**A Policy-Based Routing (PBR) Router**

**based on Distance-Vector Algorithm**

**佘宇峰 3220190860 2019/11/16**

## 1 实验目的

本实验将设计一个基于距离向量算法的路由器，对 RIP动态路由协议进行模拟。实验需要对RIP算法进行一定的改进，使RIP路由算法能够基于策略选择最短的路径发送包，并通过模拟实验对RIP算法有更深的了解。

## 2 距离-向量算法

距离向量路由算法（Distance Vector Routing）是ARPANET网络上最早使用的路由算法，主要在RIP协议中使用。距离向量路由算法的基本思想如下：每个路由器维护一个距离向量表，表中列出了当前已知的到每个目标路由节点的最短距离，以及所使用的链路。路由器通过与邻居节点交换信息而不断更新自己的路由表，最终每个路由器都了解到达其余各个目的节点的最佳路径及跳数。每个距离向量表项包括两部分：到达目的结点的最佳输出线路，和到达目的结点所需时间或距离，通信子网中的其它每个路由器在表中占据一个表项，并作为该表项的索引。每隔一段时间，路由器会向所有邻居结点发送它到每个目的结点的距离表，同时它也接收每个邻居结点发来的距离表。这样以此类推，经过一段时间后便可将网络中各路由器所获得的距离向量信息在各路由器上统一起来，这样各路由器只需要查看这个距离

|  |  |  |
| --- | --- | --- |
| 目的地 | 经由节点 | 开销 |
| A | **—** | **0** |
| B | **C** | **2** |
| C | **D** | **3** |

**表2.1 路由节点A的距离向量表**

向量表就可以为不同来源分组找到一条最佳的路由。表2.1为路由节点A的距离向量表，距离向量表的表项二表示节点A到节点B的最小损耗路径的开销为2，其使用的链路的下一跳节点为C。

## 3 实验要求

* 完成一个路由器节点模拟程序，程序名为router。
* router程序可以在一台机器上启动多个，每次启动router相当于开启了一个新的路由器进程，即部署了一个新的路由器节点。终止一个router进程相当于关闭或移除了一个路由器节点。
* 路由器节点启动命令：**./router.exe routerID myport port1 port2 port3……**

**routerID**：路由器节点ID

**myport**：该路由器节点的UDP节点号，用于发送和监听数据报和路由表

**Port1、port2、port3…**：该路由器节点的邻居节点的节点号。

* 路由器进程的路由表项基于距离-向量算法和特殊的策略规则进行更新。
* 路由器表的交换基于UDP SOCKET实现。
* 路由器能够提供以下命令：

**N** ---- 打印该路由节点邻居的活跃性

**T** ---- 打印路由表

**D** **n** ---- 向节点号n代表的路由器节点发送数据报

**P K *n1 n2 … nk*** *-*--- 优先路由器序列P

**K:** 优先路由器序列中路由器的个数

***n1 n2 … nk* :** 优先路由器序列中各路由器的端口号

若路由器节点设置了优先路由序列，则路由器发送数据报时停止使用DV最短路径，而是需要在考虑优先路由节点序列策略的基础上选择一条最短路径（即选择一条经过n1、n2、n3…路由节点的最短路由路径）

**R *n*** ----拒绝节点R

**n:** 拒绝节点的端口号

若路由器开启此命令添加拒绝节点，则路由器发送数据报时停止使用DV最短路径，而需要考虑拒绝节点R（即选择一条不经过路由器节点n的最短路由路径）

**Q** ---- 移除路由器节点

* 路由器节点每6秒向它的邻居节点广播一次路由表，路由器节点根据它接收到的邻居路由表对自己的路由表进行更新；路由节点能够检测其邻居节点的活性，若18秒内未收到邻居路由节点的路由表消息，则路由器认为该邻居节点已被移除，该邻居节点不可到达。
* 路由节点能够处理故障和恢复；能够避免无限环路导致的路由跳数无限增加；需要解决慢收敛问题，可使用的方法包括水平分割或毒性逆转。

