网络协议栈分析

ADOV协议分析

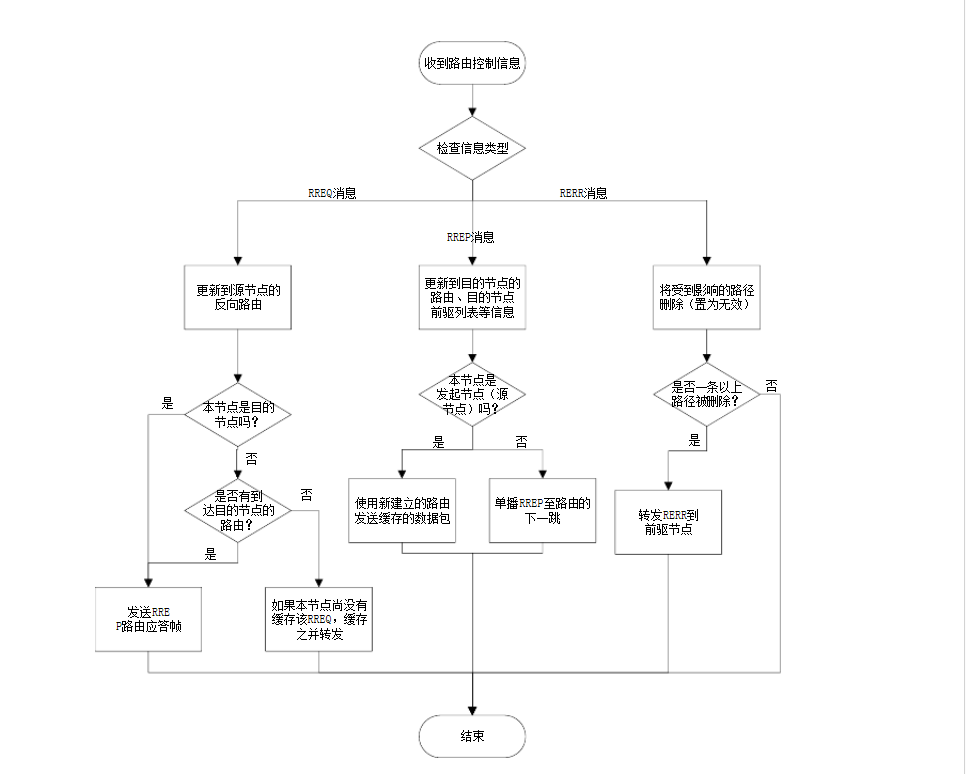
软网1703 张兆轩

软网1703 高树达

# 一、概述

AODV算法只在多个网络节点中建立和维护一个动态的，自启动的，多跳路由的专属网络。ADOV使得移动检点能快速获得通向新的目的节点的路由，并且节点仅需要维护通向它相邻一跳节点的信息。它在路由表中引入目的节点序列号（dest\_seqno），确保了任何时候都不会出现回环。同时，引入HELLO报文机制，当网络中出现连接断开或其他异动时，AODV能够让网络检点实时对这些变化做出响应，避免了传统路由算法出现的很多问题。

AODV节点的工作流程如下图所示：



## 协议报文以及路由过程

|  |  |  |
| --- | --- | --- |
| 报文名 | 解释 | 用途 |
| REEP | 路由请求 | 当某结点接收到新的目标节点时广播REEQ消息 |
| RREP | 路由回复 | 当一个节点向给它发送RREQ的节点单播一条RREQ时，原节点将相邻节点状态信息存入路由表 |
| REER | 路由错误 | 若活动路由表某一条连接发生断裂，REER将指出不能到达的节点或子网 |

### RREQ帧

当源节点S需要向目的节点D发送数据包时，但有没有目的节点D的路由入口时，会发送RREQ帧，RREQ帧会广播。

1. 对REEQ的处理

接收到RREQ 的结点做如下处理：  
（1）创建一个表项，先不分配有效序列号，用于记录反向路径。  
（2）如果在“路由发现定时”内已收到一个具有相同标识的RREQ报文，则抛弃该报文，不做任何处理;否则，对该表项进行更新如下：  
I.下一跳结点=广播RREQ的邻居。  
II.跳数=RREQ报文的“跳计数”字段值。  
III.设置表项的“过时计时器”。  
（3）如果满足以下条件，则结点产生“路由回答报文”RREP，并发送到信源；否则更新RREQ报文并广播更新后的RREQ报文。  
I.该结点是信宿。  
II.结点的路由表中有到信宿的活动表项，且表项的信宿序列号大于RREQ中的信宿序列号。  
（4）更新RREQ报文并广播更新后的RREQ报文  
I.信宿序列号=本结点收到的信宿相关的最大序列号。  
II.跳计数加1。

