SharkFest ’17 Europe

Turning Wireshark into a Traffic Monitoring Tool:

Moving from packet details to the big picture

10 november 2017

Luca Deri

ntop

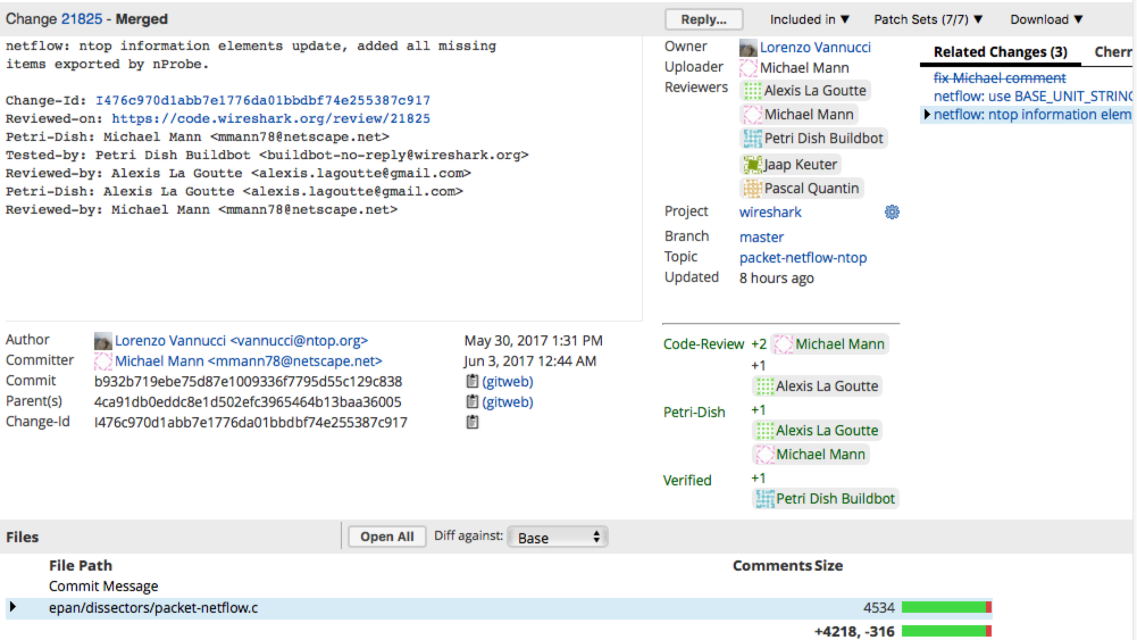
#sf17eu • Estoril, Portugal • 7-10 november 2017

Introducing ntop [1/2]

- ntop develops open source network traffic monitoring applications.
- ntop (circa 1998) is the first app we released and it is a web-based network monitoring application.
- Today our products range from traffic monitoring, high-speed packet processing, deep-packet inspection (DPI), IDS/IPS acceleration, and DDoS Mitigation.
- See http://github.com/ntop/ and in particular for Wireshark contribution https://github.com/ntop/wireshark-ntop.

Introducing ntop [2/2]

- ntop has already contributed to Wireshark by enhancing the NetFlow dissector (2005) and adding new Information Elements (2017).



Change 21825 - **Merged**

```
netflow: ntop information elements update, added all missing
items exported by nProbe.

Change-Id: I476c970d1abb7e1776da01bbdbf74e255387c917
Reviewed-on: https://code.wireshark.org/review/21825
Petri-Dish: Michael Mann <mmann78@netscape.net>
Tested-by: Petri Dish Buildbot <buildbot-no-reply@wireshark.org>
Reviewed-by: Alexis La Goutte <alexis.lagoutte@gmail.com>
Petri-Dish: Alexis La Goutte <alexis.lagoutte@gmail.com>
Reviewed-by: Michael Mann <mmann78@netscape.net>
```

Owner Lorenzo Vannucci
Uploader Michael Mann
Reviewers Alexis La Goutte
Michael Mann
Petri Dish Buildbot
Jaap Keuter
Pascal Quantin
Project wireshark
Branch master
Topic packet-netflow-ntop
Updated 8 hours ago

Related Changes (3)

~~fix Michael comment~~
netflow: use BASE_UNIT_STRIN
▸ netflow: ntop information elem

Author Lorenzo Vannucci <vannucci@ntop.org> May 30, 2017 1:31 PM
Committer Michael Mann <mmann78@netscape.net> Jun 3, 2017 12:44 AM
Commit b932b719ebe75d87e1009336f7795d55c129c838 (gitweb)
Parent(s) 4ca91db0eddc8e1d502efc3965464b13baa36005 (gitweb)
Change-Id I476c970d1abb7e1776da01bbdbf74e255387c917

Code-Review +2 Michael Mann
+1 Alexis La Goutte
Petri-Dish +1 Alexis La Goutte
Michael Mann
Verified +1 Petri Dish Buildbot

Files

Diff against: Base

File Path	Comments	Size
Commit Message		
▸ epan/dissectors/packet-netflow.c		4534
		+4218, -316

https://code.wireshark.org/review/#/c/21825/

Problem Statement [1/2]

- Wireshark is an open source packet analyser, thus a tool designed to dissect traffic in detail by analysing every packet byte.

- This approach is correct if we know in advance
  - What is the exact problem we want to solve.
  - In what part of the network is located (i.e. we know where to search for).

Problem Statement [2/2]

Unfortunately

- Sometimes we need to run Wireshark on a high-speed Internet link that can cause severe drops and thus create holes in our pcaps: 10 and 40 Gbit links are standard in most datacenter.
- Wireshark does not implement many features for giving an overview of a packet capture, and thus helping to understand what is the traffic flowing in the network.

What is an Unfamiliar Network?

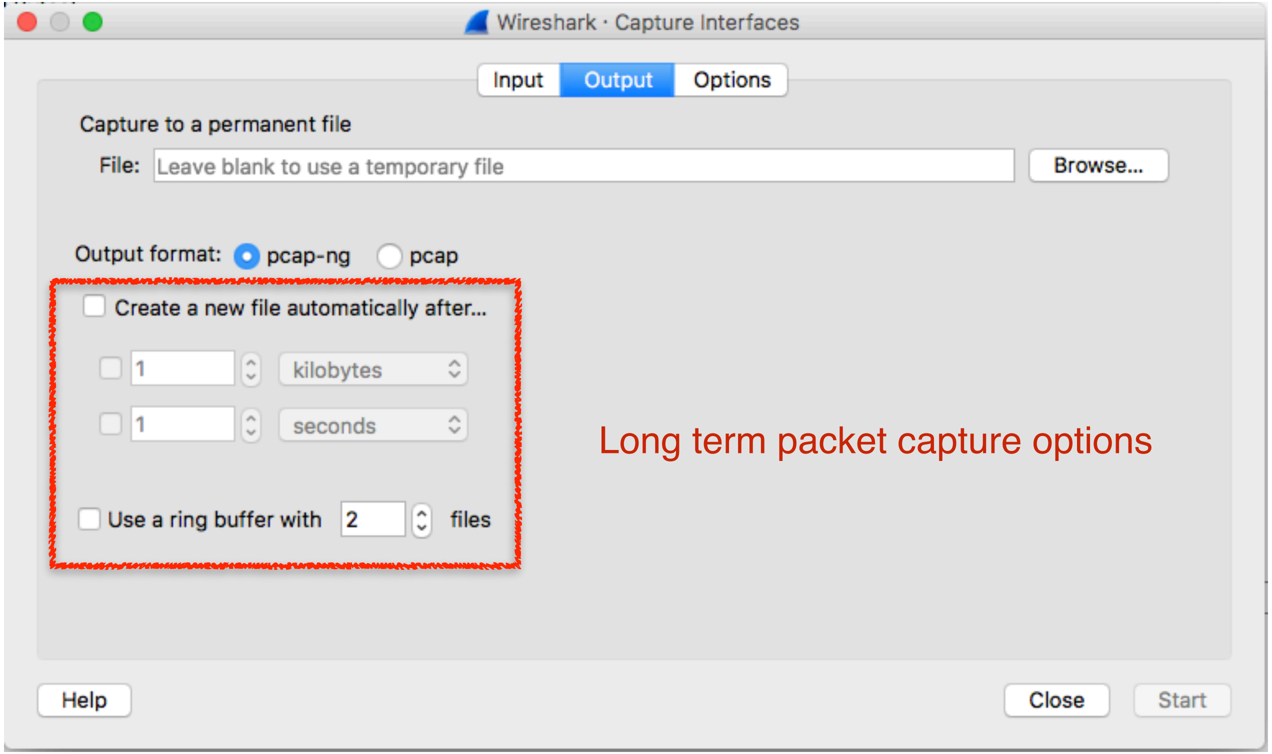
- In theory we should know “our own” network and thus this should not be unfamiliar.
- However there are many exceptions to this rule:
  - Open networks (e.g. universities) or where BYOD is standard (e.g. a coffee shop or student dorm).
  - Temporary networks (e.g. a network setup for Sharkfest) where heterogeneous people connect.
  - A network not known in advance, a consultant has been enrolled to monitor and troubleshoot

When To Monitor? [1/3]

- When using Wireshark for troubleshooting we need to know in advance where and when to search for. Example most Monday morning (when) the DHCP (what) server (where) has some problems.

- Wireshark has some options for long-term packet capture that allows people to use it as a packet-storage system, so that you can run it to capture traffic until the problem can be reproduced.

When To Monitor? [2/3]



Wireshark · Capture Interfaces

Capture to a permanent file

File: Leave blank to use a temporary file

Output format: ● pcap-ng ○ pcap

□ Create a new file automatically after...

□ 1 kilobytes

□ 1 seconds

□ Use a ring buffer with 2 files

Long term packet capture options

When To Monitor? [3/3]

- Unfortunately Wireshark does not
  - Implement a packet index for efficiently selecting specific packets out of a long packet capture (split in various files).
  - Allow packet filters to be set on application protocols. This means that “filter all Skype” traffic is not possible, and so you have to be lucky enough to troubleshoot traffic Wireshark can identify (unless you want to spend a lot of time creating complex packet filters).

Part 1
Classifying and Filtering User Traffic in Wireshark

Measure what is measurable, and make measurable what is not so
Galileo Galilei (1564 - 1642)

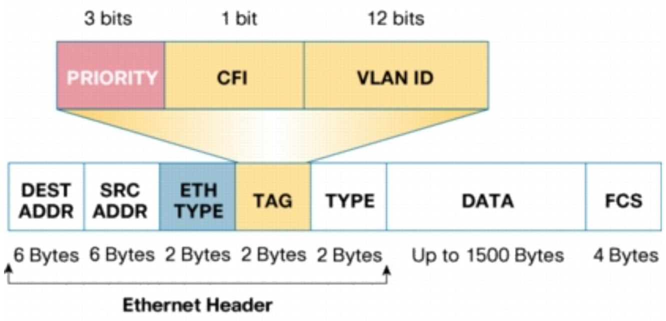
Traffic Classification: an Overview

- Traffic classification is compulsory to understand the traffic flowing on a network and enhance user experience by tuning specific network parameters.
- Main classification methods include:
  - TCP/UDP port classification
  - QoS based classification (DSCP)
  - Statistical Classification
  - Deep Packet Inspection

Port- and DSCP-based Traffic Classification

- Port-based Classification
  - In the early day of the Internet, network traffic protocols were identified by protocol and port.
  - Can classify only application protocols operating on well known ports (no rpcbind or portmap).
  - Easy to cheat and thus unreliable (TCP/80 != HTTP).

- QoS Markers (DSCP)
  - Similar to port classification but based on QoS tags.
  - Usually ignored as it is easy to cheat and forge.



Statistical Traffic Classification

- Classification of IP packets (size, port, flags, IP addresses) and flows (duration, frequency…).
- Based on rules written manually, or automatically using machine learning (ML) algorithms.
- ML requires a training set of very good quality, and it is generally computationally intensive.
- Detection rate can be as good as 95% for cases which were covered by the training set, and poor accuracy for all the other cases.

Deep Packet Inspection (DPI)

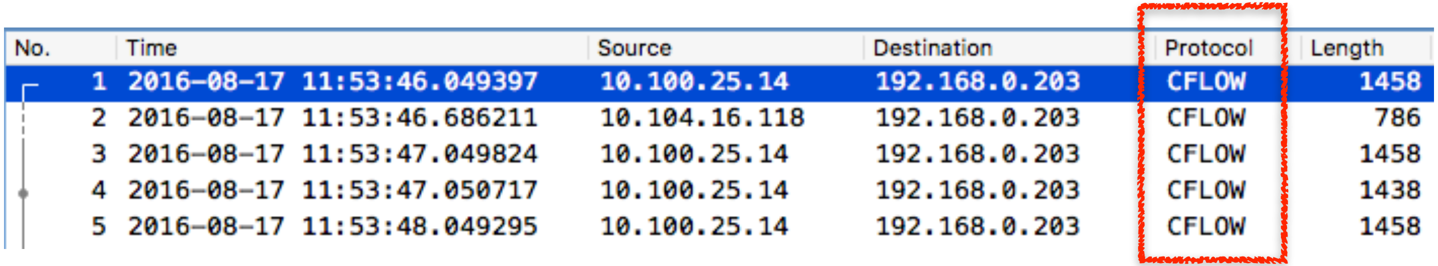
- Technique that inspects the packet payload.
- Computationally intensive with respect to simple packet header analysis.
- Concerns about privacy and confidentiality of inspected data.
- Encryption is becoming pervasive, thus challenging DPI techniques.
- No false positives unless statistical methods or IP range/flow analysis are used by DPI tools.

Using DPI in Traffic Monitoring

- Packet header analysis is no longer enough as it is unreliable and thus useless.
- Security and network administrators want to know what are the real protocols flowing on a network, this regardless of the port being used.
- Selective metadata extraction (e.g. HTTP URL or User-Agent) is necessary to perform accurate monitoring and thus this task should be performed by the DPI toolkit without replicating it on monitoring applications.

