# 实验报告

# 广播网络实验

## 一、实验内容

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，验证广播网络功能。
3. 测试广播网络效率，并对结果进行解释。
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、实验结果及分析

### （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，每次收到网络包消息时，遍历与之相邻的每个网络端口，如果不是发送该网络包的端口，就将网络包广播到这个端口。遍历过程可以通过现成的链表操作实现，具体代码如下。

```
iface_info_t *ifc = NULL;
list_for_each_entry(ifc, &instance->iface_list, list) {
    if (ifc != iface) {
        iface_send_packet(ifc, packet, len);
    }
}
```

#### 2、结果验证

三个节点各自向其他两个节点发送消息，验证其两两相互连通。

h1 节点结果如下：

"Node: h1"

```
root@joker-linux:/mnt/hgfs/share/04-broadcast# 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=0.170 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.132 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.185 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.093 ms

--- 10.0.0.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3064ms
rtt min/avg/max/mdev = 0.093/0.145/0.185/0.035 ms
root@joker-linux:/mnt/hgfs/share/04-broadcast# 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=0.155 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=0.260 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=0.157 ms
64 bytes from 10.0.0.3: icmp_seq=4 ttl=64 time=0.187 ms

--- 10.0.0.3 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3061ms
rtt min/avg/max/mdev = 0.155/0.189/0.260/0.042 ms
root@joker-linux:/mnt/hgfs/share/04-broadcast#
```

可以看到，h1 与其他两个节点连通。

h2 节点结果如下：

"Node: h2"

```
root@joker-linux:/mnt/hgfs/share/04-broadcast# 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=0.166 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=0.106 ms
64 bytes from 10.0.0.1: icmp_seq=3 ttl=64 time=0.164 ms
64 bytes from 10.0.0.1: icmp_seq=4 ttl=64 time=0.772 ms

--- 10.0.0.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3072ms
rtt min/avg/max/mdev = 0.106/0.302/0.772/0.272 ms
root@joker-linux:/mnt/hgfs/share/04-broadcast# 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=0.177 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=0.145 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=0.172 ms
64 bytes from 10.0.0.3: icmp_seq=4 ttl=64 time=0.147 ms

--- 10.0.0.3 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3071ms
rtt min/avg/max/mdev = 0.145/0.160/0.177/0.014 ms
root@joker-linux:/mnt/hgfs/share/04-broadcast#
```

可以看到，h2 与其他两个节点连通。

h3 节点结果如下：

"Node: h3"

```
root@joker-linux:/mnt/hgfs/share/04-broadcast# 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=0.254 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=0.094 ms
64 bytes from 10.0.0.1: icmp_seq=3 ttl=64 time=0.101 ms
64 bytes from 10.0.0.1: icmp_seq=4 ttl=64 time=0.107 ms

--- 10.0.0.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3068ms
rtt min/avg/max/mdev = 0.094/0.139/0.254/0.066 ms
root@joker-linux:/mnt/hgfs/share/04-broadcast# 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=0.146 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.136 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.163 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.147 ms

--- 10.0.0.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3075ms
rtt min/avg/max/mdev = 0.136/0.148/0.163/0.009 ms
root@joker-linux:/mnt/hgfs/share/04-broadcast#
```

