计算机网络课程大作业报告

王宇奇 李俊儒

The First Half

Overview

本次课程大作业前半部分总共有 8 个 Step，具体需要实现的是 Step1、Step2、Step7、Step8。剩下的一些是配置环境和代码审核的工作，这里只做简要提及。Step1 和 Step2 是利用 java sockets 实现自己版本的 ping 和 iperf，之后在 mininet 环境下中进行使用。而 Step7 是实现路由器的数据层，Step8 是实现路由器的控制层。

Step1

利用 Java Sockets(具体为 UDP packets) 实现自己的 pinger，pinger 会向某个主机发送数据包，并等待数据包的回传，最后计算 RTT。在 Java 下的运行结果如下:

```
Runing client!
size=12 from=127.0.0.1 seq=1 rtt=203.0 ms
size=12 from=127.0.0.1 seq=2 rtt=0.0 ms
size=12 from=127.0.0.1 seq=3 rtt=0.0 ms
size=12 from=127.0.0.1 seq=4 rtt=0.0 ms
size=12 from=127.0.0.1 seq=5 rtt=0.0 ms
size=12 from=127.0.0.1 seq=6 rtt=0.0 ms
size=12 from=127.0.0.1 seq=7 rtt=0.0 ms
size=12 from=127.0.0.1 seq=8 rtt=0.0 ms
size=12 from=127.0.0.1 seq=9 rtt=0.0 ms
size=12 from=127.0.0.1 seq=10 rtt=0.0 ms
sent=10 received=10 lost=0% rtt min,avg,max=0.0,20.3,203.0ms
Client stop!
```

Figure 1 Pinger Client

```
Running Server
time=1529304089583 from=127.0.0.1 seq=1
time=1529304091098 from=127.0.0.1 seq=2
time=1529304092114 from=127.0.0.1 seq=3
time=1529304093129 from=127.0.0.1 seq=4
time=1529304094145 from=127.0.0.1 seq=5
time=1529304095160 from=127.0.0.1 seq=6
time=1529304096176 from=127.0.0.1 seq=7
time=1529304097176 from=127.0.0.1 seq=8
time=1529304098192 from=127.0.0.1 seq=9
time=1529304099207 from=127.0.0.1 seq=10
Server Crashed
Server stop!
```

Figure 2 Pinger Server

Step2

利用 Java Sockets 实现 iperf，其中用的是 TCP packets。Iperfer 会在某一固定时间内向某个主机传输 TCP 包，然后用来计算这段时间内线路的带宽。在 Java 下的运行结果如下：

```
Client start!
sent=120368.0 KB rate=96.29200801985799 Mbps
Client stop!

Process finished with exit code 0
```

Figure 3 Iperfer Client

```
Server start!
received=120368.0 KB rate=96.04801962745526 Mbps
Server stop!

Process finished with exit code 0
```

Figure 4 Iperfer Server

Step3

Step 3 是用来配置 mininet 环境。建议的方法是在 mininet 官网上的第一种方法，即 VM 导入虚拟机的方法。不建议在 linux 环境下安装，因为会因为版本产生各种问题。同时建议焊装 2.2.0 的版本，因为后面的任务中需要较低版本的 mininet。

Step4

在这个步骤中，之前在 Step1 和 Step2 中写的 pinger 和 iperfer 会在 mininet 环境下被检验。所有检验的结果文件我们已经附在了附件中。这里可以做一个简单的展示。

