动态源路由协议（DSR）

高子彭　张晟瑀　贺湧钧

目录

DSR路由协议介绍

DSR路由协议介绍

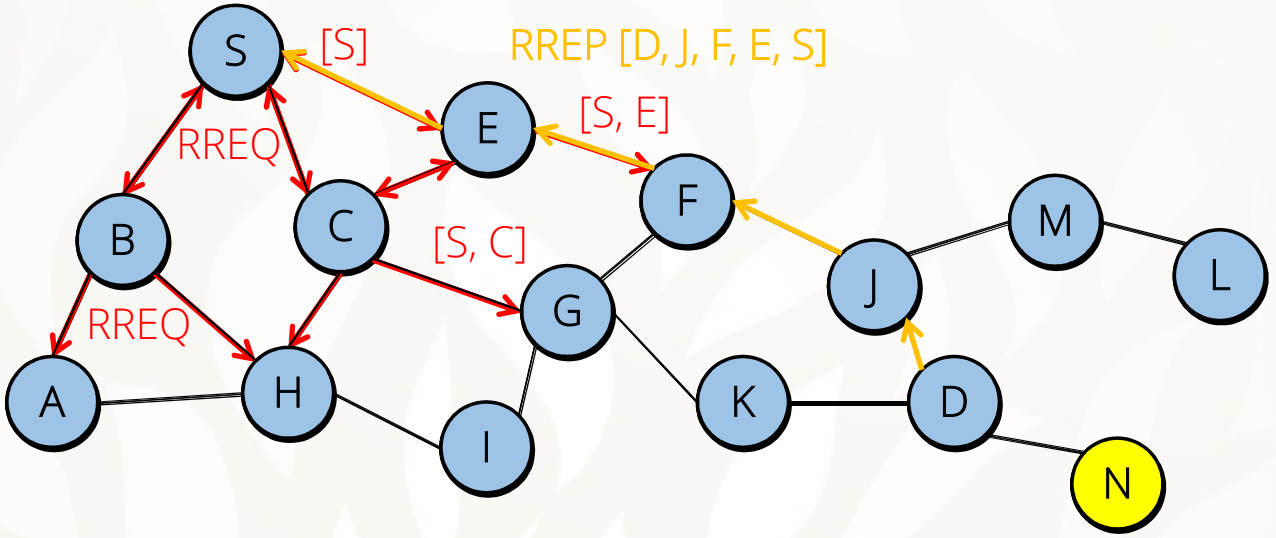
- **动态源路由**（**DSR**）是用于无线网状网络的路由协议。它类似于AODV（无线自组网按需平面距离向量路由协议），因为它在发送节点请求路由时按需形成路由。但是，它使用的是源路由而非依赖于每个中间设备上的路由表。

- DSR协议由两种主要机制组成，它们共同工作，以便发现和维护网络中的源路由:

-**路由发现**是节点S希望将数据包发送到目标节点D时，获得到D的源路由的机制。路由发现仅在S试图将数据包发送到D且到D的路由未知时使用。

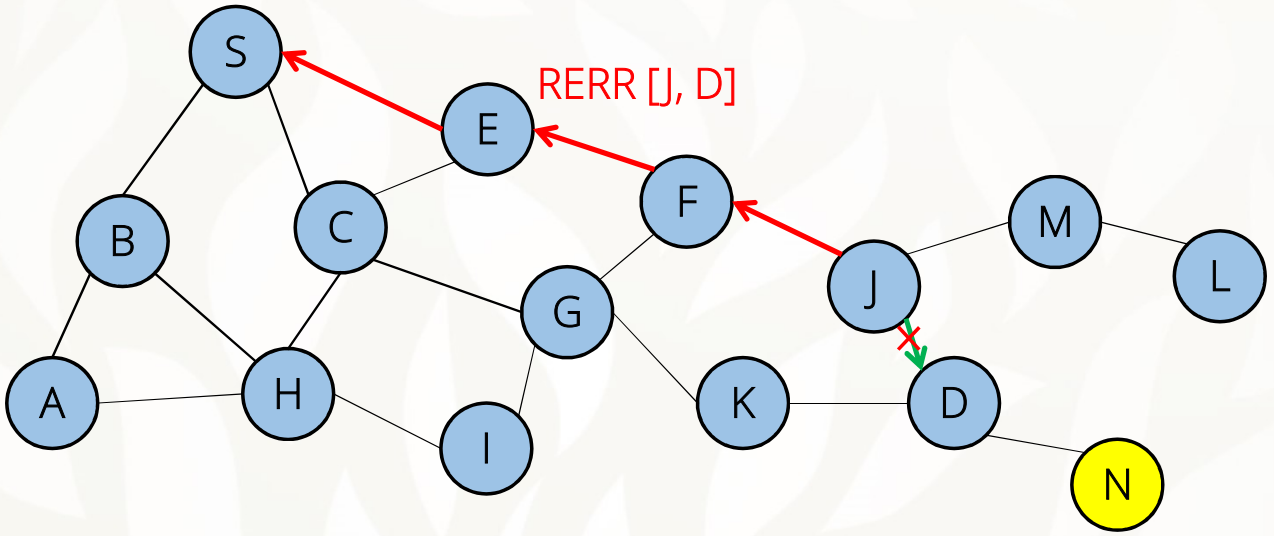
-**路由维护**是指当路由维护表明源路由故障，S可以尝试使用任何其他到D的路由，或可以调用路由发现再次找到一个新的到D的路由。路由维护仅在S已实际发送数据包到D时使用。

具体说明 路由发现



- 当S试图将数据包发送到目的节点D时，先广播一个RREQ包请求，其相邻节点若不是D，将会记录发送缓存并继续广播RREQ。
- 当目的节点收到RREQ包时，不再转发该数据包。随后目的节点会发送RREP包给源节点S，如此实现S到D的路由发现.

具体说明 路由维护



- 当节点J和D之间的连接出现问题时，J会向源节点发送RERR包报告错误。
- 源节点收到RERR包，将J和D之间的连接信息移出路由缓存，重新使用路由发现建立到节点D的路由

数据结构

路由缓存

```c
struct lc_node {
    list_t l;
    struct in_addr addr;
    unsigned int links;
    unsigned int cost;  /* Cost estimate from source when running Dijkstra */
    unsigned int hops;  /* Number of hops from source. Used to get the
                         * length of the source route to allocate. Same as
                         * cost if cost is hops. */
    struct lc_node *pred;   /* predecessor */
};

struct lc_link {
    list_t l;
    struct lc_node *src, *dst;
    int status;
    unsigned int cost;
    struct timeval expires;
};

struct link_query {
    struct in_addr src, dst;
};

struct cheapest_node {
    struct lc_node *n;
};
```

- 当节点获悉了一个新的链接时将该路由信息添加到它的路由缓存

  首先需要检查routetable内是否已经包含该条路由（包括在cache中的路由信息）

- 当节点获悉到自组织网络中的现有链接已断开时，节点从其路由缓存中删除该条路由信息

  通过接收Route Error Packet来判断链接是否断开

发送缓存

```
struct send_buf_entry { // send_buff_entry
    list_t l;
    struct dsr_pkt *dp;    //动态源路由数据包
    struct timeval qtime;
    xmit_fct_t okfn;
};
```

```
struct dsr_pkt {
    struct in_addr src; /* IP level data */ //源节点
    struct in_addr dst;  //目的节点
    struct in_addr nxt_hop;  //下个节点
    struct in_addr prv_hop;  //上一个节点
    int flags;
    int salvage;
#ifdef NS2 ...
#endif
    struct {
        union { ...
        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 ...
#endif

} « end dsr_pkt » ;
```

- 存储的是尚未发送的数据包
- 发送缓冲区的数据包与放入的顺序有关
- 在Sendbuf Timeout后会被无条件丢弃
- 为防止缓冲区溢出，可在需要时采用FIFO的方法删除数据包

