第 0 章

预 备 知 识

0.1 系统环境与代码库

本书中的所有实验皆假设所使用的实验环境为 Ubuntu 操作系统。本书另外附带了大量用于配置实验环境的工具、进行实验内容评测所需的代码与数据文件等，我们将这些文件托管于 Github 上的组织 `pkuNetLab`[①] 中。在本书的余下部分中，将用形如“`repo::path`”的文本指向 `pkuNetLab` 中的代码仓库 `repo` 中路径为 `path` 的文件。

0.2 说明文档查询工具 `man`

Linux 命令行工具 `man` 是 Linux 系统中通用的说明文档查询工具，其程序名称 `man` 是单词 Manual 的前三个字母。在本书涉及 Linux 系统命令行工具使用及编程的章节中，将假设读者已经熟练掌握了 `man` 的使用方式并能够使用该工具查询所需要使用的各个命令行工具及库函数的说明文档。在本节中，将介绍该工具的基本用法。

```
$ man printf
PRINTF(1)                        User Commands                        PRINTF(1)

NAME
       printf - format and print data

SYNOPSIS
       printf FORMAT [ARGUMENT]...
       printf OPTION

DESCRIPTION
       Print ARGUMENT(s) according to FORMAT, or execute according to OPTION:

       --help display this help and exit

       --version
              output version information and exit

       FORMAT controls the output as in C printf.  Interpreted sequences are:
```

① https://github.com/pkuNetLab.

```
       \"     double quote

       \\     backslash

       \a     alert (BEL)
Manual page printf(1) line 1 (press h for help or q to quit)
```

如上所示的命令将在当前终端中显示 printf 的帮助文档。一般情况下，如果通过 Linux 对应发行版的软件包管理工具安装了软件（假设其名称为 item），那么形如 man item 的命令会显示该软件的使用说明。对于一些函数库，也能使用 man 查询其包含的库函数的使用说明，如 GNU C 标准库 glibc 中包含的函数，我们在链路层相关实验中使用的 libpcap 包含的函数等。

细心的读者可能注意到，如上所示的命令 man printf 展示的文档并不是 C 标准库函数 printf 的说明，而是同名命令的说明。这是因为为了管理冲突的名称，man 提供了“章节”的概念，同时提供章节名称与条目名称才能唯一确定一篇文档。当条目名称唯一时，man 会自动找到这篇文档，而当名称不唯一时，它将会按照用户指定（或默认）的顺序在各章节进行搜索并展示它找到的第一篇文档。man 程序给出的规范中同时也包含了最常用的几个章节的定义：章节 1 包含了可执行程序与命令的说明文档；章节 2 包含了系统调用的说明文档；章节 3 包含了库函数的说明文档。若希望查找库函数 printf 的说明文档，则应当执行 man 3 printf。值得注意的是，章节名称并非必须是数字。一些文档编写者可能会将自己的文档放在一个名称特别的章节中以进一步防止名称冲突。man 命令的 -f 选项可以用于寻找页面所在的章节。如下所示，形如 man -f item 的命令将会列举所有名称为 item 的条目。

```
$ man -f printf
printf(1)            - format and print data
printf(3)            - formatted output conversion
```

提示 0.1 在本书的余下部分中，将用形如“man::item(x)”的文字表示“man 命令可查阅的 x 章节中的 item 页面包含了更加详细的信息”。

在阅读 man 命令输出的帮助文档时，可以用方向键与 PgUp/PgDown 键翻阅文档。文档阅读界面同时也支持 vi 形式的指令以进行更加高级的文档内导航，如在页面中输入:1000 则会跳转到第 1000 行，而在页面中输入/prin[a-z] 则会从当前页首开始搜索正则表达式 prin[a-z] 的下一个匹配。在进行搜索后，输入 n 可以前往下一个匹配，而输入 N 可以前往上一个匹配。

关于命令 man 的更多知识，如更多章节名称的定义、命令行选项的功能及说明文档的撰写方法等，详见 man::man(1)，此处不再介绍。同时也建议读者在实验过程中遇到不了解某个 Linux 命令或者库函数的使用方式的情况时，首先考虑使用 man 工具查阅帮助文档（尽管这些文档可能更难阅读），因为这些帮助文档一般由命令或库函数的作者撰写，相比于网络上常见的帮助信息而言，具有更高的准确性。

