中山大學
SUN YAT-SEN UNIVERSITY

本科生实验报告

Undergraduate Experiment Report

题目 Title: **计算机网络 (II) 实验报告**

院 系
School (Department): 数据科学与计算机学院

专 业
Major: 网络空间安全

学生姓名
Student Name: 金钰 王广烁 李晨曦

学 号
Student No.:

时间：二〇二〇年十二月三十日

Date: December 30th 2020

目录

插图目录

表格目录

第一章 背景

第二章 方案设计

2.1 课程作业目的

2.2 场景设计与预期要求

2.3 实现方案与技术细节

2.3.1 实现方案

2.3.1.1 拓扑设计

拓扑设计参照实验要求，实验时的拓扑如下：

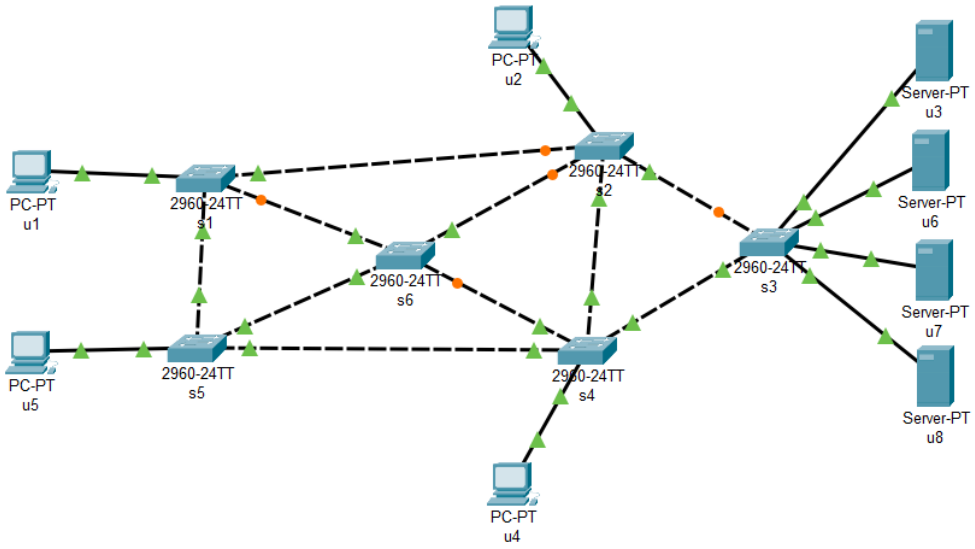

图 2-1 拓扑图

其中，u_3, u_6, u_7, u_8 作为 iperf3 测试时的服务器，u_1, u_5 作为 iperf3 测试时的客户端。各链路均有带宽和延迟的参数，如下表所示：

	$s_1 - s_2$	$s_1 - s_5$	$s_2 - s_4$	$s_4 - s_6$	其他
延迟 (ms)	15	15	15	15	10
带宽 $(Mbps)$	50	50	50	50	75

表 2.1 延迟带宽表

2.3.1.2 传统路由方案设计

使用 Ryu 控制器自带的 `ryu.app.rest_router`，可以进行传统的基于最短距离的路由。

`ryu.app.rest_router` 实现了两个功能：

- 生成路由规则并下发给 Switch。这样一来每个 Switch 都可以看作一个路由器。
- 支持通过一套 Restful API 对路由规则进行增删改查，这样一来外部可以根据需要修改路由规则。

但这个 APP 并不能自动地进行路由规则的生成，我们需要通过某种规则来生成路由并通过此 APP 的 Restful API 部署生成的路由规则。`classic/make_route.py` 文件就实现了根据最短路径算法生成路由并告知 `ryu.app.rest_router` 的过程。

主要的生成过程如下：

1) 把拓扑视作一个图，通过在每个点进行 Dijkstra 算法生成两点之间的最短路径。
2) 通过路径生成路由信息。如 u_1 到 u_3 的最短路径是 $u_1 \to u_2 \to u_3$，那么生成类似于 $(dst : u_3, next\_hop : u_2)$ 的路由信息。
3) 把生成的路由信息发送给 `ryu.app.rest_router`

