

lab07 实验报告

高鸣驹

2022 年 10 月 14 日

目录

1 实验目的	1
2 实验过程	1
2.1 发送 mospf Hello 数据包	1
2.2 处理 mospf Hello 数据包	1
2.3 邻居节点的老化更新	2
2.4 一致性链路状态数据库的节点的老化	2
2.5 一致性链路状态数据库的更新	3
2.5.1 图的创建	3
2.5.2 dijkstra 算法	4
2.5.3 数据库的更新	4
2.6 发送 mospf lsu 数据包	4
2.7 处理接收的 mospf lsu 数据包	5
3 实验结果	5
4 思考题	7
4.1 问题 1	7
4.2 问题 2	7
4.3 问题 3	7

1 实验目的

1. 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。
2. 基于 1，实现路由器计算路由表项的相关操作。

2 实验过程

2.1 发送 mospf Hello 数据包

总体思路是，遍历所有的节点的所有 iface 接口，每个接口都周期性的发送 mospf hello 数据包，宣告自己的存在。

这一部分的组包过程和之前实验的基本类似，我们的包有以下几个部分组成：

```

1  int pkt_len = ETHER_HDR_SIZE + IP_BASE_HDR_SIZE + MOSPF_HDR_SIZE + MOSPF_HELLO_SIZE;
2  struct ether_header *eh = (struct ether_header*)packet;
3  struct iphdr *ih = packet_to_ip_hdr(packet);
4  struct mospf_hdr *mh = (struct mospf_hdr*)((char*)ih + IP_BASE_HDR_SIZE);
5  struct mospf_hello *mhello = (struct mospf_hello*)((char*)mh + MOSPF_HDR_SIZE);

```

我们对各个部分分别初始化，初始化过程和之前也大致类似，主要有几点需要注意：

第一，我们的发包的目的 mac 地址为 01:00:5E:00:00:05，ip 地址为 224.0.0.5。

第二，我们在填好所有的 mospf hello content 后再填写 header 的 checksum。

第三，组包发包结束后，需要休眠一个 hello interval (5s)，间歇性发包。

2.2 处理 mospf Hello 数据包

这一部分主要是拆包操作，我们拆解出各个部分的报头和 mospf hello 报文内容。由于 mospf hello 包只可能是邻居节点发过来的，我们遍历这个接口的邻居链表，如果没有这个邻居节点的记录，我们就加入这个邻居节点到链表里，如果有的话就更新这个邻居节点的 alive 域，保证它不会被清除线程清除：

```

1  int found = 0;
2  mospf_nbr_t *nbr;
3  list_for_each_entry(nbr, &iface->nbr_list, list) {
4      if (nbr->nbr_id == ntohl(mh->rid)) {
5          found = 1;
6          nbr->alive = 0;
7          break;
8      }
9  }
10
11 if (!found) {
12     mospf_nbr_t *new_nbr = (mospf_nbr_t *)malloc(sizeof(mospf_nbr_t));
13     new_nbr->nbr_id = ntohl(mh->rid);
14     new_nbr->nbr_ip = ntohl(ih->saddr);
15     new_nbr->nbr_mask = ntohl(hello->mask);
16     new_nbr->alive = 0;
17     list_add_tail(&new_nbr->list, &iface->nbr_list);
18     iface->num_nbr++;
19 }

```

```

20     sending_mospf_lsu();
21 }

```

需要注意的是，我们如果更新这个接口的邻居节点的链表后，需要向周围节点通知，就是发 mospf lsu 包。

2.3 邻居节点的老化更新

这里我们的总体思路是：我们仍然遍历所有的 iface 接口，对于每个 iface 接口，我们看他的所有邻居是否在 3 倍的 hello-internal 时间内未更新（alive 域是否大于 3），如果是的话，则说明这个邻居已经老化，将其从邻居列表中删除，如果没有，则将邻居节点的 alive 域加 1。