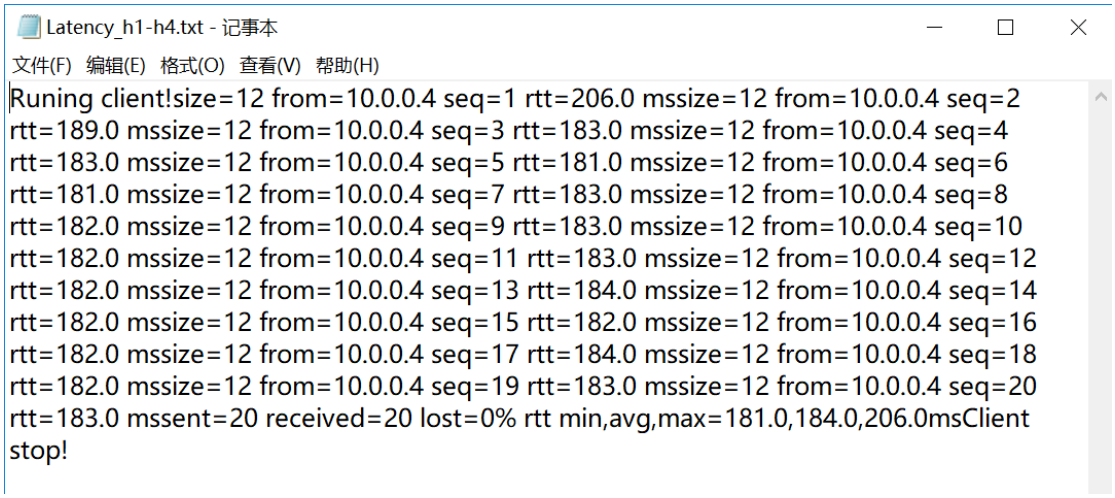

Figure 5 Latency_h1-h4

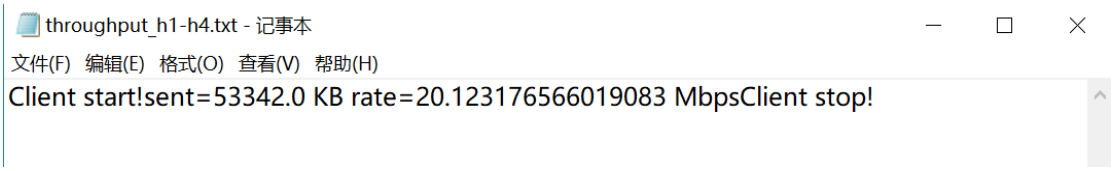

Figure 6 Throughput_h1-h4

Step5

这个步骤依然是配置环境的步骤，其中 ARP 和 ICMP 与课本上一致。Checksum 要注意需要在转发修改报文 header 之后重新生成。POX 是一个软件，提供了 mininet 的路由器与外界运行的路由器实例建立连接。

Step6

代码审核。

Step7

这部分主要来说实现 HandlePacket 的 IP forward、ARP、RIP interface、ICMP 四个功能。

```
/** * Handle an Ethernet packet received on a specific interface. * @param
public void handlePacket(Ethernet etherPacket, Iface inIface)
```

Figure 7 HandlePacket

1. rip.handlePacket
   a) Step7 中只是调用这个接口。
2. Forwarding
   a) 解析获得目的 IP
   b) 最长前缀匹配查找 RouteTable
   c) 修改链路层源 MAC-转发出口 MAC 地址
   d) 根据转发表，确定下一跳的地址，0 代表下一跳地址即为 Packet 的目的地
   e) 调用 ARP 确定下一跳 IP 代表的 MAC 地址
   f) 设置链路层信息并转发

```
private void forwardIpPacket(Ethernet etherPacket, Iface inIface)
```

Figure 8 forwardIpPacket

3. ICMP
   a) 根据不同的情况调用
   b) 在调用 sendICMP 时如果发生错误
      i. 转发表未找到转发项，直接 return
      ii. 如果 ARP 表为空，调用 arpCache.waitForArp 等待查询
   c) 注意各层设置顺序，Top-down 会导致 checksum 的问题
   d) 代码中的 ICMP 消息

```
public void sendICMP(int type, Ethernet etherPacket)
```

Figure 9 sendICMP

```
TIME_EXCEEDED: Type 11 byte 0
DEST_NET_UNREACHABLE: Type 3 byte 0 网络层
DEST_HOST_UNREACHABLE: Type 3 byte 1 链路层
DEST_PORT_UNREACHABLE: Type 3 byte 3 给router发消息是无效的
ICMP_ECHO_REPLY: Type 0 byte 0
```

Figure 10 **代码中的** ICMP **消息**

4. ARP
    a) 处理 ARP.OP_REQUEST
        i. 回复 ARP 请求
    b) 处理 ARP.OP_REPLY
        i. 更新 ARP 表
        ii. 更新表的时候返回等待列表发送等待列表中的 Packet
    c) ARP 定时查询
        i. Java Thread 定时 sleep, sleep 之后发送一遍广播
        ii. 超过次数交给 ICMP 处理
5. header 检查
    a) TLL 超时 ICMP
    b) Checksum 直接返回
    c) TLL 之后 checksum 需要重新定义
6. 最长前缀匹配 getBest()
    a) 遍历
    b) 位运算

```
private RouteTableEntry getBest(int dstAddr){
        List<RouteTableEntry> fuck = routeTable.getEntries();
        RouteTableEntry bestMatch =null;
        long t = 0;
        for (RouteTableEntry x : fuck){
                int ccc1 = ( x.getMaskAddress() & dstAddr );
                int ccc2 = ( x.getDestinationAddress() & x.getMaskAddress() );
                if (ccc1 == ccc2){
                        if (x.getMaskAddress()<t){
                                System.out.println("Find best match!");
                                t = x.getMaskAddress();
                                bestMatch = x;
                        }
                }
        }
        return bestMatch;
}
```

Figure 11 getBest()

