# 实验三 流量统计软件

## **一．实验目的**

分析IP协议，统计流量。大家可以把流量计做成流量地图，显示不同地点间数据实时发送的量，比如武汉到北京的数据包发送的数据量。 可以用echart画地图（或者D3），用whois（或者ip-api）获取IP地址的所属地。

## **二．实验原理**

WINPCAP（NPCAP）是一个基于Win32平台的，用于捕获网络数据包并进行分析的开源库.

大多数网络应用程序通过被广泛使用的操作系统元件来访问网络，比如sockets。  这是一种简单的实现方式，因为操作系统已经妥善处理了底层具体实现细节（比如协议处理，封装数据包等等），并且提供了一个与读写文件类似的，令人熟悉的接口。

然而，有些时候，这种“简单的方式”并不能满足任务的需求，因为有些应用程序需要直接访问网络中的数据包。也就是说，那些应用程序需要访问原始数据包，即没有被操作系统利用网络协议处理过的数据包。

WINPCAP（NPCAP）产生的目的，就是为Win32应用程序提供这种访问方式； WINPCAP（NPCAP）提供了以下功能 ：

1.捕获原始数据包，无论它是发往某台机器的，还是在其他设备（共享媒介）上2.进行交换的

3.在数据包发送给某应用程序前，根据用户指定的规则过滤数据包

4.将原始数据包通过网络发送出去

5.收集并统计网络流量信息

以上这些功能需要借助安装在Win32内核中的网络设备驱动程序才能实现，再加上几个动态链接库DLL。

所有这些功能都能通过一个强大的编程接口来表现出来,易于开发，并能在不同的操作系统上使用。这本手册的主要目标是在一些程序范例的帮助下，叙述这些编程接口的使用。

WINPCAP（NPCAP）可以被用来制作网络分析、监控工具。一些基于WINPCAP（NPCAP）的典型应用有：

网络与协议分析器 (network and protocol analyzers)

网络监视器 (network monitors)

网络流量记录器 (traffic loggers)

网络流量发生器 (traffic generators)

用户级网桥及路由 (user-level bridges and routers)

网络入侵检测系统 (network intrusion detection systems (NIDS))

网络扫描器 (network scanners)

安全工具 (security tools)

