交换机转发 实验设计报告

中国科学院大学

页

2022年4月9日

一、广播网络实验

# 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。

# 实验流程

1. 实现节点广播的 broadcast\_packet 函数，调⽤list\_for\_each\_entry，对整个链表进⾏遍历当前主机不是发送消息的主机则调⽤iface\_send\_packet，发包给这个主机。

void broadcast\_packet(iface\_info\_t \*iface, const char \*packet, int len)

{

*// TODO: broadcast packet*

    fprintf(stdout, "TODO: broadcast packet.\n");

    iface\_info\_t \*iface\_n = NULL;

    list\_for\_each\_entry(iface\_n, &instance->iface\_list, list) {

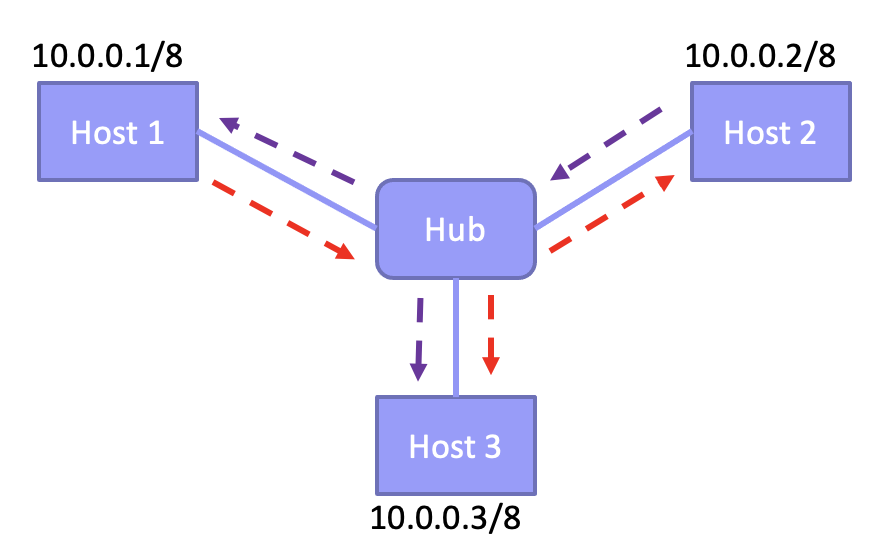
*if* (iface\_n->fd != iface->fd)

            iface\_send\_packet(iface\_n, packet, len);

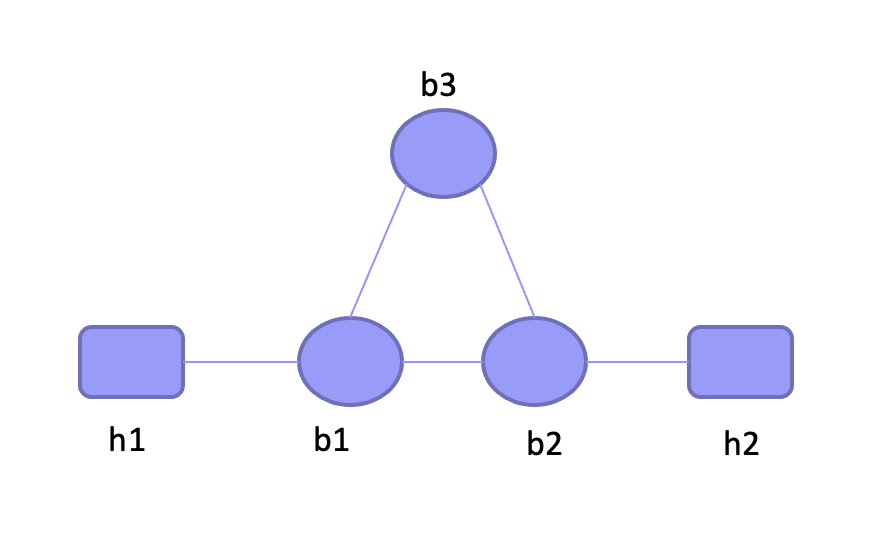
    }

}

1. 编译 hub 程序运⾏程序 three\_node\_bw.py，开启 h1、h2、h3 和b1四个结点。通过从 h1 ping h2 和h3 ，从 h2 ping h1 和 h3 ，从 h3 ping h1 和 h2 ，验证这些数据通路是否都是都是连通的。
2. 进行⼴播⽹络效率测试，运⾏ three\_node\_bw.py，开启h1、h2、h3 和b1四个结点。先⽤ h1作为服务器，h2和 h3作为客户进⾏访问。再⽤ h2 和 h3 作为服务器， h1 作为客户进⾏访问。



