计算机网络 router 实验报告

软件92 周雨豪 2018013399

1 实验环境

操作系统：Ubuntu 16.04.7

内存：4GB

2 实验方法

2.1 RoutingTable::lookup()

此部分无复杂逻辑，只需要实现最长前缀匹配算法，对路由表中的每一项均将项IP与掩码与运算的结果和参数IP与掩码与运算的结果比较，若相等则计算匹配的长度，方法是用variable precision SWAR算法求结果中比特为1的位数。选择匹配最长的项作out entry。

2.2 ArpCache::periodicCheckArpRequestsAndCacheEntries()

该函数被`ticker()`调用，定时检查ARP请求和cache表项，用一个vector存储`sent_time`超出max的ARP请求，并对每个失效的请求发送`host_unreachale`的ICMP；而未失效的请求就广播ARP request。接着用另一个vector存储失效的cache表项并全部移除。

2.3 SimpleRouter::handlePacket()

该部分逻辑相较于前者要复杂很多，也是本次实验的主要内容。简要逻辑如下：

> 从packet中拆出ether header，判断ether类型
>
> - **若是ethertype_ip**，判断IP header合法性并检查目的IP
>   - **若是router本身**，则判断协议种类
>   - **若是ICMP**，判断ICMP header合法性。如果是type和code均为echo，回复echo reply
>   - **否则** (TCP或UDP)，发送`port_unreachale`的ICMP。
>   - **否则**，作为普通的IPv4数据包处理，如果TTL失效或路由表无对应表项，则分别发送对应的无效ICMP信息，如果ARP cache无对应表项，将新的ARP请求加入队列。建立ether帧并转发。
> - **若是ethertype_arp**，进入`handleArp()`，判断ARP header合法性并检查ARP类型
>   - **若是ARP request**，则回复ARP reply
>   - **若是ARP reply**，则添加ARP cache表项，清除队列中对应的request

3 实验结果

通过mininet中的ping、traceroute、wget测试以及autograde.py测试脚本。

4 实验问题 & 解决方案

- 在刚开始测试的时候router输出" Interface list empty "，阅读实验文档发现可以通过重启mininet解决。
- 运行autograde.py或run.py偶尔会报错" Error creating interface pair: RTNETLINK answers: File exists "，只能通过重启解决。
- 在mininet启动后，router输出显示有ethertype无法识别的packet，查RFC文档得知是IPv6 (0x86DD)，并没有很清楚为什么有IPv6包，但全部作忽略处理即可不影响功能。
- 对于ICMP校验和的计算仍存有疑问，因为不同于IP校验和只用header计算，ICMP校验和计算是包括数据段的，但实验提供的`cksum()`函数应该只根据header计算。但实际使用并没有影响到功能（我也不知道为什么没有影响）。
- 测试功能的时候只能通过ping，而traceroute和wget均不通过，找了很久最终发现问题是在`handleIPv4()`中判断目的IP是否为router本身是，把判断条件写反了...

5 实验建议

- 实验后期阶段有安排在周三上午下课后教室内的集体自习和助教答疑，个人认为这个环节可以安排在实验前期，因为启动一个项目的进度往往比较缓慢而且困难，前期应该比后期更需要指导（虽然这个建议好像并不是针对实验内容本身，只是一点想法）。
- 实验文档中关于测试使用工具的说明部分也许可以写得更简明易读但更具体实用。

6 其他

- 额外使用的libraries：无