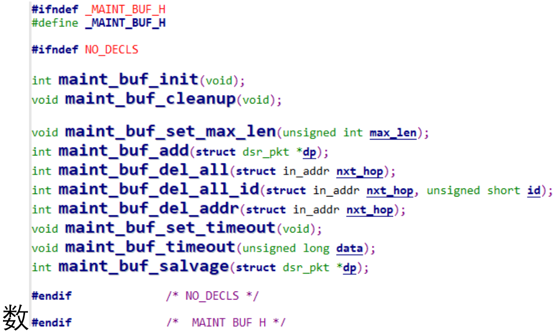
Timer.h　　定义 DSRUUTime 计时器 getset

List.h　　　链表

Tbl.h　　　链表 返回链表头、判断空、满，判断是否为头 DSR 读写锁　支持前后添加，查找，删除

```
struct tbl {
	list_t head;
	volatile unsigned int len;
	volatile unsigned int max_len;
#ifdef __KERNEL__
	rwlock_t lock;
#endif
};
```

Maint-buf.h　定义函数



```
#ifndef _MAINT_BUF_H
#define _MAINT_BUF_H

#ifndef NO_DECLS

int maint_buf_init(void);
void maint_buf_cleanup(void);

void maint_buf_set_max_len(unsigned int max_len);
int maint_buf_add(struct dsr_pkt *dp);
int maint_buf_del_all(struct in_addr nxt_hop);
int maint_buf_del_all_id(struct in_addr nxt_hop, unsigned short id);
int maint_buf_del_addr(struct in_addr nxt_hop);
void maint_buf_set_timeout(void);
void maint_buf_timeout(unsigned long data);
int maint_buf_salvage(struct dsr_pkt *dp);

#endif			/* NO_DECLS */

#endif			/*  MAINT BUF H */
```

Link-cache.h　link-cache.c 里的函数

```
struct lc_graph {
	struct tbl nodes;
	struct tbl links;
	struct lc_node *src;
#ifdef __KERNEL__
	struct timer_list timer;
	rwlock_t lock;
#endif
};
```

Dsr-pkt.h 定义 DSR 包报文格式

```
struct dsr_pkt {
	struct in_addr src;  /* IP level data */     源 ip        struct in_addr {
	struct in_addr dst;                          目的 ip          in_addr_t s_addr;
	struct in_addr nxt_hop;              下一跳 ip              };
	struct in_addr prv_hop;        上一跳 ip      表示一个 32 位的 IPv4 地址。in_addr_t 一般
	int flags;                           为 32 位的 unsigned int，其字节顺序为网络字节序，
	int salvage;    即该无符号数采用大端字节序。每 8 位表示一个 IP 地址中的一个数值。
#ifdef NS2
	union {
		struct hdr_mac *ethh;
		unsigned char *raw;
	} mac;
```

```
        struct hdr_ip ip_data;
        union {
                struct hdr_ip *iph;
                char *raw;
        } nh;
#else
        union {
                struct ethhdr *ethh;
                char *raw;
        } mac;
        union {
                struct iphdr *iph;
                char *raw;
        } nh;
        char ip_data[60];
#endif
        struct {
                union {
                        struct dsr_opt_hdr *opth;
                        char *raw;
                };
                char *tail, *end;
        } dh;

        int num_rrep_opts, num_rerr_opts, num_rreq_opts, num_ack_opts;
        struct dsr_srt_opt *srt_opt;
        struct dsr_rreq_opt *rreq_opt;      /* Can only be one */
        struct dsr_rrep_opt *rrep_opt[MAX_RREP_OPTS];
        struct dsr_rerr_opt *rerr_opt[MAX_RERR_OPTS];
        struct dsr_ack_opt *ack_opt[MAX_ACK_OPTS];
        struct dsr_ack_req_opt *ack_req_opt;
        struct dsr_srt *srt;  /* Source route */


        int payload_len;
#ifdef NS2
        AppData *payload;
        Packet *p;
#else
        char *payload;
        struct sk_buff *skb;
#endif

}
```

```
#Include<ip.h>
struct iphdr {
  #if defined(__LITTLE_ENDIAN_BITFIELD)       小端模式下
      __u8      ihl:4,          首部长度(4 位)
            version:4;          ip 协议版本 IPv4
  #elif defined (__BIG_ENDIAN_BITFIELD)       大端模式下
      __u8      version:4,
            ihl:4;
  #else
  #error    "Please fix <asm/byteorder.h>"
  #endif
     __u8      tos;                 8 位服务类型字段
     __be16    tot_len;             16 位 IP 数据报总长度
     __be16    id;                  16 位标识字段（唯一表示主机发送的每一分数据报）
     __be16    frag_off;            (3 位分段标志+13 位分段偏移数)
     __u8      ttl;                 8 位数据报生存时间
     __u8      protocol;            协议字段（8 位）
     __sum16 check;                 16 位首部校验
     __be32    saddr;               源 IP 地址
     __be32    daddr;               目的 IP 地址
     /*The options start here. */
 };
```

