路由器转发实验

学号：2016K8009908007
姓名：薛峰

一、实验内容

1、运行给定网络拓扑在 r1 上运行 router，进行数据包处理，然后在 h1 上进行 ping 实验，从而判断路由器是否能够正常工作。

2、自己构造一个包含多个路由器节点组成的网络

①手动配置每个路由器节点的路由表和终端结点的默认路由表；

②终端节点 ping 每个路由器节点的入端口 IP 地址，能够 ping 通；

③在一个终端节点上 traceroute 另一节点，能够正确输出路径上每个节点的 IP 信息。

二、实验流程

1、arp.c

① void arp_send_request(iface_info_t *iface, u32 dst_ip)函数

该函数作用是发送 arp 请求，首先分配一个 arp 包，然后对每个字段赋值，然后从 iface 端口发出。

```
void arp_send_request(iface_info_t *iface, u32 dst_ip)
{
  //fprintf(stderr, "TODO: send arp request when lookup failed in arpcache.\n");
  printf("arp_send_request iface_name = %s, dest_ip = %x\n", iface->name, dst_ip);

  int len_packet = sizeof(struct ether_arp) + ETHER_HDR_SIZE;
  char *new_pkt = (char *) malloc (len_packet);
  if ( !new_pkt ){
    printf ("Allocate failed in 'arp_send_request'.\n");
    exit(0);
  }

  struct ether_header *eth_h = (struct ether_header *)(new_pkt);
  struct ether_arp *eth_arp = (struct ether_arp *)(new_pkt + ETHER_HDR_SIZE);

  // set the Dest and Src Ether Addr
  memcpy(eth_h->ether_shost, iface->mac, ETH_ALEN);
  memset(eth_h->ether_dhost, 0xff, ETH_ALEN); // Broadcast

  // set ARP
  eth_h->ether_type = htons(ETH_P_ARP);
  eth_arp->arp_hrd  = htons(ARPHRD_ETHER);
  eth_arp->arp_pro  = htons(ETH_P_IP);
  eth_arp->arp_hln  = 6;
  eth_arp->arp_pln  = 4;
  eth_arp->arp_op   = htons(ARPOP_REQUEST);
  memcpy(eth_arp->arp_sha, iface->mac, ETH_ALEN);
  eth_arp->arp_spa = htonl(iface->ip);
  memset(eth_arp->arp_tha, 0, ETH_ALEN);
  eth_arp->arp_tpa = htonl(dst_ip);

  iface_send_packet(iface, new_pkt, len_packet);
}
```

② void arp_send_reply(iface_info_t *iface, struct ether_arp *req_hdr)函数

该函数的作用为发送 arp 应答，与 arp_send_request 类似，先分配一个 arp 包，然后对各字段赋值，然后从 iface 端口发出。

```c
void arp_send_reply(iface_info_t *iface, struct ether_arp *req_hdr)
{
  //fprintf(stderr, "TODO: send arp reply when receiving arp request.\n");

  int len_packet = sizeof(struct ether_arp) + ETHER_HDR_SIZE;
  char *new_pkt = (char *) malloc(len_packet);
  if ( !new_pkt ){
    printf ("Allocate failed in 'arp_send_reply'.\n");
    exit(0);
  }
  struct ether_header *eth_h = (struct ether_header *) (new_pkt);
  struct ether_arp *eth_arp = (struct ether_arp *)(new_pkt + ETHER_HDR_SIZE);

  // set the Dest and Src Ether Addr
  memcpy(eth_h->ether_dhost, req_hdr->arp_sha, ETH_ALEN);
  memcpy(eth_h->ether_shost, iface->mac, ETH_ALEN);

  // set ARP
  eth_h->ether_type = htons(ETH_P_ARP);
  eth_arp->arp_hrd  = htons(ARPHRD_ETHER);
  eth_arp->arp_pro  = htons(ETH_P_IP);
  eth_arp->arp_hln  = 6;
  eth_arp->arp_pln  = 4;
  eth_arp->arp_op   = htons(ARPOP_REPLY);
  memcpy(eth_arp->arp_sha, iface->mac, ETH_ALEN);
  eth_arp->arp_spa = htonl(iface->ip);
  memcpy(eth_arp->arp_tha, req_hdr->arp_sha, ETH_ALEN);
  eth_arp->arp_tpa = req_hdr->arp_spa;

  iface_send_packet(iface, new_pkt, len_packet);
}
```

