DSR-UU代码分析

组长：张伟

学号：201692204

分工：

组员：常瑞

学号：

分工：

目录

**[1.1](#_Toc501465446)****[引言](#_Toc501465446)** [3](#_Toc501465446)

**[1.2](#_Toc501465447)****[协议的主要组成部分](#_Toc501465447)** [3](#_Toc501465447)

**[1.3](#_Toc501465448)****[通信过程简介](#_Toc501465448)** [3](#_Toc501465448)

**[1.4](#_Toc501465449)****[基础知识](#_Toc501465449)** [6](#_Toc501465449)

**[1.4.1](#_Toc501465450)****[DSR选项头分析](#_Toc501465450)** [6](#_Toc501465450)

**[1.4.2](#_Toc501465451)****[DSR选项通用头分析](#_Toc501465451)** [7](#_Toc501465451)

**[1.4.3](#_Toc501465452)****[路由请求选项](#_Toc501465452)** [8](#_Toc501465452)

**[1.4.4](#_Toc501465453)****[路由回复选项](#_Toc501465453)** [8](#_Toc501465453)

**[1.4.5](#_Toc501465454)****[源路由选项](#_Toc501465454)** [9](#_Toc501465454)

**[1.4.6](#_Toc501465455)****[路由错误选项](#_Toc501465455)** [10](#_Toc501465455)

**[1.5](#_Toc501465456)****[协议具体分析](#_Toc501465456)** [10](#_Toc501465456)

**[1.5.1](#_Toc501465457)****[DSR数据包结构](#_Toc501465457)** [10](#_Toc501465457)

**[1.5.2](#_Toc501465458)****[为DSR选项头分配内存](#_Toc501465458)** [12](#_Toc501465458)

**[1.5.3](#_Toc501465459)****[协议IO分析](#_Toc501465459)** [13](#_Toc501465459)

**[1.5.4](#_Toc501465460)****[处理源路由请求](#_Toc501465460)** [14](#_Toc501465460)

**[1.5.5](#_Toc501465461)****[发出路由发现请求](#_Toc501465461)** [15](#_Toc501465461)

**[1.5.6](#_Toc501465462)****[处理路由发现请求](#_Toc501465462)** [20](#_Toc501465462)

**[1.5.7](#_Toc501465463)****[确认请求（ACK Request）选项处理](#_Toc501465463)** [22](#_Toc501465463)

**[1.5.8](#_Toc501465464)****[确认（ACK）选项处理](#_Toc501465464)** [22](#_Toc501465464)

**[1.5.9](#_Toc501465465)****[产生路由错误选项](#_Toc501465465)** [23](#_Toc501465465)

**[1.5.10](#_Toc501465466)****[处理路由错误选项](#_Toc501465466)** [23](#_Toc501465466)

**[1.6](#_Toc501465467)****[总结](#_Toc501465467)** [24](#_Toc501465467)

* 1. **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。

动态源路由协议是一个设计给多跳无线自组织网络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。

* 1. **协议的主要组成部分**

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. **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（1.5.3节）。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：

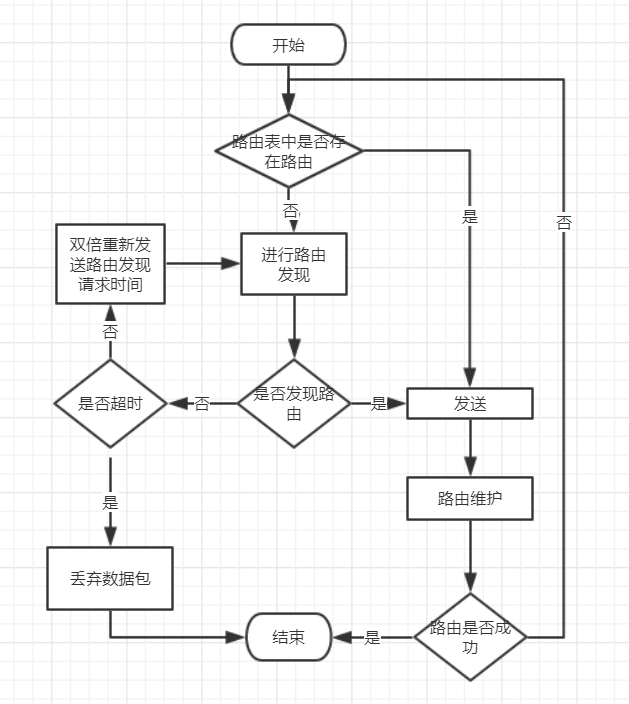
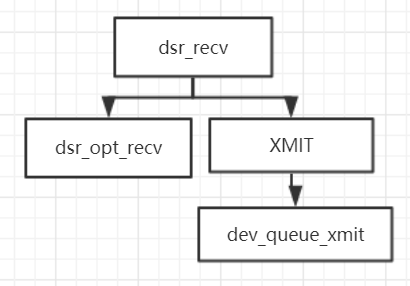
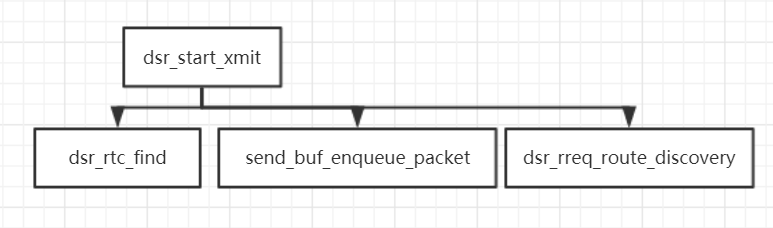
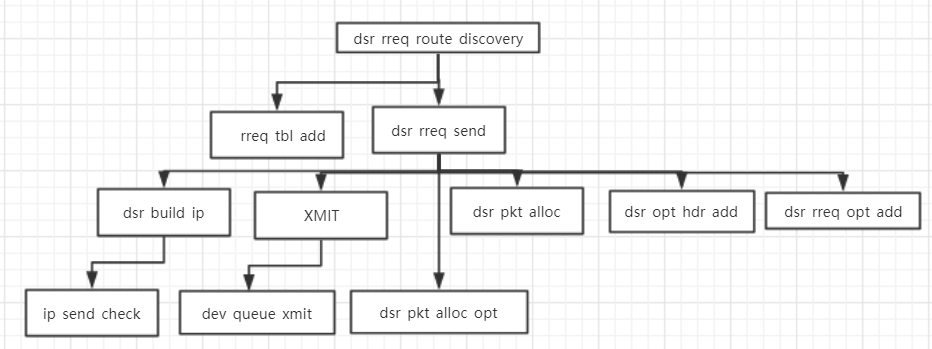


