Ad Hoc 网络中按需路由协议 AODV 分析报告

刘健

（武汉理工大学 信息工程学院，武汉 430070）

这篇文章是我对 aodv 协议的一些粗略的分析加以总结，希望能给大家起到一个抛砖引玉的作用。如有不足，欢迎指正。我的博客：http://liujian.cublog.cn/

刘健's email: liujian.mail@163.com

刘健's Msn: daniel_liu123@hotmail.com

1 AODV 协议概述

AODV 协议是在 DSDV 协议基础上，综合类似 DSR 中的按需路由机制进行改进后提出的。不同之处在于 AODV 采用了逐跳转发报文方式，而 DSR 是源路由方式。因此，AODV 在每个中间结点隐式保存了路由请求和回答的结果，而 DSR 将结果显式保存在路由请求和路由回答报文中。此外，AODV 的另一个显著特点就是加入了组播路由协议扩展，并支持 QoS。但是，AODV 是基于双信道的假设工作，路由应答报文沿路由请求报文的反方向传至信源，因而不支持单向信道。

2 AODV 路由发现

AODV 有三种基本的协议报文类型：RREQ 报文、RREP 报文和 RRER 报文。

2.1 RREQ 报文

a. RREQ 报文格式

结点在需要（没有到信宿的活动路由）时，向其邻居广播 RREQ 报文用于路由发现。RREQ 报文格式如图 1 所示。

图 1 RREQ 格式

0	1	2	3	4	5	6	7	8
Packet Type		Reserved		Hop Count	Broadcast ID			
Destination IP					Destination Sequence Number			
Source IP					Source Sequence Number			

Packet Type － 报文类型，RREQ 的值为 1。

Reserved － 保留位，为以后扩展升级预留。

Hop Count － 跳计数，RREQ 的跳计数初值为 0。

Broadcast ID － 广播 ID，惟一标识了一个 RREQ 报文。

Destination IP － 信宿 IP 地址。

Destination Sequence Number － 信宿序列号表示信源可接受的“到信源的前进路由”新旧程度，等于过去接收到的信宿的最大序列号。可见，结点需要为每一个信宿维护一个信宿序列号。

Source IP — 信源 IP 地址。
Source Sequence Number — 信源序列号由信源结点维护，用于表示“到信源反向路由”的新旧程度。

b. 对 RREQ 的处理

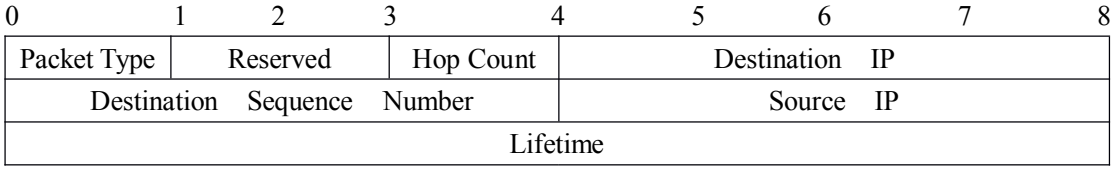
接收到 RREQ 的结点做如下处理：
（1）创建一个表项，先不分配有效序列号，用于记录反向路径。
（2）如果在“路由发现定时”内已收到一个具有相同标识的 RREQ 报文，则抛弃该报文，不做任何处理；否则，对该表项进行更新如下：
I. 信宿序列号=RREQ 报文的信源序列号。
II. 下一跳结点=广播 RREQ 的邻居。
III. 跳数=RREQ 报文的“跳计数”字段值。
IV. 设置表项的“过时计时器”。
（3）如果满足以下条件，则结点产生“路由回答报文”RREP，并发送到信源；否则更新 RREQ 报文并广播更新后的 RREQ 报文。
I. 该结点是信宿。
II. 结点的路由表中有到信宿的活动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

2.2 RREP 报文

a. RREP 报文格式

产生 RREP 的条件如上所述，图 2 为 RREP 报文的格式。

图 2 RREP 格式



Packet Type — 报文类型,RREP 的值为 2。
Reserved — 保留位，为以后扩展升级预留。
Hop Count — 跳计数，RREQ 的跳计数初值为 0。
Destination IP — 目的 IP 地址。
Destination Sequence Number — 目的序列号。
Source IP — 源 IP 地址。
Lifetime — 以毫秒为单位，自收到 RREP 开始计时以保证线路正确。

