题目：无线自组网按需距离向量路由协议代码分析

小组成员：杨韬，叶上维，徐文博

分工：

目录

[1. 基本原理 4](#_Toc533793835)

[1.1 概述 4](#_Toc533793836)

[1.1.1 定义 4](#_Toc533793837)

[1.1.2特点 5](#_Toc533793838)

[1.2 AODV工作过程 6](#_Toc533793839)

[1.2.1 路由发现过程 6](#_Toc533793840)

[1.2.2 路由维护和错误控制 7](#_Toc533793841)

[1.2.3 拥塞控制 7](#_Toc533793842)

[2. 代码介绍 7](#_Toc533793843)

[2.1 全局变量 8](#_Toc533793844)

[2.2 统计量 9](#_Toc533793845)

[2.3 SNMP变量 9](#_Toc533793846)

[3. 以太网接口 9](#_Toc533793847)

[3.1 main函数 9](#_Toc533793848)

[3.2 main函数文件内其他函数以及变量的定义： 12](#_Toc533793849)

[3.3 AODV\_socket.c 13](#_Toc533793850)

[3.3.1 \_\_cmsg\_nxthdr\_fix函数 13](#_Toc533793851)

[3.3.2 aodv\_socket\_init函数-套接字初始化函数 13](#_Toc533793852)

[3.3.3 aodv\_socket\_read函数-套接字读取 16](#_Toc533793853)

[3.3.4 aodv\_socket\_process\_packet函数-处理数据包函数 18](#_Toc533793854)

[3.3.5 recvAODVUUPacket函数-接收AODVUUPacket 19](#_Toc533793855)

[3.3.6 aodv\_socket\_send函数-套接字发送函数 20](#_Toc533793856)

[3.3.7 aodv\_socket\_new\_message函数 22](#_Toc533793857)

[3.3.8 aodv\_socket\_queue\_msg函数 23](#_Toc533793858)

[3.3.9 aodv\_socket\_cleanup函数 23](#_Toc533793859)

[3.4 AODV\_rreq.c 23](#_Toc533793860)

[3.4.1 rreq\_send函数-发送rreq消息 23](#_Toc533793861)

[3.4.2 rreq\_forward函数-转发rreq消息 23](#_Toc533793862)

[3.4.3 rreq\_process函数-处理rreq消息 24](#_Toc533793863)

[3.4.4 rreq\_route\_discovery函数-路由发现过程 28](#_Toc533793864)

[3.5 AODV\_rrep.c 30](#_Toc533793865)

[3.5.1 rrep\_ack\_process函数 30](#_Toc533793866)

[3.5.2 rrep\_send函数-发送rrep消息 31](#_Toc533793867)

[3.5.3 rrep\_forward函数-转发rrep消息 31](#_Toc533793868)

[3.5.4 rrep\_process函数-处理rrep消息 32](#_Toc533793869)

[3.5.5 其他函数 34](#_Toc533793870)

[3.6 AODV\_rerr.c 34](#_Toc533793871)

[3.6.1 rerr\_add\_udest函数-添加不可达目的节点 34](#_Toc533793872)

[3.6.2 rerr\_process函数-处理rerr消息 34](#_Toc533793873)

[3.7 Time\_queue.c 36](#_Toc533793874)

[3.7.1 NS\_CLASS timer\_init函数-记时器初始化函数 36](#_Toc533793875)

[3.7.2 NS\_CLASS timer\_timeout函数-计时器超时，调用此函数进行清理工作 37](#_Toc533793876)

[3.7.3 NS\_CLASS timer\_add函数-在计时器队列中添加计时器函数 38](#_Toc533793877)

[3.7.4 NS\_CLASS timer\_remove函数-计时器队列中移除计时器 39](#_Toc533793878)

[3.7.5 timer\_left函数-计算计时器剩余时间 39](#_Toc533793879)

[3.7.6 timer\_age\_queue函数-计算计时器队列的生存时间 40](#_Toc533793880)

[3.8 Aodv\_hello.c 40](#_Toc533793881)

[3.8.1 NC\_CLASS hello\_start函数-初始化hello消息 40](#_Toc533793882)

[3.8.2 NC\_CLASS hello\_stop函数-停止发送hello消息 41](#_Toc533793883)

[3.8.3 Hello\_process函数-处理hello消息 43](#_Toc533793884)

[3.9 Nl.c 46](#_Toc533793885)

[3.9.1 Nl\_init函数-初始化函数 46](#_Toc533793886)

[3.9.2 Nl\_kaodv\_callback函数-根据内核套接字中收到的不同命令，执行不同的回调函数 48](#_Toc533793887)

[3.9.3 Nl\_rt\_callback函数-根据路由套接字中收到的不同命令，执行不同的回调函数 51](#_Toc533793888)

[3.9.4 Nl\_send函数-发送数据包给内核模块 53](#_Toc533793889)

[3.9.5 Nl\_kern\_route函数-操作内核路由表，包括增加、修改、删除路由表条目等操作 54](#_Toc533793890)

[3.9.6 Nl\_kern\_route函数-向内核发送一个添加一个路由信息的消息，包括目的地地址、下一跳，生存期，路由状态标志，网络接口等 55](#_Toc533793891)

[3.10 Aodv\_neighbor.c 56](#_Toc533793892)

[**3.10.1更新活跃的邻居节点（更新来自非HELLO AODV控制消息的邻居）（void NS\_CLASS neighbor\_add）** 56](#_Toc533793893)

[3.10.2邻居节点链路中断情况（void NS\_CLASS neighbor\_link\_break） 56](#_Toc533793894)

[3.11 Aodv\_timeout.c 57](#_Toc533793895)

[3.11.1定义定时器到期时调用的超期函数 58](#_Toc533793896)

[3.11.2.定义本地修复超时处理函数 58](#_Toc533793897)

[3.11.3.设置路由过期超时函数 59](#_Toc533793898)

[3.11.4.路由删除函数设置 59](#_Toc533793899)

[3.11.5.HELLO消息超时函数 59](#_Toc533793900)

[3.11.6响应超时函数 59](#_Toc533793901)

# 基本原理

## 概述

AODV协议(Ad hoc on-demand distance vector routing)是应用于无线自组织网络中进行路由选择的路由协议,它能够实现单播和多播路由。该协议是自组织网络中按需生成路由方式的典型协议。用于特定网络中的可移动节点。它能在动态变化的点对点网络中确定一条到目的地的路由，并且具有接入速度快，计算量小，内存占用低，网络负荷轻等特点。它采用目的序列号来确保在任何时候都不会出现回环（在路由控制信息出现异常的时候也是如此），避免了传统的距离向量协议中会出现的很多问题。

### 1.1.1 定义

AVOD算法旨在多个移动节点中建立和维护一个动态的、自启动的、多跳路由的网络。它是的移动节点可以快速获得新目的节点的路由，而每一个节点仅需维护信号范围内节点的路由，更远的节点路由信息不必维护。

AODV定义了三种消息种类，分别是：路由请求（RREQ）、路由回复（RREP）、路由错误（RERR）。这些消息通过UDP和通常的IP协议来接收。消息不会被盲目不转发，但AODV允许特殊操作的得到广泛分布（e.gRREQ），甚至散布到整个无线自组织网络。这些RREQ的散布范围由IP头文件中的TTL来确定。

在广播RREQ消息之前，发起节点将消息的缓存一段时间。这个时间由“PATH\_DISCOVERY\_TIME”来决定。当节点从邻居节点收到具有同样RREQID和OriginatorIPaddress的RREQ消息时，他江晖认为这是一个发回来的包而将其丢弃。当某节点连接一个新节点时，广播RREQ尝试找到到目的节点的路由。在两种情况下，路由可以被找到：首先，RREQ消息到达目的节点，路由被找到。其次，如果RREQ消息到了一个中间节点，这个中间节点拥有到目的节点足够新（freshness）的路由（freshness的路由指的是有一条到目的节点的正确路由，且拥有一个较大的序列号，该序列号不得小于RREQ的序列号）。当朝发起RREQ的节点单播一个RREP消息后，该路由建立。所有接收到请求的结点都会缓存一条回到发起节点的路由，因此RREP可以通过单播从目的节点返回到发起节点，或则从一个能找到目的节点的中间节点返回到发起节点。

当活动路由表有一条连接断开时，一条RERR消息将被用来通知其他节点发生连接断裂。RERR消息指出不能到达的目的节点（或者目的子网）。为了实现这种报告机制，接地啊你将会维护一张“先驱表”，其中的信息可以轻易从回传的RREP过程中获得。先驱表中包含邻居节点IP，这些邻居节点会将它用作到达目的地的下一跳节点。先去表

AODV维护了当一个节点需要给网络中的其他节点传送信息时，如果没有到达目标节点的路由，则必须先以多播的形式发出RREQ(路由请求)报文。RREQ报文中记录着发起节点和目标节点的网络层地址，邻近节点收到RREQ，首先判断目标节点是否为自己。如果是，则向发起节点发送RREP(路由回应)；如果不是，则首先在路由表中查找是否有到达目标节点的路由，如果有，则向源节点单播RREP，否则继续转发RREQ进行查找。

在网络资源充分的情况下，AODV协议可以通过定期广播hello报文来维护路由，一旦发现某一个链路断开，节点就发送ERROR报文通知那些因链路断开而不可达的节点删除相应的记录或者对已存在的路由进行修复。

### 1.1.2特点

1. AODV协议采用的基本路由算法为距离向量算法，但有所改进。
2. AODV协议是按需路由协议。也就是说，只有在主机需要的时候才会激发出路由寻找过程。
3. AODV采用UDP封装，属于应用层协议。每次寻找路由时都要触发应用层协议，增加了实现的复杂度。
4. IEEE 802．15．4规定的帧大小为127字节，MAC头部及尾部校验最多使用了25字节，IPv6头部与UDP头部占用 48字节，只剩下54字节的空余。
5. LR—WPAN中，拓扑结构相对简单，网络的规模相对较小，节点的位置不固定，对它的设计首先要考虑的因素是简单、节能等问题。AODV的路由框架和报文设计并没有考虑这些问题。

AODV是一个路由协议，它的工作是管理路由表。AODV的每一个路由表将包含以下内容：

每个AODV协议的结点都要维护一张路由表，表的各字段如下：

1. 目的节点IP地址
2. 目的节点序列号（Sequence Number）
3. 目的节点序列号是否有效标志位
4. 下一跳节点的IP地址
5. 本节点到达目的节点需要的跳数
6. 前驱节点列表（precursor list）
7. 生存时间（路由失效或应当删除的时间）
8. 网络层接口
9. 其他的状态和路由标志位（e.g有效、无效、可修复、正在修复）

路由表的特征如下：

1. 路由表每项只记录下一跳路由信息，而不是整条路由信息，简化了路由表的建立和维护 。
2. 节点将相邻节点的连接状态保存到活动路由表中。
3. AODV在每一个路由表上使用了目的序列号。 源节点和目的节点都维护各自的序列号 。
4. 当一条链路断裂或失效导致一个节点不可达，路由表将通过对序列号的操作和标注路由表表项错误来使此条路由失效。
5. 如何管理序列号是提高路由建立和维护的关键 。

I．  序列号是用来标识路由信息新旧程度（freshness）的 。

II． 源节点发起路由请求RREQ，或者目的节点返回路由应答RREP，都要更新各自 的序列号 。

III．中间节点依据序列号的大小判断路由的新旧 。

## AODV工作过程

aodv协议包括路由发现，路由维护，路由错误控制，拥塞控制4种工作过程。它通过路由发现添加新的路由表项，通过路由维护保证出入口的通畅，路由错误控制来剔除失效的路由项，拥塞控制降低网络负载。

### 路由发现过程

AODV协议是按需平面距离向量路由协议，所以只有当上层有需要发数据且路由表项不存在的时候，才触发aodv的路由发现功能。aodv通过rreq帧和rrep帧实现本功能。

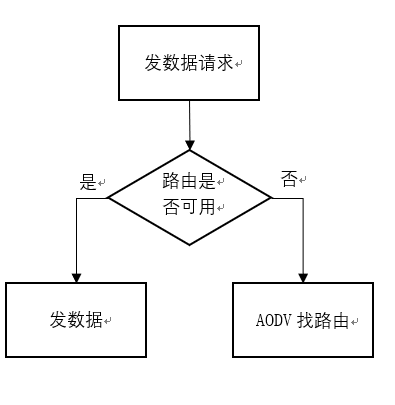


图 1 发现功能触发

首先，作为起始节点向邻居节点发送RREQ帧。邻居节点收到后，也向它的邻居节点转发RREQ帧。最后目标节点也会收到RREQ帧。

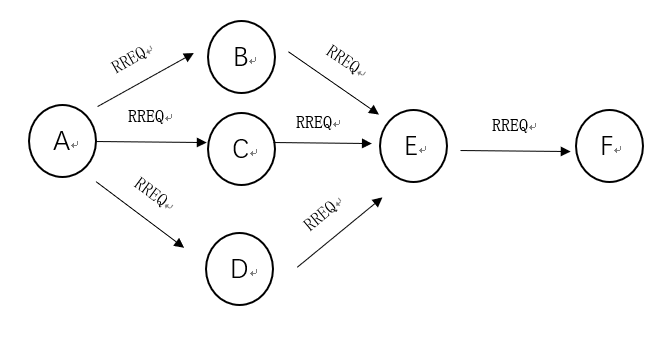


图 2 RREQ帧转发过程

在RREQ帧传播的过程中，各个中间节点在收到帧后更新到源端的路由。对于收到多个源端到一个目的端的RREQ帧的情况，取最早到达的更新到源端的路由表项。收到同一个源端到多个目的端的RREQ帧，取最新的更新到源端的路由表项。

最终目的节点收到RREQ帧后，也更新路由表项，同时查询路由表，向源端回复一个RREP帧。中间节点收到后更新路由表，查询路由表，向源端单播RREP帧。

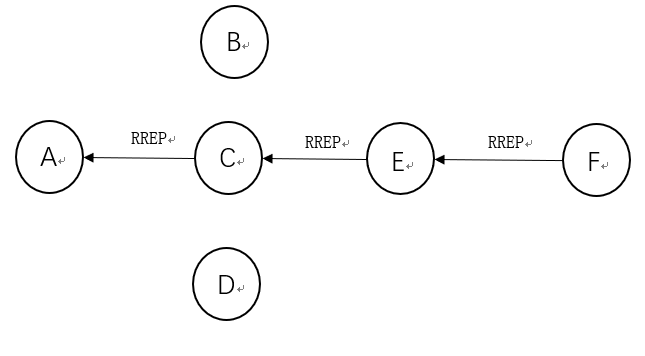


图 3 单播RREP过程

最后源端收到RREP回复帧后，更新路由表项，数据包得以发送。

还有一种情况是，当RREQ帧中相关字段允许的话，中间节点收到RREQ后，查找自己的路由表，如果发现有到达目的地的路由表项，则回复一个RREP帧。

### 路由维护和错误控制

AODV通过发送HELLO帧来实现路由的维护功能，保证了节点各个出口的有效性。

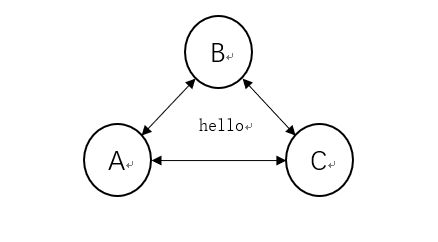


图 4 节点向邻居发hello帧

HELLO帧本质上是TTL为1的RREP帧。当节点收到RREP帧后更新对应的计时器，重

新计时。节点发现自己路由项中某个计时器超时，将该链路设置为无效，同时向相关路由表项中的前期节点发送RERR错误控制帧。

各节点收到RERR后根据帧内容修改路由表，同时把RERR转发。

### 拥塞控制

如果在规定时间内，发送RREQ请求帧的节点没有收到对应的RREP帧，那么节点可以再次发送RREQ请求帧。再次发送RREQ帧时，节点对于RREP帧的等待时间将会翻倍，这使得网络的负担得以减少。在尝试了设定次数后，节点就在路由表中标志该目的地不可达，然后构造消息发送给应用层。

# 代码介绍

该协议实现代码的根目录下共包含17个c文件和17个头文件里；一个记录更改日志的Changelog文件；一个GNU通用公共授权文件GPL；一个用于编译的Makefile文件；一个用于简要介绍AODV和使用方法的README文件；一个lnx文件夹，里面放了AODV协议用于linux 内核实现的代码，共6个c文件和7个头文件；一个patch文件夹，其中有8个补丁文件。

|  |  |
| --- | --- |
| 文件 | 说明 |
| Params.h | 各种参数的定义 |
| routing\_table.h | 路由表定义 |
| routing\_table.c | 路由表操作 |
| defs.h | 宏定义 |
| list.h | 链表结构定义 |
| list.c | 链表操作 |
| timer\_queue.h | 定时器队列定义 |
| timer\_queue.c | 定时器操作 |
| debug.h | 日志操作、转换成string、打印路由表 |
| debug.c |
| Endian.c | 补充某些缺乏〈endian.h〉的系统 |
| Aodv\_hello.h | hello消息的定义 |
| Aodv\_hello.c | hello消息的发送、处理等操作 |
| Aodv\_rerr.h | RERR消息的定义 |
| Aodv\_rerr.c | RERR消息的操作 |
| Aodv\_rrep.h | RREP消息的定义 |
| Aodv\_rrep.c | RREP消息的操作 |
| Aodv\_rreq.h | RREQ消息的定义 |
| Aodv\_rreq.c | RREQ消息的操作 |
| Aodv\_socket.h | AODV套接字定义 |
| Aodv\_socket.c | AODV套接字管理 |
| Aodv\_timeout.h | AODV超时处理 |
| Aodv\_timeout.c |
| Aodv\_neighbor.h | 添加和断开邻居 |
| Aodv\_neighbor.c |

