

# Programming Assignment: Programming using Libpcap

DongHyeonRyu (SS Lab, 2023246)

---

## Contents

[\[Requirements to compile\]](#)

[\[How to Compile\]](#)

### [1. Capturing packets using libpcap](#)

(1) [src/assignment1.cpp](#)

(2) [pcap\\_handler.h](#)

(3) [logger.h](#)

(4) Results

### [2. Packet analysis using libpcap \(Part 1\)](#)

(1) [src/assignment2.cpp](#)

(2) [analyzer.h](#)

(3) Results

### [3. Packet analysis using libpcap \(Part 2\)](#)

(1) [packetlib/packet\\_manager::make\\_packet](#)

(2) Results

---

## [Requirements to compile]

1. **cmake version**  $\geq$  3.16
2. **c++20** version required
3. **pcap** library required

```
> sudo apt install libpcap-dev
```

---

## [How to Compile]

```
> mkdir build
> cd build
> cmake ../
> cmake --build . # two executable files are created. (assignment1 & assignment2)
```

---

# 1. Capturing packets using libpcap

## (1) src/assignment1.cpp

```
5  #include <csignal>
6  #include <csetjmp>
7  #include <sstream>
8  #include "utils/pcap_handler.h"
9  #include "utils/logger.h"
10 #include "analyzer.h"
11 #include "packet_manager.h"
12
13 sig_atomic_t stopFlag = 0;
14 jmp_buf jmpbuf;
15
16 void handler(int)
17 {
18     stopFlag = 1;
19     longjmp(env: jmpbuf, val: 1);
20 }
21
22 void callback(u_char *, const struct pcap_pkthdr *pkthdr, const u_char *_packet)
23 {
24     auto packet : shared_ptr<simple_packet> = libpacket::packet_manager::make_packet(pkthdr, packet: _packet);
25     utils::logger::info(packet);
26     libpacket::analyzer::Gatcha(packet);
27 }
28
29 int main(int argc, char **argv)
30 {
31     signal( sig: SIGINT, &handler );
32
33     /* custom pcap wrapper object */
34     auto *pcap = new utils::pcap_handler();
35
36     pcap->find_all_devs();
37     pcap->print_all_devs_info();
38
39     /* make filter from user input */
40     std::stringstream filter;
41     for(int i = 1; i<argc; i++)
42         filter << " " << argv[i];
43
44     pcap_t *pcd = pcap->set_filter( device_no: 0, filter: filter.str(), buffer_size: BUFSIZ, listening_duration_in_sec: -1);
45
46     setjmp(jmpbuf);
47     if (!stopFlag) pcap->gatcha(pcd, callback);
48
49     /* print summary */
50     libpacket::analyzer::GetInstance()->analyze();
51
52     /* save dump file */
53     pcap_dumper_t *pd = pcap_dump_open(pcd, "capture.pcap");
54     pcap_dump_close(pd);
55 }
```

- This can be executed by `./assignment1 {filter charaters}`
- The program captures packets infinitely until user give it a **SIGINT** signal. After got a **SIGINT** signal, the program prints summary and save dumpfiles.
- **Code description**
  - **main**
    - **(line 31)** : register **SIGINT** signal handler
    - **(line 34)** : create my custom pcap-wrapper-object for convenience.
    - **(line 36~37)** : find all network devices in local computer. and then print their information briefly.
    - **(line 40~44)** : set filter with user input.
    - **(line 46~47)** : once the program gets a **SIGINT** signal, **stopFlag** will be set to 1, meaning “do not capture packets anymore”.
    - **(line 50)** : print summary.
    - **(line 53~54)** : save dumpfile to .pcap file.
  - **callback**
    - **(line 24)** make a “**Packet**” object with given parameters from pcap library. “**Packet**” object has only necessary informations.
    - **(line 25)** using “**Packet**” object, print current packet information.
    - **(line 26)** pass “**Packet**” object to analyzer so that the analyzer make a final summary.

