

# 广播网络实验

热伊莱·图尔贡 2018K8009929030

## 一、 实验内容

1. 实现节点广播的 broadcast\_packet 函数

2. 验证广播网络能够正常运行

从一个端节点 ping 另一个端节点

3. 验证广播网络的效率

在 three\_nodes\_bw.py 进行 iperf 测量

两种场景：

H1: iperf client; H2, H3: servers (h1 同时向 h2 和 h3 测量)

H1: iperf server; H2, H3: clients (h2 和 h3 同时向 h1 测量)

4. 自己动手构建环形拓扑，验证该拓扑下节点广播会产生数据包环路

## 二、 实验结果

1. 实现 broadcast\_packet 函数：

```
void broadcast_packet(iface_info_t *iface, const char *packet, int
{
    // TODO: broadcast packet
    // fprintf(stdout, "TODO: broadcast packet.\n");
    //非接受节点若收到数据包，需要将数据包从其他端口转发出去
    iface_info_t *iface_entry = NULL;
    list_for_each_entry(iface_entry, &instance->iface_list, list)
    {
        if (iface_entry->fd != iface->fd)
        {
            iface_send_packet(iface_entry, packet, len);
        }
    }
}
```

2. 利用 `three_nodes_bw.py` 拓扑文件打开 Mininet, 从一个端节点 ping 另一个端节点, 验证广播网络能够正常运行。

```
"Node: h1"
root@rayilam-VirtualBox:/home/rayilam/CN/03-hub+switch/hub# ping 10.0.0.2 -c 4
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.090 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.067 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.068 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.068 ms

--- 10.0.0.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3073ms
rtt min/avg/max/mdev = 0.067/0.073/0.090/0.009 ms
root@rayilam-VirtualBox:/home/rayilam/CN/03-hub+switch/hub# ping 10.0.0.3 -c 4
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=0.187 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=0.064 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=0.058 ms
64 bytes from 10.0.0.3: icmp_seq=4 ttl=64 time=0.059 ms

--- 10.0.0.3 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3056ms
rtt min/avg/max/mdev = 0.058/0.092/0.187/0.054 ms
root@rayilam-VirtualBox:/home/rayilam/CN/03-hub+switch/hub#
```

```
"Node: h2"
root@rayilam-VirtualBox:/home/rayilam/CN/03-hub+switch/hub# ping 10.0.0.1 -c 4
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=0.117 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=0.091 ms
64 bytes from 10.0.0.1: icmp_seq=3 ttl=64 time=0.061 ms
64 bytes from 10.0.0.1: icmp_seq=4 ttl=64 time=0.068 ms

--- 10.0.0.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3058ms
rtt min/avg/max/mdev = 0.061/0.084/0.117/0.021 ms
root@rayilam-VirtualBox:/home/rayilam/CN/03-hub+switch/hub# ping 10.0.0.3 -c 4
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=0.272 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=0.087 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=0.060 ms
64 bytes from 10.0.0.3: icmp_seq=4 ttl=64 time=0.063 ms

--- 10.0.0.3 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3073ms
rtt min/avg/max/mdev = 0.060/0.120/0.272/0.088 ms
root@rayilam-VirtualBox:/home/rayilam/CN/03-hub+switch/hub#
```

```
"Node: h3"
root@rayilam-VirtualBox:/home/rayilam/CN/03-hub+switch/hub# ping 10.0.0.2 -c 4
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.336 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.087 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.077 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.077 ms

--- 10.0.0.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3059ms
rtt min/avg/max/mdev = 0.077/0.144/0.336/0.110 ms
root@rayilam-VirtualBox:/home/rayilam/CN/03-hub+switch/hub# ping 10.0.0.1 -c 4
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=0.219 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=0.122 ms
64 bytes from 10.0.0.1: icmp_seq=3 ttl=64 time=0.073 ms
64 bytes from 10.0.0.1: icmp_seq=4 ttl=64 time=0.069 ms

--- 10.0.0.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3079ms
rtt min/avg/max/mdev = 0.069/0.120/0.219/0.060 ms
```