RREP 各字段的设置如下：
（1）信宿结点产生 RREP。
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. 信宿序列号=信宿维护的序列号。
III. 跳计数=0。
IV. 定时器值。

（2）中间结点产生的 RREP。

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。

I.现有表项的信宿序列号小于 RREP 报文中的序列号。

II.现有的表项没有激活。

III.信宿序列号相同，但 RREP 报文的“跳计数”值小于表项相对应的值；通过更新或创建，产生一个新的前向路由。

IV.下一跳=广播 RREP 的邻居结点。

V.信宿序列号=RREP 中的信宿序列号。

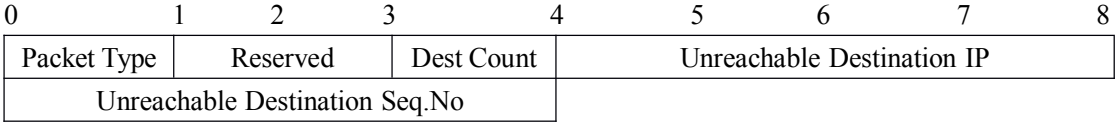
VI.跳计数加 1。

（3）按照上述的过程，任何转发 RREP 的结点，都记录了到信宿的下一跳，当 RREP 到达信源时。结点地址匹配，不再转发 RREP，信源到信宿的前向路由已经建立起来了。信源可以沿这条前向路径进行数据传输。

2.3 RRER 报文

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。RRER 格式如图 3 所示。

图 3　RRER 格式



Packet Type ― 报文类型,RRER 的值为 3。

Reserved ― 保留位，为以后扩展升级预留。

Dest Count ― 报文中包含不可达目的地址的数目，最少设置为 1.

Unreachable Destination IP ― 由于链路断，导致目的不可达的 IP 地址。

Unreachable Destination Seq.No ― 目的序列号，路由表条目到目的地无法到达目的 IP 地址领域。

3 AODV 路由维护

（1）与活动路由无关的结点移动，并不影响信源到信宿的寻径。

（2）如果信源结点移动导致路由不可用，则由信源重新发起路由发现过程。

（3）当信宿结点或活动路由的中间结点移动，导致链路中断，则链路的“上游结点”主动发送一个 RREP，该 RREP 的信宿序列号大于其所获得的信宿序列号，跳计数设为∞，并传播到所有活动邻居。该过程重复，直至所有相关信源结点被通告到。信源结点如果需要，

可重发起路由发现过程。

4 AODV 的“路由发现”举例

如图 4 所示，信源 N1 和信宿 N7 采用 AODV 的路由发现过程。如图所示，N1 通过广播 RREQ 报文进行路由请求，RREQ 最终会被中间路由转发到信宿 N7。假设 N1-N3-N6-N7 此条链路的 RREQ 报文最先传到信宿，则 N7 会丢弃其他链路传来的 RREQ 报文，并自 N7-N6-N3-N1 路径返回 RREP 报文，并对中间路由表项进行更新，以此建立信源到信宿的数据传输路径，即 N1-N3-N6-N7。