## (2) pcap\_handler.h

```

13
14 typedef void (*pcap_callback)(u_char *, const struct pcap_pkthdr *, const u_char *);
15
16 struct device_info {
17     pcap_if_t *device;
18     bpf_u_int32 *network;
19     bpf_u_int32 *mask;
20     std::string network_str;
21     std::string mask_str;
22 };
23
24 class pcap_handler {
25 public:
26     pcap_handler() = default;
27
28     void find_all_devs();
29     void print_all_devs_info();
30
31     pcap_t* set_filter(const std::string& device_name, const std::string& filter, int buffer_size, int listening_duration_in_sec, bool promiscuous_mode = false);
32     pcap_t* set_filter(int device_no, const std::string& filter, int buffer_size, int listening_duration_in_sec, bool promiscuous_mode = false);
33     void gatcha(pcap_t *pcd, pcap_callback callback);
34     void gatcha(const std::string& file_name, pcap_callback callback);
35
36 private:
37     std::vector<device_info> devices;
38     std::unordered_map<pcap_t*, int> device_no_of;
39     utils::device_info* _find_by_name(const std::string& device_name);
40
41     pcap_t* get_pcd(const std::string& device_name, int buffer_size, int listening_duration_in_sec, bool promiscuous_mode = false);
42     pcap_t* get_pcd(int device_no, int buffer_size, int listening_duration_in_sec, bool promiscuous_mode = false);
43     pcap_t* get_pcd_for_file(const std::string& file_name);
44 };
45
46

```

- `pcap_handler.h` aims to give high abstraction from **pcap**'s specification. For live capture, user can just call “**set\_filter**” then “**gatcha**” consecutively. Otherwise, using **.pcap** file, user can simply call another version of “**gatcha**” function.

### (3) `logger.h`

```

14  class logger {
15  public:
16      logger()= default;
17
18      static std::string currentDateTime();
19
20      template<typename T>
21      static void info(const T& object, const std::string& message) {
22          std::cout << "[info:]" << typeid(object).name() << " ] - [" << currentDateTime() << " ] - " << message << std::endl;
23      }
24
25      template<typename T>
26      static void warn(const T& object, const std::string &message) {
27          std::cout << "[warn:]" << typeid(object).name() << " ] - [" << currentDateTime() << " ] - " << message << std::endl;
28      }
29
30      template<typename T>
31      static void error(const T& object, const std::string &message) {
32          std::cout << "[error:]" << typeid(object).name() << " ] - [" << currentDateTime() << " ] - " << message << std::endl;
33          exit( status:1);
34      }
35
36      static void info(const libpacket::Packet& packet);
37  };

```

- `logger.h` helps to reduce redundant codes for printing messages.

### (4) Results

- The result file “**capture.pcap**” has been attacted with this report.

## 2. Packet analysis using libpcap (Part 1)

### (1) `src/assignment2.cpp`

```

13     sig_atomic_t stopFlag = 0;
14     jmp_buf jmpbuf;
15
16     void handler(int)
17     {
18         stopFlag = 1;
19         longjmp(env: jmpbuf, val: 1);
20     }
21
22     void callback(u_char *, const struct pcap_pkthdr *pkthdr, const u_char *_packet)
23     {
24         auto packet : shared_ptr<simple_packet> = libpacket::packet_manager::make_packet(pkthdr, packet: _packet);
25         utils::logger::info(packet);
26         libpacket::analyzer::Gatcha(packet);
27     }
28
29     int main(int argc, char **argv)
30     {
31         signal( sig: SIGINT, &handler );
32
33         auto *pcap = new utils::pcap_handler();
34
35         setjmp(jmpbuf);
36         if (!stopFlag) pcap->gatcha( file_name: argv[1], callback);
37
38         libpacket::analyzer::GetInstance()->analyze();
39     }

```

- This program is very similar to “**assignment1.cpp**” but much more simple.
- Note that the “**gatcha**” function is different from previous one. It’s implemented as a overloading function.
- User can run this program by `./assignment2 {filename.pcap}` , then this program read the .pcap file and show information for each packet. After reading all the packets, prints summary.

## (2) analyzer.h

```

14  ➡   class analyzer;
15      typedef analyzer* Analyzer;
16
17  ✖   class analyzer : public template_singleton<analyzer> {
18      public:
19          analyzer()=default;
20
21  ✖          ~analyzer() override;
22
23  ➡          void yummy(const Packet& packet);
24  ➡          void analyze();
25  ➡          static void Gatcha(const Packet& packet);
26
27      private:
28          std::map<protocol::protocol_type, Statistician> citizen_of_; // protocol
29          std::map<std::string, Statistician> mailman_of_; // dst
30          std::map<std::uint16_t, Statistician> developer_of_; // app (#port)
31
32          uint16_t n_packets = 0;
33          uint16_t n_drops = 0;
34          uint32_t n_bytes = 0;
35
36  ➡          void _time_to_eat(const Packet& packet);
37  ➡      };