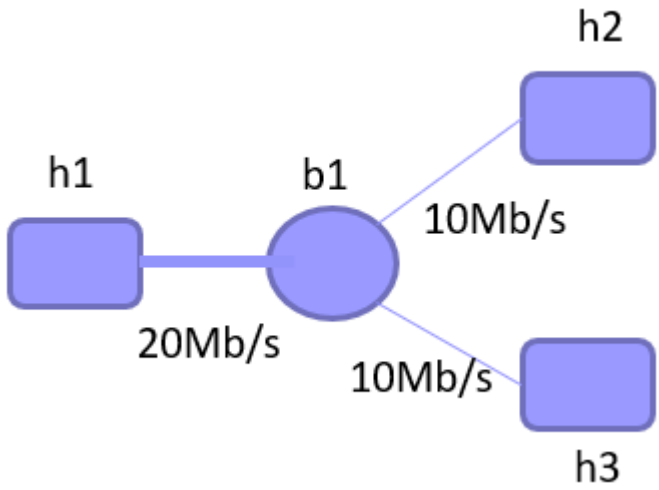
可以看到，h3 与其他两个节点连通。

因此，三个节点两两连通，广播网络能够正常运行。

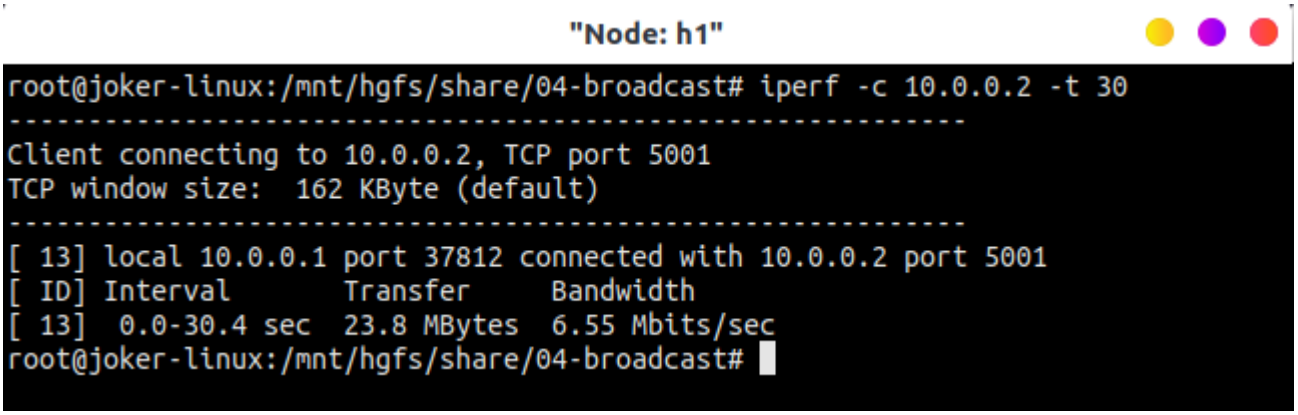
## （二）广播网络传输效率

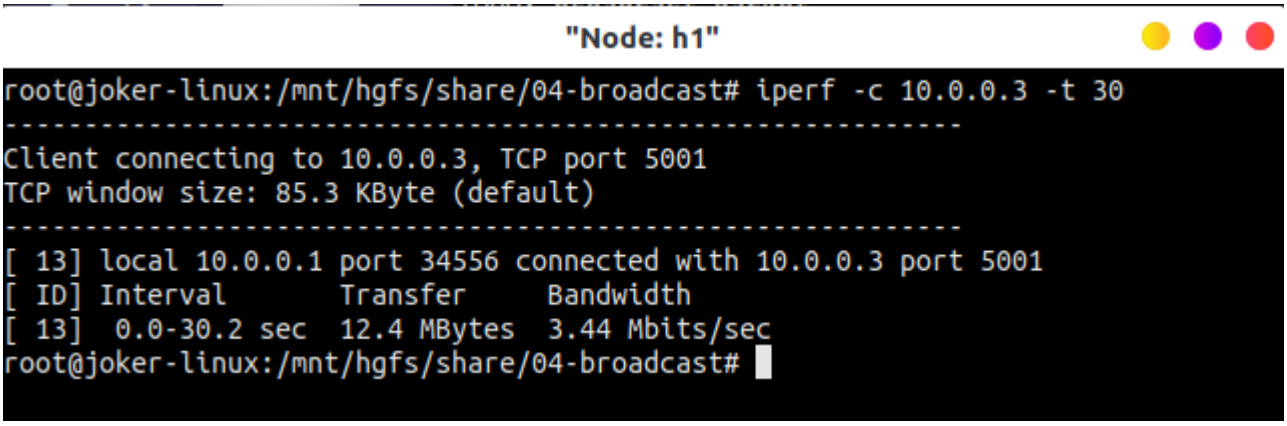
### 1、h1 同时向 h2 和 h3 测量

网络拓扑结构如下图所示：




这次由 h1 同时向 h2、h3 发送数据，测试实际传输速率。



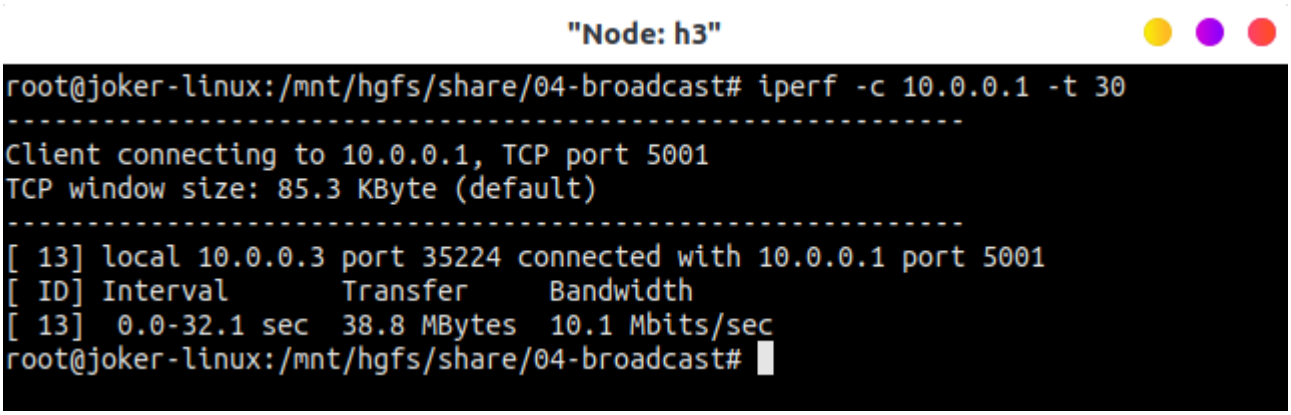

我们可以看到，h1 到 h2 实际传输速率为 6.55Mb/s，而 h1 到 h3 实际传输速率为 3.44Mb/s，都远小于各自带宽。注意到两者速率加起来刚好 9.99Mb/s，恰好达到 b1 到 h2/h3 的带宽。

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。同样地，h1 发给 h3 的数据也会占据 b1 到 h2 的传输带宽。于是 b1-h2 和 b1-h3 两条传输通路的传输数据实际是相同的，都为 h2 和 h3 的全部数据。因此 h1-h2 速率与 h1-h3 速率之和不能超过单条通路带宽 10Mb/s。这时 b1-h2 和 b1-h3 两条通路的总效率只有 50%。

至于 h1-h2 的速率与 h1-h3 的速率有些差异，会受到先后启动的影响，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，远没有完全利用带宽，广播网络效率低下。

## 2、h2 和 h3 同时向 h1 测量

这次由 h2、h3 同时向 h1 发送数据，测试实际传输速率。



```
"Node: h2"
root@joker-linux:/mnt/hgfs/share/04-broadcast# iperf -c 10.0.0.1 -t 30
------------------------------------------------------------
Client connecting to 10.0.0.1, TCP port 5001
TCP window size:  170 KByte (default)
------------------------------------------------------------
[ 13] local 10.0.0.2 port 36980 connected with 10.0.0.1 port 5001
[ ID] Interval       Transfer     Bandwidth
[ 13]  0.0-30.6 sec  34.6 MBytes  9.51 Mbits/sec
root@joker-linux:/mnt/hgfs/share/04-broadcast#
```