1.帧的格式

0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-

| Type |J|R|G|D|U| Reserved | Hop Count |

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-

| RREQ ID |

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-

| Destination IP Address |

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-

| Destination Sequence Number |

+-+-+-++-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+--+-+-+-+-+-+-+-+-+-+-+-+-+-+-++-+-+-+-+-+

| Originator IP Address |

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-

| Originator Sequence Number |

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

2.帧中字段的含义

type : 1 (默认1)

J : join

R : repair

G : Gratuitous RREP (免费路由回复标记) 说明是否应该向目标节点发送一个路由回复的报文

D : 仅允许目的节点进行回复

U : 指示目标节点序列号未知

Reserved : 填充0 会被接收端忽视的字段

Hop Count : 从发起节点到处理该节点请求的跳数

RREQ ID : ID

Destination IP Address : 目的节点IP地址

Destination Sequence Number : 目标节点序列号 发起节点在以前能通向目的节点的节点中找到的最新的序列号

Originator IP Address : 发起本条路由请求信息的IP地址

Originator Sequence Number : 发起者的路由表中现在正在使用的序列号

### 1.1.3 RREP帧

当RREQ到达时，目的节点发送的反向路由帧，使用这个帧来使网络上的各个节点建立前一个节点的路由，节点只对收到的第一次RREQ产生反应，重复发送的RREQ将不能产生重复的RREP帧。

接收到RREP的操作：

I.如果收到相应的RREQ的信宿序列号与信宿维护的当前序列号相等，则信宿将自己维护的序列号加1，否则不变。

II.跳计数=0。

III.定时器值。

（2）中间结点产生的RREP

执行如下操作：

I.本结点获取的该信宿的最大序列号。

II.跳计数=本结点到信宿的跳数（查相应表项即可得到）。

III.更新本结点维护的“前向路由表项”的下一跳和“反向路由表项”的前一跳

b. 对RREP的处理

结点对接收到的RREP 作如下处理。

（1）如果没有与RREP报文中的信宿相匹配的表项，则先创建一个“前向路表”空表项。

（2）否则，满足如下条件对已有表项进行更新。

1. 帧的格式

0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-

| Type |R|A| Reserved |Prefix Sz| Hop Count |

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-

| Destination IP address |

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-

| Destination Sequence Number |

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-

| Originator IP address |

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-

| Lifetime |

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-

1. 帧中字段的含义

type : 2 (默认2)

R : repair

A : acknowledgment required (需要确认)

Reserved : 填充0 会被接收端忽视的字段

Prefix SZ : 前缀长度 值为 0 - 31 前缀指的是 IPV4地址 - 主机地址的长度

Hop Count : 从发起节点到处理该节点请求的跳数

Destination IP Address : 目的节点IP地址

Destination Sequence Number : 目标节点序列号 从路由表中查找对应的序列号

Originator IP Address : 发起本条路由请求信息的IP地址

lifetime 生命时长 在规定的这段时间中 接受到这条消息的节点将认为这条路径是可用的

### 1.1.4 RRER帧

路由错误帧，用来进行路由的错误控制

执行操作：

邻居间周期性的互相广播“Hello”报文，用来保持联系，若在一段时间内没有收到“Hello”报文，则认定为链路断。例如当结点X、Y之间链路产生断路使数据无法通过此条链路传至信宿，则结点X会产生RRER报文向信源报告此情况。RRER通过广播形式传送，维护路由表的结点收到此报文会更新路由表（将X、Y间的路由设成无效），并转发RRER 报文。

1. 帧的格式

0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-

| Type |N| Reserved | DestCount |

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-

| Unreachable Destination IP Address (1) |

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-

| Unreachable Destination Sequence Number (1) |

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-

| Additional Unreachable Destination IP Addresses (if needed) |

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-

|Additional Unreachable Destination Sequence Numbers (if needed) |

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-

1. 帧中字段的含义

type : 3 (默认3)

N : No delete 当上一个节点对这个连接做了本地修复，就将这个标志位置真，也即不用删除这个路由

Reserved : 填充0 会被接收端忽视的字段

DestCount : 本消息内包含的不可达目的节点的数目

Unreachable Destination IP Address : 不能到的节点目的节点IP地址

Unreachable Destination Sequence Number : 与上面的目标节点对应的序列号

Additional : 可以包含多个不可达节点，用来表示多个不可达节点有一个 加一对儿消息

### 1.1.5 REEP ACK帧

收到需要确认的帧时用来回复的帧

1. 帧的格式

0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

| Type | Reserved |

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

1. 帧中字段的含义

type : 4(默认4)

