# 11-网络路由实验报告

陈彦帆 2018K8009918002

## 1、实验内容

（1）基于已有代码框架，实现路由器生成和处理 mOSPF Hello/LSU 消息的相关操作，构建一致性链路状态数据库并测试。

（2）基于实验一，实现路由器计算路由表项的相关操作。

（3）在给定网络拓扑下运行 traceroute 测试，断开一条链路后，经过一段时间，再次测试。

## 2、实验流程

**（1）完成代码编写。**

**（2）测试并检验路由器生成的数据库信息：**

首先通过 mininet 环境(topo.py)构建网络拓扑如下图：

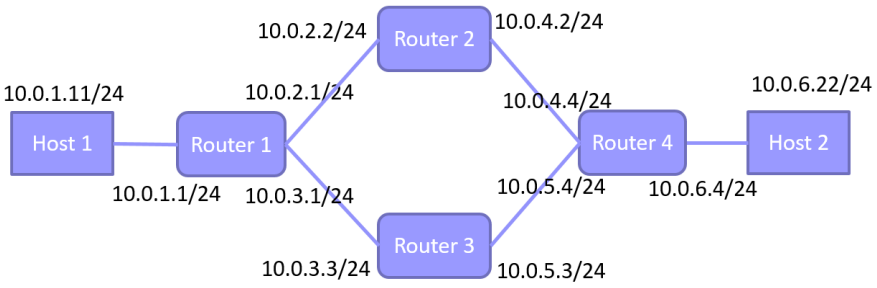


图 1 用于测试的网络拓扑

修改脚本文件部分语句如下：

```
for r in (r1, r2, r3, r4):
    r.cmd('./mospfd > %s-output.txt 2>&1 &' % r)
sleep(40)
for r in (r1, r2, r3, r4):
    r.cmd('pkill -SIGTERM mospfd')
    r.cmd('pkill -SIGTERM mospfd-reference')
```

然后运行脚本，得到测试结果。

**（3）测试 traceroute：**

构建网络拓扑如图 1。脚本修改如下：

```
for r in (r1, r2, r3, r4):
    r.cmd('./mospfd > %s-output.txt 2>&1 &' % r)
    sleep(38)
    CLI(net)
    net.stop()
```

运行脚本，输入 xterm h1，在 h1 窗口输入

traceroute 10.0.6.22

观察结果。在 mininet 窗口输入

link r2 r4 down

等待一段时间（约 30s，至少 3*HELLOINT）后再次进行 traceroute 测试。

## 3、分析与实现

### （1）总述

为了支持 mOSPF，每个路由节点需要完成的工作有：

① 发送和接收 hello 包，用于发现邻居节点。

② 对于所有邻居节点，发送和接收有效的 lsu 包并转发。lsu 包记录当前节点的链路状态（所有邻居节点的网络信息），通过洪泛，每个节点可以获得所有节点的链路状态数据。

