学习手册

THUSC2019 学习题命题组

2019 年 5 月

目录

1 二进制文件读写

本题中需要用到二进制文件读写，我们会将输入/输出文件重定向到标准输入/输出。如果你对这方面不熟悉，可能需要参考如下 C 语言代码（C++ 同样可以使用）：

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <assert.h>
```

```

// 声明要用的结构体
struct SomeType data;
#define N 20
struct SomeType more_data[N];

size_t size = sizeof(struct SomeType);

// 读入一个或多个某种类型的数据，返回值为成功读取的 size 的倍数（而非字节数）
size_t read_size = fread(&data, size, 1, stdin);
assert(read_size == 1);
size_t read_more_size = fread(more_data, size, N, stdin);
assert(read_more_size == N);

// 将一个或多个某种类型的数据写入文件中，返回值意义与上面相同
size_t written_size = fwrite(&data, size, 1, stdout);
assert (written_size == 1);

// 每次写入后，文件指针会向后移动相应的位置，可以这样重新定位
fseek(stdout, 0, SEEK_SET); // 相对开头的偏移量，此语句等价于 rewind(stdout);
fseek(stdin, 0, SEEK_CUR); // 相对当前文件指针的偏移量
fseek(stdin, 0, SEEK_END); // 相对文件结尾的偏移量
// 需要注意，只有被重定向到文件的 stdin, stdout 才能进行 seek 操作
```

当然，你也可以使用 `read(2)`，`write(2)`，`mmap(2)` 等更底层的系统 API 实现相同的功能。与传统 OI 题目不同的是，在本题中我们对它们不加以限制。如果你不了解它们，可以忽略本段。

如果你不了解某些库函数的用法，可查阅 `man` 手册获得更详细的解释。如你可以在 Linux 的终端中输入 `man 2 read` 来获得 `read(2)` 的文档，或者使用 `man fread` 来获得 `fread` 函数的文档。

2 字节操作与端序

本题中涉及到大量二进制数据的直接操作。你可以使用 `stdint.h` 中定义的各种平台无关的数据类型来方便定义和操作，如 `uint8_t`，`uint16_t`，`uint32_t`，`uint64_t` 等。前缀 `u` 表示无符号，没有 `u` 则表示有符号。在 X86 架构（正是你现在所使用的）下，`uint32_t` 对应 `unsigned int`，`int32_t` 对应 `int` 。

长于一个字节的数据类型在内存中的存储有两种顺序，分别是大端序和小端序。也就是说，对于一个多字节的基本类型（如 `uint32_t`），其中的字节存储顺序由端序决定：大端序的高位字节存储在低地址，小端序则相反。例如对于 `0x01020304` 这个四字节整数，在大端序下，其在内存中由低地址到高地址的排列顺序为 `0x01 0x02 0x03 0x04`，而在小端序下为 `0x04 0x03 0x02 0x01`。x86 架构采用**小端序**存储数据，而在网络上传输的几乎所有信息均采用**大端序**，请密切关注下面给出的字节序说明。你可以使用 GCC 内置的下列的函

数进行端序的转换：`__builtin_bswap16`，`__builtin_bswap32`，`__builtin_bswap64`。需要注意，这里只涉及到字节和字节之间的排列顺序，而不是一个字节中八个位的排列顺序。

3 PCAP 文件格式

PCAP 是常见的流量捕捉格式，可以存储各种协议类型的流量片段。一个合法的 PCAP 文件中存在三种数据格式：Global Header、Record Header 和 Record Data。

Global Header 在文件头部出现且仅出现一次。等价的文件结构定义如下。在读取时，你可以直接跳过文件头。在写入时，请严格遵守下面的约定，特别是**注意端序**。

```c
typedef struct pcap_hdr_s {
        // 所有字段都是大端序
        uint32_t magic_number;   // 用于文件类型识别，始终为 0xA1B2C3D4，
        uint16_t version_major;  // 始终为 2
        uint16_t version_minor;  // 始终为 4
        int32_t  thiszone;       // 始终为 0
        uint32_t sigfigs;        // 始终为 0
        uint32_t snaplen;        // 允许的最大包长度，始终为 262144
        uint32_t network;        // 数据类型，本次学习题中始终为 1 （以太网）
} pcap_hdr_t;
```

在其之后，就是一个或多个流量数据片段。片段与片段之间、片段与 Global Header 之间没有多余的数据。每一个片段分为两部分，首先是 Record Header：