#include<if_packet.h>   原始数据包的数据结构定义
Sockaddr_pkt
Sockaddr_ ll
定义 主机、广播、组播、其他主机
定义帧结构
用于接收原始数据包

```
#ifndef __LINUX_IF_PACKET_H
#define __LINUX_IF_PACKET_H

struct sockaddr_pkt
{
unsigned short spkt_family;
unsigned char spkt_device[14];
unsigned short spkt_protocol;
};

struct sockaddr_ll
```

```
{
unsigned short sll_family;
unsigned short sll_protocol;
int sll_ifindex;
unsigned short sll_hatype;
unsigned char sll_pkttype;
unsigned char sll_halen;
unsigned char sll_addr[8];
};

/* Packet types */

#define PACKET_HOST 0 /* To us */
#define PACKET_BROADCAST 1 /* To all */
#define PACKET_MULTICAST 2 /* To group */
#define PACKET_OTHERHOST 3 /* To someone else   */
#define PACKET_OUTGOING 4 /* Outgoing of any type */
/* These ones are invisible by user level */
#define PACKET_LOOPBACK 5 /* MC/BRD frame looped back */
#define PACKET_FASTROUTE 6 /* Fastrouted frame */

/* Packet socket options */

#define PACKET_ADD_MEMBERSHIP 1
#define PACKET_DROP_MEMBERSHIP 2
#define PACKET_RECV_OUTPUT 3
/* Value 4 is still used by obsolete turbo-packet. */
#define PACKET_RX_RING 5
#define PACKET_STATISTICS 6
#define PACKET_COPY_THRESH 7

struct tpacket_stats
```

```
{
unsigned int tp_packets;
unsigned int tp_drops;
};

struct tpacket_hdr
{
unsigned long tp_status;
#define TP_STATUS_KERNEL 0
#define TP_STATUS_USER 1
#define TP_STATUS_COPY 2
#define TP_STATUS_LOSING 4
#define TP_STATUS_CSUMNOTREADY 8
unsigned int tp_len;
unsigned int tp_snaplen;
unsigned short tp_mac;
unsigned short tp_net;
unsigned int tp_sec;
unsigned int tp_usec;
};

#define TPACKET_ALIGNMENT 16
#define TPACKET_ALIGN(x) (((x)+TPACKET_ALIGNMENT-1)&~(TPACKET_ALIGNMENT-1))
#define TPACKET_HDRLEN (TPACKET_ALIGN(sizeof(struct tpacket_hdr)) + sizeof(struct sockaddr_ll))

/*
  Frame structure:

  - Start. Frame must be aligned to TPACKET_ALIGNMENT=16
  - struct tpacket_hdr
  - pad to TPACKET_ALIGNMENT=16
```

```
    - struct sockaddr_ll
    - Gap, chosen so that packet data (Start+tp_net) alignes to TPACKET_ALIGNMENT=16
    - Start+tp_mac: [ Optional MAC header ]
    - Start+tp_net: Packet data, aligned to TPACKET_ALIGNMENT=16.
    - Pad to align to TPACKET_ALIGNMENT=16
  */

struct tpacket_req
{
unsigned int tp_block_size; /* Minimal size of contiguous block */
unsigned int tp_block_nr; /* Number of blocks */
unsigned int tp_frame_size; /* Size of frame */
unsigned int tp_frame_nr; /* Total number of frames */
};

struct packet_mreq
{
int mr_ifindex;
unsigned short mr_type;
unsigned short mr_alen;
unsigned char mr_address[8];
};

#define PACKET_MR_MULTICAST 0
#define PACKET_MR_PROMISC 1
#define PACKET_MR_ALLMULTI 2

#endif
```

```
#include<if_ether.h>
struct ethhdr
{
    unsigned char h_dest[ETH_ALEN]; //目的 MAC 地址

    unsigned char h_source[ETH_ALEN]; //源 MAC 地址

    __u16 h_proto ; //网络层所使用的协议类型 16 位
}__attribute__((packed))  //用于告诉编译器不要对这个结构体中的缝隙部分进行填充操作;
```

Dsr.opt.h   struct dsr-opt-hdr

Pkt len

Dst_

```
struct dsr_opt_hdr {
        u_int8_t nh;
        u_int16_t p_len;     /* payload length */                    有效负载
#ifdef NS2
        static int offset_;

        inline static int &offset() {                                偏移
                return offset_;
        }
        inline static dsr_opt_hdr *access(const Packet * p) {        返回 packet 的进入点?
                return (dsr_opt_hdr *) p->access(offset_);
        }

        int size() {                                                 返回字节大小  (长度+结构体大小)
                return ntohs(p_len) + sizeof(struct dsr_opt_hdr);
        }
#endif                      /* NS2 */
        struct dsr_opt option[0];
};


struct dsr_pad1_opt {
        u_int8_t type;
};

#ifdef NS2
#define DSR_NO_NEXT_HDR_TYPE PT_NTYPE
#else
#define DSR_NO_NEXT_HDR_TYPE 0
#endif
```

/ Header lengths */*

#define DSR_FIXED_HDR_LEN 4 / Should be the same as DSR_OPT_HDR_LEN, but that*
** is not the case in ns-2 */ 定义首部长度，应该与 DSR_OPT_HDR 长度相同*

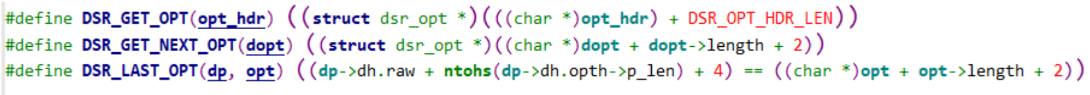
#define DSR_OPT_HDR_LEN sizeof(struct dsr_opt_hdr)

#define DSR_OPT_PAD1_LEN 1

#define DSR_PKT_MIN_LEN 24 / IP header + DSR header = 20 + 4 */ DSR 最小长度，应为 IP 报文长度 20 字节+DSR 首部长度 4 字节*

