广播网络实验

2024年11月13日

2024E8013282087-陈潇

实验内容

基于已有代码，实现生成树运行机制

- 设计思路 现有代码框架已解决了STP通信等相关问题，我们需要处理的主要任务是解析收到的包，并根据包中包含的目标端口配置与本端口及其对应节点的配置进行对比，从而更新本端口及本节点的配置。实现这一功能的代码部分将会在代码框架中的`stp_handle_config_packet`函数中完成。

  本设计对应的伪函数实现：

```
// 收到 STP 配置消息时处理
function handle_stp_config_received(stp_config, port):
    // 比较本端口和发送端口的优先级
    if compare_ports_priority(port, stp_config.sender_port) < 0:  // 本
端口优先级较高
        // 更新本端口的 STP 配置
        update_port_config(port, stp_config)

        // 更新本节点的 STP 配置
        update_node_config(stp_config)

        // 更新本节点其他端口的 STP 配置
        for each other_port in node_ports:
            if other_port != port:
                update_port_config(other_port, stp_config)

        // 如果本节点不再是根节点
        if is_root_node(node) == False:
            // 停止本节点的计时器
            stop_timer(node)

            // 将 STP 配置通过指定端口发出
            send_stp_config(node, stp_config, port)
    else:
```

```
        // 本端口的优先级不更高，忽略收到的配置
        return
```

函数实现：

```
#include "base.h"
#include <stdio.h>
extern ustack_t *instance;

void broadcast_packet(iface_info_t *iface, const char *packet, int len)
{
        // TODO: broadcast packet
        iface_info_t *ifa = NULL;
        list_for_each_entry(ifa, &instance->iface_list, list){
                if(ifa->index!=iface->index)
                {
                        iface_send_packet(ifa,packet,len);
                }
        }
}
```

为了实现上述函数，还需要实现几个子函数，下面对其进行解释：`int compare_ports_pirority(stp_port_t * p1, stp_port_t * p2)` 函数描述：比较当前端口和发送端口的config优先级，当前端口更高返回`0`，发送窗口更高返回`1`。 函数实现：

```
int compare_ports_pirority(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 1;
        }
    }
```

`int recv_has_higher_pirority(stp_port_t * p, struct stp_config *config)` 函数描述：对接收到的包进行字节时序的转换，然后比较当前端口和发送端口的config优先级，当前端口更高返回`0`，发送窗口更高返回`1`。 函数实现：

```
    int recv_has_higher_pirority(stp_port_t * p, struct stp_config
*config) {
        if (p->designated_root != ntohll(config->root_id)) {
            if(p->designated_root < ntohll(config->root_id)) {
                return 0;
            } else {
                return 1;
            }
        } else if (p->designated_cost != ntohl(config->root_path_cost)) {
            if (p->designated_cost < ntohl(config->root_path_cost)) {
                return 0;
            } else {
                return 1;
            }
        } else if (p->designated_switch != ntohll(config->switch_id)) {
            if (p->designated_switch < ntohll(config->switch_id)) {
                return 0;
            } else {
                return 1;
            }
        } else if (p->designated_port != ntohs(config->port_id)) {
            if (p->designated_port < ntohs(config->port_id)) {
                return 0;
            } else {
                return 1;
            }
        } else {
            return 1;
        }
    }
```

`void update_port_config(stp_port_t *p, struct stp_config *config)` 函数描述：根据接收端口发送的config的值来更新本端口的config，同时将本端口更新为非指定端口。 函数实现：

```
    void update_port_config(stp_port_t *p, struct stp_config *config) {
        p->designated_root = ntohll(config->root_id);
        p->designated_switch = ntohll(config->switch_id);
```

```
        p->designated_port = ntohs(config->port_id);
        p->designated_cost = ntohl(config->root_path_cost);
    }
```

`void update_root(stp_t *stp, struct stp_config *config)` 函数描述：更新节点的状态，先遍历所有端口找出根端口，如果不存在根端口则把当前节点设置为根端口。再通过root_port连接到根节点，更新节点状态。 `void update_other_ports(stp_t *stp)` 函数描述：对于所有指定端口，更新其认为的根节点和路径开销。 函数实现：

```
    void update_other_ports(stp_t *stp) {
        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;
            }
        }
    }
```