③void handle_arp_packet(iface_info_t *iface, char *packet, int len)函数

当接收到 arp 包的时候会调用该函数。如果该 arp 包是请求包并且该请求的目的 ip 与接收该包的端口 ip 相同，说明 arp 包找到了目的，因此将源 ip 和 mac 地址插入到 arpcache 中；如果该包是 arp 应答，则将源 ip 和 mac 地址插入到 arpcache 当中。

```c
void handle_arp_packet(iface_info_t *iface, char *packet, int len)
{
  struct ether_arp *eth_arp = (struct ether_arp *)(packet + ETHER_HDR_SIZE);

  if (ntohs(eth_arp->arp_op) == ARPOP_REQUEST) {
    if (ntohl(eth_arp->arp_tpa) == iface->ip) {
      printf("eth_arp->arp_tpa = %x \n", iface->ip);
      arpcache_insert(ntohl(eth_arp->arp_spa), eth_arp->arp_sha);
      arp_send_reply(iface, eth_arp);
    }
  }
  else if (ntohs(eth_arp->arp_op) == ARPOP_REPLY) {
      arpcache_insert(ntohl(eth_arp->arp_spa), eth_arp->arp_sha);
  }
  else
    printf ("Unknown ARP_OP.\n");
}
```

2、arpcache.c

①int arpcache_lookup(u32 ip4, u8 mac[ETH_ALEN])函数

```c
int arpcache_lookup(u32 ip4, u8 mac[ETH_ALEN])
{
  int i = 0;
  pthread_mutex_lock(&arpcache.lock);
  for (i = 0; i < MAX_ARP_SIZE; i++ ) {
    if (arpcache.entries[i].valid && arpcache.entries[i].ip4 == ip4) {
      memcpy(mac, arpcache.entries[i].mac, ETH_ALEN);
      pthread_mutex_unlock(&arpcache.lock);
      return 1;
    }
  }
  pthread_mutex_unlock(&arpcache.lock);
  return 0;
}
```

该函数的作用是查找 arpcache 中有没有 ip4 对应的映射关系，如果有则将其 mac 地址 copy 给参数 u8 mac[ETH_ALEN]。

②void arpcache_append_packet(iface_info_t *iface, u32 ip4, char *packet, int len)函数

该函数的作用是将 packet 插入到 wait list 当中，首先检查目的 ip 是否已经在 req_list 中，如果有则说明只需要将该 packet 插入到该等待队列即刻，否则需要创建新的 req 表项，再将该 packet 插入到该 req 表项下。最后需要发送 req 请求。

```
void arpcache_append_packet(iface_info_t *iface, u32 ip4, char *packet, int len)
{
  pthread_mutex_lock(&arpcache.lock);
  struct cached_pkt *new_pkt = (struct cached_pkt *)malloc(sizeof(struct cached_pkt));
  if ( !new_pkt ){
    printf ("Allocate failed in 'arpcache_append_packet'.\n");
    exit(0);
  }
  init_list_head(&new_pkt->list);
  new_pkt->len = len;
  new_pkt->packet = packet;

  struct arp_req *p, *q;
  list_for_each_entry_safe(p, q, &arpcache.req_list, list){
    if(p->iface == iface && p->ip4 == ip4){
      list_add_tail(&new_pkt->list, &(p->cached_packets));
      pthread_mutex_unlock(&arpcache.lock);
      // There is already an entry with the same IP address and iface
      return;
    }
  }
  // NOT FIND
  struct arp_req *new_req = (struct arp_req *)malloc(sizeof(struct arp_req));
  if (!new_req){
    printf("Allocate memory(new_req) falied in 'arpcache_append_packet'.\n");
    exit(0);
  }
  init_list_head(&new_req->list);
  init_list_head(&new_req->cached_packets);
  new_req->iface = iface;
  new_req->ip4 = ip4;
  new_req->sent = time(NULL);
  new_req->retries = 0;
  list_add_head(&new_pkt->list, &new_req->cached_packets); // Add the pkt to cached_packets
  list_add_tail(&new_req->list, &arpcache.req_list);       // Add the req to req_list
  pthread_mutex_unlock(&arpcache.lock);
  arp_send_request(iface, ip4);
  return;
}
```

