

动态源路由协议代码分析

网络协议栈分析及设计大作业



2017-12-24

大连理工大学软件学院

**目录**

[1 原理分析 3](#_Toc502735214)

[1.1 概述： 3](#_Toc502735215)

[1.2 详解： 3](#_Toc502735216)

[1.2.1 路由发现详解 3](#_Toc502735217)

[1.2.2 路由维护详解 3](#_Toc502735218)

[1.2.3 实现分析 4](#_Toc502735219)

[2 过程分析 5](#_Toc502735220)

[2.1 路由发现 5](#_Toc502735221)

[2.2 路由维护 7](#_Toc502735222)

[3 代码分析 7](#_Toc502735223)

[3.1 整体结构 8](#_Toc502735224)

[3.2 首部选项 9](#_Toc502735225)

[3.2.1 固定首部 9](#_Toc502735226)

[3.2.2 路由请求选项 10](#_Toc502735227)

[3.2.3 路由回复选项 11](#_Toc502735228)

[3.2.4 路由错误选项 12](#_Toc502735229)

[3.2.5 确认请求选项 12](#_Toc502735230)

[3.2.6 源路由选项 13](#_Toc502735231)

[3.3 数据结构分析 13](#_Toc502735232)

[3.3.1 路由缓存 13](#_Toc502735233)

[3.3.2 路由请求表 14](#_Toc502735234)

[3.3.3 发送缓冲区 14](#_Toc502735235)

[3.3.4 维护缓冲区 15](#_Toc502735236)

[3.4 操作处理 15](#_Toc502735237)

[3.4.1 通用数据分组处理 15](#_Toc502735238)

[3.4.2 路由发现处理 18](#_Toc502735239)

[3.4.3 路由维护处理 22](#_Toc502735240)

# 原理分析

## 概述：

动态源路由协议（DSR）是一种专门为多跳无线网络中移动节点设计的简易高效的路由协议。它允许网络自动调整。该协议主要包括路由发现Route Discover和路由维护Route Maintenance两部分，二者都是在节点需要发送数据时才进行，所以称该协议为动态协议。与其他协议不同，DSR不需要任何其他的周期性路由通知，也不需要链路状态和邻居节点检测，所有的路由操作都是按需进行。DSR路由协议主要是为了在大约200个节点的移动自组网络而设计的，并且在高移动率的情况下表现良好。该协议运行在网络层。本文分为三部分，首先是对该协议的综述，之后使用实例来分析系统流程，最后通过代码来分析协议的实现。

## 详解：

两大机制：路由发现和路由维护。

### 路由发现详解

路由发现发生在当网络中的某个源节点S要向目的节点D发送数据并且S不知道通向D 的路由时，目的是帮助源节点获得到达目的节点的路由。

当源节点S需要向目的节点Ｄ发送数据时，它首先查询路由缓冲区是否有通向Ｄ的路由，如果有则将该路由信息附加到数据分组头部中进行发送，如果没有，则发起路由发现过程。源节点向邻居节点以洪泛方式发送路由请求，以此启动路由发现，随后每个中间节点接收后判断是否存储有到目的节点的路由，若不含有，则将自己的地址加入请求数据中的路由记录并继续转发给邻居节点（如果是目的节点D收到该请求，则会返回路由回复，路由回复中包含所请求的路径信息）。当源节点收到路由回复后，将收到的路径信息存储在自己的路由缓存中，路由发现过程结束。

在扩展的路由发现过程里，每个节点还可以从转发的数据分组头部中学习到路径信息并保存在自己的路由缓存中。同时如果中间节点在自己的路由缓存中找到了源节点请求的路由信息，也会返回一个路由回复而不是继续转发该路由请求。

### 路由维护详解

当网络的拓扑结构发生变化后，如果S通向D的路由记录不再有效，此时DSR协议就会启动路由维护，尝试使用其他路由或者通过路由发现寻找一条新的路由。

在数据进行发送的过程中，如果路由维护发现网路拓扑结构发生改变使得某条使用中的路由无效，则会发送路由错误给源节点，源节点会在收到该报文后从路由缓存中删除所有的相关错误路由信息，并重新发起路由发现。

在扩展的路由维护过程中，如果一个要转发数据分组的中间节点通过路由维护检测到该数据分组的路径下一跳已经中断时，该节点会检查其路由缓存，如果发现有另一条路由可以到达该数据分组的目的地，节点就会抢救数据分组而不是直接丢弃。同时数据分组维护一个计数值来标记被抢救次数，以避免被无限抢救。当某个源节点收到它发起的数据分组的路由错误后，它会在下一个路由请求上附加上这个错误信息传播给邻居结点。并且，如果路由中的一个或多个中间节点不再需要，则可以自动缩短使用中的源路由，即自动路线缩短。

### 实现分析

使用DSR协议的每个通信节点维护如下主要数据结构：

* 路由缓存 (Route Cache)
* 发送缓冲区 (Send Buffer)
* 路由请求表 (Route Requestt Table)
* 维护缓冲区 (Maintenance Buffer)

**路由缓存：**

路由缓存会保存该节点所存储的路由信息。每个节点维护自己的路由缓存，当节点从网络中听到新的路径信息后便会将该信息添加进缓存，同样的，当它听到某个现存路由信息失效后，也会相应地从缓存中删除该信息。

路由缓存可以存储通向同一节点的多条路径信息，节点通过搜索目的节点地址来查找相应的路径信息。每个节点会采用相应的策略来管理缓存区，比如LRU算法，同时也可以给一些路由记录设置优先级。有两种方式来实现该缓冲区——使用path cache或者link cache，前者简单，后者实现复杂但更为强大。

本文分析的源码实现方式为link cache。

**发送缓冲区：**

每个节点的发送缓冲区会保存该节点由于无法得知目的节点路径而等待发送的数据分组队列。节点在将数据分组后加入缓冲区队列的同时发起路由发现过程，找到相应的路由信息后这些数据分组就会被发送出去。同时缓冲区中每个数据分组都与自己的加入时间相关联，一定时间后如果没有被发送出去就会被清除。如果有必要的话，缓冲区会使用先进先出FIFO策略来管理这些数据分组，从而避免缓冲区溢出。

每当某个节点向自身的路由缓存表中添加新的记录时，节点自身都会检查发送缓冲区中的每个数据分组，一旦某个数据分组的目的路由信息包含在路由缓存表中（包括新添加进的记录），该节点就会立刻发送此数据分组并将其从发送缓存区中移除

**路由请求表：**

路由请求表用来记录该节点最近发出或者转发的所有路由请求，并使用Ip地址来作为索引。该请求表记录如下信息：生存周期（TTL），最近转发时间以及路由请求发起者的相关信息等。如果某个节点在某个时刻发送或转发了通往某个目的节点的路由请求，在接下来的一定时间内，该节点无法再次想这个目的节点发送路由请求。如果某条路由请求记录超时，则该记录会被删除，每个节点使用LRU策略来管理表中记录的请求。

**维护缓冲区：**

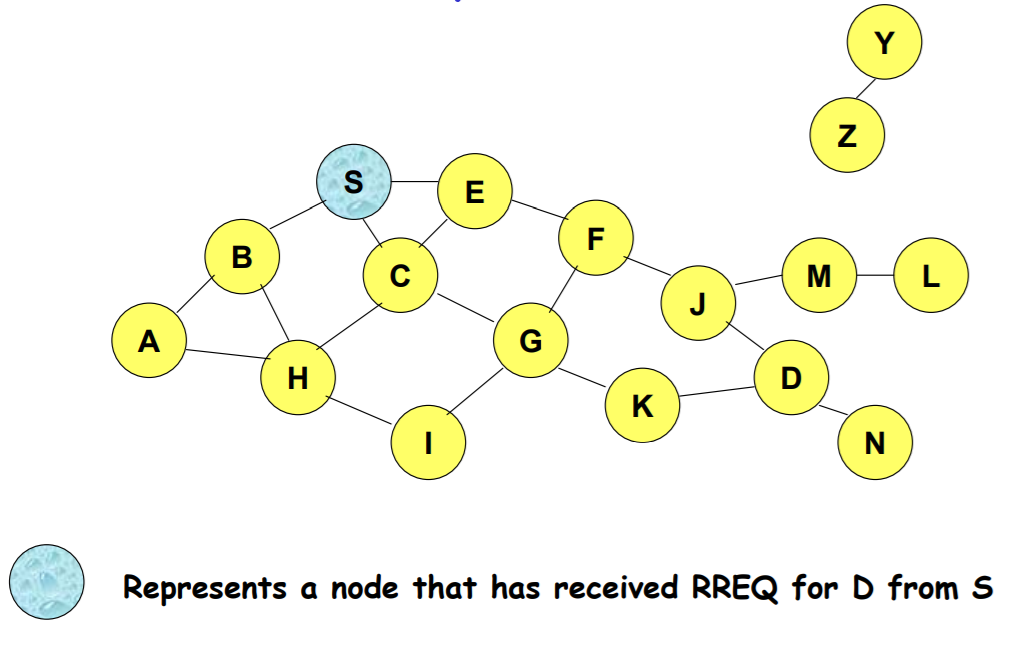
保存由该节点发出但未收到下一跳确认的数据分组。在某个节点发出或者转发数据分组之后，如果下一跳节点未成功接收或者无法确认下一跳节点已经成功接收，则需要该节点重新发送分组。每个记录标记有最大重传次数，达到最大重传次数后丢弃该数据分组，并向源节点发送错误分组。

