VIRTUAL-ROUTE-DEMO

路由选择协议的实现

按 → 继续看

目录

按 → 继续看

前言

开发及运行环境

在python3.5下开发

概述

- 在TCP socket的基础上，模拟了链路层和网络层的工作
- 在拟网络层的接口之上，开发了不同的路由选择协议
- 路由器通过额外的线程运行路由选择协议自动配置路由表。

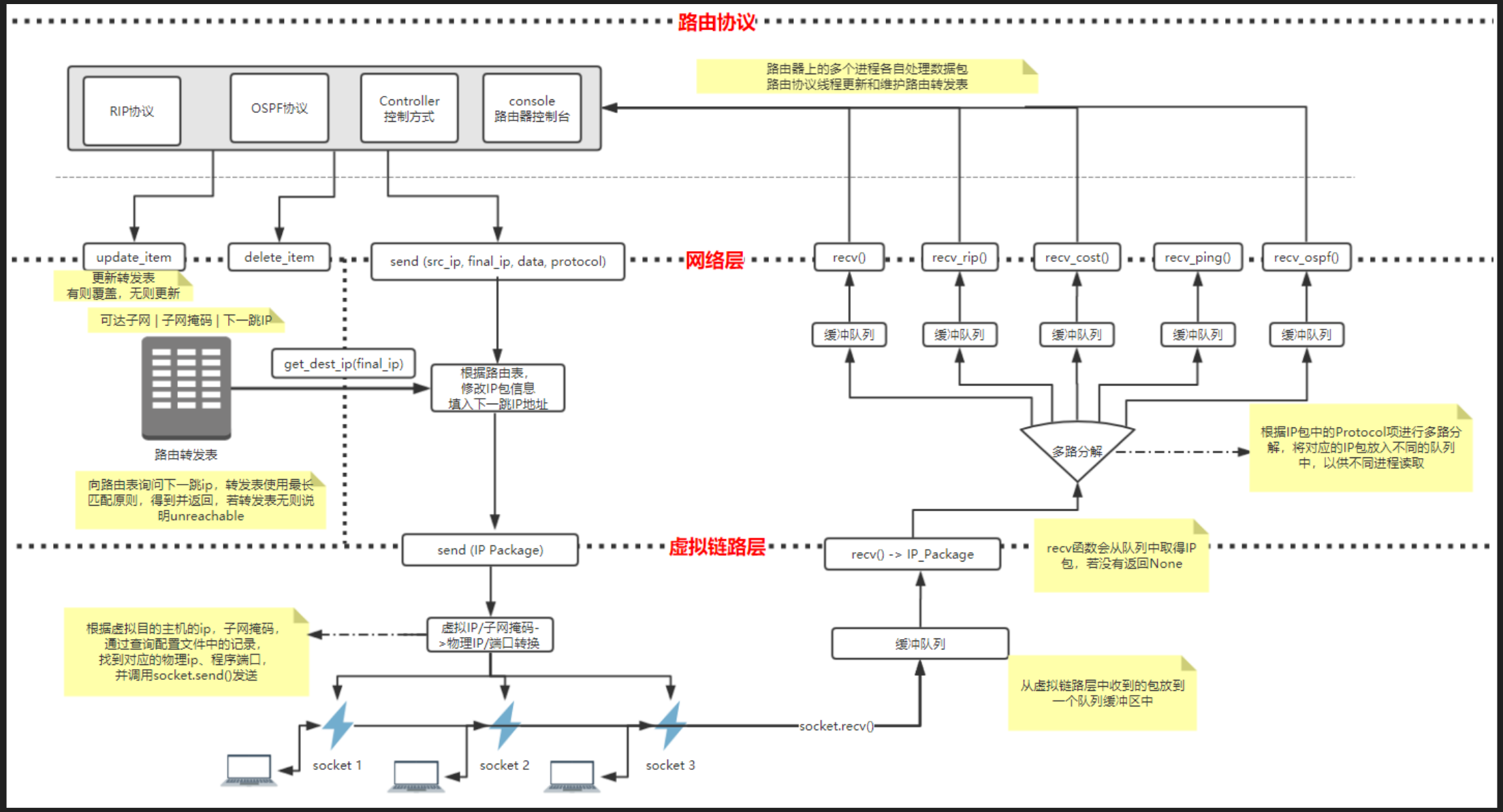
功能特性

1. 虚拟路由器具有控制台
    1. 能够在控制台实现显示路由表，距离向量表，修改路由表等操作。
2. 虚拟路由器具有基于不同算法（LS和DV）的路由协议（OSPF和RIP）
    1. 能够自动构建和维护各虚拟路由程序中的路由表。
3. 在控制台中实现了与ping类似的操作用于测试路由器连通性。
4. 实现了中心化虚拟路由程序。

技术路线

1. 在PY3原生socket库基础上实现了可靠的数据包传送协议
    1. 解决黏包，半包问题
2. 使用多线程技术支持并发操作。
3. 在架构上本次项目我们采用了分层设计的基本方法。
    1. 各层之间通过通用性强的API解耦合。

2. 架构篇


路由协议
RIP协议
OSPF协议
Controller
控制方式
console
路由器控制台
路由器上的多个进程各自处理数据包
路由协议线程更新和维护路由转发表
update_item
delete_item