③void arpcache_insert(u32 ip4, u8 mac[ETH_ALEN])函数

该函数的作用是将该 Ip 和 mac 的映射关系插入到 arpcache 当中。首先找到第一个无效的表项，如果都有效，则替换最后一个表项。然后对表项的各部分进行赋值。然后遍历 req_list，如果找到该 ip 的请求，则应该将该请求下的 packet 都发送出去，并删除该 req 表项。需要注意的是在调用 iface_send_packet_by_arp 函数之前应该释放锁，因为 iface_send_packet_by_arp 函数中也会申请锁，会造成该进程阻塞。调用之后需要重新申请该锁。（因为在调用 iface_send_packet_by_arp 函数之前没有释放锁这个 bug 调了很长时间...）

```c
void arpcache_insert(u32 ip4, u8 mac[ETH_ALEN])
{
  pthread_mutex_lock(&arpcache.lock);

  int i = 0;
  // Find an invalid entry. If there's not any invalid entry, i = MAX_ARP_SIZE.
  for(i = 0; i < MAX_ARP_SIZE - 1; i++) {
    if(arpcache.entries[i].valid == 0)
      break;
  }

  arpcache.entries[i].ip4 = ip4;
  memcpy(arpcache.entries[i].mac, mac, ETH_ALEN);
  arpcache.entries[i].added = time(NULL);
  arpcache.entries[i].valid = 1;

  struct arp_req *req_p, *req_q;  // Find the corresponding req_list
  list_for_each_entry_safe(req_p, req_q, &arpcache.req_list, list) {
    if (req_p->ip4 == ip4) {
      struct cached_pkt *pkt_p, *pkt_q;
      list_for_each_entry_safe(pkt_p, pkt_q, &req_p->cached_packets, list){
        pthread_mutex_unlock(&arpcache.lock); // Remember
        iface_send_packet_by_arp(req_p->iface, ip4, pkt_p->packet, pkt_p->len);
        pthread_mutex_lock(&arpcache.lock);
        list_delete_entry(&pkt_p->list);
        free(pkt_p);
      }
      list_delete_entry(&req_p->list);
      free(req_p);
    }
  }
  pthread_mutex_unlock(&arpcache.lock);
}
```

④**void *arpcache_sweep(void *arg) 函数**

```c
void *arpcache_sweep(void *arg)
{
  int i;
  struct arp_req *req_p, *req_q;
  struct cached_pkt *pkt_p, *pkt_q;

  while (1) {
    sleep(1);
    pthread_mutex_lock(&arpcache.lock);
    // check the IP->mac entry
    for(i = 0; i < MAX_ARP_SIZE; i++) {
      if( time(NULL) - arpcache.entries[i].added > ARP_ENTRY_TIMEOUT)
        arpcache.entries[i].valid = 0;
    }
    // check the packet
    list_for_each_entry_safe(req_p, req_q, &arpcache.req_list, list) {
      if(req_p->retries > ARP_REQUEST_MAX_RETRIES) {
        list_for_each_entry_safe(pkt_p, pkt_q, &req_p->cached_packets, list){
          pthread_mutex_unlock(&arpcache.lock); // Remember!
          icmp_send_packet(pkt_p->packet, pkt_p->len, ICMP_DEST_UNREACH, ICMP_HOST_UNREACH);
          pthread_mutex_lock(&arpcache.lock);
          free(pkt_p->packet);
          list_delete_entry(&pkt_p->list);
          free(pkt_p);
        }
        list_delete_entry(&req_p->list);
        free(req_p);
      }
      else {
        arp_send_request(req_p->iface, req_p->ip4);
        req_p->sent = time(NULL);
        req_p->retries ++;
      }
    }
    pthread_mutex_unlock(&arpcache.lock);
  }
  return NULL;
}
```

该函数的作用是定期扫描 arpcache，如果发现无效的表项则会清除。首先如果某一 ip->mac 映射生存时间超过 15s，则应清除该映射。并且没过一秒，应该对 req_list 请求下的表项发送一个 arp 请求，如果发送超过 5 此请求，那么清除该 req 表项。

