实验指导书

1. 实验背景

1.1. openwrt操作系统

路由器，作为现代网络连接的关键设备，在家庭、学校宿舍、商业场所等各类场景中广泛部署，是实现互联网接入的核心工具。从外观设计来看，不同品牌与型号的路由器各不相同，但从功能角度分析，大致可区分为普通路由器与软路由两类。对于大多数用户的日常上网行为，比如视频浏览、游戏娱乐及办公应用等基础需求，普通路由器即可有效满足。然而，当用户对网络功能有更高要求，如流量管控、 网络安全、虚拟专用网络以及负载均衡等进阶功能时，软路由则凭借其高度的灵活性与强大的扩展能力成为更优选择。

openwrt 是一个为嵌入式设备开发的高扩展度的linux操作系统，常用于搭建软路由。与许多其他路由器的发行版不同，OpenWrt 是一个完全为嵌入式设备构建的功能全面、易于修改的由现代 Linux 内核驱动的操作系统。openwrt 发展至今，已经具有超过3000个标准化应用软件包，这意味着在实践中它们几乎可以实现用户的任何功能。OpenWrt 不是一个单一且不可更改的固件，而是提供了一个完全可写的文件系统及软件包管理。对于开发人员来说，OpenWrt 是一个构建应用程序的框架，对于普通用户来说，这意味着拥有了完全定制的能力，能以意想不到的方式使用该设备。

关于openwrt系统的更多信息请参考官方网址https://openwrt.org/zh/start。

1.2. 流量监控

网络通信是通过数据包来完成的，所有信息都包含在网络通信数据包中。两台计算机通过网络“沟通”，是借助发送与接收数据包来完成的。流量监控实际上就是针对这些网络通信数据包进行管理与控制，同时进行优化与限制。流量监控的目的是允许并保证有用数据包的高效传输，禁止或限制非法数据包传输，一保一限是流量监控的本质。上网时用流量监控软件可以随时获得哪些程序正在访问互联网以及它们实时下载速度和上传速度。

2. 实验目的

1. 通过在 OpenWrt 操作系统上开展实验，深入掌握类 Linux 系统的常规操作流程与技巧，提升在类 Linux 环境下的实践能力。

2. 借助流量监控程序代码的编写工作，进一步深化对计算机网络理论知识的理解，同时在实践中强化编程技能，提高代码质量与开发效率。
3. 针对实验过程中可能出现的各类技术问题，通过主动分析、定位与解决，营造近似真实的生产环境，以此有效锻炼和提升学生在面对复杂工程场景时的应急处理与问题解决综合能力。

3. 实验要求

1. 自主完成 OpenWrt 操作系统虚拟机的安装部署，并在该虚拟机上开展本次实验。
2. 运用 C 语言进行编程开发，实现流量监控功能。需要获取的数据包括但不限于：
    a. 源IP地址/目的IP地址
    b. 累计接收(发送)流量
    c. 流量峰值
    d. 过去某个时间段的平均流量，例：过去2s、过去10s、过去40s
3. 加分项（任选其一即可）：
    a. 开发前端可视化界面与后端服务器接口。通过发送 HTTP 请求获取流量监控的实时数据信息，并将其直观呈现于前端界面，以实现数据的实时监测与可视化展示。
    b. 购买路由器固件，为其烧制openwrt操作系统，在真实的路由器环境中部署流量监控程序。

4. 实验原理

4.1. 虚拟机安装openwrt操作系统

1. 请确保自己的电脑上已经安装了可以正常使用的虚拟机软件，以vmware workstation虚拟机软件为例。
2. 从openwrt的镜像网站上下载最新版的openwrt操作系统镜像。以64位的x86架构系统为例，下载地址为：https://downloads.openwrt.org/releases/24.10.0/targets/x86/64/
3. 使用镜像格式转换工具，将从官网下载到的img格式的openwrt操作系统镜像转换为vmdk格式。示例镜像格式转换文件：StarWind V2V Converter
4. 使用转换好的镜像文件安装openwrt虚拟机，在安装过程中请根据自身需求和硬件条件进行基本配置，比如：内存大小、处理器个数等。