## 全局变量

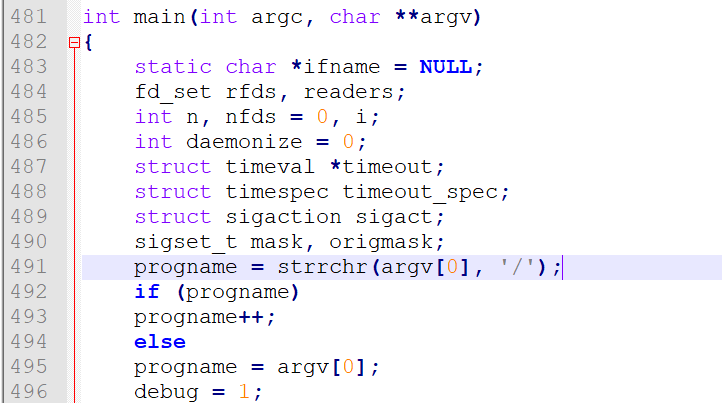
## 统计量

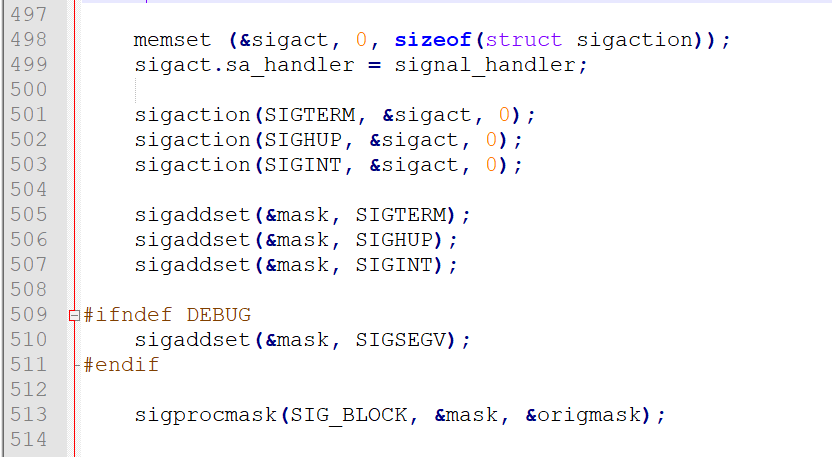
## SNMP变量

# 以太网接口

## main函数

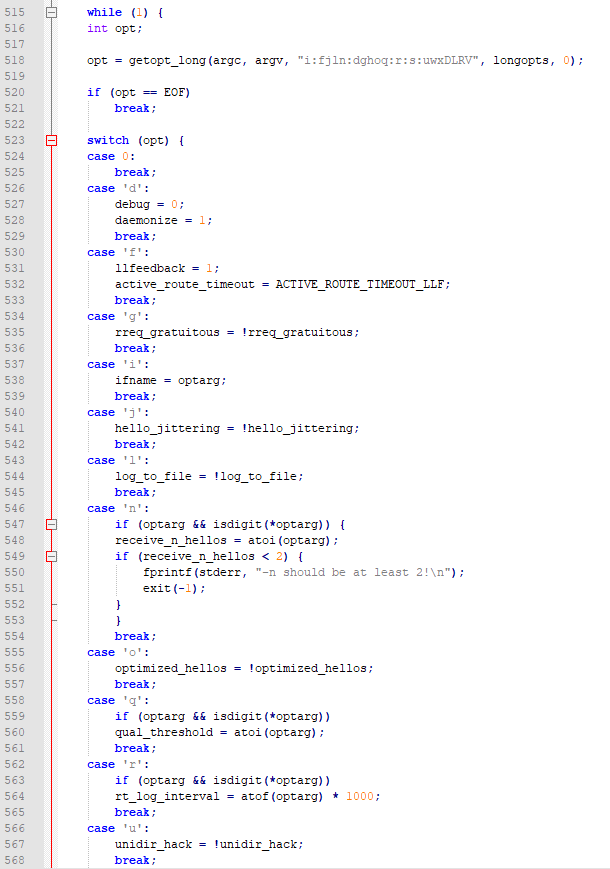
main函数主要功能是对于各个变量的初始化，包括应用到的套接字，全局链表，路由表，信号等。读取命令行参数以对相关变量进行设置，使得协议执行不同的动作。最终通过pselect轮询函数来实现节点的各个功能。

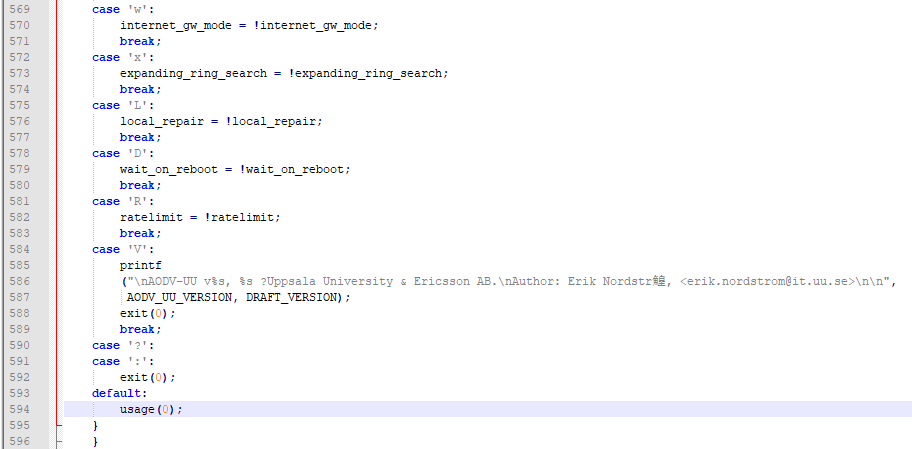




481-496 声明各变量，包括文件描述符集，计时器，信号量，读取文件名等

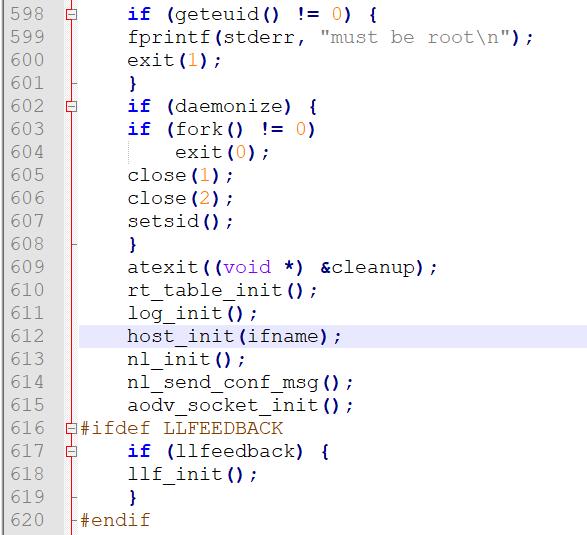
497-514 先是通过memset函数来清空结构体的内存，然后对信号结构体进行相关设置，最后将信号结构体阻塞住，将来在pselect函数中处理它。

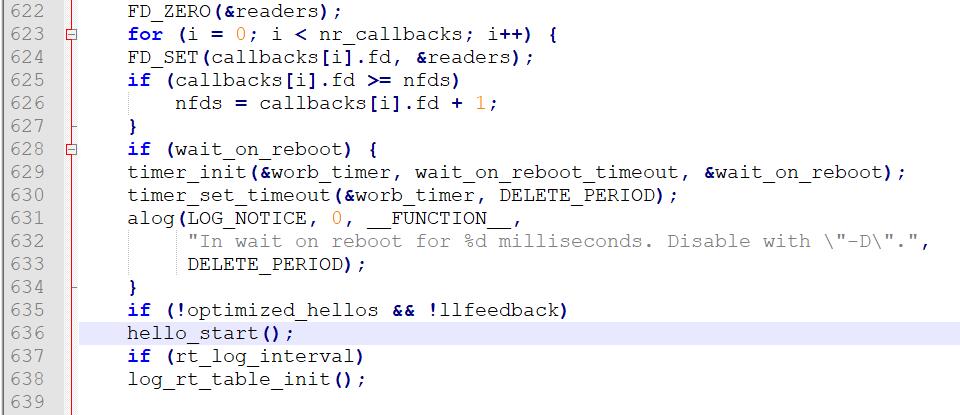




515-596 这是一个永真循环，它读取用户输入的命令行参数，然后根据输入的命令执行AODV不同的工作分支。具体操作，如下表所示：

|  |  |  |
| --- | --- | --- |
| 命令行参数 | 全称 | 说明 |
| -d | daemon | 开启守护进程模式，离开控制台窗口 |
| -f | llfeedback | 开启链路层反馈 |
| -g | force-gratuitous | 在每个传输的消息上设置gratuitous标志 |
| -i | interface | 后接端口，表示绑定到某端口；如没有，则自动分配 |
| -j | hello-jitter | 启用hello-jitter服务，默认也是启用的 |
| -l | log | 显示日志 |
| -n | n-hello | 将某节点设置成活跃邻居所需要接收到HELLO帧数 |
| -o | opt-hellos | 设置只在转发消息时才发送hello数据包 |
| -q | quality-threshold | 设置消息质量的最小阙值 |
| -r | log-rt-table | 每个一段时间记录一次路由表 |
| -u | unidir-hack | 开启检测服务 |
| -w | gateway-mode | 开启网管支持 |
| -x | no-expanding-ring | 禁用RREQ的拓展环搜索法 |
| -L | local-repair | 开启本地修复 |
| -D | no-worb | 禁用开启延迟的等待 |
| -R | ratelimit | 开启RREP和RREQ的速率限制 |
| -V | version | 输出版本信息 |
| -: | exit | 退出 |





598-601  检查AODV是否在root权限下运行，否则退出

602-608  检查变量daemonize，若不等于0，检测当前是否是父进程，是父进程则退出，不是父进程，就把输入，输出，错误流都关闭了，并且使当前进程成为新的会话组长和新的进程组长，并与原来的登录会话和进程组脱离。

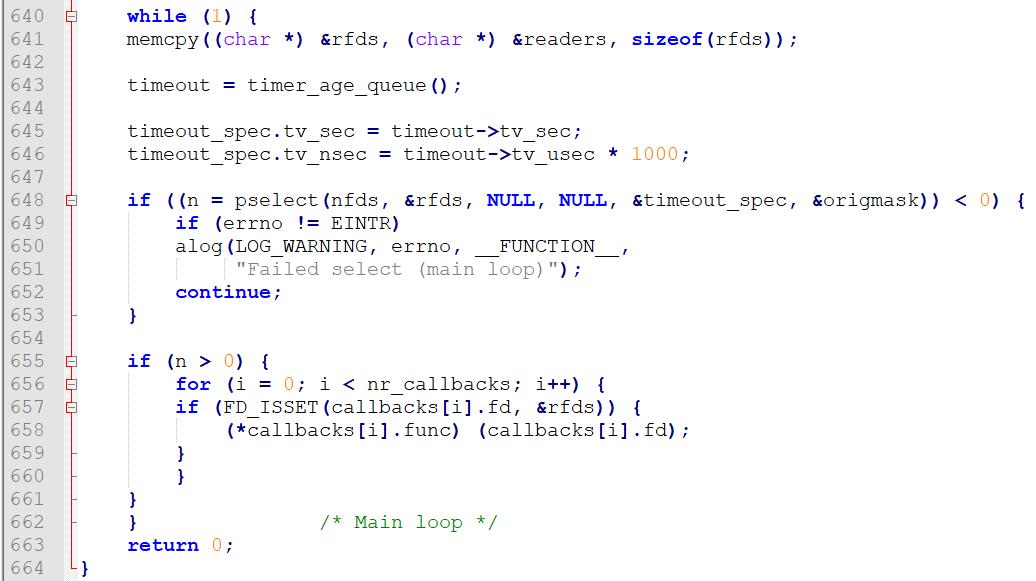
609-619  调用各初始化函数，初始化路由表，日志，各种套接字。

622-627 把需要监听的套接字描述符加入到文件描述符集中去，为以后pselect使用作准备

628-634 如果wait\_on\_reboot不为零，设置重启定时器，超出一定时间进行重启

635-637 启动HELLO帧，准备发第一个

637-369 初始化日志路由表



641-641 复制文件描述符集

643-646 设置定时器

648-653 pselect轮询函数，当文件描述符集中出现描述符对应套接字可读的时候，函数返回可读的个数n。

655-661 如果套接字可读数大于0，则通过FD\_ISSET遍历文件描述符集，判断具体是哪个可读，然后执行对应的回调函数。回调函数通过attach\_callback\_func函数来指定。

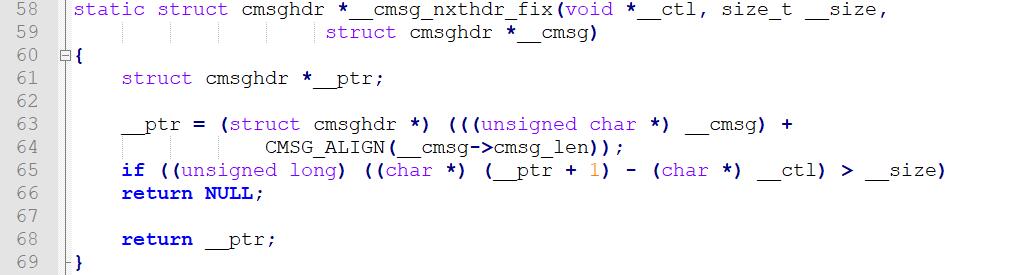
## main函数文件内其他函数以及变量的定义：

在这里以表格的形式简单说明main.c中剩下的函数功能和变量

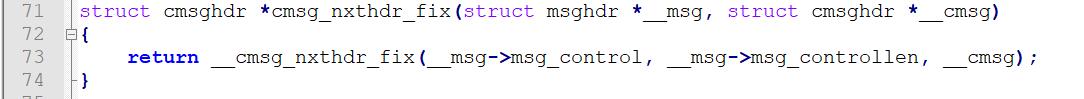
|  |  |
| --- | --- |
| 名称 | 说明 |
| voidusage(intstatus) | 打印命令行参数和说明 |
| voidcleanup() | 里面调用了各个部件的cleanup函数，实现了清空套接字、路由表等信息的功能 |
| struct callback | Callback结构体，成员变量包含一个int型的文件描述符和一个回调函数型的函数 |
| int attach\_callback\_func(int fd, callback\_func\_t func) | 增加一个callback元素 ，设置它的套接字描述符和对应的函数 |
| int find\_default\_gw(void) | 寻找节点默认网关 |
| struct sockaddr\_in \*get\_if\_info(char \*ifname, int type) | 根据套接字接口名字和其类型获取详细信息 |
| void host\_init(char \*ifname) | 初始化某个端口 |
| void load\_modules(char \*ifname) | 装载内核模块 |
| void remove\_modules(void) | 删除某个模块 |
| int set\_kernel\_options() | 设置内核选项 |
| void signal\_handler(int type) | 信号处理函数，根据不同类型的信号作不同的操作 |

## AODV\_socket.c

### \_\_cmsg\_nxthdr\_fix函数



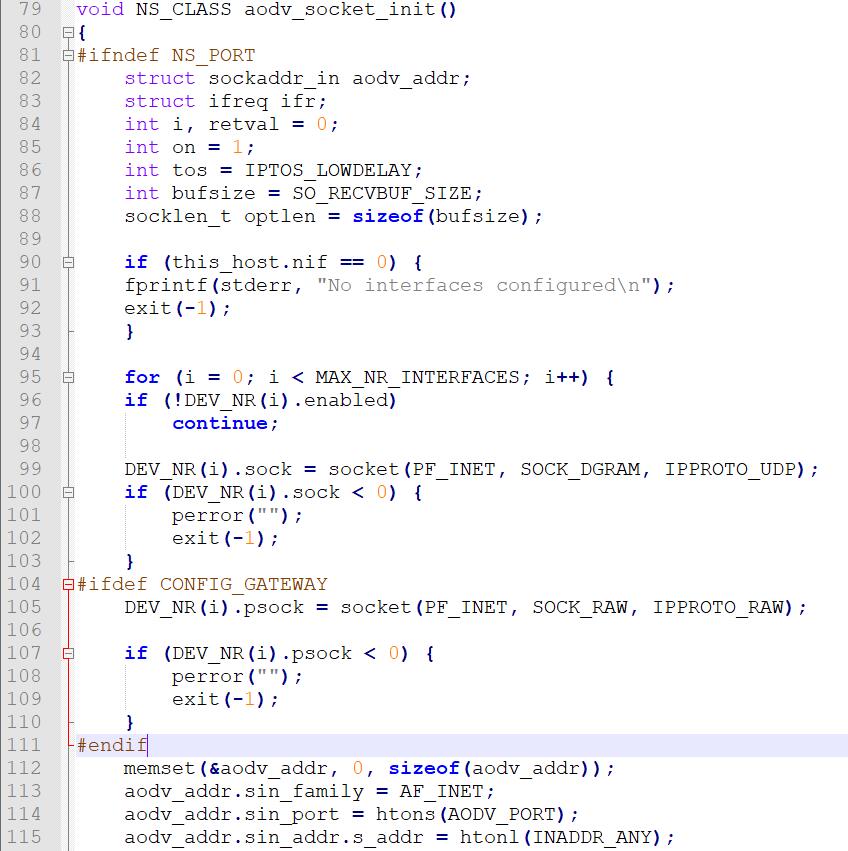
58-69首先输入的参数创建一个类型为cmsghdr的结构变量ptr，然后检查\_ptr+1减去\_\_ctl是否大于\_size,如果条件为真，则返回空，否则返回创建好的结构。该函数重写了libc里的CMSG\_NXTHDR函数。