```c
typedef struct pcaprec_hdr_s {
        // 所有字段都是大端序
        uint32_t ts_sec;         // 时间戳（秒）
        uint32_t ts_usec;        // 时间戳（微秒）
        uint32_t incl_len;       // 该片段的存储长度
        uint32_t orig_len;       // 该片段实际的长度
} pcaprec_hdr_t;
```

本题中，我们保证输入数据 `orig_len` 始终和 `incl_len` 相同，即流量中不会出现存储不完整的片段，保证时间戳（微秒）小于 10^6 。如果没有特别指定，在构造新的流量片段写入输出文件时，**请将两个时间戳均置为 0**。两个长度字段始终应该相等。

在 Record Header 之后，紧跟的 `orig_len` 字节即是这一片段的实际数据。而后就是下一个片段的 Record Header 与 Data，如此重复，直到文件结尾。

4 PCAP 文件浏览与协议解析工具

由于本题涉及到 PCAP 文件和一些网络协议的格式的解析和生成，我们为你提供了一系列软件：Wireshark, tshark 和 termshark ：

Wireshark Wireshark 是 GUI 的网络协议分析工具，在本题中用于 PCAP 文件的内容可视化，题目中也会出现针对 Wireshark 用法的提示。

tshark Wireshark 的命令行版本，它输出结果与 Wireshark 界面上看到的结果一致，但没有交互。

termshark Wireshark 的 TUI 版本，它与 Wireshark 界面基本一样，仅用字符替代了界面。

我们只在题面中提供了 Wireshark 的配置方法，其余请自行摸索。

5 网络的分层架构

Internet 工作方式基于分层的理念。我们采用比较简单的 TCP/IP 参考模型。在此模型中，整个网络自下而上分为四层：数据链路层、网络层、传输层、应用层。每一层都运行某种协议，并携带上层的数据作为本一层协议的负载。在数据从上层流动到相邻的下层时，会被下层的协议打包进它的协议格式中；反过来，则会经历拆包的过程。借助这样的设计，上层无需考虑下层的具体实现，就能与通信对端的同等层进行通信。在下面的各个部分，我们会介绍每一层传输的数据格式。

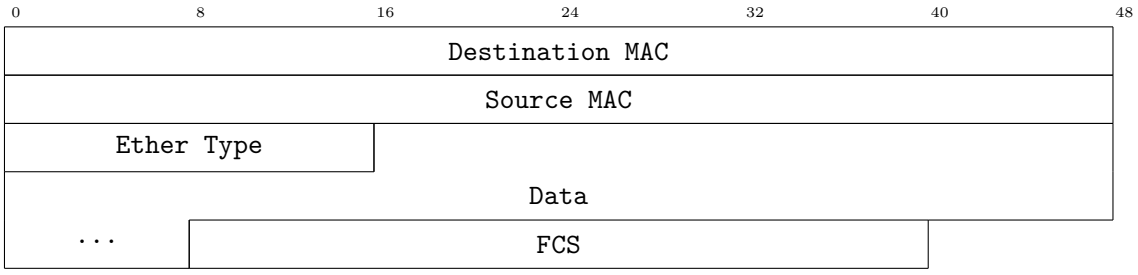
6 数据链路层协议

6.1 以太网协议简介

数据链路层是 TCP/IP 参考模型中最底层，它提供字节流在物理链路上的无差错传输。目前应用最广泛的数据链路层协议是 Ethernet （以太网）协议。以太网上传输的数据单位称为帧（Frame），其格式如图 1 所示。

请注意，本题中会出现数个如图 1 的数据结构图示。我们约定，数据中的字节编号按从左到右、从上到下的顺序递增，而每个字节中**高位在左，低位在右**。图上方的比特编号仅供定位与识读之用，**不代表在结构中的顺序**。

图 1: 以太网帧的结构



`Source MAC` 和 `Destination MAC` 长度都为 6 字节（48 比特），代表着这个帧的源地址和目标地址。在数据链路层使用的地址称为 MAC (Media Access Control) 地址，一般写作 `00:11:22:33:44:55` 的形式，在以太网帧中保存的字节顺序与书写顺序相同（如上例从低字节到高字节为 `0x00 0x11 0x22 0x33 0x44 0x55`）。每个物理接口（如网卡）都有自己唯一的 MAC 地址，称为单播地址；全部比特为 1 的 MAC 地址（`FF:FF:FF:FF:FF:FF`）称