这个线程每进行完这些操作后会休眠 1s 时间，保证 alive 域达到 $n \times \text{hello-internal}$ 时，这个邻居已经 n 倍的 hello-internal 时间未更新，便于老化判断：

```

1  while (1) {
2  pthread_mutex_lock(&mospf_lock);
3  list_for_each_entry(iface, &instance->iface_list, list) {
4      mospf_nbr_t *nbr = NULL, *q;
5      list_for_each_entry_safe(nbr, q, &iface->nbr_list, list) {
6          nbr->alive++;
7          if (nbr >= 3 * MOSPF_DEFAULT_HELLOINT) {
8              list_delete_entry(&nbr->list);
9              free(nbr);
10             iface->num_nbr--;
11             sending_mospf_lsu_thread(NULL);
12         }
13     }
14 }
15 pthread_mutex_unlock(&mospf_lock);
16 sleep(1);
17 }

```

这里还需要注意一点，这里我们的邻居节点发生了变动，我们需要给其它邻居节点更新链路状态信息，即发送 mospf-lsu 数据包。数据包包括节点 ID(mOSPF Header)、邻居节点 ID、网络和掩码 (mOSPF LSU)。

2.4 一致性链路状态数据库的节点的老化

这里的老化判断和 2.3 部分很像，这里是遍历 mospf db 的所有表项，看这个表项是否 40s 没有更新。

这个线程也是完成上述操作后会休眠 1s，用途和 2.3 相似。

```

1  while (1) {
2      pthread_mutex_lock(&mospf_db_lock);
3      mospf_db_entry_t *db_entry = NULL, *q;
4      list_for_each_entry(db_entry, q, &mospf_db, list) {
5          db_entry->alive++;
6          if (db_entry->alive >= MOSPF_DATABASE_TIMEOUT) {
7              list_delete_entry(&db_entry->list);
8              free(db_entry);
9          }
10     }
11 }
12 pthread_mutex_unlock(&mospf_db_lock);

```

```

13     sleep(1);
14 }

```

2.5 一致性链路状态数据库的更新

2.5.1 图的创建

我们这一部分的总体思路是：把 router 抽象为节点，链路抽象为边，对于每个 router，我们计算出这个 router 到其它 router 的最短路径。所以说，首先我们需要构建出这样一个抽象的图。

```

1  // Deal with graph
2  int graph[ROUTER_NUM][ROUTER_NUM] = {0};
3  int router[ROUTER_NUM] = {0};
4  int num = 0;
5
6  void init_graph(void) {
7      memset(graph, INT8_MAX, sizeof(graph));
8      mospf_db_entry_t *db;
9      router[0] = instance->router_id;
10     num = 1;
11     list_for_each_entry(db, &mospf_db, list) {
12         router[num++] = db->rid;
13     }
14     db = NULL;
15     list_for_each_entry(db, &mospf_db, list) {
16         int i, j;
17         int u, v;
18         for(i = 0; i < num; i++) {
19             if(router[i] == db->rid)
20                 break;
21         }
22         u = i;
23         for(i = 0; i < db->nadv; i++) {
24             if(db->array[i].rid) {
25                 for(j = 0; j < num; j++) {
26                     if(router[j] == db->array[i].rid)
27                         break;
28                 }
29                 v = j;
30                 graph[u][v] = 1;
31                 graph[v][u] = 1;
32             }
33         }
34     }
35 }

```

我们利用两个数组实现这个图的构建，第一个数组叫做 router 数组，这个数组用来存放图节点的下标和这个节点的 rid 的映射关系，第二个数组叫做 graph 数组，利用邻接矩阵存储节点和链路构成的图。

