**网络协议栈分析与设计课程大作业**

|  |
| --- |
| **IS-IS路由协议代码分析**  （仅供参考，分析可以更详细点） |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | | 学号 | 姓名 | 班级 | 负责模块 | 成绩 | | 201492\*\*\* | 裴\*\* | 软网1\*\*1 | 1.代码介绍  2.子网相关功能  3.PDU的处理  ，选择局域网DIS  4.排版 |  | | 201492\*\* | 王\*\* | 软网1\*\*3 | 1.引言  2.子网独立功能  3.链路状态机  ，动态主机名  4.文献搜集 |  | | 201492\*\* | 刘\*\* | 软网1\*\*3 | 1.数据结构  2.PDU结构与编码  3.SPT算法的实现  ，更新路由  4.总结 |  | |

目录

[摘要 2](#_Toc470863378)

[Abstract 3](#_Toc470863379)

[第一章 引言 4](#_Toc470863380)

[第二章 代码介绍 5](#_Toc470863381)

[2.1 符号与缩写 5](#_Toc470863382)

[2.2 文件介绍 6](#_Toc470863383)

[2.3 全局变量 7](#_Toc470863384)

[2.4 统计量 7](#_Toc470863385)

[第三章　数据结构 8](#_Toc470863386)

[3.1 struct isis 8](#_Toc470863387)

[3.2 struct isis\_area 9](#_Toc470863388)

[3.3 struct isis\_circuit 10](#_Toc470863389)

[3.4 struct isis\_lsp 11](#_Toc470863390)

[3.5 struct isis\_adjacency 12](#_Toc470863391)

[第四章 IS-IS 路由协议的实现 13](#_Toc470863392)

[4.1 PDU的处理 13](#_Toc470863393)

[4.1.1 概述 13](#_Toc470863394)

[4.1.2 isis\_handle\_pdu 函数 15](#_Toc470863395)

[4.1.3 tlv相关函数 18](#_Toc470863396)

[4.2 选择局域网DIS 20](#_Toc470863397)

[4.2.1 概述 20](#_Toc470863398)

[4.2.2 run\_dr\_elect 函数 21](#_Toc470863399)

[4.2.3 run\_dr\_resign 函数 24](#_Toc470863400)

[4.2.4 run\_dr\_commence 函数 25](#_Toc470863401)

[4.3 链路状态机 27](#_Toc470863402)

[4.1 概述 27](#_Toc470863403)

[4.2 isis\_csm\_stat\_change 函数 28](#_Toc470863404)

[4.4 动态主机名 29](#_Toc470863405)

[4.4.1 概述 29](#_Toc470863406)

[4.4.2 cache相关函数 30](#_Toc470863407)

[4.5 SPT算法的实现 32](#_Toc470863408)

[4.5.1 概述 32](#_Toc470863409)

[4.5.2 add\_to\_path 33](#_Toc470863410)

[4.5.3 isis\_run\_spf函数 34](#_Toc470863411)

[第五章 总结 37](#_Toc470863412)

# 引言

IS-IS协议是一种基于链路状态算法的路由协议，这意味着作为中间系统的路由器，必须完全知晓自己所在区域内部所有其它的路由器和它们的链路状态。IS-IS协议通过发送LSP（链路状态协议数据单元）进行路由信息通告，通过Hello PDU发现邻居和创建邻接，[5]通过SNP（序号协议数据单元）确保路由器持有最新的、有效的LSP。路由器在接收到的LSP的基础上，通过SPF（最短路径优先）算法最终确定路由表。

IS-IS协议把路由器分为三种：L1路由器、L2路由器和L1/L2路由器。其中，L1路由器是为端系统（比如个人电脑）直接提供服务的路由器，L2路由器是骨干路由器。L1/L2路由器向L1路由器隐藏骨干区域的细节，并负责将由L1路由器组成的非骨干区域的数据请求转发到由L2路由器组成的骨干区域。和OSPF协议直接使用IP地址标识路由器不同，IS-IS协议使用OSI的地址方案NASP标识路由器节点。因此，纯粹的IS-IS协议并不直接适合于IP网络；而集成IS-IS能够实现IP路由的关键，在于RFC 1195引入携带IP路由信息的CLV字段。

标准的IS-IS协议是由国际标准化组织制定的ISO/IEC 10589:2002 所规范的。但是标准的IS-IS协议是为无连接网络服务（CLNS）设计的，并不直接适合于IP网络，因此互联网工程任务组制定可以适用于IP网络的集成化的IS-IS协议，称为集成IS-IS，它由RFC 1195等RFC文档所规范。由于IP网络的普遍存在，一般所称的IS-IS协议，通常是指集成IS-IS协议。

# 第二章 数据结构

## 2.2 文件介绍

ISIS协议是由C语言实现的，头文件加上源文件一共48个，下表给出了主要的文件并进行了说明：

|  |  |
| --- | --- |
| 文件名 | 说明 |
| dict.[ch]  isis\_adjacency.[ch]  isis\_circuit.[ch]  isis\_common.h  isis\_constants.h  isis\_csm.[ch]  isis\_dr.[ch]  isis\_dynhn.[ch]  isis\_events.[ch]  isis\_flags.[ch]  isis\_lsp.[ch]  isis\_main.c  isis\_misc.[ch]  isis\_network.[ch]  isis\_pdu.[ch]  isis\_route.[ch]  isis\_spf.[ch]  isis\_tlv.[ch]  isis\_zebra.[ch]  isisd.[ch]  iso\_checksum.[ch] | Kazlib 红黑树  邻域数据库  链路相关路由  所有模块的定义  一些常量的定义  链路状态机  局域网DIS路由  动态主机名缓存  事件包装器（eg.主机类型更改事件）  SRM和SSN中flags 的实现  链路状态数据库和链路状态路由  路由协议主函数  多方辅助路由  OSI第二网络层I/O  IS-IS 协议包的处理  路由的处理  SPT（最短路径树）算法  TLV 的处理  Zebra接口  顶层数据结构和相关路由  Fletcher 校验和的实现 |

表2.2 ISIS 路由协议主要文件

## 2.3 全局变量

|  |  |  |
| --- | --- | --- |
| 变量 | 数据类型 | 说明 |
| \_caps\_p  isisd\_privs  longopts  config\_default[]  master  pid\_file  ALL\_L1\_ISS  ALL\_L2\_ISS  ALL\_ISS  ALL\_ESS  sock\_buff  isis | struct zebra\_capabilities\_t[]  struct zebra\_privs\_t  struct option  char[]  struct thread\_master\*  char\*  u\_char[]  u\_char[]  u\_char[]  u\_char[]  char[]  struct | Zebra 配置  ISIS的Zebra 配置  ISIS 路由选项  默认配置  主线程  初始化系统所需的进程ID  L1所有的IS  L2所有的IS  系统中所有的IS  系统中所有的ES  socket缓存  当前处理的IS |

表2.3 ISIS项目中的全局变量

## 2.4 统计量

结构esis\_stat中为每个系统（端系统和中间系统）收集的统计量如表2.4.