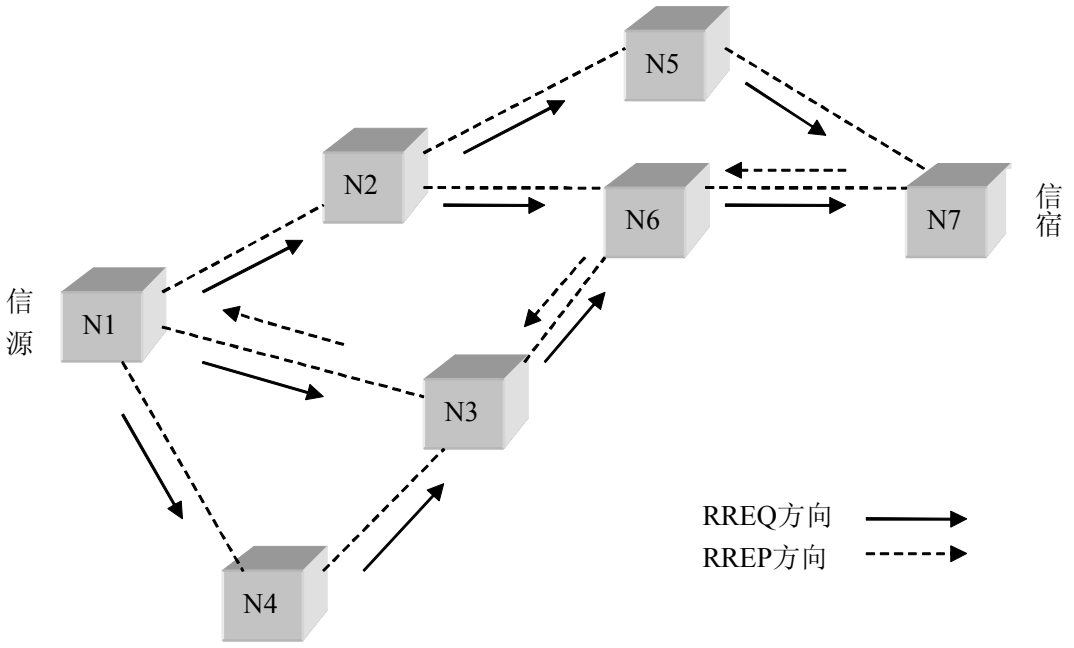


图 4　AODV 路由发现

由于 Ad Hoc 网络中路由结点具有移动性，其热点范围的改变很容易导致网络拓扑发生改变。例如图 4 中由于 N7 的移动导致 N6 与 N7 中的链路发生阻断，如图 5 所示。

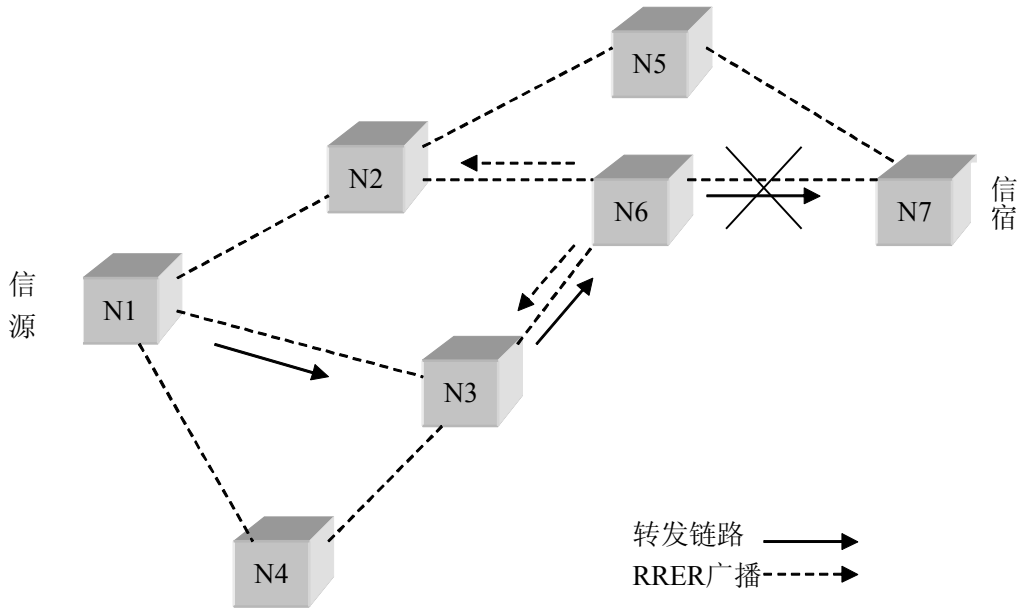


图 5 链路断广播 RRER

上图中 N6 与 N7 之间发生断路，此时 N6 会广播 RRER 报文，维护路由表的结点收到此报文会更新路由表（将 N6、N7 间的路由设成无效），并转发 RRER 报文，图中 N6 将 RRER 报文广播至 N2、N3。

