虚拟路由
项目报告
模拟网络层路由

学院：数据科学与计算机学院
专业：计算机科学与技术
年级：**2016** 级
组长（学号）：王锡淮（**16337236**）
组员（学号）：杨陈泽（**16337271**）
组员（学号）：肖遥（**16337258**）

目录

1 项目介绍

模拟网络层路由项目包含三个部分：

1. 基本 RIP 协议的实现
2. 基本 OSPF 协议的实现
3. 基本中心化路由协议的实现

在其中的每一个部分中，都实现了基本的路由功能以及对于基本算法缺点和不足的改进，比如，在基本 RIP 协议中，使用毒性逆转（split-horizon with poison inverse）、延时刷新（flush timer）、触发更新（triggered update）、抑制更新（holddown）方法缓解了无穷计数问题；在基本 OSPF 协议中，实现了多条等费用路径的路由选择，并且解决了路由循环问题和改善了路由震荡问题；在中心化路由协议中使用 Twisted 异步编程框架以减轻服务器进程的资源消耗等等。项目中 3 个部分使用的都是 UDP 协议以传输路由信息，使用 UDP 也节约了网络资源。

项目主页见https://github.com/Leo-xh/Virtual-Routing。

2 基本 RIP 协议

2.1 协议概述

本项目中实现的 RIP 协议采用了 RIPv1 和 RIPv2 的协议规定的一部分，是一个基于 DV 算法的网络层路由协议。其主要算法是 Bellman-Ford 算法，路由费用以路由跳数作为度量，运行该协议的每个路由器能够计算出到其余路由器的下一跳路由以及路由费用。定义了请求报文和响应报文两种报文格式。为了尽可能避免环路和无穷计数问题，使用的机制包括毒性逆转（split-horizon with poison inverse）、延时刷新（flush timer）、触发更新（triggered update）、抑制更新（holddown）。在协议的扩展中，实现了该协议下适用的 traceRoute 功能。

2.2 算法

2.2.1 距离矢量表更新

距离矢量表计算主要基于 Bellman-Ford 算法。在基本的 DV 算法中，路由器每次收到邻居发来的距离矢量时，会使用 Bellman-Ford 的更新公式更新自己的距离矢量表：

$$dist_{self}(dest) = \min_{neighbour}\{dist_{neighbour}(dest) + metric(self, neighbour)\}$$

为此，路由器需要在维护自己距离矢量表的同时，保存邻居发来的距离矢量表。

在 RIP 协议中，路由器只需保存和维护自己的距离矢量表。在收到邻居发来的距离矢量时，会将其与自己当前的距离矢量表进行比较，而不是用所有邻居的距离矢量表重新计算和比

较。具体而言，RIP 协议使用的更新公式为（假设收到邻居 nb 的距离矢量）：

$$dist_{self}(dest) = \begin{cases} \min(dist_{self}(dest), dist_{nb}(dest) + metric(self, nb)) & , nextHop_{self}(dest) \neq nb \\ dist_{nb}(dest) + metric(self, nb) & , nextHop_{self}(dest) = nb \end{cases}$$

采用这种更新方式避免了路由器存储邻居距离矢量表的开销，也减少了更新所需的计算量。不足之处在于最短路径费用增加时，不能马上从其余路径中找到更优的并将其替代，需要等待其他邻居发送距离矢量。（事实上，为了避免环路和无穷计数问题，原最短路径费用增加后从其余路径中找替代的更新方式是不被允许的。）

2.2.2 环路与无限计数避免机制

在本项目的实现中，为了尽可能避免环路与无限计数问题，使用了以下 4 种机制：

- 毒性逆转：一个路由器向邻居发的距离矢量中，下一跳为该邻居的项的费用改为无穷（16）。
- 延迟刷新：一个路由器在与一个邻居失去连接（链路中断）后，不立即将其从距离矢量表中清除，而是延迟一段时间。在这段时间里，链路中断的信息能够告知其他路由。
- 触发更新：一个路由器在距离矢量表更新后，将更新的项立即向所有邻居发送。特别地，当一条链路断开后，触发更新会沿着包含这一链路的所有最短路径传播，从而使这些路径被毒化（即在路由器的转发表中费用为无穷）。
- 抑制更新：一个路由器在其一条最短路径被毒化后的一段时间内，忽略收到的一切声称能到达该路径目的地的距离矢量。这是为了避免在网络不稳定时使用邻居过时的路由信息进行更新。

