实验报告

姓名：王苑铮 学号：2015K8009922002

1.实验题目： 交换机学习实验

2.实验内容：

- 1 操作数据结构mac_port_map，实现查询lookup_port,插入insert_mac_port,删除老化节点sweep_aged_mac_port_entry三个操作
- 2 实现广播操作
- 3 实现交换机学习的算法:
  - 在转发表里查找是否有源地址。
    - 如果有原地址目的地址。如果有，更新访问时间
    - 如果没有：把源地址加入到转发表，更新访问时间
  - 在转发表里查找是否有目的地址。
    - 如果有，则向这个目的地址发消息，并更新目的地址的访问时间
    - 如果没有，则广播这则消息
  - 用另一个线程检查转发表，删去一定时间内没有被访问的地址
- 4 比较广播与交换机学习两种机制的带宽利用情况

3.实验过程

mac_port_map的三个操作
1.删去老化的port

```
int sweep_aged_mac_port_entry()
{
        // TODO: implement the sweeping process here
        static int i = 0;
        int sleep_time = MAC_PORT_TIMEOUT;
        while(1){
                //fprintf(stdout, "%d sweeping\n",++i);
                pthread_mutex_lock(&mac_port_map.lock);
                mac_port_entry_t* entry;
                for(int i=0; i<HASH_8BITS ; ++i){
                        mac_port_entry_t *head,*tail;
                        tail=head=mac_port_map.hash_table[i];
                        for(entry=mac_port_map.hash_table[i]; entry!=NULL ; ){
                                //时间超过30s没访问
                                if(time(NULL)-entry->visited >= sleep_time){
                                        mac_port_entry_t* p;
                                        p=entry;
                                        //如果这个entry是头部：需要把hash表项换成下个entry
                                        if(head == entry){
                                                tail=head=mac_port_map.hash_table[i]=en
                                        }
                                        //如果不是头部，链表删掉这个节点
                                        else{
                                                tail->next = entry->next;
                                        }
                                        entry=entry->next;
                                        free(p);
                                }
                                //跳过
                                else{
                                        tail = entry;
                                        entry=entry->next;
                                }
                        }
                }
                pthread_mutex_unlock(&mac_port_map.lock);
                sleep(sleep_time);
        }
        return 0;
}
```

2.插入新的port

```
void insert_mac_port(u8 mac[ETH_ALEN], iface_info_t *iface)
{
        pthread_mutex_lock(&mac_port_map.lock);
        // TODO: implement the insertion process here
        //fprintf(stdout, "insert_mac_port\n");
        mac_port_entry_t* entry = mac_port_map.hash_table[hash8(mac,ETH_ALEN)];
        if(entry == NULL){
                entry = (mac_port_entry_t*)malloc(sizeof(mac_port_entry_t));
                memcpy(entry->mac,mac,ETH_ALEN*sizeof(u8));
                entry->iface = iface;
                entry->visited = time(NULL);
                entry->next = NULL;
                mac_port_map.hash_table[hash8(mac,ETH_ALEN)] = entry;
        }
        else{
                //沿着链表查找。如果找到，更新时间。如果没找到（查到链表尾部），插入新表
                int find = 0;
                mac_port_entry_t* p=entry;
                mac_port_entry_t* tail = NULL; //链表最后一个非空node
                for(p=entry ; p!=NULL ; p=p->next){
                        //查到
                        if(memcmp(p->mac,mac,ETH_ALEN*sizeof(u8))==0){
                                entry = p;
                                entry->visited = time(NULL);
                                find = 1;
                                break;
                        }
                        tail = p;
                }
                if(!find){
                        entry = (mac_port_entry_t*)malloc(sizeof(mac_port_entry_t));
                        memcpy(entry->mac,mac,ETH_ALEN*sizeof(u8));
                        entry->iface = iface;
                        entry->visited = time(NULL);
                        entry->next = NULL;
                        tail->next = entry;
                }
        }
        pthread_mutex_unlock(&mac_port_map.lock);
}
```

