计算机网络实验报告-08

阮星程
2015K8009929047

一、 实验题目

路由器转发实验。

二、 实验内容

理解路由器转发的内容和原理，构建路由器程序，实现相应的路由转发以及回复。并完成下列实验任务：

- 在给定网络拓扑上，依次 ping r1，h2，h3，一个不可达的终端，一个不可达的网络，路由器能够正确完成对应处理。
- 构建一个新的有多个路由器的拓扑网络，各个终端节点之间能够 ping 通，并且能够通过 traceroute 程序获取正确的路径信息。

三、 实验过程

本次代码较多，顺序叫乱，不再一一展示，选出几个较为核心的程序片段稍加说明。

程序运行从处理 ip 和 arp 的 packet 开始。

```
// handle ip packet
//
// If the packet is ICMP echo request and the destination IP address is equal to
// the IP address of the iface, send ICMP echo reply; otherwise, forward the
// packet.
void handle_ip_packet(iface_info_t *iface, char *packet, int len)
{
    struct iphdr * ip = packet_to_ip_hdr(packet);

    u32 dst = ntohl(ip->daddr);
    rt_entry_t *entry = longest_prefix_match(dst);
    if (!entry) {
```

处理 ip 的 packet，首先检测是否对应的 ip 地址存在于转发表中，如果不在，回复 ICMP 网络不可达信息。

```
//if out of ttl
if(--ip->ttl <= 0){
    //make up IP Header info of ICMP packet
    char * packet_buf = (char * )malloc(ETHER_HDR_SIZE + ntohs(ip->tot_len) + 40);
    memset(packet_buf, 0, ETHER_HDR_SIZE*sizeof(char));
    memcpy(packet_buf + ETHER_HDR_SIZE*sizeof(char), packet + ETHER_HDR_SIZE, ntohs(ip->tot_l
    memcpy(packet_buf + ETHER_HDR_SIZE*sizeof(char) + ip->ihl*4 + 8, packet + ETHER_HDR_SIZE,
    ip = packet_to_ip_hdr(packet_buf);
    ip->daddr = ip->saddr;
    ip->saddr = iface->ip;
    ip->protocol = 1;
    ip->tot_len = htons(ip->ihl*4 + 8 + ip->ihl*4 + 8);
    ip->checksum = ip_checksum(ip);

    //send
    icmp_send_packet(packet_buf, ntohs(ip->tot_len) + ETHER_HDR_SIZE, ICMP_TIME_EXCEEDED, ICM
    return ;
}
```

减 ttl，并判断是否超时，如超时便回复 ICMP 信息。中间的程序段由于最初设计的缘故，和几个发送 ICMP 信息的部分高度重复。所以看一个就好啦。这里我的架构是将 ip 部分构建好以及 icmp 的 data 部分构建好后送入 icmp_send_packet 函数，由它负责填充 icmp 头部已经对应的 checksum,再交由 ip_send_packet 添加本跳的 mac 并利用函数 iface_send_packet_by_arp 最终填充对端的 mac 完成包的发送。由于最后一个函数中的 lookup 函数需要锁，所以在调用发送包的地方都会释放 arp 的锁。

```
//if it is ICMP packet
if(ntohl(ip->daddr) == iface->ip){
    struct icmphdr * ich = (struct icmphdr *)(ip + 1);
    //log(DEBUG,"ICMP message &ip %x, ip ihl * 4 %d, &ich %x  ", ip, ip->ihl * 4, ich);

    u8 type = ich->type;
    if(type == 0){
        log(DEBUG,"ICMP received, type : 0, echo reply received \n ");
    }
    else if(type == 3){
        if(ich->code == 0){
            log(DEBUG,"ICMP received, type : 3, NET Unreachable \n ");
        }
        else{
            log(DEBUG,"ICMP received, type : 3, HOST Unreachable \n ");
        }
    }
    else if(type == 11){
        log(DEBUG,"ICMP received, type : 11,Time Exceeded  \n ");
    }
    else if(type == 8){
        log(DEBUG,"ICMP received, type : 8, echo received  \n ");
```

