分 类 号 _____ 学号 M201673000

学校代码 10487 密级 _____

華中科技大學

硕士学位论文

基于 VxWorks 系统的网络监控工具设计与实现

学 位 申 请 人： 王飞

学 科 专 业： 计算机技术

指 导 教 师： 周正勇 讲师

答 辩 日 期： 2018 年 5 月 25 日

A Thesis Submitted in Partial Fulfillment of the Requirements

for the Degree for the Master of Engineering

The network monitoring tool based on VxWorks system: Design and Implementation

Candidate : Wang Fei

Major : Computer Technology

Supervisor : Lecturer. Zhou Zhengyong

Huazhong University of Science & Technology

Wuhan 430074, P.R.China

May, 2018

独创性声明

本人声明所呈交的学位论文是我个人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除文中已经标明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的研究成果。对本文的研究做出贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。

学位论文作者签名：

日期：　　年　　月　　日

学位论文版权使用授权书

本学位论文作者完全了解学校有关保留、使用学位论文的规定，即：学校有权保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权华中科技大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

本论文属于　保密□，　　在_____年解密后适用本授权书。

　　不保密□。

（请在以上方框内打“√”）

学位论文作者签名：　　指导教师签名：

日期：　　年　　月　　日　　日期：　　年　　月　　日

摘 要

本课题所设计的基于 VxWorks 系统的网络监控工具是根据实际的应用需求实现的系统，在设计系统过程中，注重系统的可扩展性与可维护性。网络监控系统采用 C/S 结构，包括网络包捕获软件和网络包分析软件两部分。网络包捕获软件运行在 VxWorks 实时系统，负责捕获网络数据包、过滤网络数据包并推送数据到主机端等，使用 C 语言开发；网络包分析软件实现 Windows 与 Linux 跨平台使用，负责解析网络数据包与信息展示、存储网络数据包、搜索数据包等，采用 Qt 框架开发。

本文详细介绍了网络监控系统的设计与实现。首先讨论了系统所用到的关键技术，确定了自行开发网络协议解析引擎、使用 MUX 接口实现网络抓包功能、定制简化版网络数据包过滤器、使用多线程与缓冲技术控制丢包的方案。接下来给出了网络包捕获软件的整体架构，包括控制模块、通信模块、数据包捕获模块、数据包过滤模块和缓冲过滤模块，网络包捕获软件使用 MUX 接口实现终端网络数据包捕获功能，并通过串口通信将网络数据包推送到主机端。然后讨论网络包分析软件的实现，实现对网络数据包实时进行协议分析、读取本地 pcap 文件解析协议以及数据包存储的功能，用户可以根据兴趣搜索历史网络数据包。本文最后对监控系统分别进行了功能测试与性能测试，测试结果说明监控系统运行良好，性能良好，CPU 和内存占用小，满足用户的要求。

关键词：VxWorks；网络监控；C/S；协议解析；多线程

Abstract

The network monitoring tool based on VxWorks system designed in this project is a system implemented according to the actual application requirements. In the process of designing the system, we payed attention to the scalability and maintainability of the system.The network monitoring system adopts C/S structure, including monitoring server and monitoring client two parts. The monitoring server runs on the VxWorks real-time system. It is responsible for capturing network data packets, filtering network data packets, pushing data to the client and so on. It is developed in the C language.The monitoring client can runs on the Windows and Linux.It is responsible for analyzing network data packets and displaying information, storing network data packets, searching data packets and so on. It is developed by using the Qt framework.

This paper introduces the design and implementation of network monitoring system in detail. Firstly, the key technologies used in the system are discussed. We self develop the engine of network protocol parsing , use MUX interface to implement the function to capture network packets and realize a simplified network packet filter. The monitoring system uses multi-thread and buffer technology to control packet dropout. Next the whole architecture of network packet capture software is given, including control module, communication module, data packet capture module, packet filtering module and buffer filtering module. The network packet capture software uses MUX interface to implement the network packet capture function of the terminal and pushes the network packets to the host terminal through serial communication. Then the implementation of network packet analysis software is discussed. The software realizes the function of the protocol analysis of network packet in real-time, the function of reading local pcap file parsing protocol and the function of packet storage. The users can search for historical network packets according to their interest. At the end of this paper, the function test and performance test of the monitoring system are carried out respectively. The test results show that the monitoring system runs well with good performance, as well the system runs with low CPU and memory rate. The system can meet the requirements of the user.

Key words： VxWorks; Network Monitor; C/S; Protocol Analysis; Multi-thread

目 录

第一章 绪论

1.1 研究背景

随着时代的发展，计算机网络技术和信息技术日新月异，互联网已经渗透到人民生活中的各个方面，网络成为生活工作中主要的通讯方式。基于网络的应用扮演着举足轻重的角色，在公司企业日常生活中随处可见，比如平常用的云，人们已经习惯了从网上下载或者备份自己的资料，越来越多的机密信息在网络中存储传输，人们越来越依赖网络[1]。同时网络技术在生活工作中扮演着举足轻重的角色，包括军政、金融等行业[2]。随着网络涉及到的业务范围越来越广，在网络中传输的数据量爆炸式增长，提高了网络管理的要求，增加了网络管理难度[2]。主要表现在：

- 网络结构日趋复杂，网络规模不断扩大，应用场景越来越多元化[3]
- 网络流量越来越大，网络流量即是网络上传输的数据量，要求更大的带宽[3]
- 如何保障网络的持续、安全和高效安全[3]

因此，加强网络管理成为当前亟需解决的问题，监测网络行为，对网络服务质量进行有效指标的评估，找出网络存在问题，优化网络结构[2]。其中我们可以监测网络流量进而监测网络的运行状态，监测网络运行的状态，尤其是否存在异常，从而提供直观的数据找出网络的问题。研究网络流量是目前常用的方法，可以通过网络流量获取网络数据大小，网络数据类型等，可以方便的对网络状态进行监控，管理网络[3]。

本课题源于实际的应用项目，以舰船系统之间的通讯方式为背景。串口通讯作为舰船通讯的主要方式，是点对点的通讯，通过串口交换舰船系统之间的数据[4]。随着网络越来越复杂，如何监控舰船中的网络状况变得越来越困难，因此需要一个工具帮助网络管理员监控管理网络。网络监控的主要任务是捕获分析网络数据包，为网络管理提供必要的信息[4,5]。国内现在的舰艇已经在作战系统中采用网络技术，如何保证网络环境安全，使得整个舰艇系统更加安全可靠，就需要使用监控系统实

时检测网络状态。目前舰艇采用实时多任务的 VxWorks 系统，具有快速响应的特点[4]。因此我们课题也主要针对 VxWorks 操作系统进行研究。VxWorks 是一款嵌入式实时操作系统，是课题研究所使用的服务器端环境，具有可裁剪、高性能的优点[6]。基于 VxWorks 系统的网络监控工具主要提供以下功能：

1. 监控 VxWorks 服务器的网络运行状况：主要包括网络协议的解析

2. 对网络运行状态数据的分析与保存：对服务端采集的网络数据进行分析处理，生成直观的数据表，并保存相关数据

3. 部署客户端

基于要解决的问题，系统采用了经典的 C/S 机构，在 VxWorks 服务器端使用 mux 网络驱动的方式对 VxWorks 网络流量进行抓取统计，在客户端使用 Qt 设计页面反应当前的网络状态，根据图形方式直观的显示实时流量信息。C/S 架构可以满足每个用户进行网络流量的统计分析。使用此软件可以分析、掌握网络行为的基础，通过采集网络流量数据监控网络流量的变化[7,8,9]。网络管理员在网络流量监控的基础上观察网络及设备性能变化情况，同时又可以对于网络流量进行文件备份，以及查看文件备份情况，因此使得网络管理更加高效、简单和易用。

1.2 研究目标、内容和意义

网络监控工具提供了一种方式，帮助管理员研究网络环境的构造，更加直接反应当前的网络状态[7]。这也是解决网络管理难得一种方法。本课题的研究目标是设计并实现一款 C/S 架构的网络流量监控软件，对用户的特定网络进行监控，这样管理员可以及时发现问题并解决。

本课题的研究内容主要包括下面几个方面:

1. 需求分析：充分与用户沟通，了解用户的业务需求，根据需求，设计满足用户的解决方案

2. VxWorks 服务器数据包抓取：研究网络抓包原理，选择合适的在 VxWorks 下的网络抓包方式，设计良好的服务端程序

3. 客户端设计：使用 Qt 实现跨平台设计，主要是 Windows 和 Linux；对服务器发来的网络数据包进行协议解析展示，设计可靠的协议解析展示客户端。

本课题得研究意义主要如下：

1. 满足用户的需求，针对用户的特定网络环境，进行设计一款 C/S 结构的网络抓包分析软件，加强用户的网络监管，掌握网络的运行状况，更加直观的发现网络问题。同时软件本事便于部署和维护，利于使用。

2. 在基础研究方面，通过对网络流量的研究与分析，可以得知网络数据的具体信息，有利用网络的管理；同时对网络数据的抓取，对数据包的包头进行解析，可以得出数据来源，发现网络的问题。同时系统使用 C/S 结构，客户端与服务端业务可以实现分离，不局限在只对 VxWorks 系统的网络数据的抓取，对于其他类型的业务开发具有良好的扩展性。

因此，网络抓包不管是对网络管理者，或者网络研究者，还是网络用户都是有着非常大的使用价值，有着不错的前景[2]。

1.3 国内外研究现状

网络协议分析技术主要获取网络中传输的原始网络数据包，根据网络协议解析数据包的内容，提取用户感兴趣的部分，然后通过可视化的方式展示给用户，方便用户查看[11]。目前我们所熟知的网络监控框架，提供获取原始网络数据包的接口，同时也提供处理网络包的操作，程序开发人员可以进行二次开发[12]。网络流量捕获工具目前包括基于硬件的监测工具，比如 Navtelinter WATCH 和基于软件的监测工具，比如 WinCap[11,13]。在 Windows 下一般采用 Winpcap 作为软件的底层抓包程序，比如 Wireshark；在 Linux 下一般采用 Libpcap 作为底层抓包程序，比如 Tcpdump；当然也有一些自主开发底层抓包程序，科来网络分析系统就是其中的代表。本课题也采用自主开发底层抓包以及解析程序。目前网络监控对数据进行分析主要通过 TCP/IP 协议抓包存储[14]，包括分布式管理与集中式管理两种方式。分布式管理是在局域网中每个需要监控的终端上安装监控分析软件[1]，向管理中心发送每一个终端

的监控记录，再进一步进行汇总分析，这种方式实现简单，搭建容易，成本低[1]，但存在遗漏数据包等问题；集中式管理是根据兴趣在局域网中设置需要监控的网络节点，针对该节点捕获网络数据包，再进一步汇总分析[1]，这种方式具有较高的实时性与可靠性，但是开发与维护起来较难[1]。

目前国内外网络流量监控方面技术不断推陈出新，有许多网络监控产品，其中包括主流的有有 Ethereal，Sniffer Pro，Open View 等。这些软件大多是可以跨平台的。在 Unix 平台下，提供了基于 Raw socket 抓取网络数据包的函数，也可以使用开源的 Libpcap；而在 Windows 平台下，没有这么方便，主要使用 Winpcap。无论是 Libpcap 还是 Winpcap 都提供了抓包、过滤、统计、和发包四个常用的功能[3]。其中 Windump 就是由 Winpcap 团队直接维护的工具，这是 Unix 平台下 Tcpdump 的 Windows 平台移植版本[3] 。本课题开发主要参考以下：

(1)Wireshark：一款相当流量的网络包分析软件，可以跨平台使用，具有丰富的功能：捕获网络数据包，解析协议，进行统计信息等[15]，并且拥有功能强大的过滤器引擎，研究 Wireshark 对本课题开发具有一定的帮助。

(2)Tcpdump ：运行在类 Unix 平台的命令行工具，可以对网络上的数据包进行捕获分析，底层使用 libpcap 实现[15]。Tcpdump 本身开源，对实现定制网络监测工具具有一定的参考价值。

1.4 本文的组织结构

本文共分为六章，内容安排如下：

第一章 绪论：本章主要介绍了本课题的研究背景，课题的研究目标、内容及意义和国内外的研究现状。

第二章 总体设计与关键技术研究：本章首先介绍了系统的总体设计，之后讨论了实现系统需要实现的关键技术。

第三章 网络包捕获软件的设计与实现：本章首先讨论了捕获软件的总体架构设计，然后针对捕获软件的各个功能模块的设计与实现进行阐述。

第四章 网络包分析软件的设计与实现：本章首先讨论了网络包分析软件的总体架构设计，然后针对分析软件的各个功能模块的设计与实现进行阐述。

第五章 系统测试：本章主要从功能测是与性能测试两个方面对系统进行测试。

第六章 总结：本章总结本文的工作，并针对目前的课题工作情况提出存在的问题和不足以及待改进的地方。

第二章 总体设计与关键技术研究

2.1 总体设计

本论文根据用户的需求设计出基于 VxWorks 系统的网络监控系统的整体架构如图 2.1 所示。系统采用类似 C/S 的结构，包括运行在终端的网络包捕获软件和运行在主机端的网络包分析软件两个部分。软件运行的环境主要使用串口进行通信，没有多余的网络通信设备使用，因此开发过程中主机与终端之间采用串口通信，捕获软件运行在 VxWorks 实时系统上，采用 C 语言开发，使用道系统软件开发平台，分析软件是使用 Qt 框架开发的图形界面交互软件，可以运行在 Windows 和 Linux 平台上。

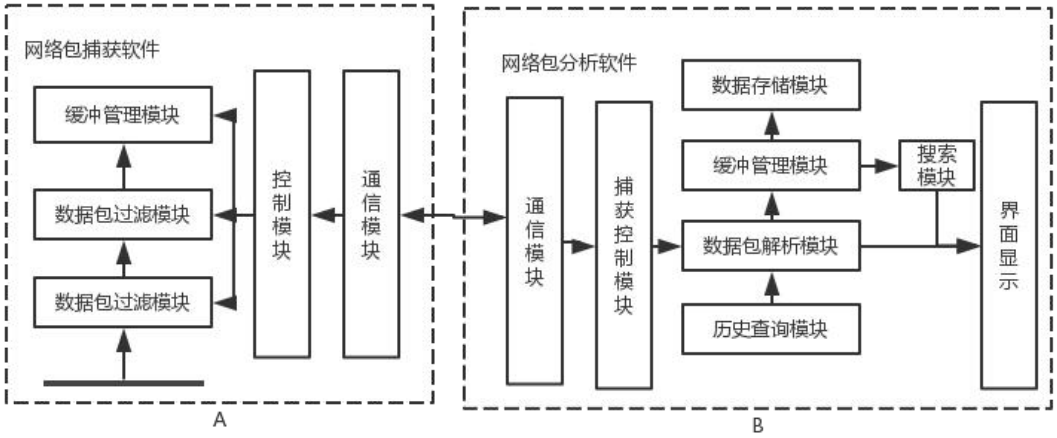


图 2.1 软件的总体架构

1. 网络包捕获软件

网络包捕获软件运行在终端，主要功能包括网络数据包的捕获功能，过滤功能和传输功能。软件的总体结构分为数据包捕获模块、过滤模块、缓存管理模块、控制模块以及通信模块。软件的总体功能模块如图 2.1.A。

网络包捕获模块负责捕获 IP 层之上包括 IP 层的网络数据；过滤模块负责将捕

获的网络数据包以多种方式过滤并保存到缓存中；缓冲管理模块负责转存捕获的网路数据包；通信模块负责从缓冲中取出捕获数据并上传，以及接受并响应主机的各种命令；控制模块负责其他模块的初始化和启动停止。

2. 网络包分析软件

网络包分析软件运行 Windows 或者 Linux 主机上，负责分析网络数据包并展示结果，总体结构如图 2.1.B 所示，主要包括通信模块、捕获控制模块、数据包解析模块、历史查询模块、数据存储模块、搜索模块、缓冲管理模块和界面显示模块。

软件包含实时监控、历史查询以及数据存储三种工作模式。

(1)用户在实时监控模式下使用捕获控制模块控制终端捕获软件的启动，暂停和结束，同时可以设定过滤条件，实时接收分析网络数据包。

(2)用户在历史查询模式下，使用历史查询模块读取保存在本地的网络数据包文件，查询历史流量信息。

(3)用户在数据存储工作模式下，使用数据存储模块读取缓冲区保存历史网络数据包到本地磁盘。

实时监控模式与历史查询模式最终会将解析的结果通过界面显示模块显示，解析结果是根据协议分层模型和数据封装规则，从链路层这层开始，自下向上逐层进行。

2.2 关键技术研究

2.2.1 网络包捕获

想要实现捕获网络包，首先需要了解捕获网络包过程中发生了什么。常用的以太网卡主要支持广播模式、多播模式、直接模式和混杂模式四中工作模式[16]。一般情况下一台网设置成广播模式，意味着网络接口在正常情况下只接受这样的数据包：①MAC 地址和自己相互匹配的数据包②广播包和属于自己的组播包[17]。对于混杂模式，网卡将接收所有经过的数据包，这种数据包是未处理的二进制数据。网卡

收到传输来的数据首先接受数据头的目的 MAC 地址，根据设置的接收模式决定对数据包的处理方式，认为该接收时网卡通过 CPU 产生一个硬件中断，并将帧中所包含的数据传给系统进一步处理；认为不该接收数据就会被网卡截断丢弃[17]。

经过上面分析，想要对网络数据进行捕获，网卡应该先设置为混杂模式[17]，接收网络数据包，最后将获取的数据进行协议解析，根据需求对每层协议分析，提取每层协议的具体信息，并可视化展示。

