -> designated_port) {
        if (p1 -> designated_port < p2 -> designated_port) {
            return 0;
        } else {
            return 1;
        }
    } else {
        return 0;
    }
}
```

图 4：比较两个端口 config 优先级的代码

注意的是，我们进行端口接收的 config 和端口的 config 比较的时候，要对接收端口的相应字段进行字节序转化，如下图：

```c
if (p -> designated_root != ntohll(recv_config -> root_id)) {
    if (ntohll(recv_config -> root_id) < p -> designated_root) {
        return 1;
    } else {
        return 0;
    }
```

图 5：字节序的转换

3. 替换端口的 config

如果我么发现端口接受的 config 比自己原本的 config 优先级要高，这个时候我们就需要把这个端口的 config 替换为这个优先级更高的 config，具体的逻辑很简单，把端口的 config 每个字段更新即可：

```c
/* Update current config. */
static inline void update_port_config(stp_port_t *p, struct stp_config *recv_config) {
    p -> designated_root = ntohll(recv_config -> root_id);
    p -> designated_cost = ntohl(recv_config -> root_path_cost);
    p -> designated_switch = ntohll(recv_config -> switch_id);
    p -> designated_port = ntohs(recv_config -> port_id);
}
```

图 6：更新端口的 config

4. 更新节点状态

这一部分主要有两个任务，第一个是找到 root 端口，第二步是更新节点状态，选择通过 root_port 连接到 root 端口。

首先我们先要找到 root 端口，需要满足两个条件：第一是该端口为非指定端口，第二为该端口的优先级要高于其它非指定端口。

具体我们的实现细节是，先遍历所有端口找到所有非指定端口，再找到非指定端口中优先级最高的一个，具体的代码逻辑如下图所示：

```
/* Update current port status. */
static void update_node_status(stp_t *stp) {
    stp_port_t *non_designated_ports[STP_MAX_PORTS];

    // Find all undesignated ports
    int num_non_ports = 0;
    for (int i = 0; i < stp -> nports; i++) {
        if (!stp_port_is_designated(&stp -> ports[i])) {
            non_designated_ports[num_non_ports++] = &stp -> ports[i];
        }
    }

    // Find the port which has the highest priority.
    stp -> root_port = non_designated_ports[0];
    for (int i = 1; i < num_non_ports; i++) {
        if (compare_ports_priority(stp -> root_port, non_designated_ports[i])) {
            stp -> root_port = non_designated_ports[i];
        }
    }
```

图 7：找到 root port 的代码

接下来我们需要更新节点的状态，这里分为两种情况，如果我们没有找到 root port（比如这个节点为 root node），我们就认为 root_node 还为本节点，cost 为 0；如果有 root node，我们利用 root port 的 config 信息更新我们节点的状态，主要是 root node 的 id 和 cost，代码如下图所示：

```
  // Update the node status.
  if (stp -> root_port == NULL) {
      stp -> designated_root = stp -> switch_id;
      stp -> root_path_cost = 0;
  } else {
      stp -> designated_root = stp -> root_port -> designated_root;
      stp -> root_path_cost = stp -> root_port -> designated_cost + \
                              stp -> root_port -> path_cost;
  }
```

图 8：更新 node 的状态

5. 更新端口的 config

我们首先需要处理非指定端口变为指定端口的情况，这种情况会在当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，具体的代码逻辑如下：

```
static void update_other_ports_config(stp_t *stp) {
    // Update designated port
    stp_port_t *curr_designated_port = (stp_port_t*)malloc(sizeof(stp_port_t));
    stp_port_t *comp_port;

    for (int i = 0; i < stp -> nports; i++) {
        comp_port = &(stp -> ports[i]);
        if (!stp_port_is_designated(comp_port)) {
            curr_designated_port -> designated_switch = stp -> switch_id;
            curr_designated_port -> designated_cost = stp -> root_path_cost;
            curr_designated_port -> designated_root = stp -> designated_root;
            curr_designated_port -> designated_port = INT16_MAX;
            if (compare_ports_priority(comp_port, curr_designated_port)) {
                comp_port -> designated_switch = stp -> switch_id;
                comp_port -> designated_port = comp_port -> port_id;
            }
        }
    }
```

图 9：非指定端口变为指定端口

接着我们对于每个指定端口，更新认为的根节点和路径开销：