# 过程分析

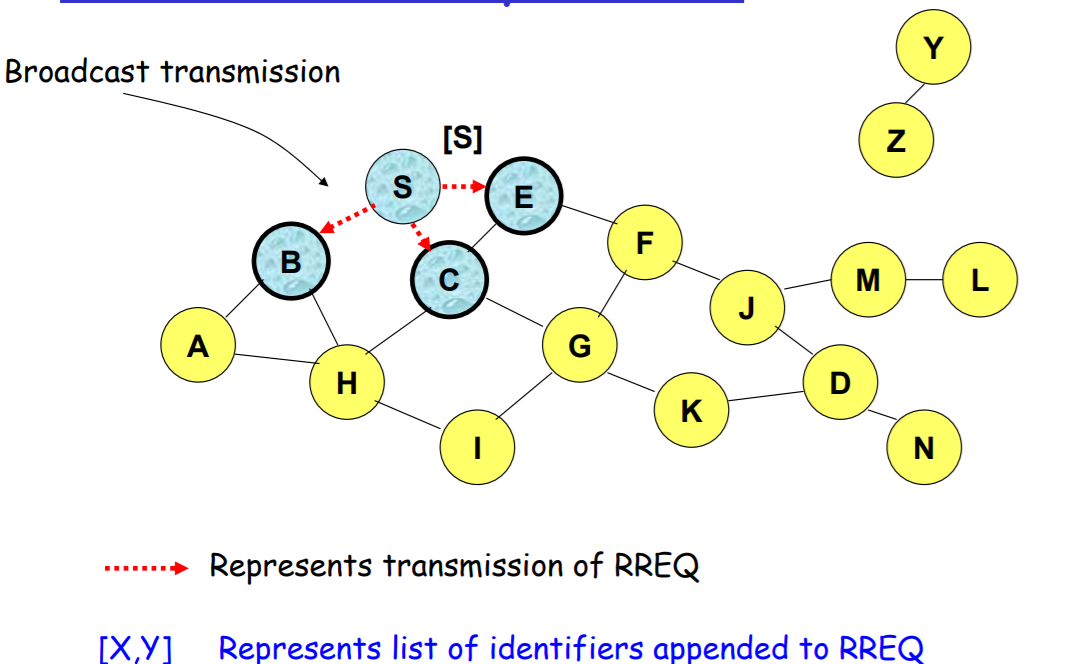
接下来通过具体实例来详细分析DSR通信过程。

## 路由发现

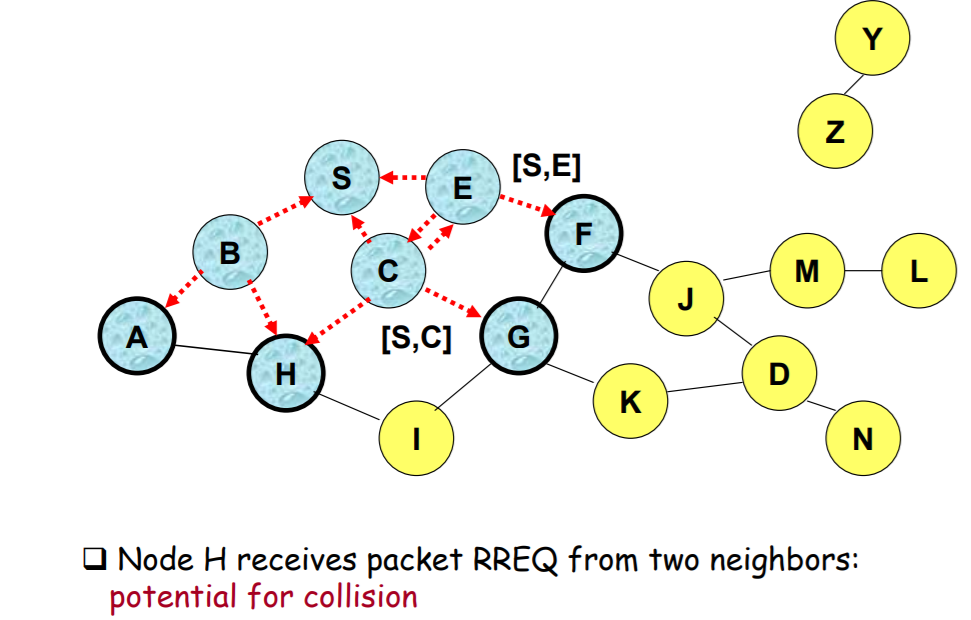
考虑在如下的网络拓扑图中：



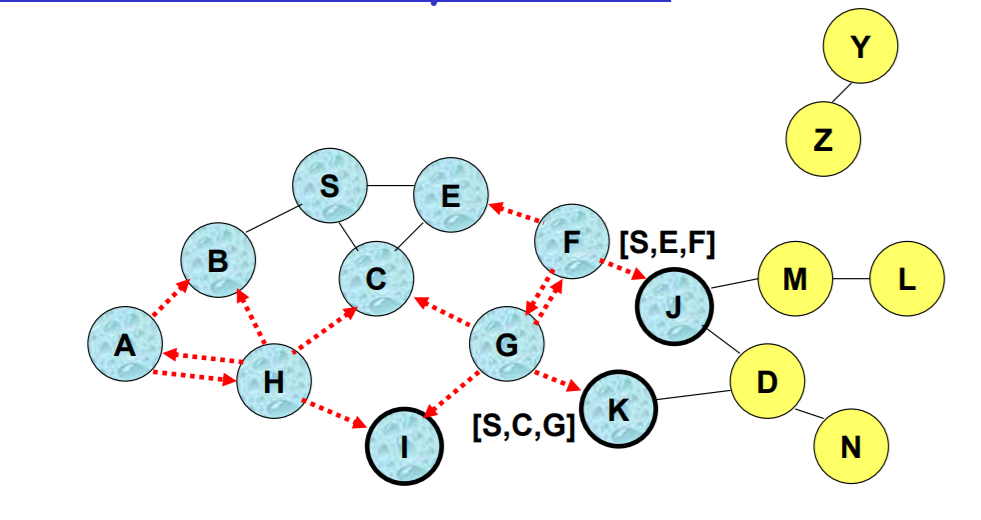
假设节点S要发送数据给节点D，但是检查路由缓存后发现不知道S->D的路由信息，于是S以洪泛Flooding的方式向邻居节点广播路由请求RREQ。

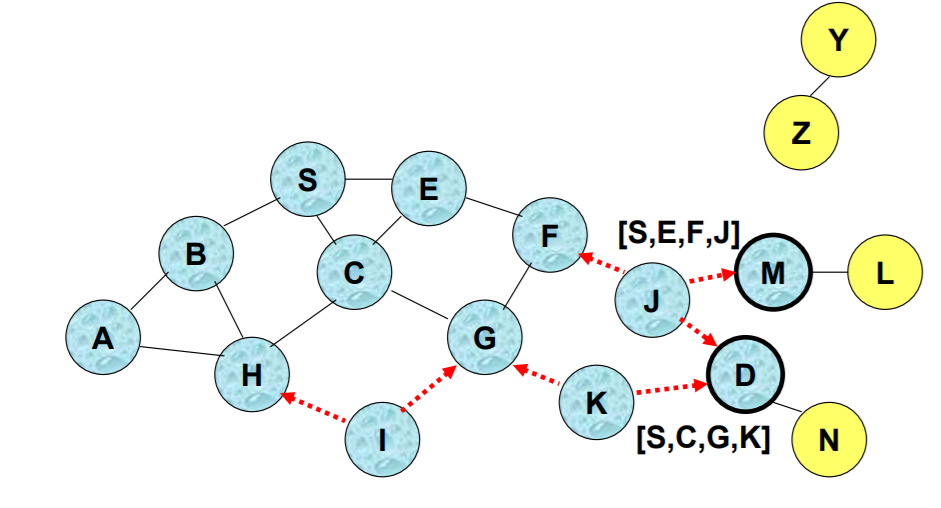


节点B，C，E收到RREQ后查询自身路由缓存表，未发现相关路由信息，于是将自身地址信息加入数据分组Header部分后继续向邻居节点广播。

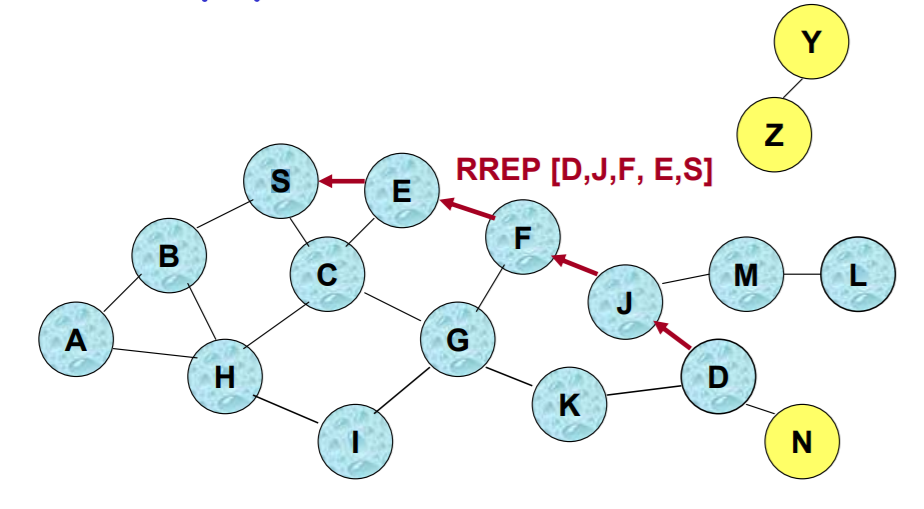


继续上述过程，其中C节点从G节点收到数据分组后不会继续转发，而是选择无视掉，因为C节点之前已经转发过这个路由请求。

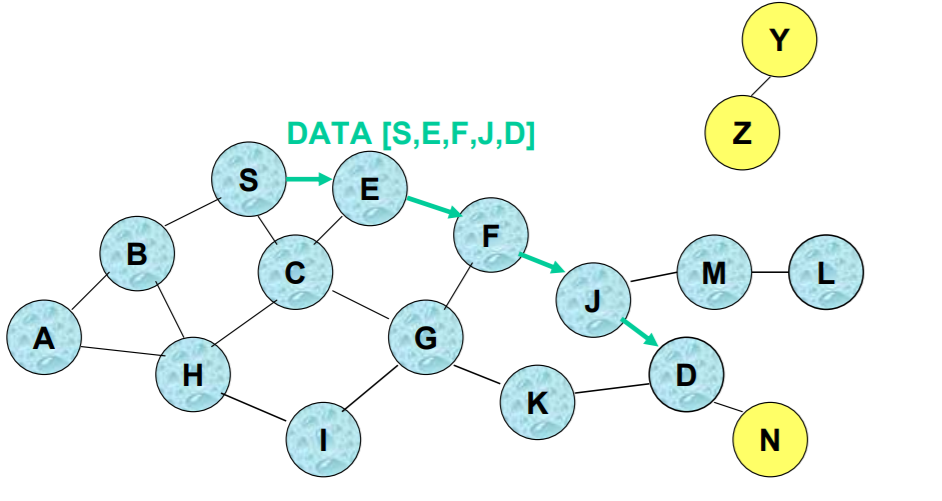




目的节点D收到路由请求后发现自身是目的节点，不再继续转发，而是返回路由回复Route Reply（RREP）数据分组。

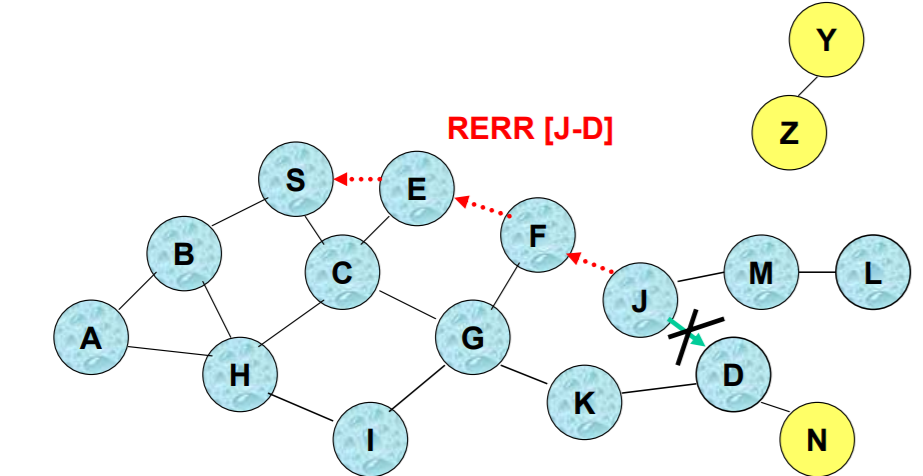


随后S存储该路由信息，每次发送的数据分组的头部都会包含相应的路由信息，从而可以正常通信



## 路由维护

假设某个时刻，节点J，D之间断开连接，当J节点试图向节点D转发数据分组时，就会向源节点S返回一个路由错误RERR，收到这个错误信息的节点会相应地更新自己的路由缓存信息