为广播地址。主机能接收到目标地址为自己具有的 MAC 地址的以太网帧，发出的帧源地址也必须是自己所具有的；同时，也能发送和接受广播帧。实际情况下 MAC 地址还有更多类型，这里不用考虑，只需要区分单播和广播地址。

`Ether Type` 字段的长度为 2 字节，**大端序**，存储了其携带的下层协议类型。常用的类型有 IPv4 (`0x0800`)、ARP (`0x0806`) 等。在本题中，我们只会用到这两种类型，请在构造对应流量时使用对应的值。

`Data` 字段长度是可变的，存储了以太网协议的负载，即上层协议数据。在不同类型的网络上，这一字段的长度有不同的上限，通常在 1500 字节左右。简化起见，在本题中我们不考虑这一限制。另一方面，合法以太网帧的最小长度（包括校验）为 64 字节。如果你构造的帧不足这一长度，则需要在后面使用全 0 填充至**恰好 64 字节**，即在最后四字节的校验前还有若干个零；否则，不需要进行任何操作。在你输出的所有以太网帧中，除了上层协议的数据和可能需要的填充以外，**不应该有任何额外的数据**，否则将被判定为错误。但是，输入数据中可能出现**一定长度的额外的数据**，这些数据仍然被包括在校验和的计算中，但你不应该解析这些额外的数据。

`FCS` 字段全称为 Frame Checks Sequence，长度为 4 字节，存储了**前面所有字段**（即 Source MAC、Destination MAC、Ether Type 和 Data，也包括填充）的校验和。以太网协议采用的校验算法为 CRC32，计算方法如下所述。

6.2 以太网协议的 CRC 计算方法

以太网协议中使用的 CRC32 校验算法使用了下列生成多项式：

$$G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$$

具体的计算流程如下：

1. 将整个以太网帧排列成二进制序列，每个字节中，低比特在前，高比特在后
2. 将序列最前面 32 比特按位取反
3. 设需要计算校验的数据共有 n 比特，从前往后将它对应到一个 $n-1$ 次多项式 $M(x)$：第一个比特对应 x^{n-1} 项的系数，最后一个比特对应常数项。
4. 计算多项式除法 $x^{32}M(x)/G(x)$ 的余数，它是一个次数不高于 31 的多项式 $R(x)$。
5. 将 $R(x)$ 的各项系数按照 3 的对应关系转换回为 32 比特的二进制序列，并按位取反。
6. 将上述序列按照 1 的排列方法解码为四个字节，即为最终的 FCS 校验值。

请注意，上面所有的多项式的系数取值都在 $\{0,1\}$ 中，**相加/减时不进行进位/借位，事实上进行的是异或操作**，在进行乘除法时需要注意利用相应的性质。接收方只要将数据根据同样的方式进行计算，并将得到的校验值与 `FCS` 字段进行对比。如果一致，则认为数据是正确的。下面给出一个例子，即待校验数据为 `0x1, 0x2, 0x3, 0x4` 时具体的计算过程。

`1. 将数据排列为每个字节低比特在前的二进制序列`
`10000000_01000000_11000000_00100000`

`2. 将最前面32 比特取反，并在后面添加 32 个 0 （即得到 x^32*M(x) 的系数表示）`

