**网络协议栈分析与设计课程大作业**

**DSR路由协议代码分析**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 学号 | 姓名 | 班级 | 负责模块 | 成绩 |
| 201692204 | 张伟 | 软网1604 | 1.通信过程分析简介  2.数据结构分析(.h)  3.具体函数代码分析（.c）  4.PPT撰写与课堂展示  5.文档撰写与总结 |  |
| 201692481 | 常瑞 | 软网1604 | 1.协议简介  2.协议组成介绍  3.函数关系图  4.文档排版 |  |

目录

[引言 3](#_Toc533804045)

[第一章 基本原理 4](#_Toc533804046)

[1.1 DSR协议简介 4](#_Toc533804047)

[1.2 DSR协议组成 4](#_Toc533804048)

[1.2.1路由发现 4](#_Toc533804049)

[1.2.2路由维护 5](#_Toc533804050)

[1.2.3路由缓存 5](#_Toc533804051)

[1.3 通信过程简介 5](#_Toc533804052)

[第二章 数据结构 7](#_Toc533804053)

[2.1 DSR选项头与选项关系 7](#_Toc533804054)

[2.2 DSR选项头分析 7](#_Toc533804055)

[2.3 DSR选项通用头分析 9](#_Toc533804056)

[2.4 DSR路由请求选项 9](#_Toc533804057)

[2.5 DSR路由回复选项 10](#_Toc533804058)

[2.6 DSR路由错误选项 11](#_Toc533804059)

[2.7 DSR路由维护选项 13](#_Toc533804060)

[第三章 函数关系图 14](#_Toc533804061)

[3.1 总体流程图 14](#_Toc533804062)

[3.2 函数关系调用图 15](#_Toc533804063)

[第四章 协议具体函数代码分析 18](#_Toc533804064)

[4.1 DSR数据包结构 18](#_Toc533804065)

[4.1.1 dsr\_pkt()数据包结构 18](#_Toc533804066)

[4.1.2 dsr\_pkt\_alloc()函数分析 20](#_Toc533804067)

[4.2 为DSR选项头分配内存 21](#_Toc533804068)

[4.2.1 dsr\_opt\_hdr\_add()函数分析 21](#_Toc533804069)

[4.3 协议IO分析 21](#_Toc533804070)

[4.3.1 dsr-recv()函数分析 22](#_Toc533804071)

[4.3.2 dsr\_start\_xmit()函数分析 23](#_Toc533804072)

[4.4 处理源路由请求 24](#_Toc533804073)

[4.4.1 dsr\_srt结构分析 24](#_Toc533804074)

[4.4.2 dsr\_srt\_opt\_add()函数分析 24](#_Toc533804075)

[4.4.3 dsr\_srt\_opt\_recv()函数分析 25](#_Toc533804076)

[4.5 发出路由发现请求 26](#_Toc533804077)

[4.5.1 rreq\_tbl\_entry 结构定义分析 26](#_Toc533804078)

[4.5.2 dsr\_rreq\_route\_discovery() 函数分析 27](#_Toc533804079)

[4.5.3 \_tbl\_find()函数分析 28](#_Toc533804080)

[4.5.4 \_rreq\_tbl\_add()函数分析 29](#_Toc533804081)

[4.5.5 rreq\_tbl\_entry\_create()函数分析 30](#_Toc533804082)

[4.5.6 dsr\_rreq\_send()函数分析 31](#_Toc533804083)

[4.5.7 dsr\_dev\_xmit()函数分析 33](#_Toc533804084)

[4.6 处理路由发现请求 34](#_Toc533804085)

[4.6.1 dsr\_rreq\_opt\_recv()函数分析 35](#_Toc533804086)

[4.7 确认请求（ACK Request）选项处理 38](#_Toc533804087)

[4.7.1 dsr\_ack\_req\_opt\_recv() 函数分析 38](#_Toc533804088)

[4.7.2 dsr\_ack\_send()函数分析 39](#_Toc533804089)

[4.8 确认（ACK）选项处理 40](#_Toc533804090)

[4.8.1 dsr\_ack\_opt\_recv()函数分析 40](#_Toc533804091)

[4.9 产生路由错误选项 41](#_Toc533804092)

[4.9.1 dsr\_rerr\_send()函数分析 41](#_Toc533804093)

[4.10 处理路由错误选项 41](#_Toc533804094)

[4.10.1 dsr\_rerr\_opt\_recv()函数分析 42](#_Toc533804095)

[第五章 小结 42](#_Toc533804096)

引言

本文对动态源路由协议（The Dynamic Source Routing protocol）的一个实现“dsr-uu”进行代码分析，该项目最初在乌普萨拉大学（Uppsala University）被完成，这也是它的名字中“UU”的含义，而DSR则是Dynamic Source Routing的缩写，本文中出现的所有“DSR”也将代指这个协议，讨论的情况也是在网络节点使用动态源路由协议的基础下进行的。

DSR协议函数众多，但是不难分析，函数结构连贯清晰，我花费了大概三周时间看懂了其中函数相互间调用的关系，并在本文第3节绘制了这些函数之间的调用关系图和总体流程图，这些图是后期进行协议代码具体分析的主要思路引导。其实只有初步分析时比较困难，当你分析好一个源文件后你会对其他文件的布局概况就已经有了初步的掌握，无非是多几个参数变量，函数等等。

我在分析时并不是一个源文件一个头文件的离散的分析各个函数，主要是以一个数据包发送为开始按照1.3节通信的流程，去找函数实现的过程，再结合函数间的调用关系，把握整个协议的内在结构逐步剖析整个协议，以此完成对DSR协议的代码分析，其中在1.2节路有缓存功能中还和AODV协议做了比较。

一边分析一边记录，在我的GitHub上有本文的所有代码截图和分析注释。README文件中有我们组员的一些基本信息。

本文的总体安排是：

第一章介绍DSR协议的基本原理。

第二章对DSR数据结构的分析，也是对主要.h文件的详细介绍，为后期函数分析做铺垫。

第三章插入主要函数之间的相互调用图，也是为后期协议函数分析理清思路做铺垫。

第四章也是最核心的一节，在这里我会详细介绍DSR协议是如何完成一个数据包发送接收应答以及错误维护的。各个函数之间如何相互调用也都会在这部分给出详细解释。有了第3节中数据结构的定义分析，在这里的分析我们将不在陌生。

第五章撰写本课程的一点总结心得。

参考RFC文档Internet-Draft and is subject to all provisions of Section 10 of RFC 2026.可以从这个网址中获取或者在我的GitHub仓库中得到这个文档。

网址：<http://www.ietf.org/shadow.html>

Repository：Sma-yida/201692204

# 第一章 基本原理

## 1.1 DSR协议简介

DSR是Dynamic Source Routing的缩写，本文中出现的所有“DSR”也将代指这个动态源路由协议（The Dynamic Source Routing protocol）的一个实现“dsr-uu协议，讨论的情况也是在网络节点使用动态源路由协议的基础下进行的。

动态源路由协议（DSR）是一个设计给多跳无线自组织网络Ad Hoc中移动节点的简单高效的路由协议。网络中的节点通过合作互相合作转发数据包使得一个节点能够和不在其无线网络覆盖范围直接可达的其他节点“跨越多跳”进行通信。同时该协议也是一种经典的按需路由协议。换言之，只有当一个节点需要发送数据包时，才会进行“路由发现”和“路由维护”过程。

## 1.2 DSR协议组成

DSR是一个被设计成能够快速应对网络变化的协议，它主要由两部分组成：

（1） 路由发现（Route Discovery）：该机制是指节点S发送给节点D的数据包中包含源路由信息，用于寻找到达D的路径。路由发现只在节点S不知道如何到达节点D时使用。

（2） 路由维护（Route Maintenance ）：该机制是指当S使用源路由给D发送数据包时，如果当网络的拓扑结构发生改变，无法再使用当前的源路由时，S能 够发现，并且使用其他已经发现的到D的路由或者发起路由发现。路由维护只在S给D实际发送数据包时使用。

这两种操作都是按需求产生的，DSR不需要发送任何周期性的数据包。

DSR维护的所有状态都是“软状态”，也就是说任何状态的丢失都不会影响

到协议的正常工作，任何必要的状态在失败后都可以很轻松地被重新发现。

举一个简单的例子，DSR中的一个节点在重启后能够很快重新加入DSR网络，并且恢复转发数据包的功能，只对路由协议造成最低的影响，甚至没有影响。

### 1.2.1路由发现

当某些节点产生了一个新的数据包，想要发送给目的节点时，源节点会将源路由放在数据包的头部，给出该数据包到目的节点所遵循的有序跳数。

通常情况下，发送方会从自己的路由缓存中获得一个合适的源路由；如果没有在缓存中找到路由，它就会启动路由发现协议动态地找到一个到达目的节点的新路由。

具体过程如下：

* 路由请求：源节点向邻居节点广播路由请求（RREQ）到达目的节点的路由；
* 路由记录：记录从源节点到目的节点中的中间节点请求ID，中间节点接收RREQ后，将自己的地址附在路由记录汇总，路由记录有addrs[0]数据维护源路由路径；
* 中间节点处理：中间节点维护序列对列表：<源节点地址、请求ID>；
* 重复RREQ检测：如果检测到重复，则中间节点丢弃该RREQ消息；
* 路由应答：目的节点收到RREQ后，给源节点返回路由应答（RREP）信息，拷贝RREQ消息中的路由记录并反向存入addrs[]数组，源节点收到RREP后，在本地路由缓存中缓存路由信息。

### 1.2.2路由维护

路由维护会在以下情况下启动：

* 在源节点向目的节点发送数据时监控当前路由的可用情况；
* 当网络拓扑变化导致路由故障时，切换到另一条路由或者重新发起路由发现过程。

通过逐跳验证机制来判断路由错误信息，从而删除失效的路由：

如果数据分组被重发了最大次数仍然没有收到下一跳的确认，则节点向源端发送路由错误信息，并指明中断的链路，源端将该路由从路由缓存中删除；如果源端路由缓存中存在另一条到目的节点的路由，则使用该路由重发分组，否则重新开始路由发现过程。

