

网络路由实验报告

张磊 2017K8009922027

一、实验题目

网络路由实验

二、实验内容

1. 基于已有代码框架，实现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；
运行 ./mospfd，使得各个节点生成一致的链路状态数据库；
2. 基于 1，实验路由器计算路由表项的相关操作；
3. 运行试验：
运行 ./mospfd，使得各个节点生成一致的链路状态数据库；
等待一段时间后，每个节点生成完整的路由表项；
在节点 h1 上 ping/traceroute 节点 h2；
关掉某个节点或链路，等待一段时间后，再次用 h1 去 traceroute 节点 h2；

三、实验流程

1. 基于附件中的代码，完成 mospf_daemon.c 中相关函数的的编写，实现路由器对 mOSPF Hello/LSU 消息的处理；
2. 实现一个打印数据库信息的函数 show_database()，每当数据库信息发生变化时就打印出数据库信息；
3. 运行 ./mospfd，观察各个节点的链路数据库是否一致；
4. 增加一个更新路由表的线程，完成对路由表的更新，每次更新后打印新的路由表信息；
5. 运行 ./mospfd，观察各个节点的路由表信息是否正确；
6. 在 h1 上 ping/traceroute 节点 h2；
7. 关闭节点 r2，等待一段时间后(大约 10s)，各节点完成更新；
8. 再次在 h1 上 ping/traceroute 节点 h2；

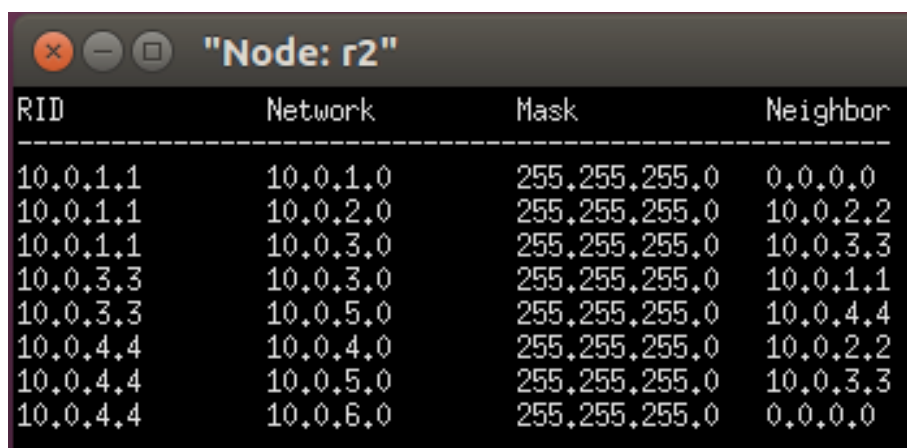
四、 实验结果

1. 实验内容一：

"Node: r1"

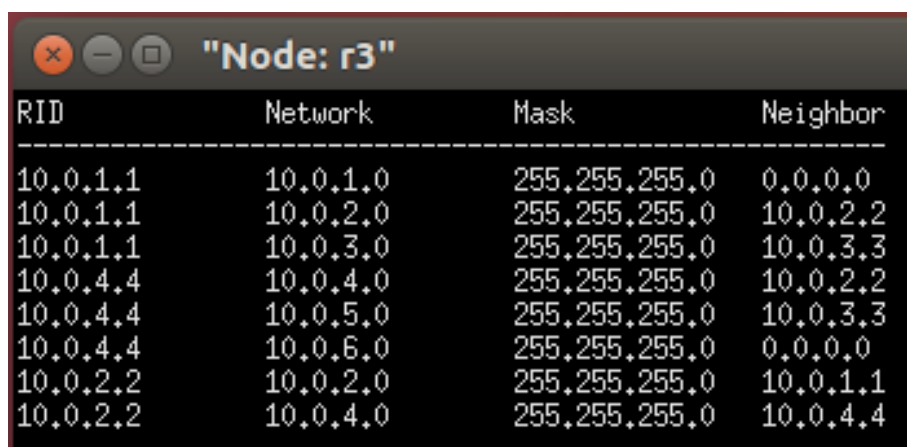
RID	Network	Mask	Neighbor
10.0.2.2	10.0.2.0	255.255.255.0	10.0.1.1
10.0.2.2	10.0.4.0	255.255.255.0	10.0.4.4
10.0.3.3	10.0.3.0	255.255.255.0	10.0.1.1
10.0.3.3	10.0.5.0	255.255.255.0	10.0.4.4
10.0.4.4	10.0.4.0	255.255.255.0	10.0.2.2
10.0.4.4	10.0.5.0	255.255.255.0	10.0.3.3
10.0.4.4	10.0.6.0	255.255.255.0	0.0.0.0

R1 Database

A screenshot of a terminal window titled "Node: r2". It displays a table with four columns: RID, Network, Mask, and Neighbor. The table contains eight rows of data representing the routing table for router R2.