**图1：3结点拓扑**

****

**图2：环形拓扑**

1. 修改拓扑结构，验证环形拓扑下数据报形成环路。参照图1和图2的拓扑结构以及源程序three\_node\_bw.py编写环形拓扑topo.py，。运⾏ topo.py ，开启 h1、 h2、 b1、 b2和b3五个结点。 b1、b2 和 b3 作为hub，然后⽤ h1 发送⼀个数据包。然后在 h2中打开 wireshark 抓包。

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)

self.addLink(h2, b2)

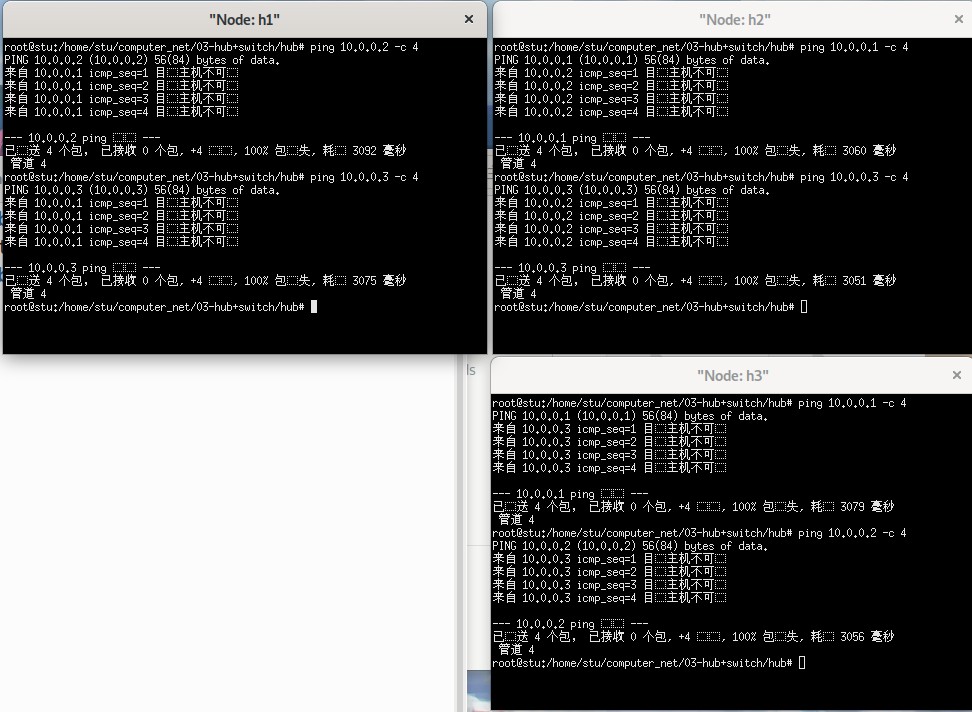
self.addLink(b1, b2)

self.addLink(b2, b3)

self.addLink(b1, b3)