## 软件功能需求描述

本流量统计软件（流量地图）需要实现以下功能：

1.数据包捕获功能：

实时捕获网络接口上的IP数据包

支持过滤非IP数据包

2.PCI解析功能：

解析每个数据包的协议控制信息

支持TCP、UDP和ICMP协议的解析

提取源/目的端口、TCP标志位等关键信息

3.流量统计功能

按源IP地址统计流量

记录每个IP的数据包数量和总字节数

实时更新流量统计信息

4.地理位置查询功能

通过IP-API查询IP地址的地理位置

获取国家、城市、经纬度等信息

自动识别局域网地址

5.可视化展示功能

使用ECharts地图展示全球流量分布

通过散点大小表示流量大小

实时更新流量地图展示

点击节点查看详细PCI信息

6.用户交互功能

提供直观的Web界面

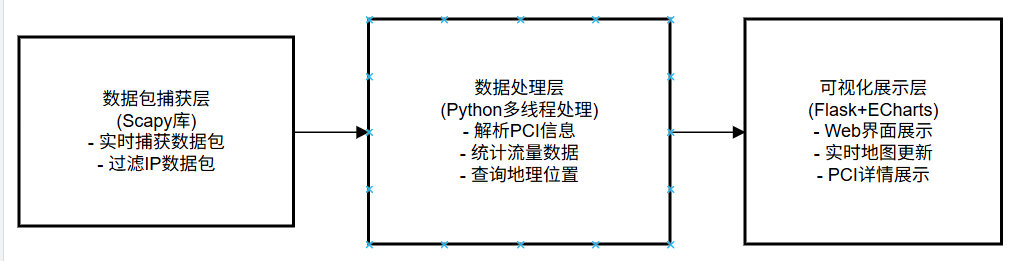
实时显示总流量统计

支持查看最近的数据包详情

## 四．流量统计软件（流量地图）的设计

### 4.1系统架构设计

系统采用三层架构：



### 4.2 模块设计

1.数据包捕获模块：使用Scapy库嗅探网络接口；设置IP过滤器；为每个捕获的数据包调用处理函数

2.PCI解析模块：识别传输层协议（TCP/UDP/ICMP）；提取协议特定信息（端口、标志位等）；添加时间戳和包大小信息

3.流量统计模块：使用字典存储各IP的流量数据；线程安全的计数器实现；保留最近5条PCI记录

4.地理位置查询模块：使用IP-API查询地理位置；缓存查询结果；特殊处理局域网地址

5.数据推送模块：使用WebSocket实时推送数据；定时更新统计信息；序列化数据结构

1. 可视化模块：ECharts地图初始化；散点图展示流量分布；交互式PCI详情面板

### 4.3 数据结构设计

# IP流量统计数据结构

traffic\_stats = {

"192.168.1.100": {

"bytes": 10240, # 总字节数

"packets": 15, # 总包数

"location": { # 地理位置

"country": "China",

"city": "Beijing",

"lat": 39.9042,

"lon": 116.4074

},

"pci\_info": [ # 最近PCI记录

{

"time": "14:30:22",

"protocol": "TCP",

"src\_port": 443,

"dst\_port": 55342,

"flags": "SA",

"size": 64

},

# ... 其他记录

]

},

# ... 其他IP

}

## 五．流量统计软件（流量地图）的实现

### 5.1 关键技术

网络捕获：使用Scapy库实现底层数据包捕获

多线程：使用Python threading模块实现并发处理

Web框架：使用Flask提供Web服务

实时通信：使用Socket.IO实现WebSocket通信

数据可视化：使用ECharts实现地理信息可视化

地理查询：使用IP-API服务获取IP地理位置

### 5.2 核心算法

数据包处理算法

def packet\_handler(packet):

if IP in packet:

src\_ip = packet[IP].src

length = len(packet)

pci\_info = get\_pci\_info(packet)

with lock:

update\_traffic\_stats(src\_ip, length, pci\_info)

PCI解析算法

def get\_pci\_info(packet):

pci = {}

if TCP in packet:

pci['protocol'] = 'TCP'

pci['src\_port'] = packet[TCP].sport

pci['dst\_port'] = packet[TCP].dport

pci['flags'] = str(packet[TCP].flags) # 转换为字符串

# 类似处理UDP和ICMP...

return pci

流量统计更新算法

def update\_traffic\_stats(ip, length, pci\_info):

# 如果该IP首次出现，初始化数据结构

if ip not in traffic\_stats:

traffic\_stats[ip] = {

'bytes': 0,

'packets': 0,

'location': None,

'pci\_info': []

}

# 更新流量数据

traffic\_stats[ip]['bytes'] += length

traffic\_stats[ip]['packets'] += 1

# 添加PCI记录

pci\_entry = {

'time': time.strftime("%H:%M:%S"),

'protocol': pci\_info.get('protocol', 'N/A'),

'src\_port': pci\_info.get('src\_port', 'N/A'),

'dst\_port': pci\_info.get('dst\_port', 'N/A'),

'flags': pci\_info.get('flags', 'N/A'),

'size': length

}

traffic\_stats[ip]['pci\_info'].append(pci\_entry)

# 保留最近5条记录

if len(traffic\_stats[ip]['pci\_info']) > 5:

traffic\_stats[ip]['pci\_info'] = traffic\_stats[ip]['pci\_info'][-5:]

### 5.3可视化实现

前端使用ECharts地图展示流量分布：

// 初始化地图

const mapChart = echarts.init(document.getElementById('map'));

// 配置地图选项

const option = {

series: [{

type: 'scatter',

coordinateSystem: 'geo',

data: trafficData.map(item => ({

name: item.city,

value: [item.lon, item.lat, item.bytes],

// ...其他属性

})),

symbolSize: function(val) {

return Math.min(30, Math.max(8, Math.log(val[2]) \* 3));