5. 关键：配置虚拟机网络。示例：选择虚拟机的网卡为**NAT模式**，分配静态ip，**并设置网关和dns列表**。
6. 可能遇到的问题：
    a. 虚拟机无网络连接。示例解决方式：
        i. 选择虚拟机的网卡为NAT模式，并确保虚拟机和宿主机处在同一网段之下。本机打开命令行，输入ipconfig，查看VMnet8的IPv4地址和子网掩码，可以获取到宿主机的网段。在openwrt虚拟机中输入vi /etc/config/network，查看lan口的ipaddr和netmask配置，可以获取到虚拟机的网段。
        ii. 为虚拟机配置网关。在openwrt虚拟机中输入vi /etc/config/network，在lan口的配置项中配置gateway，配置方式为option gateway '网关IP'。
        iii. 为虚拟机配置dns服务器。在openwrt虚拟机中输入vi /etc/config/network，在lan口的配置项中配置dns，配置方式为list dns 'dns服务器IP'。
        iv. 重启网络服务。执行命令/etc/init.d/network restart
    b. 向虚拟机中传送文件。由于向openwrt虚拟机中安装wmware tools的难度较高，推荐大家下载samba来向虚拟机中传送文件。
        i. 输入opkg update命令，更新软件源。
        ii. 输入opkg install luci-app-samba4，安装samba服务。
        iii. 可选：输入opkg install luci-i18n-samba4-zh-cn，对samba服务汉化。
        iv. 浏览器输入openwrt虚拟机的ip地址，进入图形化配置界面，选择services下的网络共享，进行samba配置。
        v. 勾选启用扩展调整、强制同步I/0。在编辑模块处，将invalid users=root注释掉。
        vi. 添加共享目录。名称自选，目录路径自选，以/mnt/p0为例。勾选可浏览、允许匿名用户，创建权限掩码设置为0777，目录权限掩码设置为0777。
        vii. 回到openwrt虚拟机中，创建对应目录。mkdir /mnt/p0。
        viii. 修改目录权限。chmod -R 777 /mnt/p0。
        ix. 使用方式：在宿主机中输入win+R，在提示符中输入"//虚拟机ip地址"，即可进入共享目录/mnt/p0。
    c. overlay空间不足。在安装gcc时，可能会遇到overlay空间不足的问题。此处提供两种解决方式：
        i. 使用vmdk磁盘安装虚拟机，由于无法在第一次安装时设置磁盘空间大小，需要对虚拟机进行扩容。

ii. 由于openwrt操作系统是为嵌入式系统搭建的操作系统，它本身就不适合执行类似编译这种占用CPU很高的任务，因此，建议在windows系统上对编写好的程序进行交叉编译，然后将编译好的可执行文件传入到虚拟机中运行。注意：如果涉及到动态链接库，需要设法将动态链接库也部署到虚拟机中，否则会导致程序无法正常运行。

4.2. libpcap原理

libpcap（Packet Capture Library）即数据包捕获函数库，是Unix/Linux平台下的网络数据包捕获函数库。它是一个独立于系统的用户层包捕获的API接口，为底层网络监测提供了一个可移植的框架。

libpcap可以实现以下功能：

1. 数据包捕获：捕获流经网卡的原始数据包
2. 自定义数据包发送：任何构造格式的原始数据包
3. 流量采集与统计：网络采集的中流量信息
4. 规则过滤：提供自带规则过滤功能，按需要选择过滤规则

libpcap主要由两部分组成：网络分接口(Network Tap)和数据过滤器(Packet Filter)。网络分接口从网络设备驱动程序中收集数据拷贝（旁路机制），过滤器决定是否接收该数据包。Libpcap利用BSD Packet Filter(BPF)算法对网卡接收到的链路层数据包进行过滤。BPF算法的基本思想是在有BPF监听的网络中，网卡驱动将接收到的数据包复制一份交给BPF过滤器，过滤器根据用户定义的规则决定是否接收此数据包以及需要拷贝该数据包的那些内容，然后将过滤后的数据给与过滤器相关联的上层应用程序。如果没有定义规则，则把全部数据交给上层应用程序。

4.3. 相关C语言API库简介

4.3.1. 数据包捕获与处理：头文件<pcap.h>

1. **打开一个网络接口以进行实时数据包捕获**：成功返回一个指向 `pcap_t` 的指针；失败返回 `NULL` 。

```c
pcap_t *pcap_open_live(const char *device, int snaplen, int promisc, int to_ms, char *errbuf);
```