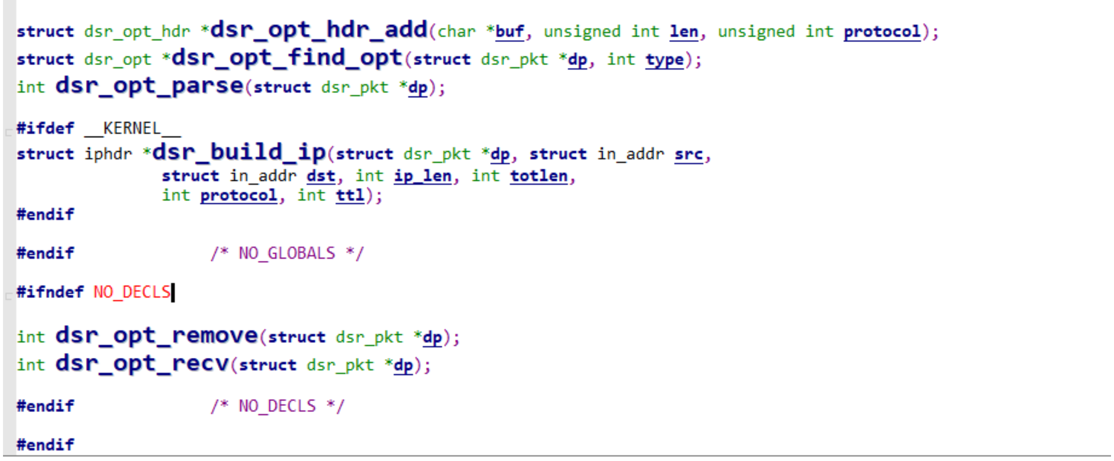
定义 DSR 首部类型

```
/* Header types */
#define DSR_OPT_PADN        0
#define DSR_OPT_RREP        1
#define DSR_OPT_RREQ        2
#define DSR_OPT_RERR        3
#define DSR_OPT_PREV_HOP    5
#define DSR_OPT_ACK         32
#define DSR_OPT_SRT         96
#define DSR_OPT_TIMEOUT     128
#define DSR_OPT_FLOWID      129
#define DSR_OPT_ACK_REQ     160
#define DSR_OPT_PAD1        224
```



```
#define DSR_GET_OPT(opt_hdr) ((struct dsr_opt *)(((char *)opt_hdr) + DSR_OPT_HDR_LEN))
#define DSR_GET_NEXT_OPT(dopt) ((struct dsr_opt *)((char *)dopt + dopt->length + 2))
#define DSR_LAST_OPT(dp, opt) ((dp->dh.raw + ntohs(dp->dh.opth->p_len) + 4) == ((char *)opt + opt->length + 2))
```

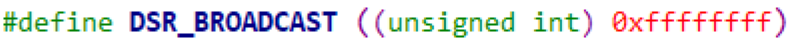
获取报文



```
struct dsr_opt_hdr *dsr_opt_hdr_add(char *buf, unsigned int len, unsigned int protocol);
struct dsr_opt *dsr_opt_find_opt(struct dsr_pkt *dp, int type);
int dsr_opt_parse(struct dsr_pkt *dp);

#ifdef __KERNEL__
struct iphdr *dsr_build_ip(struct dsr_pkt *dp, struct in_addr src,
             struct in_addr dst, int ip_len, int totlen,
             int protocol, int ttl);
#endif

#endif          /* NO_GLOBALS */

#ifndef NO_DECLS

int dsr_opt_remove(struct dsr_pkt *dp);
int dsr_opt_recv(struct dsr_pkt *dp);

#endif          /* NO_DECLS */

#endif
```

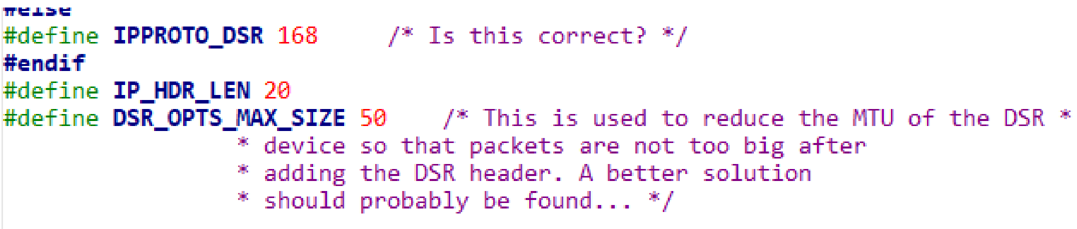
增删改查

Dsr.h



```
#define DSR_BROADCAST ((unsigned int) 0xffffffff)
```

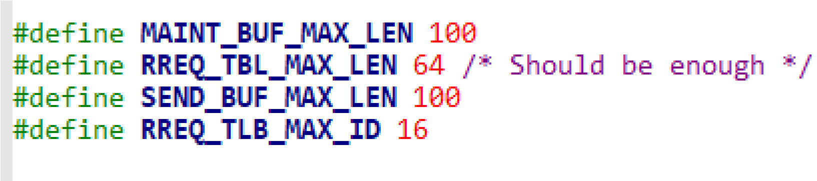
广播，255.255.255.255



```
#else
#define IPPROTO_DSR 168     /* Is this correct? */
#endif
#define IP_HDR_LEN 20
#define DSR_OPTS_MAX_SIZE 50    /* This is used to reduce the MTU of the DSR *
                 * device so that packets are not too big after
                 * adding the DSR header. A better solution
                 * should probably be found... */
```

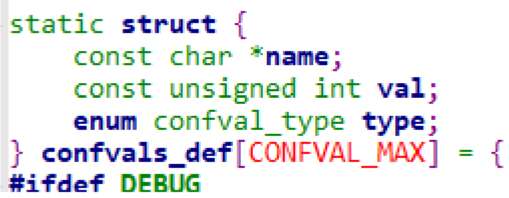
定义 MTU 最大传输长度

枚举 confval 和 confval_type



```
#define MAINT_BUF_MAX_LEN 100
#define RREQ_TBL_MAX_LEN 64 /* Should be enough */
#define SEND_BUF_MAX_LEN 100
#define RREQ_TLB_MAX_ID 16
```

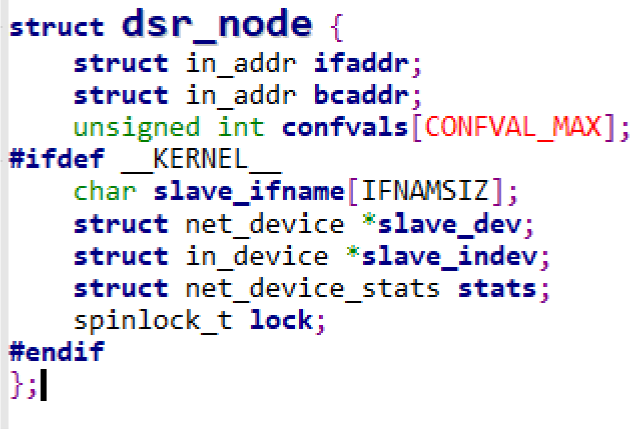
定义最大缓存长度 100，rreq（路由请求 发现更好路径）表 64 ，发送最大缓存长度 100, rreq 表 id 最长 16



```
static struct {
    const char *name;
    const unsigned int val;
    enum confval_type type;
} confvals_def[CONFVAL_MAX] = {
#ifdef DEBUG
```

