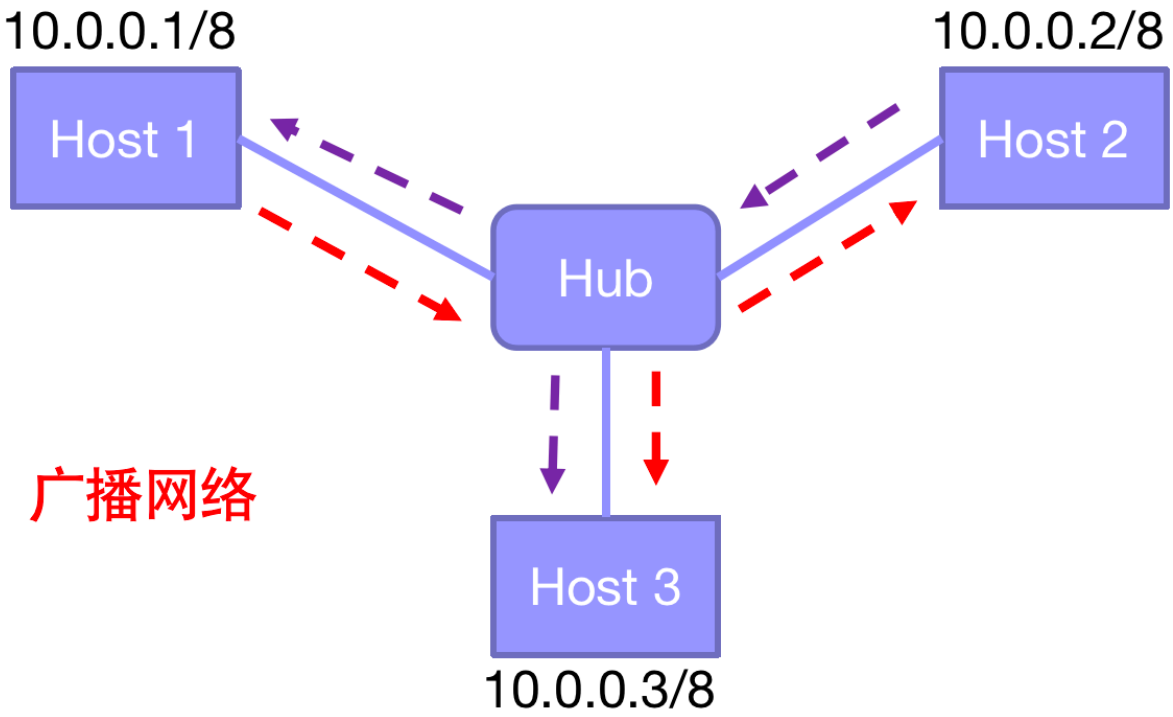
网络实验报告

2016K8009909005 雷正宇

广播网络实验

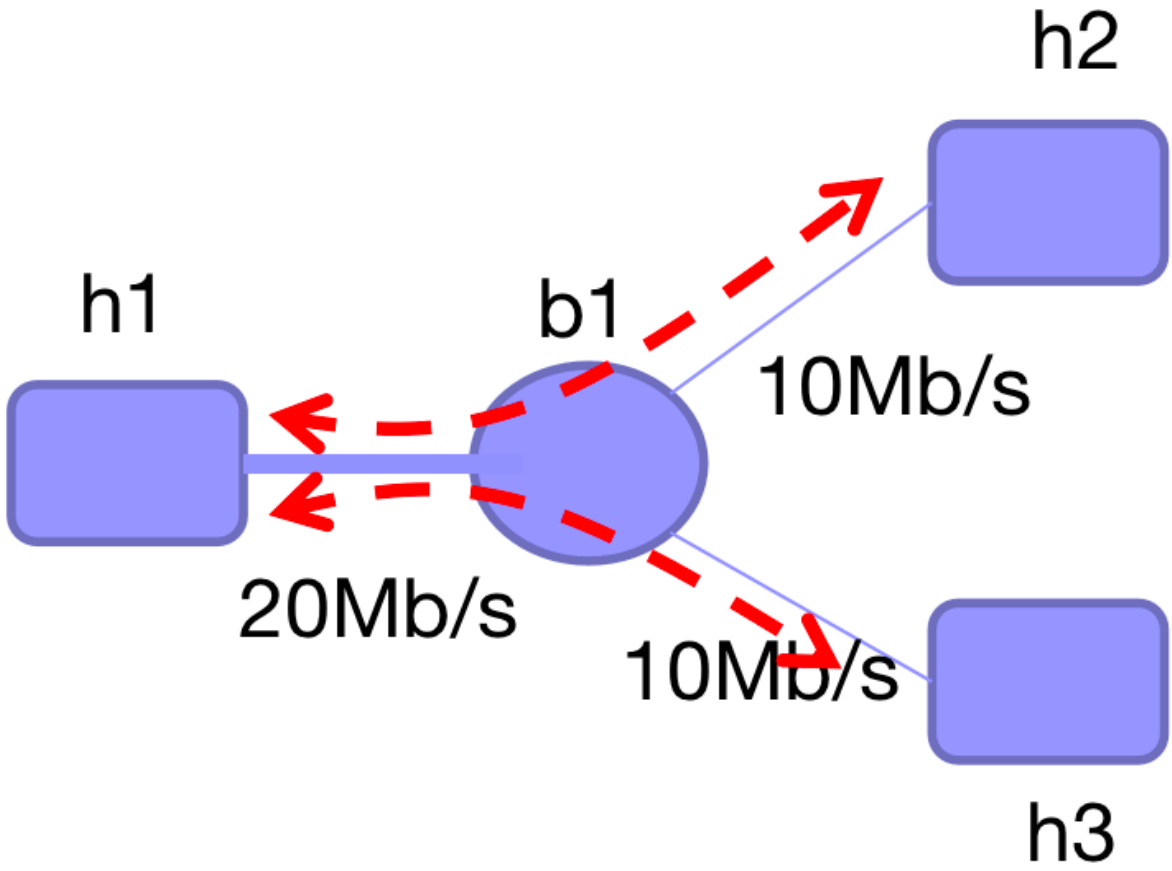
实验内容

实现节点广播的broadcast_packet函数；

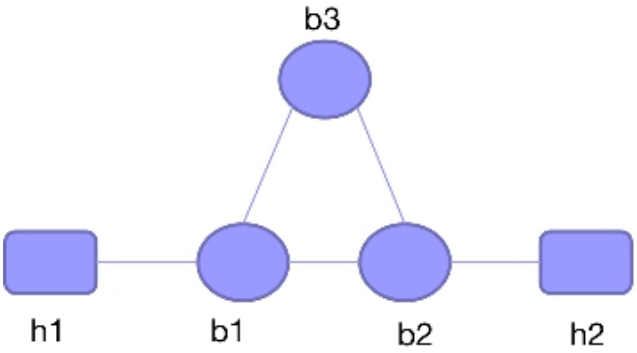


验证广播网络能够正常运行：从一个端节点ping另一个端节点；

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: iperf server; H2, H3: clients；



先构建环形拓扑，验证该拓扑下节点广播会产生数据包环路。



实验流程

1. 实现节点广播函数

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

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

这个函数用到了列表的遍历宏：

```c
#define list_for_each_entry(pos, head, member) \
  for (pos = list_entry((head)->next, typeof(*pos), member); \
      &pos->member != (head); \
      pos = list_entry(pos->member.next, typeof(*pos), member))
```

这里的 list 结构是 Linux 风格的：如果某个结构体类型需要当作被链表处理，就让它包含一个链表节点结构，从而被一个链表串起来。list_entry 宏的作用是通过链表节点找到持有它的外部结构。如果不对结点进行删除操作，用 list_for_each_entry 这个宏就能很方便地遍历整个链表。