esis\_stat 描述了一个系统的基本状况，包括系统内存，各中协议收发的分组数等。

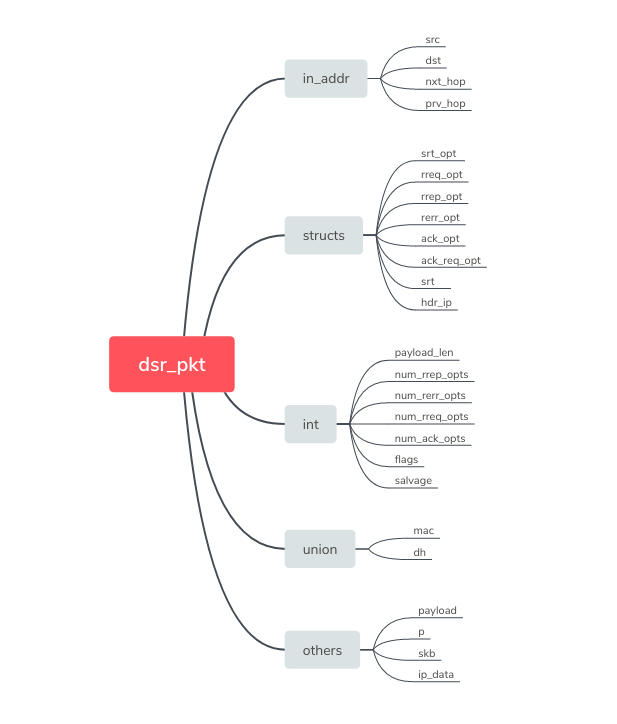
|  |  |  |
| --- | --- | --- |
| 变量 | 数据类型 | 说明 |
| es\_nomem  es\_badcsum  es\_badvers  es\_badtype  es\_toosmall  es\_eshsent  es\_eshrcvd  es\_ishsent  es\_ishrcvd  es\_rdsent  es\_rdrcvd | u\_short  u\_short  u\_short  u\_short  u\_short  u\_short  u\_short  u\_short  u\_short  u\_short  u\_short | 为发送hello包保留的内存  校验和错误的分组数  IP版本错误的分组数  PDU类型域不明的分组数  太小的分组数  ESH发送的分组数  ESH接收的分组数  ISH 发送的分组数  ISH 接收的分组数  RD 发送的分组数  RD 接收的分组数 |

表2.4 结构esis\_stat 维护的统计量

# 第三章　数据结构

## 3.1 通信单元dar\_pkt

不考虑在宏定义了NS2的情况时时，结构体的整体框架如图2.1-1所展示。它包含了各个模块的opt涉及不同的操作。



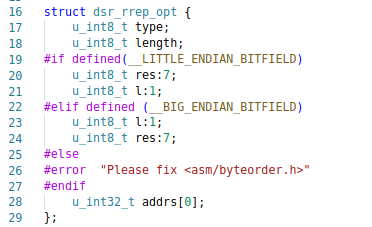
2.1-1 dsr\_pkt结构体

### 3.1.1 in\_addr

in\_addr结构体为系统所定义用于描述一个设备的ipv4地址。in\_addr.s\_addr为unsigned long类型是ip地址实际对应的32位值，是代码中多处进行地址比对的原理。

### 3.1.2 dsr\_xxx\_opt

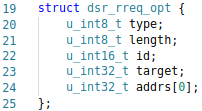
不同的操作对应不同的opt数据结构，对于某一些变量它们大体结构基本类似。以dsr\_rrep\_opt为例举出此种类型的大体存储结构。



dsr-rrep.h

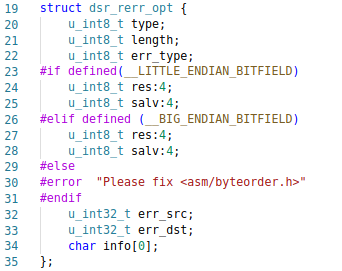
017-024行 使用3个8位无符号整型来携带各种特殊的flag, type等信息

028-028行 使用32位无符号整形进行地址的存储



dsr-rreq.h

022-023行 对于rreq来说，它与rrep的区别是增加了一个以16位无符号数为基本类型的id、和以32为无符号整形为基本类型的target。它们有着不同的功能在之后的函数功能介绍中会有具体的体现。



dsr-rerr.h

032-034行 对于dsr\_rerr\_opt而言，则添加了错误出现的节点位置和错误的类型。

考虑到结构非常类似且篇幅有限，过多的数据结构不在一一列举。省略的数据结构有dsr\_ack\_req\_opt和dsr\_ack\_opt分别对应acknowledgement request 和 acknowledgement 两个网络动作。

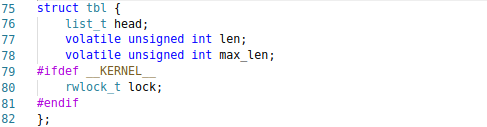
### 3.1.3 skb

skb是sk\_buff的缩写。中文可被译作套接字缓冲区。主要用来在linux网络子系统中各层之间数据传递，起到了“神经中枢”的作用。当发送数据包时，linux内核的网络模块必须建立一个包含要传输的数据包的sk\_buff,然后将sk\_buff传递给下一层，各层在 sk\_buff 中添加不同的协议头，直到交给网络设备发送。同样，当接收数据包时，网络设备从物理媒介层接收到数据后，他必须将接收到的数据转换为sk\_buff，并传递给上层，各层剥去相应的协议头后直到交给用户。

## 3.2 列表衍生、存储单元

### 3.2.1 tbl

tbl为table的缩写，有点与不足在下面分析，此种类型定义的变量有rreq\_tbl，grat\_rrep\_tbl，send\_buf，neigh\_tbl，maint\_buf由各个节点来维护。



tbl.h

076-076行 tbl结构体由list\_t封装而来

080-080行 定义程序锁避免不同进程之间读取时出错

077-078行 定义当前长度和最大长度在list\_t的基础上增加list\_t的安全性

### 3.2.2 rreq\_tbl\_entry结构体

### 

dsr-rreq.c

047-056行 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。每一个对象用来描述一次路由发现的运行。主要用于计时，统计TTL等。所有该类型的变量自创建起都保存于rreq\_tbl之中。其中元素具有唯一性。

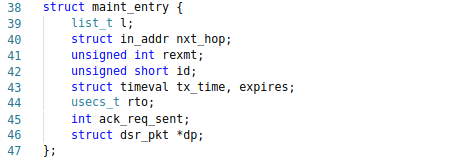
### 3.2.3 send\_buf\_entry结构体

send-buf为发送缓冲，每个包发送之前都要先压入缓冲由设备来进行发送。用send-buf-entry来描述一次buf sending可以避免由于设备故障等因素造成的过长时间的等待和资源浪费。

 send-buf.c

### 3.2.4 maint\_entry

maint-entry和maint-buf-query为maint-buf操作过程中所运用的两个结构体。在一个动态的网络中，节点状态往往会随时间的变化而发生改变，所以maint-buf需要定期的维护。使用maint-entry定期检测与邻居节点的连通性可以保证当与邻居链路断开时，路由错误信息立即被发布。保障了连接的稳定性。每个maint-entry用来描述一次路由维护的过程。