名字，值，类型

DSR 节点



```
struct dsr_node {
    struct in_addr ifaddr;
    struct in_addr bcaddr;
    unsigned int confvals[CONFVAL_MAX];
#ifdef __KERNEL__
    char slave_ifname[IFNAMSIZ];
    struct net_device *slave_dev;
    struct in_device *slave_indev;
    struct net_device_stats stats;
    spinlock_t lock;
#endif
};
```

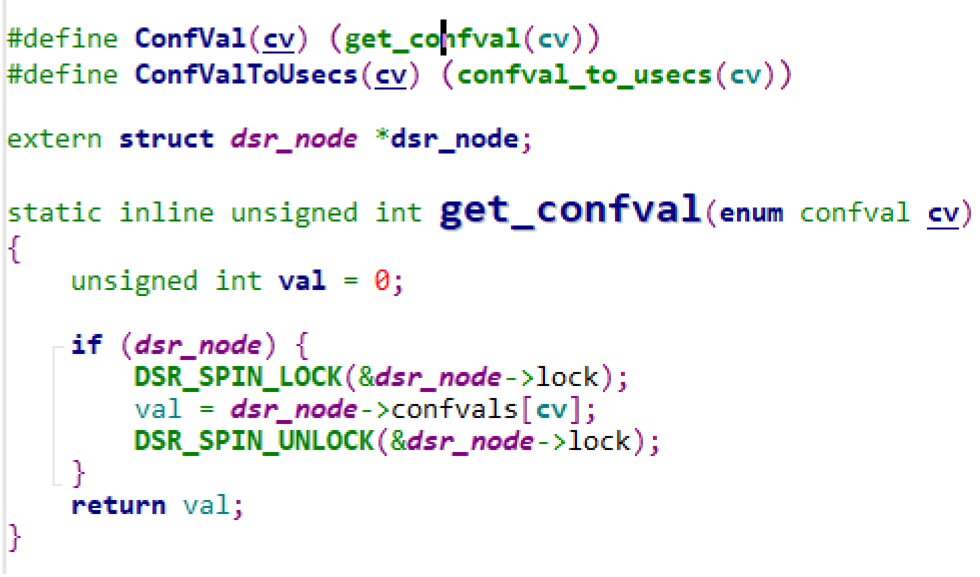
if_addr　主机地址

bc_addr 组播地址

设备状态

锁

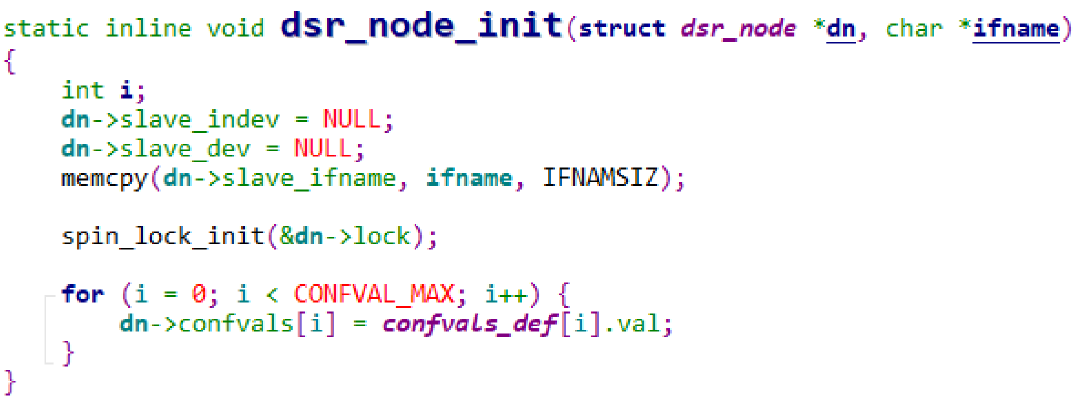
Xmit 发射



```
#define ConfVal(cv) (get_confval(cv))
#define ConfValToUsecs(cv) (confval_to_usecs(cv))

extern struct dsr_node *dsr_node;

static inline unsigned int get_confval(enum confval cv)
{
    unsigned int val = 0;

    if (dsr_node) {
        DSR_SPIN_LOCK(&dsr_node->lock);
        val = dsr_node->confvals[cv];
        DSR_SPIN_UNLOCK(&dsr_node->lock);
    }
    return val;
}
```

Get set

自旋锁保证不被修改

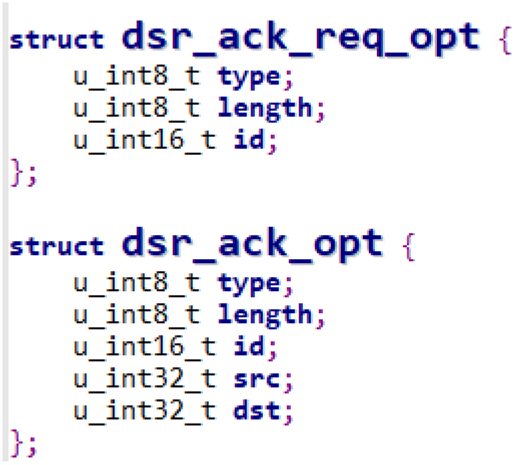


```
static inline void dsr_node_init(struct dsr_node *dn, char *ifname)
{
    int i;
    dn->slave_indev = NULL;
    dn->slave_dev = NULL;
    memcpy(dn->slave_ifname, ifname, IFNAMSIZ);

    spin_lock_init(&dn->lock);

    for (i = 0; i < CONFVAL_MAX; i++) {
        dn->confvals[i] = confvals_def[i].val;
    }
}
```