71-74 返回调用\_\_cmsg\_nxthdr\_fix函数的结果，即构建好的结构体对象

### aodv\_socket\_init函数-套接字初始化函数

该函数主要实现了对应aodv套接字的初始化。可以清晰的看到，该套接字是用UDP协议进行传输的。可以看到函数末尾调用了attach\_callback\_func函数，将自己的套接字描述符和aodv处理函数作为参数传了进去。说明这个套接字是用来接收aodv控制帧的，当套接字可读的时候，调用了aodv\_socket\_read函数来处理接收到的包。

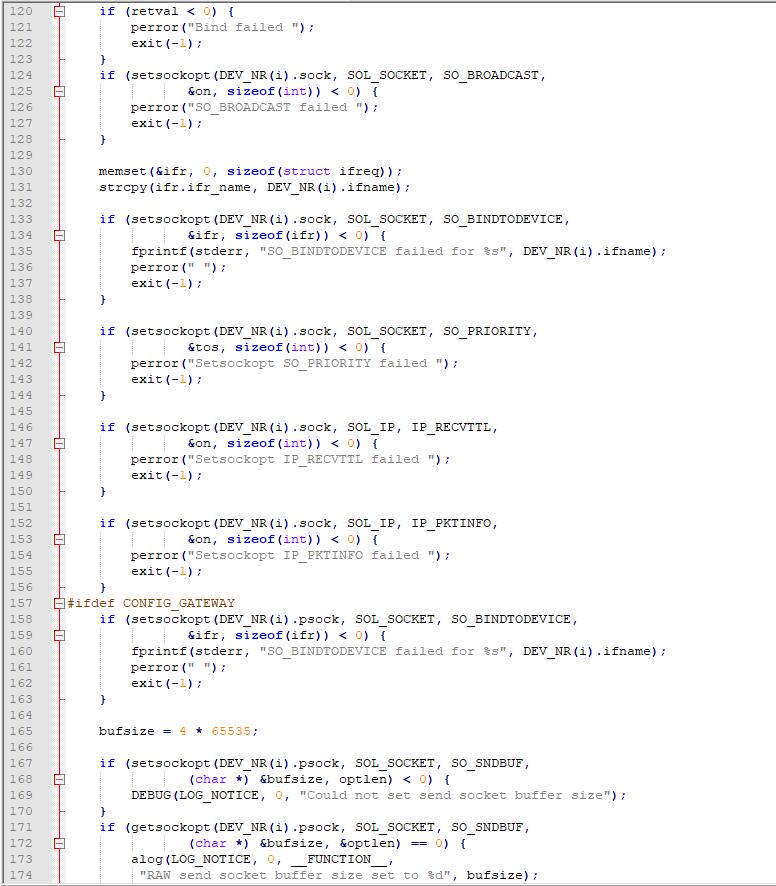


79-88 声明UDP套接字所需要的变量

90-93 如果没有可用接口的话之间退出

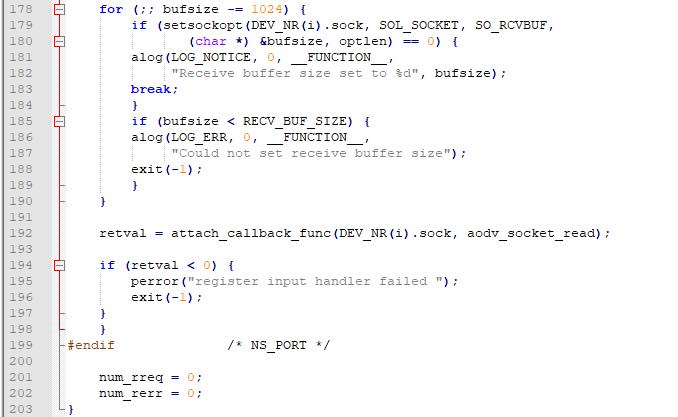
95-97 这个循环将在规定值范围内，为所有可用的接口创建套接字连接。

99-115 创建并检测生成的UDP套接字和与内核沟通的原始套接字，设置aodv地址参数





120-176 设置套接字选项，如果其中一个设定失败则显示错误，并返回



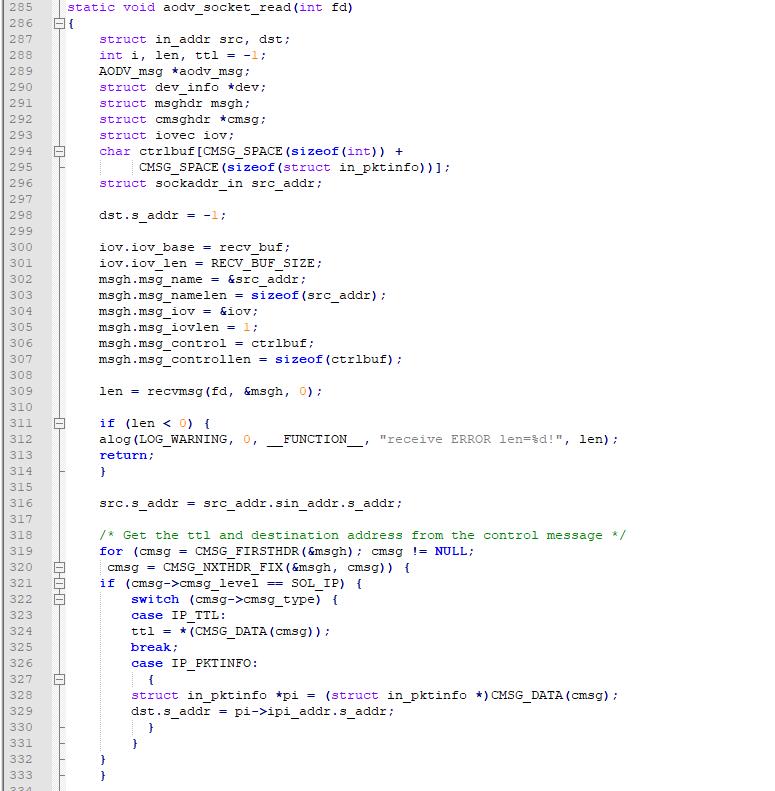
178-190 设置收控制帧的缓冲区大小，失败则显示消息并返回

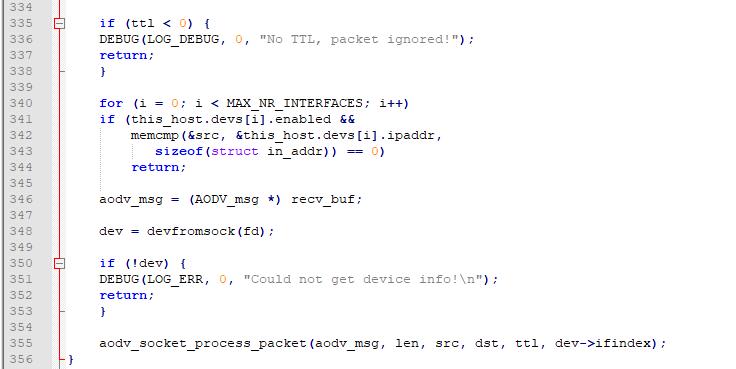
192-192 把创建好的套接字挂载到主函数的callback结构体当中

117-123 绑定套接字和AODV端口号

### aodv\_socket\_read函数-套接字读取

该函数被设定为callback结构体中某成员的回调函数，当aodv套接字可读时将会调用这个函数对收到的包进行处理。这个函数对接收到的aodv控制帧包进行过滤操作，丢弃长度小于0的和本地生成的数据包。因为发送和接受用同一个套接字进行传输，当我们需要发送帧时，把信息写入缓冲区中，套接字同样可读，会调用这个函数，所以需要判断。





287-307 声明各消息变量，以待使用

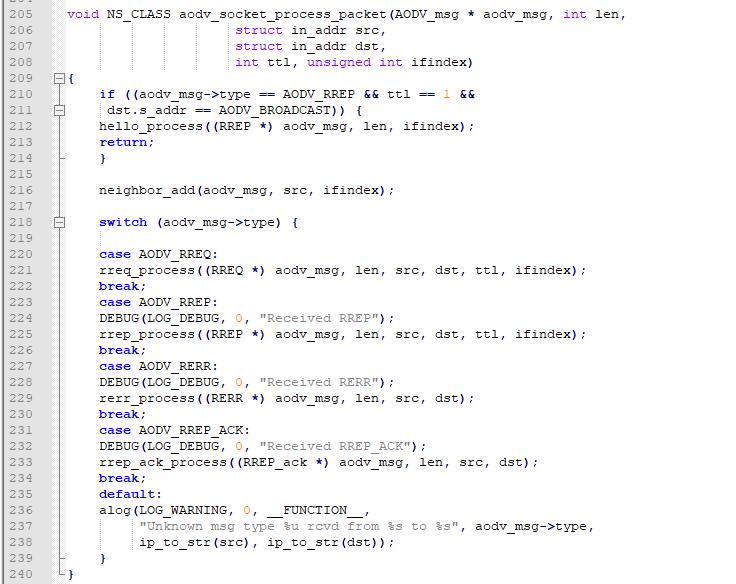
309-314 收取套接字接收到的信息，如果长度小于0，则丢弃

319-344 获取控制帧的ttl和目的地址，判断如果ttl<0或者是本节点生成的数据包则忽略掉。

346-355 提取相应消息，把包传给aodv\_socket\_process\_packet函数进行进一步处理

### aodv\_socket\_process\_packet函数-处理数据包函数

控制帧在经过aodv\_socket\_read函数初步筛选过后递交给本函数进行进一步处理，本函数主要实现的是一个分发处理功能。收到包后判断其类型，递交给对应的处理函数进行工作。

