aodv-uu源码阅读与整理

routing_table.{h,c}

简介

路由表文件，定义了路由表项结构体和路由表结构体，同时定义了相关的路由操作。

路由表项结构体

```
struct rt_table {
    list_t l;                       //链表域
    struct in_addr dest_addr;       //目标地址
    u_int32_t dest_seqno;           //序列号
    unsigned int ifindex;           //网络接口序号
    struct in_addr next_hop;        //下一跳的地址
    u_int8_t hcnt;                  //到达目的地址所需要的跳数
    u_int16_t flags;                //路由标识 前向、反向、邻居、单向、路径修复
    u_int8_t state;                 //VALID or INVALID
    struct timer rt_timer;          //当前路由表项的计时器
    struct timer ack_timer;         //RREP_ack的计时器
    struct timer hello_timer;       //hello消息的计时器
    struct timeval last_hello_time;
    u_int8_t hello_cnt;
    hash_value hash;
    int nprec;                      //先驱节点的数目
    list_t precursors;              //先驱节点链表头
};
```

路由表结构体

```
struct routing_table {              //路由表
    unsigned int num_entries;       //表项数目
    unsigned int num_active;        //活跃节点数目
    list_t tbl[RT_TABLESIZE];       //路由表项数组
};
```

功能函数

```
rt_table_init               //初始化路由表，表项数目、活跃节点数目置零，路由表项初始化
rt_table_destroy            //遍历destroy路由表项
hashing                     //获得哈希值（tbl哈希表）
rt_table_insert             /*插入路由表项
                              1、计算哈希值
                              2、检查是否有目的节点的路由表项
                              3、根据参数创建一个路由表项并初始化
                              4、其他设置（更新路由表元素、设置timeout计时器……）
                            */
rt_table_update             //更新路由表项
rt_table_update_timeout             //更新指定路由表项的timeout
rt_table_update_route_timeouts      //更新timeout以响应传入或传出的数据包
rt_table_find               //根据目标地址寻找路由表项（根据哈希函数）
rt_table_invalidate         /*使指定路由表项无效（过期或删除）
```

```
                                1、判断是否已经无效
                                2、清空相关timer
                                3、将route设置invalid
                                4、增加序列号
                                5、修复 or 删除
                              */
rt_table_delete               //删除路由表项
precursor_add                 //增加先驱节点
precursor_remove              //删除先驱节点
precursor_list_destroy        //删除所有先驱节点
```

aodv_timeout.{h,c}

简介

定义各种timeout处理函数，属于工具文件。

各种timeout处理函数

```
route_discovery_timeout       //寻路对应的处理函数
local_repair_timeout          //本地修复处理函数
route_expire_timeout          //路由失效处理函数
route_delete_timeout          //路由删除处理函数
hello_timeout                 //hello
rrep_ack_timeout              //回复确认
wait_on_reboot_timeout        //等待重启
```

aodv_neighbor.{h,c}

简介

描述邻居节点相关操作的文件，主要配合routing_table使用，只有两个函数。

两个函数

```
neighbor_add              //add or update非hello消息的邻居
neighbor_link_break       /*邻居链接中断
                            1、如果hcnt!=1，报错（通过hcnt判断是否为邻居）
                            2、rt_table_invalidate
                            3、如果不是修复则报错
                            4、遍历寻找下一跳地址等于该邻居地址的路由表项
                            5、如果当前邻居路由表项为修复，则也把遍历路由表项标为修复
                            6、rt_table_invalidate(rt_u)
                            7、如果该遍历路由表项有前驱节点，错误信息再处理
                            8、发送错误信息
                          */
```

