**网络协议栈分析与设计课程大作业**

|  |
| --- |
| **OLSR路由协议代码分析** |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | | 学号 | 姓名 | 班级 | 负责模块 | 成绩 | | 201692399 | 杨逐浪 | 软网1602 | 1.引言、代码介绍  2.数据结构、主要流程  3.进程汇报ppt  4.排版 |  | | 201692438 | 陶玉霖 | 软网1602 | 1.总结  2.路由协议流程代码分析  3.进程汇报  4.文献搜集 |  | |

目录

[第一章 引言 2](#_Toc533701523)

[第二章 代码介绍 3](#_Toc533701524)

[2.1 符号与缩写 3](#_Toc533701525)

[2.2 文件介绍 4](#_Toc533701526)

[2.3 全局变量 4](#_Toc533701527)

[第三章　数据结构 5](#_Toc533701529)

[3.1 OLSR首部 5](#_Toc533701530)

[3.2 HELLO消息 7](#_Toc533701531)

[3.3 TC消息 9](#_Toc533701532)

3.4 MID消息 11

[3.5 节点保存的信息表 12](#_Toc533701533)

3.5.1 本地链路表 12

[3.5.2 邻居信息表 12](#_Toc533701537)

[3.5.3 MPR Selector表 13](#_Toc533701537)

[3.5.4 网络拓扑表 13](#_Toc533701537)

[3.5.5 路由表 14](#_Toc533701537)

[第四章 OLSR路由协议的实现 15](#_Toc533701535)

[4.1 OLSR路由协议主要流程 16](#_Toc533701536)

[4.1.1 邻节点探测 16](#_Toc533701537)

[4.1.2 MPR节点的选择 16](#_Toc533701538)

[4.2 邻居表生成 19](#_Toc533701540)

[4.2.1 邻居表表项状态转移 19](#_Toc533701541)

[4.2.2 HELLO消息生成 19](#_Toc533701542)

[4.2.3 节点信息的存储 20](#_Toc533701543)

[4.2.4 邻居表的操作 22](#_Toc533701544)

[4.3 MPR表的生成 22](#_Toc533701545)

[4.4 泛洪TC分组 24](#_Toc533701548)

[4.4.1 TC消息的生成 24](#_Toc533701549)

[4.4.2 TC消息的处理 25](#_Toc533701550)

[4.5 路由计算与建立 27](#_Toc533701551)

[4.5.1 相关结构体和路由类型 28](#_Toc533701552)

[4.5.2 路由表计算 29](#_Toc533701553)

[第五章 总结 32](#_Toc533701555)

# 引言

OLSR(Optimized Link State Routing)是一个主要用MANET网络(Mobile Ad hoc network)的路由协议。OLSR由传统表驱动协议的改进而来，每个节点通过周期性的交换链路状态信息来维护整个网络的拓扑信息。

OLSR的特点是节点之间需要周期性交换各种控制信息，通过分布式计算来更新和建立自己的网络拓扑图。被邻节点选为多点中继站(MPR：Multipoint relay)的节点需要周期性地向网络广播控制信息，非MPR节点不需要转发控制信息。它主要采用两种控制消息分组，HELLO分组和TC(Topology Control)分组。这两种控制消息分组中都包含序列号，通过对序列号的比较，节点可以很容易的分辨出控制消息分组是否为最新的，而不会受到分组重传的影响。

OLSR路由协议分为核心功能集和扩展功能集两大部分。核心功能集定义了OLSR路由协议的基本功能，为独立MANET网络中运行OLSR路由协议的节点之间提供路由功能。扩展功能集是核心功能集的延伸，为在一-些特别场景下运行OLSR路由协议提供功能上的支持。

OLSR以路由跳数提供最优路径。这种协议尤其适合大而密集型的网络。

# 第二章 代码介绍

## 2.1 符号与缩写

|  |  |
| --- | --- |
| 符号 | 说明 |
| NSPU  NPDU  MAC  DSP  IDP  NET  MANET  MPR  MS  OLSR  TC  RSVP  QoS  MPR QoS | 网络服务数据单元  网络协议数据单元  媒体访问控制  特殊部分域  初始域部分  网络实体名称  移动Ad hoc网络  多点中继节点  MPR选择节点  优化的链路状态路由协议  拓扑控制  资源预留协议  服务质量  有QoS约束条件的多点中继节点 |

表2.1 OLSR路由协议符号缩写说明

## 2.2 文件介绍

OLSR路由协议中一共有123个源文件，下表给出了主要的文件并进行了说明：

|  |  |
| --- | --- |
| 文件名 | 说明 |
| Olsr\_ cfg. h  Packet. h  Lq\_ packet. h  Link\_set. h  Neighbor\_ table. h  Mpr\_selector\_set. h  Olsr. c  Mpr. c  Link\_ set. c  Neighbor\_ table. c  Two\_ hop\_ neighbor \_ table. c  Routing\_ table. c  TC\_ set. c  Mid\_set.c  Generate\_ msg. c  Process \_ package. c | 定义大部分常量，比如DEF \_ WILLINGNESS 等  消息数据包的结构定义  对OLSR, HELLO , TC数据包头等其他数据结构的定义  对存储节点信息的结构体的定义  对邻居信息数据结构体的定义  定义了结构体Mpr selector表示MPR选择源节点的集合  实现全局函数，比如网络拓扑结构的计算、路由表的计算与更新等  定义关于MPR的一些操作  定义邻居表的信息  表示对一跳邻居表的处理  表示对二跳邻居表的处理  实现对路由表的处理  实现对TC消息洪泛的处理  实现对MID消息的处理  表示HELLO、MID、TC、HNA消息的生成函数  实现对HELLO消息包的处理 |

表2.2 OLSR路由协议主要文件

## 2.3 全局变量

OLSR路由协议中的全局变量很多，下表给出了主要的全局变量并进行了说明：

|  |  |  |
| --- | --- | --- |
| 变量 | 数据类型 | 说明 |
| packet\_seq\_number  olsrport  hop\_count  rt\_ proto  willingness  ttl  use \_ hysteresis  min\_ tc\_ vtime  max\_ tc\_ vtime  max\_ jitter  changes\_topology  changes \_ neighborhood | nit16\_ t  nit16\_ t  nit8\_ t  nit8\_ t  nit8\_ t  nit8\_ t  bool  float  float  float  bool  bool | 发送数据包的确认号  OLSR消息发送、接收的端口号  站的计数，说明访问站数  路由表计算的所遵循的协议  WILL\_ ALWAY的邻居节点集合  IP允许通过的最大网段数量  判断消息是否迟滞  TC消息vtime的最小取值  TC消息vtime的最大取值  消息传播的最大抖动  判断拓扑信息是否变化  判断邻居信息是否变化 |