2.3 协议特性

2.3.1 协议流程

本项目中实现的 RIP 协议主要包含的流程如下：

1. 路由器在进入区域时，向所有邻居发送 Request 报文，邻居则返回一个 Response 报文，里面含有邻居的距离矢量表。
2. 该路由器会定时地向所有的邻居发送 Response 报文，即自己的距离矢量表。
3. 收到一个 Response 报文意味着要维持该邻居的状态；而如果在一段时间内没有收到某个邻居路由器的 Response 报文，则认为与该路由器的链路中断。
4. 当收到其它邻居路由器的距离矢量表时，根据该信息更新自己的距离矢量表；当自己的距离矢量更新时，向所有邻居发送自己新的距离矢量表（只发送更新的部分）。

2.4 报文格式

本实验中使用的普通报文普通数据报文、Request 报文、Response 报文、TraceRoute 报文和 Echo 报文如下：

普通数据报文：

Command
Source address
Destination address
payload

Request 报文：

Command	Version	Routing domain
Source address		
Address family	Route tag	
address		
Next hop		
metric		

Response 报文：

Command	Version	Routing domain
Source address		
Address family	Route tag	
address		
Next hop		
metric		
repeat of last 17 bytes		
...		

TraceRoute 报文：

Command
Source address
Destination address
Count

Echo 报文：

Command
Source address
Destination address
Local address

报文项解释：

1. Command（1 Byte）：指明这条报文的类型。

   0：普通数据报文。

   1：请求报文。

   2：响应（包含路由表）报文。

   3：TraceRoute 报文。

   4：Echo 报文。

2. Version（1 Byte）：RIP 协议的版本。
3. Routing domain（2 Bytes）：指明这是 RIP 中的哪一步。
4. Source address（6 Bytes）：指明发送方的源地址和监听端口。
5. Address family（2 Bytes）：使用什么作为地址，使用 IP 该项为 2。
6. Route tag（2 Bytes）：本项目中不用。
7. address（6 Bytes）：ip 和端口号。
8. Next hop（6 Bytes）：下一跳路由的 ip 和端口。
9. metric（1 Byte）：代价度量（跳数）。
10. Local address（6 Bytes）：指明 Echo 报文发送者的源地址和监听端口。
11. Count（1 Byte）：路由跳数。

2.5 协议扩展

1. 应对网络环路和无穷计数问题

   - 网络中一条链路断开后，路由器可能会收到一条经过了自身的路径，进而可能产生环路。

- 环路的产生会使环路中的路由器对该路径的费用估计逐渐增大，直到无穷，称为“无穷计数”。无穷计数最终会使环路失效，但过程漫长，显著影响收敛速度。
- 为了尽量避免以上情况，采用了毒性逆转、抑制更新、触发更新和延迟刷新等机制。

2. TraceRoute

- 基于 RIP 路由 16 跳不可达这一机理，向目的地发跳数为 1-16 共 16 个报文，当一个路由器收到该报文，则提取其中的跳数，并将它减一。
- 当跳数为 0 时，向报文的源地址返回一个 Echo 报文；否则将跳数打包进原报文中，继续向目的地转发。
- 源地址路由器收集所有返回的 Echo 报文，如果 5s 内收到目的地址返回的 Echo 报文，则输出该路径；否则认为该目的地址不可达。

2.6 协议实现

本 RIP 协议基本由 Python 实现，主要利用的是 socket 库进行套接字编程并采用的是 UDP 协议、struct 库进行数据的打包和解包、threading 库管理线程和定时器。