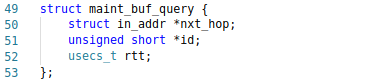


maint-buf.c

所有的该类型变量被收集与maint-buf之中。函数清单中列出了对maint-buf的各类基本操作对应的函数。

### 3.2.5 maint\_buf\_query和link\_query

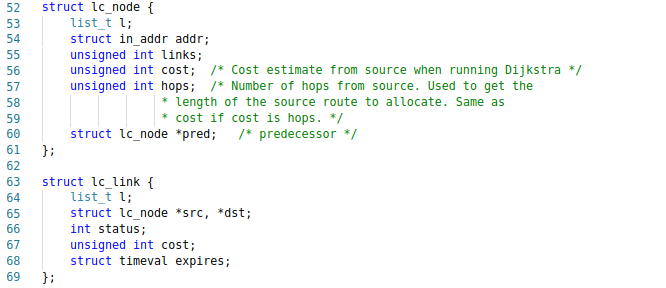
我们使用一个maint\_buf\_query来描述对一个目的节点的询问。当一条链路断开时，我们可以利用该结构体查询所有与节点相关的路径信息，并立即清空。

maint-buf.c

类似的结构link\_query和以上结构体功能类似。

* + 1. **lc\_node、lc\_link、和cheapest\_node**

与maint\_buf不同的是，lc(linc-cache)中存有两个list\_t列表。分别对应节点和边集。根据地杰斯特拉算法我们能够通过lc来得到所有联通节点的最短路径信息。lc\_node, lc\_link的功能便是如此。而cheapest\_node用来存放花销最小的节点，用来辅助迪杰斯特拉等算法的运行。





link-cache.c

# 全局变量

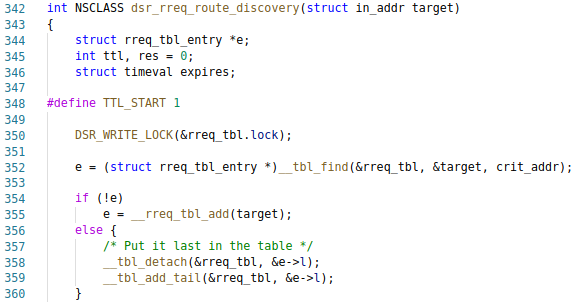
全局变量由列表形式给出，并概括了其具体功能。

# 函数详解

在函数详解一章的讲解中，我们挑出了一系列比较核心的函数进行了具体的讲解。对于函数讲解而言，我们在保证整体结构完整的前提下略去一些太过细节的部分。目的是更加清晰的刻画整个协议的大体流程。

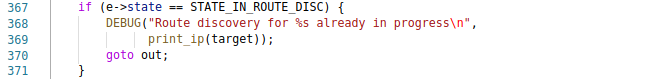
## 核心功能函数

### 4.1.1 dsr\_rreq\_route\_discovery函数



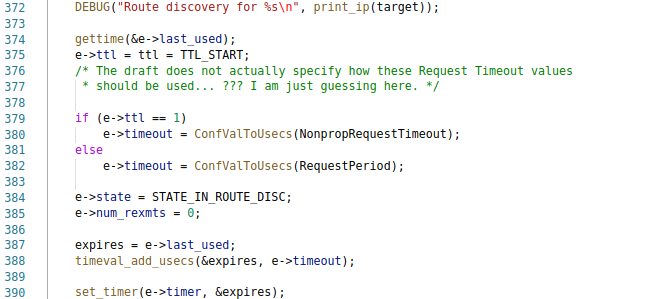
344-344行 e是一个rreq\_tbl\_entry类型的指针。

352-352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。



dsr-rreq.c

367-371行 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。



375-390行 在准备过程中，函数初始化时间，ttl等值。

dsr-rreq.c

394-394行 准备完成后，通过掉用相应函数，在网络启动discovery

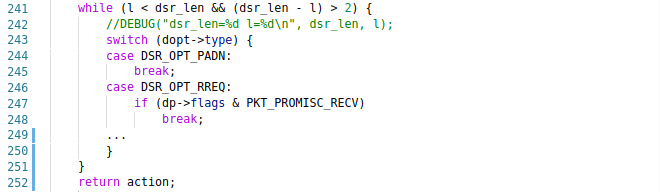
### 4.1.2 dsr\_opt\_recv函数



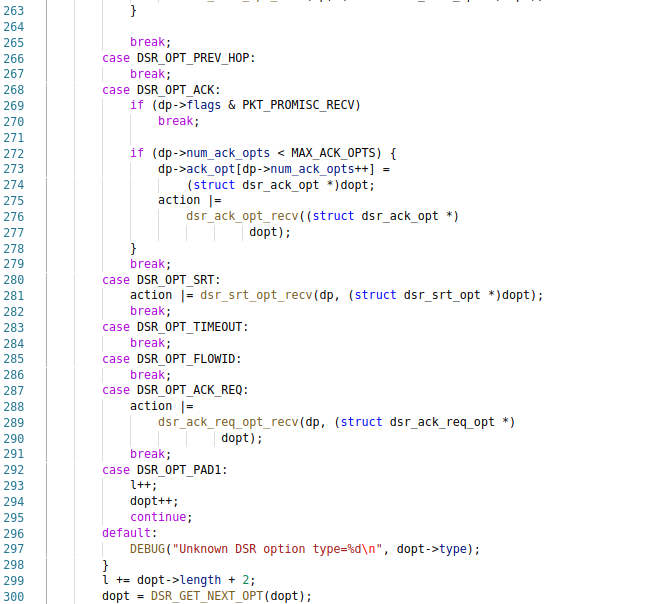
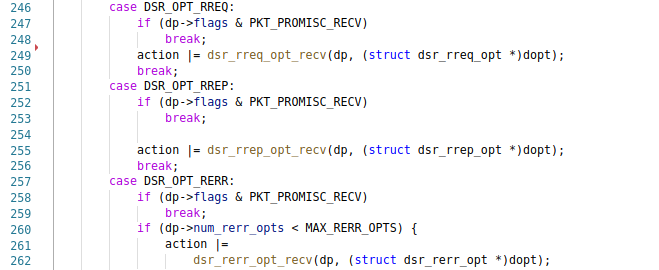
略去了初始化及其细节部分，218行是dsr\_opt\_recv进行的第一个操作。使用if条件判断进行安全性验证。



action是一个标记，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.c251-252行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

 dsr-opt.c

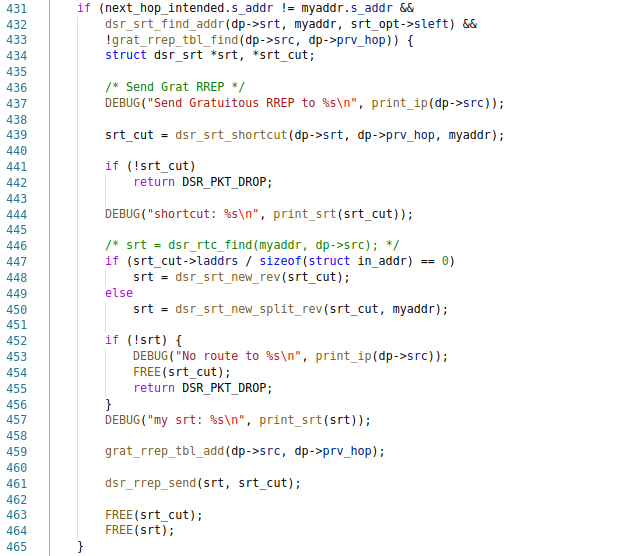
在父级函数中执行了DSR\_GET\_OPT宏并取出dp数据包中的数据包类型。在此用switch 来对不同数据包进行不同的接收操作。执行结果既返回了action，并且将数据包接收至dp指针处。不同数据包对应于不同种类的接收方式,对应返回不同的action值。

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。

### 4.1.3 dsr\_srt\_opt\_recv函数

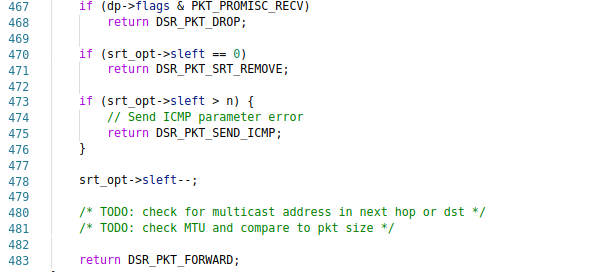
### 

### 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

431-464行进行了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。如果包中的源路由路径可以用来优化目前。则调用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。