形参	参数含义

const char *device	网络接口名称
int snaplen	捕获的数据包的最大长度
int promisc	是否将网络接口设置为混杂模式
int to_ms	超时时间（以毫秒为单位）
char *errbuf	用于存储错误信息的缓冲区

2. **捕获数据包并调用指定的处理函数**：成功返回 0；失败返回 -1。

```c
int pcap_loop(pcap_t *p, int cnt, pcap_handler callback, unsigned char *user);
```

形参	参数含义
pcap_t *p	pcap 描述符
int cnt	要捕获的数据包数量，-1表示无限循环
pcap_handler callback	数据包处理函数
unsigned char *user	用户自定义数据

3. **关闭pcap描述符，释放资源：**

```c
void pcap_close(pcap_t *p);
```

形参	参数含义
pcap_t *p	pcap 描述符

4. **编译数据包过滤规则**：成功返回 0；失败返回 -1。

```c
int pcap_compile(pcap_t *p, struct bpf_program *fp, const char *str, int optimize, bpf_u_int32 netmask);
```

形参	参数含义
pcap_t *p	pcap描述符
struct bpf_program *fp	用于存储编译后的过滤程序
const char *str	过滤规则字符串
int optimize	是否优化过滤规则
bpf_u_int32 netmask	子网掩码

5. **将编译后的过滤规则应用到捕获描述符上**：成功返回 0；失败返回 –1。

```c
int pcap_setfilter(pcap_t *p, const struct bpf_program *fp);
```

形参	参数含义
pcap_t *p	pcap描述符
struct bpf_program *fp	编译后的过滤程序

4.3.2. 线程管理：头文件<pthread.h>

1. **创建新线程**：成功返回 0，失败返回错误码。

```c
int pthread_create(pthread_t *thread, const pthread_attr_t *attr, void *(*start_routine) (void *), void *arg);
```

形参	参数含义
pthread_t *thread	指向线程标识符的指针
const pthread_attr_t *attr	线程属性，通常为 `NULL` 表示默认属性
void *(*start_routine) (void *)	线程执行的函数
void *arg	传递给线程函数的参数

2. **等待线程结束并获取返回值**：成功返回 0；失败返回错误码。

```c
int pthread_join(pthread_t thread, void **res);
```