### 1.2.3路由缓存

* 当源节点接到RREP分组后，就将RREP分组中从源节点到目的节点的路由信息缓存；
* 当源节点向目的节点发送数据分组时，此路由信息就包含在每个分组的头部；
* 所有的中间节点利用源路由信息进行分组转发。

与AODV按需路由协议的比较：

AODV协议通过向邻居节点周期性地发送hello报文来声明自己的存在，而DRS协议相当于增加了两部分开销来确保路由的有效性：1）增加DSR选项头，其中的源路由选项用来维护路由信息；2）增加路由缓存，来缓存节点到达目的节点的源路由。

## 1.3 通信过程简介

节点同过互相转发数据包来实现通信，而一个采用DSR协议的节点产生

并发出的任何一个数据包都必须经过以下四个步骤：

（1）在本节点的路由缓存中搜索是否有目标地址的字段。

（2）如果没有在路由缓存中发现对应的路由，就进行路由发现。而这一操作的实现方式就是在这个数据包的DSR选项头（DSR Options header）中添加一个路由请求选项（Route Request option）。而且这会导致这个数据包的目的地址被替换成广播地址（255.255.255.255）

（3）从第二点看来，如何一个数据包的DSR选项中没有路由请求，那么这个节点一定拥有到达目的IP地址的路由。如果有大于一个的可用路由，那么节点将会选择其中一个。如果路由长度大于一跳或者本节点想从下一个节点收到一个网络层的确认（ACK）,那么就要插入一个DSR选项头。

（4）根据选定的源路由向下一跳传输数据包，并使用路由维护判断下一跳是否可达。

通信过程的简易流程图如下：

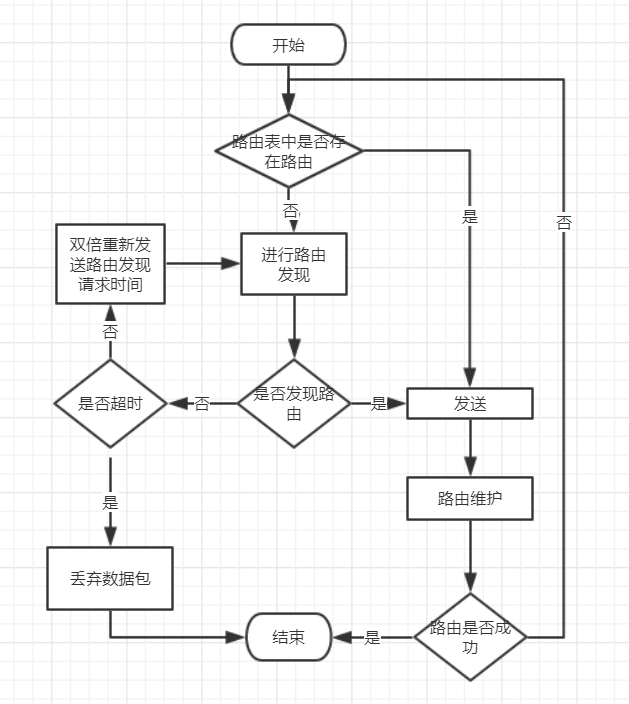


图1.3 通信过程流程图

# 第二章 数据结构

## 2.1 DSR选项头与选项关系

通过在1.3节中的介绍，我们不难发现在DSR协议中如果想要和其他节点进行通信，就需要路由发现，而路由发现就必须要通过在DSR选项头附加DSR路由选项来实现。由此可见DSR选项头和DSR选项是此协议实现的基础，要继续对协议进行分析，就必须首先对这两者进行分析。

图2.1显示了DSR选项头和四个路由选项的关系，DSR选项头通过加入不同的选项来指明DSR数据包的类型，下面将结合具体代码分别介绍它们。

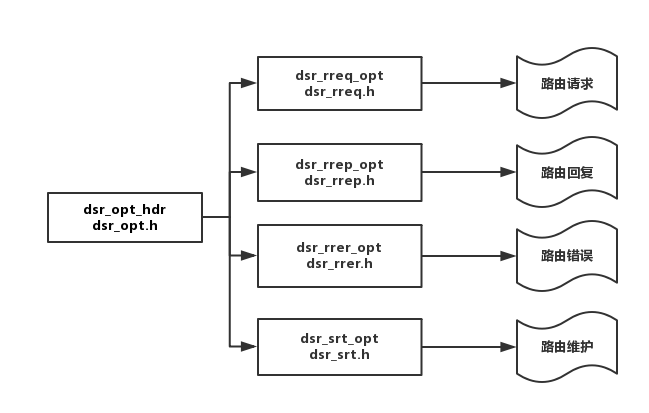


图2.1 DSR选项头与选项关系

## 2.2 DSR选项头分析

DSR通过在IP包中附加一个特殊的结构来携带控制信息，而这个结构正是DSR选项头。DSR选项头包含头部固定的4个字节数据和接下来0个或多个DSR选项组成。对于ipv4，DSR选项头必须紧跟在IP头的后面。在dsr\_opt.h头文件中定义了DSR选项头的信息，DSR选项头的结构体名字是“dsr\_opt\_hdr”，具体代码如图3所示。



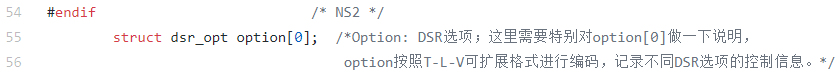


图2.2.1 dsr\_opt\_hdr定义（dsr\_opt.h）

代码分析：

|  |  |  |
| --- | --- | --- |
| 行号 | 具体代码 | 代码解释 |
| 00029 | u\_int8\_t nh | Next Header：下一报头 |
| 00032 | u\_int8\_t res:7 | Reserved：保留位，小端模式占7位 |
| 00033 | u\_int8\_t f:1 | Flag：标志位，小端模式占1位 |
| 00040 | u\_int16\_t p\_len | Payload length：负载长度 |
| 00055 | struct dsr\_opt option[0] | Option: DSR选项 |

表2.2.1 dsr\_opt\_hdr代码分析

通过分析上述代码，我们可以得出DSR选项头的格式如下：

|  |  |  |  |
| --- | --- | --- | --- |
| 字节数： 1 | 1 | | 2 |
| Next Header | Reserved（7 bits） | Flag（1 bit） | Payload length |
| Options(TLV格式：type-length-value) | | | |

表2.2.2 DSR选项头格式

这里要重点说一下option这一个数组，DSR选项就存储在其中（准确来说是存储在这个数组所指向的内存）。option按照T-L-V可扩展格式进行编码，记录不同DSR选项的控制信息。要注意虽然数组大小是0，但这并不意味着这个数组没有意义。因为把数组的大小定义为0只是C语言中一种比较方便地给结构体赋予变长空间的技巧，通过在给指向这个结构体的指针分配大于它本身的内存时(sizeof(struct dsr\_opt\_hdr))，而那些多出的内存的首地址就是option数组的首地址，使用option数组就可以访问。

而且在释放内存时也只需要释放指向结构体的指针即可，如果使用一个指针代替option数组，在释放内存时就必须先释放这个指针指向的内存，再释放结构体指针指向的内存，操作更加复杂，增大了内存泄露的风险，更何况一个指针变量本身也要占用空间，而一个大小为0的数组却不会。

在dsr-opt.h中的一条宏定义可以帮助我们获取option数组的内容，他的定义如下：



图 2.2.2 dsr-opt.h中关于获取option数组的定义

其中DSR\_OPT\_HDR\_LEN是另一条宏定义，它等同于sizeof(dsr\_opt\_hdr),把他和opt\_hdr的首地址加起来，就指向了option数组的首地址，而数组此时指向的地址是已经被分配了内存的，因此使用起来不会引发错误。

## 2.3 DSR选项通用头分析

各种不同的控制信息采用不同的控制选项，而不同的选项拥有不同的结构，并把这些结构存储在option数组中。但是纵使不同选项的结构各不相同，他们却有一个相同的头部用来区分类型和标明结构的长度，我们把他称为DSR选项通用头，其定义如下：

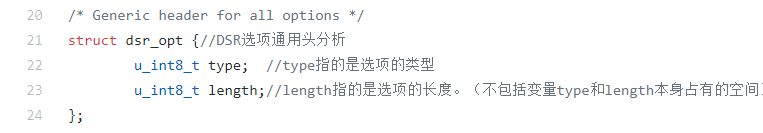


图 2.3 dsr-opt.h中关于选项通用结构的定义

很容易看出，type指的是选项的类型，而length指的是选项的长度。（不包括变量type和length本身占有的空间）。

## 2.4 DSR路由请求选项

在正是开始我们对路由算法的具体分析之前，还要对几个重要的路由选项的数据结构进行分析，这些选项控制着整个路由算法的进行，因此想要理解DSR路由算法，首先就必须要对于这些选项的构成进行了解。

而路由请求选项又是在进行路由发现时必不可少的一个选项，当进行路由发现时，发起端需要把路由请求选项添加到DSR选项头。

在dsr\_rreq.h头文件中定义了路由请求选项，具体代码如图6所示。

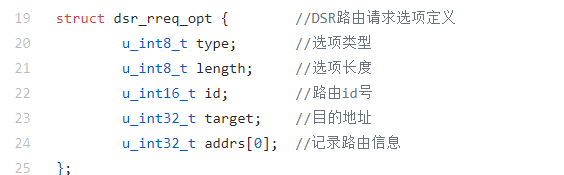


图2.4 dsr\_rreq\_opt定义（dsr\_rreq.h）

代码分析：

|  |  |  |
| --- | --- | --- |
| 行号 | 具体代码 | 代码解释 |
| 00020 | u\_int8\_t type | 选项类型 |
| 00021 | u\_int8\_t length | 选项长度 |
| 00022 | u\_int16\_t id | 路由id号 |
| 00023 | u\_int32\_t target | 目的地址 |
| 00024 | u\_int32\_t addrs[0] | 记录路由信息 |