}

}]

};

// 设置点击事件

mapChart.on('click', function(params) {

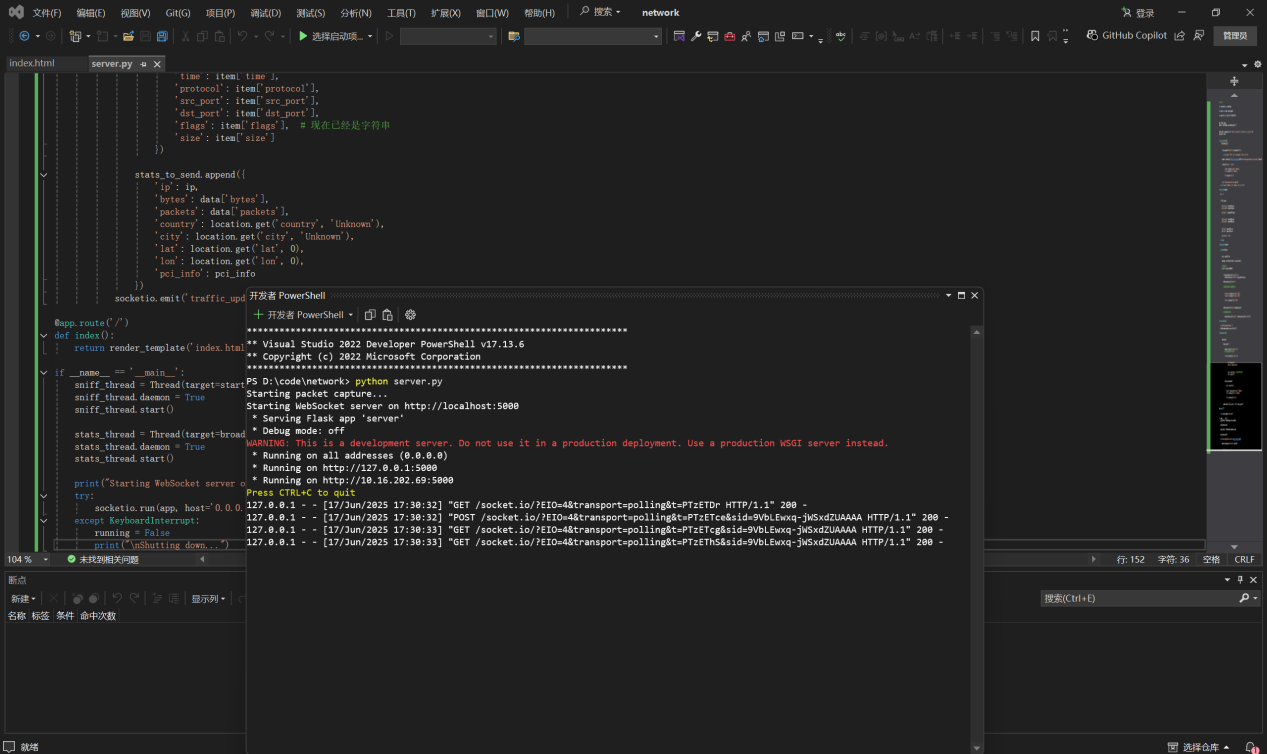
if (params.data) {

showPciDetails(params.data.pci\_info, params.data.ip);

}

});

### 5.4实现结果展示







## 总结

6.1 实验成果

本实验成功实现了一个功能完整的网络流量统计与可视化系统：

实现了实时网络数据包捕获功能

完成了对TCP/UDP/ICMP协议的PCI解析

建立了按IP地址的流量统计机制

整合了IP地理位置查询服务

开发了基于ECharts的实时流量地图

创建了直观的用户交互界面

6.2技术收获

通过本次实验，深入理解和实践了：

网络协议栈（特别是IP、TCP、UDP协议）

网络数据包捕获与分析技术

多线程编程与线程同步机制

Web实时通信技术（WebSocket）

数据可视化技术（ECharts地图）

REST API集成与使用

6.3实验心得

本次实验将理论知识转化为实际应用，加深了对网络协议的理解。通过解决数据包解析、多线程同步、实时可视化等技术挑战，提升了工程实践能力。流量地图的可视化效果直观展示了网络通信的全球分布特征，为网络监控和分析提供了有力工具。实验过程中也认识到网络协议的复杂性和网络流量的动态特性，为今后深入研究网络技术奠定了坚实基础。

