实验 5 · 交换机转发实验

吴嘉皓
2015K8009915007

一、实验内容

（一）实验目标

1. 实现对数据结构 mac_port_map 的所有操作，以及数据包的转发和广播操作：（以下操作用 Linux 多线程和互斥操作实现）
   - `iface_info_t *lookup_port(u8 mac[ETH_ALEN]);`
   - `void insert_mac_port(u8 mac[ETH_ALEN], iface_info_t *iface);`
   - `int sweep_aged_mac_port_entry();`
   - `void broadcast_packet(iface_info_t *iface, const char *packet, int len);`
   - `void handle_packet(iface_info_t *iface, char *packet, int len);`
2. 使用 iperf 和给定的拓扑进行实验，对比交换机转发与集线器广播的性能；

（二）交换机学习转发表

关键原理：当交换机从某端口收到源 MAC 地址（Ethernet 地址）为 X 的数据包时，可以确定：将目的 MAC 地址为 X 的数据包从该端口转出可以达到目的主机。

1. 查询操作：每收到一个数据包，根据目的 MAC 地址查询相应转发条目：如果查询到对应条目，则根据相应转发端口转发数据包，并更新访问时间；否则，广播该数据包；
2. 插入操作：每收到一个数据包，如果其源 MAC 地址不在转发表中，则将该地址与入端口的映射关系写入转发表；
3. 老化操作：每秒钟运行一次老化操作，删除超过 30 秒未访问的转发条目；

二、实验流程

（一）代码目录

（见下一页）

```
2015K8009915007_吴嘉皓_Lab5.tar.gz
├── 05-switching
│   ├── Makefile
│   ├── disable_offloading.sh
│   ├── example
│   │   ├── pthread_example.c
│   │   └── switch-reference
│   ├── hash.c
│   ├── include
│   │   ├── base.h
│   │   ├── ether.h
│   │   ├── hash.h
│   │   ├── headers.h
│   │   ├── list.h
│   │   ├── log.h
│   │   ├── mac.h
│   │   ├── packet.h
│   │   ├── types.h
│   │   └── utils.h
│   ├── mac.c
│   ├── main.c
│   ├── packet.c
│   ├── tags
│   └── three_nodes_bw.py
└── 实验 5-交换机实验-实验报告.pdf
```

（二）实验流程

按序在 05-switching 目录下输入如下命令：

```
make
sudo python three_nodes_bw.py
mininet> xterm h1 h2 h3 s1
```