# 实验结果与分析

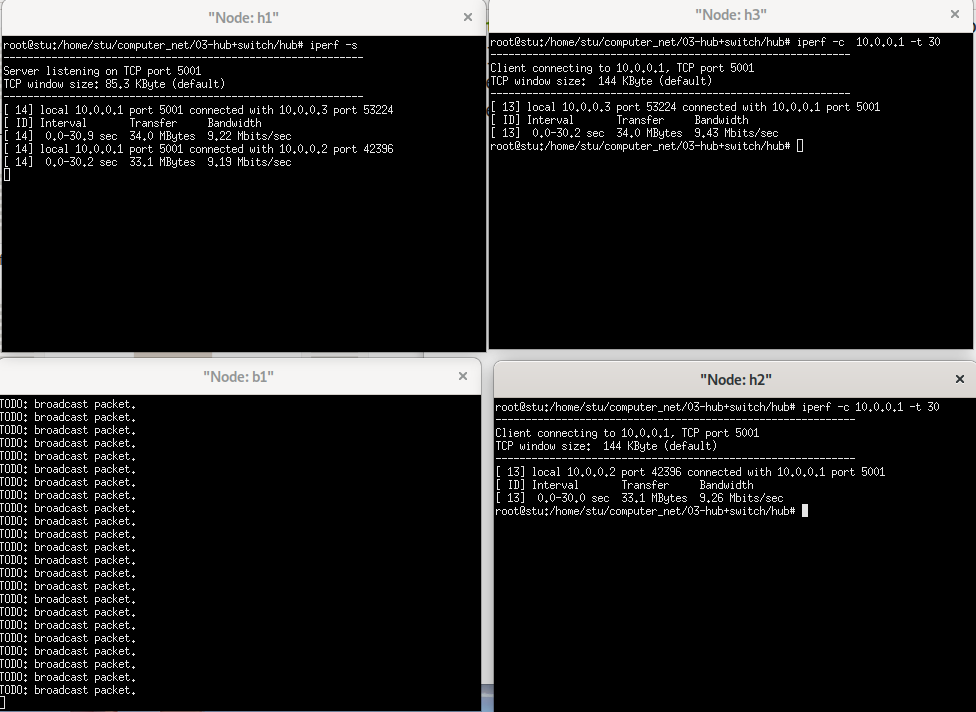
（1）广播网络功能测试(ping)



**图3：广播网络ping测试**

测试结果如图 3。结果显示，各 host 节点之间连接正常，hub可以完成正常广播的功能。

（2）⼴播⽹络效率测试

① h1作为主机，h2 和 h3 同时向 h1 测量，结果如图4所示。

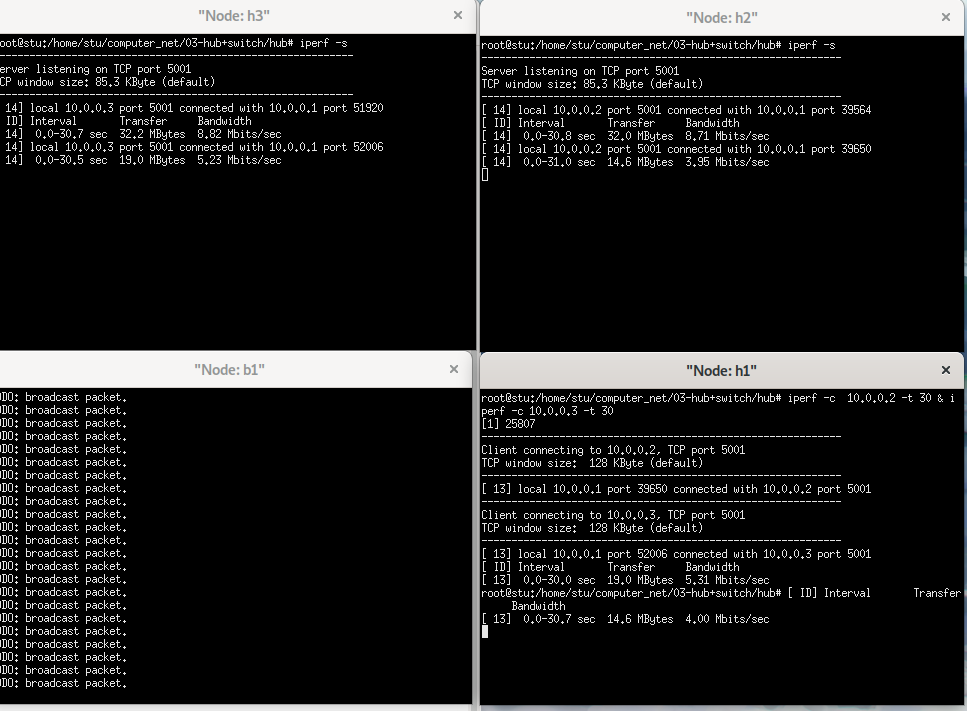
**图4：h1作主机iperf测试**

h1 处的吞吐量为 9.22+9.19 = 18.41Mbps，带宽利用率为 90.21%。h2 处的吞吐量为 9.43Mbps，利用率为 94.3%，h3 处的吞吐量为 9.26Mbps，利用率 92.6%，双向带宽利用都较高。