实现方法实现了一个 RIP 协议类，处理内部流程和为外部调用留出了收发数据、查询路由路径等接口。

2.7 结果

测试所用的拓扑图以及路由结果如下图1和2所示。

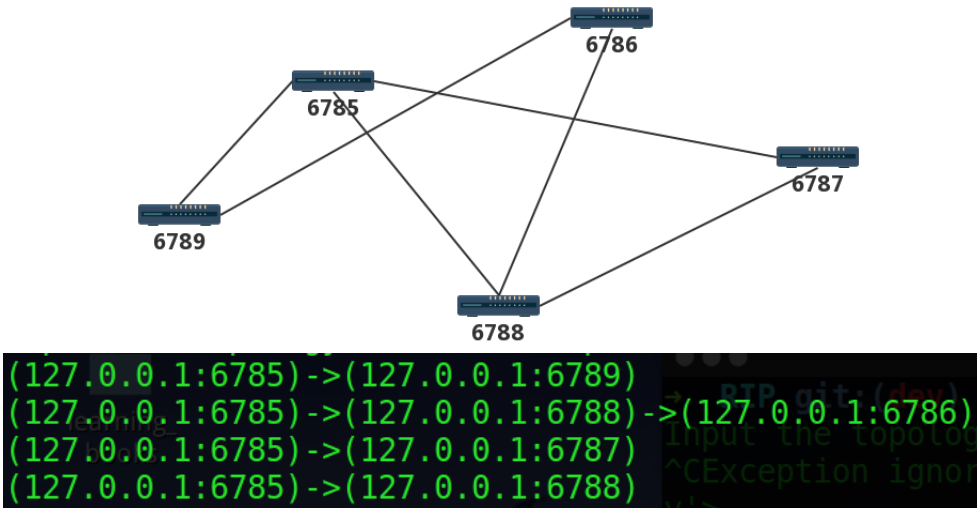


图 1: RIP 测试样例一

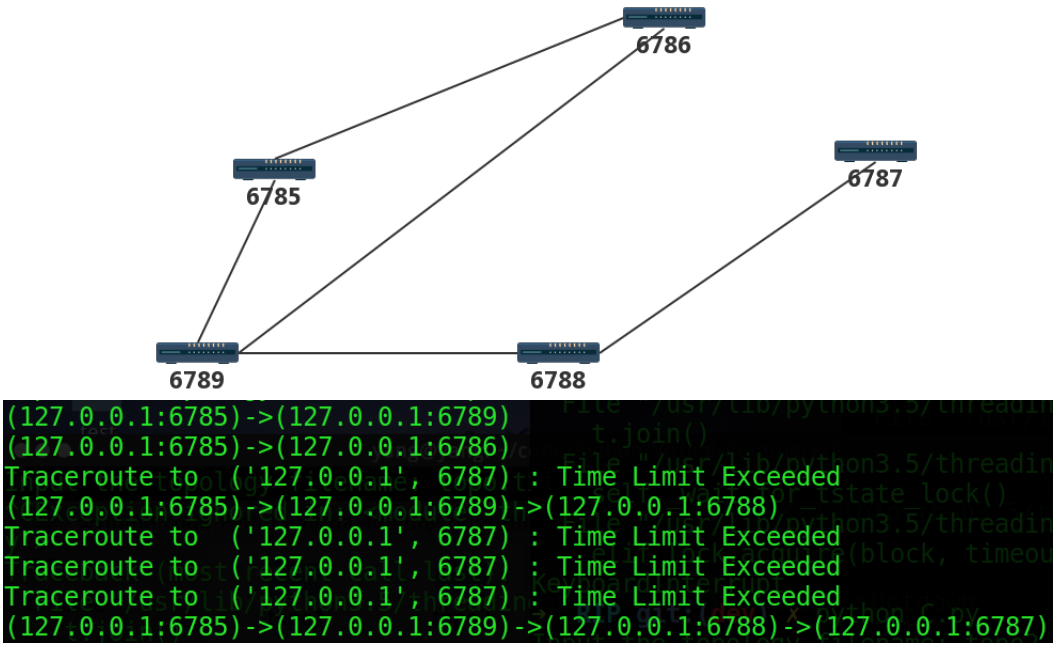


图 2: RIP 测试样例二

成功找出了其中的跳数最小的路径。

2.8 总结

- 原生的 DV 算法本身可能产生路由环路的问题，而我们的 RIP 协议中采用了多种机制尽可能解决这种问题。
- 原生的 DV 算法需要保存所有邻居的距离矢量表，而采用 RIP 协议的更新公式，避免了路由器存储邻居路由表的开销，而且减少了路由器更新时的计算量。

3 基本 OSPF 协议

3.1 协议概述

本项目中实现的 OSPF 协议是一个基于链路状态的路由协议，运行该协议的每个路由器能够根据收到的所有链路信息计算出一个最短路径树，构造对应的转发表。其链路的权重的赋值取决于带宽，算法使用的是 Dijkstra 算法和 RPF 广播算法，定义了 Hello 和 LS（link-state）两种报文格式。实现了多条等费用路径的路由选择，并且解决了路由循环问题和改善了路由震荡问题。在协议的扩展中，实现了该协议下适用的 TraceRoute 功能。

3.2 算法

3.2.1 路由计算算法

在原生 LS 算法中，一个路由器的路由计算的方法是：以自身为起点，执行 Dijkstra 算法以得到一棵最短路径树，然后根据最短路径树得到转发表。采用这种方法，计算出的到每个目的地的最短路径是唯一的。

