第六章 流量抽样实验

随着高速网络技术的发展，使得实时在线的流量测量非常困难，而基于抽样的流量测量方法作为一种可扩展的技术已经成为研究人员经常采用的一种有效的流量测量技术，网络测量一般采用主动测量和被动测量两种测量方式，而抽样测量技术是针对高速网络测量提出的一种被动测量手段。除了要在理论层面上了解抽样测量如何进行，更要通过实验过程深入的探究其基本原理。

本章共设置了四个实验，难度由浅入深。其中，实验一是基于周期抽样的流量抽样，注重于对流量抽样的认识，是一个简易的流量抽样程序；实验二是基于随机抽样的流量抽样，在了解网络流量具有高速率性质的基础上，根据预先定义的随机过程来确定抽样的起点和抽样间隔，此方法样本之间是相互独立的，能够实时生成具有高度独立性的随机数，避免了周期抽样导致的同步影响；实验三是基于掩码匹配的流量抽样，其本质上是基于报文内容的流量抽样，使用预定的比特掩码与待抽样的 IP 报头中的若干比特位进行匹配，如果匹配成功就抽取此 IP 报文。这种基于掩码匹配的抽样测量适用于多种网络环境，具有较高的效率和准确性；实验四是基于多掩码匹配的流量抽样，其是实验三的拓展，解决了实验三中至多只有 16 种抽样比率的局限性，通过将不同的抽样掩码组合以实现任意抽样比率。

实验 1：Linux 下使用 libpcap 进行基于周期抽样的流量抽样

一、实验目的

在 Linux 环境下，使用 libpcap 对检测到的报文实现等间距的抽样。

具体要求：使用 libpcap 对 99 个报文进行间距为 3 的周期抽样，并将采集到的报文保存为 pcap 文件。

二、实验基本原理

1、预备知识

本章实验需要你了解前面章节的报文抓取等实验的原理。

2、周期抽样

周期抽样采用相同的间隔对网络数据包进行采集，如每隔 N 秒钟产生 1 次抽样或每次从 N 个分组中抽样第一个分组，本章实验使用的是后者的方法。该方法简单易行，缺点是测量具有周期性和可预测性，使被测网络陷入同步状态。具体来说，如果采集的数据包本身具有周期性的行为，那么抽样过程将仅仅得到周期性行为的一部分，这样会使得抽样在较大程度上不能真实反映出被测对象的全部特性。这里我们使用计数取余的方法实现周期抽样，首先定义抽样间隔 n，然后为每个到来的报文标记序号 i，对该序号取余，即 i%n,余数为 0 则抽取该报文。

3、流量抽样系统的体系结构

本章的流量抽样测量系统体系结构均由报文采集、报文抽样和信息保存三层组成，每层实现相对独立的功能

报文采集层：其主要功能是获取经过网络的数据包。为了监听网络上所有流经数据链路层的报文，网卡应被设置为混杂模式，通过 libpcap 提供了核心函数 pcap_loop()来实现持续抓包，具体参数设置可以参考实验三中报文抓取的实验原理。

报文抽样层：其主要功能是利用概率与数理统计原理，采用一定的抽样方法，对采集的报文进行抽样处理。通过在 pcap_loop()的回调函数中，设置我们要进行抽样的规则或算法即可实现报文的抽样。

信息保存层：其主要功能是将抽样后的报文信息写入存储器，我们这里借助 libpcap 的 pcap_dump()系列函数来实现报文信息的保存。

三、实验步骤

下面，我们开始用 C 语言编写一个基于周期抽样的流量抽样程序。这里按照顺序列出了其中的关键步骤，源码为附录中的 Periodic_sampling.c 文件。

1、获取网络接口-find_alldevs()

2、打开网络接口-pcap_open_live()

3、打开用于保存捕获数据包的文件-pcap_dump_open()

pcap_dump_open() 会返回存储文件的地址。

它的函数原型为：

```
pcap_dumper_t *pcap_dump_open(pcap_t *p, char *fname)
```

- p 参数为调用 pcap_open_offline()或 pcap_open_live()函数后返回的 pcap 结构指针；
- fname 参数指定打开的文件名；

4、捕获数据包-pcap_loop()

libpcap 提供了核心函数 pcap_loop()来实现持续抓包，通过在回调函数中添加抽样的规则和算法即可完成抽样，

它的函数原型为：

```
int pcap_loop(pcap_t * p, int cnt, pcap_handler callback, u_char * user)
```

- p：第 2 步返回的 pcap_t 类型的指针；
- cnt：需要抓的数据包的个数，一旦抓到了 cnt 个数据包，pcap_loop 立即返回。负数的 cnt 表示 pcap_loop 永远循环抓包，直到出现错误。
- callback：一个回调函数指针，它必须是如下的形式：

```
void callback(u_char * user, const struct pcap_pkthdr * pkthdr, const u_char * packet)
```

在回调函数中，对每个到达的报文标记序号，并对其取余，余数为 0 则抽取该报文。

- user：传递了用户自定义一些数据，是 pcap_loop 的最后一个参数，进而传递给回调函数；
- pkthdr：收到的数据包的 pcap_pkthdr 类型的指针，表示捕获到的数据包基本信息，包括时间,长度等信息；
- data：收到的数据包数据。

5、关闭用于保存捕获数据包的文件-pcap_dump_close()

以上为实验一的主要内容，抽样原理简单易懂，完整源码及运行结果可参考附录。

实验 2：Linux 下使用 libpcap 进行随机抽样的流量抽样

一、实验目的

在 Linux 环境下，使用 libpcap 对检测到的报文实现随机抽样。

具体要求：使用 libpcap 对 500 个报文进行随机概率为 50%的周期抽样，并将采集到的报文保存为 pcap 文件。

二、实验基本原理

1、高速网络流量下随机数的生成

随机抽样根据一定的概率规则对数据总体进行抽样，从包含 N 个个体的数据总体中抽取出 n 个个体组成样本（N>n），每个个体被抽取的概率相同。本实验中，我们随机抽样的方法是对每个报文生成高度独立的随机数，对随机数取余，除数是 100，然后根据设定的抽样的概率，判断该余数在哪个余数空间（如抽样概率为 50%，则判定该余数是否小于 50，若小于 50，则抽样该报文）。