表2.3 OLSR路由协议中的部分全局变量

# 第三章　数据结构

OLSR使用统一的数据包格式，使用UDP通信，数据包嵌入到UDP数据报在网络上传输。每个数据包封装一个或多个消息，邮件分享通用报头格式，使节点能够正确接收和重传未知类型的消息。如果数据包不包含任何信息，数据包就会默默地被丟弃。每一个消息都会分配一个唯一的标识号， 用来确保消息不会被重传。

OLSR数据包一般包含以下几部分:消息类型，vtime ，数据包大小，发端地址，生存时间，跳数，消息序列号和消息。其中vtime是表示接收后很长时间节点如何，确保数据包中的消息有效;生存时间包含最大跳数，如果被重发，则减1;消息序列号被用来确保给定的消息被节点转发的次数不超过一次。

## OLSR首部

反序列化的OLSR头部如下所示：

Lq\_packet.h

54. struct olsr\_common {

55. uint8\_t type;

56. olsr\_reltime vtime;

57. uint16\_t size;

58. union olsr\_ip\_addr orig;

59. uint8\_t ttl;

60. uint8\_t hops;

61. uint16\_t seqno;

62. };

Lq\_packet.h

54- 62: olsr \_ .common是OLSR协议基本数据包。Type:消息类型; vtime :表示接收后多长时间节点可以确保数据包中的消息有效; size:消息大小; orig :发端地址; ttl:跳数，消息在传递过程中最大跳数，每转发一次，ttl减1; hops:此消息在传递过程中经历的跳数; seqno :消息的序列号，这是唯一不变的，以确保消息不回被重发。

OLSR 协议分组的基本格式如图3.1所示:

|  |  |  |
| --- | --- | --- |
| Packet Length  (16bits) | | Packet Sequence Number  (16bits) |
| Message Type  (8bits) | Vtime  (8bits) | Message Size  (16bits) |
| Originator Address  (32bits) | | |
| Time To Live  (8bits) | Hop Count  (8bits) | Message Sequence Number  (16bits) |

图3.1 OLSR路由协议分组基本格式

Packet Length:分组的长度，以bytes为单位。

Packet Sequence Number:分组的序列号，每产生一个新的分组该序列号加1。

Message Type:这个域用于说明分组中MESSAGE部分的类型。

Vtime:用于说明分组中所携带的信息的有效期。

Message Size:说明消息的长度，以bytes为单位。

Originator Address: 发送该消息的源节点，在消息的传输过程中这个域必须保持不变;

Time To Live:表示该消息被传送的最大跳数，每转发一次，该值减一，当节点收到一个TTL值为一的分组后，不再对其进行转发，通过设置TTL的值，节点可以控制分组洪泛的范围。

在OLSR协议中，主要采用了两种控制分组: HELLO分组和TC分组。HELLO分组消息执行链路检测、邻居发现的功能。TC分组消息执行MPR信息声明功能。这两种分组都作为基本分组格式中的MESSAGE部分。

## HELLO 消息

HELLO消息用于建立一个节点的邻居表，OLSR路由协议通过周期性地广播HELLO分组消息来发现邻居，建立邻居表。

Lq\_packet.h

98. struct lq\_hello\_message {

99. struct olsr\_common comm;

100. olsr\_reltime htime;

101. uint8\_t will;

102. struct lq\_hello\_neighbor \*neigh;

103. };

104.

105. /\* serialized LQ\_HELLO \*/

106. struct lq\_hello\_info\_header {

107. uint8\_t link\_code;

108. uint8\_t reserved;

109. uint16\_t size;

110. };

111.

112. struct lq\_hello\_header {

113. uint16\_t reserved;

114. uint8\_t htime;

115. uint8\_t will;

116. };

Lq\_packet.h

图3.2 HELLO消息数据包头部

98-103:结构体lq \_ hello\_ message，是消息数据包的头部。Htime：HELLO消息发射间隔; will：指定节点的意愿进行，有意愿的WILL\_ NEVER的节点被选为MPR的任意节点; neigh：消息传递的下一个节点。

105-116:结构体Iq\_hello\_ info\_ header 与结构体lq\_ hello\_ header 共同组成了HELLO消息数据包的头部。

|  |  |  |  |
| --- | --- | --- | --- |
| Reserved  (16bits) | | Htime  (8bits) | Willingness  (8bits) |
| Link Code  (8bits) | Reserved  (8bits) | Link Message Size  (16bits) | |
| Neighbor Interface Address  (32bits) | | | |
| ........... | | | |

图3.3 HELLO消息数据包头部

Reserved：保留字段，设置为“000000000000”。

Htime：此接口的HEELO消息的发送间隔。

Willingness：描述一个节点成为MPR的意愿。

Link Code：描述发送HELLO消息的节点与邻节点之间的链路类型。

Link Message Size：此链路消息的大小。

Neighbor Interface Address：邻居节点地址列表。

packet.h

58. struct hello\_message {

59. olsr\_reltime vtime;

60. olsr\_reltime htime;

61. union olsr\_ip\_addr source\_addr;

62. uint16\_t packet\_seq\_number;

63. uint8\_t hop\_count;

64. uint8\_t ttl;

65. uint8\_t willingness;

66. struct hello\_neighbor \*neighbors;

67.

68. };

packet.h

图3.4 HELLO消息数据包结构体

58-68:结构体hello\_message是消息数据包。Htime：HELLO 消息发射间隔。source\_ addr ：消息发送的原地址。packet\_ seq \_ number：数据包的序列号。hop\_ count：跳数。Ttl：数据包的生命周期。Willingness：指定节点的意愿进行，有意愿的WILL\_ NEVER的节点被选为MPR的任意节点。Neighbors：消息传递的下一个节点。

## TC消息

TC分组的作用是声明MPR信息。TC分组包含拓扑信息，节点通过该信息计算路由。网络中节点周期的发送TC分组，以声明自己的MPRSelector集，如果此节点没有被任何节点选作MPR，则它的MPRSelector集是空集，那么此节点不发送TC分组。节点的TC分组需要扩散至整个网络中的每个节点，这是与HELLO分组不同之处。HELLO分组只在产生HELLO分组的节点的--跳范围内传播。

Lq\_packet.h

118. /\* deserialized LQ\_TC \*/

119. struct lq\_tc\_message {

120. struct olsr\_common comm;

121. union olsr\_ip\_addr from;

122. uint16\_t ansn;

123. struct tc\_mpr\_addr \*neigh;

124. };

125.