这样的方法需要我们提前给定两个点之间的边权，这里把延迟作为边权。

2.3.1.3 基于 TCP 与 UDP 路由的 SDN 路由方案设计

首先给定一套把从 u_1 和 u_5 发出的 TCP 和 UDP 进行分流的方案：

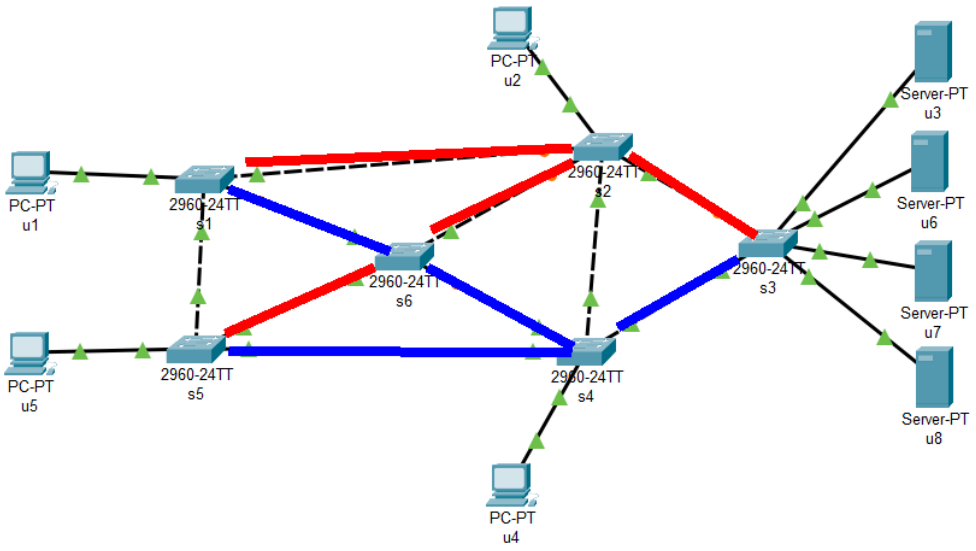

图 2-2 UDP 和 TCP 路径

如图，红色路径是 TCP 流的路径，蓝色路径是 UDP 流的路径。

实现这个路径需要用到 Ryu 控制器自带的 `ryu.app.ofctl_rest`，这个 APP 提供了一套增删改查任意 Switch 流表的 Restful API，使用它可以方便地创建所需的流表项。

在传统路由的基础上创建几个特定的高优先级流表项即可。

2.3.1.4 Qos 性能测试设计

这里采用了 iperf3 工具进行测试。iperf3 工具可以创建一段时间的 TCP 和 UDP 通信，这一段通信的速率、延迟、丢包等信息可以帮助我们获取 Qos 性能信息。也可以在测试时实时进行报文抓取，这也反映了很多 Qos 性能信息。

2.3.2 实现传统路由

2.3.2.1 搭建拓扑

首先找到 `classic/create_topo.py` 这个文件，在配置好 mininet 的主机 A 中运行

```
sudo mn --custom create_topo.py --topo mytopo \
--switch ovs --controller remote,ip={ip of B} --link tc
```

其中 `ip` 项是控制器主机 B 的 ip 地址。

在 B 主机中运行

```
ryu run --observe-links  \
ryu.app.rest_router ryu.app.gui_topology.gui_topology
```

