Định nghĩa cấu trúc header:

Định nghĩa cấu trúc gói tin giúp truy xuất cái fields trở lên thuận tiện

Ethernet header: cố định đinh là 14 bytes

```
// ĐỊNH NGHĨA CÁC HEADER
struct ether_header {
    u_char  ether_dhost[6];  // Địa chỉ MAC đích
    u_char  ether_shost[6];  // Địa chỉ MAC nguồn
    u_short ether_type;       // Loại giao thức (IPv
};
```

IPv4 header:

```
struct ip {
    u_char  ip_hl:4;       // Chiều dài tiêu đề IP (
    u_char  ip_v:4;        // Phiên bản IP (IPv4 = 4
    u_char  ip_tos;        // Loại dịch vụ
    u_short ip_len;        // Tổng chiều dài của gói
    u_short ip_id;         // ID của gói tin
    u_short ip_off;        // Độ lệch (fragment offs
    u_char  ip_ttl;        // Thời gian sống (TTL)
    u_char  ip_p;          // Giao thức (TCP, UDP, I
    u_short ip_sum;        // Tổng kiểm tra (checksu
    struct  in_addr ip_src; // Địa chỉ IP nguồn
    struct  in_addr ip_dst; // Địa chỉ IP đích
};
```

IPv6 header:

```
struct ip6_hdr {
    uint32_t ip6_flow;      // Dữ liệu dòng (flow label)
    uint16_t ip6_plen;      // Độ dài phần payload
    uint8_t  ip6_nxt;       // Giá trị của Next Header
    uint8_t  ip6_hlim;      // Thời gian sống (Hop Limit)
    struct in6_addr ip6_src; // Địa chỉ nguồn
    struct in6_addr ip6_dst; // Địa chỉ đích
};
```

TCP header:

```c
struct tcp_header {
    uint16_t source_port;        // Cổng nguồn
    uint16_t destination_port;   // Cổng đích
    uint32_t sequence_number;    // Số thứ tự
    uint32_t acknowledgment_number; // Số xác nhận
    uint8_t data_offset: 4;      // Độ dài header
    uint8_t reserved: 3;         // Dành cho tương lai
    uint8_t flags: 9;            // Các cờ điều khiển
    uint16_t window_size;        // Kích thước cửa sổ
    uint16_t checksum;           // Kiểm tra lỗi
    uint16_t urgent_pointer;     // Con trỏ khẩn cấp
};
```

UDP header:

```c
struct udp_header {
    uint16_t source_port;      // Port nguồn
    uint16_t destination_port; // Port đích
    uint16_t length;           // Chiều dài toàn bộ gói UDP
    uint16_t checksum;         // Tổng kiểm tra
};
```

DNS header:

```c
struct dns_header {
    uint16_t id;          // Mã định danh của gói DNS
    uint16_t flags;       // Cờ và mã hoạt động của DNS
    uint16_t q_count;     // Số lượng câu hỏi trong phần truy vấn
    uint16_t ans_count;   // Số lượng câu trả lời trong phần phản hồi
    uint16_t auth_count; // Số lượng bản ghi quyền lực
    uint16_t add_count;   // Số lượng bản ghi bổ sung
};
```

Thực hiện phân tích gói tin:

B1: Ta sử dụng các hàm trong thư viện trong pcap để mở file pcap và duyệt từng packet.

```
445     pcap_t* handle = pcap_open_offline(filename, errbuf);
446     if (handle == NULL) {
447         cout << " cannot open pcap file" << errbuf << endl;
448         return 1;
449     }
```

Tại dòng 445: ta thực hiện hàm pcap_open_offline() để thực hiện mở .pcap

```
452     while ((packet = pcap_next(handle, &header))!= nullptr){
453
454         cout<< endl  <<"------------------PACKET:"<<  dec <<index <<"--
455         outPut << endl  <<"------------------PACKET:"<<  dec <<index <<
456         processPacket(packet,count,outPut);
457         cout << endl;
458         outPut << endl;
459         index++;
460     }
461     pcap_close(handle);
```

Dòng 452: thực hiện vòng while để duyệt gói tin đọc từ file .pcap

Dòng 456: thực thực hiện hàm **processPacke(packet, count, outPut)** xử đếm gói tin và trích xuất thông quan trong.