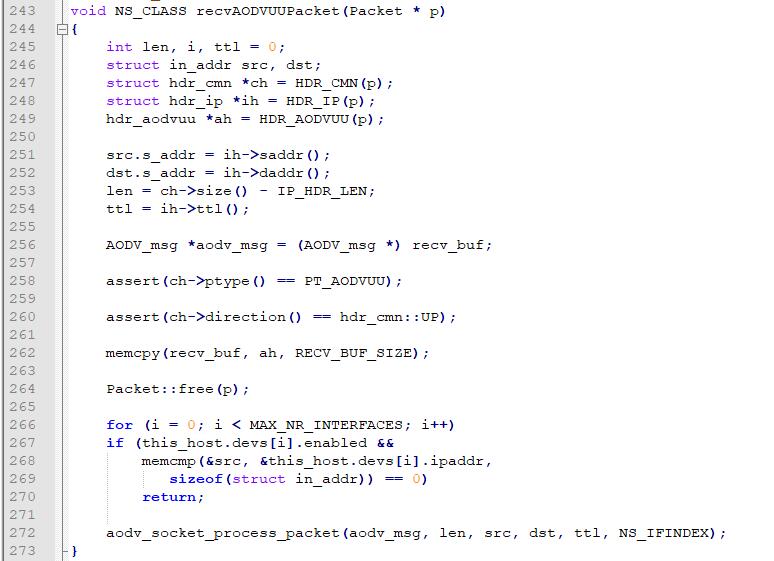


210-214 如果包是一个HELLO帧，则把该包传给hello\_process函数进行处理

226-226 根据aodv消息，src和ifindex参数添加邻居节点

218-239 提取包类型，将包递交给对应的函数进行处理

### recvAODVUUPacket函数-接收AODVUUPacket

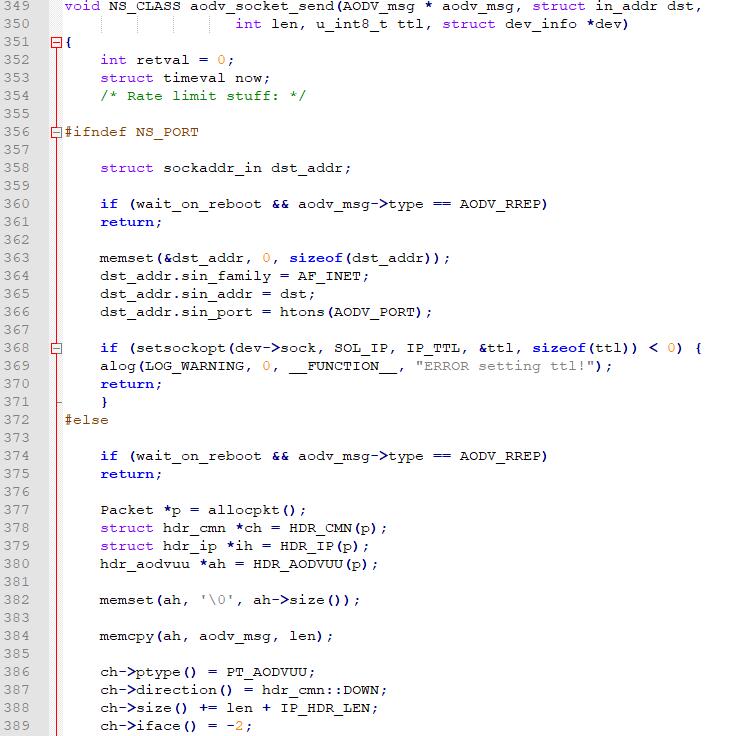


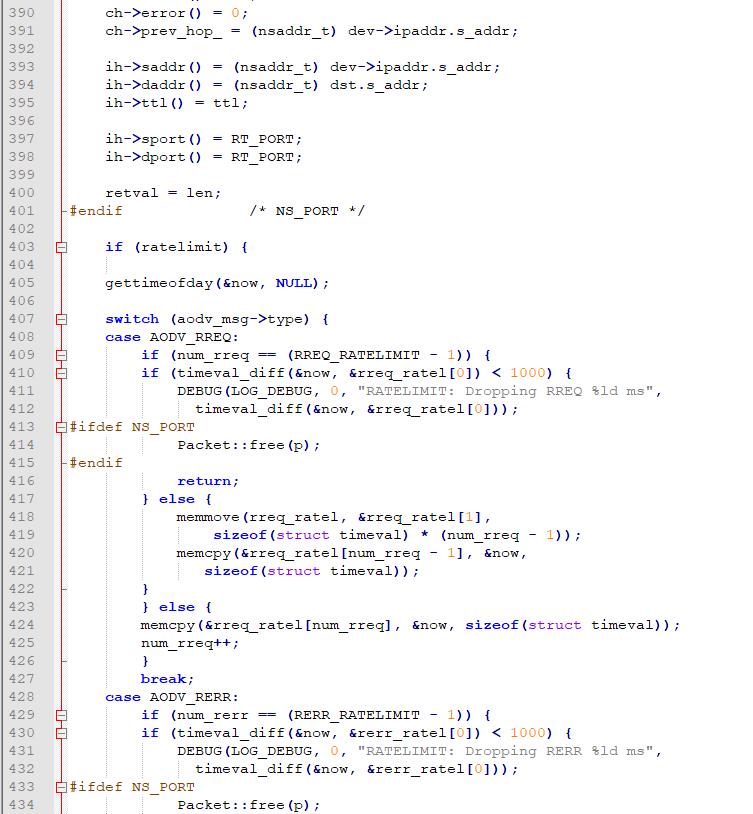
245-264 给收到的数据包先分配缓存空间，然后释放掉

266-270 检查包是否是本节点任意一个活跃接口发出的，如果是，则直接返回

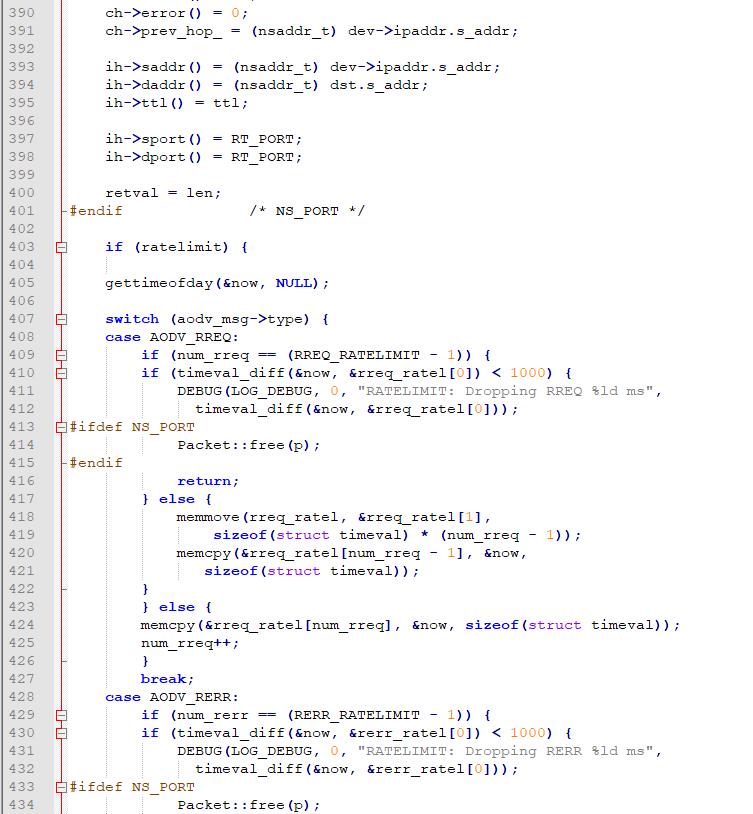
272-272 把包递交给下一函数进行处理

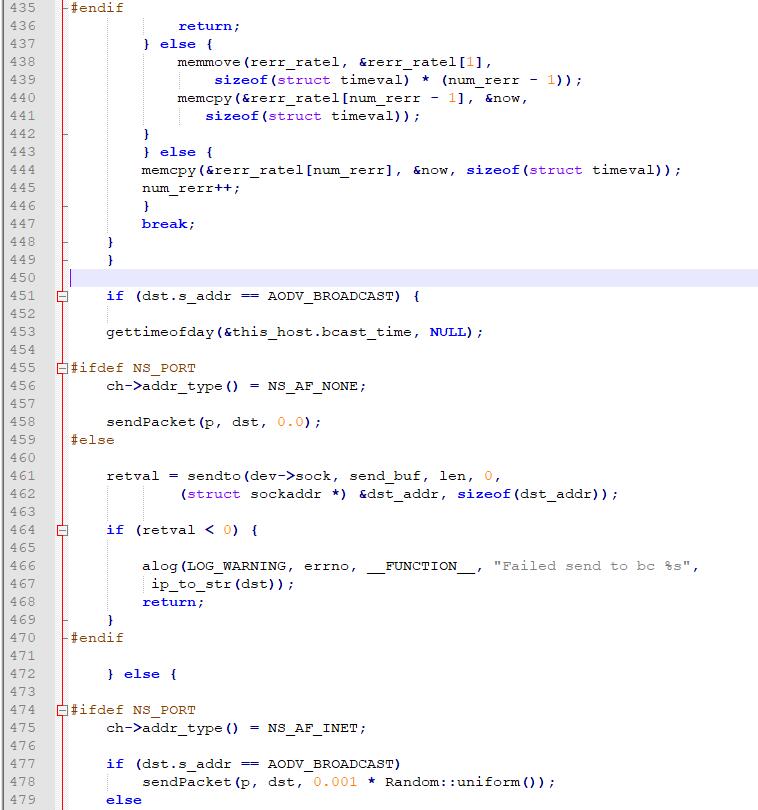
### aodv\_socket\_send函数-套接字发送函数



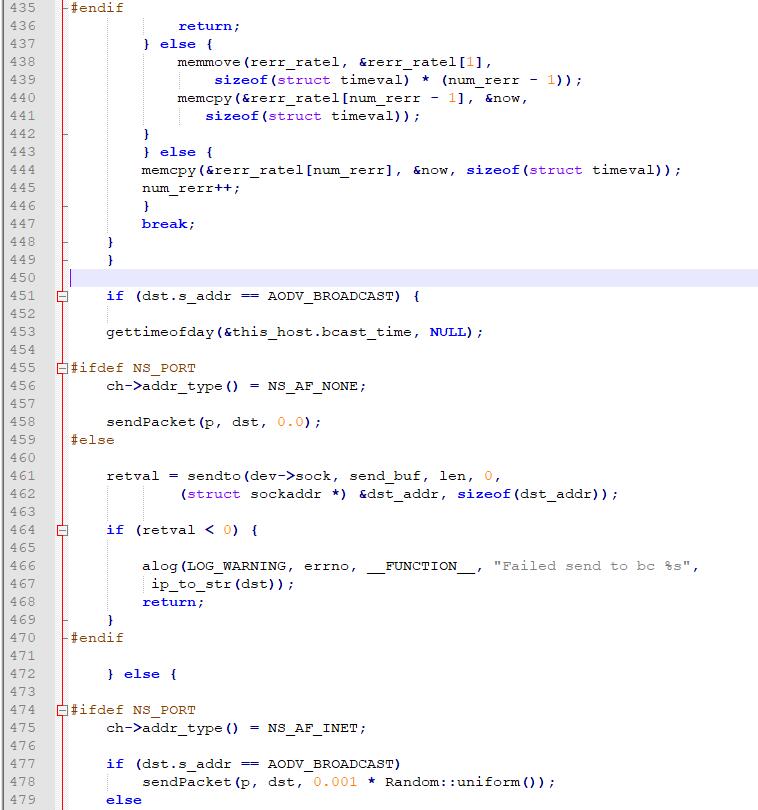


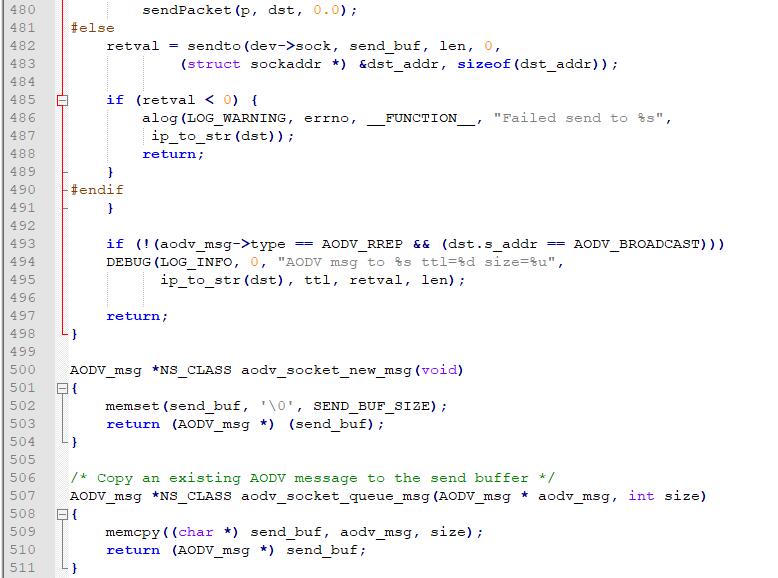
352-401 如果没有定义NS\_PORT设置头部消息和TTL，否则检查是否是RREP，不是则为包申请空间，将相关信息填入。





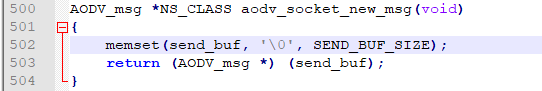
403-449 如果存在发送速率限制，检测包的最低传输速率是否比其类型对应的限定值小，如果小，则丢弃。如果符合则自身对应类型序列+1。





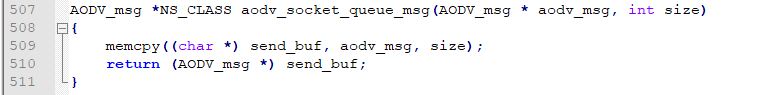
450-497 如果要发送的包是要广播的，那么更新最后广播该数据的时间，能够避免HELLO消息重复广播。

### aodv\_socket\_new\_message函数



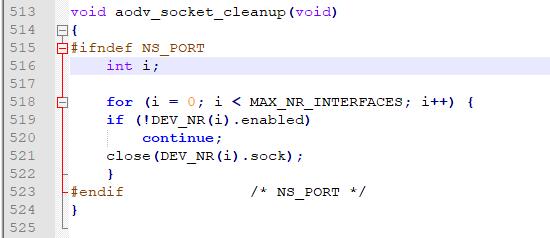
500-504 为新消息分配空间，并返回该消息的指针

### aodv\_socket\_queue\_msg函数



507-511 一个AODV消息存储在发送队列中

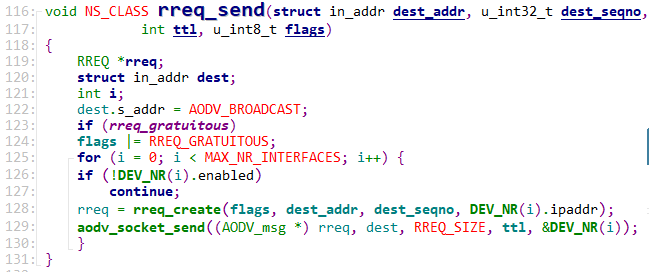
### aodv\_socket\_cleanup函数



513-525 关闭每个开放接口的套接字连接

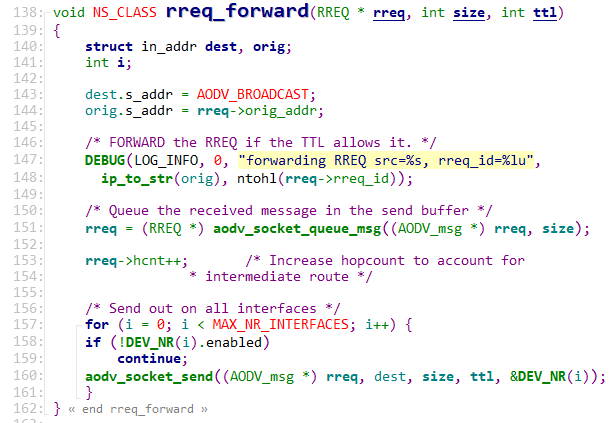
## AODV\_rreq.c

### rreq\_send函数-发送rreq消息



125-129 向所有端口广播rreq消息

### rreq\_forward函数-转发rreq消息

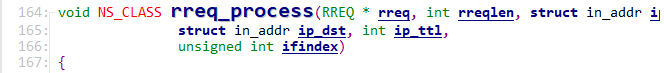


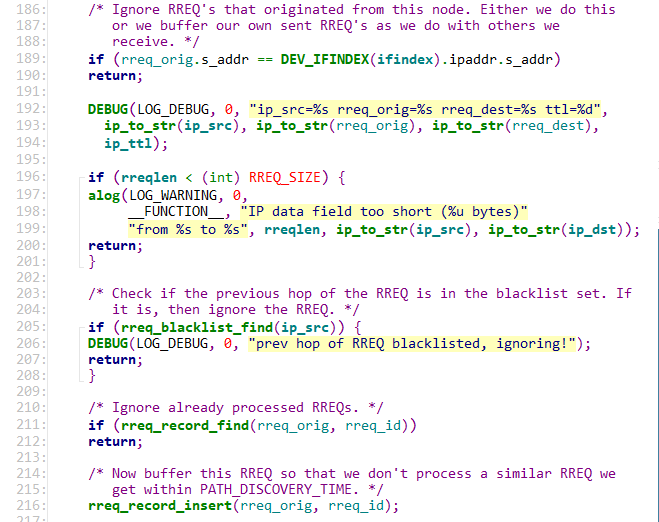
147-148 在ttl大于0之前转发接收到的消息

151-153 将接收到的rreq消息添加到接收缓存区中，并且增加跳数

157-160 在所有端口广播rreq消息

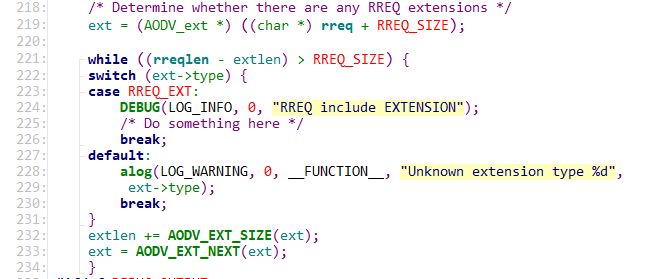
### rreq\_process函数-处理rreq消息



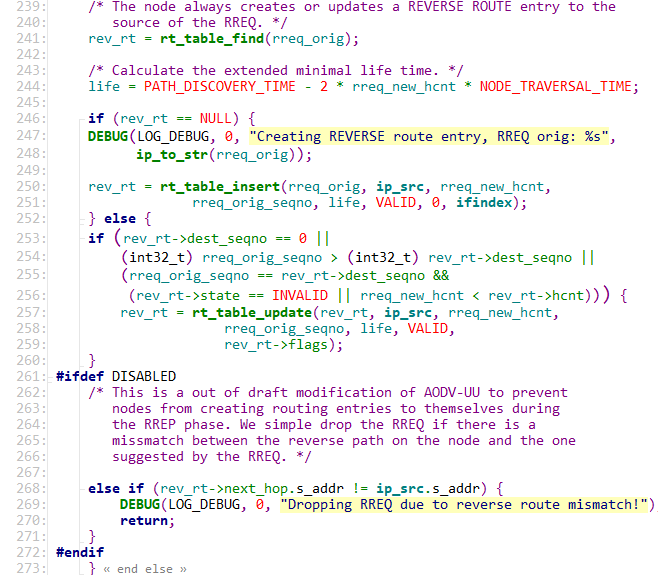


186-212 检查rreq消息，如果来自节点自身或已被加入黑名单或已有记录，则忽略该消息

216 将该消息加入record中，防止在路由发现时间内处理相同的消息



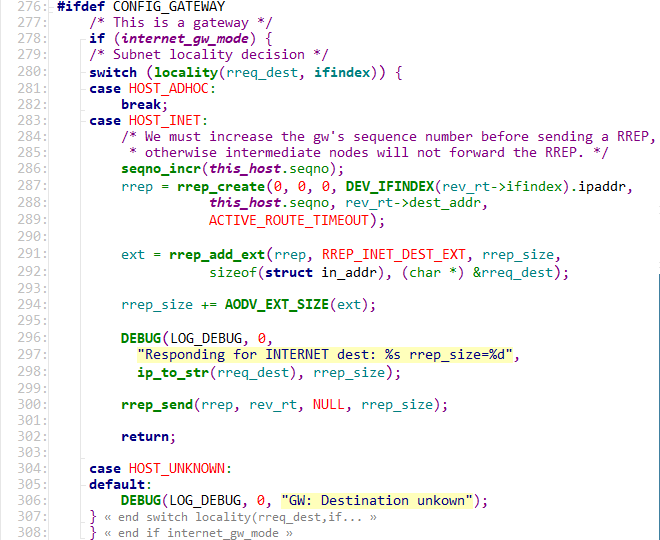
218-237 检查是否还存在RREQ拓展消息



241-273 更新反向路由，具体方法为：

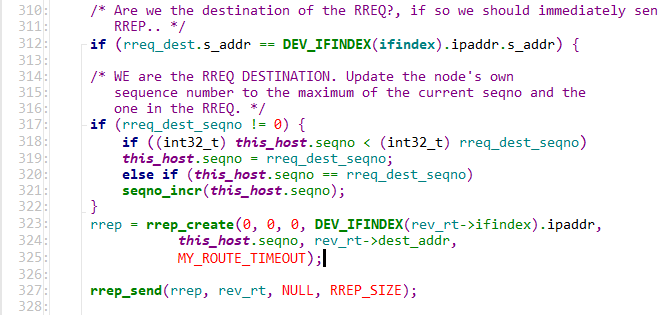
1. 如果路由表没有该rreq的源地址，则插入一条关于该源地址的路由表项；
2. 如果路由表有关于源地址的路由表项，满足下面三个条件之一则更新路由表项
3. 路由表的目的地址序列号为0；
4. Rreq的目的地址的序列号是否大于路由表项的序列号；
5. 序列号相等但是rreq跳数较少或者该路由表项因超时等失效了

268-271检查是否在RREP阶段建立了自环

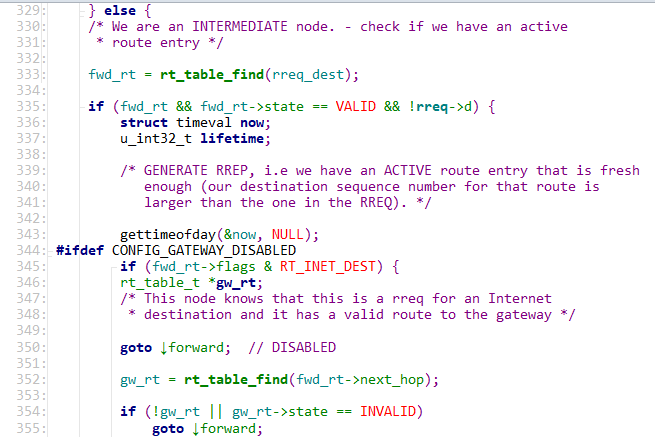


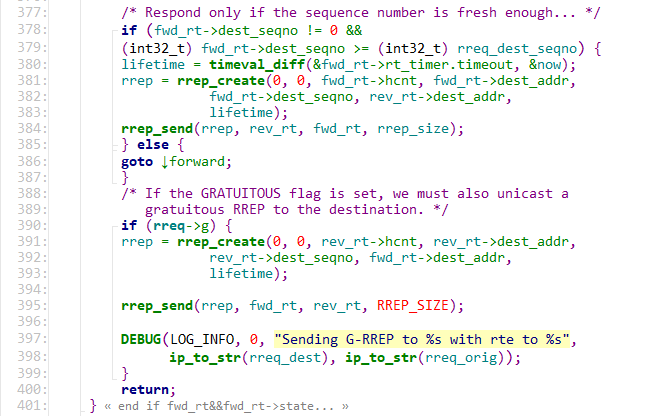
281-306 如果网关的模式，如果为HOST\_INET，则序列号自增，并创建和发送rrep消息

