

计算机网络实验 5 报告

学号 2017K8009929059
姓名 於修远

交换机转发实验

一、实验内容

- 实现对数据结构 `mac_port_map` 的查询、插入、老化检查等操作。
- 实现包括广播在内的数据包转发处理操作。
- 对比交换机转发与集线器广播的性能。

二、实验流程

(一) `mac_port_map` 构建

1、查找方法编写

由于交换机内部存在一个检查转发表老化的线程和一个处理数据收发的主线程，所以包括查找、插入、老化检查等操作都需要加入互斥锁，如下图所示。

```
pthread_mutex_lock(&mac_port_map.lock);  
mac_port_entry_t *entry;  
u8 index = hash8(mac, ETH_ALEN);  
list_for_each_entry(entry, head: &mac_port_map.hash_table[index], list) {...}  
pthread_mutex_unlock(&mac_port_map.lock);
```

同时，如上图所示，对所有需要查找的 `mac` 地址，首先进行一次哈希操作，哈希函数为将 `mac` 地址的 6 个半字按位取异或。这里对实验提供的原代码进行了部分类型修改，使之可以更好地匹配。

```
static inline u8 hash8(const u8 *buf, int len) {  
    u8 result = 0;  
    for (int i = 0; i < len; i++)  
        result ^= buf[i];  
  
    return result;  
}
```

以哈希值为索引，开始查找转发表的对应该目。遍历该条目中的所有项，若始终没有找到 `mac` 地址完全匹配的项，最后将返回一个空指针；否则意味着查询到了即将进行操作的端口，所以需要更新该条目的提前释放锁，并返回表中所记录的该 `mac` 地址对应的端口指针。

```

int match = 1;
for (int j = 0; j < ETH_ALEN; ++j) {
    if (entry->mac[j] != mac[j]) {
        match = 0;
        break;
    }
}
if (match) {
    entry->visited = time( timer: NULL);
    pthread_mutex_unlock(&mac_port_map.lock);
    return entry->iface;
}

```

2、插入方法编写

插入方法的前半部分与查找相似，在互斥锁内首先进行哈希操作找到对应条目的索引链表并遍历。若查找到了对应条目，则更新访问时间，并直接返回。

若原本的转发表中没有对应条目，则需要索引链表中插入记录该 mac 地址-端口映射的条目。完成初始化后，调用`list.h`头文件中的`list_add_tail()`函数插入条目即可。

```

mac_port_entry_t *new_entry = (mac_port_entry_t *) malloc(sizeof(mac_port_entry_t));
new_entry->iface = iface;
new_entry->visited = time( timer: NULL);
for (int i = 0; i < ETH_ALEN; ++i) {
    new_entry->mac[i] = mac[i];
}
list_add_tail(&new_entry->list, &mac_port_map.hash_table[index]);

```

3、老化检查编写

老化检查函数在专门的老化检查线程中被调用，所以同样需要用线程锁确保操作的正确。老化检查需要遍历整个转发表中的所有条目，对每个条目计算上次更新开始到当前的时间。若超过设定值（本实验中为30s），则删除该条目，并释放存储空间。

```

for (int i = 0; i < HASH_8BITS; i++) {
    list_for_each_entry_safe(entry, q, head: &mac_port_map.hash_table[i], list) {
        time_t now = time( timer: NULL);
        if (now - entry->visited > MAC_PORT_TIMEOUT) {
            list_delete_entry(&entry->list);
            free(entry);
        }
    }
}

```

（二）数据包处理逻辑

1、数据包广播

本次实验中用到的数据包广播函数与上次的广播实验的实现完全一致，故不赘述。值得一提的是由于广播时遍历的接口为独立的链表，不涉及转发表的操作，所以不需要进行锁相关的操作。

