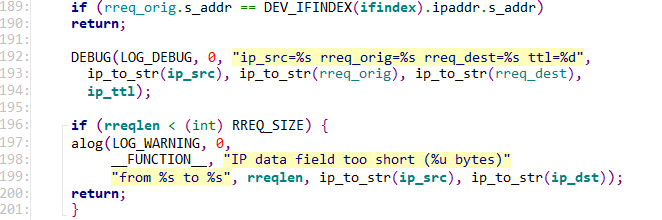
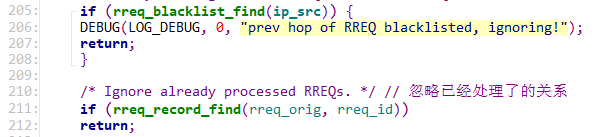
# aodv\_rreq.c



函数创建RREQ，初始化信息的类型，设置该信息的地址、序列号等信息。

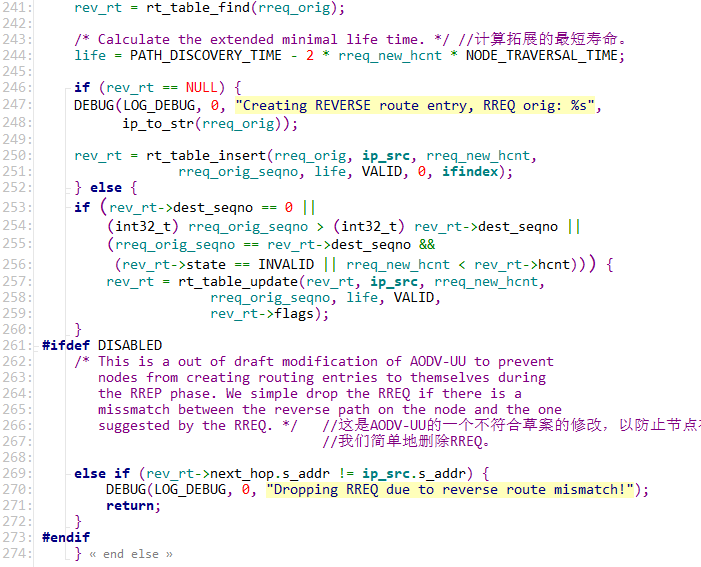


189~194：检查接收到RREQ是否为自身节点发出，如果是，则忽略，如果不是，接收信息。



205~208：检测RREQ的消息是在黑名单中，如果在，则忽略。

210~212：检测该消息是否已经被处理。



241：判断是否存在反向路由表项。

246~248：若不存在反向路由表项，则新建反向路由。若存在反向路由表项，判断是否需要更新。

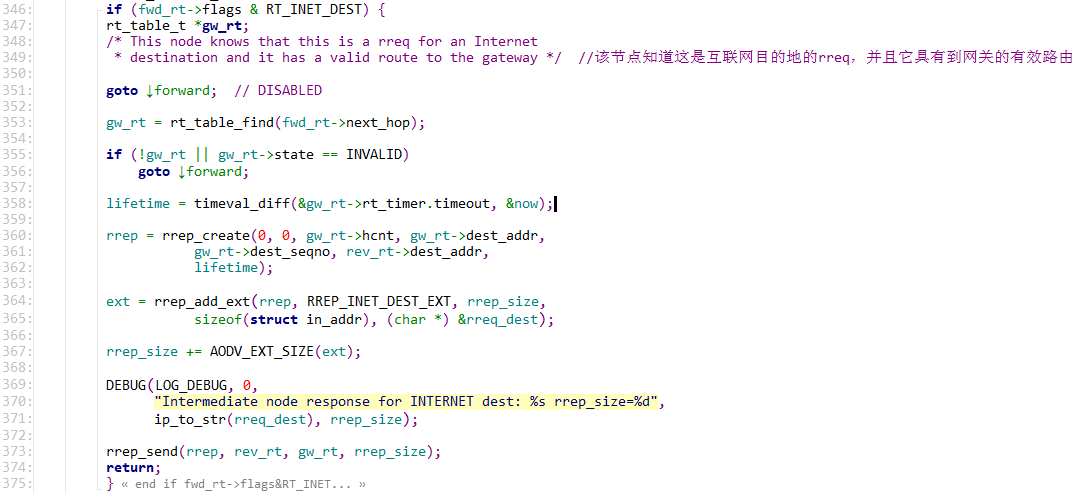
252~260：路由判断，当遇到新的RREQ时，比较在路由表中已经存在的到目的节点的路由，比较目的节点的序列号，在目的节点序列号相同的情况下，采用跳数小的。