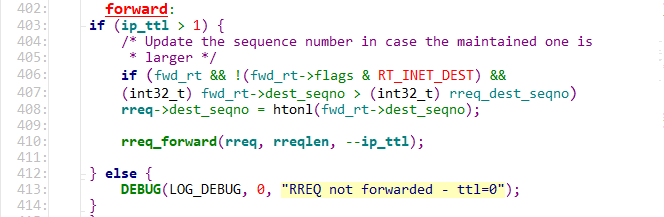
防止中间节点不转发rrep消息？



310-327 如果本节点为rreq的目标节点，更新自己的序列为rreq序列号如果自己的序列号与rreq序列目的地序列号相同，则更新自己序列号维rreq目的地序列号+1，然后回复rrep消息

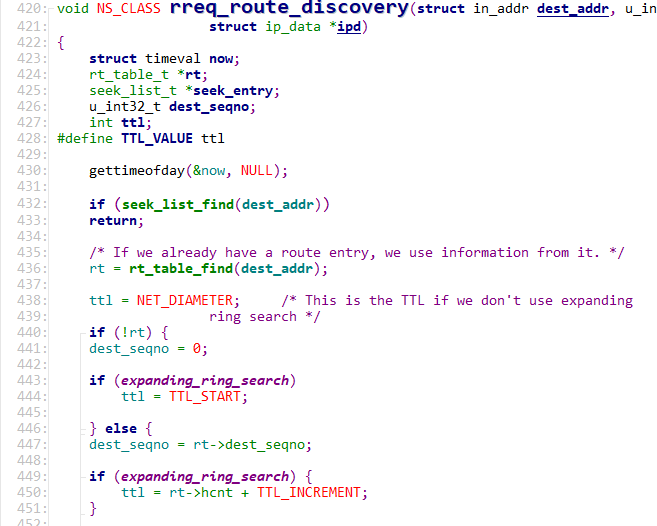


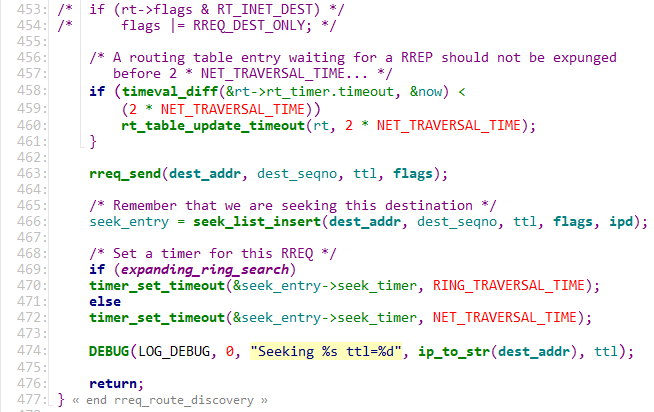




329-343,377-401，402-414如果本节点不是目标节点则查找路由表中有无该路由表项如果有有效的路由表项，当路由表的序列号比rreq目的地序列号大或者rreq的g为1的时候回复rrep否则转发ttl>1的rreq消息

### rreq\_route\_discovery函数-路由发现过程





432 如果源节点路由表中存在到目的节点的路由，则不调用AODV协议

434-463 如果没有，源节点广播rreq消息并记录在seek\_list里面

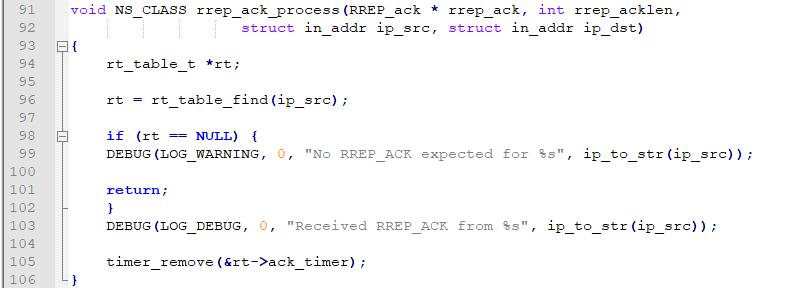
其他函数

|  |  |  |
| --- | --- | --- |
| 函数名 | 行数 | 说明 |
| RREQ \*NS\_CLASS rreq\_create(u\_int8\_t flags, struct in\_addr dest\_addr, u\_int32\_t dest\_seqno, struct in\_addr orig\_addr) | 61-96 | 创建rreq消息，设置jrgd四个参数值。当一个节点要产生RREQ洪泛之前，必须要增大序列号， |
| AODV\_ext \*rreq\_add\_ext(RREQ \* rreq, int type, unsigned int offset, int len, char \*data) | 98-114 | 增加rreq拓展消息 |
| void NS\_CLASS rreq\_local\_repair(rt\_table\_t \* rt, struct in\_addrsrc\_addr,struct ip\_data \*ipd) | 480-540 | 作用与路由发现函数类似 |
| NS\_STATIC struct rreq\_record \*NS\_CLASS rreq  \_record\_insert(struct in\_addr orig\_addr,u\_int  32\_trreq\_id) | 542-572 | 记录rreq消息 |
| NS\_STATIC struct rreq\_record \*NS\_CLASS rreq\_record\_find(struct in\_addr orig\_addr,u\_int  32\_t rreq\_id) | 574-587 | 查找rreq消息是否在记录中 |
| void NS\_CLASS rreq\_record\_timeout(void \*arg) | 589-595 | 销毁记录 |
| struct blacklist \*NS\_CLASS rreq\_blacklist\_insert  (struct in\_addr dest\_addr) | 597-621 | 将某目的地址加入黑名单 |
| struct blacklist \*NS\_CLASS rreq\_blacklist\_find  (struct in\_addr dest\_addr) | 623-634 | 查找目的地址是否在黑名单中 |
| void NS\_CLASS rreq\_blacklist\_timeout(void \*ar  g) | 636-643 | 销毁黑名单 |

## AODV\_rrep.c

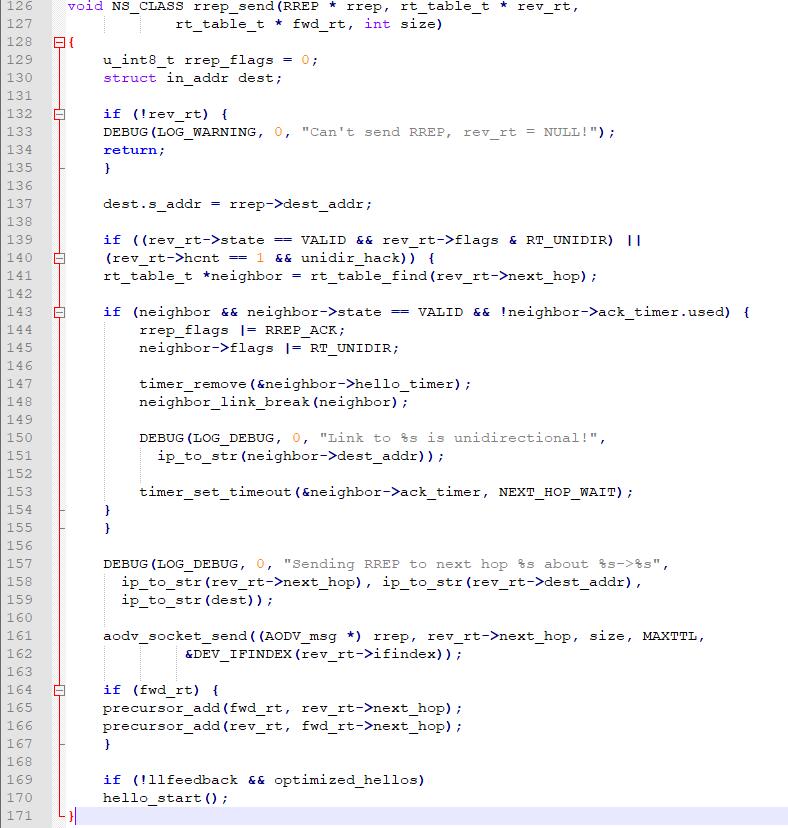
路由节点进行路由发现的时候，RREP路由控制帧会从目的节点或中间节点单播回原节点，这时候中间节点和源节点都会收到RREP帧，所以需要有RREP帧的处理函数。同时中间节点需要发送RREP帧，所以还要有生成和发送RREP帧的函数。

### rrep\_ack\_process函数



91-106 判断路由表中是否有该包的源IP相关路由项，如果没有就说明该消息非法，如果有则在控制台显示相关信息，移除ack计时器。

### rrep\_send函数-发送rrep消息



132-135 如果当前节点路由表为空，那么不能发送RREP消息

139-141 检查是否应该发送RREP\_ACK消息

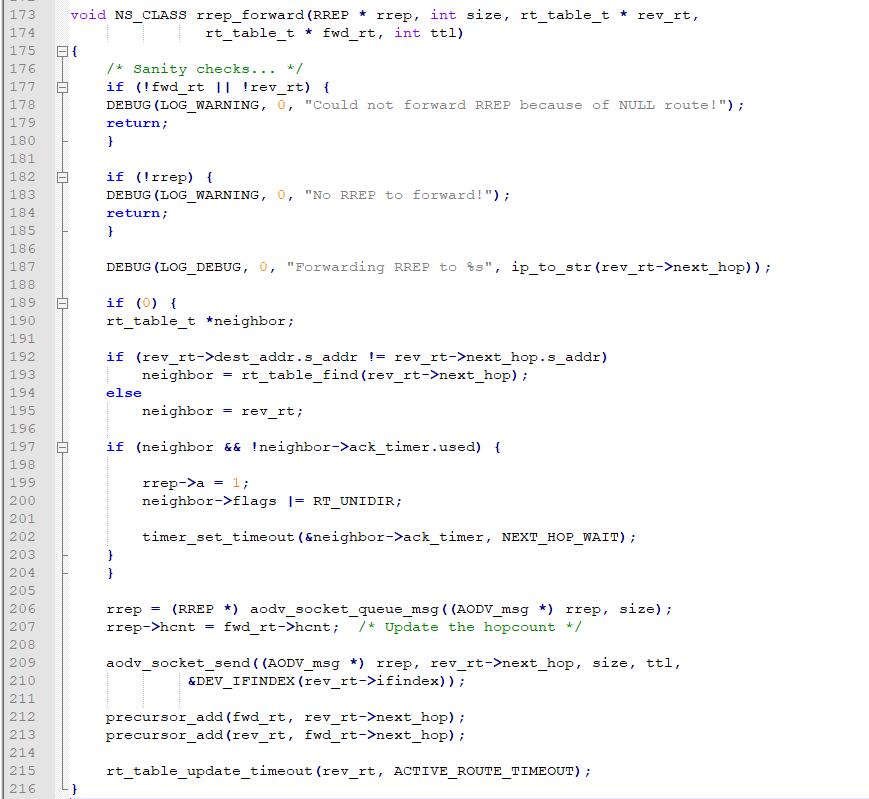
143-155 如果邻居存在并且邻居的状态是有效的，计时器还没使用，那么就将包的flag设置为RREP\_ACK，邻居标记为非单向，移除邻居的hello计时器，设置ack计时器。

157-159  更新路由表项

161-161 交由aodv\_socket\_send函数去发送消息

164-167 更新fwd\_rt和rev\_rt的前驱节点

### rrep\_forward函数-转发rrep消息



176-180 当fwd\_rt或rev\_rt不存在时，无法进行转发，因此返回；

182-185当rrep是空时，返回

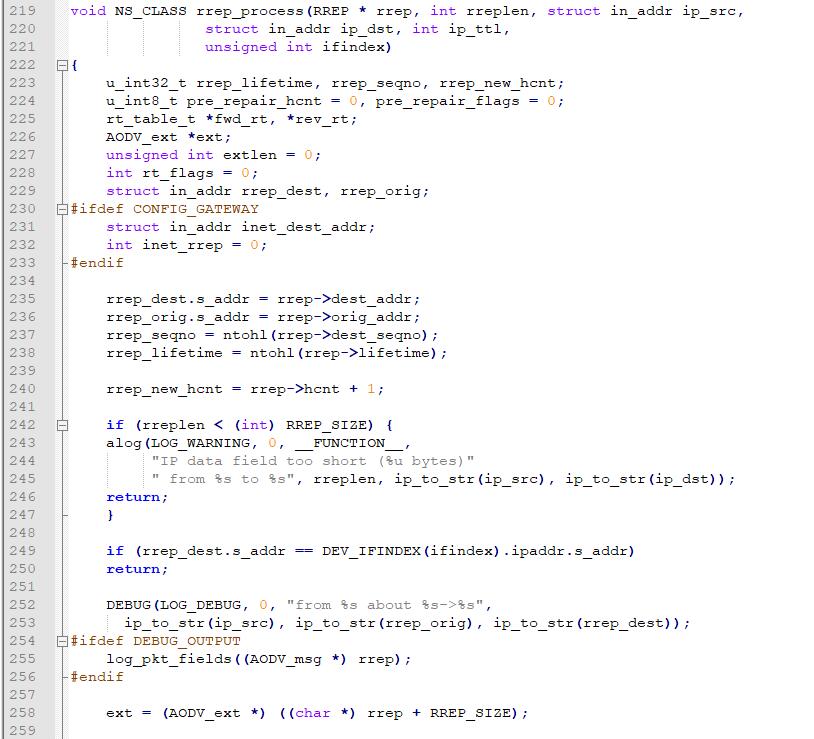
192-195 如果RREP消息的源IP地址不是相邻节点，那么我们必须找到通向源节点的下一跳IP地址。

197-204 如果我们收到邻居的RREQ控制帧，那么我们可能面临单项连接，需要将该链路设置为非单向，然后设置定时器，请求RREP\_ACK。

206-216 ，将RREP加入消息队列，交由下一函数发送，更新前驱列表，更新路由表项的有效时间

### rrep\_process函数-处理rrep消息

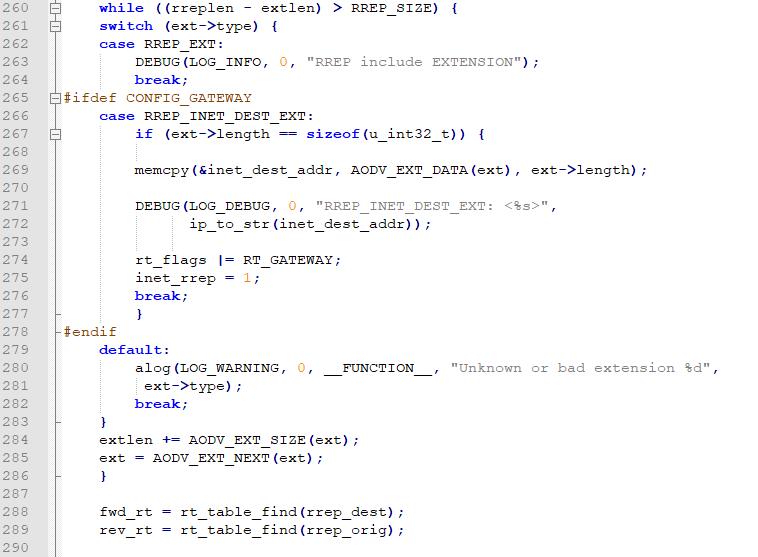
当套接字接收到RREP回复帧的时候，经过初步的过滤操作，最终会递交给这个函数进行处理。一个节点接受到RREP回复帧的时候需要判断是否是自己发出的RREQ回复的，如果不是则查询路由表项，将RREP重新单播回去。