Protocol Detection in Wireshark [1/3]

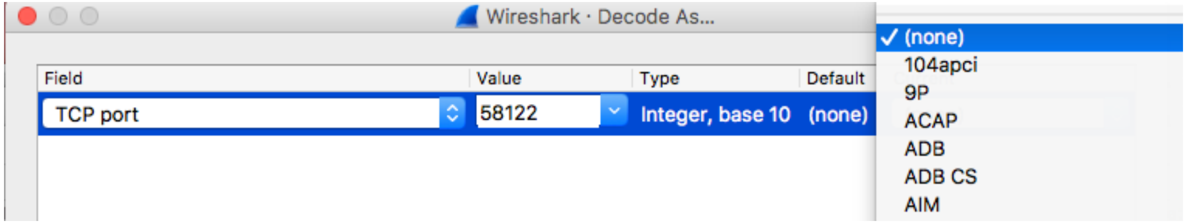
- Wireshark has "generic" protocol dissectors (e.g. TCP, TLS v1.2) for those packets that cannot be further classified.
- In case a protocol dissector is able to decode a specific traffic, Wireshark marks such traffic with the dissector name.



No.	Time	Source	Destination	Protocol	Length
1	2016-08-17 11:53:46.049397	10.100.25.14	192.168.0.203	CFLOW	1458
2	2016-08-17 11:53:46.686211	10.104.16.118	192.168.0.203	CFLOW	786
3	2016-08-17 11:53:47.049824	10.100.25.14	192.168.0.203	CFLOW	1458
4	2016-08-17 11:53:47.050717	10.100.25.14	192.168.0.203	CFLOW	1438
5	2016-08-17 11:53:48.049295	10.100.25.14	192.168.0.203	CFLOW	1458

Protocol Detection in Wireshark [2/3]

- Users can instruct Wireshark to decode a selected traffic flow using a specific protocol decoder.



Field	Value	Type	Default
TCP port	58122	Integer, base 10	(none)

- This can help when traffic that could be handled by Wireshark dissectors does not flow on standard ports and thus preventing the dissector from decoding it.

Protocol Detection in Wireshark [3/3]

- This approach has some limitations:
  - The number of protocols is limited to the supported packet decoders.
  - It is not possible to mark traffic using custom rules such as "TCP traffic on port X from host A to B is protocol Y".
  - Users cannot define new protocols based on packets fields already supported in Wireshark. Example: all HTTP requests for www.cnn.com (HTTP header Host) are marked as protocol CNN.

DPI in Wireshark [1/2]

- Limit traffic analysis at packet header level it is no longer enough (or cool).
- Network administrators want to know the real protocol without relying on the port being used.
- Once it is clear the application protocols flowing on the network, it is possible to make in-depth traffic analysis and thus selectively analyse traffic.

DPI in Wireshark [2/2]

- There are many DPI toolkits available but they are not what we look for as:
  - They are proprietary (you need to sign an NDA to use them), and costly for both purchase and maintenance.
  - Adding a new protocol requires vendor support (i.e. it has a high cost and might need time until the vendor supports it) = you’re locked-in.
- Fortunately there is another option…

Say hello to nDPI [1/2]

- ntop has decided to develop its own GPL DPI toolkit in order to build an open DPI layer for ntop and third party applications such as Wireshark.

- Supported protocols (> 225) include:
  - P2P (Skype, BitTorrent)
  - Messaging (Viber, Whatsapp, MSN, The Facebook)
  - Multimedia (YouTube, Last.gm, iTunes)
  - Conferencing (Webex, CitrixOnLine)
  - Streaming (Zattoo, Icecast, Shoutcast, Netflix)
  - Business (VNC, RDP, Citrix, *SQL)

Say hello to nDPI [2/2]

- Portable C library (Win and Unix, 32/64 bit), released according to the GNU GPLv3 license.
- Designed for user and kernel space
  - Linux ndpi-netfilter implements L7 kernel filters
- Used by many non-ntop projects (e.g. xplico.org) and part of Linux distributions (e.g. Debian).
- Able to operate on both plain ethernet traffic and encapsulated (e.g. GTP, GRE…).
- Ability to specify at runtime custom protocols (port or hostname - dns, http, https -based).

nDPI Protocol Detection [1/2]

- Similar to what Wireshark does, nDPI has a set of dissectors that try to decode packets and in case of match a positive verdict is returned.
- The match starts from the most likely protocol to match (e.g. for TCP/80 we start from HTTP) up to all the possible protocol that could match a packet.
- For instance for UDP traffic, not TCP dissectors are considered, unless the protocol could operate on both TCP and UDP (e.g. DNS)

nDPI Protocol Detection [2/2]

- nDPI protocols are specified as <network protocol>.<application protocol>.
- Example:
  - DNS query www.facebook.com = DNS.Facebook
  - HTTPS request to Facebook = SSL.Facebook
- The <application protocol> is optional (e.g. Tor traffic is simply marked as Tor).
- Match can also happen on IP address, and string applied on specific metadata such as DNS query string, HTTP host, and SSL server/client certificate.

nDPI and Wireshark

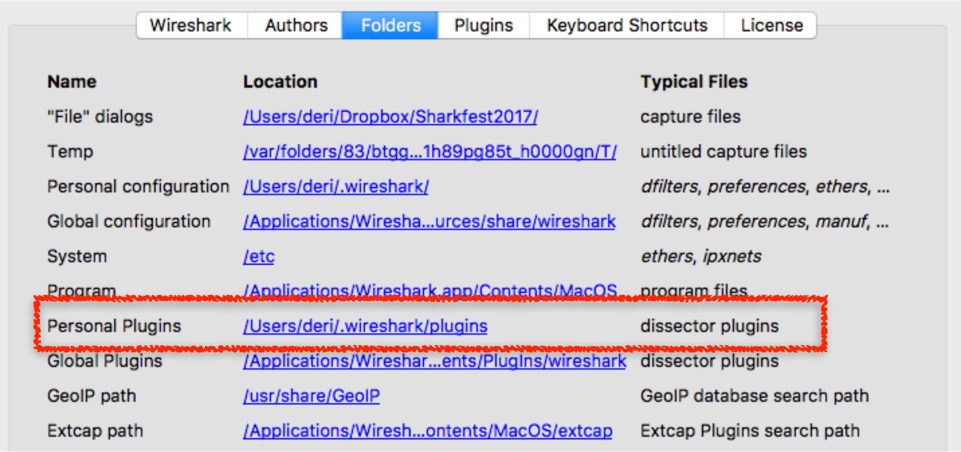
- The idea is to complement (not to replace) Wireshark dissection with nDPI in order to have the best of both worlds, and enable users to do things like:
  - Filter all the Skype traffic.
  - Compute the Facebook traffic volume on my network (i.e. how will my traffic rate change if we’ll block Facebook on the firewall?)
  - Do I have Spotify users ?

Integrating nDPI in Wireshark

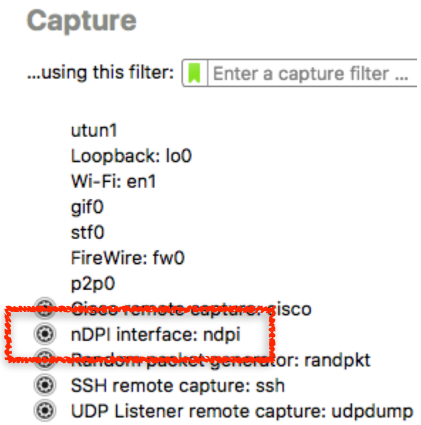
- The idea is to integrate nDPI in Wireshark “without any drawbacks”
  - Users must be able to use stock Wireshark binaries (either present in the Linux/BSD/OSX distribution or built by wireshark.org).
  - No code change (or recompilation) required.
  - Allow users to decide whether nDPI can be enabled or not.
  - Allow traffic to be filtered in Wireshark using nDPI.

Solution: Extcap + Lua [1/3]

- Extcap is an interface that allows developers to code external (i.e. not statically compiled) plugins that can be used to capture traffic and pass it to Wireshark.



Name	Location	Typical Files
"File" dialogs	/Users/deri/Dropbox/Sharkfest2017/	capture files
Temp	/var/folders/83/btgg...1h89pg85t_h0000gn/T/	untitled capture files
Personal configuration	/Users/deri/.wireshark/	*dfilters, preferences, ethers, ...*
Global configuration	/Applications/Wiresha...urces/share/wireshark	*dfilters, preferences, manuf, ...*
System	/etc	*ethers, ipxnets*
Program	/Applications/Wireshark.app/Contents/MacOS	program files
Personal Plugins	/Users/deri/.wireshark/plugins	dissector plugins
Global Plugins	/Applications/Wireshar...ents/PlugIns/wireshark	dissector plugins
GeoIP path	/usr/share/GeoIP	GeoIP database search path
Extcap path	/Applications/Wiresh...ontents/MacOS/extcap	Extcap Plugins search path



Capture

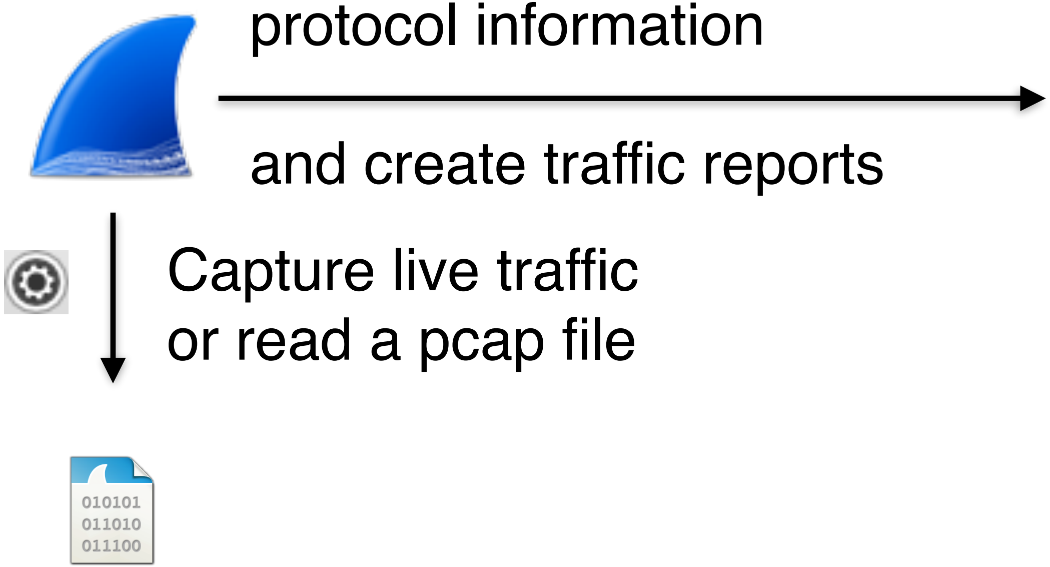
...using this filter: Enter a capture filter ...

utun1
Loopback: lo0
Wi-Fi: en1
gif0
stf0
FireWire: fw0
p2p0
Cisco remote capture: cisco
nDPI interface: ndpi
Random packet generator: randpkt
SSH remote capture: ssh
UDP Listener remote capture: udpdump

Solution: Extcap + Lua [2/3]

- Wireshark has been scriptable with Lua since many years now.
- Lua scripts can be used to dissect traffic and extend reporting capabilities.
- A new Lua script could be used to:
  - Interpret nDPI information
  - Pass it to Wireshark so that it could be used in traffic filters
  - Implement some application-based reports.

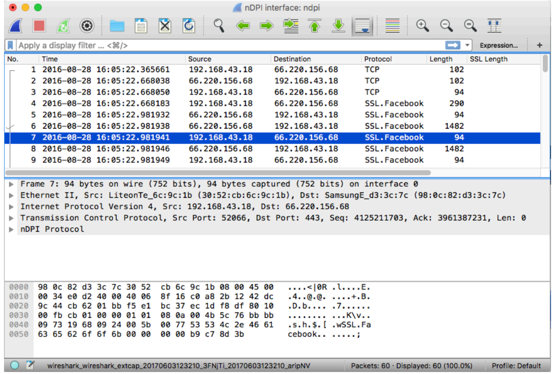
Solution: Extcap + Lua [3/3]



Lua nDPI script to visualise protocol information

and create traffic reports

Capture live traffic or read a pcap file



```
No.  Time                        Source          Destination     Protocol      Length  SSL Length
  1  2016-08-28 16:05:22.365661  192.168.43.18   66.220.156.68   TCP              102
  2  2016-08-28 16:05:22.668038  66.220.156.68   192.168.43.18   TCP              102
  3  2016-08-28 16:05:22.668050  192.168.43.18   66.220.156.68   TCP               94
  4  2016-08-28 16:05:22.668183  192.168.43.18   66.220.156.68   SSL.Facebook     290
  5  2016-08-28 16:05:22.981932  66.220.156.68   192.168.43.18   SSL.Facebook      94
  6  2016-08-28 16:05:22.981938  66.220.156.68   192.168.43.18   SSL.Facebook    1482
  7  2016-08-28 16:05:22.981941  192.168.43.18   66.220.156.68   SSL.Facebook      94
  8  2016-08-28 16:05:22.981946  66.220.156.68   192.168.43.18   SSL.Facebook    1482
  9  2016-08-28 16:05:22.981949  192.168.43.18   66.220.156.68   SSL.Facebook      94
```

