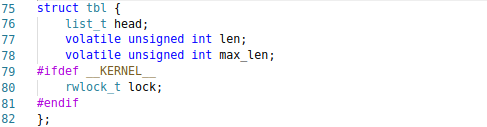
1. **tbl结构体**

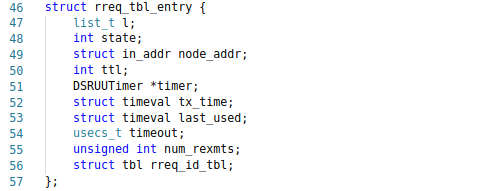
tbl结构体由list\_t封装而来并且定义了程序锁避免不同进程之间读取时出错。并且定义了当前长度和最大长度，在list\_t的基础上，增加了list\_t的安全性。



tbl.h

此种类型定义的变量有rreq\_tbl，grat\_rrep\_tbl，send\_buf，neigh\_tbl，maint\_buf

1. **rreq\_tbl\_entry结构体**

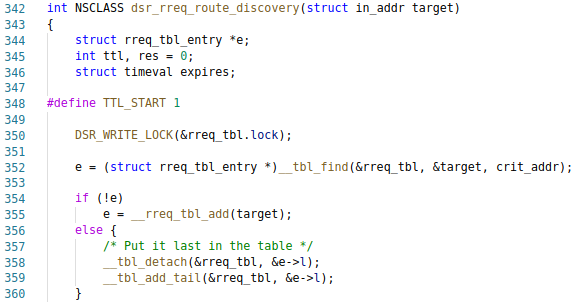


dsr-rreq.c

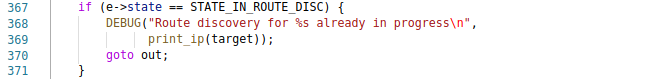
在该结构体中定义了状态，时间等一系列参数。这个结构体的功能包括判断超时退出。定义了ttl表示当前的跳数，定义了节点的地址等。

1. **dsr\_rreq\_route\_discovery函数**

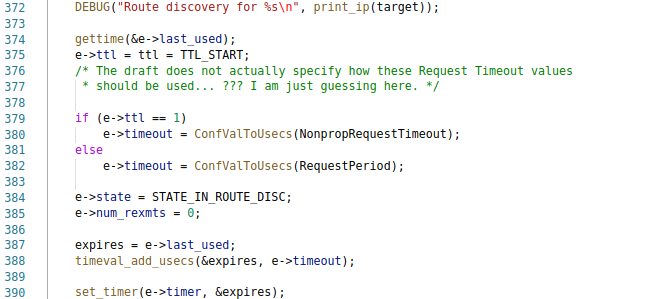
e是一个rreq\_tbl\_entry类型的指针。函数第11行，使用\_\_tbl\_find函数，对rreq\_tbl进行搜索。如果列表中存在有该过程，则将之移动至末尾。



如果同时出现两个发现同一节点的路由发现(discovery)第一个影响效率第二个有可能会引发错误。在第367行进行判断满足条件则跳转至末尾。



此外，在准备过程中，函数初始化时间，ttl等值。



准备完成后，通过掉用相应函数，在网络启动discovery



dsr-rreq.c

1. **dsr\_opt\_recv函数**

调用该函数，所有的数据会接收到参数传入的指针所指向的数据包之中，并进行安全性验证。



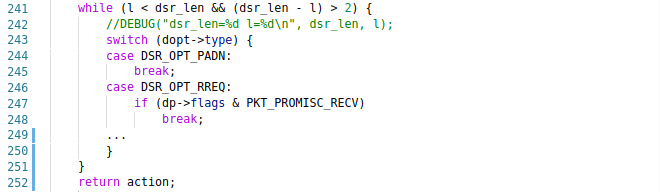
dsr-opt.c

action是一个临时标记，可以用来存放当前进行操作的类型。如果该数据包目的地是本节点并且负载长度不为0，则将action设置为DSR\_PKT\_DELIVER宏值。