3.在转发表中查找port

```
iface_info_t *lookup_port(u8 mac[ETH_ALEN])
{
        pthread_mutex_lock(&mac_port_map.lock);
        // TODO: implement the lookup process here
        //fprintf(stdout, "lookup_port\n");
        mac_port_entry_t* entry = mac_port_map.hash_table[hash8(mac,ETH_ALEN)];
        if(entry!= NULL){
                for( ; entry!=NULL ; entry=entry->next){
                        //查到
                        if(memcmp(entry->mac,mac,ETH_ALEN*sizeof(u8))==0){
                                entry->visited = time(NULL);
                                break;
                        }
                }
        }
        pthread_mutex_unlock(&mac_port_map.lock);
        return (entry==NULL)?NULL:entry->iface;
}
```

广播

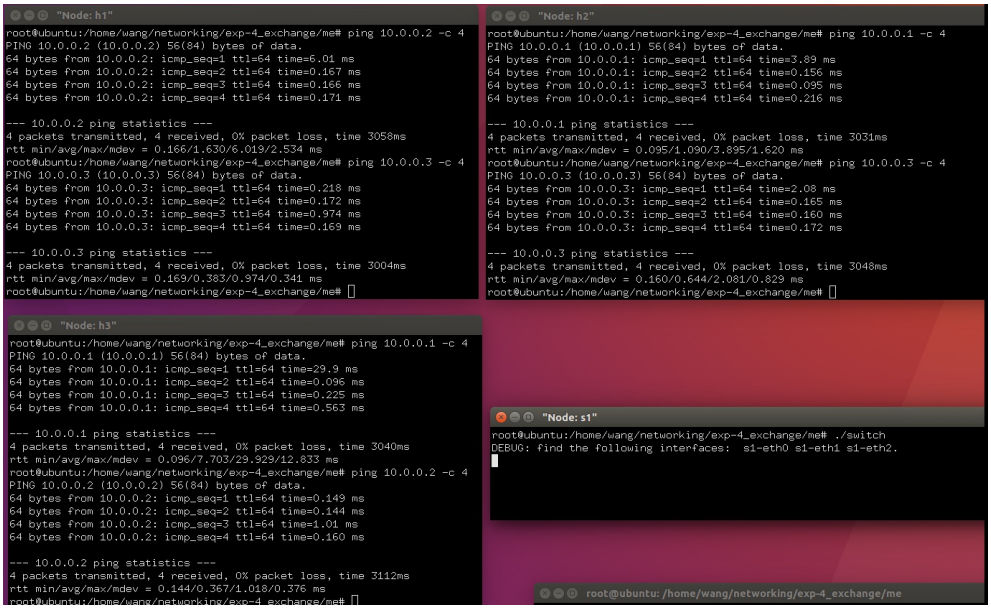
```
void broadcast_packet(iface_info_t *iface, const char *packet, int len)
{
        // TODO: implement the broadcast process here
        //fprintf(stdout, "TODO: implement the broadcast process here.\n");
        //mine
        iface_info_t *ift = NULL;
        list_for_each_entry(ift, &instance->iface_list,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

交换机学习

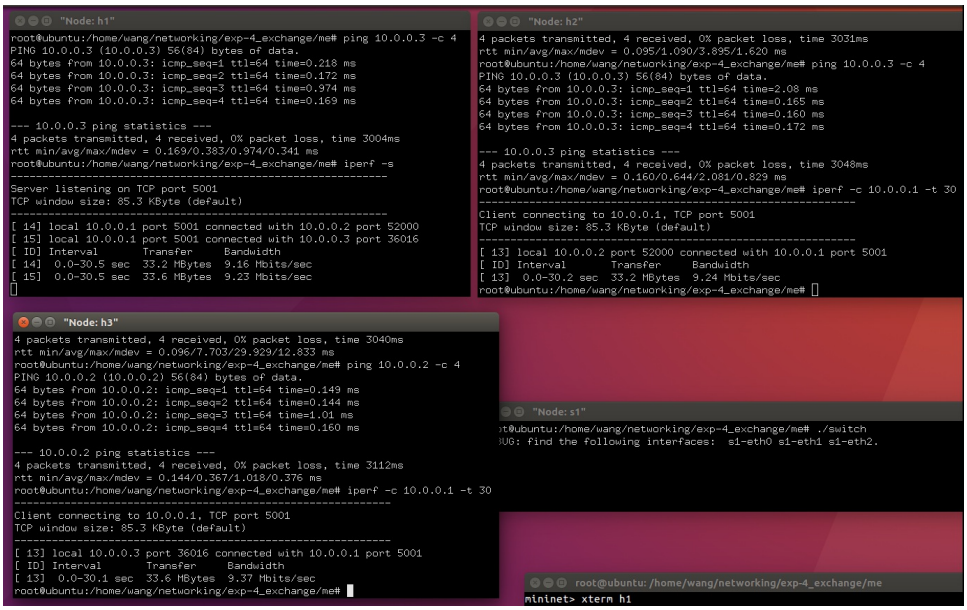
```c
void handle_packet(iface_info_t *iface, char *packet, int len)
{
        struct ether_header *eh = (struct ether_header *)packet;
        //log(DEBUG, "the dst mac address is " ETHER_STRING ".\n", ETHER_FMT(eh->ether_