表2.4 dsr\_rreq\_opt代码分析

成员type与length是通用选项头结构,不多加赘述。

下面对路由id号和记录路由信息的addrs[ ]数组以及target进行详细说明：

**ID:**

id是路由请求的发起节点（源端）所产生的唯一的一个值。每当节点发起一次新的路由请求时，就会为这次路由请求赋予一个唯一的id。这个id使得节点能够识别它最近是否转发过这个路由请求，如果一个节点在它的缓存的路由请求表中发现了相同的id，那么它将直接丢弃这个数据包以防止多次转发。如果没有发现相同的id，怎转发该数据包并把id记录到缓存中的路由请求表中。

**target:**

路由请求想要到达的目标节点地址。

**Address[1..n]:**

Address[i]表示路由请求选项的第i个记录，要注意源端地址在IP头中就可以获取，因此不应该再被记录到Address中。Address[1]表示从源端开始路由之后到达的第一个节点，之后以此类推。

一旦选择添加路由请求选项，IP头中的目的地址就会被修改为广播地址（255.255.255.255）。而且负责转发路由请求选项数据包的节点不能修改数据包的源地址。另一个需要注意到点是一个路由请求选项不能在DSR选项头中出现超过一次。

## 2.5 DSR路由回复选项

当路由发现发现目的节点之后，需要发送路由回复选项用于告诉路由请求的发起者到达目的节点的路由。

在dsr\_rrep.h头文件中定义了路由回复选项，具体代码如图2.5所示。

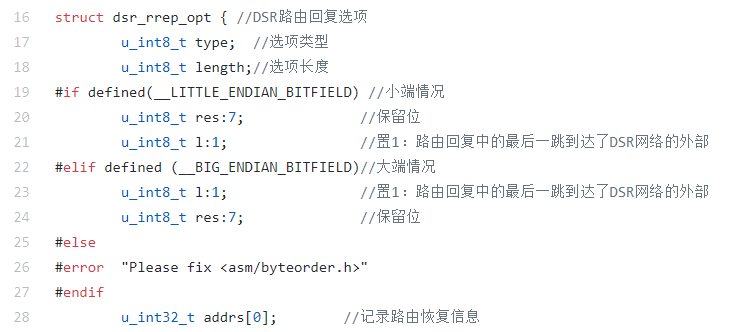


图2.5 dsr\_rrep\_opt定义（dsr\_rrep.h）

代码分析：

|  |  |  |
| --- | --- | --- |
| 行号 | 具体代码 | 代码解释 |
| 00017 | u\_int8\_t type | 选项类型 |
| 00018 | u\_int8\_t length | 选项长度 |
| 00020 | u\_int8\_t res:7 | 保留位 |
| 00021 | u\_int8\_t l:1 | 置1：路由回复中的最后一跳到达了DSR网络的外部 |
| 00028 | u\_int32\_t addrs[0] | 记录路由回复信息 |

表2.5 dsr\_rrep\_opt代码分析

这个结构体的定义对于大小端的情况做了不同的处理，但这里只针对各个成员变量的作用进行分析。

**Last Hop External (L)：**

代码第21行，l标志位代表Last Hop，置0表示最后一跳节点在DSR网络内部，置1表示最后一条到达了DSR网络的外部。

当它置1时，表明路由回复中的最后一跳到达了DSR网络的外部。DSR网络外部的路由不会再路由回复中被记录。节点在选择路由时，应该尽量选择这个标志位为0的路由。

在选取路由时应当遵循以下原则：尽量选取仅利用本网络内部节点就可以到达目的节点的路由，尽量不经过外部网络。即，在路由缓存中优先选择l标志位置0的路由。

**Reversed(res):**

保留位，置0。

**Address[1..n]：**

路由回复返回的源路由，从源点到目标节点按顺序排列。

一个路由回复选项可能在一个DSR选项头中出现一次或者多次。

## 2.6 DSR路由错误选项

路由错误选型用于通知路由的错误，

在dsr\_rrer.h头文件中定义了路由错误选项，具体代码如图2.6所示。



图2.6 dsr\_rrer\_opt定义（dsr\_rrer.h）

代码分析：

|  |  |  |
| --- | --- | --- |
| 行号 | 具体代码 | 代码解释 |
| 00022 | u\_int8\_t err\_type | 错误类型 |
| 00024 | u\_int8\_t res:4 | 保留位 |
| 00025 | u\_int8\_t salv:4 | salvage:该数据包被重发的次数 |
| 00033 | u\_int32\_t err\_src | 路由错误的源地址 |
| 00034 | u\_int32\_t err\_dst | 路由错误目的地址 |
| 00035 | char info[0] | 记录路由错误信息 |

表2.6 dsr\_rrer\_opt代码分析

成员type与length是通用选项头结构,不多加赘述。

**Error Type(err\_type)：**

置1则表明路由的第一条路由到达的是DSR网络的外部。

**Salvage(salv)：**

数据包被拯救重发的次数。salv标志位占4bits，其含义是salvage，用来表示这个数据包被重发的次数，根据这个数值来决定该数据包是否应该被丢弃。

**Resversed(res):**

保留位，置0。

**Segments Left(left):**

路由剩余的跳数

**Error Source(err\_src):**

路由错误源地址。

**Error Source(err\_dst):**

路由错误目的地址。

**Information(info):**

错误信息。

## 2.7 DSR路由维护选项

在dsr\_srt.h头文件中定义了路由维护选项，具体代码如图2.5所示。

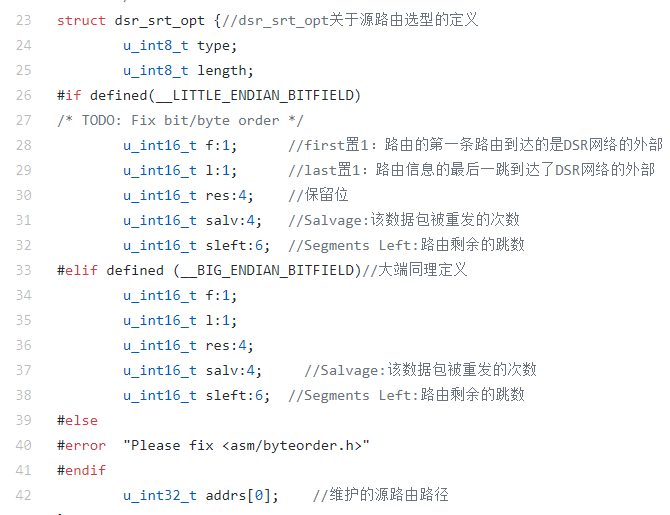


图2.7 dsr\_srt\_opt定义（dsr\_srt.h）

代码分析：

|  |  |  |
| --- | --- | --- |
| 行号 | 具体代码 | 代码解释 |
| 00028 | u\_int16\_t f:1 | 置1：路由的第一条路由到达的是DSR网络的外部 |
| 00029 | u\_int16\_t l:1 | 置1：路由信息的最后一跳到达了DSR网络的外部 |
| 00030 | u\_int16\_t res:4 | 保留位 |
| 00031 | u\_int16\_t salv:4 | Salvage:该数据包被重发的次数 |
| 00032 | u\_int16\_t sleft:6 | Segments Left:路由剩余的跳数 |
| 00042 | u\_int32\_t addrs[0] | 维护的源路由路径 |

表2.5 dsr\_rrer\_opt代码分析

路由维护选项，也可以被称为源路由选项，当一个节点想到达某一目的节点时，会从路由缓存中取得路由，并将源路由选项加入DSR选项头，具体存入数组addrs[ ]中，利用其中的路由信息到达目的节点。

成员type与length是通用选项头结构,不多加赘述。

**First Hop External (F)：**

置1则表明路由的第一条路由到达的是DSR网络的外部。

**Last Hop External (L)：**

作用同2.5节中路由回复选项的L。

**Resversed(res):**

保留位，置0。

**Segments Left(sleft):**

sleft标志位占4bits，其含义是segments Left，用来表示路由剩余的跳数。

**Salvage(salv):**

记录了这个数据包被拯救重发的次数。salv标志位占4bits，其含义是salvage，用来表示这个数据包被重发的次数，根据这个数值来决定该数据包是否应该被丢弃。

**Address[1..n]：**

源路由的路径。

# 第三章 函数关系图

## 3.1 总体流程图

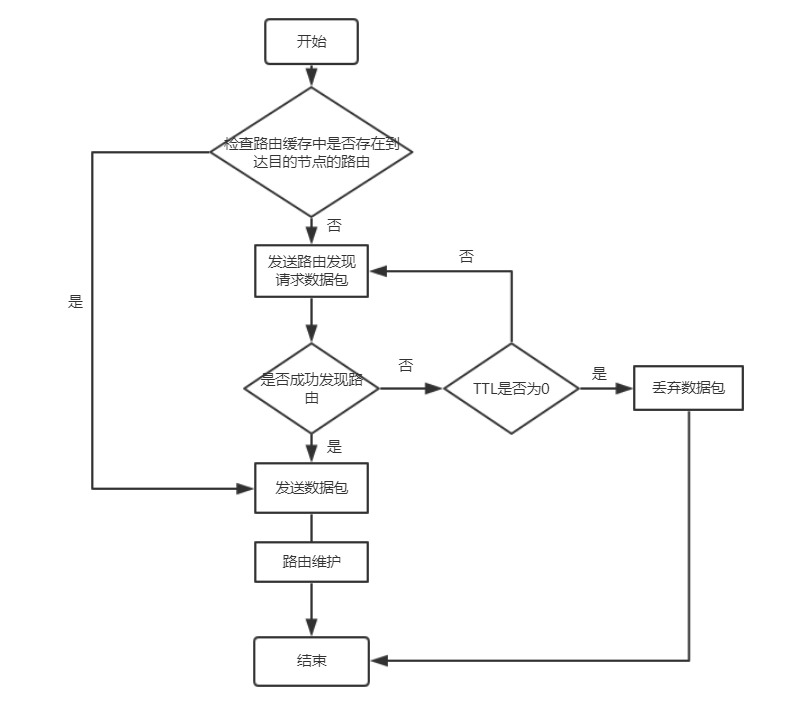


图3.1 总体流程图

在图3.1中，如第1.3节所讲，首先需要检查路由缓存中是否存在到达目的节点的路由，如果有，那么进行发送数据包，进行路由维护，然后结束；如果没有，则发送路由发现请求数据包，如果成功发现路由，那么发送数据包，进行路由维护，然后结束；如果没有，则会判断TTL是否为0，若不为0，则继续进行发送路由发现请求数据包；否则的话，丢弃数据包，然后结束。