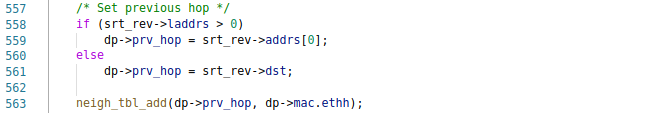


dsr-srt.c

467-483行 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。

### 4.1.4 dsr\_rreq\_opt\_recv函数

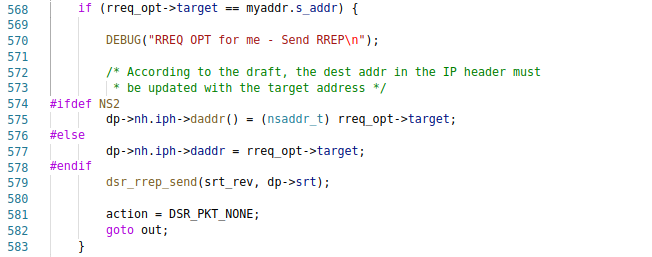
### 在接收过后将数据包传入该函数之中。避开所有的判断安全性和初始化的部分，对代码进行解析。首先添加该节点至rreq\_tbl之中。



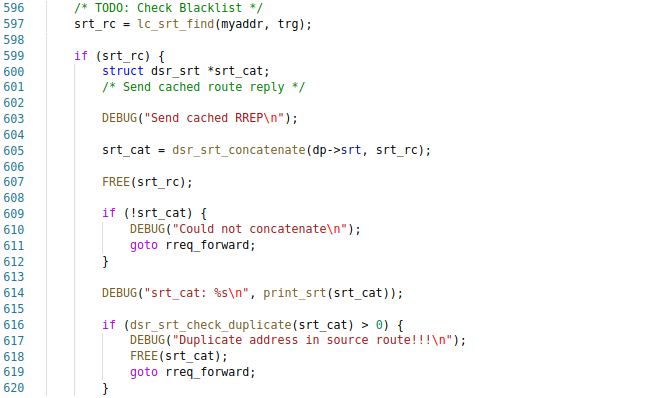
dsr-rreq.c

563-563行 rreq\_tbl\_add\_id会将新添加的节点放在列表最末尾

### 559-561行srt\_rev中，存有本身生成的反向路由。

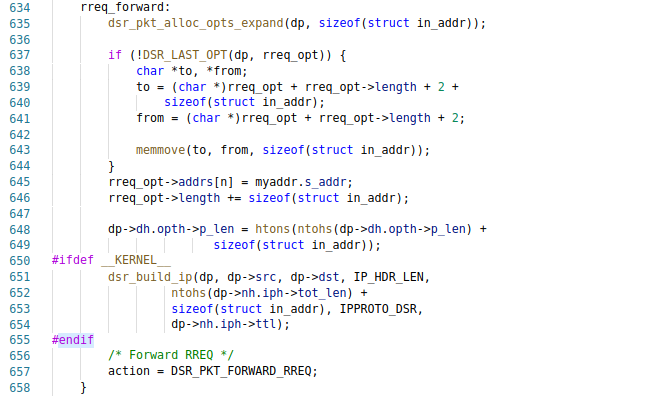


579-582行 通过反向的路径信息回传rrep消息。当消息回复以后，立即跳转，返回上层函数。



dsr-rreq.c

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

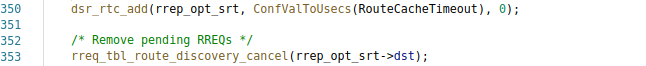
在条件满足时调用dsr\_srt\_concatenate函数处理路径信息。若条件不满足，则跳转到标签处开始执行。经过该部分action被赋值为DSR\_PKT\_FORWARD\_RREQ并将于函数最末尾返回。

### 4.1.5 dsr\_rrep\_opt\_recv函数

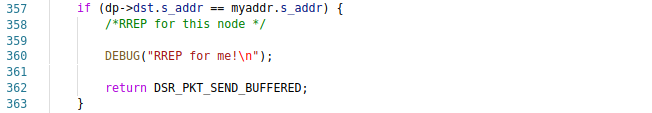
rrep的职责包括对RREP消息的转发与处理。

来看dsr\_rrep\_opt\_recv函数

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。

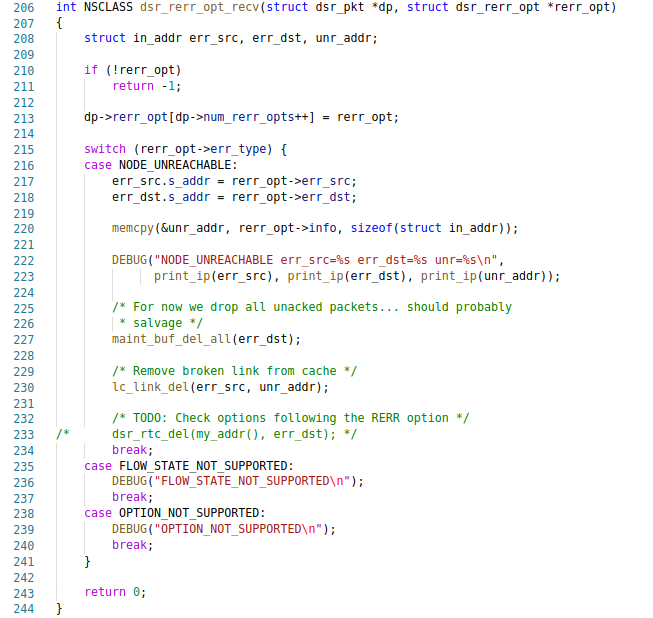


dsr-rrep.c

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。

### 4.1.6 dsr\_rerr\_opt\_recv函数

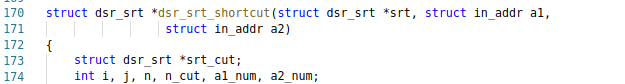
该函数的主要作用是传播路由错误信息。通过调用maint\_buf\_del\_all函数并且触发maint\_buf\_set\_timeout函数，再由maint\_buf\_set\_timeout函数调用dsr\_rerr\_send，对错误进行传输。所有节点收到数据包后，立刻将路由信息从自己的maint-buf中删除，并且帮助进行扩散。



dsr-rerr.c

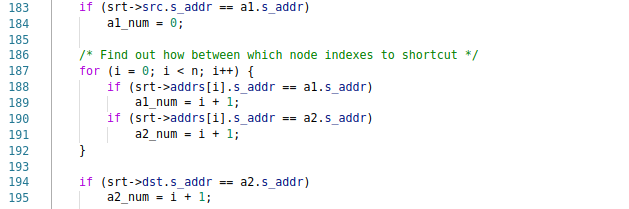
### 4.1.7 dsr\_srt\_shortcut函数

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。



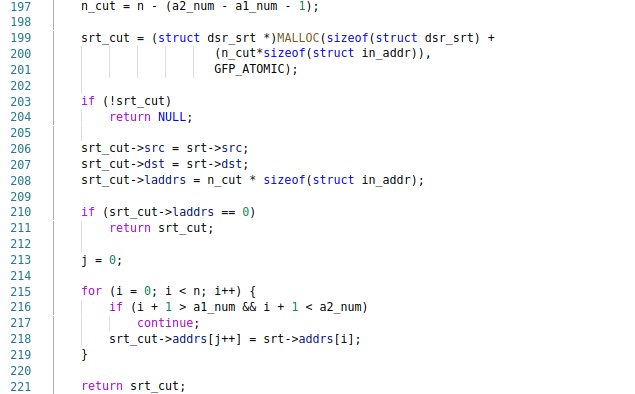
dsr-srt.c

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

### 4.1.8 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

081-084行 将源路由路径填入节点中

186-187行 将源节点地址和目的节点地址填入数据包后调用函数进行包传输

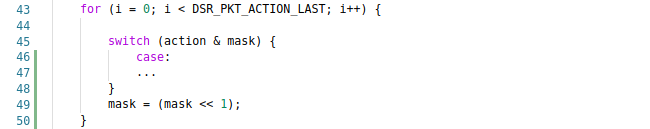
篇幅有限类似函数及其具体功能，不再通过细节展开。将采用列表的形式在之后的一章中呈现。