②h2和h3作主机， h1 同时向 h2 和 h3 测量，结果如图5所示。

h1 处的吞吐量为 5.31+4.00 = 9.31Mbps，（单向）带宽利用率为 46.55%。h2 处的吞

吐量为 5.23Mbps，利用率为 52.3%，h3 处的吞吐量为 3.95Mbps，利用率 39.5%。带宽利用率很低，从图中我们可以看到h2和h3除了收到自己的包以外还收到了对方的包，这说明当 h1 同时向 h2 和 h3 发包时，在 b1 处包会被复制，在b1->h2 有 h1 发往 h3 的包，在 b1->h3 有 b1 发往 h2 的包，这些冗余包浪费了带宽，因而带宽利用率很低。在情况①的时候h2->b1和h3->b2是独立的，因此带宽受影响很小，但是仍然达不到100%这是由于数据链路层会把数据包额外加上一些信息，这些信息占用的带宽不计入带宽计算中。



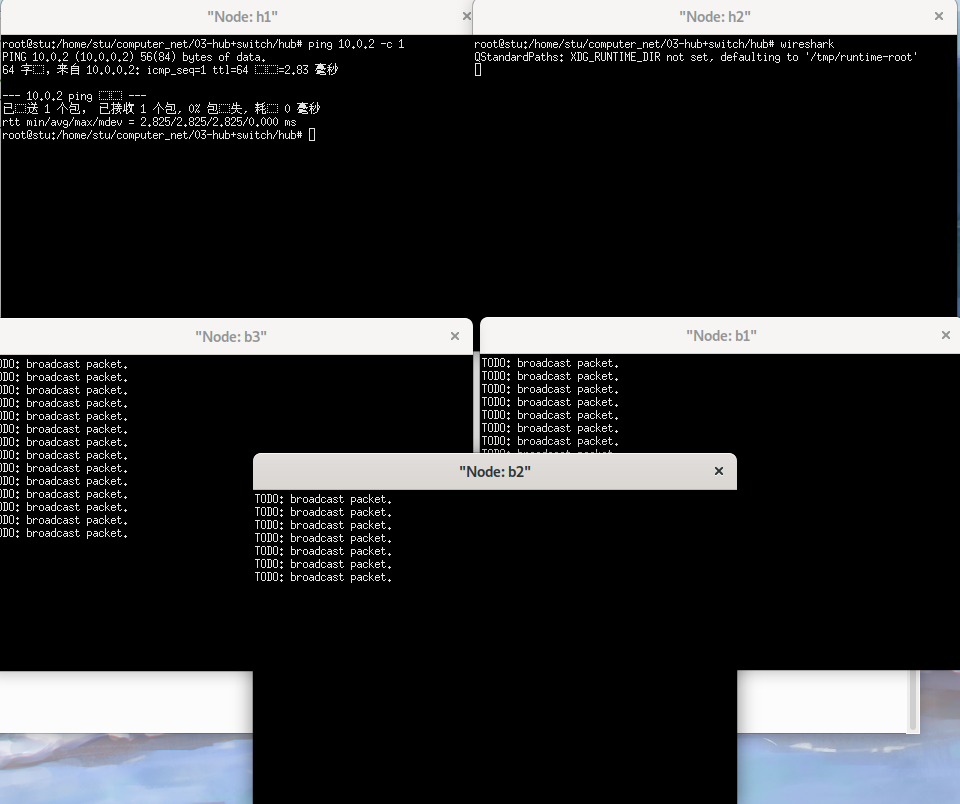
**图5：h2和h3作主机iperf**

（3）环路广播测试

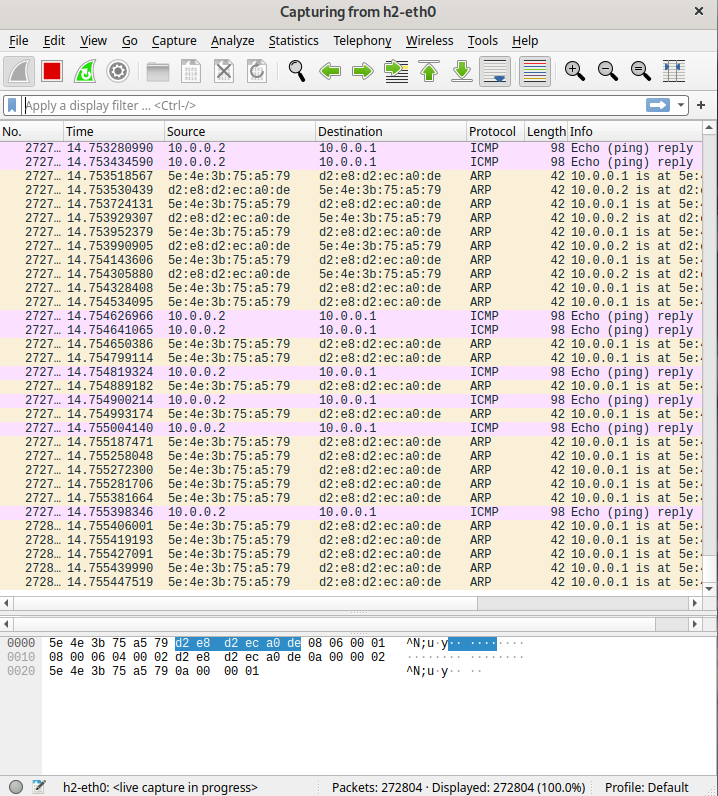
如图6所示，h1 向 h2 发出 ping 命令后，各个转发节点的 hub 程序输出如图 10 所示，说明环路中包被不断复制转发。当 h1 发送数据包后，， b1、 b2和 b3都在不停打印⼴播的信息，同时从主机h2的wireshark 如图7 中抓包也可得到一直被转发的同样的数据包，这说明数据包在环形拓扑中被不停地转发，形成了广播风暴。

广播风暴(broadcast storm)简单的讲是指当广播数据充斥网络无法处理，并占用大量网络带宽，导致正常业务不能运行，甚至彻底瘫痪，这就发生了“广播风暴”。一个数据帧或包被传输到本地网段 (由广播域定义)上的每个节点就是广播;由于网络拓扑的设计和连接问题，或其他原因导致广播在网段内大量复制，传播数据帧，导致网络性能下降，甚至网络瘫痪。

