Vysoké Učení Technické v Brně

Fakulta informačních technologií

Documentation for IPK - Project 2

Packet sniffer Implementation

Marek Sarvaš (xsarva00)

Brno, 03.05.2020

Contents

1 Preface

Documentation for packet sniffer implemented in C++ language with libraries for manipulating with packets, for neccessary header structures such as ethernet header, ip header tcp header etc., namely **pcap.h, netinet/ip.h, netinet/ip6.h, netinet/tcp.h** etc.. Programme is sniffing packets using IPv4/IPv6 and UDP/TCP protocol on various ports.

2 Theory

Transport layer

Is 4th layer which transports application-layer messages between application endpoints using TCP and UDP protocols(in the internet). It breaks application messages into segments i.e. packets and sends them into internet layer where the recieving side reassembles them and passes to application layer.

Packet

Packet is an unit that carries data over network, it represents the smallest amount of data that can be transferred over a network at once. It contains control information(source destination addresses, error detection and correction etc.) and the data it is carrying. User data are encapsulated between header and trailer where control information are carried.

TCP

It is connection-oriented protocol in which the connection between client and server is established before any data is sent. TCP uses three way handshake for better error detection and reliability but adds on latency. The minimum size of header is 20 bytes and maximum 60 bytes where main segments used in this project where *source port* and *destination port*, each of them takes 16 bits.

UDP

UDP is another transport layer protocol but is unreliable and connectionless unlike TCP. It does not use three way handshake because there is no need to establish connection before data transfer. Using UDP performance is heigher it does not check for errors, drops delayed packets and hase better latency than TCP. It is highly used in pc gaming or video communication. UDP header has fixed length to 8 bytes and contains neccessary information for this project such as *source port* and *destination port* with same 16 bits length as TCP.

3 Implementation

Sniffer is implemented in one file ipk-sniffer, whole program is divided into few functions and main function. Compilation and how to run the sniffer is

3.1 Main()

First part of main function is for parsing given arguments using check_args function.If interface was not given as argument all interfaces are printed in loop.
If program got interface (or other optional arguments such as port, tcp, etc.), opens given interface for sniffing using pcap_open_live, on success compile given filter composed from given program arguments - tcp, udp, port. If compiled successfuly filter is applied on interface handler.
For actual sniffing pcap _loop function is used with arguments interface handler, number of packets to be sniffed (stored int argument structure), callback function (documented below). There is no time limit in which packet has to be sniffed, because if user wants to sniffed eg. 2 packets program will run until 2 packets are sniffed. After wanted number of packets is sniffed programme closes interface handler and frees allocated memory.
Otherwise if interface, where tcp/udp packets could not be sniffed was given, sniffer will run infinitely until interuption (eg.: with CTRL+C).

3.2 Create_filter

Creates a string filter using tcp, udp, port number from values given as programme arguments. Because programme is sniffing only tcp or udp packets default filter will be set to **"(proto tcp) or (proto udp)"**, this filter is set either if program is run with "–tcp" and "–udp" arguments or without them. "Proto" in filter means that only IPv4 and IPv6 packets will be sniffed and default tcp/udp filter means only tcp and udp packets will be sniffed. Port number is added into filter with **"and"** e.g.: **"(proto tcp) and (port 443)"**.

3.3 Callback fucntion

Function passed into pcap_loop and is called for every packet sniffed. Is responsible for parsing packet to get necessary information such as: time, protocol of packet, source and destination ports and ip addresses, resolving ip addresses into names and printing these information and whole packet on standard output.

- **Ethernet type**

  Firstly gets ethernet header from packet data and chooses if sniffed packet uses IPv4 or IPv6. According to IP gets IPv4/IPv6 header in which are neccessary information for future use such as IP source address and IP destination address, protocol(TCP/UDP) and length of IP header. For IP addresses **inet_ntop** function is used with **AF_INET** argument for IPv4 and **AF_INET6** for IPv6.