第一部分

经典计算机网络与网络协议栈

在本书的第一部分中，我们将展开一段艰辛而令人兴奋的旅程：从链路层开始逐步向上地设计并实现一个自定义的 TCP/IP 协议栈（称为 `HomeStack`），并在这一协议栈上实现安全文件传输协议（SFTP）以支持节点间的文件传输。在这一系列实验中，将从开发者的角度系统性地认识网络协议栈的设计原理与实现方式，并通过自己的实现经验了解身边的网络主机中运行的 TCP/IP 协议栈的运行机理。本部分前三章的配套代码位于代码仓库 `pkuNetLab/lab-netstack` 中，第 4 章的配套代码位于代码仓库 `pkuNetLab/lab-sftp` 中，相信读者在完成本部分工作后对计算机网络系统的理解会大大加深。在踏上旅程之前，让我们首先看看旅行计划。

Linux 系统不仅为使用者在本地主机上的操作提供了诸多便利，同时也支持远程连接到其他 Linux 主机进行操作。Linux 系统下的远程操作主要通过安全终端（SSH）协议进行，`ssh` 程序可以在远程机器上启动一个终端程序供用户进行远程操控并保证通信的安全性（比如网络中其他用户不能看到通信双方发送的内容），而 `sftp`［`man::sftp(1)`］程序则将 SSH 协议作为媒介保证本身未加密的文件传输协议（FTP）的安全性，实现了本机与远程主机间的文件传输。实现 SFTP 协议正是本系列实验的最终目标。

从计算机用户的角度看来，安全地传输文件这一操作是简单的：在终端中输入一行命令调用 `sftp` 程序即可完成，但若从网络协议栈各层来看，这一操作则包含了极大的复杂性。图 1 展示了一次安全文件传输中协议栈各层会进行的操作。

链路层的工作是相对简单的。对于链路层来说，它所需要做的全部仅仅是将协议栈上层交给它的数据包发向局域网中下一个（具有指定地址的）节点，以及从网络中接收数据包并提交给上层。在现代计算机系统中，从电路中识别数字信号与数据包等工作都由网络设备自动完成，所以只需要建立与网卡的联系，并调用网卡的功能收发数据包即可完成这一部分的工作。下面将在第 1 章中实现这些内容。

有了链路层的收发数据包机制，便可以实现网络层的 IP 协议。IP 协议的任务是将数据包发往指定的 IP 地址（即文件服务器地址），但 IP 协议并不直接与文件服务器通信，只是将数据包发往“前往指定地址路程中的下一站”。这一过程正是一般所称的“路由”。IP 网络中其他节点收到数据包之后，也会通过路由将数据包不断地发往离指定地址更近的“下一站”，直到数据包到达目的地。为了实现这一功能，IP 模块需要与网络中其他的 IP 节点交换信息以实现路由，还需要通过地址解析协议（ARP）建立 IP 地