Init dsr_node

剩下的修改内部属性

Dsr-ack.h



```
struct dsr_ack_req_opt {
    u_int8_t type;
    u_int8_t length;
    u_int16_t id;
};

struct dsr_ack_opt {
    u_int8_t type;
    u_int8_t length;
    u_int16_t id;
    u_int32_t src;
    u_int32_t dst;
};
```

Dsr-dev.h

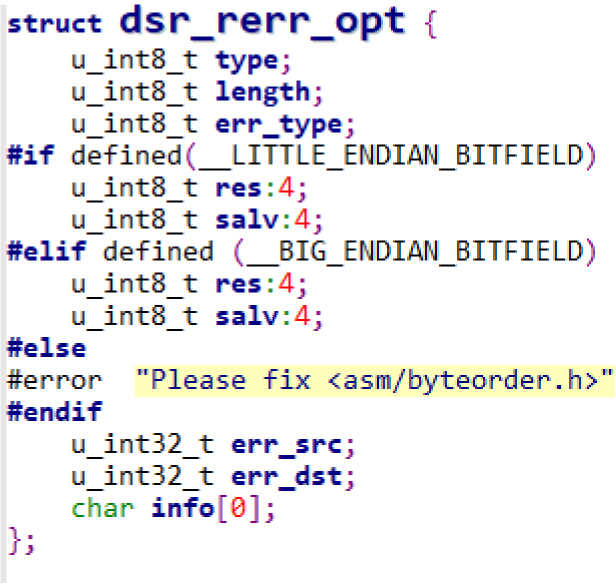
Emit 、deliver 接收发送

Dsr-io.h

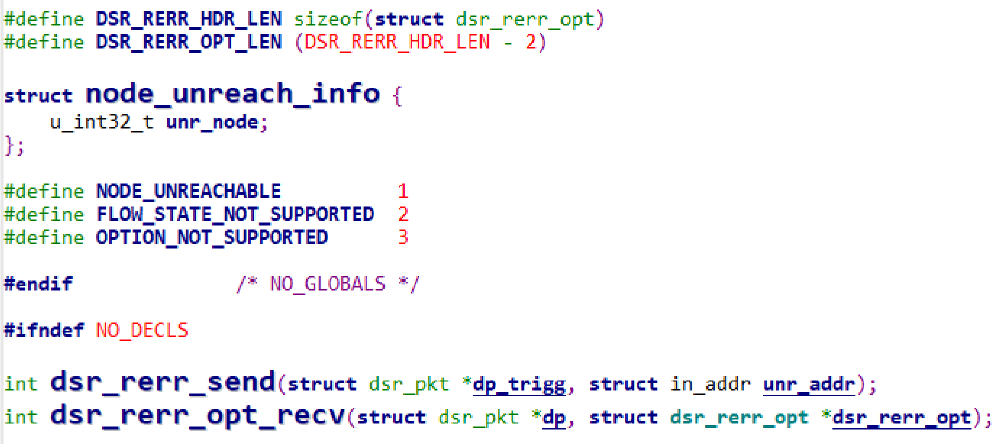
Receive Start_emit

Dsr-rerr.h

Dsr 错误处理



```
struct dsr_rerr_opt {
    u_int8_t type;
    u_int8_t length;
    u_int8_t err_type;
#if defined(__LITTLE_ENDIAN_BITFIELD)
    u_int8_t res:4;
    u_int8_t salv:4;
#elif defined (__BIG_ENDIAN_BITFIELD)
    u_int8_t res:4;
    u_int8_t salv:4;
#else
#error  "Please fix <asm/byteorder.h>"
#endif
    u_int32_t err_src;
    u_int32_t err_dst;
    char info[0];
};
```



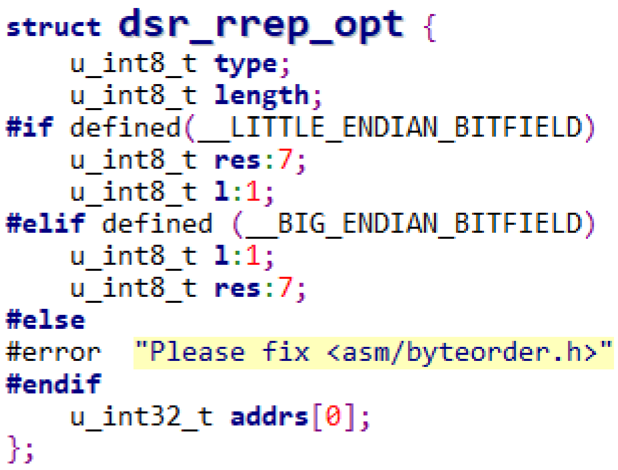
```
#define DSR_RERR_HDR_LEN sizeof(struct dsr_rerr_opt)
#define DSR_RERR_OPT_LEN (DSR_RERR_HDR_LEN - 2)

struct node_unreach_info {
    u_int32_t unr_node;
};

#define NODE_UNREACHABLE            1
#define FLOW_STATE_NOT_SUPPORTED    2
#define OPTION_NOT_SUPPORTED        3

#endif              /* NO_GLOBALS */

#ifndef NO_DECLS

int dsr_rerr_send(struct dsr_pkt *dp_trigg, struct in_addr unr_addr);
int dsr_rerr_opt_recv(struct dsr_pkt *dp, struct dsr_rerr_opt *dsr_rerr_opt);
```