2.2.1.1 网络数据包捕获方式

1.原始套接字

通常的网络数据抓包软件将网络设备设为混杂模式，从数据链路层捕获数据，也就是说在数据链路层获取 Raw Socket 进行分析。在 Linux 可以使用原始套接字访问数据链路层，原始套接字不同于 TCP/UDP 类型的套接字[18,19,20]，前者可以接收来自链路层的网络数据包，后者只能接收传输层及以上的数据。同时也可以使用 libpcap，一种提供分组捕获机制的分组捕获函数库，提供了捕获、过滤、分析和存储网络数据包的功能[18]。

在道系统中不支持 libpcap，但可以使用使用原始套接字捕获网络包。在道系统下从数据链路接收所有帧可以如下创建套接字：

```
fd = socket(PF_PACKET, type, protocol)
```

- type 字段可选 SOCK_RAW 或者 SOCK_DGRAM,这两个使用 struct sockaddr_ll{} 结构[21]。使用 SOCK_RAW 套接字可以接收包含 MAC 帧的网络数据包，使用 SOCK_DGRAM 套接字系统会自动去掉网络数据包的 MAC 帧。对于本系统的需求，选择使用 SOCK_RAW 套接字。
- protocol 字段可选 ETH_P_IP，ETH_P_ARP，ETH_P_RARP 或 ETH_P_ALL[21]，四个可选选项的含义如表 2.1。

2．MUX 接口

在 VxWorks 中，网络协议栈是一个可裁减增强网络协议栈 （SENS, Scalable Enhanced Network Stack）[22-25]，由数据链路层、MUX 接口层、网络层、传输层和

应用层组成，MUX 接口层是在传统的 TCP/IP 网络协议栈新增的特性,不是新的协议层[25],可以通过图 2.2 了解协议栈的层次。MUX 接口层可以隐藏数据链路层与网络协议层的通信细节，向网络层协议提供一致的服务使发送、接收数据变得简单。MUX 协议层关系如图 2.3 所示，图中表明网络层协议和数据链路层之间传递数据都要通过 MUX 接口层。数据捕获就是利用 MUX 接口层提供的接口实现。

表 2.1 protocol 字段含义

Protocol	Value	含义
ETH_P_IP	0x0800	接收发往目的 MAC 是本机的 IP 数据包[21]
ETH_P_ARP	0x0806	接收发往目的 MAC 是本机的 ARP 数据包[21]
ETH_P_RARP	0x8035	接收发往目的 MAC 是本机的 RARP 数据包[21]
ETH_P_ALL	0x0003	接收所有发往目的 MAC 是本机或从本机发出去的所有类型的网路包。打开混杂模式，会接收发往目的 MAC 不是本地的网络包[21]。

网络协议层或者服务程序可以使用 muxBind()或者 muxTkBind()函数绑定 END 驱动程序，将网络数据包向下传送到对应的网络接口驱动；同样，网络接口驱动也使用 MUX 层将数据包向上传送到网络协议层，完成彼此之间的通信[26,27]。最后可以调用 muxUnbind()解除 MUX 接口与网络驱动程序的绑定。捕获程序与网络接口的调用关系图如图 2.4。

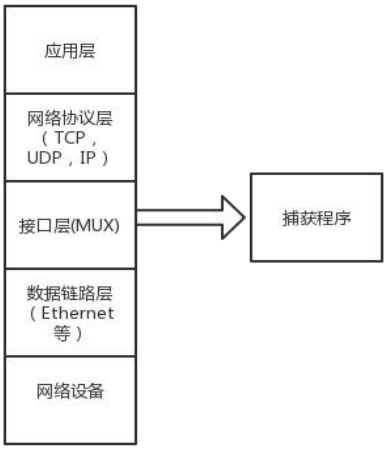


图 2.2 协议栈层次

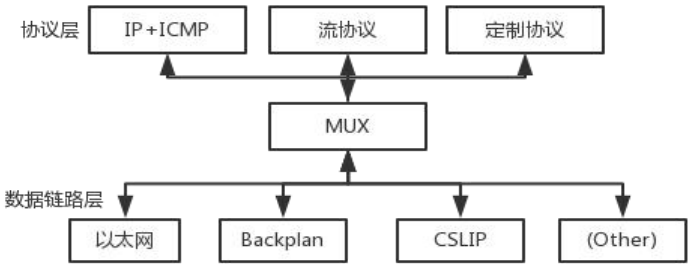


图 2.3 MUX 协议层关系图

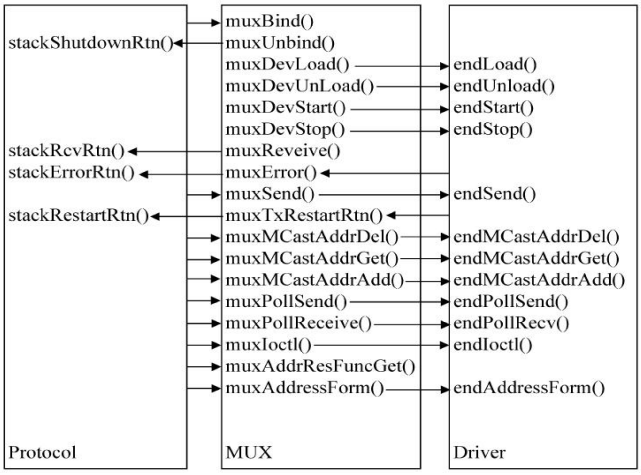


图 2.4 MUX 接口函数

在使用 muxBind 函数时，网络协议或服务程序提供其函数指针给 MUX 接口，需要我们实现下面的接口功能函数，如果不关注某一项，可以传入空指针。

① stackShutdownRtn()：关闭服务程序

② stackError()：传递一条错误信息给服务程序

③ stackRcvRtn()：传送一个数据报信息给服务程序

④ stackTxRestartRtn()：重启服务程序

MUX 使用 stackRcvRtn()回调函数将网络接口驱动接收到的网络数据包转发到协议层，其中接收到网络数据包将被存储到 M_BLK_ID 结构指针中，可以通过 pNetBuf->mBlkHdr.mData 访问，pNetBuf 为 M_BLK_ID 类型。如果返回 TRUE 时网络协议就接收了该帧[15]。

在使用 muxBind 时候，我们需要指定相关协议，协议类型和含义如下：

- MUX_PROTO_SNARF: 在所有标准协议接收之前调用 stackRcvRtn()[28]

- MUX_PROTO_PROMISC: 在所有标准协议接收之后调用 stackRcvRtn()[28]
- MUX_PROTO_OUTPUT: 在数据被送到物理层(DRIVER)之前调stackRcvRtn()[28]

当需要同时捕获所有发往目的MAC是本机或从本机发出去的所有类型的网路包时需要分别使用MUX_PROTO_SNARF和MUX_PROTO_OUTPUT绑定两次。

3. 钩子函数

VxWorks为我们提供两类钩子函数，让在VxWorks网络协议栈处理数据包之前截获数据，包括截获以太帧的EtherHook和截获IP数据包的IpFilerHook[29]。

EtherHook包括两个钩子函数：一个为EtherInputHook，用来捕获系统接收到的数据包；一个为EtherOutputHook，用来捕获系统发送的以太帧[29]。

EtherInputHookAdd将调用muxBind添加一个MUX_PROTO_SNARF协议，从而截获进去MUX接口层的所有数据包,同理EtherInputHookAdd将调用muxBind添加一个MUX_PROTO_OUTPUT协议[29],从而截获从MUX接口层发出的所有数据包；IpFilterHook使用IpFilterHookAdd钩子函数安装，不属于MUX接口层，只用来截获IP数据包，不会接收非IP数据包。

2.2.1.2 本系统采用方式

对比前面三种捕获网络数据包的方式,选择合适的开发方式。

(1)在VxWorks下使用原始套接字的方式，一方面会接收到所有的网络数据包产生多余的数据，而我们只想接收MAC地址是本机的数据包，另一方面这种方式实现使用发现不能捕获本机发送出去的网络数据包，不符合需求。

(2)使用MUX接口实现的捕获程序，通过使用MUX_PROTO_SNARF和MUX_PROTO_OUTPUT绑定两次实现接收所有发往目的MAC是本机或从本机发出去的所有类型的网络数据包。此种方式兼容性与可移植性强。

(3)使用钩子函数捕获数据，由于使用的软件开发平台道系统兼容的VxWorks版本不支持，不再考虑此种方式实现捕获网络数据包。

结合上述讨论，为了保证系统的稳健性以及系统在VxWorks版本的兼容性，制

定使用 MUX 接口实现捕获网络包的功能的方案。

2.2.2 网络包过滤

数据包过滤（packet filtering）技术是防火墙最常用的技术，通过在系统内设置过滤规则（通常称为访问控制表，Access Control List），保证让数据包有选择的通过[30,31]。数据包过滤对所有通过它进出的数据包进行检查，通过检查数据包报头， 与设置的过滤规则进行比较，对不符合过滤规则的数据包删除，符合过滤规则的数据包转存到缓冲区中[31-34]。

常见的数据包过滤有 BPF（Berkeley Packet Filter，伯克利数据包过滤器），是 Linux 是访问数据链路层的一种接口，每个数据链路驱动程序都在发送一个分组之前或者在接收一个分组之后调用 BPF。BPF 主要由两部分组成，网络转发和数据包过滤。BPF 过滤在内核中进行，并提供缓冲，将数据复制量减少到最小[30]；对于每一个符合过滤原则的数据包，BPF 将它们拷贝到与之相连的缓存中。因此 BPF 效率是很高的[30]。

由于系统的终端程序使用道系统开发，不支持 BPF 过滤，所以需要根据过滤原理以及需求设计一个简单的过滤模型。

2.2.2.1 过滤原理

数据包过滤包括无状态数据包过滤和有状态数据包过滤，不同在于后者有一个状态表，保存当前的连接列表[30]。课题设计的系统部关心连接的状态，实现简化版的无状态过滤数据包模型。

包过滤一般只检查报头信息而不验证数据包的数据部分[30]，会根据以下方式进行过滤：

（1）按照 IP 源地址和目标地址过滤：可以分析 IP 数据包报头，根据过滤规则，获取符合规则的 IP 源地址和目标地址[33]。

（2）按照数据包所属协议类型过滤：分析以太网帧，从网络协议栈中依次分析

数据包报头，过滤目标协议的网络数据包。

（3）按照 TCP 或 UDP 端口号过滤[30]：根据此过滤方式，可以得到 TCP 的源端口或目标端口和 UDP 的源端口或目标端口。这种端口过滤同样可以过滤应用层的服务信息，比如 TCP 端口号为 25 的 SMTP 邮件服务， TCP 端口号为 80 的 HTTP 服务，UDP 端口为 69 的 TFTP 服务等。

（4）按照 IP 与 TCP 或 UDP 端口结合：此过滤方式目的是过滤指定 IP 地址的服务信息。

还有许多其他的过滤方式，比如 ICMP 消息类型，TCP 报头中的 ACK 位等，根据需求，本课题设计的过滤器主要实现上面四种方式。

2.2.2.2 过滤器设计

这一小节，我们根据过滤器原理设计一个简化版过滤器。系统过滤器由两个部分组成：单个条件过滤模式和组合条件过滤模式。单个条件过滤模式主要负责按照数据包所属协议类型过滤，根据协议栈每层分析报头与过滤规则进行比较。组合条件过滤模式负责其余的三种过滤方式。系统会根据用户的输入以及过滤命令来设置当前所属的过滤模式，当符合过滤规则则将数据包转存到缓冲区中。图 2.5 给出了基于本系统设计的过滤器的捕获系统结构图。

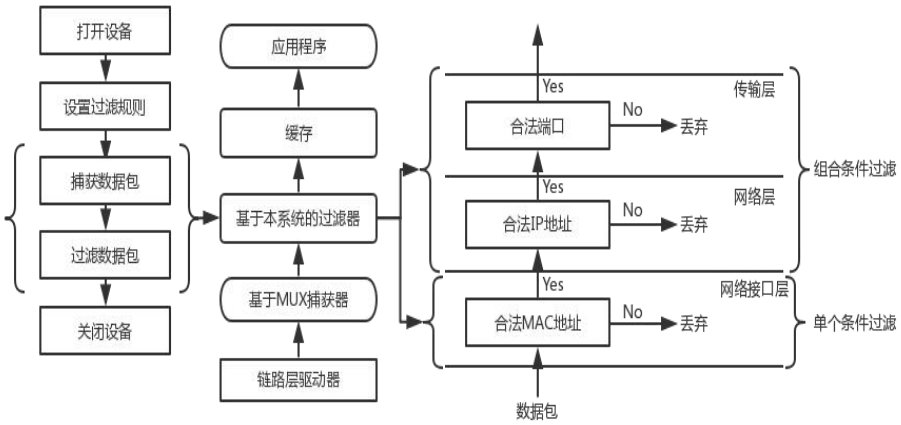


图 2.5 捕获系统结构

2.2.3 网络包读取与保存

系统本身包括两个部分，一个是实时捕获分析网络数据包，网络包捕获软件从终端捕获数据包并推送到网络包分析软件进行分析；另一个就是历史查询网络数据包，即读取本地保存的网络数据包文件分析。本系统采用 pcap 文件格式捕获的原始网络数据包存储到本地，想要从中分析网络数据包，提取以太帧信息，就需要了解 pcap 文件格式。pcap 是一种数据流格式，在 Linux 里，pcap 是一种通用的数据流格式，很多开源项目都需要用到这种格式的文件。下面以我们将详细介绍 pcap 文件格式。

一个 pcap 文件封装如表 2.2，pcap 文件头处于文件最开始处，大小固定为 24 字节，用来标识文件的属性[35]；数据报头紧随其后，大小为 16 字节，再接上数据报；然后继续数据报头和数据报，依次封装下去。

表 2.2 封装 pcap 文件

文件头	24 字节
数据报头 + 数据报	数据包头为 16 字节，紧跟着数据包[35]
数据报头 + 数据报	数据包头为 16 字节，紧跟着数据包[35]
.....	
数据报头 + 数据报	数据包头为 16 字节，紧跟着数据包[35]

可以发现pcap文件格式与网络数据包的封装类似,因此我们保存数据报为pcap文件时，首先在文件开头构造文件头，然后对每一个数据报添加一个数据报头进行封装保存；在解析 pcap 文件分析网络数据报时，首先需要根据文件头判断文件是否为 pcap 文件，然后去除文件头得到数据报头和数据包的列表，然后去除数据报头得到需要解析的数据包。如图 2.6 给出了 pcap 文件头格式与数据报格式。

1. pcap 文件头格式，每一个属性的含义如表 2.3。

2. pcap 数据报头格式，每一个属性的含义如表 2.4。

3. packet 数据报内容，即我们保存在 pcap 文件中原始网络数据包，长度大小为 caplen，与数据报头组成一个单元存储在 pcap 文件中，依次排序。因此我们需

要通过上一个 packet 包确定下一个 packet 包的位置信息。访问到 packet 包信息，就是我们熟悉的网络结构了。

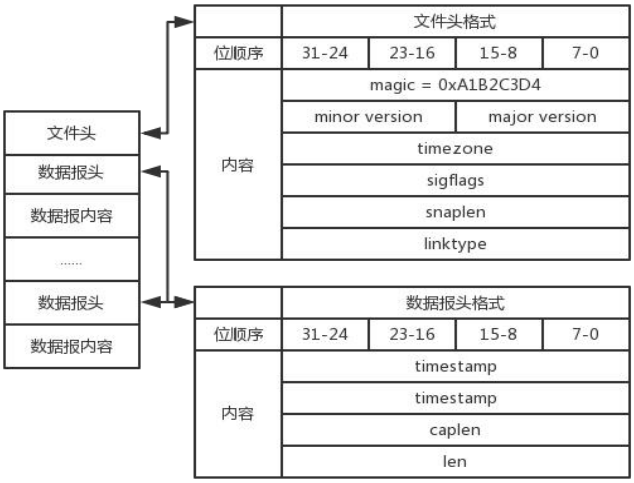


图 2.6 pcap 文件头和数据报格式

表 2.3 文件头各字段含义

字段名	含义
magic(4 字节)	识别文件和字节顺序[35]，我们固定为 0xA1B2C3D4
magor(2 字节)	文件主要版本号[35]
minor(2 字节)	文件次要版本号[35]
timezone(4 字节)	GMT 和本地时间的相差，用秒来表示，一般也设置为 0[35]
sigflags(4 字节)	时间戳的精度，一般为 0[35]
snaplen(4 字节)	捕获的网络数据包的最大存储长度，将该值最大设置为 65535，则捕获所有的网络数据包[35]
linktype(4 字节)	链路类型

2.2.4 环形缓冲区的设计

在抓包时要保障抓包的数量的正确，是保证整个系统质量的重要指标，意味着避免丢包现象的出现，这就很大程度依赖于抓包与保存包的速度[2]。因为系统采用

的 C/S 工作模式，同时网络包捕获软件使用的 VxWorks，采用多任务调度和缓冲技术的结合，网络包分析软件采用多线程和缓冲技术结合。下面具体分析整个过程，整个过程如图 2.7。

表 2.4 数据报各字段含义

字段名	含义
timestamp(4 字节)	时间戳高位，精确到秒
timestamp(4 字节)	时间戳低位，精确到微妙
caplen(4 字节)	数据包长度,标识所抓获的数据包保存在 pcap 文件中的实际长度，根据此字段计算得到下一个数据包位置[35]
len(4 字节)	数据包实际长度，如果文件中保存不完整的数据包，这个值会比 caplen 大，一般两个值相等[35]