3. 利用 three\_nodes\_bw.py 拓扑文件打开 Mininet, 进行 iperf 测量, 验证广播网络的效率。

H1: iperf client; H2, H3: servers

```
"Node: h2"
root@rayilam-VirtualBox:/home/rayilam/CN/03-hub+switch/hub# 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 48366
[ ID] Interval      Transfer    Bandwidth
[ 14] 0.0-31.1 sec  14.2 MBytes  3.84 Mbits/sec
```

```
"Node: h3"
root@rayilam-VirtualBox:/home/rayilam/CN/03-hub+switch/hub# 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 52126
[ ID] Interval      Transfer    Bandwidth
[ 14] 0.0-30.6 sec  20.8 MBytes  5.69 Mbits/sec
```

```
"Node: h1"
root@rayilam-VirtualBox:/home/rayilam/CN/03-hub+switch/hub# iperf -c 10.0.0.2 -
t 30 & iperf -c 10.0.0.3 -t 30
[1] 2421
-----
Client connecting to 10.0.0.3, TCP port 5001
TCP window size: 144 KByte (default)
-----
[ 13] local 10.0.0.1 port 52126 connected with 10.0.0.3 port 5001
-----
Client connecting to 10.0.0.2, TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[ 13] local 10.0.0.1 port 48366 connected with 10.0.0.2 port 5001
[ ID] Interval      Transfer      Bandwidth
[ 13] 0.0-30.1 sec  20.8 MBytes  5.79 Mbits/sec
root@rayilam-VirtualBox:/home/rayilam/CN/03-hub+switch/hub# [ ID] Interval
Transfer      Bandwidth
[ 13] 0.0-30.7 sec  14.2 MBytes  3.90 Mbits/sec
```

h1 节点同时向 h2 节点和 h3 节点测量，可以看出 h1 节点向 h2 节点和 h3 节点的发送带宽分别为 5.79Mbps 和 3.90Mbps。h2 节点和 h3 节点的接收带宽分别为 3.84Mbps 和 5.69Mbps。

而在拓扑文件中，h1 -> b1 的带宽为 20Mbps，b1 -> h2 的带宽为 10Mbps，b1 -> h3 的带宽为 10Mbps。因此带宽的利用率为 48.45%。

H1: iperf server; H2, H3: clients

```
"Node: h2"
root@rayilam-VirtualBox:/home/rayilam/CN/03-hub+switch/hub# iperf -c 10.0.0.1 -
t 30
-----
Client connecting to 10.0.0.1, TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[ 13] local 10.0.0.2 port 36258 connected with 10.0.0.1 port 5001
[ ID] Interval      Transfer      Bandwidth
[ 13] 0.0-30.7 sec  37.1 MBytes  10.1 Mbits/sec
root@rayilam-VirtualBox:/home/rayilam/CN/03-hub+switch/hub# █
```

```
"Node: h3"
root@rayilam-VirtualBox:/home/rayilam/CN/03-hub+switch/hub# iperf -c 10.0.0.1 -
t 30
-----
Client connecting to 10.0.0.1, TCP port 5001
TCP window size: 153 KByte (default)
-----
[ 13] local 10.0.0.3 port 45488 connected with 10.0.0.1 port 5001
[ ID] Interval      Transfer      Bandwidth
[ 13] 0.0-30.3 sec  33.1 MBytes  9.16 Mbits/sec
root@rayilam-VirtualBox:/home/rayilam/CN/03-hub+switch/hub# █
```

```
"Node: h1"
root@rayilam-VirtualBox:/home/rayilam/CN/03-hub+switch/hub# 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 10.0.0.3 port 45488
[ 15] local 10.0.0.1 port 5001 connected with 10.0.0.2 port 36258
[ ID] Interval      Transfer    Bandwidth
[ 14] 0.0-32.6 sec  33.1 MBytes  8.53 Mbits/sec
[ 15] 0.0-34.2 sec  37.1 MBytes  9.10 Mbits/sec
```

h2 节点和 h3 节点同时向 h1 节点测量，可以看出 h1 节点接收 h2 节点和 h3 节点的接收带宽分别为 9.10Mbps 和 8.53Mbps。h2 节点和 h3 节点的发送带宽分别为 10.10Mbps 和 9.16Mbps。