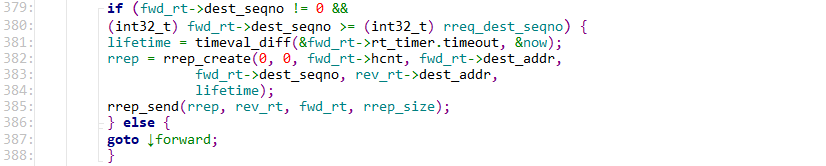
313：当不知道目的节点的位置时，判断自身是不是RREQ的目的节点。

318~329：当节点自身是RREQ的目的节点，将系欸但自身的序列号进行更新，将其更新为当前序列号和RREQ中最大的序列号，并创建新的到该节点的RREQ路径，并开始往回发送RREP消息。

330~338：若不是目的节点，则检查该节点所有的路由条目，查看是否有到达目的节点的路由条目。



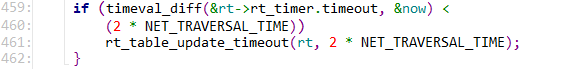
346~375：判断当前节点是否有到目的节点的有效路由，若是则回复RREQ消息，倘若没有则继续广播RREQ消息。



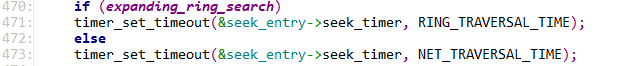
379-388：判断接收到的RREQ消息的序列号是不是足够的新，若不是说明是过时的消息，则不用转发。



437~439：如果当前节点的路由表中已经包含了目的节点的路由表项，则讲ttl设置为NET\_DIAMETER。



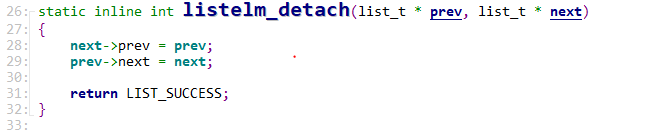
459~460：等待RREP的路由表条目不应该在2\*NET\_TRAVERSAL\_TIME遍历时间之内被删除，若时间小于2\*NET\_TRAVERSAL\_TIME，更新它的计时器，防止被删除。