- **Source and destination names using cache**

  If the IP address is not in DNS cache, it is resolved to FQDN using **getaddrinfo** function to get rid of IPv4-IPv6 dependencies. Needed information are stored in **addrinfo** structure. This structure is next used in **getnameinfo** function to get FQDN from **ai_addr** variable. **Getnameinfo** function is used with textbfNI_NAMEREQD flags which returns error if IP address cannot be resolved, this is causing small memory leak according to *valgrind* even though everything is freed same as when IP address is resolved into FQDN. In both cases(IP address can or cannot be resolved) IP address is stored in DNS cache. If IP is already in cache none of above functions is called and FQDN is got from there. DNS cache is implemented as unordered map whre *key* is IP address and *value* is either resolved FQDN or same IP address. Cache is implemented because **getaddrinfo** and **getnameinfo** are producing another packets when called, then sniffer catches these new generated packets this can cause loops and influence output.

- **Print packet**

  Firstly is printed time when packet was sniffed which is got from packet header and computed to hh:mm:ss.ffff format using **convert_time** function. Then IPv4/IPv6 source/destination IP addresses with corresponding ports. For port numbers **ntohs** function is used. Next is printed whole packet in bytes in hexadecimal and ascii format, 16 bytes per row, every row starting with hexadecimal number which represents how many bytes were printed before current row. Format of packet ouput is same as packet representation in **Wireshark**.

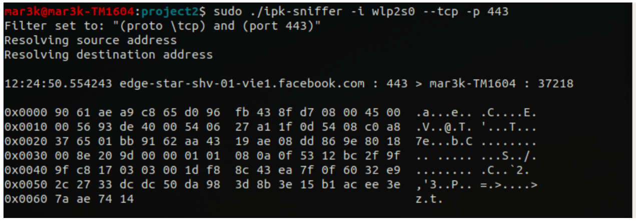
4 Testing

For testing purpose **Wireshark** application was used. Sniffer was tested only manually, filters number of packets etc., no automated testing was involved, for different configuration. Output of implemented ipk-sniffer was compared with the same packet in Wireshark for same time, source and destination ports and addresses length and data of packet.

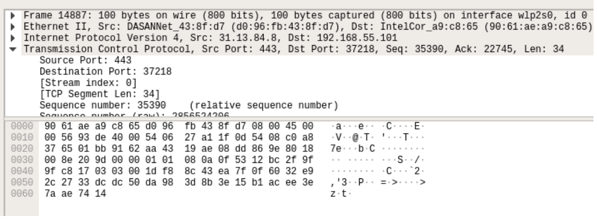
4.1 Basic filter testing

This involved testing differen configuration of filter i.e. combination of **tcp, udp** and **port number**. Also manually added in code filter for IPv6 but IPv6 address did not occure that often in ethernet traffic so that was a bit time consuming.

- **TCP protocol and port 443 filter**

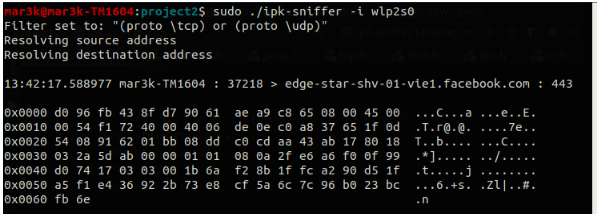


```
mar3k@mar3k-TM1604:project2$ sudo ./ipk-sniffer -i wlp2s0 --tcp -p 443
Filter set to: "(proto \tcp) and (port 443)"
Resolving source address
Resolving destination address

12:24:50.554243 edge-star-shv-01-vie1.facebook.com : 443 > mar3k-TM1604 : 37218

0x0000 90 61 ae a9 c8 65 d0 96  fb 43 8f d7 08 00 45 00  .a...e.. .C....E.
0x0010 00 56 93 de 40 00 54 06  27 a1 1f 0d 54 08 c0 a8  .V..@.T. '...T...
0x0020 37 65 01 bb 91 62 aa 43  19 ae 08 dd 86 9e 80 18  7e...b.C ........
0x0030 00 8e 20 9d 00 00 01 01  08 0a 0f 53 12 bc 2f 9f  .. ..... ...S../.
0x0040 9f c8 17 03 03 00 1d f8  8c 43 ea 7f 0f 60 32 e9  ........ .C..`2.
0x0050 2c 27 33 dc dc 50 da 98  3d 8b 3e 15 b1 ac ee 3e  ,'3..P.. =.>....>
0x0060 7a ae 74 14                                       z.t.
```