194-194行 调用定义好的函数进行包传输

### 4.1.9 dsr\_recv函数

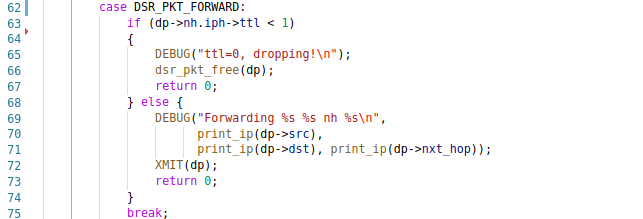


在调用dsr\_opt\_recv之后，dsr\_recv得到了一个完整的数据包（dp）。通过对action的区分，可以对不同的数据包进行不同的处理。dsr\_recv间接调用了以上所有的接收函数，是通过调用dsr\_opt\_recv函数来实现



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。

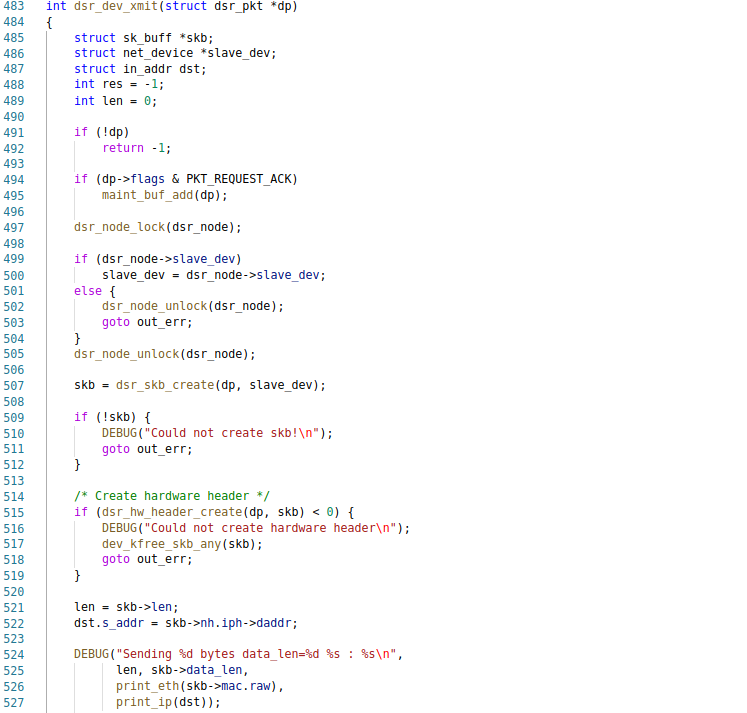


dsr-io.c

077-078行 开始进行传输

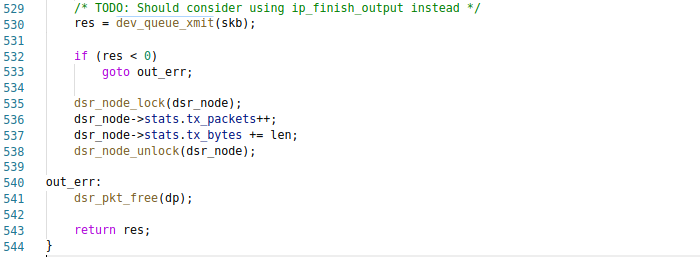
### 4.1.10 dsr\_dev\_xmit函数

所有数据包均由该函数发送。



dsr-dev.c

在完成对skb的各种初始化操作后，调用系统函数dev\_queue\_xmit对skbuff中的内容进行传输。并且释放dp的内存空间。



dsr-dev.c

## 所有函数及其功能汇总

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。我们的主要的工作内容是总结每个函数的具体功能。

### 核心功能

|  |  |  |  |
| --- | --- | --- | --- |
| 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 |

### 辅助工具函数、专有操作方法

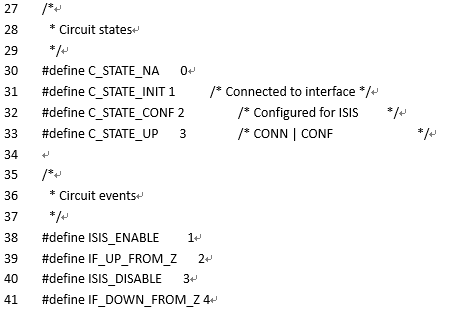
|  |  |  |  |
| --- | --- | --- | --- |
| 其他 | 函数名 | 功能简介（包括但不限于） | 声明位置 |
|  | dsr\_pkt\_alloc\_opts | 动态内存分配pkt\_opt | dsr-pkt.h |
| dsr\_pkt\_alloc\_opts\_expand | 扩展pkt\_opt |
| dsr\_pkt\_free\_opts | 释放pkt\_opt |
| dsr\_pkt\_alloc | 动态内存分配pkt |
| dsr\_pkt\_free | 释放pkt |
| dsr\_opt\_hdr\_add | 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  作为规则辅助tbl进行表内搜索 | dsr-rrep.h |
| crit\_time |
| crit\_addr | send-buf.h |
| crit\_garbage |
| crit\_addr\_id\_del | maint-buf.h |
| crit\_addr\_del |
| crit\_addr |
| crit\_ack\_req\_sent |
| crit\_expires |
| 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 |
| send\_buf\_print | send-buf基本操作 | send-buf.h |
| send\_buf\_set\_max\_len |
| send\_buf\_timeout |
| send\_buf\_entry\_create |
| send\_buf\_enqueue\_packet |
| send\_buf\_set\_verdict |
| send\_buf\_flush |
| send\_buf\_print |
| send\_buf\_get\_info |
| send\_buf\_cleanup |
|  | maint\_buf\_init | maint-buf基本操作 | maint-buf.h |
|  | maint\_buf\_cleanup |
|  | maint\_buf\_set\_max\_len |
|  | maint\_buf\_add |
|  | maint\_buf\_del\_all |
|  | maint\_buf\_del\_all\_id |
|  | maint\_buf\_del\_addr |
|  | maint\_buf\_set\_timeout |
|  | maint\_buf\_timeout |
|  | maint\_buf\_salvage |
|  | lc\_link\_del | link-cache基本操作 | link-cache.h |
|  | lc\_link\_add |
|  | lc\_garbage\_collect\_set |
|  | lc\_garbage\_collect |
|  | lc\_srt\_find |
|  | lc\_srt\_add |
|  | lc\_flush |
|  | lc\_init |
|  | lc\_cleanup |
|  | \_\_dijkstra | 迪杰斯特拉运算 |