(1)网络包捕获软件：创建两个任务，一个任务负责抓取数据包并保存数据报，另一个任务负责向网络包分析软件推送数据包；

(2)网络包捕获软件：抓包任务在抓取到数据包后，将数据包放到缓冲区 A；推送任务从缓冲区 A 读取数据包，发送到网络包分析软件；

(3)网络包分析软件：创建两个线程，一个线程负责从网络包捕获软件接收数据包，另一个线程负责分析数据包渲染页面。

(4)网络包分析软件：接收线程从网络包捕获软件接收到数据包后，将数据包放到缓冲区 B；解析线程从缓冲区 B 中读取数据包进行解析渲染页面。

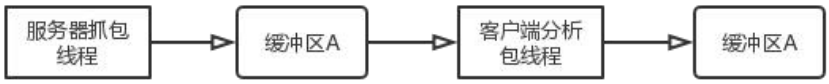


图 2-7 线程与缓冲区结合过程

可以发现整个过程的关键在当收到大量的数据包时如何合理存储，也就是说需要设计一个合理的缓冲区来保存到来的大量网络数据包，一则如果太大会占用大量系统资源，造成浪费，二则如果太小会限制存储数据包的能力，造成数据包的丢失。

本文设计了一个静态的环形缓冲区来解决这些问题，减少数据包丢失情况的发生。图 2.8 是本文采用的环形缓冲区结构，是使用循环队列实现的，特点是连续、定长。在初始化时，将环形缓冲区清空，队头位置和队尾位置设为 0,我们通过操作队头位置与队尾位置标识填充数据、移除数据，而不需要移动其存储位置，是一种先进先出缓冲区[36]。当抓包线程捕获到一个数据包时，将此包保存到当前位置 tail++，当下一个位置与 head 相等时，表示缓冲区已经满，当 head 和 tail 相等标识缓冲区空，tail+1 和 head 不等标识存在新数据包，向客户端推送数据。对于更详细的设计思想与实现将在第三章的环形缓冲区介绍。

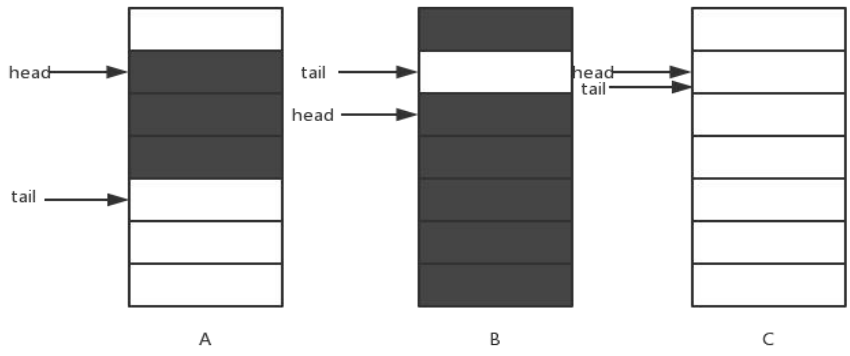


图 2.8 环形缓冲区结构

2.3 本章小结

本章重点介绍了系统的总体设计架构，并介绍了所用的道系统开发平台，最后研究了设计系统使用的关键技术，主要包括捕获网络数据包，过滤数据包、读取与保存文件、环形缓冲区的设计等。下面将讨论网络包捕获软件和网络包分析软件的详细设计和实现。

第三章 网络包捕获软件的设计与实现

3.1 总体设计

网络包捕获软件主要负责与主机端网络包分析软件交互，接受并响应来自网络包分析软件的命令信息；从网络接口抓取数据包，并实时将捕获数据推送到分析软件，有解析数据包和保存数据包等辅助功能。捕获软件的总体结构如图 3.1。

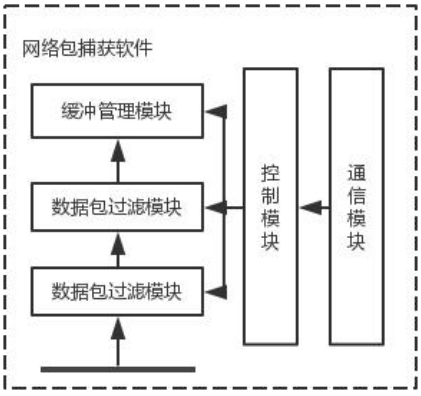


图 3.1 捕获软件总体设计

从架构图中可以看出整个捕获软件系统包括五个模块：(1)数据包捕获模块；(2)数据包过滤模块；(3)缓冲管理模块；(4)控制模块；(5)通信模块。接下来章节说明各个功能模块的设计与实现。

网络包捕获软件运行环境为 VxWorks 5.5， 使用 C 语言开发，软件开发平台为道系统。

3.2 功能模块设计与实现

3.2.1 控制模块

控制模块是整个软件的入口，主要负责捕获软件的初始化，以及其他模块的初

始化和启动停止。

1. 初始化

数据包捕获软件系统是基于VxWorks实现的,需要涉及到多个任务和串口通信，因此在启动系统时，需要初始化必要的信息。系统读取配置文件，创建命令控制任务开始等待读取来自监控客户端的命令以及用于通信的串口描述符。控制模块实现整个初始化过程。

(1) 初始化软件相关的结构体：首先使用 InitCmdList()函数初始化用于存储读取命令的双向链表cmd_list,保证命令接收正确;接下来使用InitCapInstance()函数初始化应用程序使用的全局变量结构体，全局变量结构体保存着系统业务逻辑需要操作的变量，结构如表3.1；最后初始化二进制信号量，用于共享变量的互斥同步。

(2) 初始化缓冲管理模块：使用 InitCapQueue()函数初始化缓冲循环链表，自定义的环形缓冲区，用于存储捕获的网络数据包，并推送到监控客户端，降低丢包的概率，这是缓冲管理模块需要的结构体。

(2) 初始化主机通信模块：通过创建串口通信描述符，打开并配置串口，用于监控终端捕获软件与主机分析软件的通信，终端接收来自主机的命令信息，响应并推送信息。

(3) 创建命令操作任务：在初始化相关的结构体以后，创建系统的命令操作任务，接收来自监控客户端的命令，作出对应操作。命令操作任务只是系统的其中一个任务，在接收到抓包命令时候，会在定义的抓包函数中创建抓包任务和推送数据包两个任务。

(4) 初始化捕获模块与过滤模块：这两个模块需要在接收捕获与过滤的命令时启动，开始设定为未开启模式。

2. 控制启动停止

网络包捕获软件是多任务模式，控制着任务的创建与销毁。主要涉及到下面几种任务：

(1)接收分析命令任务：当捕获软件初始化完成后，创建该任务等待接收来自

分析软件发送的命令，根据命令类型执行相应的操作。

(2)捕获网络包任务：当捕获软件接收到开始捕获命令，创建该任务开始从指定的网络接口捕获网络包并转存到缓冲区。

(3)推送网络包任务：当开始捕获网络包后，创建该任务访问缓冲区将缓冲区数据推送到分析软件。

表 3.1 全局变量结构

```
/**
 * @brief pCapInstance, 抓包全局变量结构体
 */
typedef struct {
    PROTO_COOKIE pCookie;          /**< Mux 接口句柄 */
    int quit_send_loop;            /**< 控制循环发送数据包 */
    int quit_cmd_loop;             /**< 控制循环接收处理命令 */
    SEM_ID cap_locksem_id;         /**< 信号量，用于互斥加锁解锁 */

    int connfd;                    /**< 文件描述符，描述串口或者网络 */
    int cmd_proc_tid;              /**< 启动的处理命令任务 ID */
    int fetch_proc_tid;            /**< 启动的抓包任务 ID */
    int cap_state;                 /**< 抓包所处状态 */
    int ifunit;                    /**< 目标机网络设备接口号 */
    char ifname[16];               /**< 目标机网络设备接口名称 */

    enum Cap_Type cap_type;        /**< 抓包模式，串口或者网络 */
    enum ReadyToWrite rtWrite;     /**< 是否进行写文件 */

    enum FilterMode filterMode;    /**< 过滤模式 */
    PacketFilter packetFilter;     /**< 过滤控制结构体 */
} pCapInstance;
```

当终端关闭捕获数据包软件系统时，在用户输入关闭系统命令或者接收来自分析软件的关闭命令后，将调用停止抓包函数，清理后续工作并关闭系统。停止抓包函数主要负责以下几个方面工作：(1)如果捕获网络数据包和推送网络数据包任务已经启动，调用 stopMuxPacketTask()关闭这两个任务，这也是停止捕获模块的方式；(2)修改读取命令与推送网络数据包循环条件，跳出循环；(3)清空存储命令的命令链表，释放内存；(4)删除信号量；(5)关闭串口；(6)删除接收分析命令任务。

3.2.2 通信模块

通信模块是实现与分析软件交互的关键部分，使用串口实现捕获软件与分析软件的通信，主要负责接收从分析软件发来的命令，并根据命令的类型执行操作。命令类型主要包括一是获取网络接口列表，指定网络接口；二是控制抓包流程：开始、停止、结束捕获网络数据包、关闭捕获软件；三是过滤网络数据包。如图 3.2 是命令在过程中的状态转换图。

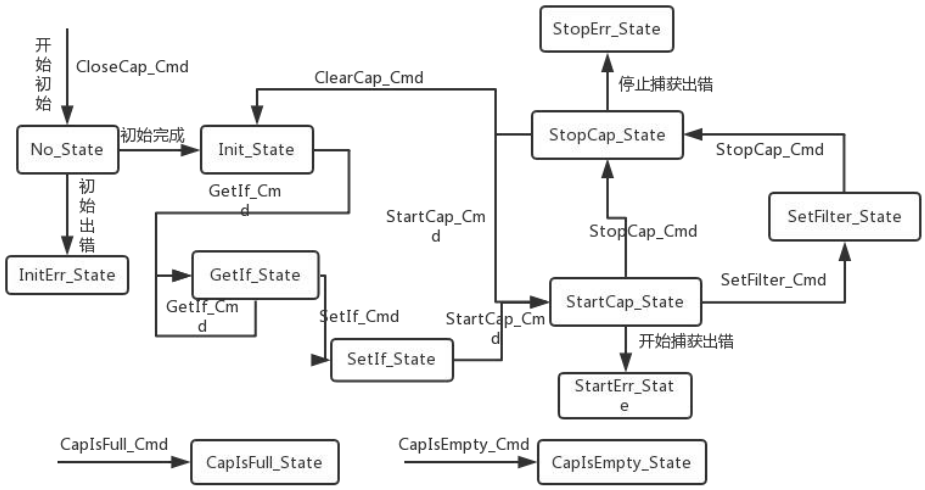


图 3.2 命令状态机

整个过程的状态转换：

(1)网络包捕获软件系统启动后配置相关信息，将状态设置为 No_State，等待初始化其他模块和创建接收分析命令任务初始化失败后状态设置为 InitErr_State，打印错误信息，等待排查；初始化完成后，将状态设置为 Init_State，等待接收来自分析软件的命令。

(2)当接收到获取网络接口列表命令 GetIf_Cmd 后，捕获软件访问网络接口列表，推送到分析软件，将状态修改未 GetIf_State；接下来分析软件发来 SetIf_Cmd 命令，捕获软件指定需要捕获的网络接口，状态切换至 SetIf_State。

(3)分析软件启动捕获，捕获软件接收到开始捕获命令 StartCap_Cmd，创建捕获数据包任务和推送数据包任务。开始捕获后，捕获模块将不断捕获指定接口的网

络数据包并转存到环形缓冲区中，当缓冲区满时，捕获软件向自身发送 CapIsFull_Cmd 命令，将状态切换至 CapIsFull_State，停止捕获，打印信息，等待管理员后续操作；推送任务循环遍历环形缓冲区将数据包推送到分析软件，直至为空。当创建捕获任务或推送任务成功状态切换至 StartCap_State 等待后续操作；出错，状态切换至 StartErr_State，打印错误信息，等待排查。

(4)分析软件设置过滤条件，捕获软件接收设置过滤条件命令 SetFilter_Cmd，等待后续操作。

(5)分析软件停止捕获，捕获软件接收到停止捕获命令 StopCap_Cmd，停止捕获与推送任务，成功切换至 StopCap_State；失败切换至 StopErr_State，打印错误信息，等待排查。

(6)分析软件重置捕获，捕获软件接收到重置捕获命令 ClearCap_Cmd，重新初始化捕获软件，将状态切换至 Init_State。

(7)分析软件关闭，捕获软件接收到关闭捕获命令 CloseCap_Cmd，捕获软件删除所有任务，清空缓冲区，初始化，将状态切换至 No_State。

这个流程是整个系统捕获软件与分析软件的交互过程，整个流程如图 3.3。

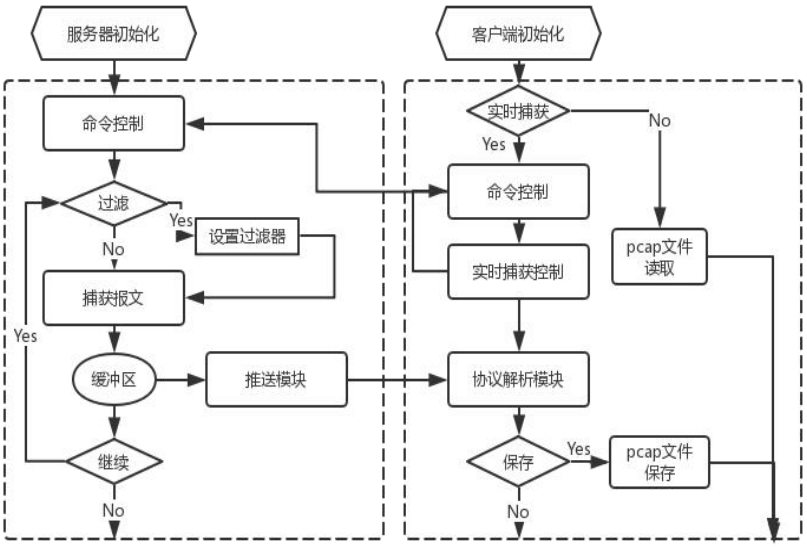


图 3.3 服务器与客户端交互

通信模块的实现关键在于两个方面：(1)串口通信，实现网络包捕获软件与网络包分析软件的信息传输；(2)命令处理，规定命令的格式，命令类型，接受并响应主机发来的命令。

3.2.2.1 串口通信

串口通信模块是网络包捕获软件命令控制与推送信息的基础，即通信模块的基础，需求简单，直接对串口操作业务逻辑进行了简单的封装，满足系统的使用。

串口通信主要接口包括打开串口，发送信息、接收信息、关闭串口，定义如表3.2。接口定义简单，在这里主要说下 SerialPortOpen(char *) 函数，函数主要包括两个步骤：①以读写方式打开串口描述符；②设置串口，使用 ioctl()函数分别设置数据位，停止位，校验位，波特率，清空缓冲区等。

表 3.2 串口通信主要接口定义

```
int SerialPortOpen(char *portName);
int SerialPortSend(int fd, char *buff, int len);
int SerialPortReceive(int fd, char *buff, int len);
int SerialPortClose(int fd);
```

串口通信不同于网络通信，如果不对数据进行特殊处理，存在发送接收不完整数据包的情况。串口通信的发送方每隔一定时间将有效信息通过信道逐个二进制位进行发送，接收相似。因此在接收数据的时候，可能接收的只是某个时间段的数据，而不是我们需要的整个数据包。针对串口通信的特性，下面设计一种保证接收方接收到完整数据包的方案。

如图 3.4 我们自定义了一种数据格式，即在数据的两端添加一个帧头和一个帧尾，帧头和帧尾是不在原始数据出现的数据，这是本文系统捕获软件与分析软件串口通信使用的方式。(1)发送方发送数据时，构造该数据格式的数据流发送；(2)接收方接收数据时，判断接收的数据流中是否存在帧头与帧尾来决定是否接收到一个完整的数据流。使用数据缓冲区累计收到的字符，当缓冲中第一次出现帧头和帧尾，表示出现一次完整数据包，读取出来，继续操作。

图 3.4 封装串口数据

表 3.3 帧头帧尾宏定义

```
#define CONTENT_END     "\04\04"        /**< 串口传输信息尾部标志 */
#define CONTENT_START   "\03\03"        /**< 串口传输信息开始标志 */
```

根据捕获软件要传送的数据是网络数据，可以将帧头和帧尾设置为两个连续的不可打印的字符，避免了在传输数据域中出现，如表 3.3 所示是帧头和帧尾的宏定义。根据上述思想提供了两个接口：①发送网络接口；②发送网络数据包接口。

发送网络接口列表接口用于发送网络接口，因为网络接口名称长度一般不会太长，以及 IP 地址长度固定，这里分配固定长度的内存存储用帧头帧尾封装的网络接口信息。如表 3.4 给出了字节分配方案，按照分配先后给出。命令响应类型 Cap_Cmd_Res 包括 GetIf_Content 和 GetIf_End 两个字段，GetIf_Content 用于标记目前发送的信息为网络接口信息，GetIf_End 用来标记目前发送的信息是获取接口列表时出错信息以及结束信息。分析软件接收到该信息后首先根据命令响应类型判断是不是需要填充窗口。

表 3.4 发送网络接口列表接口字节分配方案

名称	字节大小
帧头	4
命令响应类型	4
数据长度	4
网络接口名称	28
IP 地址	4
帧尾	4

发送网络数据包接口用于发送网络数据包，因为网络数据包本身是变长的，需要根据网络数据包的长度动态分配内存，再在两头添加帧头与帧尾封装数据，如表 3.5 给出了字节分配方案。这里不同于发送网络接口列表接口，没有使用命令响应

类型，原因在于网络数据包是从缓冲区取得，在此之前已经保证了网络数据包满足要求，不再需要进行标识。分析软件接收信息，获取网络包进行分析。

表 3.5 发送网络数据包接口接口字节分配方案

名称	字节大小
帧头	4
网络包长度	4
网络包	根据网络包长度分配内存
帧尾	4

3.2.2.2 命令处理

命令处理主要包括接收命令和应答命令两个功能。下面详细介绍两个功能的设计与实现。

1. 接收命令

接收命令负责从分析软件接收命令并分析命令，为了保证接收命令的准确性，使用携带命令信息的链表 Cap_Cmd_Info，结构体如表 3.6。

表 3.6 Cap_Cmd_Info 结构体

```
/**
 * @brief Cap_Cmd_Info, 命令携带信息链表
 */
typedef struct _Cap_Cmd_Info
{
    NODE * next;
    NODE * prev;
    int cmd;                        /**< 命令类别 */
    int info;                       /**< 命令信息 */
    char buf[CMD_BUF_LEN];          /**< 命令信息 */
} Cap_Cmd_Info;
```

