nDPI Webinar

Introduction to nDPI and Roadmap Outlook

Luca Deri <deri@ntop.org>

ntop

Monitoring Requirements

- Internet Service Providers
  - Prevent the network from collapsing (mostly DDoS).
  - Visibility of the main network activities in order to understand traffic flows (routing/AS-level, not host).
  - Device monitoring (interface drops, state changes).
- Service/Cloud/Hosting Providers
  - Monitor core services (e.g. DNS, email).
  - Detect severe source of troubles (e.g. heavy spammers) in order to avoid decreasing the overall network reputation.

In Essence... Visibility is the Problem

- In order to make decisions, it is compulsory to understand how our traffic moves in the network.
- Two scenarios:
  - End users: they "own" their traffic so they can fully inspect packets.
  - Service-Cloud Providers: the traffic belongs to their customers, but they can implement "lightweight" packet inspection in order to keep the network healthy.
  - IXPs: they transit customer traffic, not their traffic. They mostly use sFlow/NetFlow. No packets here.

What is DPI ?

- DPI (Deep Packet Inspection) enables the inspection of packet payload in order to extract metadata and characterise traffic.
- Commercial DPI libraries are often quite expensive in price, and do not cope with high-speed (> 10 Gbit).
- Network administrators are used (often due to limitations of leading hardware manufacturers) to monitor sampled data with not DPI information.
- In 2025 we need full visibility with DPI and ETA (Encrypted Traffic Analysis).

Welcome to nDPI [1/2]

- C-based open-source library providing:
  - Deep packet inspection engine for network visibility: protocol classification (450+), metadata extraction, flow risks computation
    - Basic blocks for a cyber-security application
    - Flow risks: an indication that in the flow there is something unusual/ dangerous to pay attention to
      - ~60 different flow risks: self-signed certificate, possible SQL/RCE injection, suspicious DGA domain, invalid character in SNI...
  - Algorithms for data analysis: data forecasting, anomaly detection, clustering and similarity evaluation, (sub-)string searching and IP matching, probabilistic data structures,...
- Available on GitHub, LGPL v3

Welcome to nDPI [2/2]

- Each protocol is identified as <major>.<minor> protocol. Example:
  - DNS.Facebook
  - QUIC.YouTube and QUIC.YouTubeUpload
- Caveat: Zoom or WhatsApp are application protocols in the nDPI world but not for IETF.
- The first question people ask when they have to evaluate a DPI toolkit is: how many protocol do you support? This is not the right question.