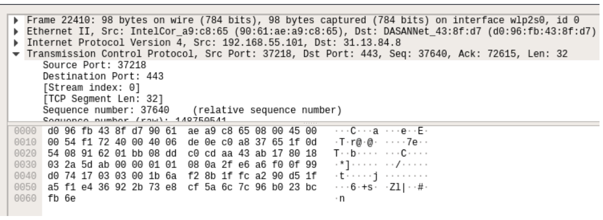


```
▸ Frame 14887: 100 bytes on wire (800 bits), 100 bytes captured (800 bits) on interface wlp2s0, id 0
▸ Ethernet II, Src: DASANNet_43:8f:d7 (d0:96:fb:43:8f:d7), Dst: IntelCor_a9:c8:65 (90:61:ae:a9:c8:65)
▸ Internet Protocol Version 4, Src: 31.13.84.8, Dst: 192.168.55.101
▾ Transmission Control Protocol, Src Port: 443, Dst Port: 37218, Seq: 35390, Ack: 22745, Len: 34
     Source Port: 443
     Destination Port: 37218
     [Stream index: 0]
     [TCP Segment Len: 34]
     Sequence number: 35390    (relative sequence number)
     Sequence number (raw): 2856524206

0000  90 61 ae a9 c8 65 d0 96  fb 43 8f d7 08 00 45 00   ·a···e·· ·C····E·
0010  00 56 93 de 40 00 54 06  27 a1 1f 0d 54 08 c0 a8   ·V··@·T· '···T···
0020  37 65 01 bb 91 62 aa 43  19 ae 08 dd 86 9e 80 18   7e···b·C ········
0030  00 8e 20 9d 00 00 01 01  08 0a 0f 53 12 bc 2f 9f   ·· ····· ···S··/·
0040  9f c8 17 03 03 00 1d f8  8c 43 ea 7f 0f 60 32 e9   ········ ·C···`2·
0050  2c 27 33 dc dc 50 da 98  3d 8b 3e 15 b1 ac ee 3e   ,'3··P·· =·>····>
0060  7a ae 74 14                                        z·t·
```

- **Without filter arguments** - TCP and UDP are default



```
mar3k@mar3k-TM1604:project2$ sudo ./ipk-sniffer -i wlp2s0
Filter set to: "(proto \tcp) or (proto \udp)"
Resolving source address
Resolving destination address

13:42:17.588977 mar3k-TM1604 : 37218 > edge-star-shv-01-vie1.facebook.com : 443

0x0000 d0 96 fb 43 8f d7 90 61  ae a9 c8 65 08 00 45 00  ...C...a ...e..E.
0x0010 00 54 f1 72 40 00 40 06  de 0e c0 a8 37 65 1f 0d  .T.r@.@. ....7e..
0x0020 54 08 91 62 01 bb 08 dd  c0 cd aa 43 ab 17 80 18  T..b.... ...C....
0x0030 03 2a 5d ab 00 00 01 01  08 0a 2f e6 a6 f0 0f 99  .*]..... ../.....
0x0040 d0 74 17 03 03 00 1b 6a  f2 8b 1f fc a2 90 d5 1f  .t.....j ........
0x0050 a5 f1 e4 36 92 2b 73 e8  cf 5a 6c 7c 96 b0 23 bc  ...6.+s. .Zl|..#.
0x0060 fb 6e                                             .n
```