为了实现多条等费用最短路径选择，本项目采用以下算法过程：

1. 对于路由器的每个邻居 nb，以 nb 为起点执行 Dijkstra 算法，得到 nb 到其他路由器的最短路径距离 $dist_{nb}$。

2. 使用 Bellman-Ford 更新公式，计算自身到其他路由器的距离：

$$dist_{self}(dest) = \min_{nb}\{dist_{nb}(dest) + metric(self, nb)\}$$

3. 对于每一个目的地 $dest$，依次判断每个邻居是否能作为下一跳（即该邻居是否在一条最短路径上）。判断一个邻居 nb 能作为下一跳的依据是：nb 能到达 $dest$ 并且满足公式

$$dist_{self}(dest) = dist_{nb}(dest) + metric(self, nb)$$

将符合条件的邻居加入转发表中去往 $dest$ 的可选下一跳列表。

3.2.2 广播算法

使用了 RPF（Reverse Path First）算法，收到一个广播数据包的路由器会向来路以外的路由器转发该广播数据包当且仅当来路位于广播源和当前路有器的任一条最短路径上。

3.3 协议特性

3.3.1 协议流程

本项目中实现的 OSPF 协议主要包含的流程如下：

1. 路由器在进入区域时，向所有邻居发送 Hello 报文（该报文用于向邻居声明自己的存在），再向该区域内的路由器广播自己的链路信息。

2. 该路由器会定时地发送 Hello 报文和广播链路信息。

3. 收到一个 Hello 报文意味着要维持该邻居的状态；而如果在一段时间内没有收到某个邻居路由器的 Hello 报文，则认为该路由器故障或离开该区域。

4. 当自身的链路信息发生改变时，广播链路信息；当收到其它路由器的链路信息时，根据该信息更新自己的关于该区域的链路信息数据库。

3.4 报文格式

该协议中包含的 Hello 报文、LSU 报文、TraceRoute 报文、Echo 报文、普通数据报文格式如下：

Hello 报文。

Command
Source address
metric

LSU 报文（广播报文）。

Command
Source address
Transmit address
Destination address
metric
repeat of last 8 bytes
. . .

TraceRoute 报文

Command
Source address
Destination address
Count

Echo 报文

Command
Source address
Destination address
Local address

普通数据报文的格式与在 RIP 中普通数据报文的格式相同。

报文项解释：

(1) Command（1 Byte）：指明这条报文的类型。

0: 普通数据报文

1: Hello

2: LSU（Link State Update）

3: TraceRoute

4: Echo

(2) Source address（6 Bytes）：指明发送方的源地址和监听端口。

(3) Destination address（6 Bytes）：指明目的地的地址和监听端口。

(4) Transmit address（6 Bytes）：指明广播中转发者的地址和监听端口。

(5) Local address（6 Bytes）：指明 Echo 报文发送方的源地址和监听端口。

(6) metric（2 Bytes）：代价度量。

(7) Count（1 Byte）：路由跳数。

3.5 协议扩展

1. 应对路由循环

- 如果所有的路由器并非按照相同的拓扑图工作，则可能形成路由环路。因为每个路由器节点的最短路径树和路由表的计算是独立的，不与任何其他节点进行交互。例如，两个相邻节点可能互相认为对方是到达给定目的地的最佳路径上的下一跳，从而任何发往目的地的数据包将在两者之间循环。涉及两个以上节点的路由环路也是可能的。
- LS 算法本身是不会产生路由自环的，因为上述路由环路会在各个路由器发出的链路信息通告完全到达其他路由器后消失。但在完成信息通告之前，需要有一定的方法避免因为环路造成的拥塞。
- 本项目中的解决方法是每个路由器缓存最近 10 个到达数据包的 md5 码，如果一个新到数据包的 md5 码与缓存的重复，则将其丢弃。缓存 md5 码的数量不能太多是因为数据包可能在之后重传。

2. TraceRoute

   - 向目的地发跳数为 1-32 共 32 个报文，当一个路由器收到该报文，则提取其中的跳数，并将它减一。
   - 如果跳数为 0，则向报文的源地址返回一个 Echo 报文；如果跳数大于 0 且报文的目的地址不等于当前路由的地址，将跳数打包进原报文中，继续向目的地转发；否则，判定为多余的 traceRoute 报文，将之舍弃。
   - 源地址路由器收集所有返回的 Echo 报文，如果 5s 内收到目的地址返回的 Echo 报文，则输出该路径；否则认为该目的地址不可达。

3.6 协议实现

本 OSPF 协议基本由 Python 实现，主要利用的是 socket 进行套接字编程、struct 库进行数据的打包和解包、threading 库管理线程和定时器。