126. /\* serialized LQ\_TC \*/

127.

128. struct lq\_tc\_header {

129. uint16\_t ansn;

130. uint8\_t lower\_border;

131. uint8\_t upper\_border;

132. };

Lq\_packet.h

图3.5 反序列化的TC分组头部

118-124: Iq \_ tc\_ message是封装后的拓扑数据包格式。from: 到达目的地的倒数第一跳地址。Ansn：记录本节点收到的最近一个TC分组的ANSN序列号。当收到一个新的TC分组时，将新的TC分组的ANSN号与拓扑表中的相对应的ANSN序列号比较，以此判断接收还是丢弃该消息。neigh: 指向广播邻居集的地址结构。

126-132: lq \_ tc\_ header是数据包的头部。Ansn：记录本节点收到的最近一个TC分组的ANSN序列号。lower \_ border表示下一级的边界，upper \_ border表示上一级的边界。

|  |  |
| --- | --- |
| ANSN  (16bits) | Reserved  (16bits) |
| MPR Selector Address  (32bits) | |
| MPR Selector Address  (32bits) | |

图3.6 TC分组格式

ANSN：当节点发现其邻居节点集发生变化时，将TC分组中的ANSN加一，其他节点收到一个TC分组时，可以通过对ANSN的比较来确定该分组是否是较新的分组。

Reserved：这个域作为保留域。

MPR Selector Address：MPR Selector节点的主地址，将所有的MPR Selector主地址封装在TC分组中。

packet.h

76. struct tc\_message {

77. olsr\_reltime vtime;

78. union olsr\_ip\_addr source\_addr;

79. union olsr\_ip\_addr originator;

80. uint16\_t packet\_seq\_number;

81. uint8\_t hop\_count;

82. uint8\_t ttl;

83. uint16\_t ansn;

84. struct tc\_mpr\_addr \*multipoint\_relay\_selector\_address;

85. };

packet.h

图3.7 TC消息数据包结构体

76-85: tc\_ message 是TC消息数据包格式。OLSR 路由协议利用拓扑表记录接收的TC消息内容。拓扑表包含四个部分：目的地址，到达目的地的倒数第二跳地址，ANSN序列号和表项有效时间。TC分组仅仅包含MPR Selector的地址，而不是所有邻居节点的地址。因为TC分组数据包必须通过MPR节点被广播到全网中，用以维护网络的拓扑信息、确保链路时刻的连通状态和更新路由表集。

## MID消息

MID(Multiple Interface Declaration)的主要作用是实现协议中多重OLSR接口之间的通信。

packet.h

101. struct mid\_message {

102. olsr\_reltime vtime;

103. union olsr\_ip\_addr mid\_origaddr;

104. uint8\_t mid\_hopcnt;

105. uint8\_t mid\_ttl;

106. uint16\_t mid\_seqno;

107. union olsr\_ip\_addr addr;

108. struct mid\_alias \*mid\_addr;

109. };

packet.h

图3.8 MID消息结构体

101-109：mid\_ message是MID消息数据包格式。Vtime根据MID\_HOLD\_TIME参数来设置。mid\_hopcnt：表示到目的地的跳数。mid\_ttl：在MID消息中TTL为225。mid\_seqno：表示序列号。olsr\_ip\_addr addr：MID消息数据包的主地址。

Mid\_set.c

65. int

66. olsr\_init\_mid\_set(void)

67. {

68. int idx;

69.

70. OLSR\_PRINTF(5, "MID: init\n");

71.

72. for (idx = 0; idx < HASHSIZE; idx++) {

73. mid\_set[idx].next = &mid\_set[idx];

74. mid\_set[idx].prev = &mid\_set[idx];

75.

76. reverse\_mid\_set[idx].next = &reverse\_mid\_set[idx];

77. reverse\_mid\_set[idx].prev = &reverse\_mid\_set[idx];

78. }

79.

80. return 1;

81. }

Mid\_set.c

图3.8 MID消息的初始化代码

65-81：表示了MID消息的初始化。其中使用了记录上一跳节点和下一跳节点的变量，reserve\_mid\_set[idx].prev表示的是上一跳的节点，reserve\_mid\_set[idx].next表示的是下一跳

的节点，表示了MID消息的通信对象。

## 节点保存的信息表

* + 1. **本地链路表**

本地链路信息表存储了该节点和邻居节点的链路信息。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| L\_local\_iface\_addr | L\_neighbor\_iface\_addr | L\_SYM\_time | L\_ASYM\_time | L\_time |

图3.9 本地链路表格式

L\_ local \_ iface \_ addr：本地节点的接口地址。

L\_ neighbor \_ iface\_ addr：邻节点的接口地址。

L\_ SYM\_ TIME：直到此时刻前，链路被认为是对称的。

L\_ ASYM\_ TIME：直到此时刻前，链路被认为是单向的。

L\_time：链路维护时刻，链路在该时刻失效，必须被删除，当L\_SYM\_TIME和L\_ ASYM \_TIME 都过期，链路被声明为丢失。

* + 1. **邻居信息表**

节点根据接收和发送获得邻居节点的信息。

|  |  |  |
| --- | --- | --- |
| N\_neighbor\_main\_addr | N\_status | N\_willingness |

图3.10 邻居表格式

N \_ neighbor\_ main\_ addr: 节点的一跳邻居地址。

N \_ status：节点与其一跳邻居之间的链路状态。

N \_ willingness ：表示邻居节点为其他节点转发分组的意愿程度。

节点存储一个两跳邻居表，表示邻居节点与对称两跳邻居节点间的对称链路。

|  |  |  |
| --- | --- | --- |
| N\_neighbor\_main\_addr | N\_2hop\_addr | N\_time |

图3.11 两跳邻居表格式

N \_ neighbor\_ main\_ addr: 节点的一跳邻居地址。

N\_ 2hop\_ addr: 表示与N \_neighbor\_ main\_addr有对称链路的两跳邻节点的地址。

N\_ time：表示表项到期必须被移除的时间。

* + 1. **MPR Selector表**

节点通过维护关于MPR Selector的信息来判断需要转发什么控制消息。

|  |  |
| --- | --- |
| MS\_main\_addr | MS\_time |

图3.12 MPR Selector表格式

MS\_ addr：表示 MPR selectors 的主地址；

MS\_time：表示了这条记录的有效时间。

* + 1. **网络拓扑表**

OLSR 路由协议中每个节点维持一张网络拓扑表，用于描述当前时刻的网络拓扑结构和计算路由。

|  |  |  |  |
| --- | --- | --- | --- |
| T\_dest\_addr | T\_last\_addr | T\_seq | T\_time |