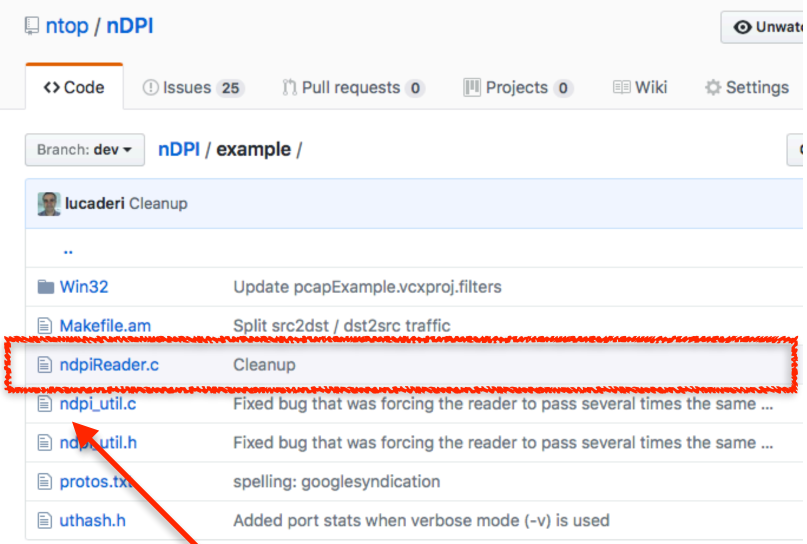
```
▶ Frame 7: 94 bytes on wire (752 bits), 94 bytes captured (752 bits) on interface 0
▶ Ethernet II, Src: LiteonTe_6c:9c:1b (30:52:cb:6c:9c:1b), Dst: SamsungE_d3:3c:7c (98:0c:82:d3:3c:7c)
▶ Internet Protocol Version 4, Src: 192.168.43.18, Dst: 66.220.156.68
▶ Transmission Control Protocol, Src Port: 52066, Dst Port: 443, Seq: 4125211703, Ack: 3961387231, Len: 0
▶ nDPI Protocol
```

```
0000  98 0c 82 d3 3c 7c 30 52  cb 6c 9c 1b 08 00 45 00   ....<|0R .l....E.
0010  00 34 e0 d2 40 00 40 06  8f 16 c0 a8 2b 12 42 dc   .4..@.@. ....+.B.
0020  9c 44 cb 62 01 bb f5 e1  bc 37 ec 1d f8 df 80 10   .D.b.... .7......
0030  00 fb cb 01 00 00 01 01  08 0a 00 4b 5c 76 bb bb   ........ ...K\v..
0040  09 73 19 68 09 24 00 5b  00 77 53 53 4c 2e 46 61   .s.h.$.[ .wSSL.Fa
0050  63 65 62 6f 6f 6b 00 00  00 00 b9 c7 8d 3b         cebook.. .....;
```

wireshark_wireshark_extcap_20170603123210_3FNjTi_20170603123210_aripNV   Packets: 60 · Displayed: 60 (100.0%)   Profile: Default

nDPI Extcap Module Code [1/2]

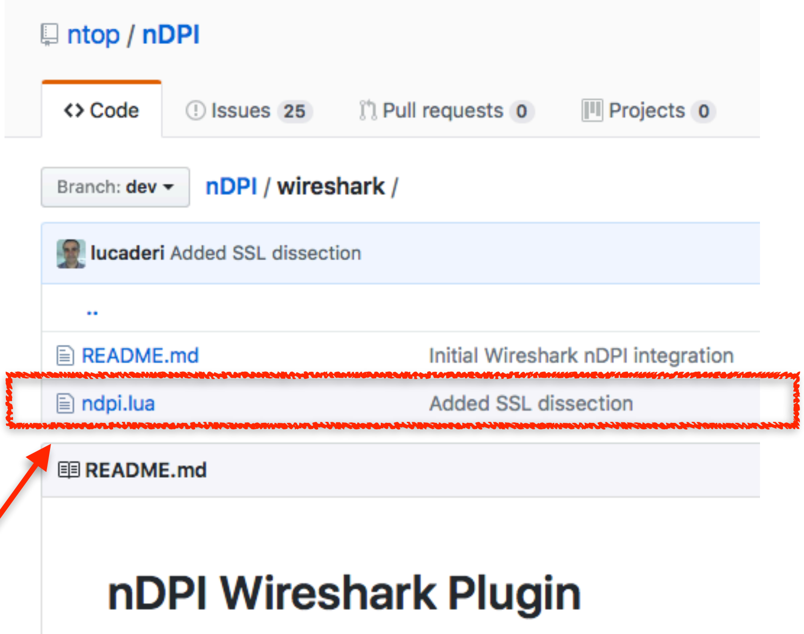
- nDPI is available from http://github.com/ntop/nDPI/



nDPI / example /

lucaderi Cleanup

..	
Win32	Update pcapExample.vcxproj.filters
Makefile.am	Split src2dst / dst2src traffic
ndpiReader.c	Cleanup
ndpi_util.c	Fixed bug that was forcing the reader to pass several times the same ...
ndpi_util.h	Fixed bug that was forcing the reader to pass several times the same ...
protos.txt	spelling: googlesyndication
uthash.h	Added port stats when verbose mode (-v) is used



nDPI / wireshark /

lucaderi Added SSL dissection

..	
README.md	Initial Wireshark nDPI integration
ndpi.lua	Added SSL dissection

README.md

nDPI Wireshark Plugin

Extcap Module

Lua Script

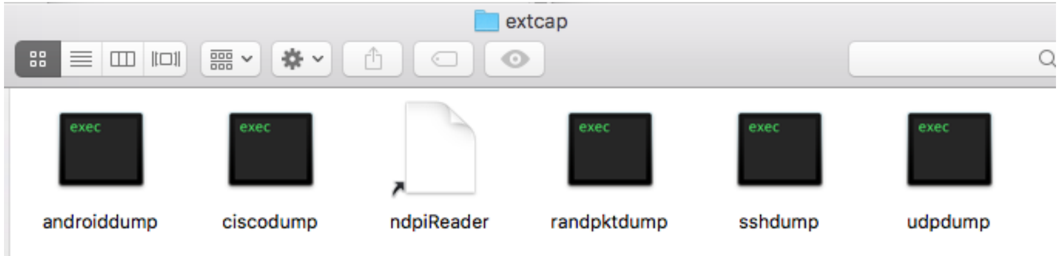
nDPI Extcap Module Code [2/2]

```
deri@Lucas-iMac.local 227> ndpiReader
Welcome to nDPI 2.1.0-860-22b7b40

ndpiReader -i <file|device> [-f <filter>][-s <duration>][-m <duration>]
          [-p <protos>][-l <loops> [-q][-d][-h][-t][-v <level>]
          [-n <threads>] [-w <file>] [-j <file>]

Usage:
  -i <file.pcap|device>     | Specify a pcap file/playlist to read packets from or a
                            | device for live capture (comma-separated list)
  -f <BPF filter>           | Specify a BPF filter for filtering selected traffic
….

Excap (wireshark) options:
  --extcap-interfaces
  --extcap-version
  --extcap-dlts
  --extcap-interface <name>
  --extcap-config
  --capture
  --extcap-capture-filter
  --fifo <path to file or pipe>
  --debug
```

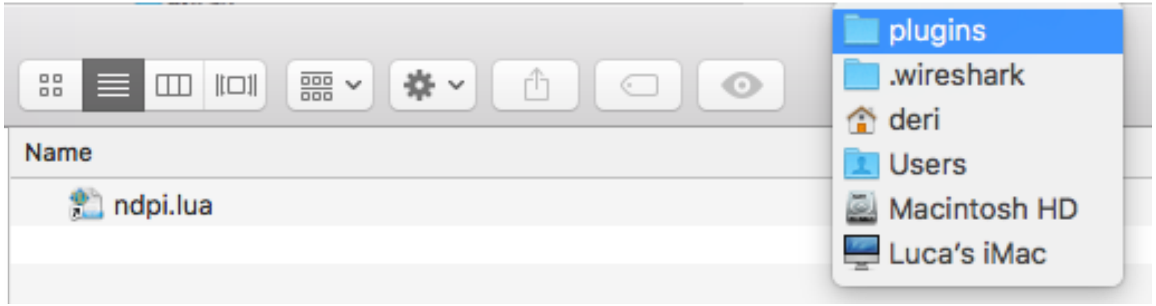
Extcap extensions

nDPI Extcap Installation

Extcap Module



Lua Script



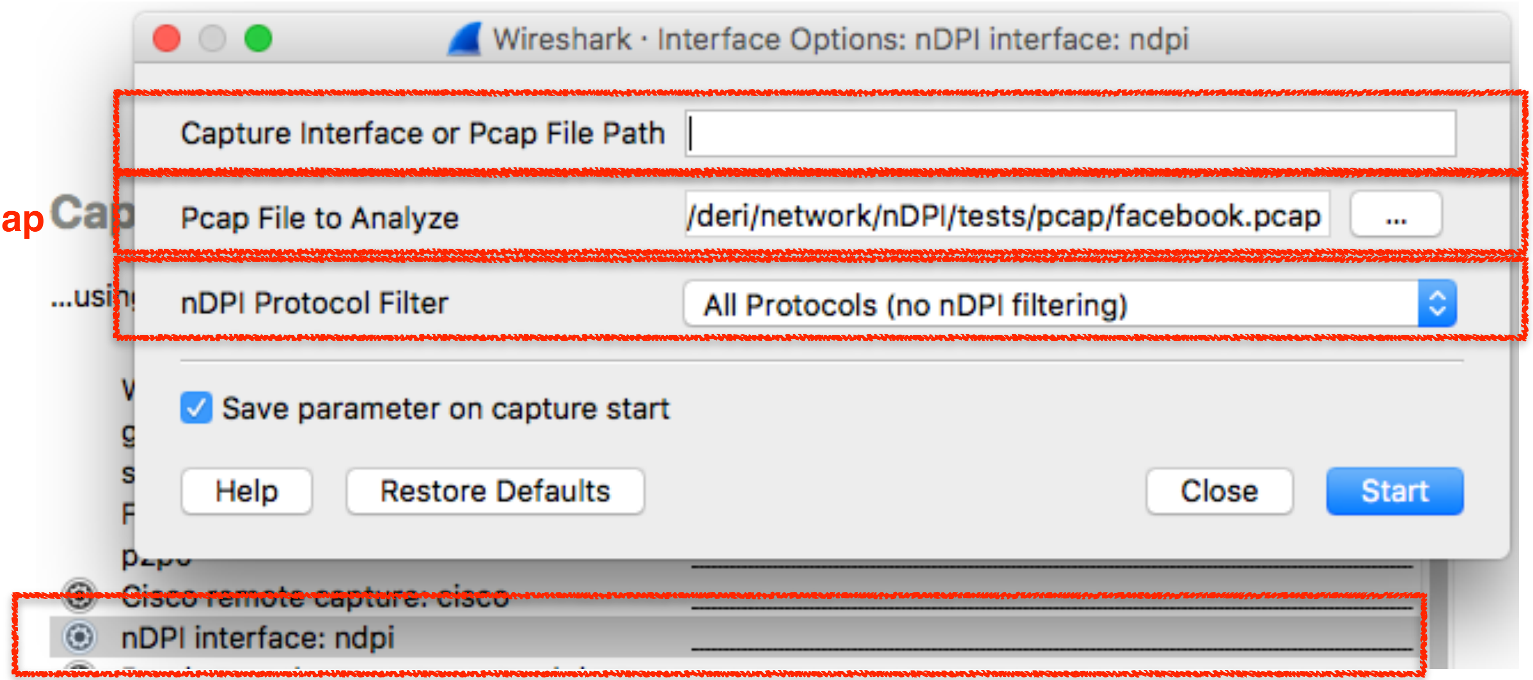
nDPI Extcap Module: Traffic Classification [1/4]

Live Capture

Capture from Pcap

nDPI Filter

Click here first



Wireshark · Interface Options: nDPI interface: ndpi

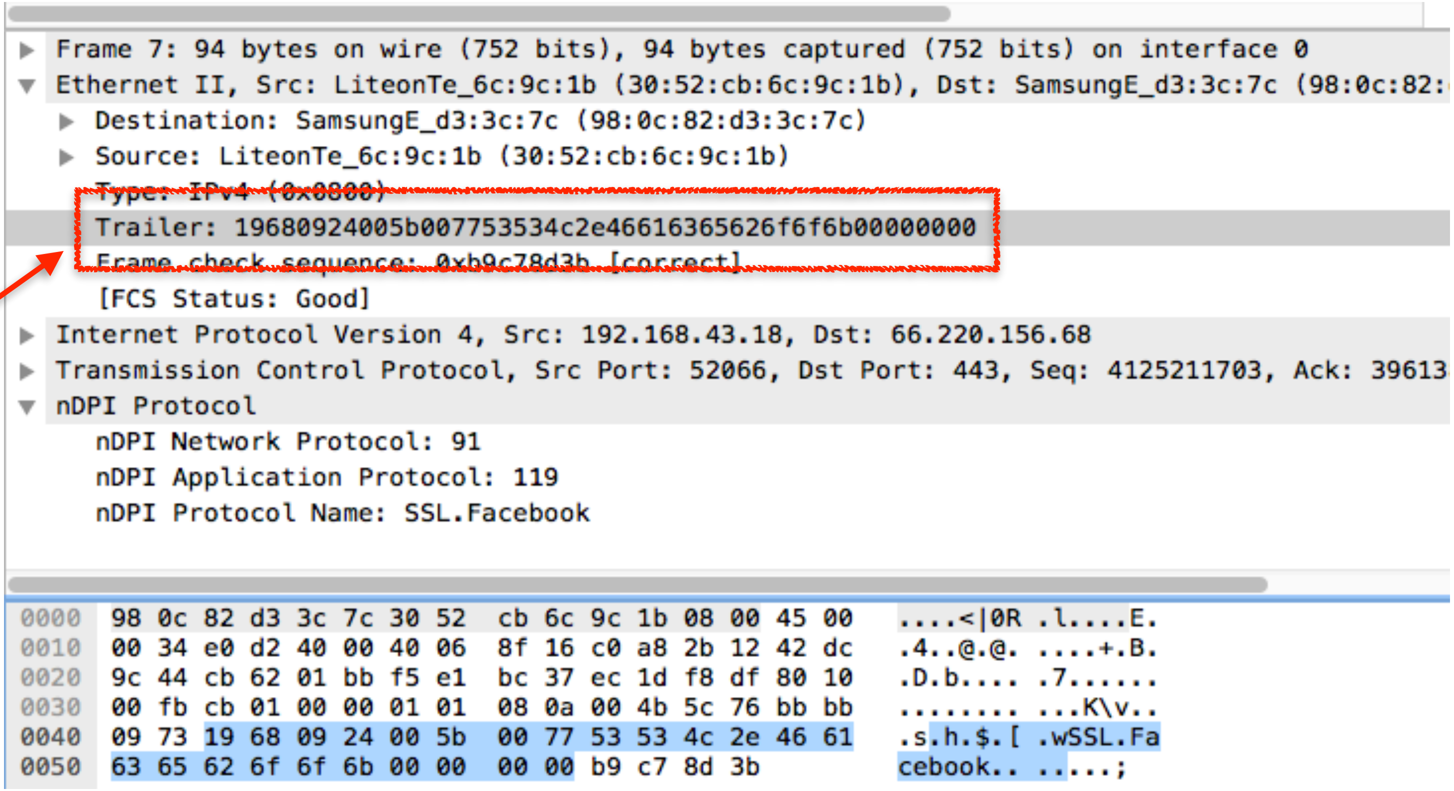
Capture Interface or Pcap File Path:

Pcap File to Analyze: /deri/network/nDPI/tests/pcap/facebook.pcap

nDPI Protocol Filter: All Protocols (no nDPI filtering)

☑ Save parameter on capture start

nDPI Extcap Module: Traffic Classification [2/4]



```
▶ Frame 7: 94 bytes on wire (752 bits), 94 bytes captured (752 bits) on interface 0
▼ Ethernet II, Src: LiteonTe_6c:9c:1b (30:52:cb:6c:9c:1b), Dst: SamsungE_d3:3c:7c (98:0c:82:
  ▶ Destination: SamsungE_d3:3c:7c (98:0c:82:d3:3c:7c)
  ▶ Source: LiteonTe_6c:9c:1b (30:52:cb:6c:9c:1b)
    Type: IPv4 (0x0800)
    Trailer: 19680924005b007753534c2e46616365626f6f6b00000000
    Frame check sequence: 0xb9c78d3b [correct]
    [FCS Status: Good]
▶ Internet Protocol Version 4, Src: 192.168.43.18, Dst: 66.220.156.68
▶ Transmission Control Protocol, Src Port: 52066, Dst Port: 443, Seq: 4125211703, Ack: 39613
▼ nDPI Protocol
    nDPI Network Protocol: 91
    nDPI Application Protocol: 119
    nDPI Protocol Name: SSL.Facebook
```

```
0000  98 0c 82 d3 3c 7c 30 52  cb 6c 9c 1b 08 00 45 00   ....<|0R .l....E.
0010  00 34 e0 d2 40 00 40 06  8f 16 c0 a8 2b 12 42 dc   .4..@.@. ....+.B.
0020  9c 44 cb 62 01 bb f5 e1  bc 37 ec 1d f8 df 80 10   .D.b.... .7......
0030  00 fb cb 01 00 00 01 01  08 0a 00 4b 5c 76 bb bb   ........ ...K\v..
0040  09 73 19 68 09 24 00 5b  00 77 53 53 4c 2e 46 61   .s.h.$.[ .wSSL.Fa
0050  63 65 62 6f 6f 6b 00 00  00 00 b9 c7 8d 3b         cebook.. ....;
```

Added via Extcap

nDPI Extcap Module: Traffic Classification [3/4]

The Extcap module

- Adds a ethernet trailer (similar to what Ixia or Gigamon devices do)

```
struct ndpi_packet_trailer {
  u_int32_t magic; /* 0x19682017 */
  u_int16_t master_protocol /* e.g. HTTP */,
            app_protocol /* e.g. FaceBook */;
  char name[16];
};
```

nDPI Protocols

- Recomputes the ethernet checksum to make sure the captured packet is not marked as invalid from Wireshark.

nDPI Extcap Module: Traffic Classification [4/4]

- Pros
  - No need to modify Wireshark as apps are decoupled thanks to the Extcap interface.
  - Whenever nDPI is updated (weekly) and new protocols dissected, Wireshark can immediately use them without being in sync with nDPI protocols.
- Cons
  - The extcap module adds a ethernet trailer that was no present on the original packet.
  - The packet trailer contains the application protocol name (to avoid misconfigurations) that adds extra payload.

nDPI Lua Module [1/2]

Responsible for:

- Interpreting the additional packet trailer, passing this information to Wireshark so that it could be used in packet filters.
- Creating nDPI reports that contain top nDPI protocols and flows.

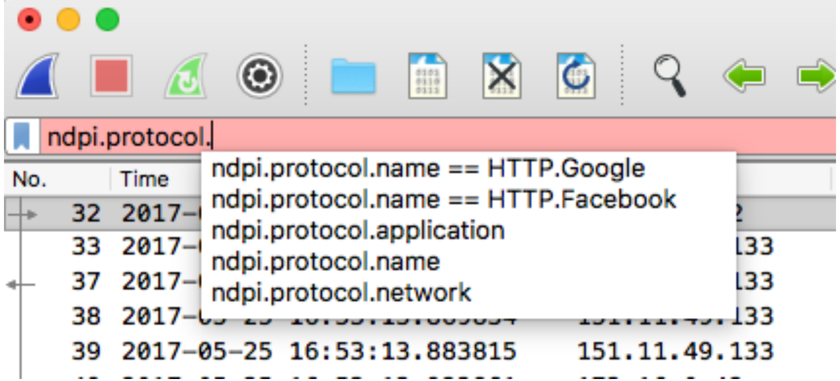
nDPI Lua Module [2/2]

```
nDPI Protocol Breakdown
-----------------------
Unknown                                    697.92 KB       [43.5 %]
LotusNotes                                 340.92 KB       [21.2 %]
RTP                                        184.85 KB       [11.5 %]
HTTP.Facebook                              103.4 KB        [6.4 %]
NFS                                        99.16 KB        [6.2 %]
SSL.Office365                              83.24 KB        [5.2 %]
HTTP                                       38.98 KB        [2.4 %]

Top nDPI Flows
-----------
192.168.17.101 / 192.168.17.40      [Unknown]         407.37 KB        [25.4 %]
192.168.18.104 / 192.168.18.106     [LotusNotes]      332.29 KB        [20.7 %]
192.168.14.139 / 192.168.14.136     [RTP]             184.85 KB        [11.5 %]
192.168.13.51 / 192.168.13.100      [Unknown]         114.32 KB        [7.1 %]
192.168.11.51 / 192.168.11.100      [Unknown]         111.99 KB        [7 %]
192.168.12.131 / 66.220.146.32      [HTTP.Facebook]   79.52 KB         [5 %]
65.55.171.156 / 192.168.16.101      [SSL.Office365]   59.25 KB         [3.7 %]
```

nDPI Filters: Runtime [1/2]

- nDPI information has been integrated into Wireshark as first citizen, thus it can be used to filter traffic after packet capture.

Runtime Filter Integration

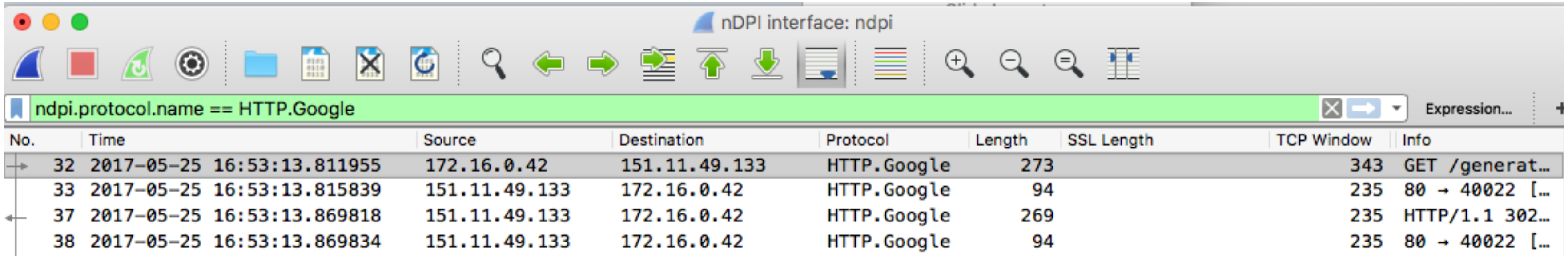


```
ndpi.protocol.
```

```
ndpi.protocol.name == HTTP.Google
ndpi.protocol.name == HTTP.Facebook
ndpi.protocol.application
ndpi.protocol.name
ndpi.protocol.network
```

```
No.  Time
32   2017-
33   2017-                                  133
37   2017-                                  133
38   2017-                                  133
39   2017-05-25 16:53:13.883815    151.11.49.133
```

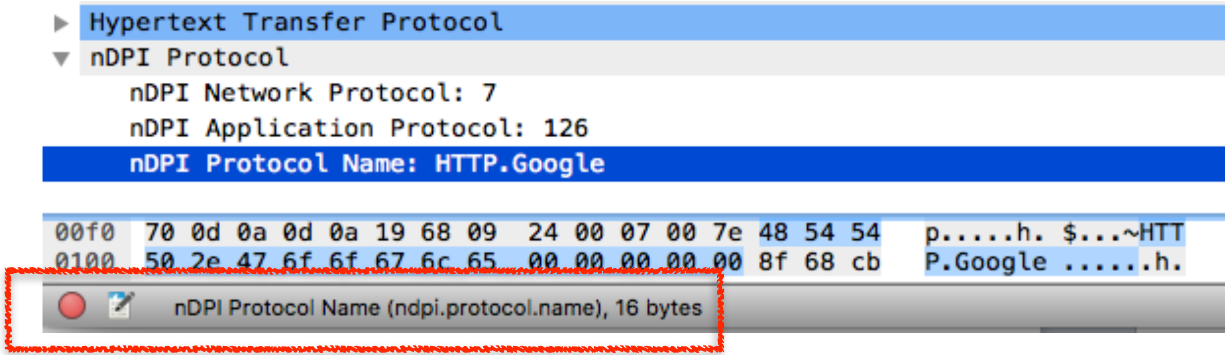
nDPI Filters: Runtime [2/2]



nDPI interface: ndpi

`ndpi.protocol.name == HTTP.Google`

No.	Time	Source	Destination	Protocol	Length	SSL Length	TCP Window	Info
32	2017-05-25 16:53:13.811955	172.16.0.42	151.11.49.133	HTTP.Google	273		343	GET /generat…
33	2017-05-25 16:53:13.815839	151.11.49.133	172.16.0.42	HTTP.Google	94		235	80 → 40022 […
37	2017-05-25 16:53:13.869818	151.11.49.133	172.16.0.42	HTTP.Google	269		235	HTTP/1.1 302…
38	2017-05-25 16:53:13.869834	151.11.49.133	172.16.0.42	HTTP.Google	94		235	80 → 40022 […



```
▶ Hypertext Transfer Protocol
▼ nDPI Protocol
    nDPI Network Protocol: 7
    nDPI Application Protocol: 126
    nDPI Protocol Name: HTTP.Google
```

```
00f0  70 0d 0a 0d 0a 19 68 09  24 00 07 00 7e 48 54 54   p.....h. $...~HTT
0100  50 2e 47 6f 6f 67 6c 65  00 00 00 00 00 8f 68 cb   P.Google ......h.
```

nDPI Protocol Name (ndpi.protocol.name), 16 bytes

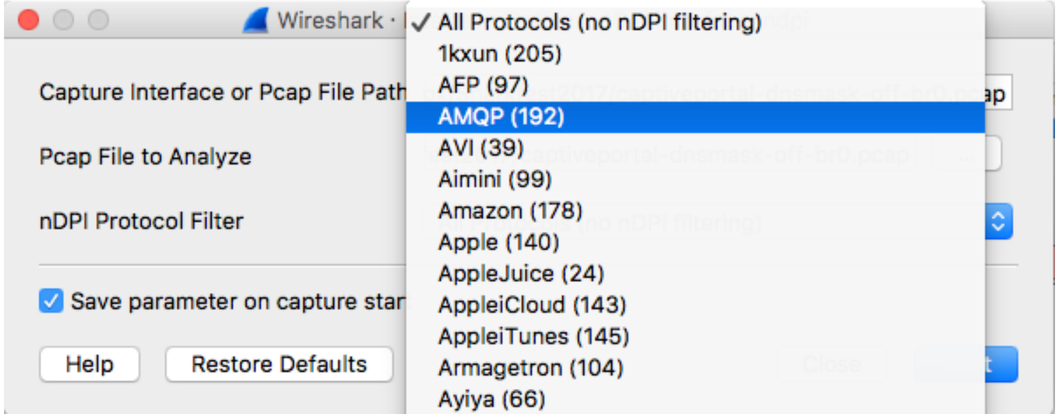
nDPI Filters: Capture [1/2]

- The nDPI Extcap module allows packet filtering to be performed during packet capture.

- This has the advantage of:
  - Discarding unwanted packets that will not reach Wireshark at all
  - Improving Wireshark responsiveness and reducing resource usage.

nDPI Filters: Capture [2/2]

- The filtered protocol is selected in the nDPI Extcap startup window by selecting the protocol name from the dropdown menu.



nDPI Filtering Limitations

- Due to the nature of DPI some protocols can be
  - Recognised with just one packet (e.g. SNMP)
  - Others need a few. For instance for detecting HTTP we need to receive at least 3 (TCP 3WH) + 2 (one per direction) packets.

- This means that the first few packets of a flow might be marked as TCP whereas the rest of the flow marked as SSL.Facebook.

Part 1 Summary

- Thanks to nDPI in Wireshark it is possible to
  - Extract efficiently packets out of large pcap traces using packet indexes and metadata.
  - nDPI can help to understand what traffic is flowing on a network, by complementing dissector information provided by Wireshark.

Part 2
Turning Wireshark in a Traffic Monitoring Application

Problem Statement

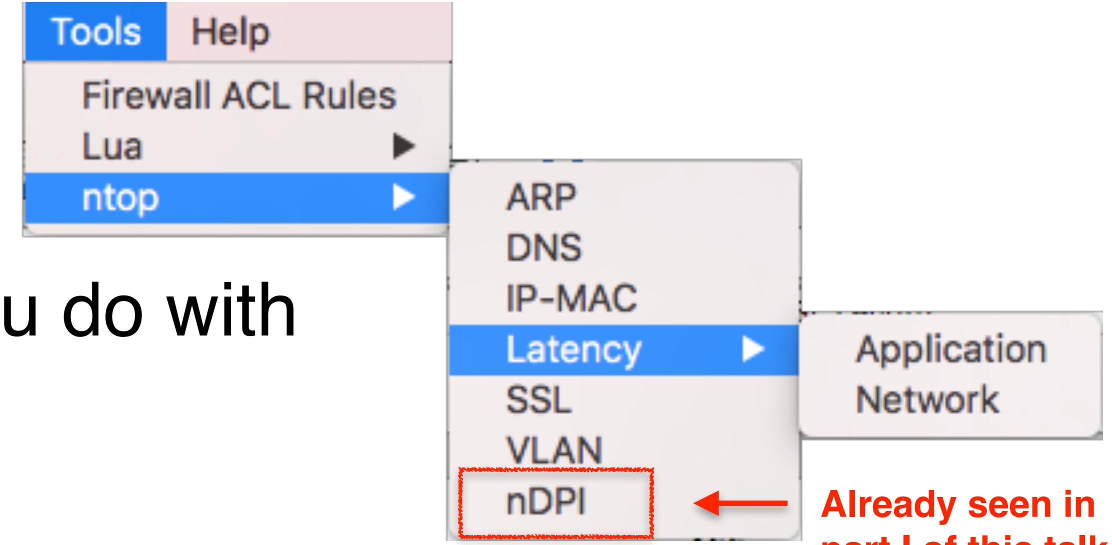
- What are the basic metrics that we are analysing when monitoring a network?
- nDPI can help us understanding what application protocols are used on our network and what are the top talkers sorted per application.
- Wireshark provides us many metrics but they are mostly packet-oriented metrics.
- We are missing a bird-eye view on the network to help us understanding the overall picture before diving into specific traffic analysis using Wireshark dissectors.