实现方法实现了一个 OSPF 协议类，处理内部流程和为外部调用留出了收发数据、查询路由等接口。

3.7 结果

测试所用的拓扑图以及路由结果如下图3和4所示。

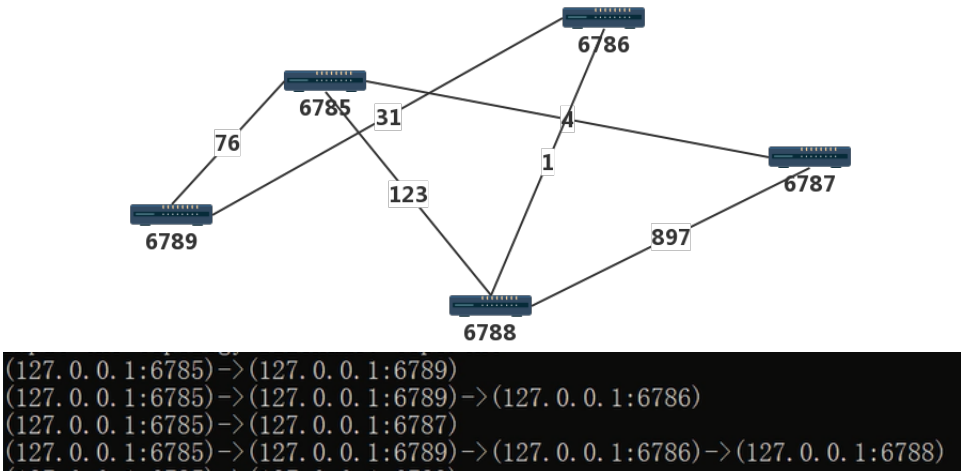


图 3: OSPF 测试样例一

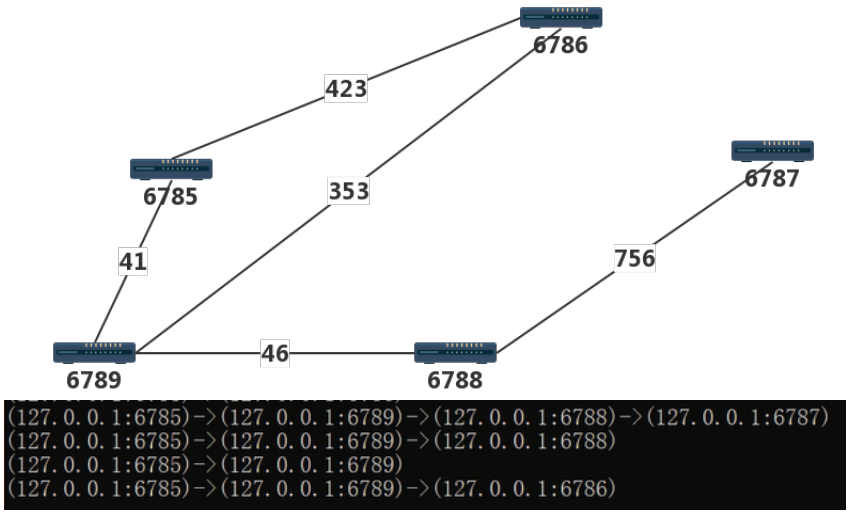


图 4: OSPF 测试样例二

成功找出了其中的最短路径。

3.8 总结

- 本项目中实现的基本 OSPF 协议淡化了其区域概念，简化了报文类型和路由器类型，同时也简化了路由器状态。
- 原生的 LS 算法会产生路由震荡问题，而 OSPF 协议中由于链路费用固定，没有路由震荡问题。迭代过程中的路由循环问题则在本项目的实现中得以缓解。

4 基本中心化路由协议

4.1 协议概述

中心化路由协议是一个基于链路状态的网络层路由协议，使用的是 Dijkstra 算法，该协议中定义了一个服务器，负责接收区域内运行该协议的路由器的链路信息并且根据这些信息计算出整个区域内路由器的最短路径树，构造出每个路由器的转发表，并发送给对应路由器。并且在服务器中能够获取指定路由器对的最短路径。

4.2 算法

在中心服务器中，对每一个路由器使用与 OSPF 中相同的方法计算其转发表，在此不再赘述。唯一区别在于，为了避免重复计算，先以每个路由器为起点执行 Dijkstra 算法，然后再计算每个路由器的转发表。（否则，在一个路由器 r 的每个邻居的计算过程中，都需要执行一次以 r 为起点的 Dijkstra 算法。）

4.3 协议特性

4.3.1 协议流程

1. 路由器加入某个区域后，向邻居发送 hello 报文，并将定期发送 hello 报文。
2. 路由器定期向服务器发送 LS 报文，服务器接收后计算对应信息并定时向路由器发送转发表。
3. 服务器和路由器都有失效机制，当应当发送信息的路由器在一段时间没有发送信息后将其置为失效。

4.4 报文格式

基本 Centralized 协议中包含的报文格式有 hello 报文、LS 报文、转发表报文三种报文格式。

LS 报文：

Command
Source address
Destination address
Metric
Repeat of last 8 bytes
...

