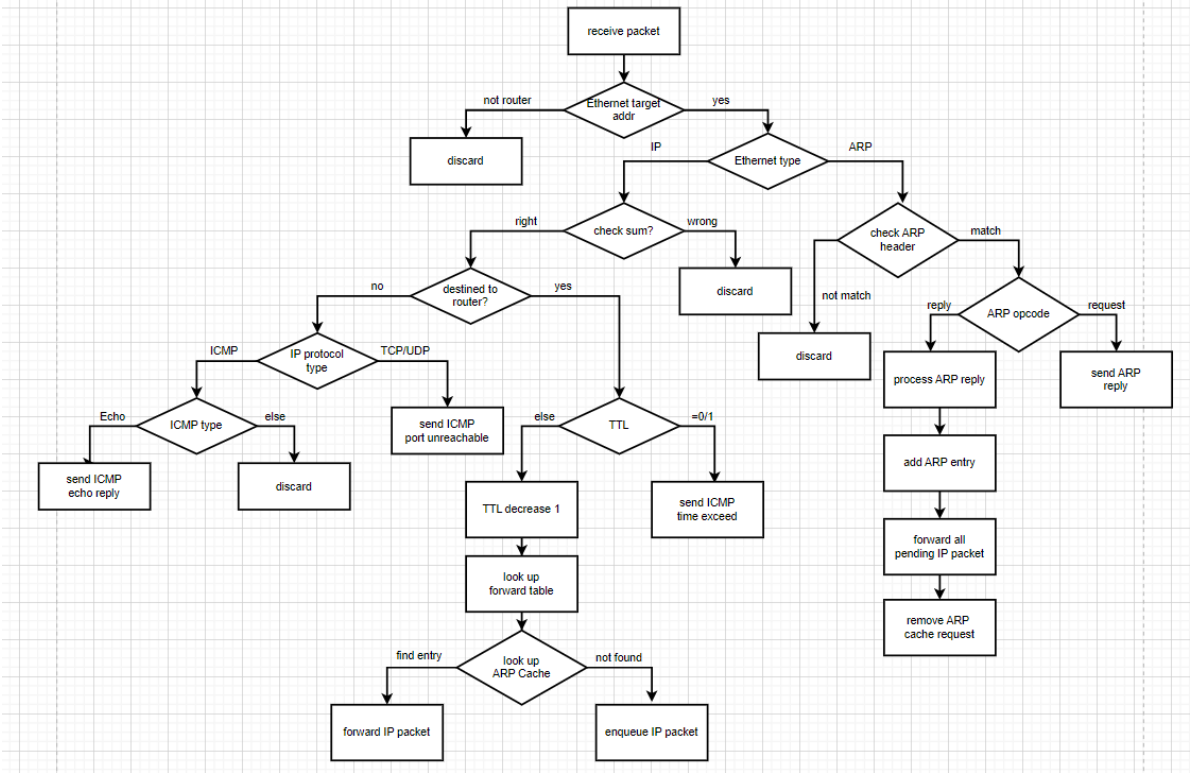
计算机网络 Lab 2 SimpleRouter

吴谦亮 软件01 2020012357

工作流程



难点与解决方案

1

最大的难点在于不知道函数 `handlePacket()` 的工作流程是什么

解决方案:

通过阅读 rfc 相关协议文档、课件和参考书，逐渐认识到处理数据包的流程基本为:

- 检查 header 相关地址、属性是否正确
- 根据数据包类型分情况进行处理，逐渐划分为最小层次，如 `ICMP echo`,`ICMP port unreachable`,`TCP/UDP`,`ARP request` 等
- 对每个最小层次的包分别根据 rfc 文档和 spec 进行实现

以处理 ARP 包为例，有以下流程:

- 读取包头，分别检查 `length`,`arp_hrd`,`arp_pro`,`arp_tip`,`arp_op`
- 根据 `arp_op` 类型分别处理 `ARP request` 和 `ARP rely`
- 对于 `ARP request`,调用已经封装好的函数 `sendARP` 发送 `ARP reply`(`sendARP` 内部以原数据包为基础，交换 `Ethernet header` 地址，再将路由器端口的 IP, MAC 等数据写入，最后完成发送)

2

前期不太了解整个项目各个模块间的交互是如何进行的

解决方案:

通过阅读项目源码和思考路由器接收不同类型数据包，在不同条件下的处理分支，观察各个模块之间的相互关系，最后认识到整个项目工作流程如下：

- router 接收数据包，调用函数 `handlePacket()` 处理该包
- 在 `handlePakcet()` 内部，在查找 forward table 时会调用成员 `m_routingTable` 的函数 `lookup` 按照 longest prefix match 找到对应端口
- 成员 `m_arp` 为管理 ARP Cache，其函数 `ticker` 会周期性调用 `periodicCheckArpRequestsAndCacheEntries` 来更新缓存的 ARP entry 和 ARPrequest
- 在函数 `periodicCheckArpRequestsAndCacheEntries` 内部，会重复发送 `ARP request` 直到最大次数，然后发送 `ICMP port unreachable` 给该包源地址

3

一开始不知道如何自己发送数据包，没有厘清纷繁复杂的数据类型之间的转换关系

解决方案:

通过参考已给代码中打印数据包信息的源码和查阅资料，逐渐认识到一套数据包数据类型的处理方式：

- 对于 `Buffer packet`，调用 `const uint8_t* buf = packet.data()` 即可得到对应的字符串数组，再通过调用 `memcpy()` 即可实现字符串数组的赋值
- 对于 `header struct`，可以利用类似于 `ip_hdr *iphdr = (ip_hdr *)(buf + sizeof(ethernet_hdr));` 的方法得到对应包头结构的指针，从而方便地获取与操作成员属性
- 对于 `interface`，通过类似 `const Interface* iface = findIfaceByName(inIface);` 得到接口指针，从而获取其 mac, ip, name

4

在向数据包写入数据时，不确定何时需要调用 `ntoh()` 转换函数，造成发送的数据包对应的数据不正确而无法被正确接收，从而误以为是其他bug，浪费了许多时间

解决方案:

在发送自己创建的数据报之前，将其内容全部打印出来(调用已有函数+自己封装额外的数据报类型)，从而可以在控制台查看路由器打印的数据报信息，方便调试过程

5

在 ARP Cache 函数 `periodicCheckArpRequestsAndCacheEntries()` 中，更新逻辑有错误，导致始终不能发送 `ICMP time exceed` 包

解决方案:

经过分析逻辑，发现对于每个 ARP request，由于将 `auto now = steady_clock::now();` 写错，其始终被识别为第一次发送请求的状态，从而不能发送 `ICMP time exceed` 包

6

在函数 `periodicCheckArpRequestsAndCacheEntries()` 中, 在删除 `ARP request` 和 `ARP Cache` 后，路由器程序崩溃退出

解决方案:

在查阅资料和与同学讨论后，了解到在使用 `for(auto x: y)` 形式遍历时，再使用 std::list 的 `remove()` 函数会有错误，因为这种遍历方法会自动将指针++

在改用用迭代器遍历 `for(auto iterator = list.begin(); iterator != list.end(); )` 后，使用 `iterator = list.erase(iterator)` 方法删除，在没有删除时将`iterator++`，即可实现正确删除对应元素

外部库

本次实验所用函数均来自于已有代码和 stl 标准库，没有引入其他外部库

课程建议

建议 spec 文档中对于各个协议的实现要求提及更多细节。因为 rfc 文档中规定的内容要求远超于本次实验，如果按照其全部要求进行完成，显然工作量不合理。而对于完成到何种程度，文档又没有很好的说明，导致很多要求是在和助教沟通、和同学交流后才了解的，从而增加了不必要的时间。