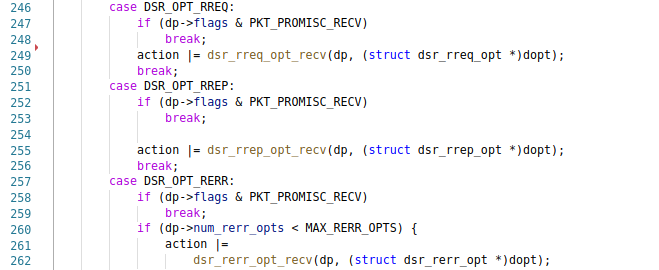
dsr-opt.c

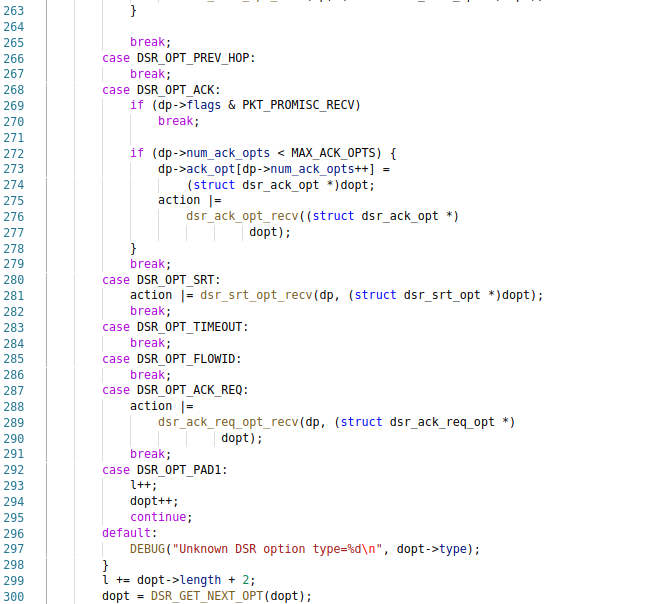
通过DSR\_GET\_OPT宏可以取出dp数据包中的数据包类型。用switch 来对不同数据包进行不同的接收操作。执行结果既返回了action，并且将数据包接收至dp指针处。



dsr-opt.c

不同数据包对应于不同种类的接收方式,对应返回不同的action值。





dsr-opt.c

其中dsr\_srt\_opt\_recv能够根据接收的内容，回复对应的reply(dsr\_rrep)而不用再回到dsr\_opt\_recv函数之中执行。类似的dsr\_ack\_req\_opt\_recv会自动回复dsr\_ack。

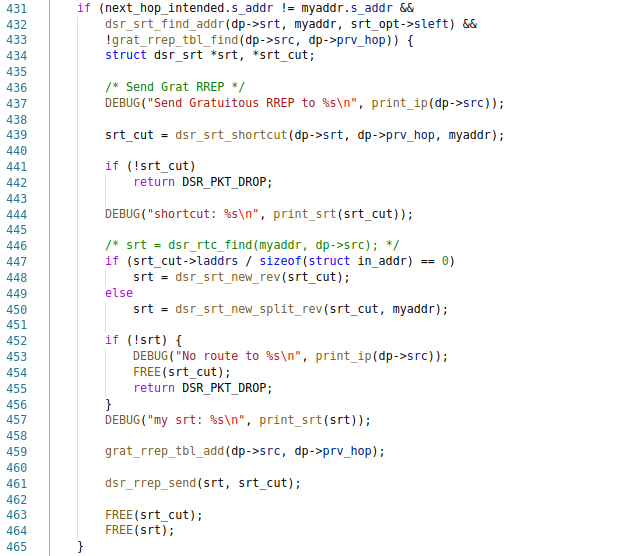
1. **dsr\_srt\_opt\_recv函数**

当dsr\_opt\_recv函数判断当前操作为DSR\_OPT\_SRT时，该函数既被调用。本函数调用的dsr\_srt\_new将源路由信息加入到cache之中。