router 错误信息及 router-error 收发

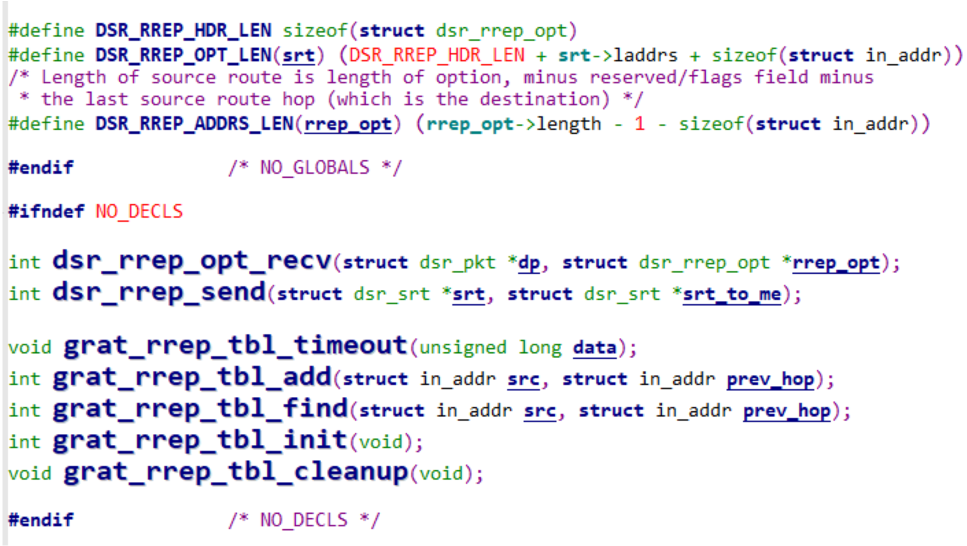
Route-error packet 用于路由维护

Dsr-rrep

Router-reply packet



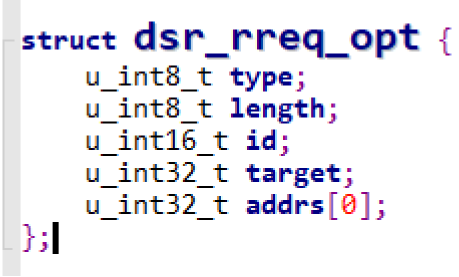
```
struct dsr_rrep_opt {
	u_int8_t type;
	u_int8_t length;
#if defined(__LITTLE_ENDIAN_BITFIELD)
	u_int8_t res:7;
	u_int8_t l:1;
#elif defined (__BIG_ENDIAN_BITFIELD)
	u_int8_t l:1;
	u_int8_t res:7;
#else
#error  "Please fix <asm/byteorder.h>"
#endif
	u_int32_t addrs[0];
};
```



```
#define DSR_RREP_HDR_LEN sizeof(struct dsr_rrep_opt)
#define DSR_RREP_OPT_LEN(srt) (DSR_RREP_HDR_LEN + srt->laddrs + sizeof(struct in_addr))
/* Length of source route is length of option, minus reserved/flags field minus
 * the last source route hop (which is the destination) */
#define DSR_RREP_ADDRS_LEN(rrep_opt) (rrep_opt->length - 1 - sizeof(struct in_addr))

#endif			/* NO_GLOBALS */

#ifndef NO_DECLS

int dsr_rrep_opt_recv(struct dsr_pkt *dp, struct dsr_rrep_opt *rrep_opt);
int dsr_rrep_send(struct dsr_srt *srt, struct dsr_srt *srt_to_me);

void grat_rrep_tbl_timeout(unsigned long data);
int grat_rrep_tbl_add(struct in_addr src, struct in_addr prev_hop);
int grat_rrep_tbl_find(struct in_addr src, struct in_addr prev_hop);
int grat_rrep_tbl_init(void);
void grat_rrep_tbl_cleanup(void);

#endif			/* NO_DECLS */
```

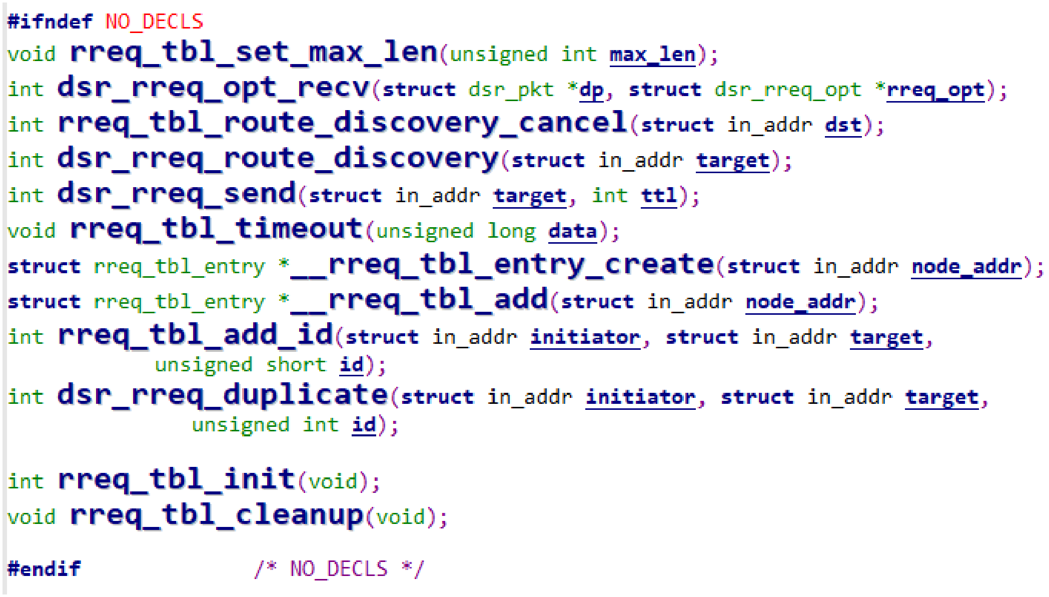
Router-reply 收发 及路由表维护

Dsr-rreq.h



```
struct dsr_rreq_opt {
	u_int8_t type;
	u_int8_t length;
	u_int16_t id;
	u_int32_t target;
	u_int32_t addrs[0];
};
```