图3.13 网络拓扑表格式

T\_dest\_addr：拓扑表中目的节点地址域保存所要到达的目的节点地址。

T\_last：被选为拓扑表中的节点的地址。

T\_seq：表项序列号，用于记录本节点收到的最后一个TC分组的序号，当收到一个新的TC

分组时，将新的 TC分组的序列号与表项序列号相比较来决定接收还是丢弃。

T\_time：表项有效时间，用于表示该表项生存时间，超过生存时间的表项不能用于路由计

算，必须删除。

* + 1. **路由表**

网络中每个节点维护一个路由表，表中保存了节点到网路中所有可达目的节点的路由，对于路由已知的网络中的每一个目的地，表项被存储在路由表中。

|  |  |  |  |
| --- | --- | --- | --- |
| R\_dest\_addr | R\_next\_addr | R\_dist | R\_iface\_addr |

图3.14 路由表格式

R\_ dest\_ addr: 路由目的节点地址。

R\_ next\_ addr: 路由的下一跳节点地址。

R\_dist：本节点到目的节点的距离。

R\_ iface\_ addr：表示下一跳节点通过本地接口R \_ iface\_ addr 到达。

# 

# 第四章 OLSR路由协议的实现

## OLSR路由协议主要流程

OLSR路由协议就是通过HELLO分组的周期性交互，执行链路检测、邻居发现的功能。通过TC分组的周期性交互执行MPR信息声明功能。最终以这些分组所建立起来的拓扑结构为基础，进行基于MPR的路由计算。下面是其主要流程：

路由协议初始化

邻居节点发现

MPR节点的计算

泛洪TC分组

拓扑表的建立

路由表的计算和建立

图4.1 OLSR路由协议主要流程

下面将对邻节点的探测和MPR节点的选择做简单的描述：

1. **邻节点探测**

基于本地链路信息表，节点建立邻居信息表，存储邻居节点信息，两跳邻居节点信息，MPR信息，MPR Selector。本地链路信息表和邻居信息表有着密切的联系，邻居信息表随着本地链路信息表内容的变化而变化，通过节点之间交换HELLO消息来维护。

当节点收到HELLO消息时，建立自己的邻居节点集，如果发送HELLO消息的邻居节点是自己的对称邻居节点：

对于HELLO消息中包含的类型为SYM\_ NEIGH和MPR\_ NEIGH的地址，如果该地址等于接受节点的地址，则不对该地址进行处理，否则，建立一条新的两跳邻居节点的记录来取代旧的记录：N\_neighbor\_addr = Originator Address，N\_2hop\_addr = address of the 2-hop nei ghbor，N\_time = current time+ validity time。

对于HELLO 消息中包含的类型为NOT NEIGH的地址，所有对应于N\_neighbor\_addr = Oniginator Address并且N\_ 2hop\_ addr = address of the 2-hopneighbor的记录将被删除。

1. **MPR节点的选择**

OLSR 路由协议与传统的链路状态路由协议最重要的区别在于MPR节点的引入，MPR 机制大幅度地降低了路由控制消息的洪泛规模，同时能够有效地将路由控制消息洪泛到全网。

MPR机制是为了最大程度的减少消息流的开销，通过减少对相邻节点之间的冗余传播消息，从而达到减少消息开销的目的。网络中所有节点都会在它的一跳对称邻居节点中选择一组节点作为MPR节点组，该MPR组会重传节点的消息。而不是该节点MPR的邻居节点只会接受和处理节点发出的广播消息，而不会重传它接受的消息。MPR 集合越小，则路由协议的控制流开销也越少。

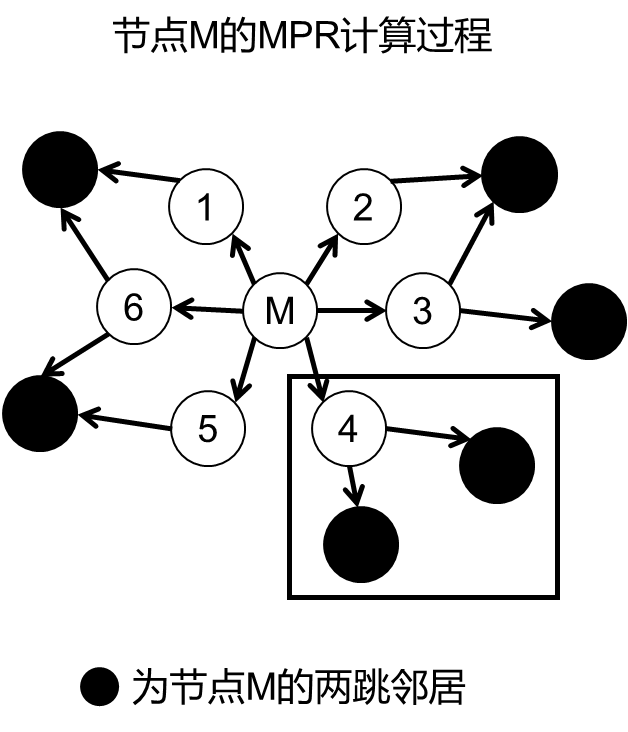


图4.2 节点M的MPR计算

1.从节点M的邻居节点中选择这样的节点——M节点必须通过此节点才能到达某些两跳邻居节点，将这个节点加入MPR，同时将能够通过MPR到达的两跳邻居节点从两跳邻居节点集中移除。

