Programming Assignment: Programming using Libpcap

DongHyeonRyu (SS Lab, 2023246)

Contents

[Requirements to compile]

1. **cmake version** ≥ 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

```cpp
#include <csignal>
#include <csetjmp>
#include <sstream>
#include "utils/pcap_handler.h"
#include "utils/logger.h"
#include "analyzer.h"
#include "packet_manager.h"

sig_atomic_t stopFlag = 0;
jmp_buf jmpbuf;

void handler(int)
{
    stopFlag = 1;
    longjmp( env: jmpbuf, val: 1);
}

void callback(u_char *, const struct pcap_pkthdr *pkthdr, const u_char *_packet)
{
    auto packet : shared_ptr<simple_packet> = libpacket::packet_manager::make_packet(pkthdr, packet: _packet);
    utils::logger::info(packet);
    libpacket::analyzer::Gatcha(packet);
}

int main(int argc, char **argv)
{
    signal( sig: SIGINT, &handler );

    /* custom pcap wrapper object */
    auto *pcap = new utils::pcap_handler();

    pcap->find_all_devs();
    pcap->print_all_devs_info();

    /* make filter from user input */
    std::stringstream filter;
    for(int i = 1; i<argc; i++)
        filter << " " << argv[i];

    pcap_t *pcd = pcap->set_filter( device_no: 0, filter: filter.str(), buffer_size: BUFSIZ, listening_duration_in_sec: -1);

    setjmp(jmpbuf);
    if (!stopFlag) pcap->gatcha(pcd, callback);

    /* print summary */
    libpacket::analyzer::GetInstance()->analyze();

    /* save dump file */
    pcap_dumper_t *pd = pcap_dump_open(pcd, "capture.pcap");
    pcap_dump_close(pd);
}
```

- This can be executed by `./assignment1 {filter charaters}`
- The program catures 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 neccesary 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_hander.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`

```
class logger {
public:
    logger()= default;

    static std::string currentDateTime();

    template<typename T>
    static void info(const T& object, const std::string& message) {
        std::cout << "[info::" << typeid(object).name() << "] - [" << currentDateTime() << "] - " << message << std::endl;
    }

    template<typename T>
    static void warn(const T& object, const std::string &message) {
        std::cout << "[warn::" << typeid(object).name() << "] - [" << currentDateTime() << "] - " << message << std::endl;
    }

    template<typename T>
    static void error(const T& object, const std::string &message) {
        std::cout << "[error::" << typeid(object).name() << "] - [" << currentDateTime() << "] - " << message << std::endl;
        exit(status: 1);
    }

    static void info(const libpacket::Packet& packet);
};
```