我们可以看到，h2-h1 和 h3-h1 的实际传输速率都接近 10Mb/s，几乎完全利用了带宽。

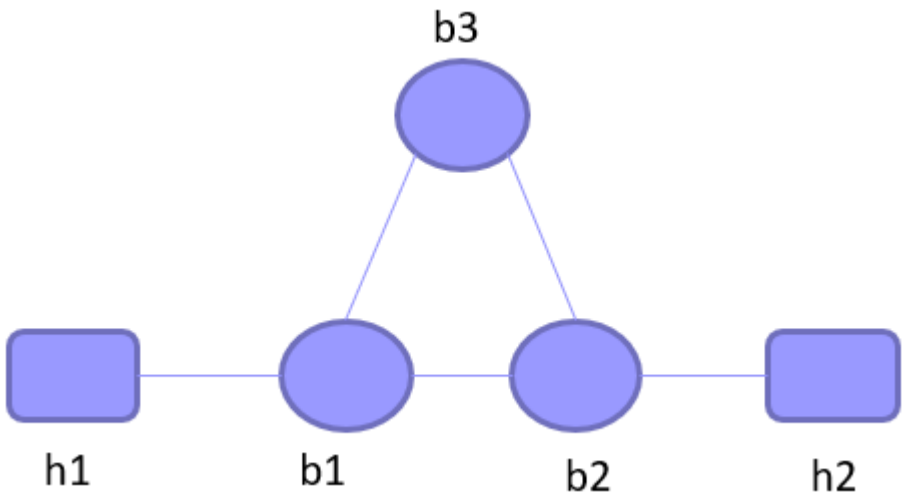
与上一种情况类似，h2 发给 h1 的数据在 b1 处，会同时广播给 h1 和 h3；h3 发给 h1 的数据在 b1 处，会同时广播给 h1 和 h2。但是这时数据并不是竞争关系，而是处于链路的两个不同方向。h2 给 h1 的数据从 b1 传到 h3；h3 给 h1 的数据从 h3 传到 b1，他们都使用了 b1-h3 链路，但却是不同方向，互不影响，都能达到链路最大带宽。而 b1-h1 链路带宽为 20Mb/s，刚好可以接收 2 个同时满带宽的 10Mb/s 数据。

因此在这种情况下，各链路带宽完全利用，广播网络效率达到最高。

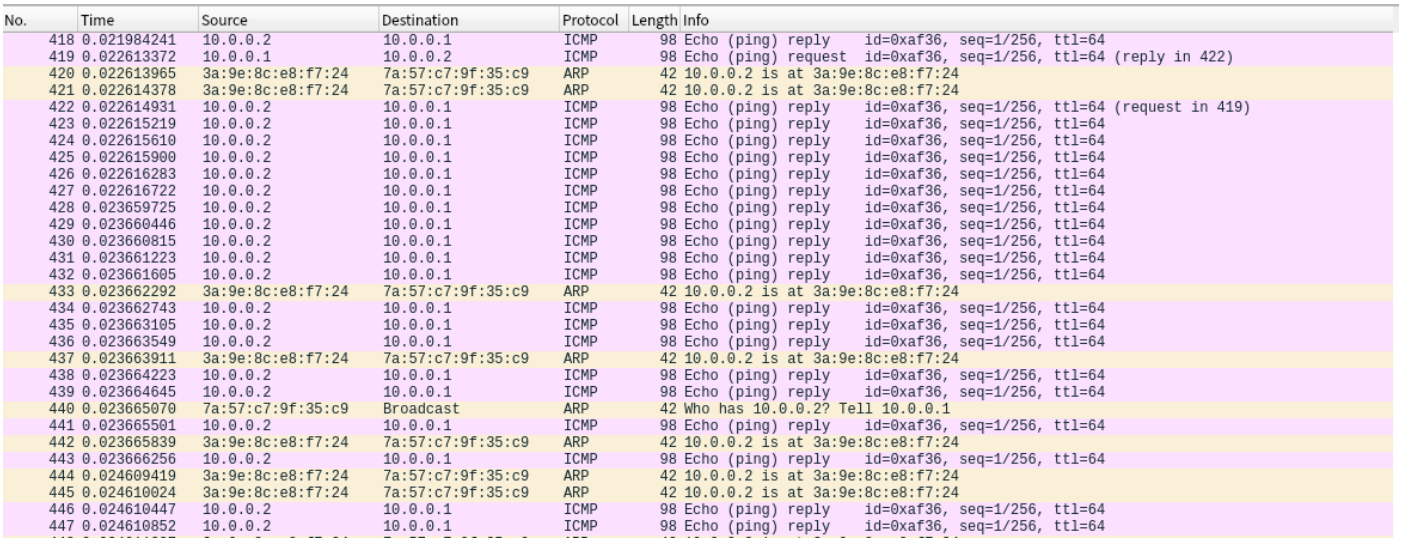
总的来说，广播网络的效率不稳定，受传输方向影响极大。并且广播的方式会产生很多无用的数据传输，会引起带宽利用率降低、无用数据抢占资源等问题。