```

analyzer.h

```

9  void libpacket::analyzer::yummy(const libpacket::Packet& packet) {
10
11      if (!packet->protocol->is_ethernet) {
12          n_drops += 1;
13          return;
14      }
15
16      if (citizen_of_[packet->protocol->type] == nullptr)
17          citizen_of_[packet->protocol->type] = std::make_shared<statistician>();
18      if (mailman_of_[packet->dst] == nullptr)
19          mailman_of_[packet->dst] = std::make_shared<statistician>();
20      if (developer_of_[packet->dst_port] == nullptr)
21          developer_of_[packet->dst_port] = std::make_shared<statistician>();
22
23
24      citizen_of_[packet->protocol->type]->eat(packet);
25      mailman_of_[packet->dst]->eat(packet);
26      developer_of_[packet->dst_port]->eat(packet);
27      _time_to_eat(packet);
28  }
29
30  void libpacket::analyzer::_time_to_eat(const libpacket::Packet &packet) {
31      n_packets += 1;
32      n_bytes += packet->bytes;
33  }
34
35  void libpacket::analyzer::Gatcha(const libpacket::Packet &packet) {
36
37      Analyzer analyst = analyzer::GetInstance();
38      analyst->yummy(packet);
39  }

```

analyzer.cpp

- Since `analyzer` class is a “**singleton**”, the very `analyzer` instance (named “analyst”) gets packets one by one via “**Gatcha**”(analyzer.cpp::line 35) function.
- `analyzer` instance looks into the packet, update figures (number of packets, bytes) according to protocol type, destination address and port number.
- After `analyzer` instance eats all the packets from input file, call “**analyze**” function to print summary. (just printing only.)