- packet: là dạng con trỏ u_char* trỏ tới gói tin cần xử lý;
- count: là một vector<int> đếm số lượng của gói tin cho từng giao thức với index từ 0 đến 9 tương ứng index trong từng giao thức: "ETHER", "TCP", "UDP", "ICMP", "DNS", "HTTP","IP","HTTPS", "ICMPv6" ,"OTHER"
- outPut: in dữ liệu xử ra file .txt

B2: Đếm Gói Tin Cho Từng Giao Thức và Trích Xuất Thông Tin Quan Trọng:

- Đếm packet của Ethernet: với mỗi packet được duyệt trong hay packet khác giá trị NULL, ta cộng giá trị count[0] lên một đơn vị (tương ứng countEthernet + 1).

- Nhưng trong phân xử lý đảm bảo an toàn, xác định cấu cấu trúc của ethernet header trước khi thực hiện đếm số lượng gói tin cho mỗi giao thức:

```
struct ether_header* ether = (struct ether_header*) packet;
```

Đoán code với mục đích ép kiểu con trỏ đến gói tin packet thành một con trỏ của ethernet header (14 bytes): điều giúp ta dễ dàng truy xuất các trường của ethernet header

Tứ đó dễ trích xuất thông quan trọng như source MAC và Destination MAC theo cấu trúc của ethernet header.

Ngoài ra, tại ethernet header có trường ether_type (2bytes)

với các giá trị phổ biến như: 0x0800 (tương ứng IPv4) , 0x086D (tương ứng với iPv6 và các giá trị khác của giao thức khác:

Như thế 3 xác định trường hợp: IPv4, IPv6 và Others:

- Tại cả 3 trường hợp: countEnther + 1.
- Trong mỗi trường hợp IPv4 và IPv6 : countIP + 1.
- Tại trường hợp others: cộng countOther + 1.
-

Trường hợp IPv4:

ta thực xác định gói tin thuộc giao thức UPD, TCP, ICMP và trích xuất thông địa chỉ IP nguồn và đích.

Vị trí bắt đầu của ip header nằm trí byte thứ 14 của packet, (do 14 bytes đầu tiên từ byte thứ 0 đến byte thứ 13 là của phần ethernet header.

```
struct ip* ip_header = (struct ip*) (packet + 14);
```

Đoạn code trên để thực hiện ép phần dữ liệu tại vị trí **packet + 14** (di chuyển con trỏ; trỏ đến byte thứ 14 đây vị trí bắt đầu IP header) thành một con trỏ kiểu struct ip, để có thể truy cập trực tiếp vào các trường trong header IP.

=> **Trích xuất địa chỉ ip nguồn và đích theo các field của IP header**

Tại trường ip_p hay ip port: với các giá trị phổ biến như: 1 (ICMP), 6 (TCP) , 17 (UDP), Other (giao thức khác):

=> **Ta tăng biến đếm count của từng giao thức nếu gói tin rơi vào trường hợp cụ thể: ICMP, TCP, UDP hoặc other giao thức khác**.

Tiếp theo truy xuất đến header của TCP và UDP: nằm sau sau header của **ethernet header (14 bytes) và IP header (ip_hl * 4 byte)** tại trường IP header không có byte xác định nó nằm trong khoảng 20 đến 60 bytes, trường ip_hl cho ta biết kích thước ip header, giá trị của trường được lưu dưới dạng **số lượng "words" 32-bit** (4 bytes) của header IP.

=> vị trí bắt đầu UDP và TCP header = packet + 14 + ip_hl * 4;

tcp header:

```
leng_iph = ip_header->ip_hl * 4; // độ dài ip header
struct tcp_header* tcp_h = (struct tcp_header*) (packet + (14 +leng_iph));
```

Udp header:

```
leng_iph = ip_header->ip_hl * 4; // độ dài ip header
struct udp_header* udp_hdr = (struct udp_header*) packet + (14 + leng_iph );
```

Tương tự ép kiểu trong di chuyển trỏ ở phần trên, và việc định nghĩa các header tại phần đầu:

=> **truy xuất source port và dest port**

- **Trường hợp TCP:**

  Nếu source port hoặc destination là các giá trị 80 (HTTP), 443 (HTTPS) và 53 (DNS)

  => **ta có tăng biến count khi rơi từng trường hợp.**

- **Trường hợp UDP:**

  Nếu source port hoặc destination là 53 (DNS) => **đếm gói tin thuộc giao thức DNS.**

Trích xuất phương thức (GET, POST, v.v.), URL và mã trạng thái HTTP

Để trích xuất method, URL và status ta cần xuất tới phần payload của packet và nằm tại vị sau TCP header cụ thể là vị trí byte thứ:

14 + length ip header (độ dài ip header) + độ dài của TCP header.