检查是否为 ICMP 包，如是的话做对应处理。如果不是的话，调用 ip_forward_packet 转发即可。

```c
void handle_arp_packet(iface_info_t *iface, char *packet, int len)
{
    log(DEBUG, "arp packet recived");
    struct ether_arp * arp = (struct ether_arp * )(packet + ETHER_HDR_SIZE);

    //if it is request
    if(arp->arp_op == htons(0x0001) && arp->arp_tpa == htonl(iface->ip)){
        log(DEBUG, "it is echo to %x", iface->ip);
        struct ether_arp buf;
        memcpy(&buf, arp, 6);
        buf.arp_op = htons(0x0002);
        memcpy(buf.arp_sha, iface->mac, ETH_ALEN);
        buf.arp_spa = htonl(iface->ip);
        memcpy(buf.arp_tha, arp->arp_sha, 10);
        arpcache_insert(ntohl(arp->arp_spa), arp->arp_sha);
        arp_send_reply(iface, &buf);

    }
    else{
        arpcache_insert(ntohl(arp->arp_spa), arp->arp_sha);
    }

}
```

对于 arp 包的处理就简洁很多，收到一个 arp 包时判断是请求还是回应，是请求的话回应对应的 arp 包，并将新的 mac 条目插入映射表。如果是回应的话，那就将 mac 条目插入映射表。在插入动作中会取出对应等待列表中的 packet 并将它们发送出去。

