



华南理工大学  
South China University of Technology

# 《计算机网络》 课程设计报告书

题目： 网络流量分析软件

学 院 计算机科学与工程学院

专 业 信息安全

学生姓名 颜徐柳

学生学号 201630610779

指导教师 袁华

课程编号 145165

课程学分 2.0

起始日期 2018-10-26

	第 14 组				
	姓名	学号	角色	主要分工	完成工作量
	颜徐柳	201630610779	组长、系统设计、程序员	整体设计、细节实现、整合联调	100%
教师评语	<div>教师签名:</div> <div>日期:</div>				
成绩评定					
备注					

# 网络流量分析软件

## 一、选题背景

一台计算机，不管是个人电脑还是服务器，每时每刻都会有大量的网络数据包通过网卡发出或接收，对一般用户这些数据是透明的，因为网络传输是在后台默默进行的，一般用户很难对这些数据进行感知和分析。

如果能设计一种软件，能够实时监控这些数据，并且能根据用户需求图形化的展示给用户，就能大大提高网络传输的可见性和可分析性。

网络上已经有如 Wireshark 之类的抓包工具，但是它们有些对操作系统和网络环境有特定要求，有些操作难度过于复杂。尤其值得注意的是对于本身没有图形界面的操作系统或者目标机位于远程位置，一般只能在目标机上获取大量数据保存为文件，下载到具有图形界面的操作系统进行分析，便携性较差，且难以做到实时监控。

针对这样的问题，本次课程设计希望能设计一种能够跨平台的网络流量实时监控、分析软件，使用 Java 语言进行后台开发，而 UI 展示则使用 web 客户端的形式。之所以选择 web 方式展示，主要是针对没有图形界面的操作系统或目标机处于远程位置的情况，可以通过互联网将数据发至用户浏览器进行展示和分析。当然，考虑到网络环境的复杂性，任务中必须要考虑数据传输的安全性和系统运行的稳定性。

## 二、方案论证(设计理念)

### 程序后台（数据模块）

#### (1) 数据抓取：pcap4j 库

Pcap4J 是一款十分优秀的网络报文抓取第三方 Java 库，要自己编写报文抓取代码是十分困难的，但是在这个第三方库的支持下，对网络报文的抓取和处理将会容易，大大降低了开发难度。尽管如此，这一部分也将是程序的核心，报文的获取、转换、存储、读取都将在这个部分完成。

#### (2) Web 框架：jetty 库

本程序虽然以 web 作为主要展示端和控制端，但 web 后台并不是核心所在，所以只需要简单轻量的内嵌式 web 后台就足够了，jetty 便是非常适合的库，能比较容易的实现 web 后台。其中文件上传下载使用 servlet，其他数据交互使用 WebSocket。

#### (3) WebSocket 交互逻辑

WebSocket 属于 web 的一部分，是前端获取数据的主要来源，它将是整

个程序中最复杂的模块之一，除了负责后端报文的转发和接收前端控制命令外，还需要检测用户的合法性，以及将用户的合法请求信息转换为可识别的控制信息。

#### (4) 认证与加密

程序启动时生成随机 256 位密钥，浏览器的认证与加密将依赖这个密钥。浏览器获取密钥之后，建立 WebSocket 连接和调用后台 API 都将使用这个密钥进行认证和加密。

为确保密钥不会在传输过程中被窃听，使用浏览器地址#hash 作为密钥的来源，这样密钥将不会通过浏览器传输，自然就没有被网络窃听的可能。

值得注意的是，在设计的初期是使用 HTTPS 协议进行认证和加密的，但在开放过程中意识到，SSL 只提供传输的保密性，却并不会提供用户认证的功能，虽然能够较好地防止信息被窃听，却无法保证用户是合法用户。

所以后期加上了 AES 加密，将与 HTTPS 同时存在，并在程序入口处加上了灵活的控制，文档后面会有详细的说明。

### 程序前端（展示模块）

- (1) 展示页面：单页面入口，使用 React 实现，进入 React 前统一从 URL 中获取密钥并保存在 sessionStorage 中（这样刷新浏览器也能继续完成认证，且在浏览器或标签关闭时失效）
- (2) 控制接口：WebSocket 接口
- (3) 数据获取接口：WebSocket 接口
- (4) 文件接口：上传与下载均使用 servlet

### 数据分析与存储

- (1) 文件存取：原生 tcpdump 标准 pcap 文件

Pcap4J 库可以直接存取 tcpdump 标准 pcap 文件。

- (2) 大量数据与性能：分页与动态读取

当数据量很大时，在一个页面显示会严重影响性能，甚至程序崩溃，必须做好分页。然而分页也需要一定的技巧，由于 pcap 文件本身是流式的，而且没有索引，所以很难从一个大的 pcap 文件中分页，只能采用多个文件的方式，借用操作系统的文件管理来辅助分页，规定一个文件最大存多少个包，这样就解决了后台分页问题。

还有就是前端的数据，除了能实时监控数据包之外，还应能够获取历史数据，另外当数据包越积越多，前端的内存消耗会越来越大，即使前端分页也无法缓解内存负荷，必须在合适的时间舍弃旧的数据包，需要的时候再向

后台请求获取。

### (3) 数据分析：独立模块

本程序的数据分析主要有两种形式：一是从开始捕获数据包开始动态统计数据包类型，加以分析；二是从一个 pcap 文件中读取并分析。

这两种分析方式的分析过程是一样的，但是数据来源不同，如果能将分析程序独立出来，就能从不同的来源获取数据后使用统一的接口。另外，动态分析将是一直存在与全局空间的，只有一个，而文件分析则对应每一个文件，可以有多个。

从连接上看，两种连接都应该支持多客户端，不同之处只是实时捕获是多个客户端控制同一个捕获程序，而文件管理部分则是每个客户端对应一个文件管理程序。

## 三、过程论述

### 1. 总体架构与引用第三方库

按运行环境分为前端和后台，按程序模块分为实时捕获和文件管理，按前后端连接方式分为 servlet 和 websocket。

前端使用 React<sup>[1]</sup>脚手架<sup>[2]</sup>构建，使用了 Ant Design<sup>[3]</sup>来组建基本的页面元素，图表绘制使用了 Ant Design Pro<sup>[4]</sup>。另外，前端的加解密使用了 crypto-js<sup>[5]</sup>。

后台使用 jetty<sup>[6]</sup>作为 web 库与前端交互，使用 pcap4j<sup>[7]</sup>作为数据报文捕获与处理工具，使用 Gson<sup>[8]</sup>作为 JSON 格式处理工具，使用 slf4j<sup>[9]</sup>作为日志系统，另外还用到了 Apache 的开源库 commons-fileupload<sup>[10]</sup>来处理文件的上传。

程序使用 gradle<sup>[11]</sup>构建，使用 bootJar<sup>[12]</sup>组织依赖并打包成 jar。

### 2. 定义前后端交互接口

#### 客户端 -> 服务器

有如下三种类型的消息：

#### ① action=hello 握手消息

每当新的连接建立时，会首先有三次握手，成功握手并建立互信关系后才能进行后续消息传输。

第一次握手：由服务器发给客户端，action=hello 并附带随机字符串 seed 和用 AES 密钥加密 seed 后的 cipher。客户端可以用本地密钥对 seed 进行加密并与 cipher 进行比较来确认本地的密钥是否正确，如正确则进行第二次握手，否则断开连接并提示用户密钥错误。

第二次握手：由客户端发给服务器，`action=hello` 并附带用客户端密钥对第一次握手时收到的 `seed` 的哈希值进行加密后的密文，同时附带客户端身份（实时捕获还是文件管理）。服务器同样用服务器密钥对 `seed` 的哈希进行加密与收到的密文比较，并进行第三次握手。

第三次握手：由服务器发给客户端，`action=verify` 并附带验证结果。如果第二次握手中验证通过则发送成功通知，否则发送失败通知。验证失败后客户端会主动关闭连接。

## ② `action=ping` 存活确认消息

当接收到客户端的 `ping` 消息后，服务器会回复一个 `action=pong` 的空消息来确认服务器的存活状态。

## ③ `action=command` 控制命令消息

控制消息有三类，捕获模块消息、文件模块消息、通用消息。

捕获模块消息：

- ✓ `list_interfaces`：列出网卡列表
- ✓ `capture_status`：获取当前的捕获状态
- ✓ `start_capture`：开始实时捕获
- ✓ `stop_capture`：结束捕获

文件模块消息：

- ✓ `list_files`：列出 `dump` 目录的文件列表，文件可能来自于实时捕获，也可能来自于用户上传，也可以直接把文件拷贝到对应目录。
- ✓ `delete_files`：删除指定的文件，附带请求删除的文件列表，文件必须是 `.pcap` 文件且必须在 `dump` 目录下。
- ✓ `bind_file`：打开文件获取数据，附带请求文件名和过滤器规则，文件必须是 `.pcap` 文件且必须在 `dump` 目录下，过滤器规则必须符合 BPF 过滤规则<sup>[13]</sup>（可参考 BPF 语法的简单介绍<sup>[14]</sup>）。
- ✓ `request_packets`：请求打开的文件的数据报文，附带 `start` 和 `number` 参数，`start` 是第一个希望获取的数据报文的位置，`number` 是希望获取的个数。
- ✓ `unbind_file`：关闭打开的文件。

通用消息只有一个：`statistics`，用于获取当前的统计数据，没有附带参数。

## 服务器 -> 客户端

有如下五种类型的状态消息：

- ✓ `action=pong`：服务器对 `ping` 消息的响应。

- ✓ `action=hello`: 第一次握手消息。
- ✓ `action=verify`: 第三次握手消息。
- ✓ `action=info`: 附带 `info` 参数, 向用户提供提示信息。
- ✓ `action=error`: 附带 `error` 参数, 向用户提供错误信息。

有如下五种类型的数据消息:

- ✓ `action=status`: 附带 `status` 参数, 向用户提供当前的捕获状态。
- ✓ `action=interfaces`: 附带 `interfaces` 参数, 向用户提供网卡接口信息。
- ✓ `action=packet`: 附带 `packet` 参数和报文时间戳 `time` 参数, 向用户提供实时捕获的数据报文。
- ✓ `action=statistics`: 附带 `statistics` 参数和更新时间 `time` 参数, 向用户提供数据报文的统计信息。
- ✓ `action=file_list`: 附带 `data` 参数, 向用户提供 `dump` 目录下的 `.pcap` 文件列表, 除了用户主动请求获取文件列表外, 当用户上传或删除文件成功后也会自动触发该类型消息。