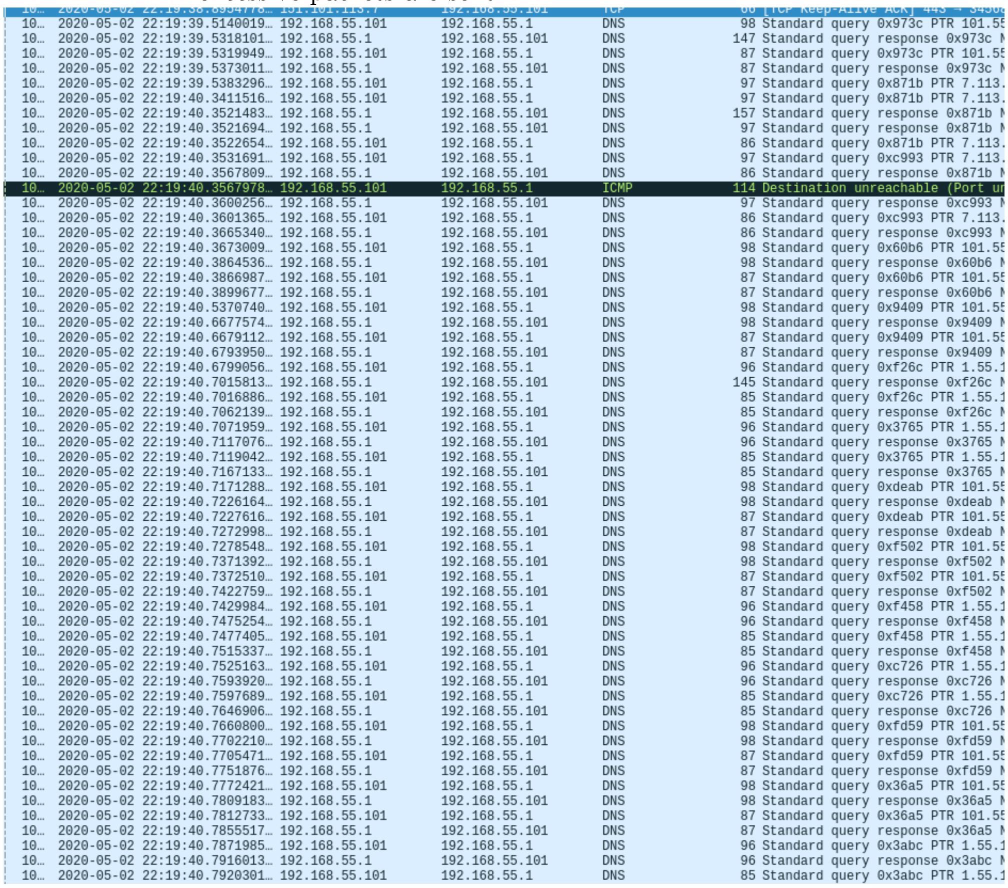


```
▸ Frame 22410: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface wlp2s0, id 0
▸ Ethernet II, Src: IntelCor_a9:c8:65 (90:61:ae:a9:c8:65), Dst: DASANNet_43:8f:d7 (d0:96:fb:43:8f:d7)
▸ Internet Protocol Version 4, Src: 192.168.55.101, Dst: 31.13.84.8
▾ Transmission Control Protocol, Src Port: 37218, Dst Port: 443, Seq: 37640, Ack: 72615, Len: 32
     Source Port: 37218
     Destination Port: 443
     [Stream index: 0]
     [TCP Segment Len: 32]
     Sequence number: 37640    (relative sequence number)
     Sequence number (raw): 148750541

0000  d0 96 fb 43 8f d7 90 61  ae a9 c8 65 08 00 45 00   ···C···a ···e··E·
0010  00 54 f1 72 40 00 40 06  de 0e c0 a8 37 65 1f 0d   ·T·r@·@· ····7e··
0020  54 08 91 62 01 bb 08 dd  c0 cd aa 43 ab 17 80 18   T··b···· ···C····
0030  03 2a 5d ab 00 00 01 01  08 0a 2f e6 a6 f0 0f 99   ·*]····· ··/·····
0040  d0 74 17 03 03 00 1b 6a  f2 8b 1f fc a2 90 d5 1f   ·t·····j ········
0050  a5 f1 e4 36 92 2b 73 e8  cf 5a 6c 7c 96 b0 23 bc   ···6·+s· ·Zl|··#·
0060  fb 6e                                              ·n
```