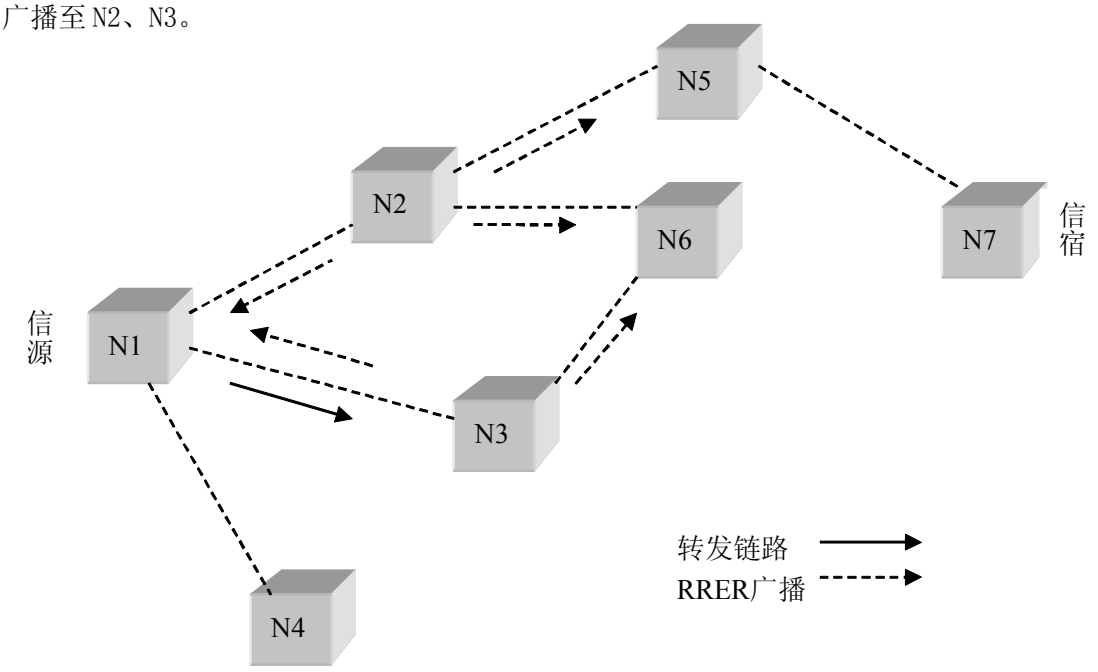


图 6 RRER 通过广播通报所有广播域中成员

图 6 所示的 RRER 报文已由 N6 传至 N2、N3，此时 N3 收到 RRER 后收到 N6、N7 链路阻断的消息并停止将信源的数据包转发至 N6。RRER 报文由 N2、N3 继续广播发出。

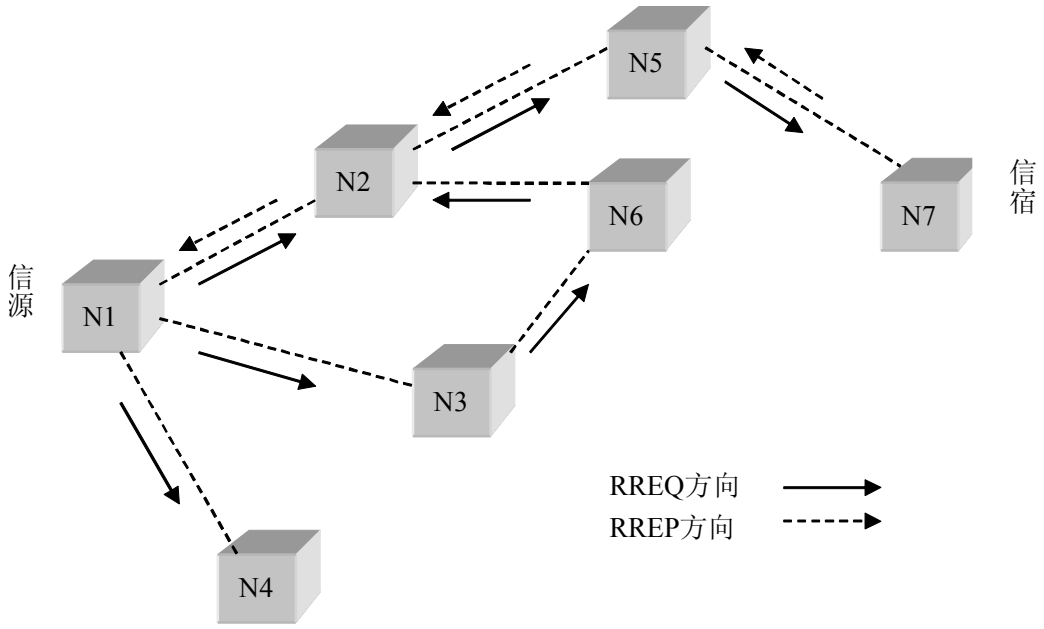


图 7 信源重新发布路由请求

当 RRER 传至信源后，信源得知原传输路径由于 N6、N7 间的链路阻断。信源重新发布路由请求，广播 RREQ，并最终得到新的传输路径 N1-N2-N5-N7。