send (src_ip, final_ip, data, protocol)
网络层
recv()
recv_rip()
recv_cost()
recv_ping()
recv_ospf()
更新转发表
有则覆盖，无则更新
可达子网 | 子网掩码 | 下一跳IP
缓冲队列
缓冲队列
缓冲队列
缓冲队列
缓冲队列
get_dest_ip(final_ip)
根据路由表，
修改IP包信息
填入下一跳IP地址
路由转发表
多路分解
根据IP包中的Protocol项进行多路分解，将对应的IP包放入不同的队列中，以供不同进程读取
向路由表询问下一跳ip，转发表使用最长匹配原则，得到并返回，若转发表无则说明unreachable
send (IP Package)
虚拟链路层
recv() -> IP_Package
recv函数会从队列中取得IP包，若没有返回None
根据虚拟目的主机的ip，子网掩码，通过查询配置文件中的记录，找到对应的物理ip、程序端口，并调用socket.send()发送
虚拟IP/子网掩码->物理IP/端口转换
缓冲队列
从虚拟链路层中收到的包放到一个队列缓冲区中
socket.recv()
socket 1
socket 2
socket 3

3. 部署使用篇

3.1 获取源码

运行下面这一条指令获得源码及demo：

```
git clone https://github.com/WalkerYF/virtual-routing-demo
cd virtual-routing-demo/
```