## 链路状态机

### 概述

ISO10589中给链路定义了两种状态：up 和down。ISIS路由协议中使用了Zebra，因此ISIS路由协议中链路状态拓展成了四种：N/A（链路处于未知的状态），init（链路具有Zebra规定的基本特征，但还没有配置），configured（链路为了ISIS路由进行了配置但是没有连接到网络接口），up（链路根据ISIS配置好了并且连接到了网络接口）。

isis\_csm.h



isis\_csm.h

图4.5 链路状态机状态和事件的常量定义

链路状态机对实现在文件isis\_csm.[ch] 中。链路状态机的运行仅由一个函数isis\_csm\_state\_change控制。这个函数的两个参数分别是图5.1中定义的链路和事件。

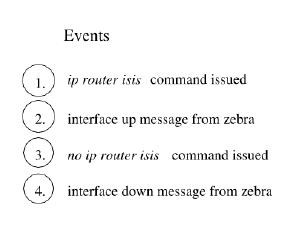
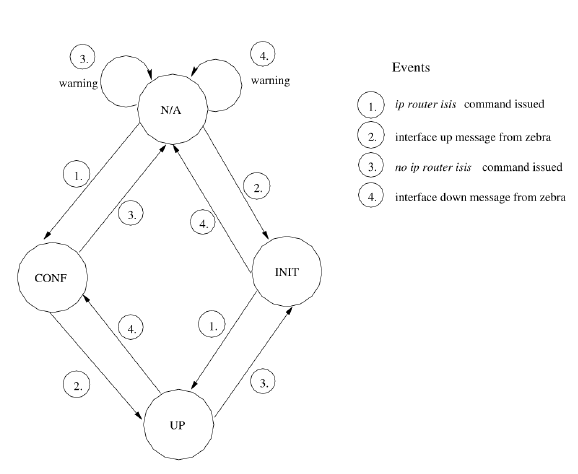


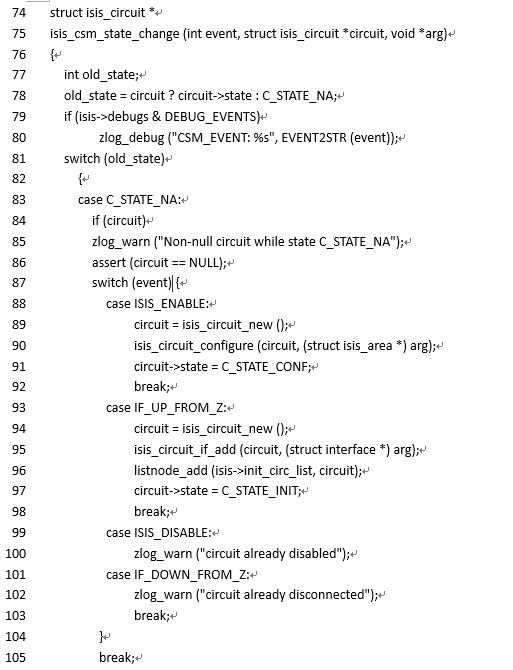
图4.6 链路状态机各个状态间的转换

图5.2表示了链路状态机的各个状态之间的转换。每个状态都可由两种其他的状态转换而来。

### isis\_csm\_stat\_change 函数

isis\_csm\_stat\_change函数的功能是控制链路状态机的运行，当不同的事件发生时，该函数会根据具体的事件类型进行处理。以下是在当链路状态机处于N/A状态时，各种事件触发的反应。

isis\_csm.c



isis\_csm.c

77 – 78 获取并保存状态机当前状态

79 – 80 当系统处于debug 状态时，打印log 日志。

83 – 86 若old\_state 的值为STATE\_NA，则加上断言circuit不为空。

88 – 92 若事件为ISIS\_ENABLE（即isis 路口器传递命令要求状态机准备使用），则创建个新 链路并把值赋给circuit，之后调用函数isis\_circuit\_figure配置链路，并把当前链路 状态改为C\_STATE\_CONF。

93 – 98 若事件为 IF\_UP\_FROM\_Z （即zebra 传递消息要求打开某端口），则创建个新链路 并把事件参数所指定的端口加入到链路中。最后把当前状态改为 C\_STATE\_INIT。

99 – 103 若事件值为ISIS\_DISABLE（即 路由器要求关闭链路状态机），则系统在log 中打印 出警告信息。

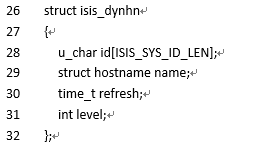
101 – 103 若事件为IF\_DOWN\_FROM\_Z（即 zebra 传递命令要求关闭某端口），则系统在log 中打印出警告信息。

## 动态主机名

### 概述

ISIS路由协议支持主机名动态更新机制。主机名缓存的实现在文件isis\_dynhn.[ch]中。节点从链路中获取的主机名保存在结构体isis\_dynhn中（如图5.3），这个结构体能够保存主机名，进程ID和节点的等级（level）。节点每遇到一个TLV，就会更新结构体中变量refresh的值，从而删除过时失效的主机名。需要注意的是，函数的print函数打印缓存中适配了进程ID的主机名而不是节点的进程ID。

isis\_dynhn.h

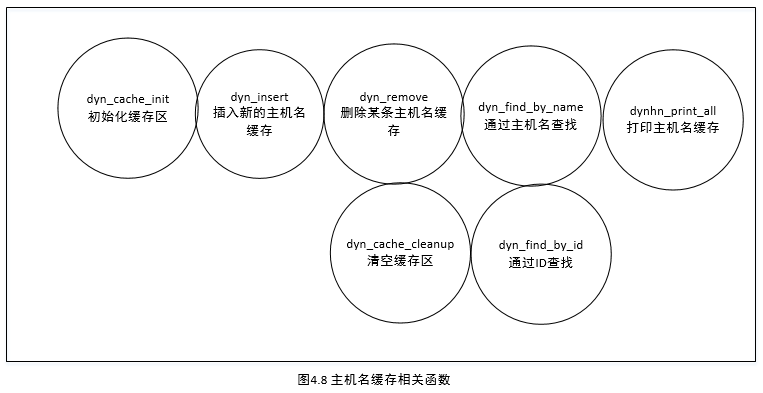


isis\_dynhn.h

图4.7 结构体isis\_dynhn的定义

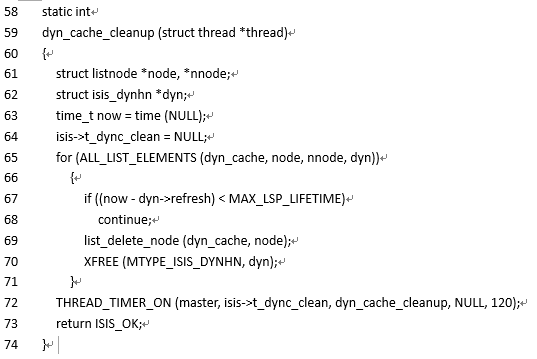
### cache相关函数

主机名动态更新机制的实现在文件isis\_dynhn.c 中。



Isis\_dynhn中一共有7个函数。其中，dyn\_cache\_cleanup, dyn\_cache\_init, dynh\_print\_all 函数是关于主机名缓存的处理函数； isis\_dynhn\_insert, isis\_dynhn\_remove 是处理结构体isis\_dynhn的函数，分别负责往缓存中插入和删除结构体；dynhn\_find\_by\_id, dynhn\_find\_by\_name 是在缓存中查找主机名缓存条的函数。

isis\_dynhn.c



isis\_dynhn.h

dyn\_cache\_cleanup 函数的作用是删除主机名缓存区内所有的缓存条目。它的参数是一个线程结构体。删除的操作将会在这个线程内异步进行。

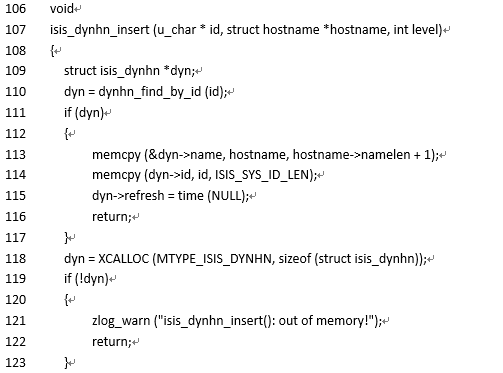
61 – 62 声明局部变量： 缓存条目结构体，和缓存区结构体

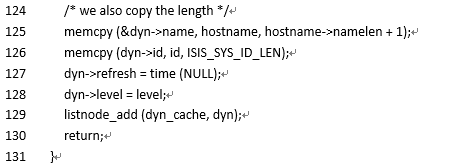
63 – 64 记录当前时间，并把isis 结构体的t\_dync\_clean值设置为空（意味着清理未完成）。