### 其他接口

- ✓ WebSocket 地址: `/websocket`
- ✓ 上传接口: `/upload`, POST
- ✓ 下载接口: `/download/{file_name}?auth={encrypted}`, GET, `{encrypted}` 参数是对请求 `{file_name}` 的 AES 加密

### 3. 后台: 定义前后端交互的 Packet 模型

首先, `Pcap4J` 是有自带的 `Packet` 类库的, 并且包装的很好, 但毕竟是一个库, 提供的信息非常全面, 却有太多的冗余信息, 如果不加处理的直接传给前端, 会浪费大量的网络资源, 而且前端的处理也会十分不便, 所以我对其进行了一定的重新封装, 但主体数据依然来自于它的类库。

在早期借鉴 `Pcap4J` 的设计理念, 用了一个统一的 `SPacket` 接口, 所有的实现类都实现这个接口。但是后来发现, 我的目的只是为了实现一个数据容器而不是数据接口, 我只需要把数据装进去发给前端而不需要对这些数据进行任何操作, 所以后期撤销了接口式的封装而使用了数据式的封装。图 3-3-1 提供了顶层 `frame` 的封装代码作为示例。

```

public class SFrame {
    private String type;
    private SEthernetPacket ethernet;
    private int length;

    public void load(Packet packet) {
        length = packet.length();
        if (packet instanceof EthernetPacket) {
            EthernetPacket ethernetPacket = (EthernetPacket) packet;
            type = "ethernet";
            ethernet = new SEthernetPacket();
            ethernet.load(ethernetPacket);
        } else
            throw new UnknownPacketException();
    }

    public String getType() { return type; }
    public SEthernetPacket getEthernet() { return ethernet; }
    public int getLength() { return length; }
}

```

图 3-3-1 顶层 frame 的封装代码

其次，通过观察，发现所有的数据包的最上层都是以太网协议，虽然无法保证 100%，但目前观测到的数据包都是如此，所以程序只分析以太网协议，就如上图 3-3-1 所示，如果捕获到非以太网协议的数据报文，将抛出未知数据报文异常（这是一个自定义异常）。同样，其实本次实验中感兴趣的只有那几类报文，对于不感兴趣的报文，将会做丢弃处理（文件依然会记录下无法识别的报文，用户可以下载捕获文件后用其他工具如 WireShark 查看）。

最后特别说明一下 IP 报文和 ICMP 报文，由于 IPv6 的出现，IP 报文分为 IPv4 和 IPv6，ICMP 报文也分为 v4 和 v6，而 ICMP 报文本身就有非常多的类型，所以最后的种类非常繁多。

对于 IP 报文，虽然 v4 和 v6 的头部不同，但是其负载（IPv6 以最后一个负载作为实际负载）是完全相同的，故抽取出来一个抽象类避免重复代码。同时通过观察发现，IPv6 的绝大部分报文的下一个头其实就是最终的负载，所以为简化操作，将忽略解析下一个头不是感兴趣的负载的 IPv6 报文。

对于 ICMP 报文，虽然 Pcap4J 对其进行了大量的分类，但其实本程序并不需要分得那么细，故只分了 ICMPv4 和 ICMPv6 两类。

#### 4. 后台：定义可能用到的辅助类

前期只定义了生成随机字符串的辅助类，如图 3-3-2 所示。



```

public class RandomStringHelper {
    private static final Random random = new Random(System.currentTimeMillis());

    public static String randomString(int length) {
        String source = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789";
        StringBuilder builder = new StringBuilder();
        for(int i = 0; i < length; i++) {
            int number = random.nextInt(source.length());
            builder.append(source.charAt(number));
        }
        return builder.toString();
    }
}

```

图 3-3-2 生成随机字符串的辅助类

后期新增的辅助类主要有 AES 加解密类、命令行参数解析类和计算 MD5 的类，将在后面详细介绍。

## 5. 后台：完成实时捕获后台核心部分

首先，本程序中实时捕获模块使用单例模式进行，所有客户端共享同一个捕获实例，如图 3-3-3 所示。

```

// 单例模式
private CaptureHandler() { init(); }
private static CaptureHandler instance = new CaptureHandler();
public static CaptureHandler getInstance() { return instance; }

```

图 3-3-3 使用单例模式的 CaptureHandler 类

其次，实时捕获和文件管理两个模块的 websocket 连接使用的同一个类，因为它们之间有很多的共同之处，没有必要写太多的重复代码。所以这两个模块的 Handler 将具有相同的特征，实现了同一个接口，如图 3-3-4 所示。

```

public interface SourceHandler {
    void bindWebSocketHandler(WShandler handler);
    CaptureInformation getInformation();
}

```

图 3-3-4 两个模块的 Handler 实现的共同接口

图 3-3-5 和图 3-3-6 分别提供了核心处理代码和数据报文处理代码。在核心处理代码中，不会处理由本程序端口发出或接收的报文，因为如果本程序与前端交互过程中也会产生数据流量，如果不进行过滤的话，将会引起循环流量。在数据报文

处理过程中，因为加密需要耗费较多的时间，所以提前对报文进行加密再转发给所有的客户端。

```
// 开始循环获取报文
handle.loop( packetCount: -1, (PacketListener) packet -> {
    IpPacket ipPacket = packet.get(IpPacket.class);
    TcpPacket tcpPacket = packet.get(TcpPacket.class);
    if (ipPacket != null && tcpPacket != null) {
        InetAddress src_address = ipPacket.getHeader().getSrcAddr();
        int src_port = tcpPacket.getHeader().getSrcPort().value();
        InetAddress dst_address = ipPacket.getHeader().getDstAddr();
        int dst_port = tcpPacket.getHeader().getDstPort().value();
        // 过滤本应用发出的报文
        if (interface_ips.contains(src_address) &&
            (src_port == Application.http_port || src_port == Application.https_port))
            return;
        // 过滤本应用收到的报文
        if (interface_ips.contains(dst_address) &&
            (dst_port == Application.http_port || dst_port == Application.https_port))
            return;
    }
    handlePacket(packet);
});
```

图 3-3-5 核心处理代码

```
private void handlePacket(Packet packet) {
    // 解析
    SFrame frame;
    try {
        frame = new SFrame();
        frame.load(packet);
    } catch (UnknownPacketException e) {
        logger.debug("handler an unknown packet.");
        logger.debug(new Gson().toJson(packet));
        frame = null;
    }

    // 类型统计
    information.count(packet);

    // WebSocket
    if (frame != null) {
        String encryptedFrame = AescryptHelper.encrypt(new Gson().toJson(frame));
        for (WSHandler websocket : websockets) {
            if (websocket.isOpen())
                websocket.sendPacket(encryptedFrame);
        }
    }

    // 保存为文件
    try {
        if (dumper != null)
            dumper.dump(packet);
    } catch (NotOpenException e) {
        e.printStackTrace();
        logger.error(e.getMessage());
    }
}
```

图 3-3-6 数据报文处理代码

其他具体细节请查看源代码。

## 6. 后台：实现实时捕获模块的后端接口

后台接口只是处理简单的交互逻辑，将由对应 Handler 处理的结果转发给前端，并提供一些与前端交互的接口方法，具体可查看 WSHandler.java，这里不再赘述。

## 7. 前端：实现向接口发送控制命令和从接口获取实时数据

Socket.js 实现了与后台的 websocket 交互。PacketTree.jsx 提供了对报文内容的解析并按树的形式展示出来。

图 3-3-7 展示了配置 websocket 连接地址的代码。因为前端调试的时候使用了 React 脚手架提供的调试 server 和热更新功能，所以需要特别配置与后台的连接端口。

```
let ws_url = null;

if (process.env.NODE_ENV === "development") {
  ws_url = (window.location.protocol.indexOf("s") > -1 ? "wss://" : "ws://")
    + window.location.hostname + ":12611/websocket/";
  // ws_url = "wss://localhost:8002/websocket/";
}
else {
  ws_url = (window.location.protocol.indexOf("s") > -1 ? "wss://" : "ws://")
    + window.location.host + "/websocket/";
}
```

图 3-3-7 配置 websocket 连接地址

图 3-3-8 展示了配置 AES 参数的代码，具体加解密代码请查看源代码。

```
import CryptoJS from 'crypto-js';

let key = CryptoJS.enc.Hex.parse(window.key);
let iv = CryptoJS.enc.Hex.parse(window.key.substring(0, 32));

function resetKey(new_key) {
  window.key = new_key;
  key = CryptoJS.enc.Hex.parse(window.key);
  iv = CryptoJS.enc.Hex.parse(window.key.substring(0, 32));
  if (window.sessionStorage)
    window.sessionStorage.setItem("key", window.key);
}
```

图 3-3-8 配置 AES 参数

其他具体细节请查看源代码。

## 8. 后台：添加 HTTPS 支持

如果配置了 HTTPS 连接端口，将创建一个 HTTPS 的 Connector，如图 3-3-9 所示。

```

if (https_port > 0) {
    // SSL Context Factory
    SslContextFactory sslContextFactory = new SslContextFactory();
    sslContextFactory.setKeyStorePath(keystore_path);
    sslContextFactory.setKeyStorePassword(keystore_password);

    // HTTPS Configuration
    HttpConfiguration https_config = new HttpConfiguration();
    https_config.setSecureScheme("https");
    https_config.setSecurePort(https_port);
    https_config.addCustomizer(new SecureRequestCustomizer());

    // HTTPS Connector
    ServerConnector httpsConnector = new ServerConnector(server,
        new SslConnectionFactory(sslContextFactory, HttpVersion.HTTP_1_1.asString()),
        new HttpConnectionFactory(https_config));
    httpsConnector.setPort(https_port);
    server.addConnector(httpsConnector);
}