```
for (int i = 0; i < stp -> nports; i++) {
    comp_port = &stp -> ports[i];
    if (stp_port_is_designated(comp_port)) {
        comp_port -> designated_cost = stp -> root_path_cost;
        comp_port -> designated_root = stp -> designated_root;
    }
}
```

图 10：更新指定端口的状态

6. 生成最小生成树拓扑的例子

我们建立最初的拓扑结构：

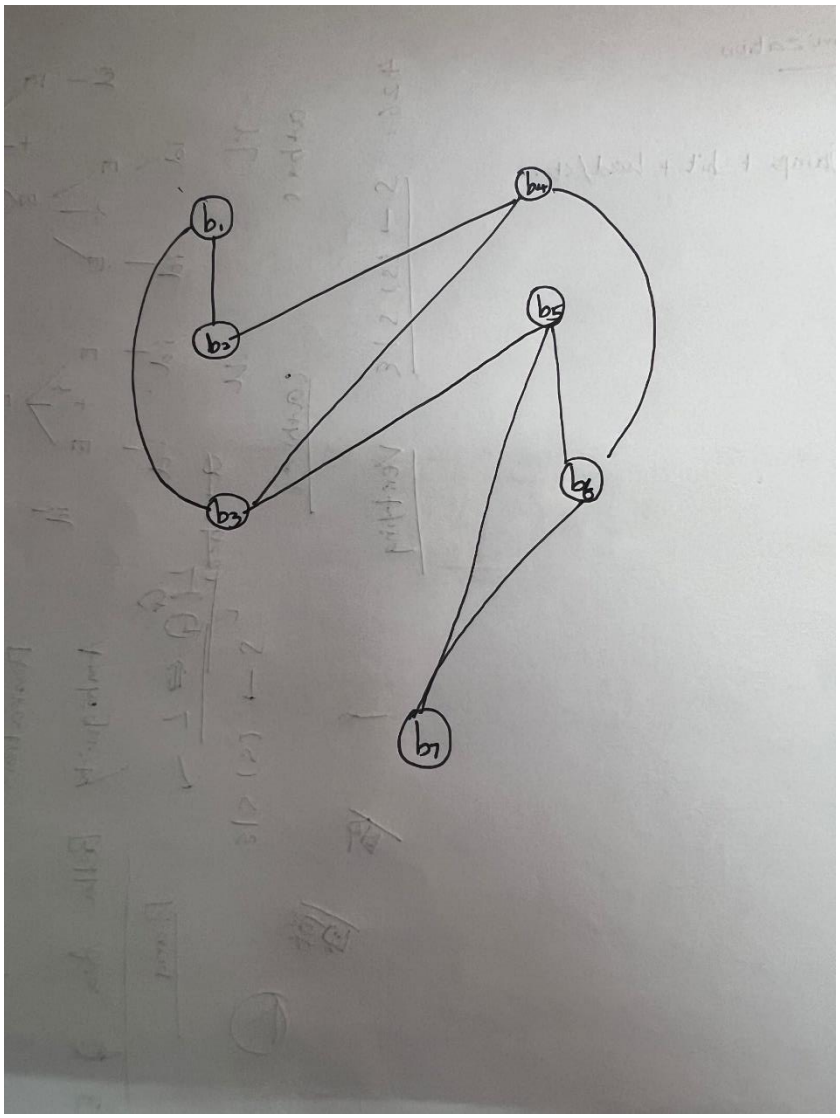


图 11：最初的拓扑结构

跑完 stp 算法，我们得到如下结果：

```
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.
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: 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.
INFO: port id: 03, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0401, ->port: 03, ->cost: 2.
```

```
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: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0501, ->port: 02, ->cost: 2.
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: 3.
INFO: port id: 01, role: ROOT.
INFO:   designated ->root: 0101, ->switch: 0401, ->port: 03, ->cost: 2.
INFO: port id: 02, role: ALTERNATE.
INFO:   designated ->root: 0101, ->switch: 0501, ->port: 02, ->cost: 2.
INFO: port id: 03, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0601, ->port: 03, ->cost: 3.