对应此图，节点M必须通过节点4才能到达右下角两个两跳邻居节点，节点4加入MPR，同时将右下角的两个两跳邻居节点从两跳邻居节点集中移除。

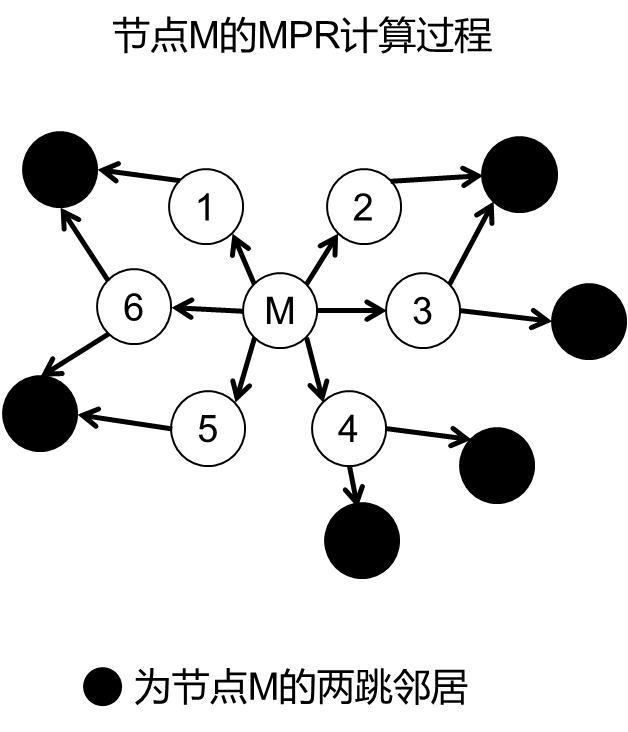


图4.3 节点M的MPR计算

2.如果此时在两跳邻居节点集中还存在没有被MPR覆盖的两跳邻居节点，则重复下面步骤，直到所有的两跳邻居节点都能通过MPR到达：

在M节点的邻居节点中选择覆盖度最大的节点，如有多个节点最大，选择其中连接度最高的节点加入MPR，同时将能够通过MPR到达的两跳邻居节点从两跳邻居节点集中移除。

（节点的覆盖度：该节点能够覆盖的两跳邻居节点集中没有被MPR覆盖的两跳邻居节点个数。节点的连接度：该节点以对称链路连接的两跳邻居节点个数。）

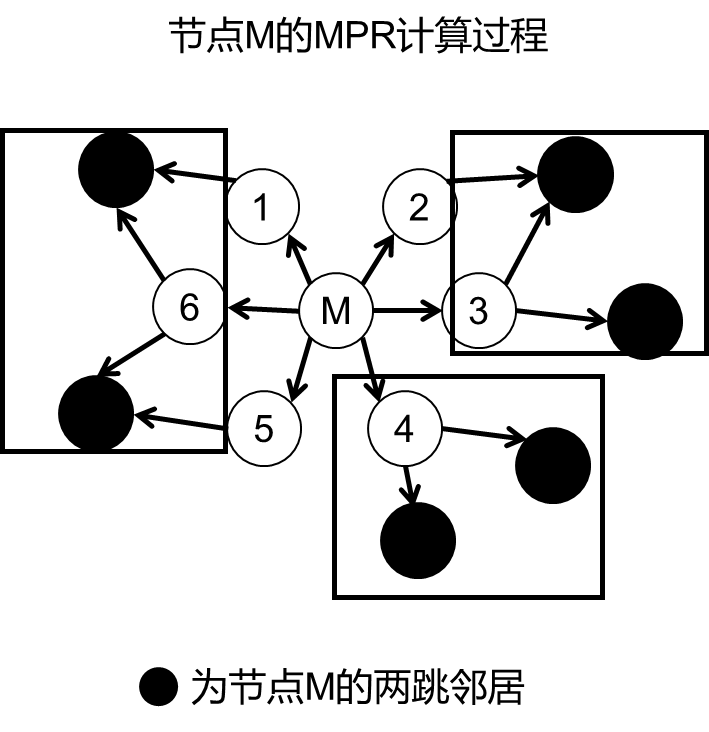


图4.4 节点M的MPR计算

对应此图，此时两跳邻居节点集中还存在没有被MPR覆盖的两跳邻居节点，则选择具有最大覆盖度的节点3和节点6，它们的覆盖度均为2，又因为它们的连接度同为2，所以将节点3和节点6同时加入MPR。

此时，两跳邻居节点集中的节点均被MPR覆盖，节点M的MPR计算过程完成。节点3，节点4，节点6成为节点M的MPR，使节点M通过MPR能到达所有的两跳邻居节点。

## 邻居表的生成

1. **邻居表表项状态转移**

邻居发现是基于节点的邻居信息库，通过 HELLO消息的传播实现。邻居信息库包含关于邻居、两跳邻居、MPRs和MPR的信息。

为了检测本节点与周围邻居的链路情况，OLSR协议规定各个节点周期性地发送HELO消息。每隔" Hello intenal"周期，移动节点就会广播一次 HELLO消息，来更新网络拓扑信息。 HELLO消息包含着本节点的邻居及其链路状态信息。通过 HELLO消息周期性地交互，各个节点不仅知道自己的邻居信息，还能知道两跳邻居。拥有邻居和两跳邻居信息后，移动节点独立选择本节点的MPR为邻居节点集中，能覆盖所有两跳邻居节点的个数最少的那些节点。HELO消息只在一跳范围内广播，邻居节点收到 HELLO消息处理后立即丢弃，不再进行转发。

1. **HELLO消息的生成**

Generate\_msg.c

66.void

67.generate\_hello(void \*p)

68.{

69. struct hello\_message hellopacket;

70. struct interface \*ifn = (struct interface \*)p;

71.

72. olsr\_build\_hello\_packet(&hellopacket, ifn);

73.

74. if (queue\_hello(&hellopacket, ifn))

75. net\_output(ifn);

76.

77. olsr\_free\_hello\_packet(&hellopacket);

78.

79.}

Generate\_msg.c

66-79: generate\_hello函数用于产生一个消息，再调用queue\_hell函数将生成带有参数 hellopacket内容的HELLO包，是发送消息的接口，如果创建成功，则通过net\_output函数

将此包通过给定接口ifn发送出去，最后调用函数释放消息。

1. **节点信息的存储**

Link\_set.h

58. struct link\_entry {

59. union olsr\_ip\_addr local\_iface\_addr;

60. union olsr\_ip\_addr neighbor\_iface\_addr;

61. const struct interface \*inter;

62. char \*if\_name;

63. struct timer\_entry \*link\_timer;

64. struct timer\_entry \*link\_sym\_timer;

65. uint32\_t ASYM\_time;

66. olsr\_reltime vtime;

67. struct neighbor\_entry \*neighbor;

68. uint8\_t prev\_status;

69.

70. /\*

71. \* Hysteresis

72. \*/

73. float L\_link\_quality;

74. int L\_link\_pending;

75. uint32\_t L\_LOST\_LINK\_time;

76. struct timer\_entry \*link\_hello\_timer;

77. olsr\_reltime last\_htime;

78. bool olsr\_seqno\_valid;

79. uint16\_t olsr\_seqno;

80.

81. /\*

82. \* packet loss

83. \*/

84. olsr\_reltime loss\_helloint;

85. struct timer\_entry \*link\_loss\_timer;

86.

87. /\* user defined multiplies for link quality, multiplied with 65536 \*/

88. uint32\_t loss\_link\_multiplier;

89.

90. /\* cost of this link \*/

91. olsr\_linkcost linkcost;

92.

