Npcap抓包、解包编程

2111252 李佳豪

Npcap抓包、解包编程
- 学长检查时的问题1：
- 问题2：
- 实验目的
- 实验过程
  - 实验准备：Npcap 用到的函数
  - 代码逻辑解析：
- 完整代码链接

学长检查时的问题1：

- **UDP和TCP为什么要分开设计结构体，他们是不一样吗，哪里不一样（下面回答是自己的理解，可能有不准确的地方 ...）**

  当时我回答说是他们协议内容不同，但具体怎么个不同、为什么不同并不知。结合这次计算机网络关于udp协议上的可靠传输，以及自行搜索的ip协议上实现数据可靠传输的tcp协议、三次握手方面的知识，关于这个问题我想先得从应用层协议封装过程说起；

  - 从应用层根据其需要封装应用层http、ftp等协议之一，为区别同一主机不同应用，需要给予他们不同端口号来区别，我们使用了udp、tcp协议进行封装这个port，tcp、udp最根本的作用应该就是记录port区别了不同应用程序；而要实现主机到主机的传输，则需要通过电脑的ip地址，进行逻辑层面网络的通信，而ip协议最核心的功能便是实现了主机到主机的记录。

  - 当ip协议将上层传来的数据封装成packet后，便可以针对原主机-路由器、路由器-路由器、路由器-目的主机之间的路径进行传输（ip层具体如何配合物理链路层实现数据在交换机之间传递还不懂...），进一步在物理链路层封装mac地址进行物理网络传输。

  - 上述过程的基础上，信息在底层传递时出现差错、丢失现象，因此保证传输的可靠性，设计tcp协议，而对于视频流这种无需保证每一个packet准确的信息则使用udp协议，无需保证可靠性以达到更高速。

  - 因此tcp和udp传输的结构也不同，tcp有很多字段用来保证他的安全和可靠，udp则仅仅记录俩port、长度、一个简单的校验和和数据。

```
TCP:
+--------------------+--------------------+--------------------+--------------
--------+
|   Source Port   |   Destination Port   |      Seq Number    |    Ack Number
   |
+--------------------+--------------------+--------------------+--------------
--------+
| Hdr Len |  Res  |Flags|         Window Size         |   Checksum     |  Urg
Pointer  |
+--------------------+--------------------+--------------------+--------------
--------+
|             Options (if Hdr Len > 5) & Padding                          |
+--------------------+--------------------+--------------------+--------------
--------+
|                  Data (Payload) - Variable Length                      |
+--------------------+--------------------+--------------------+--------------
--------+
```

```
UDP:
+--------------------+--------------------+--------------------+
|   Source Port   |   Destination Port   |           Length      |
+--------------------+--------------------+--------------------+
|                    Checksum                  |                 |
+--------------------+--------------------+--------------------+
|                  Data (Payload) - Variable Length              |
+--------------------+--------------------+--------------------+
```

- 当应用层决定采取tcp还是udp传输这个数据时，会按照tcp和udp协议规定的样子包装这则信息，而为了自下而上解析数据包时，ip层也必须事先知道这个传输层是按照tcp还是udp、或者别的协议的内容来进行解析；因此ip的protocol 8位便起到这个作用。
- 也因此我们先拆了物理链路层的帧结构后，再根据ip结构拆解ip包，再根据ip层的protocol字段内容判断到底传输层是tcp还是udp协议，再使用他们对应的结构拆解的原因。

问题2：

- **浅拷贝深拷贝问题**
    - 从过程上看：网卡数据先拷贝进入内核区域一片`buffer`内，这片数据本应该经过正常网络协议栈的处理，逐层解析后发送到对应程序`Socket`的缓存区，但Libcab可以直接从这片`buffer`复制数据到用户空间，这种处理应该是必要的，因为我们用户空间若能直接处理这些包的内容，也就改变了协议栈后续的处理，就出问题了...
    - 因此`pcap_next_ex()`返回的是指向用户空间buffer的指针，这片`buffer`正是Libcab从内核空间复制到用户空间的数据暂存区，这是深拷贝，后面`pcap_next_ex()`访问这片buffer，只是通过指针，算是浅拷贝。