4.2 Name resolving with cache

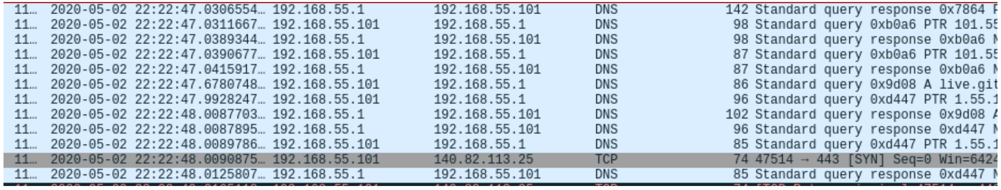
When resolving FQDN from ip address using *getaddrinfo()* and *getnameinfo()* additional packets are sent. This fact with combination of more packet sniffing e.g.: argument -n 10 can result into repeatedly sniffing only these packet for name resolving.
Examples are run with ***-n 20*** configuration and tested in **Wireshark** application.

- **without cache** excessive packets are sent



```
10…  2020-05-02 22:19:38.8954778… 151.101.113.7     192.168.55.101   TCP        66 [TCP Keep-Alive ACK] 443 → 3450
10…  2020-05-02 22:19:39.5140019… 192.168.55.101    192.168.55.1     DNS        98 Standard query 0x973c PTR 101.55
10…  2020-05-02 22:19:39.5318101… 192.168.55.1      192.168.55.101   DNS       147 Standard query response 0x973c M
10…  2020-05-02 22:19:39.5319949… 192.168.55.101    192.168.55.1     DNS        87 Standard query 0x973c PTR 101.55
10…  2020-05-02 22:19:39.5373011… 192.168.55.1      192.168.55.101   DNS        87 Standard query response 0x973c M
10…  2020-05-02 22:19:39.5383296… 192.168.55.101    192.168.55.1     DNS        97 Standard query 0x871b PTR 7.113.
10…  2020-05-02 22:19:40.3411516… 192.168.55.101    192.168.55.1     DNS        97 Standard query 0x871b PTR 7.113.
10…  2020-05-02 22:19:40.3521483… 192.168.55.1      192.168.55.101   DNS       157 Standard query response 0x871b M
10…  2020-05-02 22:19:40.3521694… 192.168.55.1      192.168.55.101   DNS        97 Standard query response 0x871b M
10…  2020-05-02 22:19:40.3522654… 192.168.55.101    192.168.55.1     DNS        86 Standard query 0x871b PTR 7.113.
10…  2020-05-02 22:19:40.3531691… 192.168.55.101    192.168.55.1     DNS        97 Standard query 0xc993 PTR 7.113.
10…  2020-05-02 22:19:40.3567809… 192.168.55.1      192.168.55.101   DNS        86 Standard query response 0x871b M
10…  2020-05-02 22:19:40.3567978… 192.168.55.101    192.168.55.1     ICMP      114 Destination unreachable (Port un
10…  2020-05-02 22:19:40.3600256… 192.168.55.1      192.168.55.101   DNS        97 Standard query response 0xc993 M
10…  2020-05-02 22:19:40.3601365… 192.168.55.101    192.168.55.1     DNS        86 Standard query 0xc993 PTR 7.113.
10…  2020-05-02 22:19:40.3665340… 192.168.55.1      192.168.55.101   DNS        86 Standard query response 0xc993 M
10…  2020-05-02 22:19:40.3673009… 192.168.55.101    192.168.55.1     DNS        98 Standard query 0x60b6 PTR 101.55
10…  2020-05-02 22:19:40.3864536… 192.168.55.1      192.168.55.101   DNS        98 Standard query response 0x60b6 M
10…  2020-05-02 22:19:40.3866987… 192.168.55.101    192.168.55.1     DNS        87 Standard query 0x60b6 PTR 101.55
10…  2020-05-02 22:19:40.3899677… 192.168.55.1      192.168.55.101   DNS        87 Standard query response 0x60b6 M
10…  2020-05-02 22:19:40.5370740… 192.168.55.101    192.168.55.1     DNS        98 Standard query 0x9409 PTR 101.55
10…  2020-05-02 22:19:40.6677574… 192.168.55.1      192.168.55.101   DNS        98 Standard query response 0x9409 M
10…  2020-05-02 22:19:40.6679112… 192.168.55.101    192.168.55.1     DNS        87 Standard query 0x9409 PTR 101.55
10…  2020-05-02 22:19:40.6793950… 192.168.55.1      192.168.55.101   DNS        87 Standard query response 0x9409 M
10…  2020-05-02 22:19:40.6799056… 192.168.55.101    192.168.55.1     DNS        96 Standard query 0xf26c PTR 1.55.1
10…  2020-05-02 22:19:40.7015813… 192.168.55.1      192.168.55.101   DNS       145 Standard query response 0xf26c M
10…  2020-05-02 22:19:40.7016886… 192.168.55.101    192.168.55.1     DNS        85 Standard query 0xf26c PTR 1.55.1
10…  2020-05-02 22:19:40.7062139… 192.168.55.1      192.168.55.101   DNS        85 Standard query response 0xf26c M