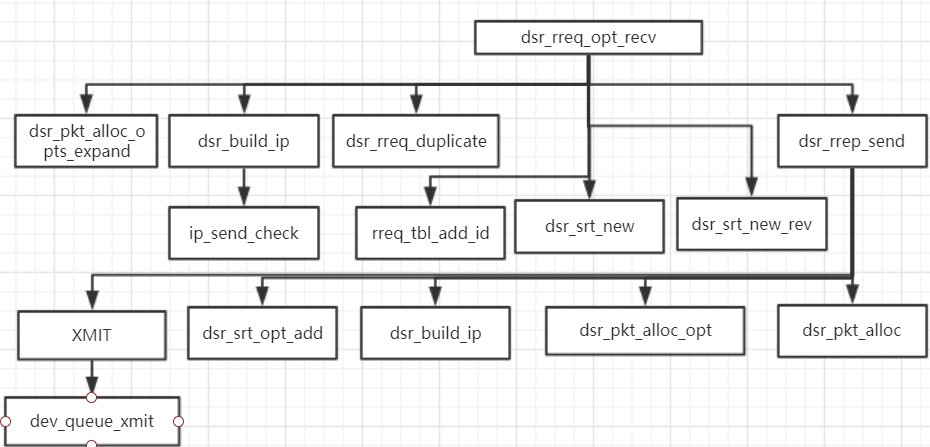
图 1 通信过程流程图

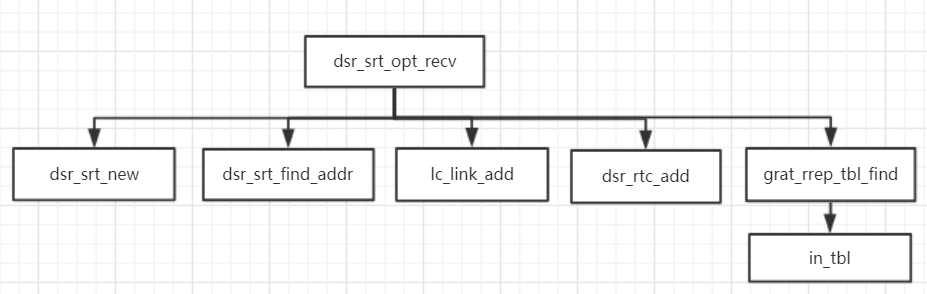
下图是项目中关键函数的调用图，我们在之后的分析中也会详细介绍。

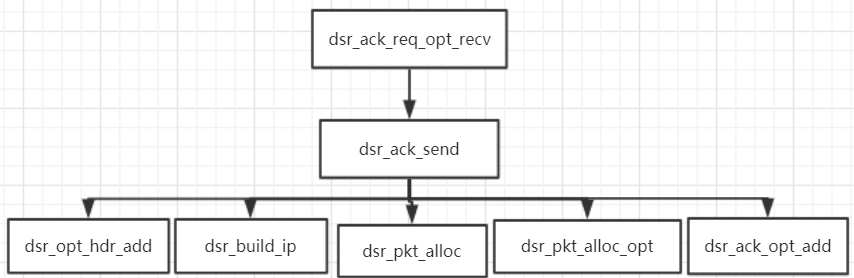












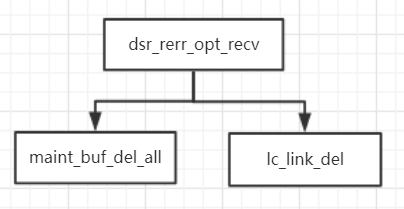


图 2 函数调用图

* 1. **基础知识**

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。

* + 1. **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。DSR选项头的结构体名字是“dsr\_opt\_hdr”，它的定义如图1所示：