3、iacmp.c

①void icmp_send_packet(const char *in_pkt, int len, u8 type, u8 code)函数

该函数的作用是发送 ICMP 数据包。在四种情况下需要发送 ICMP 数据包：路由表查找失败、ARP 查询失败、TTL 值减为 0、收到 Ping 本端口的数据包。其中前三种情况对应的 ICMP 数据包除了 type 和 code 不同之外，剩余部分相同。因此，第四种情况要单独讨论。如果 type 为 8，说明收到 ping 本端口的包，此时，ICMP 数据包的 type = 8, code = 0，剩余部分为 ping 包中的相应字段。对于前三种情况，剩余部分则为收到数据包的头部和随后的 8 字节。

```
void icmp_send_packet(const char *in_pkt, int len, u8 type, u8 code)
{
  char *packet;
  int packet_len;
  struct iphdr *ip = packet_to_ip_hdr(in_pkt);
  struct ether_header *in_eth_h = (struct ether_header*)(in_pkt);

  if (type == ICMP_ECHOREPLY){
    packet_len = len;
  }
  else {
    packet_len = ETHER_HDR_SIZE + IP_BASE_HDR_SIZE + ICMP_HDR_SIZE + IP_HDR_SIZE(ip) + 8;
  }

  packet = (char *)malloc(packet_len);
  if ( !packet ){
    printf ("Allocate failed in 'icmp_send_packet'.\n");
    exit(0);
  }
  struct ether_header *eth_h = (struct ether_header *)(packet);
  memcpy(eth_h->ether_dhost, in_eth_h->ether_dhost, ETH_ALEN);
  memcpy(eth_h->ether_shost, in_eth_h->ether_dhost, ETH_ALEN);
  eth_h->ether_type = htons(ETH_P_IP);

  struct iphdr *iph = (struct iphdr *)(packet + ETHER_HDR_SIZE);
  rt_entry_t *entry = longest_prefix_match(ntohl(ip->saddr));
  ip_init_hdr(iph, entry->iface->ip, ntohl(ip->saddr), packet_len-ETHER_HDR_SIZE, 1);
```

```
  struct icmphdr *icmp = (struct icmphdr *)(packet + ETHER_HDR_SIZE + IP_BASE_HDR_SIZE);
  if (type == ICMP_ECHOREPLY) {
    icmp->type = 0;
    icmp->code = 0;
    char *packet_rest = packet + ETHER_HDR_SIZE + IP_BASE_HDR_SIZE + ICMP_HDR_SIZE - 4;
    char *in_pkt_rest = (char *)(in_pkt + ETHER_HDR_SIZE + IP_HDR_SIZE(ip) + ICMP_HDR_SIZE - 4);
    int data_size = len - ETHER_HDR_SIZE - IP_HDR_SIZE(ip) - ICMP_HDR_SIZE + 4;
    memcpy(packet_rest, in_pkt_rest, data_size);
    icmp->checksum = icmp_checksum(icmp, data_size + ICMP_HDR_SIZE - 4);
  }
  else {
    printf("icmp_send_packet is NOT ICMP_ECHOREPLY\n");
    icmp->type = type;
    icmp->code = code;
    char *packet_rest = packet + ETHER_HDR_SIZE + IP_BASE_HDR_SIZE + ICMP_HDR_SIZE;
    memset(packet_rest - 4, 0, 4);
    int data_size = IP_HDR_SIZE(ip) + 8;
    memcpy(packet_rest, ip, data_size);
    icmp->checksum = icmp_checksum(icmp, data_size + ICMP_HDR_SIZE);
  }

  ip_send_packet(packet, packet_len);
}
```