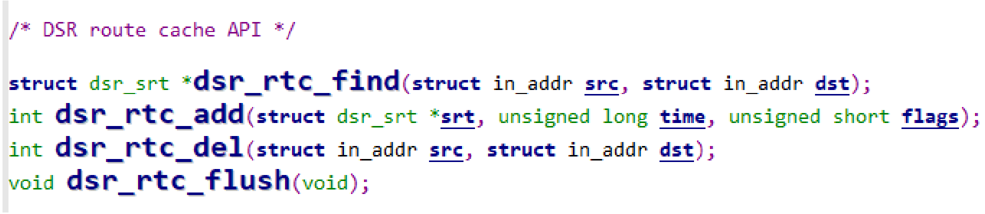
结构



```
#ifndef NO_DECLS
void rreq_tbl_set_max_len(unsigned int max_len);
int dsr_rreq_opt_recv(struct dsr_pkt *dp, struct dsr_rreq_opt *rreq_opt);
int rreq_tbl_route_discovery_cancel(struct in_addr dst);
int dsr_rreq_route_discovery(struct in_addr target);
int dsr_rreq_send(struct in_addr target, int ttl);
void rreq_tbl_timeout(unsigned long data);
struct rreq_tbl_entry *__rreq_tbl_entry_create(struct in_addr node_addr);
struct rreq_tbl_entry *__rreq_tbl_add(struct in_addr node_addr);
int rreq_tbl_add_id(struct in_addr initiator, struct in_addr target,
            unsigned short id);
int dsr_rreq_duplicate(struct in_addr initiator, struct in_addr target,
               unsigned int id);

int rreq_tbl_init(void);
void rreq_tbl_cleanup(void);

#endif              /* NO_DECLS */
```

对路由表进行维护及路由发现

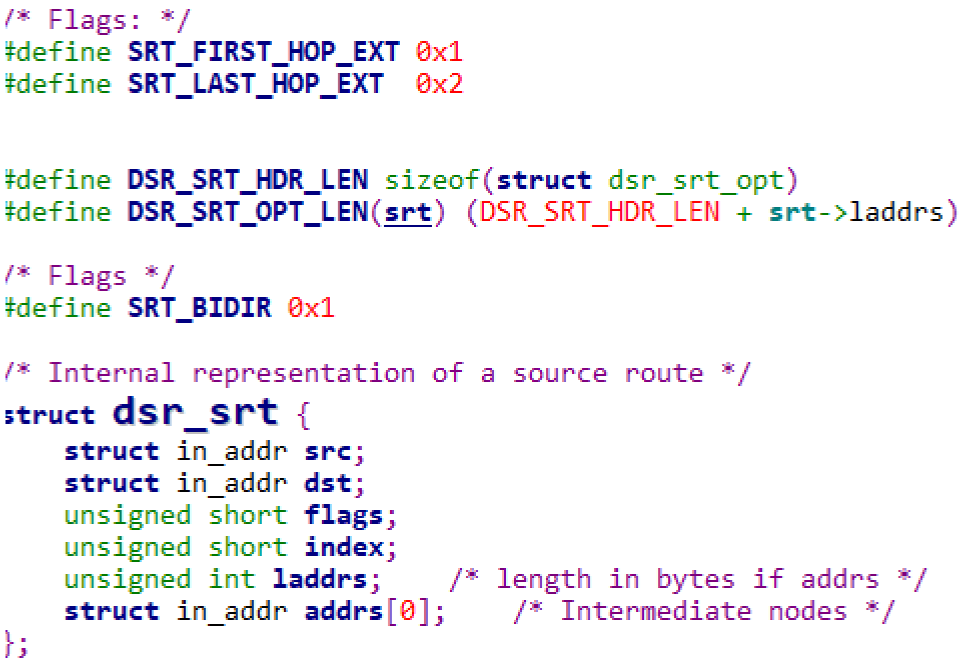
Dsr-rtc.h　路由表 cache



```
/* DSR route cache API */

struct dsr_srt *dsr_rtc_find(struct in_addr src, struct in_addr dst);
int dsr_rtc_add(struct dsr_srt *srt, unsigned long time, unsigned short flags);
int dsr_rtc_del(struct in_addr src, struct in_addr dst);
void dsr_rtc_flush(void);
```

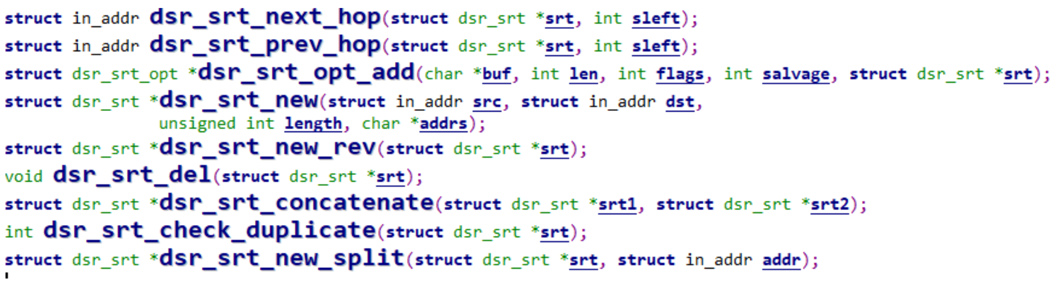
Dsr_srt.h

Source route



```
/* Flags: */
#define SRT_FIRST_HOP_EXT 0x1
#define SRT_LAST_HOP_EXT  0x2


#define DSR_SRT_HDR_LEN sizeof(struct dsr_srt_opt)
#define DSR_SRT_OPT_LEN(srt) (DSR_SRT_HDR_LEN + srt->laddrs)

/* Flags */
#define SRT_BIDIR 0x1

/* Internal representation of a source route */
struct dsr_srt {
    struct in_addr src;
    struct in_addr dst;
    unsigned short flags;
    unsigned short index;
    unsigned int laddrs;    /* length in bytes if addrs */
    struct in_addr addrs[0];    /* Intermediate nodes */
};
```