在随机数生成环节，利用 C 语言的 srand()函数和 rand()函数，其函数原型如下：

```
void srand(unsigned int seed)
int rand(void)
```

在调用 rand()函数之前，可以使用 srand()函数设置随机数种子，如果没有设置随机数种子，rand()函数在调用时，自动设置随机数种子为 1。而如果随机种子相同，则每次产生的随机数也会相同。

考虑到实验环境下每秒有近百条报文的网络情况，仅仅使用基于秒级的系统时间作为随机数种子则会产生大量相同的随机数，因此，本实验借助 C 语言中的 sys/time.h 头文件，提取每个报文到达时的系统微秒级时间作为随机数种子，保证了高度独立的随机性。

2、流量抽样系统的体系结构

本实验的体系结构与实验一相同，也是由报文采集、报文抽样和信息保存三层组成，只是在报文抽样里的抽样规则发生了改变，根据预先定义的随机过程来确定抽样的起点和抽样间隔，此方法样本之间是相互独立的，避免了实验一中周期抽样导致的同步影响。

三、实验步骤

1、完成实验一

首先，依据实验一的三层结构搭建流量抽样系统。实验二相比实验一的区别，主要体现在抽样规则上。

2、获取网络接口-find_alldevs()

3、打开网络接口-pcap_open_live()

4、打开用于保存捕获数据包的文件-pcap_dump_open()

5、捕获数据包-pcap_loop()

处理抓取到的数据包-process_packet()，在抓取到数据包后，我们还可以进一步地来分析数据包。这里利用 pcap_loop()中的第三个参数回调函数来处理抓取到的数据包，本实验采用的是自定义的回调函数 process_packet()，具体操作如下：

5(a)、获取报文到达时的系统微秒级时间-sysTime()

sysTime()是用户自定义的函数，需要引用头文件<sys/time.h>，利用 gettimeofday()和

localtime()两个函数提取当前系统微秒级时间，并返回该数值。

5(b)、生成随机数-srand()和 rand()

利用上述步骤中返回的微秒级时间作为随机数种子，输出具有高度独立性的随机数。

5(c)、判定是否抽样

根据预先定义的抽样率，对随机数进行取余，除数是 100，判断余数在是否小于阈值，如果小于，则对该报文抽样。

6、关闭用于保存捕获数据包的文件-pcap_dump_close()

以上为实验二的主要内容，主要功能是在报文到达时能够实时生成高度独立的随机数，完整源码及运行结果可参考附录。

实验 3：Linux 下使用 libpcap 进行掩码匹配的流量抽样

一、实验目的

在 Linux 环境下，使用 libpcap 对检测到的报文实现基于标识字段的掩码抽样。

具体要求：使用 libpcap 对 500 个报文进行指定位数的掩码抽样（如输入位数为 3，则抽样概率为 $1/2^3$），并将采集到的报文保存为 pcap 文件。

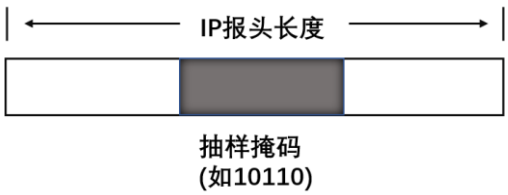
二、实验基本原理

1、IP 报文头的组成

这部分内容是计算机网络的基础，可以查阅相关文献书籍得知。

2、基于标识字段的掩码抽样原理

基于掩码匹配的抽样测量是一种基于统计分析 的流量抽样测量方法，实际上是一种基于报文内容的抽样技术，使用预定的比特掩码与待抽样的 IP 报头中的若干比特位进行匹配，如果匹配成功就抽取此 IP 报文。这种基于掩码匹配的抽样测量是用于多种网络测量环境，具有较高的效率和准确性。基于掩码匹配的抽样测量以比特串匹配为基础，通过比较比特掩码和每个报文中的特定比特串来确定报文抽样与否，比特掩码长度直接影响测量系统的精度和效率，如下图所示：



理论上，抽样比率由抽样掩码比特长度决定，若掩码长度为 m，则存在 $M=2^m$ 掩码取值，理论上的数据包抽样比率 p=1/M。而 IP 报文的标识字段随机性高且相互独立，非常适合充当掩码匹配比特串，并且可以通过调整 m 来控制抽样样本的数量。

三、实验步骤

1、获取网络接口-find_alldevs()

2、根据输入的抽样比特位数，得到抽样掩码

3、打开网络接口-pcap_open_live()

4、打开用于保存捕获数据包的文件-pcap_dump_open()

5、捕获数据包-pcap_loop()

处理抓取到的数据包-process_packet()，在抓取到数据包后，我们还可以进一步地来分析数据包。这里利用 pcap_loop()中的第三个参数回调函数来处理抓取到的数据包，本实验采用的是自定义的回调函数 process_packet()，具体操作如下：

依据步骤 2 得到的抽样掩码，执行抽样算法-ip_character_mask()函数，然后根据返回结果判定是否对该报文抽样。

6、关闭用于保存捕获数据包的文件-pcap_dump_close()