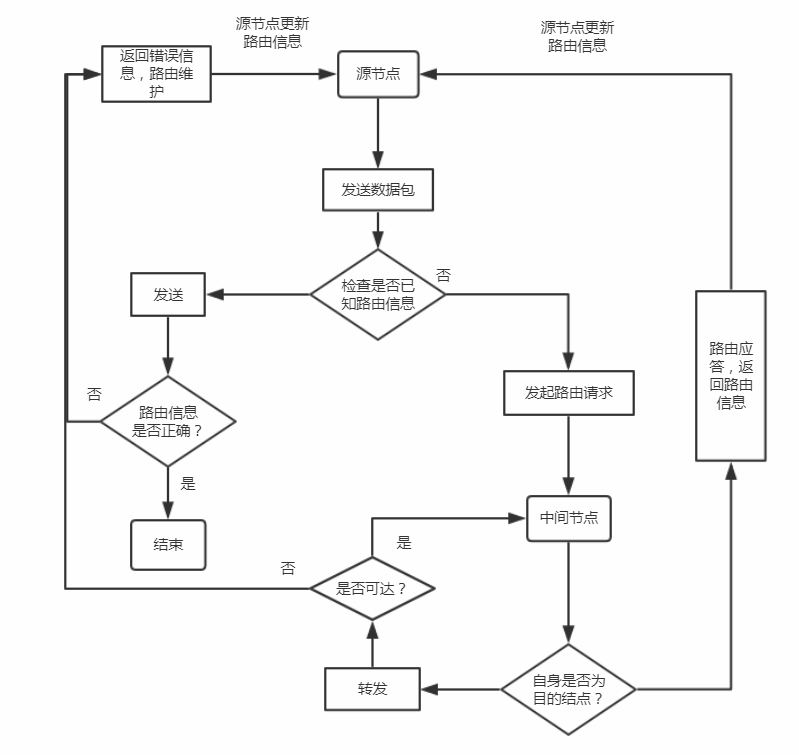


# 代码分析

为了实现NS2平台下的模拟，该源码使用了大量的宏定义来实现跨平台兼容，为了更加清晰地了解具体结构，以下分析中不考虑关于NS2的实现部分，仅考虑linux平台。

本部分包括以下内容：整体结构，首部选项，数据结构，以及操作处理。

整体简易流程图如下：

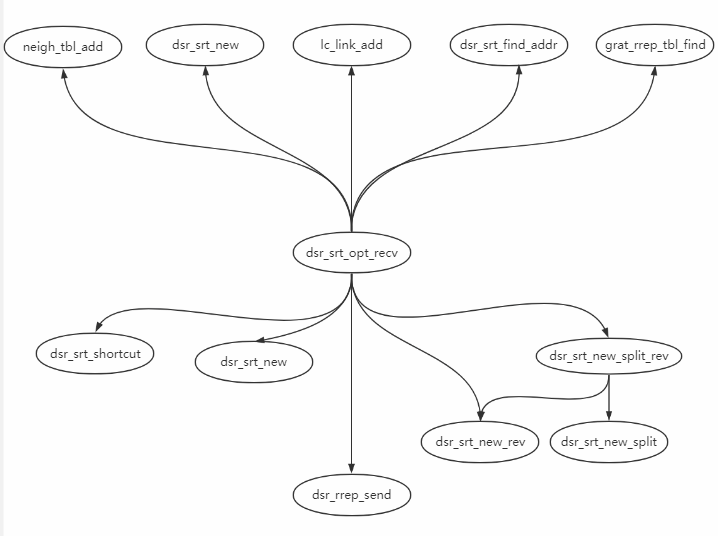


## 整体结构

函数调用图如下



其中，局部函数中重点分析dsr\_srt\_opt\_recv函数调用图如下：



源码中整体主要文件结构如下表：

|  |  |
| --- | --- |
| 文件 | 描述 |
| Dsr-opt.c | 封装DSR选项头以及添加删除选项等操作 |
| Dsr-srt.c | 定义源路由选项以及添加处理等操作 |
| Dsr-rrep.c | 定义路由请求选项以及相关操作 |
| Dsr-rreq.c | 定义路由回复选项以及相关操作 |
| Dsr-io.c  Dsr-dev.c  Link-cache.c  Maint-buf.c | 封装接收数据分组操作  调用硬件设备封装发送等操作  每个节点维护路有缓存  维护缓存用来存储未被确认的数据分组 |
| Send-buf.c | 发送缓冲区待发送的数据分组 |

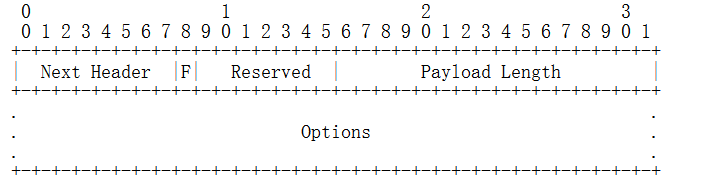
其他文件，如timer.h定义定时器，list.h和tbl.h定义相关链表，endian.c判断系统大端小端，NS-agent.h兼容NS2平台等等。

## 首部选项

分析一个协议是如何实现的，其首要部分应当是分析该协议的首部选项。动态源路由协议使用选项头来传送控制信息，这些信息可以包含在数据包的IP首部中。DSR选项头由固定部分和可选部分DSR选项组成。

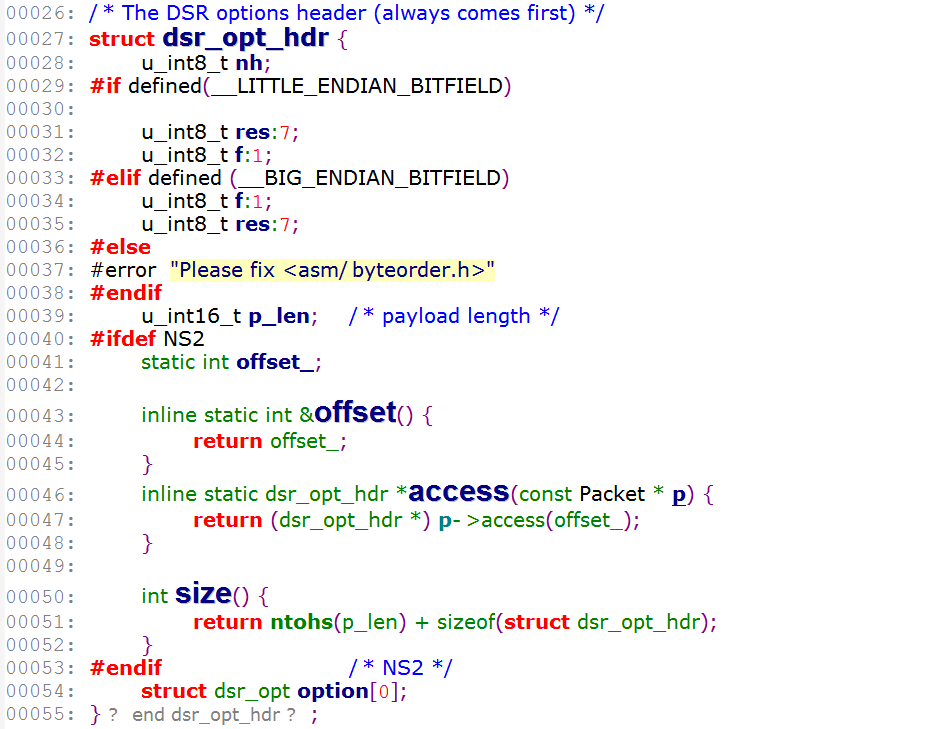
### 固定首部

DSR首部格式如下：



其中固定部分，或称之为基本首部，是固定大小的4个字节，结构定义如下：



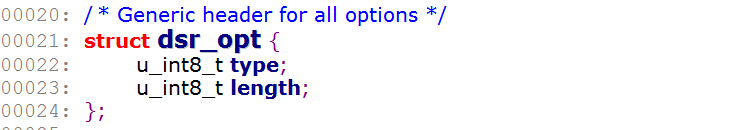


27-54：Nh字段，8位选择器，使用与Ipv4协议字段相同的值，如果没有下一个header则值为59。然后根据大端小端来定义不同的标记位域，以及2个字节的p\_len记录有效负载长度（定义所有选项的总长度），所以共4个字节。最后option变长数组记录可选的DSR选项部分。其中40-53行关于NS2的宏定义可以略去。

该实现源码中，可选部分选项的主要种类包括：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 名称 | | 作用 | | | Type | 值 |
| 路由请求Route Request | 启动路由发现 | | | DSR\_OPT\_RREQ | | 2 |
| 路由回复 Route Reply | 返回路由信息 | | | DSR\_OPT\_RREP | | 1 |
| 路由错误 Route Error | 返回不可达等错误信息 | | | DSR\_OPT\_RERR | | 3 |
| 源路由DSR Source Route | | | 携带源路径信息 | DSR\_OPT\_SRT | | 96 |
| 确认请求 ACK Request | 确认数据分组收到 | | | DSR\_OPT\_ACK\_REQ | | 160 |

这些选项的通用部分定义如下：

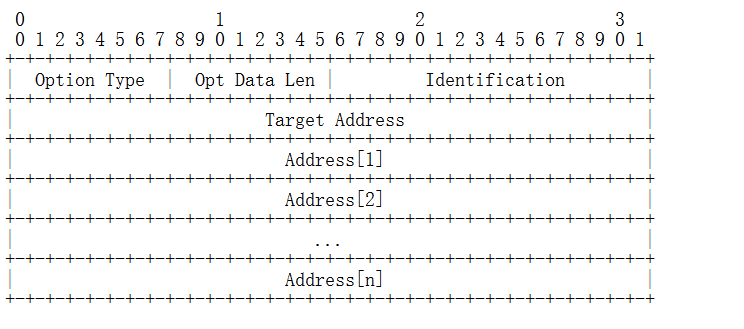


Type字段标记选项类型，length标记选项长度，其中每种type的值在opt.h文件中定义（如上图）。

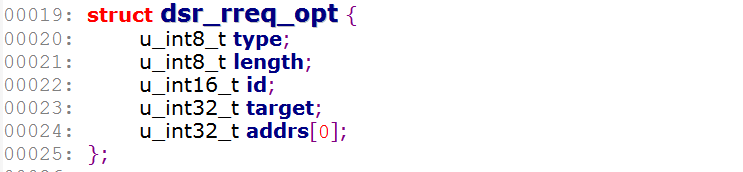
这里主要详细分析以下几种类型的选项。

### 路由请求选项

结构图：



源码定义：



**20-21：**选项通用部分，选项类型和长度，如上所述。不理解此类型的节点会忽略此选项。

**22：**id由路由请求发起者设置，以此作为每个请求的唯一标识，同时id值也使得每个接收节点可以识别它是否已经接收过同一请求，如果接收节点在自己的路由请求表中发现了该id值（对于特定的源IP地址和目的IP地址该id值是唯一的），那么这个节点就会忽略本次路由请求。