## 代码附录

Server.py

# server.py

import time

from collections import defaultdict

from threading import Thread, Lock

from flask import Flask, render\_template

from flask\_socketio import SocketIO

from scapy.all import sniff, IP, TCP, UDP, ICMP

import requests

app = Flask(\_\_name\_\_)

socketio = SocketIO(app, cors\_allowed\_origins="\*")

# 全局变量

traffic\_stats = defaultdict(lambda: {'bytes': 0, 'packets': 0, 'location': None, 'pci\_info': []})

stats\_lock = Lock()

running = True

def get\_ip\_location(ip):

"""获取IP地理位置信息"""

try:

# 简单判断局域网地址

if (ip.startswith('192.168.') or ip.startswith('10.') or

ip.startswith('172.') and 16 <= int(ip.split('.')[1]) <= 31):

return {'country': 'LAN', 'city': 'Local Network', 'lat': 0, 'lon': 0}

response = requests.get(f"http://ip-api.com/json/{ip}?fields=status,message,country,city,lat,lon,query", timeout=2)

data = response.json()

if data.get('status') == 'success':

return {

'country': data.get('country', 'Unknown'),

'city': data.get('city', 'Unknown'),

'lat': data.get('lat', 0),

'lon': data.get('lon', 0)

}

except Exception as e:

print(f"IP location fetch error for {ip}: {e}")

return {'country': 'Unknown', 'city': 'Unknown', 'lat': 0, 'lon': 0}

def get\_pci\_info(packet):

"""提取PCI信息"""

pci = {}

# 协议类型

if TCP in packet:

pci['protocol'] = 'TCP'

pci['src\_port'] = packet[TCP].sport

pci['dst\_port'] = packet[TCP].dport

# 修复：将TCP标志位转换为字符串

pci['flags'] = str(packet[TCP].flags)

elif UDP in packet:

pci['protocol'] = 'UDP'

pci['src\_port'] = packet[UDP].sport

pci['dst\_port'] = packet[UDP].dport

elif ICMP in packet:

pci['protocol'] = 'ICMP'

pci['type'] = packet[ICMP].type

pci['code'] = packet[ICMP].code

else:

pci['protocol'] = 'Other'

return pci

def packet\_handler(packet):

"""处理捕获的数据包"""

global traffic\_stats

if IP in packet:

src\_ip = packet[IP].src

length = len(packet)

timestamp = time.strftime("%H:%M:%S", time.localtime())

# 获取PCI信息

pci\_info = get\_pci\_info(packet)

with stats\_lock:

if traffic\_stats[src\_ip]['location'] is None:

traffic\_stats[src\_ip]['location'] = get\_ip\_location(src\_ip)

traffic\_stats[src\_ip]['bytes'] += length

traffic\_stats[src\_ip]['packets'] += 1

# 添加PCI信息到列表（保留最近的5条）

pci\_entry = {

'time': timestamp,

'protocol': pci\_info.get('protocol', 'N/A'),

'src\_port': pci\_info.get('src\_port', 'N/A'),

'dst\_port': pci\_info.get('dst\_port', 'N/A'),

'flags': pci\_info.get('flags', 'N/A'),

'size': length

}

traffic\_stats[src\_ip]['pci\_info'].append(pci\_entry)

# 只保留最近5条PCI记录

if len(traffic\_stats[src\_ip]['pci\_info']) > 5:

traffic\_stats[src\_ip]['pci\_info'] = traffic\_stats[src\_ip]['pci\_info'][-5:]

def start\_sniffing():

