

11-OSPF实验-report

作者：苗屹松

日期：2017/12月/3日

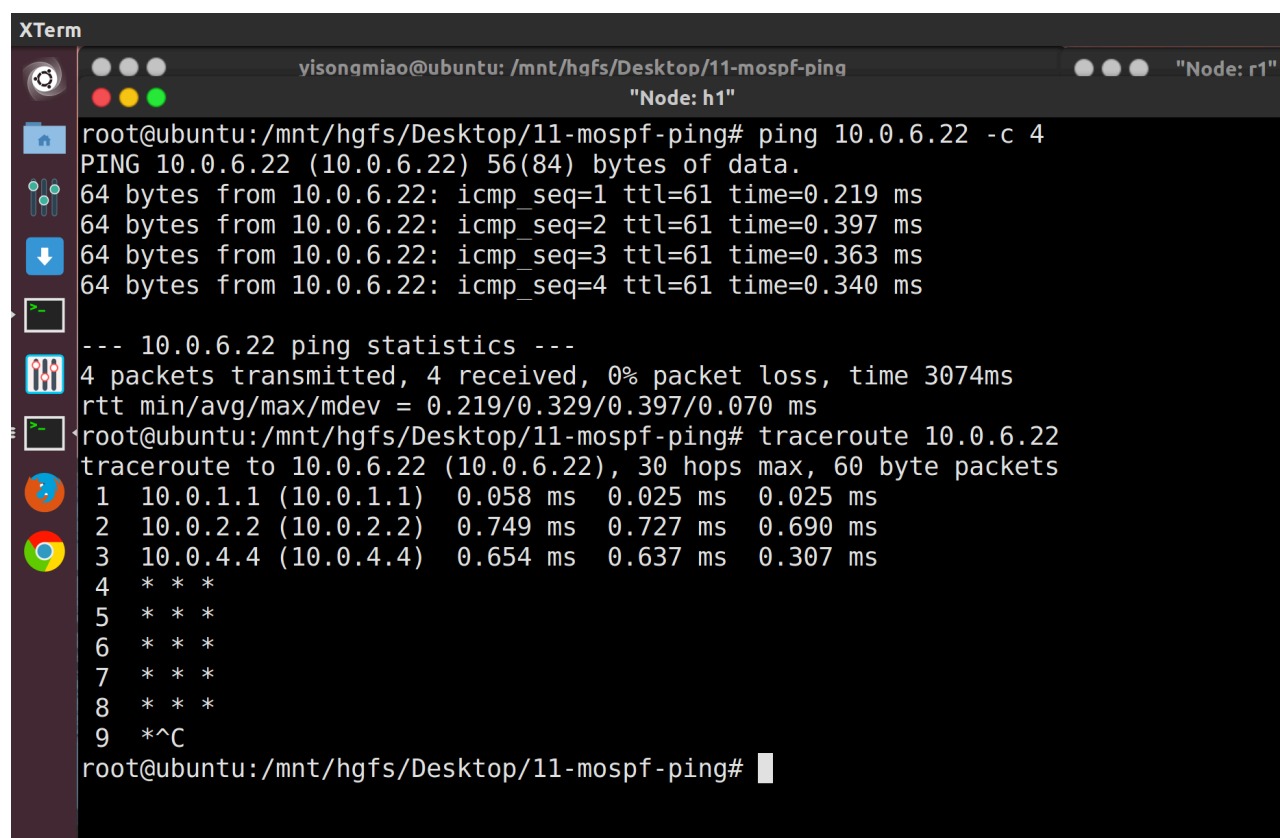
1. Overview

这次实验是OSPF的第二部分，用已经生成好的链路状态数据库来完成各个router的route_table。

因为上一次实验我做的比较好，所以这次并不是很费力。

我最大的收获时领会到 信息的编码 的感觉，算是最大的收获！将会在第三部分详细描述。

2. 实验结果



The screenshot shows an XTerm window with the title bar "visongmiao@ubuntu: /mnt/hgfs/Desktop/11-mospf-ping". The terminal output is as follows:

```
root@ubuntu:/mnt/hgfs/Desktop/11-mospf-ping# ping 10.0.6.22 -c 4
PING 10.0.6.22 (10.0.6.22) 56(84) bytes of data.
64 bytes from 10.0.6.22: icmp_seq=1 ttl=61 time=0.219 ms
64 bytes from 10.0.6.22: icmp_seq=2 ttl=61 time=0.397 ms
64 bytes from 10.0.6.22: icmp_seq=3 ttl=61 time=0.363 ms
64 bytes from 10.0.6.22: icmp_seq=4 ttl=61 time=0.340 ms

--- 10.0.6.22 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3074ms
rtt min/avg/max/mdev = 0.219/0.329/0.397/0.070 ms
root@ubuntu:/mnt/hgfs/Desktop/11-mospf-ping# 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)  0.058 ms  0.025 ms  0.025 ms
 2  10.0.2.2 (10.0.2.2)  0.749 ms  0.727 ms  0.690 ms
 3  10.0.4.4 (10.0.4.4)  0.654 ms  0.637 ms  0.307 ms
 4  * * *
 5  * * *
 6  * * *
 7  * * *
 8  * * *
 9  * ^C
root@ubuntu:/mnt/hgfs/Desktop/11-mospf-ping#
```

如上图所示，一个ping包从h1出发，历经3个路由器，它们的网关分别是10.0.1.1, 10.0.2.2, 10.0.4.4，最终到达h2。

达到实验目的！

(武老师在群里说的，想办法改进一下，在第四条traceroute就退出，目前还没有思路，之后会想办法)

3. 最大收获

这次实验的最大收获是感受到 信息的编码 的感觉。

我只学了一点微小的计算机网络知识，以我愚见，一个很重要的问题就是把信息编码成方便传输的形式。

在第7次router实验中，我第一次接触各种形式的packet，有 arp, ICMP, 还有 IP.