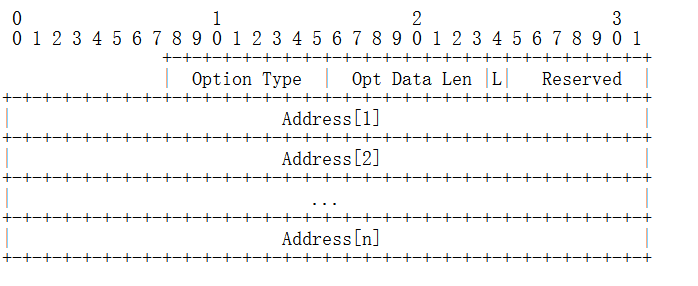
**23：**target记录路由请求的目的节点地址

**24：**addrs变长数组用以记录路由请求传播路径上的每个节点的地址信息。地址数量由该选项中的length指定，id和target共占6个字节，每个ip地址4个字节，故n=（length-6）/4

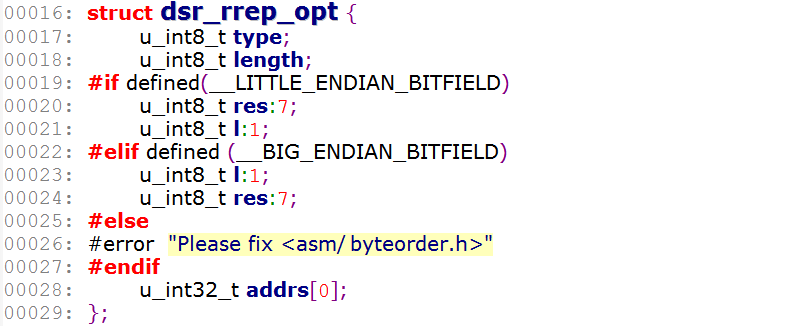
路由请求选项不得在一个DSR选项头中出现超过一次。

### 路由回复选项

结构图：



源码定义如下：

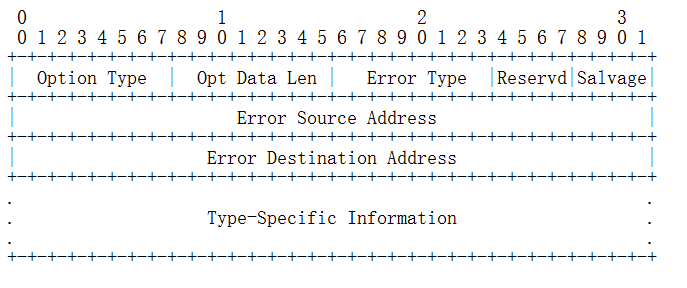


**17-28：**根据系统大小端的区别，定义不同的位域作为标志位。Res表示保留位，置0。L标识网络内外部，置1时，表明路由回复中的最后一跳到达了DSR网络的外部。变长数组addrs记录返回的路由信息。同样，其地址数量由length字段指定。

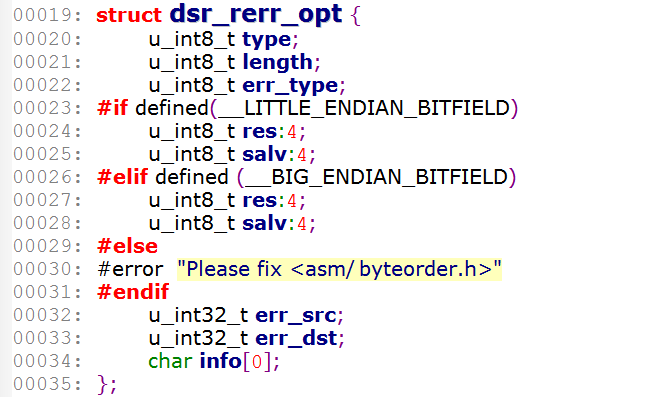
一个路由回复选项可能会在一个DSR选项头中出现不止一次。

### 路由错误选项

结构图：

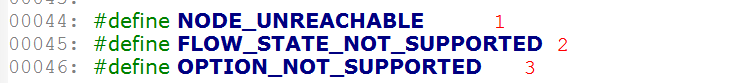


源码定义：



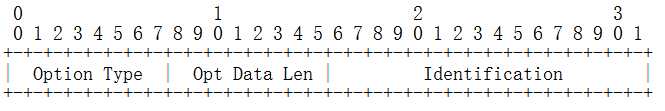
**22-29：**err\_type字段标记错误类型（其定义如图标号），salv从触发错误的数据包的源路由选项中的Salvage字段复制而来，标记数据分组被抢救的次数，避免数据包被无限抢救。

**32-34：**err\_src记录发起路由错误的节点地址，err\_dst记录错误目标地址，例如，当错误类型字段为NODE\_UNREACHABLE时，err\_dst会被设置为生成该路由信息的节点地址（即该节点认为此路径可达但事实上不可达）。Info数组记录有关此错误的信息。

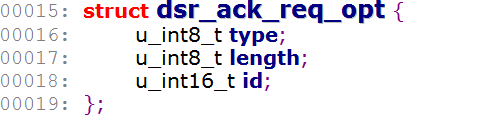


### 确认请求选项

结构图：



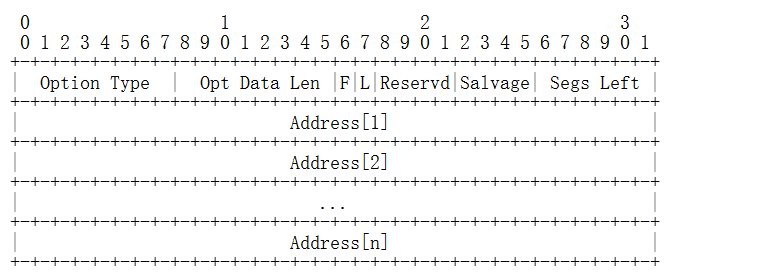
源码定义：



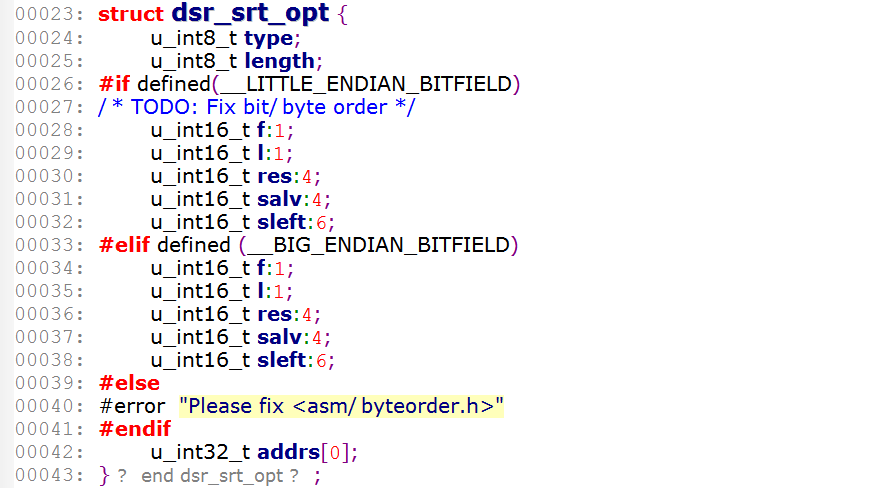
18：id字段被设置为一个唯一的值，在接收到数据分组的节点返回时被复制到确认选项的id字段中。确认请求选项在DSR选项头中只能出现一次。

### 源路由选项

结构图：



源码定义：



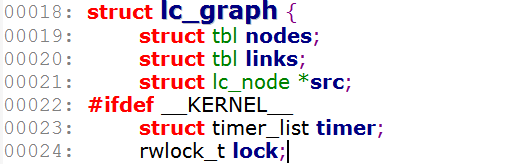
**23-42：**sleft记录剩余路段的数量，即在到达最终目的地之前仍然需要访问的中间节点数量。变长数组addrs记录源路由的路径信息。其他字段同上。

## 数据结构分析

协议中除了首部之外，接下来应该关注的部分就是数据结构。数据结构完成了协议中传递数据的任务，本文分析以协议相关内容为主，因此重点关注的是数据结构的成员分析，相关的增删改查等操作属于基础部分，不作分析。本节从源码中着重分析以下数据结构的组成：

### 路由缓存

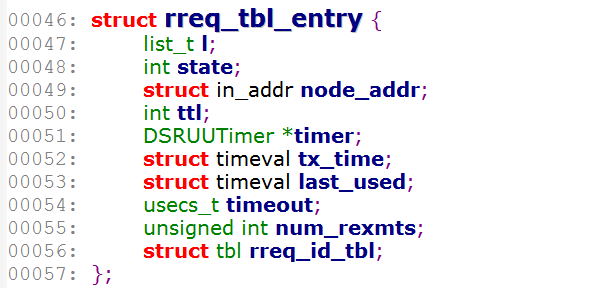
在本源码中，每个节点使用图结构来存储自身的路由缓存， 源码中List.h文件实现了一个简单的双向链表节点（接下来的缓冲区结构都有用到），以及相应的初步增删等操作，在头文件Tbl.h中，对于此链表节点进行了进一步的封装，增加了链表长度等信息，完善了增加，删除，查找等操作。而在link-cache文件中，通过存储节点信息以及链接信息，以图的方式将路由缓存信息存储下来，并使用dijkstra算法来查找最短路径。这里图的实现方式使用的不是矩阵，而是之前定义的链表（邻接表表示法）。源码定义如下：



19-24：nodes和links构成图的两大要素，记录节点以及链路关系，包含有长度以及前后指针，src记录源节点，包括地址，跳数等信息，timer\_list是linux内核的定时器，rwlock\_t读写锁实现对读写的保护。

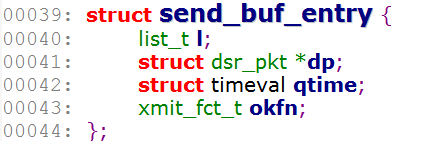
这里实现的是单条路由表项。

### 路由请求表



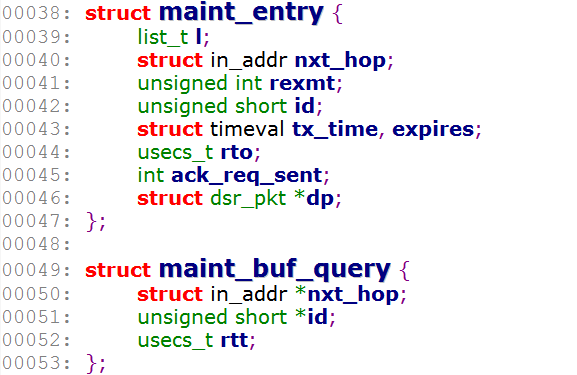
47-56：state : 请求状态，node\_addr：目标地址，ttl：为该目标节点发起的最后路由发现的生存期，rreq\_id\_tbl： last\_used该节点最后发起该目标节点的路由请求的时间，timeout：超时时间间隔