首先我们将图里每个节点的距离设置为无穷，然后对于我们所有的 router，遍历它相邻的节点，将这个节点和它相邻节点的距离设置为 1。

2.5.2 dijkstra 算法

Dijkstra 算法就是我们经典的算法，分为两步：找到离源点最小距离的为访问过的节点加到访问节点集合中；边的松弛：

```

1  void dijkstra(int prev[], int dist[]) {
2      int visit[ROUTER_NUM];
3      for (int i = 0; i < ROUTER_NUM; i++) {
4          dist[i] = INT8_MAX;
5          prev[i] = -1;
6          visit[i] = 0;
7      }
8
9      dist[0] = 0;
10
11     for (int i = 0; i < num; i++) {
12         // 找到距离最小节点
13         int j = -1;
14         for (int k = 0; k < num; k++) {
15             if (visit[k] == 0) {
16                 if (j == -1 || dist[k] < dist[j])
17                     j = k;
18             }
19         }
20         int u = j;
21         visit[u] = 1;
22         // 边的松弛
23         for (int v = 0; v < num; v++) {
24             if (!visit[v] && dist[u] + graph[u][v] < dist[v]) {
25                 dist[v] = dist[u] + graph[u][v];
26                 prev[v] = u;
27             }
28         }
29     }
30 }

```

2.5.3 数据库的更新

该函数根据 Dijkstra 算法获得的节点拓扑信息来进行更新路由表。对于每个节点，会根据 Dijkstra 算法匹配到前序节点。用递归的方式前递可以找到对于本节点而言，每一个其他的节点的下一条节点是多少，并以此更新路由表，从而确定到其他网络的下一跳网关地址、源节点的转发端口。

另外，实验初始化时，会从内核中读入到本地网络的路由条目，更新路由表时需要区分这些条目和计算生成的路由条目。本设计中非默认路由表会在一开始删去。

2.6 发送 mospf lsu 数据包

我们分为以下几步：

第一步：确定我们 array 域的元素数量，如果这个 iface 接口没有邻居，那么我们算这个 iface 接口一个节点，如果有邻居，我们计算它的所有邻居。

```

1  list_for_each_entry (iface, &instance->iface_list, list) {
2      if (!iface->num_nbr) {
3          nbr_sum++;
4      } else {
5          nbr_sum += iface->num_nbr;
6      }
7  }

```

第二步：对 array 域元素进行初始化。

```

1  list_for_each_entry (iface, &instance->iface_list, list) {
2      if (!iface->num_nbr) {
3          ml_array[pos].mask = htonl(iface->mask);
4          ml_array[pos].network = htonl(iface->ip & iface->mask);
5          ml_array[pos].rid = 0;
6          pos++;
7      } else {
8          mospf_nbr_t *nbr;
9          list_for_each_entry (nbr, &iface->nbr_list, list) {
10             ml_array[pos].mask = htonl(nbr->nbr_mask);
11             ml_array[pos].network = htonl(nbr->nbr_mask & nbr->nbr_ip);
12             ml_array[pos].rid = htonl(nbr->nbr_id);
13             pos++;
14         }
15     }
16 }

```

第三步：组包过程与第一部分大同小异，不多赘述。

2.7 处理接收的 mospf lsu 数据包

处理 mospf lsu 数据包的过程基本上就是组装这个数据包的逆过程，我们进行拆包。拆包之后我们找现在这个接口的邻居链表中是否有这个节点，如果有，并且发的包的 rid 较大，那么我们更新这个节点的信息。如果没有，我们在链表中加入这个节点的信息。

