交换机网络实验

2021K8009929006
范子墨

一、 实验内容

1、hub

（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

2、switch

（1）实现对数据结构 max_port_map 的所有操作，以及数据包的转发和广播操作

（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、 设计思路及验证

1、hub

（1） broadcast 函数

节点广播的伪代码如下

```
foreach iface in iface_list:
    if iface != rx_iface:
        iface_send_packet(iface, packet, len);
```

根据伪代码，使用 list_for_each_entry 遍历每个节点，node 指针暂存当前节点。除了当前接收消息的节点端口，向所有端口转发接收到的消息。

```
void broadcast_packet(iface_info_t *iface, const char *packet, int len)
{
    iface_info_t *node = NULL;
    list_for_each_entry(node, &instance->iface_list, list) {
        if (node->index != iface->index) {
            iface_send_packet(node, packet, len);
        }
    }
}
```

（2） ping 通

10.0.0.1 是 h1 节点，10.0.0.2 是 h2 节点，10.0.0.3 是 h3 节点，三个节点互相执行 ping 指令，可以看到三个节点能够相互 ping 通

```
root@leona-virtual-machine:/home/leona/CN/04-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=0.492 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.302 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.317 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.265 ms

--- 10.0.0.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3068ms
rtt min/avg/max/mdev = 0.265/0.344/0.492/0.087 ms
root@leona-virtual-machine:/home/leona/CN/04-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=0.391 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=0.325 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=0.289 ms
64 bytes from 10.0.0.3: icmp_seq=4 ttl=64 time=0.328 ms

--- 10.0.0.3 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3079ms
rtt min/avg/max/mdev = 0.289/0.333/0.391/0.036 ms
root@leona-virtual-machine:/home/leona/CN/04-hub+switch/hub#
```

图表 1：h1 ping h2 h3

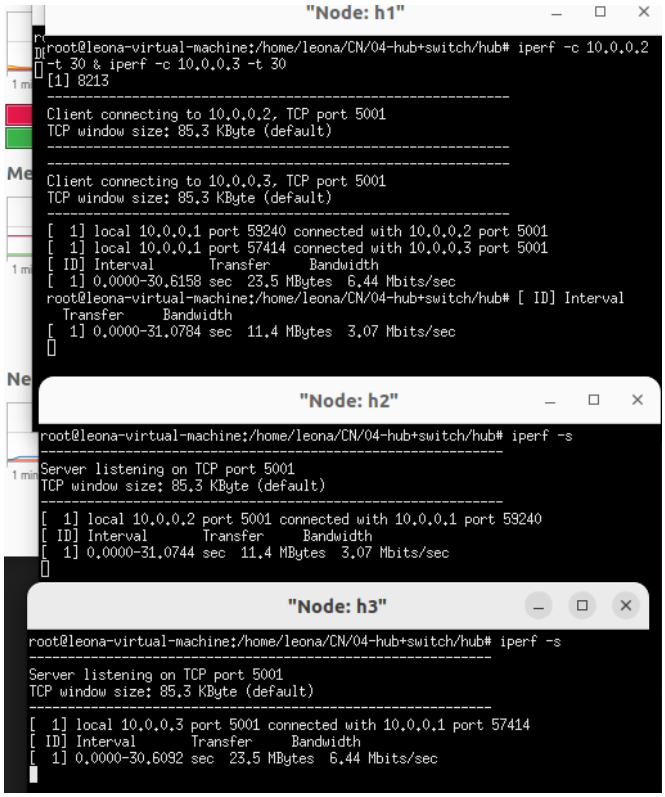
```
root@leona-virtual-machine:/home/leona/CN/04-hub+switch/hub# ping 10.0.0.2 -c 4
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.541 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.292 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.327 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=1.07 ms

--- 10.0.0.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3080ms
rtt min/avg/max/mdev = 0.292/0.556/1.065/0.308 ms
root@leona-virtual-machine:/home/leona/CN/04-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=0.477 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=0.331 ms
64 bytes from 10.0.0.1: icmp_seq=3 ttl=64 time=0.263 ms
64 bytes from 10.0.0.1: icmp_seq=4 ttl=64 time=0.266 ms

--- 10.0.0.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3061ms
rtt min/avg/max/mdev = 0.263/0.334/0.477/0.086 ms
root@leona-virtual-machine:/home/leona/CN/04-hub+switch/hub#
```