93. struct list\_node link\_list;

94. uint32\_t linkquality[0];

95.};

Link\_set.h

59-62:local\_iface\_addr存储该节点接口IP地址，neighbor\_iface\_ addr存储邻居节点IP地址。也定义了一些结构体存储了接口其他相关信息。

63-68:时间相关信息，包括link\_timer定时器，lk\_sym\_timer定时器 因vimer表示接收后多长时间节点可以确保数据包中的消息有效。neighbo以链表形存储邻居节点信息，prev\_status存储上一个节点的状态。

73-79:是一种滞后作用，L\_link\_ quality记录链路质量，Link\_hello\_ timer是结构体 timer\_entry类型的指针，用以记录何时收到一个HELLO消息 。

81-95:记录数据包丢失的相关信息，并且由loss\_link\_ multiplier存储用户定义的链路质量倍数，由 linkcost存储链路的代价值。

Neighbor\_table.h

58. struct neighbor\_entry {

59. union olsr\_ip\_addr neighbor\_main\_addr;

60. uint8\_t status;

61. uint8\_t willingness;

62. bool is\_mpr;

63. bool was\_mpr;

64. bool skip;

65. int neighbor\_2\_nocov;

66. int linkcount;

67. struct neighbor\_2\_list\_entry neighbor\_2\_list;

68. struct neighbor\_entry \*next;

69. struct neighbor\_entry \*prev;

70.};

Neighbor\_table.h

58-70: neighbor\_ entry结构体，用来存储邻居节点的信息。记录了邻居节点的主地址状态，作为MPR的 willingness值，是否是MPR，是否曾是MPR，覆盖的两跳邻居节点的数量，节点连接链路的数量以及指向 neighbor\_2\_list链表的指针。其中，成员变量was\_mpr用来发现MPR的变化。

1. **邻居表的操作**

Neighbor\_table.c

56.void

57.olsr\_init\_neighbor\_table(void)

58.{

59. int i;

60.

61. for (i = 0; i < HASHSIZE; i++) {

62. neighbortable[i].next = &neighbortable[i];

63. neighbortable[i].prev = &neighbortable[i];

64. }

65.}

Neighbor\_table.c

61-64: 函数功能是初始化邻居表。将毎一个邻居表 neighbortable[i]初始化为指向自身的仅有一个节点的链表。

## MPR表的生成

Mpr.c

357.static uint16\_t

358.add\_will\_always\_nodes(void)

359.{

360. struct neighbor\_entry \*a\_neighbor;

361. uint16\_t count = 0;

362.

363.#if 0

364. printf("\nAdding WILL ALWAYS nodes....\n");

365.#endif

366.

367. OLSR\_FOR\_ALL\_NBR\_ENTRIES(a\_neighbor) {

368. struct ipaddr\_str buf;

369. if ((a\_neighbor->status == NOT\_SYM) || (a\_neighbor->willingness != WILL\_ALWAYS)) {

370. continue;

371. }

372. olsr\_chosen\_mpr(a\_neighbor, &count);

373.

374. OLSR\_PRINTF(3, "Adding WILL\_ALWAYS: %s\n", olsr\_ip\_to\_string(&buf, &a\_neighbor->neighbor\_main\_addr));

375.

376. }

377. OLSR\_FOR\_ALL\_NBR\_ENTRIES\_END(a\_neighbor);

378.

379.#if 0

380. OLSR\_PRINTF(1, "Count: %d\n", count);

381.#endif

382. return count;

383.}

Mpr.c

367-383：函数作用是添加 willingness为WLL\_ALWAYS的邻居节点到MPR集合中。首先将非对称的邻居节点或是不是WLL\_ ALWAYS的部居节点忽略，然后将剩余的节点a \_neighbor添加到mpr中，并返回添加节点的数量Count。

Mpr.c

236.static void

237.olsr\_clear\_mprs(void)

238.{

239. struct neighbor\_entry \*a\_neighbor;

240. struct neighbor\_2\_list\_entry \*two\_hop\_list;

241.

242. OLSR\_FOR\_ALL\_NBR\_ENTRIES(a\_neighbor) {

243.

244. /\* Clear MPR selection. \*/

245. if (a\_neighbor->is\_mpr) {

246. a\_neighbor->was\_mpr = true;

247. a\_neighbor->is\_mpr = false;

248. }

249.

250. /\* Clear two hop neighbors coverage count/ \*/

251. for (two\_hop\_list = a\_neighbor->neighbor\_2\_list.next; two\_hop\_list != &a\_neighbor->neighbor\_2\_list;

252. two\_hop\_list = two\_hop\_list->next) {

253. two\_hop\_list->neighbor\_2->mpr\_covered\_count = 0;

254. }

255.

256. }

257. OLSR\_FOR\_ALL\_NBR\_ENTRIES\_END(a\_neighbor);

258.}

Mpr.c

245-248：如果节点a\_neighbor的is\_mpr值为真的话，那么就将其置为假，代表其不是MPR节点，同时将was\_mpr置为真，代表该邻居节点曾经被选为MPR。

251-254：遍历邻居节点a\_ neighbor 覆盖的两跳邻居节点的数量置为0。

## 泛洪TC分组

网络中的MPR节点每隔一段时间就向全网洪泛广播TC( Topology Control)消息，来维护网络的拓扑信息。为了减少洪泛，OLSR规定，对于相同的TC消息，节点只在第一次收到并且选择其作为MPR的情况下转发，避免了广播风暴。TC分组仅仅包含( MPR selector)(将本节点选为MPR节点的邻居节点)的地址，而不是所有邻居节点地址。节点通过TC消息的扩散获得全网拓扑图，再根据两跳邻居表和拓扑表，独立地按照Dijkstra算法计算出路由表。

* + 1. **TC消息的生成**

Generate\_msg.c

81.void

82.generate\_tc(void \*p)

83.{

84. struct tc\_message tcpacket;

85. struct interface \*ifn = (struct interface \*)p;

86.

87. olsr\_build\_tc\_packet(&tcpacket);

88.

89. if (queue\_tc(&tcpacket, ifn) && TIMED\_OUT(ifn->fwdtimer)) {

90. set\_buffer\_timer(ifn);

91. }

92.