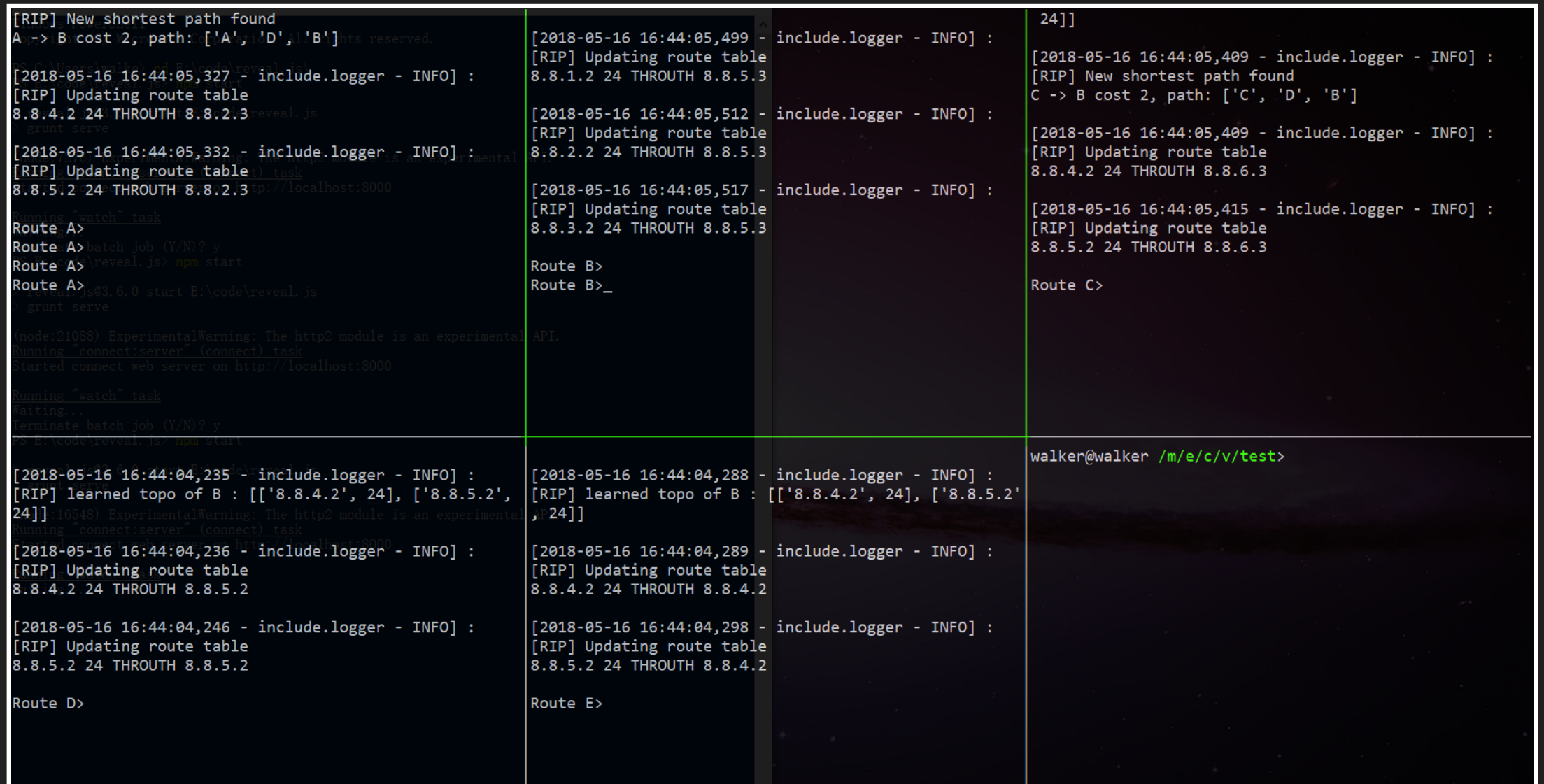
获取源码后，运行一键测试脚本：

```
cd test
./test_rip.sh # 运行测试RIP协议的脚本
```

运行结果可按↓

运行脚本后，开启了样例1中的5台路由器

并分别进入控制台等待控制台输入指令。按→继续



3.2 使用控制台

路由器的控制台支持以下命令

```
show route # 查看路由表
show interface # 查看接口情况
show tcp # 查看底层TCP socket的情况
show dv # 在RIP协议中可用，用于查看距离向量表
add (dest_net) (net_mask) (next_ip) # 用于在路由表中新增一项
delete (dest_net) (net_mask) # 用于在路由表中删除一项
send (src_ip) (final_ip) (data) # 用于发送消息到指定路由器
recv # 从本地队列中取得一个IP包并显示
```