RID	Network	Mask	Neighbor
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
10.0.3.3	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

R2 Database

A screenshot of a terminal window titled "Node: r3". It displays a table with four columns: RID, Network, Mask, and Neighbor. The table contains eight rows of data representing the routing table for router R3.

RID	Network	Mask	Neighbor
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
10.0.2.2	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

R3 Database

"Node: r4"			
RID	Network	Mask	Neighbor
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
10.0.1.1	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

R4 Database

2. 实验内容二(所有节点正常运行):

"Node: r1"			
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
10.0.4.0	255.255.255.0	10.0.2.2	r1-eth1
10.0.5.0	255.255.255.0	10.0.3.3	r1-eth2
10.0.6.0	255.255.255.0	10.0.2.2	r1-eth1

R1 Rtable

"Node: r2"			
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
10.0.3.0	255.255.255.0	10.0.2.1	r2-eth0
10.0.5.0	255.255.255.0	10.0.4.4	r2-eth1
10.0.6.0	255.255.255.0	10.0.4.4	r2-eth1

R2 Rtable

"Node: r3"

Routing Table:

dest	mask	gateway	if_name
10.0.3.0	255.255.255.0	0.0.0.0	r3-eth0
10.0.5.0	255.255.255.0	0.0.0.0	r3-eth1
10.0.1.0	255.255.255.0	10.0.3.1	r3-eth0
10.0.2.0	255.255.255.0	10.0.3.1	r3-eth0
10.0.4.0	255.255.255.0	10.0.5.4	r3-eth1
10.0.6.0	255.255.255.0	10.0.5.4	r3-eth1

R3 Rtable

"Node: r4"

Routing Table:

dest	mask	gateway	if_name
10.0.4.0	255.255.255.0	0.0.0.0	r4-eth0
10.0.5.0	255.255.255.0	0.0.0.0	r4-eth1
10.0.6.0	255.255.255.0	0.0.0.0	r4-eth2
10.0.2.0	255.255.255.0	10.0.4.2	r4-eth0
10.0.3.0	255.255.255.0	10.0.5.3	r4-eth1
10.0.1.0	255.255.255.0	10.0.4.2	r4-eth0

R4 Rtable

"Node: h1"

```

root@zhanglei-VirtualBox:~/Desktop/08-mospf# ping 10.0.6.22 -c 10
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=0.236 ms
64 bytes from 10.0.6.22: icmp_seq=2 ttl=61 time=0.188 ms
64 bytes from 10.0.6.22: icmp_seq=3 ttl=61 time=0.179 ms
64 bytes from 10.0.6.22: icmp_seq=4 ttl=61 time=0.257 ms
64 bytes from 10.0.6.22: icmp_seq=5 ttl=61 time=0.179 ms
64 bytes from 10.0.6.22: icmp_seq=6 ttl=61 time=0.572 ms
64 bytes from 10.0.6.22: icmp_seq=7 ttl=61 time=0.171 ms
64 bytes from 10.0.6.22: icmp_seq=8 ttl=61 time=0.179 ms
64 bytes from 10.0.6.22: icmp_seq=9 ttl=61 time=0.120 ms
64 bytes from 10.0.6.22: icmp_seq=10 ttl=61 time=0.182 ms

--- 10.0.6.22 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9201ms
rtt min/avg/max/mdev = 0.120/0.226/0.572/0.121 ms
root@zhanglei-VirtualBox:~/Desktop/08-mospf#

```

H1 ping 10.0.6.22 (H2)

```

Node: h1
root@zhanglei-VirtualBox:~/Desktop/08-mospf# 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)  0.421 ms  0.381 ms  0.368 ms
 2  10.0.2.2 (10.0.2.2)  0.640 ms  0.631 ms  0.623 ms
 3  10.0.4.4 (10.0.4.4)  0.679 ms  0.672 ms  0.665 ms
 4  10.0.6.22 (10.0.6.22)  0.655 ms  0.648 ms  0.639 ms
root@zhanglei-VirtualBox:~/Desktop/08-mospf#
```

H1 traceroute 10.0.6.22 (H2)