在 cache 部分，sweep 函数要做的事比之前的 sweep 要多不少，主要体现在对于等待队列的 arp 请求重发以及超过发送次数后的 ICMP 包回复。

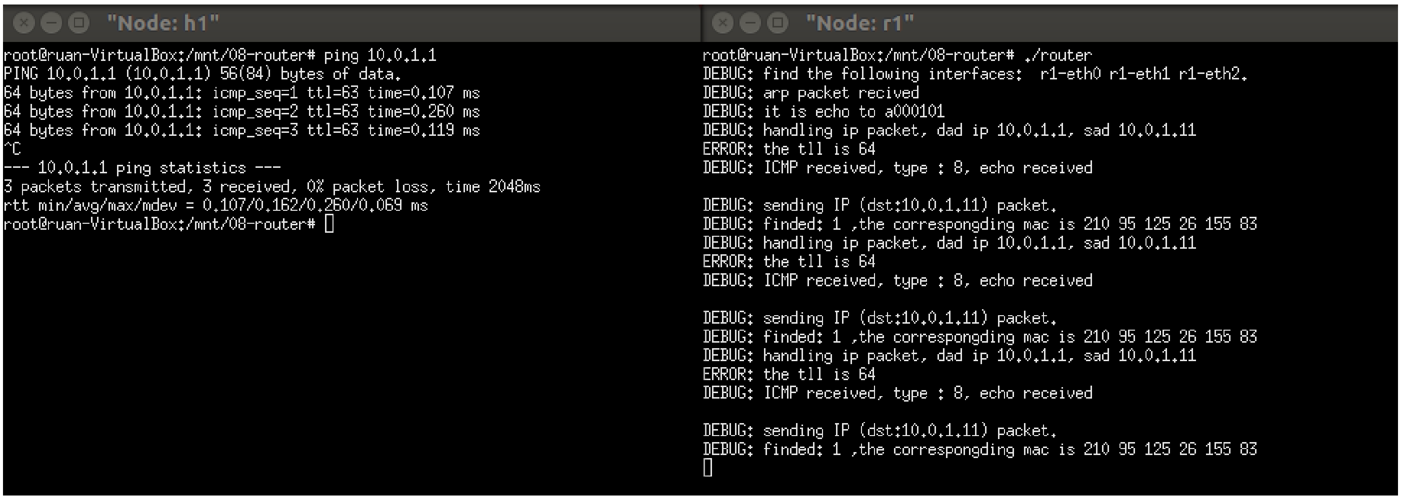
运行 router，开始完成对应实验条目，结果见下节。

四、 实验结果

运行脚本

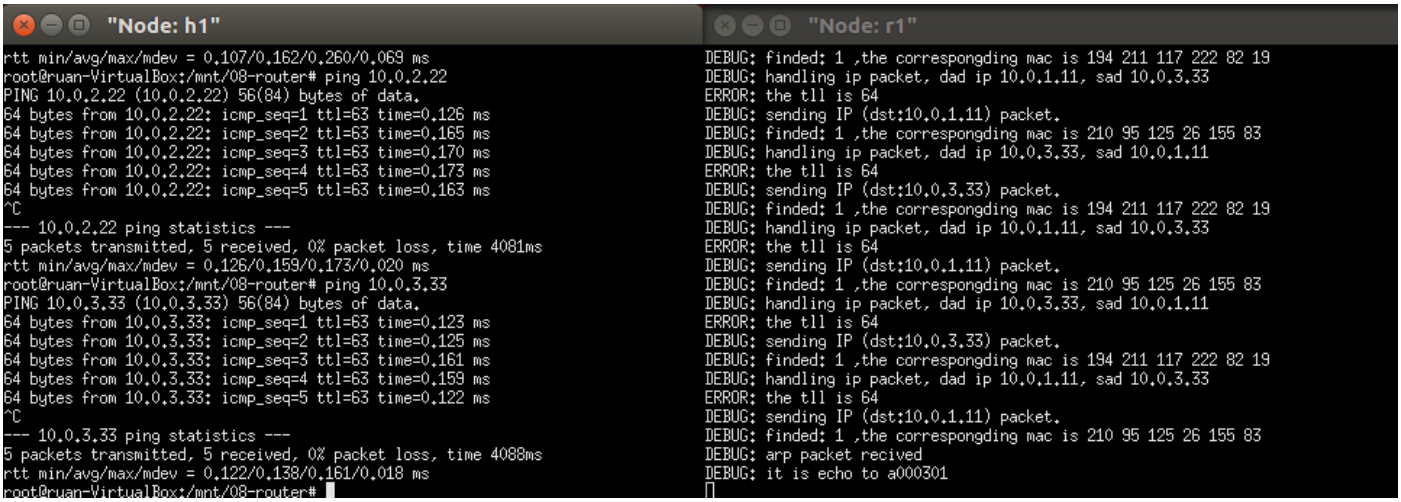
```
ruan@ruan-VirtualBox:/mnt/08-router$ sudo python router_topo.py
mininet> xterm h1 h2 h3 r1
```