广播风暴的产生有多种原因，如蠕虫病毒、交换机端口故障、网卡故障、链路冗余没有启用生成树协议、网线线序错误或受到干扰等。从目前来看，蠕虫病毒和arp攻击是造成网络广播风暴最主要的原因。虽说如今网络广播风暴已经很少见了，但在一些使用集线器的网络中仍然非常常见。解决网络广播风暴最快捷的方法是给集线器断电然后上电启动即可，但这只是治标不治本的方法，要彻底解决，最好使用交换机设备，并划分vlan、通过端口控制网络广播风暴。否则，如果广播风暴是由于网卡损坏所致，要从上百台计算机找出故障计算机是十分困难的事情。



**图6：环路拓扑测试结果**



**图7：wireshark抓包环路拓扑**

1. 交换机转发实验

# 实验内容

（1）实现对数据结构 mac\_port\_map 的所有操作，包括查询，插入和老化，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。

# 实验流程

1. 理解现有的代码框架，补充完善boradcast\_packet.c中的boradcast\_packet函数，main.c中的handle\_packet函数，mac.c中的lookup\_port、insert\_mac\_port和sweep\_aged\_mac\_port\_entry函数。
2. make编译生成switch文件。
3. 执行命令生成h1、h2、h3三给主机和一个s1交换机结点，在h2、h3服务端，从h1发送数据包测试性能。

sudo python three\_nodes\_bw.py

mininet> xterm h1 h2 h3 s1

s1# ./switch

h2# iperf -s

h3# iperf -s

h1# iperf -c 10.0.0.2 -t 30 & iperf-c 10.0.0.3 -t 30

1. 观察实验结果，完成实验报告。

# 实验函数设计

1. 数据包处理

交换机收到数据包后调⽤lookup查找转发表中是否有对应的条⽬，如果有对应的条⽬从记录的端⼝中转发出去，否则⼴播此数据包。如果没有对应的的条目根据这个数据包的源和⼊⼝，修改转发表