### (3) Results

- The output file “[**result**] Packet analysis using libpcap (Part 1).txt” has been attached with this report.
- Screenshots of this program’s output are followed:

```
22:59:08.567558 IP4 UDP 40.99.67.226.443 > 141.223.121.126.60151, length 70
22:59:08.567648 IP4 UDP 40.99.67.226.443 > 141.223.121.126.60151, length 72
22:59:08.570444 IP4 UDP 141.223.121.126.60151 > 40.99.67.226.443, length 84
22:59:08.591529 IP4 UDP 40.99.67.226.443 > 141.223.121.126.60151, length 1514
22:59:08.591529 IP4 UDP 40.99.67.226.443 > 141.223.121.126.60151, length 735
22:59:08.591728 IP4 UDP 141.223.121.126.60151 > 40.99.67.226.443, length 96
22:59:08.592028 IP4 UDP 40.99.67.226.443 > 141.223.121.126.60151, length 70
22:59:08.594064 IP4 UDP 141.223.121.126.60151 > 40.99.67.226.443, length 84
22:59:08.602461 IP4 UDP 40.99.67.226.443 > 141.223.121.126.60151, length 74
22:59:08.605112 IP4 UDP 141.223.121.126.60151 > 40.99.67.226.443, length 84
22:59:08.732895 IP4 UDP 0.0.0.0.68 > 255.255.255.255.67, length 324
22:59:08.778818 IP4 ICMP 141.223.121.126 > 172.217.25.174, length 98
22:59:08.801322 IP4 ICMP 172.217.25.174 > 141.223.121.126, length 98
22:59:08.836739 IP4 UDP 141.223.121.126.17500 > 141.223.124.255.17500, length 188
22:59:08.891568 IP4 TCP 141.223.121.126.63774 > 35.73.126.78.443, length 55
22:59:08.931831 IP4 TCP 35.73.126.78.443 > 141.223.121.126.63774, length 66
22:59:08.962873 Non-ethernet packet.
22:59:09.322153 IP4 UDP 141.223.124.8.50281 > 239.255.255.250.1900, length 206
22:59:09.373955 IP4 UDP 141.223.124.13.58536 > 239.255.255.250.1900, length 217
22:59:09.468905 IP4 UDP 141.223.124.8.54915 > 141.223.124.255.54915, length 305
22:59:09.513471 IP4 UDP 141.223.124.4.5353 > 224.0.0.251.5353, length 85
22:59:09.513675 Non-ethernet packet.
22:59:09.578959 IP4 TCP 211.115.106.79.80 > 141.223.121.126.63827, length 60
22:59:09.578959 IP4 TCP 141.223.121.126.63827 > 211.115.106.79.80, length 54
22:59:09.579281 IP4 TCP 211.115.106.79.80 > 141.223.121.126.63828, length 60
22:59:09.579297 IP4 TCP 141.223.121.126.63828 > 211.115.106.79.80, length 54
22:59:09.772593 IP4 ICMP 141.223.121.126 > 172.217.25.174, length 98
22:59:09.803015 IP4 ICMP 172.217.25.174 > 141.223.121.126, length 98
22:59:09.915992 IP4 TCP 211.115.106.79.80 > 141.223.121.126.63826, length 60
22:59:09.916020 IP4 TCP 141.223.121.126.63826 > 211.115.106.79.80, length 54
22:59:10.180261 IP4 UDP 0.0.0.0.68 > 255.255.255.255.67, length 324
22:59:10.469691 IP4 UDP 141.223.124.8.54915 > 141.223.124.255.54915, length 305
22:59:10.472424 Non-ethernet packet.
22:59:10.521811 IP4 UDP 141.223.124.4.5353 > 224.0.0.251.5353, length 85
22:59:10.521951 Non-ethernet packet.
22:59:10.774266 IP4 ICMP 141.223.121.126 > 172.217.25.174, length 98
22:59:10.781301 IP4 UDP 0.0.0.0.68 > 255.255.255.255.67, length 298
22:59:10.804676 IP4 ICMP 172.217.25.174 > 141.223.121.126, length 98
22:59:10.967132 Non-ethernet packet.
22:59:11.009412 IP4 Others 141.223.109.99.0 > 224.0.0.10.0, length 74
22:59:11.295115 IP4 TCP 141.223.121.126.65439 > 77.88.21.90.443, length 55
22:59:11.478731 IP4 UDP 141.223.124.8.54915 > 141.223.124.255.54915, length 305
22:59:11.573796 IP4 TCP 77.88.21.90.443 > 141.223.121.126.65439, length 66
22:59:11.741565 IP4 UDP 0.0.0.0.68 > 255.255.255.255.67, length 324
22:59:11.776005 IP4 ICMP 141.223.121.126 > 172.217.25.174, length 98
22:59:11.808553 IP4 ICMP 172.217.25.174 > 141.223.121.126, length 98
22:59:12.839956 IP4 UDP 192.168.0.239.5353 > 224.0.0.251.5353, length 128
22:59:12.237100 IP4 UDP 141.223.124.17.50763 > 239.255.255.250.1900, length 217
22:59:12.448194 IP4 UDP 141.223.124.8.54915 > 141.223.124.255.54915, length 305
22:59:12.482899 Non-ethernet packet.
22:59:12.777828 IP4 ICMP 141.223.121.126 > 172.217.25.174, length 98
22:59:12.808118 IP4 ICMP 172.217.25.174 > 141.223.121.126, length 98
22:59:12.822384 IP4 UDP 141.223.83.147.17500 > 255.255.255.17500, length 199
22:59:12.828641 IP4 UDP 141.223.83.147.17500 > 255.255.255.17500, length 199
22:59:12.828641 IP4 UDP 141.223.83.147.17500 > 141.223.83.255.17500, length 199
22:59:12.828683 IP4 UDP 141.223.83.147.17500 > 255.255.255.255.17500, length 199
22:59:12.828683 IP4 UDP 141.223.83.147.17500 > 255.255.255.255.17500, length 199
22:59:12.828803 IP4 UDP 141.223.83.147.17500 > 255.255.255.255.17500, length 308
22:59:12.948589 IP4 UDP 141.223.121.79.17500 > 255.255.255.255.17500, length 308