关于这些命令的使用样例及测试，可见console测试

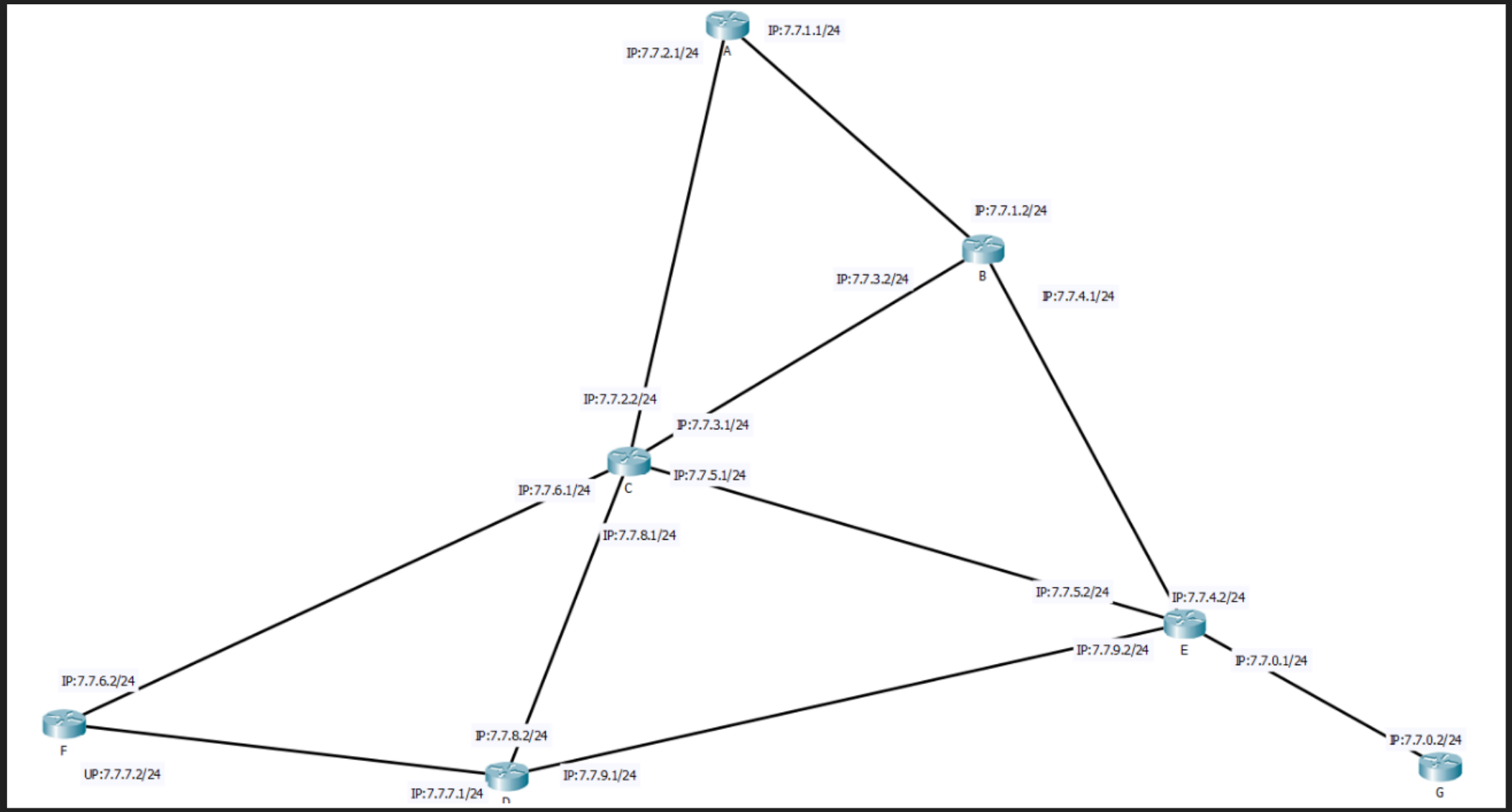
4 测试篇

该部分我们主要测试了一下项目。

- 路由器控制台的使用，转发表的设置
- 基于RIP协议的路由表动态配置
- 基于OSPF协议的路由表动态配置
- 中心化路由

按↓查看测试所用拓扑 按→查看测试1结果

考虑到篇幅限制，我们在使用的三个测试样例中，仅使用test2样例来说明。

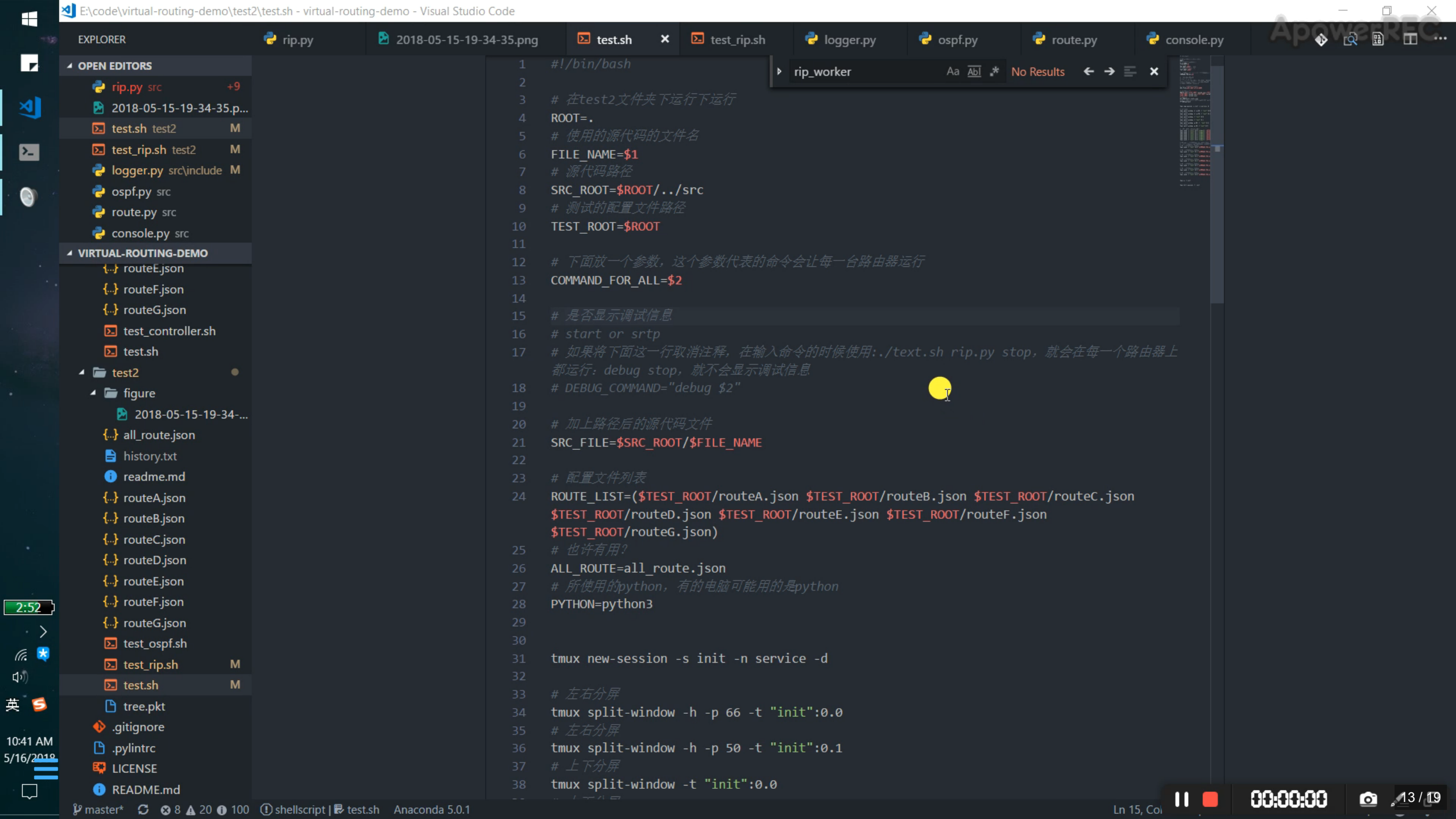


4.1 测试一：路由器控制台的使用

此处主要展示了各项命令的输出，

以及设置转发路由表对不同子网间端口连通性影响。

详情请按↓查看测试视频


E:\code\virtual-routing-demo\test2\test.sh - virtual-routing-demo - Visual Studio Code
EXPLORER
rip.py
2018-05-15-19-34-35.png
test.sh
test_rip.sh
logger.py
ospf.py
route.py
console.py
OPEN EDITORS
rip.py src +9
2018-05-15-19-34-35.p...
test.sh test2 M
test_rip.sh test2 M
logger.py src\include M