以上为实验三的主要内容，主要功能是基于报文标识的掩码匹配抽样算法，完整源码及运行结果可参考附录。

实验 4：Linux 下使用 libpcap 进行多掩码匹配的流量抽样

一、实验目的

在 Linux 环境下，使用 libpcap 对检测到的报文实现基于标识字段的多掩码抽样。

具体要求：使用 libpcap 对 500 个报文进行任意指定概率的多掩码匹配抽样，并将采集到的报文保存为 pcap 文件。

二、实验基本原理

1、基于标识字段的多掩码抽样原理

实验三中描述掩码法定义一串字符串作为掩码 M，掩码长度 L，如果标识字段的指定比特串同掩码 M 匹配，则抽样该报文，否则丢弃报文。理论上，长度 L 的掩码抽样的比率为 $1/2^L$。掩码法的缺点是其控制的抽样比率范围很小，长度为 16 比特的标识字段仅有 16 种抽样比率，分别为 1/2、$1/2^2$、$1/2^3$、$1/2^4$、$1/2^5$、$1/2^6$、$1/2^7$、$1/2^8$、$/1/2^9$、$1/2^{10}$、$1/2^{11}$、$1/2^{12}$、$1/2^{13}$、$1/2^{14}$、$1/2^{15}$、$1/2^{16}$。这 16 种抽样比率无法描述实际测量需求，为此本实验提出将不同的抽样掩码组合以实现任意抽样比率。

使用多个抽样掩码，使得不同抽样掩码对应的抽样比率能够直接叠加，必须保证不同 的抽样掩码之间独立不相关。设 n 个抽样掩码的长度分别为 L_i：i = 1 to n，其对应的抽样比率分别为 $1/2^L$，则使用这 n 个抽样掩码的抽样概率为：

$$ratio = f(\sum_{i=1}^{n} 1/2^{L_i})$$

如果这 n 个掩码之间是独立不相关的，则抽样概率为：

$$ratio = f\left(\sum_{i=1}^{n}\frac{1}{2^{L_i}}\right) = \sum_{i=1}^{n}\frac{1}{2^{L_i}}$$

为此抽样掩码的定义规则为：

(1) 抽样掩码第 1 个比特对应标识字段第 1 个比特，即偏移为 0；

(2) 长度为 L 的抽样掩码，其前面 L-1 个比特取值为 0，第 L 个比特取值为 1。

根据以上规则，抽样参数定义为：设 n 个抽样掩码长度分别为 L_i：i = 1 to n，其掩码值定义为子掩码的抽样参数，即如果子抽样掩码长度为 L_i，则根据抽样掩码定义规则，其子抽样参数 parameter = 2^{L_i}，同时，知道抽样参数，可以将其转化为用 2 进制表示的抽样掩码 因此根据上面规则，n 个子抽样掩码的参数定义为：

$$parameter = \sum_{2^{L_i}}^{n} 2^{L_i}$$

由于任何整数可以用多个不同的 2i 组合得到因此可以通过对 parameter 进行 2 幂分解得到各个抽样掩码的抽样参数，任何抽样参数可以转化为抽样掩码，下面给出多掩码抽样算法及其参数转化算法。

2、基于标识字段的多掩码抽样算法

2(a)、抽样比率分解算法

任意抽样比率 ratio∈(0,1), 使用以下的抽样比率分解算法将其转化为多个子抽样掩码：

```
设抽样参数为 parameter = 0;
for i from 1 to 16
{   if ratio>=1/2^i then
        ratio = ratio – 1/2^i
        parameter = parameter + 2^i
    end if   }
return parameter
```

抽样比率分解算法可以保证抽样比率可以在 16 次循环之内转化为抽样参数，其抽样精度可控制在 1/65536 范围之内，相对精度误差为 1/65536*ratio。

2(b)、参数分解算法

测量器得到抽样参数以后, 可以使用以下参数分解算法将参数分解为对应的抽样掩码长度：

```
设抽样掩码参数存放数组为 mask[ ] = 0;
抽样掩码个数 number = 0;
for i from 1 to 16
{   if parameter>=2^i then
        mask[number] = i
        number++;
    end if   }
return number , mask[ ]
```

2(c)、抽样算法

知道抽样掩码长度，根据抽样掩码定义规则，即可知道对应的抽样掩码，对于到达的报

文，根据由以下的抽样算法决定是否抽样报文：

```
设到达报文的标识字段第一个出现非零的位置是第 L 个比特;
bool flag = 0; //如果 flag 为 0 表示该报文不抽样
for i from 0 to number-1
{   if mask[ i ] == L then
          flag = 1
          return flag;
    end if   }
return flag
```

下面举例分析各算法，设要求抽样比率为 0.638，根据抽样比率分解算法将 0.638 分解为： 0.638 $=1/2+1/2^3+1/2^7+1/2^8+1/2^{10}+1/2^{12}+1/2^{15}+1/2^{16}$ +0.0000148。相应的抽样参数 parameter = $2+2^3+2^7+2^8+2^{10}+2^{12}+2^{15}+2^{16}$ = 103818。分析器可以将抽样参数 103818 传送到测量器，测量器根据抽样参数分解算法将抽样参数分解成子掩码长度，其 number = 8，抽样掩码位置数组为 mask[ ] ={1,3,7,8,10,12,15,16} ，其对应的各子抽样掩码分别为 1、001、0000001、00000001、0000000001、000000000001、0000000000000001、0000000000000001。最后根据抽样算法抽样测量到达的报文。其中抽样比率分解算法和抽样参数分解算法为测量之前的任务，而抽样算法为测量过程中使用的算法。

三、实验步骤

1、完成实验三

本实验是在实验三掩码匹配抽样的基础上，进行了拓展，实现基于报文标识的多掩码匹配，能够以任意比率进行流量抽样。

2、获取网络接口-find_alldevs()

3、根据输入的抽样比率，计算出抽样参数并分解-Sampling_ratio_decomposition()

4、打开网络接口-pcap_open_live()

5、打开用于保存捕获数据包的文件-pcap_dump_open()

6、捕获数据包-pcap_loop()

处理抓取到的数据包-process_packet()，在抓取到数据包后，我们还可以进一步地来分析数据包。这里利用 pcap_loop()中的第三个参数回调函数来处理抓取到的数据包，本实验采用的是自定义的回调函数 process_packet()，具体操作如下：