```

图 3-3-9 添加 HTTPS 支持

HTTPS 需要一个 keystore 数字证书，程序提供了默认的自己签名证书，默认密码是 traffic，有效期是 10 年，支持对 localhost 和 127.0.0.1 认证，必须首先将默认提供的 keystore.crt 安装到受信任的根证书列表中，图 3-3-10 展示了默认 keystore 的生成命令。

```

generate_cert.txt x
1 keytool -genkey -keyalg RSA -validity 3650 -keystore keystore -alias localhost -ext "san=dns:localhost,ip:127.0.0.1" -dname "CN=localhost, OU=xinsane, O=xinsane, L=Guangdong, ST=Guangzhou, C=CN" -storepass traffic -keypass traffic
2
3 keytool -importkeystore -srckeystore keystore -destkeystore keystore.p12 -srcstoretype JKS -deststoretype PKCS12 -srcstorepass traffic -deststorepass traffic
4
5 openssl pkcs12 -clcerts -nokeys -out keystore.crt -in keystore.p12 -passin pass:traffic

```

图 3-3-10 默认 keystore 的生成命令

## 9. 综合：添加 AES 加密传输

前端的 AES 配置前面已经介绍过了，这里简单说明一下后台的 AES 处理过程。

后台的 AES 加解密由一个辅助类 AESCryptHelper 完成，其中包括密钥的生成、加密和解密。如果指定了 no\_aes 特性，将不会生成密钥，加解密将直接以原文返回。图 3-3-11 提供了密钥生成的代码，密钥生成之后将同时以 16 进制形式输出到控制台和程序运行目录下的 key.txt 文件。

```

static {
    KeyGenerator generator = null;
    SecretKey secretKey = null;
    if (!Application.no_aes) {
        try {
            generator = KeyGenerator.getInstance("AES");
        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
            logger.error("can not generate key: " + e.getMessage());
            System.exit(status: -1);
        }
        generator.init(128);
        secretKey = generator.generateKey();
    }
    key = secretKey;
}

```

图 3-3-11 密钥生成代码

具体加解密细节不再赘述。

#### 10. 后台：添加解析命令行参数功能，灵活配置程序参数

ArgumentsResolver 辅助类提供了对命令行参数的简单解析，其中有三种类型的命令行参数，一是 option，以单个-开头并附带一个参数，一个是 feature，以双-开头且没有附带参数，最后剩下的参数作为普通参数返回。

图 3-3-12 展示了程序使用 ArgumentsResolver 解析的代码。这里省略了合法性验证的代码。

```

Map<String, String> validOptions = new HashMap<>();
validOptions.put("http", "specify the http listen port.");
validOptions.put("https", "specify the https listen port.");
validOptions.put("dump", "specify the pcap files directory. default ./dump/");
validOptions.put("slice", "specify how many packets per pcap file in file mode. default 500");
validOptions.put("keystore", "specify the path of SSL keystore file. default ./certs/keystore");
validOptions.put("keystore_pass", "specify the password of SSL keystore file. default traffic");
Map<String, String> validFeatures = new HashMap<>();
validFeatures.put("no_aes", "data will be transferred without encryption if --no_aes is set");
config = ArgumentsResolver.resolve(args, validOptions, validFeatures);

```

图 3-3-12 命令行参数的解析

#### 11. 前端：添加断线重连功能

当网络不稳定时，如果 websocket 异常断开连接，应当能合理的重新连接。部分实现代码如图 3-3-13 所示。

```

onClose = () => {
  if (this.ws_notification) {
    notification.error({
      message: "连接已断开",
      description: "与服务器的连接中断, 无法获取新数据",
      duration: null
    });
  }
  this.ws_notification = false;
  this.ws = null;
  setTimeout( handler: () => {
    this.initWebSocket( reconnect: true )
  }, timeout: 1000)
};

```

图 3-3-13 断线重连

## 12. 后台：添加代理识别，不向代理客户端发送实时数据

由于我的服务器只开放 22 和 80/443 端口，其他服务通过 80/443 进行反向代理，而程序本身只能识别并过滤掉自身端口的流量，这导致大量转发的数据报文被重复捕获到并继续转发，造成循环流量。故添加简单的代理识别功能，只要检测到请求头中含有 X-Forwarded-For 参数就不再向该客户端转发报文数据，但依然可以通过代理查看统计信息。

## 13. 前端：重构提取可重用部分，为实现文件管理模块做准备

前端可重用的部分主要有两块：统计图和数据表格。统计图被提取到了 PacketStatistics.jsx，数据表格被提取到了 PacketDataTable.jsx。

## 14. 综合：实现文件管理模块接口

由于前面已经做好了铺垫，所以这个模块的实现相对容易的多。

了解了 Pcap4J 对数据报文的读取方式后，立刻感觉到不能简单地使用。这是由于.pcap 的格式本身决定的，因为.pcap 文件不带索引，所以想要定位到某一个数据报文只能从头开始搜索直到找到需要的报文，而一次性将全部数据报文加载到内存中是不可能的，一个文件的大小动辄上 GB，不能粗暴的用内存来换效率。

最好的方法就是单独为.pcap 文件建立索引，但是由于时间、精力的关系并没有选择这种方式，而是使用了操作系统的文件系统作为辅助索引，具体做法就是，当用户请求加载某个文件时，从头到尾扫描文件一遍，并按照一定的数量，将文件拆分成等量数据报文数的多个小文件，从而实现简单的索引。这样一来，虽然定位某个数据包依然需要搜索，但搜索的范围就小得多，另外前端需要的数据通常是连续

的一块，只要前端的分页配置的恰当，后台的搜索代价瓶颈基本上就在于文件系统的检索速度了。

在实现文件下载时，发现浏览器下载文件时不显示下载进度条，即使我已经添加了 Content-Length 的响应头。最后查看前端收到的响应头才想起前期配置时使用了 gzip 对数据进行压缩处理，为了更好的用户体验，上传下载将不再使用 gzip 进行压缩。

15. 整合调试，打包发布

## 四、结果分析

### 运行结果及分析

1. 运行结果

程序分两个模块，分别是实时捕获和文件管理。其基本构成如图 4-1-1 所示。下面分模块分别介绍。这里统一介绍电脑端，手机端之后会专门介绍。



图 4-1-1 程序前端基本构成

### 实时捕获

密钥认证：如果无法从 URL 的 hash 获取密钥将要求用户输入 16 进制格式密钥，如图 4-1-1 所示。

选择网卡：认证成功后，将会列出安装在电脑上的网卡，用户可以选择一个网卡和筛选条件来开始捕获数据包，如图 4-1-2 和图 4-1-3 所示。

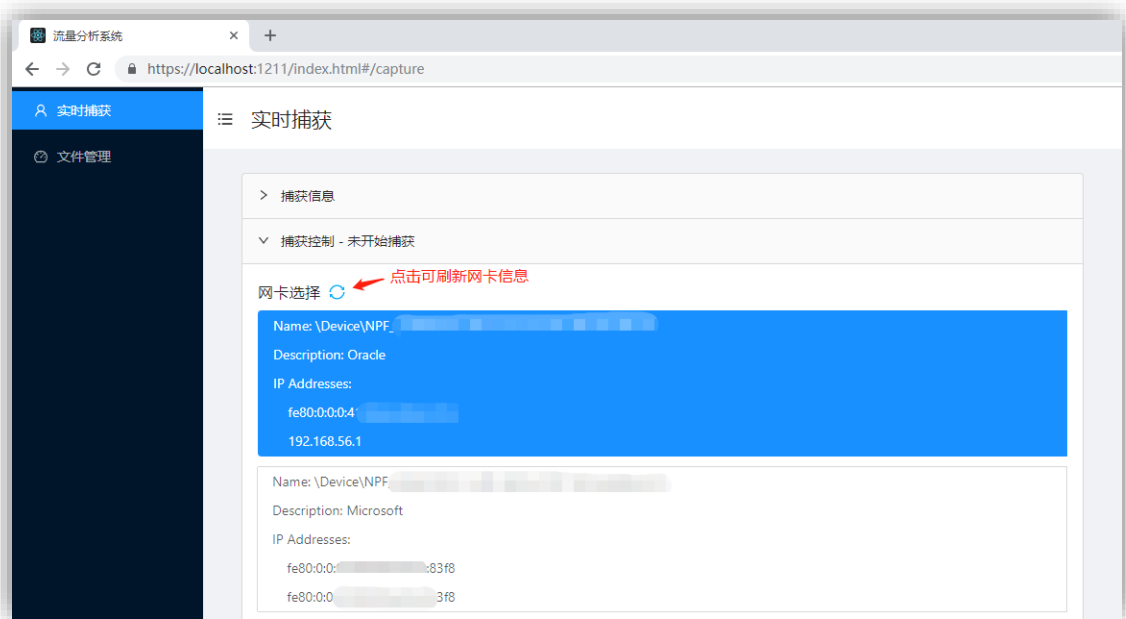


图 4-1-2 网卡选择



图 4-1-3 设置过滤器与开始捕获

实时数据：开始捕获后，捕获的数据将实时显示在表格中，每页将显示 15 条，如图 4-1-4 所示，同时可点击一项来获取具体数据，可展开到网络层，如图 4-1-5 所示。此时还可以按类型筛选数据，如图 4-1-6 所示。