```
01111111_10111111_00111111_11011111_00000000_00000000_00000000_00000000
```

3. 进行基于异或的除法，得到最终的余数

```
 01111111101111110011111111101111100000000000000000000000000000000
^ 10000010011000001000111011011 0111
=00111110100011110111100010110010110000000000000000000000000000000
^  10000010011000001000111011011 0111
=00011110000101110101101100000100001000000000000000000000000000000
（中间省略大量类似步骤）
=00000000000000000000000000000100100010110100010111110001111110 0
^                                  10000010011000001000111011011 0111
=00000000000000000000000000000001001100001000001100001110010010
```

余数为 01001100_00100000_11000011_10010010

4. 将余数按位取反，并按照1的排列方法分解为4个字节

11001101_11111011_00111100_10110110

即最终得到的校验值序列为 0xCD 0xFB 0x3C 0xB6，请仔细**阅读并理解以上过程**。

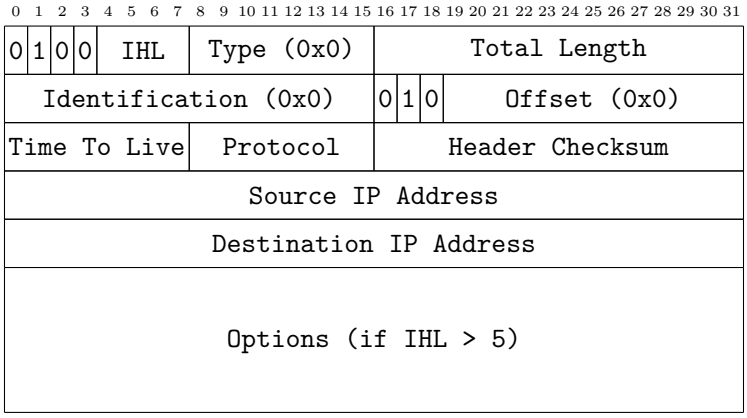
7 网络层协议

7.1 IP 协议头格式

在网络层运行的最常用的协议是 IP (Internet Protocol) 协议，它也是目前 Internet 的核心协议。它能够忽略数据链路层的差异，实现不同类型网络的互联互通。在本题中，我们只考虑 IPv4 协议。

IPv4 协议的数据最小单位称为分组，由分组头和数据构成。IPv4 地址的分组头结构如图 2 所示。请注意所有长于 8 比特的字段都是**大端序**的。

图 2: IPv4 分组头的结构



其中标记了默认值的字段不需要关心，在构建时原样填写即可。其他字段的含义如下：

- `IHL` 字段全称是 Internet Header Length，标记了 IP 分组头的长度（单位是 4 字节，所以 IHL=5 代表分组头长度为 20）；从图中可以看出，分组头的长度至少是 20 字节，所以总是不小于 5；当分组头中有额外的可选字段 `Options` 时，需要相应更改 IHL 字段。
- `Total Length` 字段即为 IP 分组（头与数据）的总长度，单位是字节，它至少应该为 20（代表此时不包含数据）。
- `Time to Live` 字段代表了该分组还能被转发的剩余次数，每个转发分组的路由器都需要将其减少 1，当到 0 时丢弃。一个新发出而非被转发的分组的 TTL 需要被置为 64。
- `Protocol` 字段代表分组中承载的高层协议类型，常见的有 TCP(`0x06`)、UDP(`0x11`) 和 ICMP(`0x01`) 等。下列各题中，你需要使用这个字段来判断不同的分组类型和处理方式。
- `Header Checksum` 代表了整个分组头（包含 `Options` 字段）的校验和，具体的计算方式参见下面的叙述。
- `Source/Destination IP Address` 代表分组的源和目标地址，关于 IP 地址下文也有更详细的描述。
- `Options` 字段用于放置额外的选项，进行转发时，你需要原样保留这一字段；而在构建的分组中，请不要加入这一字段。

7.2 IP 地址与网络划分

IPv4 协议使用 32 位长的整数作为地址，为了方便记忆，人们通常使用点分十进制的形式来书写 IP 地址：按字节从高到低切分，依次转为十进制，并将四个十进制数用小数点连接起来。如 `0x65060606` 对应的点分十进制形式为 `101.6.6.6`，`0xFFFFFFFF` 对应 `255.255.255.255`，`0x03020100` 对应 `3.2.1.0` 等等。

我们将物理上连接在同一个链路上（如连接在同一个交换机上）的所有主机组成的网络称为一个**广播域**，同一个广播域内的主机的 IP 地址通常会在同一个**子网**中。为了划分子网，我们引入一个另一个整数 $0 \le p \le 32$，称作前缀长度。为了简化记号，我们将一个（IP，前缀长度）的二元组写成 `a.b.c.d/p` 的形式，它是标识任何网络接口不可缺少的两要素。如果两个 IP 的前 p 个二进制位相同，那么我们就称这两个 IP 在这个前缀长度意义下处于同一个子网中。这种划分网络的方式称为无类别域间路由（Classless Inter-domain Routing, CIDR）。

