Simple Router 实验报告

基本信息

- 姓名：卢敏思
- 学号：2019013240

实验背景与目的

本项目要求实现一个简单的路由器（Simple Router），来加深对网络层和链路层协议的理解，熟悉路由器的基本工作流程和转发逻辑。

在该实验中，我们使用由教学团队提供的`simple-router`骨架代码，并根据需求完善其处理ARP报文、IPv4报文（包括ICMP、TCP和UDP的简单处理）以及ARP缓存（Arp Cache）和路由表（Routing Table）匹配逻辑。最终我们期望实现一个在Mininet和POX控制器环境下正常运转的简单路由器，通过ARP解析获取下一跳MAC地址并成功转发分组。

实验环境

- 使用Mininet模拟网络拓扑
- 使用POX作为控制器
- 使用Ice框架进行RPC交互
- Ubuntu 16.04.7

设计思路

整体流程

1. **接收分组**：通过`handlePacket`函数处理POX转发过来的以太网帧。
2. **以太网头解析**：检查`ether_type`、`ether_dhost`。如果`ether_dhost`是路由器接口的MAC或是广播地址，则进一步处理，否则直接丢弃。
3. **分类型处理**：
    - 若`ether_type == ethertype_arp`，调用`handleArp`。
    - 若`ether_type == ethertype_ip`，调用`handleIPv4`。

ARP处理逻辑

`handleArp`中先对ARP报文合法性进行检验：

- 检查packet size、hardware type、protocol type、hw addr len、proto addr len、opcode.

若是ARP Request且目标IP正是路由器接口IP，则发送ARP Reply：

- 填写ARP Reply中的`arp_sip`为本路由器接口IP，`arp_sha`为路由器接口MAC，`arp_tip`为request发送者的IP，`arp_tha`为request发送者的MAC。
- 将Ethernet头中的源目的MAC地址对调后发送Reply。

若是ARP Reply：

- 解析Reply中的`sender_ip`和`sender_mac`，调用`m_arp.insertArpEntry`更新ARP缓存。
- 同时清理过期或无效的ARP请求。

IPv4处理逻辑

`handleIPv4`中先检查IPv4头部是否合法：

- 检查IP报文长度、`ip_sum`checksum。
- 若报文目标IP为路由器本身：
  - 根据协议字段`ip_p`判断类型：
    - ICMP：处理ICMP Echo请求，发送ICMP Echo Reply。
    - TCP/UDP：由于路由器不实现完整TCP/UDP处理，此处只能发送ICMP Port Unreachable响应，表示该端口不可达。
  - 若是不支持的协议类型则忽略。
- 若报文目标不是本路由器：
  - 检查`ip_ttl`，若`ttl <= 1`，发送ICMP Time Exceeded报文。
  - 调用routingTable.lookup，查找目的IP对应的MAC地址。如果arp_table里没有，则queueRequest。
  - 若`ttl > 1`则将`ttl`减1，并通过`forwardIPv4`执行路由查表、ARP查询和分组转发。

路由表匹配逻辑

`m_routingTable.lookup(ip)`实现最长前缀匹配：

- 遍历路由表中每条目，检查`(entry.dest & entry.mask) == (ip & entry.mask)`。
- 保留匹配掩码最长的一条（mask值最大），返回该条目。
- 若找不到匹配路由，抛出异常“Routing entry not found”。

ARP缓存与请求队列维护

`ArpCache`中对ARP请求和缓存进行周期性检查（`periodicCheckArpRequestsAndCacheEntries`）：

- 如果某个ARP请求发送超过5次无应答，则认为目标不可达，对挂起该请求的分组全部发送ICMP Host Unreachable，并删除该ARP请求。
- 对ARP缓存条目进行检查，移除已失效的条目。
- 若ARP请求还未超5次限制，则重发ARP请求并增加计数。

这样通过定期维护ARP请求和ARP缓存，保证ARP缓存的时效性和正确性，以及对长期无法解析ARP的请求进行清理处理。

遇到的问题与解决方法

1. **大端小端转换不清**：
   初期对`htons/ntohs/htonl/ntohl`理解不足。通过反复对比RFC和示例代码，才意识到IPv4 header的长度、标识、校验和IP地址等字段传输需统一为网络字节序。在解析报文时用`ntohs/ntohl`转主机序，发送前用`htons/htonl`转网络序。 对于uint_16，uint_32需要进行转换，但对uint_8等不应该使用转换函数，只有一字节会导致错误。

2. **Checksum计算问题**：
   对IP和ICMP的校验和(Checksum)计算中，初次不理解计算过程。通过阅读提供的`cksum`函数与RFC1071

中的校验和算法描述，学会了先将`sum`置0后计算Checksum，并在验证Checksum时应当计算若不为0xffff则错误。

3. **锁与死锁问题**：
   在处理高流量下大量ARP请求时，最初使用`mutex`锁在`ArpCache`访问上出现死锁。通过改用`recursive_mutex`以及优化请求处理流程，避免嵌套锁导致死锁。

4. **长文件传输时卡顿**：
   传输大文件时出现停滞。仔细检查后发现是队列处理以及ARP查询延迟导致，再结合死锁问题的解决使大文件传输得以顺畅。

使用的额外库

- 标准C++库与C++11特性。
- `Ice`框架（已有）用于与POX通信。
- `Mininet`与`POX`为实验环境标准组件，本身不算第三方库。

未额外使用其他非标注库。

对本项目的建议

- 刚设计时需要花很多时间理清转发逻辑，在初学时可添加更多注释和调试信息来帮助理解ARP和IP转发流程。
- 应当提供更多`htons/ntohs/htonl/ntohl`函数的用法信息，避免字节序问题。
- 在Mininet测试前先`sudo mn -c`清理旧状态，否则易出现"File exists"类型错误，我遇到过很多次，这个需要在文档中特别提醒

感想

通过本项目，我加深了对路由器工作原理的理解，对以太网、ARP、ICMP、IPv4转发有了更直观认识。