aodv_hello.{h,c}

简介

一个特殊的rrep消息，用于维护邻居信息，定时广播，定时更新路由表。

功能函数

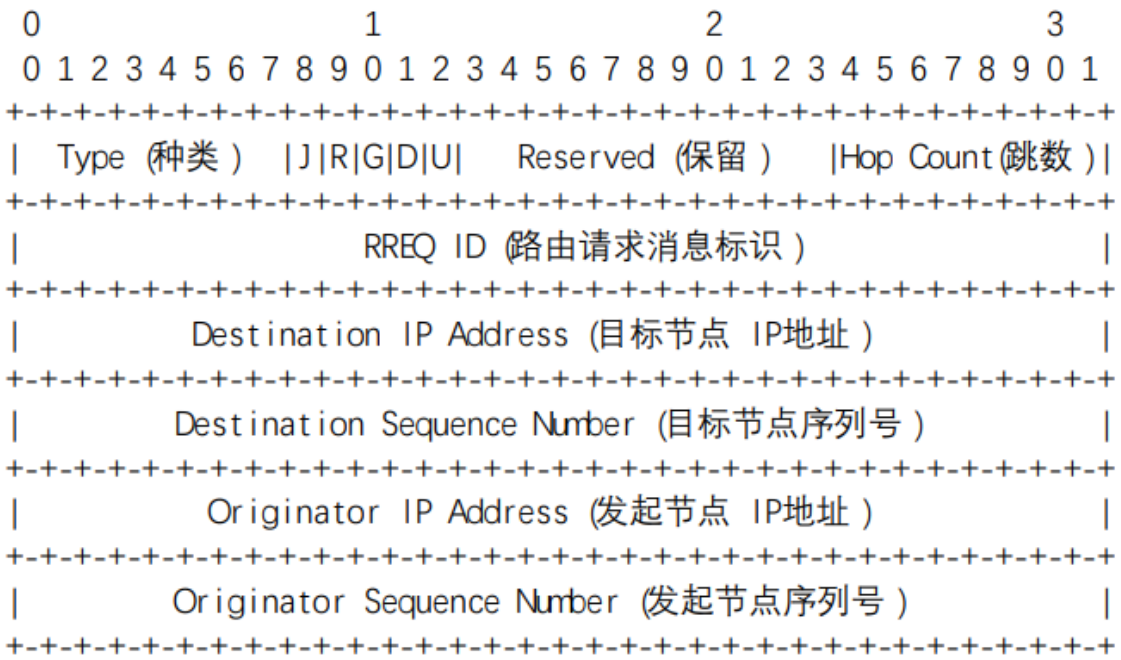
```
hello_start             //启动hello
hello_stop              //停止hello
hello_send              /*发送hello消息
                          1、判断两次间隔是否大于HELLO_INTERVAL，若是则继续
                          2、遍历所有接口并生成目的节点地址和序列号都是自己的RREP
                          3、设置lifetime并广播
                          4、重新设置timeout
                        */
hello_process           //处理hello消息并对路由表进行更新
```

aodv_rreq.{h,c}

简介

rreq：路由的request消息
rreq_record：对收到的rreq消息进行缓存
blacklist：单向路径情况下的黑名单

消息格式



rreq_record结构体

```
struct rreq_record {
    list_t l;
    struct in_addr orig_addr;           //源节点地址
    u_int32_t rreq_id;                  //rreq广播id
    struct timer rec_timer;
};
```

blacklist结构体

```
struct blacklist {
    list_t l;
    struct in_addr dest_addr;
    struct timer bl_timer;
};
```