对于给定拓扑(four_node_ring.py)，计算输出相应状态下的生成树拓扑

- 修改python脚本，添加下面的命令

```
node.cmd('./stp > %s-output.txt 2>&1 &' & name)
```

或者在shell中键入以下命令：

```
mininet> xterm b1 b2 b3 b4
b1# ./stp > b1-output.txt 2>&1
b2# ./stp > b2-output.txt 2>&1
b3# ./stp > b3-output.txt 2>&1
b4# ./stp > b4-output.txt 2>&1
```

等待一段时间，在新终端中：

```
sudo pkill -SIGTERM stp
./dump_output.sh 4
```

结果如下：

```
xiao@xiao:~/compute-net/net-lab-1/3-stp$ sudo pkill -SIGTERM stp
xiao@xiao:~/compute-net/net-lab-1/3-stp$ ./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.
```

可以看出该4节点的测试成功。

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程序计算输出生成树拓扑

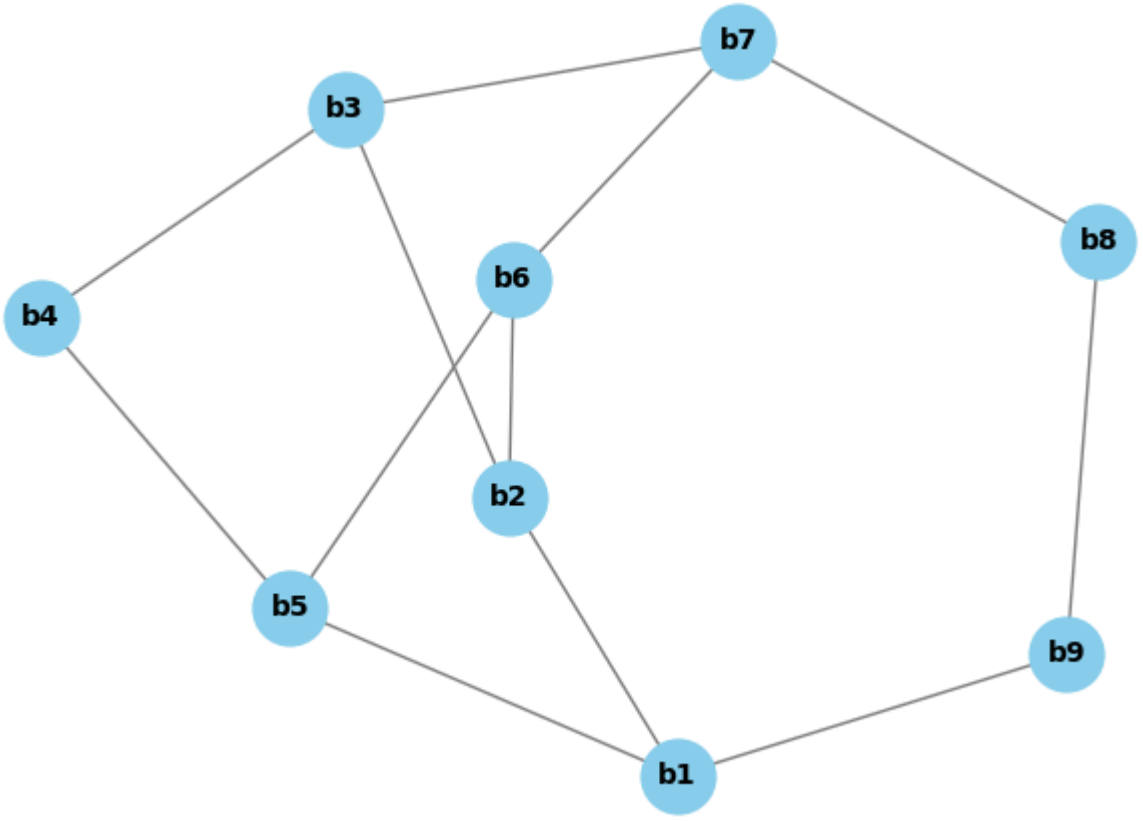
- 构造的九节点三冗余链路的拓扑代码如下：

```
import os
import sys
import glob

from mininet.topo import Topo
from mininet.net import Mininet
from mininet.cli import CLI

script_deps = [ 'ethtool' ]

def check_scripts():
    dir = os.path.abspath(os.path.dirname(sys.argv[0]))

    for fname in glob.glob(dir + '/' + 'scripts/*.sh'):
        if not os.access(fname, os.X_OK):
            print('%s should be set executable by using chmod +x
$script_name' % (fname))
            sys.exit(1)
```

```
    for program in script_deps:
        found = False
        for path in os.environ['PATH'].split(os.pathsep):
            exe_file = os.path.join(path, program)
            if os.path.isfile(exe_file) and os.access(exe_file,
os.X_OK):
                found = True
                break
        if not found:
            print('%s is required but missing, which could be installed
via apt or aptitude' % (program))
            sys.exit(2)

def clearIP(n):
    for iface in n.intfList():
        n.cmd('ifconfig %s 0.0.0.0' % (iface))

class RedundantTopo(Topo):
    def build(self):
        # 创建9个节点
        nodes = []
        for i in range(9):
            nodes.append(self.addHost(f'b{i+1}'))

        # 创建冗余链路（3条）
        self.addLink(nodes[0], nodes[1])
        self.addLink(nodes[1], nodes[2])
        self.addLink(nodes[2], nodes[3])
        self.addLink(nodes[3], nodes[4])
        self.addLink(nodes[4], nodes[5])
        self.addLink(nodes[5], nodes[6])
        self.addLink(nodes[6], nodes[7])
        self.addLink(nodes[7], nodes[8])
        self.addLink(nodes[8], nodes[0])  # 冗余链路1

        self.addLink(nodes[0], nodes[4])  # 冗余链路2
        self.addLink(nodes[1], nodes[5])  # 冗余链路3

if __name__ == '__main__':
    check_scripts()

    topo = RedundantTopo()
    net = Mininet(topo=topo, controller=None)

    for idx in range(9):
        name = f'b{idx+1}'
        node = net.get(name)
        clearIP(node)
        node.cmd('./scripts/disable_offloading.sh')
        node.cmd('./scripts/disable_ipv6.sh')
        node.cmd('./stp > %s-output.txt 2>&1 &' % name)

        # 设置每个接口的 MAC 地址
        for port in range(len(node.intfList())):
            intf = f'{name}-eth{port}'
            mac = f'00:00:00:00:0{idx+1}:0{port+1}'
```

```
            node.setMAC(mac, intf=intf)

        # node.cmd('./stp > %s-output.txt 2>&1 &' % name)
        # node.cmd('./stp-reference > %s-output.txt 2>&1 &' % name)

    net.start()
    CLI(net)
    net.stop()
```