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);

    }

    insert\_mac\_port(eh->ether\_shost, iface);

    free(packet);

}

1. 查询：收到数据包时，根据⽬的MAC地址查询转发表，如果查询到则从该端⼝转发，否则进⾏⼴播。函数首先获得转发表的互斥锁，计算当前mac地址对应的hash值，然后利用list\_for\_each\_entry()和memcmp()函数查找当前hash值对应的转发链表中是否有对应的条⽬，找到后占用的互斥锁。

iface\_info\_t \*lookup\_port(u8 mac[ETH\_ALEN])

{

*// TODO: implement the lookup process here*

*// fprintf(stdout, "TODO: implement the lookup process here.\n");*

    mac\_port\_entry\_t \* mac\_entry = NULL;

    u8 mac\_hash = hash8((void \*) mac, ETH\_ALEN);

    pthread\_mutex\_lock(&mac\_port\_map.lock);

    list\_for\_each\_entry(mac\_entry, &mac\_port\_map.hash\_table[mac\_hash], list) {

*if* (memcmp(mac\_entry->mac, mac, ETH\_ALEN) == 0) {

            pthread\_mutex\_unlock(&mac\_port\_map.lock);

*return* mac\_entry->iface;

        }

    }

    pthread\_mutex\_unlock(&mac\_port\_map.lock);

*return* NULL;

}

1. 插⼊：收到数据包时，可以获得收到该数据包的端⼝和数据包的源MAC地址，更新转发表中的条⽬。首先便利查找该mac对应的hash表中是否有对应条目，如果有的话修改信息和访问时间。没有对应的条目的话创建新的结点写入信息并将对应的结点插入到表中。

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 \* mac\_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(mac\_entry, &mac\_port\_map.hash\_table[mac\_hash], list) {

*if* (memcmp(mac\_entry->mac, mac, ETH\_ALEN) == 0) {

            mac\_entry->iface = iface;

            mac\_entry->visited = now;

            pthread\_mutex\_unlock(&mac\_port\_map.lock);

*return*;

        }

    }

    mac\_entry = malloc(sizeof(mac\_port\_entry\_t));

    memcpy(mac\_entry->mac, mac, ETH\_ALEN);

    mac\_entry->iface = iface;

    mac\_entry->visited = now;

    list\_add\_head(&mac\_entry->list, &mac\_port\_map.hash\_table[mac\_hash]);

    pthread\_mutex\_unlock(&mac\_port\_map.lock);

}

1. ⽼化：删除30s内未访问的条⽬，提⾼查表的效率。遍历hash转发表，如果发现超时的结点就删除对应的条目，释放相应的空间。

int sweep\_aged\_mac\_port\_entry()

{

*// TODO: implement the sweeping process here*

*//fprintf(stdout, "TODO: implement the sweeping process here.\n");*

    mac\_port\_entry\_t \*mac\_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(mac\_entry, next\_entry, &mac\_port\_map.hash\_table[i], list) {

*if* ((int)(now - mac\_entry->visited) > MAC\_PORT\_TIMEOUT) {

                list\_delete\_entry(&mac\_entry->list);

                free(mac\_entry);

            }

        }

    }

    pthread\_mutex\_unlock(&mac\_port\_map.lock);

