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.
Introducing ntop [2/2]

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

https://code.wireshark.org/review/#/c/21825/
• 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).
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.
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 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]

Long term packet capture options
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-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.
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.
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.

<table>
<thead>
<tr>
<th>No.</th>
<th>Time</th>
<th>Source</th>
<th>Destination</th>
<th>Protocol</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2016-08-17</td>
<td>10.100.25.14</td>
<td>192.168.0.203</td>
<td>CFLOW</td>
<td>1458</td>
</tr>
<tr>
<td>2</td>
<td>2016-08-17</td>
<td>10.104.16.118</td>
<td>192.168.0.203</td>
<td>CFLOW</td>
<td>786</td>
</tr>
<tr>
<td>3</td>
<td>2016-08-17</td>
<td>10.100.25.14</td>
<td>192.168.0.203</td>
<td>CFLOW</td>
<td>1458</td>
</tr>
<tr>
<td>4</td>
<td>2016-08-17</td>
<td>10.100.25.14</td>
<td>192.168.0.203</td>
<td>CFLOW</td>
<td>1438</td>
</tr>
<tr>
<td>5</td>
<td>2016-08-17</td>
<td>10.100.25.14</td>
<td>192.168.0.203</td>
<td>CFLOW</td>
<td>1458</td>
</tr>
</tbody>
</table>
Users can instruct Wireshark to decode a selected traffic flow using a specific protocol decoder. 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.
• 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.
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...
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 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.
• 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
nDPI Extcap Module Code [1/2]

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

Extcap Module

- ndpiReader.c
- ntopUtil.c
- ntopUtil.h
- protos.h
- uthash.h

Lua Script

- ndpi.lua
- README.md
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
How to rule the world... by looking at packets!

nDPI Extcap Installation

Extcap Module

Lua Script
nDPI Extcap Module: Traffic Classification [1/4]

- **Live Capture**
- **Capture from Pcap**
- **nDPI Filter**

Click here first
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: 19680924005b0007753534c2e46616365626f66b00000000
  Frame check sequence: 0xb9c78d3b [correct].
[FCS Status: Good]
Internet Protocol Version 4, Src: 192.168.43.18, Dst: 66.220.156.68
nDPI Protocol
  nDPI Network Protocol: 91
  nDPI Application Protocol: 119
  nDPI Protocol Name: SSL.Facebook

Added via Extcap
The Extcap module

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

```c
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];
};
```

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

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Size</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unknown</td>
<td>697.92 KB</td>
<td>43.5 %</td>
</tr>
<tr>
<td>LotusNotes</td>
<td>340.92 KB</td>
<td>21.2 %</td>
</tr>
<tr>
<td>RTP</td>
<td>184.85 KB</td>
<td>11.5 %</td>
</tr>
<tr>
<td>HTTP.Facebook</td>
<td>103.4 KB</td>
<td>6.4 %</td>
</tr>
<tr>
<td>NFS</td>
<td>99.16 KB</td>
<td>6.2 %</td>
</tr>
<tr>
<td>SSL.Office365</td>
<td>83.24 KB</td>
<td>5.2 %</td>
</tr>
<tr>
<td>HTTP</td>
<td>38.98 KB</td>
<td>2.4 %</td>
</tr>
</tbody>
</table>

Top nDPI Flows

<table>
<thead>
<tr>
<th>Source</th>
<th>Destination</th>
<th>Size</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>192.168.17.101 / 40</td>
<td>192.168.17.40</td>
<td>407.37 KB</td>
<td>25.4 %</td>
</tr>
<tr>
<td>192.168.18.104 / 106</td>
<td>192.168.18.106</td>
<td>332.29 KB</td>
<td>20.7 %</td>
</tr>
<tr>
<td>192.168.13.51 / 100</td>
<td>192.168.13.100</td>
<td>114.32 KB</td>
<td>7.1 %</td>
</tr>
<tr>
<td>192.168.12.131 / 32</td>
<td>66.220.146.32</td>
<td>79.52 KB</td>
<td>5 %</td>
</tr>
<tr>
<td>65.55.171.156 / 101</td>
<td>192.168.16.101</td>
<td>59.25 KB</td>
<td>3.7 %</td>
</tr>
</tbody>
</table>
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.
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.

![Screenshot of nDPI Extcap startup window with protocol selection dropdown open and AMQP protocol highlighted.](image)
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.
• 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).
• Wireshark is able to dissect VLAN-tagged packets both inside the packet header or in other packet types (e.g. CDP)
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 [2/3]
• 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.
• 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.
• Is it normal that one host has sent most ARP requests? Why?
• What role has this host on our network?
• Is anybody doing an ARP scan perhaps?
• How much is the ARP traffic when compared to overall network traffic.
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.
This is probably the gateway

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
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](https://en.wikipedia.org/wiki/Domain_generation_algorithm)).
# How to rule the world... by looking at packets!