功能函数

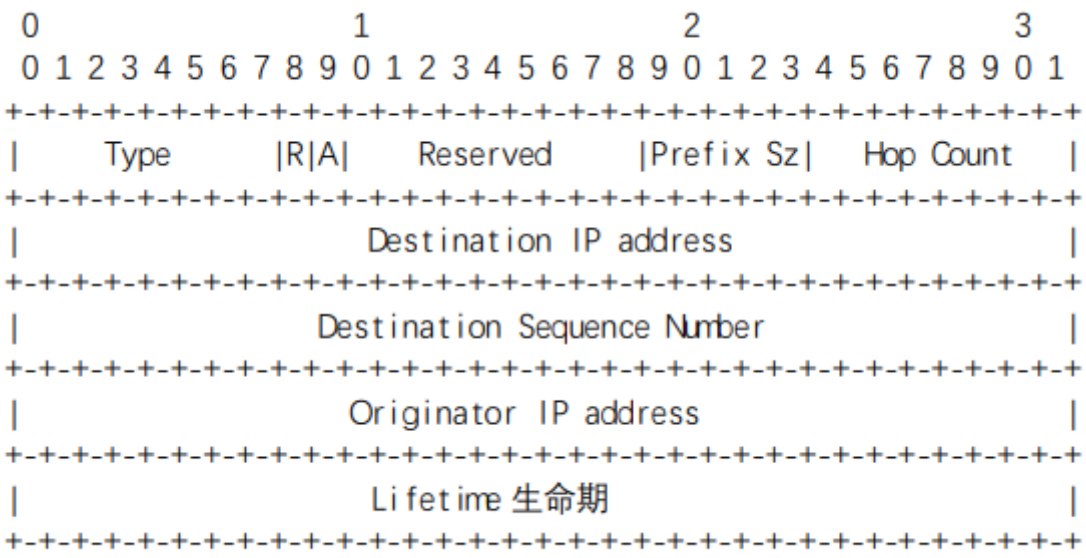
```
rreq_create             /*生成rreq消息
                          1、初始化
                          2、序列号自增
                        */
rreq_send               /*发送rreq消息
                          1、设置免费flag
                          2、遍历生成rreq消息并广播
                        */
rreq_forward            /*向前传递
                          1、合法性检查
                          2、从缓冲中取出rreq并将hcnt自增
                          3、遍历广播
                        */
rreq_process            /*处理rreq消息
                          1、是否由自己发出
                          2、上一跳是否在黑名单中
                          3、是否收到过该rreq
                          4、若前三项都没问题则插入rreq_record
                          5、建立反向路由：路由表add or update
                          6、检查自己是否为目标节点，若是则更新序列号并生成rrep
                          7、否则，检查路由表里面是否有目的节点的路由表项
                          8、如果有并且相应的序列号大于等于RREQ里面的序列号则生成RREP然后
直接返回
                          9、否则调用rreq_forward同时ttl自减
                        */
rreq_route_discovery    /*rreq寻路
                          1、是否已经存在与seek_list
                          2、路由表是否已经存在目的节点信息
                          3、调用rreq_send，并且把当前寻路信息加入seek_list
                          4、设置timeout
                        */
rreq_local_repair       /*rreq本地修复，与上面相似*/
```

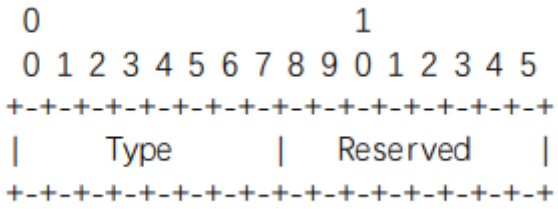
aodv_rrep.{h,c}

简介

rrep：路由的reply消息
rrep_ack：若rrep的a位为1，则需要发送一个rrep_ack消息用于回复

消息格式





RREP_ack结构体

```
typedef struct {
    u_int8_t type;
    u_int8_t reserved;
} RREP_ack;
```

功能函数

```
rrep_create             //创建rrep消息
rrep_ack_create         //创建rrep_ack消息
rrep_ack_process        //处理ack_rrep，移除ack_timer
rrep_send               /*发送rrep消息
                          1、是否需要返回rrep_ack
                          2、设置ack_timer
                          3、发送
                          4、添加precursor信息
                        */
rrep_forward            /*向前传递
                          1、是否需要返回rrep_ack
                          2、作为中间节点向前传递rrep
                          3、更新timeout
```