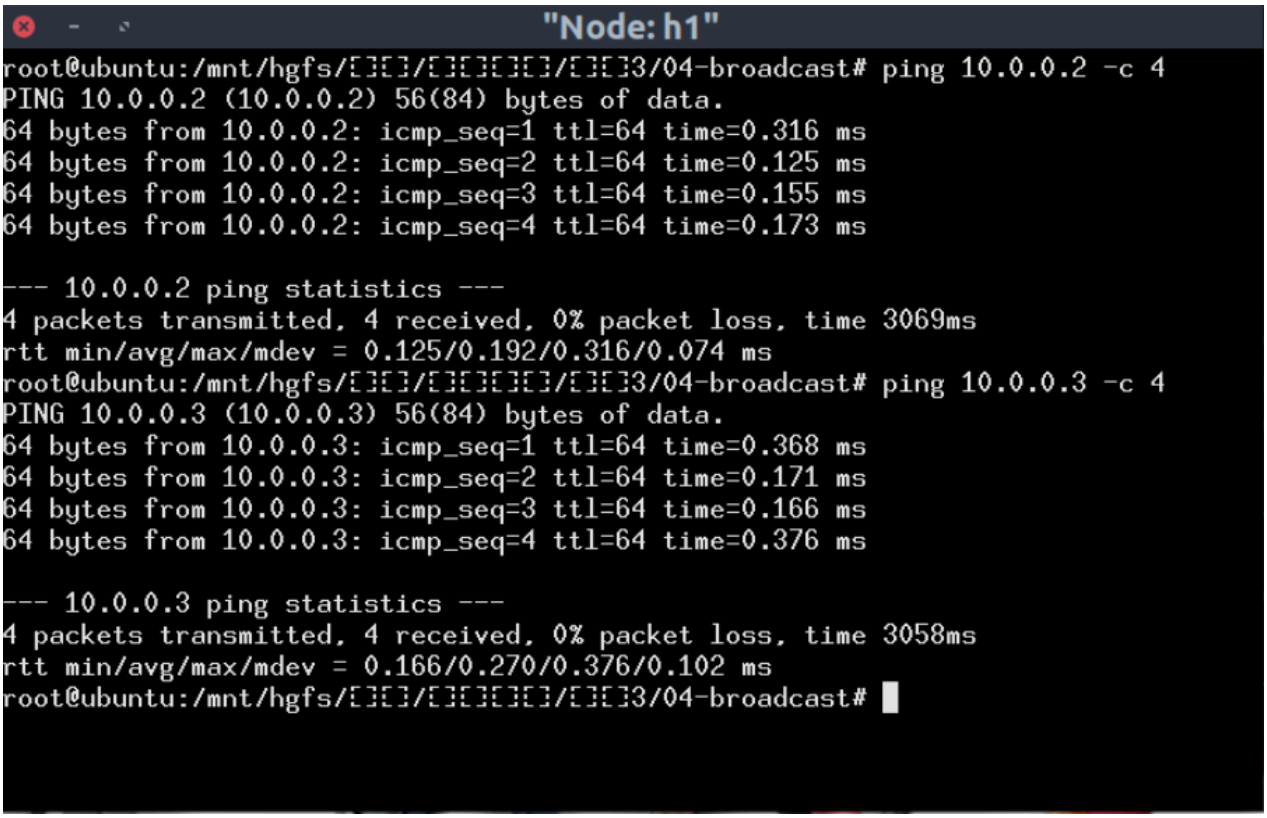
编写完成后，利用 Makefile 编译即可完成 hub 的制作。

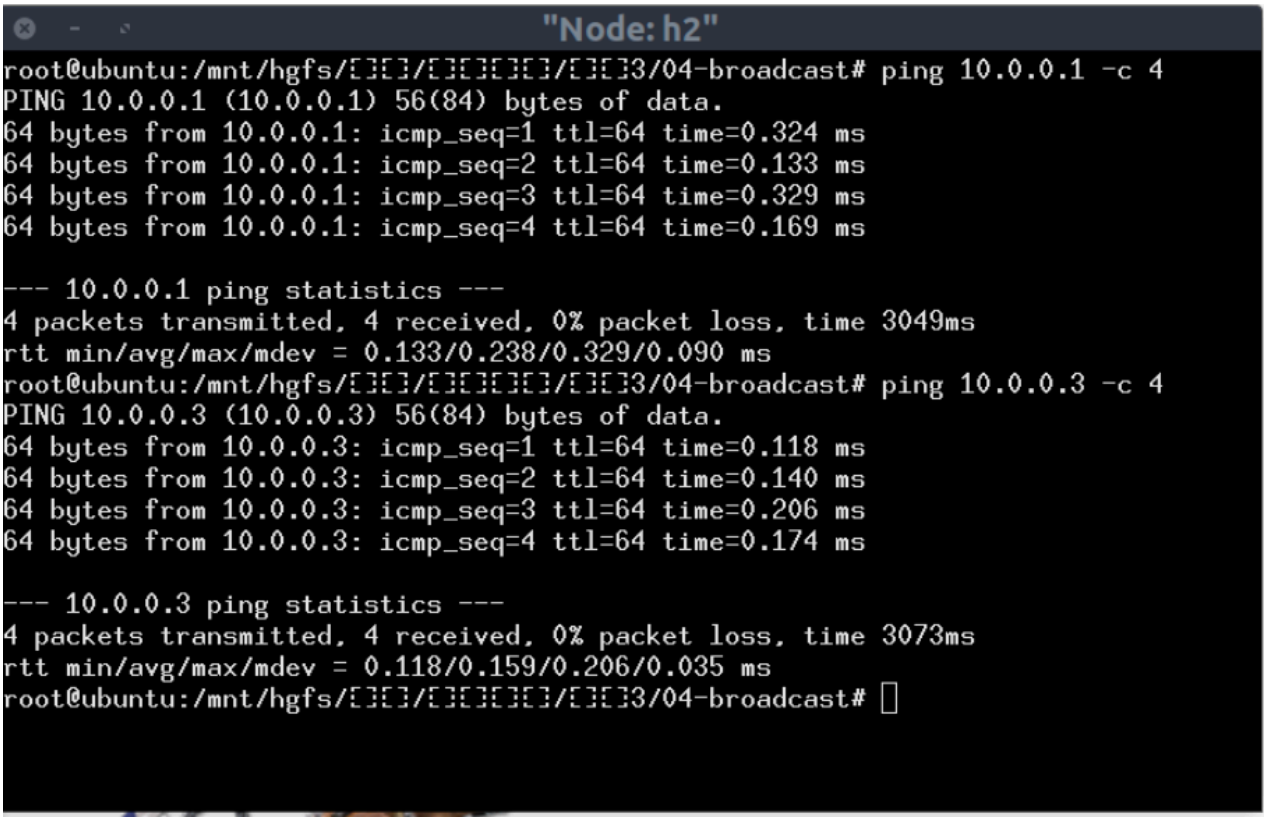
2. 验证广播网络

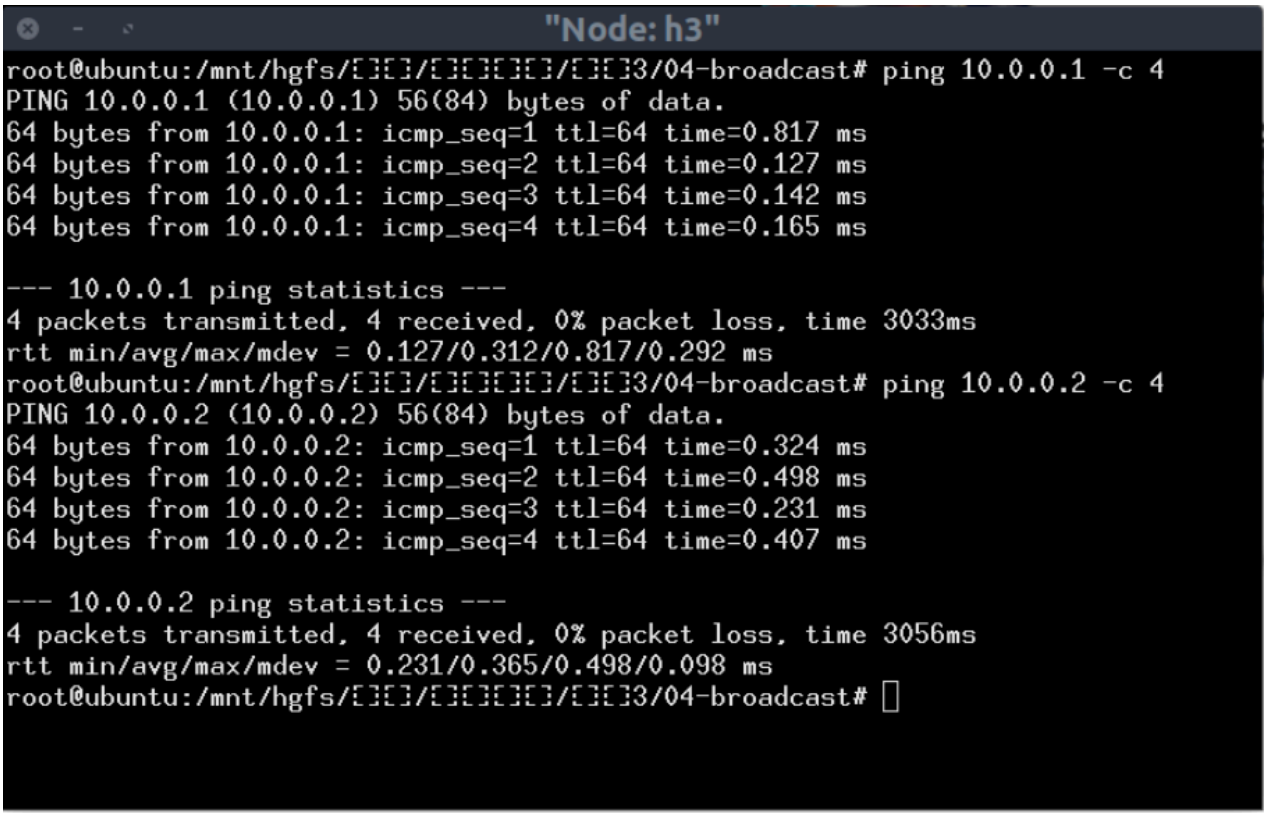
先将 three_nodes_bw.py 文件中的 hub 节点后台程序换成刚刚编写的版本：

```python
# three_nodes_bw.py
    # ...
    # b1.cmd('./hub-reference &')
    b1.cmd('./hub &')
```

然后启动 mininet，看看 h1, h2, h3 能不能相互 ping 通：


"Node: h1"
root@ubuntu:/mnt/hgfs/□□□/□□□□□□/□□□3/04-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=0.316 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.125 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.155 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.173 ms
--- 10.0.0.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3069ms
rtt min/avg/max/mdev = 0.125/0.192/0.316/0.074 ms
root@ubuntu:/mnt/hgfs/□□□/□□□□□□/□□□3/04-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=0.368 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=0.171 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=0.166 ms
64 bytes from 10.0.0.3: icmp_seq=4 ttl=64 time=0.376 ms
--- 10.0.0.3 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3058ms
rtt min/avg/max/mdev = 0.166/0.270/0.376/0.102 ms
root@ubuntu:/mnt/hgfs/□□□/□□□□□□/□□□3/04-broadcast#


"Node: h2"
root@ubuntu:/mnt/hgfs/□□□/□□□□□□/□□□3/04-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=0.324 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=0.133 ms
64 bytes from 10.0.0.1: icmp_seq=3 ttl=64 time=0.329 ms
64 bytes from 10.0.0.1: icmp_seq=4 ttl=64 time=0.169 ms
--- 10.0.0.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3049ms
rtt min/avg/max/mdev = 0.133/0.238/0.329/0.090 ms
root@ubuntu:/mnt/hgfs/□□□/□□□□□□/□□□3/04-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=0.118 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=0.140 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=0.206 ms
64 bytes from 10.0.0.3: icmp_seq=4 ttl=64 time=0.174 ms
--- 10.0.0.3 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3073ms
rtt min/avg/max/mdev = 0.118/0.159/0.206/0.035 ms
root@ubuntu:/mnt/hgfs/□□□/□□□□□□/□□□3/04-broadcast#