3 实验结果

我们中间 router 的配置如图一所示：我们等待生成一致的链路状态数据库后，在 h1 上 traceroute h2，结果如图 2 所示：

从图 2 结果中我们可以看到，Host1 经过了 r1, r2, r3 到达 h2，结果正确。

我们利用 link down 命令删掉 r2, r4 节点之间的链路，重复以上过程，结果如图三所示：

我们看出，链路状态发生了变化，Host1 经过 r1, r3, r4 到达了 r2，因为 r2, r4 之间的链路被禁用了，符合我们的预期。

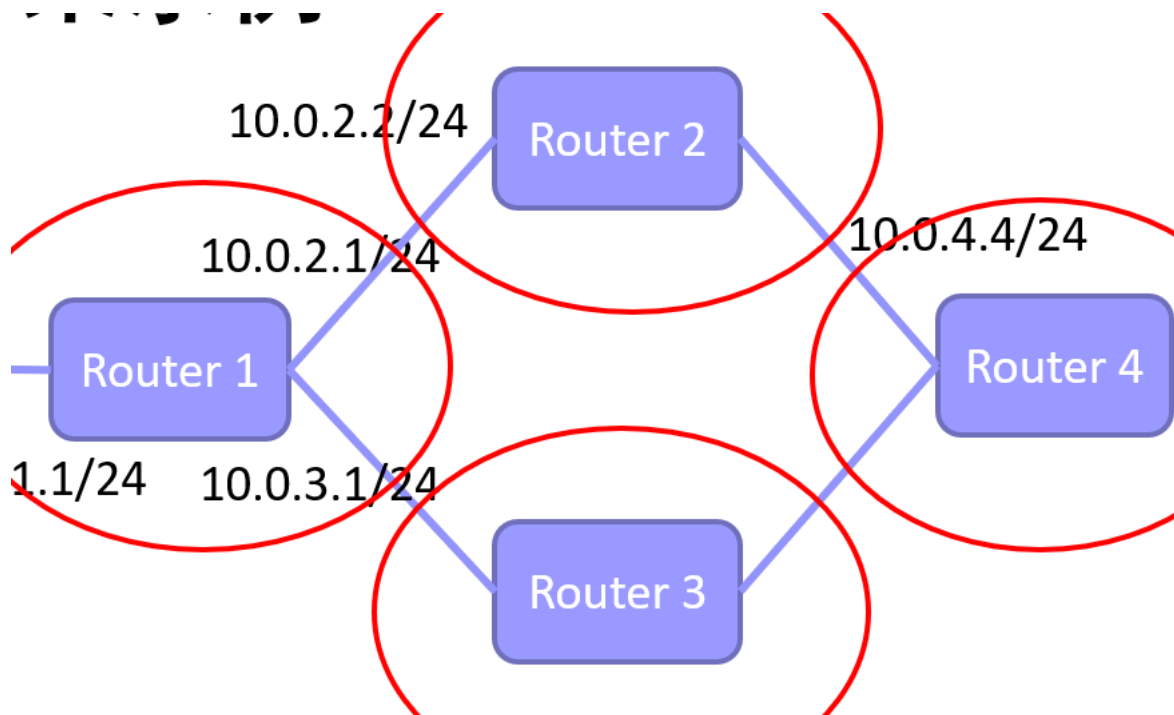
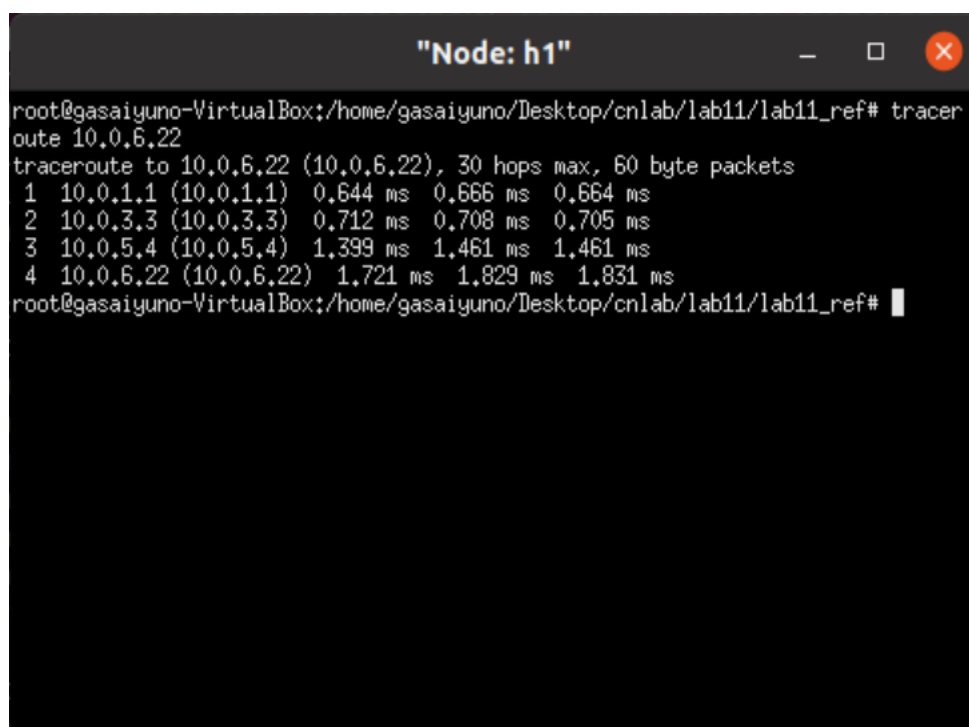