而在拓扑文件中，h1 -> b1 的带宽为 20Mbps，b1 -> h2 的带宽为 10Mbps，b1 -> h3 的带宽为 10Mbps。因此带宽的利用率为 88.15%。

#### 4. 构建环形拓扑网络

```
39 class BroadcastTopo(Topo):
40     def build(self):
41         h1 = self.addHost('h1')
42         h2 = self.addHost('h2')
43         b1 = self.addHost('b1')
44         b2 = self.addHost('b2')
45         b3 = self.addHost('b3')
46
47         self.addLink(h1, b1, bw=20)
48         self.addLink(h2, b2, bw=20)
49         self.addLink(b1, b2, bw=20)
50         self.addLink(b1, b3, bw=20)
51         self.addLink(b2, b3, bw=20)
52
53 if __name__ == '__main__':
54     check_scripts()
55
56     topo = BroadcastTopo()
57     net = Mininet(topo = topo, link = TCLink, controller = None)
58
59     h1, h2, b1, b2, b3 = net.get('h1', 'h2', 'b1', 'b2', 'b3')
60     h1.cmd('ifconfig h1-eth0 10.0.0.1/8')
61     h2.cmd('ifconfig h2-eth0 10.0.0.2/8')
62
63     clearIP(b1)
64
65     for h in [ h1, h2, b1, b2, b3 ]:
66         h.cmd('./scripts/disable_offloading.sh')
67         h.cmd('./scripts/disable_ipv6.sh')
```

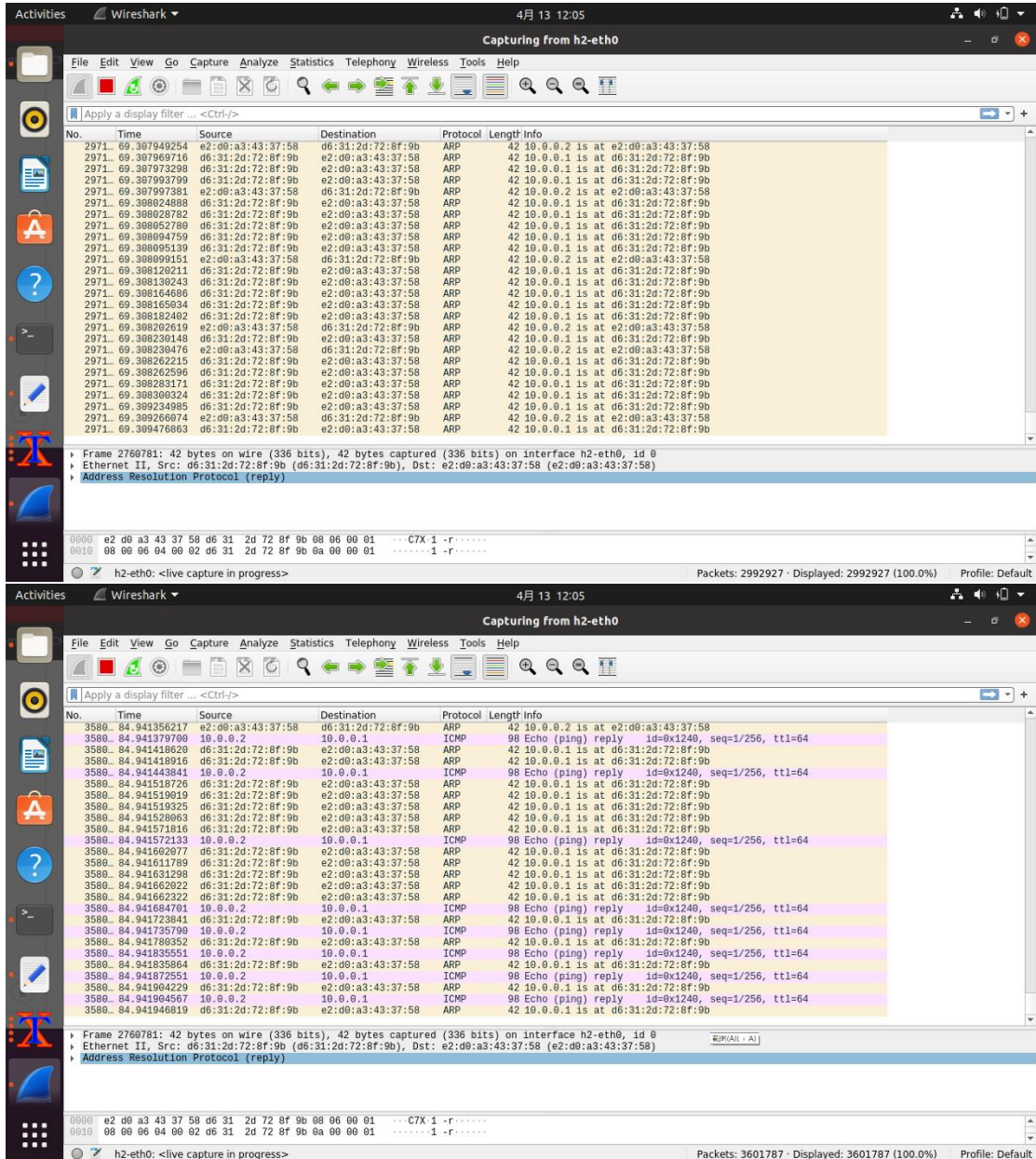
h1# ping -c 1 10.0.0.2 指令



```
"Node: h1"
root@rayilam-VirtualBox:/home/rayilam/CN/03-hub+switch/hub# ping -c 1 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.240 ms

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.240/0.240/0.240/0.000 ms
root@rayilam-VirtualBox:/home/rayilam/CN/03-hub+switch/hub#
```