在一个子网中，最大和最小的地址都被保留。最小的地址被称为**网络地址**（在历史上也被称为网络号），用于标识整个子网，而最大的地址被称为**广播地址**；中间所有地址与网络地址的差值被称为**主机号**。所有主机都接收目标地址为广播地址的 IP 分组，而路由器会将目标地址为广播地址的分组转发给整个子网。作为一个例子，`192.168.1.0/24` 子网的网络地址是 `192.168.1.0` ，广播地址是 `192.168.1.255`；而 `10.2.0.0/16` 子网的网络地址是 `10.2.0.0`，广播地址是 `10.2.255.255`。

需要注意，两个 IP 在不同的前缀长度下，未必属于同一个子网。考虑 `192.168.1.1` 与 `192.168.1.5`。当前缀长度为 24 时，这两个 IP 的所处子网的网络地址均为 `192.168.1.0`，

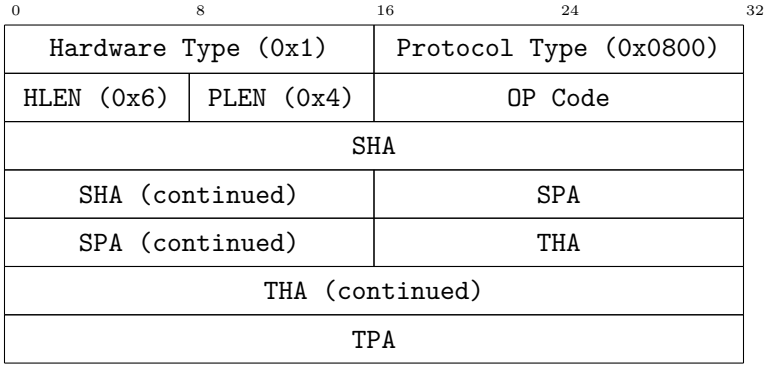
主机号分别为 1 和 5。而当前缀长度为 30 时，前者网络号为 `192.168.1.0`，主机号为 1；后者所处子网的网络地址为 `192.168.1.4`，主机号也是 1。

从上面的叙述可以看出，`a.b.c.d/p` 这种表示方法既可以用于指代一个特定的 IP 地址及其前缀长度，也可以只用于指代一段 IP 地址（一个子网）。因此，在遇到这一记号时，请特别**注意上下文**，以确定属于哪种情况。

7.3 主机的通信与 ARP 协议

当同一个广播域上的两台主机处于同一个子网中时，它们可以直接进行通信。由于发送以太网帧需要填写正确的目标 MAC 地址，因此需要能够从 IP 地址查询目标主机的 MAC 地址。ARP (Address Resolution Protocol) 协议就能完成这一工作。ARP 协议直接作为以太网帧的负载，长度为固定的 28 字节，构成如图 3 所示。

图 3: ARP 协议的数据格式



在构造 ARP 协议头时，所有整数字段都应该采用**大端序**。其中标记了默认值的字段只需要对应填写即可。`OP Code` 是操作码，1 代表请求，2 代表应答。`SHA` 与 `THA` 分别是发送者和接收者的硬件地址（MAC 地址），`SPA` 和 `TPA` 分别是发送者和接受者的协议地址（IP 地址），这些字段的字节顺序与上面提到的两种协议中相同。在部分题目中，数据保证 `SHA` 与以太网帧中发送者的 MAC 地址相同；如果题面没有相应的表述，则不能保证。

通常，ARP 的协议流程分为两步，即请求和应答：

1. 发起询问的主机发送 ARP 请求包，（链路层帧头中）源 MAC、`SHA`、`SPA` 为自身接口单播地址，目标 MAC 是广播地址，`THA` 为零，`TPA` 为其请求的 IP 地址。

2. 拥有该 IP 地址的主机发送 ARP 应答包，（链路层帧头中）源 MAC、`SHA`、`SPA` 为自身接口单播地址，目标 MAC、`THA`、`TPA` 为请求者的单播地址

需要注意，当一个主机上有多个网络接口时，响应 ARP 请求时应该选择接收到请求的接口，即使用该接口的 MAC 地址和 IP 地址作为源地址和应答内容。

每当查询到对应关系后，主机会将它存入自己的 ARP 缓存中，并设置一定的超时时间；在这段时间内，发往相同 IP 地址的分组可以直接从缓存中查询到对应 MAC 地址。除此之外，网络上还可能有无偿（Gratuitous）的 ARP 应答，表示某主机主动公告自己的 IP 和 MAC 对应关系。与一般的应答不同，它是一个广播的以太网帧，并且 `SPA` 与 `TPA` 相同为该主机的 IP 地址，`SHA` 为该主机的 MAC 地址，而 `THA` 为全零。主机应当关注这些消息，以及时更新自身的缓存。

7.4 IP 协议头校验和的计算

IP 协议采用的校验和称为 Internet Checksum，它的一种计算流程如下所述：

1. 将分组头中的校验和区域填充为 0
2. 将整个分组头视作大端序 16 比特整数的数组，将所有 16 比特整数相加
3. 如果和发生溢出，则将其截断为低 16 比特和溢出部分，将溢出部分加到低 16 比特（如 `0x1CB2F -> 0xCB2F + 0x1`）
4. 如果上述操作中又发生了溢出，则重复上述操作，直到不发生溢出
5. 将得到的结果按位取反，使用大端序填充到校验和区域

7.5 路由表与路由转发

处于两个子网的主机无法直接通信，需要借助同时连接了两个子网的主机（称为网关）进行分组的转发。

一台主机上的路由表由多个表项构成，每个表项的格式为 `A.B.C.D/m via E.F.G.H if N metric M`。其中 `A.B.C.D/m` 称为目标网络，`E.F.G.H` 称为下一跳，`N` 称为出接口编号，`M` 称为跃点数。这条表项的含义是，如果分组的目标地址的网络号与目标网络匹配，则将这个包的数据链路层帧头的目标 MAC 修改为下一跳的地址，从接口 `N` 发出，发送的代价是 `M`。

每当主机需要发送一个目标地址与其不在同一子网中的分组，它会查询自己的路由表寻找匹配的项目。如果有超过一个项目匹配，它会选择**网络前缀长度最长**的，这称为最长前缀匹配。如果依旧有多条匹配，再选择跃点数最小的。作为一个简单的例子，如果有下列路由表：