Turning Wireshark in a Traffic Monitoring Tool [1/3]

- Wireshark provides rich packet-level metrics (i.e. we already have the data we need).
- Lua scripting can be used to glue this information and create a summary that can help traffic administrators to understand what is happening on our network.
- In essence Wireshark has all the ingredients we’re looking for: we just need to glue them up.

Turning Wireshark in a Traffic Monitoring Tool [2/3]

- Just to demonstrate how easy is to turn Wireshark in a general-purpose network monitoring tool that can be used to create a basic knowledge of traffic flowing on an unfamiliar network.
- This is just an example of what you do with Lua and Wireshark.



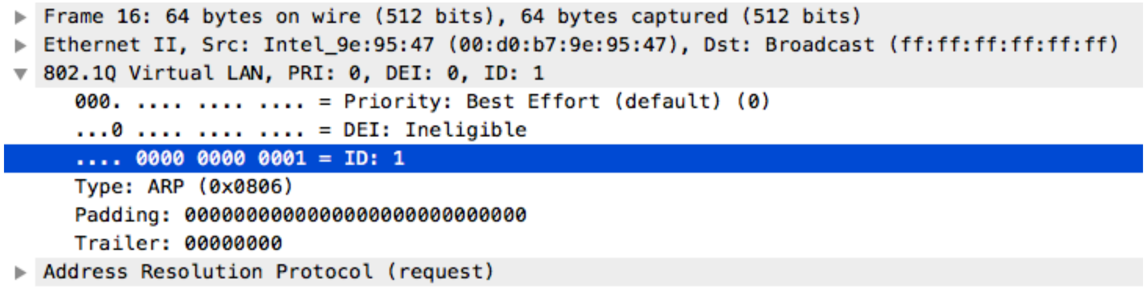
Already seen in part I of this talk

Turning Wireshark in a Traffic Monitoring Tool [3/3]

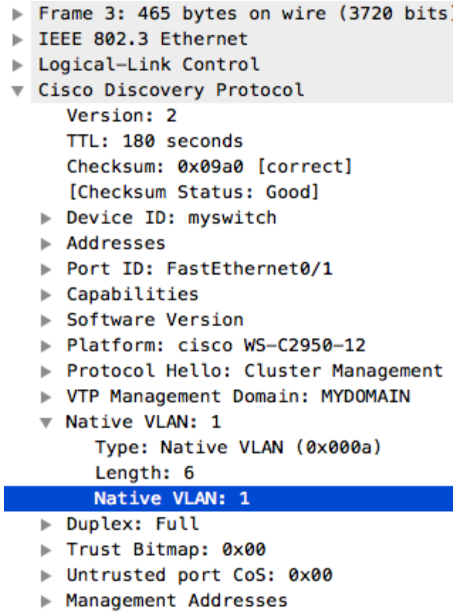
- In this talk we do not plan to cover Wireshark+Lua. If interested please see Stig Bjørlykke, Lua Scripting in Wireshark, SHARKFEST '09.
- As said earlier on this talk, all the Lua code is on https://github.com/ntop/nDPI/tree/dev/wireshark and it is integrated in the ndpi.lua script (one script does all).

VLANs [1/3]

- Wireshark is able to dissect VLAN-tagged packets both inside the packet header or in other packet types (e.g. CDP)



```
▶ Frame 16: 64 bytes on wire (512 bits), 64 bytes captured (512 bits)
▶ Ethernet II, Src: Intel_9e:95:47 (00:d0:b7:9e:95:47), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
▼ 802.1Q Virtual LAN, PRI: 0, DEI: 0, ID: 1
    000. .... .... .... = Priority: Best Effort (default) (0)
    ...0 .... .... .... = DEI: Ineligible
    .... 0000 0000 0001 = ID: 1
    Type: ARP (0x0806)
    Padding: 0000000000000000000000000000
    Trailer: 00000000
▶ Address Resolution Protocol (request)
```



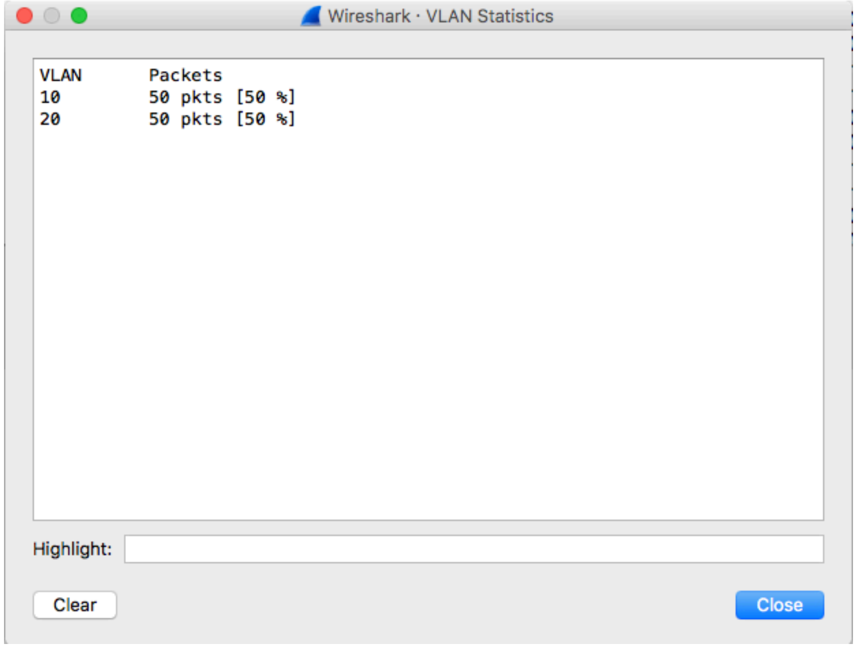
```
▶ Frame 3: 465 bytes on wire (3720 bits
▶ IEEE 802.3 Ethernet
▶ Logical-Link Control
▼ Cisco Discovery Protocol
    Version: 2
    TTL: 180 seconds
    Checksum: 0x09a0 [correct]
    [Checksum Status: Good]
  ▶ Device ID: myswitch
  ▶ Addresses
  ▶ Port ID: FastEthernet0/1
  ▶ Capabilities
  ▶ Software Version
  ▶ Platform: cisco WS-C2950-12
  ▶ Protocol Hello: Cluster Management
  ▶ VTP Management Domain: MYDOMAIN
  ▼ Native VLAN: 1
      Type: Native VLAN (0x000a)
      Length: 6
      Native VLAN: 1
  ▶ Duplex: Full
  ▶ Trust Bitmap: 0x00
  ▶ Untrusted port CoS: 0x00
  ▶ Management Addresses
```

VLANs [2/3]

- However what is missing in Wireshark is the ability to answer to a simple question. What are all the VLAN-tagged packets flowing on my network?
- This can help identifying traffic that:
  - Was not properly tagged (e.g. the wrong VLAN was used).
  - Invalid VLAN ID transported on trunk ports (sometimes the configurations are set to "just transport all VLANs")

VLANs [3/3]

- Are these two VLANs all we expect?
- Is the traffic supposed to be evenly distributed or this is a symptom of a problem?
- NOTE: Wireshark takes care of encapsulations so all VLAN encapsulations are supported by the script.



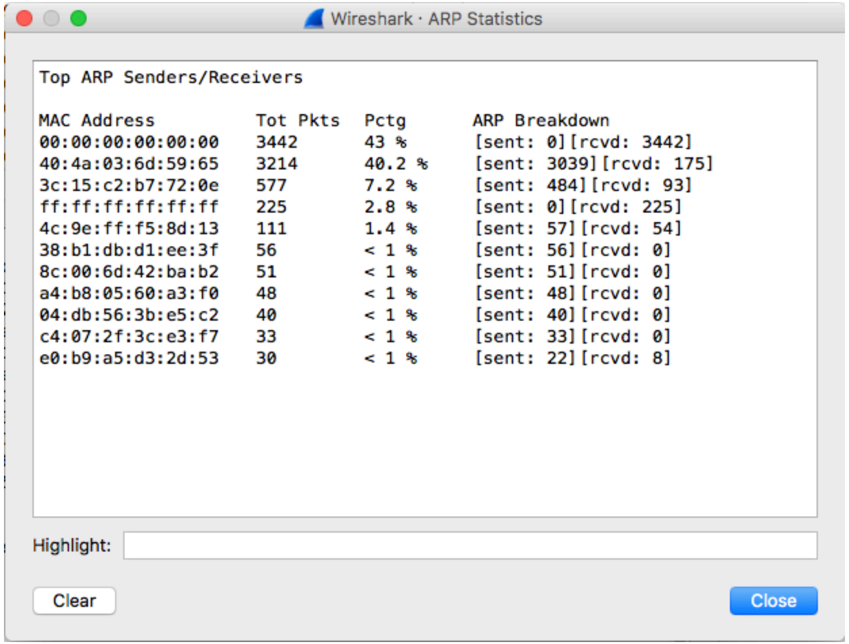
```
VLAN       Packets
10         50 pkts [50 %]
20         50 pkts [50 %]
```

ARP [1/2]

- Layer 2 protocols have been considered as 2nd-class protocols (e.g. NetFlow is a layer 3 technology).
- In addition to IPv4 address resolution, ARP is important for :
  - Monitoring devices activities (is my device up?).
  - Detecting scans and contacts towards hosts down or unreachable. Tools like nmap.org or fing.io (also) use ARP for scanning networks.

ARP [2/2]

- Is it normal that one host has sent most ARP requests? Why?
- What role has this host on out network?
- Is anybody doing an ARP scan perhaps?
- How much is the ARP traffic when compared the overall network traffic.



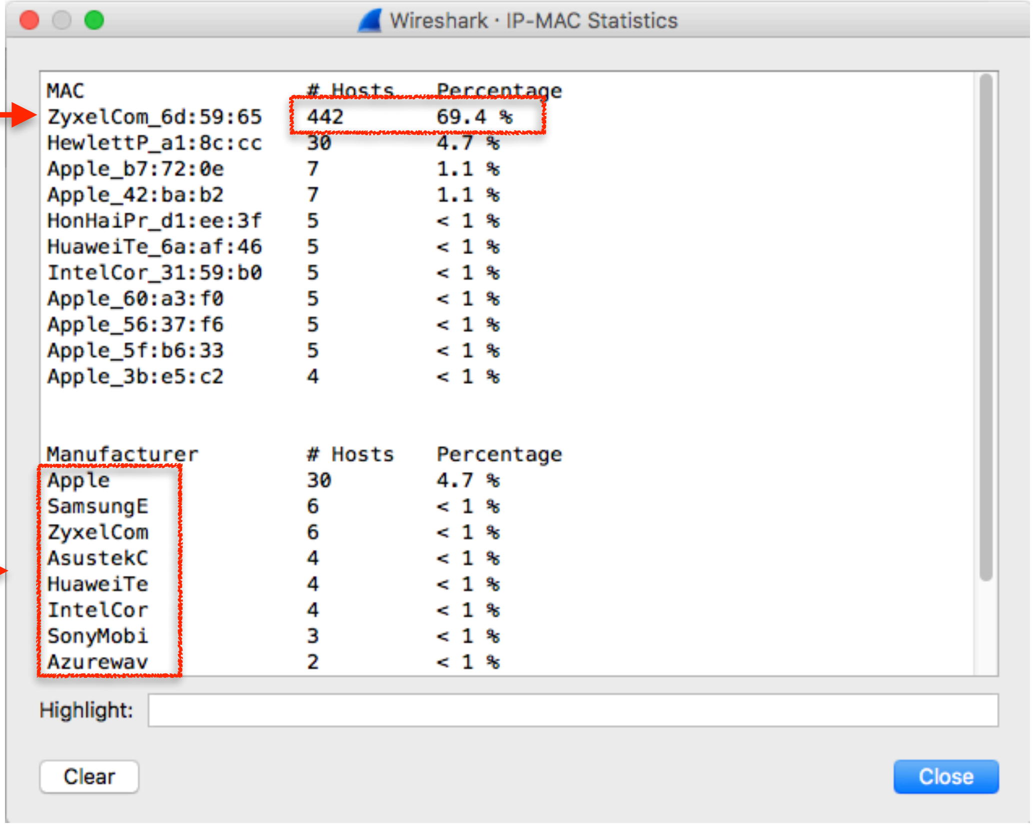
```
Top ARP Senders/Receivers

MAC Address        Tot Pkts  Pctg      ARP Breakdown
00:00:00:00:00:00  3442      43 %      [sent: 0][rcvd: 3442]
40:4a:03:6d:59:65  3214      40.2 %    [sent: 3039][rcvd: 175]
3c:15:c2:b7:72:0e  577       7.2 %     [sent: 484][rcvd: 93]
ff:ff:ff:ff:ff:ff  225       2.8 %     [sent: 0][rcvd: 225]
4c:9e:ff:f5:8d:13  111       1.4 %     [sent: 57][rcvd: 54]
38:b1:db:d1:ee:3f  56        < 1 %     [sent: 56][rcvd: 0]
8c:00:6d:42:ba:b2  51        < 1 %     [sent: 51][rcvd: 0]
a4:b8:05:60:a3:f0  48        < 1 %     [sent: 48][rcvd: 0]
04:db:56:3b:e5:c2  40        < 1 %     [sent: 40][rcvd: 0]
c4:07:2f:3c:e3:f7  33        < 1 %     [sent: 33][rcvd: 0]
e0:b9:a5:d3:2d:53  30        < 1 %     [sent: 22][rcvd: 8]
```

IP-Mac [1/2]

- The association IP-MAC is a metric that is often measured (does the tool arpwatch ring a bell?) as then it changes there must be a good reason (DHCP?).
- Counting IPs behind a MAC can also help us figuring out:
  - Whether it is a (hidden?) network gateway.
  - If a host runs VMs
  - Looking at MAC-IP cardinality we can guess if we’re analyzing a span port or a single-host traffic.