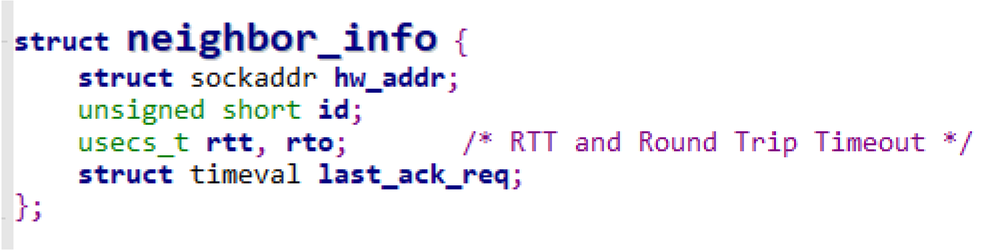
源 IP　目的 IP　跳数　中间节点



```
struct in_addr dsr_srt_next_hop(struct dsr_srt *srt, int sleft);
struct in_addr dsr_srt_prev_hop(struct dsr_srt *srt, int sleft);
struct dsr_srt_opt *dsr_srt_opt_add(char *buf, int len, int flags, int salvage, struct dsr_srt *srt);
struct dsr_srt *dsr_srt_new(struct in_addr src, struct in_addr dst,
                unsigned int length, char *addrs);
struct dsr_srt *dsr_srt_new_rev(struct dsr_srt *srt);
void dsr_srt_del(struct dsr_srt *srt);
struct dsr_srt *dsr_srt_concatenate(struct dsr_srt *srt1, struct dsr_srt *srt2);
int dsr_srt_check_duplicate(struct dsr_srt *srt);
struct dsr_srt *dsr_srt_new_split(struct dsr_srt *srt, struct in_addr addr);
```

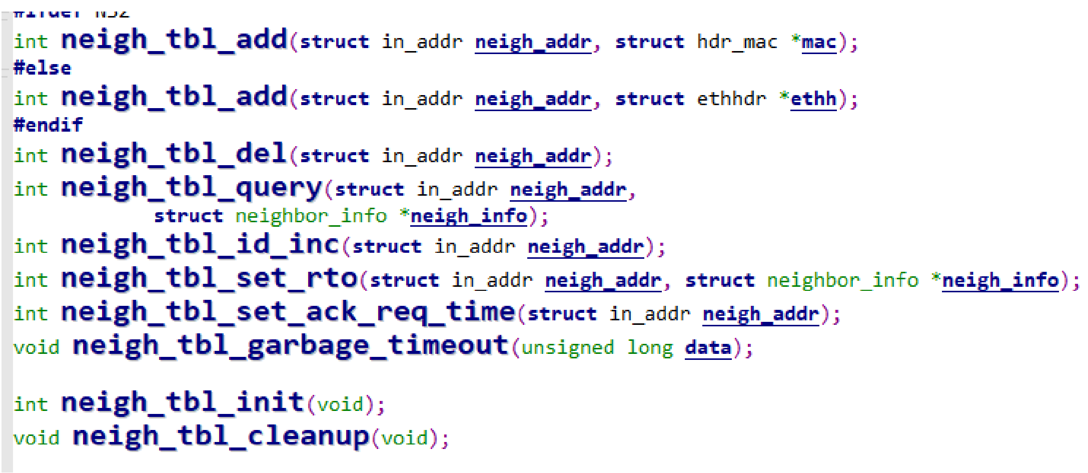
节点间连接 分开 检测重复

Neigh.h 相邻节点



```
struct neighbor_info {
     struct sockaddr hw_addr;
     unsigned short id;
     usecs_t rtt, rto;         /* RTT and Round Trip Timeout */
     struct timeval last_ack_req;
};
```

结构

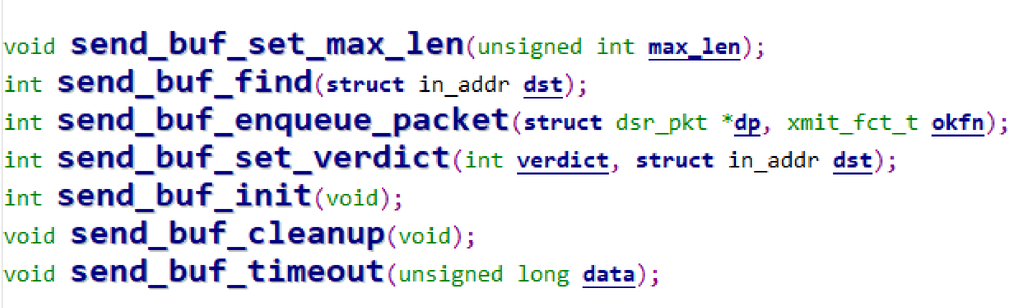


```
#ifdef NS2
int neigh_tbl_add(struct in_addr neigh_addr, struct hdr_mac *mac);
#else
int neigh_tbl_add(struct in_addr neigh_addr, struct ethhdr *ethh);
#endif
int neigh_tbl_del(struct in_addr neigh_addr);
int neigh_tbl_query(struct in_addr neigh_addr,
            struct neighbor_info *neigh_info);
int neigh_tbl_id_inc(struct in_addr neigh_addr);
int neigh_tbl_set_rto(struct in_addr neigh_addr, struct neighbor_info *neigh_info);
int neigh_tbl_set_ack_req_time(struct in_addr neigh_addr);
void neigh_tbl_garbage_timeout(unsigned long data);

int neigh_tbl_init(void);
void neigh_tbl_cleanup(void);
```

信息维护

Ns-agent.h

Send-buf.h



```
void send_buf_set_max_len(unsigned int max_len);
int send_buf_find(struct in_addr dst);
int send_buf_enqueue_packet(struct dsr_pkt *dp, xmit_fct_t okfn);
int send_buf_set_verdict(int verdict, struct in_addr dst);
int send_buf_init(void);
void send_buf_cleanup(void);
void send_buf_timeout(unsigned long data);
```

发送缓冲区