## 3.2 函数关系调用图

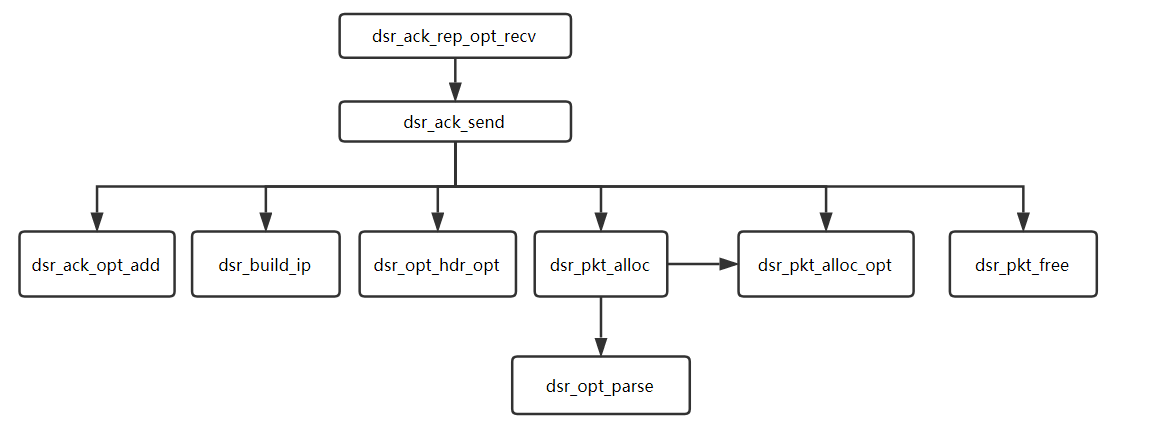


图3.2 dsr\_ack\_rep\_opt\_recv() 函数关系调用图

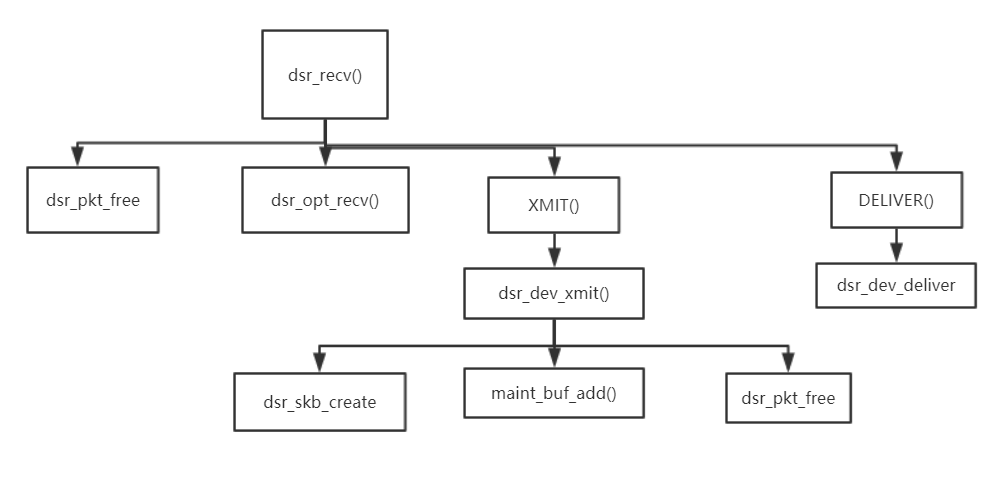


图3.3 dsr\_recv() 函数关系调用图

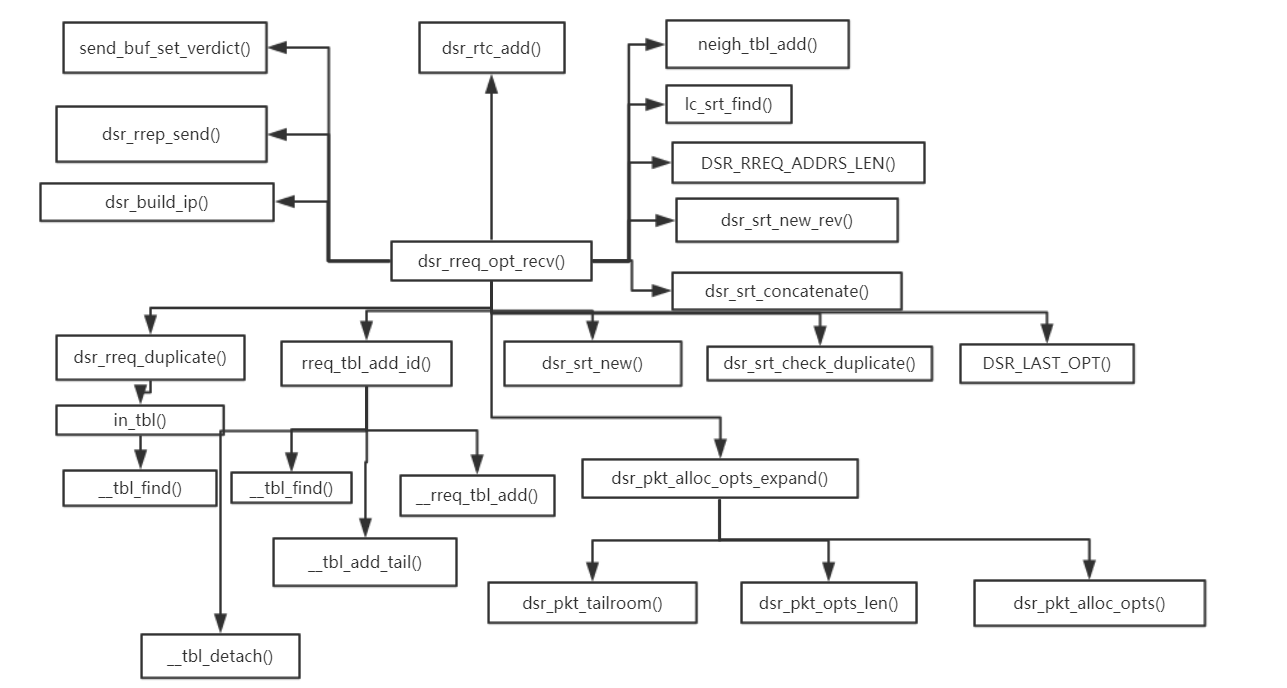


图3.4 dsr\_rreq\_opt\_recv() 函数关系调用图

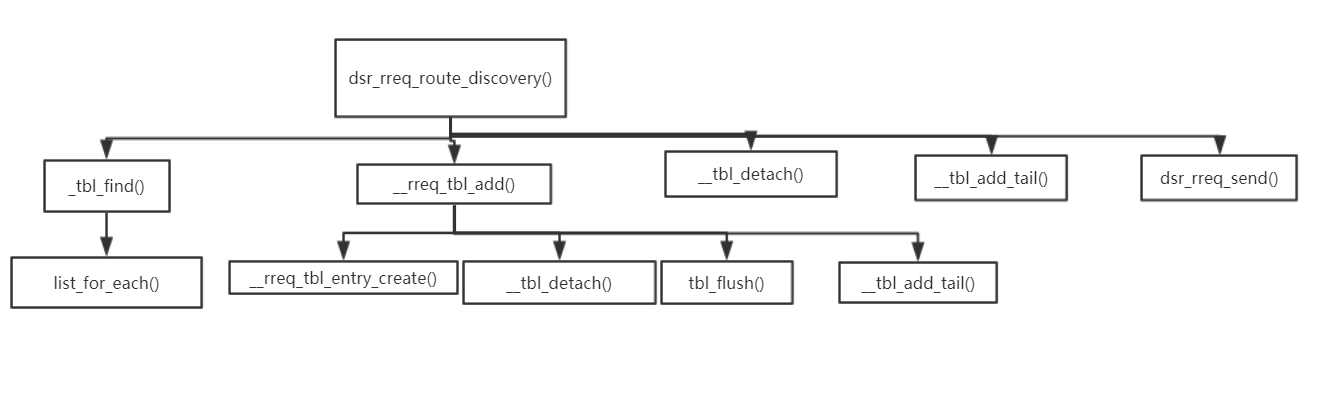


图3.5 dsr\_rreq\_route\_discovery() 函数关系调用图

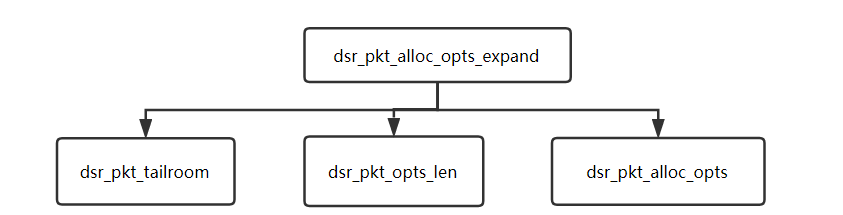


图3.6 dsr\_pkt\_alloc\_opts\_expand() 函数关系调用图

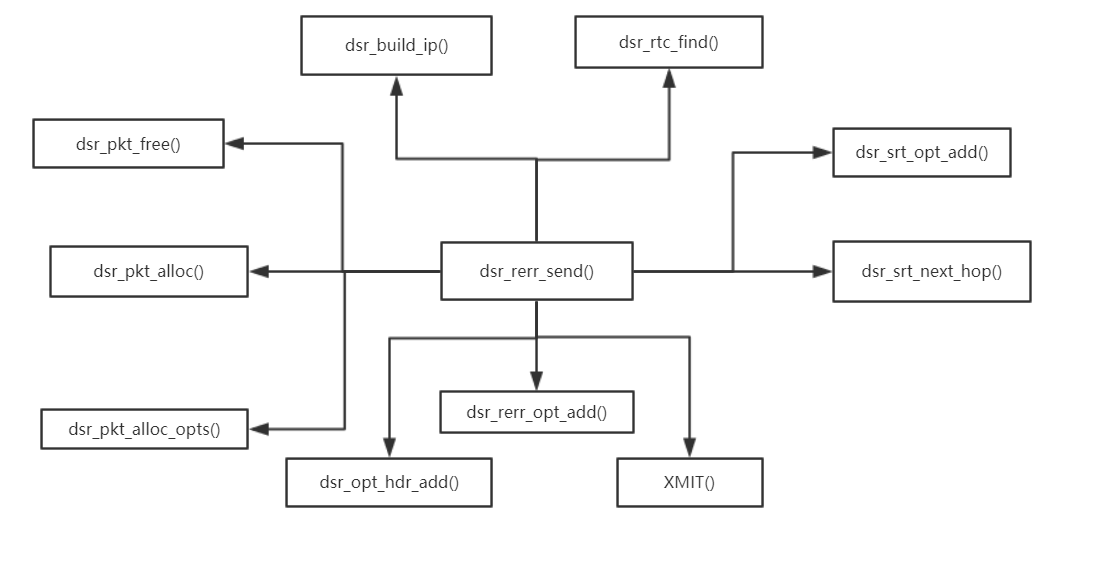


图3.7 dsr\_rerr\_send() 函数关系调用图

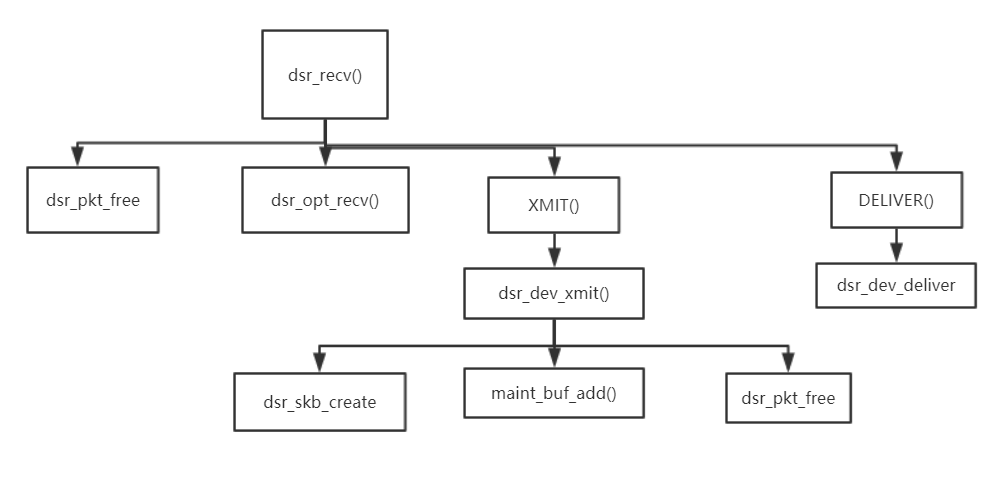


图3.8 dsr\_rrep\_send() 函数关系调用图

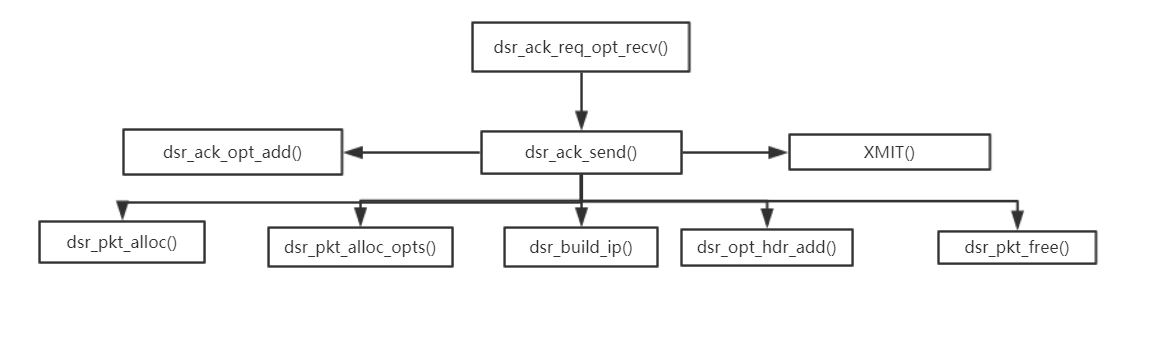


图3.9 dsr\_ack\_req\_opt\_recv () 函数关系调用图

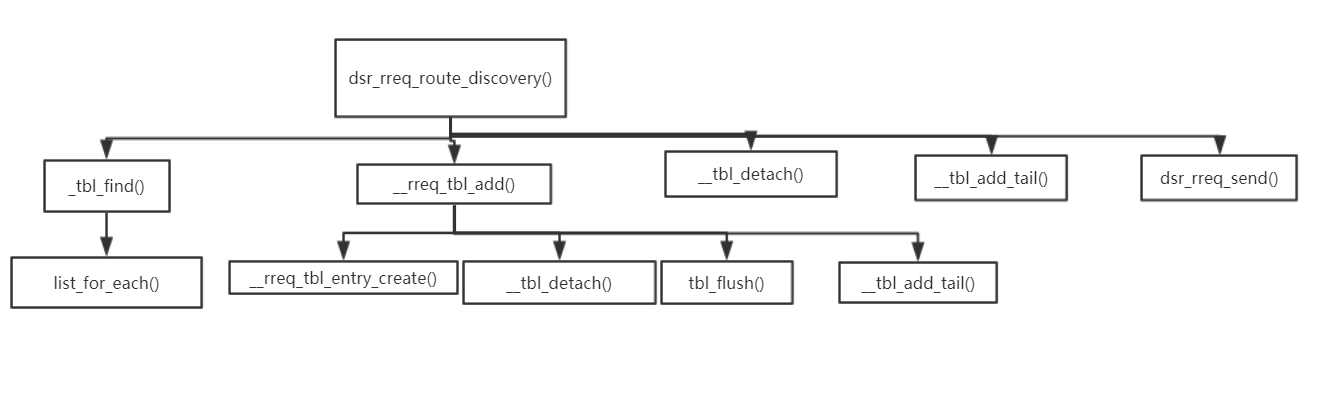


图3.10 dsr\_rreq\_send () 函数关系调用图

# 第四章 协议具体函数代码分析

在完成了对于DSR选项和DSR选项头等基础知识的铺垫以及对DSR协议的运行流程有了大致的了解之后，我们将正式开始对于协议的运行过程进行详细分析。

## 4.1 DSR数据包结构

为了更加方便地对DSR协议中的数据包进行管理，DSR协议对于协议中的数据包采用以下数据结构：

### 4.1.1 dsr\_pkt()数据包结构





图4.1.1 dsr\_pkt()数据包结构

