计算机网络实验
简单路由程序的实现

计算机科学与技术
1511186
梁宸

February 25, 2020

1 简介

本次作业实现了一个简单的路由程序. 使用基于JVM的Java和Kotlin语言, 测试环境为Windows 10和Ubuntu, jdk8.

2 运行

程序界面为命令行驱动. 首先输入一个网络接口ID, 之后程序可以接受如下命令:

1. start: 开始捕获数据包并处理路由和ARP回应等.
2. quit: 通知结束, 程序会在所有子线程结束后停止.
3. save: 手动地将路由表和ARP缓存表等数据结构缓存到本地.
4. flush: 清空ARP缓存.
5. insert: 添加路由表项, 接受12个整数, 依次表示网络的IP地址, 子网掩码和对应的下一跳IP地址.
6. delete: 删除路由表项, 参数-a表示全部清空; 参数-i表示按索引删除, 接受一个整数i表示删除第i项.
7. show: 打印数据结构到控制台, 参数-a表示打印ARP缓存表; 参数-f表示打印路由表.

程序实时地打印一些反馈和处理状态到控制台, 更为详细的日志文件保存在根目录的log文件中.

3 实现

3.1 数据结构

本程序主要数据结构有4个, 分别为IP地址, Ethernet地址, 路由表和ARP缓存表, 均为Kotlin实现.

3.1.1 IP地址

IP地址的数据结构定义在IPAddress.kt中, 类名为IPAddress, 为data class. 其中保存了IP地址和子网掩码的int型表示[1]和byte型表示. 在构造函数中判断是否为有效的IPv4地址和子网掩码, 否则抛出异常. 实现了match方法, 判断一个IP地址a是否在另一个IP地址b所表示的网络中, 方法是将a与b的IP地址都和b的子网掩码进行与操作, 判断结果是否相等, 这将用于路由选路; 重载了toString方法用于打印; 重载了hashCode方法用于构建ARP缓存表.; 重载了equals方法用于构建路由表.

3.1.2 Ethernet地址

在EthAddress.kt中定义了Ethernet地址的数据结构, 类名为EthAddress, 为data class, 也保存了MAC地址的int表示和byte表示. 重载了toString和equals方法.

3.1.3 路由表

路由表在ForwardTable.kt文件中定义, 类名为ForwardTable. 其中表项为两个IP地址类型, 表示网络地址和对应的下一跳地址. 表项维护为简单的链表. 实现了插入(或更新), 删除, 清空和查找匹配等方法, 它们都是同步方法以保证一致性. 查找算法为查找最长前缀匹配, 先将链表按前缀长度降序排序, 然后依次调用IP地址的match方法判断是否和所给的IP地址匹配, 若匹配则直接返回该表项.

更好的数据结构是用字典树(Trie)维护前缀. 而且我也不知道为什么当初我没用trie写.

3.1.4 ARP缓存表

ARP缓存表定义在ARPCacheTable.kt的同名类中. 主要结构是哈希表, Key为IP地址, Value为Ethernet地址和时间戳构成的对. 实现了插入(或更新), 清空和查找方法, 它们也都是同步方法. 其中插入时, 将当前系统时间加入到Value中; 查找时, 若哈希表中存在记录, 还需调用expire方法判断该记录是否已经过期失效[2], 否则依然认为未找到, 抛出异常.

[1] 即0-255可读表示.

[2] 默认取5分钟生存期.

3.2 程序

本程序的控制主体在Router.java文件中实现. 基于责任链模式和多线程处理. 使用Java自带的synchronized机制来保证线程安全.

3.2.1 主要的类

Router类是表述整个路由器程序. 它包含一个PacketHandler类和KeyboardCommandListenner类.

KeyboardCommandListenner类用于监听控制台的键盘输入指令, 并按指令调用Router的方法, 事实上作为程序的UI.

PacketHandler类用于Jnetpcap捕获到数据包的回调, 它包含一个RouteHandler类和一个ARPResponseHandler类.

ARPResponseHandler类用于处理一个ARP回应.

RouteHandler类用于处理一个路由行为, 它包含一个ARPRequestHandler类.

ARPRequestSender类用于发送一个ARP请求.

3.2.2 控制逻辑

程序启动之后, 申请一个Router类的实例, 创建一个线程A运行该实例的KeyboardCommandListenner类来监听用户指令. 当它收到用户的start指令后, 创建一个线程B运行Router的startLoop方法开始捕获数据包.

线程B回调Router类的PacketHandler类的nextPacket方法来处理每一个数据包, 它先缓存该数据包, 如果含有ARP的header, 则创建新线程ARPResponseHandler来处理; 若通过路由仲裁, 在本程序中即为有host的Ethernet地址而没有host的IP地址, 则创建新线程RouteHandler来处理.

RouteHandler先提取该数据包的IP header, 检查并更新其TTL, 若小于等于1, 则抛弃; 再提取其目的IP地址, 若不是有效的IPv4地址, 则抛弃; 接下来在路由表中查找对应的下一跳地址, 若未找到, 则抛弃; 若找到并等于特殊地址0.0.0.0, 则说明可直接投递, 下一跳地址为原始的目的IP地址;

得到下一跳IP地址后, 要在ARP缓存表中得到对应的Ethernet地址. 此时, 要先取得ARP缓存表的锁, 查找表中对应项, 若不存在, 则调用ARPRequestSender的方法发送一个ARP请求, 之后调用Java的wait方法, 释放锁并等待. 每当ARP缓存表被修改后, 它会被唤醒[3], 并检查是否能找到响应的记录, 若仍未找到, 则继续等待; 若等待超时, 则抛出异常, 表明这是一个不可到达的IP地址.

[3]这是ARPResponseHandler负责的

得到目的Ethernet地址后, 直接修改数据包Ethernet头部的对应字段, 并将其发送.

ARPResponseHandler先提取数据包中ARP header的Source Protocol Address和Source Hardware Address, 检查是否为有效的IPv4和Ethernet地址, 若不是则抛弃; 否则先取得ARP缓存表的锁, 然后更新此记录到表中, 并调用notifyall方法, 唤醒所有等待ARP缓存表的锁的线程.

更细粒度的锁机制是, 维护一个哈希表, 对其中每个IP地址维护一个锁和计数器, 每个等待ARP回应的线程都使计数器加1, 得到结果后使计数器减1, 同时, 如果计数器为0还要负责销毁哈希表中的记录. 然而这种机制在保证一致性时将会更为复杂, 尤其是在哈希表的添加和销毁记录时.