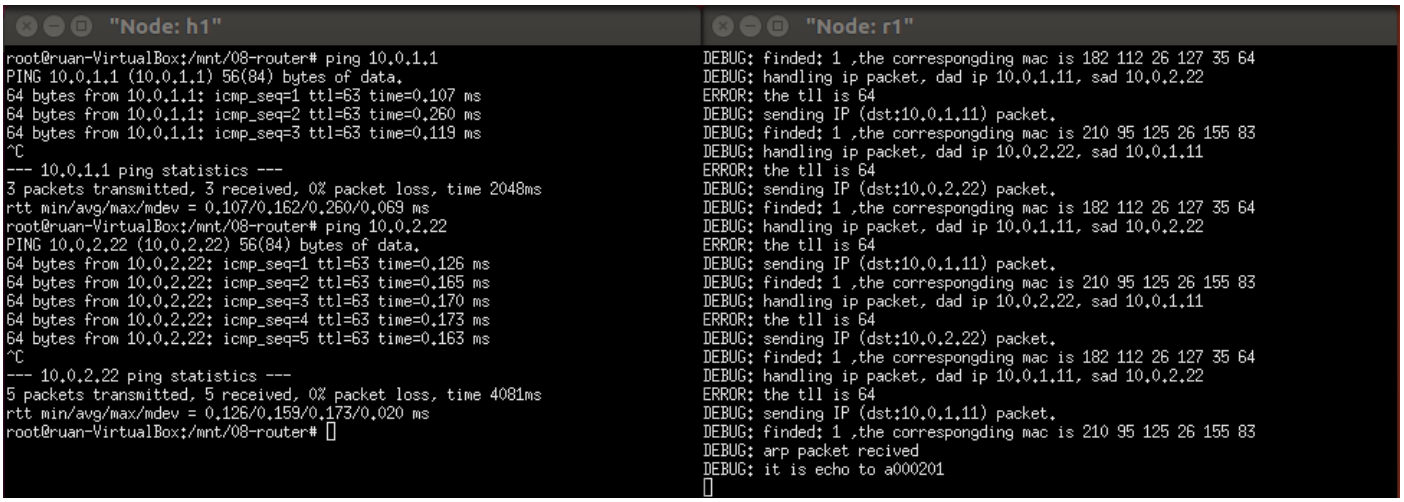
Ping r1



成功

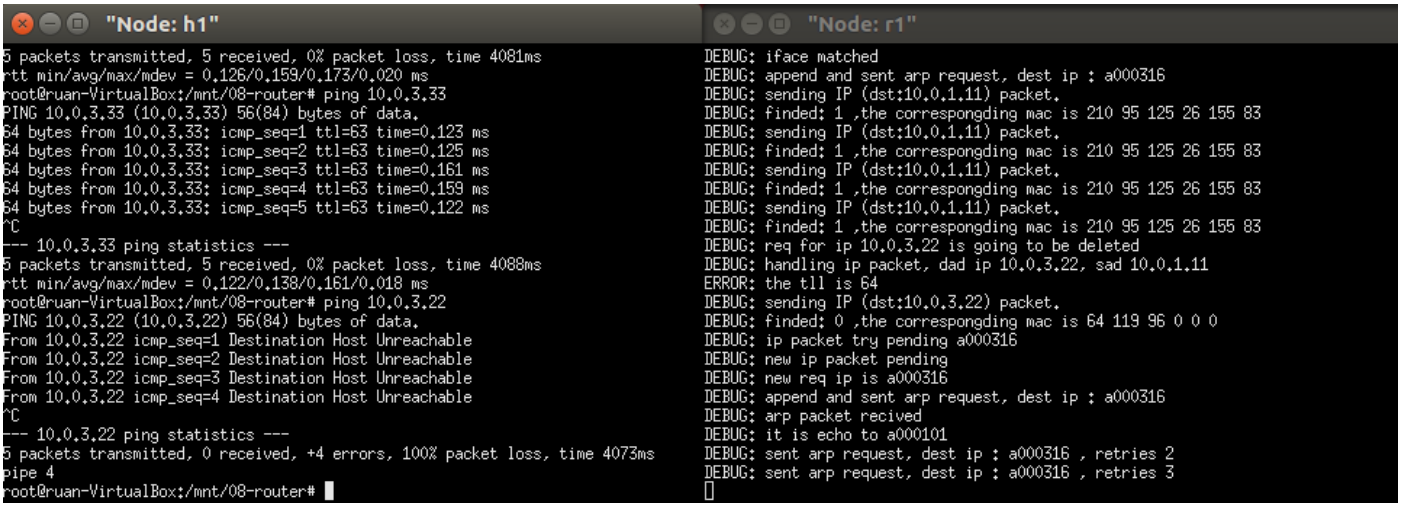
Ping h2 h3





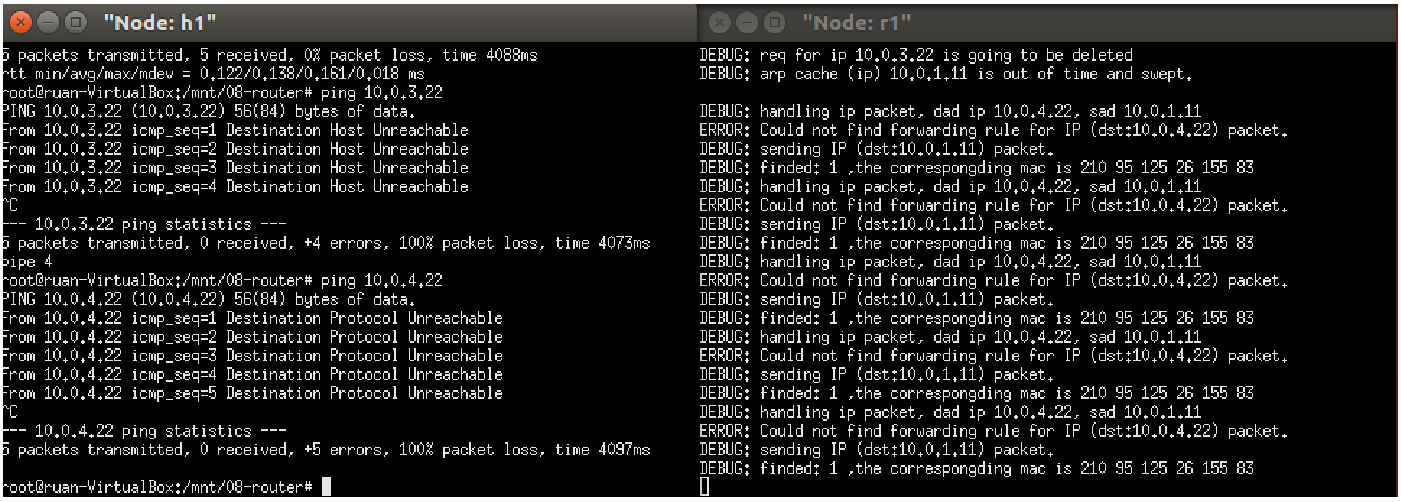
成功

Ping unreachable host



成功