3. 实验内容二(关闭 R2 之后):

```

Node: h1
root@zhanglei-VirtualBox:~/Desktop/08-mospf# ping 10.0.6.22 -c 10
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=0.167 ms
64 bytes from 10.0.6.22: icmp_seq=2 ttl=61 time=0.181 ms
64 bytes from 10.0.6.22: icmp_seq=3 ttl=61 time=0.182 ms
64 bytes from 10.0.6.22: icmp_seq=4 ttl=61 time=0.189 ms
64 bytes from 10.0.6.22: icmp_seq=5 ttl=61 time=0.172 ms
64 bytes from 10.0.6.22: icmp_seq=6 ttl=61 time=0.254 ms
64 bytes from 10.0.6.22: icmp_seq=7 ttl=61 time=0.186 ms
64 bytes from 10.0.6.22: icmp_seq=8 ttl=61 time=0.180 ms
64 bytes from 10.0.6.22: icmp_seq=9 ttl=61 time=0.180 ms
64 bytes from 10.0.6.22: icmp_seq=10 ttl=61 time=0.195 ms

--- 10.0.6.22 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9194ms
rtt min/avg/max/mdev = 0.167/0.188/0.254/0.027 ms
root@zhanglei-VirtualBox:~/Desktop/08-mospf#
```

H1 ping 10.0.6.22 (H2)

```

Node: h1
root@zhanglei-VirtualBox:~/Desktop/08-mospf# 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)  0.054 ms  0.012 ms  0.010 ms
 2  10.0.3.3 (10.0.3.3)  0.036 ms  0.023 ms  0.023 ms
 3  10.0.5.4 (10.0.5.4)  0.046 ms  0.037 ms  0.038 ms
 4  10.0.6.22 (10.0.6.22)  0.067 ms  0.042 ms  0.044 ms
root@zhanglei-VirtualBox:~/Desktop/08-mospf#
```

H1 traceroute 10.0.6.22 (H2)

五、 实验分析

1. 实验一中, 我们观察到各个节点都形成了一致的链路数据库, 实验内容一成功;
2. 实验二中, 我们观察到各个节点都生成了正确的路由表项, 在 h1 上 ping 节点 h2 时, 可以 ping 通; 在 h1 上 traceroute 节点 h2 时, 可以看到消息依次经过了 r1, r2, r4 到达 h2;
3. 在关闭 r2 之后, 等待一段时间, 各个节点完成链路数据库和路由表项的更新, 在此在 h1 上 ping 节点 h2, 成功 ping 通, 并且在 h1 上 traceroute 节点 h2 时, 发现路由路径由原来的 h1->r1->r2->r4->h2 改变为了现在的 h1->r1->r3->r4->h2, 实验二内容成功;

六、 反思总结

1. 本次实验代码量还是比较大, 并且在使用 Dijkstra 算法计算最短路径时比较麻烦, 所以耗费时间比较长, 但是通过编写代码和 DEBUG, 让我对路由器如何交换路由信息, 自动生成路由表有了更加深刻的理解, 对 Hello 和 LSU 两种 mOSPF 消息的格式都有了更加深刻的记忆和理解;
2. 在进行路由表的更新的时候我尝试了两种操作, 一种是在完成 handle_lsu_packet 之后更新路由表, 这样做的好处是每次路由信息发生变化都能及时的更新路由表, 缺点是, 当网络中的节点数量变多后, 更新路由表的信息的操作会变得非常频繁;
3. 因此我采用了第二种方法, 单独创建一个用于更新路由表的线程, 每隔一段时间 (实验中我设置为 10 秒) 更新一次路由表, 这样做的好处是, 即使网络中的节点很多, 也不会频繁的更新路由表, 因为我们知道, 网络中链路很少会在极短时间内发生变化, 如果每次收到 LSU 消息都去更新一次路由表的话, 很多都是无效的操作, 因为路由表并没有产生变化, 这样做的缺点是可能无法及时的更新路由表中变化的信息, 但是这个可以通过手动配置刷新时间来改变;

4. 最理想的方式是只在收到的 LSU 消息中包含变化了的路由信息时才更新路由表，但是我暂时没找到如何区别路由信息是否发生变化的方式，一个可行的方法是在 LSU 消息中增加一个数据项，用来表示链路数据库是否发生变化，然后每个节点就可以在 `handle_lsu_packet` 时根据发送消息节点的链路数据是否发生变化来确定是否需要更新路由表；

七、 参考文献

i

ii

ⁱ 中国科学院大学 2020 春计算机网络研讨课 08-网络路由实验课件

ⁱⁱ 中国科学院大学 2020 春计算机网络研讨课 08-网络路由实验附件代码