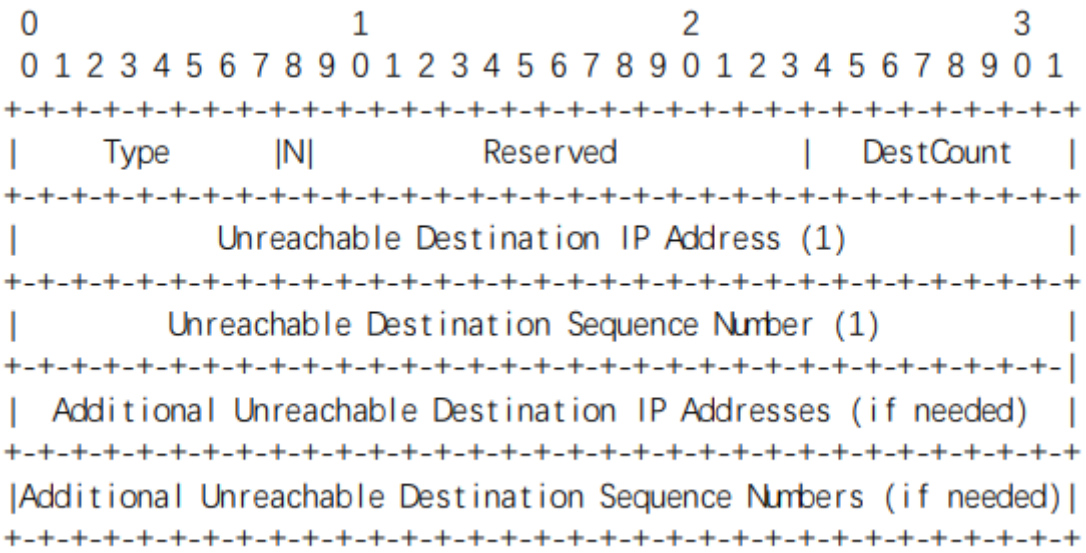
```
                        */
rrep_process            /*处理rrep消息
                          1、检查正确性、完备性
                          2、找源节点和目的节点的路由表项rev_rt和fwd_rt，若rev_rt不存在
则插入
                          3、检测是否有a标志，有则发送RREP_ack并移除a标志
                          4、检查RREP的源节点是否是自己
                          5、若是，判断是否为修复且是否修复成功
                          6、若否，调用rrep_forward同时ttl自减
                        */
```

aodv_rerr.{h,c}

简介

rerr：路由的error消息
rerr_udest：不可到达的目标节点

消息格式：



RERR_udest结构体

```
typedef struct {
    u_int32_t dest_addr;
    u_int32_t dest_seqno;
} RERR_udest;
```

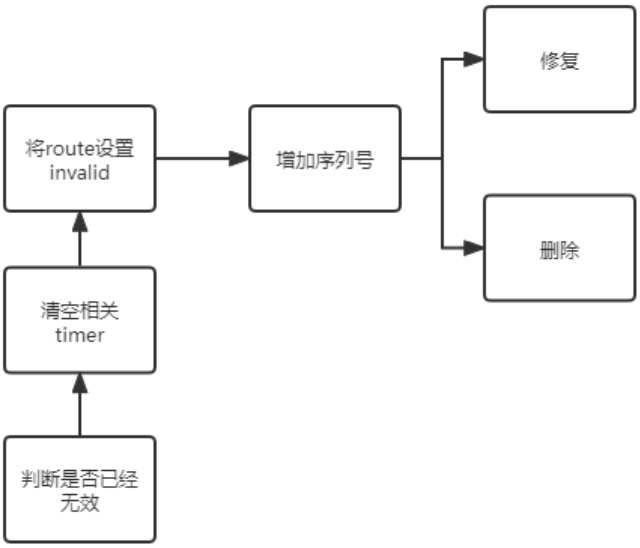
功能函数

```
rerr_create              //创建rerr消息
rerr_add_udest           //添加udest
rerr_process             /*处理rerr消息
                           1、检查正确性、完备性
                           2、遍历udest，根据目的节点地址找到路由表项
                           3、如果路由表项为VALID且下一跳为ip_src
                           4、检查序列号
                           5、invalidate路由表项
                           6、更新路由表项的dest_seqno为该udest的seqno
                           7、若没有标识路径修复且有前驱节点，则把当前udest节点信息加入新的
rerr消息
                           8、若有新的rerr生成，则发送
                         */
```

