交换机转发实验

学号： 2021K8009929010

姓名： 贾城昊

实验任务一（hub）

一、 实验任务：

了解广播网络的原理,实现节点广播的 broadcast_packet 函数。验证广播网络能够正常运行,并通过 iperf 测试广播网络的效率,掌握其运行特点。最后构建环形拓扑网络,验证该拓扑下节点广播会产生数据包环路。

二、 实验流程

1. 根据广播网络的原理,实现节点广播的 broadcast_packet 函数。

2. 测试三个节点互相连通,验证广播网络能正常运行。

3. 验证广播网络效率,并对结果进行解释。

   1) 在 three_nodes_bw.py 进行 iperf 测量

   2) 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. 构建环形拓扑网络,验证该拓扑下节点广播会产生数据包环路。

三、 实验结果与分析

（一）实现节点广播

1. 广播节点设计思路

广播节点的逻辑较为简单，即每次收到网络包消息时，遍历与之相邻的每个网络端口，只要不是发送该网络包的端口，就将网络包广播到这个端口。

而上述遍历过程可以通过现成的链表操作实现，具体代码如下：

```c
#include "base.h"
#include <stdio.h>

extern ustack_t *instance;

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_entry;
    list_for_each_entry(iface_entry, &instance -> iface_list, list) {
        if (iface_entry -> fd != iface -> fd)
            iface_send_packet(iface_entry, packet, len);
    }
}
```

2. 结果验证

三个节点各自向其它两个节点发送消息，验证其两两相互连通。

h1 节点的验证结果如下：

```
root@Computer:~/workspace/Network/lab4/04-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=0.110 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.058 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.059 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.069 ms

--- 10.0.0.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2999ms
rtt min/avg/max/mdev = 0.058/0.074/0.110/0.021 ms
root@Computer:~/workspace/Network/lab4/04-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=0.142 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=0.046 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=0.053 ms
64 bytes from 10.0.0.3: icmp_seq=4 ttl=64 time=0.061 ms

--- 10.0.0.3 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2997ms
rtt min/avg/max/mdev = 0.046/0.075/0.142/0.039 ms
```

可以看到，h1 和另外两个节点连通。

h2 节点的验证结果如下：

```
root@Computer:~/workspace/Network/lab4/04-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=0.117 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=0.051 ms
64 bytes from 10.0.0.1: icmp_seq=3 ttl=64 time=0.053 ms
64 bytes from 10.0.0.1: icmp_seq=4 ttl=64 time=0.070 ms

--- 10.0.0.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2998ms
rtt min/avg/max/mdev = 0.051/0.072/0.117/0.028 ms
root@Computer:~/workspace/Network/lab4/04-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=0.146 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=0.051 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=0.055 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 2997ms
rtt min/avg/max/mdev = 0.051/0.078/0.146/0.040 ms
```

可以看到，h2 和另外两个节点连通。

h3 节点的验证结果如下：

```
root@Computer:~/workspace/Network/lab4/04-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=0.114 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=0.048 ms
64 bytes from 10.0.0.1: icmp_seq=3 ttl=64 time=0.054 ms
64 bytes from 10.0.0.1: icmp_seq=4 ttl=64 time=0.059 ms

--- 10.0.0.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2997ms
rtt min/avg/max/mdev = 0.048/0.068/0.114/0.028 ms
root@Computer:~/workspace/Network/lab4/04-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=0.119 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.053 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.053 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.063 ms

--- 10.0.0.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2998ms
rtt min/avg/max/mdev = 0.053/0.072/0.119/0.027 ms
```

