《计算机网络》实验报告 4

IPv4 分组转发实验

童超宇

院 （系）：计算机科学与技术学院　　专　　业：计算机科学与技术

学　　号 ：1152130106　　指导教师：刘亚维

2018 年 5 月

目录

1 实验目的

本实验需要将实验模块的角色定位从通信两端的主机转移到作为中间节点的路由器上，在 IPv4 分组收发处理的基础上，实现分组的路由转发功能。

网络层协议最为关注的是如何将 IPv4 分组从源主机通过网络送达目的主机，这个任务就是由路由器中的 IPv4 协议模块所承担。路由器根据自身所获得的路由信息，将收到的 IPv4 分组转发给正确的下一跳路由器。如此逐跳地对分组进行转发，直至该分组抵达目的主机。IPv4 分组转发是路由器最为重要的功能。

本实验设计模拟实现路由器中的 IPv4 协议，可以在原有 IPv4 分组收发实验的基础上，增加 IPv4 分组的转发功能。对网络的观察视角由主机转移到路由器中，了解路由器是如何为分组选择路由，并逐跳地将分组发送到目的主机。本实验中也会初步接触路由表这一重要的数据结构，认识路由器是如何根据路由表对分组进行转发的。

2 实验内容

在前面 IPv4 分组收发实验的基础上，增加分组转发功能。具体来说，对于每一个到达本机的 IPv4 分组，根据其目的 IPv4 地址决定分组的处理行为，对该分组进行如下的几类操作:

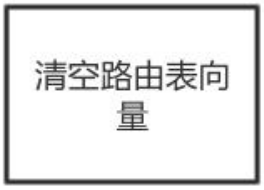
1)向上层协议上交目的地址为本机地址的分组;

2)根据路由查找结果，丢弃查不到路由的分组;

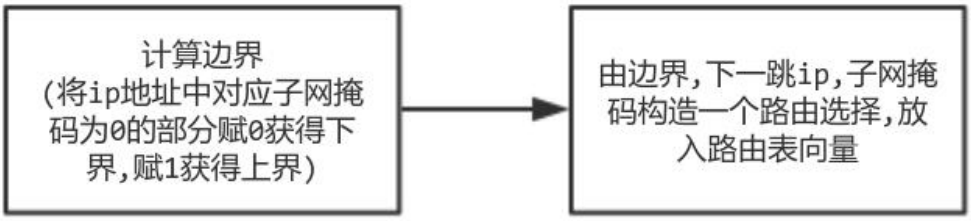
3)根据路由查找结果，向相应接口转发不是本机接收的分组。

3 程序流程图

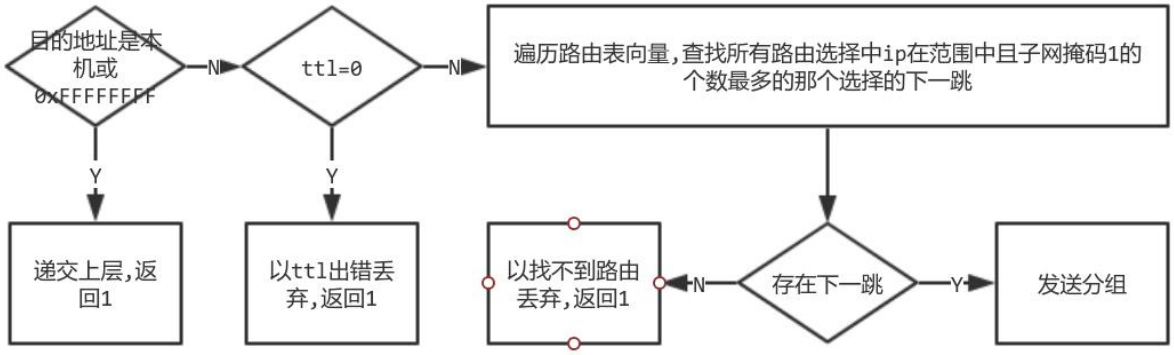
3.1 路由表初始化



3.2 路由增加



3.3 路由转发



4 数据结构

路由表为一个向量，存储多个路由选择，结构如下

```
25  struct route {
26    uint low, high, masklen, nextIP;
27    route(uint low, uint high, uint masklen, uint nextIP) {
28      this->low = low;
29      this->high = high;
30      this->masklen = masklen;
31      this->nextIP = nextIP;
32    }
33  };
```