223-240 声明变量，提取RREP回复帧中的各种消息，方便后续使用

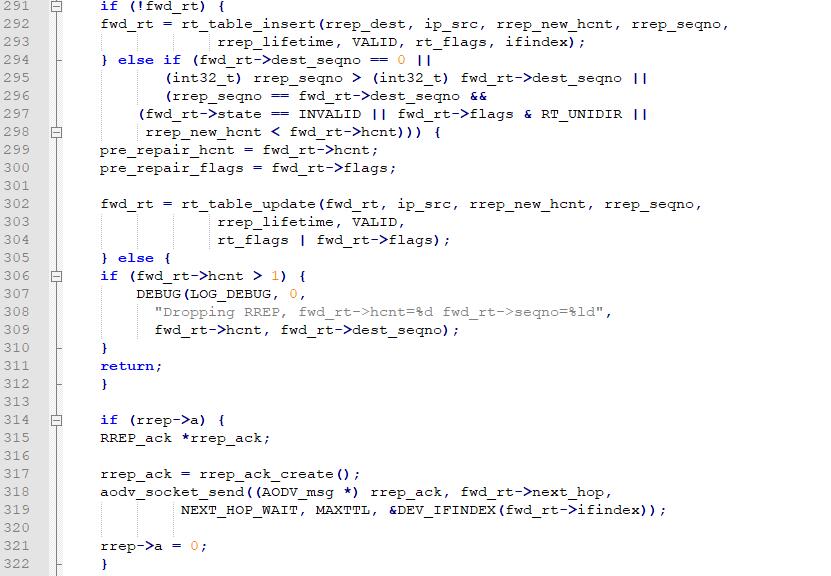
242-247 如果RREP包的长度小于限定值，显示错误信息，返回

249-250 如果是自己发出的包，直接忽略



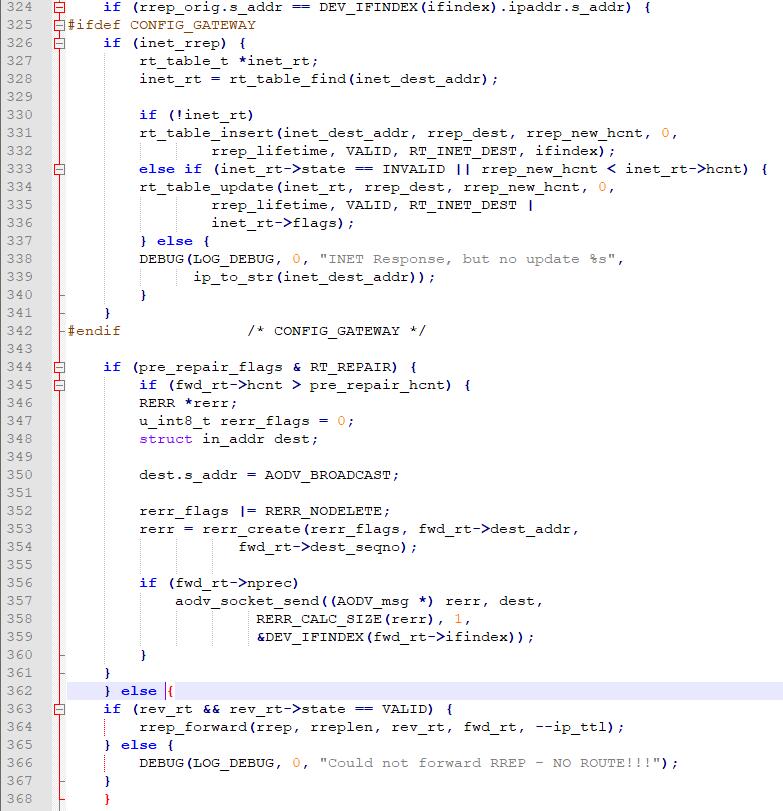
260-286 如果包的长度减去扩展长度仍大于RREP包的设定大小则进入该循环。该循环对RREP扩展和目的节点的拓展进行处理

288-290 获取有关RREP目的节点和源节点的路由表项以待后续使用



291-312 如果没有关于目的节点的表项，那么就在路由表新加入该表项，如果有而且序列号大于路由表项记录的序列号，那么更新路由表，否则忽略

314-322 检测是否需要发送RREP\_ACK，需要则发送



324-368 如果源地址就是本节点，那么如果定义了默认网关，就进行对应的更新和插入操作。如果断开的节点处于被修复状态，一个NODELETERERR应该被发送到源节点，源节点重新发起路由发现过程。如果源地址不是本节点，判断是否需要转发RREP包，需要转发则转发。

369-370 根据条件，决定是否开启HELLO帧。

### 其他函数

一些实现上与RREQ帧重复的和不是那么重要的函数以表格的形式展示，就不进行深入解读了

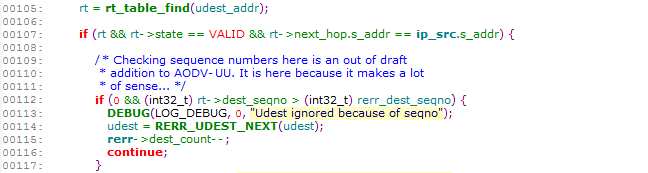
|  |  |
| --- | --- |
| 名称 | 功能说明 |
| RREP \*NS\_CLASS rrep\_create(u\_int8\_t flags,  u\_int8\_t prefix, u\_int8\_t hcnt, struct in\_addr dest\_addr, u\_int32\_t dest\_seqno, struct in\_addr orig\_addr, u\_int32\_t life) | 该函数通过传入的参数设置RREP包的prefix，hop count，目的IP和对应序列号，源IP和对应序列号等信息，返回一个新创建好的RREP包 |
| RREP\_ack \*NS\_CLASS rrep\_ack\_create() | 创建RREP\_ACK控制帧 |
| AODV\_ext \*NS\_CLASS rrep\_add\_ext(RREP \* rrep, int type, unsigned int offset, int len, char \*data) | 为RREP包添加扩展，并返回拓展后的包 |
| int rrep\_add\_hello\_ext(RREP \* rrep, int offset, u\_int32\_t interval) | 在RREP中包括Hello Interval扩展并返回新的偏移量 |

## AODV\_rerr.c

### rerr\_add\_udest函数-添加不可达目的节点

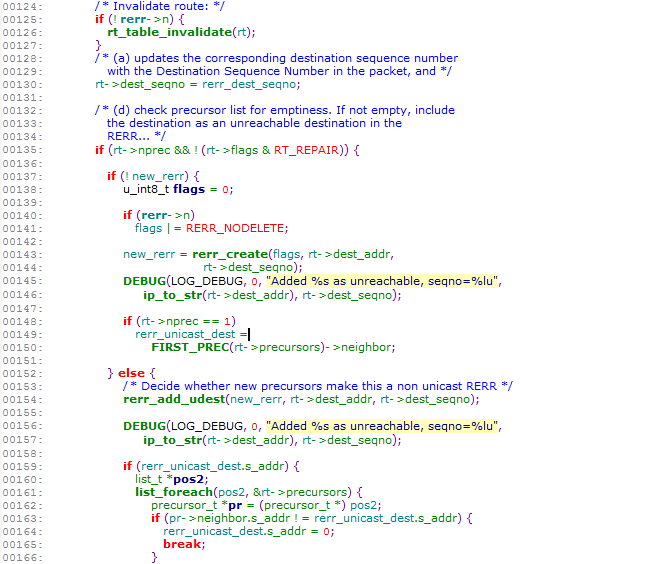
定义rerr\_udest类型的ud，设置其值

### rerr\_process函数-处理rerr消息



105-107 在不可达目的地列表中查找

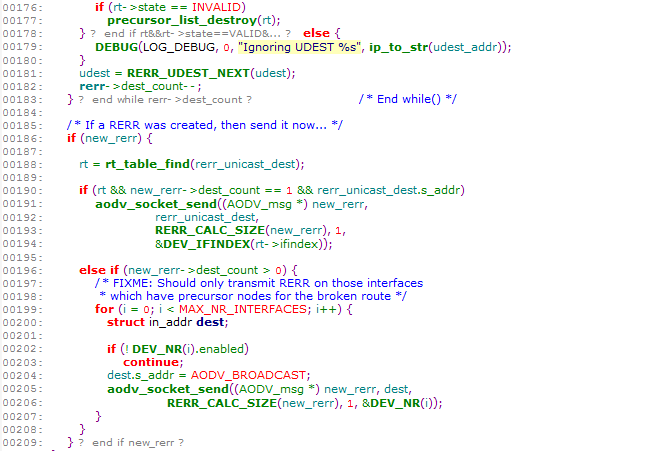
112-116 检查rerr消息中的不可达结点的目的序列号与路由表项中保存的最新的相应结点的序列号，若后者大于前者，则说明消息已过期，直接返回



124-130 调用rt\_table\_invalidate使rt无效，随后用rrer消息中的目的节点序列号更新rt的目标节点序列号

135-150 如果先驱列表为空，则将目的IP地址加入不可达目的地列表中

152-170

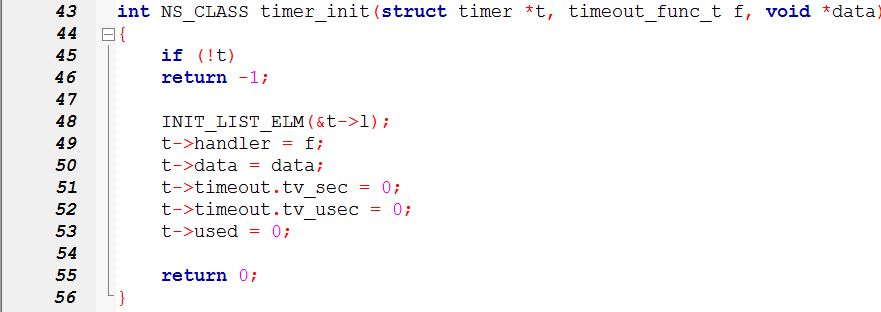


176-183 为所有不可达节点删除先驱列表

186-209 如果rerr的目标节点只有一个，则单播该消息；否则，我们仅仅向那些包含断裂路由的先驱节点的端口发送rerr消息

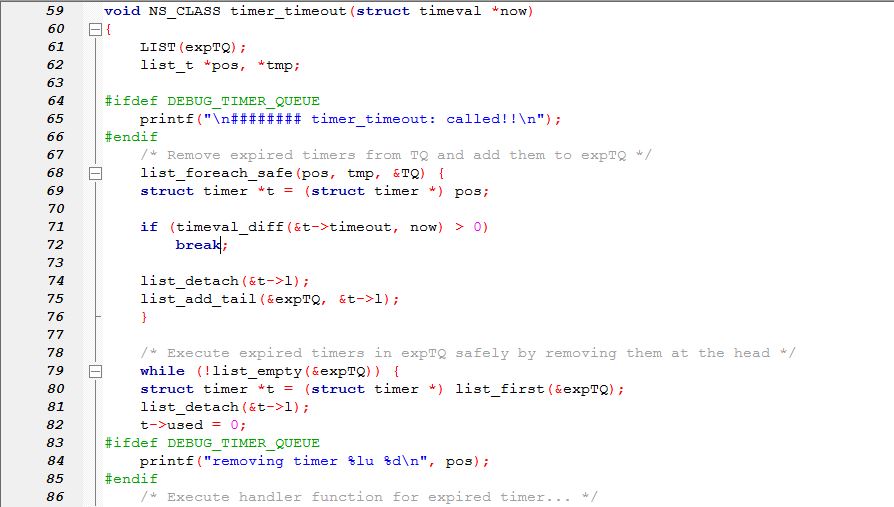
## Time\_queue.c

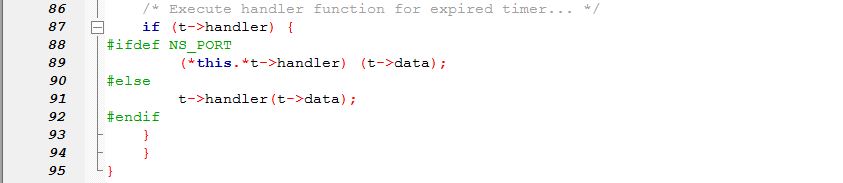
### NS\_CLASS timer\_init函数-记时器初始化函数



设定一个新的计时器，如果没有计时器的结构体便退出，如果有，便对计时器进行赋值。

### NS\_CLASS timer\_timeout函数-计时器超时，调用此函数进行清理工作





当计时器超时，调用此函数进行清理工作

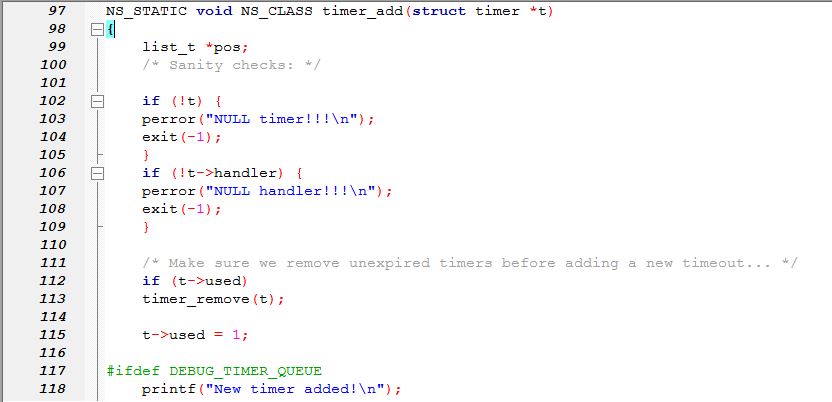
61：创建list\_t的结构体，结构体名为expTQ，并且将该结构体的两个指针赋值

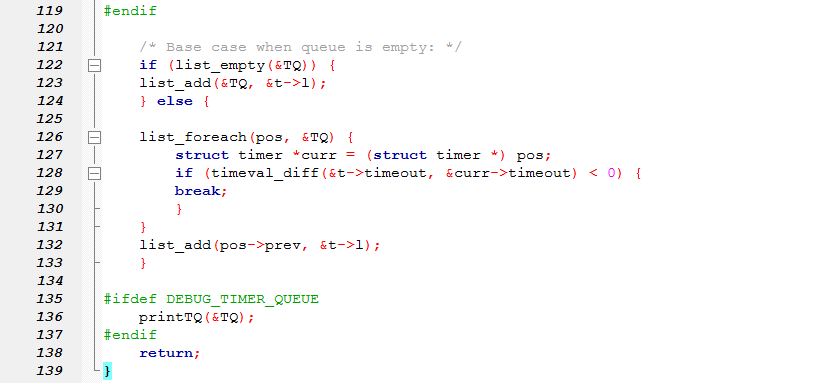
62：创建list\_t的结构体的指针，pos，tmp

68-76：遍历头指针为TQ的队列，如果此时计时器超时则退出循环，并且将整个队列的排列顺序倒置

79-94：如果队列expTQ不为空，就将expTQ的首指针赋值给结构体\*t，并将该指针从队列中删除，然后t中的uesd值设为0，

### NS\_CLASS timer\_add函数-在计时器队列中添加计时器函数





99：定义了list\_t的指针\*pos

102-105：如果t构造失败，返回-1

106-109：如果t->handler构造失败，返回-1

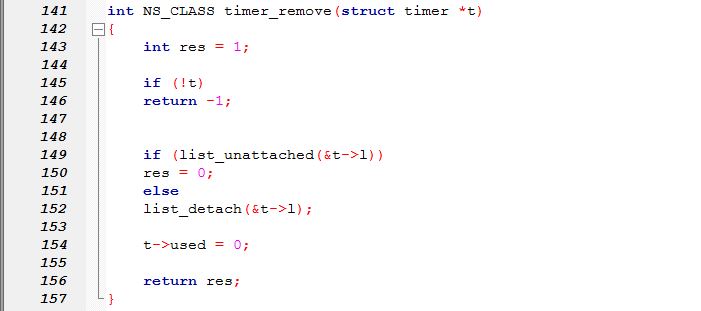
112-113：如果t->used值为1，表明此定时器已经被使用，便移除此定时器