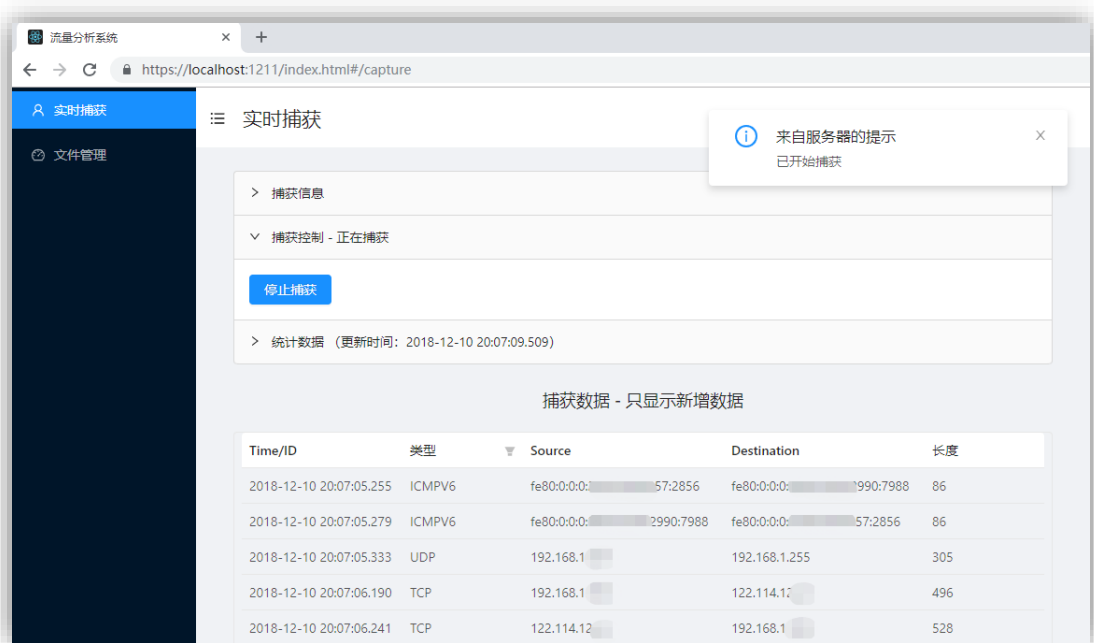


图 4-1-4 捕获的实时数据

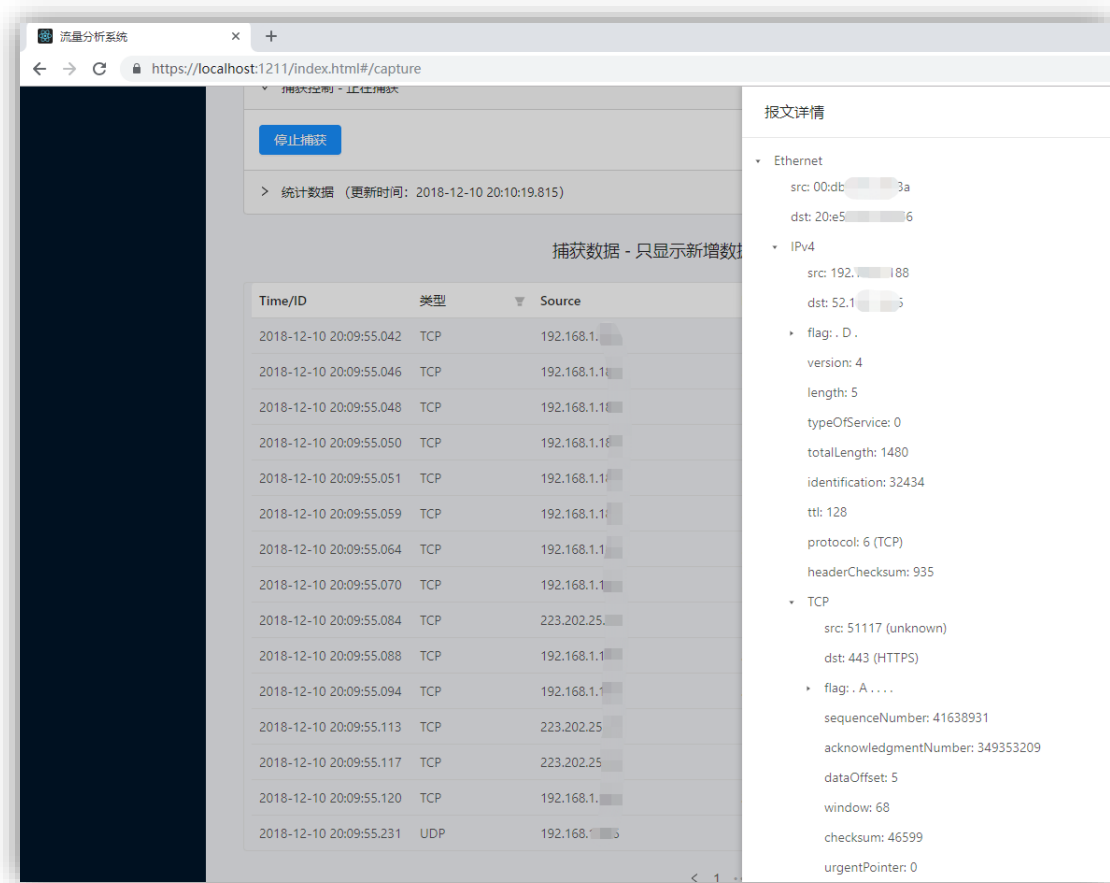


图 4-1-5 逐字段展开一个数据包

捕获数据 - 只显示新增数据

按类型筛选

Time/ID	类型		Source	Destination	长度
2018-12-10 20:21:17.976	TCP	<input type="checkbox"/> TCP	52.109.120.3	192.168.1.188	1514
2018-12-10 20:21:17.979	TCP	<input checked="" type="checkbox"/> UDP	192.168.1.188	52.109.120.3	66
2018-12-10 20:21:17.996	TCP	<input type="checkbox"/> ARP	52.109.120.3	192.168.1.188	1514
2018-12-10 20:21:18.007	TCP	<input type="checkbox"/> ICMP	192.168.1.188	52.109.120.3	54
2018-12-10 20:21:18.008	TCP	OK Reset	52.109.120.3	192.168.1.188	1514
2018-12-10 20:21:18.012	TCP		52.109.120.3	192.168.1.188	1318
2018-12-10 20:21:18.013	TCP		192.168.1.188	52.109.120.3	54
2018-12-10 20:21:18.017	TCP		192.168.1.188	52.109.120.3	268
2018-12-10 20:21:18.240	UDP		192.168.1.226	192.168.1.255	305
2018-12-10 20:21:18.492	TCP		52.109.120.3	192.168.1.188	161
2018-12-10 20:21:18.500	TCP		192.168.1.188	52.109.120.3	443
2018-12-10 20:21:18.503	TCP		192.168.1.188	52.109.120.3	1494
2018-12-10 20:21:18.505	TCP		192.168.1.188	52.109.120.3	1494
2018-12-10 20:21:18.508	TCP		192.168.1.188	52.109.120.3	1494
2018-12-10 20:21:18.510	TCP		192.168.1.188	52.109.120.3	1494

< 1 ... 220 221 222 223 224 ... 231 > Goto

图 4-1-6 按类型筛选数据

统计数据：点击捕获信息标签可以查看捕获信息，如图 4-1-7 所示，点击统计数据标签可查看统计信息，如图 4-1-8 所示。

捕获信息

当前状态	正在捕获
网卡信息	Name: \Device\NPF_{FAF16846-72EA-4025-8542-AA13F34EA7D5} Description: Microsoft IP Addresses: fde2:de93:9454:0:0:0:0:188 fc00:100:100:1:280c:a26d:2990:7988 fc00:100:100:1:0:0:0:188 192.168.1.188
过滤器	无
开始时间	2018-12-10 20:14:49.362