2、数据包中转

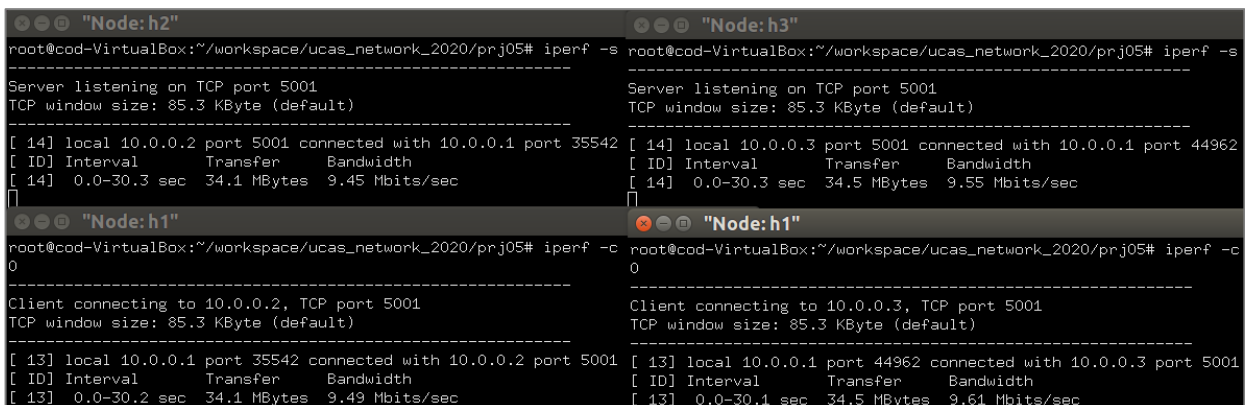
数据包的中转在`main.c`文件中的`handle_packet()`函数中实现，该函数实现了接收到数据包后的中转逻辑。不管后续处理如何，接收到数据包后，首先进行一次插入操作，即将数据包的源 mac 地址与接收端口的映射填入或更新到转发表中。紧接着需要根据目的 mac 地址查询转发端口，即调用查询函数，记录返回值。当返回值为空指针时，意味着转发表中不存在对应的 mac 地址-端口映射信息，从而调用广播函数直接对所有端口转发；否则，返回值即是目标端口，直接向该端口传递消息即可。

```
insert_mac_port(eh->ether_shost, iface);

iface_info_t *dest = lookup_port(eh->ether_dhost);
if (dest) iface_send_packet(dest, packet, len);
else broadcast_packet(iface, packet, len);
```

（三）交换机转发的性能测量

使用课程提供的`three_nodes_bw.py`脚本建立包含 3 个主机节点 h1、h2、h3 和 1 个交换机节点 s1 的 mininet 网络，其中 h1 与 s1 的带宽上限为 20Mbps，h2 和 h3 与 s1 的带宽上限均为 10Mbps。在 s1 节点上运行按上述代码编译的交换机程序 switch。此后，首先以 h2 和 h3 作为服务端，h1 作为客户端，通过两个 shell 窗口实现同时向 h2 和 h3 执行 iperf 操作，测得实际带宽如下图所示。即 h2 与 s1 之间带宽利用率为 94.5%，h3 与 s1 之间为 95.5%，h1 与 s1 之间为 95.5%。



```

"Node:h2"
root@cod-VirtualBox:~/workspace/ucas_network_2020/prj05# iperf -s
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
[ 14] local 10.0.0.2 port 5001 connected with 10.0.0.1 port 35542
[ ID] Interval      Transfer    Bandwidth
[ 14] 0.0-30.3 sec  34.1 MBytes  9.45 Mbits/sec

"Node:h3"
root@cod-VirtualBox:~/workspace/ucas_network_2020/prj05# iperf -s
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
[ 14] local 10.0.0.3 port 5001 connected with 10.0.0.1 port 44962
[ ID] Interval      Transfer    Bandwidth
[ 14] 0.0-30.3 sec  34.5 MBytes  9.55 Mbits/sec

"Node:h1"
root@cod-VirtualBox:~/workspace/ucas_network_2020/prj05# iperf -c 0
Client connecting to 10.0.0.2, TCP port 5001
TCP window size: 85.3 KByte (default)
[ 13] local 10.0.0.1 port 35542 connected with 10.0.0.2 port 5001
[ ID] Interval      Transfer    Bandwidth
[ 13] 0.0-30.2 sec  34.1 MBytes  9.49 Mbits/sec

"Node:h1"
root@cod-VirtualBox:~/workspace/ucas_network_2020/prj05# iperf -c 0
Client connecting to 10.0.0.3, TCP port 5001
TCP window size: 85.3 KByte (default)
[ 13] local 10.0.0.1 port 44962 connected with 10.0.0.3 port 5001
[ ID] Interval      Transfer    Bandwidth
[ 13] 0.0-30.1 sec  34.5 MBytes  9.61 Mbits/sec
```