结果显示相互 ping 的过程是正常的。

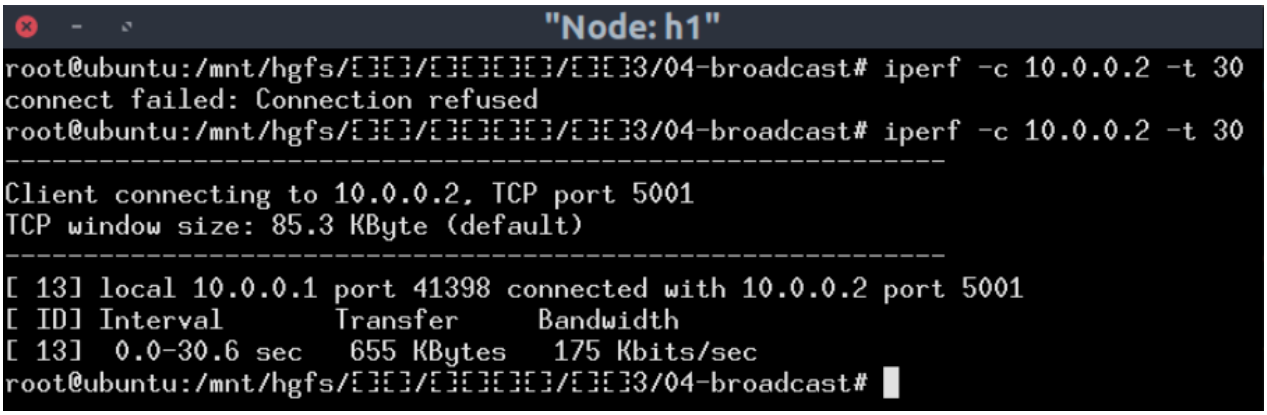
3. 验证广播网络的效率

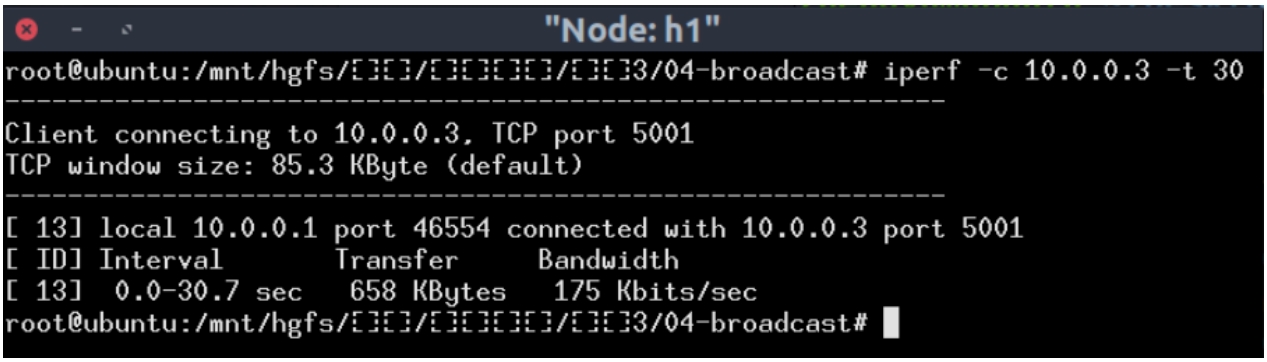
h2, h3 作为服务器

命令如下：

```
h2 # iperf -s
h3 # iperf -s
```

h1 连通 h2 和 h3:

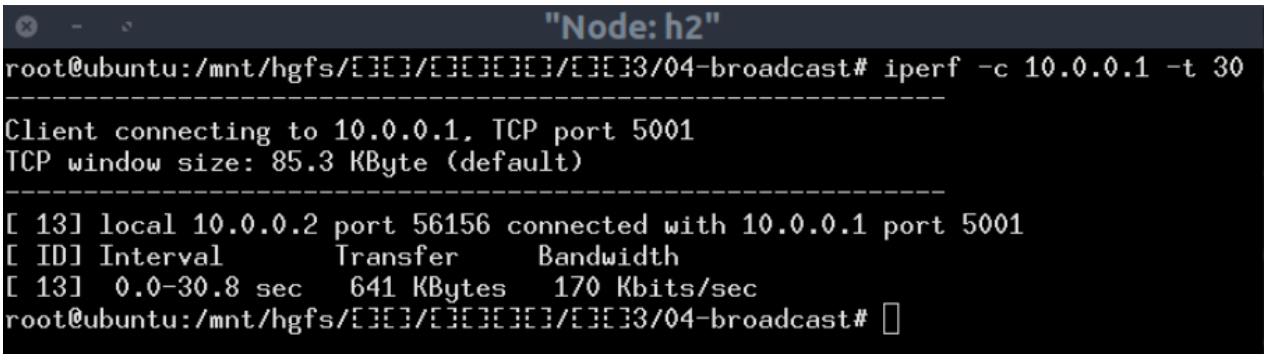


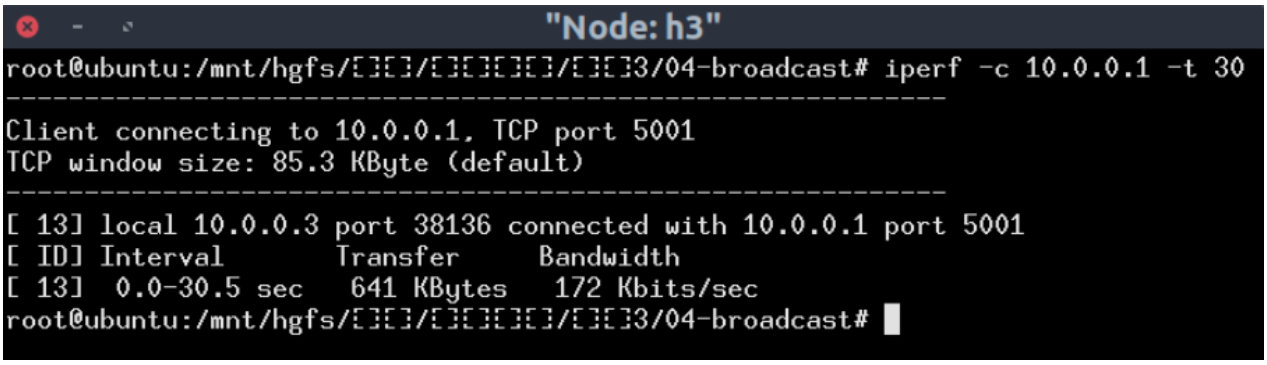


h1 作为服务器

```
h1 # iperf -s
```

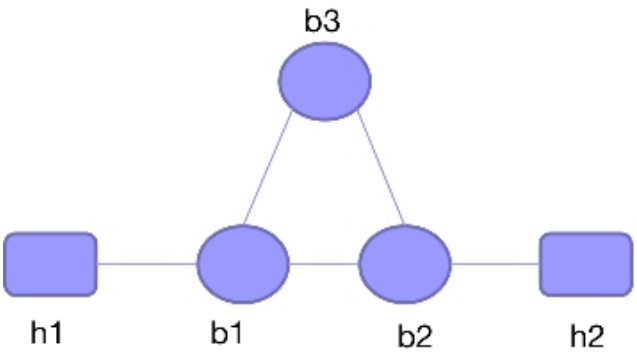
h2, h3 连 h1:





4. 环路转发

在上面这个 naive 广播网络中，如果出现环路，就会无限转发。以如下网络拓扑为例：



构建该网络的代码如下：

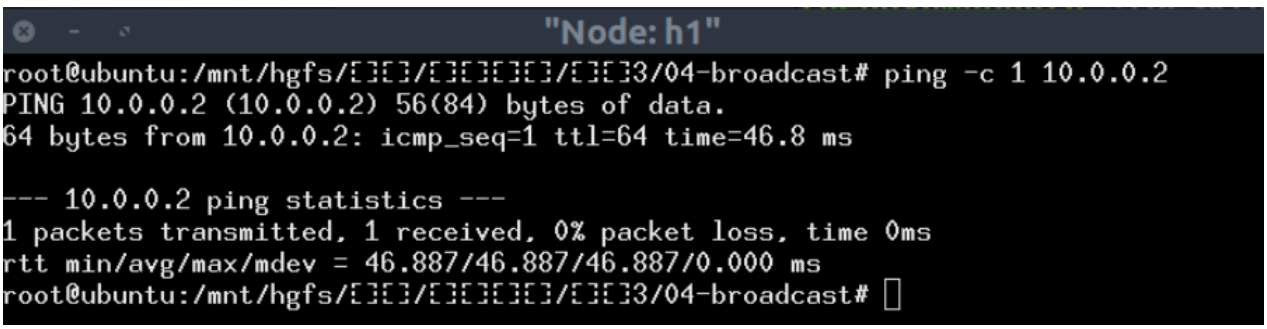
circle.py

```
#!/usr/bin/python

import sys
import os.path

from mininet.topo import Topo
from mininet.net import Mininet
from mininet.link import TCLink
from mininet.cli import CLI

# Mininet will assign an IP address for each interface of a node
# automatically, but hub or switch does not need IP address.
def clearIP(n):
    for iface in n.intfList():
        n.cmd('ifconfig %s 0.0.0.0' % (iface))

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

if __name__ == '__main__':
    if not os.path.exists('/sbin/ethtool'):
        print('ethtool not found, please install it using `apt install ethtool`')
        sys.exit(1)

    topo = BroadcastTopo()
    net = Mininet(topo = topo, link = TCLink, controller = None)

    h1, h2, b1, b2, b3 = net.get('h1', 'h2', 'b1', 'b2', 'b3')
    h1.cmd('ifconfig h1-eth0 10.0.0.1/8')
    h2.cmd('ifconfig h2-eth0 10.0.0.2/8')

    clearIP(b1)
    clearIP(b2)
    clearIP(b3)

    for h in [h1, h2]:
        h.cmd('./disable_offloading.sh')
        h.cmd('./disable_ipv6.sh')

    net.start()

    b1.cmd('./hub &')
    b2.cmd('./hub &')
    b3.cmd('./hub &')
    # b1.cmd('./hub-reference &')
    # b2.cmd('./hub-reference &')
    # b3.cmd('./hub-reference &')
    CLI(net)
    net.stop()
```