图 4-1-7 捕获信息

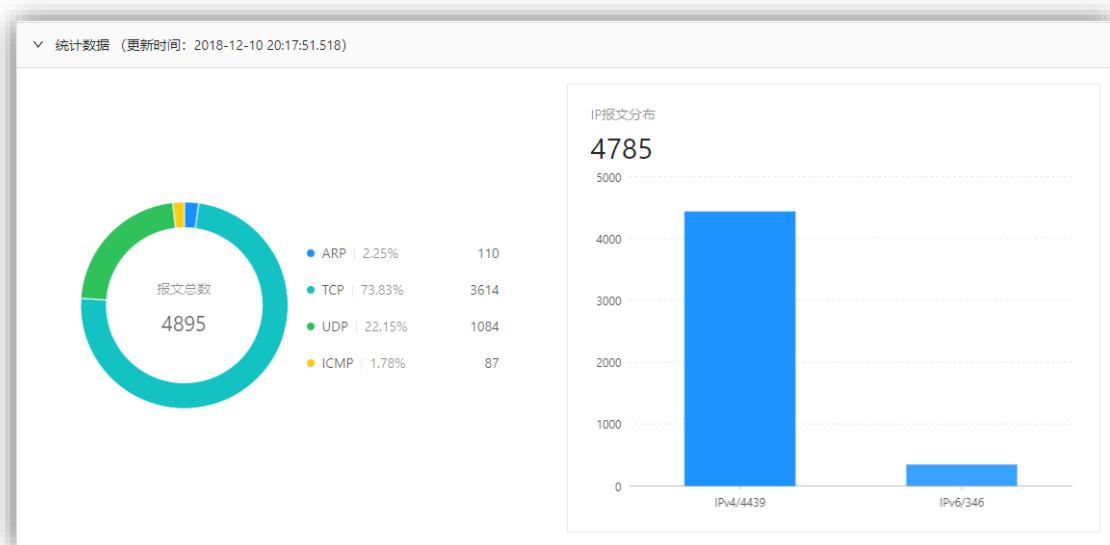


图 4-1-8 统计信息

**停止捕获：**点击捕获控制标签中的停止捕获可以停止捕获过程，如图 4-1-9 所示。注意，整个程序共享一个捕获子程序，在一个客户端中开始或停止捕获会向所有客户端发送开始或停止捕获消息，捕获数据和统计信息也会向所有客户端发送。

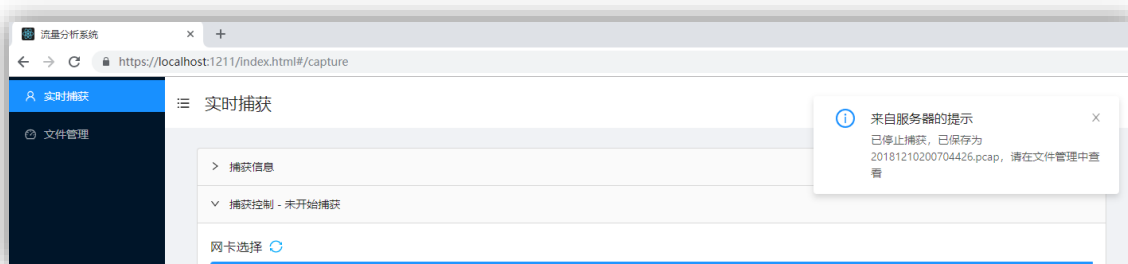


图 4-1-9 停止捕获

## 文件管理

**密钥认证：**与实时捕获模块相同。只要一个模块成功认证，另一个模块会同步认证。

**文件列表：**认证成功后会显示文件列表，点击刷新按钮可刷新文件列表，点击下载可将指定文件下载到本地，如图 4-1-10 所示。

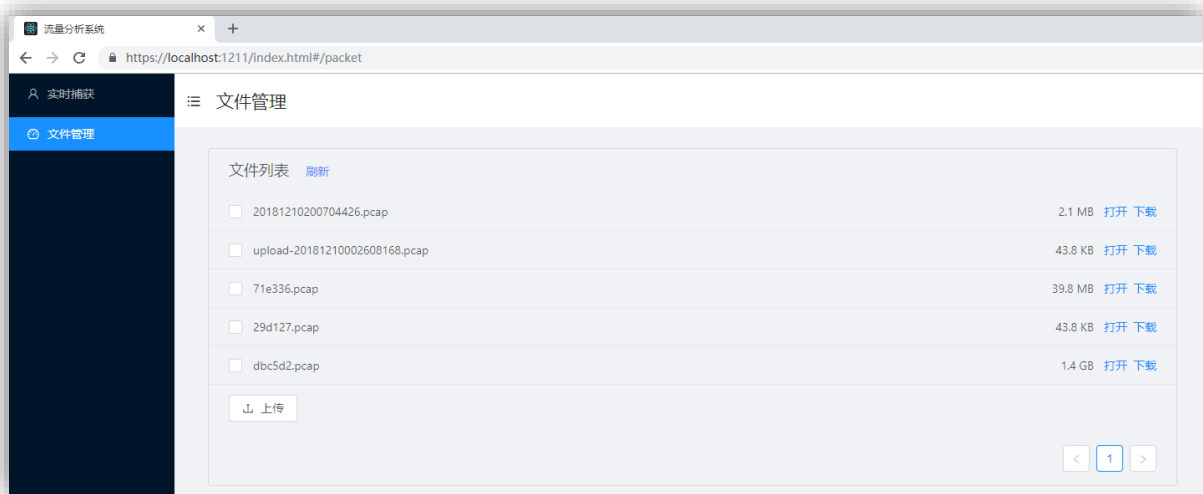


图 4-1-10 文件列表

上传文件：点击上传按钮可上传文件，只支持 pcap 格式，如图 4-1-11 所示。无论是否上传成功都会显示在上传历史区域，上传历史不会记录在本地，刷新页面后上传历史会被清空。上传会显示上传进度条，可随时取消上传。上传的文件如果被接受将被重命名为 upload-〈当前的时间戳〉. pcap。

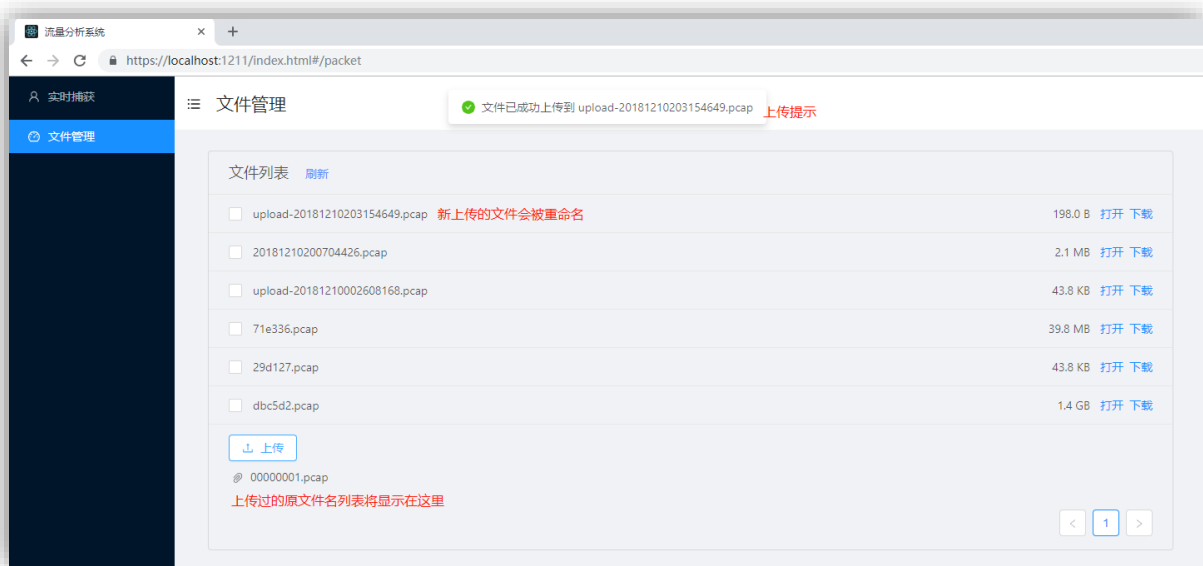


图 4-1-11 上传文件

删除文件：点击左侧的复选框可以删除文件，删除文件不可恢复，如图 4-1-12 所示。

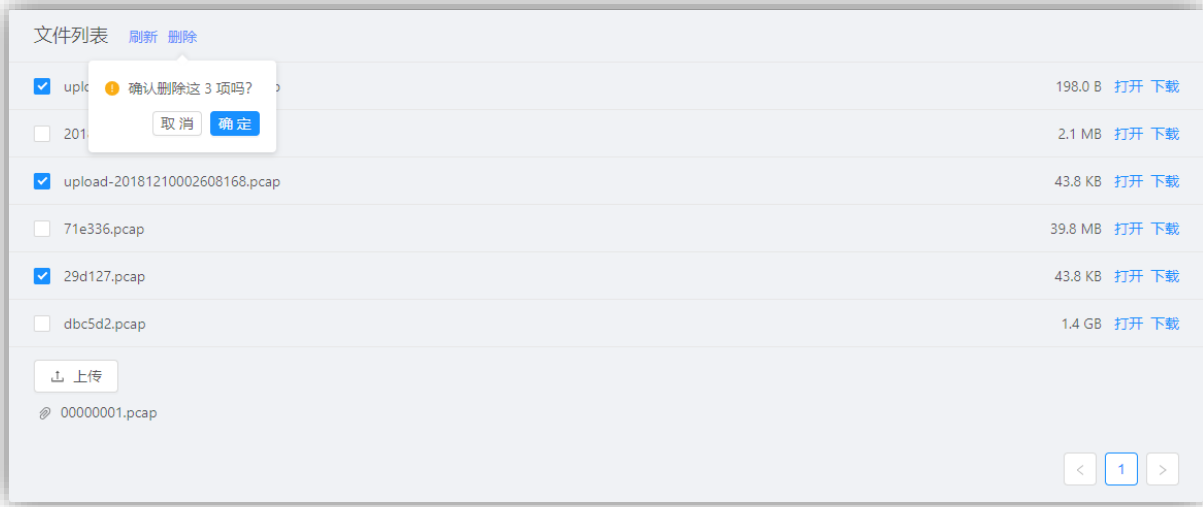


图 4-1-12 删除文件

打开文件：点击打开可打开文件，如图 4-1-13 所示。注意，打开较大的文件将会花费较多的时间和空间，**数据会逐步加载并每秒更新，直到完全加载完毕**。为了减轻前端的负荷，分页功能使用懒加载模式，切换页后如果该页本地没有数据将会向服务器获取，如果本地数据量过多（超过 10000 条时）将会删除前面的数据以保证系统稳定运行。

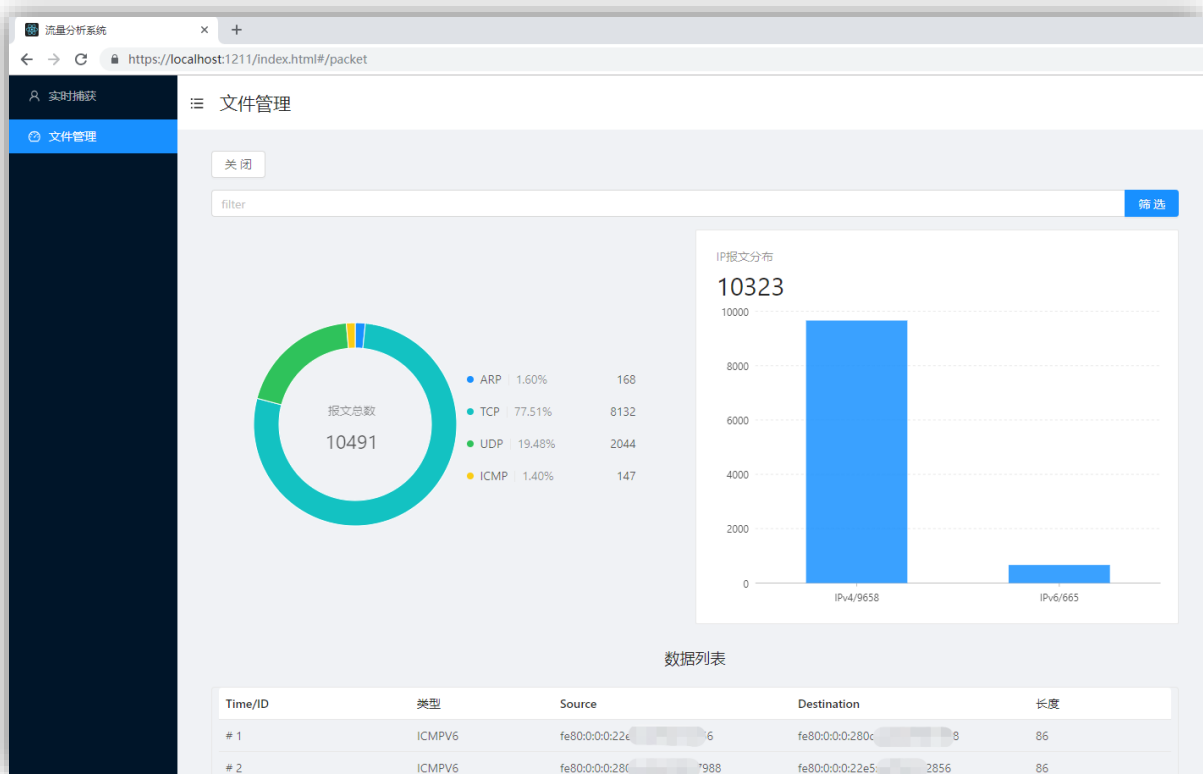


图 4-1-13 打开文件

筛选过滤：输入过滤条件可以进行过滤，如图 4-1-14 所示。注意，为了分页的稳定性，每次筛选将会重新向后台请求数据，将重新读取文件，筛选大文件同样需要更多的时间和空间。