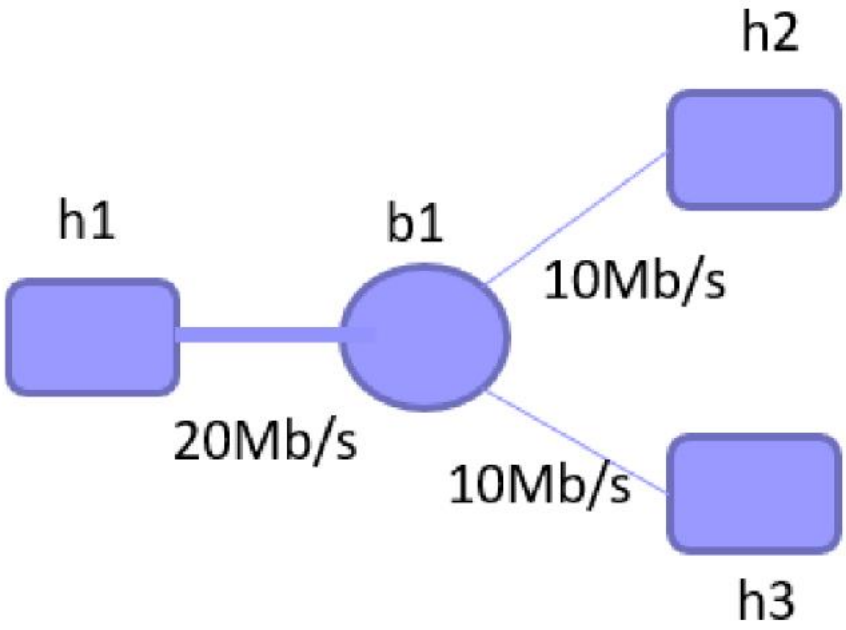
可以看到，h3 和另外两个节点连通。

综上，三个节点两两连通，广播网络能够正常运行

（二）广播网络传输效率

1. h1 向 h2 和 h3 同时传输

网络的拓扑结构如下所示：



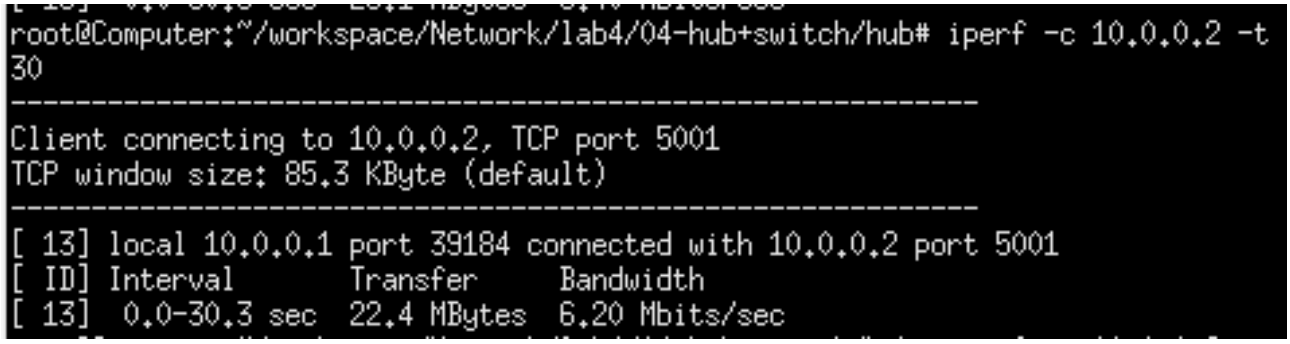
h1 同时向 h2 和 h3 发送数据，测试传输速率（一共做了三次实验）：

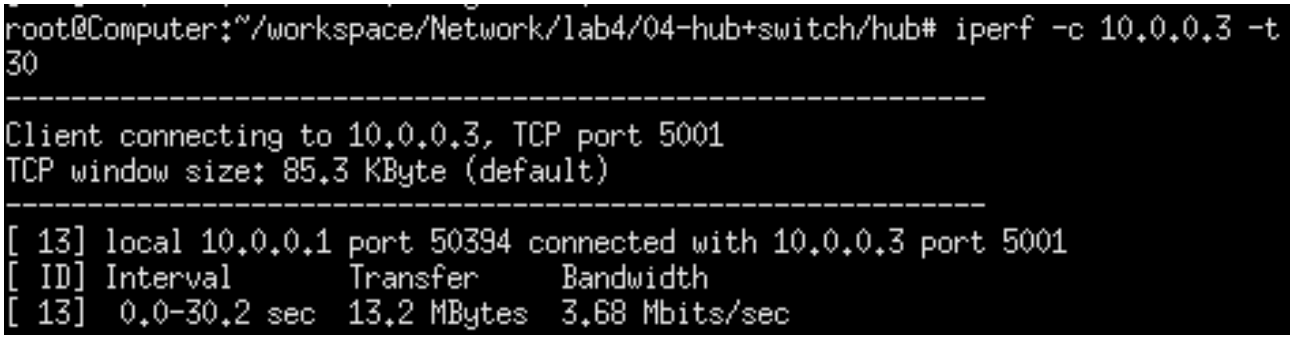
第一次实验结果如下：

```
root@Computer:~/workspace/Network/lab4/04-hub+switch/hub# iperf -c 10.0.0.2 -t
30
------------------------------------------------------------
Client connecting to 10.0.0.2, TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[ 13] local 10.0.0.1 port 39180 connected with 10.0.0.2 port 5001
[ ID] Interval       Transfer     Bandwidth
[ 13]  0.0-30.3 sec  23.1 MBytes  6.40 Mbits/sec
```

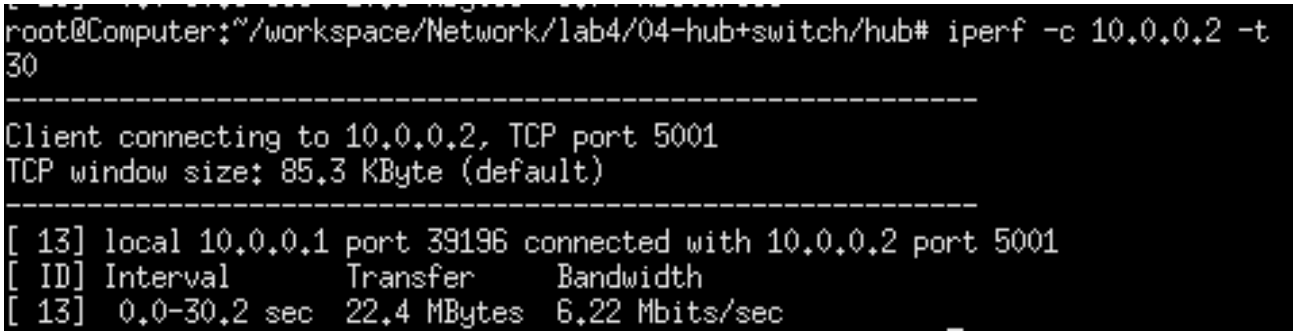
```
root@Computer:~/workspace/Network/lab4/04-hub+switch/hub# iperf -c 10.0.0.3 -t
30
------------------------------------------------------------
Client connecting to 10.0.0.3, TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[ 13] local 10.0.0.1 port 50390 connected with 10.0.0.3 port 5001
[ ID] Interval       Transfer     Bandwidth
[ 13]  0.0-30.1 sec  11.9 MBytes  3.31 Mbits/sec
```

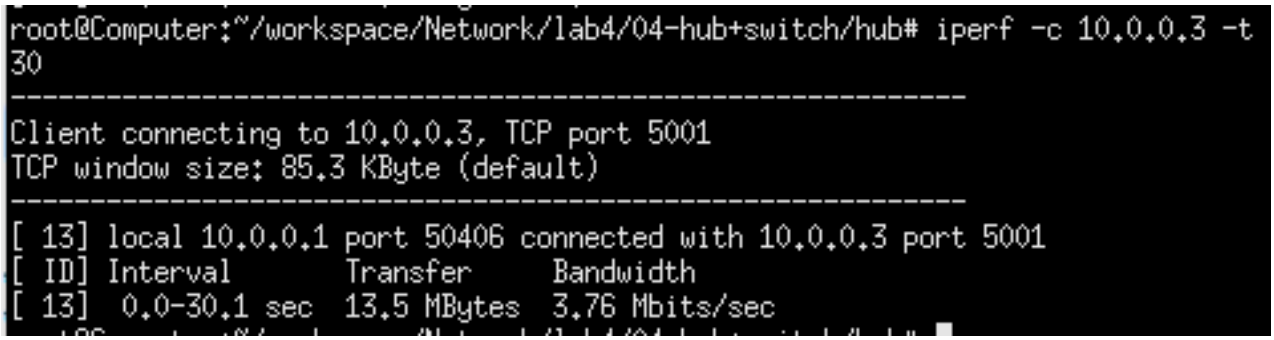
第二次实验结果如下：





第三次实验结果如下：





从上面的几张图我们可以看出，h1 到 h2 实际传输速率和 h1 到 h3 实际传输速率都小于 h2/h3 到 b1 的带宽（10.0Mb/s）。同时我们也能注意到两者速率加起来差不多正好达到 b1 到 h2/h3 的带宽（10.0Mb/s）。