NODE b7 dumps:
INFO: non-root switch, designated root: 0101, root path cost: 3.
INFO: port id: 01, role: ROOT.
INFO:   designated ->root: 0101, ->switch: 0501, ->port: 03, ->cost: 2.
INFO: port id: 02, role: ALTERNATE.
INFO:   designated ->root: 0101, ->switch: 0601, ->port: 03, ->cost: 3.
```

图 12：最后生成树结果

我们具体化一些：

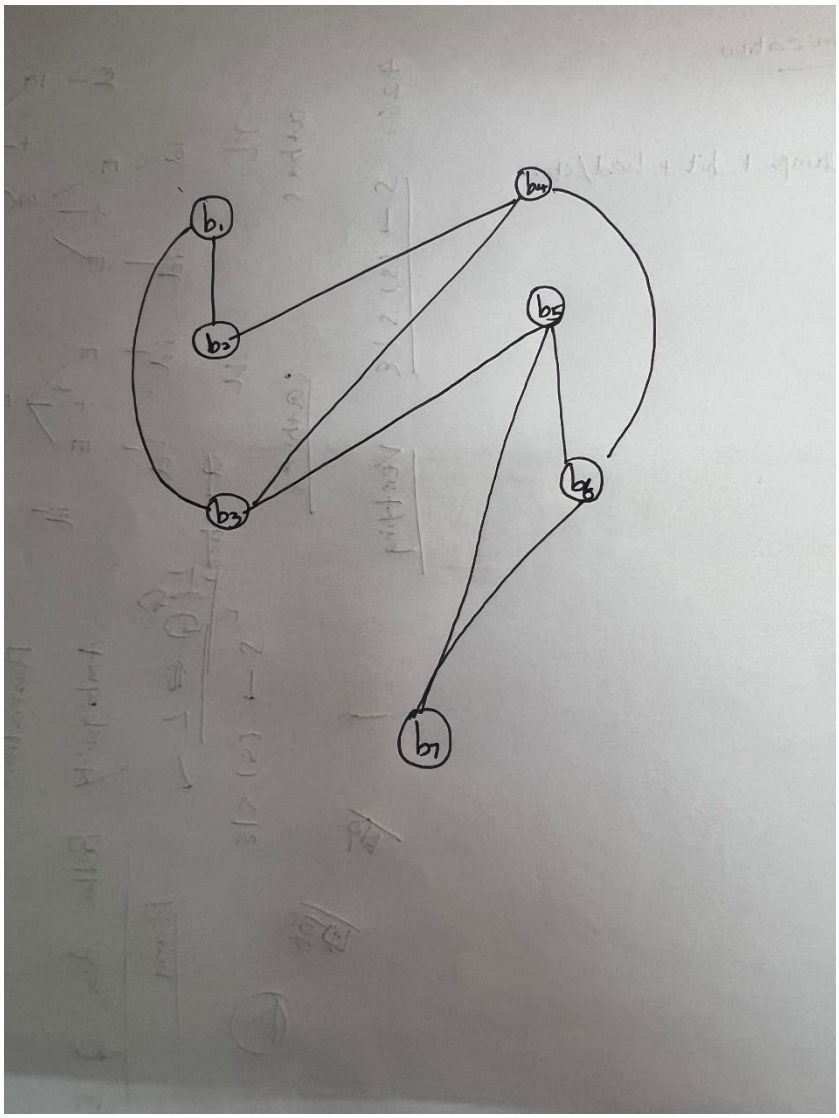
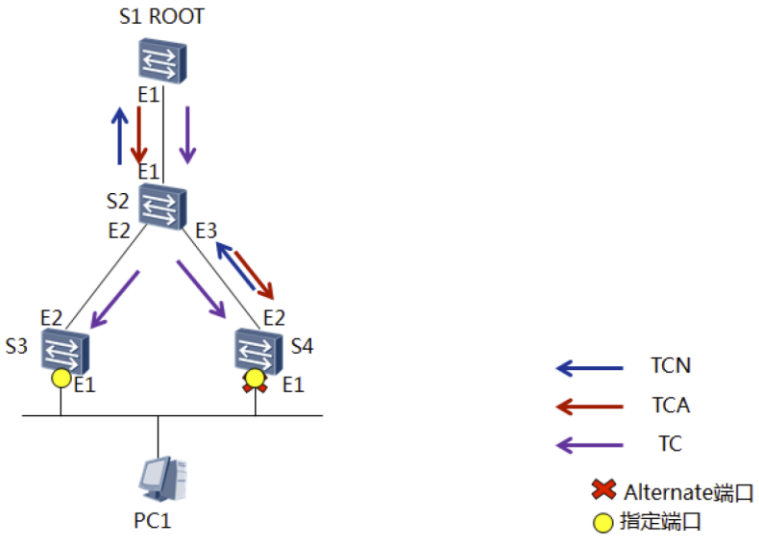


图 13：最后的生成树结果（图示）

三. 实验思考

1. 标准生成树协议中，如何处理网络拓扑变动的情况？

节点加入或者离开拓扑结构时候，都要重新进行 STP 计算，更新节点状态和端口的配置信息，如果生成树拓扑变化，发送数据的路径也会发生变化。拓扑变化时，应该及时更新 MAC 地址表项。拓扑变化时候，发生改变的交换机通过 TCN BPDU 报文(Topology Change Notification)告知根桥生成树拓扑里发生了变化，只有指定端口会处理 TCN BPDU。上游设备收到 TCN BPDU，回复 TCA 置位的配置 BPDU，用于终止下游设备发送 TCN BPDU。上游继续向根桥方向发送 TCN BPDU，直到根桥收到后。根桥回复 TCA+TC，TC 全网泛洪，用于清除交换机上的 mac 地址表。TC 的作用为：避免旧的 mac 表造成数据的转发错误。



TCN消息处理过程：

1. 在网络拓扑发生变化后，有端口转为转发状态的下游设备会不间断地向上游设备发送TCN BPDU报文。
2. 上游设备收到下游设备发来的TCN BPDU报文后，只有指定端口处理TCN BPDU报文。其它端口也有可能收到TCN BPDU报文，但不会处理。
3. 上游设备会把配置BPDU报文中的Flags的TCA位设置1，然后发送给下游设备，告知下游设备停止发送TCN BPDU报文。
4. 上游设备复制一份TCN BPDU报文，向根桥方向发送。
5. 重复步骤1、2、3、4，直到根桥收到TCN BPDU报文。
6. 根桥收到TCN BPDU后，会将下一个要发送的配置BPDU中的TCA位置位，作为对收到的TCN的确认，还会将该配置BPDU报文中的Flags的TC位置1，用于通知所有网桥拓扑发生了变化。
7. 根桥在之后的max（age+forwardingdelay）时间内，将发送BPDU中的TC置位的报文，收到该配置BDPU的网桥，会将自身MAC地址老化时间缩短为forwarding delay。

2. 标准生成树协议如何在构建生成树的过程中保持网络联通？

端口状态	发送/接收配置BPDU	MAC地址学习	转发数据	作用说明