### 发送缓冲区



40-43：如前文所述，通过l字段维护链表，dp指向待发送的数据分组，qtime标记超时时间，超时后该记录会被删除，okfn为函数指针，指向xmit\_fct\_t类型的函数，该函数参数为某个DSR数据分组的地址，将该数据分组发送出去。

### 维护缓冲区



39-46：nxt\_hop标记下一跳地址，rexmt记录重传次数，id标识每个数据分组，随后定义最后重传时间，该数据分组地址等，其中ack\_req\_sent事实上相当于一个bool类型变量，初始值为0，如果有PKT\_REQUEST\_ACK选项则被置为1。

49-52：维护该缓冲表项队列，rtt记录往返时间。

## 操作处理

接下来分为三个部分来分析具体操作处理函数，每部分分别讨论如何发起和如何处理（即输入和输出）。

1. 通用数据分组处理
2. 路由发现处理
3. 路由维护处理

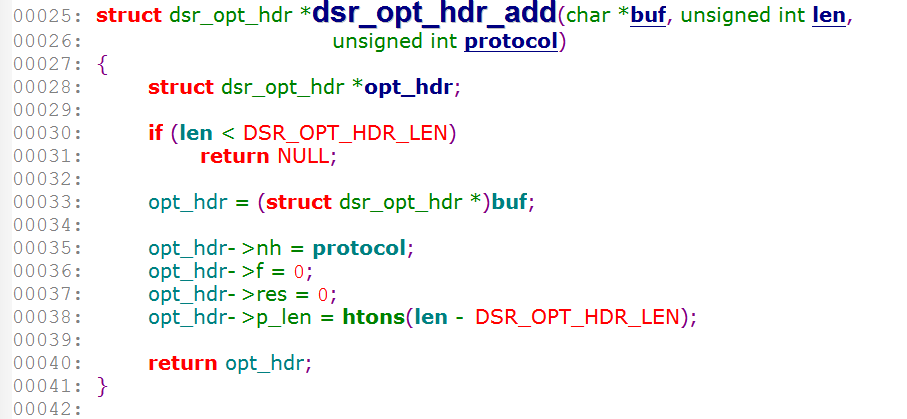
### 通用数据分组处理

1. 始发数据包：

发起任何数据分组时，网络中的节点都需要执行以下步骤：首先，在节点的路由缓存中搜索到数据包报头中目标IP地址字段中给定地址的路由。

其次如果未找到，则执行路由发现，节点会在此现有数据分组的DSR选项头中添加路由请求选型，或保存到发送缓冲区，

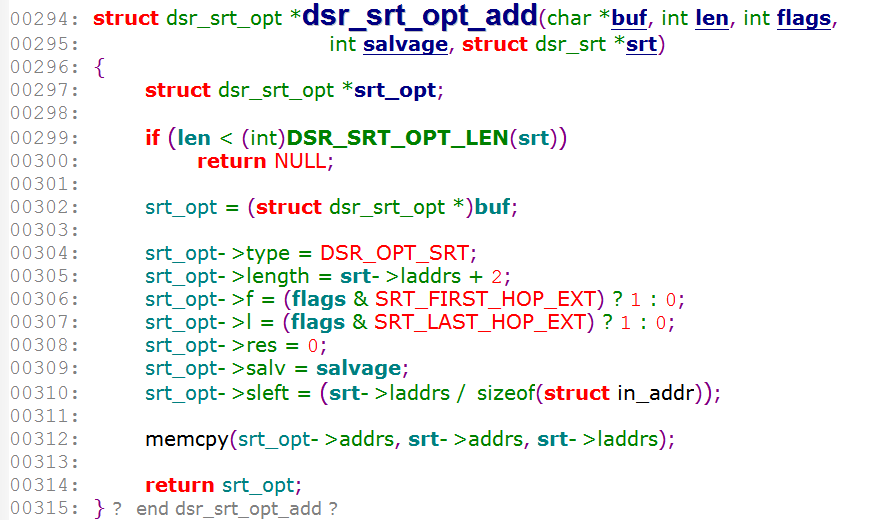
将DSR选项头添加到数据分组代码实现如下：



28-40：首先定义每个选项头的固定部分，之后判断如果长度足够，则开始装载相关设置，将参数中缓冲区指针强制类型转换为一个指向DSR选项头的指针，Next Header字段设置为数据分组的IP头的协议号，最后返回该指针。

将DSR源路由选项添加到数据分组

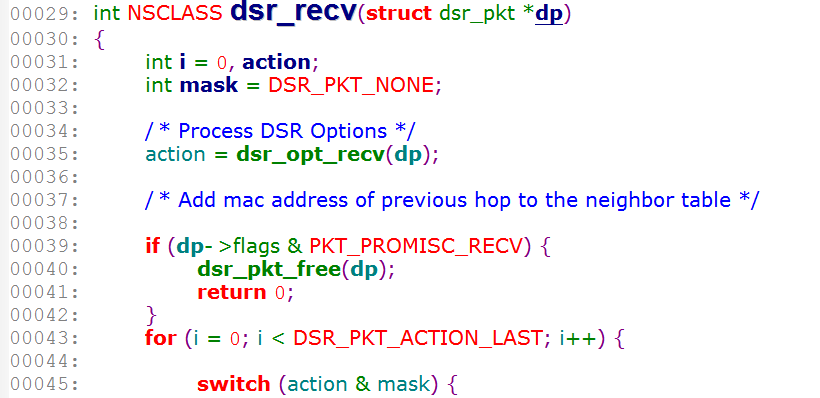
发起分组的节点在需要时向分组添加DSR源路由选项，以便将来自该始发节点的源路由携带到分组的最终目的地地址，代码实现如下：



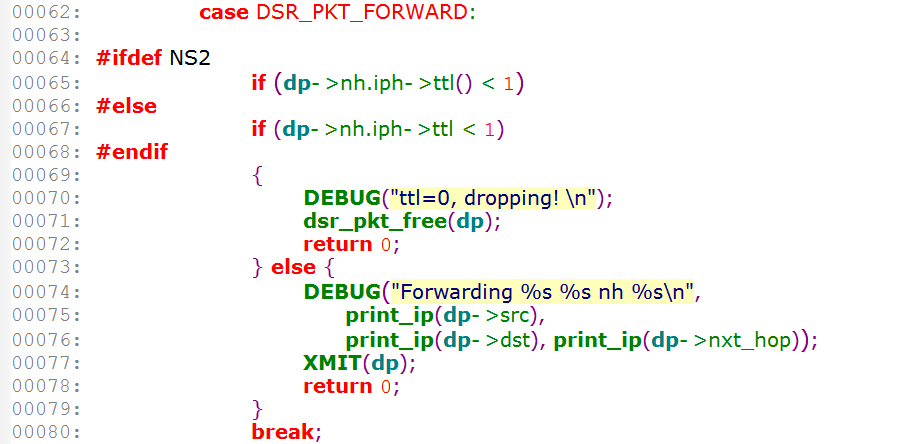
297-312：选项头固定部分定义，缓冲区长度判断以及指针转换同上，接下来开始装载选项字段，将类型设置为源路由选项，初始化长度，标志位，抢救次数，segmens left，随后数据分组中的地址被复制到DSR源路由选项中的addrs数组字段。

1. 处理收到的数据分组：

当一个节点收到任何数据分组时，如果该数据分组包含一个DSR选项头，那么该节点必须处理该DSR选项头中包含的所有选项。处理路由请求的代码实现如下：

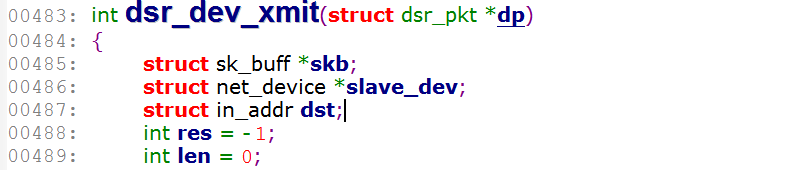


31-43：节点会根据数据分组携带的选项来进行不同的处理操作，调用的dsr\_opt\_recv函数通过对数据分组中选项的类型进行识别然后返回对应的操作类型，随后使用switch语句根据返回的操作类型进行不同的操作，在这些分支中这里着重分析较为复杂的转发请求，代码如下：

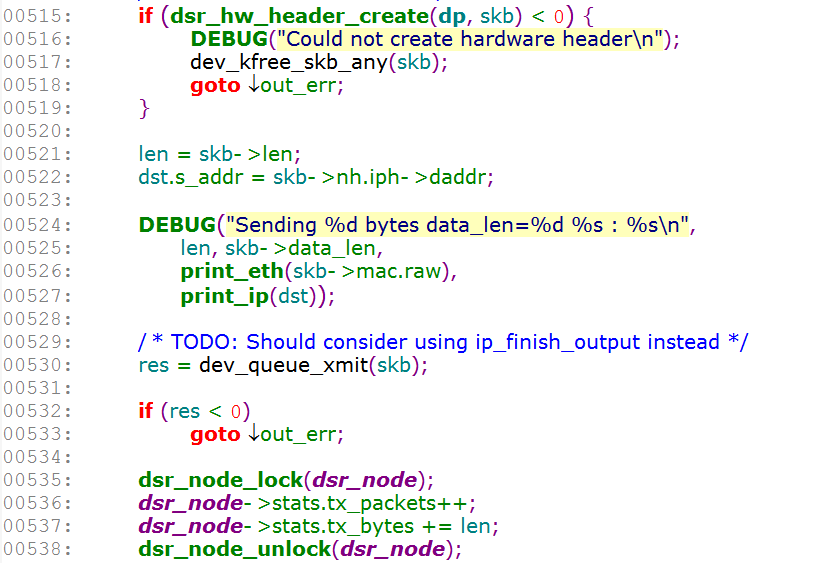


首先进行生存期ttl的判断，如果 ttl为零则丢弃数据分组，不为零则调用XMIT函数进行发送数据，XMIT是如下函数dsr\_dev\_xmit的宏定义，代码如下：