这是由于 h1 发给 h2 的数据在 b1，会同时广播给 h2 和 h3，这样给 h2 的数据

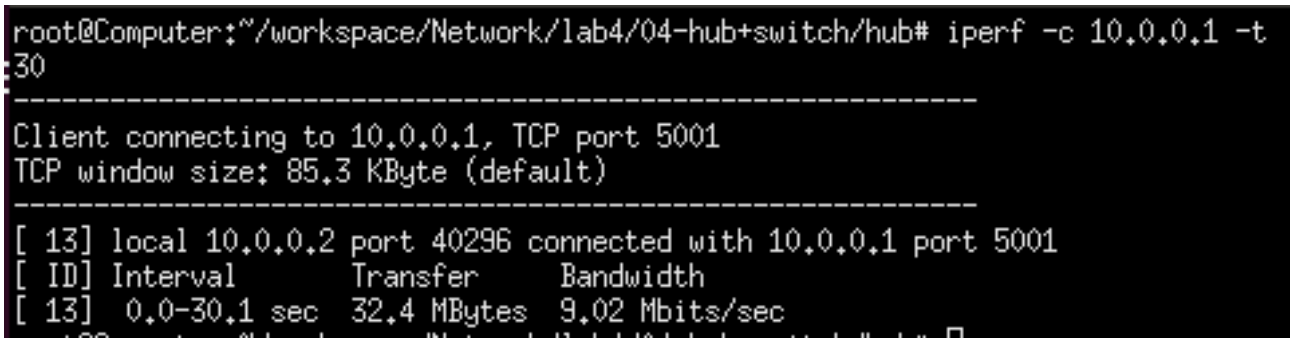
也会占据 h3 的传输带宽。同理，h1 发给 h3 的数据也会占据 b1 到 h2 的传输带宽。于是 b1 到 h2 和 b1 到 h3 两条传输通路都会传输 h1 发送给 h2 和 h3 的全部数据。因此 h1 到 h2 的传输速率与 h1 到 h3 的传输速率都会小于 b1 到 h2/h3 的带宽 10Mb/s。

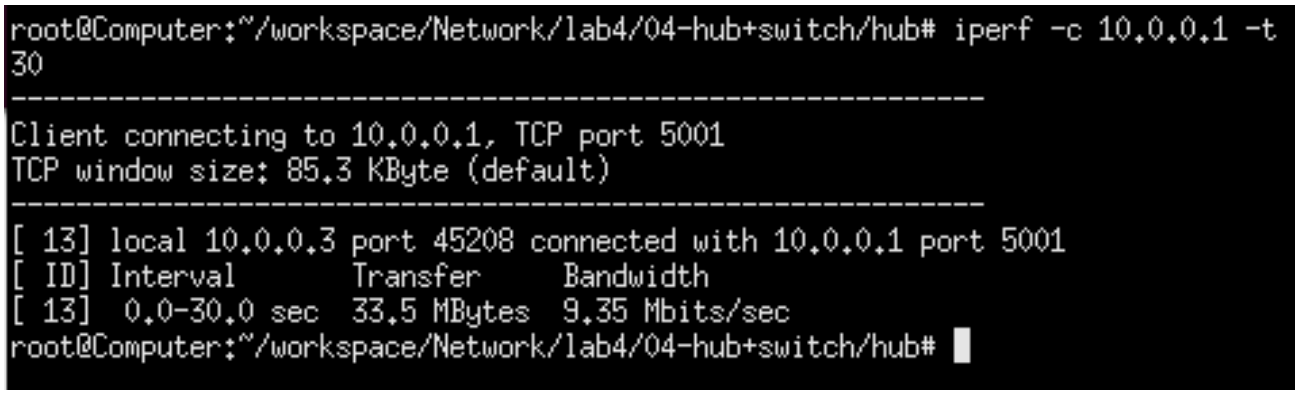
理论上来说，b1 到 h2 和 b1 到 h3 两条通路的效率应都为 50%左右。但实际中 h1 到 h2 的速率与 hl 到 h3 的速率有些差异，本人认为这是受到了先后启动的影响，测试进程先启动的一方 TCP 窗口更大，速率会略大一些。但无论如何两者速率之和上限只有 10Mb/s，传输速率远没有达到带宽，可以看出广播网络效率低下。

2. h2 和 h3 向 h1 同时传输

这次由 h2、h3 同时向 h1 发送数据，测试实际传输速率（一共三次实验）。

第一次实验结果如下：





第二次实验结果如下：

```
root@Computer:~/workspace/Network/lab4/04-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 40326 connected with 10.0.0.1 port 5001
[ ID] Interval       Transfer     Bandwidth
[ 13]  0.0-30.1 sec  33.6 MBytes  9.36 Mbits/sec
```

```
root@Computer:~/workspace/Network/lab4/04-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 10.0.0.3 port 45234 connected with 10.0.0.1 port 5001
[ ID] Interval       Transfer     Bandwidth
[ 13]  0.0-30.1 sec  32.9 MBytes  9.16 Mbits/sec
```

第三次实验结果如下：

```
root@Computer:~/workspace/Network/lab4/04-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 40330 connected with 10.0.0.1 port 5001
[ ID] Interval       Transfer     Bandwidth
[ 13]  0.0-30.2 sec  33.6 MBytes  9.35 Mbits/sec
```

```
root@Computer:~/workspace/Network/lab4/04-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 10.0.0.3 port 45238 connected with 10.0.0.1 port 5001
[ ID] Interval       Transfer     Bandwidth
[ 13]  0.0-30.2 sec  33.5 MBytes  9.32 Mbits/sec
```