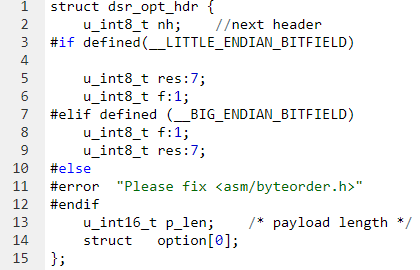


图 3 dsr-opt.h中关于DSR Option Header的定义

其中各个不同变量的意义如下：

**nh:**

Next Header,与IP头中的协议字段取值相同。

**f:**

标志位，在DSR选项头中置0，在DSR流状态头中置1。

**res:**

保留位，置0。

**p\_len:**

Payload length, 负载长度，所有DSR选项的总长度。

**option:**

变长域，记录了DSR选项，以type-length-value(类型-长度-值)的形式编码。

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）。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 4 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。

* + 1. **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：

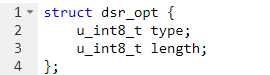


图 5 dsr-opt.h中关于选项通用结构的定义

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）。

* + 1. **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，路由选项的定义如下：

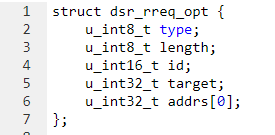


图 6 dsr-rreq.h中关于路由请求选项结构的定义

成员type与length是通用选项头结构,不多加赘述。

**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。

* + 1. **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，路由回复选项的格式如下：

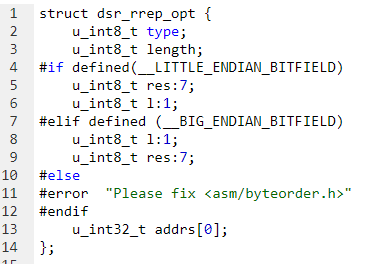


图 7 dsr-rrep.h中关于路由回复选项结构的定义

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)：**

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

**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。

* + 1. **源路由选项**

源路由选项的定义如下：

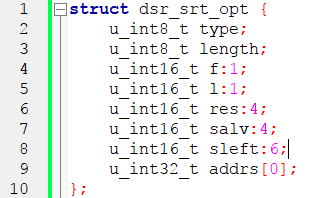


图 8 dsr-srt.h中关于源路由选型的定义

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1.4.4节中路由回复选项的L。

**Resversed(res):**

保留位，置0。

**Segments Left(left):**

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。