③ 处理所有链路状态数据，用最短路算法生成每个网络的转发端口和网关，并相应地更新路由表。

④ 定期删除超时的邻居节点和链路状态数据。

其中，hello 包和 lsu 包都属于 mOSPF 协议，它工作在 IP 层，具有 IP 层的包头和自己的头部。其格式图 2-3 所示。

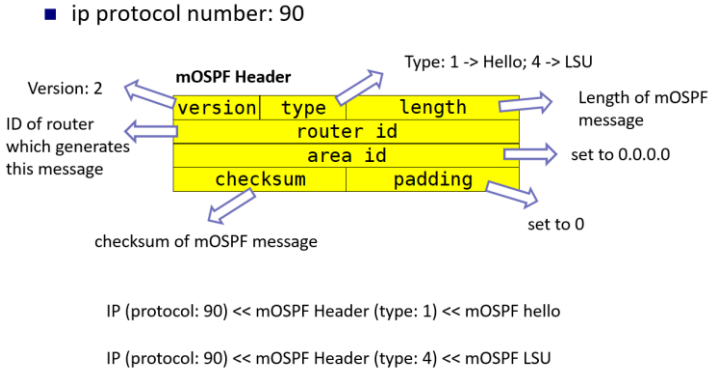


图 2 mOSPF 头部格式

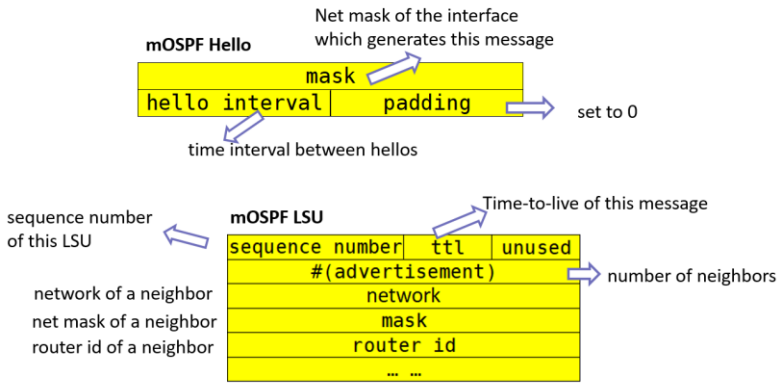


图 3 lsu 和 hello 包数据部分格式

**（2）发送和接收 Hello 包**

sending_mospf_hello_thread

间隔每个 HELLOINT(5s)，按图 3 的格式填充 hello 包，组播本节点的信息。相邻路由节点会首先接收到 hello 包，hello 包不会被转发，故每个节点得到的 hello 包信息都是邻居节点发出的。

handle_mospf_hello

对于收到的 hello 包，获得其节点信息。查找邻居节点链表，若存在此节点，则更新其生存时间，否则新增一个链表项存储该节点信息，并向每个邻居节点发送 lsu 包。

访问邻居节点链表时，需要上锁，以和清除过期邻居节点的线程互斥。

**（3）发送、接收和转发 lsu 包**

sending_mospf_lsu_thread

每隔一个 LSUINT(30s)，按图 3 的格式填充 lsu 包，向所有邻居节点发送。当端口没有相邻路由器时，也要表达该网络，邻居节点 ID 为 0。

handle_mospf_lsu

对于 ttl 不为 0 的 lsu 包，读取其链路状态信息。遍历链路状态数据库，若找到相应的节点且其信息较旧，则更新信息，否则新建一个节点存储其信息。若更新了信息或新建了节点，则更新路由表，然后向所有邻居节点转发该 lsu 包。

访问链路状态数据库时，需要上锁以实现互斥。（与访问邻居节点链表的锁不同）

**（4）计算最短路，更新路由表**

当链路状态数据库发生变化后，需要更新路由表。本实验采取的是重新生成路由表的方法。

首先移除所有非默认的路由表项。然后读取链路信息建立图结构：

采用邻接矩阵表示。先遍历链路状态数据库把链表转化为数组，然后遍历每个节点的链路信息，构建邻接矩阵。

运行 dijkstra 算法计算最短路并更新路由表。在本实验中，相邻路由之间的代价都相同，退化为广度优先搜索。用一个队列维护正在访问的项，每访问一个项，插入该项对应网络的路由表。

实现代码：

```
typedef struct {
  u32 rid;
  u32 gw;
  u32 mask;
  u32 network;
  int nadv;
  iface_info_t *iface;
}rnode_t;
static void update_rtable()
{
  //init data structure
  rnode_t rnode[MN];
  char vis[MN] = {0};
  int graph[MN][MNBR];
  int queue[MN];
  int i1=0,i2=0;
  int k=0;
  i1=i2=0;

  // delete rentries that are not nbr
  rt_entry_t *rentry, *q;
  list_for_each_entry_safe(rentry,q,&rtable,list){
    if(rentry->gw)
        remove_rt_entry(rentry);
  }

  // add node and id
  pthread_mutex_lock(&db_lock);
  mospf_db_entry_t *entry = NULL;
  list_for_each_entry(entry,&mospf_db,list){
    rnode[k].nadv = entry->nadv;
    rnode[k++].rid = entry->rid;
  }
  // add arc and info
  int found;
```

```
  int i=0;
  list_for_each_entry(entry,&mospf_db,list){
    for(int j=0;j<entry->nadv;j++){
      found = 0;
      for(int l=0;l<k;l++){
        if(rnode[l].rid==entry->array[j].rid){
          graph[i][j] = l;
          rnode[l].mask = entry->array[j].mask;
          rnode[l].network = entry->array[j].network;
          found = 1;
        }
      }
      if(!found) {
        rnode[k].rid = entry->array[j].network;
        graph[i][j] = k;
        rnode[k].mask = entry->array[j].mask;
        rnode[k].nadv = 0;
        rnode[k++].network = entry->array[j].network;
      }
    }
    i++;
  }
  pthread_mutex_unlock(&db_lock);
  pthread_mutex_lock(&mospf_lock);
  // init iface and gw
  iface_info_t *iface = NULL;
  mospf_nbr_t *nbr = NULL;
  list_for_each_entry(iface,&instance->iface_list,list){
    list_for_each_entry(nbr,&iface->nbr_list,list){
      for(int i=0;i<k;i++){
        if(rnode[i].rid == nbr->nbr_id){
          rnode[i].iface = iface;
          rnode[i].gw = nbr->nbr_ip;
          vis[i] = 1;
          fprintf(stdout,"vis "IP_FMT"\n",HOST_IP_FMT_STR(nbr->nbr_id));
          queue[i2++] = i;
        }
      }
      if(!found) fprintf(stdout,"init iface!"IP_FMT"\n",HOST_IP_FMT_STR(nbr->nbr_id
));
    }
  }
  // dijkstra when each arc's weight is the same,
  // i.e. BFS.
  // At the same time add rt entry.
```

```
  while(i2>i1){
    int th = queue[i1++];
    for(int i=0;i<rnode[th].nadv;i++){
      int next = graph[th][i];
      if(vis[next]) continue;
      rnode[next].iface = rnode[th].iface;
      rnode[next].gw = rnode[th].gw;
      rt_entry_t *rentry = new_rt_entry(rnode[next].network,rnode[next].mask,rnode[
th].gw,rnode[th].iface);
      add_rt_entry(rentry);
      vis[next] = 1;
      fprintf(stdout,"vis "IP_FMT"\n",HOST_IP_FMT_STR(rnode[next].rid));
      queue[i2++] = next;
    }
  }
  pthread_mutex_unlock(&mospf_lock);
}
```