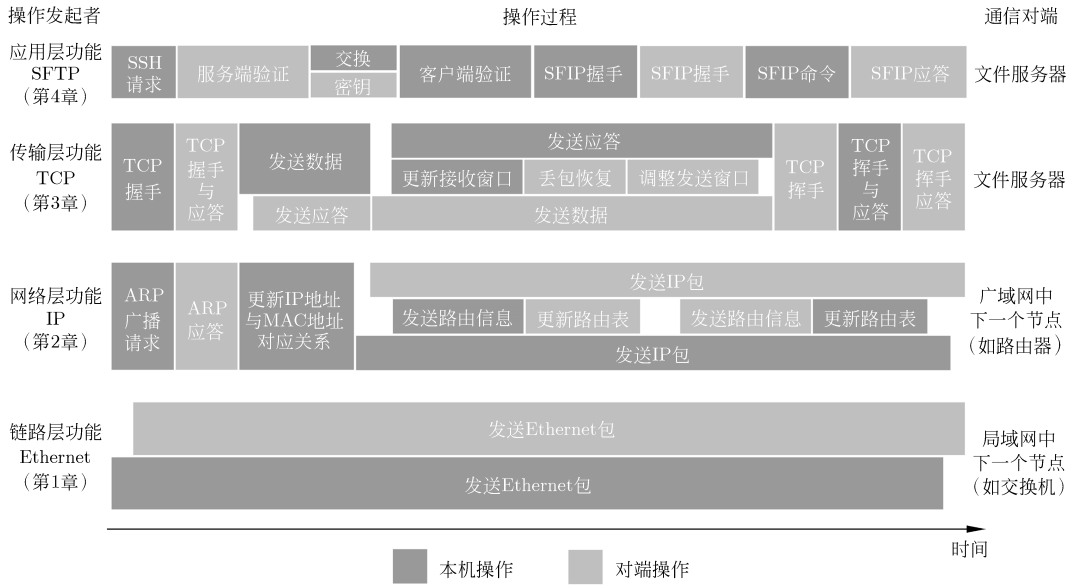

图 1　协议栈各层在一次安全文件传输中进行的操作

址与链路层地址之间的映射，从而利用链路层功能进行数据包的收发。下面将在第 2 章中实现这些内容。

有了网络层将数据包发送到指定的地址，便可以实现传输层的 TCP 协议。TCP 协议的任务是连接到指定的 IP 上指定的 TCP 端口，并可靠地传输数据。在这一过程中，TCP 模块首先需要与通信的对端进行“握手”，即连接的建立。此后，本地的 TCP 模块需要将文件传输的指令发送到对端，之后等待对端将文件数据传回，并提交给用户。传输数据这一任务也并不简单：在传输数据的过程中，TCP 模块需要在可能有数据在发送过程中丢失的情况下通过重传数据包保证数据的可靠性，也需要控制自身收发数据的速率，防止网络过载或本地内存不足。在完成数据传输后，TCP 模块还需要与通信的对端进行“挥手”，即连接的关闭。下面将在第 3 章中实现这些内容。

终于，借助传输层提供的接口，可以实现应用层的 SFTP 协议并真正使用网络协议栈。SFTP 协议的任务是利用 TCP 协议已经建立的数据传输通道安全地传输文件，这里的“安全”指的是客户端和服务器都能够自证身份，且传输的数据不会被监听和篡改。为达到这一目的，首先需要通过 SSH 协议在客户端和服务器之间建立一个安全的数据传输通道，这包含双方交换密钥等一系列“握手”过程。此后客户端便可以利用 SFTP 协议定义的操作来读写服务器的文件。下面将在第 4 章中实现这些内容。

当到达旅程的终点时，我们将能在 Linux 主机上通过实现的文件传输客户端与其他 Linux 主机上的文件服务器进行通信并进行正常的交互，也能使其他网络应用使用实现的 TCP/IP 协议栈相关接口进行通信。

第 1 章

链路层：Ethernet

从本章开始，我们将从链路层到传输层递进式地设计与实现一个具有完整功能、能够与一般的网络主机通信的 TCP/IP 协议栈（将之命名为 `HomeStack`），深刻理解网络协议栈的设计思想与实现原理。

在本章中，我们将第一次接触网络数据包分析工具 Wireshark [1]。Wireshark 是进行网络分析、错误定位的主流工具，在实现 `HomeStack` 的相关实验内容中，我们也将学习如何使用 Wireshark 进行协议栈各层的数据分析与问题排查，从而掌握使用该工具对自己的程序实现与实际网络问题进行调试的能力。

在本章中，我们也将实现 `HomeStack` 中链路层的相关内容。链路层的主要功能（如图 1.1 所示）包括 0-1 数据流与数据包间的相互转换、提供多用户访问能力，以及区分上层协议等。在现代计算机系统中，网卡或网卡驱动一般会自动完成拆装数据包以及识别数据包的发送者与接收者的工作。此外，不同的链路层协议也会通过不同方式让开发者能够指定数据发送的目标用户及上层协议。在本书中，我们不考虑网络设备自身的内部实现，而是在拥有网络设备的情况下，尝试利用链路层协议为上层实现提供功能接口。我们将从以太网（Ethernet）设备与现行的一种以太网数据包格式标准 Ethernet II 入手，学习进行以太网数据包的收发与以太网数据包头中各数据域的填充。