ospf.py src
route.py src
console.py src
VIRTUAL-ROUTING-DEMO
routeE.json
routeF.json
routeG.json
test_controller.sh
test.sh
test2
figure
2018-05-15-19-34-...
all_route.json
history.txt
readme.md
routeA.json
routeB.json
routeC.json
routeD.json
routeE.json
routeF.json
routeG.json
test_ospf.sh
test_rip.sh M
test.sh M
tree.pkt
.gitignore
.pylintrc
LICENSE
README.md
rip_worker
No Results
#!/bin/bash
# 在test2文件夹下运行下运行
ROOT=.
# 使用的源代码的文件名
FILE_NAME=$1
# 源代码路径
SRC_ROOT=$ROOT/../src
# 测试的配置文件路径
TEST_ROOT=$ROOT
# 下面放一个参数，这个参数代表的命令会让每一台路由器运行
COMMAND_FOR_ALL=$2
# 是否显示调试信息
# start or srtp
# 如果将下面这一行取消注释，在输入命令的时候使用:./text.sh rip.py stop，就会在每一个路由器上都运行：debug stop，就不会显示调试信息
# DEBUG_COMMAND="debug $2"
# 加上路径后的源代码文件
SRC_FILE=$SRC_ROOT/$FILE_NAME
# 配置文件列表
ROUTE_LIST=($TEST_ROOT/routeA.json $TEST_ROOT/routeB.json $TEST_ROOT/routeC.json $TEST_ROOT/routeD.json $TEST_ROOT/routeE.json $TEST_ROOT/routeF.json $TEST_ROOT/routeG.json)
# 也许有用?
ALL_ROUTE=all_route.json
# 所使用的python，有的电脑可能用的是python
PYTHON=python3
tmux new-session -s init -n service -d
左右分屏
tmux split-window -h -p 66 -t "init":0.0
左右分屏
tmux split-window -h -p 50 -t "init":0.1
上下分屏
tmux split-window -t "init":0.0
master*
8 20 100
shellscript | test.sh
Anaconda 5.0.1
Ln 15, Co
2:52
10:41 AM
5/16/2018
00:00:00
13 / 19