IP-Mac [2/2]

This is probably the gateway



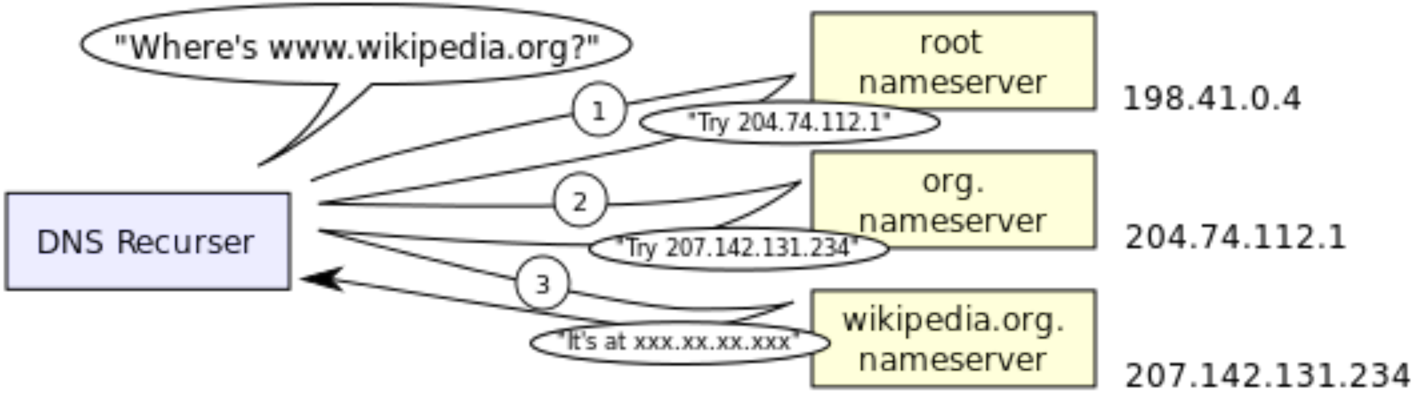
```
MAC                    # Hosts    Percentage
ZyxelCom_6d:59:65      442        69.4 %
HewlettP_a1:8c:cc      30         4.7 %
Apple_b7:72:0e         7          1.1 %
Apple_42:ba:b2         7          1.1 %
HonHaiPr_d1:ee:3f      5          < 1 %
HuaweiTe_6a:af:46      5          < 1 %
IntelCor_31:59:b0      5          < 1 %
Apple_60:a3:f0         5          < 1 %
Apple_56:37:f6         5          < 1 %
Apple_5f:b6:33         5          < 1 %
Apple_3b:e5:c2         4          < 1 %


Manufacturer           # Hosts    Percentage
Apple                  30         4.7 %
SamsungE               6          < 1 %
ZyxelCom               6          < 1 %
AsustekC               4          < 1 %
HuaweiTe               4          < 1 %
IntelCor               4          < 1 %
SonyMobi               3          < 1 %
Azurewav               2          < 1 %
```

Most users seem to use mobile devices

DNS [1/4]

- DNS is used to resolve IP addresses (i.e. find the number IP corresponding to a symbolic IP and vice-versa), and it is a protocol used by many other protocols (e.g. HTTP).
- Monitoring it, it's not just about resolving an address but also understanding what is happening on the network.
- In Wireshark there is an advanced dissector and Statistics -> DNS but it just focused on aggregated DNS protocol stats.

DNS [2/4]



https://en.wikipedia.org/wiki/Domain_Name_System

DNS [3/4]

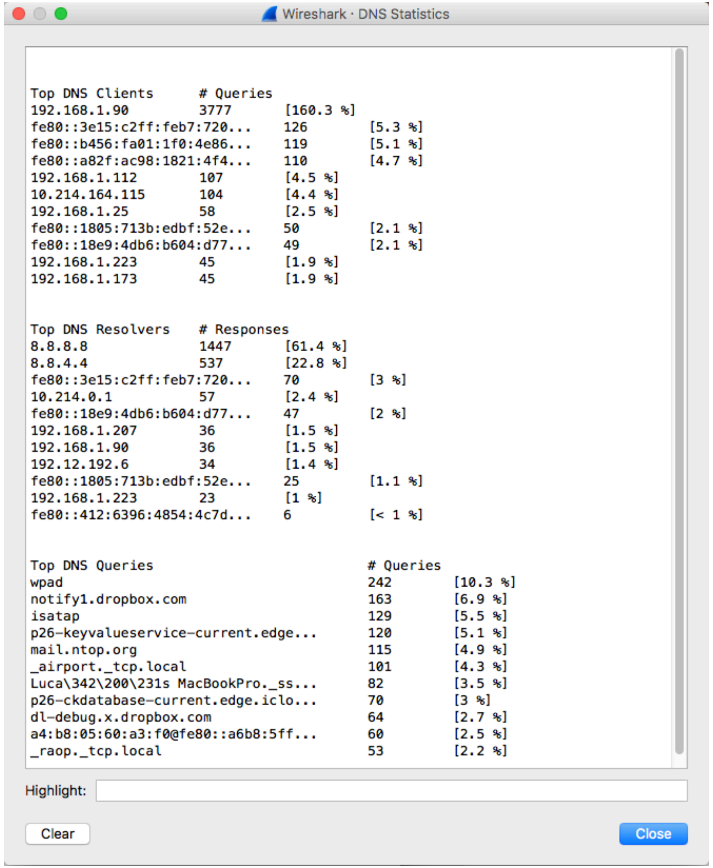
- What is missing is the ability to see what top addresses are resolved, what are the network resolvers, if there are hosts in the network that are not using the DNS resolver sysadmins expect.
- Also the ratio DNS requests vs valid/error responses can be a great indicator for spotting misconfigurations, scanners, ransomware (https://en.wikipedia.org/wiki/Domain_generation_algorithm).

DNS [4/4]

Top DNS Clients →

Top DNS Resolvers →

Top DNS Queries →



```
Top DNS Clients      # Queries
192.168.1.90         3777       [160.3 %]
fe80::3e15:c2ff:feb7:720...     126        [5.3 %]
fe80::b456:fa01:1f0:4e86...     119        [5.1 %]
fe80::a82f:ac98:1821:4f4...     110        [4.7 %]
192.168.1.112        107        [4.5 %]
10.214.164.115       104        [4.4 %]
192.168.1.25         58         [2.5 %]
fe80::1805:713b:edbf:52e...     50         [2.1 %]
fe80::18e9:4db6:b604:d77...     49         [2.1 %]
192.168.1.223        45         [1.9 %]
192.168.1.173        45         [1.9 %]


Top DNS Resolvers    # Responses
8.8.8.8              1447       [61.4 %]
8.8.4.4              537        [22.8 %]
fe80::3e15:c2ff:feb7:720...     70         [3 %]
10.214.0.1           57         [2.4 %]
fe80::18e9:4db6:b604:d77...     47         [2 %]
192.168.1.207        36         [1.5 %]
192.168.1.90         36         [1.5 %]
192.12.192.6         34         [1.4 %]
fe80::1805:713b:edbf:52e...     25         [1.1 %]
192.168.1.223        23         [1 %]
fe80::412:6396:4854:4c7d...     6          [< 1 %]


Top DNS Queries                         # Queries
wpad                                    242        [10.3 %]
notify1.dropbox.com                     163        [6.9 %]
isatap                                  129        [5.5 %]
p26-keyvalueservice-current.edge...     120        [5.1 %]
mail.ntop.org                           115        [4.9 %]
_airport._tcp.local                     101        [4.3 %]
Luca\342\200\231s MacBookPro._ss...     82         [3.5 %]
p26-ckdatabase-current.edge.iclo...     70         [3 %]
dl-debug.x.dropbox.com                  64         [2.7 %]
a4:b8:05:60:a3:f0@fe80::a6b8:5ff...     60         [2.5 %]
_raop._tcp.local                        53         [2.2 %]
```

TLS/SSL [1/4]

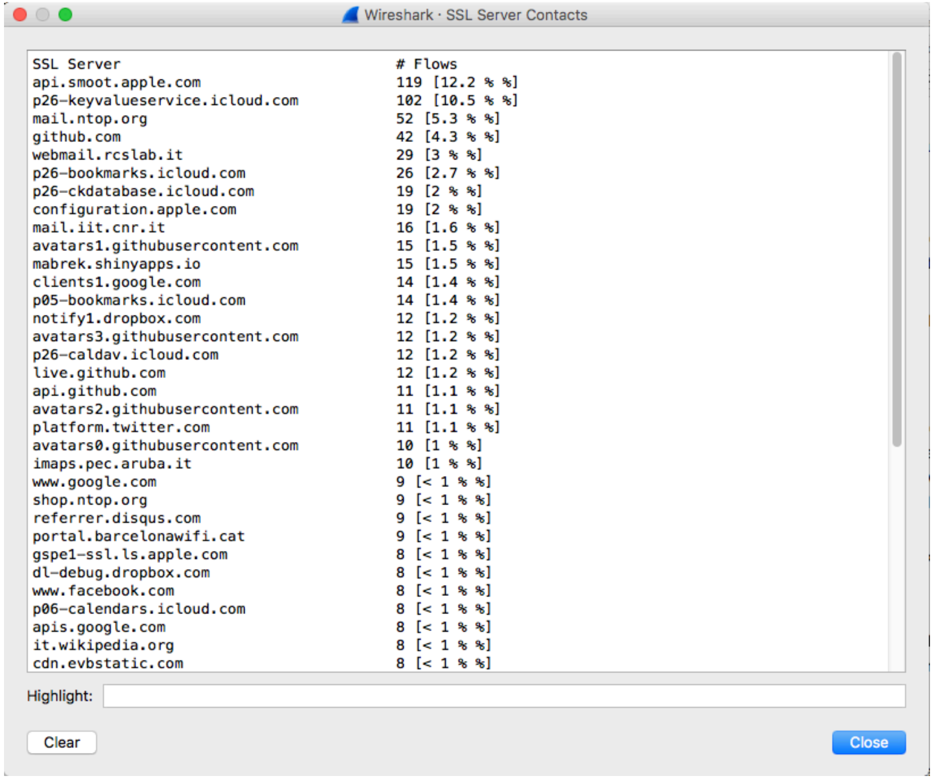
- Analysis of SSL certificates is becoming increasingly important for identifying application protocols. In particular (nDPI does it), for “protocols” that are alike as Google Search, Google Maps, Google Mail….
- Wireshark is able to decode SSL certificates (exchanged in clear before the encrypted part starts), so we just need to automate a feature that is already part of the tool.

TLS/SSL [2/4]

- At the beginning of the connection, in TLS the server is required to present a certificate.
- The client verifies that:
  - The subject of the certificate matches the hostname it is connecting to.
  - The certificate is used by a trusted certificate authority.
- Connection with invalid (or missing) server certificate are definitively suspicious.

TLS/SSL [3/4]

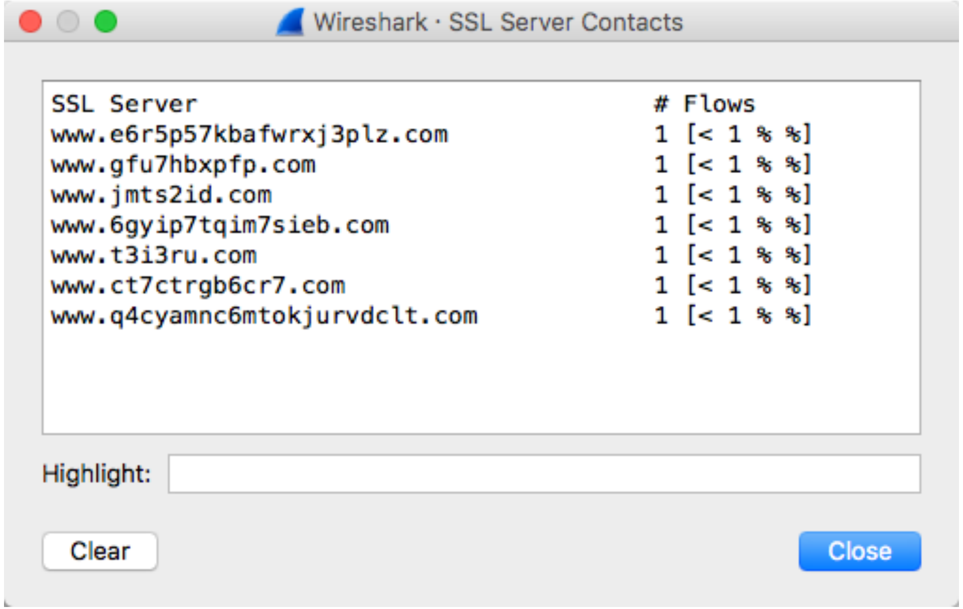
- Our Lua plugin is able to keep track of server certificates and count the number of SSL connections per certificate.
- However not all server names are alike…



```
SSL Server                                 # Flows
api.smoot.apple.com                        119 [12.2 % %]
p26-keyvalueservice.icloud.com             102 [10.5 % %]
mail.ntop.org                              52 [5.3 % %]
github.com                                 42 [4.3 % %]
webmail.rcslab.it                          29 [3 % %]
p26-bookmarks.icloud.com                   26 [2.7 % %]
p26-ckdatabase.icloud.com                  19 [2 % %]
configuration.apple.com                    19 [2 % %]
mail.iit.cnr.it                            16 [1.6 % %]
avatars1.githubusercontent.com             15 [1.5 % %]
mabrek.shinyapps.io                        15 [1.5 % %]
clients1.google.com                        14 [1.4 % %]
p05-bookmarks.icloud.com                   14 [1.4 % %]
notify1.dropbox.com                        12 [1.2 % %]
avatars3.githubusercontent.com             12 [1.2 % %]
p26-caldav.icloud.com                      12 [1.2 % %]
live.github.com                            12 [1.2 % %]
api.github.com                             11 [1.1 % %]
avatars2.githubusercontent.com             11 [1.1 % %]
platform.twitter.com                       11 [1.1 % %]
avatars0.githubusercontent.com             10 [1 % %]
imaps.pec.aruba.it                         10 [1 % %]
www.google.com                             9 [< 1 % %]
shop.ntop.org                              9 [< 1 % %]
referrer.disqus.com                        9 [< 1 % %]
portal.barcelonawifi.cat                   9 [< 1 % %]
gspe1-ssl.ls.apple.com                     8 [< 1 % %]
dl-debug.dropbox.com                       8 [< 1 % %]
www.facebook.com                           8 [< 1 % %]
p06-calendars.icloud.com                   8 [< 1 % %]
apis.google.com                            8 [< 1 % %]
it.wikipedia.org                           8 [< 1 % %]
cdn.evbstatic.com                          8 [< 1 % %]
```