Forward Table 报文：

Command
Destination address
nextHop
Repeat of last 12 bytes
...

Hello 报文：

Command
Source address
Metric

普通报文：

Command
source address
destination address
payload

报文项解释：

1. Command（1 Byte）：指明这条报文的类型。

   0: 普通报文

   1：hello

   2：LS

   3：转发表

2. Source address（6 Bytes）：指明发送方的源地址和监听端口。
3. Destination address（6 Bytes）：指明目的地的源地址和监听端口。
4. nextHop（6 Bytes）：下一跳 IP 地址和监听端口。
5. Metric（2 Bytes）：代价度量。

4.5 协议扩展

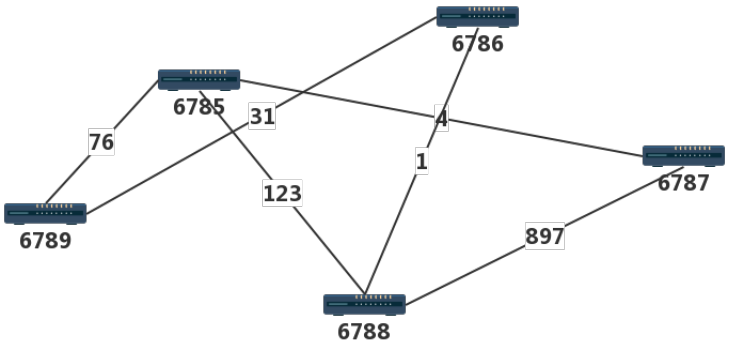
查找路径功能。在服务器中，实现了指定路由器对查找两者之间的最短路径的功能。

4.6 协议实现

1. 客户端程序使用 python 编写，主要使用的是 socket、threading 等库。
2. 使用 Twisted 异步编程框架编写服务器端。使用 Twisted 框架编写的服务器端具有低功耗、资源利用率高的特点，同时事件驱动的内在逻辑也符合该服务器的功能特性。

4.7 结果

测试结果如下图5和6所示：




```
('127.0.0.1', 6785)->('127.0.0.1', 6789)
('127.0.0.1', 6785)->('127.0.0.1', 6789)->('127.0.0.1', 6786)->('127.0.0.1',
6788)
('127.0.0.1', 6785)->('127.0.0.1', 6787)
('127.0.0.1', 6785)->('127.0.0.1', 6789)->('127.0.0.1', 6786)
('127.0.0.1', 6789)->('127.0.0.1', 6786)->('127.0.0.1', 6788)
('127.0.0.1', 6789)->('127.0.0.1', 6785)->('127.0.0.1', 6787)
('127.0.0.1', 6789)->('127.0.0.1', 6786)
('127.0.0.1', 6789)->('127.0.0.1', 6785)
('127.0.0.1', 6788)->('127.0.0.1', 6786)->('127.0.0.1', 6789)
('127.0.0.1', 6788)->('127.0.0.1', 6786)->('127.0.0.1', 6789)->('127.0.0.1',
6785)->('127.0.0.1', 6787)
('127.0.0.1', 6788)->('127.0.0.1', 6786)
('127.0.0.1', 6788)->('127.0.0.1', 6786)->('127.0.0.1', 6789)->('127.0.0.1',
6785)
('127.0.0.1', 6787)->('127.0.0.1', 6785)->('127.0.0.1', 6789)
('127.0.0.1', 6787)->('127.0.0.1', 6785)->('127.0.0.1', 6789)->('127.0.0.1',
6786)->('127.0.0.1', 6788)
('127.0.0.1', 6787)->('127.0.0.1', 6785)->('127.0.0.1', 6789)->('127.0.0.1',
6786)
('127.0.0.1', 6787)->('127.0.0.1', 6785)
('127.0.0.1', 6786)->('127.0.0.1', 6789)
('127.0.0.1', 6786)->('127.0.0.1', 6788)
('127.0.0.1', 6786)->('127.0.0.1', 6789)->('127.0.0.1', 6785)->('127.0.0.1',
6787)
('127.0.0.1', 6786)->('127.0.0.1', 6789)->('127.0.0.1', 6785)
```