Tượng tự IP header, độ dài TCP header được xác tại trường data offset và cũng được biểu diễn bằng 32-bit words (4 bytes) => **len = data_offset * 4**

```
leng_iph = ip_header->ip_hl * 4; // độ dài ip header
struct tcp_header* tcp_h = (struct tcp_header*) (packet + (14 +leng_iph));
int len_tcp_hdr = tcp_h->data_offset * 4; // độ dài cải tcp_header;
```

Chuyển con trỏ payload trong packet:

```
count[5]++;
const u_char * payload = packet + (14 + leng_iph + len_tcp_hdr);
```

Tính độ dài payload = tổng độ dài gói tin ip (Trong trường ip_len của IP HEADER) - (độ dài ip header + đồ dài TCP header);

```
int total_length = ntohs(ip_header->ip_len);
// Độ dài TCP payload
int tcp_payload_length = total_length - (leng_iph + len_tcp_hdr);
```

Get các thông tin của HTTP:

```
void getHttpInfo(const u_char* payload, int length,  ofstream& outPut) {···
```

Trong hàm này truyền vào các tham payload, length (độ dài payload), outPut (in file txt)

Hàm sử dụng function trong thư viện string để find method, host, URL xác định vị trí, nếu có thực hiện cắt chuỗi để lấy thông tin.

```
void getHttpInfo(const u_char* payload, int length,  ofstream& outPut) {
    // Chuyển đổi payload thành chuỗi
    string httpPayload(reinterpret_cast<const char*>(payload), length);
    string filteredPayload;

    // Lọc ra các ký tự có
    for (char c : httpPayload) {···

    istringstream stream(filteredPayload); // chuyển chuyển dữ liệu thành dàng luồng hay từng dòng
    string line;
    string method, url1, url2, status;

    // Đọc từng dòng trong chuỗi đã lọc
    while ( getline(stream, line)) {
        // Kiểm tra phương thức HTTP
        if (indexMethod(line) != -1) // kiểm tra method
        {
            int position = indexMethod(line); // trả về vị trí method.
            if (position !=  string::npos) {
                method = line.substr(position, line.find(" ") - position); // Trích xuất phương thức
                int nextSpace = line.find(" ", position + 1); // Tìm dấu cách thứ hai
                if (nextSpace !=  string::npos) {
                    int nnSpace = line.find(" ", nextSpace + 1);
                    if (nnSpace !=  string::npos) {
                        url1 = line.substr(nextSpace + 1, nnSpace - nextSpace - 1); // Trích xuất URL
                    }
                }
            }
        }
        // Trích xuất thông tin Host
        else if (line.find("Host:") !=  string::npos) {
            int hostPosition = line.find(":"); // Tìm dấu ":"
            if (hostPosition !=  string::npos) {
                url2 = line.substr(hostPosition + 1); // Trích xuất Host
                // Xóa khoảng trắng ở đầu URL
                url2.erase(0, url2.find_first_not_of(" \n\r\t"));
            }
        }
        // Kiểm tra mã trạng thái
        else if (line.find("HTTP/") !=  string::npos) {
            int statusPosition = line.find("HTTP/");
            if (statusPosition !=  string::npos) {
                status = line.substr(statusPosition); // Trích xuất mã trạng thái
            }
        }
    }
    // Tạo chuỗi đầu ra
    stringstream write;
    write<<"HTTP:\n";
    if (!method.empty()) {···

    if (!url1.empty() || !url2.empty()) {···
    if (!status.empty()) {···

    // In và ghi ra file
    cout << write.str();
    outPut << write.str();
}
```

DNS trên UDP và TCP

Để truy vấn tên miền ta phải truy vấn tới phần Question Section và địa chỉ IP trả về tà cần truy xuất tới Answer Sections

Vị trí phần Question Section: năm sau DNS header (có độ dài cố định là 12 bytes)

Vị trí Question Section của nó trong packet là:

- Với TCP: **offset** = 14 + độ dài ip header (ip_hl * 4) + độ dài TCP header (offset * 4) + 12 (độ dài của DNS header)
- Vói UDP: **offset** = 14 + độ dài ip header (ip_hl * 4) + 8 (độ dài UDP header) + 12 (độ dài của DNS header)
-
- Cấu trúc của Question Section gồm: **qName -> qType (2bytes) -> qClass (2bytes);**

qName chính là phần domain nó không xác định số byte nhất định và tùy thuộc độ dài tên của domain và tổ chức:

ví dụ: www.example.com

Phân tích từng phần:

- 3 |"www"
- 7 |cho "example"
- 3 |"com"
- 0 |(đánh dấu kết thúc)

Từ đó sử dụng hàm này để get domain.

```
220  string getDomain( const u_char * packet,int &offset){
221      string domain;
222      int len = packet[offset];
223      while( len > 0){
224          const u_char * domainPart = &packet[offset+1];
225          domain.append(reinterpret_cast<const char*>(domainPart), len);
226          offset = offset + len + 1;
227          len = packet[offset];
228          if(len>0) domain += 1;
229      }
230      offset++; // bo qua ket thu ten mien
231      return domain;
232  }
233
```

Giải thích:

- giá trị **offset** (vị trí byte đầu tiên của Question Section)
- Dòng 222: lấy giá trí độ dài như ví trên **len** = 3;
- dòng 224 và 225: thêm (nối chuỗi) chuối con có dài từ domainPart vào string domain (khai bao dòng 221)

- Dòng 226: cập nhật lại offset và chỉ tới vị trí chứa **len** mới là 7 độ đại của "example"
- Tiếp cho đến khi offset là vị trí 0 đánh kết thúc **qName**

Địa chỉ IP trả về tà cần truy xuất tới Answer Section:
Phần Answer section nằm sau Question Section:
Name(? byte) -> Type (2->bytes) -> class(2 bytes) -> TTL(4 bytes) -> LENGTH (độ dài dữ liệu trả về) 2 byte -> **RDATA**
Như thế để trích xuất ip cần truy cập trơi RDATA như đoạn code dưới để cập nhật **offset** tới vị trí byte của RDATA trong packet.

```
//AwnQus(name ?byte, type 2 bytes, class 2 bytes, TTL: 4 bytes, date len: 2 bytes, data 4 bytes )
int offset = 14 + leng_iph + 8 + 12; // 12bytes la do dai DNS headers
if(ntohs(dns_hdr->q_count) > 0)
{   for(int i = 0; i < ntohs(dns_hdr->q_count); i ++){
        string domain = getDomain(packet, offset);
        outPut<< "Domain: " << domain<< endl;
        cout<< "Domain: " << domain<< endl;
        offset += 4; // bỏ qua 4bytes ( 2byte type và 2 bytes của class)
    }
}
// get ip from DNS AWS Section;
if(ntohs(dns_hdr->ans_count) > 0)
{
    for( int i = 0; i < dns_hdr->ans_count ; i++){
        string anwser_domain_name = getDomain(packet, offset); // bỏ qua số byte tại Name files;
        offset += 10; // Bỏ qua Type (2 byte), Class (2 byte), TTL (4 byte), Data length (2 byte)
        struct in_addr addr;
        memcpy(&addr, &packet[offset], sizeof(struct in_addr));
        outPut << "DNS respond IP:" << ipToString(addr) << endl;
        cout << "DNS respond IP:" << ipToString(addr) << endl;
        offset+= 4; // bỏ qua 4 byte phần data.
    }

}
```

Trường hợp IPv6:

Trượng tự như IPv4:
Vị trí IPv6 header nằm vị byte thứ 14 sau ethernet header
=> dựa vào cấu trúc ip header để lấy địa chỉ nguồn và đích.
=> dựa vào trường **ip6_next** để xác header tiếp theo: 6 (TCP) , 17 (UDP), 58 (ICMPv6) và các giao thức khác (OTHER)
=> vị trí byte của TCP và UDP header làtrong parke 14 + 40 (IPv6 header có độ dài cố định là 40)

- Sau đó ta thực hiện in port theo các trường đã được định nghĩa và đếm các giao thức như giao thức IPv4:

NOTE: Tại phần HTTPS em chưa xử lý được trích xuất được thông tin về domain được truy vấn do đó em chưa giải thích tại về kỹ thuật Tại IPv6 em chưa xử lý lấy thông tin http, https và DNS.