形参	参数含义
pthread_t thread	要等待的线程标识符
void **res	用于存储线程返回值的指针（可选）

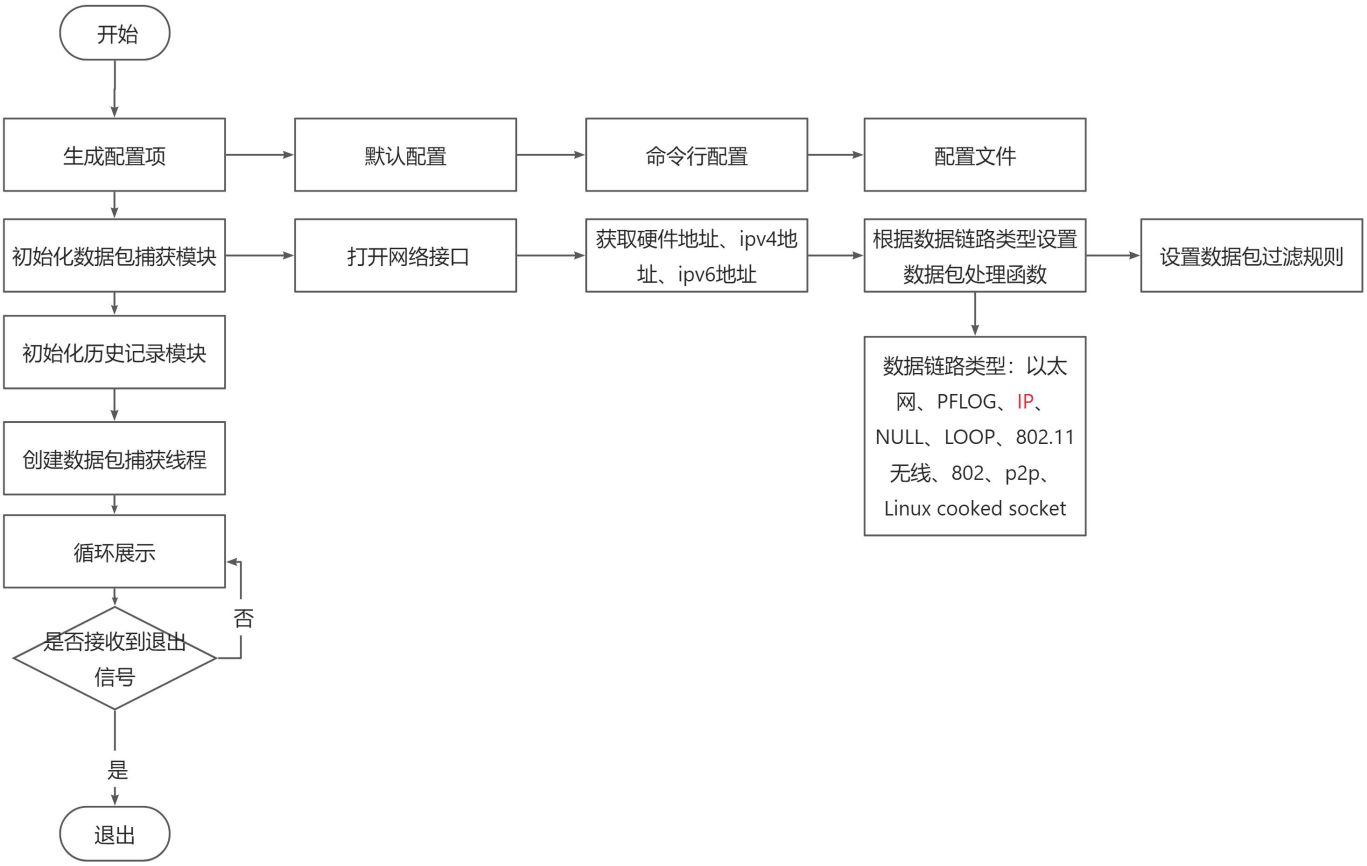
3. **取消线程**：成功返回 0；失败返回错误码。

```c
int pthread_cancel(pthread_t thread);
```