5 AODV 源代码分析

当协议接收到一个分组，即 recv(Packet*, Handler*)函数被调用，函数根据分组类型调用不同的处理函数进行处理。

5.1 Void AODV::recv(Packet *p, Handler*)

/*判断是否是 aodv 包，是则调用 recvAODV(Packet*)函数进行处理。recvAODV 函数再根据分组的不同类型来调用不同的函数进行处理。*/

```
if(ch->ptype() == PT_AODV) { recvAODV(p);

//本结点产生的数据包，添加 IP 头

if((ih->saddr() == index) && (ch->num_forwards() == 0)) Add the IP Header

//收到本结点发送的包，说明有路由环路，丢包

else if(ih->saddr() == index) drop(p, DROP_RTR_ROUTE_LOOP);

//本结点是中间结点

else {if(--ih->ttl_ == 0) drop(p, DROP_RTR_TTL);//TTL 是分组最多能转发的次数

//收到的不是广播分组，解析分组
```

```
if ( (u_int32_t)ih->daddr() != IP_BROADCAST)    rt_resolve(p);

else   forward((aodv_rt_entry*) 0, p, NO_DELAY); //转发
```

5.2 `void AODV::recvAODV(Packet *p) {`

```
case AODVTYPE_RREQ:  recvRequest(p); break;

case AODVTYPE_RREP:   recvReply(p); break;

case AODVTYPE_RERR:   recvError(p); break;

case AODVTYPE_HELLO:    recvHello(p); break;

default: fprintf(stderr, "Invalid AODV type (%x)\n", ah->ah_type);
```

`exit(1);}` //根据包类型调用不同函数

5.21 如果接收到的是路由请求分组，则调用 recvRequest(Packet*)函数进行处理。如果该分组由结点自身产生或已经接收过的，会被结点丢弃，并结束处理。否则，结点将缓存该分组的序列号，并将该分组发送来的路径添加到反向路由中，转发相应分组。然后，结点根据该分组的目的地址进行判断并调用不同函数进行处理。

如果结点自身即为目的结点，则调用 sendReply(nsaddr_t, u_int32_t, nsaddr_t, u_int32_t, u_int32_t, double)函数进行响应。如果结点不是目的结点，但知道通往目的结点的路由，则调用 sendReply 函数进行响应，并在源和目的前驱列表中分别插入到源和目的的下一跳结点。否则，不能直接响应该请求，将跳数加1，并调用 forward(aodv_rt_entry*, Packet*, double)函数转发该分组。

在 sendReply 函数中，结点首先查找到达目的结点（即发送路由请求分组的结点）的路由，创建并填充分组，然后调用 Scheduler::instance().schedule()函数来发送该分组。

5.22 如果接收到的是路由响应分组，则调用 recvReply(Packet*)函数进行处理。结点首先查询前往分组目的结点的路由，如果不存在则新增一条路由项。然后，结点更新到该目的结点的路由项，并发送所有相关分组。

如果结点为目的结点则更新路由发现延迟并发送所有相关的分组。如果结点不是目的结点，但知道通往目的结点的路由，则将跳数加1，调用 forward 函数转发该分组，并修改响应的前驱列表。如果结点不是目的结点，也不知道通往目的结点的路由，则丢弃该分组。

5.23 如果接收到的是路由错误分组，则调用 recvError(Packet*)函数进行处理。

结点首先清除所有受到影响的路由项，丢弃所有受影响的分组。然后，如果前驱结点中存在会受该路由错误影响的分组，则调用 sendError(Packet*, bool)函数转发该分组。

sendError 函数创建并填充分组，然后调用 Scheduler::instance().schedule()函数来发送该分组。

5.24 如果接收到的是 Hello 消息分组，则调用 recvHello(Packet*)函数进行处理。结点会将该邻居的信息添加到邻居列表中（或更新该邻居的信息）。