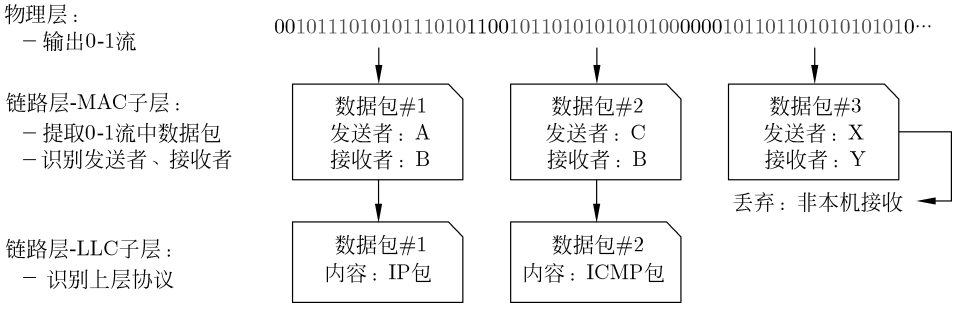

图 1.1　链路层的主要功能

1.1　实 验 目 的

在本章中，我们将掌握如下内容：

1. 熟悉数据包分析工具 Wireshark 的基本功能。
2. 熟悉网络主机使用 Ethernet 包进行通信的流程。

3. 使用 libpcap 进行网络设备的管理及 Ethernet 包的收发。

1.2 实验环境配置

在本章中，我们可能需要用到的新命令行工具及其安装方式如表 1.1 所示。

表 1.1 本章新命令行工具及其安装方式

工具名称	工具类型	包含该工具的 apt 软件包
Wireshark	应用程序	wireshark
Tshark	命令	tshark
ip	命令	iproute2
libpcap	函数库	libpcap-dev

Wireshark 工具可以在官网 [1] 或包管理器下载。我们可以从 Wireshark 的手册和问答页面中找到很多有用的帮助信息。

提示 1.1 使用 Wireshark/Tshark 嗅探数据包时常出现的一种报错是提示权限不足。一般而言权限不足报错可以通过以超级用户身份执行程序的方式解决，但 Wireshark/Tshark 出于安全性考量，并不支持以超级用户身份执行。官方对该问题建议的解决方案是将 Wireshark/Tshark 分配至一个具有数据包嗅探权限的用户组，但存在该解决方案在一些情况下不能正确解决问题的报告。一种更加有效（但也会泄露更多权限）的解决方案是赋予 Wireshark/Tshark 使用的数据包嗅探子程序 dumpcap 超级用户权限，并将之设置为允许所有用户执行。执行这一操作所需的命令为 chmod 4755 path [详见 man::chmod(1)]，其中 path 是 dumpcap 的程序路径。

本章另需使用工具 ip 进行 Linux 虚拟网络设备的创建。Linux 虚拟网络设备有多种类型，而在本章中我们将主要使用虚拟以太网设备（veth）。本书不介绍 ip 的具体使用方法，而是通过提供一组实用工具（vnetUtils）封装所需使用的功能，这一组实用工具在网络层与传输层的实验中也将会有对应的应用。下面介绍使用这一组实用工具创建 Linux 虚拟以太网设备的方法。

```
$ cd lab-netstack::3rd/vnetUtils/helper
$ ./addVethPair name1 name2
```

如上所示的调用会创建一对 Linux 虚拟以太网设备。Linux 虚拟以太网设备总是成对出现，每一对 Linux 虚拟以太网设备可以看作两张由网线连接的有线网卡。当用户向其中一端（即其中一个虚拟以太网设备）发送数据包后，在另一端（即另一个虚拟以太网设备）可以收到发送的数据包。Linux 虚拟以太网设备对也是搭建虚拟网络所需的核心组件之一。可以使用 Linux 虚拟以太网设备对和数据包嗅探工具（Wireshark 或