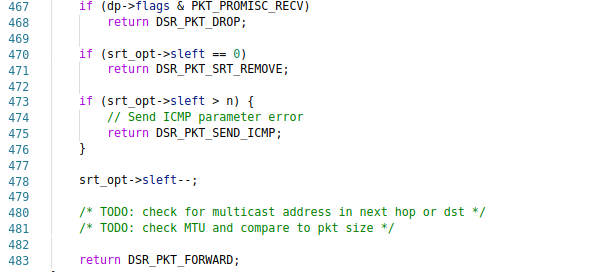
dsr-srt.c

然后进行一系列操作包括地址比对，检索列表等。寻找到需要进行路由缩短的位置。



dsr-srt.c

经过前系的条件判断，包中的源路由路径可以用来优化目前。经调用dsr\_srt\_shortcut函数，从dp->srt中得到一条从dp->srt到dp->prv\_hop的路由。使用grat\_rrep\_tbl\_add，进行路由reply消息的发送，通知前列节点路径信息已经被缩短。本地不会缓存缩短后的路径信息，路由信息由前置节点存储。

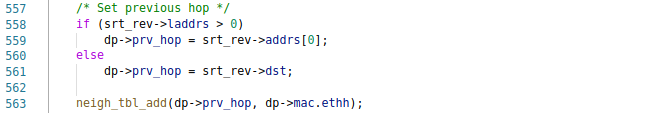


完成操作后，设定一系列规则返回不同值。代表上层函数会执行不同的操作。

1. **dsr\_rreq\_opt\_recv函数**

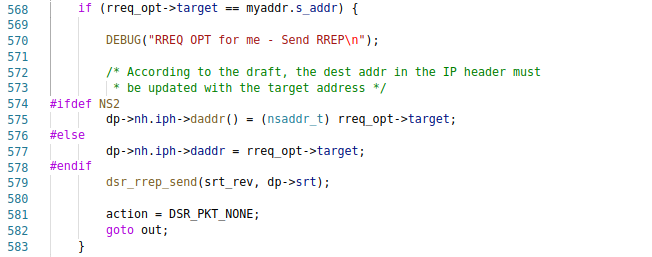
在接收过后将数据包传入该函数之中。避开所有的判断安全性和初始化的部分，对代码进行解析。首先添加该节点至rreq\_tbl之中。rreq\_tbl\_add\_id会将新添加的节点放在列表最末尾。