路由请求表

记录有关该节点最近发起或转发的路由请求的信息。

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

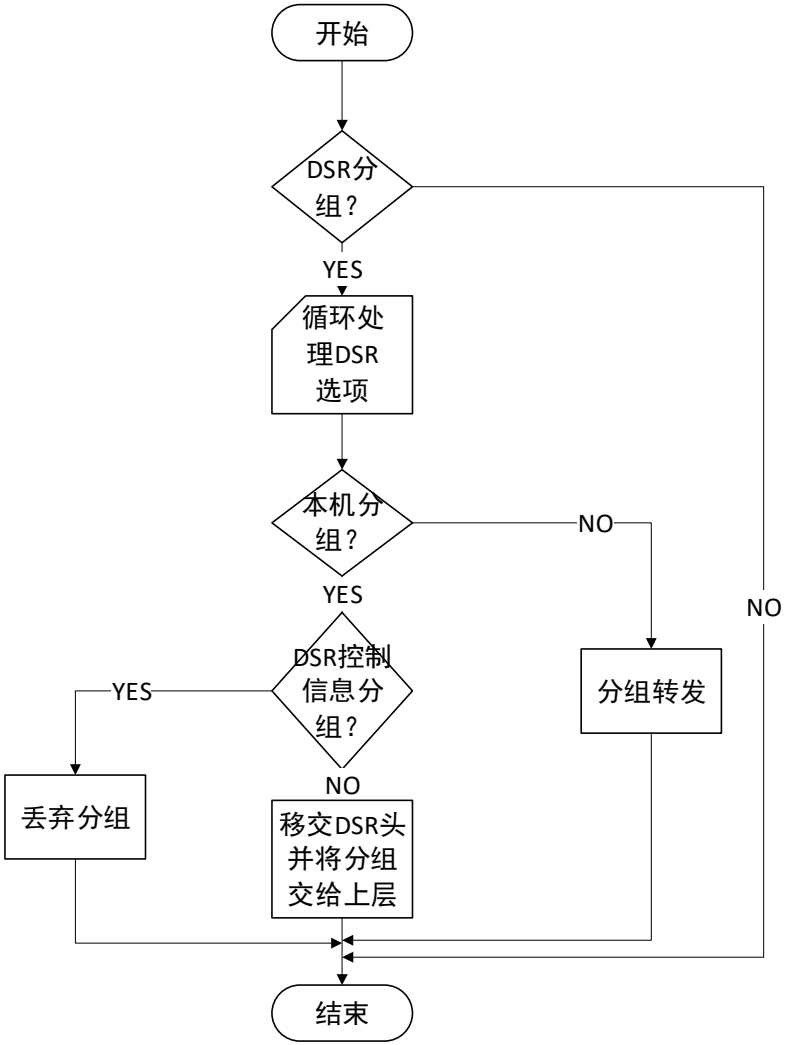
```
struct rreq_tbl_entry {
    list_t l;
    int state;
    struct in_addr node_addr;
    int ttl;//Time to Live
    DSRUUTimer *timer;
    struct timeval tx_time;
    struct timeval last_used;//直到最后使用的时间
    usecs_t timeout;
    unsigned int num_rexmts;//寻找的次数
    struct tbl rreq_id_tbl;
};

struct id_entry {
    list_t l;
    struct in_addr trg_addr;
    unsigned short id;
};
struct rreq_tbl_query {
    struct in_addr *initiator;
    struct in_addr *target;
    unsigned int *id;
};
```