（该函数实现了dsr数据包到IP数据包的向下封装）



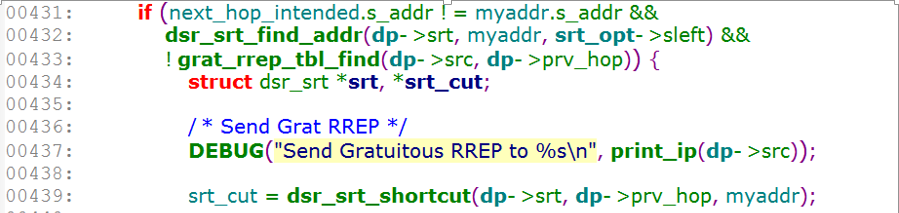
其中，结构体sk\_buff与m\_buf类似，是linux系统中各层协议之间传递数据的重要载体，net\_device结构体用来描述一个网络设备，dst字段标记目标地址

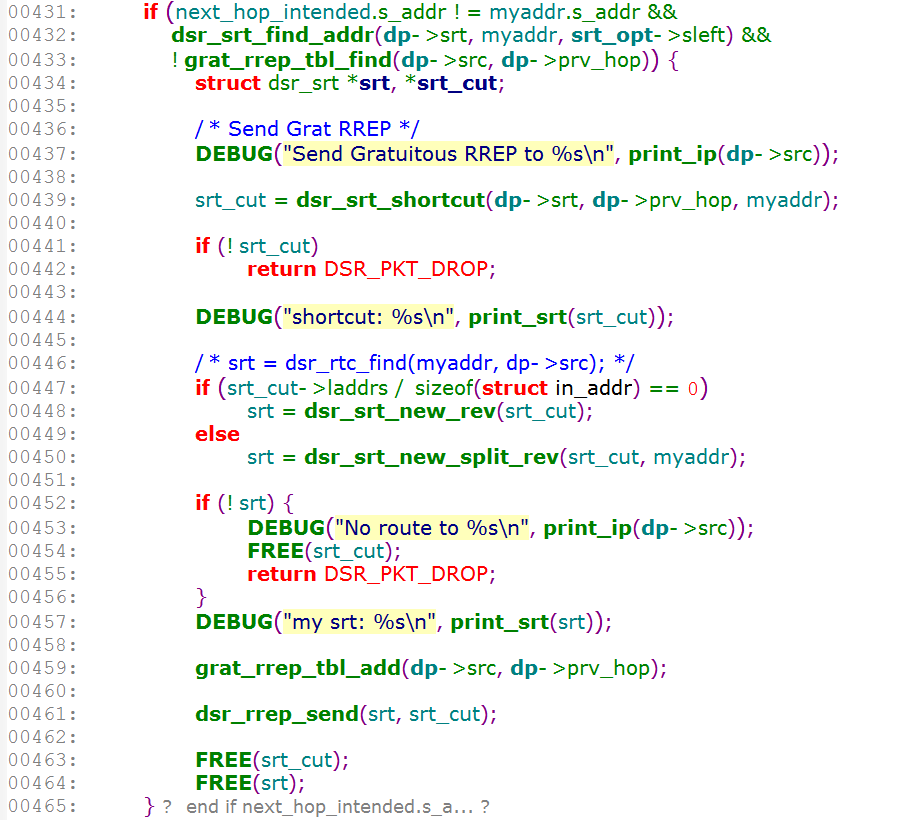


497-507：通过锁来实现dsr\_node（描述当前节点信息）的并发安全性，将抢救网卡设备指定为当前节点的相应设备字段，随后调用dsr\_skb\_create函数（装填相关字段信息并返回一个sk\_buff的指针）创建数据分组。

515-538：随后开始向创建后的数据分组中装填mac地址相关字段，并调用dev\_queue\_xmit函数进行发送。

1. 处理收到的DSR源路由选项

当一个节点收到一个包含DSR源路由选项的数据包时，该节点应该检查数据包以确定是否可以进行自动缩短路由，具体来说，如果一个节点不是该数据分组的下一跳地址，但是在路由信息的后续地址中有该节点的地址，则应执行自动路径缩短，代码实现如下：Dsr-srt.c 



432-439：判断是否可以进行路由缩短：如果节点自身地址并非下一跳地址，并且在路由后续地址中查找到了自己的地址的同时自身应答列表中尚没有该路由回复，则调用dsr\_srt\_shortcut函数进行路由缩短，返回一个缩短后的源路由选项结构src\_cut。

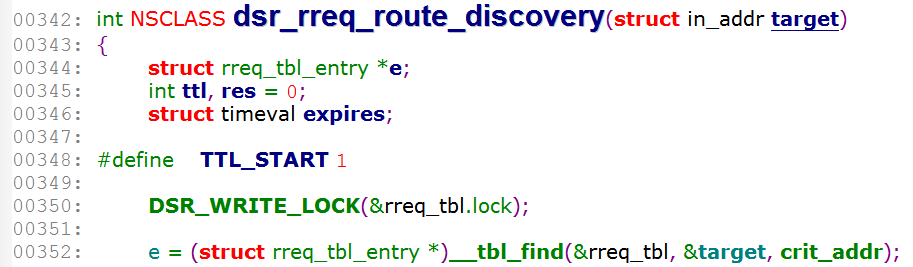
447-461：如果缩短后剩余节点个数为0（即自身为目的节点），则直接调用dsr\_srt\_new\_rev函数将src\_cut中的路径进行反转（因为要向源节点发送应答，需原路返回），如果自身不是目的节点，则调用dsr\_srt\_new\_split\_rev函数，该函数会先调用dsr\_srt\_new\_split函数进行分割（切割掉多余的路径），将路径信息切割至自身节点后再调用dsr\_src\_new\_rev函数进行反转。随后该节点会使用反转后的路由信息向源节点发送应答分组，该分组携带有源节点到该节点的缩短后的路由信息。

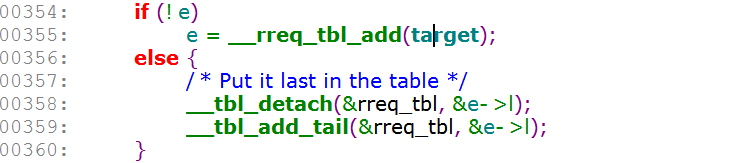
如果不能缩短则直接转发。

### 路由发现处理

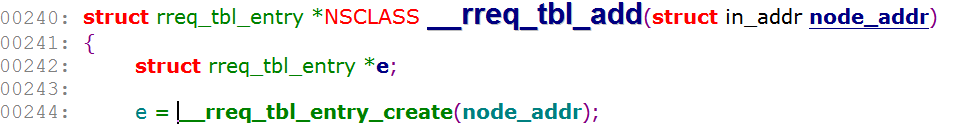
1. 发起路由发现请求

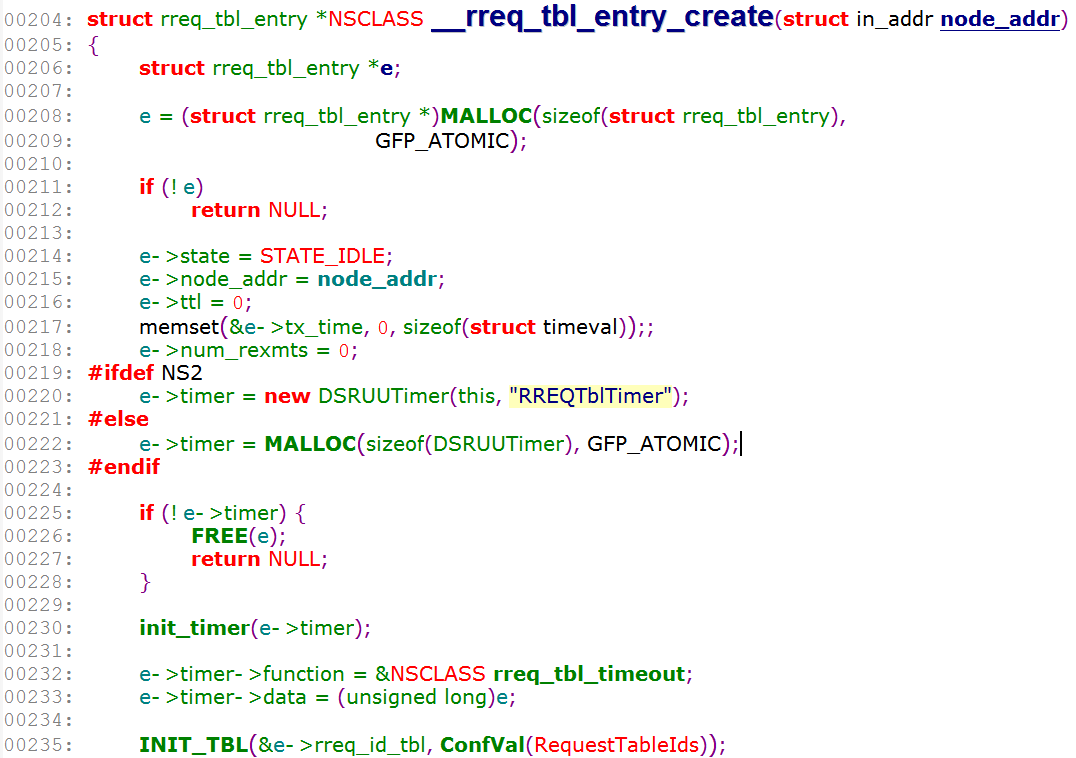
启动路由发现的节点会在某个IP数据包的DSR选项头中创建并初始化一个路由请求选项。这可以是一个单独的IP数据包，仅用于执行此路由请求选项，也可以在某些需要发送到目标节点的现有数据包中包含路由请求选项。需要完成的步骤有1添加到本地路由请求表2发送路由请求。代码实现如下：dsr\_rreq.c





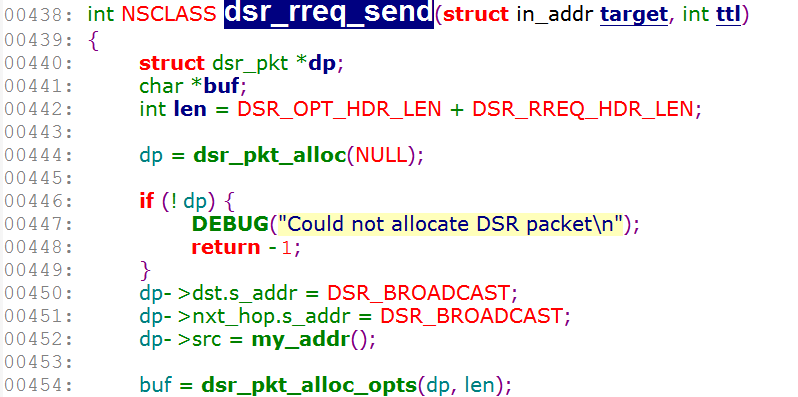
352-360：首先通过\_tbl\_find函数在路由请求表中查找是否有相同目的地的路由请求，如果没有则调用\_\_rreq\_tbl\_add函数直接添加进路由请求表，否则先删除原有记录，再调用\_tbl\_add\_tail添加进路由请求表中的最后（使用先进先出的管理策略，如果请求表满会先删除第一条记录）。其中\_\_rreq\_tbl\_add函数主要调用\_rreq\_tbl\_entry\_create函数，代码如下：