10…  2020-05-02 22:19:40.7071959… 192.168.55.101    192.168.55.1     DNS        96 Standard query 0x3765 PTR 1.55.1
10…  2020-05-02 22:19:40.7117076… 192.168.55.1      192.168.55.101   DNS        96 Standard query response 0x3765 M
10…  2020-05-02 22:19:40.7119042… 192.168.55.101    192.168.55.1     DNS        85 Standard query 0x3765 PTR 1.55.1
10…  2020-05-02 22:19:40.7167133… 192.168.55.1      192.168.55.101   DNS        85 Standard query response 0x3765 M
10…  2020-05-02 22:19:40.7171288… 192.168.55.101    192.168.55.1     DNS        98 Standard query 0xdeab PTR 101.55
10…  2020-05-02 22:19:40.7226164… 192.168.55.1      192.168.55.101   DNS        98 Standard query response 0xdeab M
10…  2020-05-02 22:19:40.7227616… 192.168.55.101    192.168.55.1     DNS        87 Standard query 0xdeab PTR 101.55
10…  2020-05-02 22:19:40.7272998… 192.168.55.1      192.168.55.101   DNS        87 Standard query response 0xdeab M
10…  2020-05-02 22:19:40.7278548… 192.168.55.101    192.168.55.1     DNS        98 Standard query 0xf502 PTR 101.55
10…  2020-05-02 22:19:40.7371392… 192.168.55.1      192.168.55.101   DNS        98 Standard query response 0xf502 M
10…  2020-05-02 22:19:40.7372510… 192.168.55.101    192.168.55.1     DNS        87 Standard query 0xf502 PTR 101.55
10…  2020-05-02 22:19:40.7422759… 192.168.55.1      192.168.55.101   DNS        87 Standard query response 0xf502 M
10…  2020-05-02 22:19:40.7429984… 192.168.55.101    192.168.55.1     DNS        96 Standard query 0xf458 PTR 1.55.1
10…  2020-05-02 22:19:40.7475254… 192.168.55.1      192.168.55.101   DNS        96 Standard query response 0xf458 M
10…  2020-05-02 22:19:40.7477405… 192.168.55.101    192.168.55.1     DNS        85 Standard query 0xf458 PTR 1.55.1
10…  2020-05-02 22:19:40.7515337… 192.168.55.1      192.168.55.101   DNS        85 Standard query response 0xf458 M
10…  2020-05-02 22:19:40.7525163… 192.168.55.101    192.168.55.1     DNS        96 Standard query 0xc726 PTR 1.55.1
10…  2020-05-02 22:19:40.7593920… 192.168.55.1      192.168.55.101   DNS        96 Standard query response 0xc726 M
10…  2020-05-02 22:19:40.7597689… 192.168.55.101    192.168.55.1     DNS        85 Standard query 0xc726 PTR 1.55.1
10…  2020-05-02 22:19:40.7646906… 192.168.55.1      192.168.55.101   DNS        85 Standard query response 0xc726 M
10…  2020-05-02 22:19:40.7660800… 192.168.55.101    192.168.55.1     DNS        98 Standard query 0xfd59 PTR 101.55
10…  2020-05-02 22:19:40.7702210… 192.168.55.1      192.168.55.101   DNS        98 Standard query response 0xfd59 M
10…  2020-05-02 22:19:40.7705471… 192.168.55.101    192.168.55.1     DNS        87 Standard query 0xfd59 PTR 101.55
10…  2020-05-02 22:19:40.7751876… 192.168.55.1      192.168.55.101   DNS        87 Standard query response 0xfd59 M
10…  2020-05-02 22:19:40.7772421… 192.168.55.101    192.168.55.1     DNS        98 Standard query 0x36a5 PTR 101.55
10…  2020-05-02 22:19:40.7809183… 192.168.55.1      192.168.55.101   DNS        98 Standard query response 0x36a5 M
10…  2020-05-02 22:19:40.7812733… 192.168.55.101    192.168.55.1     DNS        87 Standard query 0x36a5 PTR 101.55
10…  2020-05-02 22:19:40.7855517… 192.168.55.1      192.168.55.101   DNS        87 Standard query response 0x36a5 M
10…  2020-05-02 22:19:40.7871985… 192.168.55.101    192.168.55.1     DNS        96 Standard query 0x3abc PTR 1.55.1
10…  2020-05-02 22:19:40.7916013… 192.168.55.1      192.168.55.101   DNS        96 Standard query response 0x3abc M
10…  2020-05-02 22:19:40.7920301… 192.168.55.101    192.168.55.1     DNS        85 Standard query 0x3abc PTR 1.55.1
```