## 4 程序设计语言/平台/工具

* **程序设计语言：**Golang
* **IDE：**IntelliJ IDEA
* **平台：**Windows 10

## 5 实验设计思路

### 5.1 多线程设计和端口设计

在路由器节点启动后四个线程同时工作：活跃性检测线程、路由表同步线程、UDP监听线程、主线程，四个线程协同工作，共同实现路由器的各部分功能；此外，由于路由器节点需要基于UDP SOCKET同时兼备发送数据和接收数据的功能，因此需要至少两个端口（监听端口会阻塞）。本程序中路由器节点对外显示的端口为监听端口myport，资料发送端口号实为myport+100。节点对外只表现出一个端口，发送和接收数据时接收端口和发送端口的转换在程序内部进行，此过程对外部透明，四个线程的功能如下所述。

* **主线程：**主线程用于用户与路由器节点的交互，执行路由器的各个命令（打印路由表、打印邻居节点的活跃性、发送数据报、设置路由优先队列及拒绝节点等等）
* **UDP监听线程：**基于UDP SOCKET监听端口，接收数据报或路由表资料，并根据接收的信息类型做相应处理（更新路由表、转发数据报等等）
* **路由表同步线程：**设置定时器，每隔六秒基于UDP协议向邻居节点的监听端口发送本节点的路由表资料。
* **活跃性检测线程：**用于检测邻居节点的活跃度。路由器节点为每个邻居设置一个活跃性标志位，初始值为1。当路由器节点收到某节点的路由表信息时，将该邻居的标志位置为1；活跃性检测线程设置定时器，每隔十八秒检查路由器节点的邻居活跃性标志位，若标志位为1，则将标志位设为0；若标志位为0，则活跃性检测线程认为邻居节点被移除，在路由表中将邻居节点设置为不可到达。

### 5.2 路由表更新算法及基于策略的包转发

本实验的RIP算法基于最短距离（跳数）实现，路由器节点到自己的最小距离为0，到直接邻居的最小距离为1，并以此类推。RIP的跳数度量值取值范围为0-15之间的整数，距离大于等于16的目的节点被定义为无穷大，即永远无法到达。若不考虑路由优先节点序列及拒绝节点等路由策略，RIP路由表结构如表2.1所示，路由表项更新规则如下所述：

* 如果更新的某路由表项在路由表中没有，则直接在路由表中添加该路由表项
* 如果路由表中已有相同目的网络的路由表项，且路由信息的来源端口相同，那么无条件根据最新的路由信息更新其路由表
* 如果路由表中已有相同目的网络的路由表项，但来源端口不同，则要比较它们的度量值，将度量值较小的一个作为自己的路由表项
* 如果路由表中已有相同目的网络的路由表项，且度量值相等，保留原来的路由表项

然而，由于RIP中每个节点仅保存局部的路由信息，上述的路由规则无法实现基于策略的路由路径选择，因此本实验对RIP路由表的结构和路由表更新算法进行改进，使其能够实现基于策略的RIP路由算法。表5.1表示改进RIP中节点A的路由表

|  |  |  |  |
| --- | --- | --- | --- |
| 目的节点 | 跳数 | 经由节点 | 路径 |
| A | 0 | A | A |
| B | 1 | B | A B |
| C | 2 | B | A B C |

表5.1 节点A的改进RIP路由表

|  |  |  |  |
| --- | --- | --- | --- |
| 目的节点 | 跳数 | 经由节点 | 路径 |
| B | 0 | B | B |
| A | 1 | A | B A |
| C | 1 | C | B C |

表5.2 节点B的改进RIP路由表

结构，在改进的路由表中，我们新增了路径表项并在更新路由表的同时对路径表项进行更新。表5.3的第三行表示节点A到目的节点C的一条路径的跳数为2，该路径下一跳节点为B，数据传输的路径为[A B C]。由于路由表中新增了路径表项，因此节点能够基于路由表选择经过某些特定节点（优先节点序列）且不经过某节点（拒绝节点）的最短路由路径向目的节点发送数据报。改进的RIP路由表基于以下步骤对路由表进行更新：

* 每个路由节点路由表的初始状态如表5.3所示。节点A的路由表最初只有一条路由信息，即A到它本身的路由信息：A到A的一条路径的跳数为0，该路径下一跳节点为A，数据报发送路径为[A]。当接收到来自邻居节点的路由信息时，对路由表中每条路由信息进行如下判断并对自己的路由表按照以下规则进行更新。

|  |  |  |  |
| --- | --- | --- | --- |
| 目的节点 | 跳数 | 经由节点 | 路径 |
| A | 0 | A | A |