- `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

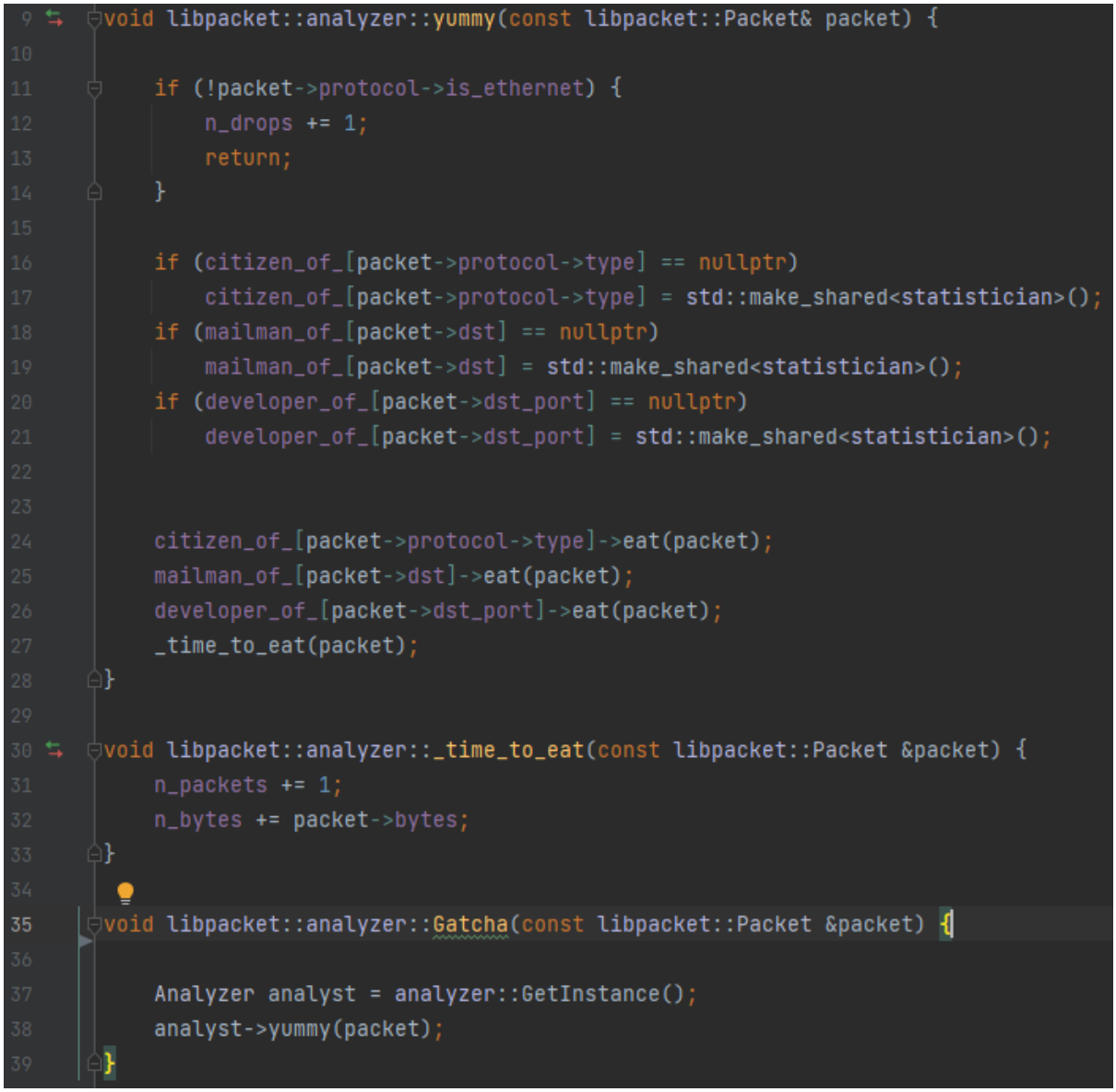
```cpp
sig_atomic_t stopFlag = 0;
jmp_buf jmpbuf;

void handler(int)
{
    stopFlag = 1;
    longjmp( env: jmpbuf, val: 1);
}

void callback(u_char *, const struct pcap_pkthdr *pkthdr, const u_char *_packet)
{
    auto packet : shared_ptr<simple_packet> = libpacket::packet_manager::make_packet(pkthdr, packet: _packet);
    utils::logger::info(packet);
    libpacket::analyzer::Gatcha(packet);
}

int main(int argc, char **argv)
{
    signal( sig: SIGINT, &handler );

    auto *pcap = new utils::pcap_handler();

    setjmp(jmpbuf);
    if (!stopFlag) pcap->gatcha( file_name: argv[1], callback);

    libpacket::analyzer::GetInstance()->analyze();
}
```

- 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



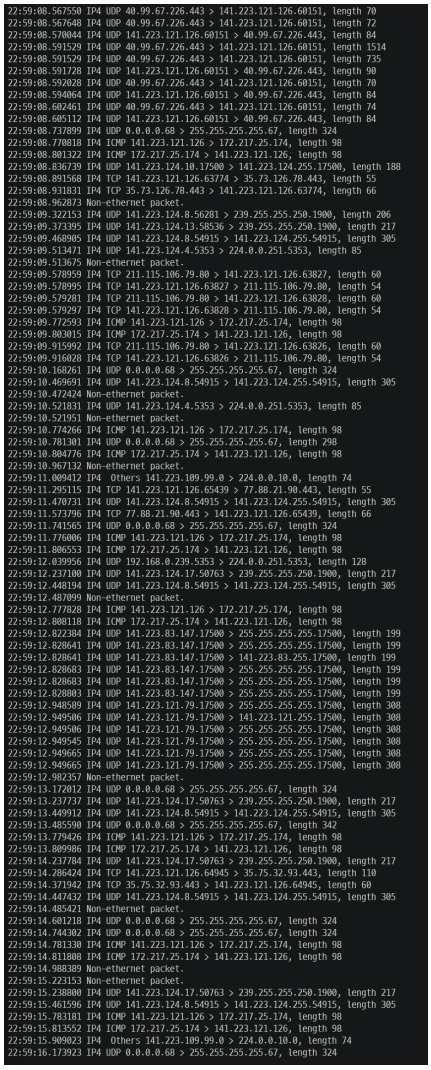
```cpp
void libpacket::analyzer::yummy(const libpacket::Packet& packet) {

    if (!packet->protocol->is_ethernet) {
        n_drops += 1;
        return;
    }

    if (citizen_of_[packet->protocol->type] == nullptr)
        citizen_of_[packet->protocol->type] = std::make_shared<statistician>();
    if (mailman_of_[packet->dst] == nullptr)
        mailman_of_[packet->dst] = std::make_shared<statistician>();
    if (developer_of_[packet->dst_port] == nullptr)
        developer_of_[packet->dst_port] = std::make_shared<statistician>();


    citizen_of_[packet->protocol->type]->eat(packet);
    mailman_of_[packet->dst]->eat(packet);
    developer_of_[packet->dst_port]->eat(packet);
    _time_to_eat(packet);
}

void libpacket::analyzer::_time_to_eat(const libpacket::Packet &packet) {
    n_packets += 1;
    n_bytes += packet->bytes;
}

void libpacket::analyzer::Gatcha(const libpacket::Packet &packet) {

    Analyzer analyst = analyzer::GetInstance();
    analyst->yummy(packet);
}
```