22:59:12.949506 IP4 UDP 141.223.121.79.17500 > 141.223.121.255.17500, length 308
22:59:12.949506 IP4 UDP 141.223.121.79.17500 > 255.255.255.255.17500, length 308
22:59:12.949545 IP4 UDP 141.223.121.79.17500 > 255.255.255.255.17500, length 308
22:59:12.949665 IP4 UDP 141.223.121.79.17500 > 255.255.255.255.17500, length 308
22:59:12.949665 IP4 UDP 141.223.121.79.17500 > 255.255.255.255.17500, length 308
22:59:12.982357 Non-ethernet packet.
22:59:13.172012 IP4 UDP 0.0.0.0.68 > 255.255.255.255.67, length 324
22:59:13.237737 IP4 UDP 141.223.124.17.50763 > 239.255.255.250.1900, length 217
22:59:13.449912 IP4 UDP 141.223.124.8.54915 > 141.223.124.255.54915, length 305
22:59:13.485590 IP4 UDP 0.0.0.0.68 > 255.255.255.255.67, length 342
22:59:13.778426 IP4 ICMP 141.223.121.126 > 172.217.25.174, length 98
22:59:13.809986 IP4 ICMP 172.217.25.174 > 141.223.121.126, length 98
22:59:14.237784 IP4 UDP 141.223.124.17.50763 > 239.255.255.250.1900, length 217
22:59:14.286424 IP4 TCP 141.223.121.126.64945 > 35.75.32.83.443, length 110
22:59:14.571942 IP4 TCP 35.75.32.83.443 > 141.223.121.126.64945, length 60
22:59:14.447432 IP4 UDP 141.223.124.8.54915 > 141.223.124.255.54915, length 305
22:59:14.485431 Non-ethernet packet.
22:59:14.603718 IP4 UDP 0.0.0.0.68 > 255.255.255.255.67, length 324
22:59:14.744302 IP4 UDP 0.0.0.0.68 > 255.255.255.255.67, length 324
22:59:14.781330 IP4 ICMP 141.223.121.126 > 172.217.25.174, length 98
22:59:14.811808 IP4 ICMP 172.217.25.174 > 141.223.121.126, length 98
22:59:14.988389 Non-ethernet packet.
22:59:15.223153 Non-ethernet packet.
22:59:15.238800 IP4 UDP 141.223.124.17.50763 > 239.255.255.250.1900, length 217
22:59:15.461596 IP4 UDP 141.223.124.8.54915 > 141.223.124.255.54915, length 305
22:59:15.783181 IP4 ICMP 141.223.121.126 > 172.217.25.174, length 98
22:59:15.813552 IP4 ICMP 172.217.25.174 > 141.223.121.126, length 98
22:59:15.909023 IP4 Others 141.223.109.99.0 > 224.0.0.10.0, length 74
22:59:16.173923 IP4 UDP 0.0.0.0.68 > 255.255.255.255.67, length 324
```

```
===== SUMMARY =====
:::Protocols::
[Others]
  packets: 7, bytes: 518
[TCP]
  packets: 349, bytes: 35837
[UDP]
  packets: 226, bytes: 77638
[ICMP]
  packets: 68, bytes: 6664

::IP addresses::
[108.177.125.188]
  packets: 1, bytes: 55
[141.223.1.2]
  packets: 170, bytes: 10682
[141.223.121.126]
  packets: 198, bytes: 40887
[141.223.121.255]
  packets: 5, bytes: 1399
[141.223.124.255]
  packets: 41, bytes: 11578
[141.223.83.255]
  packets: 3, bytes: 597
[162.159.130.234]
  packets: 2, bytes: 162
[172.217.25.174]
  packets: 34, bytes: 3332
[211.115.106.73]
  packets: 3, bytes: 162
[211.115.106.79]
  packets: 17, bytes: 2374
[224.0.0.10]
  packets: 7, bytes: 518
[224.0.0.251]
  packets: 9, bytes: 748
[224.0.0.252]
  packets: 4, bytes: 256
[239.255.255.250]
  packets: 74, bytes: 26584
[255.255.255.255]
  packets: 59, bytes: 16977
[35.73.126.78]
  packets: 1, bytes: 55
[35.75.32.93]
  packets: 9, bytes: 760
[40.99.67.226]
  packets: 9, bytes: 3266
[52.111.234.0]
  packets: 1, bytes: 100
[52.193.186.57]
  packets: 2, bytes: 110
[77.88.21.90]
  packets: 1, bytes: 55
```

```
:::Applications(#port)::
[0]
  packets: 75, bytes: 7182
[53]
  packets: 170, bytes: 10682
[67]
  packets: 26, bytes: 8444
[80]
  packets: 20, bytes: 2536
[137]
  packets: 3, bytes: 276
[138]
  packets: 2, bytes: 486
[443]
  packets: 25, bytes: 4508
[1900]
  packets: 52, bytes: 11228
[3702]
  packets: 22, bytes: 15356
[5278]
  packets: 1, bytes: 55
[5353]
  packets: 9, bytes: 748
[5355]
  packets: 4, bytes: 256
[17500]
  packets: 43, bytes: 10975
[49264]
  packets: 1, bytes: 66
[49278]
  packets: 1, bytes: 66
[49894]
  packets: 1, bytes: 182
[52066]
  packets: 1, bytes: 66
[54915]
  packets: 34, bytes: 10370
[57916]
  packets: 2, bytes: 160
[60151]
  packets: 9, bytes: 2778
[63643]
  packets: 2, bytes: 146
[63774]
  packets: 1, bytes: 66
[63797]
  packets: 4, bytes: 538
[63798]
  packets: 4, bytes: 547
[63799]
  packets: 4, bytes: 558
[63800]
  packets: 4, bytes: 344
[63801]
  packets: 4, bytes: 585
[63802]
  packets: 4, bytes: 371
[63803]
  packets: 4, bytes: 669
[63804]
  packets: 4, bytes: 572
[63805]
  packets: 4, bytes: 638
[63806]
  packets: 4, bytes: 328
[63807]
  packets: 4, bytes: 551
[63808]
  packets: 5, bytes: 391
```

```
:::Total::
  packets: 650, non-ethernet: 314, bytes: 120657
  END
```