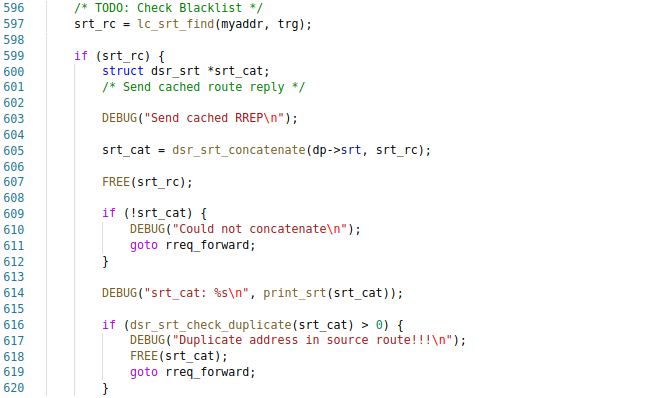


dsr-rreq.c

srt\_rev中，存有本身生成的反向路由。通过反向的路径信息回传rrep消息。当消息回复以后，立即跳转结束函数的执行。

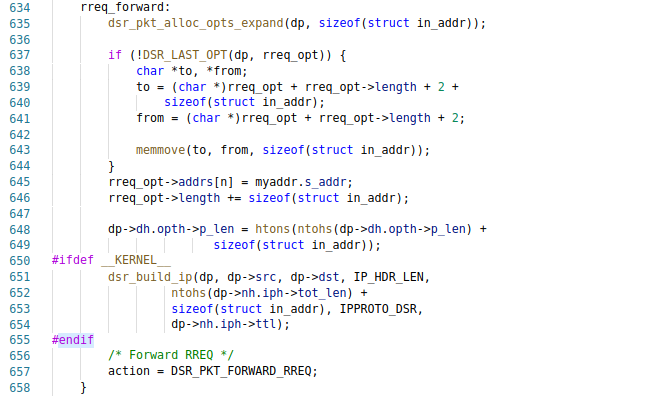


如果函数未跳转，程序会继续执行。dsr\_srt\_concatenate函数可以将路由串联起来，将本地路由连接到原路由之后。



dsr-rreq.c

在条件满足时，调用dsr\_srt\_concatenate函数处理路径信息。若条件不满足（该节点位于blacklist之中），则跳转到标签处开始执行。

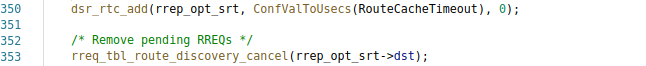


dsr-rreq.c

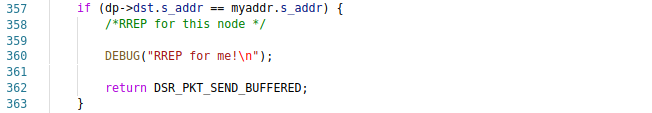
经过该部分action被赋值为DSR\_PKT\_FORWARD\_RREQ并将于函数最末尾返回。

1. **dsr\_rrep\_opt\_recv函数**

rrep的职责包括对RREP消息的转发与处理。首先进行路由发现关闭过程并且将搜索到的路径信息缓存。表示一次发现过程已经完成。



然后进行地址比对。



dsr-rrep.c

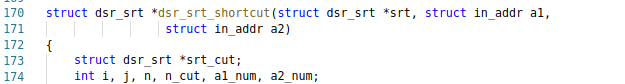
如果目的地址是本机地址，action返回DSR\_PKT\_SEND\_BUFFERED并且结束程序。如果不是则返回DSR\_PKT\_FORWARD交由上一层处理。

1. **dsr\_rerr\_opt\_recv函数**

该函数的主要作用是传播路由错误信息。通过调用maint\_buf\_del\_all函数并且触发maint\_buf\_set\_timeout函数，再由maint\_buf\_set\_timeout函数调用dsr\_rerr\_send，对错误进行传输。所有节点收到数据包后，立刻将路由信息从自己的maint-buf中删除，并且帮助进行扩散。