Cap_Cmd_Info 结构体中包含着 cmd，info 和 buf 三个字段，其中 cmd 作用标记命令类别，以此响应操作，比如开始捕获、停止捕获、清除捕获、关闭等；其他两个字段用来携带命令的相关信息，如出错信息。命令类别使用自定义的 enum 类

型 Cap_Cmd，定义如表 3.7。

表 3.7 Cap_Cmd 定义

```
/**
 * @enum Cap_Cmd
 * enum 抓包命令类型
 */
enum Cap_Cmd
{
    GetIf_Cmd=0,          /**< 获取网络接口命令 */
    SetIf_Cmd,            /**< 指定网络接口命令 */
    SetFilter_Cmd,        /**< 设置过滤器命令 */
    StartCap_Cmd,         /**< 开始捕获命令 */
    StopCap_Cmd,          /**< 停止捕获命令 */
    ClearCap_Cmd,         /**< 重置捕获命令 */
    CloseCap_Cmd          /**< 关闭捕获命令 */
};
```

因为捕获软件与分析软件之间使用串口通信，使用的二进制数据流，需要定义数据流与命令链表结构体之间的转换，则此模块主要包括两个方面， 这两个方面所对应的操作接口声明在表 3.8 给出。

表 3.8 存取命令接口

```
/** 从接收的数据中分析命令*/
void parse_cmd_buf(char *buf, int len, LIST *cmd_list, SEM_ID key);
/** 从命令列表读取命令*/
Cap_Cmd_Info * read_cmd_fromlist(LIST *cmd_list, SEM_ID key);
```

(1)捕获软件循环等待读取来自分析软件的命令请求，当接收到一个命令的二进制数据流，便通过函数 parse_cmd_buf 分析转化成命令链表结构体存放到命令链表中。

(2)捕获软件通过使用函数 read_cmd_fromlist 从链表中读取命令，合法时根据命令类型响应操作。

上述操作接口易实现，不过要注意两个问题：(1)用来存储命令结构体的链表是使用 VxWorks 的基本数据结构中的双向链表实现，使用 lstAdd 和 lstGet 操作，避免了自定义复杂的链表操作，同时在清空命令链表时候要安全释放内存。(2)捕获软件使用多任务工作模式，因此要保证整个存取命令过程在多个任务之间的安

全，也就是说要实现整个过程的互斥同步，多任务之间的互斥同步本质上就是我们在多线程熟悉的“读写锁”问题。我们使用在缓冲管理模块章节提到的互斥机制。(3)在整个读取处理过程，需要考虑到使用串口与分析软件通信的时候，如果直接使用 read 读串口会造成阻塞或者数据读取错误，使得读取的命令出错。可以使用 select 先查询串口，再去 read 读，这样可以避免阻塞，保证读取的命令的正确性，同时在串口延时时，程序会退出，不会造成程序的崩溃。(4)串口需要发送命令结果与发送网络数据包，为了保证两种类别的数据正确发送，严格保证命令状态的切换顺序，即发送命令结果需要在接收到获取网络接口命令后，发送网络数据包需要在接收到开始捕获命令后。

2. 应答命令

应答命令的职责是捕获软件在接收分析完命令后，从命令链表中取出命令，并根据命令类别作出相应操作。如上一节提到的，应答命令需要对获取网络接口列表、指定网络接口、控制捕获网络数据包、过滤数据包等命令的处理。

(1)获取网络接口列表并指定网络接口

服务器在接收到来自分析软件的获取网络接口列表命令时，将调用获取网络接口函数获取终端所有的网络接口列表，并将列表信息推送至分析软件。然后用户指定一个网络接口，分析软件发送指定网络接口命令，捕获软缉拿接收该命令指定对应的网络接口，创建捕获网路数据包任务。ioctl 使用宏 SIOCGIFBRDADDR 作为参数可以获得网络接口名称，如下：ifr 是 ifreq 结构体，包含了需要的网络接口的名称、IP 地址字段。

```
(sockfd = socket(AF_INET, SOCK_DGRAM, 0))
ioctl(sockfd, SIOCGIFADDR, &ifr)
```

(2)应答开始捕获网络数据包

捕获软件在接收到开始捕获网络数据包命令后，将创建系统所需要的最后两个任务：①捕获网络数据包；②推送网络数据包。其业务逻辑在 fetch_packets 接口中实现，需要向接口中传入网络接口名称和网络接口序号。抓包任务会使用自定义的 muxPacketAttach 函数进行抓包，当捕获到一个网络数据包后，首先判断是否采

用过滤器，然后在缓冲区未满时，保存此包。推送包任务循环访问缓冲区，判断环形缓冲区非空，取出一个数据包进行推送到客户端，释放内存。

(3)停止捕获

捕获软件在接收到停止捕获命令后，将调用 stopMuxPacketTask 接口，解除 MUX 与驱动的绑定，删除推送网络网络包任务。

(4)过滤数据包

捕获软件在接收过滤数据包命令后，将全局变量结构体中的 filterMode 字段根据命令携带的信息进行赋值，然后根据 filterMode 在捕获网络数据包过程启用单个条件过滤或组合条件过滤，过滤器的具体实现将在下面章节中展开。

3.2.3 数据包捕获模块

数据包捕获模块是整个系统的核心功能模块，在接收到捕获命令后启动，负责捕获指定网络接口的网络数据包，并转存到缓冲区，等待后续处理。流程如图 3.5 所示。通过在第二章中我们讨论了网络数据包捕获方式，最终选择了使用 MUX 接口实现满足需求的捕获网络数据包的模块，接下来具体讨论使用 MUX 方案。

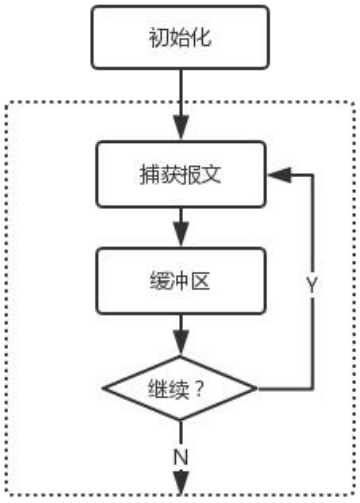


图 3.5 网络包捕获模块

在捕获网络数据包接口中使用 MUX 提供的 muxBind 函数启动抓取网络数据包的任务，根据用户需求，需要抓取从设置的网络适配器进出的网络数据包，因此需要使用 muxBind 分别针对 MUX_PROTO_SNARF 和 MUX_PROTO_OUTPUT 两个协议进行绑定。在停止捕获时，也需要分别针对两个协议进行解绑。表 3.9 给出了实现的伪代码。

表 3.9 muxPacketAttach 实现伪代码

```
if ((capInstance_p.pCookie_s =muxBind(device, unit,
        &muxPacketRcvRtn_S,&muxPacketShutdownRtn,&muxPacketRestartRtn,
        &muxPacketErrorRtn, MUX_PROTO_NAME_S,NULL, NULL)) == NULL)
        return ERROR;
if ((capInstance_p.pCookie_o =muxBind(device,unit,
        &muxPacketRcvRtn_O,&muxPacketShutdownRtn,&muxPacketRestartRtn,
        &muxPacketErrorRtn,MUX_PROTO_NAME_O,NULL,NULL))==NULL)
{       muxUnbind(capInstance_p.pCookie_s,MUX_PROTO_NAME_S,muxPacketRcvRtn_S);
        capInstance_p.pCookie_s = 0;
        return ERROR;   }
```

注意 muxPacketRcvRtn_O 和 muxPacketRcvRtn_S 是对 muxPacketRcvRtn 函数的再次封装，muxPacketRcvRtn 是接收到网络数据包处理过程：①当接收到数据包后，判断是否启动滤器，启动根据规则过滤；②将符合条件的数据包查询放入环形缓冲区中，当缓冲区满停止捕获，将捕获状态设置为 CapIsFull_State，打印信息提示调整缓冲区大小；③当捕获出错，将捕获状态设置为 StartCapErr_State，打印出错信息。

3.2.4 数据包过滤模块

数据包过滤模块是系统的核心功能模块之一，主要负责根据分析软件的命令设置相关的过滤器，过滤符合规则的数据包。图 3.6 给出了使用数据包过滤模块的流程图。本系统的过滤模块使用的是定制简易版的过滤器，由图 3.6 可以发现过滤器主要针对四方面过滤：(1)IP 源地址和目标地址；(2)TCP/UDP 源端口和目的端口；(3)协议类型；(4)IP 地址与端口结合。

过滤器实现的原理：用当前访问网络数据包指针位置加上当前网络协议头部长度得到下一个网络协议访问的位置，进行访问协议头部，判断是否满足过滤条件。

如下得到 IP 头部访问的位置：

```
char *data = buf + sizeof(struct ether_header);
```

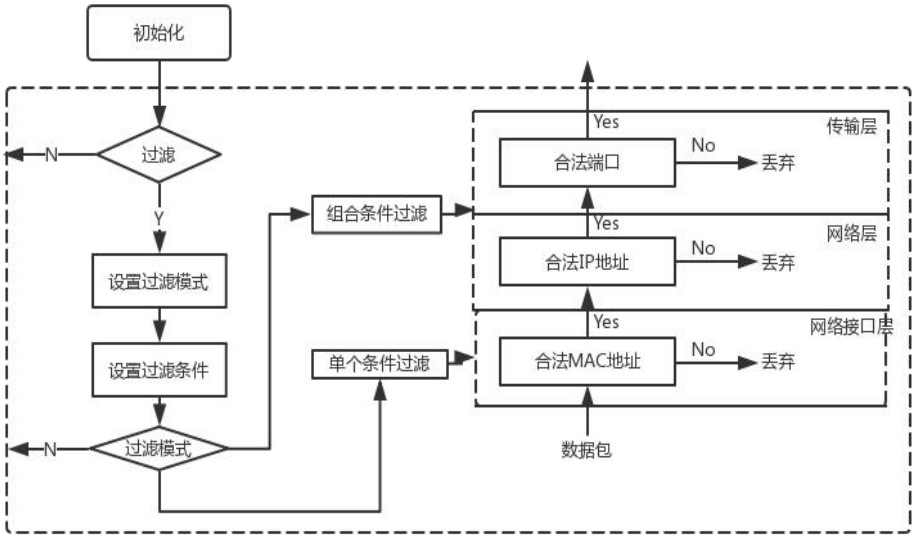


图 3.6 数据包过滤模块

过滤器分为单个条件过滤模式与混合条件过滤模式，当用户设置过滤规则，将发送携带过滤规则信息的过滤命令到捕获软件，捕获软件解析命令，并填充过滤器结构体，即设置过滤模式以及设置过滤条件。如下是填充过滤器的过程。

```
memcpy(&capInstance_p.packetFilter, local_cmd->buf, sizeof(local_cmd->buf));
capInstance_p.filterMode = local_cmd->info
```

packetFilter 是一个过滤器结构体变量，定义如表 3.10。PacketFilter 分为两种过滤模式：①单个条件过滤，此时只关注结构体的前两个字段，即只过滤协议类型；②组合条件过滤，此时根据 firstFilterType 字段的类型，决定是否访问结构体前四个字段，根据 nextFilterType 字段类型决定是否访问后四个字 firstFilterType 为 NoFilter 类型时，不访问前四个字段，nextFilterType 同理。当是组合条件过滤时，前四个字段负责与 IP 相关的过滤条件，后四个字段负责与 TCP/UDP 相关的字段。组合条件过滤负责除了协议类型的其他单个方面。表 3.11 给出了所用到 FilterType 的定义。

表 3.10 过滤器结构体

```
/**
 * 结构体 PacketFilter， 过滤控制体
 * 当 IP 与 TCP 或 UDP 组合过滤时候，则第一个协议设置为 IP 相关
 */
typedef struct {
    enum ProtoName firstProtoName;        /**< 过滤第一个协议名称 */
    enum FilterType firstFilterType;      /**< 过滤类型，决定第一个协议过滤方式 */
    uint32_t ip_src;                      /**< IP 源地址 */
    uint32_t ip_dst;                      /**< IP 目标地址 */
    enum ProtoName nextProtoName;         /**< 组合过滤第二个协议名称 */
    enum FilterType nextFilterType;       /**< 过滤类型，决定第二个协议过滤方式 */
    uint16_t port_src;                    /**< 源端口号 */
    uint16_t port_dst;                    /**< 目标端口号 */
    } PacketFilter;
```

表 3.11 enum FilterType 的定义

```
enum FilterType {
    NoFilter        = 0x00,
    ArpFilte        = 0x05,
    RarpFilter      = 0x07,
    IcmpFilter      = 0x09,
    IgmpFilter      = 0x03,
    IpOnlyFilter    = 0x01,
    IpSrcFilter     = 0x02,
    IpDstFilter     = 0x04,
    IpAllFilter     = 0x02 | 0x04,
    TcpOnlyFilter   = 0x0B,
    TcpSrcFilter    = 0x08,
    TcpDstFilter    = 0x10,
    TcpAllFilter    = 0x08 | 0x10,
    UdpOnlyFilter   = 0x0D,
    UdpSrcFilter    = 0x20,
    UdpDstFilter    = 0x40,
    UdpAllFilter    = 0x20 | 0x40,
    };
```

过滤网络数据包的业务逻辑在packetFilter接口中实现，接口根据filterMode类型，决定执行两种过滤模式其中的一种，之后根据过滤条件选择符合条件的网络数据包。

表 3.12 enum FilterMode 定义

```
enum FilterMode {
    NoneFilter = 0,          /**< 无过滤 */
    SingleFilter,            /** < 单条件过滤 */
    CombineFilter            /**< 组合条件过滤 */  };
```

3.2.5 缓冲管理模块

在前面章节叙述过使用环形缓冲区减少数据包丢失的情况，缓冲管理模块承担这样的角色，负责存储捕获的网络数据包，并等待通信模块取出推送到分析软件进行分析。过程流程图如图 3.7。

图 3.7 缓冲管理模块使用流程图

3.2.5.1 环形缓冲区实现

终端捕获软件使用循环队列来定制一种特定环形缓冲区，满足系统要求。循环队列的结构体定义如表 3.13。

表 3.13 循环队列定义

```
typedef struct {
    int packqueue_head;                          /**< 队列头位置 */
    int packqueue_tail;                          /**< 队列尾位置 */
    PacketInfo packet_list[MAX_PACK_NUM];        /**< 存储数据包 */
} pCapQueueControl;
```

PacketInfo 是存储网络数据包的结构体，表 3.14 给出了其定义。

表 3.14 PacketInfo 结构体

```
typedef struct _Packet_Info
{
    int state;                                   /**< 网络数据包状态 */
    int len;                                     /**< 网络数据包长度 */
    unsigned char buf[MAX_PACK_LEN];             /**< 存储网络数据包 */
} PacketInfo;
```

从上面可以看出定制的循环队列有下面几个特点：(1)可存储的数据包的个数

是固定的，是在宏 MAX_PACK_NUM 定义，不能动态修改。在 MAX_PACK_NUM 设置太小或者抓包速度远大于发送数据包速度，就会出现丢包现象，如果太大造成不必要的资源浪费；(2)存储网络数据包字段为静态数组，避免了繁琐的分配内存和释放内存的操作，也因此一开始确定了大小，会造成一定程度空间的浪费；(3)向队列中添加与读取数据都满足互斥同步原则；(4)入队与出队的时间复杂度为 O(1)。

下面详细讲解使用定制的循环队列作为环形缓冲区的基本思想：(1)先创建一个全局变量保存当前循环队列的状态，将这个循环队列看做一个环；(2)packqueue_tail 表示当前抓包任务处理的位置，packqueue_head 表示当前推送数据包任务处理的位置，初始 packqueue_tail = packqueue_head = 0；(3)当收到一个数据包之后，分配空间保存此包，当前处理的位置为(packqueue_tail+1)% MAX_PACK_NUM，如图 2.13.A；当它的下一个位置等于 packqueue_head，说明缓冲区已满，如图 2.13.B 所示；当 packqueue_head == packqueue_tail，表示缓冲区为空，如图 2.13.C 所示；(4)当 packqueue_head != packqueue_tail，推送网络数据包任务开始工作，每推送一个包，packqueue_head=(packqueue_head+1)% MAX_PACK_NUM，并释放该包，一直处理完缓冲区。

3.2.5.2 互斥同步

缓冲区在软件设计中要被多个任务使用到，因此要保证数据存储与读取安全，在 VxWorks 下不支持多线程，系统使用二进制信号量完成互斥同步，二进制信号量（Binary Semaphores）是完成互斥、同步操作的最佳方式，VxWorks 提供的信号量经过了高度优化，在所有任务间通信机制中，速度最快。

为了方便后续使用，这里自定义互斥同步的操作，基于二进制信号量封装了简单的加锁解锁操作，可以安全的访问临界资源，主要使用的是 semTake 和 semGive 两个函数。这里的互斥同步在系统的其他模块涉及到读写安全处用到。表 3.15 给出相应接口。

表 3.15 信号量互斥同步接口

STATUS capLock(SEM_ID key);
STATUS capUnLock(SEM_ID key);

3.3 本章小结

本章重点介绍了网络包捕获软件的设计与实现方案。首先讨论了捕获软件的总体设计，给出了捕获软件的总体架构；然后讨论捕获软件各个模块的设计与代码实现。下一章将讨论网络包分析软件设计与实现的细节。