```
10.1.0.0/16 via 1.1.1.1 if 2 metric 10
10.1.1.0/24 via 2.2.2.2 if 3 metric 20
10.1.1.0/24 via 2.2.2.3 if 3 metric 15
0.0.0.0/0   via 3.3.3.3 if 4 metric 10
```

当主机需要向 `10.1.1.23` 发送分组时，这四条路由表都能够匹配目标 IP 地址，因此主机会选取前缀最长的，即第二和三条；而后，再比较跃点数，最终选择第三条。需要注意，其中的第四条路由比较特殊：根据匹配规则，它能够匹配所有的目标地址，一般称为默认路由。只连接了一个子网的主机往往只需要一条默认路由即可工作，它的下一跳就是这个子网的网关。

虽然大部分主机上的路由表项都不多，但是网络中存在一些节点被称为边界路由器，它们会保存整个 Internet 上的路由表项（目前数量已经接近一百万条）。而每一个分组经过都需要查询对应的路由，因此保存、更新、查询这些表项的效率是影响网络的吞吐量的关键因素。网络设备通常使用硬件来帮助存储和转发，而现代操作系统中也使用了各种优化的数据结构，一般都在 Trie 树的基础上加以针对性优化。我们提供了下列三篇文献供你参考，或许会对你的实现有所帮助。

`poptrie.pdf` *Poptrie: A Compressed Trie with Population Count for Fast and Scalable Software IP Routing Table Lookup*

`lc-trie.pdf` *IP-Address Lookup Using LC-Tries*

`linux.pdf` *IPv4 route lookup on Linux*

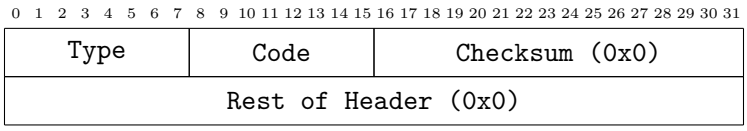
同时，在路由过程中还有一个重要的机制是 TTL。每当一个分组经过路由器，它的 IP 分组头中的 `TTL` 字段就会被减一（因此整个分组头的校验和也需要重新计算）。如果 `TTL` 在到达目标前减到 0，它就会被路由器丢弃，并给源主机返回错误信息（详见下一节）。这样的设计能够有效地防止网络中的环带来的危害，减轻路由器的负担。

IP 协议的核心就在于路由转发的过程，一个分组往往需要经过多跳才能到达其目的地，这其中经历了多次路由的选择。各个路由器之间的连接关系错综复杂，分组到达目的地的路径往往不止一条，这保证了整个网络不会因为某些节点的故障而瘫痪，但是也使得某些转发途径不是更优的。这是 Internet 设计过程中的一个重要取舍，使它足够健壮，以至于能够发展到今天的宏大规模。

7.6 ICMP 协议简介

IP 协议中承载一个重要协议为 ICMP (Internet Control Message Protocol) 协议，用于传递与 IP 协议相关的控制信息。ICMP 协议在 IP 分组头中的协议号为 `0x01`，协议头总是 8 字节长，数据格式如图 4 所示。

图 4: ICMP 协议头的数据格式



其中 `Type` 代表消息的分类，而 `Code` 是具体的类别。我们将用到的只有四个组合：Echo Request(`Type=8, Code=0`)，Echo Reply(`Type=0, Code=0`)、Destination Network Unreachable(`Type=3, Code=0`) 和 Time Exceeded(`Type=11, Code=0`)。`Checksum` 为整个 ICMP 协议头和数据的校验和，但为了简化，本题中始终为 0（因此你可以忽略 Wireshark 中显示的校验和错误）。在我们的简化实现中，协议头的剩余 4 字节总是 0。在协议头之后还可能附带一定长度的负载，负载的长度需要从 IP 头中长度字段推断。