## （三）数据包在环路中不断广播

首先对 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。




由 h1 向 h2 发送 ping 消息，用 wireshark 抓包结果如下。



| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 418 | 0.021984241 | 10.0.0.2 | 10.0.0.1 | ICMP | 98 | Echo (ping) reply id=0xaf36, seq=1/256, ttl=64 |
| 419 | 0.022613372 | 10.0.0.1 | 10.0.0.2 | ICMP | 98 | Echo (ping) request id=0xaf36, seq=1/256, ttl=64 (reply in 422) |
| 420 | 0.022613965 | 3a:9e:8c:e8:f7:24 | 7a:57:c7:9f:35:c9 | ARP | 42 | 10.0.0.2 is at 3a:9e:8c:e8:f7:24 |
| 421 | 0.022614378 | 3a:9e:8c:e8:f7:24 | 7a:57:c7:9f:35:c9 | ARP | 42 | 10.0.0.2 is at 3a:9e:8c:e8:f7:24 |
| 422 | 0.022614931 | 10.0.0.2 | 10.0.0.1 | ICMP | 98 | Echo (ping) reply id=0xaf36, seq=1/256, ttl=64 (request in 419) |
| 423 | 0.022615219 | 10.0.0.2 | 10.0.0.1 | ICMP | 98 | Echo (ping) reply id=0xaf36, seq=1/256, ttl=64 |
| 424 | 0.022615610 | 10.0.0.2 | 10.0.0.1 | ICMP | 98 | Echo (ping) reply id=0xaf36, seq=1/256, ttl=64 |
| 425 | 0.022615900 | 10.0.0.2 | 10.0.0.1 | ICMP | 98 | Echo (ping) reply id=0xaf36, seq=1/256, ttl=64 |
| 426 | 0.022616283 | 10.0.0.2 | 10.0.0.1 | ICMP | 98 | Echo (ping) reply id=0xaf36, seq=1/256, ttl=64 |
| 427 | 0.022616722 | 10.0.0.2 | 10.0.0.1 | ICMP | 98 | Echo (ping) reply id=0xaf36, seq=1/256, ttl=64 |
| 428 | 0.023659725 | 10.0.0.2 | 10.0.0.1 | ICMP | 98 | Echo (ping) reply id=0xaf36, seq=1/256, ttl=64 |
| 429 | 0.023660446 | 10.0.0.2 | 10.0.0.1 | ICMP | 98 | Echo (ping) reply id=0xaf36, seq=1/256, ttl=64 |
| 430 | 0.023660815 | 10.0.0.2 | 10.0.0.1 | ICMP | 98 | Echo (ping) reply id=0xaf36, seq=1/256, ttl=64 |
| 431 | 0.023661223 | 10.0.0.2 | 10.0.0.1 | ICMP | 98 | Echo (ping) reply id=0xaf36, seq=1/256, ttl=64 |
| 432 | 0.023661605 | 10.0.0.2 | 10.0.0.1 | ICMP | 98 | Echo (ping) reply id=0xaf36, seq=1/256, ttl=64 |
| 433 | 0.023662292 | 3a:9e:8c:e8:f7:24 | 7a:57:c7:9f:35:c9 | ARP | 42 | 10.0.0.2 is at 3a:9e:8c:e8:f7:24 |
| 434 | 0.023662743 | 10.0.0.2 | 10.0.0.1 | ICMP | 98 | Echo (ping) reply id=0xaf36, seq=1/256, ttl=64 |