93. olsr\_free\_tc\_packet(&tcpacket);

94.}

Generate\_msg.c

81-94：MID消息通过olsr\_build\_\_ packet()函数生成之后放在MID队列中。当时间戳期满的时候，调用set\_ buffer\_ timer()设置定时器。最后从给定接口ifn释放消息，同时调用olsr\_free\_tc\_ packet()释放所占内存。

为了构建拓扑信息库，每个被选择为MPR的节点必须广播拓扑控制(TC)消息，这些通过TC消息扩散到网络中的信息将有所帮助每个节点计算其路由表。并且TC消息必须是根据“默认转发算法”进行转发的。MPRs使得拓扑信息的分布具有更好的可扩展性。

当节点的通告链路集变为空时，该节点应当在等于其先前发送的TC消息的无效，直到有节点加入到通告链路集。“有效时间”的持续时间期间仍然发送(空)TC消息，以便使先前的TC消息无效，直到有节点加入到通告链路集。

节点可以传送附加的TC消息以增加其链接故障的反应性。当检测到对MPR选择器集合发生改变并且这种改变可以归因于链路故障时，TC消息应当在短于TC\_ INTERVAL的时间间隔内被发送。

* + 1. **TC消息处理**

Tc\_set.c

810. /\* We are only interested in TC message types. \*/

811. pkt\_get\_u8(&curr, &type);

812. if ((type != LQ\_TC\_MESSAGE) && (type != TC\_MESSAGE)) {

813. return false;

814. }

815.

816. /\*

817. \* If the sender interface (NB: not originator) of this message

818. \* is not in the symmetric 1-hop neighborhood of this node, the

819. \* message MUST be discarded.

820. \*/

821. if (check\_neighbor\_link(from\_addr) != SYM\_LINK) {

822. OLSR\_PRINTF(2, "Received TC from NON SYM neighbor %s\n", olsr\_ip\_to\_string(&buf, from\_addr));

823. return false;

824. }

Tc\_set.c

811-814：当节点接收到TC消息时，只关心其消息类型。当其类型不等于LQ\_TC\_ MESSAGE或者TC\_ MESSAGE时，直接返回 false，将包丢弃。

816-820：TC消息接收者在接收到消息时判断发送者接回信息，若发送者并非是对称一跳邻居，那么该包将会被丢弃。

821-824：一旦接收到TC消息，必须根据消息头的Vtime字段计算“有效时间”。

Tc\_set.c

848. if (olsr\_seq\_inrange\_high((int)tc->msg\_seq - TC\_SEQNO\_WINDOW, tc->msg\_seq, msg\_seq)

849. && olsr\_seq\_inrange\_high((int)tc->ansn - TC\_ANSN\_WINDOW, tc->ansn, ansn)) {

850.

851. /\*

852. \* Ignore already seen seq/ansn values (small window for mesh memory)

853. \*/

854. if ((tc->msg\_seq == msg\_seq) || (tc->ignored++ < 32)) {

855. return false;

856. }

858. OLSR\_PRINTF(1, "Ignored to much LQTC's for %s, restarting\n", olsr\_ip\_to\_string(&buf, &originator));

859.

860. }

Tc\_set.c

854-858：如果该消息中的msg\_seq和外部变量msg\_seq相等且ignored变量小于32，则代表该消息已经处理过，应该忽视。

Tc\_set.c

882.if (!tc) {

883. tc = olsr\_add\_tc\_entry(&originator);

884. }

885.

886. /\*

887. \* Update the tc entry.

888. \*/

889. tc->msg\_hops = msg\_hops;

890. tc->msg\_seq = msg\_seq;

891. tc->ansn = ansn;

892. tc->ignored = 0;

893. tc->err\_seq\_valid = false;

894.

895. OLSR\_PRINTF(1, "Processing TC from %s, seq 0x%04x\n", olsr\_ip\_to\_string(&buf, &originator), tc->msg\_seq);

Tc\_set.c

882-895：如果拓扑表中不存在和TC消息中消息产生这地址字段相同的条目，则添加新的条目并且保存序列号，之后根据之后获取的TC消息数据包的头部信息更新tc\_entry。

## 路由计算与建立

节点通过TC消息的扩散获得全网拓扑图，再根据邻居表、两跳邻居表和拓扑表，独立地按照 Dijkstra算法计算出路由表。每个节点都有一张路由表，通过路由表寻找路径信息。对于路由已知的网络中的每个目的地，将路由信息记录在路由表中。所有路由被破坏或仅部分已知的目的地的路由信息不被记录在表中。每一条路由信息都包含信息目的地址、下一跳地址、总跳数、下一跳接口地址。

路由表的更新既不在网络中，也不在一跳邻居域中生成或触发任何消息。操作系统的路由体系结构按功能可以分成两个部分。一部分是负责与其它节点交换信息，计算到其它节点的正确路由，称之为“路由功能模块”；一部分则是根据内核路由表，将需要发送到网络中的数据分组，诵过正确的网络接口发送到下一跳节点，称为“转发功能模块”。这样，操作系统就可以在“转发功能模块”保持不变的情况下，通过修改“路由功能模块”，从而实现不同的路由协议。

OLSR协议的实现通过端口号为698的UDP端口收发路由控制分组，然后维护邻居表，进行逻辑计算，最后生成路由表并反映到内核路由表中。数据分组和协议控制分组按照内核路由表中的最佳匹配表项进行发送和转发。当网络中有分组到达本节点时，内核将判断该分组的目的地是否是自己，如果不是，则“转发功能模块”根据内核路由表转发该分组；如果是，则根据分组的不同交给相应的模块进行处理，当收到OLSR协议控制分组时，转由OLSR路由协议模块处理。

1. **相关结构体和路由类型**

Routing\_table.h

66.struct rt\_metric {

67. olsr\_linkcost cost;

68. uint32\_t hops;

69.};

70.

71./\* a nexthop is a pointer to a gateway router plus an interface \*/

72.struct rt\_nexthop {

73. union olsr\_ip\_addr gateway; /\* gateway router \*/

74. int iif\_index; /\* outgoing interface index \*/

75.};

Routing\_table.h

66-69：在路由选择的时候使用的复合矩阵，其中包括两个路由点之间的花销或者跳数。

72-75：该结构体包含下一跳的网关(IPv4或IPv6)与接口索引。

Routing\_table.h

84.struct rt\_entry {

85. struct olsr\_ip\_prefix rt\_dst;

86. struct avl\_node rt\_tree\_node;

87. struct rt\_path \*rt\_best; /\* shortcut to the best path \*/

88. struct rt\_nexthop rt\_nexthop; /\* nexthop of FIB route \*/

89. struct rt\_metric rt\_metric; /\* metric of FIB route \*/

90. struct avl\_tree rt\_path\_tree;

91. struct list\_node rt\_change\_node; /\* queue for kernel FIB add/chg/del \*/

92.};

Routing\_table.h

84-92：每一个RIB节点都会有一个路由的接口，这个接回很重要，里面包含了最佳路径的下一个网关信息，而且是rt\_Path\_tree的根，同样也包含了个在所有路由信息里的一个最好的路径。rt\_dst包含了该信息的路由地址与前级长度。rt\_tree\_node包含了val\_tree的一些信息。该结构体里面也包含了父节点、子节点、左右节点、表示一棵树的信息值。

1. **路由表计算**

函数的功能是创建一个可用的路由条目，对于提供的参数IP前缀分配一个路由条目空间，并做一些相应的初始化并把该入口插入到avl树里。

Routing\_table.c

167.void

168.olsr\_init\_routing\_table(void)

169.{

170. OLSR\_PRINTF(5, "RIB: init routing tree\n");

171.