TLS/SSL [4/4]

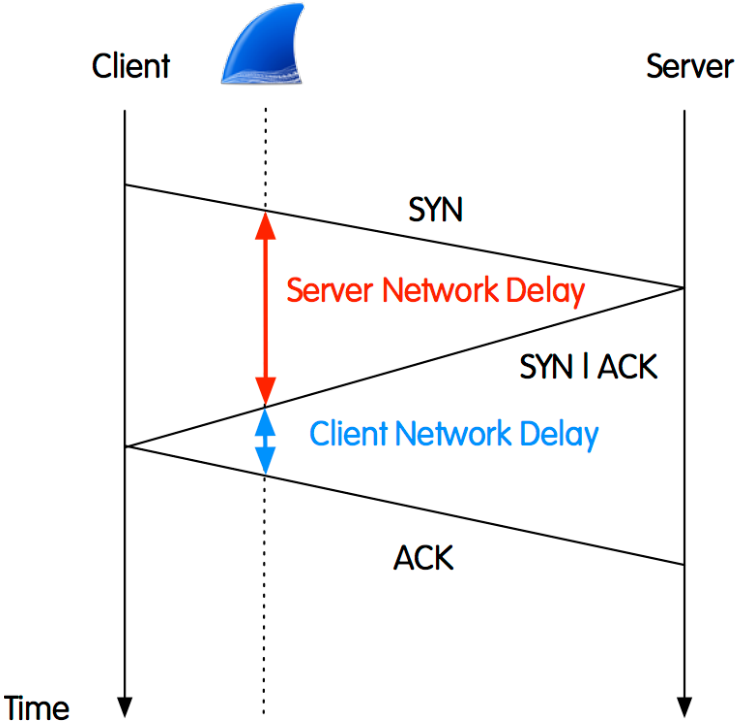
- Earlier in this walk we covered DGA. They are not used just by ransomware(s). Below you can see what (fake) server names Tor connections use.
- When you see these name, it means it's time to investigate more in details what our users are doing.



```
SSL Server                                # Flows
www.e6r5p57kbafwrxj3plz.com               1 [< 1 % %]
www.gfu7hbxpfp.com                        1 [< 1 % %]
www.jmts2id.com                           1 [< 1 % %]
www.6gyip7tqim7sieb.com                   1 [< 1 % %]
www.t3i3ru.com                            1 [< 1 % %]
www.ct7ctrgb6cr7.com                      1 [< 1 % %]
www.q4cyamnc6mtokjurvdclt.com             1 [< 1 % %]
```

Network/Application Latency [1/4]

- Network Latency (or delay): amount of time (ms) it takes a packet from source to destination (one-way). It cannot be measured from a single point of observation.
- Very important for interactive applications (e.g. online games).
- Round-trip latency (source -> destination -> source) is less accurate but more popular as it can be measured from a single point.



Network/Application Latency [2/4]

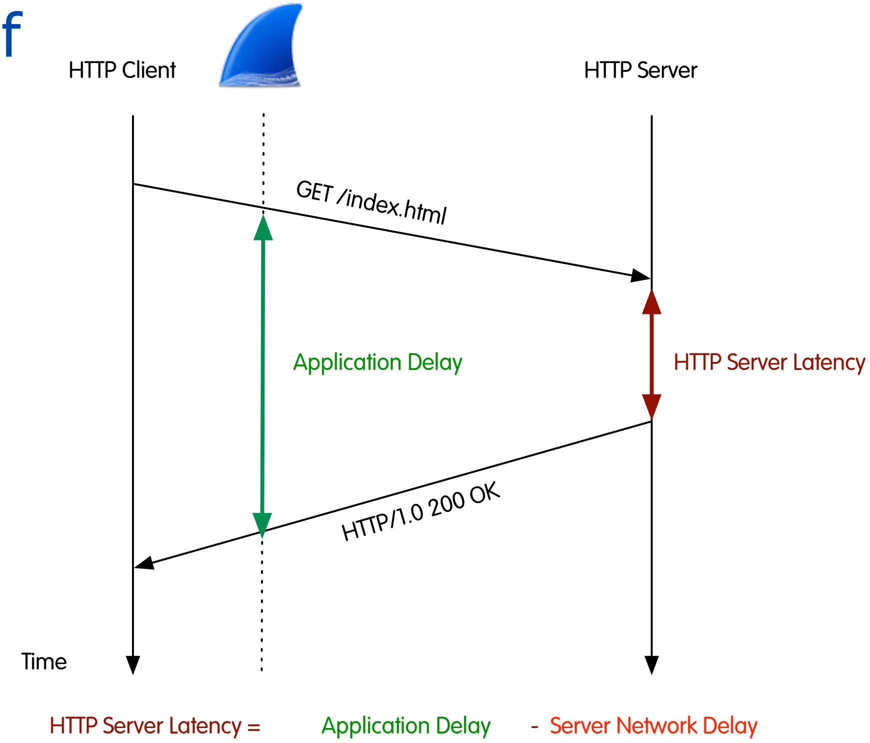
- Network latency is affected by:
  - Media type (fibre vs. satellite communications).

  - Memory Buffers
    - Operating system buffering (sockets, queues)
    - Network devices buffering (I/O ports).

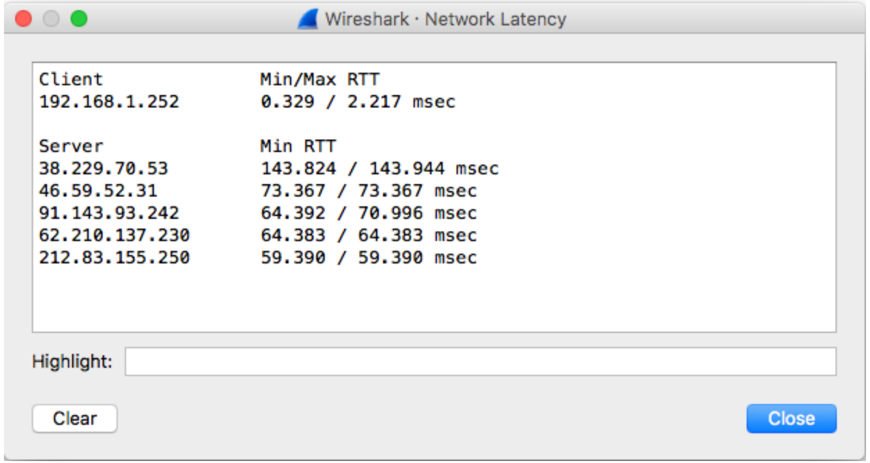
- The more network elements a packet has to traverse, the more is the latency it can accumulate.

Network/**Application** Latency [3/4]

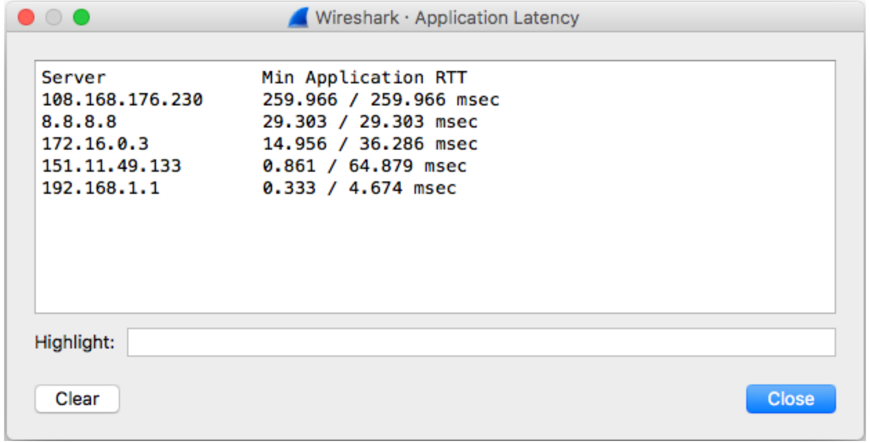
- Application Latency: companion of network latency, when measured at application level instead of network level.
- It computes the delay added by application processing to the packet journey.
- It basically measures the time taken by the application to serve the request (e.g. SQL query execution time).



Network/Application Latency [4/4]



```
Client              Min/Max RTT
192.168.1.252       0.329 / 2.217 msec

Server              Min RTT
38.229.70.53        143.824 / 143.944 msec
46.59.52.31         73.367 / 73.367 msec
91.143.93.242       64.392 / 70.996 msec
62.210.137.230      64.383 / 64.383 msec
212.83.155.250      59.390 / 59.390 msec
```



```
Server              Min Application RTT
108.168.176.230     259.966 / 259.966 msec
8.8.8.8             29.303 / 29.303 msec
172.16.0.3          14.956 / 36.286 msec
151.11.49.133       0.861 / 64.879 msec
192.168.1.1         0.333 / 4.674 msec
```

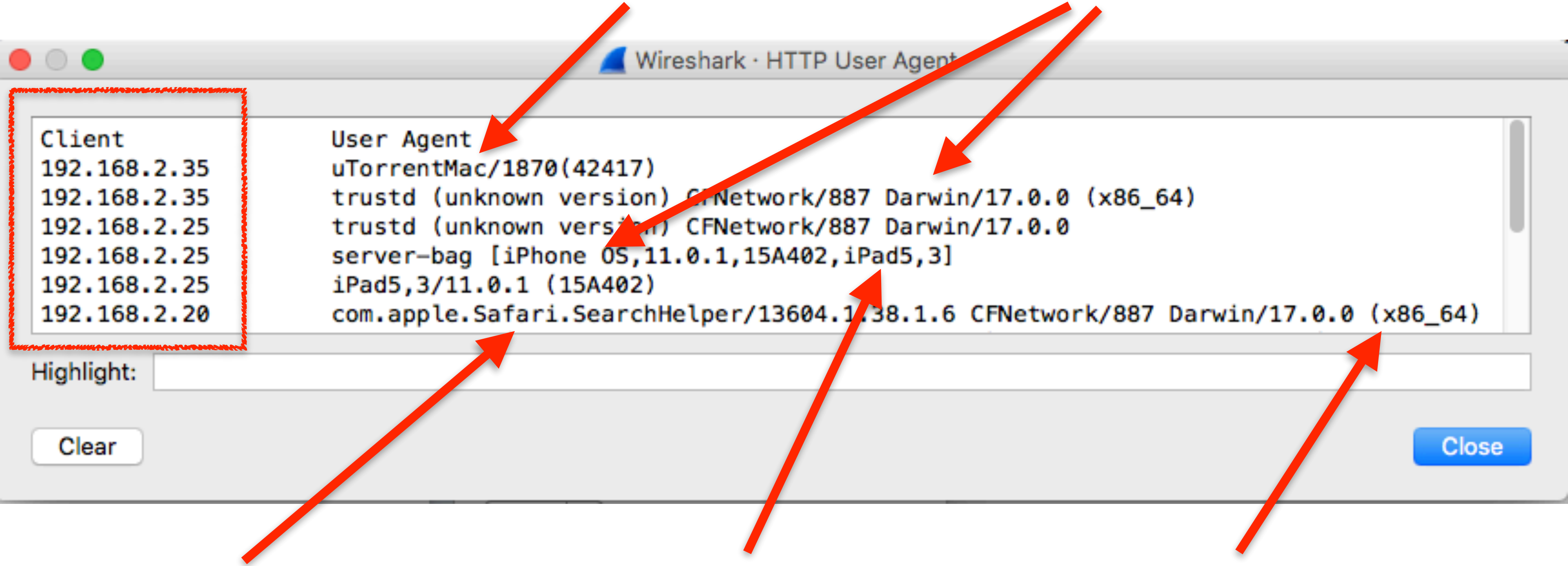
Passive HTTP Discovery [1/3]

- The HTTP user agent field can disclose a lot of information about the device issuing requests, including its:
  - Operating System Version
  - Device Type and Model
  - Network Services (and thus running applications) accessed by the device
  - Browser type/version (and thus vulnerabilities/CVEs)
  - NAT (> 1 Operating System/MAC -> NAT is in place)

Passive HTTP Discovery [2/3]

L7 Protocol + Application

Operating System (+ Version)



```
Client              User Agent
192.168.2.35        uTorrentMac/1870(42417)
192.168.2.35        trustd (unknown version) CFNetwork/887 Darwin/17.0.0 (x86_64)
192.168.2.25        trustd (unknown version) CFNetwork/887 Darwin/17.0.0
192.168.2.25        server-bag [iPhone OS,11.0.1,15A402,iPad5,3]
192.168.2.25        iPad5,3/11.0.1 (15A402)
192.168.2.20        com.apple.Safari.SearchHelper/13604.1.38.1.6 CFNetwork/887 Darwin/17.0.0 (x86_64)
```