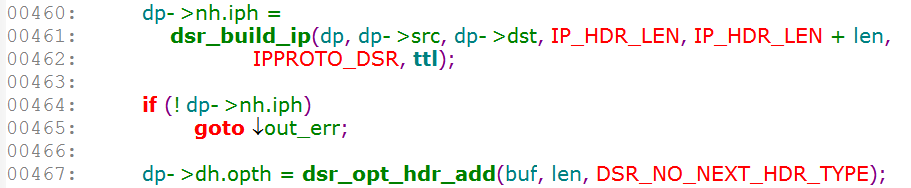


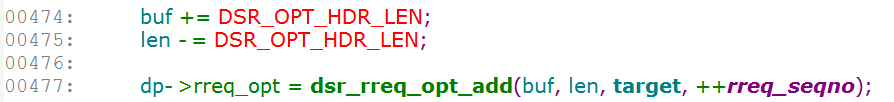


206-235：首先为该条请求记录申请内存空间，然后开始装填状态，目标地址，生存期ttl等字段。最后需要为每条请求初始化一个唯一标识id，避免被无限转发。

发送请求部分代码如下：









440-491：首先申请构造一个DSR数据包，将源地址设置为自身地址，目的地址和下一跳地址都设置为广播地址，然后调用dsr\_build\_ip函数构造IP报头，调用dsr\_opt\_hdr\_add添加DSR选型头，再调用dsr\_rreq\_opt\_add函数向DSR选项头中添加路由请求选项，最后调用XMIT函数发送数据分组。

1. 处理路由发现请求

当一个节点收到一个包含路由请求选项的数据包时，该节点的操作如下：

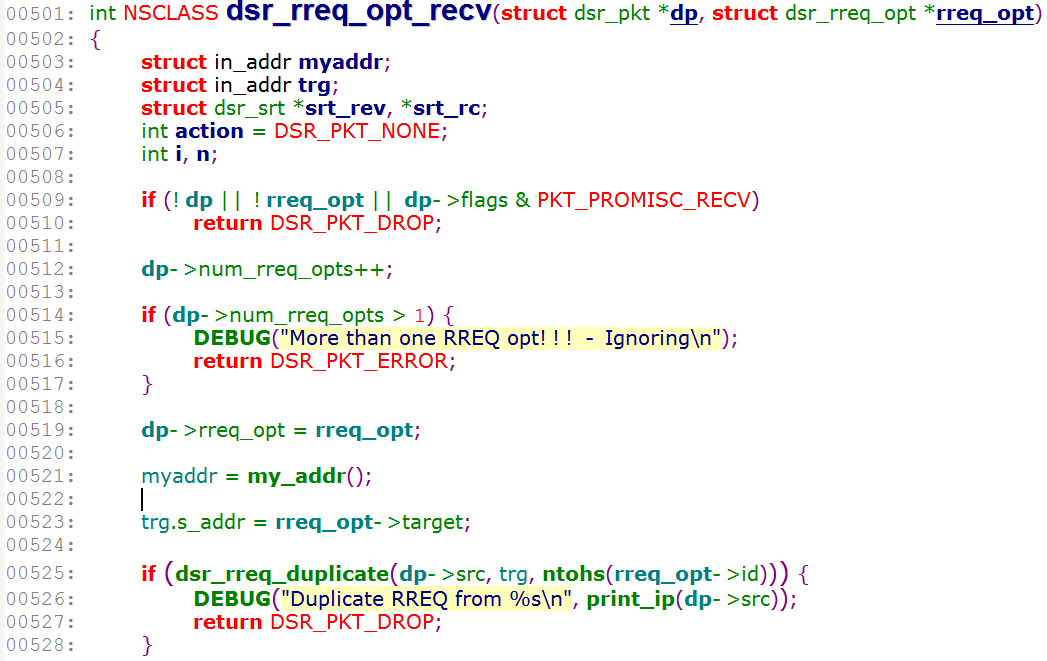
首先判断是否满足丢弃条件或返回条件

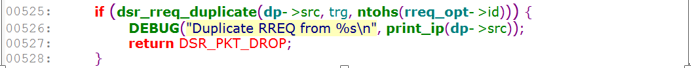
1. 如果路由请求中的目标地址字段与这个节点自己的IP地址相匹配，那么节点应该返回一个路由回复给这个路由请求的发起者
2. 如果不匹配，节点必须检查路由请求选项（IP源地址字段和地址字段序列）中记录的路由，以确定此节点自己的IP地址是否已经出现在该地址列表中。如果查找到（即自己已经处理过），则该节点会丢弃整个分组。如果未查找到，则该节点会在其路由请求表中搜索该路由请求的发起者（IP源地址字段）的记录。
3. 如果在表中找到这样一个记录，那么节点必须搜索该表项中最近接收到的路由请求的标识值缓存表，以确定在该路由中是否存在与该标识值和目标节点地址匹配的高速缓存中的条目请求。如果在这个缓存中发现这样一个（标识，目标地址）条目在路由请求表中（即自己已经接收过），那么节点必须丢弃整个携带路由请求选项的分组。

判断完毕后如果未丢弃或是返回，则这个节点应该按照以下步骤进一步处理路由请求：

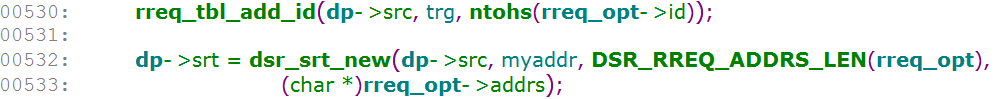
1. 在最近收到的路由请求的缓存中为此路由请求添加一个条目。
2. 将此节点自己的IP地址追加到路由请求中的地址值列表中（即addr数组），并将路由请求中的Opt Data Len字段的值增加4（IP地址的大小）。
3. 同时这个节点会搜索自己的路由缓存，查找是否有到这个目标地址的路由。如果找到，则向这个路由请求的发起者返回路由答复。如果节点未找到相关路由，那么这个节点会把这个数据包的副本作为链路层广播发送，

代码如下：



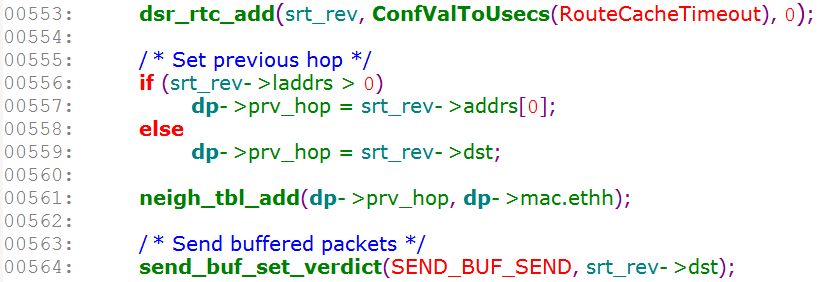


503-527：这里调用dsr\_rreq\_duplicate函数判断是否收到的路由请求是否已经在自己的路由请求表中，如果已经包含则丢弃该请求包。

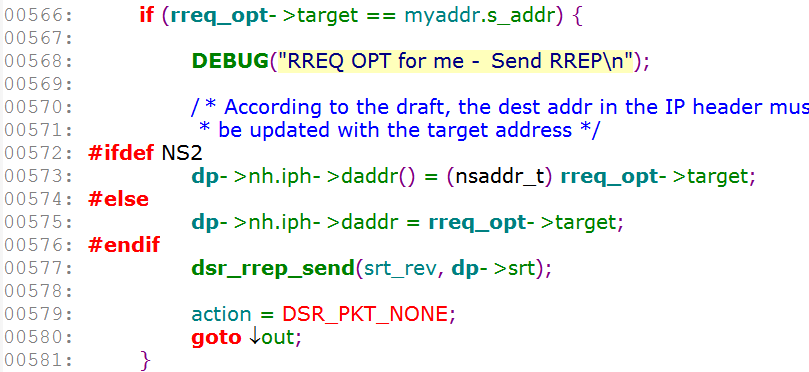




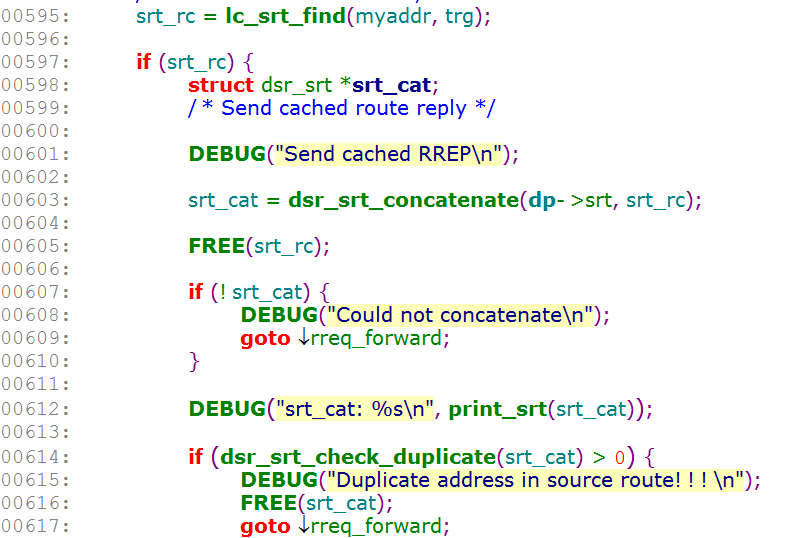
532-544：先将该请求添加进路由请求表中并关联该请求的标识id，再新建一个源路由选项，随后调用dsr\_srt\_new\_rev函数将所收到的数据包中的路径信息反转（参考3.4.1），因为之后可能需要返回路由回复信息。



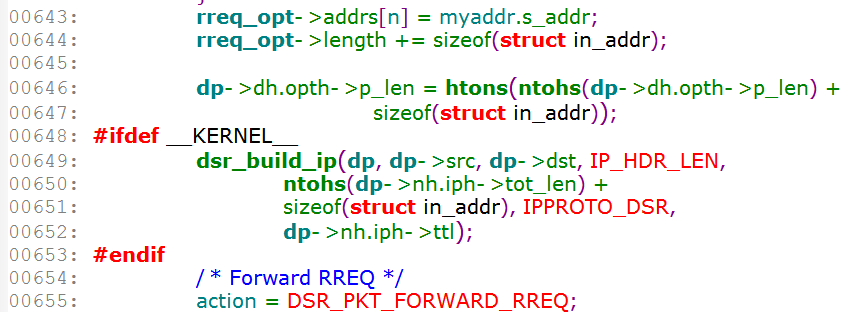
553-564：先调用dsr\_rtc\_add函数将该路由信息添加进自身的路由缓存中（节点可以从路由请求听到路由信息），之后如果剩余地址数大于0则将自身节点地址最后将该数据分组复制到发送缓冲区中。