65 – 71 循环遍历缓存区链表上的缓存条目，依次删除后释放指针空间

函数dynhn\_insert 的作用是往主机名缓存区中插入的新主机名缓存条。它的三个参数分别是缓存条id，主机名名， 以及该主机名所在网络层级。函数会首先确认缓存区中是否已经存在该主机名条目，根据结果进行不同的插入处理。

isis\_dynhn.h





isis\_dynhn.h

109 – 110 声明局部结构体变量dyn存储dynhn\_find\_by\_id函数的返回结果。即函数在缓存 区中查找ID等于id的缓存条目，使用dyn保存查找结果。

111 – 117 如果缓存中已经包含ID 为id 的缓存条目，更新其保存的主机名，并记录当前时 间后函数返回。

118 – 123 如果缓存区中存在ID为id的缓存条目，则在缓存区中创建新的缓存条目，结果 保存在局部变量dyn 中。如果创建失败，则在log中打印出警告信息后返回。

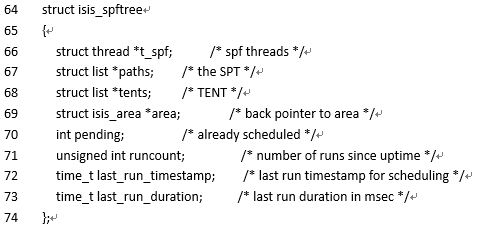
125 – 130 如果主机名缓存条目创建成功，则把参数传过来的信息拷贝到创建的缓存条目中， 并记录当前时间。把该缓存条目加入到缓存区中的条目链表后函数返回。

## SPT算法的实现

### 概述

中间节点必须定期执行最小生成树算法（SPF）来更新链路状态数据库或者邻域数据库。SPT算法的实现在文件isis\_spf.[ch] 中。每个域都要为所有的层（level）构建一个最小生成树（SPT）。SPT保存在结构体isis\_spftree中，如图4.9所示。这个结构体保存了定期运行SPF 算法的进程ID和上一次 SPF算法运行的时间戳。同时，也可以通过这个结构体连接到PATH和TENT数据库。注意，SPF 的运行频率应该小于系统配置变量minimumSPFInterval。参数pending和lastrun 用来规划SPF的持续运行，从而保证满足系统的需要。

isis\_spf.h

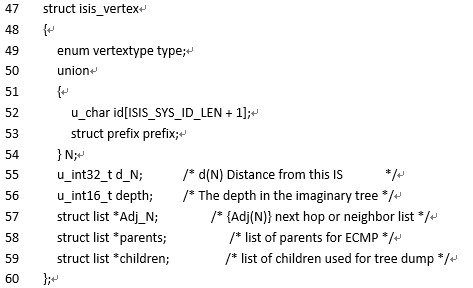


isis\_spf.h

图4.9 结构体isis\_spftree 的定义

PATH和TENT数据库中保存的内容是结构体isis\_vertex 中所描述的的网络节点。结构体中每个节点的包括一个指向节点本身来源（LSP）的指针和这个节点邻域（由一组与该节点具有同等花费连接的节点表示）。每个节点邻域的数量都被参数maximumPathSplits 限制。

isis\_spf.h



isis\_spf.h

图4.10 结构体isis\_vertex的定义

节点的类型由union体N 的值决定。对于中间节点和端节点来说，N保存的值是它们的系统ID。如果该节点是一个伪节点，N 还会追加一位非零值。如果节点描述的是一个网络可达地址，那么N 保存的就是该地址的网络地址和子网掩码。

在执行SPF算法时，节点到算法执行节点的距离(distance)保存在变量d\_N中，树的度（depth）保存在变量depth中。

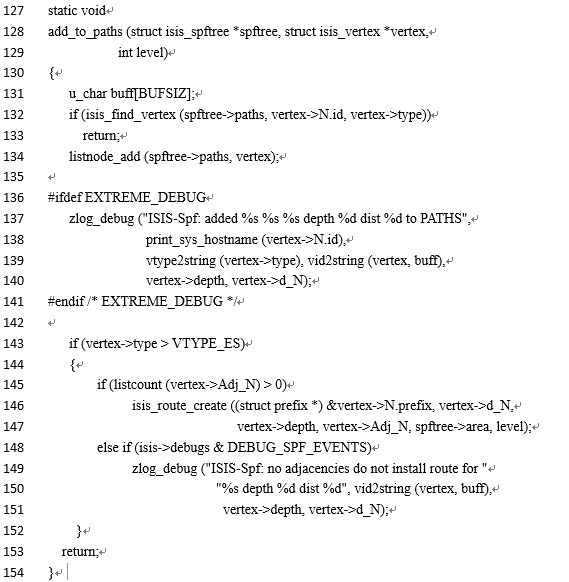
SPT算法的实现基于Annex C of ISO 10589[8]。在这个协议中，SPT 算法在基于Dijkstra 算法上增加了对多路同等花费路径的复用。这种对多路同等花费路径的复用的实现使用了负载均衡的技术，这意味着如果存在多条同等花费的路径到达目的地，系统会选择把数据流均衡的分布到这些路径中。

需要注意的是，如果节点同时支持IPV4和IPV6，那么该节点会分别运行SPT算法。

### add\_to\_path

函数add\_to\_path的功能是在节点运行SPF 算法时，把新的节点加入到最短路径树中（SPT）。函数首先检查SPT中是否已含有该节点，含有就直接返回；然后函数会检查新节点的类型是否是端节点，符合要求的话就会调用isis\_route\_create 函数为该节点创建路由并插入到SPT 中。

isis\_spf.c



isis\_spf.c

129 – 131 函数局部变量：isis\_spftree结构体spftree，它会定期运行SPF 算法的进程ID和上一次 SPF算法运行的时间戳 ；isis\_vertex 结构体vertex描述网络节点； level整形变量描述优先级。

132 – 134 isis\_find\_vertex函数如果找到了结构体spftree连接到的PATH数据库，vertex对应的网络节点编号及网络节点类型，则结束函数返回；若没有找到符合条件的，则利用listnode\_add将该节点添加进去。即找到SPT是否已经存在节点，存在则直接返回；不存在则添加该节点。

136 – 141 利用zlog\_debug进行终端调试，查看vertex描述的网络节点的id,类型，buff以及网络节点树的度（depth）的变量depth，节点的距离(distance)的变量d\_N是否正常；

143 – 147 条件如果vertex对应type> VTYPE\_ES,继续一个条件如果vertex->Adj\_N数量>0,则调用isis\_route\_create将结构体vertex的N.prefix, vertex的d\_N,vertex的depth, vertex的Adj\_N,及结构体spftree的area与 level添加进去。