## 3. Packet analysis using libpcap (Part 2)

### (1 packetlib/packet\_manager::make\_packet



```

13 #define GTP_E_MASK      0x04
14 #define GTP_S_MASK      0x02
15 #define GTP_PN_MASK     0x01
16 struct gtp_header {
17     uint8_t flags;
18     uint8_t message_type;
19     uint16_t length;
20     uint32_t TEID;
21 };
22
23
24 libpacket::Packet libpacket::packet_manager::make_packet(const struct pcap_pkthdr *pkthdr, const u_char *_packet)
25 {
26     auto *ep = (struct ether_header *)_packet; // get ethernet header
27     _packet += sizeof(struct ether_header); // get IP header
28     unsigned short ether_type = ntohs(netshort:ep->ether_type); // get protocol
29
30     Packet packet(p::new simple_packet{});
31     protocol::Protocol_info proto_info(p::new protocol::protocol_info{});
32
33     /* set packet size */
34     packet->bytes = pkthdr->caplen;
35
36     /* set captured time */
37     packet->time = std::make_shared<timeval>(pkthdr->ts);
38
39     /* only process if this packet is ethernet type */
40     if ((proto_info->is_ethernet = (ether_type == ETHERTYPE_IP)))
41         _process_ip_packet(packet._packet, &packet, &proto_info);
42
43     if(packet->is_GTP())
44     {
45         auto *ip_header = (struct ip *) _packet;
46         _packet += ip_header->ip_hl * 4; // udp pointer
47         _packet += sizeof(struct udphdr); // gtp pointer
48
49         int offset = 8;
50         auto *gtp_hdr = (gtp_header*)_packet; // gtp header
51         if (gtp_hdr->flags & (GTP_S_MASK|GTP_PN_MASK|GTP_E_MASK))
52         {
53             offset += 4;
54             if(gtp_hdr->flags & GTP_E_MASK){
55                 uint8_t next_extension = 0;
56                 do {
57                     uint8_t extension_length = (*(uint8_t *) (_packet + offset)) * 4;
58                     offset += extension_length;
59                     next_extension = *(uint8_t *) (_packet + offset - 1);
60                 } while (next_extension != 0);
61             }
62             _packet += offset;
63             _process_ip_packet(packet._packet, &packet, &proto_info);
64         }
65     }
66
67     /* set protocol info to packet */
68     packet->protocol = proto_info;
69
70     return packet;
71 }
72
73

```

- The code marked with red box is newly added to uncapusulate **GTP** packets.
  - (line 43) **GTP** packets usually use 2152 port. i decided which packet is **GTP** using the fact.
  - (line 45~47) Unwrap “**IP** header” and “**UDP** header”.

- **(line 49)** The minimum size of **GTP** header is 8 bytes.
- **(line 50)** Get minimum size of GTP header.
- **(line 51~53)** If one of the three flags is set, the header size is 12 bytes. (Extension Header Flag, Sequence Number Flag, N-PDU Number Flag)
- **(line 54~62)** When the **Extension Header Flag** is set, there may be more extended headers. The first byte of extended header is length of the header. When its last byte is non-zero, this means that there is a following extended header.
- **(line 65)** Finally, unwrap GTP header!
- **(line 66)** Process remaining packets as normal case.