依据步骤 3 得到的子掩码长度数组，执行抽样算法-multmask_matching()函数，然后根据返回结果判定是否对该报文抽样。

7、关闭用于保存捕获数据包的文件-pcap_dump_close()

8、注意事项

此实验中需要注意的地方有：

① 基本的抽样系统架构均为报文采样，报文抽样，信息保存这三层。
② 此实验重在对抽样算法的设计，需要深入理解其原理。
③ 鉴于实验室有限的网络环境，实验过程中可能存在误差，多切换网卡设备。
④ 实验函数设计时多考虑低耦合高内聚，方便代码的移植。

以上为实验四的主要内容，主要功能是基于报文标识的多掩码抽样算法，完整源码及运行结果可参考附录。

附录

1、源码

实验一：Periodic_sampling.c

```
/************************************************************
*       File: Periodic_sampling.c
*       Experiment 1: Use libpcap to capture and sample packets Periodically
*       print the details of the captured packets
*       in each computer network hierarchy protocol,
*       and save them to a pcap file.
*
*       To compile:
*       >gcc Periodic_sampling.c -lpcap -o Periodic_sampling
*
*       To run:
*       >sudo ./Periodic_sampling
*
*       Enter the number of the device you want to sniff:
*       >1
*
************************************************************/

#include<pcap.h>
#include<stdio.h>
#include<stdlib.h> // for exit()
#include<string.h> //for memset
#include<sys/socket.h>
#include<arpa/inet.h> // for inet_ntoa()
#include<net/ethernet.h>
#include<netinet/ip.h>//Provides declarations for ip header
#include <unistd.h>

void process_packet(u_char *, const struct pcap_pkthdr *, const u_char *);

int tcp=0,udp=0,icmp=0,others=0,igmp=0,total=0,i,j;
pcap_t *handle;
int timeLen = 1;
int count = 1;
int interval; //Sampling interval
```

```
int main()
{
	pcap_if_t *alldevsp , *device;
	pcap_t *handle; //Handle of the device that shall be sniffed
	pcap_dumper_t* out_pcap;

	char errbuf[100] , *devname , devs[100][100];
	int count = 1 , n;

	//First get the list of available devices
	printf("Finding available devices ... ");
	if( pcap_findalldevs( &alldevsp , errbuf) )
	{
		printf("Error finding devices : %s" , errbuf);
		exit(1);
	}
	printf("Done");

	//Print the available devices
	printf("\nAvailable Devices are :\n");
	for(device = alldevsp ; device != NULL ; device = device->next)
	{
		printf("%d. %s - %s\n" , count , device->name , device->description);
		if(device->name != NULL)
		{
			strcpy(devs[count] , device->name);
		}
		count++;
	}

	//Ask user which device to sniff
	printf("Enter the number of the device you want to sniff: ");
	scanf("%d" , &n);
	devname = devs[n];

	//Ask user the sampling interval
	printf("Enter the interval of the sampling: ");
	scanf("%d" , &interval);

	printf("\nOpening device %s for sniffing ... " , devname);
	handle = pcap_open_live(devname , 65536 , 0 , -1 , errbuf);

	if (handle == NULL)
	{
```

```
            fprintf(stderr, "Couldn't open device %s : %s\n" , devname , errbuf);
            exit(1);
        }
        printf("Done\n");

        // open pcap file and wait the packet update
        out_pcap =
pcap_dump_open(handle,"/home/jiapengli/sampling/Periodic_sampling.pcap");

        // Put the device in sniff loop
        pcap_loop(handle , 99 , process_packet , (u_char*)out_pcap);

        // refresh buffer
        pcap_dump_flush(out_pcap);

        // close resource
        pcap_dump_close(out_pcap);
        return 0;
}

/**********************************************************************************************
*************/
void process_packet(u_char *args, const struct pcap_pkthdr *header, const u_char *buffer)
{

        if(count%interval==0) //Sampling interval
        {
            // Export traffic data to pcap file
            pcap_dump(args, header, buffer);

            //Get the IP Header part of this packet , excluding the ethernet header
            struct iphdr *iph = (struct iphdr*)(buffer + sizeof(struct ethhdr));
            ++total;
            switch (iph->protocol) //Check the Protocol and do accordingly...
            {
                case 1:  //ICMP Protocol
                    ++icmp;
                    break;

                case 2:  //IGMP Protocol
                    ++igmp;
                    break;

                case 6:  //TCP Protocol
```

```
                    ++tcp;
                    break;

                case 17: //UDP Protocol
                    ++udp;
                    break;

                default: //Some Other Protocol like ARP etc.
                    ++others;
                    break;
            }
            printf("TCP : %d   UDP : %d   ICMP : %d   IGMP : %d   Others : %d
Total : %d\n", tcp , udp , icmp , igmp , others , total);

        }
        ++count;
}
```