第四章 网络包分析软件的设计与实现

4.1 总体设计

网络包分析软件运行在 Windows 或 Linux 客户端，拥有实时监测和历史查询两种工作模式。分析软件在实时监测工作模式下，负责控制整个捕获流程和接收来自捕获软件推送的网络数据包，并转存至缓冲区；分析软件在历史查询工作模式下，可以读取本地保存的历史网络数据包文件，并转存至缓冲区。两种工作模式下，后续操作将使用数据包解析模块，并将解析结果填充到分组列表窗口，当用户点击时，展示详细的协议树信息。

网络分析软件主要包括如下几个功能模块：(1)界面显示模块；(2)缓冲管理模块；(3)搜索模块；(4)数据包解析模块；(5)历史查询模块；(6)数据存储模块；(7)捕获控制模块；(8)通信模块。如图 4.1 给出了网路包分析软件的总体结构图。

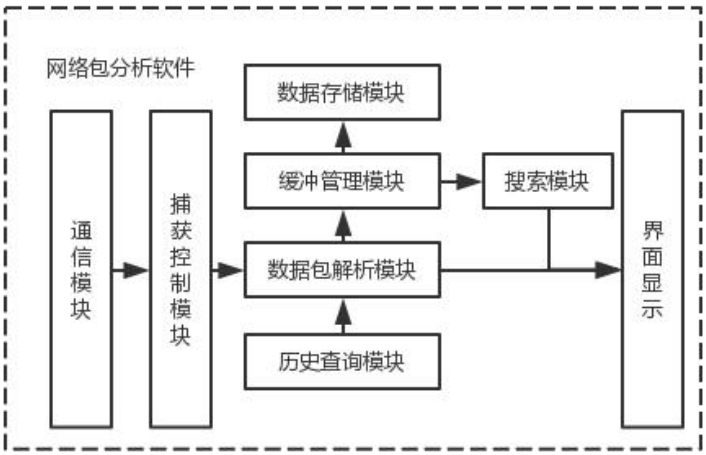


图 4.1 网络包分析软件总体结构

网络包分析软件使用 C++开发，开发框架 Qt，可以运行在 Windows 平台和 Linux 平台。接下来我们详细介绍分析软件功能模块的设计与实现。

4.2 功能模块设计与实现

4.2.1 界面显示

界面显示为用户提供操作软件的入口，以及显示信息的载体。界面接收用户的操作信息，切换工作模式，调用核心功能层完成动作。界面提供如下动作：

(1) 实时监测模式相关：获取网路接口列表和设置网络接口动作；控制捕获流程，开始、停止与结束捕获动作；设置过滤动作；保存文件动作。

(2) 历史查询模式相关：打开文件动作。

(3) 两种模式通用：搜索动作。

网络分析软件的界面设计如图 4.2 所示。

图 4.2 界面设计

菜单栏方便用户切换不同的需要，主要包括以下几个部分：(1)项目：设计到展示已有的网络接口，用户选择其中一个网络接口进行抓包、关闭、退出等；(2)文件：包括打开文件、保存文件；(3)捕获：包括开始捕获、停止捕获、清除三个抓包交互功能；设置捕获过滤器；(4)视图：控制捕获数据的显示方式，包括字体缩放，根据协议类型进行着色；搜索网络数据。

菜单栏下面是网络数据包信息展示的三个窗口：(1)分组列表窗口：网络数据

包分析的概要信息在这里展示，包括网络数据包的源地址与目标地址、网络数据包的协议类型、网络数据包的捕获时间、网络数据包的大小、网络数据包最高层携带的简要信息。(2)分组详情窗口：当点击分组列表窗口的一个条目的时候，客户端会对当前网络数据包进行解析，将每层协议进行解析并将解析的详细信息填充到分组详情窗口,以协议树显示。(3)分组字节窗口：当点击分组列表窗口的一个条目的时候，客户端会将原始网络数据包转换成十六进制字节，并与原始网络数据包在分组字节窗口处一一对应的显示。

分组详情窗口与分组字节窗口之间存在互动，当点击协议树信息的某一条信息，会在分组字节窗口高亮显示这条信息对应的字节信息；同样当点击字节信息时，会根据当前点击的范围定位协议树信息。为了实现这种效果，自定义协议树结构 ProtoTree 存储协议树每条信息对应的字节起始位置和偏移量，便于后续的与字节匹配信息。

4.2.2 通信模块

通信模块是捕获软件与分析软件通信的关键，使用串口进行通信，具有两个主要任务：(1)根据捕获控制模块的动作向捕获软件发送捕获命令（开始捕获、停止捕获结束捕获等），并从捕获软件接收响应信息；(2)从捕获软件接收网络数据包。图 4.3 给出了通信模块的类关系图。

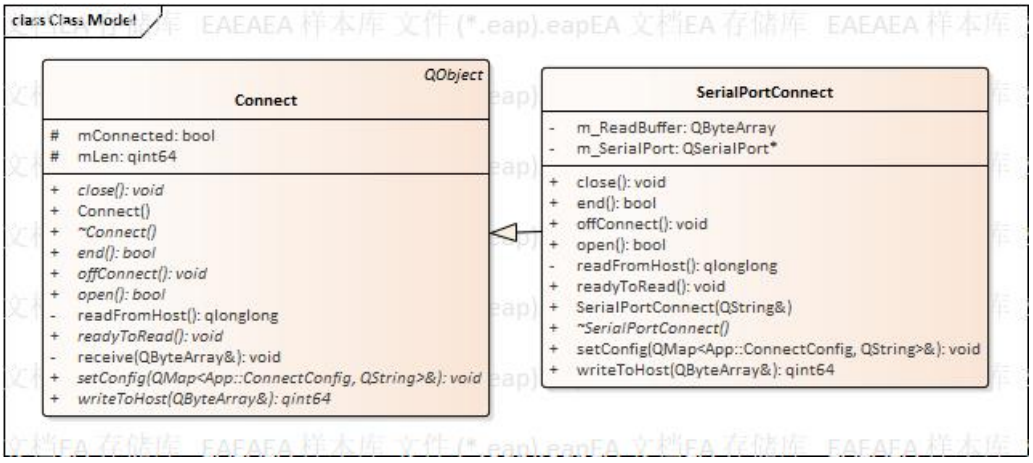


图 4.3 通信模块类关系图

为了让系统具有良好的扩展性，通信模块定义了类 Connect 作为所有通信模块的父类，当需要实现其他通信方式，只用继承类 Connect。readFromHost 定义通信接收数据接口，writeToHost 定义通信发送数据接口。

SerialPortConnect 继承 Connect 类，实现相关接口，是通信模块的业务逻辑实现类。在第三章中我们提出了保证串口通信接收完整包的方式，即使用数据缓冲区累计收到的字符，当缓冲中第一次出现帧头和帧尾，表示出现一次完整数据包，具体代码实现在 SerialPortConnect 定义的槽函数中。实现的伪代码如表 4.1.

```
private slots:
    /**
    * 槽函数-从目标主机读取数据
    * @return qint64, 成功返回收到数据的字节数；失败返回-1
    */
    qlonglong readFromHost();
```

表 4.1 分析软件串口通信接收数据伪代码

```
QByteArray m_ReadBuffer;                          //定义缓冲区
m_ReadBuffer.append(datagram);                    //datagram 从串口接收数据
if (m_ReadBuffer.contains(CONTENT_END)) {//表示接收到一完整数据包
    int start = m_ReadBuffer.indexOf(CONTENT_START);
    int end= m_ReadBuffer.indexOf(CONTENT_END);
    ...
    ...
    处理完整数据包
}
```

4.2.3 捕获控制模块

实时监测模式是网络分析软件工作模式的一种，负责控制捕获网络数据包流程，包括获取网络接口列表、设置网络接口、开始捕获、停止捕获、结束捕获等方面。捕获控制模块通过调用通信模块，在用户执行动作，向捕获软件发送对应的命令，并实时接收来自终端发来的信息，并将信息传送到网络解析模块进行解析转存到缓冲区 A 中，等待后续操作。图 4.4 给出了捕获控制模块工作流程。

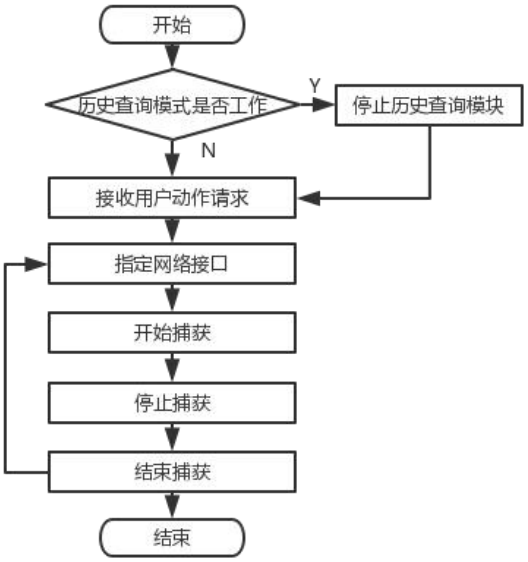


图 4.4 捕获控制模块工作流程

(1)当用户点击获取网络接口菜单时，分析软件初始化通信模块并打开串口，建立与捕获软件的连接，同时发送获取网络接口列表的命令。捕获软件将网络接口列表推送至分析软件在窗口中展示。

(2)用户指定网络接口开始捕获，此时会向捕获软件先后发送指定网络接口命令和开始捕获命令。分析软件在发送指定网络接口命令后，开启一个定时器，在一定时间后发送开始捕获命令，确保捕获软件具有足够的设置命令的时间，命令顺序执行。开始捕获后，分析软件初始化在线读设备，实时接收来自串口的网络数据包进行分析并保存至缓冲区，并发信号至主界面将分析结果展示出来。在线读设备在实时监控模式下启动，图 4.5 给出了类关系图。实时读取网络包的业务逻辑在类 PcapLiveReaderDevice 中实现，该类继承 IReaderDevice 与 QObject，是一个在线读设备。这里使用多线程读取网络数据包，使用 QObject 函数的 moveToThread，将 PcapLiveReaderDevice 的运行线程转移。因为读取网络包并进行分析是一个耗时的操作，需要开辟新线程与 UI 线程独立开，避免程序的崩溃。

(3)用户设置过滤条件，执行过滤操作，分析软件构造携带过滤条件的过滤命令发送至捕获软件，捕获软件设置过滤条件，捕获符合过滤条件的网络数据包。

(4)用户执行停止捕获操作，分析软件向捕获软件发送停止捕获命令，捕获软件停止捕获，分析软件停止读取分析网路数据包的线程。

(5)用户执行重置捕获操作，分析软件提示用户保存当前捕获的网路数据包，之后初始化在线读设备，销毁分析数据包线程，关闭串口，初始化通信模块，等待用户继续操作。

(6)当用户关闭分析软件时，会根据当前的状态提示是否需要保存网络包，停止所有正在运行的模块以及线程，向捕获软件发送关闭捕获命令，之后软件退出。

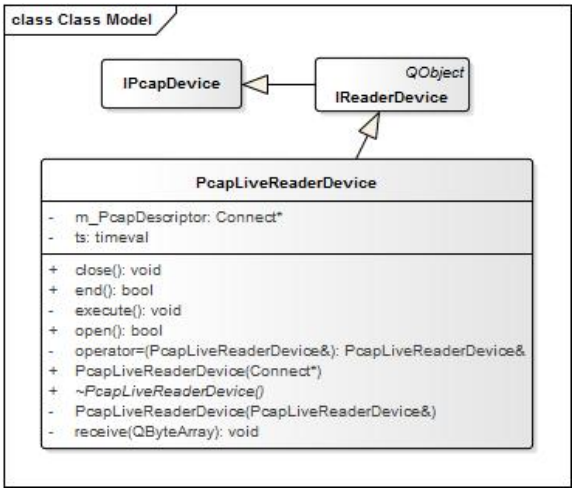


图 4.5 实时监控读取网络包类关系图

4.2.4 数据包解析模块

数据包解析模块根据 TCP/IP 协议层模型以及数据封装规则对接收到的网络数据包层层分析，提取每层协议的关键信息[37-40]。数据在 TCP/IP 模型中从下往上传递，我们需要层层去掉头部信息，获取该层协议的原始数据进行分析[41,42]。根据用户需求，系统将针对 Ethernet，ARP，IPV4，IPV6，ICMP，TCP 与 UDP 协议进行解析。

在分析软件启动网络数据包分析功能后，首先初始化分析线程，分析线程循环从服务器或者文件中获取网络数据包，如果合法，解析数据包，发送信号通知显示模块显示，并将原始数据包和解析结果分别放到缓冲区 A 和缓冲区 B。解析数据包流程如图 4.6 所示。

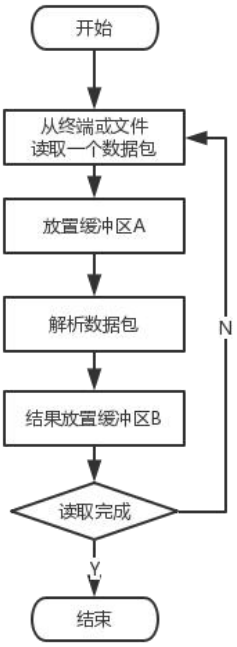


图 4.6 解析数据包流程图

4.2.4.1 数据包解析模块设计

解析网络协议就是根据 TCP/IP 协议簇从下到上按照协议栈一层一层进行协议分析，分析完成之后将分析的结果存放到制定的数据域中[3]。解析协议的过程就是解析各个协议的首部，从数据包提取我们感兴趣的信息，因此需要自定义系统用到的各个协议首部。

网络解析首先根据自定义的各协议首部，解析 Ethernet 首部内容，确定协议类型，ARP 或 IP；之后分别解析对应的首部。我们以解析 IP 为例：定义一个 IPV4 头指针 ipHdr，通过移动 ipHdr 指针实现解析网路数据包。接下来根据定义的 IPV4 首部结构，读取对应的各项信息。根据 IPV4 首部 protocol 字段确定下一层协议为 TCP 数据包、UDP 数据包还是 ICMP 数据包[37]。

为了保证系统良好的扩展性，利于系统维护，这里我们采用面向对象的思想设计协议解析引擎。如图 4.7 所示是类之间的关系。类 Layer 是所有协议层的父类，定义了共有字段以及操作，表 4.2 给出了 Layer 关键字段与函数的含义。

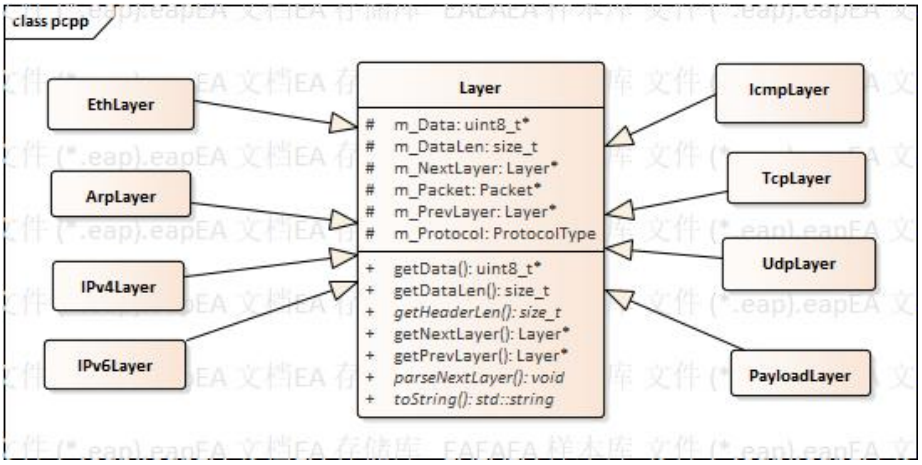


图 4.7 协议解析引擎类关系图

表 4.2 Layer 关键字段与函数含义

名称	含义
m_Data	当前协议层的数据
m_DataLen	当前协议层的数据长度
m_Packet	类 Packet 实例，获取每一层协议的入口,比如获取 TCP 层: TcpLayer* layer = packet.getLayerOfType<TcpLayer>();
m_Protocol	当前协议层的类型
parseNextLayer	解析下一层协议的业务逻辑实现位置

4.2.4.2 数据包解析模块具体实现

1. EthLayer 层处理

EthLayer 层处理主要根据 MAC 帧格式，将数据包前 14 个字节解析成对应的数据结构,再从数据结构中读取每个字段代表的详细信息。根据 MAC 帧首部 etherType 字段，可以确定当前捕获哪种协议类型的网络数据包，当该字段为 0x0800，IP 协议数据包；当该字段为 0x0806，ARP 协议数据包[4]。然后根据协议类型决定下一步的处理过程。图 4.8 给出 EthLayer 层的处理流程。

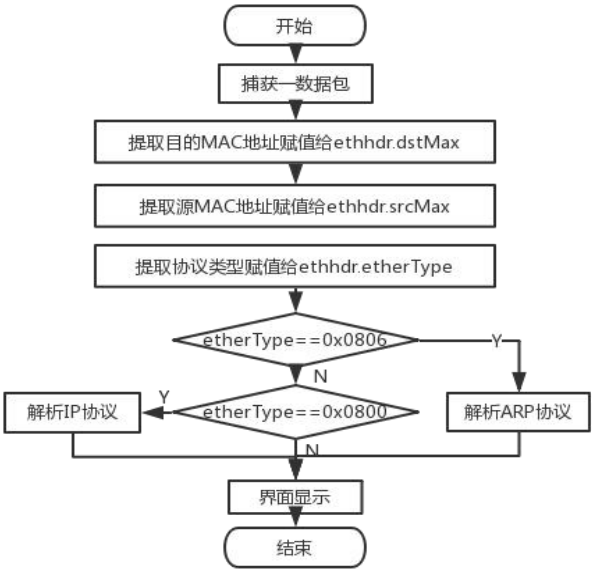


图 4.8 EthLayer 层处理流程

2. ArpLayer 层处理

ARP 协议，地址解析协议，是一种 TCP/IP 协议，作用根据 IP 地址获取物理地址。可以从 ArpLayer 层提取硬件类型、协议类型、MAC 地址、IP 地址等关键信息[4]。图 4.9 给出了 ArpLayer 层处理流程。