每个路由选择包括边界，长度（掩码中 1 的个数），下一跳

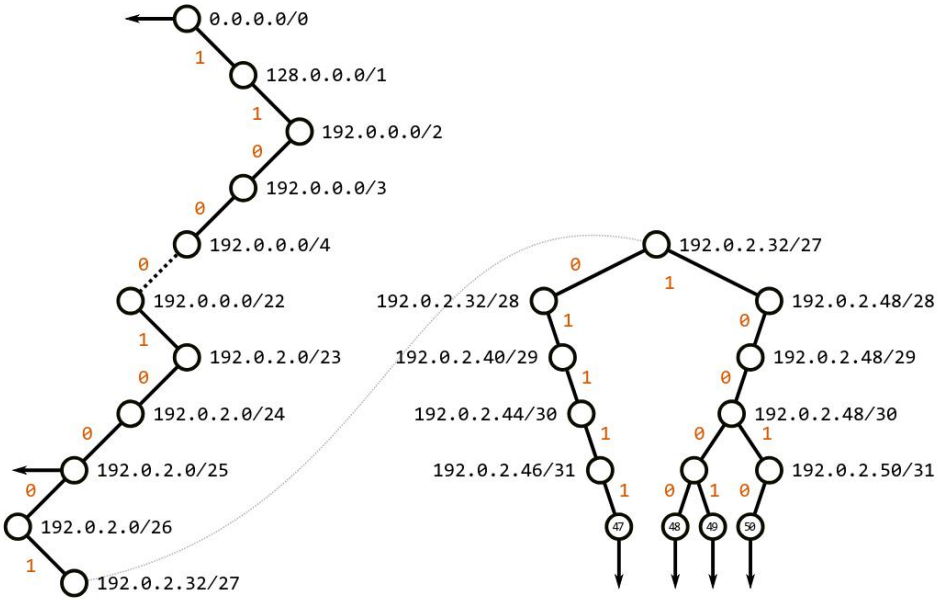
边界的计算方法是将 ip 地址中对应子网掩码为 0 的部分赋 0 获得下界,赋 1 获得上界

5 提高转发效率

本实验中的转发可以抽象为这样一个问题

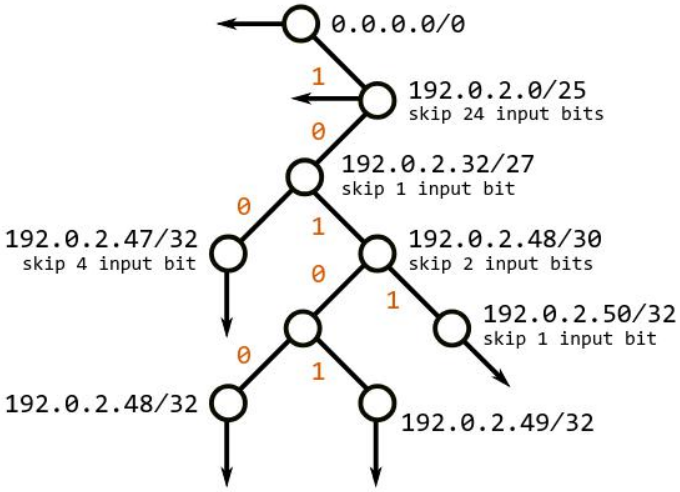
输入一组整数区间，和一个整数，输出其中一个区间包含这个数且区间长度最小，整数与区间边界在 0 到 2^32-1

可以构造一颗 trie 树，如下图所示

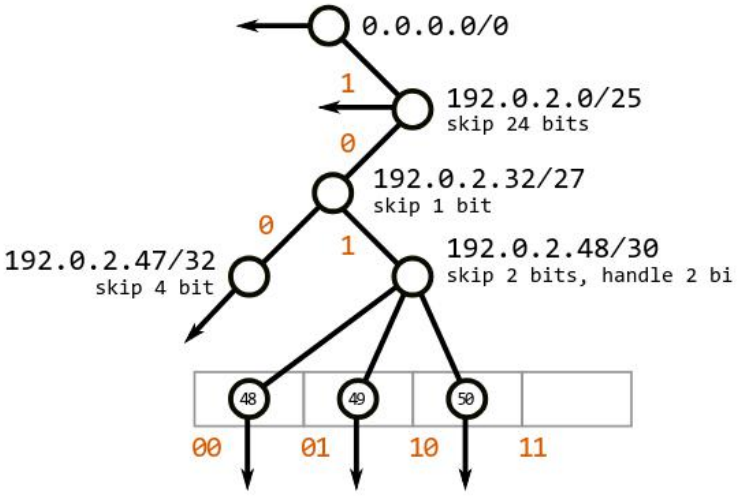


树的最大高度为 32，每次查询从上往下遍历，最多遍历 32 个节点，相比线性查表有很大提升

在 trie 的基础上可以使用路径压缩技术，只有一个孩子的节点被删除(除非它还包含一个路由条目)，其余每个节点获取一个新的属性表示应跳过多少输入位，由此构造一颗 Patricia 树，这在 OpenBSD,FreeBSD 上使用