实验二： Random_sampling.c

```
/************************************************************
*       File: Random_sampling.c
*
*       Experiment 2: Use libpcap to capture and sample packets randomly
*       print the captured packets
*       in each computer network hierarchy protocol,
*       and save them to a pcap file.
*
*       To compile:
*       >gcc Random_sampling.c -lpcap -o Random_sampling
*
*       To run:
*       >sudo ./Random_sampling
*
*       Enter the number of the device you want to sniff:
*       >1
*
*
************************************************************/

#include<pcap.h>
#include<stdio.h>
#include<stdlib.h> // for exit()
#include<string.h> //for memset
```

```
#include<sys/socket.h>
#include<arpa/inet.h> // for inet_ntoa()
#include<net/ethernet.h>
#include<netinet/ip.h>//Provides declarations for ip header
#include <unistd.h>
#include <time.h>
#include <sys/time.h>

void process_packet(u_char *, const struct pcap_pkthdr *, const u_char *);
int sysTime(void);


int tcp=0,udp=0,icmp=0,others=0,igmp=0,total=0,i,j;
pcap_t *handle;
int threshold;


int main()
{
	pcap_if_t *alldevsp , *device;
	pcap_t *handle; //Handle of the device that shall be sniffed
	pcap_dumper_t* out_pcap;

	char errbuf[100] , *devname , devs[100][100];
	int count = 1 , n;

	//First get the list of available devices
	printf("Finding available devices ... ");
	if( pcap_findalldevs( &alldevsp , errbuf) )
	{
		printf("Error finding devices : %s" , errbuf);
		exit(1);
	}
	printf("Done");

	//Print the available devices
	printf("\nAvailable Devices are :\n");
	for(device = alldevsp ; device != NULL ; device = device->next)
	{
		printf("%d. %s - %s\n" , count , device->name , device->description);
		if(device->name != NULL)
		{
			strcpy(devs[count] , device->name);
		}
```

```
            count++;
        }

        //Ask user which device to sniff
        printf("Enter the number of the device you want to sniff: ");
        scanf("%d" , &n);
        devname = devs[n];

        //Ask user the sampling ratio
        printf("Enter the threshold of the sampling ratio: ");
        scanf("%d" , &threshold);

        printf("\nOpening device %s for sniffing ... " , devname);
        handle = pcap_open_live(devname , 65536 , 0 , -1 , errbuf);

        if (handle == NULL)
        {
            fprintf(stderr, "Couldn't open device %s : %s\n" , devname , errbuf);
            exit(1);
        }
        printf("Done\n");

        // open pcap file and wait the packet update
        out_pcap =
pcap_dump_open(handle,"/home/jiapengli/sampling/random_sampling.pcap");

        // Put the device in sniff loop ,
        pcap_loop(handle , 500 , process_packet , (u_char*)out_pcap);

        // refresh buffer
        pcap_dump_flush(out_pcap);

        // close resource
        pcap_dump_close(out_pcap);
        return 0;
}

/*********************************************************************************************
*************/
void process_packet(u_char *args, const struct pcap_pkthdr *header, const u_char *buffer)
{
        int temp;
        srand((unsigned)sysTime() );
        temp=rand()%100+1;
```

```
    if(temp < threshold) //Sampling ratio
    {
        // Export traffic data to pcap file
        pcap_dump(args, header, buffer);

        //Get the IP Header part of this packet , excluding the ethernet header
        struct iphdr *iph = (struct iphdr*)(buffer + sizeof(struct ethhdr));
        ++total;
        switch (iph->protocol) //Check the Protocol and do accordingly...
        {
            case 1:  //ICMP Protocol
                ++icmp;
                break;

            case 2:  //IGMP Protocol
                ++igmp;
                break;

            case 6:  //TCP Protocol
                ++tcp;
                break;

            case 17: //UDP Protocol
                ++udp;
                break;

            default: //Some Other Protocol like ARP etc.
                ++others;
                break;
        }
        printf("TCP : %d   UDP : %d   ICMP : %d   IGMP : %d   Others : %d
Total : %d\n", tcp , udp , icmp , igmp , others , total);
    }
}

int sysTime(void)
{
    struct timeval tv;
    struct timezone tz;
    struct tm *t;
    gettimeofday(&tv, &tz);
    t = localtime(&tv.tv_sec);
    printf("time_now:%d-%d-%d %d:%d:%d.%ld\n", 1900+t->tm_year, 1+t->tm_mon,
```

```
t->tm_mday, t->tm_hour, t->tm_min, t->tm_sec, tv.tv_usec);
	int now_T =
t->tm_year+t->tm_mon+t->tm_mday+t->tm_hour+t->tm_min+t->tm_sec+tv.tv_usec;
	return now_T;
}
```