图 5: 中心化路由测试样例一

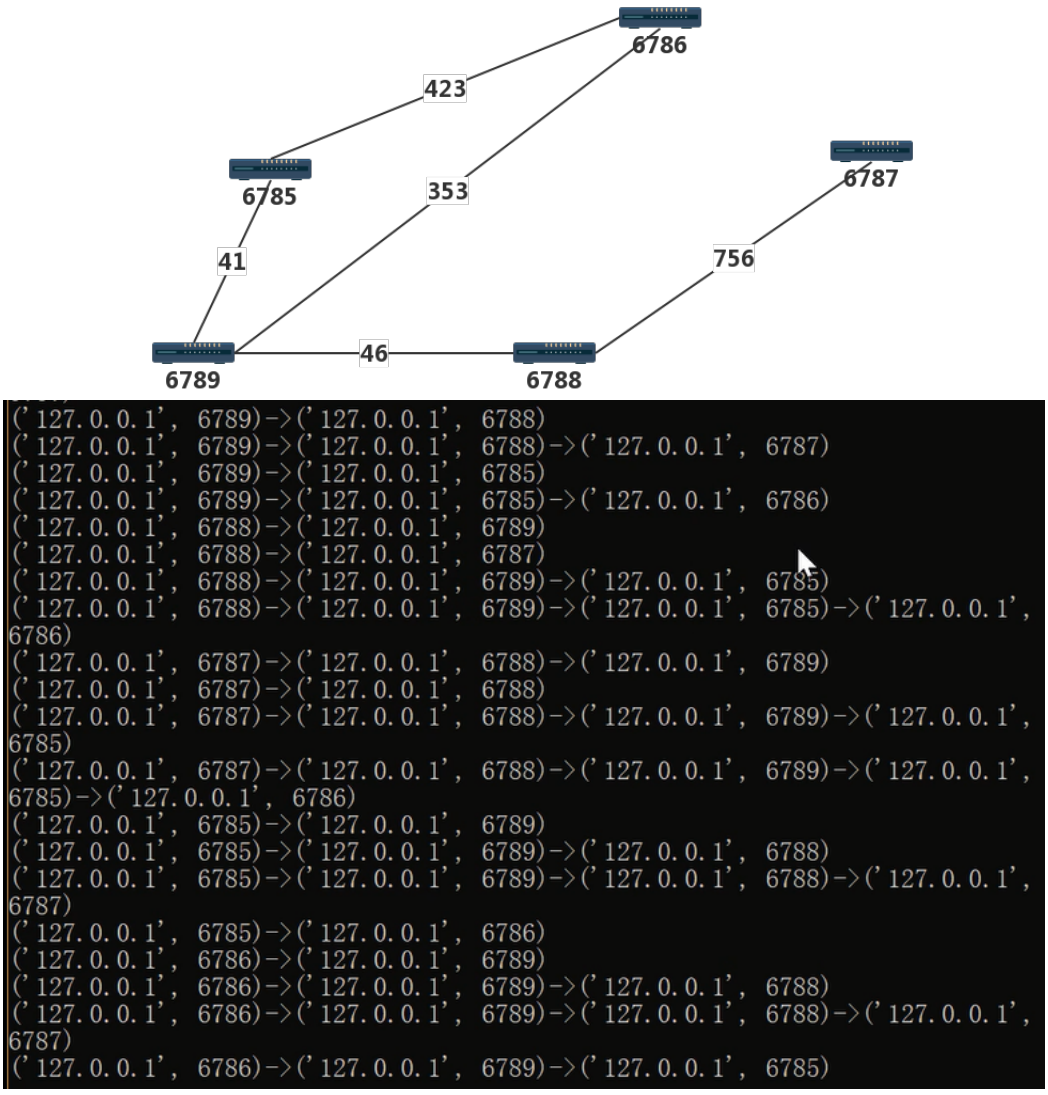


图 6: 中心化路由测试样例二

成功找出了全部最短路径。

4.8 总结

- 由于在中心化路由协议中每个路由器的转发表都是由服务器计算并且同时几乎获得，即转发表基于同一张图结构，则不存在路由循环、路由震荡等问题。
- 不能及时地检测到路由的变化，造成某些数据报文发送的浪费。

5 安装与部署

- 可执行文件的安装与部署：

  1. 三个项目都已打包成可执行文件 (.exe)，直接在 Windows 系统上双击打开，便可以使用，无需安装 python 及其各个使用到的 python 模块。
  2. 只需打开项目中 executable 文件夹中的 A.exe 到 E.exe 文件（中心化项目还需要打开 Server.exe 文件），并在各个窗口中输入：topo.txt 或 topo2.txt，便自动按照拓扑图 1 或拓扑图 2 部署好 5 个路由，并且 A 路由执行 traceRoute，测试到其余路由的路径；
  3. 或者，可用 Windows8 或 Windows10 的 powershell 命令行工具，执行各个项目中 executable 文件中的 run.ps1 脚本文件（.\run.ps1），便自动打开各个需要的 exe 文件，只需在各个窗口中输入：topo.txt 或 topo2.txt 即可。