        // TODO: implement the packet forwarding process here
        //fprintf(stdout, "handle.\n");
        iface_info_t* d_ift=lookup_port(eh->ether_dhost),
                                * s_ift=lookup_port(eh->ether_shost);

        if(d_ift != NULL){
                //fprintf(stdout, "find.\n");
                iface_send_packet(d_ift,packet,len);
        }
        else{
                //fprintf(stdout, "broadcast****.\n");
                broadcast_packet(iface,packet,len);
        }

        if(s_ift==NULL){
                insert_mac_port(eh->ether_shost,iface);
        }
}
```

4.实验结果

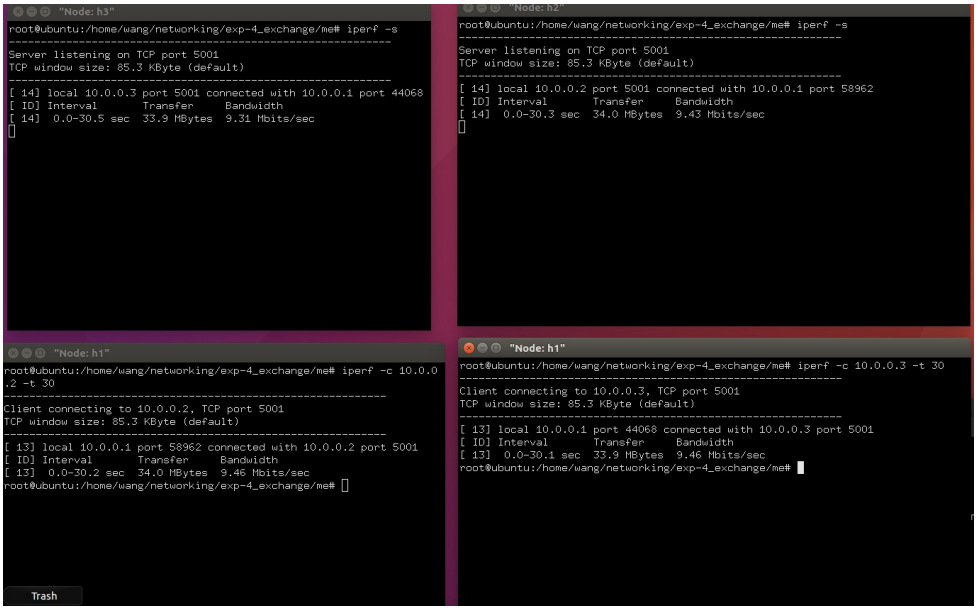
1.互相ping



2.性能测试：h1做service，h2,h3做clinet

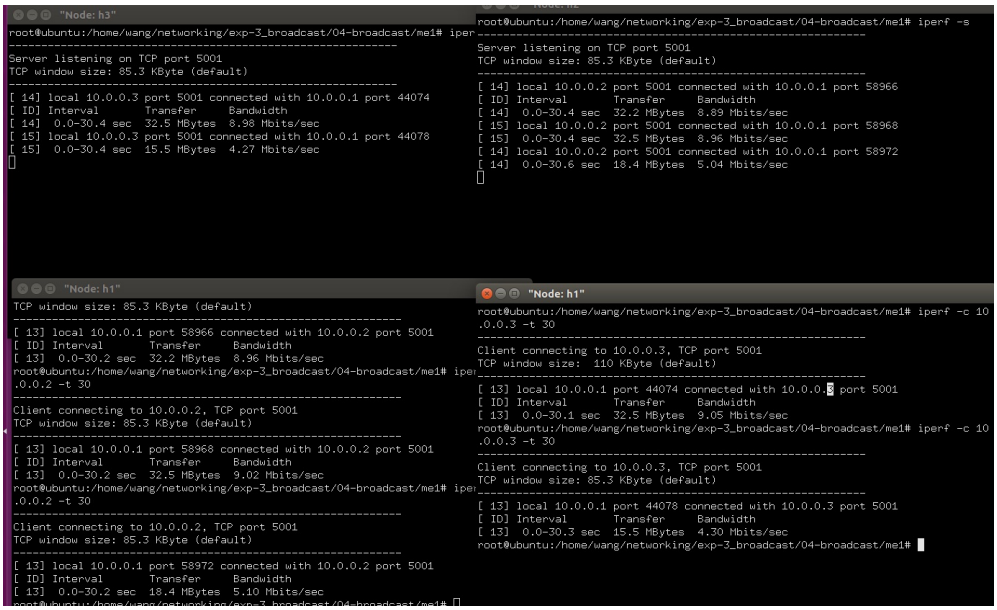


3.性能测试：h1做clinet，h2,h3做service



5.结果分析

广播算法性能测试：h1做clinet，h2,h3做service



h1,h2,h3最大带宽为20Mb/s,10Mb/s,10Mb/s