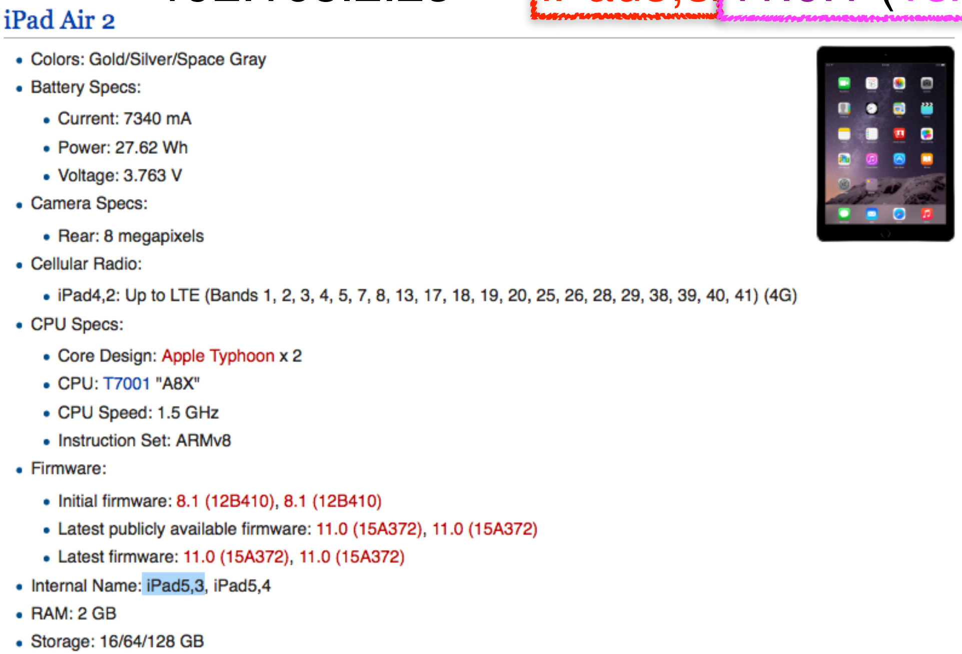
Client Application

Device Model (+ Mobile User)

Architecture

Passive HTTP Discovery [3/3]

```
192.168.2.25    server-bag [iPhone OS,11.0.1,15A402,iPad5,3]
192.168.2.25    iPad5,3/11.0.1 (15A402)
```



iPad Air 2

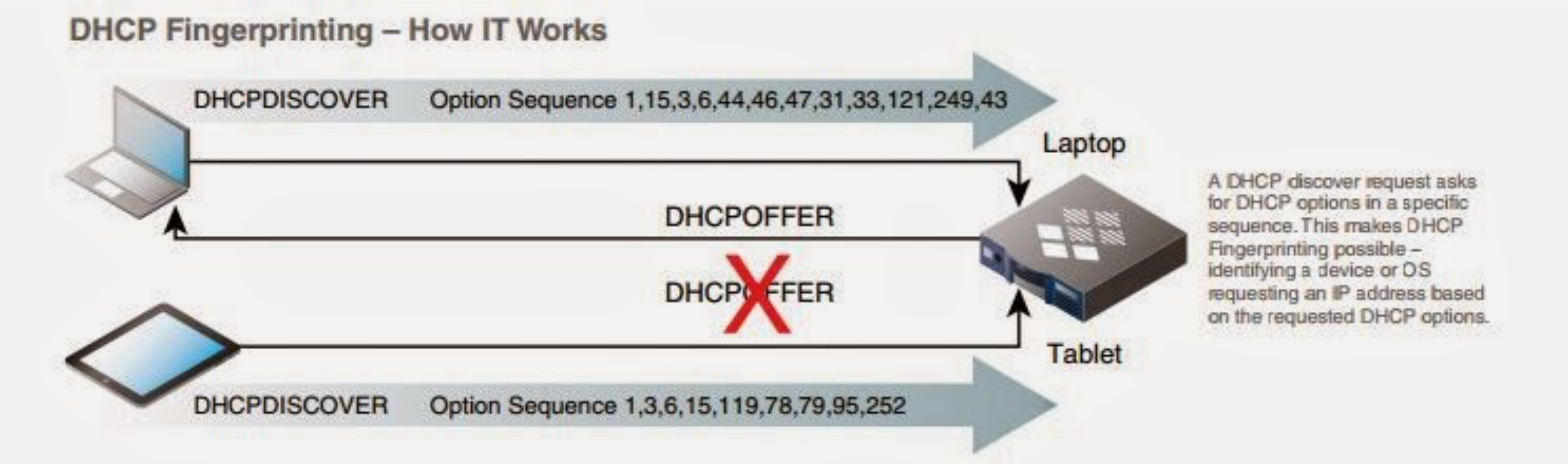
- Colors: Gold/Silver/Space Gray
- Battery Specs:
  - Current: 7340 mA
  - Power: 27.62 Wh
  - Voltage: 3.763 V
- Camera Specs:
  - Rear: 8 megapixels
- Cellular Radio:
  - iPad4,2: Up to LTE (Bands 1, 2, 3, 4, 5, 7, 8, 13, 17, 18, 19, 20, 25, 26, 28, 29, 38, 39, 40, 41) (4G)
- CPU Specs:
  - Core Design: Apple Typhoon x 2
  - CPU: T7001 "A8X"
  - CPU Speed: 1.5 GHz
  - Instruction Set: ARMv8
- Firmware:
  - Initial firmware: 8.1 (12B410), 8.1 (12B410)
  - Latest publicly available firmware: 11.0 (15A372), 11.0 (15A372)
  - Latest firmware: 11.0 (15A372), 11.0 (15A372)
- Internal Name: iPad5,3, iPad5,4
- RAM: 2 GB
- Storage: 16/64/128 GB

https://www.theiphonewiki.com/wiki/List_of_iPads

DHCP Fingerprinting [1/4]

- As TCP state-based detection is becoming outdated (p0f, ettercap…) new passive fingerprinting tools are emerging.
- One of the most popular is DHCP fingerprinting that allows
  - Devices to be identified quite reliably
  - As DHCP is one of the first packets to be observed on a network, it can be used by devices such as access points to “block at the edge” devices that are marked as unsafe and potentially dangerous

DHCP Fingerprinting [2/4]



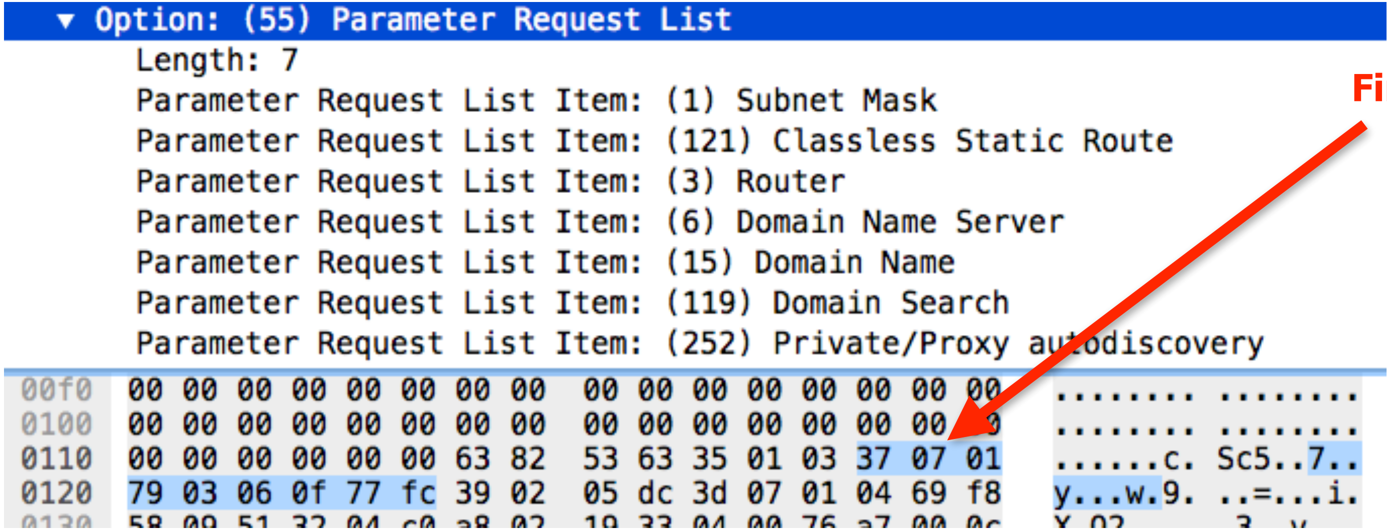
DHCP Fingerprinting – How IT Works

DHCPDISCOVER
Option Sequence 1,15,3,6,44,46,47,31,33,121,249,43
Laptop
DHCPOFFER
DHCPOFFER
Tablet
DHCPDISCOVER
Option Sequence 1,3,6,15,119,78,79,95,252

A DHCP discover request asks for DHCP options in a specific sequence. This makes DHCP Fingerprinting possible – identifying a device or OS requesting an IP address based on the requested DHCP options.

DHCP Fingerprinting [3/4]

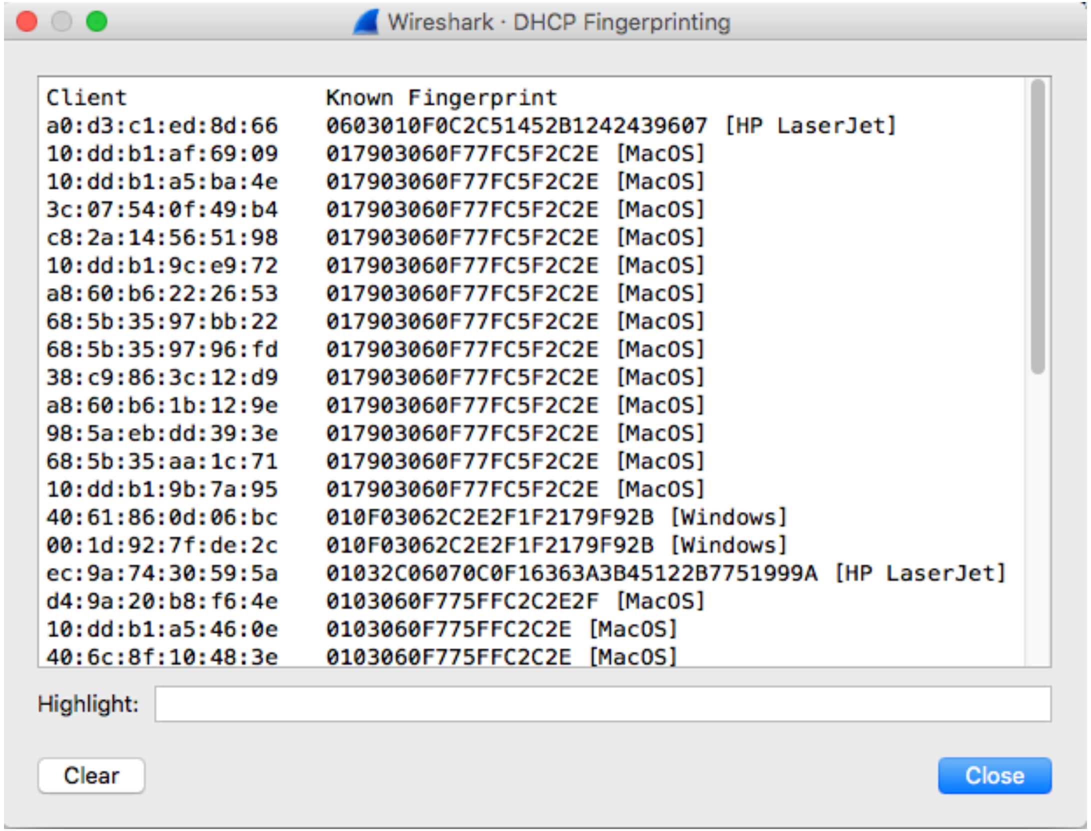
- DHCP fingerprinting uses option 55 of DHCP request to create a fingerprint then searched on a database (e.g. https://fingerbank.org)



```
▼ Option: (55) Parameter Request List
      Length: 7
      Parameter Request List Item: (1) Subnet Mask
      Parameter Request List Item: (121) Classless Static Route
      Parameter Request List Item: (3) Router
      Parameter Request List Item: (6) Domain Name Server
      Parameter Request List Item: (15) Domain Name
      Parameter Request List Item: (119) Domain Search
      Parameter Request List Item: (252) Private/Proxy autodiscovery
00f0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0100  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0110  00 00 00 00 00 00 63 82  53 63 35 01 03 37 07 01   ......c. Sc5..7..
0120  79 03 06 0f 77 fc 39 02  05 dc 3d 07 01 04 69 f8   y...w.9. ..=...i.
0130  58 09 51 32 04 c0 a8 02  19 33 04 00 76 a7 00 0c   X.Q2.... .3..v...
```

Fingerprint

DHCP Fingerprinting [4/4]



```
Client              Known Fingerprint
a0:d3:c1:ed:8d:66   0603010F0C2C51452B1242439607 [HP LaserJet]
10:dd:b1:af:69:09   017903060F77FC5F2C2E [MacOS]
10:dd:b1:a5:ba:4e   017903060F77FC5F2C2E [MacOS]
3c:07:54:0f:49:b4   017903060F77FC5F2C2E [MacOS]
c8:2a:14:56:51:98   017903060F77FC5F2C2E [MacOS]
10:dd:b1:9c:e9:72   017903060F77FC5F2C2E [MacOS]
a8:60:b6:22:26:53   017903060F77FC5F2C2E [MacOS]
68:5b:35:97:bb:22   017903060F77FC5F2C2E [MacOS]
68:5b:35:97:96:fd   017903060F77FC5F2C2E [MacOS]
38:c9:86:3c:12:d9   017903060F77FC5F2C2E [MacOS]
a8:60:b6:1b:12:9e   017903060F77FC5F2C2E [MacOS]
98:5a:eb:dd:39:3e   017903060F77FC5F2C2E [MacOS]
68:5b:35:aa:1c:71   017903060F77FC5F2C2E [MacOS]
10:dd:b1:9b:7a:95   017903060F77FC5F2C2E [MacOS]
40:61:86:0d:06:bc   010F03062C2E2F1F2179F92B [Windows]
00:1d:92:7f:de:2c   010F03062C2E2F1F2179F92B [Windows]
ec:9a:74:30:59:5a   01032C06070C0F16363A3B45122B7751999A [HP LaserJet]
d4:9a:20:b8:f6:4e   0103060F775FFC2C2E2F [MacOS]
10:dd:b1:a5:46:0e   0103060F775FFC2C2E [MacOS]
40:6c:8f:10:48:3e   0103060F775FFC2C2E [MacOS]
```

Final Remarks

- ntop is contributing to Wireshark since a decade.
- We are focusing on network traffic monitoring, and this talk has demonstrated that with little effort Wireshark can be extended well beyond its native packet-oriented metrics.

- All the code presented is open-source and it is available at https://github.com/ntop/wireshark-ntop

Enjoy!