我们可以看到，三次实验中 h2 到 h1 和 h3 到 h1 的实际传输速率都接近 10Mb/s，与 h2/h3 到 b1 的带宽一致。

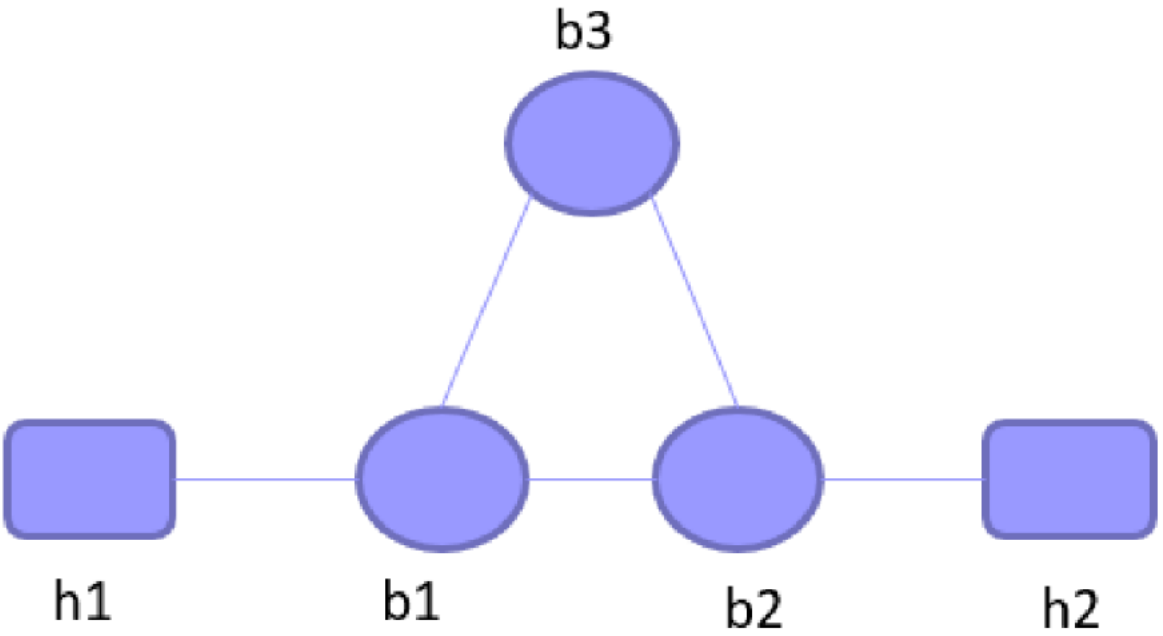
这是因为，h2 发给 h1 的数据在 b1 处，会同时广播给 h1 和 h3；而 h3 发给 h1 的数据也会在 b1 处，同时广播给 h1 和 h2。但是这时数据并不是竞争关系，而是

处于链路的两个不同方向。h2 给 h1 的数据从 b1 传到 h3，而 h3 给 h1 的数据从 h3 传到 b1，虽然他们都使用了 b1-h3 链路，但却是不同方向，互不影响，所以能达到链路最大带宽。而 b1 到 h1 链路带宽为 20Mb/s,刚好可以接收 2 个同时满带宽的 10Mb/s 数据。因此在，路上的每一个链路带宽都被完全利用了，广播网络的效率达到最高。

但总的来说，广播网络的效率不稳定，受传输的节点数量与传输方向影响很大。不管怎么说，广播的方式会产生很多无用的数据传输，会引起带宽利用率降低、无用数据抢占资源等问题。

3. 数据包在环路中不断广播

对 three_nodes_bw.py 文件进行更改,将网络改为由 2 个主机节点、3 个 Hub 节点构成的环状网络。



其中，除了增加 b2 和 b3 节点的声明及相关定义外,最重要的是重新构建节点间的互联关系,以实现实验要求的环形拓扑。如下图所示,需要建立共计 5 条连接: b1 和 b2 和 b3 互相的连接、b1 和 h1 的连接、b2 和 h2 的连接。

```python
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=10)
        self.addLink(h2, b2, bw=10)
        self.addLink(b1, b2, bw=10)
        self.addLink(b2, b3, bw=10)
        self.addLink(b1, b3, bw=10)
```

由 h1 向 h2 发送 ping 消息，用 wireshark 抓包结果的部分截图如下。

260	0.026695733	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
261	0.026696201	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
262	0.026696530	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
263	0.026697254	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
264	0.026697648	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
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
285	0.026707181	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
286	0.026707597	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
287	0.026708054	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
288	0.026708606	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
289	0.026709155	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
290	0.026763022	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

1135…	82.639417224	4e:23:0c:13:b2:03	26:9c:5d:0e:52:13	ARP	42	10.0.0.1 is at 4e:23:0c:13:b2:03
1135…	82.639454228	4e:23:0c:13:b2:03	26:9c:5d:0e:52:13	ARP	42	10.0.0.1 is at 4e:23:0c:13:b2:03
1135…	82.639484947	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id=0x312e, seq=1/256, ttl=64
1135…	82.639566195	4e:23:0c:13:b2:03	26:9c:5d:0e:52:13	ARP	42	10.0.0.1 is at 4e:23:0c:13:b2:03
1135…	82.639604928	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id=0x312e, seq=1/256, ttl=64
1135…	82.639683463	4e:23:0c:13:b2:03	26:9c:5d:0e:52:13	ARP	42	10.0.0.1 is at 4e:23:0c:13:b2:03
1135…	82.639710544	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id=0x312e, seq=1/256, ttl=64
1135…	82.639786620	4e:23:0c:13:b2:03	26:9c:5d:0e:52:13	ARP	42	10.0.0.1 is at 4e:23:0c:13:b2:03
1135…	82.639819344	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id=0x312e, seq=1/256, ttl=64
1135…	82.639896534	4e:23:0c:13:b2:03	26:9c:5d:0e:52:13	ARP	42	10.0.0.1 is at 4e:23:0c:13:b2:03
1135…	82.639931168	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id=0x312e, seq=1/256, ttl=64
1135…	82.640009814	4e:23:0c:13:b2:03	26:9c:5d:0e:52:13	ARP	42	10.0.0.1 is at 4e:23:0c:13:b2:03
1135…	82.640157231	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id=0x312e, seq=1/256, ttl=64
1135…	82.640157544	4e:23:0c:13:b2:03	26:9c:5d:0e:52:13	ARP	42	10.0.0.1 is at 4e:23:0c:13:b2:03
1135…	82.640157878	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id=0x312e, seq=1/256, ttl=64
1135…	82.640244565	4e:23:0c:13:b2:03	26:9c:5d:0e:52:13	ARP	42	10.0.0.1 is at 4e:23:0c:13:b2:03
1135…	82.640273700	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id=0x312e, seq=1/256, ttl=64
1135…	82.640348351	4e:23:0c:13:b2:03	26:9c:5d:0e:52:13	ARP	42	10.0.0.1 is at 4e:23:0c:13:b2:03
1135…	82.640384377	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id=0x312e, seq=1/256, ttl=64
1135…	82.640470577	4e:23:0c:13:b2:03	26:9c:5d:0e:52:13	ARP	42	10.0.0.1 is at 4e:23:0c:13:b2:03
1135…	82.640495609	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id=0x312e, seq=1/256, ttl=64
1135…	82.640573235	4e:23:0c:13:b2:03	26:9c:5d:0e:52:13	ARP	42	10.0.0.1 is at 4e:23:0c:13:b2:03
1135…	82.640603862	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id=0x312e, seq=1/256, ttl=64
1135…	82.640682363	4e:23:0c:13:b2:03	26:9c:5d:0e:52:13	ARP	42	10.0.0.1 is at 4e:23:0c:13:b2:03
1135…	82.640717964	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id=0x312e, seq=1/256, ttl=64
1135…	82.640796370	4e:23:0c:13:b2:03	26:9c:5d:0e:52:13	ARP	42	10.0.0.1 is at 4e:23:0c:13:b2:03
1135…	82.640834172	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id=0x312e, seq=1/256, ttl=64
1135…	82.640914562	4e:23:0c:13:b2:03	26:9c:5d:0e:52:13	ARP	42	10.0.0.1 is at 4e:23:0c:13:b2:03
1135…	82.640950129	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id=0x312e, seq=1/256, ttl=64
1135…	82.641025083	4e:23:0c:13:b2:03	26:9c:5d:0e:52:13	ARP	42	10.0.0.1 is at 4e:23:0c:13:b2:03