平时最常使用的用于测试网络连通性的工具 `ping` 就利用了 ICMP 的 Echo Request 和 Echo Reply 消息，探测的主机向目标主机发出 Request 报文，目标主机如果收到，则会返回 Reply 报文。如果能够收到 Reply 报文，就说明网络的两个方向都是连通的。

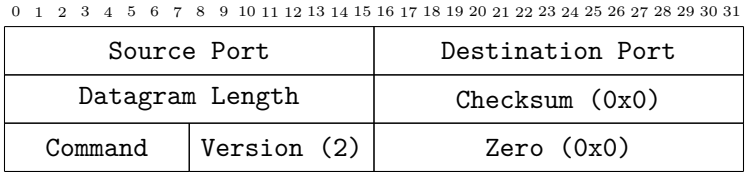
Destination Network Unreachable 消息在无法找到某个分组的下一跳去向时发送，Time Exceeded 在某个分组的 TTL 被减到 0 无法继续转发时发送。这两种错误消息都应该由路由器发送给产生错误的分组的源主机，发送时的源 IP 地址、MAC 地址都应该选择能够到达目标主机的网络接口的对应地址。在现代操作系统中，由于多种机制的共同作用，这种源地址选择的过程可能非常复杂。但在本题中，你只需要直接使用收到的分组的对应信息即可。

ICMP 协议同样可以在协议头后携带负载，Echo Reply 要求复制响应 Echo Request 中的负载；而错误信息的负载是 **产生错误的 IP 分组的分组头与前 8 字节数据**。

8 RIP 路由协议

一个路由器可能连接多个不同的子网，它的路由表可能也会非常复杂。同时，网络的连接关系可能因为人为或者非人为的原因发生变动。因此，为了让正确路由信息能够迅速传播，我们使用各类路由协议来在路由器之间互相通告路由信息的变化。一个简单而常用的路由协议是 RIP（Routing Information Protocol）。RIP 协议使用传输层协议 UDP 的 520 端口进行通信，因此它被承载在 UDP 数据包中，而 UDP 数据包被承载在 IP 数据包中，协议号为 `0x11`。一个包装在 UDP 数据包中的 RIP 协议头数据格式如图 5 所示。由于不要求构造 RIP 数据包，因此我们无需过多关注协议头中的字段信息。

图 5: 包装在 UDP 协议中的 RIP 协议头格式

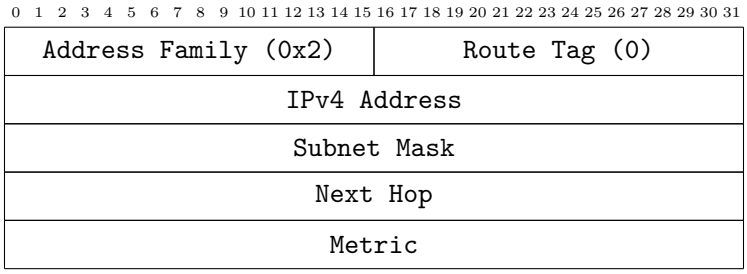


UDP 数据包中 `Source Port` 和 `Destination Port` 是端口号，对于 RIP 协议，两个端口号都为 `520` 。`Datagram Length` 为 UDP 协议头加上数据的总长度，在这里就是 UDP 协议头加上 RIP 数据的总长度。`Checksum` 为 UDP 协议头和数据的校验和，但为了简化，本题中始终为 0，即使 Wireshark 中会显示校验和错误。

`Command` 代表 RIP 的请求类型，`Version` 为 RIP 协议的版本号，始终为 2 。

在协议头后，跟随着的是多个 RIP 表项，每个 RIP 报文中最多含有 25 个表项。每个表项的格式如图 6 所示，长度为固定的 20 字节。表项的所有字段都是**大端序**的，包含了一条路由表所有必要的信息，其中目标网络的网络地址、下一跳 IP 地址都以原始的 32 位整数形式存储。其中的 `Subnet Mask` 称为子网掩码，每一个前缀长度 p 可唯一地对应一个前导 p 个二进制位均为 1 且剩余 $32-p$ 位均为 0 的子网掩码（如 $p=24$ 对应的子网掩码可写为 `255.255.255.0`）。RIP 协议有一个特殊的约定：**如果表项的下一跳地址为全零，表示下一跳是发出该 RIP 数据包的主机的 IP 地址**。

图 6: 每个 RIP 表项的数据格式



在本题中，你需要处理的 RIP 报文类型只有 `Command` 字段为 `0x2` 的，即 RIP Response 报文。这种类型的报文一般来自于主机连接的网络上的其他路由器定时主动播发自己具有的路由信息。作为简化的处理流程，当主机收到一个 RIP Response，它需要遍历每一个 RIP 表项，进行如下处理：

- 如果本机路由表中没有到目标网络的表项，并且收到表项的跃点数小于 16，则将收到

表项的跃点数**加一**，并插入本机路由表。

- 如果本机路由表中有到目标网络的表项，并且收到表项的跃点数等于 16，则移除本机路由表中相应的表项。