还可以使用层压缩，检测 trie 的密集部分（densily populated），用 2^k 的向量替换单一节点，构造一颗 LC-trie（LPC-trie）



这是 linux 的 ipv4 路由采用的结构

参考 https://vincent.bernat.im/en/blog/2017-ipv4-route-lookup-linux#lookup-with-a-level-compressed-trie

查看本机的路由结构

```
~ cat /proc/net/fib_trie
Main:
  +-- 0.0.0.0/0 3 0 5
     |-- 0.0.0.0
        /0 universe UNICAST
     +-- 127.0.0.0/8 2 0 2
        +-- 127.0.0.0/31 1 0 0
           |-- 127.0.0.0
              /32 link BROADCAST
              /8 host LOCAL
           |-- 127.0.0.1
              /32 host LOCAL
        |-- 127.255.255.255
           /32 link BROADCAST
     +-- 192.168.199.0/24 2 0 1
        |-- 192.168.199.0
           /32 link BROADCAST
           /24 link UNICAST
        |-- 192.168.199.100
           /32 host LOCAL
        |-- 192.168.199.255
           /32 link BROADCAST
```

6 源代码

```
#include <iostream>
#include <vector>
#include "sysInclude.h"
using namespace std;
typedef unsigned int uint;
typedef unsigned short ushort;
extern void fwd_LocalRcv(char *pBuffer, int length);
extern void fwd_SendtoLower(char *pBuffer, int length, uint nexthop);
extern void fwd_DiscardPkt(char *pBuffer, int type);
extern uint getIpv4Address();
// set tail = 00000...
uint getLow(uint IP, uint masklen) {
    masklen = 32 - masklen;
    IP >>= masklen;
    IP <<= masklen;
    return IP;
}
// set tail = 11111...
uint getHigh(uint IP, uint masklen) {
    masklen = 32 - masklen;
    IP |= (1 << masklen) - 1;
    return IP;
}

struct route {
    uint low, high, masklen, nextIP;
    route(uint low, uint high, uint masklen, uint nextIP) {
        this->low = low;
        this->high = high;
        this->masklen = masklen;
        this->nextIP = nextIP;
    }
};

vector<route> vec;

void stud_Route_Init() {
```

```
    vec.clear();
    return;
}

void stud_route_add(stud_route_msg *proute) {
    uint dest = htonl(proute->dest);
    uint masklen = htonl(proute->masklen);
    uint nextIP = htonl(proute->nexthop);
    uint low = getLow(dest, masklen);
    uint high = getHigh(dest, masklen);
    vec.push_back(route(low, high, masklen, nextIP));
    return;
}

bool get_next_ip(uint destIP, uint &nextIP) {
    uint len = 0;
    bool ret = false;
    for (uint i = 0; i < vec.size(); i++)
        if (vec[i].low <= destIP && vec[i].high >= destIP)
            if (vec[i].masklen >= len) {
                len = vec[i].masklen;
                nextIP = vec[i].nextIP;
                ret = true;
            }
    return ret;
}

int stud_fwd_deal(char *pBuffer, int length) {
    uint destIP = ntohl(*(uint *)(pBuffer + 16));
    if (destIP == 0xFFFFFFFF || destIP == getIpv4Address()) {
        fwd_LocalRcv(pBuffer, length);
        return 1;
    }

    uint ttl = (uint)(pBuffer[8]);
    if (ttl == 0) {
        fwd_DiscardPkt(pBuffer, STUD_FORWARD_TEST_TTLERROR);
        return 1;
    }
```

```
    uint nextIP;
    if (!get_next_ip(destIP, nextIP)) {
        fwd_DiscardPkt(pBuffer, STUD_FORWARD_TEST_NOROUTE);
        return 1;
    }

    // ttl-1
    pBuffer[8] = (ttl - 1) & 0xff;
    // re calculate checksum
    *(ushort *)(pBuffer + 10) = 0;
    int sum = 0;
    for (int i = 0; i < 10; ++i) sum += (int)(*(ushort *)(pBuffer + i * 2));
    while (sum > 0xffff) sum = (sum & 0xffff) + (sum >> 16);
    sum = ~((ushort)sum);
    *(ushort *)(pBuffer + 10) = (ushort)sum;

    fwd_SendtoLower(pBuffer, length, nextIP);
    return 0;
}
```