- **using cache** number of these packets is reduced to minimum



```
11…  2020-05-02 22:22:47.0306554… 192.168.55.1      192.168.55.101   DNS       142 Standard query response 0x7864 P
11…  2020-05-02 22:22:47.0311667… 192.168.55.101    192.168.55.1     DNS        98 Standard query 0xb0a6 PTR 101.55
11…  2020-05-02 22:22:47.0389344… 192.168.55.1      192.168.55.101   DNS        98 Standard query response 0xb0a6 M
11…  2020-05-02 22:22:47.0390677… 192.168.55.101    192.168.55.1     DNS        87 Standard query 0xb0a6 PTR 101.55
11…  2020-05-02 22:22:47.0415917… 192.168.55.1      192.168.55.101   DNS        87 Standard query response 0xb0a6 M
11…  2020-05-02 22:22:47.6780748… 192.168.55.101    192.168.55.1     DNS        86 Standard query 0x9d08 A live.git
11…  2020-05-02 22:22:47.9928247… 192.168.55.101    192.168.55.1     DNS        96 Standard query 0xd447 PTR 1.55.1
11…  2020-05-02 22:22:48.0087703… 192.168.55.1      192.168.55.101   DNS       102 Standard query response 0x9d08 A
11…  2020-05-02 22:22:48.0087895… 192.168.55.1      192.168.55.101   DNS        96 Standard query response 0xd447 M
11…  2020-05-02 22:22:48.0089786… 192.168.55.101    192.168.55.1     DNS        85 Standard query 0xd447 PTR 1.55.1
11…  2020-05-02 22:22:48.0090875… 192.168.55.101    140.82.113.25    TCP        74 47514 → 443 [SYN] Seq=0 Win=6424
11…  2020-05-02 22:22:48.0125807… 192.168.55.1      192.168.55.101   DNS        85 Standard query response 0xd447 M
```

5 References

[1] James F. Kurose, Keith W. Ross. *Computer networking : a top-down approach.* -6th edition

[2] Protocol Numbers,
`https://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml`

[3] Linux manual,
`https://linux.die.net/man/`

[4] WinPcap Unix-compatible Functions,
`https://www.winpcap.org/docs/docs_40_2/html/group__wpcapfunc.html`

[5] LibPcap,
`https://www.tcpdump.org/pcap.html`

[6] tcpdump pcap_loop,
`https://www.tcpdump.org/manpages/pcap_loop.3pcap.html`

[7] Packet filter,
`https://www.tcpdump.org/manpages/pcap-filter.7.html`