**Address[1..n]：**

源路由的路径。

* + 1. **路由错误选项**

路由错误选型用于通知路由的错误，它的定义如下：

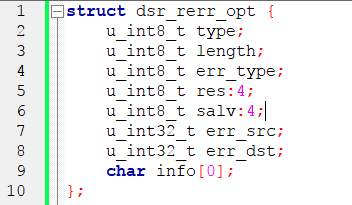


图 9 dsr-rerr.h中关于路由错误的定义

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。

**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。

* 1. **协议具体分析**

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。

* + 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：

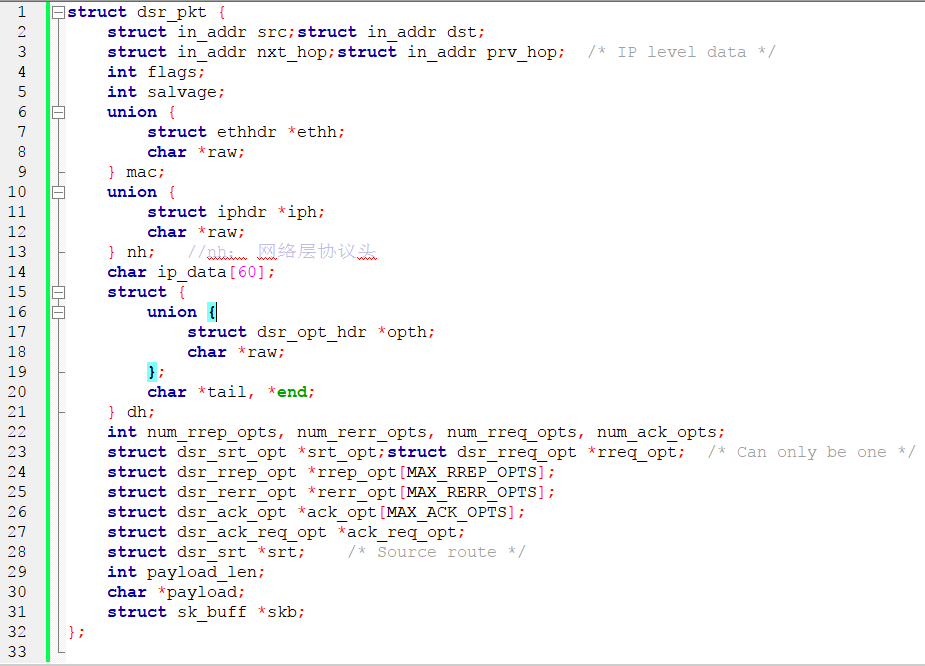


图 10 dsr-pkt.h中关于dsr数据包的定义

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。

而生成数据包的函数dsr\_pkt\_alloc的实现如下：

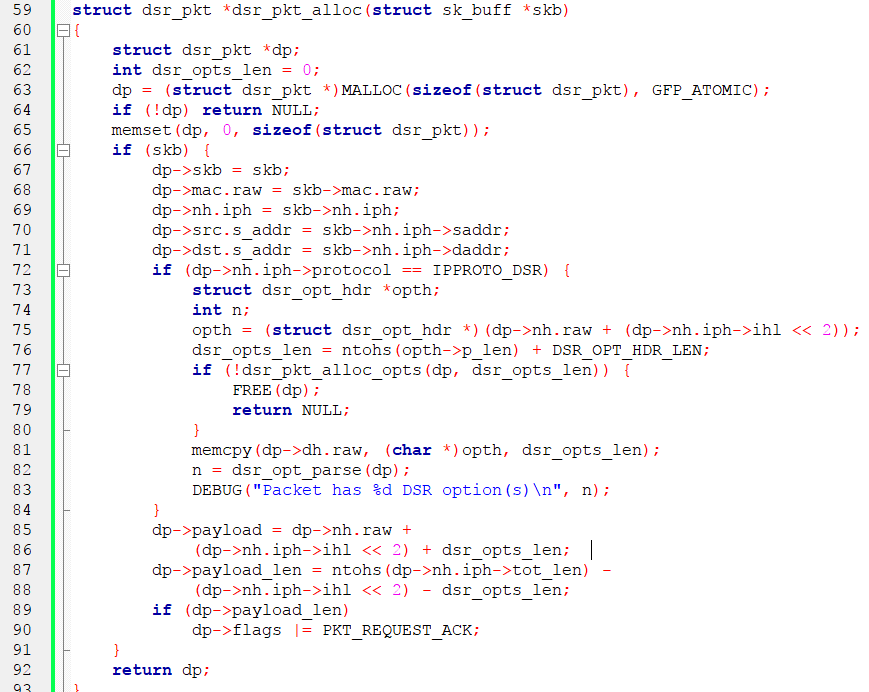


图 11 dsr-pkt.c中关于生成数据包的实现

从第63行可以看出，这一步骤首先会申请数据包所需要的内存，并全部清零（65行），如果skb这个参数不为空，则数据包根据skb（struct sk\_buff是linux网络系统中的核心结构体，linux网络中的所有数据包的封装以及解封装都是在这个结构体的基础上进行）的内容进行初始化。完成之后返回指向这个数据包的指针。

* + 1. **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：

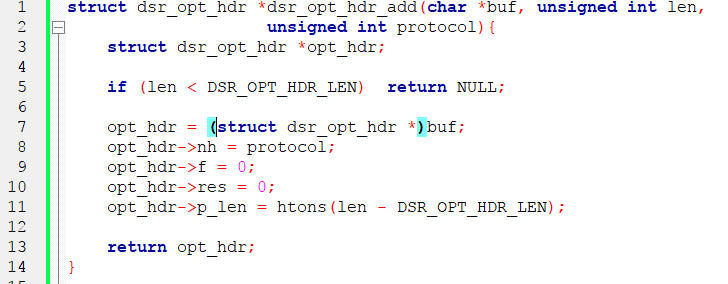


图 12 dsr-opt.c中关于为选项头分配内存的实现

这一过程首先判断了缓冲区的长度是否足够放下DSR选项头（第5行），如果足够，则把选项头指针指向这个缓冲区（第6行），并且对其相关的信息进行初始化（第8-11行），最后返回这个指针。

* + 1. **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：







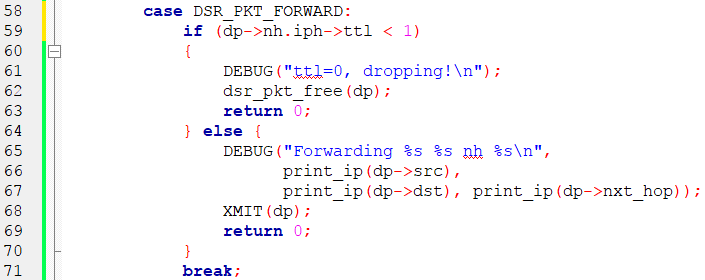
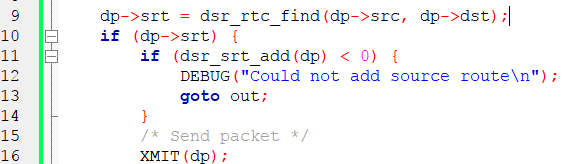


图 13 dsr-io.c中关于接受数据包的处理

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（第35行），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（第45行），我们着重分析转发数据包的情况（第58-71行），当ttl为零时，丢弃数据包，而其他情况下调用XMIT函数（1.5.5节会详细介绍）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。