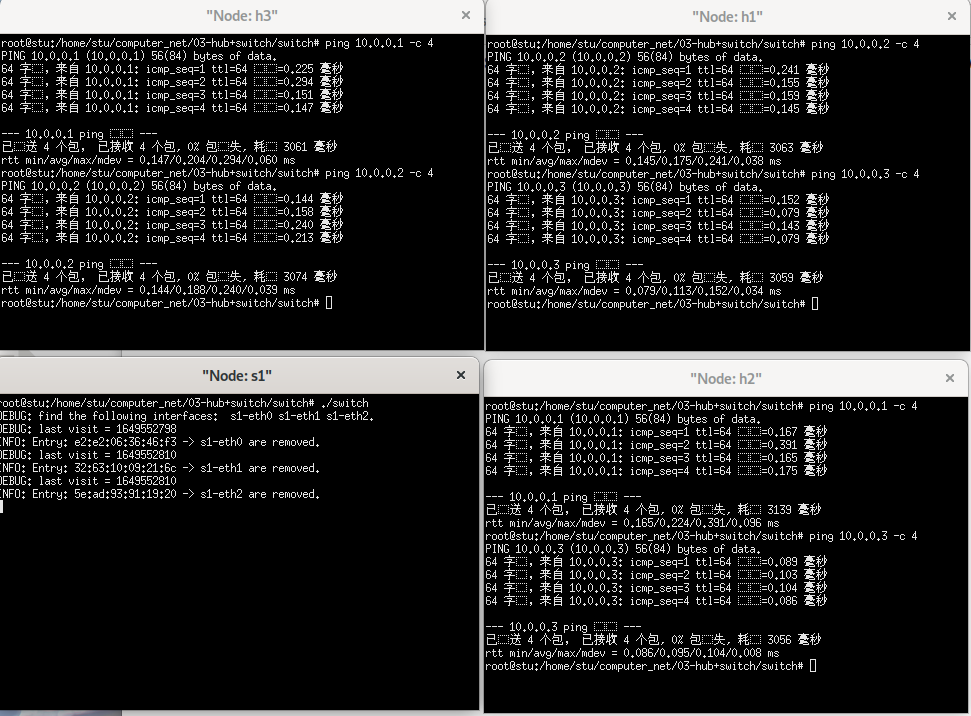
*return* 0;

}

# 实验结果与分析

1. 广播功能测试（ping）

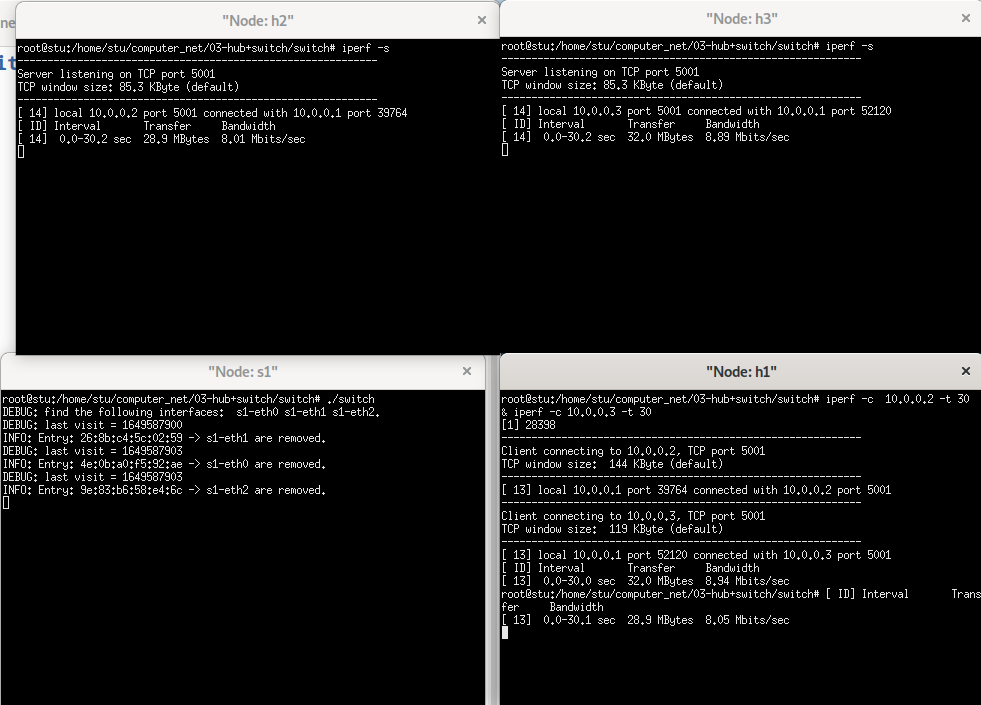
如图8所示，运⾏ three\_node\_bw.py ，开启h1、h2、h3 和s1四个结点。在s1打开switch,从h1 ping h2 和h3 ，从 h2 ping h1和h3 ，从h3 ping h1和 h2 ，可以发现这些数据通路都是ping通的，这说明实现的 switch 可以正常完成转发的功能。