## **(2) Results**

- The output file “[**result**] **Packet analysis using libpcap (Part 2).txt**” has been attached with this report.
- Screenshots of this program’s output are followed:

```

user ~/pcap/build master ± ./assignment2 ../src/internet_trace.pcap
19:13:13.590523 IP4 TCP 10.6.54.132.56636 > 202.131.29.100.80, length 118
19:13:13.591263 IP4 TCP 202.131.29.100.80 > 10.6.54.132.56636, length 110
19:13:13.633577 IP4 TCP 10.6.54.132.56636 > 202.131.29.100.80, length 720
19:13:13.640859 IP4 TCP 202.131.29.100.80 > 10.6.54.132.56636, length 1498
19:13:13.640860 IP4 TCP 202.131.29.100.80 > 10.6.54.132.56636, length 1498
19:13:13.644664 IP4 TCP 202.131.29.100.80 > 10.6.54.132.56636, length 1498
19:13:13.644745 IP4 TCP 202.131.29.100.80 > 10.6.54.132.56636, length 1498
19:13:13.644746 IP4 TCP 202.131.29.100.80 > 10.6.54.132.56636, length 1498
19:13:13.648650 IP4 TCP 202.131.29.100.80 > 10.6.54.132.56636, length 1498
19:13:13.648651 IP4 TCP 202.131.29.100.80 > 10.6.54.132.56636, length 1498
19:13:13.648814 IP4 TCP 202.131.29.100.80 > 10.6.54.132.56636, length 1498
19:13:13.650480 IP4 TCP 10.6.54.132.56637 > 202.131.29.100.80, length 98
19:13:13.651127 IP4 TCP 202.131.29.100.80 > 10.6.54.132.56637, length 104
19:13:13.652439 IP4 TCP 202.131.29.100.80 > 10.6.54.132.56636, length 1498
19:13:13.652553 IP4 TCP 202.131.29.100.80 > 10.6.54.132.56636, length 1498
19:13:14.070475 IP4 TCP 10.6.54.132.56636 > 202.131.29.100.80, length 98
19:13:14.070533 IP4 TCP 10.6.54.132.56636 > 202.131.29.100.80, length 98
19:13:14.070547 IP4 TCP 10.6.54.132.56636 > 202.131.29.100.80, length 98
19:13:14.070548 IP4 TCP 10.6.54.132.56636 > 202.131.29.100.80, length 98
19:13:14.070552 IP4 TCP 10.6.54.132.56636 > 202.131.29.100.80, length 98
19:13:14.070565 IP4 TCP 10.6.54.132.56636 > 202.131.29.100.80, length 98
19:13:14.070565 IP4 TCP 10.6.54.132.56636 > 202.131.29.100.80, length 98
19:13:14.070565 IP4 TCP 10.6.54.132.56636 > 202.131.29.100.80, length 98
19:13:14.070603 IP4 TCP 10.6.54.132.56636 > 202.131.29.100.80, length 98
19:13:14.070606 IP4 TCP 10.6.54.132.56636 > 202.131.29.100.80, length 98
19:13:14.070991 IP4 TCP 202.131.29.100.80 > 10.6.54.132.56636, length 1498
19:13:14.070994 IP4 TCP 202.131.29.100.80 > 10.6.54.132.56636, length 1498
19:13:14.070996 IP4 TCP 202.131.29.100.80 > 10.6.54.132.56636, length 1498
19:13:14.070998 IP4 TCP 202.131.29.100.80 > 10.6.54.132.56636, length 1498
19:13:14.071000 IP4 TCP 202.131.29.100.80 > 10.6.54.132.56636, length 1498
19:13:14.071002 IP4 TCP 202.131.29.100.80 > 10.6.54.132.56636, length 1498
19:13:14.071003 IP4 TCP 202.131.29.100.80 > 10.6.54.132.56636, length 421
19:13:14.100489 IP4 TCP 10.6.54.132.56636 > 202.131.29.100.80, length 98
19:13:14.111506 IP4 TCP 10.6.54.132.56636 > 202.131.29.100.80, length 98
19:13:14.111550 IP4 TCP 10.6.54.132.56636 > 202.131.29.100.80, length 98
19:13:14.111558 IP4 TCP 10.6.54.132.56636 > 202.131.29.100.80, length 98
19:13:14.111558 IP4 TCP 10.6.54.132.56636 > 202.131.29.100.80, length 98
19:13:14.111558 IP4 TCP 10.6.54.132.56636 > 202.131.29.100.80, length 98
19:13:14.111558 IP4 TCP 10.6.54.132.56636 > 202.131.29.100.80, length 98
19:13:14.111589 IP4 TCP 10.6.54.132.56636 > 202.131.29.100.80, length 98
19:13:14.112037 IP4 TCP 202.131.29.100.80 > 10.6.54.132.56636, length 104
19:13:15.290457 IP4 TCP 10.6.54.132.56710 > 202.131.24.245.80, length 118
19:13:15.291299 IP4 TCP 202.131.24.245.80 > 10.6.54.132.56710, length 110
19:13:15.301487 IP4 TCP 10.6.54.132.45914 > 202.179.179.65.80, length 118
19:13:15.303108 IP4 TCP 202.179.179.65.80 > 10.6.54.132.45914, length 110
19:13:15.324521 IP4 TCP 10.6.54.132.56710 > 202.131.24.245.80, length 98
19:13:15.325065 IP4 TCP 202.131.24.245.80 > 10.6.54.132.56710, length 104
19:13:15.333565 IP4 TCP 10.6.54.132.56710 > 202.131.24.245.80, length 750
19:13:15.333566 IP4 TCP 10.6.54.132.45914 > 202.179.179.65.80, length 98
19:13:15.334341 IP4 TCP 202.179.179.65.80 > 10.6.54.132.45914, length 104
19:13:15.340581 IP4 TCP 10.6.54.132.45914 > 202.179.179.65.80, length 746
19:13:15.340922 IP4 TCP 202.131.24.245.80 > 10.6.54.132.56710, length 514
19:13:15.341106 IP4 TCP 202.131.24.245.80 > 10.6.54.132.56710, length 104
19:13:15.346790 IP4 TCP 202.179.179.65.80 > 10.6.54.132.45914, length 508
19:13:15.347080 IP4 TCP 202.179.179.65.80 > 10.6.54.132.45914, length 104
19:13:15.370455 IP4 TCP 10.6.54.132.56710 > 202.131.24.245.80, length 98
19:13:15.381485 IP4 TCP 10.6.54.132.56710 > 202.131.24.245.80, length 98
19:13:15.381507 IP4 TCP 10.6.54.132.45914 > 202.179.179.65.80, length 98
19:13:15.381510 IP4 TCP 10.6.54.132.45914 > 202.179.179.65.80, length 98
19:13:15.381893 IP4 TCP 202.131.24.245.80 > 10.6.54.132.56710, length 104
19:13:15.381992 IP4 TCP 202.179.179.65.80 > 10.6.54.132.45914, length 104
19:13:15.392501 IP4 TCP 10.6.54.132.49576 > 125.209.210.67.80, length 118
19:13:15.393183 IP4 TCP 125.209.210.67.80 > 10.6.54.132.49576, length 110
19:13:15.420467 IP4 TCP 10.6.54.132.49576 > 125.209.210.67.80, length 98
19:13:15.421495 IP4 TCP 125.209.210.67.80 > 10.6.54.132.49576, length 104
19:13:15.435582 IP4 TCP 10.6.54.132.49576 > 125.209.210.67.80, length 727
19:13:15.452906 IP4 TCP 125.209.210.67.80 > 10.6.54.132.49576, length 1498
19:13:15.453026 IP4 TCP 125.209.210.67.80 > 10.6.54.132.49576, length 1498
19:13:15.453040 IP4 TCP 125.209.210.67.80 > 10.6.54.132.49576, length 1498
19:13:15.453042 IP4 TCP 125.209.210.67.80 > 10.6.54.132.49576, length 1498
19:13:15.453045 IP4 TCP 125.209.210.67.80 > 10.6.54.132.49576, length 1498
19:13:15.453049 IP4 TCP 125.209.210.67.80 > 10.6.54.132.49576, length 1498
19:13:15.453053 IP4 TCP 125.209.210.67.80 > 10.6.54.132.49576, length 1498