图 1: router 的配置

```
root@gasaiyuno-VirtualBox:/home/gasaiyuno/Desktop/cnlab/lab11/lab11_ref# traceroute 10.0.6.22
traceroute to 10.0.6.22 (10.0.6.22), 30 hops max, 60 byte packets
 1 10.0.1.1 (10.0.1.1) 0.538 ms 0.633 ms 0.631 ms
 2 10.0.2.2 (10.0.2.2) 1.103 ms 1.101 ms 1.098 ms
 3 10.0.4.4 (10.0.4.4) 1.095 ms 1.076 ms 1.071 ms
 4 10.0.6.22 (10.0.6.22) 1.067 ms 1.063 ms 1.058 ms
root@gasaiyuno-VirtualBox:/home/gasaiyuno/Desktop/cnlab/lab11/lab11_ref#
```

图 2: 完整链路的 tracetroute



```
"Node: h1"
root@gasaiyuno-VirtualBox:/home/gasaiyuno/Desktop/cnlab/lab11/lab11_ref# traceroute
oute 10.0.6.22
traceroute to 10.0.6.22 (10.0.6.22), 30 hops max, 60 byte packets
 1  10.0.1.1 (10.0.1.1)  0.644 ms  0.666 ms  0.664 ms
 2  10.0.3.3 (10.0.3.3)  0.712 ms  0.708 ms  0.705 ms
 3  10.0.5.4 (10.0.5.4)  1.399 ms  1.461 ms  1.461 ms
 4  10.0.6.22 (10.0.6.22)  1.721 ms  1.829 ms  1.831 ms
root@gasaiyuno-VirtualBox:/home/gasaiyuno/Desktop/cnlab/lab11/lab11_ref#
```

图 3: 缺失 r2 到 r4 链路的 tracetroute

4 思考题

4.1 问题 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？

邻居发现过程中每个结点周期性地向邻居结点发送 mospf hello 包，发送周期一般较短，因此使用组播能够大幅度地减轻网络负载。然而，组播与单播相比没有纠错机制，发生丢包错包后难以弥补，链路扩散对信息准确度要求比较高，因此使用单播机制保证发送信息的准确性。

4.2 问题 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？

网络规模较大的时候，可以把网络划分区域，在这个每个特定的区域使用洪泛机制进行网络拓扑的构建。在一个区域内部的路由器只知道本区域的完整网络拓扑，而不知道其他区域的网络拓扑的情况。增强了可扩展性。

4.3 问题 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？

由于我们这里的链路数据库查找和更新是不可以同步进行的，因为更新的时候会给数据库上锁，查找进程就只

能等待，大大影响了效率。我们可以采用数据库的概念，将表进行不同层次的划分（数据库，页，块，记录等等），同时拓宽锁的类型，在不同层次分别上 S, X, IS, SIX, IX 等不同类型的锁，保证对一个数据的更改最小程度影响对其它数据的查找，而不是直接把整个数据库锁住