表5.3 改进RIP路由表的初始状态

1. 若邻居路由表的一条路由信息的目的节点是本节点，则忽略此路由信息。

2. 若本节点已存在于一条路由信息的路径中，则忽略此路由信息（防止环出现）否则将本节点添加至该条路由信息的路径表项中，并转3。

3. 若本节点路由表中存在一条具有相同目的节点、相同路径、相同经由节点的路由信息，且路由信息的来源端口相同，则无条件更新本节点路由表的跳数。

4. 若本节点路由表中没有此条路由信息，则将该路由信息加入本地路由表。

* 基于上述路由表结构和路由表更新算法，改进RIP算法能够将到达其余节点的路径（包括最短路径和非最短路径且无包含环路的路径）全部记录下来。在基于优先节点序列和拒绝节点的策略向目的节点发送数据报时，由于可以从路由表中得知全部的路径信息，因此节点能够选择符合策略要求的路由路径，按照该路径进行数据报的发送，同时此路径信息将保存在发送数据报中，收到该数据报的路由节点按照此路径信息选择接收或转发数据报。

### 5.3 毒性逆转

为解决RIP慢收敛问题，本实验采用毒性逆转方法。模拟节点在向邻居节点广播路由表时，不直接将自己的路由表发送给邻居，而是选择在向某一邻居节点发送路由表时，将该邻居节点作为来源端口的路由信息的跳数设置为∞（16跳），将修改后的路由表发送给相应的邻居节点，从而避免环路问题。

## 6 数据结构

* **路由器数据结构**

|  |  |
| --- | --- |
| 数据表项 | 解释 |
| RouterId | Int 路由器ID号 |
| ListenPort | Int 监听端口 |
| SendPort | Int 发送端口= ListenPort+100 |
| Adjcents | Int[] 邻居节点 用相应的监听端口表示 |
| RouterTable | RouterTableItem[] 路由表 |
| ActiveFlag | Map[int]int 邻居节点的活跃性标志位 |
| PriorityRouters | []int 优先节点序列 |
| RefuseNNode | []int 拒绝节点 |

* **路由表项结构**

|  |  |
| --- | --- |
| 数据表项 | 解释 |
| DesPort | Int 目标节点的对外端口 |
| Hops | Int 该路径的跳数 |
| Nextport | Int 下一跳经由节点 |
| path | []int 该路径经过的各节点的端口号 |

* **消息信封结构**

在发送数据过程中，路由表和数据包都被统一封装成消息信封结构发送，邻居节点在接收到消息信封时通过查看该消息中Type字段的描述获取其中封装的数据类型，并根据具体的数据类型处理数据。

|  |  |
| --- | --- |
| 数据表项 | 解释 |
| Type | String 数据类型（路由表或） |
| Msg | []byte 具体的数据 |

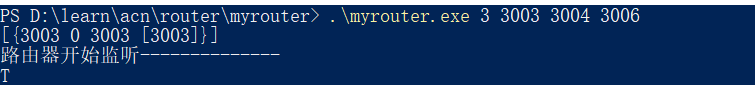
* **数据包**

|  |  |
| --- | --- |
| 数据表项 | 解释 |
| NextPort | Int 经由节点 |
| Path | []int 路由路径 |
| Data | []byte 数据包数据 |

## 7 处理流程

## 8 实验结果展示

* 我们按照实验要求中给出如图8.1所示的拓扑结构启动、连接路由器节点，当路由表同步稳定后，我们打印出3003节点的路由表如图8.2所示，路由表保存了3003节点到其余节点的全部路由路径。