115：used置为1，表明此定时器已经开始被使用

122-124：如果定时器为空，则直接添加一个定时器

126-131：如果定时器不为空，则遍历整个定时器队列，如果这个定时器已经超时，则退出这个循环，否则便将定时器插在定时器队列的末尾。

### NS\_CLASS timer\_remove函数-计时器队列中移除计时器



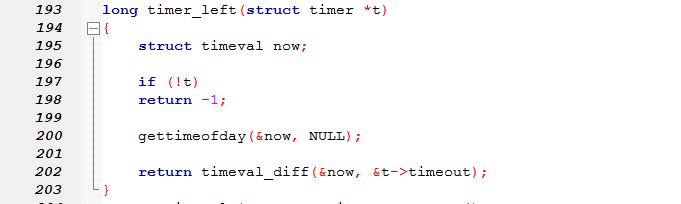
143：定义并初始化res

145-146：如果计时器t的定义错误，name便返回-1

149-150：如果这个定时器并没有添加到队列中，那么便将res值置为0

151-152：否则，便将定时器t删除，并将used值置为0，并返回res

### timer\_left函数-计算计时器剩余时间



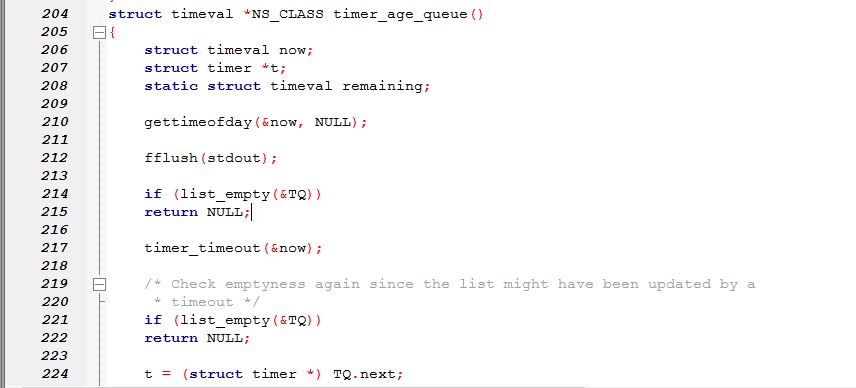
195：定义timeval结构体 now

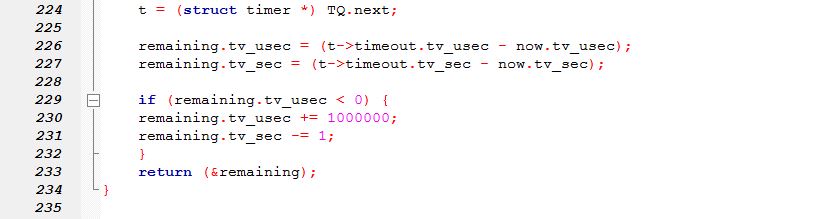
197-198：如果t的结构体构造不正确，返回-1退出

200：得到当前时间

202：返回计时器的剩余时间

### timer\_age\_queue函数-计算计时器队列的生存时间





206-208：定义两个timeval结构体，分别为now，remaining，和一个timer的结构体 t

210：获取当前时间

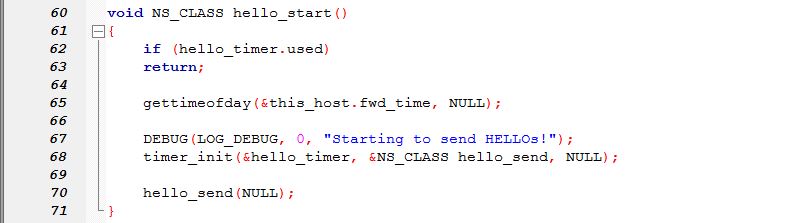
212：对标准输出流的清理，及时地打印数据到屏幕上

214-215：如果队列为空，则返回空

221-233：再次检查空白，因为列表可能已经超时更新，并对更新的值重新赋值

## Aodv\_hello.c

### NC\_CLASS hello\_start函数-初始化hello消息



62-63:检查hello消息的定时器有没有开启，如果hello\_timer.used的值为1，则表明计时器已经开启，hello消息已经发送，则返回。

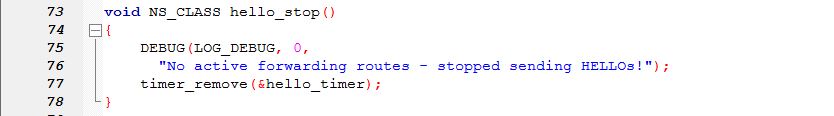
65：获取当前时间

67：将DEBUG的参数设定，表明即将发送hello消息

68：初始化hello消息的定时器

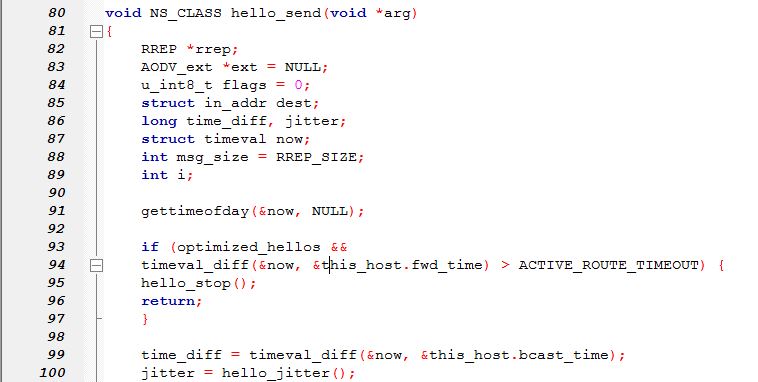
70：发送hello消息

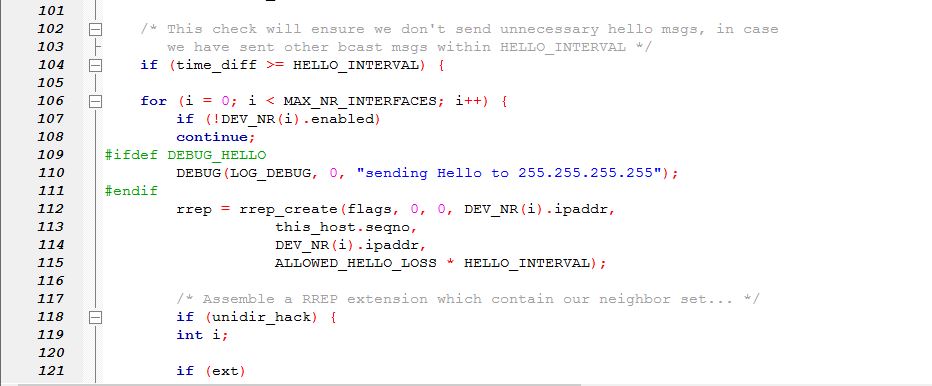
### NC\_CLASS hello\_stop函数-停止发送hello消息

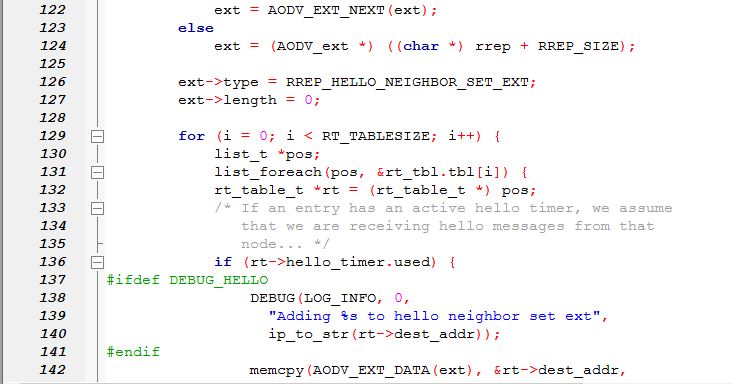


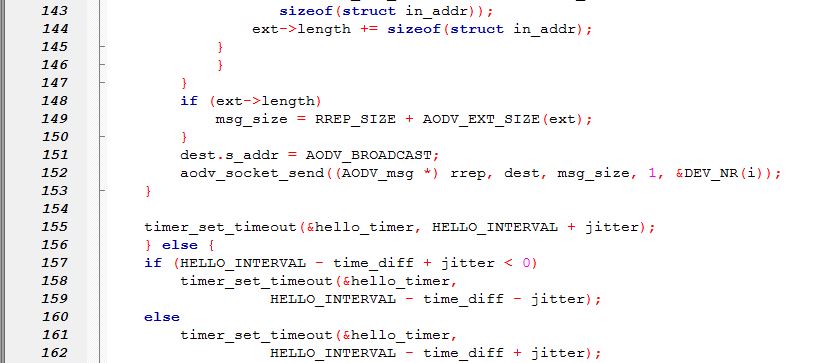
75-76：调用DEBUG函数输出信息：No active forwarding routes - stopped sending HELLOs!

77：从计时器队列中移除hello消息的计时器









81-89：定义一个RREP类型的指针：rrep

dest:定义in\_addr类型的结构体，用于存放目标地址

now：定义timeval类型的结构体，用于存放当前时间

msg\_size：用于存放RREP消息的长度

91：获取当前时间

93-97：如果当前时间和上次发送数据包的时间差大于路由超时时间，则表明此节点和周围的节点的连接已经断开，便取消发送hello消息。

99：time\_diff：当前时间和上一次广播的时间差

104：判断time\_diff和HELLO\_INTERVAL的大小，以避免发送不必要的hello消息

106-108：构建循环查看每个设备的情况

109-110：如果已经宏定义了DEBUG\_HELLO则广播hello消息，并输出sending Hello to 255.255.255.255的消息

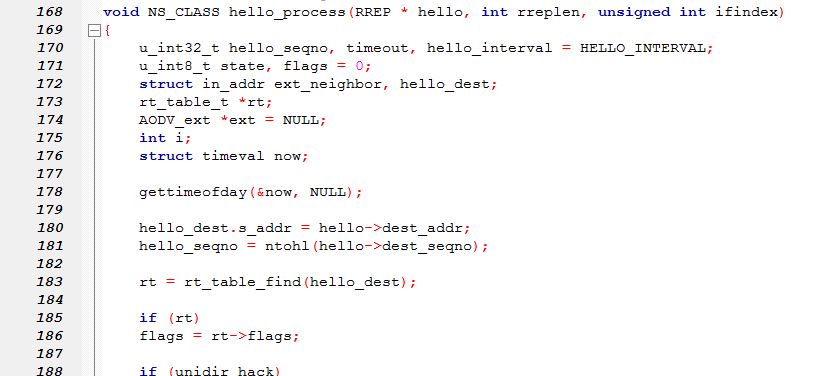
112-115：创建了hello消息

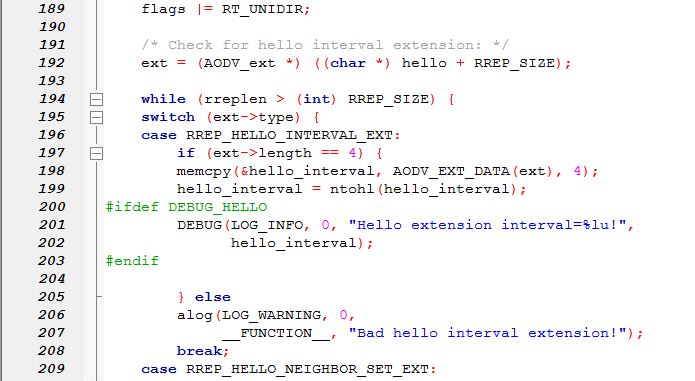
118-127：组装一个包含邻居组的RREP扩展

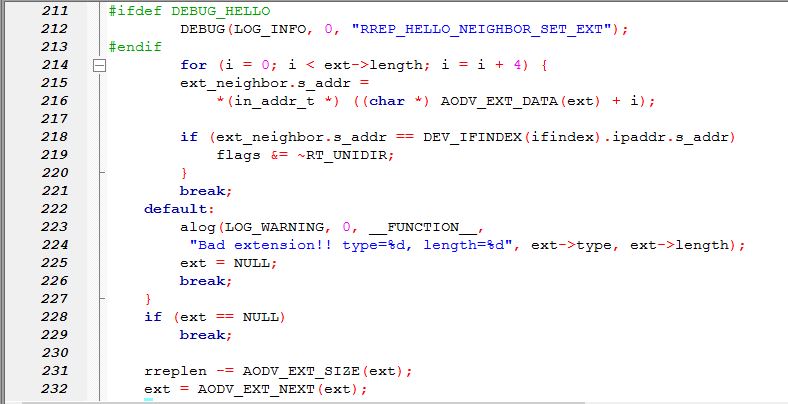
137-140：如果定义了DEBUG\_HELLO，那么我们便把rt的所有dest\_addr都添加到hello neighbor set ext

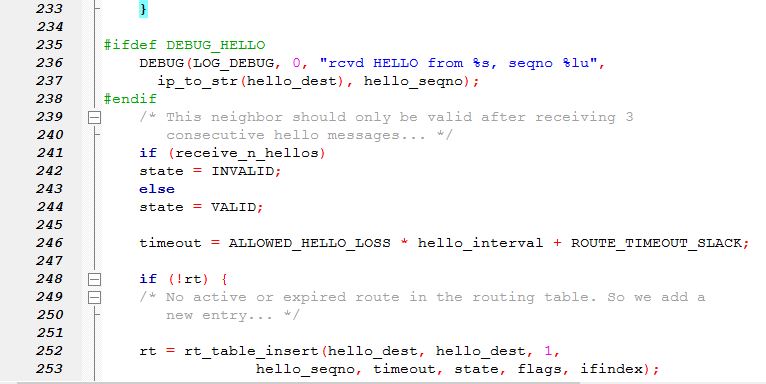
141-147：否则，我们便将rt->dest\_addr的长度为 in\_addr的结构体长度的字节数全部复制到AODV\_EXT\_DATA中，并且更新ext的长度。

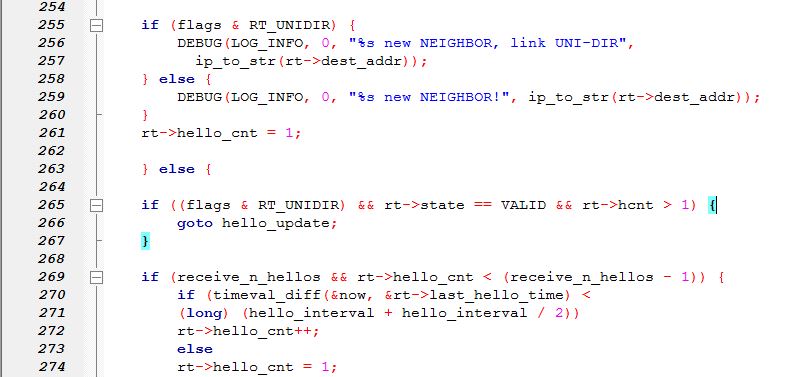
### Hello\_process函数-处理hello消息











170-175：hello\_seqno：32位的无符号的数，代表hello消息目的节点的序列号

timeout：32位的无符号的数，代表hello消息发出的时间

hello\_interval：32位的无符号的数，代表两次发送hello消息的时间间隔

ext\_neighbor：in\_addr类型，代表ext的邻居节点的地址

hello\_dest：in\_addr类型，代表hello消息的目的地址

178：获取当前时间

194-233：构成switch语句，根据ext->type检查两次hello消息的时间间隔

235-237：如果成功接收目的端发来的消息，则调用debug函数输出

241-244：只有当邻居节点收到三个连续的hello消息时才有效，根据此设置state的值为“INVALID”或“VALID。即：如果receive\_n\_hellos的值为0，则将state置为INVALID

否则，便将state置为VALID

246：计算出hello消息超时的时长

248-254：如果rt的值为0，这说明路由表中没有活动或过期的路由。 所以我们添加一个

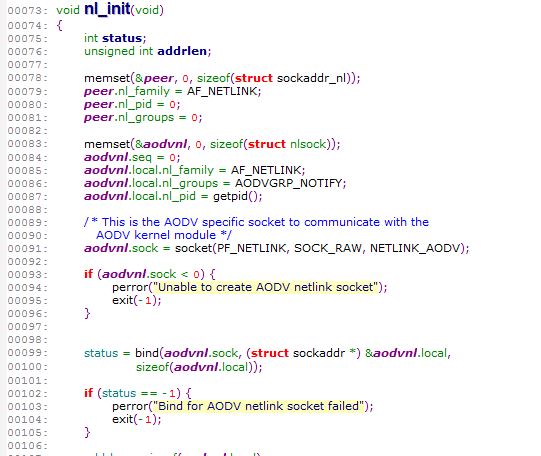
新条目

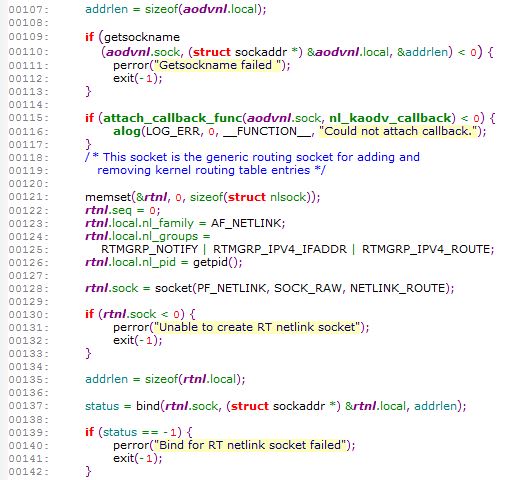
255-258：如果flags 和 RT\_UNIDIR值均为1，则说明新的邻居已经添加成功，否则便只是找到了新的邻居，但是却没有添加成功

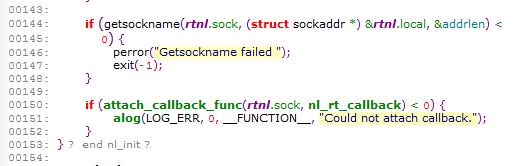
265-280 当路由表中已有活跃的路由表项时，那么这条路的生命周期应该增加，至少为：ALLOWED\_HELLO\_LOSS\*hello\_interval。这条通往临近结点的路由，必须包含hello消息中最新的目的序列号。rt\_table\_update()函数完成了此功能

## Nl.c

### Nl\_init函数-初始化函数







75-87：将aodvnl，peer分别进行初始化

91：创建一个原始套接口，这是AODV特定的用于与AODV内核模块通信的套接口，socket函数第一参数为PF\_NETLINK，表明套接字遵循协议族NETLINK。第二参数SOCK\_RAW，表明采用原始套接字。第三参数NETLINK\_AODV，表明采用的协议为AODV协议