- 如果本机路由表中有到目标网络的表项，并且收到表项的跃点数**小于**现有表项的跃点数，则将收到表项的跃点数**加一**，并替换现有的表项。

这意味着，跃点数填充为 16 等同于告知“网络不可达”。假设默认路由表为空，依照上述规则维护路由表；则同一时刻对于某个特定的目标网络，**至多存在一条**表项，读者自行证明不难。

9 参考资料

互联网协议的制定组织是 IETF (Internet Engineering Task Force)，它是一个松散的民间组织，任何人都可以加入和提出标准。这些标准都被称为 RFC (Requests for Comments)，不具有行政或法律的强制力。尽管如此，大量的 RFC 都成为了世界通行的公认标准。正是这种自由开放的精神，促进了世界范围内互联网的高速发展。

下面我们将给出题目中涉及的各个技术的 RFC 文档编号，并将这些 RFC 全文作为题目附件一同发布。请注意，出于严谨性考虑，RFC 的行文措辞通常冗长而详细；同时，我们对题目的要求也经过了简化，**未必符合原有 RFC 中的规范**。因此，虽然我们鼓励对细节的追求，但请不要沉迷其中，以免浪费宝贵的答题时间。

RFC 791 `Internet Protocol` 描述 IP 协议的消息格式、实现方式等

RFC 2474 `Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers` 描述 IP 协议中的 `Type of Service` 字段的语义

RFC 6864 `Updated Specification of the IPv4 ID Field` 描述 IP 协议中 `Identification` 字段的语义

RFC 950 `Internet Standard Subnetting Procedure` 描述使用有类地址划分子网的方式（目前已不再采用）

RFC 4632 `Classless Inter-domain Routing (CIDR): The Internet Address Assignment and Aggregation Plan` 描述使用无类域间路由（CIDR）进行网络划分的方式

RFC 826 `An Ethernet Address Resolution Protocol` 描述了 ARP 协议的消息格式、交互流程等

RFC 5227 `IPv4 Address Conflict Detection` 描述了使用 ARP 协议进行 IP 地址冲突检测的过程

RFC 5494 `IANA Allocation Guidelines for the Address Resolution Protocol (ARP)` 描述了对 ARP 协议内某些字段的再分配

RFC 792 `Internet Control Message Protocol` 描述了 ICMP 协议的消息格式和使用场景等，规定 ICMP 消息类型

RFC 4884 `Extended ICMP to Support Multi-Part Messages` 为 ICMP 协议添加多 IP 分组支持

RFC 6633 `Deprecation of ICMP Source Quench Messages` 由于安全等原因废除 ICMP Source Quench 消息类型

RFC 6918 `Formally Deprecating Some ICMPv4 Message Types` 废除更多未使用的 ICMP 消息类型

RFC 768 `User Datagram Protocol` 描述了 UDP 协议的消息格式和用途等

RFC 2453 `RIP Version 2` 描述了 RIP 协议第二版的消息格式、交互流程等

RFC 4822 `RIPv2 Cryptographic Authentication` 为 RIPv2 协议添加加密和认证支持

最后，希望这道题能够增进你对网络的了解和兴趣。祝你成功!