图8.1 实验路由节点的拓扑结构

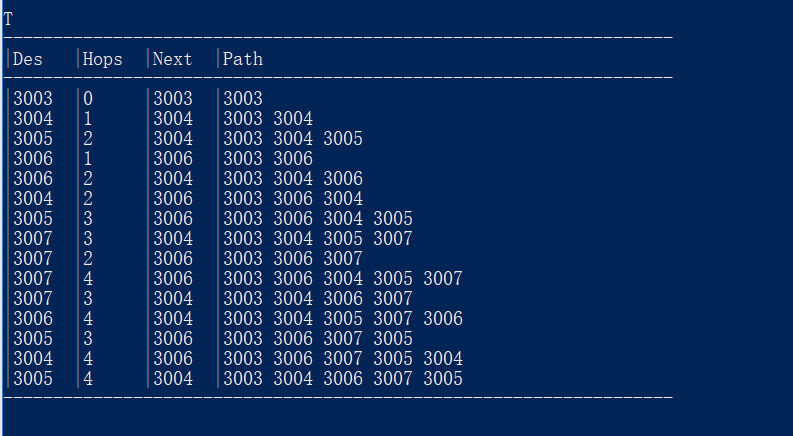
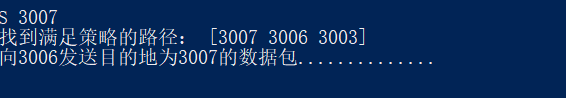


图 8.2 全部节点启动后3003节点路由表

* 我们首先在不对3003路由节点设置路由优先序列和拒绝节点等策略的情况下，向3007节点发送数据包，命令如图8.3所示，结果如图8.4所示。如图8.3所示，3003节点基于路由表找到一条

图 8.3 3003向3007发送数据包（无路由策略）

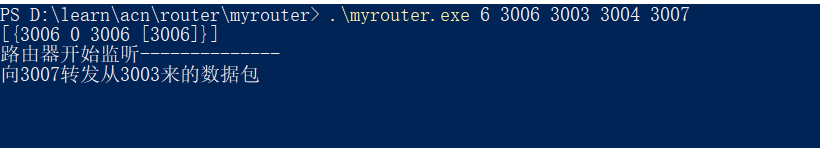
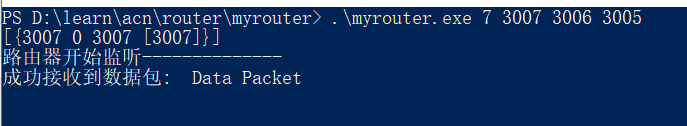
最短路径3003-3006-3007(截图中的路径需要反过来看)。图8.4描述了节点3006对数据包进行了转发，3007节点接收此数据包。

图 8.4 包的转发和接收

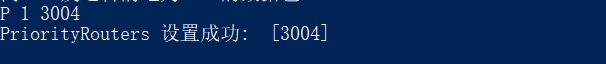
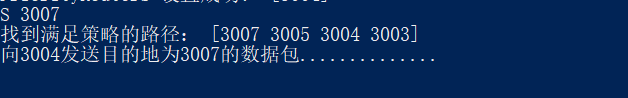
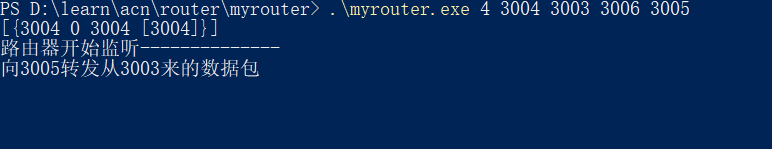
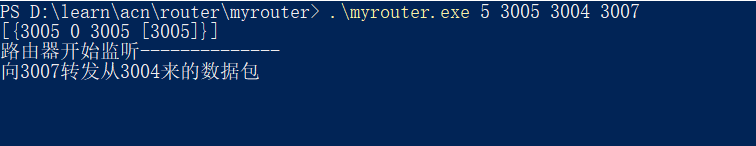
* 而后，我们基于策略对3003节点设置优先节点序列，使他发送的包必须经过节点3004 ，必经节点设置如图8.5所示。此时我们将

图 8.5 必经节点设置命令

 再次向3007节点发送消息，路径选择和包转发过程如图8.6所示。



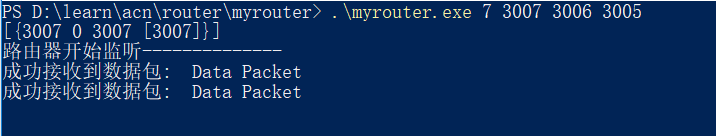
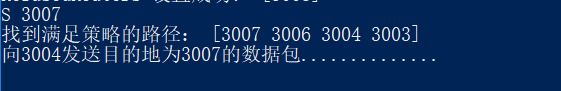