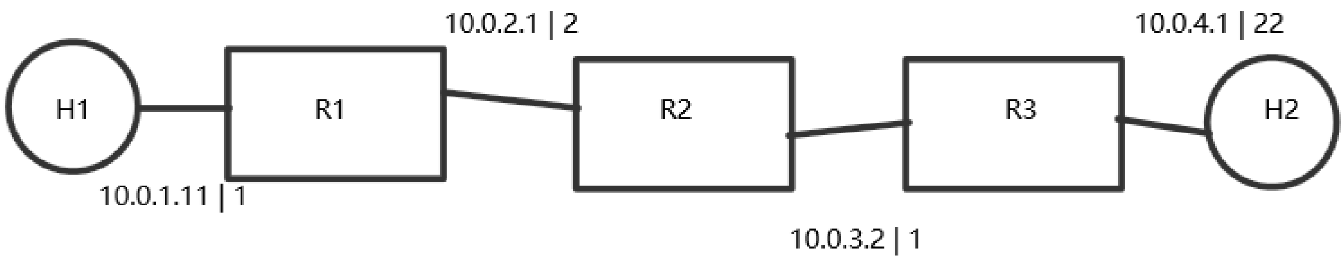
Ping unreachable net



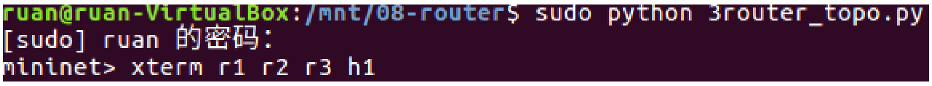
成功

运行新的拓扑脚本

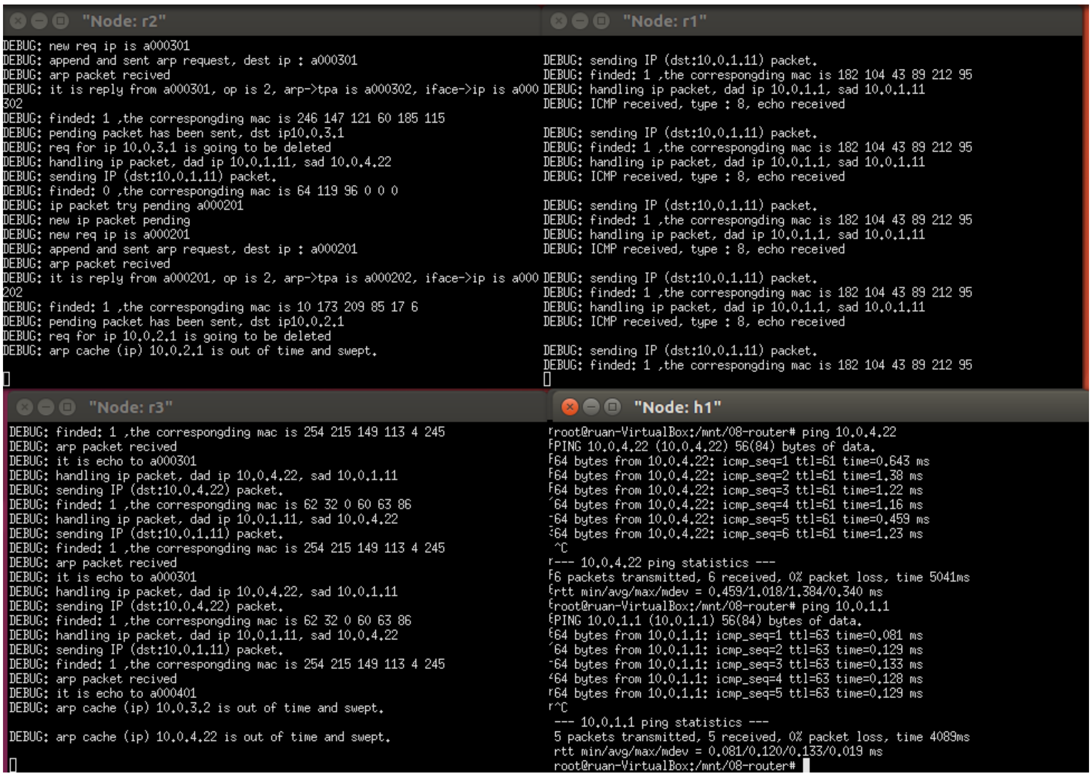
拓扑关系见下图



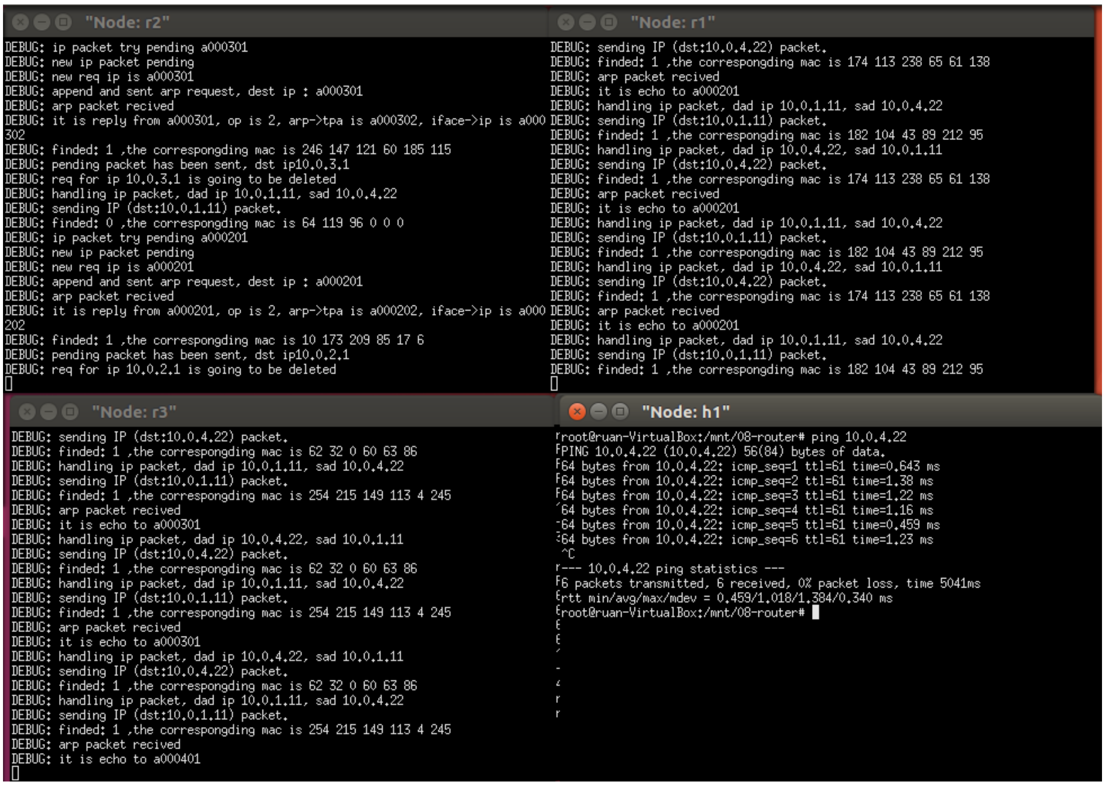
