NetRiver 3–IPv4 协议转发实验

张煌昭，1400017707，元培学院

摘要—本次作业对 IPv4 协议路由转发进行了实现，完成了配置静态路由表，根据路由表进行分组转发的功能，并于 NetRiver 平台进行提交和测试。本次报告使用 Overleaf L^AT_EX 在线平台编写[1]，作业源码附于报告之后。

I. 介绍

本次作业，要求使用 C/C++ 语言，在 NetRiver 的 IPv4 层虚拟环境中，在上一次 IPv4 收发实验的基础之上，完成 IPv4 路由器分组转发的功能。较为具体的功能包括初始化并配置静态路由表，以及根据路由表对分组进行逐跳转发。在转发过程中发现 TTL 错误和路由信息不存在时，丢弃分组并报错。

路由表的实现，配置，和维护详情见第 II节，IPv4 分组转发的详情请见第 III节。

II. 路由表

A. 要求

要求实现和维护一个静态路由表，路由表中至少需要包括目标地址和下一跳地址。需要实现路由表的初始化以及配置函数。。

本次作业下，路由表初始化和配置的函数，以及路由表配置信息结构体均如下。

```
void stud_route_add(stud_route_msg *proute);
void stur_Route_Init();
typedef struct stud_route_msg
{
    unsigned int dest;
    unsigned int masklen;
    unsigned int nexthop;
} stud_route_msg;
```

调用初始化函数，路由表进行初始化并清空。调用配置函数，根据穿入参数向路由表添加新的一行，并对该行进行设置。

[1]本报告源码可通过以下 git 命令获得，
git clone https://git.overleaf.com/15853721gmnmbcjkdywj

B. 实现

路由表仅有目标地址和下一跳地址两列，查表时需要根据目标地址进行查找，并且对查表效率的要求很高，因此采用 C++ STL 中的 map 数据结构对其进行实现。

map 基于平衡树实现，插入和查找效率很高，并且提供非常方便的 API。将路由表定义为如下结构。其元素为 <int, int> 的 pair，每个 pair 中的 first 为目标地址，second 为下一跳地址。如此便自然实现了路由表的数据结构。

```
map<int, int> RouteTable;
```

对路由表进行初始化的函数实现如下。直接采用 map.clear() 的 API 进行实现。

```
void stud_Route_Init()
{
    RouteTable.clear();
    return;
}
```

对路由表进行配置的函数实现如下。下一跳地址直接由参数中提取得到，目标地址使用参数中的 masklen 生成的 mask 对参数中的目标地址进行过滤得到。

```
void stud_route_add(stud_route_msg* proute)
{
    int mask = 0xFFFFFFFF <<
                (32 - htonl(proute->masklen));
    int DstAddr = ntohl(proute->dest) & mask;
    int NextHop = ntohl(proute->nexthop);
    RouteTable.insert(std::pair<int, int>(DstAddr,
                                          NextHop));
    return;
}
```

III. IPv4 分组转发

A. 要求

路由器接收到 IPv4 分组后，需要根据分组的目标地址选择下一跳进行转发，转发过程若发现 TTL 耗尽，或路由表中不存在该分组的目标地址的表项，则丢弃该

帧并报相应错误。完成检查后，对 TTL 和校验和进行更新并向下一跳转发。

本次作业下，IPv4 分组的接收检查通过以下函数实现。

```
int stud_fwd_deal(char *pBuffer, int length);
```

其中 pBuffer 指向接收到的 IPv4 分组内容，length 为分组长度。转发完成，函数返回值为 0，否则出错返回 1。

B. 需要使用的系统接口

获取本机 IPv4 地址的接口如下，没有参数，返回值为本机的 IPv4 地址。

```
unsigned int getIpv4Address();
```

丢弃分组并报错的接口如下，pBuffer 为需要丢弃的分组，type 为错误类型编号，包括 TTL 出错，和找不到路由表项。

```
// TTL值出错
#DEFINE STUD_FORWARD_TEST_TTLERROR
// 找不到路由表项
#DEFINE STUD_FORWARD_TEST_NOROUTE

void fwd_DiscardPkt(char *pBuffer, int type);
```

分组目标地址为本机时，直接向上层协议提交分组，接口如下。

```
void fwd_LocalRcv(char *pBuffer, int length);
```

进行转发时，通过如下接口向下层发送并标明下一跳地址。

```
void fwd_SendtoLower(char *pBuffer, int length,
                     unsigned int nexthop);
```

C. 实现

接收分组后，从中获取 IHL（用于更新校验和），TTL 和目标地址。首先根据目标地址是否为本机判断是否需要进行转发，若不需要转发则直接向上层协议提交分组并返回 0。之后检查 TTL，查找目的地址对应的路由表项，若出错则报相应错误并返回 1。最后复制分组，更新 TTL（TTL <- TTL - 1）和校验和（利用新的头部计算）。完成后进行转发并返回 0.