1949…	163.597175904	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id=0x312e, seq=1/256, ttl=64
1949…	163.597252511	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id=0x312e, seq=1/256, ttl=64
1949…	163.597333746	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id=0x312e, seq=1/256, ttl=64
1949…	163.597411133	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id=0x312e, seq=1/256, ttl=64
1949…	163.597489670	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id=0x312e, seq=1/256, ttl=64
1949…	163.597569815	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id=0x312e, seq=1/256, ttl=64
1949…	163.597644545	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id=0x312e, seq=1/256, ttl=64
1949…	163.597726130	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id=0x312e, seq=1/256, ttl=64
1949…	163.597803324	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id=0x312e, seq=1/256, ttl=64
1949…	163.597883290	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id=0x312e, seq=1/256, ttl=64
1949…	163.597958835	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id=0x312e, seq=1/256, ttl=64
1949…	163.598038094	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id=0x312e, seq=1/256, ttl=64
1949…	163.598116310	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id=0x312e, seq=1/256, ttl=64
1949…	163.598194882	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id=0x312e, seq=1/256, ttl=64
1949…	163.598272846	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id=0x312e, seq=1/256, ttl=64
1949…	163.598352329	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id=0x312e, seq=1/256, ttl=64
1949…	163.598426293	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id=0x312e, seq=1/256, ttl=64
1949…	163.598504648	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id=0x312e, seq=1/256, ttl=64
1949…	163.598586332	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id=0x312e, seq=1/256, ttl=64
1949…	163.598664340	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id=0x312e, seq=1/256, ttl=64
1949…	163.598739977	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id=0x312e, seq=1/256, ttl=64
1949…	163.598854203	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id=0x312e, seq=1/256, ttl=64
1949…	163.598903156	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id=0x312e, seq=1/256, ttl=64
1949…	163.598981869	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id=0x312e, seq=1/256, ttl=64
1949…	163.599060640	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id=0x312e, seq=1/256, ttl=64
1949…	163.599139391	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id=0x312e, seq=1/256, ttl=64
1949…	163.599217916	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id=0x312e, seq=1/256, ttl=64
1949…	163.599296442	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id=0x312e, seq=1/256, ttl=64
1949…	163.599374282	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id=0x312e, seq=1/256, ttl=64
1949…	163.599461682	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id=0x312e, seq=1/256, ttl=64
1949…	163.599539301	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id=0x312e, seq=1/256, ttl=64

可以看到，h1 节点频繁收到来自 h2 的相同的信息，这是因为其在网络中不断循环广播，一直重复（即数据包在 b1，b2，b3 中进行循环，所以 h1 不断从 b1 中收到来自 b2 的相同的信息）。

造成这种数据包环路现象的原因是网络中 Hub 节点构成了一个环。由于广播网络的工作模式，当网络包从 h2 达到 b2 后，b2 将数据包广播到 b3、b1。而下一时刻，b3 又将数据包广播到 b1，b1 又将数据包广播到 b3。之后它们又将数据包传回 b2，然后在 hub 环中重复上面的过程，这使得数据包的传输在网络中不断循环转发。

四、 实验总结

通过本次实验，我对广播网络有了更多的了解。在本次实验中，我更加深入的明白了广播网络的工作方式，并直接体会到了广播网络的效率特点，知道了数据传输方向对其的影响，也明白了广播的方式效率不高。而在最后一个实验中，我深刻认识到广播网络有着致命的弱点。其要求拓扑结构不能有环路，否则会造成数据包在环路中不断被转发，占据资源，冬儿对网络产生极大破坏。

实验任务二（switch）

一、 实验任务：

了解交换机的转发原理和转发表的构建方式,理解交换机如何学习和维护转发表。实现转发表的数据结构,支持转发表的查询、插入、老化操作,完成一个能自动学习转发表的交换机。使用 iperf 和给定的拓扑进行测试,对比交换机转发与之前集线器广播的性能差异。

二、 实验流程

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

三、 实验结果与分析

（一）实现交换机转发

1. 转发表的查询操作

转发表为了快速查询了转发表中端口对应的 256（ETH_ALEN）个链表，在查询时先对 MAC 地址 hash，根据 key 值找到所对应链表。然后遍历该链表，查看有无与输入 MAC

地址相同的表项。若有，查询成功，并返回该表项中端口结构 iface；若无，则查询失败，返回 NULL。

由于交换机转发过程中，会存在另一个线程进行超时表项的清理工作，因此查找操作需要加上锁来确保原子性。

具体的代码现实如下:

```
// lookup the mac address in mac_port table
iface_info_t *lookup_port(u8 mac[ETH_ALEN])
{
    // TODO: implement the lookup process here
    //fprintf(stdout, "TODO: implement the lookup process here.\n");
    int idx = (int)hash8((char*)mac, ETH_ALEN);
    mac_port_entry_t *entry;
    pthread_mutex_lock(&mac_port_map.lock);

    list_for_each_entry(entry, &(mac_port_map.hash_table[idx]), 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;
}
```

2. 转发表的插入操作

该操作的目的是当转发表中没有源 mac 地址和对应的 iface 的映射表项时，将源 mac 地址与该 iface 插入到转发表当中。

首先根据源 mac 地址在转发表中查找表项，如果找到，那么更新表项、更新访问时

间；如果没有找到，那么将源地址与端口的映射关系写入转发表。

同样的,由于交换机转发过程中,会存在另一个线程进行超时表项的清理工作,因此插入操作同样需要加上锁来确保原子性。

具体的代码现实如下:

```
// insert the mac -> iface mapping into mac_port table
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");
    int idx = (int)hash8((char*)mac, ETH_ALEN);
    mac_port_entry_t *entry;
    time_t now = time(NULL);
    pthread_mutex_lock(&(mac_port_map.lock));

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

            return ;
        }
    }
    mac_port_entry_t *new = malloc(sizeof(mac_port_entry_t));
    new->iface = iface;
    new->visited = now;
    for(int i=0;i<ETH_ALEN;i++)
        new->mac[i] = mac[i];

    list_add_head(&new->list, &(mac_port_map.hash_table[idx]));
    pthread_mutex_unlock(&(mac_port_map.lock));

    return ;
}
```