- 源代码的安装与部署：

  1. 首先需要解决项目所需要的依赖环境，可在 Linux 或 Windows 环境下，安装 python3.0 及 3.0 以上版本，然后安装 python3-pip，再用 pip 直接下载安装或者到网站https://pypi.org/上下载 wheel 文件或者源代码，安装 twisted 模块（中心化项目需要）。
  2. 然后，便可以在 5（或 6）个终端/命令行进行测试，先进入项目的 executable 文件夹中，分别输入：python A.py，到：python E.py（中心化项目还需要在第六个终端或命令行输入 python Server.py）；最后再在各个终端或者命令行输入：topo.txt 或 topo2.txt，便可以完成 5 个路由器按照拓补图 1 或拓扑图 2 的安装和部署，并由 A 路由器执行 traceRoute，测试到其余路由器的路径。

6 项目管理记录

6.1 管理概述

该项目放在 github 上，见https://github.com/Leo-xh/Virtual-Routing，组内三人共同开发。

6.2 阶段管理

任务确定为设计并实现基本的 RIP 协议、OSPF 协议、中心化路由协议。任务确定前进行资料检索，总结原生 DV 和 LS 算法以及可能的解决方案，以及现存的协议版本。任务确定后检索实现和设计相关的资料。

6.3 人员分工

任务	主要参与者
协议设计	全体组员
代码框架搭建	王锡淮
算法设计以及实现	肖遥
通信功能	杨陈泽
线程、定时器以及数据结构的管理功能	王锡淮
调试	全体组员
项目打包以及文档编写	全体组员

7 总结

7.1 三种路由方式的比较

比较方面	RIP	OSPF	Centralized
算法基础	Bellman-Ford	Dijkstra	Dijkstra
算法结构	迭代的、分布的	非迭代、基于全局信息	非迭代、基于全局信息
时间复杂性	$O(n)$（一次更新）	$O(mn^2)$（m 为邻居数）	$O(n^3)$
收敛速度	慢	快	快
跳数的限制	15 跳	无	无
可扩展性	较差	较好	较差
环路	短时间内容易出现	不会出现	不会出现

由此，得出三者主要的优缺点。

- RIP 协议最大的优点就是简单，而其缺点也明显，就是路由跳数的限制与路由器之间交换完整的路由信息的开销太大。原本的 DV 算法还有其“坏消息传播慢”的特点，使得收敛过程太长，但是本项目中实现的 RIP 协议通过解决无穷计数的策略使得“坏消息”传播也快。以上的缺点主要在大型网络中表现出来，由此 RIP 协议更加适合使用在小型网络中。
- 本项目中的 OSPF 协议的优点在于长期来看不存在路由环路问题，短期的路由环路现象也采取了措施进行缓解，还有就是其收敛速度较快，可以采用多条等费用路径。其主要缺点是实现较为复杂。
- 对于中心化路由协议，其中的每个路由器的转发表都是由服务器计算并且同时几乎获得，即转发表基于同一张图结构，则不存在路由循环、路由震荡等问题。但是中心化路由协

议具有滞后性，虽然该滞后性也使得路由更加稳定，但是不能及时地检测到路由的变化，造成某些数据报文发送的浪费。

7.2 关于在应用层模拟网络层

要实现在应用层模拟网络层的协议，本项目以 IP 和其监听端口为路由索引，这是因为要在一台电脑上运行多个路由器程序进行模拟。使用 UDP 协议发送路由信息报文时，注意控制 UDP 协议的源地址和目的地址之间一定要有一条虚拟链路，这样才能模拟链路层的信息传递。

A 参考文献

1. RFC2328, Open Shorst Path First Protocol.
2. RFC2453, Route Information Protocol.
3. https://en.wikipedia.org/wiki/Open_Shortest_Path_First, Wikipedia for OSPF.
4. https://en.wikipedia.org/wiki/Routing_Information_Protocol, Wikipedia for RIP.
5. http://www.clnchina.com.cn/discussion_forum/2010/0721/8402.shtml, Four Timer in RIP.
6. https://blog.csdn.net/friendbkf/article/details/48808533, LS 和 DV 路由协议的分析与比较.