附件

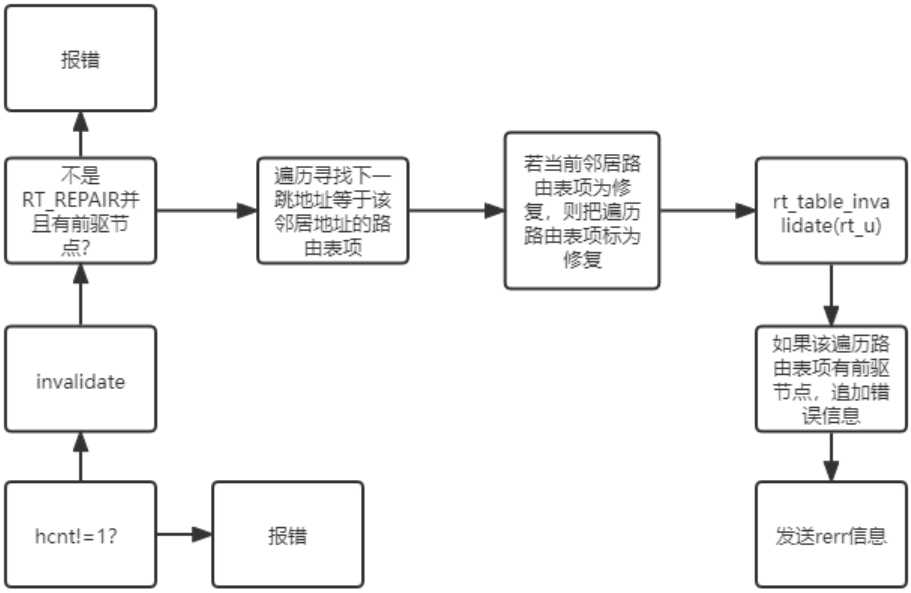
rt_table_invalidate

路由表项无效(过期或移除)处理函数



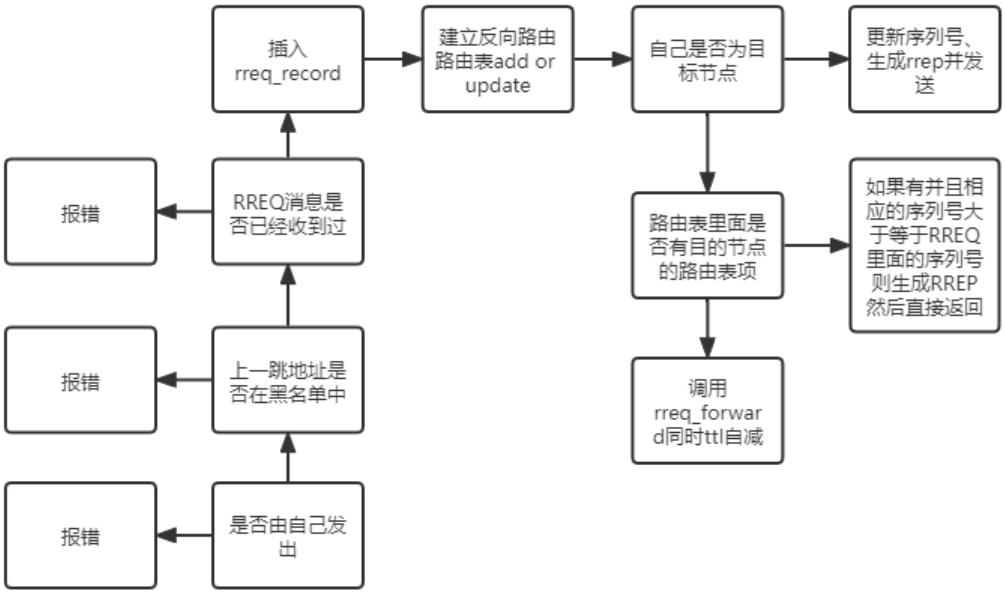
neighbor_link_break

邻居节点链接中断处理函数



rreq_process

处理RREQ消息的函数



参考资料

1. aodv-uu 路由协议手册
2. aodv-uu 源码解读博客 . aodv-uu 源码解读_dragonylee的博客-CSDN博客

——《无线网络系统》第二组 2021.10.27