计算机网络实验 4 报告

学号　2017K8009929059
姓名　於修远

广播网络实验

一、实验内容

- 实现节点广播的 broadcast_packet 函数。
- 验证广播能够正常运行。
- 验证广播网络的效率。
- 构建环形拓扑，并验证该拓扑下节点广播会产生数据环路。

二、实验流程

（一）节点广播构建

1、broadcast_packet 函数编写

如下图所示，调用宏定义中的方法，遍历全局列表中的每个结点，pos 指针暂存处理中的当前列表节点。然后，除了当前接受消息的节点端口外，向所有端口转发接收到的消息。本实现中使用了`list_for_each_entry_safe `的遍历方式，尽管在目前的操作下使用` list_for_each_entry `可以实现同样的效果，但出于扩展性考虑还是选择了 safe 模式的遍历，方便后续增加功能拓展。

```
iface_info_t *pos, *q; //temp iface for traversal
list_for_each_entry_safe(pos, q, head: &instance->iface_list, list) { //traverse global list
    if (pos != iface) { // dont send to the receiving entry
        iface_send_packet(pos, packet, len); // send packet
    }
}
```

2、广播网络连通性测试

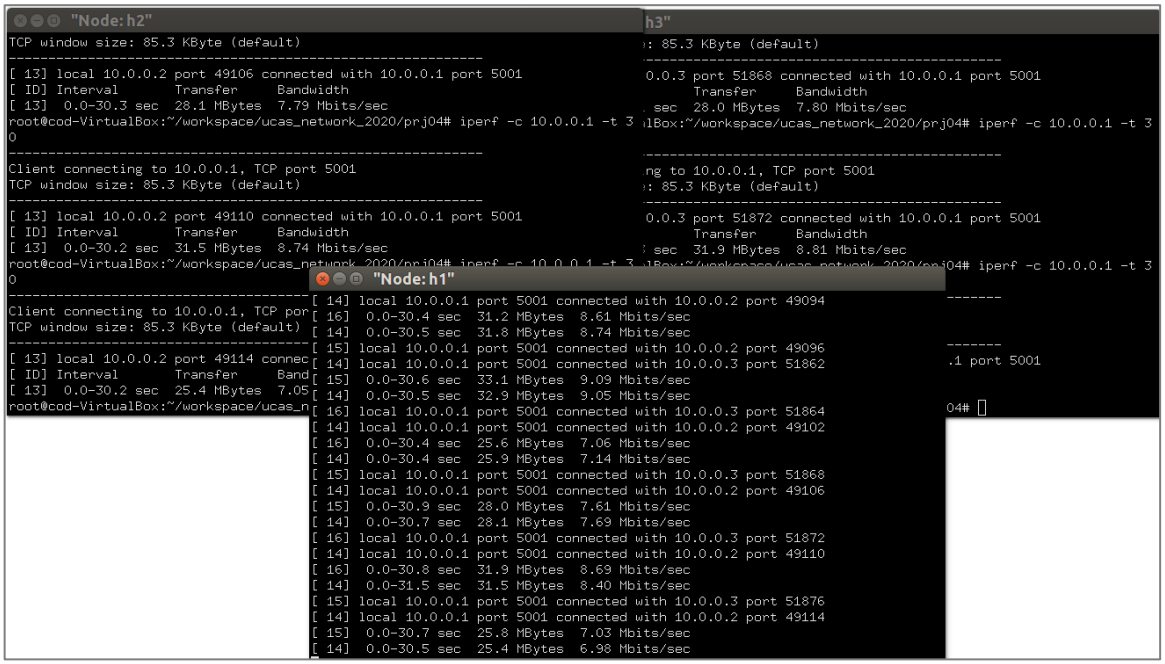
运行 python 脚本启动 mininet 环境后，在 b1 结点中启动使用 main.c 编译得到的 hub 文件，形成一个连通 3 个主机节点的 hub 结点。随后对 3 个主机节点互相执行 ping 指令，检查连通性。如下图所示为和 h1 向 h2、h2 向 h3、h3 向 h1 分别执行 ping 指令的结果，可以看到连接均有效。反向 ping 依然有效，所得结果的截图省略。

```
mininet> xterm b1
mininet> h1 ping h2 -c 2
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=0.324 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.082 ms

--- 10.0.0.2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1032ms
rtt min/avg/max/mdev = 0.082/0.203/0.324/0.121 ms
mininet> h2 ping h3 -c 2
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=0.235 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=0.135 ms

--- 10.0.0.3 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1026ms
rtt min/avg/max/mdev = 0.135/0.185/0.235/0.050 ms
mininet> h3 ping h1 -c 2
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=0.273 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=0.068 ms

--- 10.0.0.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1015ms
rtt min/avg/max/mdev = 0.068/0.170/0.273/0.103 ms
mininet>
```