我们对其中的重点成员进行分析。

**src:**

指向数据包源地址。

**dst:**

指向数据包目的地址。

**nh:**

指向网络层协议头（IP头），联合体使得我们能够使用结构体的方式访问IP头或者直接读取IP头的二进制数据。

**dh:**

指向DSR头，联合体的作用同上。

**rreq\_opt:**

指向路由请求选项。

在对数据包结构体的定义中（图4.1），在26-29行，分别定义了源端IP，目的端IP，下一跳IP和上一跳IP, 在32-52行，根据是否是在NS2平台进行协议模拟定义了可以访问IP协议头部的nh联合体，可以访问MAC协议头部的mac联合体，以及IP数据包的ip\_data。在53-59行，定义了可以访问dsr选项的结构体dh。在62-68行，分别定义了指向源路由选项、路由请求选项、路由回复选项、路由错误选项、确认选项、确认请求选项、源路由结构体的指针。在71-78行，定义了负载、负载长度及一个分片结构体。

而生成数据包的函数**dsr\_pkt\_alloc**的实现如下：

### 4.1.2 dsr\_pkt\_alloc()函数分析



图 4.1.2 dsr-pkt.c中关于生成数据包的实现dsr\_pkt\_alloc() 函数代码

在**dsr\_pkt\_alloc()**函数中，实现了生成数据包的功能，同样会根据是否是在NS2平台上进行模拟而分成两种情况，此处对NS2平台上的宏定义进行分析。

首先在89-90行，分别定义了一个数据包结构体和一个数据包的common头指针。

在92行，为数据包申请了一个内存空间，然后将其置0，方便后续初始化。

在96-125行，判断数据包非空的情况下，对其进行初始化操作，其中**address()**是接口函数，用来访问数据包的包头部分；

在105-122行中，会根据common头的类型执行不同的操作。

无论在96行判断的结果如何，最后都会在126行，返回dp结构体指针。

## 4.2 为DSR选项头分配内存

DSR选项头用于携带协议的控制信息，在一个DSR数据包中指针dh.opth指向了DSR选项头，当我们需要为数据包添加DSR选项头时，就需要为其分配内存，具体的实现如下：