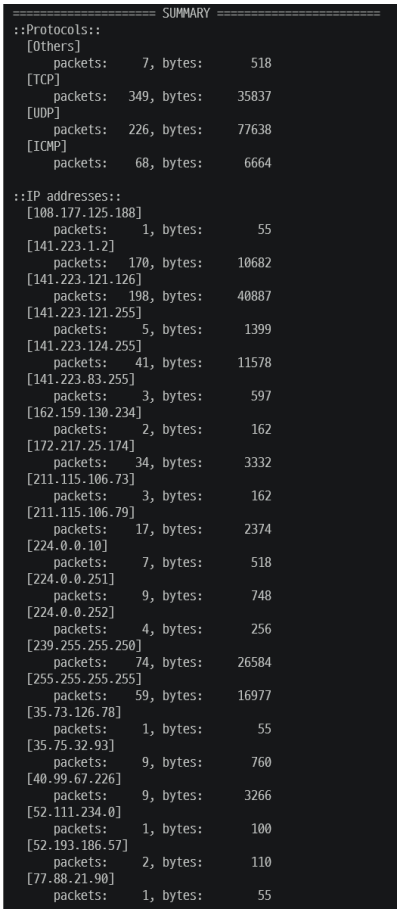
analyzer.cpp

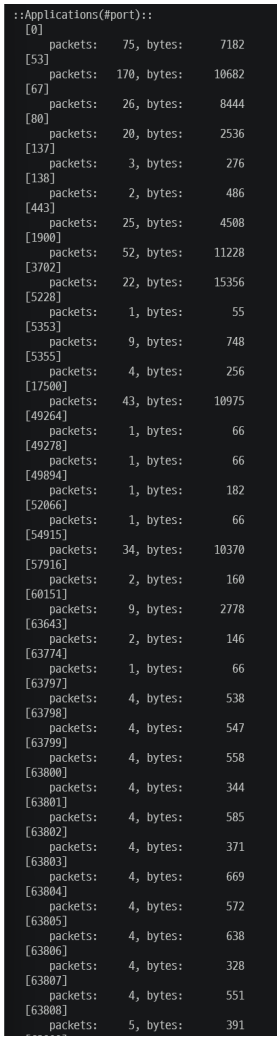
- Since `analyzer` class is a "**singleton**", the very `analyzer` instance (named "analyst") gets packets one by one via "**Gatcha**"(analyer.cpp::line 35) function.
- `analyzer` instance looks into the packet, update figures (number of packets, bytes) according to protocol type, destination address 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 attacted with this report.
- Screenshots of this program's output are followed:







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
56                  uint8_t next_extension = 0;
57                  do {
58                      uint8_t extension_length = (*(uint8_t *) (_packet + offset)) * 4;
59
60                      offset += extension_length;
61                      next_extension = *(uint8_t *) (_packet + offset - 1);
62                  } while (next_extension != 0);
63              }
64          }
65          _packet += offset;
66          _process_ip_packet(packet: _packet, &packet, &proto_info);
67      }
68
69      /* set protocol info to packet */
70      packet->protocol = proto_info;
71
72      return packet;
73  }
```

- The code marked with red box is newly added to uncapsulate **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 minimub 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 extented 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 attacted 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
19:13:14.111589 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 ========================
```