（其中大部分内容 copy 自 three_nodes_bw.py）

现在在 h1 节点 ping h2：



在 h2 打开 wireshark 跟踪网络包，会发现 h2 不断地在接收网络包，截取了其中一个部分如下：

```
18874 1.006950534   f6:ba:77:b5:c6:89    8a:1f:66:25:77:d2    ARP         42 10.0.0.2 is at f6:ba:77:b5:c6:89
18875 1.006951125   f6:ba:77:b5:c6:89    8a:1f:66:25:77:d2    ARP         42 10.0.0.2 is at f6:ba:77:b5:c6:89
18876 1.006998250   f6:ba:77:b5:c6:89    8a:1f:66:25:77:d2    ARP         42 10.0.0.2 is at f6:ba:77:b5:c6:89
18877 1.007030253   f6:ba:77:b5:c6:89    8a:1f:66:25:77:d2    ARP         42 10.0.0.2 is at f6:ba:77:b5:c6:89
18878 1.007057307   f6:ba:77:b5:c6:89    8a:1f:66:25:77:d2    ARP         42 10.0.0.2 is at f6:ba:77:b5:c6:89
18879 1.007093121   f6:ba:77:b5:c6:89    8a:1f:66:25:77:d2    ARP         42 10.0.0.2 is at f6:ba:77:b5:c6:89
18880 1.007442370   f6:ba:77:b5:c6:89    8a:1f:66:25:77:d2    ARP         42 10.0.0.2 is at f6:ba:77:b5:c6:89
18881 1.007442978   f6:ba:77:b5:c6:89    8a:1f:66:25:77:d2    ARP         42 10.0.0.2 is at f6:ba:77:b5:c6:89
18882 1.007443607   f6:ba:77:b5:c6:89    8a:1f:66:25:77:d2    ARP         42 10.0.0.2 is at f6:ba:77:b5:c6:89
18883 1.007444085   f6:ba:77:b5:c6:89    8a:1f:66:25:77:d2    ARP         42 10.0.0.2 is at f6:ba:77:b5:c6:89
18884 1.007444556   f6:ba:77:b5:c6:89    8a:1f:66:25:77:d2    ARP         42 10.0.0.2 is at f6:ba:77:b5:c6:89
18885 1.007445263   f6:ba:77:b5:c6:89    8a:1f:66:25:77:d2    ARP         42 10.0.0.2 is at f6:ba:77:b5:c6:89
18886 1.007445767   f6:ba:77:b5:c6:89    8a:1f:66:25:77:d2    ARP         42 10.0.0.2 is at f6:ba:77:b5:c6:89
18887 1.007446256   f6:ba:77:b5:c6:89    8a:1f:66:25:77:d2    ARP         42 10.0.0.2 is at f6:ba:77:b5:c6:89
18888 1.007446801   f6:ba:77:b5:c6:89    8a:1f:66:25:77:d2    ARP         42 10.0.0.2 is at f6:ba:77:b5:c6:89
18889 1.007447505   f6:ba:77:b5:c6:89    8a:1f:66:25:77:d2    ARP         42 10.0.0.2 is at f6:ba:77:b5:c6:89
18890 1.007937953   f6:ba:77:b5:c6:89    8a:1f:66:25:77:d2    ARP         42 10.0.0.2 is at f6:ba:77:b5:c6:89
18891 1.007939475   f6:ba:77:b5:c6:89    8a:1f:66:25:77:d2    ARP         42 10.0.0.2 is at f6:ba:77:b5:c6:89
```