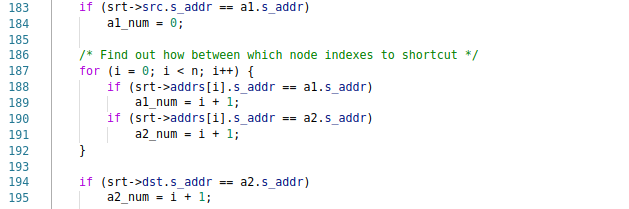
1. **dsr\_srt\_shortcut函数**

函数传入了一个源路由信息和两个地址信息。并定义了一系列临时变量。



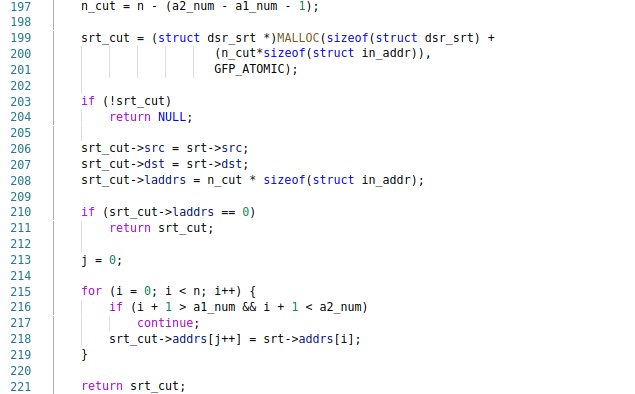
dsr-srt.c

然后寻找a1,a2节点在地址中对应的位置。

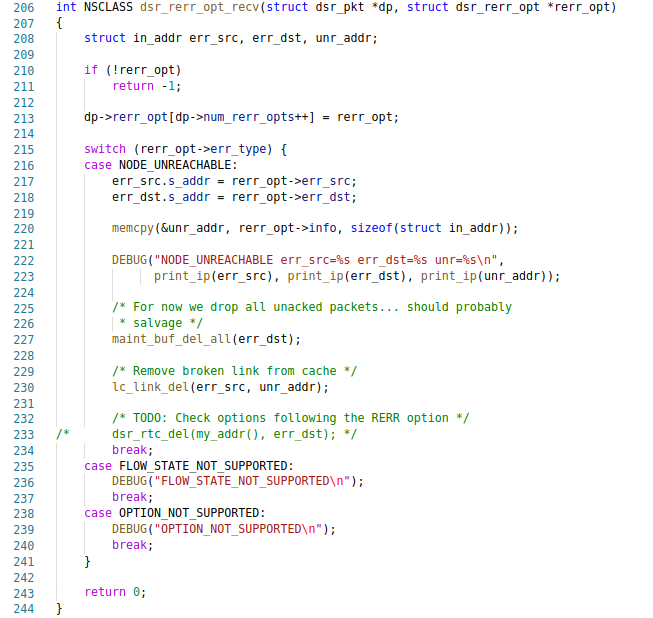


dsr-srt.c

最后生成两点之间的源路由并返回。



dsr-srt.c



dsr-rerr.c

1. **dsr\_rerr\_send函数**

设定目的地和事故路段后，调用传输函数将数据包传送至目的地。根据原理进行操作所以略去一部分代码选择功能最核心的部分后得到以下代码段。







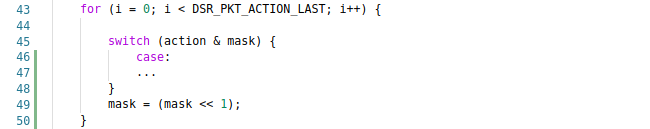
dsr-rerr.c

篇幅有限类似函数及其具体功能，不再通过细节展开。将采用列表的形式在之后的一章中呈现。

1. **dsr\_recv函数**

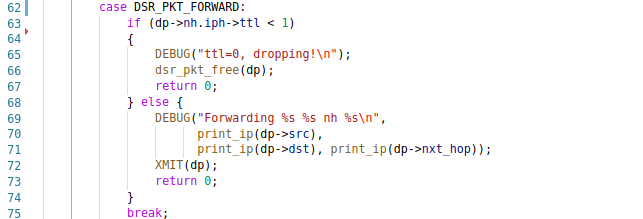
在调用dsr\_opt\_recv之后，dsr\_recv得到了一个完整的数据包（dp）。通过对action的区分，可以对不同的数据包进行不同的处理。dsr\_recv间接调用了以上所有的接收函数，是通过调用dsr\_opt\_recv函数来实现





dsr-io.c

如果收到数据包，并且通过action隐含的信息判断是否需要将该数据包转发。



dsr\_recv函数收到DSR\_PKT\_FORWARD\_RREQ消息之后，会直接将数据包传送并且立即结束运行。



dsr-io.c

基本函数清单及其功能

为避免篇幅过长，这部分内容使用列表将函数名，函数对应的文件和功能总结列于表格中。主要包括一些具体的数据操作，如向列表中添加、删除元素等。其中的各种函数并不独立，可以存在一定的相互调用关系。

|  |  |  |  |
| --- | --- | --- | --- |
| 核心功能 | 函数名 | 功能（包括但不限于） | 声明位置 |
|  | dsr\_recv | 接收处理函数 | dsr-io.h |
| dsr\_opt\_recv | dsr-opt.h |
| dsr\_srt\_opt\_recv | dsr-srt.h |
| dsr\_rerr\_opt\_recv | dsr-rerr.h |
| dsr\_rreq\_opt\_recv | dsr-rreq.h |
| dsr\_rrep\_opt\_recv | dsr-rrep.h |
| dsr\_srt\_shortcut | 从srt中得到短路径信息 | dsr-srt.h |
| dsr\_rerr\_send | 发送路由错误 | dsr-rerr.h |
| dsr\_rrep\_send | 发送路由reply | dsr-rrep.h |
| dsr\_rreq\_send | 再网络中启动路由发现 | dsr-rreq.h |
| dsr\_ack\_req\_send | ack请求消息 | dsr-ack.h |
| dsr\_ack\_send | 发送ack | dsr-ack.h |