Tshark）验证 HomeStack 链路层实现的正确性：在一端写入（发送）数据包之后，应当能在另一端抓取到写入的数据包；抓取到的数据包应当具有正确的 Ethernet II 数据包头，并且能被 Wireshark 正确解析。

提示 1.2 使用 vnetUtils 工具可以方便地创建虚拟网络拓扑，HomeStack 将会运行在这样的网络拓扑上。请仔细阅读 lab-netstack::3rd/vnetUtils/README.md 并掌握其用法。

1.3 实 验 内 容

1.3.1 Wireshark

Wireshark 是一款功能丰富的开源网络数据包分析软件。Wireshark 带有图形用户界面，其对应的命令行工具是 TShark。在本次实验中，我们将学习 Wireshark 的基本使用，并利用它完成网络数据包的分析。在本书的所有实验中，用到的 Wireshark 功能包括：

- 实时地嗅探经过指定网络接口的数据包。
- 查看数据包中每一字节所属的网络协议和具体含义。
- 对网络数据的流量、时延等属性进行统计。
- 使用过滤器，只保留我们关心的数据包。

下面，我们将学习 Wireshark 的两个主要功能——数据包嗅探和数据包分析。

数据包嗅探

首先是数据包嗅探（也叫作“抓包”“数据包捕获”）。顾名思义，Wireshark 会“嗅探”指定网络接口发送和接收的数据帧，而 Wireshark 本身不会发送任何数据。被嗅探的数据可以保存在文件中，便于后续的回放 [2] 和分析。常用的文件格式有 PCAP [3] 及其扩展 PCAPNG（PCAP Next Generation [4]）。在后面的实验中，使用 Wireshark 或 TShark 来调试实现的协议栈。

打开 Wireshark 后，欢迎主界面列举了当前可以进行嗅探的网络接口。如果接入的是 Wi-Fi 无线局域网，双击类似“WLAN”或“Wi-Fi en0”的字样即可开始嗅探。也可以在菜单栏“捕获-选项”中进行更详细的嗅探配置。

旁注 1.1（Wireshark 的嗅探原理） Wireshark 利用 pcap 进行实时的数据包嗅探。pcap（全称 Packet Capture）是一组用于嗅探和生成网络流量数据的应用程序接口。pcap 接口在不同的操作系统下有不同的具体实现。libpcap 是类 UNIX 系统下的 pcap 接口的一种实现；tcpdump 是封装了 libpcap 的命令行工

具。而对于 Windows 操作系统，pcap 接口有 WinPcap 和 Npcap 两种实现，其中 WinPcap 的维护在 2013 年已经停止。

pcap 接口在类 UNIX 系统下通常使用原始套接字 [5] 实现。如图1.2所示，当有新的数据包要发送和接收时，内核会将数据包复制一份，然后转发给打开了原始套接字的应用程序（比如 Wireshark）。

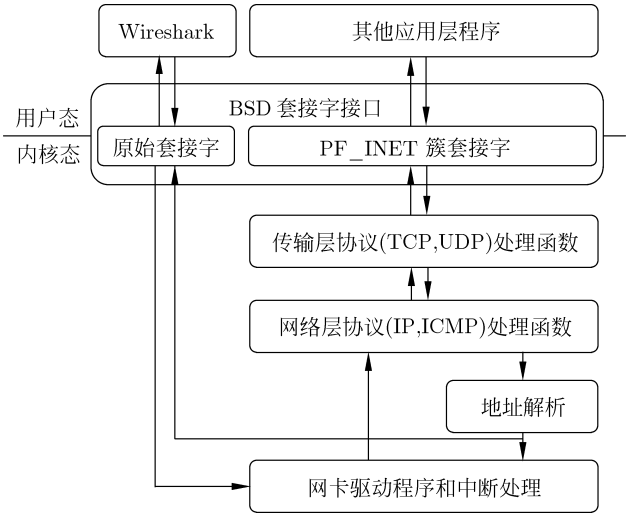

图 1.2　原始套接字可用于实现数据包嗅探

练习 1.1

尝试在 Wireshark 开始数据包嗅探。嗅探开始后在浏览器中访问自己喜欢的网站，并观察 Wireshark 的输出。最后在菜单栏中选择“文件-保存”将嗅探结果保存到文件中。