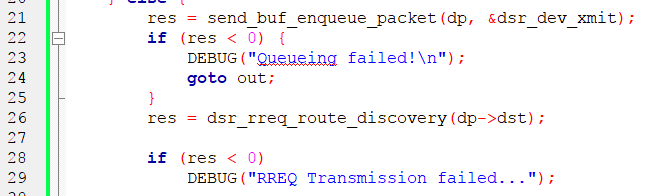


图 14 dsr-io.c中关于发送数据包的处理

首先找到是否有到达目标节点的源路由，有的话则添加源路由并且发现数据包（第9-16行）。没有的话就把数据包加入发送缓存并且进行路由发现（第21-29行）。

* + 1. **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。

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：

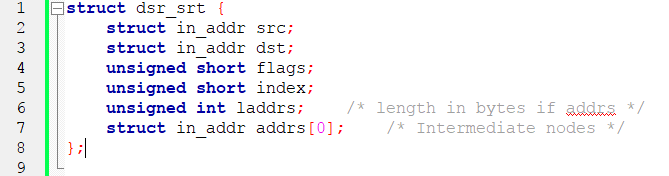


图 15 dsr-srt.h中关于源路由的路径

**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。

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1.4.5节），具体的代码实现如下：

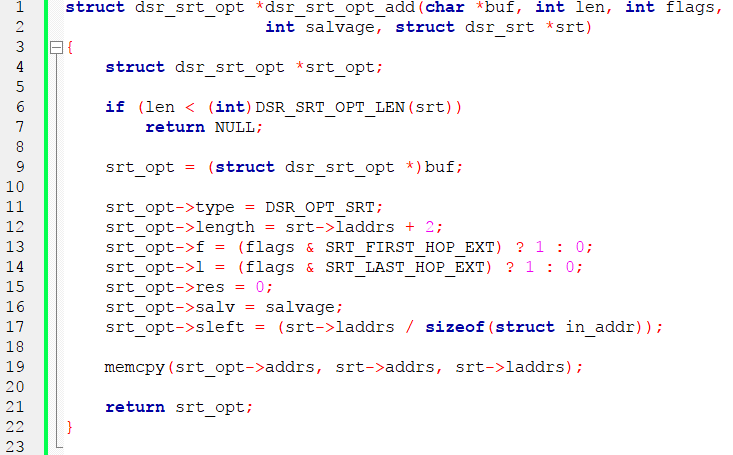


图 16 dsr-srt.c中关于添加源路由选项的实现

首先判断缓冲区的长度是否足够（第6行），如果足够，初始化相关的类型，长度，标志位（第11-17行）并且复制地址列表（第19行）。至此，源路由选项添加完成。

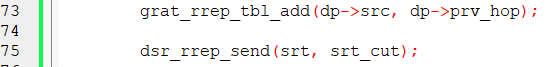
源路由一个基本的功能就是转发数据包，但是另一个值得一提的功能就是路由缩减，也就是缩短路由路径。当节点监听到一个数据包但是该数据包的下一跳地址不是它自己时，节点就会在这个数据包的剩余源路由中搜索是否有自身的地址，如果存在，意味着路由能够缩减，这一段逻辑的实现代码如下：   



图 17 dsr-srt.c中关于源路由选型的处理

第45-47行用于判断是否可以经行路由缩减，第73-75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97行）。

* + 1. **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。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，首当其冲的就是路由请求表，它的定义如下：

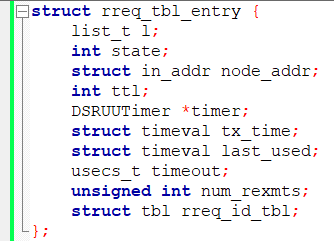


图 18 dsr-rrep.c中路由请求表的定义

我们对其中重要的成语进行分析：

**node\_addr:**

路由请求的目标地址。

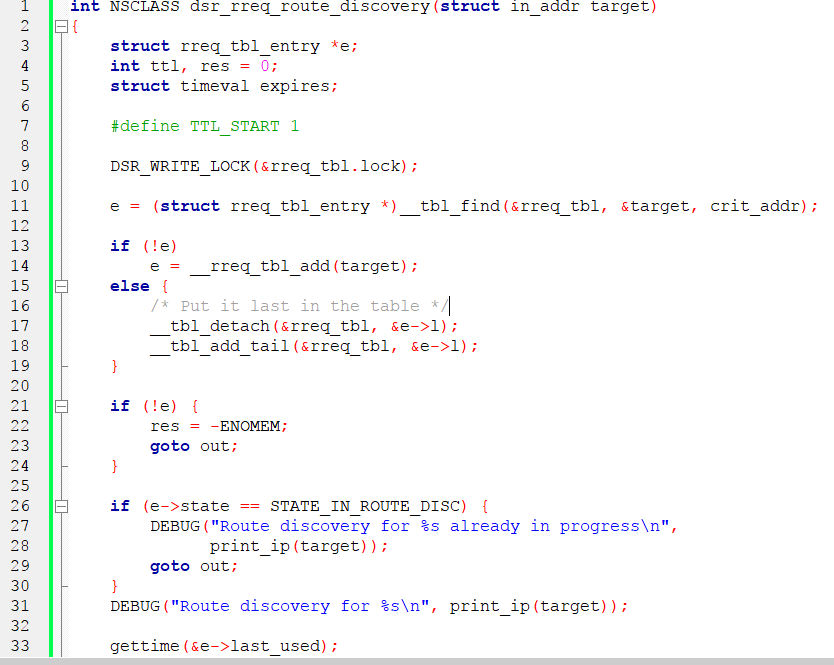
**state:**

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：



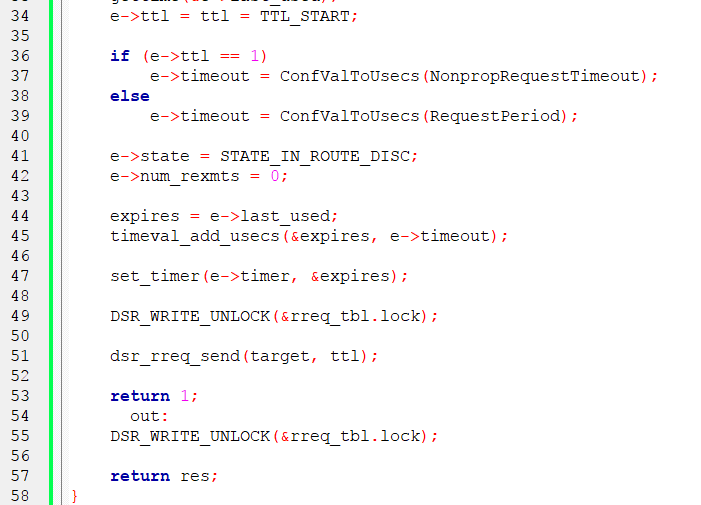


图 19 dsr-rreq.c中关于路由发现的实现

从代码中可以很清晰的看出路由发现的运行逻辑。第11行代码是路由发现的第一个步骤：在路由请求表中寻找是否已经有相同目的地的路由请求。\_\_tbl\_find函数的实现如下：