"""启动数据包捕获线程"""

print("Starting packet capture...")

sniff(prn=packet\_handler, store=0, filter="ip")

def broadcast\_stats():

global running

while running:

time.sleep(2)

with stats\_lock:

stats\_to\_send = []

for ip, data in traffic\_stats.items():

location = data.get('location') or {}

# 确保所有数据都是可序列化的

pci\_info = []

for item in data.get('pci\_info', []):

# 确保所有字段都是基本类型

pci\_info.append({

'time': item['time'],

'protocol': item['protocol'],

'src\_port': item['src\_port'],

'dst\_port': item['dst\_port'],

'flags': item['flags'], # 现在已经是字符串

'size': item['size']

})

stats\_to\_send.append({

'ip': ip,

'bytes': data['bytes'],

'packets': data['packets'],

'country': location.get('country', 'Unknown'),

'city': location.get('city', 'Unknown'),

'lat': location.get('lat', 0),

'lon': location.get('lon', 0),

'pci\_info': pci\_info

})

socketio.emit('traffic\_update', {'data': stats\_to\_send})

@app.route('/')

def index():

return render\_template('index.html')

if \_\_name\_\_ == '\_\_main\_\_':

sniff\_thread = Thread(target=start\_sniffing)

sniff\_thread.daemon = True

sniff\_thread.start()

stats\_thread = Thread(target=broadcast\_stats)

stats\_thread.daemon = True

stats\_thread.start()

print("Starting WebSocket server on http://localhost:5000")

try:

socketio.run(app, host='0.0.0.0', port=5000)

except KeyboardInterrupt:

running = False

print("\nShutting down...")

Index.html

<!-- index.html -->

<!DOCTYPE html>

<html>

<head>

<title>实时网络流量地图</title>

<script src="https://cdn.jsdelivr.net/npm/echarts@5.3.2/dist/echarts.min.js"></script>

<script src="https://cdnjs.cloudflare.com/ajax/libs/socket.io/4.0.1/socket.io.js"></script>

<style>

body {

margin: 0;

padding: 0;

font-family: Arial;

overflow: hidden;

}

#map {

width: 100vw;

height: 100vh;

}

#pci-details {

position: absolute;

top: 10px;

right: 10px;

width: 300px;

background: rgba(255, 255, 255, 0.9);

border-radius: 8px;

padding: 15px;

box-shadow: 0 0 10px rgba(0,0,0,0.2);

z-index: 1000;

max-height: 80vh;

overflow-y: auto;

}

.pci-header {

font-weight: bold;

margin-bottom: 10px;

color: #333;

border-bottom: 1px solid #eee;

padding-bottom: 5px;

}

.pci-item {

margin-bottom: 10px;

padding: 8px;

background: #f9f9f9;

border-radius: 4px;

font-size: 13px;

}

.pci-time {

font-weight: bold;

color: #d94e5d;

}

.pci-protocol {

color: #50a3ba;

font-weight: bold;

}

.close-btn {

position: absolute;

top: 5px;

right: 10px;

cursor: pointer;

font-weight: bold;

color: #666;

}

</style>

</head>

<body>

<div id="map"></div>

<div id="pci-details" style="display: none;">

<span class="close-btn" onclick="document.getElementById('pci-details').style.display='none'">×</span>

<div class="pci-header">PCI信息 - <span id="pci-ip"></span></div>

<div id="pci-container"></div>

</div>

<script>

// 初始化地图

const mapChart = echarts.init(document.getElementById('map'));

let totalBytes = 0;

let selectedIp = null;

// 连接WebSocket服务器

const socket = io('http://localhost:5000');

// 处理流量更新

socket.on('traffic\_update', function (data) {

updateMap(data.data);

});

function updateMap(trafficData) {

// 计算总流量

totalBytes = trafficData.reduce((sum, item) => sum + item.bytes, 0);

const option = {

title: {

text: '实时网络流量地图',

subtext: `总流量: ${formatBytes(totalBytes)}`,

left: 'center',

textStyle: {

fontSize: 18,

color: '#333'