实验三： Mask_match_sampling.c

```
/*************************************************************
*	File: Mask_match_sampling.c
*
*	Experiment 3: Use libpcap to capture and sample packets based on Mask matching
*	Match the IP identification information with the mask
*	print the captured packets
*	in each computer network hierarchy protocol,
*	and save them to a pcap file.
*
*	To compile:
*	>gcc Mask_match_sampling.c -lpcap -o Mask_match_sampling
*
*	To run:
*	>sudo ./Mask_match_sampling
*
*	Enter the number of the device you want to sniff:
*	>1
*
************************************************************/

#include<pcap.h>
#include<stdio.h>
#include<stdlib.h> // for exit()
#include<string.h> //for memset
#include<sys/socket.h>
#include<arpa/inet.h> // for inet_ntoa()
#include<net/ethernet.h>
#include<netinet/ip.h>//Provides declarations for ip header
#include <unistd.h>


void process_packet(u_char *, const struct pcap_pkthdr *, const u_char *);
int ip_character_mask(const u_char * Buffer);
void tobin(u_int16_t a,char* str);

int tcp=0,udp=0,icmp=0,others=0,igmp=0,total=0,i,j;
```

```
pcap_t *handle;
int m;
char cpre[16] = {0};

int main()
{
	pcap_if_t *alldevsp , *device;
	pcap_t *handle; //Handle of the device that shall be sniffed
	pcap_dumper_t* out_pcap;

	// m = strlen(cpre);
	char errbuf[100] , *devname , devs[100][100];
	int count = 1 , n;

	//First get the list of available devices
	printf("Finding available devices ... ");
	if( pcap_findalldevs( &alldevsp , errbuf) )
	{
		printf("Error finding devices : %s" , errbuf);
		exit(1);
	}
	printf("Done");

	//Print the available devices
	printf("\nAvailable Devices are :\n");
	for(device = alldevsp ; device != NULL ; device = device->next)
	{
		printf("%d. %s - %s\n" , count , device->name , device->description);
		if(device->name != NULL)
		{
			strcpy(devs[count] , device->name);
		}
		count++;
	}

	//Ask user which device to sniff
	printf("Enter the number of the device you want to sniff: ");
	scanf("%d" , &n);
	devname = devs[n];

	//Ask user the number of the bits used for mask sampling
	printf("Enter the number of the bits used for mask sampling: ");
	scanf("%d" , &m);
	for(int i=0;i<m;i++)
```

```
	{
		cpre[i]='1';
	}

	printf("\nOpening device %s for sniffing ... " , devname);
	handle = pcap_open_live(devname , 65536 , 0 , -1 , errbuf);

	if (handle == NULL)
	{
		fprintf(stderr, "Couldn't open device %s : %s\n" , devname , errbuf);
		exit(1);
	}
	printf("Done\n");

	// open pcap file and wait the packet update
	out_pcap                                                                    =
pcap_dump_open(handle,"/home/jiapengli/sampling/Mask_match_sampling.pcap");

	// Put the device in sniff loop
	pcap_loop(handle , 100 , process_packet , (u_char*)out_pcap);

	// refresh buffer
	pcap_dump_flush(out_pcap);

	// close resource
	pcap_dump_close(out_pcap);
	return 0;
}

/*********************************************************************************************
*************/
void process_packet(u_char *args, const struct pcap_pkthdr *header, const u_char *buffer)
{
	if(ip_character_mask(buffer)) //Sampling interval
	{
		// Export traffic data to pcap file
		pcap_dump(args, header, buffer);

		//Get the IP Header part of this packet , excluding the ethernet header
		struct iphdr *iph = (struct iphdr*)(buffer + sizeof(struct ethhdr));
		++total;
		switch (iph->protocol) //Check the Protocol and do accordingly...
		{
			case 1:  //ICMP Protocol
```

```
                        ++icmp;
                        break;

                case 2:    //IGMP Protocol
                        ++igmp;
                        break;

                case 6:    //TCP Protocol
                        ++tcp;
                        break;

                case 17: //UDP Protocol
                        ++udp;
                        break;

                default: //Some Other Protocol like ARP etc.
                        ++others;
                        break;
            }
            printf("TCP : %d      UDP : %d      ICMP : %d      IGMP : %d      Others : %d
Total : %d\n", tcp , udp , icmp , igmp , others , total);
        }
}

int ip_character_mask(const u_char * Buffer)
{
        unsigned short iphdrlen;
        u_int16_t ip_id;
        char str[16] = {0};
        struct iphdr *iph = (struct iphdr *)(Buffer    + sizeof(struct ethhdr) );

        ip_id= ntohs(iph->id);
        // Convert IP_identification to binary
        tobin(ip_id,str);
        int len = strlen(str);
        // Mask matching
        for(int i =0;i<m;i++)
        {
                if(cpre[m-i-1] != str[len-i-1])
                return 0;
        }
        printf( "     |-Identification       : %d\n",ntohs(iph->id));
        printf("%s\n",str);
        return 1;
```

```
}

void tobin(u_int16_t a,char* str)
{
	// p points to the first address of a
	char *p=(char*)&a,c=0,f=0,pos=-1;
	for(int o=0;o<2;++o)
	{
		for(int i=0;i<8;++i)
		{
		c=p[1-o]&(1<<(7-i));
		if(!f&&!(f=c))continue;
		str[++pos]=c?'1':'0';
		}
	}
}
```