4.1.1 实验结果

1. 基于TCP socket实现的虚拟链路层和网络层工作正常
2. 控制台能够正确的控制路由器内部信息。

4.2 测试二：RIP协议

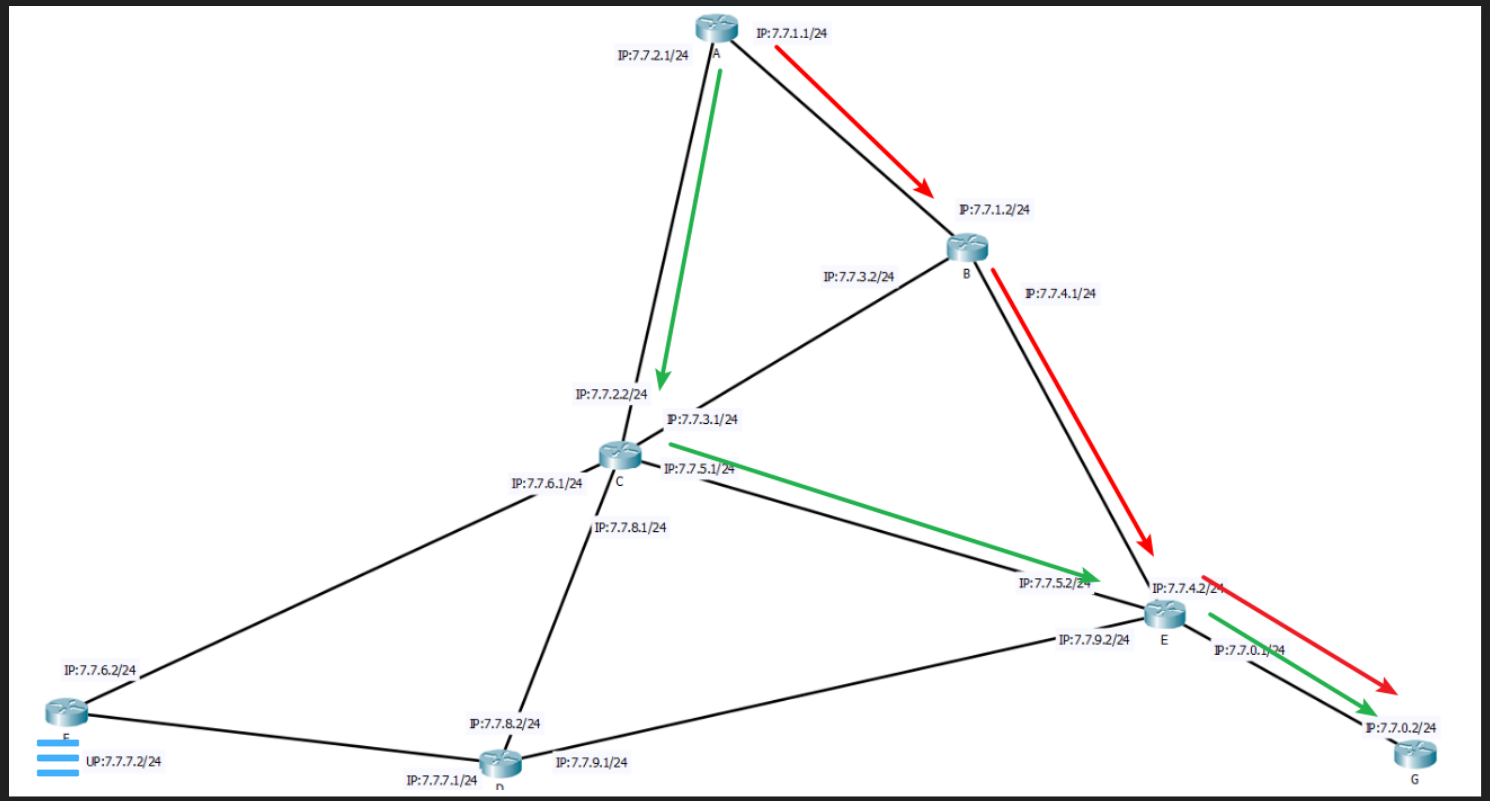
我们主要测试了以下三个场景。

每一个场景的测试过程可点相应超链接观看相应的视频了解结果。

1. 场景1：【路由器刚开机】
    1. 每台路由器只有本地链路信息，如何转发？
2. 场景2：【网络拓扑结构改变】
    1. 路由器下线，其他路由器的路由表能更新吗？
3. 场景3：【链路费用改变】
    1. 某线路阻塞，能自适应更新吗？

测试场景2说明

按→查看*测试三：ospf协议*结果



情景	路线
B还在线	A->B->E->F(如红线)
B下线	A->C->E->F(如绿线)

4.3 测试三：OSPF协议

OSPF是一个使用SPFA算法，通过全局拓扑信息计算得到最优的路的协议。

对该协议的测试我们分为以下两个场景：

1. 场景1：【路由器刚开机】
    1. 开启OSPF协议后，应该能够做到以下两点
        1. 每一个路由器能够得到全局的拓扑信息
        2. 设置好本地的转发表。
2. 场景2：【网络拓扑结构改变】
    1. 路由器发现拓扑结构的改变
    2. 广播此改变信息，并修改自己的转发表和数据库。

以上两个场景的视频可点击此处

4.4 测试四：中心化路由

在这种实现，网络拓扑中具有一个中心路由。

网络上的每一台主机依赖这一台主机得到转发表，而不需自己获得拓扑信息来计算。

这里是我们的测试视频

THANKS !