图表 2:h3 ping h1 h2

```
root@leona-virtual-machine:/home/leona/CN/04-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=0.355 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=0.232 ms
64 bytes from 10.0.0.1: icmp_seq=3 ttl=64 time=0.141 ms
64 bytes from 10.0.0.1: icmp_seq=4 ttl=64 time=0.198 ms

--- 10.0.0.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3074ms
rtt min/avg/max/mdev = 0.141/0.231/0.355/0.078 ms
root@leona-virtual-machine:/home/leona/CN/04-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=0.454 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=0.327 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=0.212 ms
64 bytes from 10.0.0.3: icmp_seq=4 ttl=64 time=0.109 ms

--- 10.0.0.3 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3067ms
rtt min/avg/max/mdev = 0.109/0.275/0.454/0.128 ms
root@leona-virtual-machine:/home/leona/CN/04-hub+switch/hub#
```

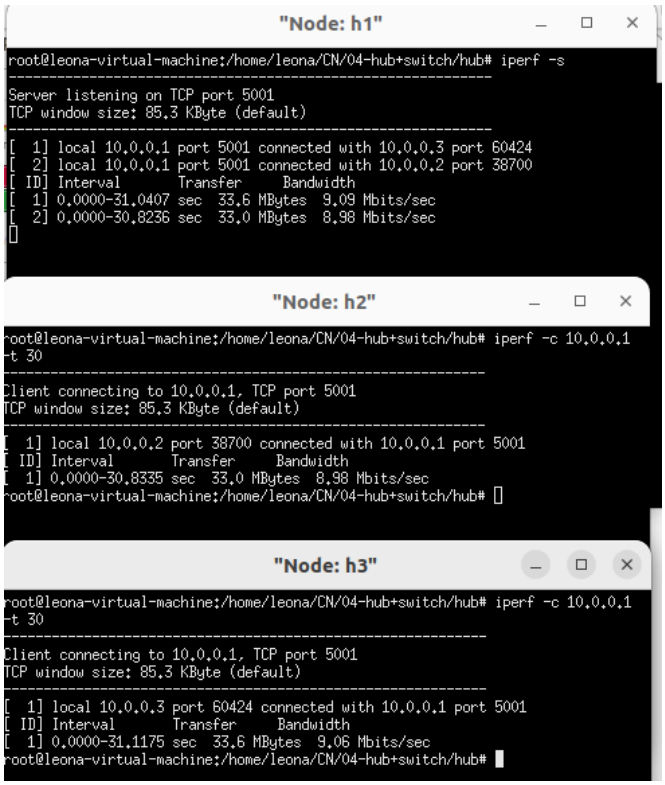
图表 3: h2 ping h1 h3

（3） iperf 测量



图表 4: h1 为 client、h2h3 为 server

以 h1 为 client、h2h3 为 server 执行 iperf，可以看出发送带宽分别为 3.07Mbps 和 6.44Mbps，接受带宽分别为 3.07Mbps 和 6.44Mbps，则带宽利用率为 47.55%。

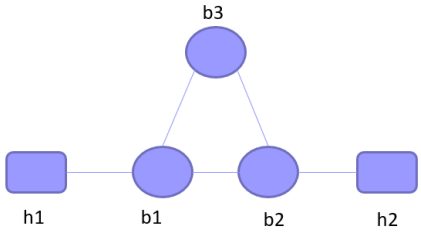


图表 5： h1 为 server、h2h3 为 client

以 h1 为 server、h2h3 为 client 时接受带宽分别为 8.98Mbps 和 9.06Mbps，则带宽利用率为 90.2%

h1 为 client 时，带宽利用率较低，这是由于 h2 和 h3 连接 b1 节点的带宽均为 10Mbps，所以 h1 向两个节点发送数据包的总带宽不能超过 10Mbps。而以 h1 为 server 时，带宽利用率较高，h2 和 h3 的发送不会相互影响，也就不会受到 hub 广播效应的影响。