### 4.2.1 dsr\_opt\_hdr\_add()函数分析



图 4.2 dsr-opt.c中dsr\_opt\_hdr\_add()函数关于为选项头分配内存的实现

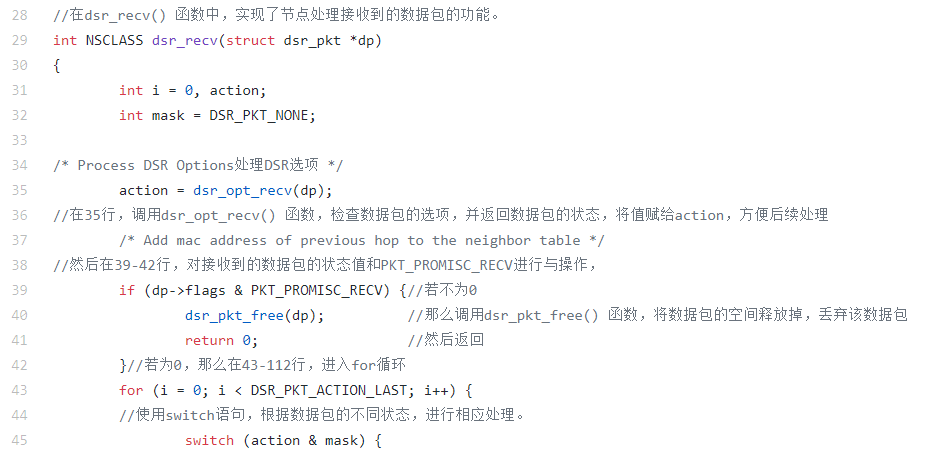
在**dsr\_opt\_hdr\_add()**函数中，实现了将dsr选项头添加到头部的功能，在30-31行，先判断长度是否能放下该选项头，然后将头指针指向该缓冲区，然后在35-38行，对选项进行初始化操作，并返回该指针。

## 4.3 协议IO分析

协议想要运行，IO自然是必不可少的一部分，有了输入和输出的能力，一个节点才能够和其他的节点进行交互。

我们先来讲一讲协议的输入，也就是DSR协议对于接受数据包的处理，他的具体实现在**dsr-recv()**函数中：

### 4.3.1 dsr-recv()函数分析





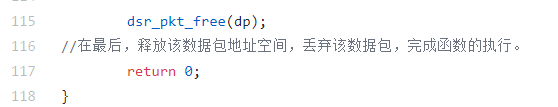


图4.3.1 dsr\_recv() 函数代码

在**dsr\_recv()** 函数中，实现了节点处理接收到的数据包的功能。

在35行，调用**dsr\_opt\_recv()** 函数，检查数据包的选项，并返回数据包的状态，将值赋给action，方便后续处理。

然后在39-42行，对接收到的数据包的状态值和PKT\_PROMISC\_RECV进行与操作，若不为0，那么调用**dsr\_pkt\_free()** 函数，将数据包的空间释放掉，丢弃该数据包，然后返回。

若为0，那么在43-112行，进入for循环，使用switch语句，根据数据包的不同状态，进行相应处理。

其中，在62-80行，如果是需要转发的数据包，会检查其TTL，若为0，则丢弃该数据包，否则打印源端IP，目的端IP和下一跳IP地址，然后执行XMIT()传输数据包。在最后，释放该数据包地址空间，丢弃该数据包，完成函数的执行。

总结：首先，协议会处理数据包的选项，并决定对数据包进行何种操作，之后会用一条switch语句对各种不同的选项进行判断，并进行处理，我们着重分析转发数据包的情况，当ttl为零时，丢弃数据包，而其他情况下调用XMIT函数（下节的分析中会详细介绍）转发数据包。

而**dsr\_start\_xmit()**函数很好的展示了节点在发送数据包时会发生什么。

### 4.3.2 dsr\_start\_xmit()函数分析



图4.3.2 dsr\_start\_xmit() 函数代码中关于发送数据包的处理

在**dsr\_start\_xmit()** 函数中：

首先在123-126行，进行判断是否属于数据包，若不属于则打印错误并退出；

然后在128-138行，进行寻找到目的节点的路由，若存在，则发送数据包；

在144-156行，若不存在则调用**send\_buf\_enqueue\_packet()**将数据包加入到发送缓存中，然后调用**dsr\_rreq\_route\_discovery()**函数，进行路由发现。

## 4.4 处理源路由请求

在DSR协议众多的操作之中，我们首先分析对于源路由选项的处理，因为这是整个协议中相当基础的一个操作——在知道路由路径的情况下如何将数据包发送到目标地址。

### 4.4.1 dsr\_srt结构分析

源路由在这里是指发送的数据包沿途经过的节点，也就是路由的路径。而在协议内部，使用结构dsr\_srt来描述源路由，它的定义如下：

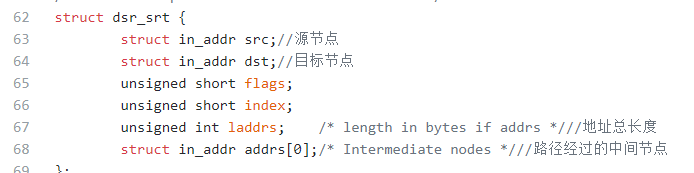


图 4.4.1 dsr-srt.h中关于源路由的路径

**Src:**

源节点。

**Dst：**

目标节点。

**Laddrs:**

地址总长度。

**Addrs[1..n]:**

路径，经过的中间节点。

### 4.4.2 dsr\_srt\_opt\_add()函数分析

想要进行源路由第一步自然就是要为数据包添加一个DSR源路由选项（具体格式见2.7节），具体的代码实现如下：



图 4.4.2 dsr-srt.c中dsr\_srt\_opt\_add()函数关于添加源路由选项的实现

源路由请求是指，源路由的用户可以指定他所发送的数据包沿途经过的部分或者全部路由器。

在进行源路由的第一步，需要向数据包的头中加入一个选项。和**dsr\_opt\_hdr\_add()**函数类似：

首先在299行判断缓冲区的长度是否足够，其长度是否可以容纳的下该选项长度；

然后在302行，将选项的指针指向该缓冲区。

如果足够，在304-310行，初始化相关的类型，长度，标志位并且复制地址列表。

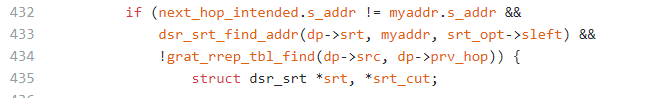
在312行对地址列表进行复制。最后在314行，返回该指针。

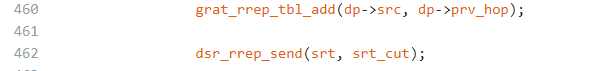
至此，源路由选项添加完成。

### 4.4.3 dsr\_srt\_opt\_recv()函数分析

源路由一个基本的功能就是转发数据包，但是另一个值得一提的功能就是路由缩减，也就是缩短路由路径。当节点监听到一个数据包但是该数据包的下一跳地址不是它自己时，节点就会在这个数据包的剩余源路由中搜索是否有自身的地址，如果存在，意味着路由能够缩减，这一段逻辑的实现代码如下：

srt.c-recv1





srt.c-recv4

图 4.4.3 dsr-srt.c中dsr\_srt\_opt\_recv()函数关于源路由选型的处理

第432-435行用于判断是否可以经行路由缩减，第460-462行代码用于在缩减列表中添加本次路由缩减并且发送一个路由回复用于通知新的路由变更。如果不能进行路由缩减，则转发数据包（第484行）。

## 4.5 发出路由发现请求

当一个节点不知道应该如何到达目标节点时，就会采用路由发现来寻找路由。

### 4.5.1 rreq\_tbl\_entry 结构定义分析

在开始分析这个过程的具体实现代码之前，首先要对一些数据结构进行介绍，首当其中的就是路由请求表，它的定义如下：

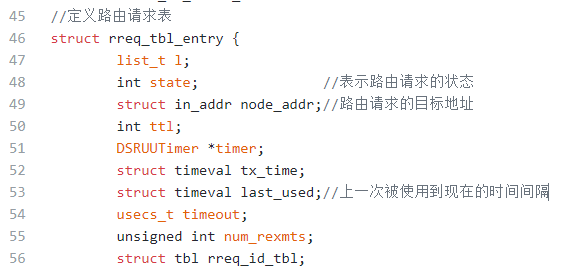


图 4.5.1 dsr-rrep.c中路由请求表的定义

我们对其中重要的成员进行分析：

**node\_addr:**

路由请求的目标地址。

**state:**

表示路由请求的状态。

**last\_used:**

上一次被使用到现在的时间间隔。

在了解了这一点后，我们来看路由发现的具体实现，它的代码实现如下：

### 4.5.2 dsr\_rreq\_route\_discovery() 函数分析





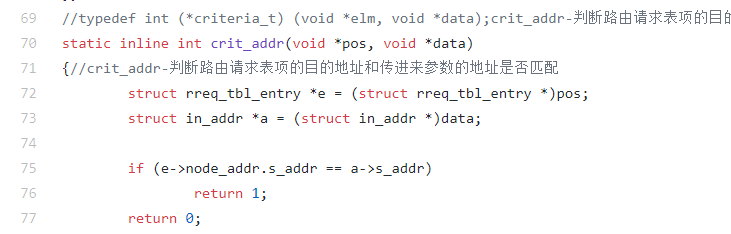
图 4.5.2 dsr-rreq.c中dsr\_rreq\_route\_discovery() 函数关于路由发现的实现

在dsr\_rreq\_route\_discovery() 函数中，实现了发送路由请求，进行路由发现的功能。从代码中可以很清晰的看出路由发现的运行逻辑。

首先，在344-346行定义一个指向路由请求表的指针；在路由请求表的结构体中，存放着路由求情的目的地址，当前状态，以及关于时间的一些变量；还定义了TTL，控制缓存失效期限的变量expires。

在352行，会调用\_tbl\_find()函数对当前目标在路由请求表中先进行搜索。在\_tbl\_find()中，是对链表中的数据进行遍历，如果发现满足条件的节点，则返回指向该结点的指针，否则返回NULL。\_\_tbl\_find()函数的实现如下：