拓扑结构如下：



STP测试如下：

```
xiao@xiao:~/compute-net/net-lab-1/3-stp$ sudo pkill -SIGTERM stp
xiao@xiao:~/compute-net/net-lab-1/3-stp$ ./dump_output.sh 9
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.

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: ROOT.
INFO:   designated ->root: 0101, ->switch: 0503, ->port: 02, ->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: 02, ->cost: 0.
INFO: port id: 02, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0503, ->port: 02, ->cost: 1.
INFO: port id: 03, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0503, ->port: 03, ->cost: 1.

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: 0503, ->port: 03, ->cost: 1.
INFO: port id: 03, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0603, ->port: 03, ->cost: 2.

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: 0603, ->port: 03, ->cost: 2.
INFO: port id: 02, role: ALTERNATE.
INFO:   designated ->root: 0101, ->switch: 0801, ->port: 01, ->cost: 2.

NODE b8 dumps:
INFO: non-root switch, designated root: 0101, root path cost: 2.
INFO: port id: 01, role: DESIGNATED.
INFO:   designated ->root: 0101, ->switch: 0801, ->port: 01, ->cost: 2.
INFO: port id: 02, role: ROOT.
INFO:   designated ->root: 0101, ->switch: 0901, ->port: 01, ->cost: 1.
```