这样一来整个拓扑已经搭建成功。浏览 `http://ip_of_B:8080` 可以查看拓扑图像如下：

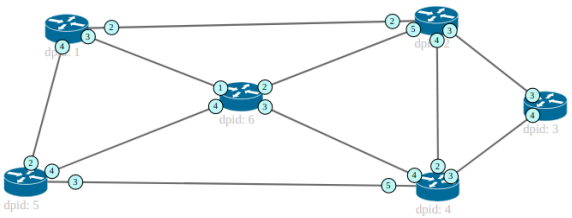

图 2-3 拓扑图

注意到这里没有画出主机的拓扑情况。

2.3.2.2 生成路由

首先找到 `classic/make_route.py`，在主机 B 上运行

```
python make_route.py
```

这会在控制器上创建基于最短路算法的静态路由。此时路由已经创建好，但是，由于主机的默认网关均未进行配置，要手动配置默认网关。

在 mininet 的命令行中运行

```
u1 ip route add default via 172.18.0.2 \
u2 ip route add default via 172.18.1.2 \
u3 ip route add default via 172.18.2.2 \
u4 ip route add default via 172.18.3.2 \
u5 ip route add default via 172.18.4.2 \
u6 ip route add default via 172.18.2.2 \
u7 ip route add default via 172.18.2.2 \
u8 ip route add default via 172.18.2.2
```

这样一来路由就搭建成功了。

2.3.2.3 测试路由

在 mininet 的命令行中运行

```
pingall
```

得到输出如下：

```
mininet> pingall
*** Ping: testing ping reachability
u1 -> u2 u3 u4 u5 u6 u7 u8
u2 -> u1 u3 u4 u5 u6 u7 u8
u3 -> u1 u2 u4 u5 u6 u7 u8
u4 -> u1 u2 u3 u5 u6 u7 u8
u5 -> u1 u2 u3 u4 u6 u7 u8
u6 -> u1 u2 u3 u4 u5 u7 u8
u7 -> u1 u2 u3 u4 u5 u6 u8
u8 -> u1 u2 u3 u4 u5 u6 u7
*** Results: 0% dropped (56/56 received)
```

这样一来就可以确定路由已经正常工作。

2.3.2.4 性能测试

在 mininet 的命令行中运行

```
u3 iperf3 -s -D
u6 iperf3 -s -D
u7 iperf3 -s -D
u8 iperf3 -s -D
u1 iperf3 -c u3 -t 20 > u1.tcp &
u1 iperf3 -c u6 -u -b 40M > u1.udp &
u2 iperf3 -c u7 -t 20 > u2.tcp &
u2 iperf3 -c u8 -u -b 40M > u2.udp &
```

性能测试的结果在 `u1.tcp, u1.udp, u2.tcp, u2.udp` 这四个文件中。

2.3.3 实现基于 UDP 和 TCP 选择的 SDN 路由

2.3.3.1 搭建拓扑

首先找到 `classic/create_topo.py` 这个文件，在配置好 mininet 的主机 A 中运行

```
sudo mn --custom create_topo.py --topo mytopo \
--switch ovs --controller remote,ip={ip of B} --link tc
```

其中 `ip` 项是控制器主机 B 的 ip 地址。

在 B 主机中运行

```
ryu run --observe-links  \
ryu.app.rest_router ryu.app.gui_topology.gui_topology \
ryu.app.ofctl_rest
```

这样一来整个拓扑已经搭建成功。实际上，这里的拓扑和传统路由时的拓扑没有区别。

2.3.3.2 生成路由

首先找到 `sdn-tcp-udp` 文件夹中的 `make_classic_route.py` 和 `make_sdn_route.py`，在主机 B 上运行

```
python make_classic_route.py \
python make_sdn_route.py
```

这会在控制器上创建基于最短路算法的静态路由和基于 UDP 和 TCP 的 SDN 路由。

此时路由已经创建好，但是，由于主机的默认网关均未进行配置，要手动配置默认网关。

在 mininet 的命令行中运行

```
u1 ip route add default via 172.18.0.2
u2 ip route add default via 172.18.1.2
u3 ip route add default via 172.18.2.2
u4 ip route add default via 172.18.3.2
u5 ip route add default via 172.18.4.2
u6 ip route add default via 172.18.2.2
u7 ip route add default via 172.18.2.2
u8 ip route add default via 172.18.2.2
```