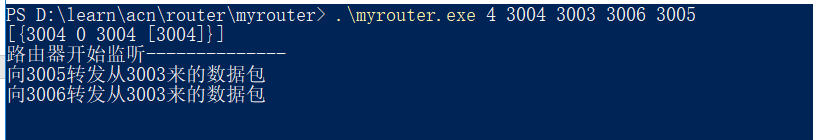
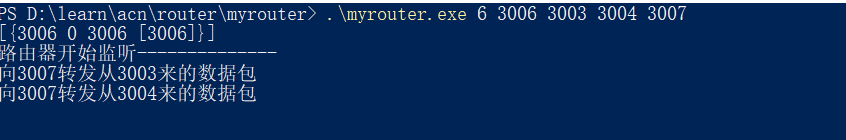
图 8.6 基于必经节点序列的数据包转发和接收过程

此时3003节点并未选择3003-3006-3007的路由路径，而是考虑到必经节点序列[3004]，选择了3003-3004-3005-3007这条路由路径。

* 再次设置拒绝节点，我们将3005节点设置为3003的拒绝节点，再次向3007发送数据包，设置命令如8.7所示。此时，3003节点同时考虑到优先路由节点序列和拒绝节点，选择路径3003-3004-3006-3007进行转发，过程如下所示。

图 8.7 设置拒绝节点





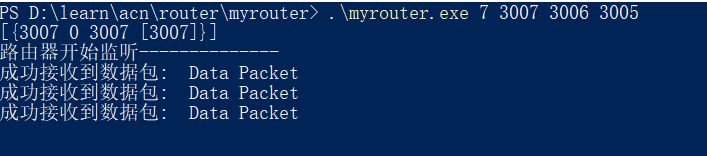


图 8.8 基于必经节点和拒绝节点选择路径进行包转发

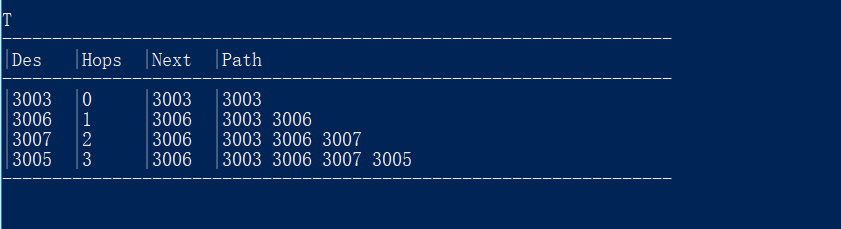
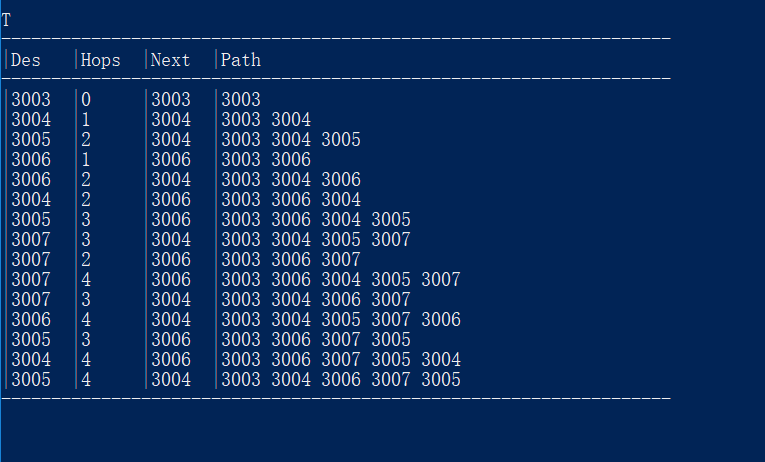
* 我们将3004节点关闭，等待一段时间后观察节点3003路由表的变化，可以发现和3004节点相关的路由信息都被置为无法到达，从本地路由表中删除，如图8.9所示。

图 8.9 关闭节点3004后3003的路由表

* 我们将3004节点再次开启，等待一段时间，发现节点3003的路由表恢复。

图 8.10 开启节点3004后节点3003的路由表的恢复