（二）广播网络效率的测量

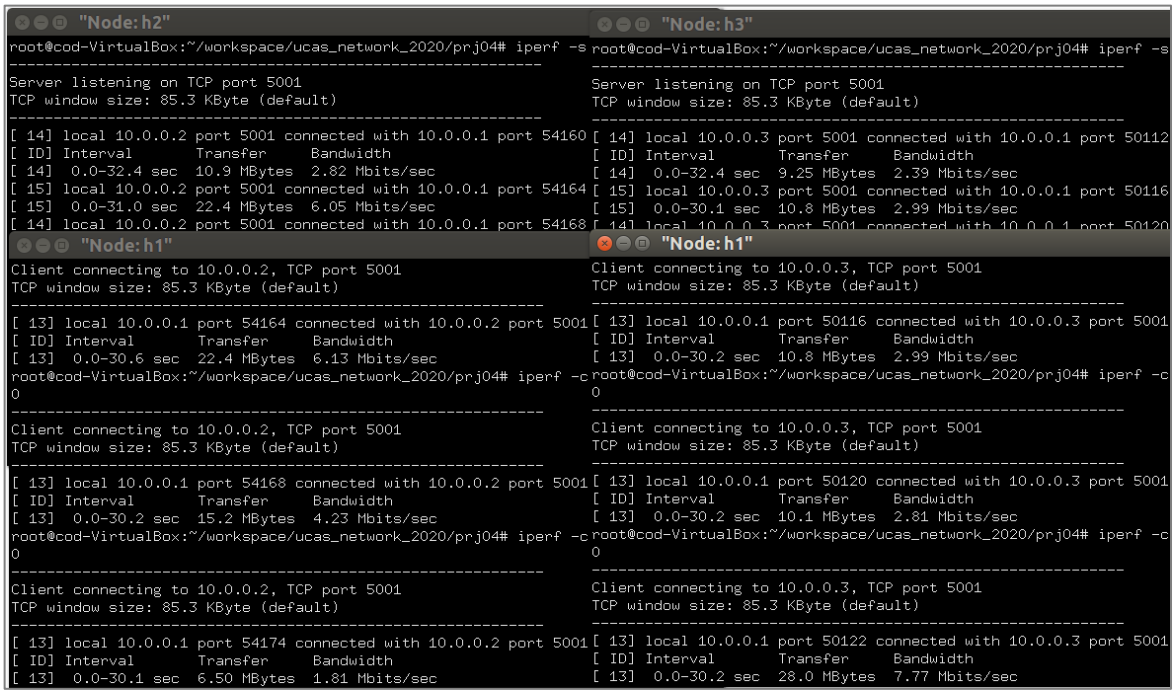
1、以 h1 为 server 执行 iperf

如下图所示，在 h1 节点执行` iperf -s `命令，在 h2 和 h3 节点同时执行` iperf -c 10.0.0.1 -t 30 `命令，观察得到的带宽测试结果。可以看到每次 iperf 的测量值都有变化，取 3 次平均得到 h1 和 h2 和 h3 到 hub 节点 b1 的平均带宽均为 7.91Mbps，即带宽利用率约为 79.1%。



2、以 h1 为 client 执行 iperf

如下图所示，在 h2 和 h3 节点执行` iperf -s `命令，在 h1 节点同时执行` iperf -c 10.0.0.2 -t 30 `和` iperf -c 10.0.0.3 -t 30 `命令，观察得到的带宽测试结果。可以看到每次 iperf 的测量值都有变化，取 3 次平均得到 h1 与 b1 的平均带宽约为 8.87Mbps，h2 与 b1 的平均带宽约为 4.06Mbps，h3 与 b1 的平均带宽约为 4.81Mbps，即带宽利用率分别约为 44.4%，40.6%，48.1%。