5.3 如果是数据分组，则节点丢弃已经发送过或者 ttl 为0的分组，并结束处理。如果分组是由上层协议产生的，则节点添加 IP 报头。随后，节点根据目的路由进行不同处理。

5.31 如果目的节点路由未知，则调用 rt_resolve(Packet*)函数进行路由解析和转发。

如果目的节点路由在路由表中存在，则直接调用 forward 函数进行转发。如果分组是由节点自身产生的，则将分组保存到缓冲队列中，并调用 sendRequest(nsaddr_t)函数查询目的路由。如果目的路由已知，但正在进行本地修复，则将分组保存到缓冲队列中。否则，丢弃该分组，并调用 sendError 函数报错。

```
void AODV::rt_resolve(Packet *p) {//

rt = rtable.rt_lookup(ih->daddr()); //查找是否有到目的结点的路由

if(rt == 0)       rt = rtable.rt_add(ih->daddr());

//没有，则添加到该目的结点的路由，此时添加的路由是无效的

if(rt->rt_flags == RTF_UP)   forward(rt, p, NO_DELAY);

//有效路由，则根据路由表中信息转发分组

else if(ih->saddr() == index) rqueue.enque(p);   sendRequest(rt->rt_dst);

//如果本结点是该分组的源结点，说明没有到目的结点的路，此时发送 RREQ 找路

 else if (rt->rt_flags == RTF_IN_REPAIR) rqueue.enque(p);//链路中断，在维护中

else   sendError(rerr, false);

// 本结点转发数据分组，但是不知道该到目的结点的路，发送 RERR 说明链路中断
```

5.32 如果接收到的是数据分组，且自身为目的节点，则通过调用 PortClassifier 对象的 recv(Packet*, Handle*)函数将分组交递给高层协议，并结束处理。否则，节点设置分组属性，并调用 Scheduler::instance().schedule(Handler*, Event*, double)函数来发送分组。 其中，Handler 为基类中的属性 target_（会根据脚本中的设置指向相应的协议实体），Event 为要发送的分组即可。

6. 几个重要函数部分原代码分析

a. 发送 RREQ AODV::sendRequest(nsaddr_t dst) {

```
aodv_rt_entry *rt = rtable.rt_lookup(dst);

//添加到目的节点的路由，此时的路由不可用，是无效的

if (rt->rt_req_timeout > CURRENT_TIME)   Packet::free((Packet *)p);
```

//不到发送 RREQ 的时间，若没收到 RREP，源节点需要定时发送 RREQ

```
if (rt->rt_req_cnt > RREQ_RETRIES) rt->rt_req_timeout = CURRENT_TIME +
MAX_RREQ_TIMEOUT;      drop(buf_pkt, DROP_RTR_NO_ROUTE);
```

//AODV 中多次发送 RREQ，多次发送 RREQ 后仍找不到路，则丢包

//余下部分是填充路由表以及 RREQ 分组的内容

b. 收到 RREQ `void AODV::recvRequest(Packet *p) {`

```
if(rq->rq_src == index)   Packet::free(p);
```

// 信源收到路由请求，则丢弃该 RREQ

```
 if (id_lookup(rq->rq_src, rq->rq_bcast_id))   Packet::free(p);
```

// * Cache the broadcast ID ，用于判断是否已收到过该 RREQ

```
   id_insert(rq->rq_src, rq->rq_bcast_id);
```

//查找是否有到源节点的路由，有则更新，无则添加 a

//缓存中是否有到源节点的数据分组，有，则建立好路由后开始发送数据

//查找是否有到目的节点的有效路由，有则向源节点回复 RREP，没有则继续转发该 RREQ

c. 发送 RREP `void AODV::sendReply(nsaddr_t ipdst, u_int32_t hop_count,`
`nsaddr_t rpdst,u_int32_t rpseq, u_int32_t lifetime, double timestamp) {`

```
aodv_rt_entry *rt = rtable.rt_lookup(ipdst); //查找到上游结点的路由条目
```

//余下部分是给 RREP 中参数赋值

d. 收到 RREP `void AODV::recvReply(Packet *p) {`

```
rt = rtable.rt_lookup(rp->rp_dst); //从路由表中搜索是否有 RREP 包含的路由条目

 if(rt == 0) rt = rtable.rt_add(rp->rp_dst); //如果没有加入路由表中
```