这样一来路由就搭建成功了。

2.3.3.3 测试路由

在 mininet 的命令行中运行

```
pingall
```

得到输出如下：

```
mininet> pingall
*** Ping: testing ping reachability
u1 -> u2 u3 u4 u5 u6 u7 u8
u2 -> u1 u3 u4 u5 u6 u7 u8
u3 -> u1 u2 u4 u5 u6 u7 u8
u4 -> u1 u2 u3 u5 u6 u7 u8
u5 -> u1 u2 u3 u4 u6 u7 u8
u6 -> u1 u2 u3 u4 u5 u7 u8
u7 -> u1 u2 u3 u4 u5 u6 u8
u8 -> u1 u2 u3 u4 u5 u6 u7
*** Results: 0% dropped (56/56 received)
```

这样一来就可以确定路由已经正常工作。

2.3.3.4 性能测试

这里进行了两次性能测试，一次测试和传统路由相同，限制 udp 的速率为 $40Mbp/s$，一次把 udp 的速率限制到 $100Mbp/s$。

在 mininet 的命令行中运行

```
u3 iperf3 -s -D
u6 iperf3 -s -D
u7 iperf3 -s -D
u8 iperf3 -s -D

u1 iperf3 -c u3 -w 1M -t 20 > u1-40.tcp &
u1 iperf3 -c u6 -u -b 40M > u1-40.udp &
u5 iperf3 -c u7 -w 1M -t 20 > u5-40.tcp &
u5 iperf3 -c u8 -u -b 40M > u5-40.udp &

u1 iperf3 -c u3 -w 1M -t 20 > u1-100.tcp &
u1 iperf3 -c u6 -u -b 100M > u1-100.udp &
u5 iperf3 -c u7 -w 1M -t 20 > u5-100.tcp &
u5 iperf3 -c u8 -u -b 100M > u5-100.udp &
```

第一次性能测试的结果在 `u1-40.tcp`, `u1-40.udp`, `u5-40.tcp`, `u5-40.udp` 这四个文件中。

第二次性能测试的结果在 `u1-100.tcp`, `u1-100.udp`, `u5-100.tcp`, `u5-100.udp` 这四个文件中。

2.4 组员的分工与工作

1[1]

2.5 实验中遇到的挑战与问题及其解决方法

2.5.1 Ryu 控制器的内部问题

在金钰同学的计算机上测试时，Ryu 控制器的内置 `ryu.app.rest_router` 不能正常地进行工作。报错信息为 python 中的 `KeyError`，即一个哈希表的键不存在。这个问题并不源于 `ryu.app.rest_router`，真正的错误来源是 Ryu 控制器的内部组件。

在李晨曦同学的计算机上测试时，Ryu 控制器的内置 `ryu.app.gui_topology` 不能正常地工作。表现为浏览 `http://ip_of_B:8080` 后，应该出现拓扑图的区域空无一物。

这两个问题都涉及到 Ryu 控制器的内部问题。李晨曦同学通过重新安装了 Ryu 控制器解决了问题，金钰同学通过拷贝李晨曦同学主机上的的 Ryu 控制器并覆盖到相应位置解决了问题。

2.5.2 Mininet 内置 Shell 的不便

Mininet 内置的 Shell 语法类似于

$$host\ command$$

乍看之下，这没有什么问题。然而，实际上这有比较严重的问题。首先，这样的设计使得它无法正常地使用 Shell 脚本批量地对**不同主机**执行命令。这导致每次配置拓扑中主机的默认网关时都要手动输入 8 条命令。

其次，无论是 `&&` 操作符还是 \ 分行符号，在 Mininte Shell 中的语义均是『在一个主机上执行』。也就是说，

```
u1 ls . && u2 cd ..
```

会被理解为在 u_1 上执行

```
ls . && u2 cd ..
```

这样导致无法通过『复制粘贴写好的命令』来解决这个问题。

实际上，这个问题我们最终也未能解决，只能留待日后探索了。

第三章 实验分析

第四章 总结

参考文献:

[1] LONG J, SHELHAMER E, DARRELL T. Fully convolutional networks for semantic segmentation[C] // Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2015 : 3431 – 3440.