## DNS Statistics

### Top DNS Clients

<table>
<thead>
<tr>
<th>IP Address</th>
<th>Queries</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>192.168.1.90</td>
<td>3777</td>
<td>168.3%</td>
</tr>
<tr>
<td>fe80::6e15:cfff:fe80:726...</td>
<td>126</td>
<td>5.3%</td>
</tr>
<tr>
<td>fe80::6e45:fe80:11f8:6d08...</td>
<td>110</td>
<td>5.1%</td>
</tr>
<tr>
<td>fe80::f82f:6c08:3823:4f4...</td>
<td>110</td>
<td>4.7%</td>
</tr>
<tr>
<td>192.168.1.112</td>
<td>187</td>
<td>4.5%</td>
</tr>
<tr>
<td>10.214.164.115</td>
<td>184</td>
<td>4.4%</td>
</tr>
<tr>
<td>192.168.1.25</td>
<td>58</td>
<td>2.5%</td>
</tr>
<tr>
<td>fe80::7e3b:edbf:52e...</td>
<td>50</td>
<td>2.1%</td>
</tr>
<tr>
<td>fe80::5e6d:6b08:d77...</td>
<td>49</td>
<td>2.1%</td>
</tr>
<tr>
<td>192.168.1.223</td>
<td>45</td>
<td>1.9%</td>
</tr>
<tr>
<td>192.168.1.173</td>
<td>45</td>
<td>1.9%</td>
</tr>
</tbody>
</table>

### Top DNS Resolvers

<table>
<thead>
<tr>
<th>IP Address</th>
<th>Responses</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.8.8.8</td>
<td>1447</td>
<td>61.4%</td>
</tr>
<tr>
<td>8.8.4.4</td>
<td>527</td>
<td>22.8%</td>
</tr>
<tr>
<td>fe80::2e12:cf2ff:fe80:726...</td>
<td>70</td>
<td>3%</td>
</tr>
<tr>
<td>10.214.0.1</td>
<td>27</td>
<td>2.4%</td>
</tr>
<tr>
<td>fe80::1e8d:4dbd:6804:d77...</td>
<td>47</td>
<td>2%</td>
</tr>
<tr>
<td>192.168.1.207</td>
<td>26</td>
<td>1.5%</td>
</tr>
<tr>
<td>192.168.1.98</td>
<td>36</td>
<td>1.5%</td>
</tr>
<tr>
<td>192.12.192.6</td>
<td>34</td>
<td>1.4%</td>
</tr>
<tr>
<td>fe80::1e8d:713b:edbf:52e...</td>
<td>25</td>
<td>1.1%</td>
</tr>
<tr>
<td>192.168.1.223</td>
<td>23</td>
<td>1%</td>
</tr>
<tr>
<td>fe80::412:6396:4654:4c7d...</td>
<td>6</td>
<td>&lt;1%</td>
</tr>
</tbody>
</table>

### Top DNS Queries

<table>
<thead>
<tr>
<th>Query</th>
<th>Queries</th>
</tr>
</thead>
<tbody>
<tr>
<td>wpad</td>
<td>342</td>
</tr>
<tr>
<td>notify1.dropbox.com</td>
<td>163</td>
</tr>
<tr>
<td>lsstat</td>
<td>129</td>
</tr>
<tr>
<td>p26-keyvalueservice-current.edge...</td>
<td>129</td>
</tr>
<tr>
<td>mail.ntop.org</td>
<td>115</td>
</tr>
<tr>
<td>_airport._tcp.local</td>
<td>101</td>
</tr>
<tr>
<td>local//342/200//231s MacbookPro...</td>
<td>82</td>
</tr>
<tr>
<td>p26-c-dashboard-current.edge...</td>
<td>70</td>
</tr>
<tr>
<td>sl-debug.dropbox.com</td>
<td>64</td>
</tr>
<tr>
<td>64.81.61.68:61f1f8ff...</td>
<td>60</td>
</tr>
<tr>
<td>_resp._tcp.local</td>
<td>53</td>
</tr>
</tbody>
</table>
• 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.
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.
• 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...
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.
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 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.
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

\[ \text{HTTP Server Latency} = \text{Application Delay} - \text{Server Network Delay} \]
Network/Application Latency [4/4]

How to rule the world... by looking at packets!
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]

Client Application

Operating System (+ Version)

L7 Protocol + Application

Device Model (+ Mobile User)

Architecture

Client
192.168.2.35
192.168.2.35
192.168.2.25
192.168.2.25
192.168.2.25
192.168.2.20

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

192.168.2.25  iPad5,3/11.0.1 (15A402)

https://www.theiphonewiki.com/wiki/List_of_iPads
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 [3/4]

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

![DHCP Fingerprinting Example](image)
### DHCP Fingerprinting [4/4]

**Client** | **Known Fingerprint**
---|---
a0:d3:c1:ed:8d:66 | 0603010F0C2C51452B1242439607 [HP LaserJet]
10:db:b1:af:89:09 | 017903060F77FC5F2C2E [MacOS]
10:db:b1:a5:ba:4e | 017903060F77FC5F2C2E [MacOS]
3c:07:54:0f:49:b4 | 017903060F77FC5F2C2E [MacOS]
c8:2a:14:56:51:98 | 017903060F77FC5F2C2E [MacOS]
10:db:b1:9c:ea:97 | 017903060F77FC5F2C2E [MacOS]
a8:60:b6:22:26:51 | 017903060F77FC5F2C2E [MacOS]
60:5b:35:97:bb:22 | 017903060F77FC5F2C2E [MacOS]
60:5b:35:97:96:fd | 017903060F77FC5F2C2E [MacOS]
38:cb:86:3c:12:dc | 017903060F77FC5F2C2E [MacOS]
a8:60:b6:1b:12:9e | 017903060F77FC5F2C2E [MacOS]
98:5a:eb:dd:39:3e | 017903060F77FC5F2C2E [MacOS]
60:5b:35:aa:1c:71 | 017903060F77FC5F2C2E [MacOS]
10:db:b1:9b:7a:95 | 017903060F77FC5F2C2E [MacOS]
00:1d:92:7f:de:2c | 010F03062C2E2F12179F92B [Windows]
d4:9a:20:b5:6d:4e | 0103060F775FC2CE2F [MacOS]
10:db:b1:a5:46:99 | 0103060F775FC2CE2 [MacOS]
40:6c:8f:10:48:3e | 0103060F775FC2CE2 [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!