在 xterm 的 s1 终端中运行 ./switch

然后分别以 h1，h2，h3 为服务端，其余两个为客户端，使用 iperf 测试性能。

三、实验结果

（一）交换机转发的结果

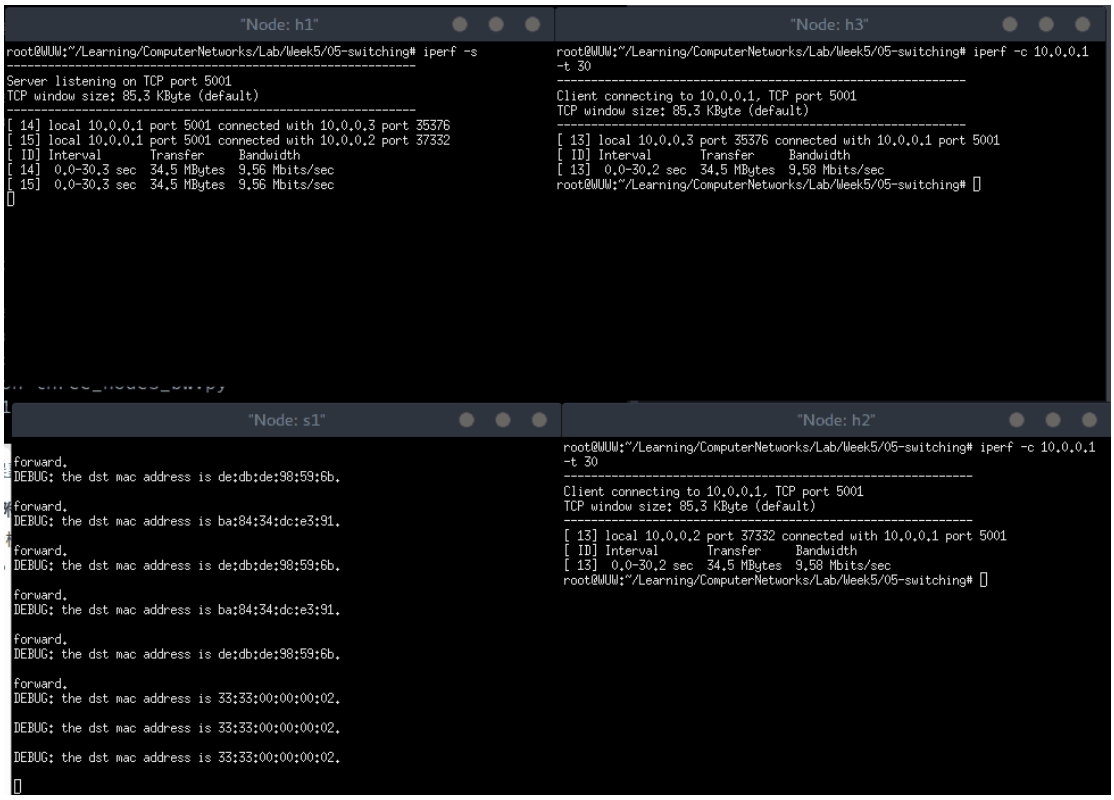

图 1 以 h1 为服务端，h2、h3 为客户端的交换机转发性能

（二）广播转发的结果

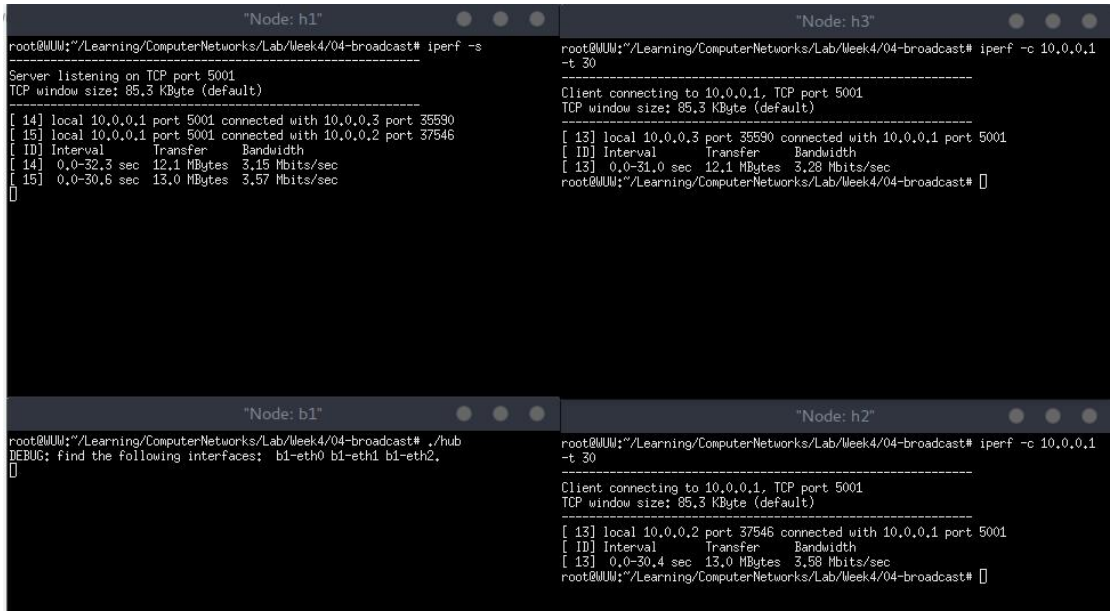

图 2 以 h1 为服务端，h2、h3 为客户端的广播转发性能

四、结果分析

（一）交换机转发与广播转发对比

由上述三(实验结果)可知,广播转发的带宽只有交换机转发的带宽的一半左右;

可知交换机转发的效率高于广播转发。

（二）代码实现分析

- **iface_info_t *lookup_port(u8 mac[ETH_ALEN])**

```
iface_info_t *lookup_port(u8 mac[ETH_ALEN])
{
    // TODO: implement the lookup process here
    // fprintf(stdout, "TODO: implement the lookup process here.\n");
    pthread_mutex_lock(&mac_port_map.lock);
    u8 hash_value = hash8((unsigned char *)mac, sizeof(u8)*ETH_ALEN);
    mac_port_entry_t * entry = mac_port_map.hash_table[hash_value];
    while(entry){
        if(!memcmp(entry->mac, mac, sizeof(u8)*ETH_ALEN)){
            // fprintf(stdout, "Port comparing succeeded.\n");
            pthread_mutex_unlock(&mac_port_map.lock);
            return entry->iface;
        }
        entry = entry->next;
    }
    // fprintf(stdout, "Port comparing failed.\n");
    pthread_mutex_unlock(&mac_port_map.lock);
    return NULL;
}
```

查询交换转发表的时候，需要用到互斥锁的操作，防止该操作与老化操作冲突，以免查询到的端口转发表和实际不一致。

先给待查询的 mac 地址计算哈希值，然后在哈希表对应的 hash cell 中查找与该 mac 地址匹配的 iface;

如果查找到，则释放锁，并返回指向该 iface 的指针;