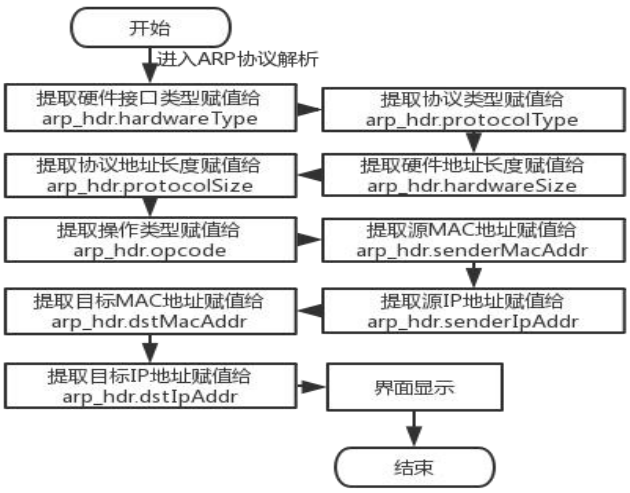


图 4.9 ArpLayer 层处理流程

3.IP 层处理

IP 层处理包括 IPvLayer 和 IPv6Layer，IPV4 根据首部中的 protocol 域字段，IPV6 根据首部的 Next Header 字段判断下一层的协议类型[4]，当值为 6 时下一层协议为 TCP 协议，当值为 17 时下一层协议为 UDP 协议，当值为 1 时下一层协议为 ICMP 协议。处理流程如图 4.10。

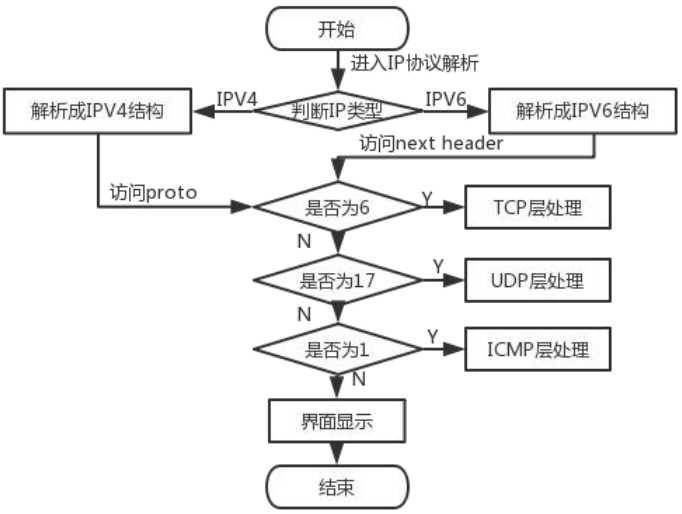


图 4.10 IP 层处理流程

4. TcpLayer 层处理

在 TcpLayer 层，需要根据 TCP 首部结构获取想要的信息，这里我们将对 TCP 所有字段进行解析，主要包括端口号、标识位、窗口大小、校验和、紧急数据偏移量等[4]。处理流程如图 4.11。

5.UdpLayer 层处理

UdpLayer 层与 TcpLayer 层相同，对 UDP 首部所有字段进行解析，不过相比 TCP 而言，要分析的信息就少的多，主要是端口号，数据包长度以及校验和[4]等几个内容。处理流程如图 4.12。

6.IcmpLayer 层处理

IcmpLayer 层是对 ICMP 进行处理，在 TCP/IP 网络协议中常被当做是 IP 层的一部分，主要负责传递差错报文，并携带其他关键信息。ICMP 报文是在 IP 数据包内

被传输的，处理流程如图 4.13。

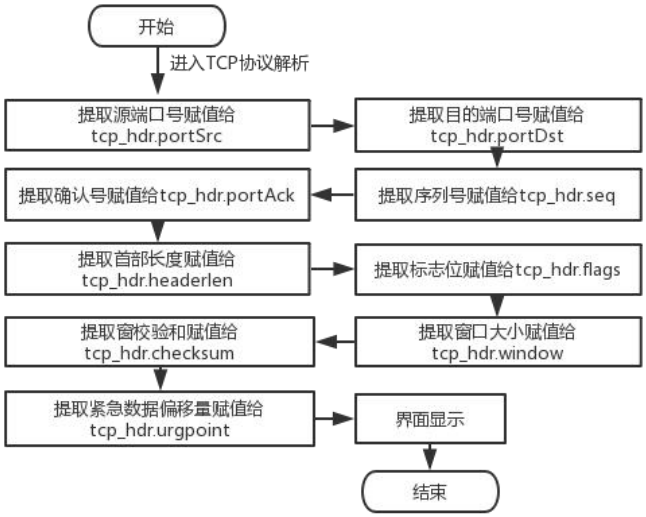


图 4.11 TCP 层处理流程

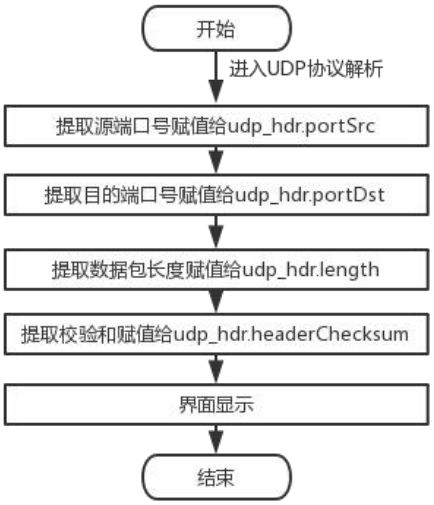


图 4.12 UDP 层处理流程

7.PayloadLayer 层处理

这一层不是实际意义上的协议层，用于存储目前系统不支持解析的协议原始数据包数据，可通过 getPayload 函数获取，该层数据主要以字节形式在分组字节窗口中显示。

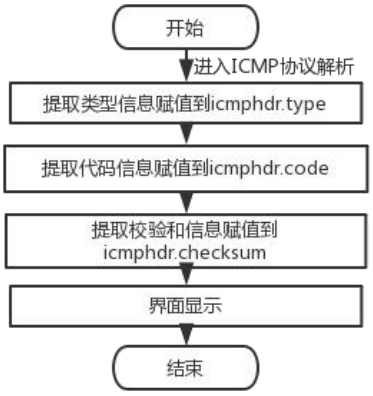


图 4.13 ICMP 层处理流程

4.2.5 历史查询模块

历史查询模式是网络分析软件工作模式的一种，调用历史查询模块，负责从本地读取历史网络数据包文件，调用数据包解析模块进行解析。当用户选中本地文件打开时，会自动调用该模块查看流量信息。如图 4.14 给出了历史查询模块的工作流程。

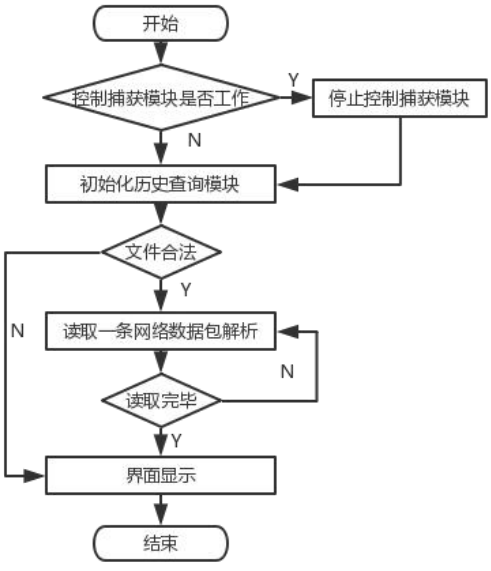


图 4.14 历史查询模块工作流程

首先在切换到历史查询模式下工作后，历史查询模块会判断捕获控制模块是否已经工作，即实时监控，如果已经工作，则将其关闭；接着初始化历史查询模块，读取文件，当文件符合 pcap 文件格式则继续，否则退出；文件读取模块循环从文件读取网络数据包，使用网络解析模块进行解析，并将解析结果转从到缓冲区 A 中，直至读取完毕；读取缓冲区 A 的结果渲染界面。

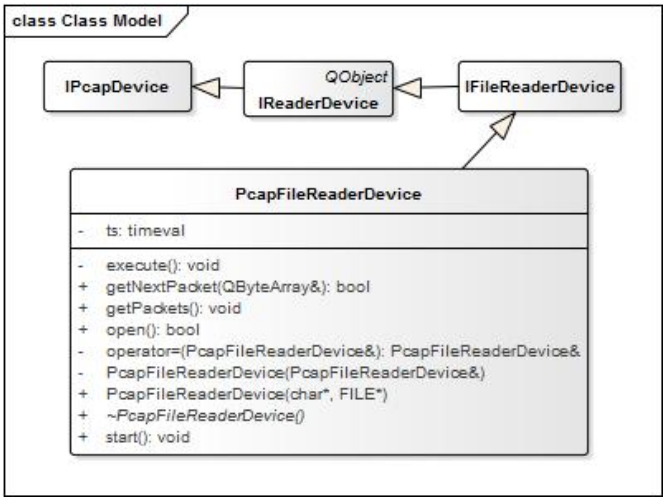


图 4.15 历史查询模块类关系图

历史查询模块的关键在于正确的从 pcap 文件读取网络数据包，我们在第二章已经详细的介绍了 pcap 文件格式，这里将详细介绍如何实现。这里的结构也是数据存储模块需要的。表 4.3 和表 4.4 给出自定义的文件头和数据包头。

表 4.3 pcap 文件头结构体

```
struct pcap_file_header{
    uint32_t magic;
    uint16_t version_major;
    uint16_t version_minor;
    int32_t thiszone;
    uint32_t sigfigs;
    uint32_t snaplen;
    uint32_t linktype;} ;
```

表 4.4 pcap 数据包头结构体

```
struct pcap_packet_header
{
    uint32_t tv_sec;
    uint32_t tv_usec;
    uint32_t caplen;
    uint32_t len;
};
```

读取 pcap 文件需要下面步骤：(1)首先需要读取文件头，判断文件合法，这里判断 pfhdr.magic 字段与定义的 MAGIC_NUMBER 是否相等，如果相等去除文件头；(2)去除网络包头，根据包头中的 caplen 属性，向后读取 caplen 字节从文件得到

原始网络数据包；(3)使用网络解析模块解析网络数据包；(4)重复执行 2，3 步骤。经过上面四个步骤可以实现历史查询模块的业务逻辑。图 4.15 给出了历史查询模块的类关系图。模块业务逻辑在类 PcapFileReaderDevice 中实现，该类继承自 IReaderDevice，是一个本地读设备。所有扩展读设备，应该继承类 IReaderDevice。

4.2.6 数据存储模块

数据存储模块为用户提供了一种保存已经捕获的历史数据包的方式，负责从缓冲区中读取数据包并保存成 pcap 格式文件，目前只支持 pcap 格式。如图 4.16 给出了数据存储模块的工作流程。首先使用数据存储模块确保捕获控制模块已经停止，然后判断缓冲区 A 是否存在存在数据，如果存在则从中取出一条数据保存到本地，直至读取完成或停止保存。

保存 pcap 文件需要如下步骤：(1)首先构建文件头；(2)根据网络数据包信息构建数据包头；(3)拼接数据包；(4)重复 2,3 步骤。通过上述就可以实现本地保存 pcap 文件了。

图 4.17 给出了数据存储模块的类关系图。数据存储模块的业务逻辑在类 PcapFileWriterDevice 中实现，该类继承自 IWriterDevice，是一个本地写设备。所有扩展的写设备应该继承类 IWriterDevice。

4.2.7 搜索模块

搜索模块是分析软件的核心功能模块之一，为了用户从已经捕获的数据包中查找感兴趣的数据包而设计。与捕获软件的过滤模块对比，不同在于前者是静态的，是历史查找。下面从搜索模块的设计与实现两个方面介绍。

4.2.7.1 搜索模块设计

搜索模块根据用户的输入设置从缓冲区中搜索匹配的一个数据包。根据展示方式的不同这里将搜索模块设置成如下几种搜索模式，如表 4.5 所示。

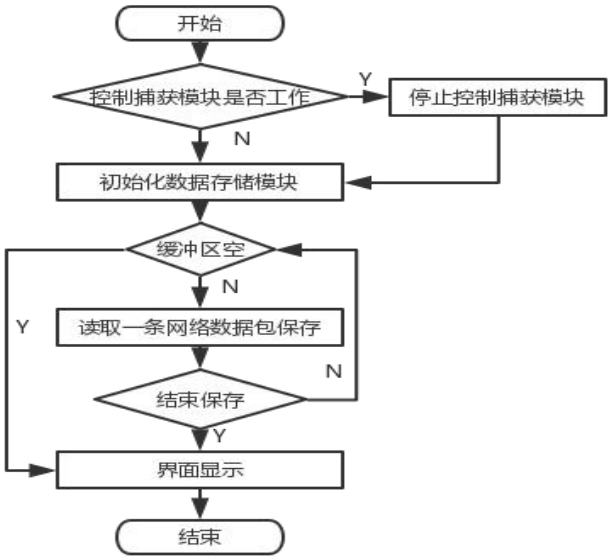


图 4.16 数据存储模块工作流程

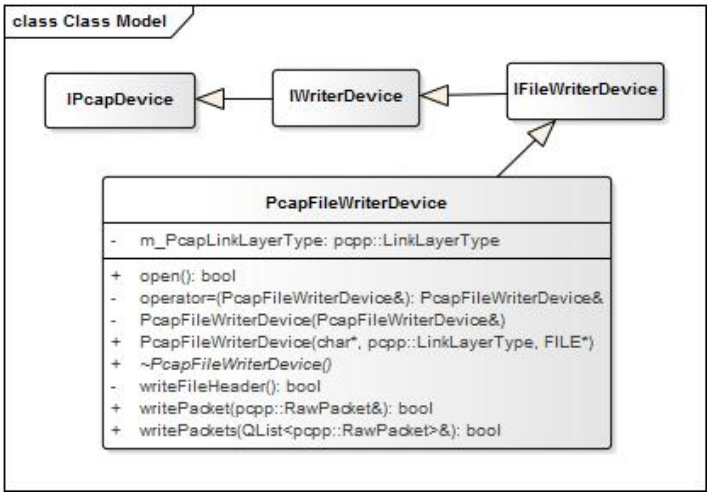


图 4.17 数据存储模块类关系图

表 4.5 搜索模式

字符串搜索	分组列表	区分大小写
	分组详情	
	分组字节	
十六进制搜索	双字节搜索	

(1)字符串搜索：此模式下可以选择分组列表、分组详情、分组字节其中一种以字符串的信息遍历历史网络数据包,分别针对提取的关键信息搜索，协议树信息搜索，原始数据进行搜索，同时用户可以根据自己的兴趣区分大小写搜索;

(2)十六进制搜索：此模式下是对分组字节以十六进制字节的信息遍历历史网络数据包。此模式下用户的输入必须是双字节或者双字节的倍数，否则会被认为是非法输入，这跟分组字节窗口展示信息的方式有关。

当用户设置搜索条件，只有合理的条件才会被模块采用，主要流程如图 4.18 所示。(1)初始化搜索模块(2)用户设置搜索条件，点击搜索按钮进行搜索，此时会对用户输入进行检测，只有合法才会进行搜索。①当搜索到符合条件的数据包，分析软件跳转到对应条目的数据包，高亮显示；②继续搜索，当搜索条件与上次相同，搜索模块会从当前位置往后搜索；当搜索条件与上次不同，重新设置搜索模块；当未搜索到，跳出提示。(3)退出搜索模块

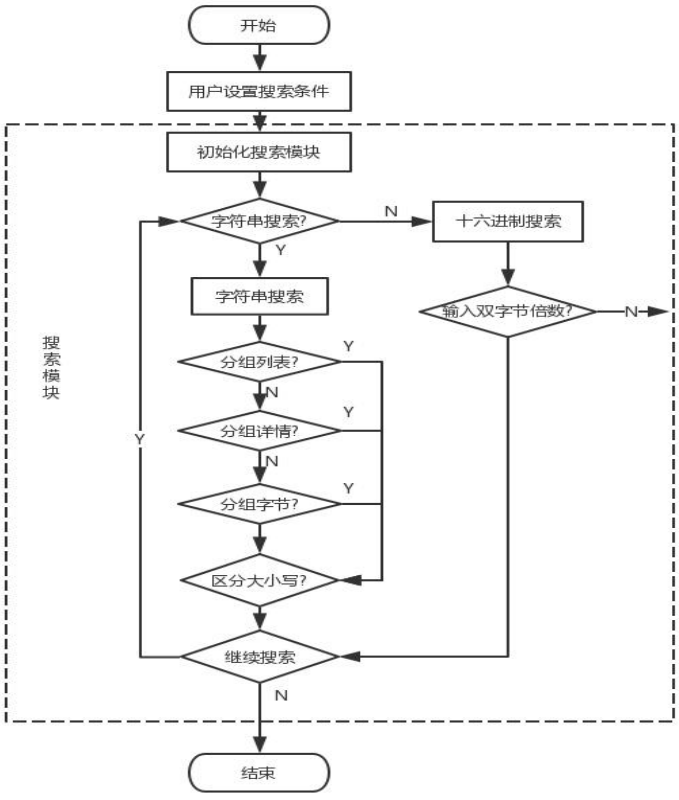


图 4.18 使用搜索模块流程

4.2.7.2 搜索模块具体实现

上一小节介绍了搜索模块的设计以及使用流程，接下来将介绍模块的实现细节。图 4.19 给出了搜索模块的类关系图。搜索历史数据包是一个耗时操作，如果直接将搜索模块放在 UI 线程中，将阻塞 UI 线程，导致程序崩溃，为此设计一个搜索线程专门负责搜索模块业务逻辑。