3. 转发表的老化操作

该操作的作用是：当转发表中的表项超过 30s 没有被查询，则删除冗旧的表项

执行老化操作时遍历整个转发表,查看当前时间与每个表项访问时间之差是否超过30 秒,如果超过就删除掉该表项。

同样的,由于交换机转发过程中,会存在另一个线程进行超时表项的清理工作,因此插入操作同样需要加上锁来确保原子性。

具体的代码现实如下:

```
// sweeping mac_port table, remove the entry which has not been visited in the
// last 30 seconds.
int sweep_aged_mac_port_entry()
{
    // TODO: implement the sweeping process here
    //fprintf(stdout, "TODO: implement the sweeping process here.\n");
    int n = 0;
    mac_port_entry_t *entry, *q;
    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, q, &mac_port_map.hash_table[i], list){
            if((int)(now - entry->visited) > MAC_PORT_TIMEOUT){
                list_delete_entry(&entry->list);
                free(entry);
                n++;
            }
        }
    }
    pthread_mutex_unlock(&mac_port_map.lock);

    return n;
}
```

4. 广播操作

该操作的作用是：广播收到的包,代码复用实验任务一的广播代码即可。

5. 交换机的处理操作

交换机需要完成 3 种操作,查询操作、插入操作、老化操作。其中老化操作由单独线程处理，在初始化转发表时（init_mac_port_table 函数）就封装调用了上面的 sweep_aged_mac_port_entry 函数，这里不再赘述了。

而 handle_packet 函数中需要完成的，就是查询操作和插入操作。查询操作对目的 MAC 地址进行查询，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，否则就广播该数据包。而插入操作对源 MAC 地址进行查询，如果查到相应条目，更新访问时间；如果没有查到，那么将该地址与端口的映射关系写入到转发表（这里只需要调用已经写好的 insert_mac_port 函数即可）。

具体的代码实现如下

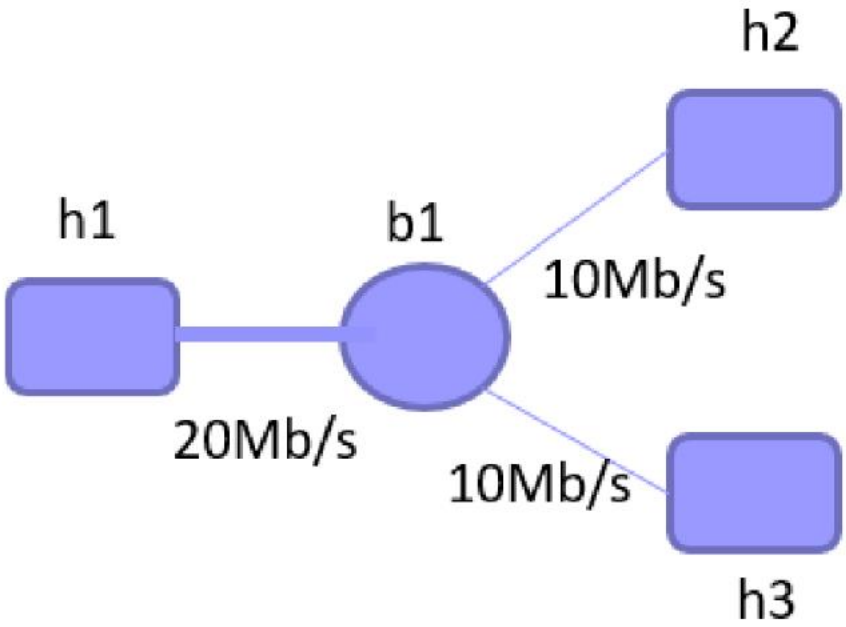
```
void handle_packet(iface_info_t *iface, char *packet, int len)
{
    // TODO: implement the packet forwarding process here
    //fprintf(stdout, "TODO: implement the packet forwarding process here.\n");

    struct ether_header *eh = (struct ether_header *)packet;
    //log(DEBUG, "the dst mac address is " ETHER_STRING ".\n", ETHER_FMT(eh->ether

    iface_info_t * dest_iface = lookup_port(eh->ether_dhost);
    if (dest_iface) {
        // log(DEBUG, "Send this packet to %s.", dest_iface->name);
        iface_send_packet(dest_iface, packet, len);
    } else {
        // log(DEBUG, "Broadcast this packet.");
        broadcast_packet(iface, packet, len);
    }

    // log(DEBUG, "Insert into mac_port_map: " ETHER_STRING " -> %s.", ETHER_FMT(e
    insert_mac_port(eh->ether_shost, iface);

    free(packet);
}
```

（二）测量交换机转发性能

在测试性能前先注释掉代码中的 debug 打印信息，避免打印占据较多时间而对传输速率产生影响。

1. h1 向 h2 和 h3 同时传输

网络的拓扑结构与上个实验任务一致(b1 节点名称改为了 s1,在本质上没有区别)，如下所示：



h1 同时向 h2 和 h3 发送数据，测试传输速率（一共做了三次实验）：

第一次实验结果如下：

```
root@Computer:~/workspace/Network/lab4/04-hub+switch/switch# iperf -c 10.0.0.2
-t 30
------------------------------------------------------------
Client connecting to 10.0.0.2, TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[ 13] local 10.0.0.1 port 39390 connected with 10.0.0.2 port 5001
[ ID] Interval       Transfer     Bandwidth
[ 13]  0.0-30.2 sec  34.5 MBytes  9.58 Mbits/sec
```

```
root@Computer:~/workspace/Network/lab4/04-hub+switch/switch# iperf -c 10.0.0.3
-t 30
------------------------------------------------------------
Client connecting to 10.0.0.3, TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[ 13] local 10.0.0.1 port 50600 connected with 10.0.0.3 port 5001
[ ID] Interval       Transfer     Bandwidth
[ 13]  0.0-30.2 sec  34.6 MBytes  9.63 Mbits/sec
```