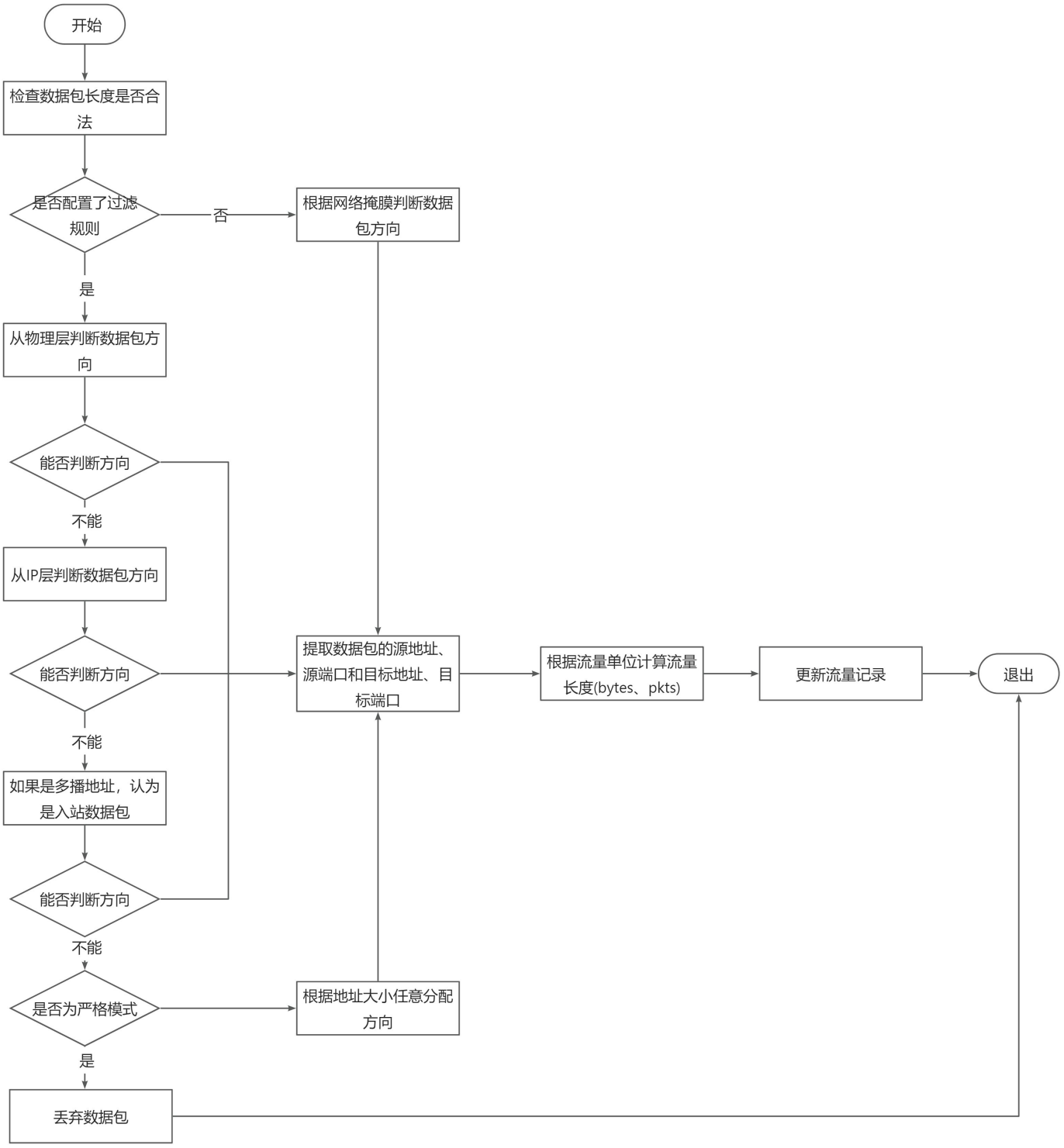
5. 示例

5.1. 流程示例

1. 整体流程示例：



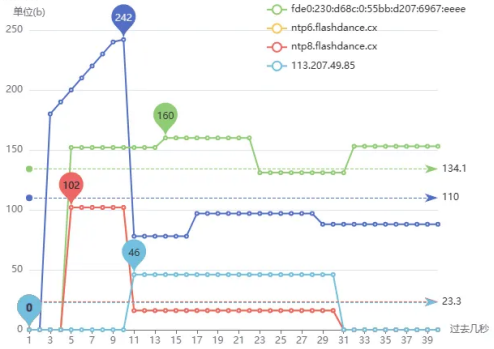
2. IP数据包解析流程示例：



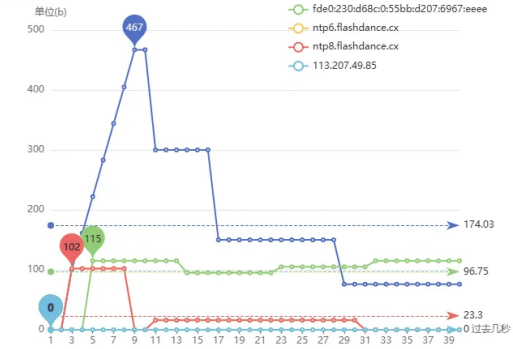
5.2. 运行示例

```
192.168.60.254                           => public-dns-a.baidu.com                0b     171b     110b
                                         <=                                       0b     245b     174b
fde0:230:d68c::1                         => fde0:230:d68c:0:55bb:d207:6967:ecce   0b      91b     134b
                                         <=                                       0b      69b      97b
192.168.60.254                           => ntp6.flashdance.cx                    0b      61b      23b
                                         <=                                       0b      61b      23b
192.168.60.254                           => ntp8.flashdance.cx                    0b      61b      23b
                                         <=                                       0b      61b      23b
192.168.60.254                           => 113.207.49.85                         0b       0b      23b
                                         <=                                       0b       0b       0b
```

发送速率



接收速率



目的(源)ip地址	数据包方向	累计流量	峰值流量	平均流量（过去2s）	平均流量（过去10s）	平均流量（过去40s）
public-dns-a.baidu.com	发送	4400	242	0	171	110
	接收	6961	467	0	245	174
public-dns-a.baidu.com	发送	5364	160	0	91	134
	接收	3870	115	0	69	97
public-dns-a.baidu.com	发送	932	102	0	61	23
	接收	932	102	0	61	23
public-dns-a.baidu.com	发送	932	102	0	61	23