### 4.5.3 \_tbl\_find()函数分析



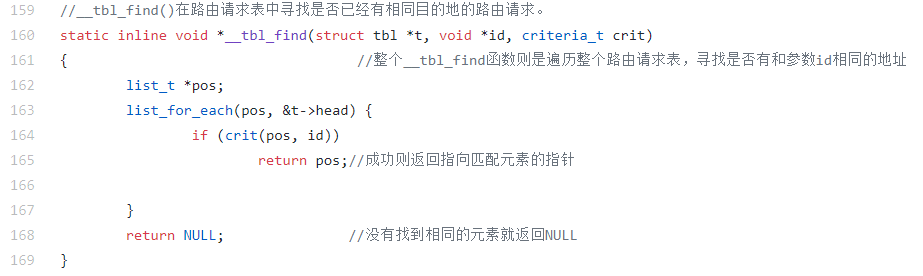


图 4.5.3 tbl.h中\_\_tbl\_find的定义和dsr\_rreq.c中crit\_addr的定义

我们发现criteira\_t是一个参数为两个void型指针，返回值为int的函数指针，而crit\_addr则是在路由发现的过程中传递给该参数的值，从图4.5.3 第75行代码可以看出它用于判断路由请求表项的目的地址和传进来参数的地址是否匹配。

而整个\_\_tbl\_find函数则是遍历整个路由请求表，寻找是否有和参数id相同的地址成功则返回指向匹配元素的指针，没有找到相同的元素就返回NULL。

回到dsr\_rreq\_route\_discovery()函数中,在355-356行中，若没有找到在路由请求表中相同的路由请求，那么就调用\_rreq\_tbl\_add()函数，把target添加到路由请求表中。

### 4.5.4 \_rreq\_tbl\_add()函数分析



图4.5.4 \_rreq\_tbl\_add()函数代码

在244行，首先会调用\_rreq\_tbl\_entry\_create()函数，对表项进行内存分配，然后初始化项目值，然后返回指向路由请求表的指针。

在249-265行，会对请求表进行判断，检查表中条目是否已满，如果已经满了，就删除表中的第一条请求。

在266行，调用\_tbl\_add\_tail()函数，将新的请求添加到请求表的最后，然后返回指针。

清楚了这几点之后再看图4.5.2的dsr\_rreq\_route\_discovery()函数中代码就很清晰了：

第355-356行，表明了如果没有相同的路由请求，就把该请求添加到路由请求表中，如果有，则删除原先的路由请求，并把新的路由请求添加到路由请求表的末尾。

在367行，检查state的值，如果表示“当前状态已经在路由发现中”，那么打印信息，然后函数执行结束；

若不是，将在374-390行，对时间相关的变量进行设置。最后返回res结束还是的执行。

添加路由请求表项的过程的具体实现如下：

### 4.5.5 rreq\_tbl\_entry\_create()函数分析

rreq.c-creat



图4.5.5 dsr-rreq.c中rreq\_tbl\_entry\_create()函数关于创建路由请求选项的实现

当调用\_\_rreq\_tbl\_add时，首先就会调用rreq\_tbl\_entry\_create()函数来分配给表项分配一段内存并把这个表项初始化，成功之后返回指向这个表项的指针。

接下来我们看图4.5.5的第249行代码，在返回到\_\_rreq\_tbl\_add()函数之后，会先判断当前的路由请求表是否已经满了；

如果已经满了，第258-260行，那么就会将请求表中的第一个表项删除，因为其等待的时间最长，最先进入队列却还停留在请求表中。在完成这个操作之后就会把这个路由请求添加到请求表的结尾。

之后我们再次回到4.5.2节dsr\_rreq\_route\_discovery()函数中。接下来的代码是关于请求超时的一些设置。第375行把本次路由请求选项的上次使用时间设置成当前事件。第380-383行把路由请求选项设置了请求的超时时间，一旦超出了这段时间路由发现还没有完成，这一路由请求就会被取消。第389-391行设置失效时间expires。

在完成了对时间的相关步骤之后，就到了最后一步——发送请求了，dsr\_rreq\_send()函数负责了这一任务，他的实现如下：

### 4.5.6 dsr\_rreq\_send()函数分析



图 4.5.6 dsr\_rreq.c中dsr\_rreq\_send()函数关于发送路由请求的实现

dsr\_rreq\_send()实现了发送路由请求的功能。

第445行，首先，在发送前，会先申请分配一个DSR数据包。

第451到第453行，之后会把这个数据包的目的地址设置成广播地址，并把源地址设置成自身的地址,将数据包的目的地址和下一跳地址都定义成广播地址，将源地址调用my\_addr()函数，设置为自身地址；

第455行，同时为DSR选项分配相应的内存。

第461-463行，如果这几步都顺利完成的话，就构造数据包的IP头，包括源地址，目标地址，协议类型和负载长度等信息。

第469行，在完成IP头的构造之后就是DSR选项头的构造。



图 4.5.6 dsr\_rreq.c中关于发送路由请求的实现dsr\_rreq\_send()函数代码

完成之后自然就是路由请求选项的添加，它的实现如下：这一段代码和添加DSR选项头的代码大致相同，主要是判断缓冲区的空间是否足够，如果足够，就把选项指针指向缓冲区，并初始化id、长度和目的地址。

总结：为选项申请内存地址并检查是否申请成功，若失败，则调用dsr\_pkt\_free()函数，将该数据包的空间释放，丢弃该数据包，然后退出函数；若成功，则调用dsr\_build\_ip()函数，构造数据包的头部，包括源端IP地址，目的端IP地址，TTL等信息，此处若构造失败，则丢弃该数据包并退出函数。然后在469行，构造选项头，初始化信息，并检查是否构造成功，若失败就打印信息并丢弃数据包；若成功，则调用dsr\_opt\_hdr\_add()对路由请求选项进行添加，同样需要检查是否添加成功。在492行，调用XMIT()函数（实际是调用dsr\_dev\_xmit()函数），进行数据包的发送。

### 4.5.7 dsr\_dev\_xmit()函数分析





图 4.5.7 dsr-dev.c中关于数据包发送的实现 dsr\_dev\_xmit()函数代码

整个发送流程十分清晰，简而言之，首先是要根据dsr数据包的数据创建一个sk\_buf数据包，因为发送的数据包的格式必须为sk\_buf格式。在完成创建后接着完成MAC头的封装并检查其合法性（第514-518行），最后一步在530行调用dev\_queue\_xmit(skb)将数据包发送出去并统计发送的数据包信息（第535到538行）。

所有流程顺利结束的话，将返回0。如果有错误产生，将会跳转至标签out\_err处，我们不再需要这个数据包，因此将它释放（第541行）并返回-1。

## 4.6 处理路由发现请求

当一个节点收到的数据包中包含路由发现请求时，这个节点必须通过以下步骤处理：

1. 如果路由请求的目的地址的自己的IP地址吻合，那么这个节点就需要发出一个路由回复给路由发现的发起者。
2. 如果路由请求的目的地址的自己的IP地址不同，那么节点就需要在自己的收到路由请求缓存中搜索是否已经发出过相同的请求，如果已经发出过，那么就必须丢弃这个数据包以防止重复转发。
3. 1. 在最近收到的路由请求缓存中添加这个请求

2. 将自己的IP地址添加到路由请求选项的Address数组中，并更改选项的length值。

3. 在自己的路由缓存中搜索是否有到达目标节点的路由，如果有，则发送“缓存路由回复”给起始节点。

4. 如果没有发现，则以链路层广播的形式发送这个修改后的数据包。

仔细阅读上面的步骤，我们不难发现对于路由请求的处理就是判断是否因该生成路由回复和如何生成路由回复。而路由回复的生成方式有两种，一种是自己不是目标节点，但是有到达目标节点的路由，通过缓存的方式产生的路由回复。另一种则是自己就是目标节点而产生的路由回复，这一段逻辑的具体实现在dsr-rreq.c的**dsr\_rreq\_opt\_recv()**函数中，在此截取其中重要部分进行分析：

### 4.6.1 dsr\_rreq\_opt\_recv()函数分析



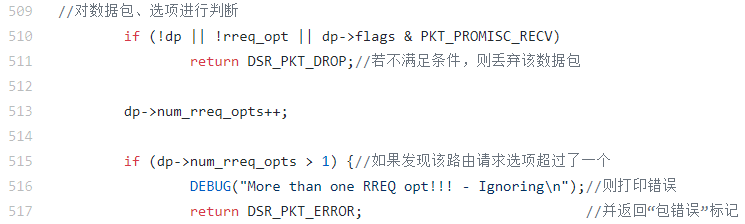




图4.6.1 dsr-rreq.c中dsr\_rreq\_opt\_recv()函数关于处理收到的路由请求的实现

**dsr\_rreq\_opt\_recv()**函数实现了接收处理路由发现请求选项的功能。

在510-511行，对数据包、选项进行判断，若不满足条件，则丢弃该数据包。

在515-517行，如果发现该路由请求选项超过了一个，则打印错误并返回“包错误”标记。

在525-528行，调用dsr\_rreq\_duplicate()函数，检查当前节点是否收到过该请求，若收到过，则打印错误并丢弃该数据包；

在531行，若未重复收到，则调用rreq\_tbl\_add\_id()函数，在路由请求表中添加该请求。

在533-534行，调用dsr\_srt\_new()函数，根据当前节点的路由，建立源路由。然后进行判断。

在540-542行，打印RREQ的信息。

在546-550行，调用dsr\_srt\_new\_rev()函数，根据当前路由，寻找逆向源路由来进行路由回复，接下来进行判断并打印信息。



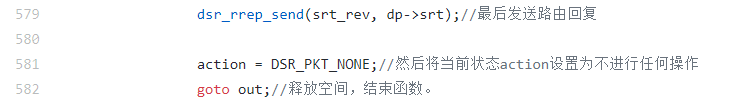
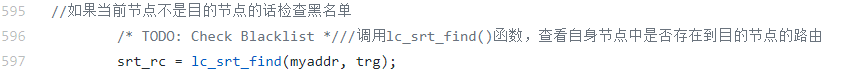