运行脚本



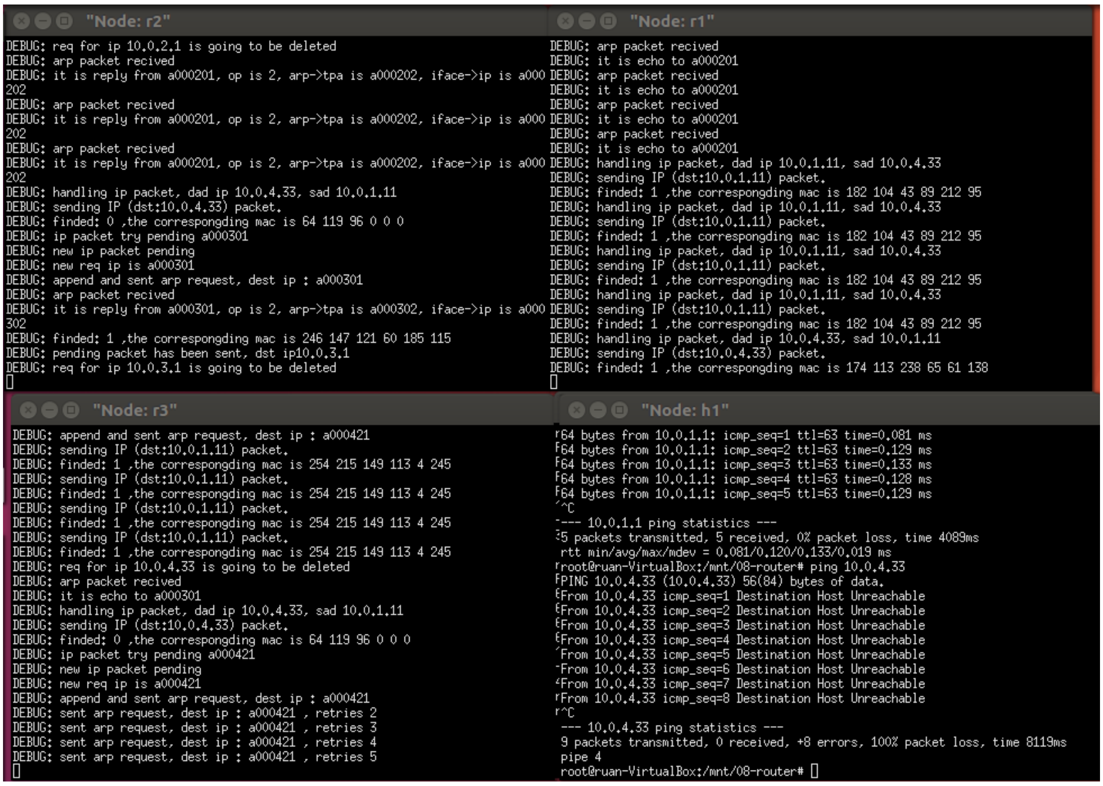
在各端口配置好转发表后，直接用 h1 ping r1，得如下结果

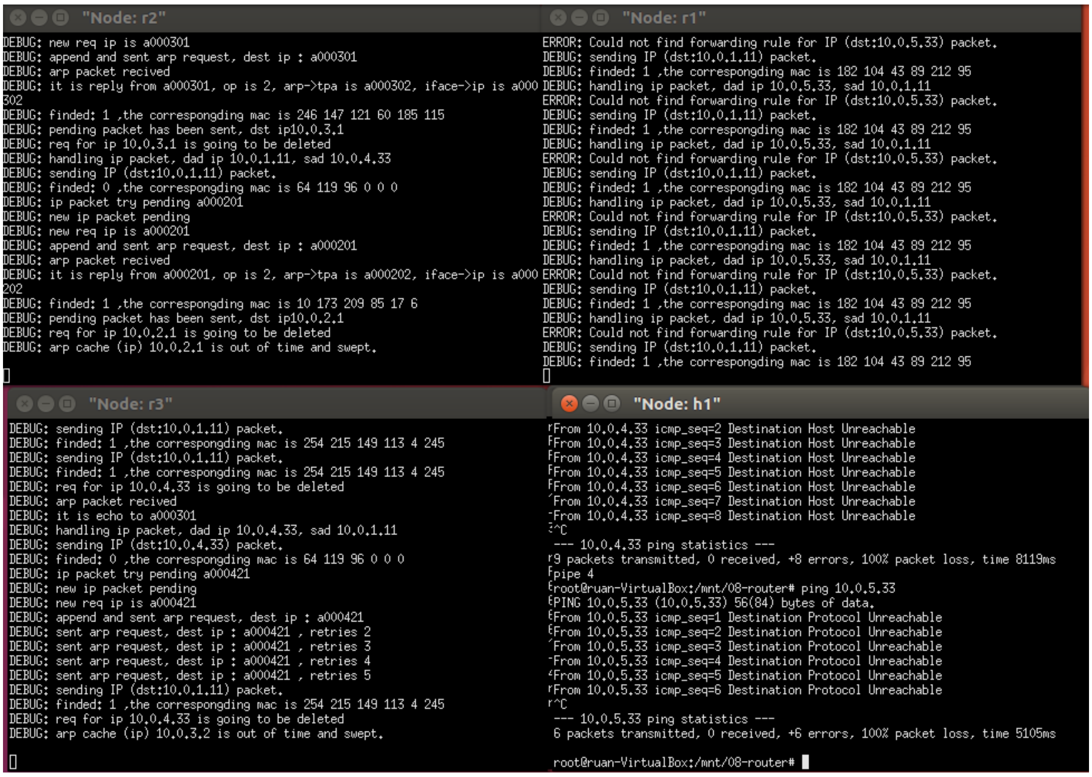


用 h1 ping h2 ，得如下结果，可以证明各个端口之间正常连通。

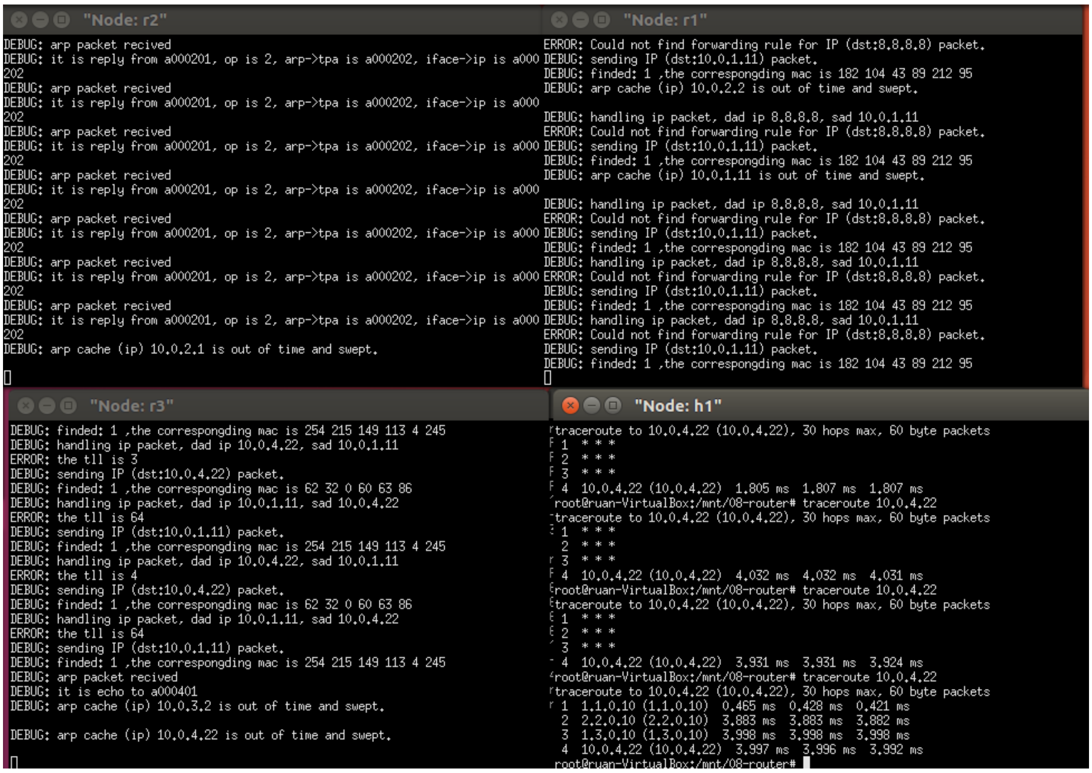


Ping unreachable host/net 也得到了对应的正确回应





运行 traceroute 程序，正确获得了路径



五、 实验总结

通过本次实现，我更加细致的了解了 IP 协议中各项内容的结构以及对应的作用，对路由的转发有了更加透彻的理解。当然，背后经理了很多错误观念的改正以及未知问题的尝试解决。这次实验大概花费了两个整天的时间，一天书写

一天调试，这样长度的调试时间其实也很能说明我之前对实验的架构掌握得不够好，对各个时候应该做什么并没有了然于心，导致走了不少弯路；另一个问题是网络序和本地序的转化，确实是太容易忘记了，而且在初期时，也不大确定各个函数输入的东西究竟是怎样的，导致了不少问题，另外这个确实是很容易出错，不管是类型还是是否需要转换，而且往往影响挺大，好在有 wireshark 的帮助，调试的压力小了不少。