      另外https://www.jianshu.com/p/ed6db49a3428博客中提到`libpcap`绕过了`Linux`内核收包流程中协议栈部分的处理，使得用户空间`API`可以直接调用套接字`PF_PACKET`从链路层驱动程序中获得数据报文的拷贝，将其从内核缓冲区拷贝至用户空间缓冲区

实验目的

获得本机网卡设备，抓包解析。

实验过程

实验准备：Npcap 用到的函数

- 所需对象：完成捕获流量首先捕获本地设备的网络物理、虚拟接口
  - `pcap_if_t` ->一个捕获设备

```
struct pcap_if_t {
    struct pcap_if_t *next;       /* 指向下一个接口的指针 */
    char *name;                   /* 设备名称，例如 "eth0" */
    char *description;           /* 设备描述 */
    struct pcap_addr_t *addresses; /* 设备地址 */
    bpf_u_int32 flags;           /* PCAP_IF_标志 */
};
```

  - `pcap_addr_t` -> 表示与捕获设备关联的网络地址

```
struct pcap_addr_t {
    struct pcap_addr_t *next;    /* 一个接口多个地址指针 */
    struct sockaddr *addr;       /* 地址 */
    struct sockaddr *netmask;   /* 子网掩码 */
    struct sockaddr *broadaddr; /* 广播地址 */
    struct sockaddr *dstaddr;    /* P2P目的地址 */
};
```

  - `pcap_findalldevs_ex( )` 函数获得接口列表

```
int pcap_findalldevs_ex(const char *source,          /* 指定从哪个地方获得接口*/
                    struct pcap_rmtauth *auth,    /* NULL*/
                    pcap_if_t **alldevs,          /* 将第一个pcap_if_t的指针写到alldevs内*/
                    char *errbuf                  /* 错误缓存区 */
                    );
```

- 选择设备后，打开捕获句柄
  - `pcap_t` 表示用于捕获的会话句柄

```
typedef struct pcap pcap_t
```

  - `pcap_open()` 函数打开捕获设备

```
pcap_t *pcap_open(const char device,  /* 要捕获数据的网络设备的名称 */
                  snaplen:         /* 每个数据包捕获的最大字节数 */
                  promisc,         /* 是否启用混杂模式（0表示关闭，非0表示开启）
*/
                  to_ms,           /* 读取数据包的超时，单位为毫秒 */
                  errbuf) /* 错误缓冲区，用于存储错误或警告信息）*/
```

- 开始捕获流量
  - `pcap_next_ex()` 获得下一流量

```
int pcap_next_ex(pcap_t *p,     /* pcap_open返回的handle */
                 struct pcap_pkthdr **pkt_header,
                 const u_char **pkt_data);
```

  - `pcap_pkthdr`：

```
struct pcap_pkthdr {
    struct timeval ts;   /* 时间戳：该数据包被捕获时的时间  timeval{tv_sec 时
间戳和tv_usec 微妙}*/
    bpf_u_int32 caplen;  /* 数据包捕获的长度：实际捕获到的数据长度（可能小于数据
包的实际长度，例如因为混杂模式的设置） */
    bpf_u_int32 len;     /* 数据包的总长度：原始数据包的长度 */
};
```

代码逻辑解析：

- 协议对应结构体

```
struct eth_header; //物理链路层的14字节
struct ip_header; //ip头结构
struct udp_header; //udp头结构
```

- 封装函数

  具体流程，打开本机的所有网卡，输出信息，找到我想捕捉的网卡流量，opendevice打开捕获事件，使用`pcap_next_ex`捕获到数据存入一个buffer，自底向上逐层解析，得到包类型、长度等信息；根据解析返回的pcap_pkthdr值，得到时间、总长度信息。

```
int get_device_list(pcap_if_t** alldevs, char* errbuf); //返回列表位置给alldevs
成功返回列表数目 否则-1
int open_device(pcap_t** adhandle, int& num, pcap_if_t* alldevs, char*
errbuf); //返回设备handle给adhandle,num是要打开设备列表的序号
int capture(pcap_t* adhandle); //对adhandle句柄捕获流量，进行处理
```

完整代码链接

https://github.com/FondH/NetTech/blob/main/wireshark/Fwireshark/Fwireshark/main.cpp