图4.6.1 dsr-rreq.c中dsr\_rreq\_opt\_recv()函数关于处理收到的路由请求的实现（续）

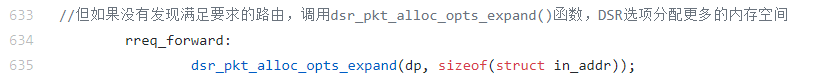
在558-570行，设置上一跳信息，然后发送缓存包。

在568-582行，进行帕努单，如果目标节点就是当前节点的话，打印信息，然后根据是否在NS2平台进行模拟来设置，最后发送路由回复，然后将当前状态action设置为不进行任何操作，释放空间，结束函数。



在596行，如果当前节点不是目的节点的话，调用lc\_srt\_find()函数，查看自身节点中是否存在到目的节点的路由。





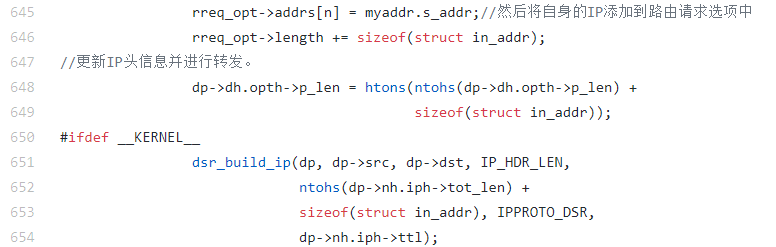


图4.6.1 dsr-rreq.c中dsr\_rreq\_opt\_recv()函数关于处理收到的路由请求的实现（续）

在605行，如果发现了对应的路由就调用dsr\_srt\_concatenate()函数将路由连接起来，然后返回路由回复。但如果没有发现满足要求的路由，就在635行调用dsr\_pkt\_alloc\_opts\_expand()函数，DSR选项分配更多的内存空间，然后将自身的IP添加到路由请求选项中，更新IP头信息并进行转发。

**总结：**判断是否收到过重读的路由请求。如果没收到过，就在路由请求表中添加这个请求,否则将其丢弃，并更新该数据包的源路由建立逆向源路由以便于发送回复。如果路由请求的目的地址等同与自身的地址，就发送路由回复，回复的选项为路由回复选项，具体数据格式见1.4.4节。如果路由请求的目的地址等同与自身的地址不同，那么就在自己的路由缓冲中搜获是否有到目的地的路由。发现对应路由就够在新的源路由，发送路由回复请求。如果没有发现，为DSR选项分配更多的内存，并且将自身的ip添加到路由请求选项之中，更新IP头信息后进行转发。

至此为止，路由发现的部分已经全部分析完成。接下来我们将要对路由维护的部分进行详细分析。

## 4.7 确认请求（ACK Request）选项处理

这个选项十分简单，就是在DSR通用选项头后面附加一个ID。DSR协议使用这个选项来判断邻居节点是否存活。当给一个邻居节点重发多次ACK请求却任然没有收到回复的话，节点就会认为它的这个节点的链路已经被破坏，从而在路由缓存中删除这条路由，并且给上一次收到ACK之后在这条链路上发送过数据包的节点发送路由错误。

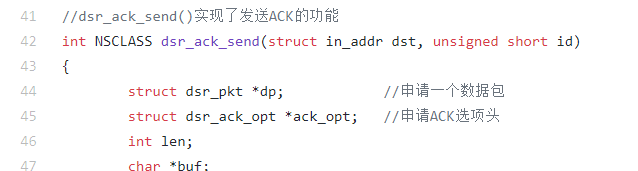
### 4.7.1 dsr\_ack\_req\_opt\_recv() 函数分析

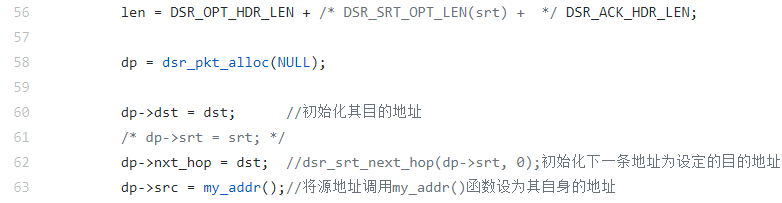


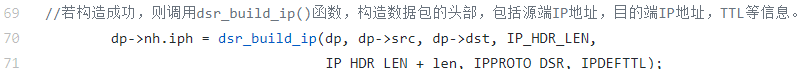
图 4.7.1 dsr-ack.c中关于ACK请求选项的处理dsr\_ack\_req\_opt\_recv() 函数代码

在dsr\_ack\_req\_opt\_recv() 函数中，首先在267-268行进行判断，如果不满足条件就返回包错误；然后使用ntohs()函数给id赋值，然后将该id添加到DSR通用选线头的后面，如RFC文档中那样。经过上面的分析，当一个节点收到ACK请求时会发生什么就很简单了，当然就是给请求方会发一个ACK，在280行，执行dsr\_ack\_send()函数，将回复请求方一个ACK数据包。

### 4.7.2 dsr\_ack\_send()函数分析











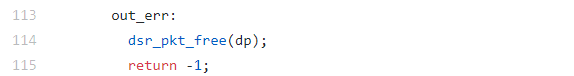


图4.7.2 dsr\_ack\_send() 函数代码

**dsr\_ack\_send()**实现了发送ACK的功能。

首先在44-45行申请一个数据包和ACK选项头；

从56行开始对数据包进行初始化，初始化其目的地址和下一条地址为设定的目的地址，将源地址调用my\_addr()函数设为其自身的地址。

第70行，若构造成功，则调用dsr\_build\_ip()函数，构造数据包的头部，包括源端IP地址，目的端IP地址，TTL等信息。

第113-115行，若失败则调用dsr\_pkt\_free()函数，将该数据包的空间释放，丢弃该数据包，然后退出函数。

第78行构造选项头对路由选项进行添加。

第98行调用dsr\_ack\_opt\_add()函数用于将ack选项头添加至数据包之中，第109行调用XMIT()函数，传输数据包。

## 4.8 确认（ACK）选项处理

当数据包顺利送达目标节点之后就会产生一个确认选项，而节点收到这个选项之后的处理也很简单，就是在维护缓存中删除发送给ACK源端的数据包，因为收到了ACK表明数据包已经顺利送达。

### 4.8.1 dsr\_ack\_opt\_recv()函数分析



图 4.8.1 dsr-ack.c中关于ACK选项的处理dsr\_ack\_opt\_recv() 函数代码

在291-292行，首先判断是否是ACK，如果不是的话，则返回包错误标志。

在294-298行，对当前节点地址、源端IP、目的端IP和id进行赋值。

在300行，打印ACK的信息。

在306行，调用maint\_buf\_del\_all\_id()函数，删除维护缓存中发送给ACK源端的数据包。

## 4.9 产生路由错误选项

当发生路由错误时，比如网络拓扑结构发生变化，数据包无法送达，或者DSR选项头中的选项不支持时，节点就需要发出路由错误通知给数据包的源节点具体步骤如下：

### 4.9.1 dsr\_rerr\_send()函数分析







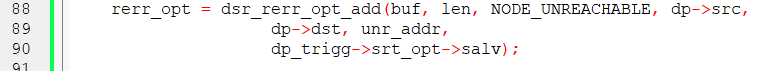


图 4.9.1 dsr-rrer中dsr\_rerr\_send()函数关于发送路由错误请求的实现

第一步自然就是申请一个dsr数据包（第24行），接下来是建立一条到达数据包源端的路由以便发送错误（第31行）。之后添加路由错误选项（都88-90行）。最后发送数据包（第131行）。

## 4.10 处理路由错误选项

在路由维护的过程中，一个重要的选项就是路由错误选项。这个选项用于通知路由的过程中发生的错误，以让节点做出相应的对策，代码实现下如下：

### 4.10.1 dsr\_rerr\_opt\_recv()函数分析



图 4.10.1 dsr-rerr.c中关于路由错误处理的实现 dsr\_rerr\_opt\_recv()函数代码

**dsr\_rerr\_opt\_recv()**函数中，实现了对于接收到的路由错误选项进行处理的功能。从中可以看出，对于节点无法到达的错误，节点会在自己的路由缓存中删除这条路由，而对于其它的错误，则会输出调试信息指明错误类型。

在215行，使用switch()对于不同的错误执行不同的命令。

在216-234行，如果是节点不可达，则打印错误信息，然后调用maint\_buf\_del\_all()函数及lc\_link\_del()函数，先丢弃所有未返回ack的数据包，然后再将这条损坏的链路从缓存中删除。

在235-238行，如果是流状态不支持或者是选项不支持，则打印错误信息。

# 第五章 小结

至此为止，DSR协议的分析已经全部完成。通过分析，我们很清楚地看到协议寻找路由的方法以及应对网络结构变化的方法。

无线网络覆盖是大势所趋，因此一个在无线网络下的可靠的路由也很重要，DSR协议就给了我们一个简单但又高效的解决方式，值得借鉴。

在本篇分析中，我们说明了动态源路由协议代码的原理。结合平时上课所学，和RFC文档的解读，最终形成了该篇分析。由于能力有限时间有限，dsr-uu中还有几个bug.h/.c以及补丁漏洞包没有进行主要分析，不能说是将DSR协议完美解读，但是我们的主要目的还是尽自己所能把协议的主要流程代码分析理解透彻并以PPT和大作业形式展现给大家。希望课程结束达到此目的，文档中分析不对的地方还请多多指正。

对于此协议的分析过后，无论是对于代码阅读与分析能力，还是说对于路由协议的理解，以及英文阅读（RFC文档）我们自认为都会有很大的提升。

最终还是要感谢老师助教同学们在这门课程中带给我的知识。