4、ip.c

①void handle_ip_packet(iface_info_t *iface, char *packet, int len)函数

该函数的作用是处理 ip 包，在收到 ip 包之后会调用该函数。首先如果该包是 ICMP 请求并且目的地址与该收包端口一致则需要

返回 icmp 应答(ping 便是通过该机制实现)；否则向下传递该包调用 forward_ip_packet 函数。void forward_ip_packet(u32 ip_dst, char *packet, int len)函数的作用是转发数据包，首先将 ttl 减一，若 ttl<=0 则说明该包需要被清除，回复 icmp 包。然后，重新计算校验和，通过最长前缀匹配查找转出端口，并将数据包转发出去，如果查找失败，则说明网络不可达，回复 ICMP Destination Net Unreachable。

```
void handle_ip_packet(iface_info_t *iface, char *packet, int len)
{
  struct iphdr *ip_hdr = packet_to_ip_hdr(packet);
  u32 daddr = ntohl(ip_hdr->daddr);
  struct icmphdr *icmp_hdr = (struct icmphdr*)((char*)ip_hdr + IP_HDR_SIZE(ip_hdr));

  if(daddr == iface->ip && icmp_hdr->type == ICMP_ECHOREQUEST) {
    icmp_send_packet(packet, len, ICMP_ECHOREPLY, 0);
    free(packet);
    return;
  }
  else {
    forward_ip_packet(daddr, packet, len);
    return;
  }
}
```

```
void forward_ip_packet(u32 ip_dst, char *packet, int len)
{
  struct iphdr *ip_hdr = packet_to_ip_hdr(packet);

  ip_hdr->ttl--;  //ttl-1
  if(ip_hdr->ttl <= 0) {
    icmp_send_packet(packet, len, ICMP_TIME_EXCEEDED, ICMP_EXC_TTL);
    free(packet);
    return;
  }
  ip_hdr->checksum = ip_checksum(ip_hdr); // reset the checksum

  rt_entry_t *entry = longest_prefix_match(ip_dst);
  if(entry != NULL) {
    printf("entry != NULL!, ip_dst = %x, entry->if_name = %s\n",ip_dst, entry->if_name);
    ip_send_packet(packet, len);
  }
  else {
    printf("entry = NULL!\n");
    icmp_send_packet(packet, len, ICMP_DEST_UNREACH, ICMP_NET_UNREACH);
    free(packet);
  }
}
```

5、ip_base.c

①rt_entry_t *longest_prefix_match(u32 dst)函数

该函数的作用是用最长前缀匹配方法查找 dst 对应的路由表项。

```
rt_entry_t *longest_prefix_match(u32 dst)
{
  rt_entry_t *pos, *maxpos = NULL;
  u32 maxmask = 0;
  list_for_each_entry(pos, &rtable, list){
    u32 ip = dst & pos->mask;
    u32 pos_ip = pos->dest & pos->mask;
    if ( pos_ip == ip && pos->mask > maxmask ) {
      maxpos = pos;
      maxmask = pos->mask;
    }
  }
  return maxpos;
}
```

②void ip_send_packet(char *packet, int len)