在终端环境中，使用命令行工具会便于操作和编写自动化脚本。Wireshark 提供了很多嗅探、分析、处理数据包的工具，比如 Tshark、capinfos、editcap。可以在 Wireshark 的菜单栏中选择“帮助-说明文档”找到这些工具的使用方法。下面介绍 Tshark 这一命令行工具。具体的使用方法见 `man::tshark(1)`，这里简单列举几个实用的例子。

```
# 从网络接口 eth0 嗅探数据包，并保存到文件 result.pcap
tshark -i eth0 -w result.pcap
# 显示文件 result.pcap 的内容
tshark -r result.pcap
# 显示文件 result.pcap 的每个数据包的字节，这对后面调试可能会很有帮助
tshark -r result.pcap -x
# 显示文件 result.pcap 的包含 HTTP 信息的数据包
tshark -r result.pcap -Y http
# 统计文件 rseult.pcap 中 IP 层会话的数据
tshark -r result.pcap -q -z conv,ip
```

旁注 1.2（Tcpdump） Tcpdump 是另一个常用的命令行工具。Tshark 的很多命令行选项都是参考 Tcpdump 设计的，所以这两个工具的使用方法相差不大，不过 Tshark 提供了一些新的用于分析数据包的功能。

数据包分析

接下来，我们学习使用 Wireshark 进行数据包分析。图1.3是 Wireshark 的数据包分析界面，从上往下依次是：

- 菜单栏：包含了 Wireshark 所有的功能和配置。目前只用到了“文件”和“捕获”两个主菜单。
- 工具栏：包含了常用的快捷方式，比如开始新的捕获、停止捕获、打开已保存的捕获文件等。工具栏中的工具也能在菜单栏中找到功能一样的栏目。
- 过滤器：接收一个过滤表达式。Wireshark 将根据过滤表达式，只显示和表达式匹配的数据包。
- 已捕获的数据包列表：每一行显示一个捕获的数据包，每行的主要信息包括数