（4） 构建环形拓扑，产生数据包环路



图表 6:环路示意图

如图所示需要建立五条连接，分别是 b1 和 b2 和 b2 互相的连接，b1 和 h1 的连接，b2 和 h2 的连接。

```
class BroadcastTopo(Topo):
    def build(self):
        h1 = self.addHost('h1')
        h2 = self.addHost('h2')
        b1 = self.addHost('b1')
        b2 = self.addHost('b2')
        b3 = self.addHost('b3')

        self.addLink(h1, b1, bw=20)
        self.addLink(h2, b2, bw=20)
        self.addLink(b1, b2, bw=20)
        self.addLink(b2, b3, bw=20)
        self.addLink(b3, b1, bw=20)
```

使用 wireshark 监视 h1 端口，在 h1 中执行 ping 指令，可以看出数据包在不断被广播。

265	0.026698263	26:9c:5d:0e:52:13	4e:23:0c:13:b2:03	ARP	42 10.0.0.2 is at 26:9c:5d:0e:52:13
266	0.026698701	26:9c:5d:0e:52:13	4e:23:0c:13:b2:03	ARP	42 10.0.0.2 is at 26:9c:5d:0e:52:13
267	0.026699069	26:9c:5d:0e:52:13	4e:23:0c:13:b2:03	ARP	42 10.0.0.2 is at 26:9c:5d:0e:52:13
268	0.026699525	26:9c:5d:0e:52:13	4e:23:0c:13:b2:03	ARP	42 10.0.0.2 is at 26:9c:5d:0e:52:13
269	0.026700081	26:9c:5d:0e:52:13	4e:23:0c:13:b2:03	ARP	42 10.0.0.2 is at 26:9c:5d:0e:52:13
270	0.026700556	26:9c:5d:0e:52:13	4e:23:0c:13:b2:03	ARP	42 10.0.0.2 is at 26:9c:5d:0e:52:13
271	0.026700971	26:9c:5d:0e:52:13	4e:23:0c:13:b2:03	ARP	42 10.0.0.2 is at 26:9c:5d:0e:52:13
272	0.026701336	26:9c:5d:0e:52:13	4e:23:0c:13:b2:03	ARP	42 10.0.0.2 is at 26:9c:5d:0e:52:13
273	0.026701727	26:9c:5d:0e:52:13	4e:23:0c:13:b2:03	ARP	42 10.0.0.2 is at 26:9c:5d:0e:52:13
274	0.026702167	26:9c:5d:0e:52:13	4e:23:0c:13:b2:03	ARP	42 10.0.0.2 is at 26:9c:5d:0e:52:13
275	0.026702516	26:9c:5d:0e:52:13	4e:23:0c:13:b2:03	ARP	42 10.0.0.2 is at 26:9c:5d:0e:52:13
276	0.026702833	26:9c:5d:0e:52:13	4e:23:0c:13:b2:03	ARP	42 10.0.0.2 is at 26:9c:5d:0e:52:13
277	0.026703358	26:9c:5d:0e:52:13	4e:23:0c:13:b2:03	ARP	42 10.0.0.2 is at 26:9c:5d:0e:52:13
278	0.026703789	26:9c:5d:0e:52:13	4e:23:0c:13:b2:03	ARP	42 10.0.0.2 is at 26:9c:5d:0e:52:13
279	0.026704241	26:9c:5d:0e:52:13	4e:23:0c:13:b2:03	ARP	42 10.0.0.2 is at 26:9c:5d:0e:52:13
280	0.026705045	26:9c:5d:0e:52:13	4e:23:0c:13:b2:03	ARP	42 10.0.0.2 is at 26:9c:5d:0e:52:13
281	0.026705514	26:9c:5d:0e:52:13	4e:23:0c:13:b2:03	ARP	42 10.0.0.2 is at 26:9c:5d:0e:52:13
282	0.026705976	26:9c:5d:0e:52:13	4e:23:0c:13:b2:03	ARP	42 10.0.0.2 is at 26:9c:5d:0e:52:13
283	0.026706411	26:9c:5d:0e:52:13	4e:23:0c:13:b2:03	ARP	42 10.0.0.2 is at 26:9c:5d:0e:52:13
284	0.026706807	26:9c:5d:0e:52:13	4e:23:0c:13:b2:03	ARP	42 10.0.0.2 is at 26:9c:5d:0e:52:13