实验结果与分析

本次实验主要还是在于过程体验，需要编写代码、构建网络的部分并不是很多，所以很大部分实验结果已经在流程中表达出来了。

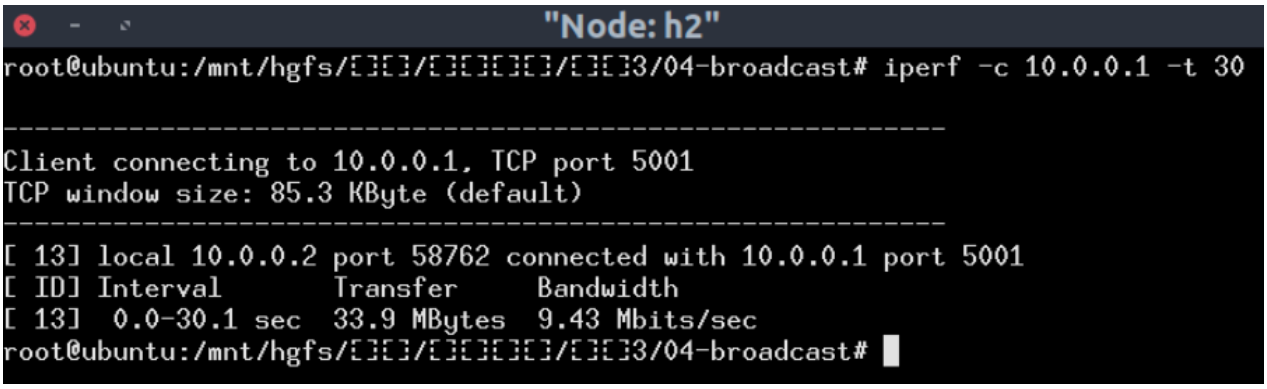
这次实验中遇到了一个当时（这部分报告撰写之前）发现的问题，一个是广播网络效率问题：我的网络广播 iperf 测试的带宽很低。后来我将节点广播函数中的 fprintf 函数的调用删去以后：

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

  // fprintf(stdout, "TODO: broadcast packet here.\n");
}
```

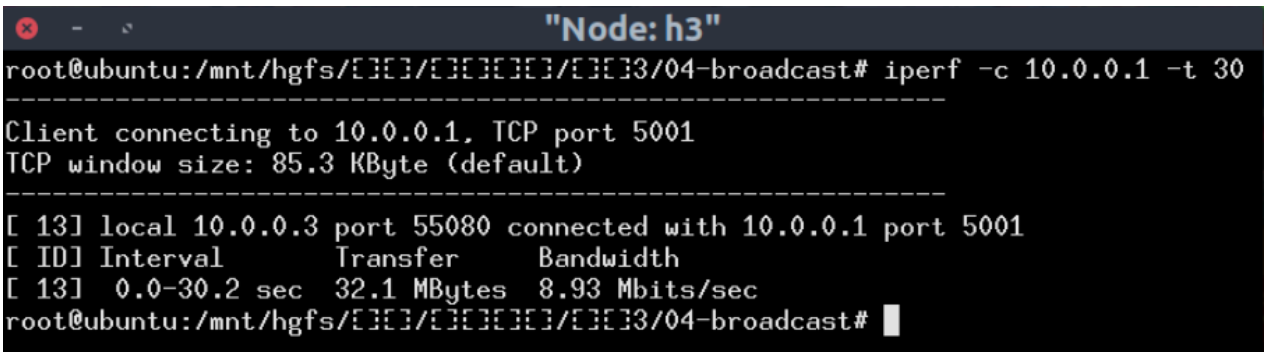
用修改后的 hub 完成的实验呈现的结果终于与 reference 相符：

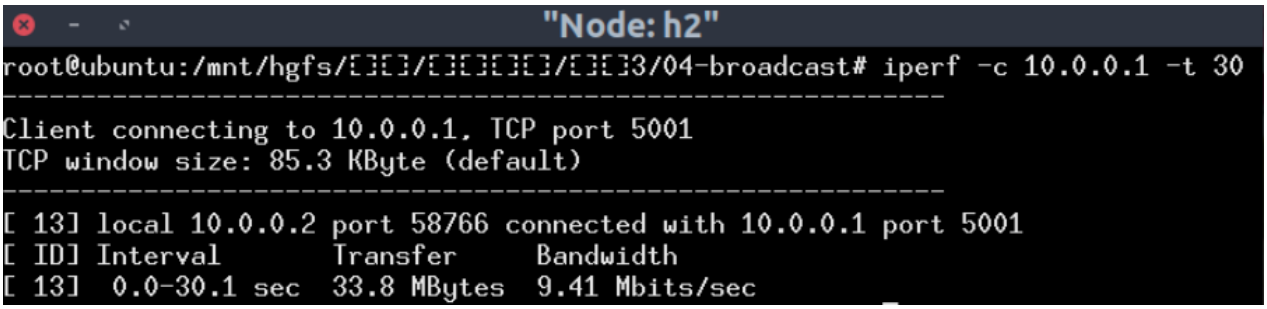
```
root@ubuntu:/mnt/hgfs/[illegible]/04-broadcast# iperf -c 10.0.0.2 -t 30
------------------------------------------------------------
Client connecting to 10.0.0.2, TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[ 13] local 10.0.0.1 port 40086 connected with 10.0.0.2 port 5001
[ ID] Interval       Transfer     Bandwidth
[ 13]  0.0-30.1 sec  33.8 MBytes  9.40 Mbits/sec
```

这是 h1 作为 client 连接 h2 的带宽：9.40Mbits/s. h1 连接 h3 server 带宽与此值相近。以下是 h1 作为服务器的情形：



和 h2 作为服务器几乎没有区别。当 h2, h3 同时连接 h1 时：





与 h1 作为服务器的情形几乎也相同，h3 先于 h2 执行 iperf，最终带宽稍低一些，总体上都小于 10Mbits/s。