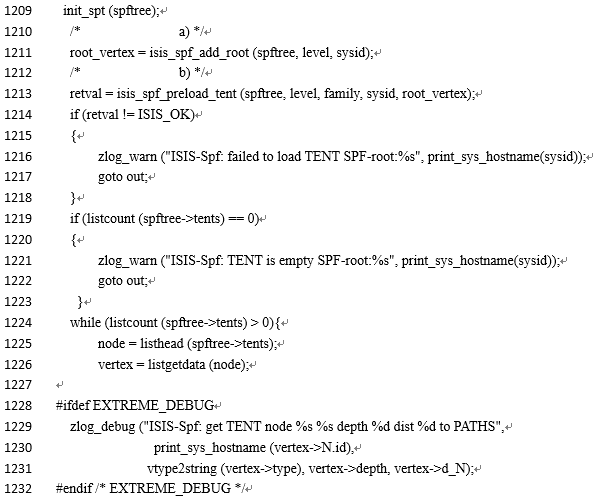
148 – 152 如果上述条件不符合，则又进行极限调试zlog\_debug。

153 – 153 执行完上述代码，函数返回。

### isis\_run\_spf函数

函数isis\_run\_spf 的功能就是在网络中运行Dijestra算法求得最短路径树。首先节系统会预加载节点数据库（TENT database），然后通过一个循环处理数据库中的节点并把产生的路径信息依次添加到路径数据库中（PATH database）。

isis\_spf.c





isis\_spf.c

1209 – 1213 init\_spt初始化构造spftree, root\_vertex为利用isis\_spf\_add\_root函数添加的节点，retval为利用 isis\_spf\_preload\_tent加载的节点数据库。

1214 – 1218 如果retval不等于ISIS\_OK，则利用zlog\_warn()函数输出异常信息，说明没有成功加载节点数据库。

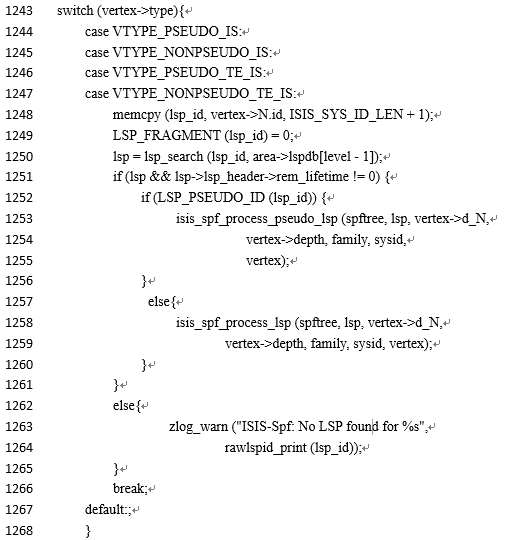
1219 – 1226 如果结构体spftree对应的tents数据库数量为0,利用zlog\_warn()函数输出异常信息，说明此时的tent数据库节点为空。

1228 – 1232 进行极限调试，输出各项相关有用信息。

1234 – 1236 执行完上述代码，删除节点，并把产生的路径信息依次添加到路径数据库

（PATH database）

isis\_spf.c



isis\_spf.c

1243 – 1247 switch结构，vertext对应的type有VTYPE\_PSEUDO\_IS, VTYPE\_NONPSEUDO\_IS, VTYPE\_PSEUDO\_TE\_IS, VTYPE\_NONPSEUDO\_TE\_IS四种类型

1248 – 1250 如果对应的是以上的四种类型之一，则继续执行，利用内存拷贝函数memcpy函数(的功能是从源src所指的内存地址的起始位置开始拷贝n个字节到目标dest所指的内存地址的起始位置中) vertex->N.id的起始位置开始拷贝 ISIS\_SYS\_ID\_LEN + 1个字节到lsp\_id所在的起始位置中。LSP\_FRAGMENT (lsp\_id) 赋值为0，lsp 定义为lsp\_search (lsp\_id, area->lspdb[level - 1])搜索到的lsp的编号以及area对应的数据库优先级。

1251 – 1261 条件判断，如果lsp不为空并且lsp对应的header对应的lefetime不为0，则继续执行。利用LSP\_PSEUDO\_ID函数分析lsp\_id，如果执行通过，将spftree, lsp, vertex->d\_N,vertex->depth, family, sysid,vertex信息添加到isis\_spf\_process\_pseudo\_lsp函数中；否则执行isis\_spf\_process\_lsp函数。

1260 – 1266 否则利用zlog\_warn输出异常信息

1267 – 1268 结束，如果vertex不是以上的四种类型，则走default结构,什么都不执行。

# 第五章 总结

ISIS作为一个分级的链路状态路由协议，基于DECnet PhaseV [路由算法](http://baike.baidu.com/view/2276401.htm" \t "_blank)，它使用Hello协议寻找临近[节点](http://baike.baidu.com/view/47398.htm" \t "_blank)，使用一个传播协议发送链接信息。ISIS可以在不同的[子网](http://baike.baidu.com/view/65511.htm" \t "_blank)上操作，包括广播型的LAN、WAN和点到点链路。ISIS是一个开放式协议，适用于中型及特大型网络。在分析该ISIS协议的过程中，我们深刻体会到了这一点，同时也感受到ISIS在不断发展中体现出在中型乃至大型的重要地位，也深深佩服发明ISIS协议以及不断发展该协议的人的伟大之处。

ISIS是一个在不断发展的协议，虽然IS-IS协议最初是用于OSI路由器协议的,但是,ISO(国际标准组织)开发IS-IS是为了支持无连接网络服务/无连接网络协议(CLNS/CLNP)｡支持IP协议的集成的IS-IS是后来开发的｡其目的是提供一个能够为无连接网络服务(CLNS)提供路由的单一的路由器协议｡目前ISP(互联网服务提供商)正在使用IS-IS｡我们小组认为这一点非常好，事物发展才能彰显强大的生命力，发展才能使一个协议始终紧跟时代的发展，这也是ISIS的一个独特的优势。

在分析代码的过程中，我们可以清晰地看到，ISIS整体的代码架构十分清晰，风格极其严谨，当然作为一份给大家学习的代码，它的注释也非常棒。每每看到这些，不禁令人叹为观止，实在是经典之作，巅峰之作。同时，各个代码段间函数功能也特别清楚，命名及其规范，各个模块之间充分体现了“高内聚，低耦合”的程序设计原则。通过阅读以及分析ISIS的代码，我们真的学到了很多，为我们的以后的代码整体布局与编写都做到了一个极好的模范作用。

当然，任何事物都不是完美的。在我们分析的过程中，我们发现了ISIS存在的两个技术问题：第一是ISIS使用一个小的[度量值](http://baike.baidu.com/view/466367.htm" \t "_blank)（6 比特），严重限制了能与它进行转换的信息；而且链接状态也只有8 比特长，[路由器](http://baike.baidu.com/view/1360.htm" \t "_blank)能通告的记录只有256个。第二是ISIS受OSI约束，使得与OSPF相比它的发展比较缓慢。这个限制的原因是由于SPF的要求；但Wide-metric使这个范围变成24位的扩展解决了这个问题。

我们可以看到，身处当代，随着人类的不断发展，技术的不断创新，无线网络技术的运用越来越广泛。而我们作为网络专业的学生，这正是我们应该研究的问题，同时我们也希望ISIS能继续不断发展，在今后的网络中发挥越来越重要的作用。