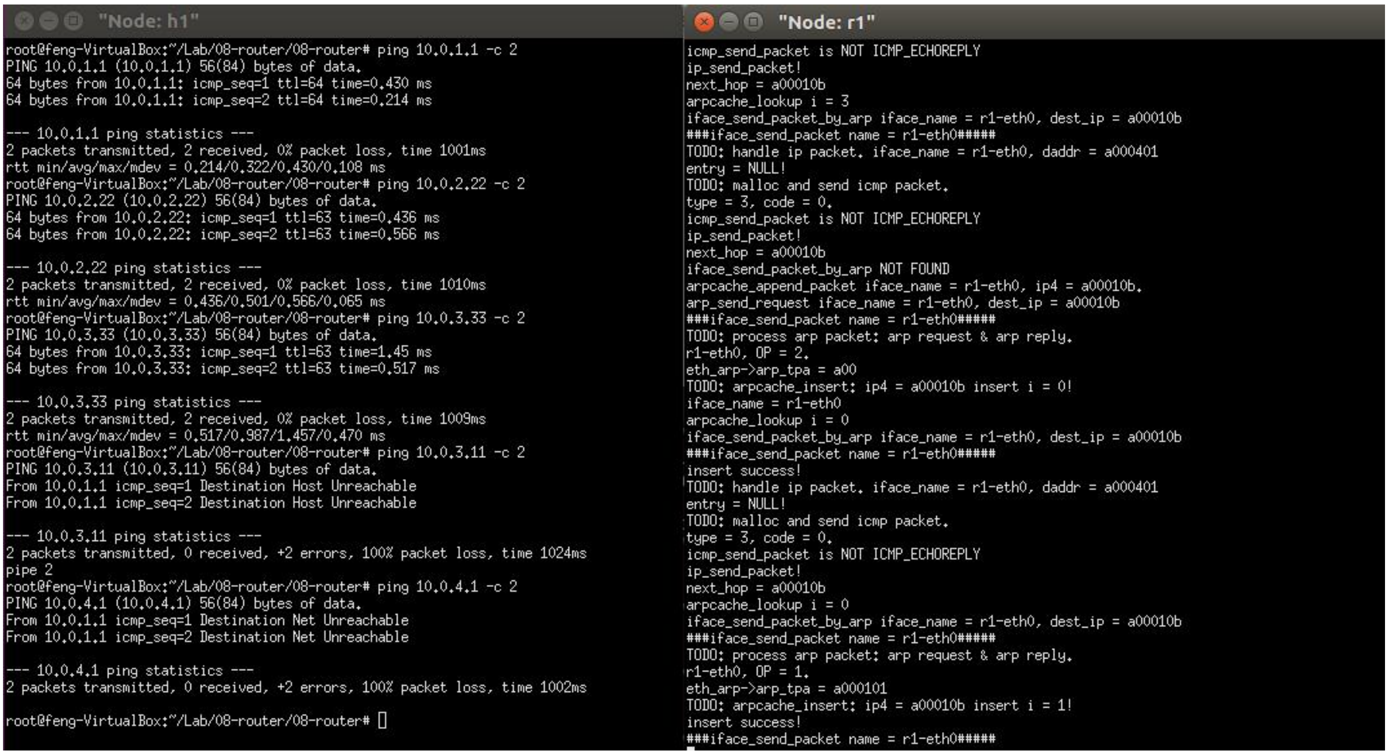
该函数的作用是发送 ip 包。首先查找路由表查找目的 ip 对应的表项。对于下一跳地址，如果路由器端口与目的地址不在同一网段则 next_hop = entry->gw，如果路由器端口与目的地址在同一网段，则 next_hop = dst_ip。

```c
void ip_send_packet(char *packet, int len)
{
  struct iphdr *ip = packet_to_ip_hdr(packet);
  u32 daddr = ntohl(ip->daddr);
  rt_entry_t *entry = longest_prefix_match(daddr);
  if (entry == NULL) {
    printf("No corresponding ip in rtable");
    return ;
  }

  u32 next_hop = entry->gw;
  if (!next_hop)
    next_hop = daddr;

  iface_send_packet_by_arp(entry->iface, next_hop, packet, len);
}
```