Disable	否/否	否	否	端口状态为down
Blocking	否/是	否	否	阻塞端口的最终状态
Listening	是/是	否	否	过渡状态，选举根桥、确定端口角色
Learning	是/是	是	否	过渡状态，构建MAC地址表
Forwarding	是/是	是	是	只有根端口和指定端口才能进入Forwarding状态

图 14：端口状态表

在构建生成树的过程中，STP会将部分冗余链路强制转化为阻塞状态，其余则链路处于转发状态。当处于转发状态的链路不可用时，STP可以重新配置网络，集合合适的备用链路，恢复部分冗余链路的转发状态来确保网络的连通性。

3. **调研说明快速生成树机制的原理。**

RSTP概述：

快速生成树协议RSTP（Rapid Spanning Tree Protocol）在STP基础上实现了快速收敛，并增加了边缘端口的概念及保护功能。

RSTP在STP基础上新增加了2种端口角色：Backup端口和边缘端口。通过端口角色的增补，简化了生成树协议的理解及部署。

Backup端口：由于学习到自己发送的配置BPDU报文而阻塞的端口，指定端口的备份，提供了另外一条从根节点到叶节点的备份通路。

边缘端口：如果端口位于整个交换区域边缘，不与任何交换设备连接，这种端口叫做边缘端口。边缘端口一般与用户终端设备直接连接。

边缘端口的特点

1. 边缘端口会节省30S的延时，端口UP后会立即进入转发状态。

2. 边缘端口的UP/DOWN不会触发拓扑改变。

3. 边缘端口收的TC置为的配置BPDU报文不会将MAC地址的老化时间设置为15s。

4. 边缘端口如果收到配置的BPDU报文会马上变为一个普通端"口，进行STP的收敛

5. 边缘端口也会发送配置BPDU报文。

6. PA协商不会阻塞边缘端口。

RSTP的端口状态

RSTP的端口状态在STP的基础上进行了改进。由原来的五种缩减为三种。

端口状态	说明
Forwarding（转发）	在这种状态下，端口既转发用户流量又处理BPDU报文。
Learning（学习）	这是一种过渡状态。在Learning下，交换设备会根据收到的用户流量，构建MAC地址表，但不转发用户流量，所以叫做学习状态。Learning状态的端口处理BPDU报文，不转发用户流量。
Discarding(丢弃)	Discarding状态的端口只接收BPDU报文。

图15：RSTP端口状态

RSTP中只有一种BPDU包，称为RST BPDU。对配置BPDU的flags字段进行了填充，类型值做了改变，如下：

1. Type：RST BPDU包此字段类型是2

2. Flags字段,RST填充了STP BPDU Flags中的保留字段，填充了如下信息：

（1）Aggrement：确认标识位，用于RSTP中定义的Proposal/Aggrement机制，对于Proposal报文的确认

（2）Forwarding：转发状态标识位，1：表示发送该BPDU报文的端口处于Forwarding状态

（3）Learning：学习状态标识位，1：表示发送该BPDU报文的端口处于Learning状态