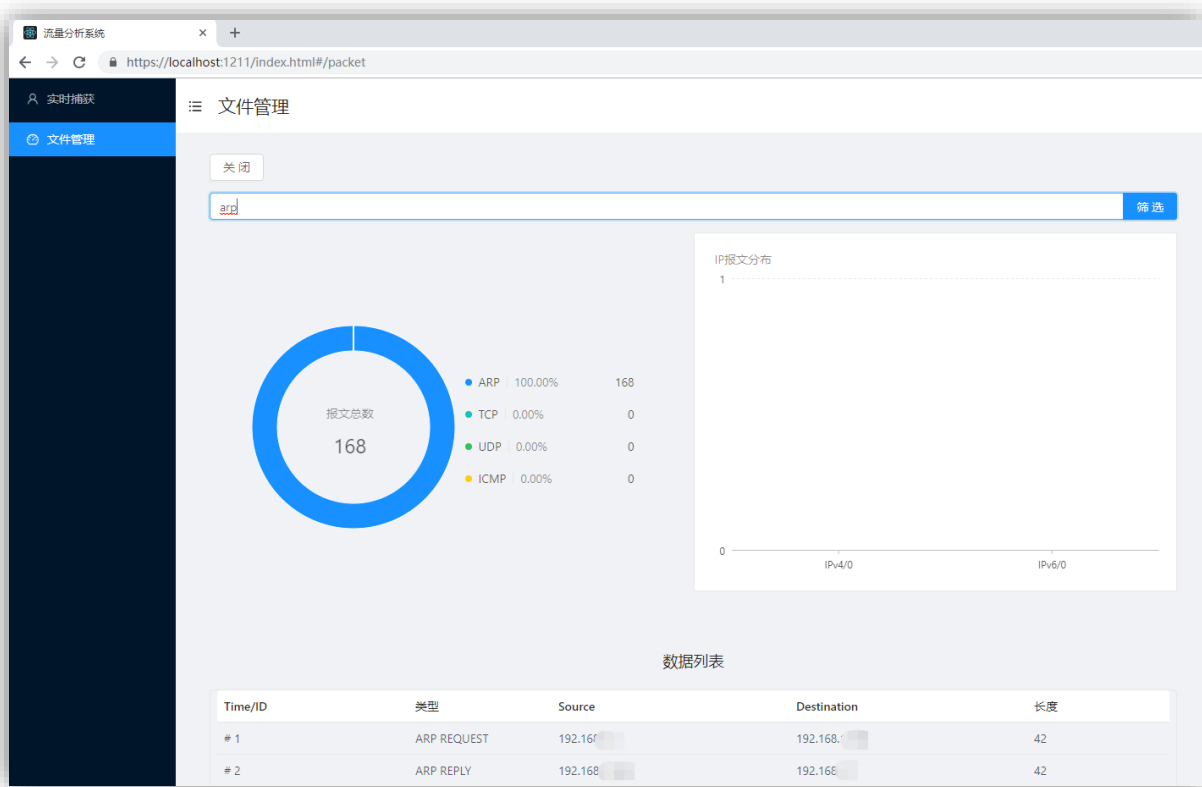


图 4-1-14 筛选 arp 报文

## 2. 捕获模块稳定性和资源占用分析

程序在个人 PC 机进行了长时间的捕获，从 12 月 7 日 5:37 到 12 月 10 日 19:39 共 86 小时的运行结果如图 4-1-15 所示，共计捕获 600 余万的数据包，输出文件有 4.75GB。程序运行时不影响电脑的正常使用。

同时在远端服务器上也运行了 4 天多总计超过 100 个小时，统计数据如图 4-1-16 所示。由于服务器的网络流量并不是很频繁，虽然时间更久数据包总量却相对较少，有 100 多万数据包，输出文件 1.38GB。由于服务器没有 IPv6 地址，所以没有捕获到 IPv6 数据包。程序运行时不影响服务器的正常使用（服务器上部署的多个服务均能正常使用）。