实验四： MultiMask_match_sampling.c

```
/************************************************************
*	File: MultiMask_match_sampling.c
*
*	Experiment 4: Use libpcap to capture and sample packets based on MultiMask matching
*	Match the IP identification information with the mask
*	print the captured packets
*	in each computer network hierarchy protocol,
*	and save them to a pcap file.
*
*	To compile:
*	>gcc MultiMask_match_sampling.c -lpcap -lm -o MultiMask_match_sampling
*
*	To run:
*	>sudo ./MultiMask_match_sampling
*
*	Enter the number of the device you want to sniff:
*	>2
*
*
************************************************************/

#include<pcap.h>
#include<stdio.h>
```

```
#include<stdlib.h> // for exit()
#include<string.h> //for memset
#include<sys/socket.h>
#include<arpa/inet.h> // for inet_ntoa()
#include<net/ethernet.h>
#include<netinet/ip.h>//Provides declarations for ip header
#include<math.h>    //Provides the pow() function
#include<unistd.h>


void process_packet(u_char *, const struct pcap_pkthdr *, const u_char *);
int multmask_matching(const u_char * Buffer);
void tobin(u_int16_t a,char* str);
int Sampling_ratio_decomposition(float ratio);
int tcp=0,udp=0,icmp=0,others=0,igmp=0,total=0;
pcap_t *handle;


// set the sampling ratio
float ratio;
// set the sampling parameter
int parameter = 0;
// set mask[]
int mask[16] = {0};
// set the number of mask
int number = 0;


int main()
{
	pcap_if_t *alldevsp , *device;
	pcap_t *handle; //Handle of the device that shall be sniffed
	pcap_dumper_t* out_pcap;

	char errbuf[100] , *devname , devs[100][100];
	int count = 1 , n;

	//First get the list of available devices
	printf("Finding available devices ... ");
	if( pcap_findalldevs( &alldevsp , errbuf) )
	{
		printf("Error finding devices : %s" , errbuf);
		exit(1);
	}
```

```
    printf("Done");

    //Print the available devices
    printf("\nAvailable Devices are :\n");
    for(device = alldevsp ; device != NULL ; device = device->next)
    {
        printf("%d. %s - %s\n" , count , device->name , device->description);
        if(device->name != NULL)
        {
            strcpy(devs[count] , device->name);
        }
        count++;
    }

    //Ask user which device to sniff
    printf("Enter the number of the device you want to sniff: ");
    scanf("%d" , &n);
    devname = devs[n];

    //Ask user the sampling ratio
    printf("Enter the ratio of the sampling: ");
    scanf("%f" , &ratio);

    //get the parameter
    parameter = Sampling_ratio_decomposition(ratio);
    printf("\n");
    printf("parameter:%d\n",parameter);
    printf("number:%d\n",number);

    // print device
    printf("\nOpening device %s for sniffing ... " , devname);
    handle = pcap_open_live(devname , 65536 , 0 , -1 , errbuf);

    if (handle == NULL)
    {
        fprintf(stderr, "Couldn't open device %s : %s\n" , devname , errbuf);
        exit(1);
    }
    printf("Done\n");

    // open pcap file and wait the packet update
    out_pcap =
pcap_dump_open(handle,"/home/jiapengli/sampling/MultiMask_match_sampling.pcap");
```

```
	// Put the device in sniff loop
	pcap_loop(handle , 500 , process_packet , (u_char*)out_pcap);
	printf("Total : %d\n",total);

	// refresh buffer
	pcap_dump_flush(out_pcap);

	// close resource
	pcap_dump_close(out_pcap);
	return 0;
}

/*********************************************************************************************
*************/
void process_packet(u_char *args, const struct pcap_pkthdr *header, const u_char *buffer)
{
	if(multmask_matching(buffer)) //Sampling interval
	{
		// Export traffic data to pcap file
		pcap_dump(args, header, buffer);

		//Get the IP Header part of this packet , excluding the ethernet header
		struct iphdr *iph = (struct iphdr*)(buffer + sizeof(struct ethhdr));
		++total;
		switch (iph->protocol) //Check the Protocol and do accordingly...
		{
			case 1:  //ICMP Protocol
				++icmp;
				break;

			case 2:  //IGMP Protocol
				++igmp;
				break;

			case 6:  //TCP Protocol
				++tcp;
				break;

			case 17: //UDP Protocol
				++udp;
				break;

			default: //Some Other Protocol like ARP etc.
				++others;
```

```
                        break;
            }
            printf("TCP : %d    UDP : %d    ICMP : %d    IGMP : %d    Others : %d
Total : %d\n", tcp , udp , icmp , igmp , others , total);
        }
}

int multmask_matching(const u_char * Buffer)
{
        // Get the ID of IPheader
        unsigned short iphdrlen;
        u_int16_t ip_id;
        char str[16] = {0};
        struct iphdr *iph = (struct iphdr *)(Buffer    + sizeof(struct ethhdr) );
        int flag = 0;
        ip_id= ntohs(iph->id);

        // Convert IP_identification to binary
        tobin(ip_id,str);
        int len = strlen(str);

        // Get the first non-zero occurrence of the Identifies , NO.L
        int L = 17-len;

        // Multimask sampling algorithm
        for(int i =0;i<number-1;i++)
        {
            if(mask[i] == L)
            {
                flag = 1;
                printf("The L is:%d\n",L);
                return flag;
            }
        }
        return flag;
}

void tobin(u_int16_t a,char* str)
{
        // p points to the first address of a
        char *p=(char*)&a,c=0,f=0,pos=-1;
        for(int o=0;o<2;++o)
        {
            for(int i=0;i<8;++i)
```

```
            {
            c=p[1-o]&(1<<(7-i));
            if(!f&&!(f=c))continue;
            str[++pos]=c?'1':'0';
            }
        }
}

int Sampling_ratio_decomposition(float ratio)
{
      float a =ratio;
      int P = 0;
      printf("mask:");
      for(int i =1;i<17;i++)
      {
            if(a>1.0/(pow(2,i)))
            {
                  a = a -(1.0/pow(2,i));
                  P = P + pow(2,i);
                  //get the mask and its number
                  mask[number] = i;
                  printf("%d,",i);
                  number++;
            }
      }
      return P;
}
```

2、执行程序

实验一

运行过程：

```
jiapengli@xiaoyanhu-System-Product-Name:~/sampling$ gcc Periodic_sampling.c -lpcap -o Periodic_sampling
jiapengli@xiaoyanhu-System-Product-Name:~/sampling$ sudo ./Periodic_sampling
Finding available devices ... Done
Available Devices are :
1. enp114s0 - (null)
2. any - Pseudo-device that captures on all interfaces
3. lo - (null)
4. enp5s0 - (null)
5. wlo1 - (null)
6. bluetooth0 - Bluetooth adapter number 0
7. bluetooth-monitor - Bluetooth Linux Monitor
8. nflog - Linux netfilter log (NFLOG) interface
9. nfqueue - Linux netfilter queue (NFQUEUE) interface
10. dbus-system - D-Bus system bus
11. dbus-session - D-Bus session bus
Enter the number of the device you want to sniff: 1
Enter the interval of the sampling: 3
```

对到达的 99 个报文进行间隔为 3 的周期抽样出的报文数：

```
TCP : 19   UDP : 2   ICMP : 0   IGMP : 0   Others : 1   Total : 22
TCP : 20   UDP : 2   ICMP : 0   IGMP : 0   Others : 1   Total : 23
TCP : 21   UDP : 2   ICMP : 0   IGMP : 0   Others : 1   Total : 24
TCP : 22   UDP : 2   ICMP : 0   IGMP : 0   Others : 1   Total : 25
TCP : 23   UDP : 2   ICMP : 0   IGMP : 0   Others : 1   Total : 26
TCP : 24   UDP : 2   ICMP : 0   IGMP : 0   Others : 1   Total : 27
TCP : 25   UDP : 2   ICMP : 0   IGMP : 0   Others : 1   Total : 28
TCP : 25   UDP : 2   ICMP : 0   IGMP : 0   Others : 2   Total : 29
TCP : 26   UDP : 2   ICMP : 0   IGMP : 0   Others : 2   Total : 30
TCP : 27   UDP : 2   ICMP : 0   IGMP : 0   Others : 2   Total : 31
TCP : 28   UDP : 2   ICMP : 0   IGMP : 0   Others : 2   Total : 32
TCP : 29   UDP : 2   ICMP : 0   IGMP : 0   Others : 2   Total : 33
jiapengli@xiaoyanhu-System-Product-Name:~/sampling$
```