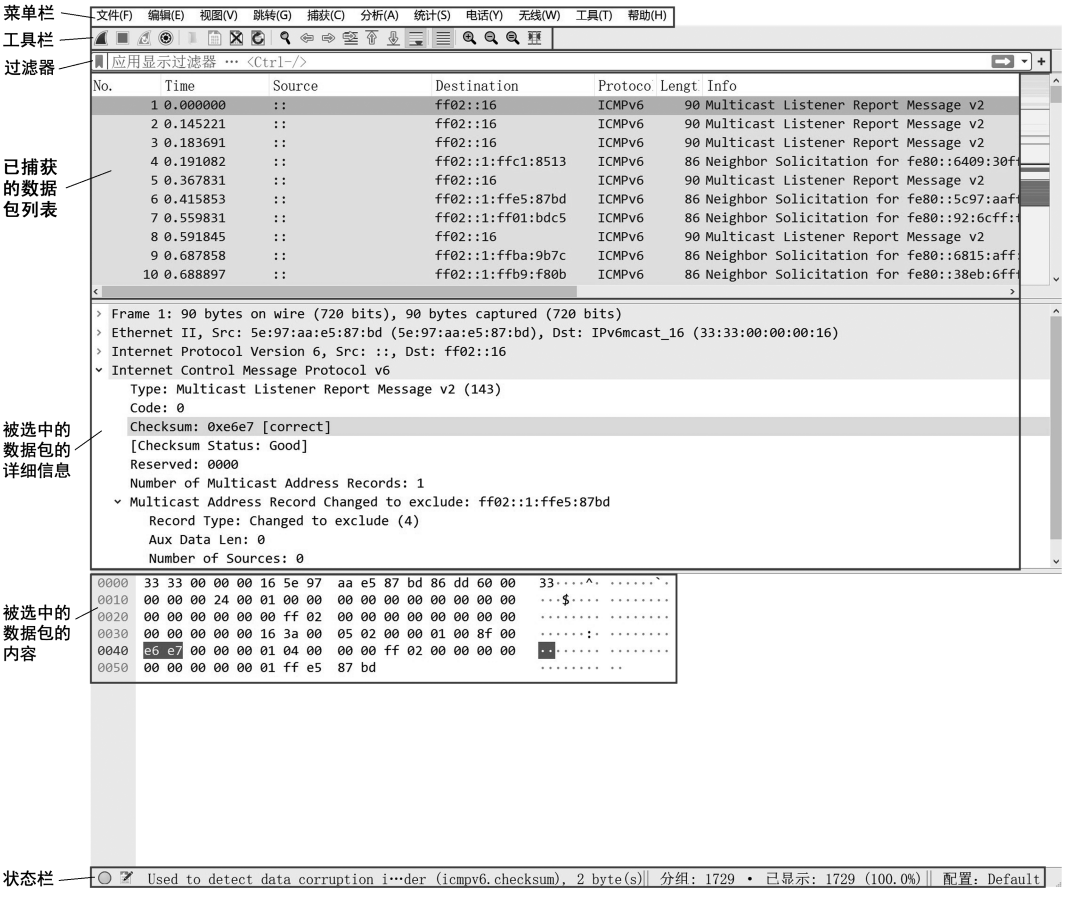

图 1.3　Wireshark 的数据包分析界面

据包编号、捕获的时间、数据包的源地址和目的地址、协议类型和协议携带的信息总结。

- 被选中的数据包的详细信息：显示了数据包列表中被选中的数据包的详细信息。
- 被选中的数据包的内容：显示了数据包列表中被选中的数据包的内容。数据包的每个字节将以十六进制打印出来。
- 状态栏：显示当前 Wireshark 的工作状态。

练习 1.2

用 Wireshark 打开文件 `lab-netstack::pcap-trace/trace.pcap`，完成以下实验。首先在过滤器中键入表达式 `eth.src == 6a:15:0a:ba:9b:7c`，只保留 Ethernet 源地址为 `6a:15:0a:ba:9b:7c` 的数据包。然后找到过滤结果中的第三个数据包，回答以下问题：

1. 符合过滤表达式的数据包有多少个？（提示：观察状态栏）
2. 这个数据包的 Ethernet 目的地址是什么？有什么特殊含义？
3. 这个数据包的第 71 个字节（从 0 开始计数）是什么？

1.3.2 基于 libpcap 的以太网数据包收发

在 1.3.1 节中，我们已经通过使用 Wireshark 抓取与分析 Ethernet 数据包对 Ethernet 中主机间通信的流程有了一定的认知。在本节中，我们将自己动手实现这一流程，使用 libpcap [6] 库进行以太网（Ethernet II）数据包的收发。以太网数据包广泛用于有线网络传输，绝大多数千兆速率有线网卡皆支持 GigE 特性，即支持以太网数据包。同时，Linux 虚拟以太网卡 [7] 也使用此类型数据包进行通信，而 Linux 虚拟网络也将是 `HomeStack` 运行的环境。在网络层的实现中，我们将基于本实验实现的函数接口进行数据通信。

与应用程序编写者所熟悉的套接字编程所不同的是，在链路层无法使用任何协议栈上层所提供的抽象，如实际应用中常见的 IP 地址或 TCP 端口等。相反，我们将直接与硬件建立联系，指定收发以太网数据包所使用的网络设备。在本实验中，我们将完成两部分任务：第一部分任务为在 `HomeStack` 中管理网络设备，第二部分任务则是使用在 `HomeStack` 中注册的设备进行数据包的收发。

其中，第一部分任务如下：

```
void DeviceManager::addDevice(pcap_if_t *dev);
```

- 该函数打开 `libpcap` 找到的一个网络设备并将其添加到设备管理数据结构中。
- `dev` 是通过 `pcap_findalldevs` 函数发现的 `pcap_if_t` 设备。

在第二部分任务中，我们将利用网络设备编号来指定网络设备，并在其上进行数据包的收发。在了解第二部分任务之前，让我们首先对 Ethernet II 数据包头格式进行一些介绍：