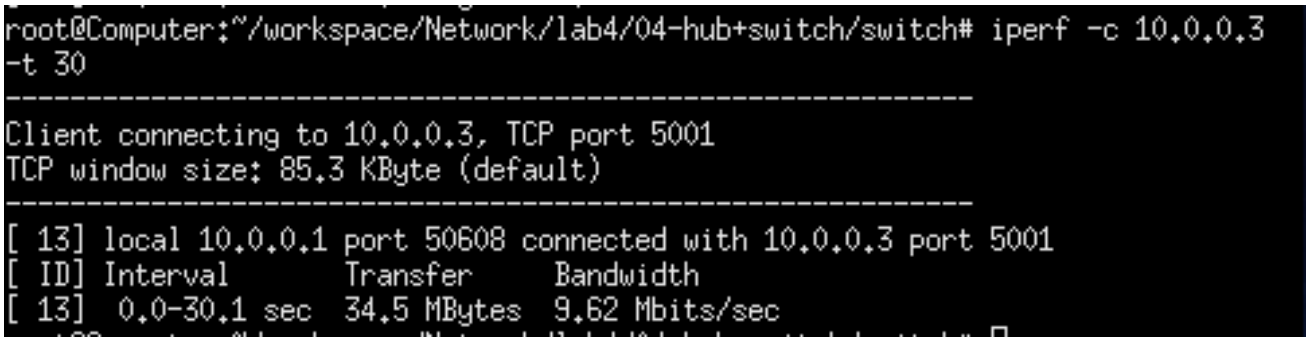
第二次实验结果如下：

```
root@Computer:~/workspace/Network/lab4/04-hub+switch/switch# iperf -c 10.0.0.2
-t 30
------------------------------------------------------------
Client connecting to 10.0.0.2, TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[ 13] local 10.0.0.1 port 39396 connected with 10.0.0.2 port 5001
[ ID] Interval       Transfer     Bandwidth
[ 13]  0.0-30.2 sec  34.8 MBytes  9.65 Mbits/sec
```

```
root@Computer:~/workspace/Network/lab4/04-hub+switch/switch# iperf -c 10.0.0.3
-t 30
------------------------------------------------------------
Client connecting to 10.0.0.3, TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[ 13] local 10.0.0.1 port 50602 connected with 10.0.0.3 port 5001
[ ID] Interval       Transfer     Bandwidth
[ 13]  0.0-30.2 sec  34.6 MBytes  9.61 Mbits/sec
```

第三次实验结果如下：

```
root@Computer:~/workspace/Network/lab4/04-hub+switch/switch# iperf -c 10.0.0.2
-t 30
------------------------------------------------------------
Client connecting to 10.0.0.2, TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[ 13] local 10.0.0.1 port 39402 connected with 10.0.0.2 port 5001
[ ID] Interval       Transfer     Bandwidth
[ 13]  0.0-30.2 sec  34.8 MBytes  9.65 Mbits/sec
```



从上面几张图我们可以看到, h1 到 h2 平均传输速率为 9.63Mb/s，而 h1 到 h3 平均传输速率为 9.62Mb/s，都基本接近各自带宽。可以说各自链路都充分利用了各自带宽。

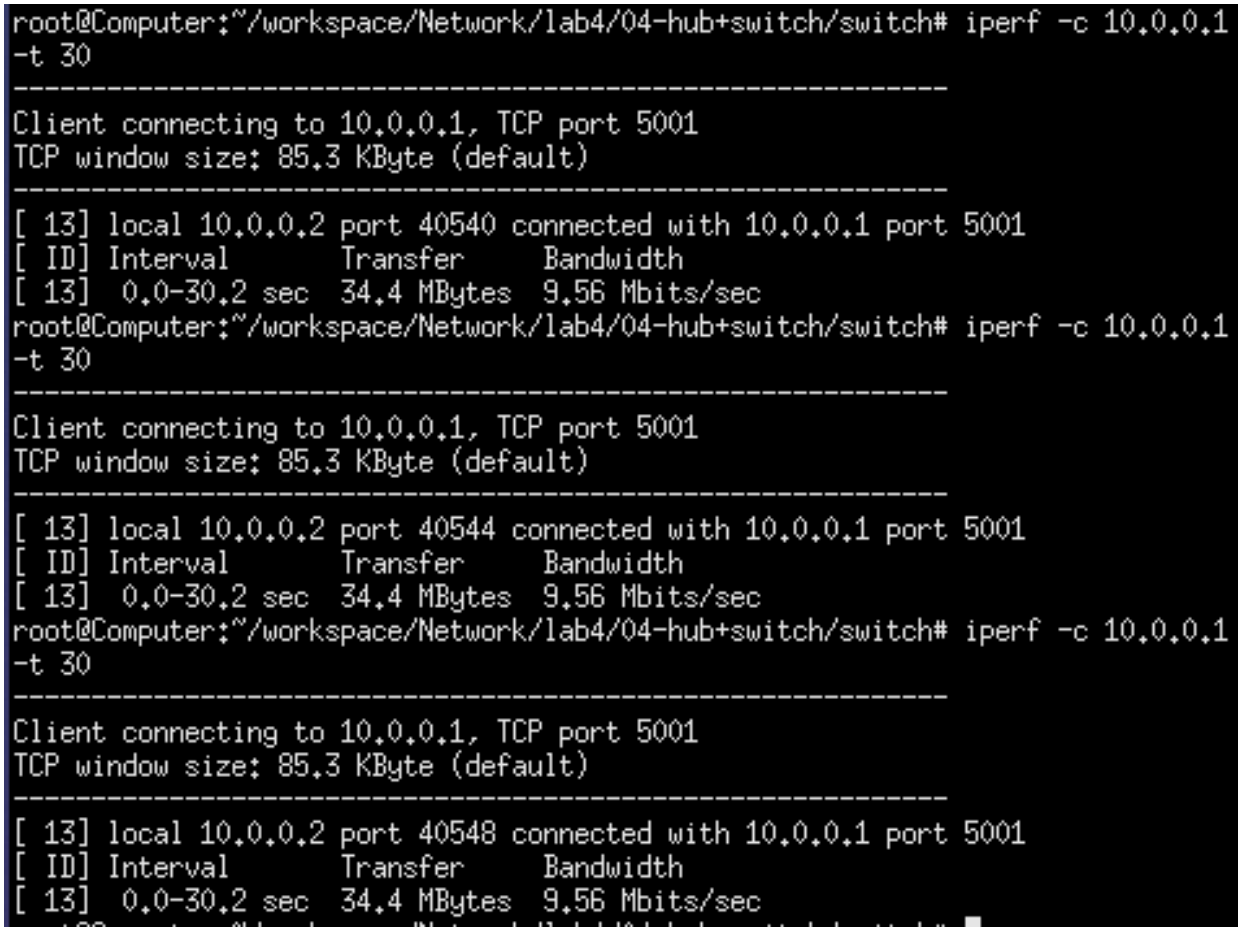
可以发现，在 h1 同时向 h2 和 h3 发送数据的情况下，交换机节点中转的效率明显高于广播节点。这是因为广播节点必须同时向两条带宽上限为 10Mbps 的数据通路分发数据包,严重影响了数据包传输的效率（具体分析见实验任务一）；而在交换机的情况下，除了第一次通讯时由于转发表为空需要广播数据包以外，由于服务端会发送响应包，所以，从第二个数据包开始，交换机就直接只向对应的端口进行转发，使得连接 h2 和 h3 的两条通路可以相对独立地进行数据收发。这样带宽得到了充分的利用，此时，只有交换机转的软件处理的时间(如交换表查询等操作)可能影响传输的效率。