**（5）定期删除超时的邻居节点和链路状态数据**

checking_database_thread

定期删除超时的链路状态数据，访问链路状态数据库时，需要上锁(db_lock)。注意，删除后不能立即更新路由表，否则可能和正在进行路由查找的主线程冲突。路由表也不宜上锁，因为申请和释放锁需要开销，可能影响路由查找的效率。

checking_nbr_thread

定期删除超时的邻居节点，访问邻居节点链表时，需要上锁(mospf_lock)。若删除成功，需要向所有邻居节点发送 lsu 包。

## 4、结果与讨论

**（1）测试并检验路由器生成的数据库信息：**

测试的网络拓扑如图 1。

一段时间后，r1,r2 打印的数据库信息和路由表信息如下所示。（r3,r4 为镜像节点，结果省略）

r1

RID Network Mask Nbr

10.0.2.2 10.0.2.0 255.255.255.0 10.0.1.1

10.0.2.2 10.0.4.0 255.255.255.0 10.0.4.4

10.0.3.3 10.0.3.0 255.255.255.0 10.0.1.1

10.0.3.3 10.0.5.0 255.255.255.0 10.0.4.4

10.0.4.4 10.0.4.0 255.255.255.0 10.0.2.2

10.0.4.4 10.0.5.0 255.255.255.0 10.0.3.3

10.0.4.4 10.0.6.0 255.255.255.0 0.0.0.0

Routing Table:

dest mask gateway if_name

---

10.0.1.0 255.255.255.0 0.0.0.0 r1-eth0

10.0.2.0 255.255.255.0 0.0.0.0 r1-eth1

10.0.3.0 255.255.255.0 0.0.0.0 r1-eth2

10.0.2.0 255.255.255.0 10.0.2.2 r1-eth1

10.0.5.0 255.255.255.0 10.0.2.2 r1-eth1

10.0.3.0 255.255.255.0 10.0.3.3 r1-eth2

10.0.6.0 255.255.255.0 10.0.2.2 r1-eth1

**r2**

RID Network Mask Nbr

10.0.1.1 10.0.1.0 255.255.255.0 0.0.0.0

10.0.1.1 10.0.2.0 255.255.255.0 10.0.2.2

10.0.1.1 10.0.3.0 255.255.255.0 10.0.3.3

10.0.4.4 10.0.4.0 255.255.255.0 10.0.2.2

10.0.4.4 10.0.5.0 255.255.255.0 10.0.3.3

10.0.4.4 10.0.6.0 255.255.255.0 0.0.0.0

10.0.3.3 10.0.3.0 255.255.255.0 10.0.1.1

10.0.3.3 10.0.5.0 255.255.255.0 10.0.4.4

Routing Table:

dest mask gateway if_name

---

10.0.2.0 255.255.255.0 0.0.0.0 r2-eth0

10.0.4.0 255.255.255.0 0.0.0.0 r2-eth1

10.0.1.0　255.255.255.0 10.0.2.1　r2-eth0

10.0.2.0　255.255.255.0 10.0.2.1　r2-eth0

10.0.5.0　255.255.255.0 10.0.2.1　r2-eth0

10.0.4.0　255.255.255.0 10.0.4.4　r2-eth1

10.0.6.0　255.255.255.0 10.0.4.4　r2-eth1

（2）测试 traceroute，删除 r2-r4 链路后经过一段时间再次测试：

测试的网络拓扑如图 1。

结果如下图：

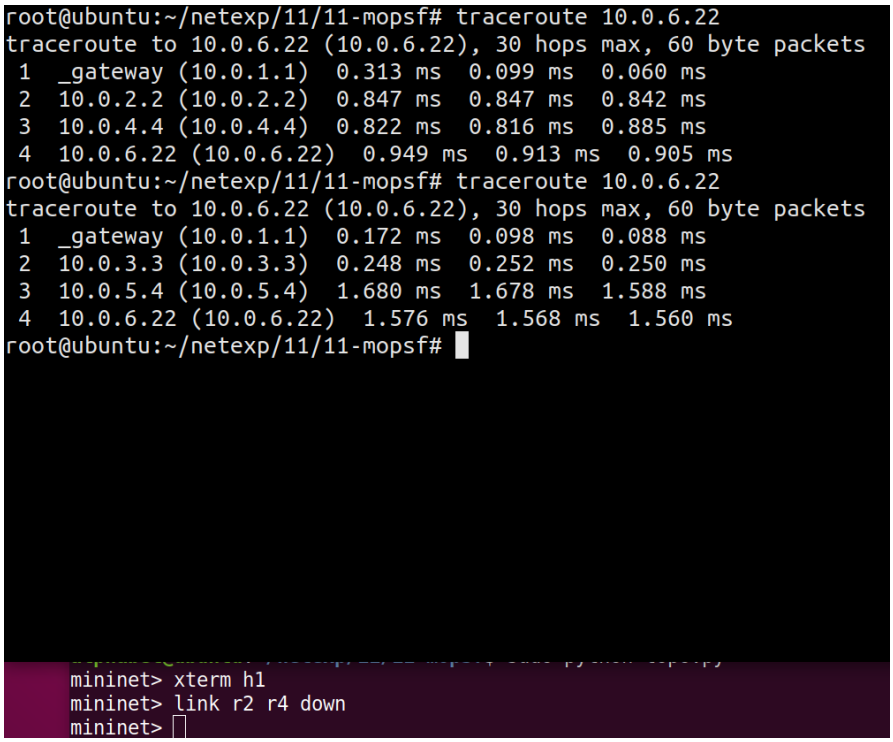


图 4

两次测试间隔应至少为 3*HELLOINT，这样路由节点才能发现并删除失效的邻居。比照图 1 的网络拓扑，图 4 中的结果是正确的。

## 5. 思考题

**（1）在构建一致性链路状态数据库中，为什么邻居发现使用组播(Multicast)机制，链路状态扩散用单播(Unicast)机制？**

邻居发现时还不知道邻居的信息，也就不知道其 IP 地址，无法使用单播，故采用组播，把参与 mOSPF 的路由器设置相同的组播地址，则可以收到相邻节点的 hello 包。

链路状态扩散时已经知道了邻居节点的 IP 地址信息，可以使用单播。而且无须向发送者转发 lsu 包，故不采用组播。

**（2）该实验的路由收敛时间大约为 20-30 秒，网络规模增大时收敛时间会进一步增加，如何改进路由算法的可扩展性？**

可以采用分层结构。划分为多个区域，每个区域内部采用 OSPF 方法，区域之间采用别的方法（如 BGP）。

**（3）路由查找的时间尺度为~ns，路由更新的时间尺度为~10s，如何设计路由查找更新数据结构，使得更新对查找的影响尽可能小？**

本实验中每次更新路由表都要把原来的非默认路由项全部删除后重新计算。在计算最短路时，也是重新计算。作为改进，可以不重新计算，而在计算最短路时，从更新链路信息的节点开始，只更新有受到影响的节点的路由信息，只对这些节点更新路由表。路由表可采用改进的 trie 树结构，其插入、删除和更新一项的时间是 O(log(n))的。由于更新项的数目应远小于原有项，故可以显著减少更新时间。

## 6. 遇到的问题

（1）“当端口没有相邻路由器时，也要表达该网络，邻居节点 ID 为 0。”在计算最短路时需要特别考虑此类情况。本实验采取的做法是，在图中为其新建一个结点，其 id 改为其网络号，出度为 0，入边指向与其相邻的路由节点。若不特殊处理，若有多个这样的网络，其 id 会产生冲突。