Step8

1. 一些注意的点
   a) UDP 的 520 端口
   b) 广播 IP 为 224.0.0.9 广播地址为 FF:FF:FF:FF:FF:FF
   c) 初始化的时候发送 RIP 查询请求
   d) 每 10s 发送一次通告
   e) 除了直接相连其余 30s 失效
   f) 为了防止表过大,需要有一个 maxinum=16 意思时为只记录长度小于等于 16 的转发表项
2. 收到 RIP Packet
   a) 收到 Request
      i. 返回自己的表
   b) 收到 Response
      i. 遍历
      ii. 更新转发表
         1. 路径长度+1
         2. 如果找到相同的项目,更新
         3. 如果没有相同的项目,并且长度小于 maxhop,add
         4. 如果没有相同项目,长度大于 maxhop,如果不存在转发项(能够转发,即满足最长匹配),add(hoptim=maxhop)
         5. 否则,什么也不需要做
3. 发送 RIP Packet
   a) 分情况讨论即可
      i. 正常按时通告
      ii. 在初始化的时候,发送广播查询
      iii. 回答 RIP 询问定向的
   b) 其他说明
      i. 毒性逆转,有助于加速 RIP 的实时更新,但是对于复杂情况效果一般
      ii. 定时刷新,定时超时,与 Step7 中 ARP 定时询问类似

The Second Half

Overview

大作业的第二部分主要是实现 SDN（软件定义网络），虽然总共只有三个 Step，但是后面两个 Step 工作量还是比较大的。其中 Step1 是在原先 mininet 的环境上配置新的环境。Step2 是在 Step1 基础上实现 routing，只不过用的信息是全局的信息。而 Step3 则是实现分布式负载均衡。

Step1

这个步骤是配置环境，建议直接使用 mininet 2.2.0 VM，这样后面的配置会比较容易一些。

Step2

1. 相比大作业前半部分的改变
    a) 不再是分布式的最短路了，而是全局最短路
    b) 不再使用路由表，使用通用的流表，即路由表+MAC 表
        i. 有不同的匹配规则
        ii. 根据 Ethernet IP TCP UDP 等
    c) 重要的函数，见 Code Overview
    d) 删除流表的内容
    e) 添加流表的内容
    f) 利用全局最短路算法，求得转发表，为每一个 switch 添加转发项目
    g) 规则的其余问题
        i. 超时
        ii. 优先级
    h) 试验过程很简单，ping 一下就 ok 了，然后查看一下流表内容
2. Device
    a) deviceAdded (updateHost)
    b) deviceRemoved
        i. 删除设备
        ii. 其余 switch 中（removeRule)
    c) deviceMoved (updateHost)
    d) switchAdded (updateAllHost)
    e) switchRemoved (updateAllHost)
    f) computeShortestPaths 在每次更改之后都会被调用

```
private void updateAllHosts() {
  Collection<Host> hosts = this.getHosts();
  Map<Long, IOFSwitch> switches = this.getSwitches();
  this.predMaps = new ConcurrentHashMap<Long,Map<Long,Link>>();
  for (Long source : switches.keySet()) {
      this.predMaps.put(source, computeShortestPaths(source));
          for (Host host : hosts) {
              removeRules(switches.get(source), host);
          }
      }
  for (Host host : hosts) {
      updateHost(host);
  }
}
```

Figure 12 updateAllHosts()

3. computeShortesPaths
    a) 单源多汇最短路
    b) 记录路径即可
    c) 根据路径更新每个路由器

Step3

1. Install rules
    a) 发起与虚拟 IP 的链接时，交给 Controller

b) 对于虚拟 IP 的 MAC 地址的 APR 查询需要回复
c) 对于虚拟 IP 通信，需要重写 header
d) 优先级更高
e) 20s 删除

2. switchAdded
   a) 为虚拟 IP 添加发给 controller 的规则低优先级
   b) ARP 查询
   c) 其余的规则

```
ArrayList<OFInstruction> instructions = null;
for (Integer vip : this.instances.keySet()) {
        OFMatch matchCriteria = new OFMatch();
        matchCriteria.setDataLayerType(OFMatch.ETH_TYPE_IPV4);
        matchCriteria.setNetworkDestination(vip);
        OFAction action = new OFActionOutput(OFPort.OFPP_CONTROLLER);
        ArrayList<OFAction> actions = new ArrayList<OFAction>();
        actions.add(action);
        OFInstruction instruction = new OFInstructionApplyActions(actions);
        instructions = new ArrayList<OFInstruction>();
        instructions.add(instruction);
        SwitchCommands.installRule(sw, table, (short) (SwitchCommands.DEFAULT_PRIORI
}
OFMatch blk = new OFMatch();
instructions = new ArrayList<OFInstruction>();
OFInstructionGotoTable instruction = new OFInstructionGotoTable(L3Routing.table);
instructions.add(instruction);
SwitchCommands.installRule(sw, table, SwitchCommands.DEFAULT_PRIORITY, blk, instruct
```

Figure 13

3. 指令处理

```
if (ethPkt.getEtherType() == Ethernet.TYPE_ARP)
        handleArp(sw, ethPkt, pktIn);
        if (ethPkt.getEtherType() == Ethernet.TYPE_IPv4) {
                IPv4 ipv4Pkt = (IPv4) ethPkt.getPayload();
                if (ipv4Pkt.getProtocol() == IPv4.PROTOCOL_TCP) {
                        TCP tcpPkt = (TCP) ipv4Pkt.getPayload();
                if (tcpPkt.getFlags() == TCP_FLAG_SYN)
                        handleTcpSyn(sw, ipv4Pkt);
        }
}
```

Figure 14 **指令处理**