| 435 | 0.023663105 | 10.0.0.2 | 10.0.0.1 | ICMP | 98 | Echo (ping) reply id=0xaf36, seq=1/256, ttl=64 |
| 436 | 0.023663549 | 10.0.0.2 | 10.0.0.1 | ICMP | 98 | Echo (ping) reply id=0xaf36, seq=1/256, ttl=64 |
| 437 | 0.023663911 | 3a:9e:8c:e8:f7:24 | 7a:57:c7:9f:35:c9 | ARP | 42 | 10.0.0.2 is at 3a:9e:8c:e8:f7:24 |
| 438 | 0.023664223 | 10.0.0.2 | 10.0.0.1 | ICMP | 98 | Echo (ping) reply id=0xaf36, seq=1/256, ttl=64 |
| 439 | 0.023664645 | 10.0.0.2 | 10.0.0.1 | ICMP | 98 | Echo (ping) reply id=0xaf36, seq=1/256, ttl=64 |
| 440 | 0.023665070 | 7a:57:c7:9f:35:c9 | Broadcast | ARP | 42 | Who has 10.0.0.2? Tell 10.0.0.1 |
| 441 | 0.023665501 | 10.0.0.2 | 10.0.0.1 | ICMP | 98 | Echo (ping) reply id=0xaf36, seq=1/256, ttl=64 |
| 442 | 0.023665839 | 3a:9e:8c:e8:f7:24 | 7a:57:c7:9f:35:c9 | ARP | 42 | 10.0.0.2 is at 3a:9e:8c:e8:f7:24 |
| 443 | 0.023666256 | 10.0.0.2 | 10.0.0.1 | ICMP | 98 | Echo (ping) reply id=0xaf36, seq=1/256, ttl=64 |
| 444 | 0.024609419 | 3a:9e:8c:e8:f7:24 | 7a:57:c7:9f:35:c9 | ARP | 42 | 10.0.0.2 is at 3a:9e:8c:e8:f7:24 |
| 445 | 0.024610024 | 3a:9e:8c:e8:f7:24 | 7a:57:c7:9f:35:c9 | ARP | 42 | 10.0.0.2 is at 3a:9e:8c:e8:f7:24 |
| 446 | 0.024610447 | 10.0.0.2 | 10.0.0.1 | ICMP | 98 | Echo (ping) reply id=0xaf36, seq=1/256, ttl=64 |
| 447 | 0.024610852 | 10.0.0.2 | 10.0.0.1 | ICMP | 98 | Echo (ping) reply id=0xaf36, seq=1/256, ttl=64 |

上图是截取的一部分网络包消息，可以看到其中有：如 440 的 ARP 广播包，查询 h2 的 mac 地址；如 442 的 ARP 回应包，回答 h2 的 mac 地址；如 419 的 ping 请求包；如 422 的 ping 应答包。

这 4 种包在网络中不断循环广播，一直重复，将网络资源占满，造成网络卡死。

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 h1 达到 b1，b1 将数据包广播到 b2、b3。而下一时刻，b2 又将数据包广播到 b3，b3 又将数据包广播到 b2。之后它们又将数据包传回 b1，然后 b1 将 b2 给它的包传给 b3；b3 给它的包传给 b2。以此类推，数据包的传输构成一个环路，在网络中不断循环转发，将网络卡死。

## 四、实验总结

通过本次实验，我对广播网络有了更多的了解。在第一个实验中，我学到了广播网络的工作方式，掌握如何实现一个广播网络节点。在第二个实验中，我了解到广播网络的效率特点，数据传输方向对其影响极大，效率不高。在最后一个实验中，我认识到广播网络的一个致命弱点。拓扑结构不能有环路，否则会造成数据包环路，一个数据包在网络里不断转发，占据资源，对网络产生极大破坏。