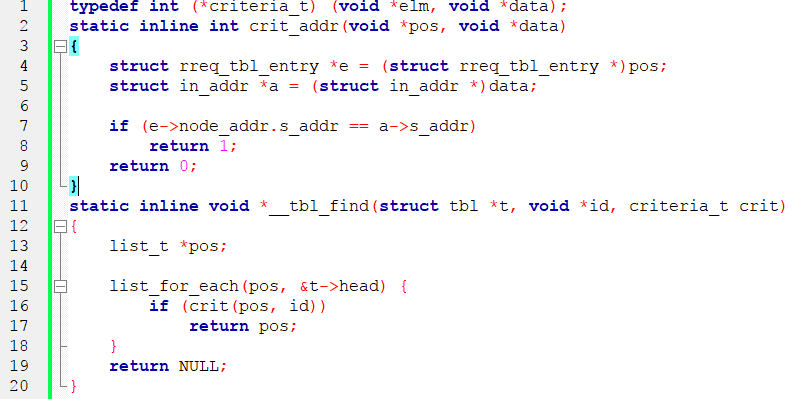


图 20 tbl.h中\_\_tbl\_find的定义和dsr\_rreq.c中crit\_addr的定义

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9第7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。

清楚了这一点之后再看图9的代码就很清晰了，第13-14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。

添加路由请求表项的过程的具体实现如下：

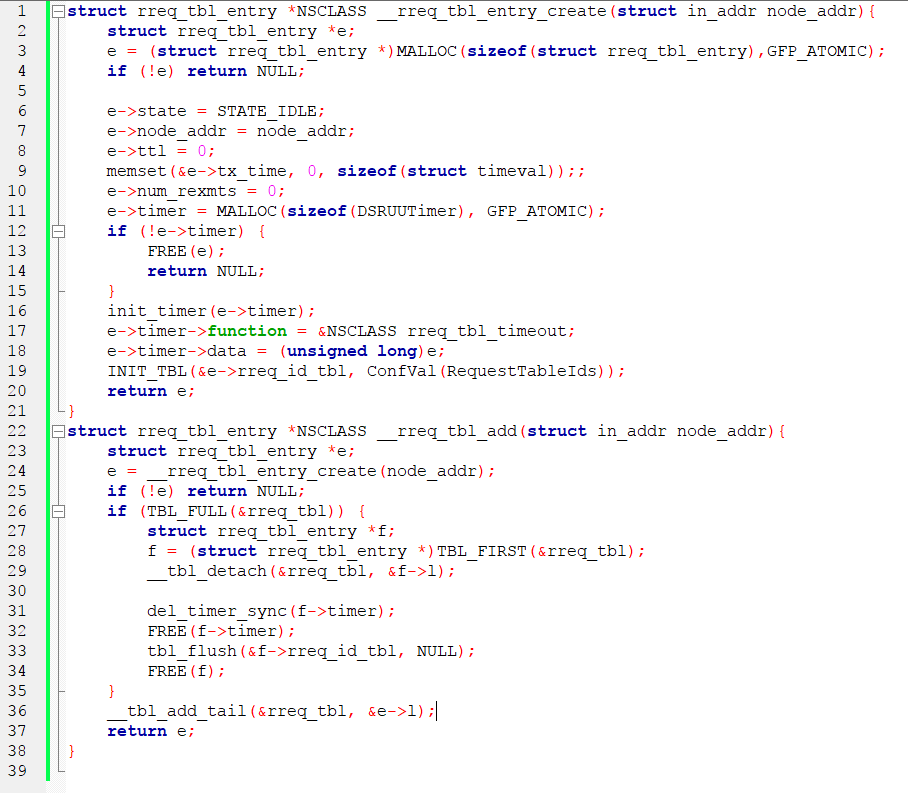


图 21 dsr-rreq.c中关于创建路由请求选项的实现

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10的第26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，那么就会将请求表中的第一个表项删除（27-35行），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33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36-39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44-45行设置失效时间expires。