以上函数和一些其他的函数存在一定的调用关系，这些细节操作可以单独列出来作为一个补充。

|  |  |  |  |
| --- | --- | --- | --- |
| 其他 | 函数名 | 功能（包括但不限于） | 声明位置 |
|  | dsr\_pkt\_alloc\_opts | 动态内存分配pkt\_opt | dsr-pkt.h |
| dsr\_pkt\_alloc\_opts\_expand | 扩展pkt\_opt |
| dsr\_pkt\_free\_opts | 释放pkt\_opt |
| dsr\_pkt\_alloc | 动态内存分配pkt |
| dsr\_pkt\_free | 释放pkt |
| dsr\_opt\_hdr\_add | 将缓冲区转换为dsr\_opt\_hdr | dsr-opt.h |
| dsr\_build\_ip | 构建iphdr |
| dsr\_opt\_find\_opt | 传入type，通过type寻找对应的opt |
| dsr\_opt\_remove | 删除pkt的opt |
| dsr\_opt\_parse | 通过一定规则解析opt |
| dsr\_rerr\_opt\_add | 创建一个dsr\_rerr\_opt用来发送路由错误 | dsr-rerr.h |
| crit\_query | 比对内存 | dsr-rrep.h |
| crit\_time |
| crit\_addr | dsr-rreq.h |
| crit\_duplicate |
| grat\_rrep\_tbl\_init | rrep\_tbl基本操作 | dsr-rrep.h |
| grat\_rrep\_tbl\_cleanup |
| grat\_rrep\_tbl\_add |
| grat\_rrep\_tbl\_find |
| grat\_rrep\_tbl\_print |
| grat\_rrep\_tbl\_proc\_info |
| grat\_rrep\_tbl\_timeout |
| dsr\_rrep\_add\_srt | 添加source route | dsr-rrep.h |
| \_\_rreq\_tbl\_entry\_create | 创建rreq\_tbl基本单元 | dsr-rreq.h |
| \_\_rreq\_tbl\_add | 添加元素至rreq\_tbl\_add |
| rreq\_tbl\_timeout | 检测超时并进行相应操作 |
| rreq\_tbl\_add\_id | 使用id信息构造rreq\_tbl基本单元并添加 |
| dsr\_rreq\_route\_discovery | 开启一次路由发现 |
| rreq\_tbl\_route\_discovery\_cancel | 关闭特定的路由发现 |
| rreq\_tbl\_set\_max\_len | 得到最大长度 |
| dsr\_rreq\_duplicate | 检测重复，有则返回1，没有返回0 |
| dsr\_srt\_next\_hop | 寻找下一跳地址 | dsr-srt.h |
| dsr\_srt\_prev\_hop | 寻找上一跳地址 |
| dsr\_srt\_find\_addr | 是否包含对应地址有返回1没有返回0 |
| dsr\_srt\_new | 使用一定信息创建一个dsr\_srt |
| dsr\_srt\_new\_rev | 复制一个dsr\_srt |
| dsr\_srt\_new\_split | 返回指定地址之前的路径地址 |
| dsr\_srt\_new\_split\_rev | 从接收的dsr\_srt中分出地址 |
| dsr\_srt\_check\_duplicate | 查重 |
| tbl\_empty | tbl结构体基本操作 | tbl.h |
| \_\_tbl\_add |
| \_\_tbl\_add\_tail |
| tbl\_add\_tail |
| \_\_tbl\_find |
| \_\_tbl\_detach |
| \_\_tbl\_del |
| \_\_tbl\_find\_do |
| tbl\_find\_do |
| \_\_tbl\_do\_for\_each |
| tbl\_do\_for\_each |
| tbl\_find\_detach |
| tbl\_detach |
| tbl\_detach\_first |
| tbl\_add |
| tbl\_del |
| tbl\_find\_del |
| tbl\_for\_each\_del |