93-96：如果创建失败，则输出Unable to create AODV netlink socket，并返回-1

99-100：把aodvnl的本地协议地址赋予state

102-105：如果出错则输出Bind for AODV netlink socket failed，并返回-1

109-113：返回aodvnl.sock套接字的IP地址，如果返回失败则返回-1退出

122-126：对rtnl进行初始化并赋值

128：将rtnl.sock创建为一个原始套接口，这个套接口通用路由套接口，是用于添加和的删除内核路由表项

130-133：如果这个套接口创建失败，则输出nable to create RT netlink socket并返回-1退出

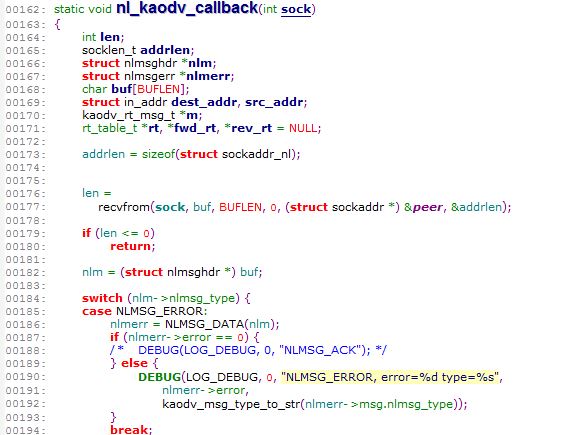
137：把rtnl.sock的本地协议地址赋予state

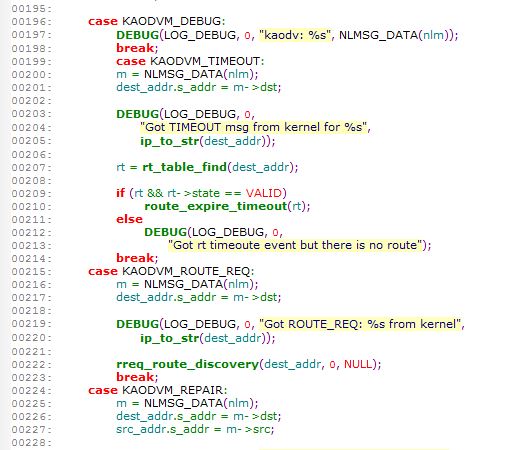
139-142：如果赋予协议地址出错，则输出Bind for RT netlink socket failed并返回-1退出

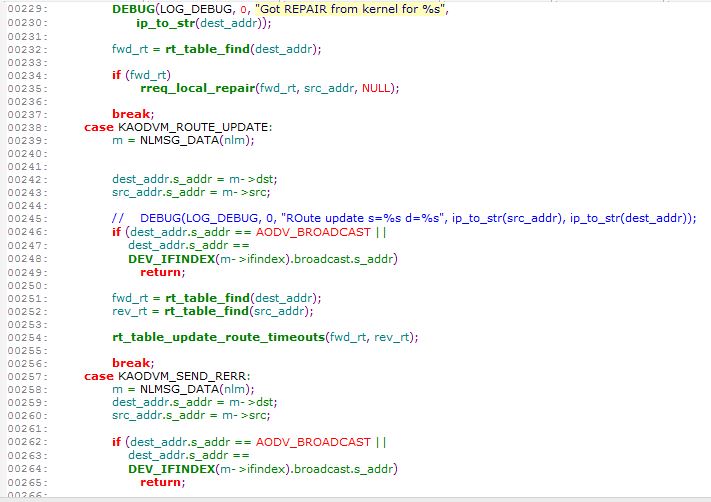
144-148：如果获取rtnl.sock的ip地址失败，则输出Getsockname failed 并返回-1退出

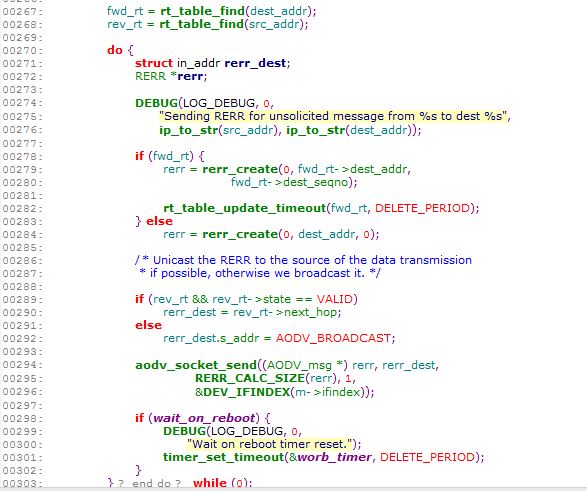
150-153：如果向rtnl.sock添加一个callback失败，则输出Could not attach callback.

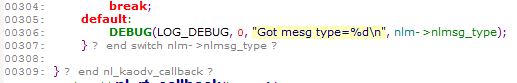
### Nl\_kaodv\_callback函数-根据内核套接字中收到的不同命令，执行不同的回调函数











164-171：addrlen：socklen\_t类型

nlm：nlmsghdr类型的指针

nlmerr：nlmsgerr类型的指针

dest\_addr，src\_addr：in\_addr类型的指针，分别表示目的地址

m：kaodv\_rt\_msg\_t类型的指针

rt, fwd\_rt, rev\_rt：rt\_table\_t类型的指针

179-180：如果读取失败，便返回空

184：根据内核套接字nlm接受到的不同命令类型选择不同的处理方式

185-197：如果nlm的类型为NLMSG\_ERROR，且nlmerr->error值不为0，则输出NLMSG\_ERROR, error=%d type=%s，

238-256：如果类型为 KAODVM\_ROUTE\_UPDATE，如果要广播目的地址，那么便返回

251：调用rt\_table\_find(dest\_addr)函数找到dest\_addr的路由

252：本地主机更新自己的路由表

257：如果类型为KAODVM\_SEND\_RERR

262-268：如果目的地址是广播地址，那么便返回空，否则便找到目的地址并更新本地主机的路由表

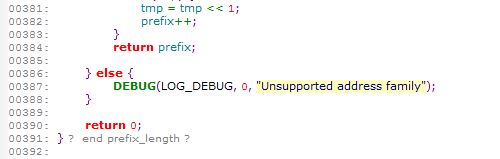
270-287：进入循环，首先输出Sending RERR for unsolicited message from %s to dest %s，如果fwd\_rt的值为1，则创建一个rerr 的消息类型，并且更新fwd\_rt，否则便创建一个将序列号置为0的rerr类型的消息

289-303：如果rev\_rt类型的状态是VALID，那么便将这个rev\_rt的目的地址指向下一跳，否则便广播目的地址，也就是说如果可能我们将单播RERR从源地址的数据传输，否则我们蒋广波它

### Nl\_rt\_callback函数-根据路由套接字中收到的不同命令，执行不同的回调函数







312-318：addrlen：socklen\_t类型，用于表示地址长度

nlm：nlmsghdr类型的指针

nlmerr：nlmsgerr类型的指针

ifm：ifaddrmsg类型的指针

rta：rtattr类型的指针

322-326：如果recvfrom调用失败，则返回空

330：根据nlm->nlmsg\_type即路由套接字收到的不同命令选择分支语句

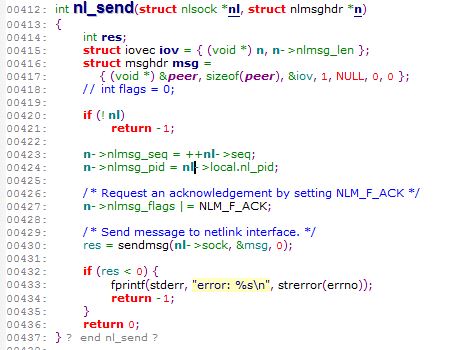
331-329：如果值为NLMSG\_ERROR且nlmerr->error的值不为0，则输出NLMSG\_ERROR, error=%d type=%d。

340-342如果值为RTM\_NEWLINK，则输出RTM\_NEWADDR

343-364如果值为RTM\_NEWADDR，则首先为ifm、rta、attrlen赋初值。在rta队列中寻找谁的rta\_type的值为IFA\_ADDRESS，则将RTA\_PAYLOAD(rta)长度的RTA\_DATA(rta)复制到&ifaddr，然后输出

365-368：如果值的类型为：RTM\_NEWROUTE，便直接退出函数

### Nl\_send函数-发送数据包给内核模块



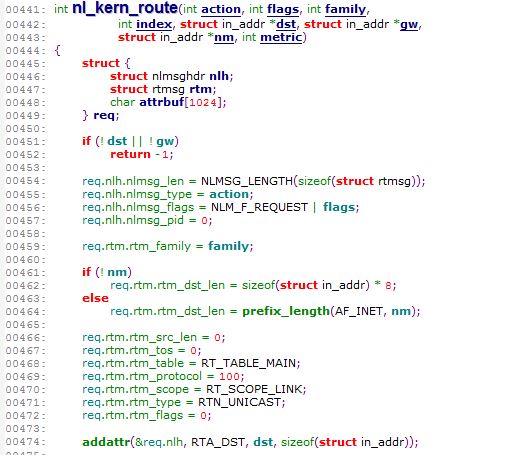
420-424：如果nl指针不为空，那么便对指针n进行赋值，否则便返回-1

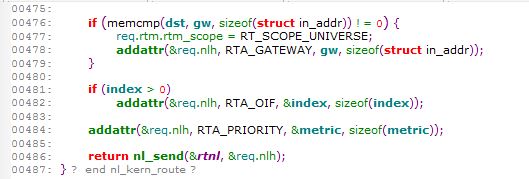
427：通过设置NLM\_F\_ACK来请求确认

430：发送消息到netlink接口，res为负，表明函数调用失败

432-435：如果函数sendmsg（）调用失败，便打印出error，退出函数，返回-1

### Nl\_kern\_route函数-操作内核路由表，包括增加、修改、删除路由表条目等操作





445-449：定义结构体req

451：如果dst和gw中有一个值为0，则退出函数，返回-1

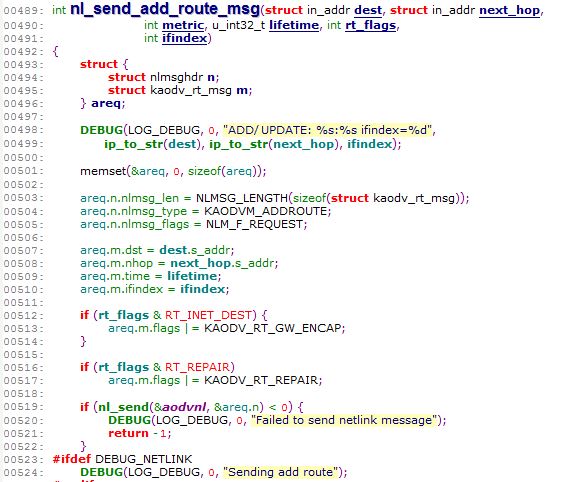
454-459：对req的成员变量进行赋值

461-462：如果nm的值为0，则req.rtm.rtm\_dst\_len便重新赋值，并再添加一个RTA\_GATEWAY类型的路由表项

481-483：如果index的值大于0便添加一个类型为RTA\_OIF的attribute

484：添加一个类型为RTA\_PRIORITY的attributer

### Nl\_kern\_route函数-向内核发送一个添加一个路由信息的消息，包括目的地地址、下一跳，生存期，路由状态标志，网络接口等



addroute02

492-496：定义areq的结构体

501：清空areq结构体

503-510：为areq的结构体进行赋值

512-513：如果rt\_flags 和RT\_INET\_DEST值均为1，则对areq.m.flags赋值为 KAODV\_RT\_GW\_ENCAP

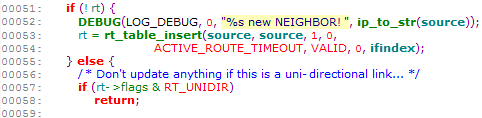
516-517：如果rt\_flags 和RT\_REPAIR值均为1，则对areq.m.flags赋值为 KAODV\_RT\_REPAIR

519-521：如果nl发送失败，则打印消息后退出函数返回-1

* 1. Aodv\_neighbor.c

**3.10.1更新活跃的邻居节点（更新来自非HELLO AODV控制消息的邻居）（void NS\_CLASS neighbor\_add）**

 代码如下：

 043-045 行代码定义了变量类型，now的作用是记录当前时间； rt作用是用作路由表（初始值设置为NULL）；seqno作用是序列号（初始值设置为0）。

051-055行代码判断路由表上是否存在sourse指向的目的节点。若不存在，利用rt\_table\_insert()函数传入参数、添加路由信息。

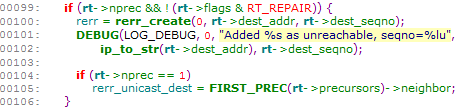
 056-059 行代码判断如果为单向链路直接返回，不用更新路由信息。

067-068 行代码表示如果路由信息是存在且活跃的，则需要用ALLOWED\_HELLO\_LOSS \* HELLO\_INTERVAL来添加对应路由的活跃周期。

### 3.10.2邻居节点链路中断情况（void NS\_CLASS neighbor\_link\_break）

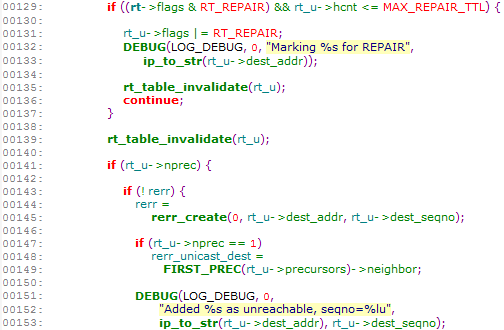
084-086 行代码判断路由表是否为空若为空则直接返回。

087-092 行代码判断若rt->hcnt跳数不为1时，则不是直接邻居节点。将目的地址和跳数传入函数并返回。

 096 行代码若判断函数为跳数为1的直接邻居节点，输出链路断开信息并将目的地址传入。同时将对应的路由表置为无效化。

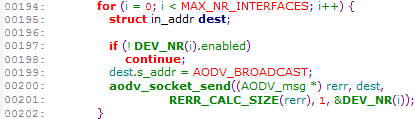
099-106行代码，除去路由被回复的情况，创建一个路由错误信息rerr。调用rerr\_create()函数来传入目的地址和序列号。

109-110 行代码用来清除了路由前驱列表。

129-152 行代码用来检测路由表的下一跳条目中知否存在不可达的目标。如果条目是不可达的，应包含在RERR中。

131-137 行代码，将损失的链接标记为要修复。

139-153 行代码将损失的链接标记为要修复的情况下，在对其他所有不可达条目应执行相同的操作。

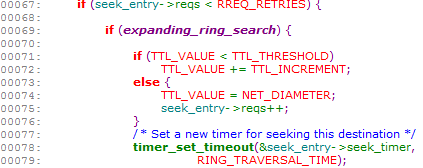
194-202 行代码表示只在不可达路由结点的先驱结点的接口上传输RERR。

## 3.11 Aodv\_timeout.c

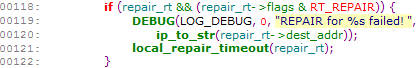
### 3.11.1定义定时器到期时调用的超期函数

059-061 行代码检查完整性。

063-065 行代码记录当前时间并写在日志中。

067-079 行代码当不满足超时的条件时，设置一个新的计时器来寻找目的地址。

088-090 行代码AODV应该使用二进制指数来退避RREP等待时间。

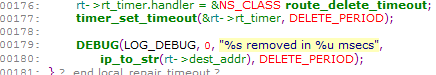


118-122 行代码保证了当路径被修复后，节点返回超时信号。

### 3.11.2.定义本地修复超时处理函数

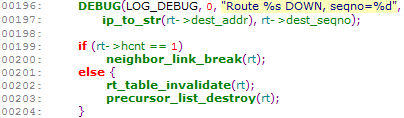


137-140 行代码默认了目的地，并清除RERAIR标志。



176-181 行代码表当修复超时，清除可能在排队中的数据包，并记录到日志中。

### 3.11.3.设置路由过期超时函数



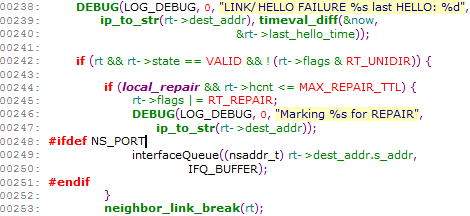
196-204 行代码当路由消息长时间没有收到对应的HELLO()函数消息时，设置这条路由超时，将其断开，并从路由表里删除。

### 3.11.4.路由删除函数设置



214-217行代码进行路由表完整性检测。从路由表中删除路由信息，并记录在日志中。

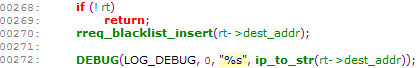
### 3.11.5.HELLO消息超时函数



238-253 行代码表示当我们停止接收来自邻居节点的HELLO()信息时，这个函数就会被调用。Hello消息超时函数与路由超时功能基本一致。当我们在修理路线时，将其标记（标志位设置为REPAIR）进行维修.

缓冲区内容来自接口队列的数据包。

### 3.11.6响应超时函数



268-272 行代码，当RREP传输失败时（即缺少RREP\_ACK）时，添加到黑名单设置中。