Router Report

生11 刘云 2021011932

1 路由器处理逻辑 simple-router.cpp

当router收到链路上的任何包后, 调用`handlePacket()`函数. 首先检查链路层MAC地址是否为接收接口MAC地址, 或广播地址`FF-FF-FF-FF-FF-FF`. 若是, 继续检查负载数据, 分为ARP数据和IP数据.

1.1 ARP

ARP数据分为两种, 分别是收到ARP请求和收到ARP回复.

1.1.1 ARP请求

ARP请求首先需要检查目的IP是否为当前端口IP. 若是, 则创造ARP回复数据, 将收到ARP请求的当前端口MAC地址写入. 随后直接将包发给接收这个包的端口.

1.1.2 ARP回复

ARP回复也需要先检查目的IP是否为当前端口IP, 若是, 则将ARP消息中的IP和MAC键值关系利用`insertArpEntry()`函数添加到ARP缓存中.

此时`insertArpEntry()`会返回当前目的IP所有未发送的包信息(包括包和需要发出的端口). 此处实现方法是在是在`simple-router.cpp`的`handlePacket()`中处理这些消息. 这些未处理的消息并不包括链路层包头. 在此填充发出端口的MAC地址和目的MAC地址, 然后从需要的端口发出.

1.2 IP

IP数据具有checksum, 首先对IP包头进行校验. IP数据也分为两种, 分别是目的IP地址为本端口地址, 和目的IP地址不为本端口地址.

1.2.1 目的IP为本端口IP: ICMP

此时需要解析IP负载数据中的信息. 本实验中只需要处理负载在IP中的ICMP消息. ICMP具有checksum, 首先对ICMP全长数据报进行检验. 若无误, 向源IP发送Type0 ICMP Echo消息.

如果收到的协议不为ICMP, 为UDP或TCP, 则向源IP发送Type3 Code3 ICMP消息.

1.2.2 目的IP不为本端口IP: 转发

检查IP包头中TTL信息, 若TTL此时为1或0, 则向源IP地址发送Type11 ICMP消息. 随后需要将IP包头的TTL信息减一, 随后重新计算包头的checksum. 然后再将IP的负载数据添加.

1.2.3 发送消息

本实验中, 除发送ARP回复和处理ARP缓存消息时直接调用`sendPacket()`函数(这些情况已经确定了发出端口), 其余情况均调用新函数`forwardPacket()`发送消息. 该函数输入除链路层外的包, 和目的IP.

函数中, 首先检查路由器的转发表, 寻找下一跳的IP和本机路由器需要发出的端口. 若没有找到匹配条目, 则向源IP发送Type3 Code1 ICMP消息(目的主机不可达). 若找到条目, 则进一步查询下一跳IP对应的MAC地址.

1.2.2.1 ARP缓存存在

若存在对应MAC地址缓存, 则构建出端口MAC地址和下一跳MAC地址的以太网包头, 然后调用`sendPacket()`函数从对应端口发出.

1.2.2.2 ARP缓存不存在

此时调用`queueRequest()`函数, 将包, 下一跳IP, 发出端口存入ARP缓存, 交给ARP处理ARP请求和回复等内容. 待ARP得到对应MAC地址后, 会将发出端口MAC和目的MAC添加到包中, 然后调用`sendPacket()`函数发出.

2 ARP处理逻辑 arp-cache.cpp

ARP同时维护ARP表和待发送的包. 前者负责记录IP和MAC的键值对, 后者负责维护尚不知道MAC地址的待发送包. 计时器定时调用`ticker()`函数, 进而调用`periodicCheckArpRequestsAndCacheEntries()`函数. 该函数有两个任务, 分别是去除过期ARP缓存, 和重新发送ARP请求.

2.1 清除ARP缓存

由于`ticker()`已将所有需要清除的ARP条目标记为`isValid == false`, 因此直接循环所有`cacheEntries`中的条目, 随后去除所有被标记的条目即可.

2.2 处理ARP请求

循环所有未处理的`arpRequests`条目(包含下一跳IP, 发往该IP的每一个待发送的包, 以及每个包的内容和发出端口, 此处认为发往同一个IP地址的端口相同).

对于请求次数少于5次的条目, 重新构建源地址为发出端口, 目的MAC地址为广播地址, 目的IP为下一跳IP的ARP请求.

对于请求次数已经超过5次的条目, 则提取IP包头中源IP发送Type3 Code1 ICMP消息(目的主机不可达), 并清除该ARP请求缓存.

3 路由表处理逻辑 routing-table.cpp

`lookup()`函数根据最长匹配原则寻找输入IP的下一跳信息. 如果下一跳(gw)为0.0.0.0, 则说明下一跳直接为目的地. 在本实验中, 将gw为0.0.0.0时, 自动返回请求的IP地址(即下一跳直接为目的地址).

4 经验感受

4.1 ntoh & hton

在存储大于1字节的数据时, 分为大端字节序和小端字节序, 即低地址是存储最高位还是最低位. TCP/IP要求网络传输时使用大端字节序. 因此在包头中传递大于1字节数据的时, 需要在本机的字节序和网络的大端字节序转换.

其中`ntoh`和`hton`函数用于转换. `n`代表网络的大端字节序, `h`代表主机的字节序. 由于字节序的转换与数据全长有关, 因此两个函数分别由对于2字节的`s`版, 和对4字节的`l`版.

在本实验中, 如`ether_type`, `ip_len`等数据涉及从本机和网络的字节序转换. 理论上4字节的IP也涉及字节序转换. 但是本实验中一般不涉及硬编码IP地址, 而是直接提取包头中的IP, 因此操作时均以网络字节序操作, 不涉及转换.

在checksum时, 应直接将收到的包进行运算, 即使用网络字节序. 本实验中, 本地构建的包再写入的时候均转换成网络字节序, 因此可以直接调用`utils.cpp`中的`cksum()`函数计算网络字节序版的checksum, 随后写入包中.

参考资料: https://blog.csdn.net/damanchen/article/details/112424874

4.2 死锁

在arp-cache中, 为避免冲突访问ARP缓存, 每个对外接口都有锁. 函数`periodicCheckArpRequestsAndCacheEntries()`在`ticker()`内部, `ticker()`函数已经获取锁, 因此`periodicCheckArpRequestsAndCacheEntries()`函数内部不可以继续调用对外接口, 如`removeRequest()`等操作修改ARP缓存, 应直接用列表的`remove`方法等操作. 否则会出现死锁的情况.

4.3 从上层向下层构建包

构建包的时候, 最好从高层向底层构建, 如先构建ICMP, 再构建IP. 因为一般底层的包需要描述上层整体的情况, 如长度, 因此最好的逻辑是从高层向下层构建.

另外作业中给出的是各个层级的包头, 不要忘记把负载数据加在包头后.