三、实验结果及分析

1、实验结果

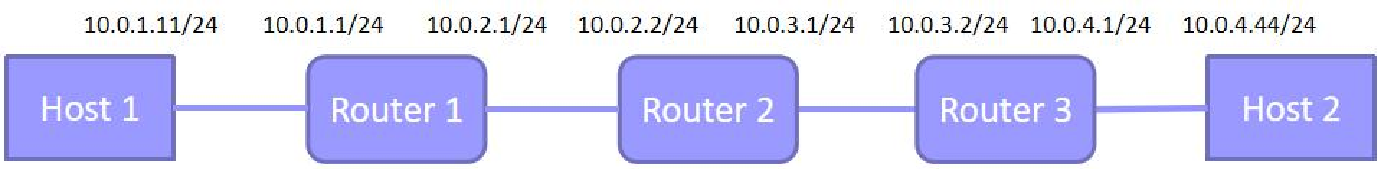
对于给定拓扑

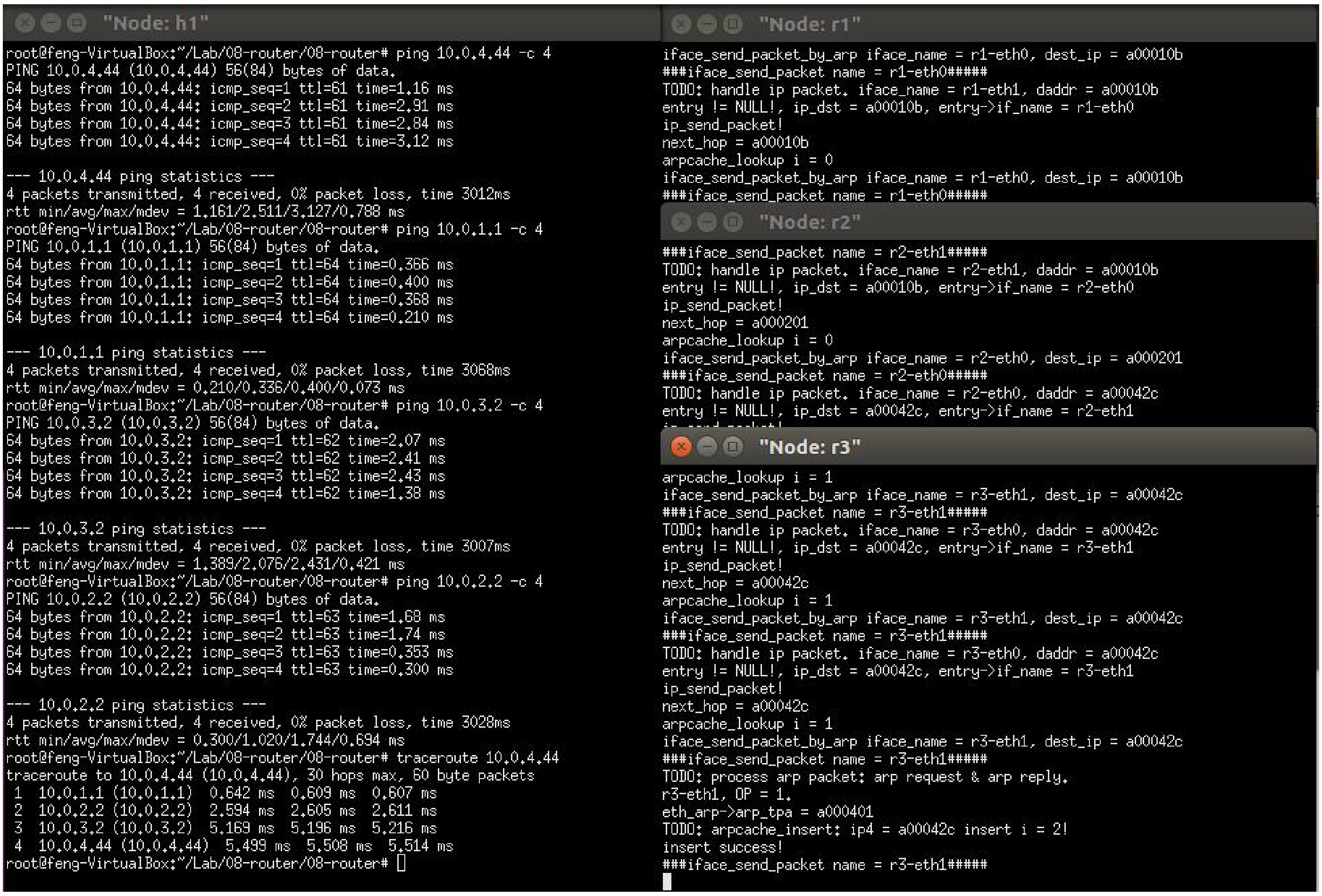


```
"Node: h1"
root@feng-VirtualBox:~/Lab/08-router/08-router# ping 10.0.1.1 -c 2
PING 10.0.1.1 (10.0.1.1) 56(84) bytes of data.
64 bytes from 10.0.1.1: icmp_seq=1 ttl=64 time=0.430 ms
64 bytes from 10.0.1.1: icmp_seq=2 ttl=64 time=0.214 ms

--- 10.0.1.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 0.214/0.322/0.430/0.108 ms
root@feng-VirtualBox:~/Lab/08-router/08-router# ping 10.0.2.22 -c 2
PING 10.0.2.22 (10.0.2.22) 56(84) bytes of data.
64 bytes from 10.0.2.22: icmp_seq=1 ttl=63 time=0.436 ms
64 bytes from 10.0.2.22: icmp_seq=2 ttl=63 time=0.566 ms

--- 10.0.2.22 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1010ms
rtt min/avg/max/mdev = 0.436/0.501/0.566/0.065 ms
root@feng-VirtualBox:~/Lab/08-router/08-router# ping 10.0.3.33 -c 2
PING 10.0.3.33 (10.0.3.33) 56(84) bytes of data.
64 bytes from 10.0.3.33: icmp_seq=1 ttl=63 time=1.45 ms
64 bytes from 10.0.3.33: icmp_seq=2 ttl=63 time=0.517 ms

--- 10.0.3.33 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1009ms
rtt min/avg/max/mdev = 0.517/0.987/1.457/0.470 ms
root@feng-VirtualBox:~/Lab/08-router/08-router# ping 10.0.3.11 -c 2
PING 10.0.3.11 (10.0.3.11) 56(84) bytes of data.
From 10.0.1.1 icmp_seq=1 Destination Host Unreachable
From 10.0.1.1 icmp_seq=2 Destination Host Unreachable

--- 10.0.3.11 ping statistics ---
2 packets transmitted, 0 received, +2 errors, 100% packet loss, time 1024ms
pipe 2
root@feng-VirtualBox:~/Lab/08-router/08-router# ping 10.0.4.1 -c 2
PING 10.0.4.1 (10.0.4.1) 56(84) bytes of data.
From 10.0.1.1 icmp_seq=1 Destination Net Unreachable
From 10.0.1.1 icmp_seq=2 Destination Net Unreachable

--- 10.0.4.1 ping statistics ---
2 packets transmitted, 0 received, +2 errors, 100% packet loss, time 1002ms