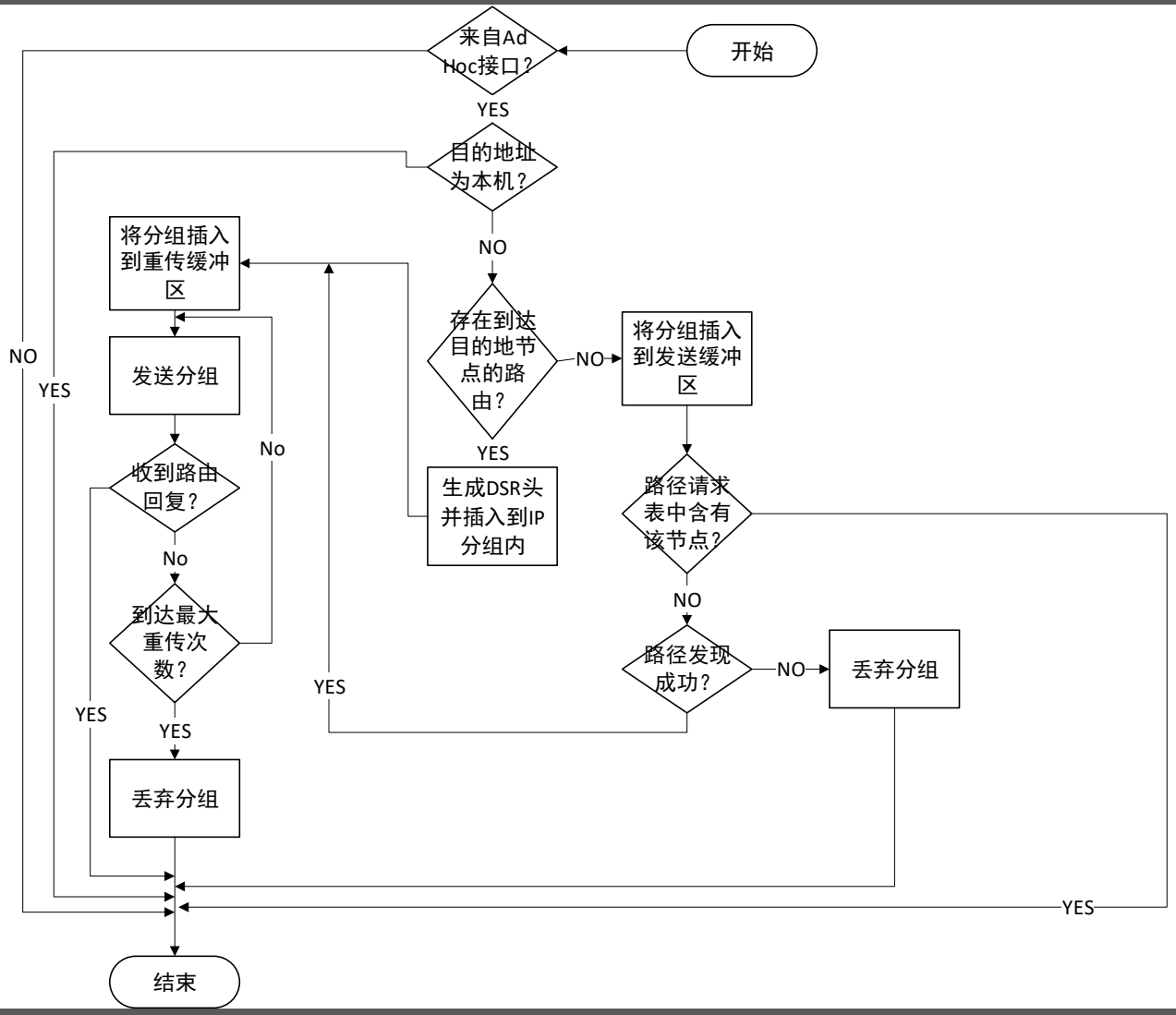
- TTL（Time To Live）
- 该节点上次向目标节点发起请求的时间。
- 为了得到有效的路由应答，该节点发出的路由寻找的次数。
- 该节点下一次尝试针对该目标节点的路由寻找的剩余时间量。
- 大小为RequestTableIds条目的FIFO缓存，其中包含来自源节点的最近路由请求的标识值和路由请求的表示地址目标地址。

实现

接收分组程序流程图


开始
DSR分组？
YES
NO
循环处理DSR选项
本机分组？
NO
YES
DSR控制信息分组？
YES
NO
丢弃分组
移交DSR头并将分组交给上层
分组转发
结束

发送分组程序流程图


开始
来自Ad Hoc接口？
YES
目的地址为本机？
NO
存在到达目的地节点的路由？
NO
将分组插入到发送缓冲区
YES
生成DSR头并插入到IP分组内
路径请求表中含有该节点？
NO
路径发现成功？
NO
丢弃分组
YES
YES
将分组插入到重传缓冲区
发送分组
收到路由回复？
No
No
到达最大重传次数？
YES
YES
丢弃分组
NO
YES
YES
结束

4 总结

DSR的优、缺点

- **DSR的优点：**
  - 路由维护只有节点之间需要沟通，以此减少路由维护开销；
  - 路由缓存可以进一步减少路由发现开销；
  - 单一的路由发现可能产生多条到目的节点的路由。

- **DSR的缺点：**
  - 包头大小随路由长度的增加而增加；
  - 路由请求可能在网络中产生洪泛；
  - 必须注意避免路由请求传播通过相邻节点之间的碰撞；
  - 可能出现路线回复风暴等类似的竞争问题。

谢谢大家！