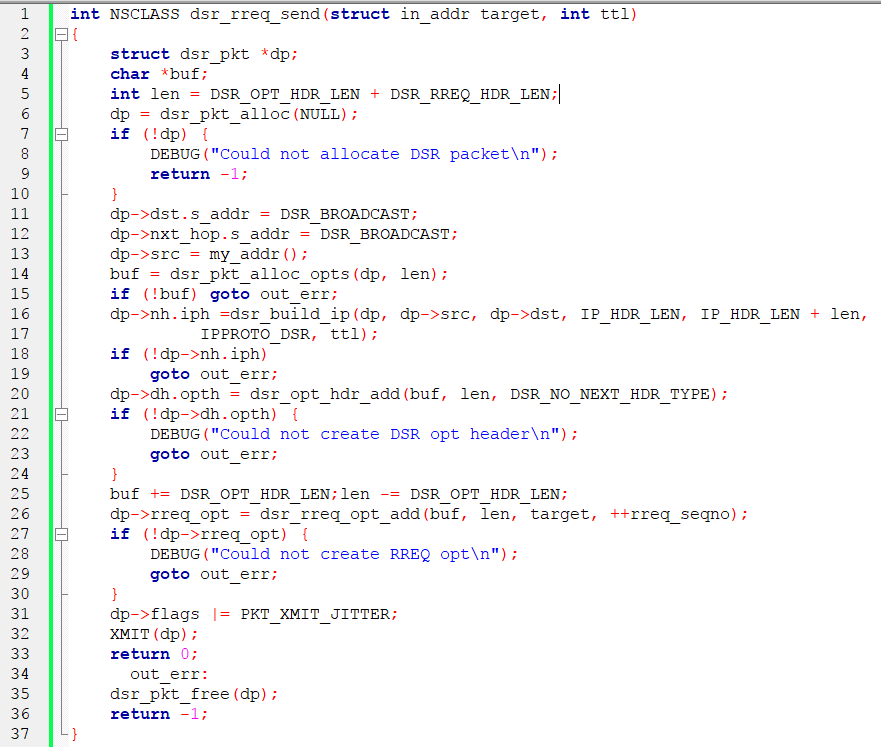
在完成了对时间的相关步骤之后，就到了最后一步——发送请求了，dsr\_rreq\_send函数负责了这一任务，他的实现如下：

图 22 dsr\_rreq.c中关于发送路由请求的实现

首先，在发送前，会先申请分配一个DSR数据包（第6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（第11到第13行），同时为DSR选项分配相应的内存（第14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16-17行）。在完成IP头的构造之后就是DSR选项头的构造（第20行）。完成之后自然就是路由请求选项的添加（第26行），它的实现如下：

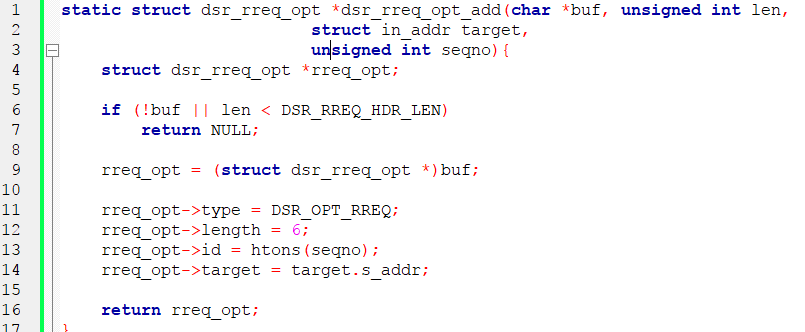


图 23 dsr-rreq.c中关于添加路由请求的实现

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。

至此为止，数据包的构造已经全部完成。接下来的工作就是发送数据包了。而XMIT(dp)是dsr\_dev\_xmit（dp）的宏定义（xmit的含义是transmit，与就是传输的意思），它的实现如下：

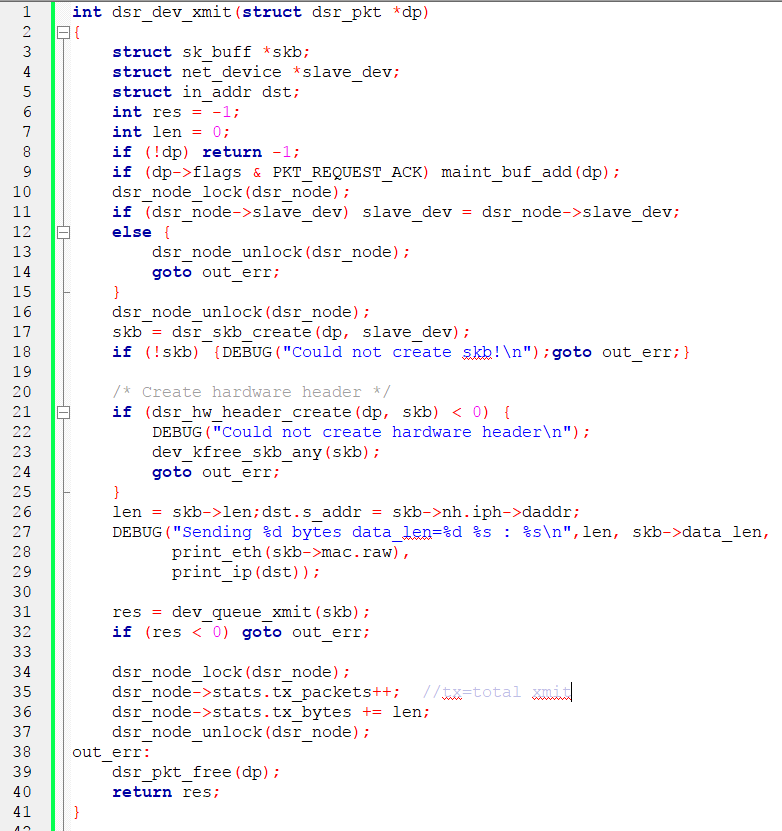


图 24 dsr-dev.c中关于数据包发送的实现

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（第21-26行），最后一步调用dev\_queue\_xmit(skb)将数据包发送出去（第31行）并统计发送的数据包信息（第35到36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35行）并返回-1。