h2 利用 wireshark 抓包结果:



由上图可知，数据包在该环形拓扑中被不断广播。

# 交换机转发实验

热伊莱·图尔贡 2018K8009929030

## 一、 实验内容

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

```
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、 设计思路

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

```
iface_info_t *lookup_port(u8 mac[ETH_ALEN]);
```

该函数的作用是：在转发表中查找对应 `mac` 地址和 `iface` 映射的表项。若找到对应的表项，则返回查询 `mac` 地址对应的 `iface`。

```

iface_info_t *lookup_port(u8 mac[ETH_ALEN])
{
    // TODO: implement the lookup process here
    /* fprintf(stdout, TODO: implement the lookup process here.\n");*/
    u8 mac_hash = hash8((void *)mac, ETH_ALEN);
    mac_port_entry_t *entry = NULL;

    pthread_mutex_lock(&mac_port_map.lock);
    list_for_each_entry(entry, &mac_port_map.hash_table[mac_hash], list)
    {
        if (memcmp(entry->mac, mac, ETH_ALEN) == 0)
        {
            pthread_mutex_unlock(&mac_port_map.lock);
            return entry->iface;
        }
    }
    pthread_mutex_unlock(&mac_port_map.lock);

    return NULL;
}

```

void insert\_mac\_port(u8 mac[ETH\_ALEN], iface\_info\_t \*iface);

该函数的作用是当转发表中没有源 mac 地址和对应 iface 的映射表项时，将源 mac 地址与该 iface 插入到转发表当中。



```

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");
    mac_port_entry_t *entry = NULL;
    u8 mac_hash = hash8((void *)mac, ETH_ALEN);
    time_t now = time(NULL);

    pthread_mutex_lock(&mac_port_map.lock);
    list_for_each_entry(entry, &mac_port_map.hash_table[mac_hash], list)
    {
        if (memcmp(entry->mac, mac, ETH_ALEN) == 0)
        {
            entry->iface = iface;
            entry->visited = now;

            pthread_mutex_unlock(&mac_port_map.lock);
            return;
        }
    }

    entry = malloc(sizeof(mac_port_entry_t));
    memcpy(entry->mac, mac, ETH_ALEN);
    entry->iface = iface;
    entry->visited = now;
    list_add_tail(&entry->list, &mac_port_map.hash_table[mac_hash]);
    pthread_mutex_unlock(&mac_port_map.lock);
}

```