}

},

tooltip: {

trigger: 'item',

formatter: params => {

if (params.data) {

return `

<div style="font-weight:bold">${params.data.city}</div>

<div>IP: ${params.data.ip}</div>

<div>国家: ${params.data.country}</div>

<div>流量: ${formatBytes(params.data.bytes)}</div>

<div>包数: ${params.data.packets}</div>

<div style="margin-top:5px;color:#d94e5d">点击查看PCI详情</div>

`;

}

return params.name;

}

},

visualMap: {

min: 0,

max: Math.max(...trafficData.map(d => d.bytes)),

text: ['高流量', '低流量'],

realtime: false,

calculable: true,

inRange: {

color: ['#50a3ba', '#eac736', '#d94e5d']

},

textStyle: {

color: '#333'

}

},

series: [{

name: '流量分布',

type: 'scatter',

coordinateSystem: 'geo',

data: trafficData.map(item => ({

name: item.city,

value: [item.lon, item.lat, item.bytes],

ip: item.ip,

country: item.country,

city: item.city,

bytes: item.bytes,

packets: item.packets,

pci\_info: item.pci\_info

})),

symbolSize: function (val) {

return Math.min(30, Math.max(8, Math.log(val[2]) \* 3));

},

label: {

show: true,

formatter: '{b}',

position: 'right',

color: '#333'

},

itemStyle: {

color: '#ddb926'

},

emphasis: {

label: {

show: true

},

itemStyle: {

shadowBlur: 10,

shadowColor: 'rgba(0, 0, 0, 0.5)'

}

}

}]

};

// 加载中国地图

fetch('https://geo.datav.aliyun.com/areas\_v3/bound/100000\_full.json')

.then(response => response.json())

.then(mapJson => {

echarts.registerMap('china', mapJson);

option.geo = {

map: 'china',

roam: true,

zoom: 1.2,

label: {

show: true,

color: '#333',

fontSize: 10

},

itemStyle: {

areaColor: '#f0f8ff',

borderColor: '#404a59',

borderWidth: 0.5

},

emphasis: {

label: {

show: true

},

itemStyle: {

areaColor: '#e6f7ff'

}

}

};

mapChart.setOption(option);

// 添加点击事件

mapChart.on('click', function (params) {

if (params.data) {

selectedIp = params.data.ip;

showPciDetails(params.data.pci\_info, params.data.ip);

}

});

});

}

function formatBytes(bytes) {

if (bytes >= 1000000000) return (bytes / 1000000000).toFixed(2) + ' GB';

if (bytes >= 1000000) return (bytes / 1000000).toFixed(2) + ' MB';

if (bytes >= 1000) return (bytes / 1000).toFixed(2) + ' KB';

return bytes + ' B';

}

function showPciDetails(pciInfo, ip) {

if (!pciInfo || pciInfo.length === 0) {

alert('该IP没有可用的PCI信息');

return;

}

document.getElementById('pci-ip').textContent = ip;

const container = document.getElementById('pci-container');

container.innerHTML = '';

pciInfo.forEach(item => {

const pciItem = document.createElement('div');

pciItem.className = 'pci-item';

let flagsInfo = '';

if (item.flags !== 'N/A') {

flagsInfo = `<div>Flags: ${item.flags}</div>`;

}

pciItem.innerHTML = `

<div class="pci-time">${item.time}</div>

<div>协议: <span class="pci-protocol">${item.protocol}</span></div>

<div>源端口: ${item.src\_port}</div>

<div>目标端口: ${item.dst\_port}</div>

${flagsInfo}

<div>大小: ${formatBytes(item.size)}</div>

`;

container.appendChild(pciItem);

});

document.getElementById('pci-details').style.display = 'block';

}

// 窗口大小变化时重绘图表

window.addEventListener('resize', function () {

mapChart.resize();

});

</script>

</body>

</html>