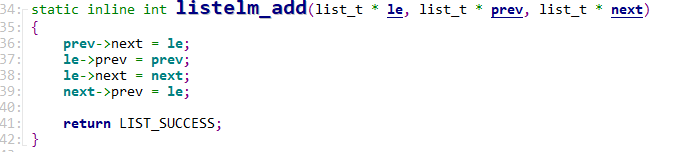
470~473：为RREQ消息设置定时器。

# List.c

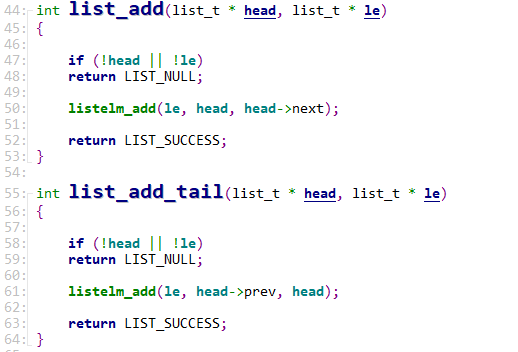
List.c中主要包含了一些链表的操作。



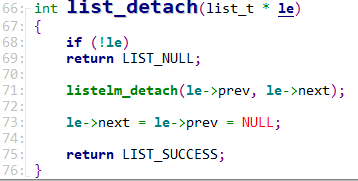
26~34：这个函数进行的是链表中间节点删除的操作。



34~42：这个函数所进行的是在链表中间入新节点的操作。



44~64：这两个函数分别实现类的在链表头和链表尾插入新的节点。

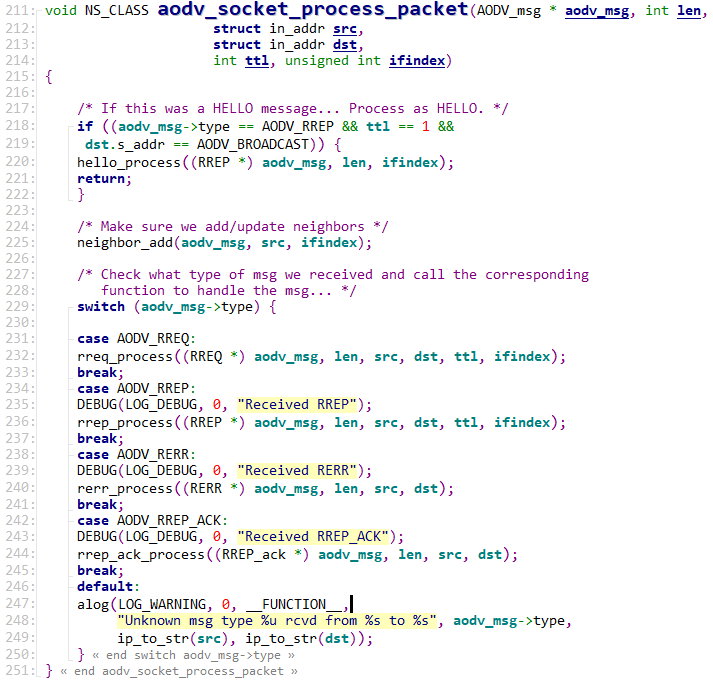


66~76：将链表中的某一指定节点删除。

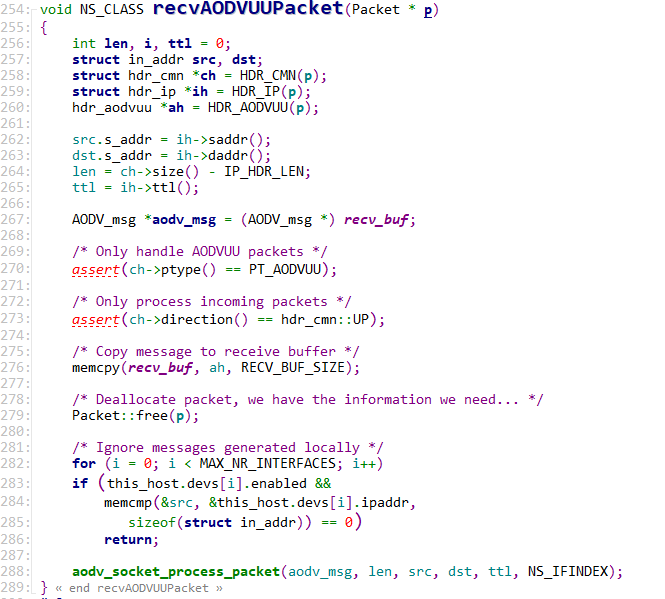
# Llf.c

连路层反馈

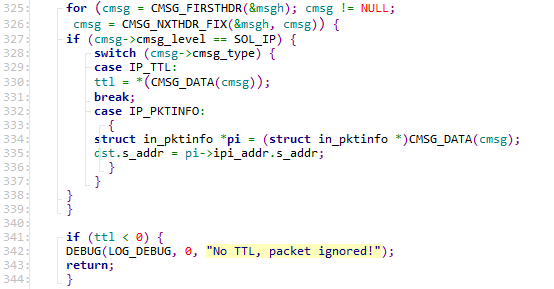
# Aodv\_socket.c



211~251：为函数aodv\_socket\_process\_packet,该函数对所接收到的信息做出判断，判断是属于哪一类的信息，并根据不同的信息做出不一样的处理。