图表 7：抓包

2、switch

（1） 实现 mac_port_max 相关操作

在转发表中查找对应的 mac 地址和 iface 映射的表项，若找到对应的表项，则返回对应的 iface。同时，由于存在另一个线程会进行超时表项的清理工作，因此在

查找操作时需要上锁（这个操作在插入和老化检查的函数中也需要）。

```
iface_info_t *lookup_port(u8 mac[ETH_ALEN])
{
    u8 hash_value = hash8((char*)mac, ETH_ALEN);
    mac_port_entry_t *ptr = NULL;
    pthread_mutex_lock(&mac_port_map.lock);
    list_for_each_entry(ptr, &mac_port_map.hash_table[hash_value], list) {
        if (memcmp(ptr -> mac, mac, ETH_ALEN) == 0) {
            pthread_mutex_unlock(&mac_port_map.lock);
            return ptr -> iface;
        }
    }
    pthread_mutex_unlock(&mac_port_map.lock);
    return NULL;
}
```

当转发表中没有源 mac 地址和对应 iface 的映射表项时，将该地址与 iface 插入到转发表中。

```
void insert_mac_port(u8 mac[ETH_ALEN], iface_info_t *iface)
{
    u8 hash_value = hash8((char*)mac, ETH_ALEN);
    mac_port_entry_t *ptr = NULL;

    pthread_mutex_lock(&mac_port_map.lock);

    list_for_each_entry(ptr, &mac_port_map.hash_table[hash_value], list) {
        if (memcmp(ptr -> mac, mac, ETH_ALEN) == 0) {
            ptr -> iface = iface;
            ptr -> visited = time(NULL);
            pthread_mutex_unlock(&mac_port_map.lock);
            return;
        }
    }
    // miss
    mac_port_entry_t *new_entry = (mac_port_entry_t *)malloc(sizeof(mac_port_entry_t));
    new_entry -> iface = iface;
    memcpy(new_entry -> mac, mac, ETH_ALEN);
    new_entry -> visited = time(NULL);
    list_add_head(&new_entry -> list, &mac_port_map.hash_table[hash_value]);

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

当转发表中的表项超过 30s 没有被查询，则删除该表项。

```
int sweep_aged_mac_port_entry()
{
```

```
    int num_removed_entry = 0;

    pthread_mutex_lock(&mac_port_map.lock);

    for (int i = 0; i < HASH_8BITS; i += 1) {
        mac_port_entry_t *ptr = NULL;
        mac_port_entry_t *nxt = NULL;
        list_for_each_entry_safe(ptr, nxt, &mac_port_map.hash_table[i], list) {
            if (time(NULL) - ptr -> visited > MAC_PORT_TIMEOUT) {
                list_delete_entry(&ptr -> list);
                num_removed_entry += 1;
                free(ptr);
            }
        }
    }