* + 1. **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

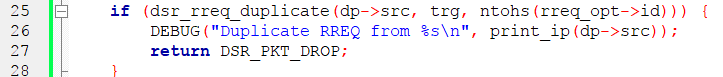
2. 将自己的IP地址添加到路由请求选项的Address数组中，并更改选项的length值。

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。

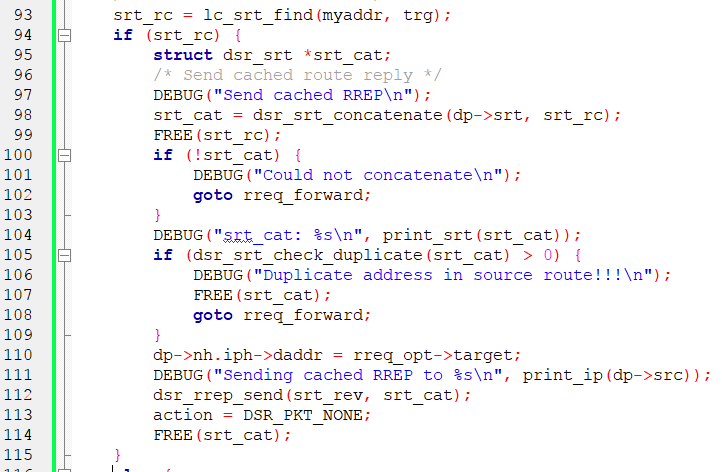
4. 如果没有发现，则以链路层广播的形式发送这个修改后的数据包。

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：









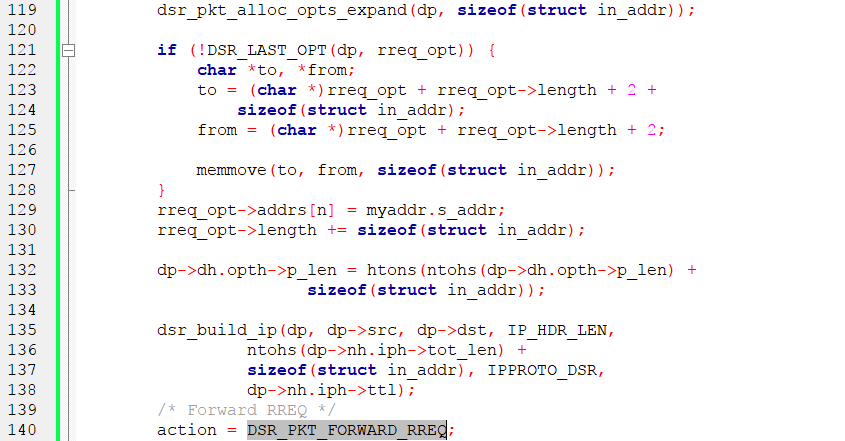


图 25 dsr-rreq.c中关于处理收到的路由请求的实现

第25-28行代码用于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（第30行），并更具该数据包的源路由建立逆向源路由以便于发送回复（第44行）。如果路由请求的目的地址等同与自身的地址，就发送路由回复第66-80行），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（第93行）。发现对应路由就够在新的源路由（第98行）并发送路由回复请求（第112行）。如果没有发现，为DSR选项分配更多的内存（第119行），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（第129-138行）进行转发（第140行）。

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。

* + 1. **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。





图 26 dsr-ack.c中关于ACK请求选项的处理

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（第19行）。

* + 1. **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（第13行和第22行），因为收到了ACK表明数据包已经顺利送达。







图 27 dsr-ack.c中关于ACK选项的处理

* + 1. **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：







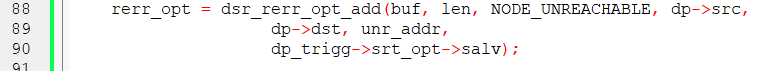




图 28 dsr-rrer中关于发送路由错误请求的实现

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）。

* + 1. **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：

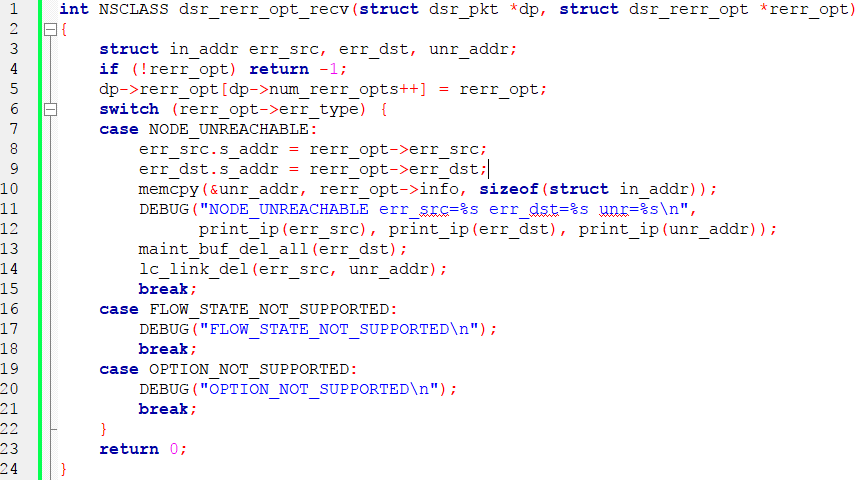


图 29 dsr-rerr.c中关于路由错误处理的实现

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（第7-15行），而对于其它的错误，则会输出调试信息指明错误类型（第16到21行）。

* 1. **总结**

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。