566-580：如果请求的目标地址是自身地址，则将IP Header字段设置为自身地址，随后该节点调用dsr\_rrep\_send函数返回路由回复。



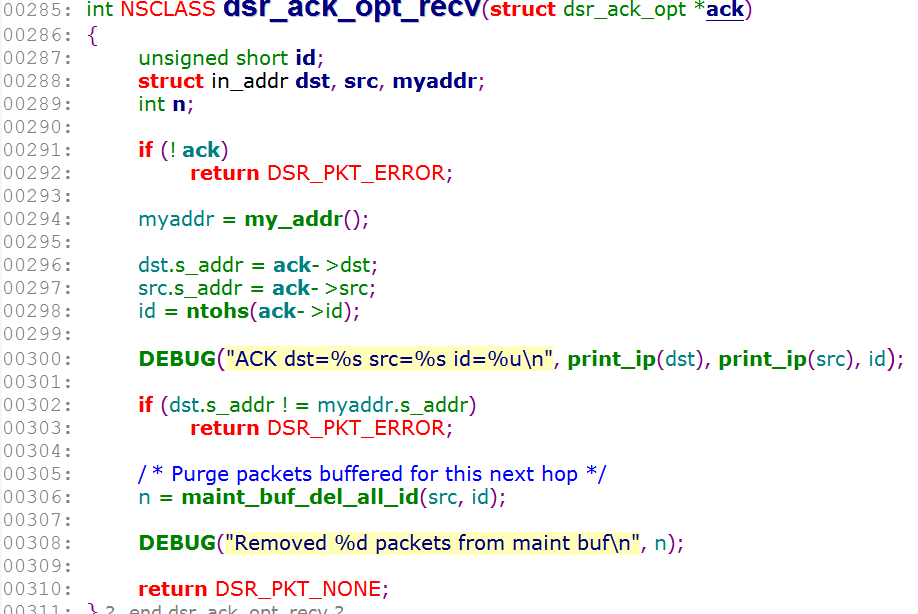
595-625：接着在自身的路由缓存表中查找该目的地址的路由，如果找到，就将请求中的源路由部分和查找到的路由部分拼接起来，在检查确认拼接后的路由没有重复节点后，调用dsr\_rrep\_send函数使用之前反转后的路径发送该路由回复信息（携带拼接后的路径）。



643-655：如果拼接后的路由信息包含重复节点，则不得向源节点发送路由回复，而是选择将自身节点地址添加到addrs数组中，构造IP数据报头，之后转发该数据分组。

### 路由维护处理

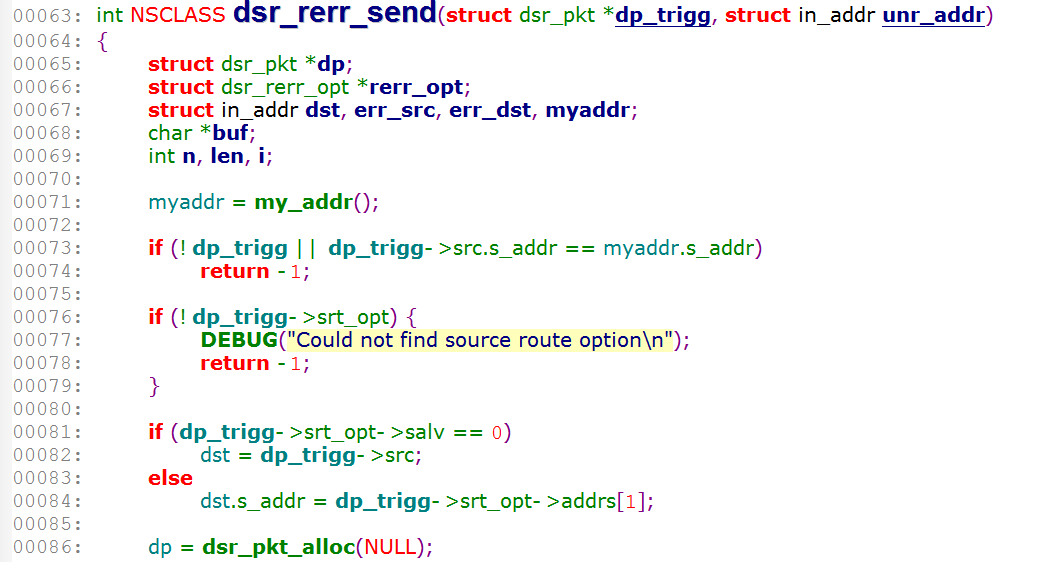
1. 使用确认

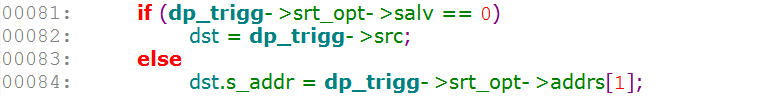


287-303：如果未确认，则返回错误，检查ack中的目的地址是否为自身地址，确认成功则将维护缓冲区中相应id的数据分组删除。

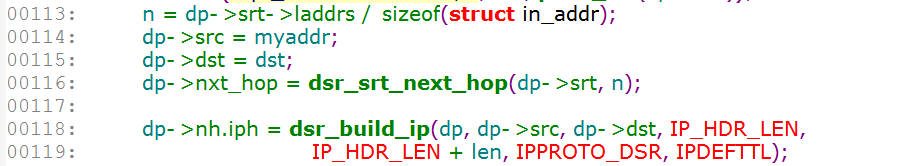
1. 发起路由错误

当一个节点在达到最大重发次数后依然不能验证下一跳节点的可达性时，它应该向该分组的IP源地址发送路由错误，代码实现如下：

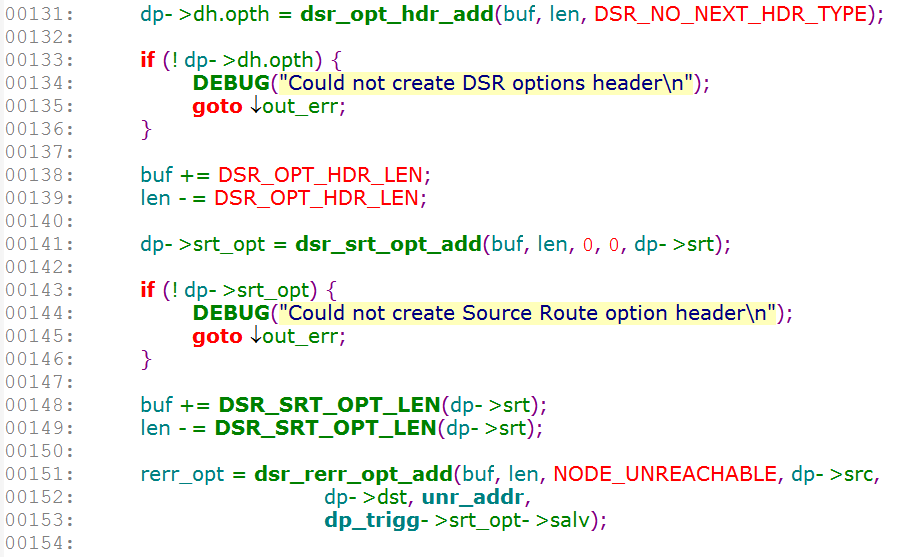








65-119：新建一个数据包，将源地址设置为节点自身地址。如果触发路由错误的数据分组中抢救次数为0，则将目的地址设置为该触发此路由错误的数据分组中的源节点地址，否则将目的地址设置为addrs数组中的addr[1]。随后调用dsr\_build\_ip装填IP报头相关字段。其中93行在自身节点路由缓存中查找通往目的节点的路由，便于后续发送路由错误。

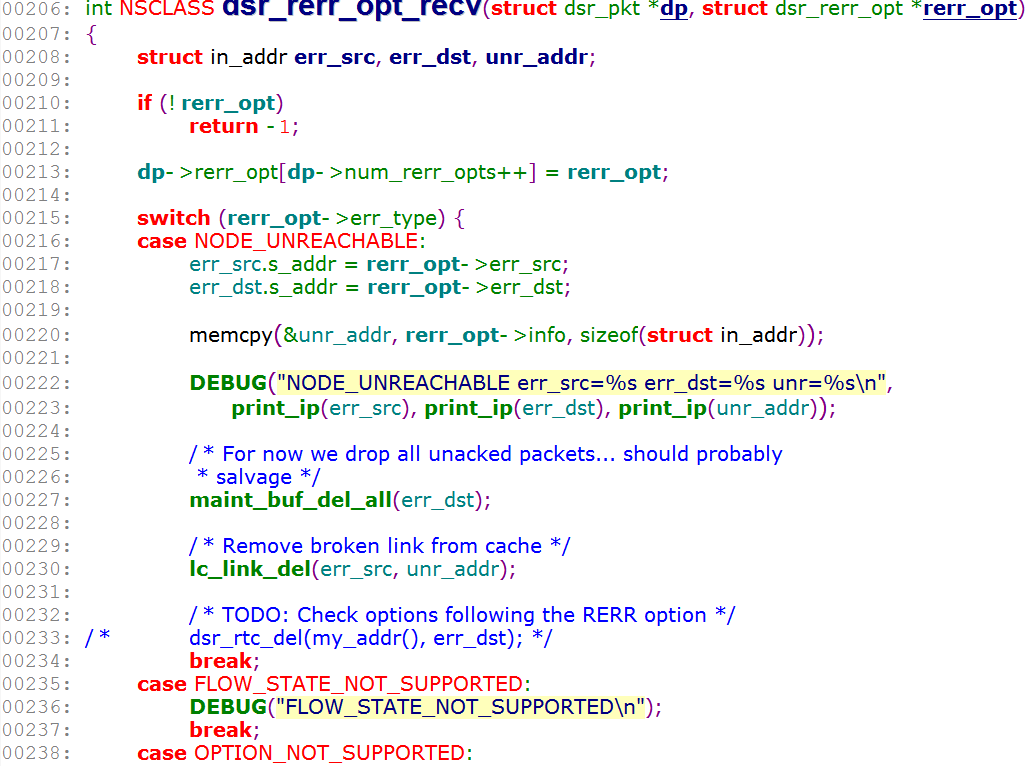


131-154：随后添加DSR路由选项头，向选项头中添加源路由选项，携带之前查找到的路由信息。同时向选项头中添加路由错误选项，类型设置为NODE\_UNREACHABLE，不可达地址设置为源路由中下一跳地址。

同时如果触发路由错误的数据包包含任何路由错误或确认选项，则节点可以将这些选项中的每一个追加到其路由错误中。后续代码中使用for循环实现追加选项。

1. 路由错误处理

当节点收到一个包含路由错误选项的数据包时，该节点从其路由缓存中删除从由错误源地址节点到由不可达节点（如果该路由存在于其路由缓存中）的路径信息。:



206-230：如果错误类型为不可达，则输出不可达地址信息，并将维护缓冲区中所有等待目标地址返回ack确认的数据包删除，同时删除不可达路径，更新自己的路由缓存。

所有的路由选项在函数dsr\_opt\_recv中处理，在while循环中使用switch语句对不同的选项调用相关的处理函数进行处理，所以所有附加的错误选项在213行通过自增操作都可以被处理。