程序占用的资源如图 4-1-17 (Windows) 和图 4-1-18 (Linux) 所示。前端浏览器的资源占用如图 4-1-19 所示。

可见程序后台的内存占用大约是 100MB 左右，前端浏览器大约是 150MB，而对 CPU 的资源占用相对较少。

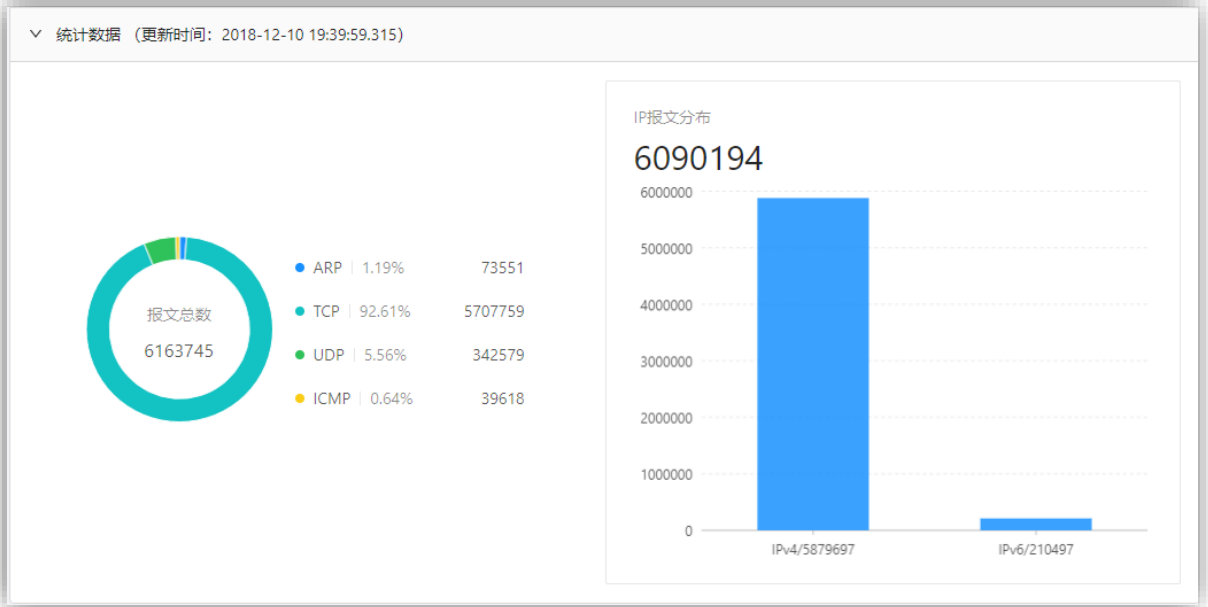


图 4-1-15 捕获 PC 机 86 小时的数据统计

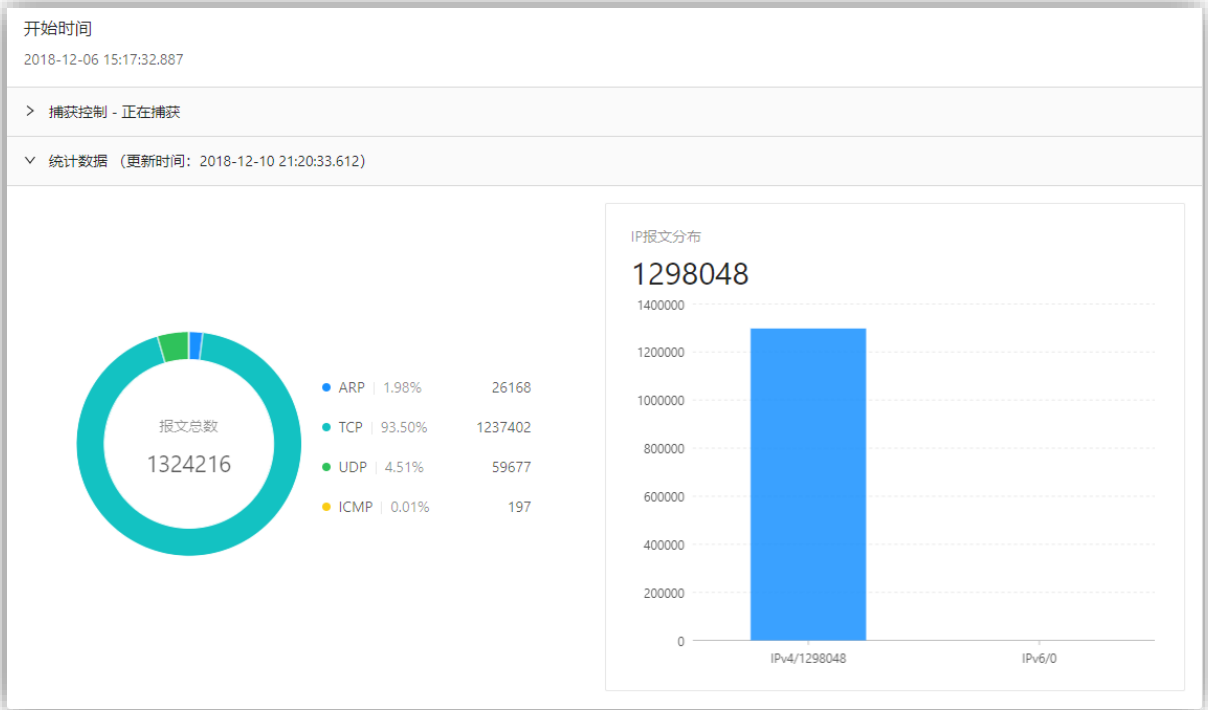


图 4-1-16 捕获服务器超过 100 小时的数据统计

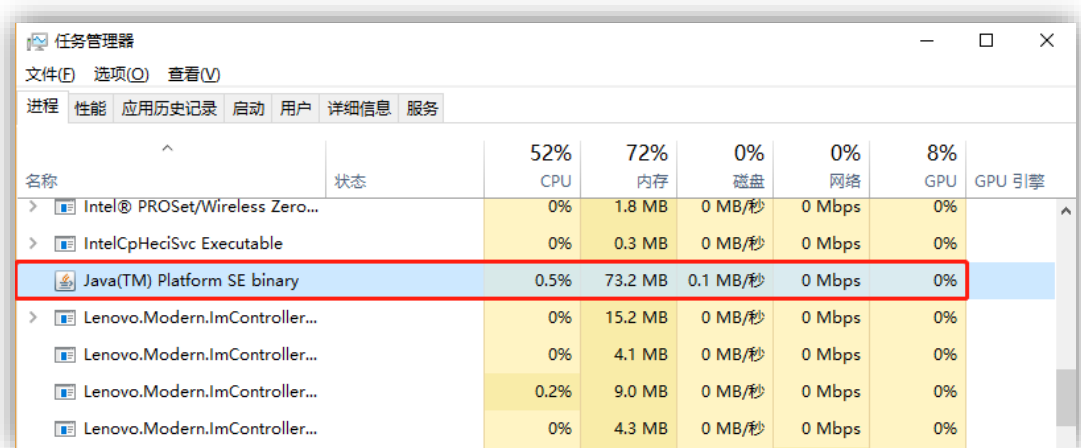


图 4-1-17 Windows 系统运行的资源占用



图 4-1-18 Linux 系统运行的资源占用

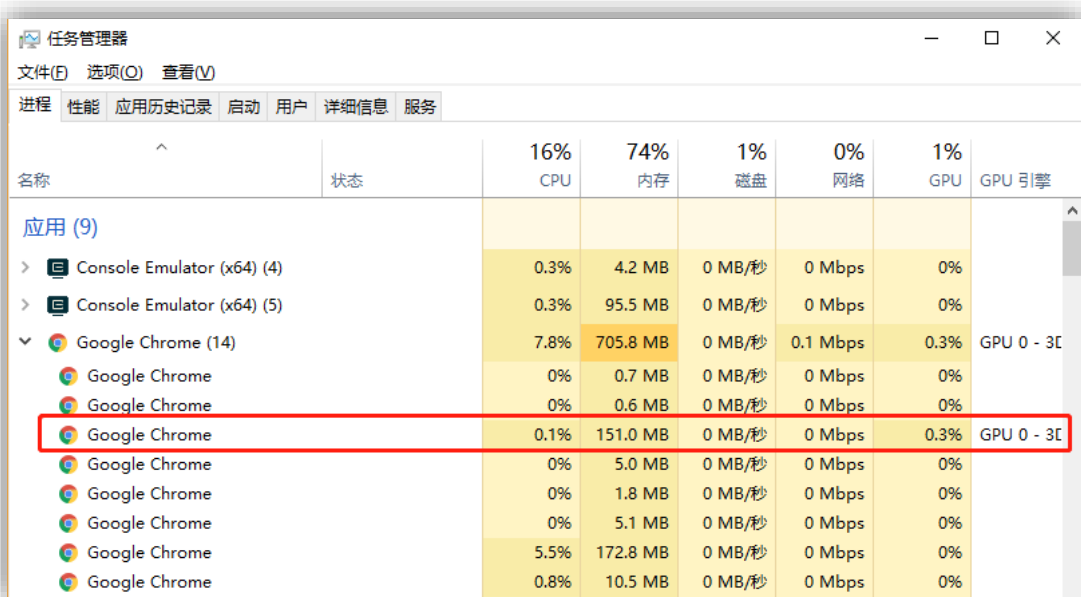


图 4-1-19 前端浏览器的资源占用（红框内为正在接收数据的客户端）



### 3. 手机端

由于以 web 方式展示内容，可以方便的同时兼容电脑端和手机端，手机端的功能与电脑端完全一致，部分截图如图 4-1-20 到 4-1-24 所示。

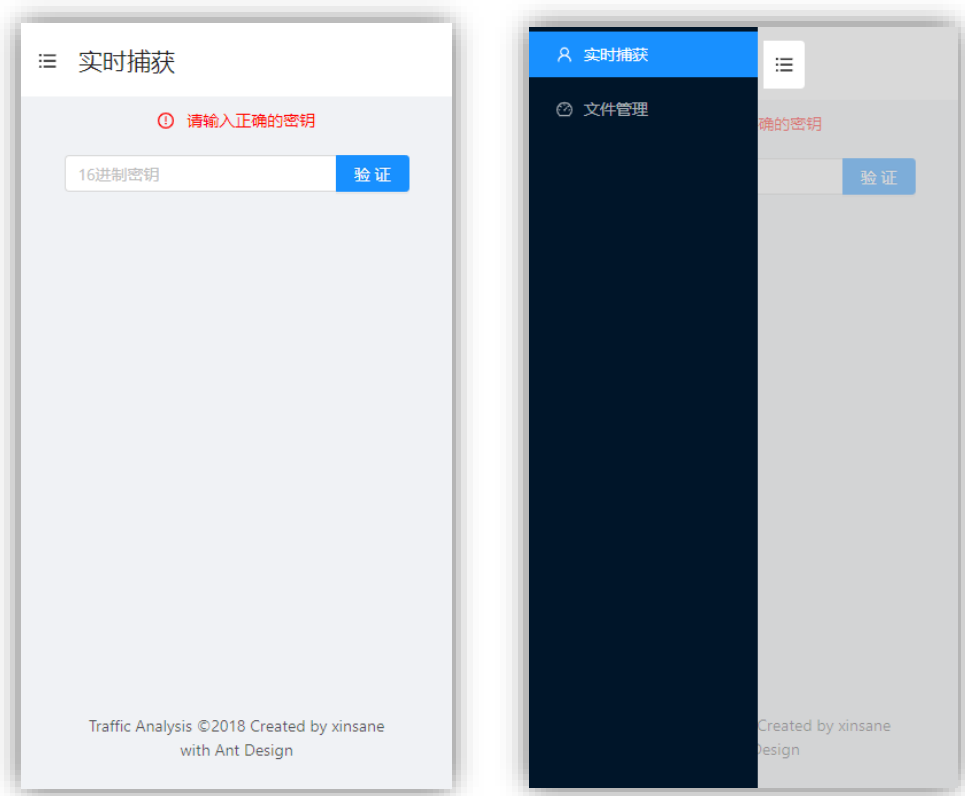


图 4-1-20 基本结构

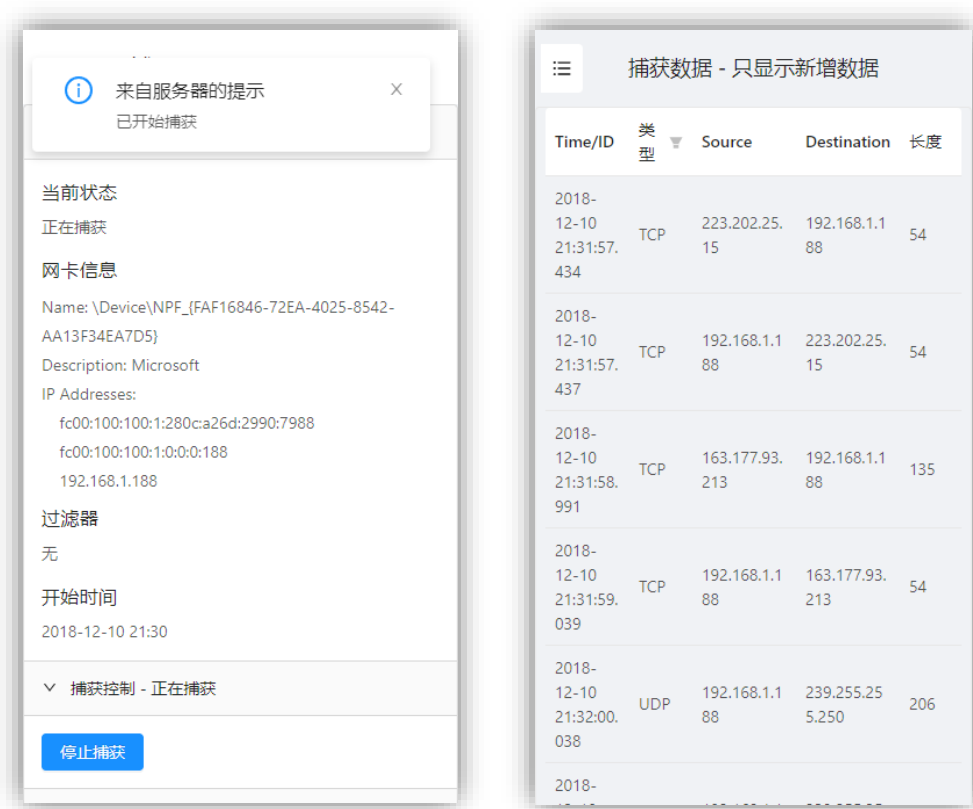


图 4-1-21 开始捕获

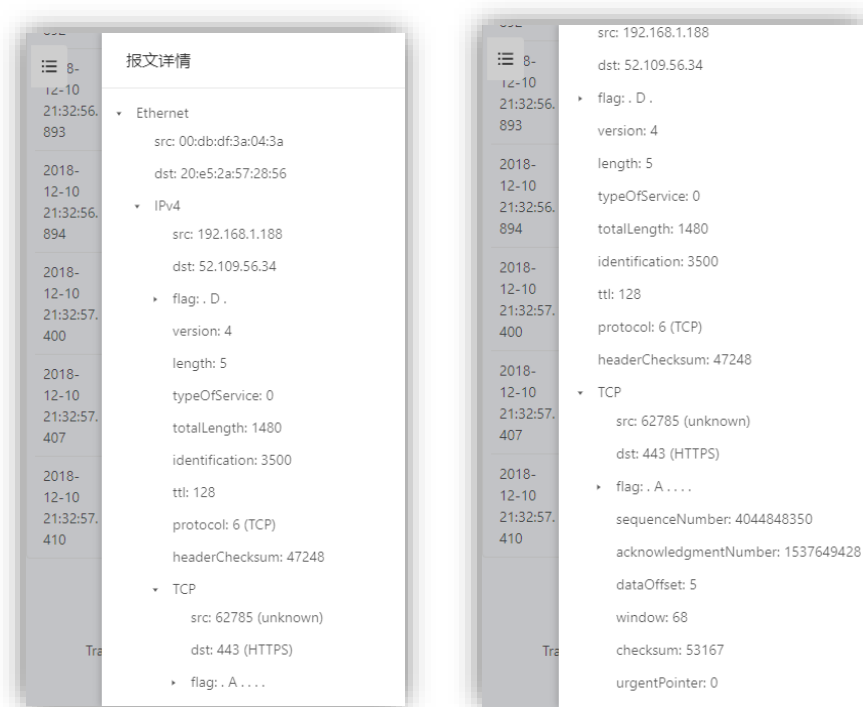


图 4-1-22 数据报文详情



图 4-1-23 文件管理

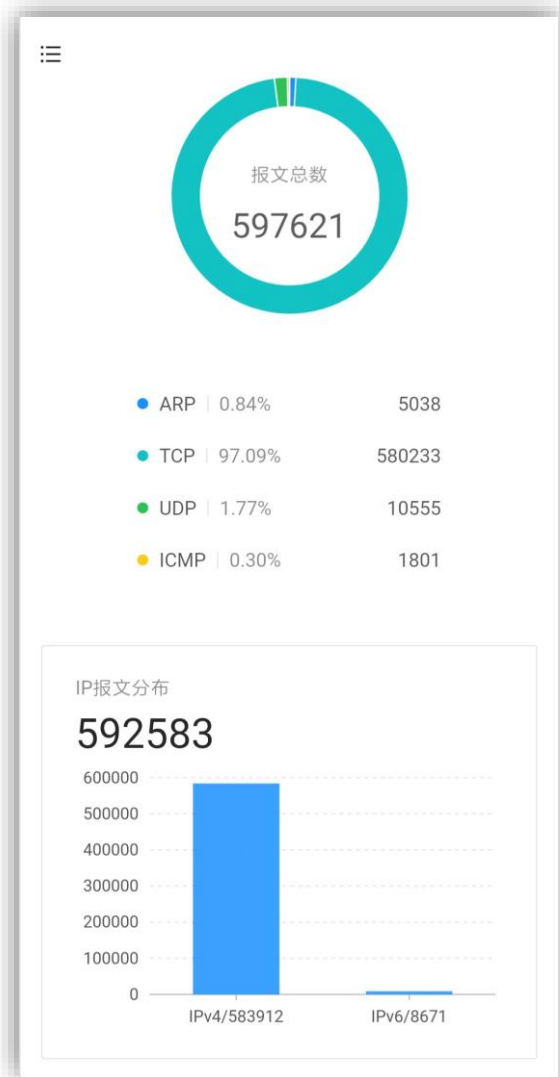


图 4-1-24 统计数据

### 附：使用手册（程序使用方法）

#### 1. 环境要求

- ✓ Java 运行环境 JDK/JRE 1.8.0，如图 4-2-1 所示。
- ✓ libpcap（Windows 环境可用 winpcap 代替）

```
$ java -version
java version "1.8.0_192"
Java(TM) SE Runtime Environment (build 1.8.0_192-b12)
Java HotSpot(TM) 64-Bit Server VM (build 25.192-b12, mixed mode)
```

图 4-2-1 Java 环境示例

## 2. 使用方法

**运行程序：** java -jar traffic\_analysis-1.0.jar <参数列表>

参数列表的解释参考图 4-2-2

图 4-3 是一些可能的配置方式

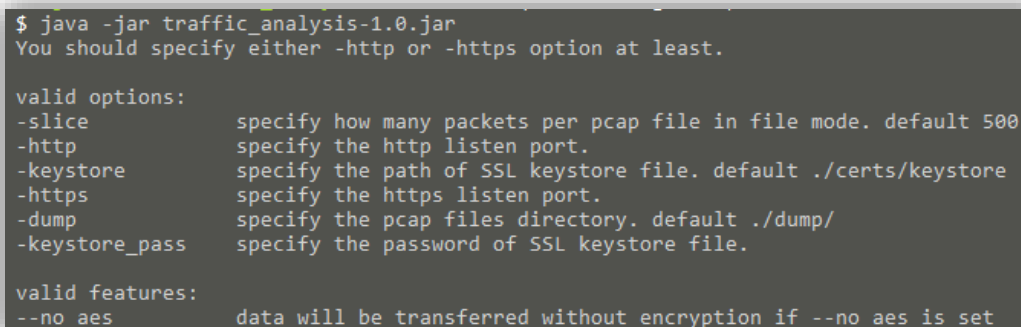
**AES 密钥的获取：** 程序运行时控制台将会调试输出 16 进制密钥，同时会将密钥写入程序运行目录下的 key.txt。可通过这两种方式获取密钥。

**前端浏览：** 打开浏览器访问对应协议对应端口，如：

https://localhost:12611/

URL 地址可直接带上 AES 密钥以直接认证，使用英文叹号包围的 hash，如：

https://localhost:12611/#!{aes\_hex\_key}!



```
$ java -jar traffic_analysis-1.0.jar
You should specify either -http or -https option at least.

valid options:
-slice          specify how many packets per pcap file in file mode. default 500
-http           specify the http listen port.
-keystore       specify the path of SSL keystore file. default ./certs/keystore
-https         specify the https listen port.
-dump          specify the pcap files directory. default ./dump/
-keystore_pass  specify the password of SSL keystore file.

valid features:
--no_aes       data will be transferred without encryption if --no_aes is set
```

图 4-2-2 运行参数的解释



```
常用配置.txt
1 # 默认配置
2 java -jar traffic_analysis-1.0.jar -http <http_port> -https <https_port>
3
4 # 不使用https
5 java -jar traffic_analysis-1.0.jar -http <http_port>
6
7 # 只使用https
8 java -jar traffic_analysis-1.0.jar -https <https_port>
9
10 # 不使用AES加密
11 java -jar traffic_analysis-1.0.jar -http <http_port> -https <https_port>
12 --no_aes
13
14 # 使用自定义证书
15 java -jar traffic_analysis-1.0.jar -http <http_port> -https <https_port>
16 -keystore <keystore_file_path> -keystore_pass <keystore_password>
```

图 4-3 常用配置