root@feng-VirtualBox:~/Lab/08-router/08-router#
```

```
"Node: r1"
icmp_send_packet is NOT ICMP_ECHOREPLY
ip_send_packet!
next_hop = a00010b
arpcache_lookup i = 3
iface_send_packet_by_arp iface_name = r1-eth0, dest_ip = a00010b
###iface_send_packet name = r1-eth0#####
TODO: handle ip packet. iface_name = r1-eth0, daddr = a000401
entry = NULL!
TODO: malloc and send icmp packet.
type = 3, code = 0.
icmp_send_packet is NOT ICMP_ECHOREPLY
ip_send_packet!
next_hop = a00010b
iface_send_packet_by_arp NOT FOUND
arpcache_append_packet iface_name = r1-eth0, ip4 = a00010b.
arp_send_request iface_name = r1-eth0, dest_ip = a00010b
###iface_send_packet name = r1-eth0#####
TODO: process arp packet: arp request & arp reply.
r1-eth0, OP = 2.
eth_arp->arp_tpa = a00
TODO: arpcache_insert: ip4 = a00010b insert i = 0!
iface_name = r1-eth0
arpcache_lookup i = 0
iface_send_packet_by_arp iface_name = r1-eth0, dest_ip = a00010b
###iface_send_packet name = r1-eth0#####
insert success!
TODO: handle ip packet. iface_name = r1-eth0, daddr = a000401
entry = NULL!
TODO: malloc and send icmp packet.
type = 3, code = 0.
icmp_send_packet is NOT ICMP_ECHOREPLY
ip_send_packet!
next_hop = a00010b
arpcache_lookup i = 0
iface_send_packet_by_arp iface_name = r1-eth0, dest_ip = a00010b
###iface_send_packet name = r1-eth0#####
TODO: process arp packet: arp request & arp reply.
r1-eth0, OP = 1.
eth_arp->arp_tpa = a000101
TODO: arpcache_insert: ip4 = a00010b insert i = 1!
insert success!
###iface_send_packet name = r1-eth0#####
```

对于自己设计的拓扑





2、结果分析

实验一：

h1 分别 ping 10.0.1.1 10.0.2.22 10.0.3.33 均成功，说明路由器成功地连接了各 host；h1 ping 10.3.11，回复 Host Unreachable，说明路由器有此网段的路由表项，但是发出 arp 请求后一直无法收到 arp 回应，目的主机不可达。 ping 10.0.4.1 回复 Net Unreachable，说明路由器没有此网段的路由表项，目的网段不可达。

实验二：

h1 ping 各个 路由器节点的入端口 IP 地址，能够 ping 通，并且 ping h2 也能 ping 通。

traceroute h2 正确地输出了路径上每个节点的 IP 信息。

说明本次实验目的已经实现。