当时觉得比较好玩，我发送了一个包，居然可以被我的程序and/or Linux协议栈自动的解析出里面的信息。

在一次晚餐的时候我和汪诗洋同学交流，他说：

计算机科学某种程度上只 处理信息,但并不产生新的信息！

我觉得这句话很有道理，比如在这次实验中，我有以下断言：

一致性链路状态数据库 与 路由表 本质上是一样的！

具体而言：

1.我们从一致性链路状态数据库出发，构建出路由器的抽象拓普，写出其邻接矩阵

2.从这个邻接矩阵，计算最短路径，得到路由表

在1,2两步里，没有输入别的信息，也没有输出任何新信息！

很奇妙！

4. 具体实现

4.1 把没有接收到hello的interface也告知这个网络

我使用的方法是用LSU。

比如r4，它发送的LSU的 `nadv = 3`

其中3个entry分别是：r2, r3 和 r4-eth2

这样r1~r3就可以知道10.0.6.0这个子网的存在！

4.2 求最短路径

```

void dijkstra(int input_graph[NUM_VERTICES][NUM_VERTICES], int src, int
dist[NUM_VERTICES], int prev[NUM_VERTICES], int purpose[NUM_VERTICES], int
subnet_graph[NUM_VERTICES][NUM_VERTICES]){
    int sptSet[NUM_VERTICES];
    int i;
    for(i = 0; i < NUM_VERTICES; i++){
        dist[i] = INT_MAX;
        sptSet[i] = 0;
        prev[i] = -1;
    } // do the initialization

    dist[src] = 0;

    int count;
    for(count = 0; count < NUM_VERTICES - 1; count++){
        int u = minDistance(dist, sptSet);
        sptSet[u] = 1;
        int v;
        for(v = 0; v < NUM_VERTICES; v++){
            if(sptSet[v] == 0 && input_graph[u][v] && dist[u] + input_graph[u]
[v] < dist[v]){
                dist[v] = dist[u] + input_graph[u][v];
                prev[v] = u;
                purpose[v] = subnet_graph[u][v];
            }
        }
    }
}

```

其中，我有两个辅助的list:

- (1) `prev` 用来 `回溯`： `prev[v] = u`代表: `v`节点的前一跳是`u`，我们不断回溯就寻找到整条路径：

```

int find_desired_index(int prev[NUM_VERTICES], int dest_index){ // actually a
backsource
    int current_index = dest_index;
    while(prev[prev[current_index]] != -1){
        current_index = prev[current_index];
    }
    return current_index;
}

```

用一个简单的while循环就可完成回溯，找到从源点出发后的第一跳节点——即我们所渴求的信息！

- (2) `purpose` 用来记录: `purpose[v]`表示从源点到`v`节点的目的是什么

`purpose[v] = subnet[u][v` 表示，这个目的是 连接节点u和v的子网。

而它也就是每个router_entry中的 主键: subnet!

4.3 如何获取下一跳网关?

我采用的方法是这样的:

首先抽象一下问题:

通过最短路径算法和我的两个数组，我们已知所要解决的subnet是谁，还知道当前路由器的rid，也已知下一跳路由器的rid，现在要得到下一跳的网关!

而下一跳的网关是什么? 其实就是目的路由器的一个interface的IP地址!

所以我的方法是:

```
for each iface in current router:
    for each subnet in ospf_db for the target router:
        if (iface->ip & mask) == subnet:
            return ...
```

以上代码就找到了当前路由器的转发端口，以及下一跳网关

4.4 更新路由表

```

int found = 0;
rt_entry_t *entry = NULL;
list_for_each_entry(entry, &rtable, list){
    if(entry->dest == current_subnet){ // refresh the content of
the entry

        printf("Refresh the content of current entry\n");
        found = 1;
        entry->gw = desired_gw;
        entry->mask = desired_mask;
        strcpy(entry->if_name, desired_if_name);
        entry->iface = desired_iface;
    }
}
if(found == 0){
    printf("Need to add a new entry\n");
    rt_entry_t * new_rt_entry = malloc(sizeof(rt_entry_t));
    new_rt_entry->dest = current_subnet;
    new_rt_entry->gw = desired_gw;
    new_rt_entry->mask = desired_mask;
    strcpy(new_rt_entry->if_name, desired_if_name);
    new_rt_entry->iface = desired_iface;

    list_add_tail(&new_rt_entry->list, &rtable);
}

```

更新路由表的方法非常简单：我们每4秒完成以下流程： 一致性链路数据库 —> 计算邻接矩阵与最短路径 —> 更新路由表

更新路由表的细节是：如果当前的subnet：可以在路由表中找到，则更新其内容： gw, mask, if_name, iface

如果找不到，则插入新的entry!

非常简单～

5. Reference & Give Credit

1. 最短路径Dijkstra算法，虽然大二的算法课我写过，但我还是参考了Geekforgeek的实现，我觉得这个网站对于基础数据结构与算法是个很好的reference! :

<http://www.geeksforgeeks.org/greedy-algorithms-set-6-dijkstras-shortest-path-algorithm/>

2. 谢谢方言歌同学，他转发给了我老师关于没有收到hello消息的interface的处理方法
3. 谢谢林海涛同学，在我很puzzled的时候，他建议我用LSU的方式来发送没有收到hello消息的interface的信息，来告知其他路由器这个interface的存在。我后来也的确采用了这个方法。