- **h1 server, h2,h3 clinet**

  结果：

  h2,h3不管是同时向h1发消息，还是分别发送，都是接近10Mb/s.学习算法和广播都是这样的结果。

  分析：

  对于学习算法，h2向h1发消息时，交换机不向h3发消息。h3亦然。由于h1的最大带宽为h2，h3最大带宽之和，所以即使h2，h3同时向h1发消息，h1的带宽也能容纳，因此h2，h3都接近最大发送带宽。

  对于广播算法，虽然h2向h1发消息时也向h3发消息，但占用的是h3的接收带宽而不占用h3发送带宽。因而h2,h3同时发送也接近最大发送带宽

- **h1 clinet, h2,h3 server**

  结果：

  对于学习算法，h1无论是单独还是同时向h2、h3发消息，都接近最大带宽。对于广播算法，h1单独向h2、h3发消息时接近h2、h3最大带宽，同时向h2、h3发消息时带宽大约减半

  分析：

  对于学习算法，h1向h2发消息时，交换机不向h3发消息，并且h1带宽为h2，h3之和，因此h1同时向h2，h3发消息，两边互不干扰，能接近h2、h3最大带宽。

  对于广播算法，h1向h2发消息时h3也会收到消息，占用h3接收带宽。h1同时向h2、h3发消息时，h2的接收带宽内，除了收到发给h2的消息，还有发给h3的消息。h3亦然。因而最终结果是h2、h3的最大带宽的大约一半

结论：

学习算法可以防止占用除了源、目的节点以外的节点的带宽，从而在多个节点同时发出多个请求时，提高了有效带宽，因而优于广播算法

6.bug以及原因：

现象1： ping不通
结果： 交换机的学习算法一开始设计的不对。最初版我是在没有查到目的地址时广播并把目的地址插进转发表。正确的做法应该是在没有查到源地址时把源地址插入到转发表，在没有查到目的地址时广播（不是把目的地址插入转发表）

现象2： 交换机学习算法在iperf时带宽只有3.几M
结果： 屏幕打印的信息太多。去掉打印信息就达到9.几M了