```
int sweep_aged_mac_port_entry();
```

该函数的作用是当转发表中的表项超过 30s 没有被查询，则删除冗旧的表项。

```

int sweep_aged_mac_port_entry()
{
    // TODO: implement the sweeping process here
    // fprintf(stdout, "TODO: implement the sweeping process here.\n");
    // int q = 0;
    mac_port_entry_t *entry = NULL;
    mac_port_entry_t *next_entry = NULL;
    time_t now = time(NULL);
    pthread_mutex_lock(&mac_port_map.lock);
    for (int i = 0; i < HASH_8BITS; i++)
    {
        list_for_each_entry_safe(entry, next_entry, &mac_port_map.hash_table[i],
        {
            if ((int)(now - entry->visited) > MAC_PORT_TIMEOUT)
            {
                list_delete_entry(&entry->list);
                free(entry);
                // q++;
            }
        }
    }
    pthread_mutex_unlock(&mac_port_map.lock);
    // return q++;
    return 0;
}

```

```

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

```

该函数为广播收到的包，代码复用广播网络实验的代码即可。

```

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

```

该函数为处理收到的包。

处理包的逻辑为先调用 lookup\_port 函数，检查目的 mac 与端口的映射有无在映射表中。若存在，则根据这个表项进行发包，若没有则广播。另外需要检查源 mac 地址与转发端口的映射是否存在表中，若没有，则调用 insert\_mac\_port 函数插入表中。

```

void handle_packet(iface_info_t *iface, char *packet, int len)
{
    // TODO: implement the packet forwarding process here
    // fprintf(stdout, TODO: implement the packet forwarding process here. \n")

    struct ether_header *eh = (struct ether_header *)packet;
    iface_info_t *dest_iface = lookup_port(eh->ether_dhost);
    if (dest_iface)
    {
        iface_send_packet(dest_iface, packet, len);
    }
    else
    {
        broadcast_packet(iface, packet, len);
    }

    /*log(DEBUG, "the dst mac address is " ETHER_STRING ". \n", ETHER_FMT(eh->ether_dhost));*/
    insert_mac_port(eh->ether_shost, iface);
    free(packet);
}

```

### 三、 结果验证

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

switch-reference 的结果如下：

```

"Node: h2"
root@rayilam-VirtualBox:/home/rayilam/CN/03-hub+switch/switch# 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 48374
[ ID] Interval      Transfer      Bandwidth
[ 14] 0.0-30.3 sec  34.5 MBytes  9.56 Mbits/sec

"Node: h3"
root@rayilam-VirtualBox:/home/rayilam/CN/03-hub+switch/switch# 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 52132
[ ID] Interval      Transfer      Bandwidth
[ 14] 0.0-30.2 sec  34.4 MBytes  9.56 Mbits/sec

```

```
"Node: h1"
root@rayilam-VirtualBox:/home/rayilam/CN/03-hub+switch/switch# iperf -c 10.0.0.
2 -t 30 & iperf -c 10.0.0.3 -t 30
[1] 3753
-----
Client connecting to 10.0.0.3, TCP port 5001
TCP window size: 144 KByte (default)
-----
[ 13] local 10.0.0.1 port 52132 connected with 10.0.0.3 port 5001
-----
Client connecting to 10.0.0.2, TCP port 5001
TCP window size: 102 KByte (default)
-----
[ 13] local 10.0.0.1 port 48374 connected with 10.0.0.2 port 5001
[ ID] Interval      Transfer    Bandwidth
[ 13] 0.0-30.1 sec  34.4 MBytes  9.59 Mbits/sec
root@rayilam-VirtualBox:/home/rayilam/CN/03-hub+switch/switch# [ ID] Interval
      Transfer    Bandwidth
[ 13] 0.0-30.1 sec  34.5 MBytes  9.62 Mbits/sec
```

h1 节点同时接收 h2 节点和 h3 节点，可以看出 h2 节点和 h3 节点的发送带宽分别为 9.62Mbps 和 9.59Mbps，利用率为 96.05%。

hub-reference 的结果如下：