2. h2 和 h3 向 h1 同时传输

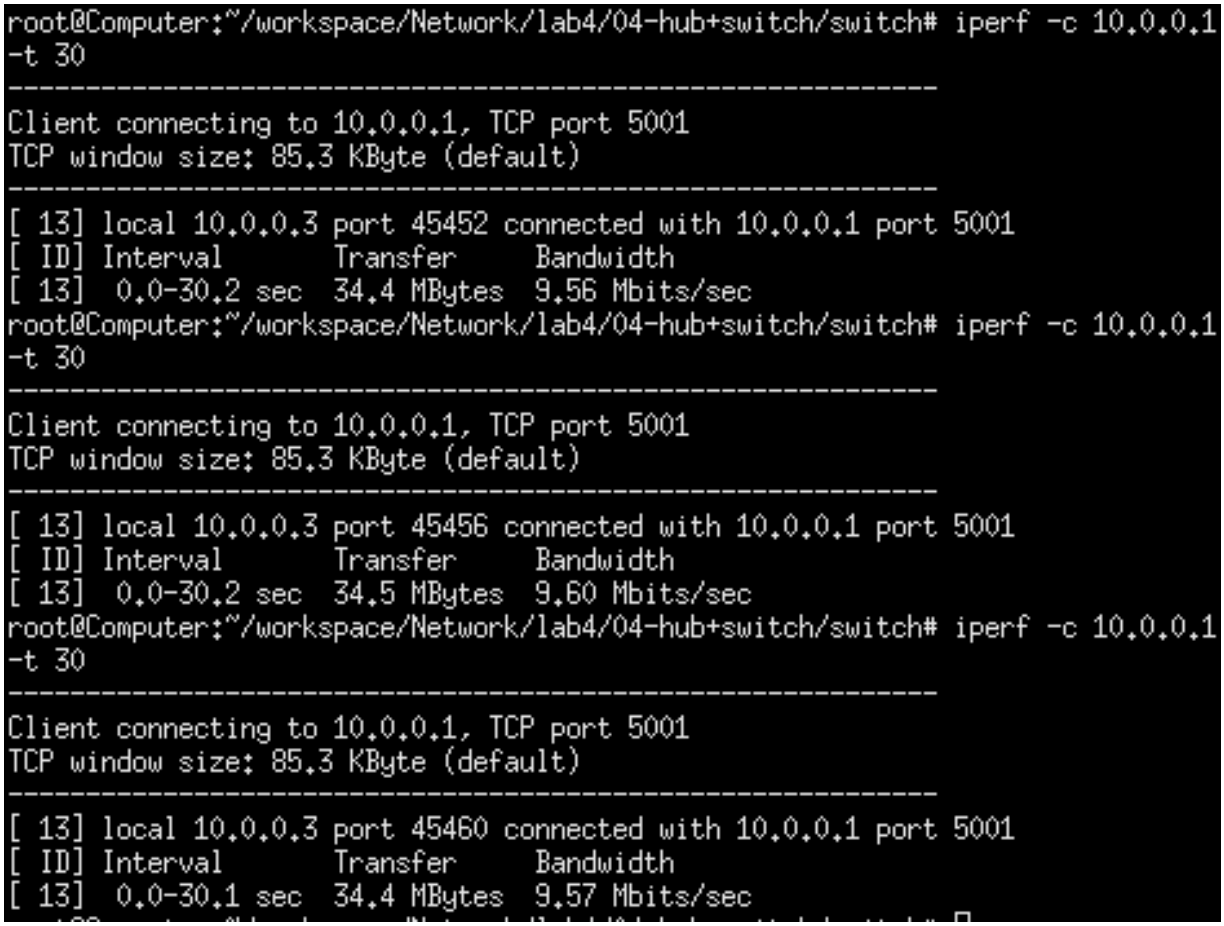
这次由 h2、h3 同时向 h1 发送数据，测试实际传输速率（一共三次实验）。

三次的实验结果如下:

h2 节点向 h1 节点的三次测量结果:



h3 节点向 h1 节点的三次测量结果：



从上面几张图我们可以看到, h1 到 h2 平均传输速率为 9.56Mb/s，而 h1 到 h3 平均传输速率为 9.58Mb/s，都基本接近各自带宽，可以说各自链路都充分利用了各自带宽。而具体原因与上一个测量实验（h1 向 h2 和 h3 同时传输）一致，这里不进行赘述了，这种情况下广播网络也能充分利用各自的带宽，具体原因可见实验任务一。

3. 交换机转发与集线器广播性能对比

对于集线器广播方式，从实验任务一中可以得知, h1 向 h2 和 h3 发送时，平均速率为 6.27Mb/s、 3.58Mb/s,总体速率远小于其对应的带宽。这是因为广播方式会将数据包向所有端口发送，占用其他链路的带宽。交换机在第一次转发时,也是广播方式。但之后学习到 MAC 地址和端口的对应关系之后,就只向目的端口发送数据包，只占用该链路带宽。因此总体上利用率可以拉满，实际传输速率接近满带宽。

而对于 h2 和 h3 同时向 h1 发送数据时,集线器和交换机没有什么差异,都能完全利用带宽（具体分析可参见前文）。

但总的来说，交换机的性能较好，它在学习完毕后可以定向发送数据包，没有无用数据挤占带宽。同时它不受传输方向的影响，上下行效率一致。而集线器效率受传输方向影响很大，上下行不对等，坏的情况下效率很低。此外在节点更多后无用数据会更加挤占带宽，性能受到局限。

四、 实验总结

通过本次实验，我对交换机及其工作原理有了更加深入的了解。首先，我学到了交换机的工作方式，即通过转发表来学习 MAC 地址与端口的对应关系，以此优化转发。其次，我知道了转发表的组织结构，明白了转发表的一些基本操作，并且本次实验中，我实现了转发表和交换机，并对比分析了集线器和交换机的性能差异，这让我对交换机的优势以及其巧妙的设计有了更准确的认识，这让我对交换机有了更深的理解。