172. /\* the routing tree \*/

173. avl\_init(&routingtree, avl\_comp\_prefix\_default);

174. routingtree\_version = 0;

175.

176. /\*

177. \* Get some cookies for memory stats and memory recycling.

178. \*/

179. rt\_mem\_cookie = olsr\_alloc\_cookie("rt\_entry", OLSR\_COOKIE\_TYPE\_MEMORY);

180. olsr\_cookie\_set\_memory\_size(rt\_mem\_cookie, sizeof(struct rt\_entry));

181.

182. rtp\_mem\_cookie = olsr\_alloc\_cookie("rt\_path", OLSR\_COOKIE\_TYPE\_MEMORY);

183. olsr\_cookie\_set\_memory\_size(rtp\_mem\_cookie, sizeof(struct rt\_path));

184.}

Routing\_table.c

173-174：调用avl\_init()初始化一个avl树。维护一个版本号rountingtree\_version用以检测每一个rt\_entry和rt\_Path子树中过时的信息，并在此处将该值初始化为0。

179-183：为rt\_entry和rft\_path分配内存，创建相应的 cookie。

Routing\_table.c

231.static struct rt\_entry \*

232.olsr\_alloc\_rt\_entry(struct olsr\_ip\_prefix \*prefix)

233.{

234. struct rt\_entry \*rt = olsr\_cookie\_malloc(rt\_mem\_cookie);

235. if (!rt) {

236. return NULL;

237. }

238.

239. memset(rt, 0, sizeof(\*rt));

240.

241. /\* Mark this entry as fresh (see process\_routes.c:512) \*/

242. rt->rt\_nexthop.iif\_index = -1;

243.

244. /\* set key and backpointer prior to tree insertion \*/

245. rt->rt\_dst = \*prefix;

246.

247. rt->rt\_tree\_node.key = &rt->rt\_dst;

248. avl\_insert(&routingtree, &rt->rt\_tree\_node, AVL\_DUP\_NO);

249.

250. /\* init the originator subtree \*/

251. avl\_init(&rt->rt\_path\_tree, avl\_comp\_default);

252.

253. return rt;

254.}

Routing\_table.c

231-237：申请内存空间并把空间内清零。

239-245：标识该入口为新分配入口并把该入口的目的地址设置成为参数提供的入口地址。

247-254：把入口的树节点插入到整个路由表中并初始化树。

Routing\_table.c

292.void

293.olsr\_insert\_rt\_path(struct rt\_path \*rtp, struct tc\_entry \*tc, struct link\_entry \*link)

294.{

295. struct rt\_entry \*rt;

296. struct avl\_node \*node;

297.

298. /\*

299. \* no unreachable routes please.

300. \*/

301. if (tc->path\_cost == ROUTE\_COST\_BROKEN) {

302. return;

303. }

304.

305. /\*

306. \* No bogus prefix lengths.

307. \*/

308. if (rtp->rtp\_dst.prefix\_len > olsr\_cnf->maxplen) {

309. return;

310. }

311.

312. /\*

313. \* first check if there is a route\_entry for the prefix.

314. \*/

315. node = avl\_find(&routingtree, &rtp->rtp\_dst);

316.

317. if (!node) {

318.

319. /\* no route entry yet \*/

320. rt = olsr\_alloc\_rt\_entry(&rtp->rtp\_dst);

321.

322. if (!rt) {

323. return;

324. }

325.

326. } else {

327. rt = rt\_tree2rt(node);

328. }

329.

330. /\* Now insert the rt\_path to the owning rt\_entry tree \*/

331. rtp->rtp\_originator = tc->addr;

332.

333. /\* set key and backpointer prior to tree insertion \*/

334. rtp->rtp\_tree\_node.key = &rtp->rtp\_originator;

335.

336. /\* insert to the route entry originator tree \*/

337. avl\_insert(&rt->rt\_path\_tree, &rtp->rtp\_tree\_node, AVL\_DUP\_NO);

338.

339. /\* backlink to the owning route entry \*/

340. rtp->rtp\_rt = rt;

341.

342. /\* update the version field and relevant parameters \*/

343. olsr\_update\_rt\_path(rtp, tc, link);

344.}

Routing\_table.c

292-310：创建参数变量检查参数是否符合要求，如果传入的tc\_entry为ROUTE\_COST\_ BROKEN或者传入的rtp的目的地址长度大于所设置的最大 地址长度则直接返回。

315-337：调用avl\_find()函数检查传入的rtp的节点是否在路由表中，如果节点不在路由表的avl树中则重新分配一个节点，如果在就把节点类型从avl\_node类型转化成rt\_entry类型。

340-344：把新节点添加进avl树里，然后再改变相应的参数，更新整个路由表。

# 第五章 总结

通过对代码的分析，我们知道了OLSR协议的优点是它是一种先应式路由协议，具有查找路由时延小的优点。采用MPR机制，只有MPR节点负责向全网泛洪TC控制消息，从而减少了协议开销。它也采用最短路径选路算法进行选路，缩减了控制分组的大小。同时OLSR路由协议的局限在于核心MPR机制只考虑覆盖度 ，忽略了链路质量的影响。仍有改进的空间。

我们对它实现这些优点的方式产生了兴趣，这也让我们能有动力去分析大部分是完全陌生的代码。在代码分析和完成实验报告的过程中，我们也明白了研究方法的重要性。因为在阅读代码的初期我们遇到了理解性的困难，找不到突破口。我们组于是从基础出发。在查阅资料了解OLSR的基本内容后，从协议的核心机制MPR开始，结合协议的流程由主干到枝节、循序渐进地完成代码分析。通过这次难得的经历，我们明白了只有合理地运用研究方法，才能达到事半功倍的效果。我们最终收获了丰富的知识和宝贵的经验。

最后感谢覃老师的课程讲解和学长的帮助，让我们能更多地了解各个协议的基本内容、掌握许多学习工具的使用。我们也会在课后积极学习，知晓更多的网络协议栈的知识。