实验二

运行过程:

```
jiapengli@xiaoyanhu-System-Product-Name:~/sampling$ gcc Random_sampling.c -lpcap -o Random_sampling
jiapengli@xiaoyanhu-System-Product-Name:~/sampling$ sudo ./Random_sampling
Finding available devices ... Done
Available Devices are :
1. enp114s0 - (null)
2. any - Pseudo-device that captures on all interfaces
3. lo - (null)
4. enp5s0 - (null)
5. wlo1 - (null)
6. bluetooth0 - Bluetooth adapter number 0
7. bluetooth-monitor - Bluetooth Linux Monitor
8. nflog - Linux netfilter log (NFLOG) interface
9. nfqueue - Linux netfilter queue (NFQUEUE) interface
10. dbus-system - D-Bus system bus
11. dbus-session - D-Bus session bus
Enter the number of the device you want to sniff: 1
Enter the threshold of the sampling ratio: 50
```

对到达的 500 个报文进行 50%随机抽样出的报文数:

```
time_now:2022-11-19 13:42:39.37612
time_now:2022-11-19 13:42:39.45611
TCP : 239   UDP : 0   ICMP : 0   IGMP : 0   Others : 0   Total : 239
time_now:2022-11-19 13:42:39.53614
time_now:2022-11-19 13:42:39.53617
time_now:2022-11-19 13:42:39.61615
time_now:2022-11-19 13:42:39.69615
TCP : 240   UDP : 0   ICMP : 0   IGMP : 0   Others : 0   Total : 240
time_now:2022-11-19 13:42:39.69618
time_now:2022-11-19 13:42:39.77608
TCP : 241   UDP : 0   ICMP : 0   IGMP : 0   Others : 0   Total : 241
time_now:2022-11-19 13:42:39.85616
jiapengli@xiaoyanhu-System-Product-Name:~/sampling$
```

实验三

运行过程:

```
jiapengli@xiaoyanhu-System-Product-Name:~/sampling$ gcc Mask_match_sampling.c -lpcap -o Mask_match_sampling
jiapengli@xiaoyanhu-System-Product-Name:~/sampling$ sudo ./Mask_match_sampling
[sudo] password for jiapengli:
Finding available devices ... Done
Available Devices are :
1. enp114s0 - (null)
2. any - Pseudo-device that captures on all interfaces
3. lo - (null)
4. enp5s0 - (null)
5. wlo1 - (null)
6. bluetooth0 - Bluetooth adapter number 0
7. bluetooth-monitor - Bluetooth Linux Monitor
8. nflog - Linux netfilter log (NFLOG) interface
9. nfqueue - Linux netfilter queue (NFQUEUE) interface
10. dbus-system - D-Bus system bus
11. dbus-session - D-Bus session bus
Enter the number of the device you want to sniff: 1
Enter the number of the bits used for mask sampling: 3
```

对到达的 500 个报文进行掩码位数为 3（比率为 $1/2^3$）的掩码匹配抽样出的报文数:

```
100011001001111
TCP : 45   UDP : 6   ICMP : 0   IGMP : 0   Others : 1   Total : 52
   |-Identification    : 50695
1100011000000111
TCP : 46   UDP : 6   ICMP : 0   IGMP : 0   Others : 1   Total : 53
   |-Identification    : 18007
100011001010111
TCP : 47   UDP : 6   ICMP : 0   IGMP : 0   Others : 1   Total : 54
   |-Identification    : 49215
1100000000111111
TCP : 47   UDP : 7   ICMP : 0   IGMP : 0   Others : 1   Total : 55
   |-Identification    : 50703
1100011000001111
TCP : 48   UDP : 7   ICMP : 0   IGMP : 0   Others : 1   Total : 56
   |-Identification    : 18015
100011001011111
TCP : 49   UDP : 7   ICMP : 0   IGMP : 0   Others : 1   Total : 57
   |-Identification    : 50711
1100011000010111
TCP : 50   UDP : 7   ICMP : 0   IGMP : 0   Others : 1   Total : 58
   |-Identification    : 18023
100011001100111
TCP : 51   UDP : 7   ICMP : 0   IGMP : 0   Others : 1   Total : 59
jiapengli@xiaoyanhu-System-Product-Name:~/sampling$
```

实验四

运行过程:

```
jiapengli@xiaoyanhu-System-Product-Name:~/sampling$ gcc sampling0.c -lpcap -lm -o sampling0
jiapengli@xiaoyanhu-System-Product-Name:~/sampling$ sudo ./sampling0
Finding available devices ... Done
Available Devices are :
1. enp114s0 - (null)
2. any - Pseudo-device that captures on all interfaces
3. lo - (null)
4. enp5s0 - (null)
5. wlo1 - (null)
6. bluetooth0 - Bluetooth adapter number 0
7. bluetooth-monitor - Bluetooth Linux Monitor
8. nflog - Linux netfilter log (NFLOG) interface
9. nfqueue - Linux netfilter queue (NFQUEUE) interface
10. dbus-system - D-Bus system bus
11. dbus-session - D-Bus session bus
Enter the number of the device you want to sniff: 2
Enter the ratio of the sampling: 0.638
mask:1,3,7,8,10,12,15,16,
parameter:103818
number:8
```

对到达的 500 个报文进行 0.638 概率多掩码抽样出的报文数:

```
The L is:10
TCP : 0   UDP : 0   ICMP : 0   IGMP : 0   Others : 300   Total : 300
The L is:10
TCP : 0   UDP : 0   ICMP : 0   IGMP : 0   Others : 301   Total : 301
The L is:10
TCP : 0   UDP : 0   ICMP : 0   IGMP : 0   Others : 302   Total : 302
The L is:10
TCP : 0   UDP : 0   ICMP : 0   IGMP : 0   Others : 303   Total : 303
The L is:10
TCP : 0   UDP : 0   ICMP : 0   IGMP : 0   Others : 304   Total : 304
The L is:10
TCP : 0   UDP : 0   ICMP : 0   IGMP : 0   Others : 305   Total : 305
Total : 305
jiapengli@xiaoyanhu-System-Product-Name:~/sampling$
```