程序的 C/C++ 风格的伪代码如下所示。

```
int stud_fwd_deal(char* pBuffer, int length)
{
    int IHL = GET_IHL(pBuffer);
    int TTL = GET_TTL(pBuffer);
    int DstAddr = GET_DSTADDR(pBuffer);
    if (DstAddr == GET_MY_IP())
    {
        // 目标地址为本地，不转发，交付上层
        return 0;
    }
    if (TTL <= 0)
    {
        // TTL错误，报错并丢弃分组
        return 1;
    }
    // 查表寻找DstAddr的表项iter
    if (isNull(iter)) {
        // 没有路由表项，报错并丢弃分组
        return 1;
    }
    // 复制分组至Buffer
    SET_TTL(Buffer, TTL-1);
    SET_CHECKSUM(Buffer, IHL);
    // 转发至下一跳
    return 0;
}
```

IV. 代码

本次作业详细代码请见附录部分。

附录

```
/*
* THIS FILE IS FOR IP FORWARD TEST
*/
#include "sysInclude.h"
#include<map>
using namespace std;

// system support
extern void fwd_LocalRcv(char *pBuffer, int length);

extern void fwd_SendtoLower(char *pBuffer, int length, unsigned int nexthop);

extern void fwd_DiscardPkt(char *pBuffer, int type);

extern unsigned int getIpv4Address( );

// implemented by students

/* RouteTable <DstAddr, nexthop> */
map<int, int> RouteTable;

/* Initializing the RouteTable */
void stud_Route_Init()
{
    RouteTable.clear();
    return;
}

/* Adding a new entry to the RouteTable */
void stud_route_add(stud_route_msg* proute)
{
    int mask = 0xFFFFFFFF << (32 - htonl(proute->masklen));
    int DstAddr = ntohl(proute->dest) & mask;
    int NextHop = ntohl(proute->nexthop);
    RouteTable.insert(std::pair<int, int>(DstAddr, NextHop));
    return;
}

/* Dealing with the reception and forwarding */
int stud_fwd_deal(char* pBuffer, int length)
{
    int IHL = pBuffer[0] & 0xf;
    int TTL = (int)pBuffer[8];
    int DstAddr = ntohl(*(unsigned *)(&pBuffer[16]));

    // Local, no forwording.
    if (DstAddr == getIpv4Address())
    {
        fwd_LocalRcv(pBuffer, length);
        return 0;
    }
    // TTL error.
    if (TTL <= 0)
    {
        fwd_DiscardPkt(pBuffer, STUD_FORWARD_TEST_TTLERROR);
        return 1;
    }
```

```
    map<int, int>::iterator ii = RouteTable.find(DstAddr);
    // No route.
    if (ii == RouteTable.end())
    {
        fwd_DiscardPkt(pBuffer, STUD_FORWARD_TEST_NOROUTE);
        return 1;
    }

    unsigned char* Buffer = (unsigned char *)malloc(length);
    memcpy(Buffer, pBuffer, length);

    // Set TTL and header check sum.
    Buffer[8] = TTL - 1;
    unsigned int HeaderCheckSum = 0;
    memset(&Buffer[10], 0, sizeof(short));
    for (int i = 0; i < 4 * IHL; i += 2)
        HeaderCheckSum += ((Buffer[i] & 0xFF) << 8) + (Buffer[i + 1] & 0xFF);
    HeaderCheckSum += (HeaderCheckSum >> 16);
    HeaderCheckSum = ~HeaderCheckSum;
    Buffer[10] = (unsigned short)HeaderCheckSum >> 8;
    Buffer[11] = (unsigned short)HeaderCheckSum & 0xFF;

    // Send the package.
    fwd_SendtoLower((char *)Buffer, length, (*ii).second);

    return 0;
}
```