## 3. 注意事项

- ✓ 程序需要管理员权限，否则将无法捕获数据包。
- ✓ 至少指定 http 端口和 https 端口的其中一个，也可以同时指定以同时监听 http 端口和 https 端口。
- ✓ 如果指定了 https 端口，则必须正确配置 keystore 证书和密码。

程序附带有一个默认的 keystore 证书，默认密码为 traffic（使用默认证书需要先将 certs 目录下的 keystore.cer 安装到受信任的 CA 根证书列表中，使用默认证书只能通过 localhost 或 127.0.0.1 访问）。在实际使用中应配置实际的证书。如果在服务器上使用，可使用正规的 CA（如免费的 Let's Encrypt）提供的证书并生成对应的 keystore，这样在通过域名访问时能够让浏览器正确识别。如果是自签名证书，需要信任根证书才能让浏览器正确识别，不建议通过关闭浏览器的安全警告方式访问，这样很容易被中间人攻击。
- ✓ 如果指定了 http 端口而关闭了 AES 加密，整个通讯将暴露在网络中，一般只有在本地使用且正确配置了防火墙时才使用这种配置。

## 五、课程设计总结

本次课程设计实现一个简单的流量分析软件，对运行本程序的机器的网络流量进行捕获并展示，同时将会保存为文件，用户可以查看保存的文件，也可以对这些文件进行简单的管理（上传、下载或删除）。

在最初的设计中，很多问题没有预先考虑周全。

最大的问题就是，虽然使用 web 作为展示前端可以大大简化前端的开发并且使远图形化查看数据成为了可能，但却带来了很多的麻烦，比如虽然当初考虑到了循环流量的过滤，却没有考虑到代理的存在可能会打破这种美梦，另外如果一台机器开了两个程序捕获同一个网卡，也会出现循环流量的问题，这种爆发式的流量增长虽然实测中没有造成系统崩溃，却带来了较大的性能影响，而且这种洪流式的数据流量会使正常的流量被淹没掉，难以分辨哪些是真实的流量。当然，只要在运行程序时多加注意，不同时使用多个客户端，不捕获同一个网卡，使用代理时使用添加 X-Forwarded-For 头标记，就可以避免这种情况。

同时，使用 web 传输数据将带来安全性问题，数据报文不能直接暴露在复杂的网络环境中，在使用 AES 加密后，观察程序性能并没有较大的额外消耗，这是个好消息。另外 HTTPS 给网站带来了传输层的保障，想要正确配置却没有那么容易，这次课程设计中花了大量的时间在证书的配置上，最终通过 HTTPS 和 AES 实现了前后端的互信。

另外，最初设计时认为使用第三方组件库将会把大部分精力投入到逻辑和后台开

发中，但实际却没有那么容易，虽然 Ant Design 提供了大量的组件，使用也比较方便，但不得不承认，学习这些组件也需要一定的成本，当初没有把这部分时间算进去也算是一大失误，导致后期时间非常紧张，很多问题只是简单粗暴的解决或没有解决，比如.pcap 文件的索引问题最终也只是粗暴的用操作系统的文件系统来解决，另外还有一些小问题没有处理，比如后期调试时发现捕获开始的时间不见了，这些影响较小的问题由于时间关系便没有进行处理。

但即便如此，最终还是完成了任务，这一路上的收获也是非常丰富。比如没有考虑到的 Ant Design 学习成本，现在已经基本学会了很多组件，下次有需要时便可以快速投入使用。花费了很多时间的证书配置，所有成功使用配置的命令都记录下来了，以后在这方面就会相对比较容易实现，并且对实现原理也做了部分了解，更容易理解 HTTPS 为什么可以建立客户端对服务器的信任。当然，除了这些印象深刻的记忆，还有很多零零碎碎的知识，比如嵌入式 jetty 的使用，websocket 的使用等等。

另外，先期的考虑很难做到滴水不漏，需要留出大量的时间来处理意想不到的情况和后期的代码调试，避免后期时间紧张，来不及完善程序或草草开发收获太少。

总之，尽管这个过程中遇到了很多问题，但在解决这些问题的过程中收获了大量的经验，也算是收获甚丰了。

## 参考文献

- [1] React – A JavaScript library for building user interfaces, <https://reactjs.org/>
- [2] Create React App, <https://github.com/facebook/create-react-app>
- [3] Ant Design, <https://ant.design>
- [4] Ant Design Pro, <https://pro.ant.design/>
- [5] crypto.js, <https://github.com/brix/crypto-js>
- [6] jetty, <https://www.eclipse.org/jetty/>
- [7] pcap4j, <https://github.com/kaitoy/pcap4j>
- [8] gson, <https://github.com/google/gson>
- [9] slf4j, <https://www.slf4j.org/>
- [10] commons-fileupload, <https://commons.apache.org/proper/commons-fileupload/>
- [11] gradle, <https://gradle.org/>
- [12] Spring Boot Gradle Plugin Reference Guide, <https://docs.spring.io/spring-boot/docs/current/gradle-plugin/reference/html/>
- [13] BPF 过滤规则, [https://en.wikipedia.org/wiki/Berkeley\\_Packet\\_Filter](https://en.wikipedia.org/wiki/Berkeley_Packet_Filter)
- [14] BPF 语法简单介绍, <https://www.cnblogs.com/zhongxinWang/p/4303153.html>