否则，释放锁，并返回空指针。

➢ **void insert_mac_port(u8 mac[ETH_ALEN], iface_info_t *iface);**

```c
void insert_mac_port(u8 mac[ETH_ALEN], iface_info_t *iface)
{
    // TODO: implement the insertion process here
    // fprintf(stdout, "TODO: implement the insertion process here.\n");
    pthread_mutex_lock(&mac_port_map.lock);

    // create a new mac_port_entry && initialize its info
    mac_port_entry_t * new_entry = (mac_port_entry_t *)malloc(sizeof(mac_port_entry_t));
    bzero(new_entry, sizeof(mac_port_entry_t));
    new_entry->next = NULL;
    new_entry->iface = iface;
    new_entry->visited = time(NULL);
    memcpy(new_entry->mac, mac, sizeof(u8)*ETH_ALEN);
    // caculate the hash value of MAC && insert it into hash table
    u8 hash_value = hash8((unsigned char *)mac, sizeof(u8)*ETH_ALEN);
    mac_port_entry_t ** entry = &(mac_port_map.hash_table[hash_value]);
    if(!(*entry)) *entry = new_entry; // if the hash cell is empty
    else{
        while((*entry)->next) entry = &((*entry)->next); // find the tail of the hash cell
        (*entry)->next = new_entry; // add the entry to the hash cell
    }
    pthread_mutex_unlock(&mac_port_map.lock);
}
```

插入操作开始时也应当对端口转发表用上互斥锁，防止数据冲突；

在 mac_port_map 中插入新的元素时，先初始化新的元素的信息（next、iface、visited 以及 mac 地址），然后计算新节点 mac 值的哈希值，并将其插入 hash cell 的尾端。

【遭遇的小坑】 PS：在该函数中进行修改 mac_port_map 的操作时，第一次忘记使用指针的指针进行修改，导致了实际测试中，lookup 的操作永远找不到。

➢ **int sweep_aged_mac_port_entry();**

```c
int sweep_aged_mac_port_entry()
{
    // TODO: implement the sweeping process here
    // fprintf(stdout, "TODO: implement the sweeping process here.\n");
    mac_port_entry_t *entry = NULL, *tmp = NULL;
    time_t now = time(NULL);
    // fprintf(stdout, "Sweeping aged mac port entry.\n");
    pthread_mutex_lock(&mac_port_map.lock);
    for (int i = 0; i < HASH_8BITS; i++) {
        entry = mac_port_map.hash_table[i];
        if(!entry) continue;
        tmp = entry->next;
        while(tmp){
            entry->next = tmp->next;
            if(((int)(now - tmp->visited)) > MAC_PORT_TIMEOUT){
                fprintf(stdout, ETHER_STRING " rm -> %s, %d\n", ETHER_FMT(tmp->mac), \
                        tmp->iface->name, (int)(now - tmp->visited));
                free(tmp);
            }
            tmp = entry->next;
        }
    }
    pthread_mutex_unlock(&mac_port_map.lock);
    return 0;
}
```

遍历整个端口转发表，用当前时间减去 visited 计算出老化时间，free 掉 age_time > MAC_PORT_TIMEOUT 的项。

- **void broadcast_packet(iface_info_t *iface, const char *packet, int len);**

```c
void broadcast_packet(iface_info_t *iface, const char *packet, int len)
{
    // TODO: implement the broadcast process here
    // fprintf(stdout, "TODO: implement the broadcast process here.\n");
    iface_info_t * ptr = NULL;
    list_for_each_entry(ptr, &instance->iface_list, list) {
        if (ptr->fd != iface->fd)
            iface_send_packet(ptr, packet, len);
    }
}
```

与实验四中广播操作一致。

- **void handle_packet(iface_info_t *iface, char *packet, int len);**

```c
void handle_packet(iface_info_t *iface, char *packet, int len)
{
    struct ether_header *eh = (struct ether_header *)packet;
    log(DEBUG, "the dst mac address is " ETHER_STRING ".\n", ETHER_FMT(eh->ether_dhost));

    // TODO: implement the packet forwarding process here
    // fprintf(stdout, "TODO: implement the packet forwarding process here.\n");

    // check whether the source mac address is in the FDB
    // if not in the FDB, add it into FDB
    if(!lookup_port(eh->ether_shost))
        insert_mac_port(eh->ether_shost, iface);

    // look up the destination mac address in the FDB
    // if not in FDB, then broadcast the packet
    // else, forward the packet
    iface_info_t * lookup = lookup_port(eh->ether_dhost);
    if(!lookup){
        // printf("broadcast.\n");
        broadcast_packet(iface, packet, len);
    }
    else {
        // printf("forward.\n");
        iface_send_packet(lookup, packet, len);
    }
}
```

1. 查看 source host 的 mac 地址是否在转发表中，如果不在，将其添加到转发表中；
2. 在转发表中查找 dest host 的 mac 地址，
   a) 如果没有查到，广播该 packet；
   b) 如果找到，沿该路径传输该 packet。