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

（2） 实现数据包处理逻辑

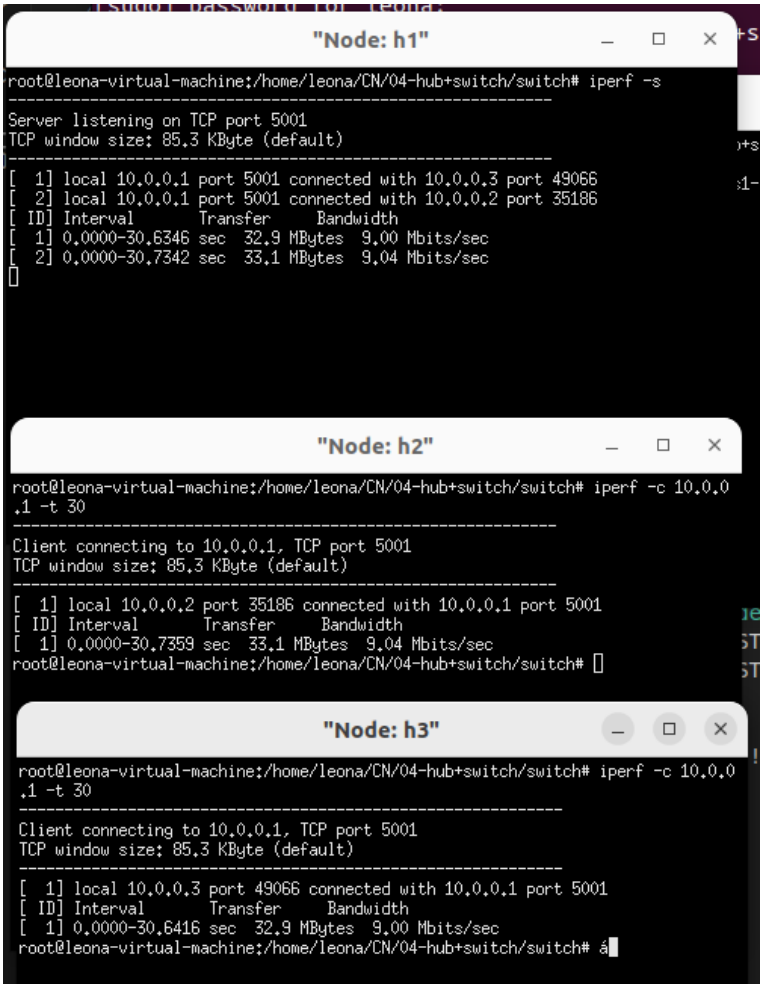
首先调用 look_up_port 函数检查目的 mac 与端口的映射是否在表中，若存在，则根据表项发包，若没有则进行广播（广播函数即与之前相同）。同时调用 insert_mac_port 函数插入表中。

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

    iface_info_t *port = NULL;
    if ((port = lookup_port(eh -> ether_dhost)) != NULL) {
        iface_send_packet(port, packet, len);
      }
      else {
        broadcast_packet(iface, packet, len);
    }

    insert_mac_port(eh -> ether_shost, iface);

    free(packet);
}
```

（3） 测量

图表 8：h1 为 client、h2h3 为 server

以 h1 为 client、h2h3 为 server 执行 iperf，可以看出发送带宽分别为 9.27Mbps 和 9.54Mbps，接受带宽分别为 9.27Mbps 和 9.54Mbps，则带宽利用率为 94.05%。



图表 9：h1 为 server、h2h3 为 client

以 h1 为 server、h2h3 为 client 时接受带宽分别为 9.04Mbps 和 9.00Mbps，则带宽利用率为 90.2%

3、对比交换机转发与集线器广播的性能

可以看出当 h1 作为 client 时，带宽利用率相差很大，在交换机中利用率接近百分之百。这是由于广播节点对于 h1 和 h2 的总带宽有限制，因为所有数据包都会向所有节点发出。而交换机转发是只向目的接口转发，因此不必受限。

当 h1 作为 server 时，带宽利用率相差无几。这是由于在广播中对于带宽没有限制，因此不像 h1 作为 client 时会极大降低带宽利用率。

总体来说，交换机转发性能远远高于集线器广播性能。交换机在数据链路层工作，能够快速执行地址查找和转发，速度较快。集线器在物理层工作，没有地址选择，会广播到所有接口，因此会限制网络性能。