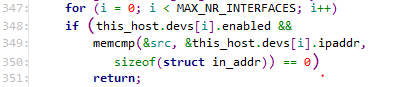


254~289：对接收到的数据包做处理。在确定这是aodv的数据包并且是从外部接受到的数据包时，复制该数据包里的消息到缓冲区，在获取完所需要的数据后，将数据包分解、释放。如果是自身产生的数据包，就将其忽略掉。最后在其做完处理判断后，调用aodv\_socket\_process\_packet对数据包的类型做出相应判断并做相应处理。

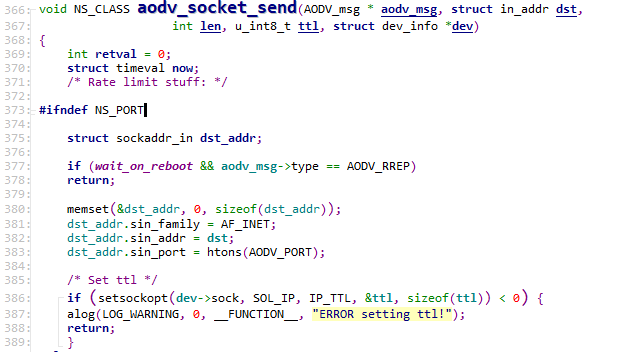


325~339：这段代码在函数aodv\_socket\_read中，这段代码的的作用是从读到的消息中获取ttl（跳数）以及目的地址的信息。

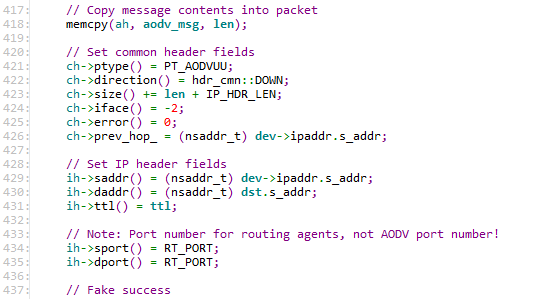
341~344：根据读到的ttl做判断，如果ttl小于0，则忽略该条消息。



347~351：对消息做出判断，检查这条消息是不是自身发出的，如果是自身发出的消息，则将其忽略。

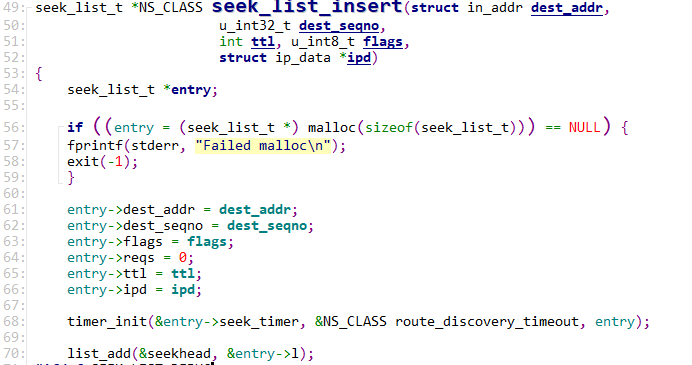


366~389：在函数aodv\_socket\_send中，先判断要发送的信息的类型，先判断信息是否在等待被发送以及信息的类型，若信息不是RREP，则设置该信息的基础信息，并设置ttl。

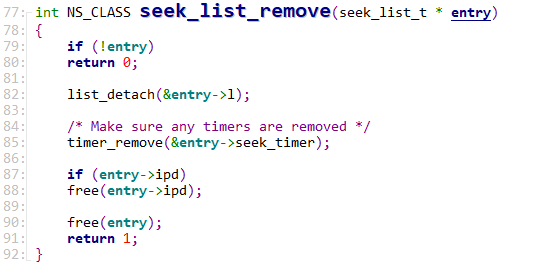


417~437：设置其他首部信息，并设置接收方的端口号。

# Seek\_list.c



49~70：函数seek\_list\_insert完成的操作是在搜索裂变中加入新的节点。先开辟一个新的节点空间，将地址、序列号，ttl，ip等信息存放进去，并设置计时器，最后将其插入到链表中。



77~92：函数seek\_list\_find完成的操作是将某一结点从搜索链表中删除。先判断该节点是否存在，当节点存在时，删除该节点，并将该节点所占用的空间释放。

# Locality.c