```
"Node: h2"
root@rayilam-VirtualBox:/home/rayilam/CN/03-hub+switch/switch# 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 48376
[ ID] Interval      Transfer    Bandwidth
[ 14] 0.0-30.6 sec   21.9 MBytes  5.99 Mbits/sec
```

```
"Node: h3"
root@rayilam-VirtualBox:/home/rayilam/CN/03-hub+switch/switch# 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 52134
[ ID] Interval      Transfer    Bandwidth
[ 14] 0.0-31.1 sec   13.0 MBytes  3.51 Mbits/sec
```

```
"Node: h1"
root@rayilam-VirtualBox:/home/rayilam/CN/03-hub+switch/switch# iperf -c 10.0.0.
2 -t 30 & iperf -c 10.0.0.3 -t 30
[1] 3809
-----
Client connecting to 10.0.0.3, TCP port 5001
TCP window size: 144 KByte (default)
-----
[ 13] local 10.0.0.1 port 52134 connected with 10.0.0.3 port 5001
-----
Client connecting to 10.0.0.2, TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[ 13] local 10.0.0.1 port 48376 connected with 10.0.0.2 port 5001
[ ID] Interval      Transfer      Bandwidth
[ 13] 0.0-30.1 sec   21.9 MBytes   6.09 Mbits/sec
[ ID] Interval      Transfer      Bandwidth
[ 13] 0.0-30.6 sec   13.0 MBytes   3.57 Mbits/sec
[1]+  Done                  iperf -c 10.0.0.2 -t 30
root@rayilam-VirtualBox:/home/rayilam/CN/03-hub+switch/switch#
```

h1 节点同时接收 h2 节点和 h3 节点，可以看出 h2 节点和 h3 节点的发送带宽分别为 5.99Mbps 和 3.51Mbps，平均利用率为 47.50%。  
switch 的带宽利用率明显比 hub 高。因此 switch 利用转发表的方式明显比 hub 的直接广播模式效率要高。

## 四、 思考题

1. 交换机在转发数据包时有两个查表操作：根据源 MAC 地址、根据目的 MAC 地址，为什么在查询源 MAC 地址时更新老化时间，而查询目的 MAC 地址时不更新呢？

提示：1、查询目的 MAC 地址时是否有必要更新；2、如果更新的话，当一个主机从交换机的一个网口切换到了另一个网口，会有什么问题？

查询目的 MAC 地址时没有必要更新，老化操作可以把过期的目的地址删除。

如果更新的话，当一个主机从交换机的一个网口切换到了另一个网口，原来的地址的老化时间一直被更新，导致无法被老化删除。

2. 网络中存在广播包，即发往网内所有主机的数据包，其目的 MAC 地址设置为全 0xFF，例如 ARP 请求数据包。这种广播包对交换



机转发表逻辑有什么影响？

当在整个传输网络刚刚启动时，转发表的内容为空，主机 A 想与主机 B 进行通信，会经过如下过程：

A 检查自己的 ARP 表，发现 B 的 MAC 地址不在自己的 ARP 表里，因此 A 只能向交换机发出 ARP 请求数据包。交换机学习 A 的 MAC 地址到自己的转发表，并广播 ARP 请求报文；B 接收到 ARP 请求报文，学习 A 的 MAC 地址到自己的 ARP 表并向交换机发出 ARP 回应报文（当其他主机接收这个广播包之后，不会接受它）；交换机学习 B 的 MAC 地址到自己的转发表，并向 A 发出 ARP 回应报文；A 接受到 B 的 ARP 回应报文，并学习 B 的 MAC 的地址；

在这个过程中帮助所有参与的交换机学习建立起了映射转发表。

3. 理论上，足够多个交换机可以连接起全世界所有的终端。请问，使用这种方式连接亿万台主机是否技术可行？并说明理由。

不行。

足够多的交换机虽然可以链接全世界的终端，但是这样缺少层次化的网络结构难以维护。当网络连接中遇到环形拓扑时，需要通过生成树算法来破坏环形结构，以免出现在环形拓扑结构不停转发的情况。而假如用足够多个交换机连接起全世界所有的终端，生成树算法难以收

敛。另外全部用交换机进行连接会带来安全性问题。