SearchThread 类是实现搜索模块业务逻辑的多线程类，继承自 Qt 的 QThread 类，我们需要重载 run 函数来实现我们的搜索业务逻辑代码，为了方便控制搜索流程，自定义 stop 和 start 函数。启动搜索线程后，在 run 函数中会根据模式调用对应的搜索接口，接口定义如表 4.6 所示。四个接口需要共同调用字符串搜索函数 matchBinary，其定义如下表 4.7。

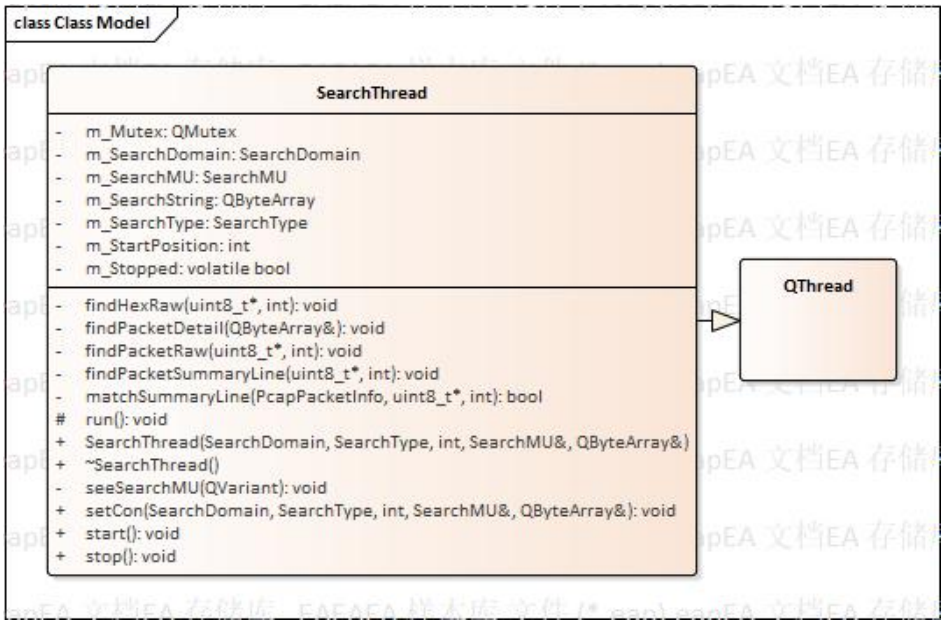


图 4.19 搜索模块类关系图

表 4.6 搜索接口定义说明

接口名称	接口说明
findPacketSummaryLine	字符串模式查找分组列表
findPacketDetail	字符串模式查找分组详情
findPacketRaw	字符串模式查找分组字节
findHexRaw	十六进制模式查找分组字节

表 4.7 matchBinary 函数定义

```
bool matchBinary(const uint8_t *data, int dataLen, const uint8_t *criterion,
int criterionLen, SearchMU *searchMu)
```

matchBinary 函数是搜索模块的核心功能接口，根据本系统需求实现了 KMP 算法，让字符串搜索复杂度达到 O(m+n)，让搜索模块简单高效。KMP 算法利用匹配失败后的信息，减少模式串与主串的匹配次数，是一种快速匹配字符串的高效算法，代码的实现思想这里就不再赘述。

表 4.8 SearchMU 结构体

```
class SearchMU {
public:
    int m_CurrentRow;                    /**< 数据包所在的行数，用于定位 */
    int m_SearchPos;                     /**< 匹配字符串的最后位置 */
    int m_SearchLen;                     /**< 搜索字符串长度 */
    SearchType _searchType;              /**< 搜索模式 */
    SearchDomain _searchDomain;          /**< 字符串搜索模式下的搜索方式 */
    bool _sensitive;                     /**< 区分大小写 */
    int _treeInfo_id;                    /**< 协议树信息叶子 ID */
};
```

matchBinary 接口搜索成功，将返回 true，并将字符串匹配的最后一个位置以及所搜索的字符串长度保存在 SearchMU 结构体中，用于分组字节的高亮显示在这里需要注意的是，十六进制模式下搜索也调用了 matchBinary 函数，具体是读取用户输入的双字节十六进制条件，并根据规则转化成原始数据包字符串再进行字符串搜索。表 4.8 定义了 SearchMU，SearchMU 存储搜索模式条件以及搜索结果，通过信号 seeSearchMU 传递到与之绑定的槽函数中，实现搜索界面交互。

当用户进行搜索操作，点击搜索按钮时，将调用 MainWindow 文件中的 setSearchCon 函数。该函数首先检查当前是否有搜索线程已经启动，如果有，则停止，重新设置搜索条件启动；如果没有，则创建新的搜索线程，建立信号槽连接。表 4.9 给出了创建搜索线程伪代码。

表 4.9 SearchThread 创建伪代码

```
if(线程已经启动) {
    stopSearchThread();
    resetSearchCondition();
```

表 4.9 SearchThread 创建伪代码

```
    restartSearchThread();
} else {
    createSearchThread();
    connectSearchSignalAndSlots();
    startSearchThread();
}
```

4.2.8 缓冲管理模块

缓冲管理模块作用主要包括如下方面：(1)缓冲区 A 存储来自从捕获软件推送的或者从本地文件读取的原始数据包；(2)缓冲区 B 存储解析网络数据包的简要信息；(3)搜索模块使用缓冲区 A 与缓冲区 B 根据用户请求搜索；(4)数据存储模块读取缓冲区 A 保存数据包。（4）根据缓冲区 B 填充分组列表窗口。图 4.20 给出了使用缓冲管理模块的流程。

缓冲管理模块使用的缓冲区不同于捕获软件的缓冲区，这里使用的线性缓冲区。由于分析软件采用多线程技术，需要保证缓冲区数据的读写安全，为了避免在程序中大量的加锁与释放锁操作，程序自定义了一个线程安全的缓冲区类，即 ThreadSafeList 类。使用 Qt 的 Qlist 与 QReadWriteLock 将加锁与释放锁的细节封装到类中，保证读写操作的安全，我们使用 QReadWriteLock 可以减少锁定的时间，例如，如果没有启用写入锁，任何数量的线程都可以获得只读锁。此方法将返回指定位置的值，ThreadSafeList 有数个其他方法，结构与这个方法相同，不同在于使用锁的类型。

```
Type value(int i) const {
    QReadLocker locker(&m_Lock); //获取只读锁
    return m_List.value(i);
}
```

至此，我们可以实现一个线程安全的缓冲区，这种数据结构意味着分析数据包与搜索程序所用的线程可以把 ThreadSafeList 当作一个正常的数据结构来操作，不必担心锁定的事情。表 4.10 给出了所使用的 ThreadSafeList 定义。

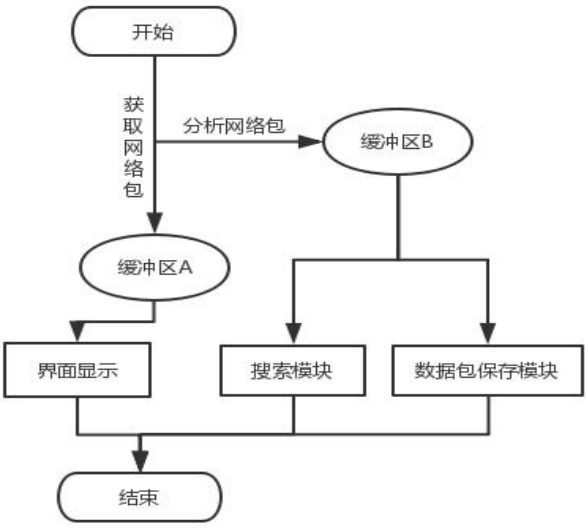


图 4.20 使用缓冲管理模块流程图

表 4.10 ThreadSafeList 定义

```
template<typename Type>
class ThreadSafeList    {
public:
      Type value(int i) const;                          /**< 下标访问元素 */
      bool isEmpty() const;
      bool contains(const Type &value) const;
      void clear();
      void append(const Type &value);                   /**< 尾部添加元素 */
      void append(const QList<Type> &value);            /**< 尾部添加元素 */
      Type takeFist();                                  /**< 取出元素，并释放内存 */
   private:
      mutable QReadWriteLock m_Lock;                    /**< 读写锁 */
      QList<Type> m_List;                               /**< 用于缓冲区 */
};
```

4.3 辅助功能

4.3.1 辅助函数

分析软件需要做到跨平台使用，主要运行在 Linux 和 Windows 上，由于两个操

作系统对于分析软件需要的一些功能不能很好的兼容，在此需要自定义一些相关平台无关函数。接口主要是网络字节序与主机字节序之间的转换和将 Ipv4 字符串转化为整形，集中在 Util.h 文件中。相关接口定义如下表 4.11

表 4.11 辅助函数定义

```
/** 模拟 htonl 函数，本机字节序转网络字节序 */
inline unsigned long int u_htonl(unsigned long int h)
/** 模拟 ntohl 函数，网络字节序转本机字节序 */
inline unsigned long int u_ntohl(unsigned long int n)
/** 模拟 htons 函数，本机字节序转网络字节序 */
inline unsigned short int u_htons(unsigned short int h)
/** 模拟 ntohs 函数，网络字节序转本机字节序 */
inline unsigned short int u_ntohs(unsigned short int n)
/** 将 Ipv4 字符串转化为 Int */
inline int ipv4StrToInt(const char *ipAddress)
```

4.3.2 样式定制

样式定制是为网络分析软件风格提供服务，为软件样式提供方案。目前只定义相关的配色方案与字体，当分析软件需要更多的定制服务时候，可以在 StyleUtils 中进行扩展。如表 4.12.

表 4.12 样式定制

```
class StyleUtils {
public:
    static StyleUtils instance() {
        StyleUtils instance;
        return instance;
    }
    //获取字体路径
    QString getFontPath(FontType fontType);
    //用户自定义字体
    QFont userDefinedFont(QString fontPath, int size = 11);
    //根据协议类型转化为相关的颜色字符串
    QString protoColorHex(pcpp::ProtocolType protocolType);
};
```

4.3.3 调试支持

调试对于分析软件系统开发非常重要，是开发信息的记录组件，调试支持是为了快速定位故障，方便维护系统。Qt 中自带有 QDebug 调试类，为了更好的服务系统，选择自定义调试类。调试类的定义如表 4.13。

表 4.13 调试类定义

```
class LogUtil {
private:
    LogUtil() {}
    enum _LogLevel_{
        _VERBOSE_, _DEBUG_, _INFO_, _WARN_, _ERROR_, _NOLOG_
    };
    static const int LOG_LEVEL_ = _VERBOSE_;
public:
    static void v(QString tag, QString msg);
    static void d(QString tag, QString msg);
    static void i(QString tag, QString msg);
    static void w(QString tag, QString msg);
    static void e(QString tag, QString msg);
};
```

调试类有以下特点：(1)当 LOG_LEVEL_设置为_NOLOG_等级时，调试支持被关闭，系统不会打印任何相关的调试信息。(2)当 LOG_LEVEL_设置为除了_NOLOG_以外等级时候，调试支持被打开，系统会根据等级需求打印 verbose, debug, info, warn, error 调试信息，例如当设置等级是_DEBUG_，系统会打印 verbose 和 debug 调试信息。

4.4 本章小结

本章主要介绍了网络分析软件的设计与实现，首先介绍了该软件的整体结构，接下来详细介绍了各个功能模块的设计与实现，最好讨论了实现系统需要的辅助功能。分析软件采用面向对象的思想设计核心功能模块，保证软件的维护性与可扩展性，本文给出相关的类图与流程图。

第五章 系统测试

前面章节我们已经分别讨论了网络捕获软件与网络分析软件的设计与实现，为了更好的理解本系统的功能与特点，本章我们将对系统从功能与性能两个方面进行测试。

5.1 测试环境

表 5.1 给出了本系统的测试环境，我们用以太网集线器将主机和终端通过相连接。主机端运行网络报文捕获和分析工具，终端运行网络捕获软件，为此我们使用已有的三台电脑搭建局域网进行测试，一台 VxWorks 终端，一台 Window7 主机，一台 Ubuntu 16.04 主机。

表 5.1 测试环境

系统类别	测试环境
服务器	VxWorks 5.5
客户端	Windows XP/7/8; Ubuntu 16.04

5.2 功能测试

5.2.1 测试方法制定

系统本身是一个基于 C/S 架构的监控设备，因此测试系统需要证明整个系统的功能，在此针对本系统进行了如下的测试：

(1)使用搭建好的网络环境中的所有 PC 机对终端进行 ping 包操作并且使用网络环境中的 PC 机与终端建立 SSH 连接，并通过彼此间传输文件。目的在于测试服务器可以正常捕获网络数据包以及客户端可以正常解析网络数据包，并执行相关操作。

(2)使用网络环境中的所有 PC 机对服终端进行 ping 大包操作并执行 4 小时，并保存文件，目的在于测试服务器与客户端承载能力、串口传输速度以及保存功能。

(3)读取本地文件，并在读取本地文件与实时捕获不断切换，目的在于测试客户端读取本地文件功能以及切换不同功能。

5.2.2 测试结果与分析

5.2.2.1 启动

捕获软件运行在 VxWorks 上，是控制台应用程序，当启动后会提示用户选择相关操作。当用户选择操作后，终端会执行相关的初始化操作，等待与主机交互。如图 5.1 所示。

打开网络分析软件，初始化相关配置，等待与终端交互。

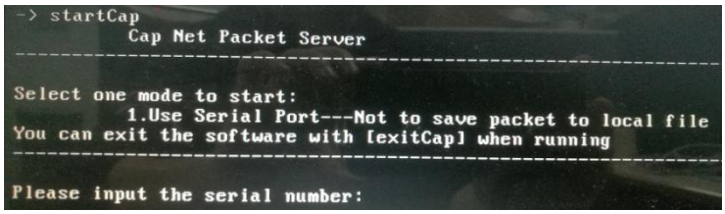


图 5.1 捕获软件加载完成

5.2.2.2 测试结果

1. 测试一结果

此次测试内容主要包括：①测试终端捕获数据包功能；②测试分析软件解析数据包功能；③测试终端与主机端捕获控制通信功能；④测试网络分析软件搜索功能。

(1)测试终端捕获数据包功能：图 5.2 给出了终端在捕获网络数据包时的打印信息。

(2)测试网络分析软件解析数据包功能：图 5.3 给出了分组列表的窗口，当点击任一条目时候，会在分组详情与分组字节窗口展开相关信。以条目 4 为例，点击后分组详情如图 5.4，是以分层协议树形显示该数据包的协议内容，分组字节如图

5.5，以十六进制形式显示该数据包。

图 5.2 服务器系统抓包信息

当点击分层协议树的一行信息时，分组字节会高亮显示对应的字节区间；同样当点击分组字节，分层协议树会根据点击字节区间高亮显示当行信息。

No	Time	Source	Destination	Protocol	Size	Info.
1	下午2:09	60:e…2:52	ff:ff:ff:ff:ff:ff	ARP	60	ARP Layer, ARP …6.55.55 ? Tell
2	下午2:09	60:4…1:0b	ff:ff:ff:ff:ff:ff	ARP	60	ARP Layer, ARP ….55.213 ? Tell
3	下午2:09	fe80…e799	ff02::c	UDP	208	UDP Layer(User …rt: 50287, Dst
4	下午2:09	172.…5.88	64.233.162.84	TCP	74	TCP Layer(User …YN], Src port:
5	下午2:09	30:b…2:b9	ff:ff:ff:ff:ff:ff	Ethernet	60	Ethernet II Lay…0b:e2:b9, Dst:
6	下午2:09	00:0…8:06	01:80:c2:00:00:00	Ethernet	60	Ethernet II Lay…f6:c8:06, Dst:
7	下午2:09	60:4…1:0b	ff:ff:ff:ff:ff:ff	ARP	60	ARP Layer, ARP ….55.213 ? Tell
8	下午2:09	30:b…2:b9	ff:ff:ff:ff:ff:ff	Ethernet	60	Ethernet II Lay…0b:e2:b9, Dst:
9	下午2:09	60:4…1:0b	ff:ff:ff:ff:ff:ff	ARP	60	ARP Layer, ARP ….55.213 ? Tell
10	下午2:09	30:b…2:b9	ff:ff:ff:ff:ff:ff	Ethernet	60	Ethernet II Lay…0b:e2:b9, Dst:
11	下午2:09	00:0…8:06	01:80:c2:00:00:00	Ethernet	60	Ethernet II Lay…f6:c8:06, Dst:
12	下午2:09	60:4…1:0b	ff:ff:ff:ff:ff:ff	ARP	60	ARP Layer, ARP ….55.213 ? Tell
13	下午2:09	30:b…2:b9	ff:ff:ff:ff:ff:ff	Ethernet	60	Ethernet II Lay…0b:e2:b9, Dst:

图 5.3 分组列表窗口

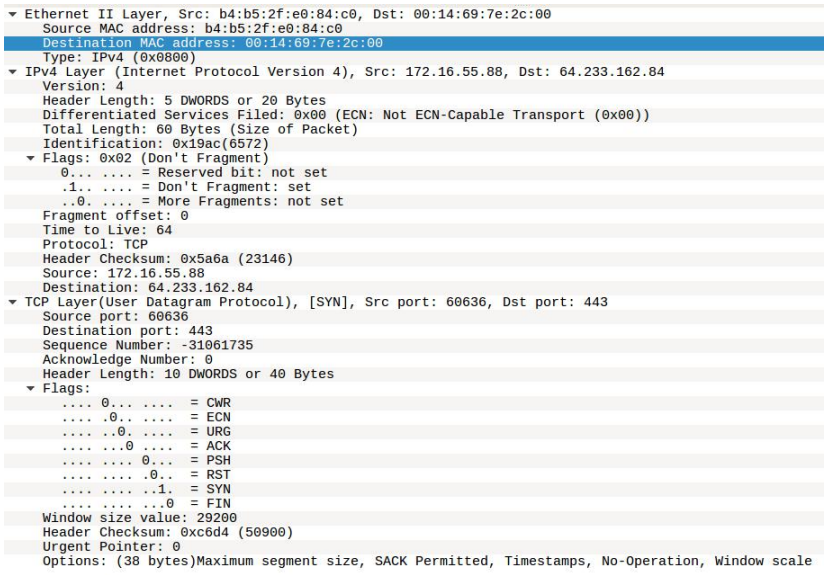


图 5.4 分组列表窗口

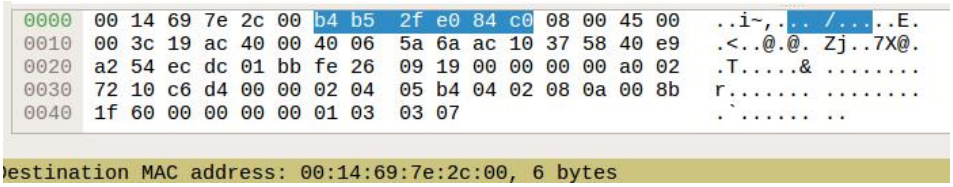


图 5.5 分组列表窗口

(4)测试终端与主机端捕获控制通信功能：图 5.6 给出了在主机端进行获取选择网络接口、开始捕获、停止捕获、清除捕获、关闭终端软件以及过滤操作时，终端作出的响应。

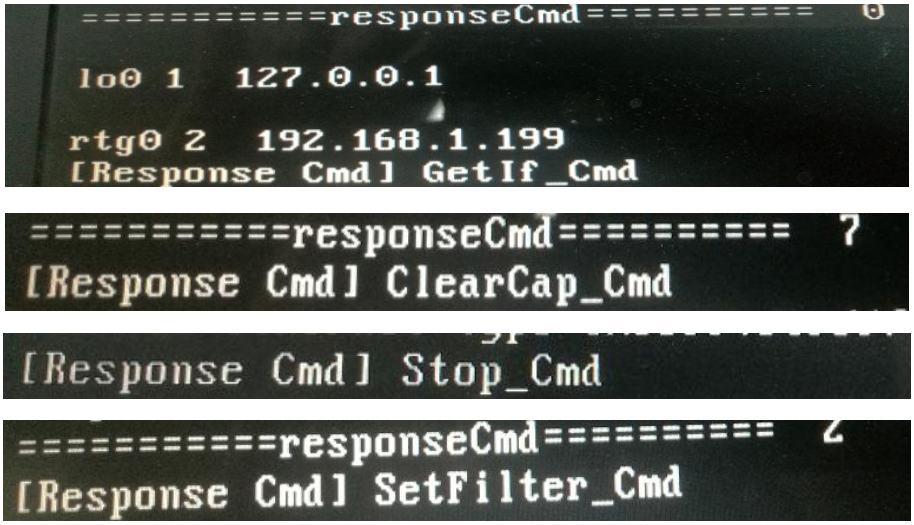


图 5.6 服务器通信打印信息

(5)测试网络分析软件搜索功能：用户在搜索输入框中输入搜索条件，点击搜索后的结果，如图 5.7 所示。

从四项测试的结果得出，系统的捕获、分析数据包功能正常，搜索功能正常，系统运行良好。

2. 测试二结果

此次测试系统捕获分析 ping 大包 4 小时，保存数据包正常，测试结果符合预期。系统运行良好。

3. 测试三结果

此次测试分别读取表 5.2 中的文件，对应完成时间如表 5.2 所示。

表 5.2 读取 pcap 文件时间

文件名称	文件大小	数据条目	完成时间
htttp.pcap	15.0KB	78	0.022s
http.pcap	7.6MB	7506	0.301s
https.pcap	18.3MB	17654	0.707s
icmp.pcap	25.7MB	109719	3.618s
big.pcap	75.7MB	222010	6.948s
big1.pcap	92.MB	85247	3.278s

由表 5.2 可以发现，在读取 222010 条目所需要的时间为 6.948s，满足要求。读取文件与实时捕获功能不断切换，可以运行正常。测试结果说明系统运行良好。

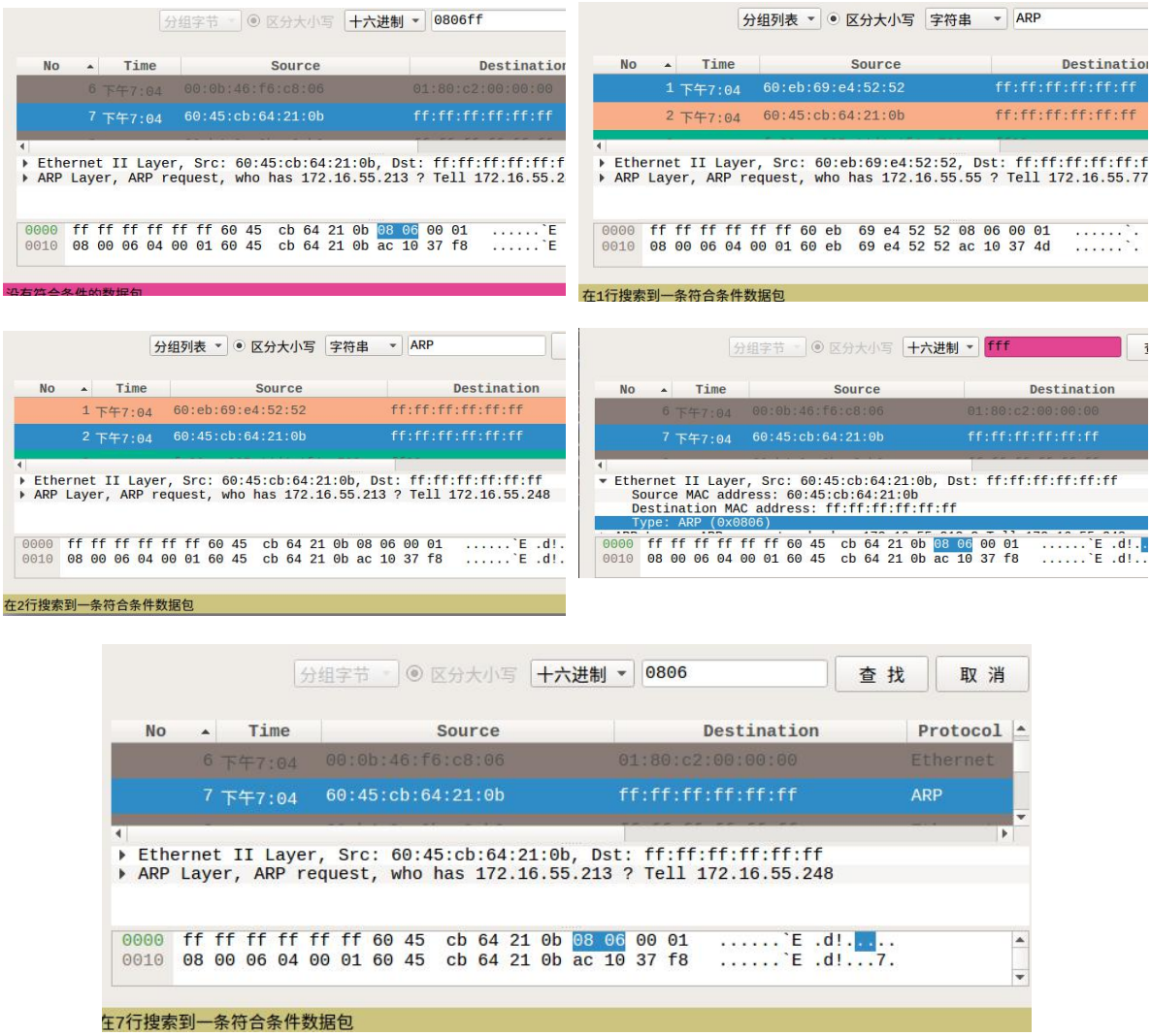


图 5.7 搜索测试结果

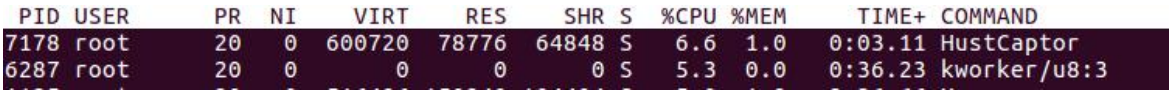
5.3 性能测试

本小节针对系统的性能进行测试，表 5.3 给出了系统的主要性能的测试结果。

表 5.3 性能测试

序号	测试内容	操作	结果	结论
2	文件读取速度	读取 222010 条目	11.040s	满足需求
3	文件保存速度	保存 222010 条目	0.226s	满足需求
4	数据搜索速度	搜索 222010 条目的协议树，搜索条件不出现在协议树	11.040s	满足需求

图 5.8.A 和图 5.8.B 分别给出了主机端系统运行在 Windows 7 和 Ubuntu 16.04 实时捕获时内存和 CPU 使用情况。测试结果表明本系统占用内存小，CPU 使用率占用低，并能满足用户的性能要求。



A

B

图 5.8 内存占用情况

5.4 本章小结

本章主要针对系统进行了功能测试与性能测试，通过测试结果可以看出本系统满足监控网络状态的需求。

第六章 总　结

本文主要研究了基于 VxWorks 系统的网络抓包工具设计与实现，采用 C/S 架构实现网络数据包的分析与存储。本文首先介绍了实现系统的关键技术，包括对网络抓包原理的介绍；阐述了数据包过滤的原理，通过原理自定义简化版过滤器；介绍了系统分析所需要的网络协议，讨论了解析网络数据包的思路。在系统设计与实现部分，本文分别给出了网络包捕获软件与网络包分析软件的整体架构，并对架构中每层的功能模块的设计思路与代码实现进行了讨论。最后本文对课题所设计的系统进行了功能测试与性能测试，分析相关测试结果。本文的主要成果与总结如下：

1. 通过分析嗅探器的工作原理、功能需求以及使用的关键技术，并结合 VxWorks 的网络架构，提出基于 VxWorks 系统的网络抓包分析系统的整体设计方案。

2. 在 VxWorks 平台通过使用 MUX 接口实现服务器网络数据包捕获功能，并通过串口通信将网络数据包推送到主机端；网络包分析软件实现 Windows 与 Linux 的跨平台使用，实现对网络数据包实时进行协议分析、读取本地 pcap 文件解析协议以及数据包存储的功能。

3. 通过对系统的功能测试与性能测试验证系统运行状态。

本文所设计的系统运行良好，可以很好的解析网络数据包，但还存在以下几点有待改进：

1.网络协议解析只做到对传输层之前的层次（包括传输层）的解析，并没有对应用层进行进一步解析，需要进一步研究。

2.本文设计的系统只是完成了解析数据包的核心功能，没有做流量统计，需要进一步研究。

3.本文设计的系统读取保存本地文件，只局限于 pcap 格式的文件，对于其他格式尚不支持；同时对大文件保存没有做分片处理，需要进一步研究。

致 谢

光阴似箭，不知不觉中我快要度过宝贵的两年研究生时光。这两年的研究生时光是我成长和收获比较大的两年。两年来，我悉心听取导师的教诲，与同学共同成长。所以说我的每一份成长都和关心帮助我的人分不开。我十分感谢这两年来导师对我的指导和同学们对我的关心，他们的存在，让我的生活更加精彩，学习更加顺利。

本论文的工作是在周正勇导师的悉心指导下完成的，周老师自己丰富的项目经验，渊博的知识让自己学到很多。整个过程中期间自己的论文有不少问题，周老师针对性的提出修改意见，帮自己更好的完善论文。其次胡贯荣教授对我启发颇多，针对我的研究工作和论文都提出了许多珍贵的意见，在此表示由衷的感谢。

最后感谢养育我的父母和培养我的华中科技大学。

参考文献

[1] 基于 HTTP 协议的 Web 访问监测系统. 张一. 电子科技大学，2012.05

[2] 基于 WinpCap 的网络流量统计分析系统设计与实现. 于天予. 河北科技大学，2013.12

[3] 基于 C_S 结构的网络服务质量监控系统的设计与实现. 杨晗. 电子科技大学， 2012.03

[4] 基于 VxWorks 平台的网络通讯监控技术的应用研究. 蔺妍. 哈尔滨工程大学. 2007.01

[5] 基于 VxWorks 导航系统网络技术的应用研究. 贾桂芬. 哈尔滨工程大学.2006.01

[6] 关于 VxWorks 系统及其现状的评述. 周勇军. 中国人民解放军第 5720 工厂，安徽 芜湖 241007

[7] 一种网络抓包软件的实现. 赵敏，陈志平，张巨勇. 杭州电子科技大学，机械工程学院，浙江 杭州 310018

[8] 一种网络协议抓包数据引擎优化控制机制的研究. 徐琳. 信息技术与信息化研究与探讨，2015:11-218

[9] T. Mc Gregor, H. Braun, J. Brown. The NLANR Network Analysis Infrastructure .IEEE Communication Magazine, 2000, 38(5): 122-128

[10] Z. Wei. Stability Analysis of Networked Control Systems .Cleveland, USA: Case Western Reserve University, 2001

[11] 网络流量实时监测系统设计与实现. 穆杰. 河北科技大学. 2013.05

[12] The WinPcap Team, The Chinese version of the Win Pcap 4.0.1 manual, http://www.coffeecat.net.cn/winpcap/html/index.html, 2008.03

[13] 基于 WinPcap 库的网络封包嗅探器实现. 康晓东，裴昌幸. 电子科技，2005，5(2): 34-50

[14] 基于 LINUX 的网络数据抓取识别系统的设计与实现. 吕鹏飞. 青岛理工大学. 2014.06

[15] 高速大数据量的网络监视与数据包捕获解析技术研究. 池小强. 华中师范大学，2015.05

[16] M.S.SENDIL, N.N.JAN. Analyzing the Peer to Peer Traffic Aggregation Using an Optimized Method. Journal of Computer Science. 2009, 5(10): 738-742

[17] 网络数据包捕获及分析. 卢建华，蒋明，陈淑芳. 网络安全技术与应用，2009.2

[18] W. Richard Stevens,Bill Fenner,Andrew M. Rudoff.Unix 网络编程 卷 1：套接字联网 API（第三版）.杨继张译. 人民邮电出版社.2017.580-645

[19] 基于网络抓包的数据库即时备份方法. 傅瑞军. 计算机工程与设计，2010,31(24)

[20] S. BOCCALETTI, V. LATORA, Y. MORENO, etal. Complex Networks: Structure and Dynamics. Physics Reports, 2006, 424(4/5): 175-308

[21] Linux 网络编程：原始套接字的魔力（下）. https://www.cnblogs.com/masterpanda/p/5700481.html. 2014.04

[22] VxWorks END 网络驱动软件的开发与实现. 寇云林，陈怀民，段晓军，陈伟. 计算机测量与控制，2009.17(1)

[23] 基于 VxWorks 的设备驱动和网络通信. [D]西安科技大学.2006

[24] VxWorks 网络协议栈的 MUX 接口. 张晓华，李智涛，徐钊. 计算机与嵌入式系统应用. 2002.5

[25] 基于 VxWorks 的网卡驱动设计. 王景刚，邓如玉，杨小平. 中国新技术新产品.2010.NO.14

[26] VxWoks NetWork Programmer's Guide. WindRiver System, Inc, 1998

[27] Wind River. VxWorks 网络程序员指南（第一版）. 王金刚，宫霄霖，熊辉译. 清华大学出版社. 2003:155-162, 193-197

[28] VxWorks 网络协议栈的 MUX 接口. http://blog.sina.com.cn/s/blog_a44175a901015yj7.html. 2012.05

[29] NAT 原理及在 VxWorks 上的实现. https://wenku.baidu.com/view/faa2cd61a6c30c2258019e25.html. 2014.10

[30] BPF 数据包过滤器的分析与研究. 王景琪，左明，张功杰. 计算机工程与设计,2005.09 26-9

[31] 数据包过滤技术的分析与讨论. 曾志高，谭骏珊，易胜秋. 株洲师范高等专科学校学报. 第 11 卷 第五期. 2006.10

[32] A.HARI, S.SURI, G.PARULKAR. Detecting and Resolving Packet Filter Conflicts. Califorbia: IEEE Press, 2000, 1203-1212

[33] L. DEGIOANMI. Development of An Architecture for Packet Capture and Network Traffic Analysis. Turin, Italy: Politecnico Di Torino, 2000

[34] Jenny J. He,Dimitra Simeonidou. Flow Routing and its Performance Analysis in Optical IP Networks.Photonic Network Communications.2001.

[35] pcap 文件格式分析. https://www.cnblogs.com/2017Crown/p/7162303.html. 2017.07

[36] Linux 环境下基于 Raw socket 技术的多线程追击抓包法. 喻曦，文静华. 计算机光盘软件与应用 2012-17

[37] TCP/IP 协议及网络编程技术. 罗军舟，黎波涛，杨明，吴俊. 北京：清华大学出版社，2006

[38] Uyless Black. TCP/IP and Related Protocols(Third Edition)[M]. Mc Graw-Hill Companies, 1999, 03.

[39] Behrouz A.Forouzan. TCP/IP Protocol Suite 4 edition[M]. Mc Graw-Hill Science/Engineering/Math. 2011,01.

[40] Jon C. Snader. Effective TCP/IP Programming: 44 Tips to Improve Your Network Programs[M]. Addison-Wesley

Professional, 2011, 04.

[41] Jacobson V, Leres C, Mc Canne S, et al. Berkeley Laboratory[J]. Berkeley, CA Initial public release, 1994,(6):121-124.

[42] M.Yu, L.Wang, T.G.Chu, G.M.Xie. Stabilization of Networked Control Systems with Data Packet Dropout and Network Delays via Switching System Approach .Proceedings of the 43rd IEEE Conference on Decision and Control. Bahamas. 2004: 3539-3544