Reserved : 填充0 会被接收端忽视的字段。

## 路由表

#### 路由表结构分析

struct rt\_table {

list\_t l;

struct in\_addr dest\_addr; /\* IP address of the destination \*/

u\_int32\_t dest\_seqno;

unsigned int ifindex; /\* Network interface index... \*/

struct in\_addr next\_hop; /\* IP address of the next hop to the dest \*/

u\_int8\_t hcnt; /\* Distance (in hops) to the destination \*/

u\_int16\_t flags; /\* Routing flags \*/

u\_int8\_t state; /\* The state of this entry \*/

struct timer rt\_timer; /\* The timer associated with this entry \*/

struct timer ack\_timer; /\* RREP\_ack timer for this destination \*/

struct timer hello\_timer;

struct timeval last\_hello\_time;

u\_int8\_t hello\_cnt;

hash\_value hash;

int nprec; /\* Number of precursors \*/

list\_t precursors; /\* List of neighbors using the route \*/

};

|  |  |  |
| --- | --- | --- |
| 变量名称 | 数据类型 | 作用 |
| dest\_addr(目的节点地址) | in\_addr | 用于标志使用此路由的最终目的节点，决定了数据分组转发方向。 |
| dest\_seqno(目的节点序列号) | u\_int32\_t | 反映此路由的新鲜度，一般序列号越大路由越新鲜， 这是保证开环的重要措施，在路由发现和路由应答更新路由时需要进行序列号的比较。 |
| flags(路由状态标志) | u\_int16\_t | 反映此路由目前的状态，主要用于告知数据分组经过此节点的时候处理方式。如果路 由处于无效状态，那么数据分组将丢失；如果处于修复状态，那么数据分组进入等待路由队列中；如果有效状态，那么直接转发。 |
| next\_hop(下一跳的ip地址) | in\_addr | 据分组经过本节点之后，数据分组将被直接转发的中继节点，通常下一跳节点应该出现在当前节点的邻节点列表中。 |
| hcnt(到达目的ip的跳数) | u\_int8\_t | 到达目的ip所需经历的子网的数量 |
| precursors(前驱节点表) | list\_t | 使用这条路由的所有直接前驱节点列表。在出现断链的时候可以通过前驱节点列表中是否存有节点而决定是否广播RERR消息 |
| rt\_timer(路由生命周期) | timer | 路由有效的生命期，在数据分组转发使用当前路由时会更新路由的有效生命期，当较长时间不使用此路由时，此路由的有效期将会过期，在路由管理时将会使路由失效 |

## 路由发现

当一个节点无法找到一个可用的到某个节点的路由时，它就会广播一条RREQ消息。中间节点各自更新到源节点的路由表，并维护指向发起节点的反向路由。目的节点和有效路由的中间节点产生RREP路由应答帧，该帧通过反向路由发送至源节点，源节点收到该帧后开始发送数据包。

## 1.2.3路由算法

# 、二、代码分析

该源代码共有46个子文件，其中主文件夹包含aodv\_hello.c、aodv\_neighbor.c等17个C文件，以及aodv\_rrer.h、aodv\_rreq.h等17个头文件，用于声明和定义aodv协议的工作流程。同时，在main函数中，声明了该协议栈的全局变量和全局函数。还包含了Makefile文件，用于编译源代码，README，用于简述代码使用方式和工作流程,。RFC（Request For Comments）3561，为该协议的说明文档。TAGS，用于编辑索引。子文件夹lnx中包含了AODV协议用于Linux的实现代码，前缀kaodv全称为kernel aodv即该实现应用于Linux内核。子文件夹patches包含了AODV的补丁文件，本次协议栈分析主要针对于主文件夹的C文件和H文件，子文件夹的内容不过多赘述。

## 2.1 文件介绍

|  |  |
| --- | --- |
| 文件名 | 简介 |
| list.h | 定义链表结构 |
| routing\_table.h | 路由表定义 |
| aodv\_hello.c | hello 消息的定义和相关操作 |
| aodv\_neighbor.c | 添加邻居节点并且处理邻居节点断开事件 |
| aodv\_rerr.c | rerr 消息的定义发送及处理 |
| aodv\_rrep.c | rrep 消息的定义发送及处理 |
| aodv\_rreq.c | rreq 消息的定义发送及处理 |
| aodv\_socket.c | 处理aodv的socket套接字 |
| aodv\_timeout.c | 超时处理 |

#### 全局变量

|  |  |  |
| --- | --- | --- |
| 变量名称 | 数据类型 | 作用 |
| rreq\_gratuitous | int |  |
| ratelimit | timer |  |
| rt\_log\_interval | int |  |
| receive\_n\_hellos | int | 记录收到的hello报文数目 |
| optimized\_hellos | int |  |
|  |  |  |
|  |  |  |