**交换机网络实验**

2021K8009929006

范子墨

1. **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

1. **设计思路及验证**
2. hub
3. 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);

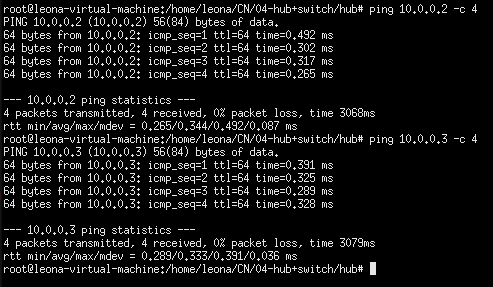
        }

    }

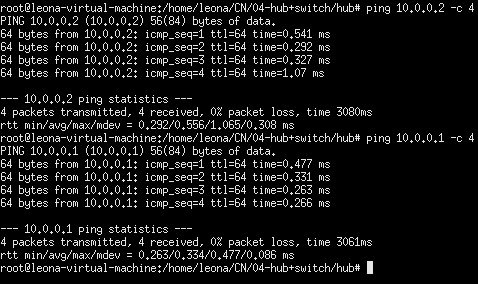
}

1. ping通

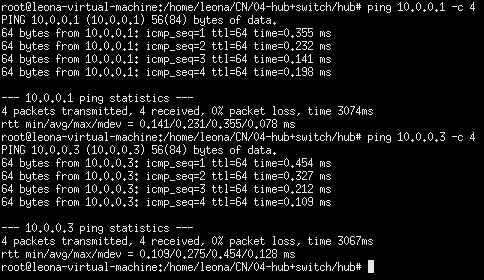
10.0.0.1是h1节点，10.0.0.2是h2节点，10.0.0.3是h3节点，三个节点互相执行ping指令，可以看到三个节点能够相互ping通



图表 1：h1 ping h2 h3

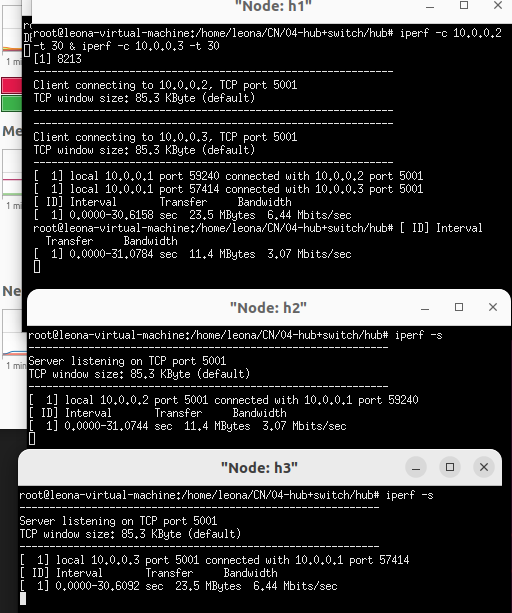


图表 2:h3 ping h1 h2



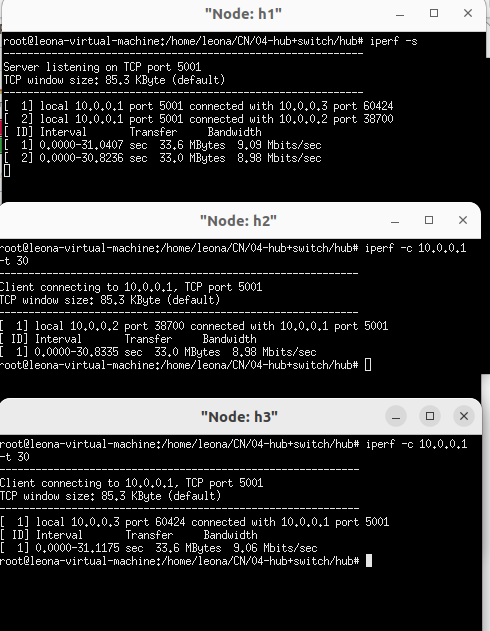
图表 3: h2 ping h1 h3

1. 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%。

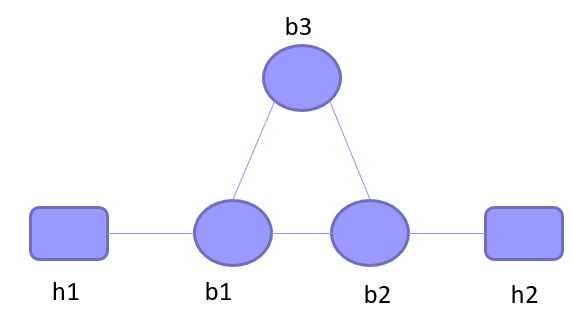


图表 5： h1为server、h2h3为client

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。

1. 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)

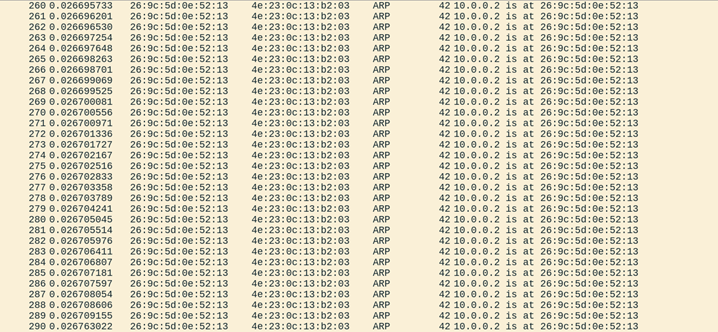
        self.addLink(h2, b2, bw=20)

        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。



图表 7：抓包

1. switch
2. 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;

}

1. 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);

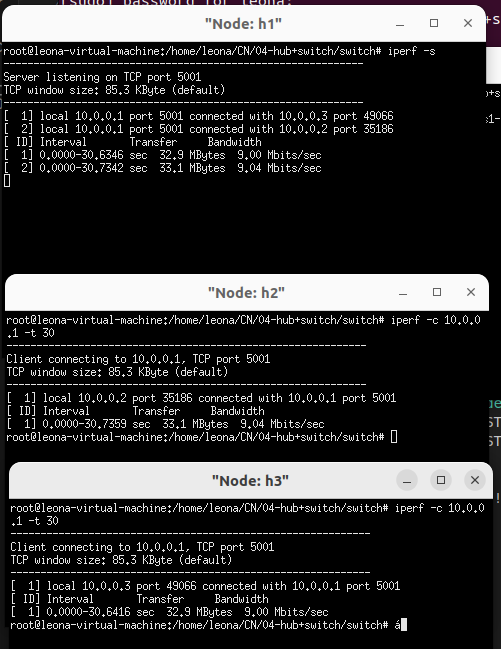
}

1. 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。