随后以 h1 作为服务端，h2 和 h3 作为客户端，同时向 h1 执行 iperf 操作，测得实际带宽如下图所示。即 h2 与 s1 之间带宽利用率为 95.4%，h3 与 s1 之间为 95.6%，h1 与 s1 之间为 94.4%。

```
"Node:h2"
root@cod-VirtualBox:~/workspace/ucas_network_2020/prj05# iperf -c 10.0.0.1
Client connecting to 10.0.0.1, TCP port 5001
TCP window size: 85.3 KByte (default)
[ 13] local 10.0.0.2 port 50668 connected with 10.0.0.1 port 5001
[ ID] Interval      Transfer    Bandwidth
[ 13] 0.0-30.0 sec  34.1 MBytes  9.54 Mbits/sec
root@cod-VirtualBox:~/workspace/ucas_network_2020/prj05#

"Node:h3"
root@cod-VirtualBox:~/workspace/ucas_network_2020/prj05# iperf -c 10.0.0.1
Client connecting to 10.0.0.1, TCP port 5001
TCP window size: 85.3 KByte (default)
[ 13] local 10.0.0.3 port 56096 connected with 10.0.0.1 port 5001
[ ID] Interval      Transfer    Bandwidth
[ 13] 0.0-30.3 sec  34.5 MBytes  9.56 Mbits/sec
root@cod-VirtualBox:~/workspace/ucas_network_2020/prj05#

"Node:h1"
root@cod-VirtualBox:~/workspace/ucas_network_2020/prj05# iperf -s
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
[ 14] local 10.0.0.1 port 5001 connected with 10.0.0.2 port 50668
[ 15] local 10.0.0.1 port 5001 connected with 10.0.0.3 port 56096
[ ID] Interval      Transfer    Bandwidth
[ 14] 0.0-30.4 sec  34.1 MBytes  9.43 Mbits/sec
[ 15] 0.0-30.6 sec  34.5 MBytes  9.45 Mbits/sec
```

三、结果分析

(一) 广播网络的效率

通过测试可以看到，以 h1 作为服务端，h2 和 h3 向 h1 发送数据时，中转结点为交换机节点时的带宽利用特征与广播节点类似，三条数据通路的利用率都接近满载。广播节点的这一结果在上次实验中已有分析；而对于交换机节点，这一结果是显然的，h1 与 s1 之间的带宽应当近似于 s1 与 h2 和 h3 的带宽和，且受限于后两者的带宽上限，这一和也必不超过 h1 与 s1 的带宽上限。

但可以发现，在 h1 同时向 h2 和 h3 发送数据的情况下，交换机节点中转的效率明显高于广播节点，且与上一种情况的效率并无明显差异。这是因为广播节点必须同时向两条带宽上限为 10Mbps 的数据通路分发数据包，所以 h1 到 s1 的通路被额外施加了带宽不得超过 10Mbps 的限制，这严重影响了数据包传输的效率。而在交换机的情况下，除了第一次通讯时由于转发表为空所以需要广播数据包，由于服务端会返还响应包，交换机从第二个数据包开始就可以直接分端口转发，使得连接 h2 和 h3 的两条通路可以相对独立地进行数据收发。这样一来，充分利用带宽成为了可能，只有交换机转发部分的软件处理（如交换表查询）可能影响效率。

(二) 思考题

1、广播包对交换机行为逻辑的影响

广播包本身也是数据包的一种，所以在接收后同样需要进行插入操作，将源 mac 地址和收取端口的映射填入转发表中。这一行为是合理的，因为一般来说一个主机在发送广播包后会等待一个单播的回复。

而在此之后，广播包不需要进行转发表的检索。也就是说，需要在调用转发表函数前首先检查数据包的目的 mac 地址是否为全 0XFF。若是，则直接调用广播函数，完成后直接结束对该数据包的处理；否则就回

到前述代码的处理，即查询转发表后决定是单播转发还是广播。

综上，广播包并不影响交换机的主体行为逻辑，只是增加了一种需要预处理的情况。

2、用交换机直接连接终端是否可行

单从理论上讲，交换机可以实现任意多个主机的连接。然而，从技术上看这并不现实。主要有以下两方面的原因。

首先，交换机的扩展能力有限。用交换机与交换机连接，是一种线性的扩展，扩展效率不高；同时由于 MAC 地址不反映实际的拓扑信息，转发表的规模会变得极其庞大，消耗巨大的存储空间；即使保存了转发表，也很难进行有效的路由寻址，维护和查询转发表的时间开销会对交换机的运行效率产生严重影响。同时，这也会使得交换机之间确定通路的生成树算法的时间开销急剧膨胀。这些都从实现效果上制约了用交换机直接构建全世界的互联网。

其次，当今的互联网环境中其实存在着许多异构网络，例如以太网、WiFi 等，不同网络应用不同的地址模式、介质访问协议、服务模型，只使用交换机转发的情况下，不仅会大大增加内部实现的复杂度和处理压力，且并非所有类型的网络都如以太网一样使用 MAC 地址作为转发信息，交换机的转发表无法实现或实现难度极大。

综上，用足够多的交换机直接连接世界上所有终端的方式在技术上不可行。