```

```

===== SUMMARY =====
::Protocols::
[TCP]
    packets: 492, bytes: 424969
[UDP]
    packets: 4, bytes: 799
::IP addresses::
[1.226.51.189]
    packets: 132, bytes: 14893
[10.6.54.132]
    packets: 301, bytes: 399810
[111.91.132.17]
    packets: 5, bytes: 1218
[113.217.240.31]
    packets: 2, bytes: 237
[125.209.210.67]
    packets: 14, bytes: 2021
[175.158.2.26]
    packets: 5, bytes: 1213
[202.131.24.245]
    packets: 5, bytes: 1162
[202.131.28.38]
    packets: 5, bytes: 1258
[202.131.29.100]
    packets: 22, bytes: 2798
[202.179.179.65]
    packets: 5, bytes: 1158
::Applications(#port)::
[53]
    packets: 2, bytes: 237
[80]
    packets: 193, bytes: 25721
[30387]
    packets: 1, bytes: 299
[38547]
    packets: 5, bytes: 625
[44378]
    packets: 3, bytes: 318
[45914]
    packets: 5, bytes: 930
[46149]
    packets: 5, bytes: 1614
[49576]
    packets: 15, bytes: 18106
[53058]
    packets: 1, bytes: 263
[54895]
    packets: 239, bytes: 351804
[56636]
    packets: 19, bytes: 24603
[56637]
    packets: 3, bytes: 312
[56710]
    packets: 5, bytes: 936
::Total::
    packets: 496, non-ethernet: 0, bytes: 425768
===== END =====

```