**图8：交换机ping测试结果**

1. 广播性能效率测试（iperf）

运行three\_node\_bw.py，开启h1、h2、h3和s1四个结点，用h3和h3作为服务器，h1作为客户机同时访问h2和h3。



**图9：交换机广播性能测试结果**

如图9测试结果显示：h1->h2 的实测带宽为 8.94Mbps，h1->h3 的实测带宽为 8.06Mbps。带宽利用率平均在85.0%，h2的处的吞吐量为8.01Mbps，h3处的为8.89，而在hub同样实验实验结果为h1 处的吞吐量为 5.31+4.00 = 9.31Mbps，带宽利用率为 46.55%。h2 处的吞吐量为 5.23Mbps，利用率为 52.3%，h3 处的吞吐量为 3.95Mbps，利用率 39.5%。带宽上涨100%，这是因为相比hub的复制发送switch 只向目的端口转发数据包而不再向所有端口转发数据包，避免了网络拓扑中的冗余包占用带宽。

与此同时我们可以从图9的s1节点发现switch的查询，插入和老化等功能正常运行。

1. 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？）

**答：**

目的地址不需要更新，老化操作可以把过期的目的地址删除。如果更新目的地址，当目的MAC 地址变更时，原来地址的老化时间可能一直被更新，从而无法被老化删除。考虑⼀种情况：假如结点 h1 ⼀直不停地向结点 h2 发送数据包，这个过程中结点 h2更换了MAC地址，那么交换机就⽆法通过转发表中记录的⽹⼝将数据包发送给 h2 ，但是由于h1 ⼀直在查询 h2 的 MAC 地址，所以转发表中的条⽬就⼀直不会被⽼化删除，最终这个数据包⼀致⽆法送达。所以不能在查询⽬的 MAC 地址时更新⽼化时间。

（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？

**答：**

这种⼴播包不会影响交换表转发逻辑。因全 0xFF 地址只会作为目的地址出现而不会作为源地址出现，将会被广播，不会更新交换机转发表。

（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

**答：**

不⾏。⾸先，所有终端构成的⽹络中势必存在很多环路，需要⽤⽣成树算法去处理这些环路，两个节点之间只能有一条路径。当节点数量足够多时，网络中的一台交换机需要处理大量结点的请求，转发表需要为所有地址进行学习，交换机的计算和存储能力有限，达不到要求。其次，为了转发表中的信息不被很快刷新，转发表中需要记录⼤量的条⽬，查找时非常慢，效率反⽽会降低。

从交换机的连接方式上来说，单独一台交换机的端口数量是有限的，不足以满足网络终端设备接入网络的需求。为此我们需要使用多台交换机来提供终端接入功能，并将多台交换机互连，形成一个局域网。交换机的互连主要有级联(Uplink)和堆叠(Stack)两种方式。当交换机级联层数较多时，层次之间存在较大的收敛比，将出现较大的延时。超过一定数量的交换机进行级联，最终会因此网络广播风暴，导致网络性能严重下降，甚至瘫痪。因此交换机不应该无限制级联。而堆叠方式的数目也是有限的，一般最多允许对8台交换机进行堆叠，并且要求堆叠成员离自己的位置足够近，堆叠线缆最长只有几米，一般堆叠的交换机处于同一个机柜中，这种方式难以远程连接。

同时将所有终端连接在⼀起会降低⽹络⼊侵的难度，⽹络安全隐患很大

**参考资料：**

1. 广播风暴：<https://www.21ic.com/article/702749.html>。
2. 交换机的互联技术：<https://www.cnblogs.com/gzxbkk/p/7910611.html>。