（三）数据环路的复现

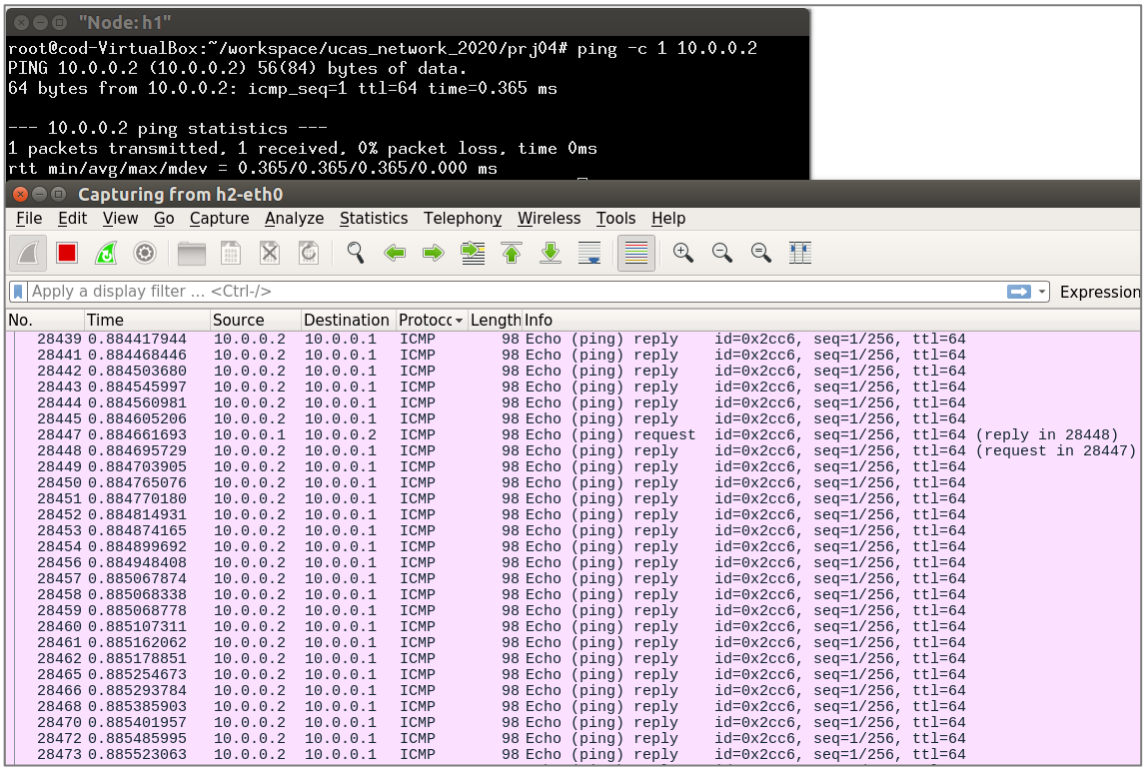
1、round_nodes_bw.py 的编写

复现数据环路的 python 脚本基本基于` three_nodes_bw.py `得到。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。

```
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)
```

2、广播网络连通性测试

用上述 python 脚本创建 mininet 环境后，在 b1 和 b2 和 b3 结点中分别启动使用 main.c 编译得到的 hub 文件，形成一个 3 个 hub 节点互联且可以互相广播的拓扑。随后在主机节点 h2 中启动 wireshark，监控` h2-eth0 `端口；并在主机节点 h1 中执行 ping 指令，检查连通性。如下图所示，可以看到 h1 节点接收到了 ping 包的返回。观察 wireshark 的抓包结果，可见 ping 请求和 ping 答复包都被循环转发，且后者在环路中所占比重大得多。



三、结果分析

（一）广播网络的效率

从先前的计算结果中可以看到，在仅以 h1 为 server 端时测得的有效带宽明显高于仅以 h1 为 client 端时的有效带宽。这一点可以这样解释：当 h1 向 h2 进行 iperf 操作发送数据包时，由于 hub 的广播效应，实际的效果是 h1 向 h2 和 h3 同时发送数据包；h1 向 h3 进行 iperf 操作时也有同样的效果。由于 h2 和 h3 连接 b1 节点的带宽上限都是 10Mbps，所以 h1 向两个节点发送数据包的总带宽不能超过 10Mbps，而非 h1 与 b1 之间的带宽上限 20Mbps。检视测量数据，可以发现对 h2 和 h3 的带宽求和，结果分别为 9.12Mbps，7.04Mbps，9.58Mbps，均满足上述解释。而当 h2 和 h3 向 h1 执行 iperf 操作时，即使 h2 的实际带宽达到上限 10Mbps，由于带宽双向，h3 的发送不会受到影响；h3 同理。而二者的实际带宽总和也不超过 20Mbps，即 h1 和 b1 的带宽上限，所以此时测量带宽不会受到 hub 广播效应的影响，测出的实际带宽和带宽利用率也就更高。

另外，以 h1 为 server 端时，h2 及 h3 与 b1 之间的实际带宽的均值距离带宽上限任有一定距离。这点可以解释为 TCP 慢启动机制的影响，同时 hub 节点的处理时间也不可忽略。尝试将广播函数中的遍历方式改为非` safe `的后，可以看到带宽测量值有了显著提高，平均达到了 9Mbps 左右。

（二）数据环路的产生

考虑如下图所示的一个网络拓扑。当 h1 向 h2 发送 ping 包时，最直接的通路是 h1→b1→b2→h2，设此时收到的包为 P1。但由于环形拓扑的存在，数据包在 b1 同样也会向 b3 进行发送，被转发向 b2 后又会被拷贝一份发送到 b1，继而转发到 b3，从而形成一个顺时针的环路，环路中始终有一个 ping 包在发送；同理，直接通路中 b2 也会向 b3 广播一份数据包，从而类似地形成一个逆时针的环路。在这两个环路中，b2 每次收到来自 b1 或 b3 的数据包，都会向 h2 转发一份 ping 包，所以 h2 会源源不断地收到完全一样的数据包。

同时，由于 h2 始终以为这是来自 h1 的 ping 包，所以相应地会不断发送 ping 响应包。这一数据包也会与初始的 ping 包一样形成两个环路，并在每次经过 b1 时向 b1 发送一份拷贝。虽然 h1 的 ping 接收显示完成，但其实后续还在不断收到来自 h2 的响应数据包。由此可见数据环路作用下会造成主机与 hub 节点间链路资源的极大浪费。