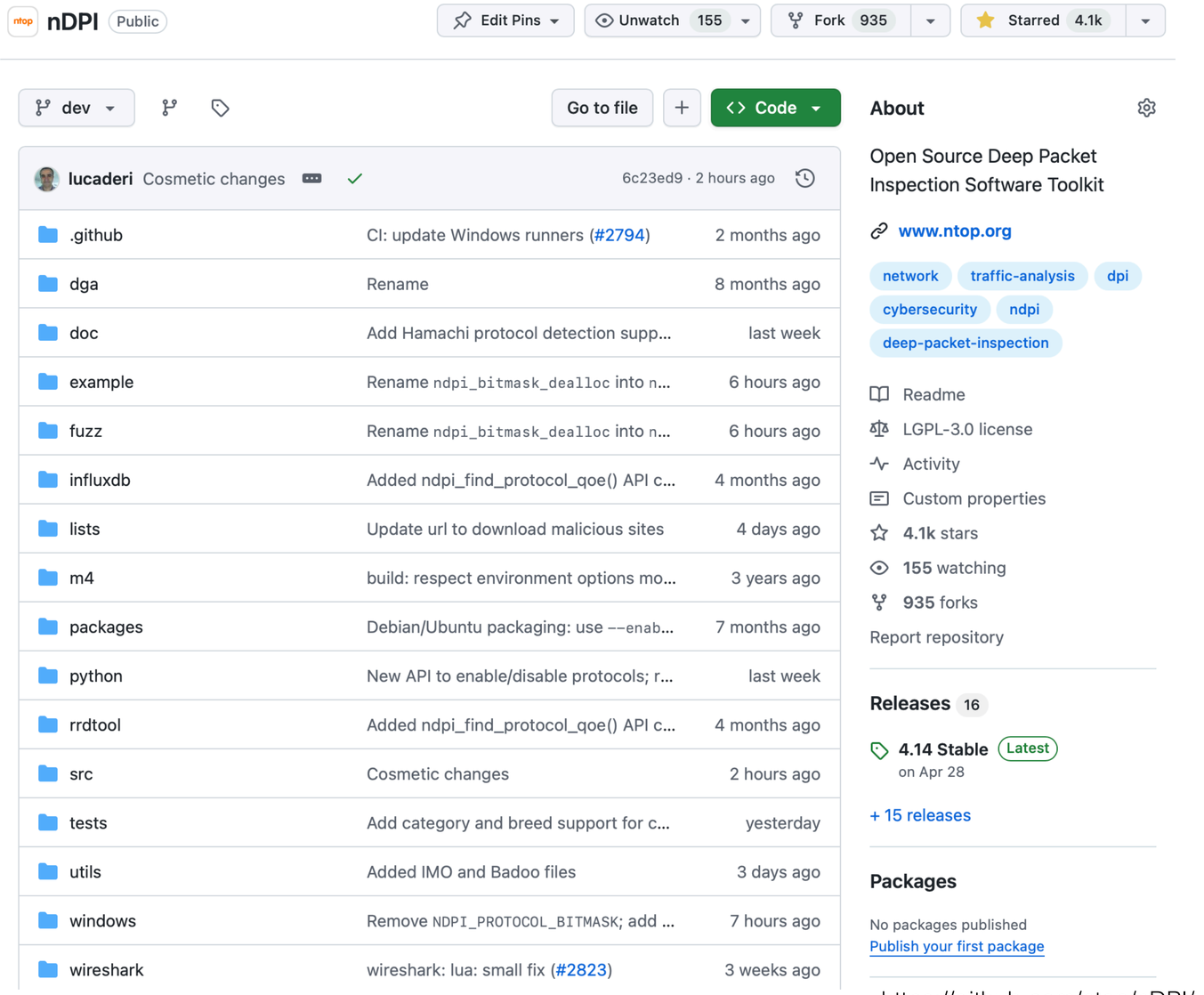
nDPI Code Layout

Test App Testing

Main Code Testing

Wireshark

ntop



https://github.com/ntop/nDPI/

How nDPI Dissection Works

- nDPI implements protocol dissectors, divided per L4 protocol (TCP and UDP).
- Packets are classified in flows (similar to NetFlow).
- Depending on the L4 protocol, dissectors are applied to the flow:
  - At every packet, only potential protocol dissectors are applied.
  - As soon as a protocol dissector fails, it is not applied to future flow packets.
  - As soon as there is a match or when no dissector matched (Unknown), the process is over.
  - In the worst case, after too many iterations, Unknown protocol is reported.

nDPI Dissectors

```
#include "ndpi_protocol_ids.h"

#define NDPI_CURRENT_PROTO NDPI_PROTOCOL_ETHERSIO

#include "ndpi_api.h"
#include "ndpi_private.h"


static void ndpi_search_ethersio(struct ndpi_detection_module_struct *ndpi_struct,
                                 struct ndpi_flow_struct *flow)
{
  struct ndpi_packet_struct const * const packet = &ndpi_struct->packet;

  NDPI_LOG_DBG(ndpi_struct, "search Ether-SIO\n");

  if (packet->payload_packet_len >= 20) {
    if ((memcmp(packet->payload, "ESIO", 4) == 0) &&
        (packet->payload[4] == 0) && (packet->payload[5] <= 0x2) &&
        (packet->payload[6] == 0)) {
      ndpi_set_detected_protocol(ndpi_struct, flow, NDPI_PROTOCOL_ETHERSIO,
                                 NDPI_PROTOCOL_UNKNOWN, NDPI_CONFIDENCE_DPI);
      return;
    }
  }

  NDPI_EXCLUDE_DISSECTOR(ndpi_struct, flow);
}

void init_ethersio_dissector(struct ndpi_detection_module_struct *ndpi_struct)
{
  register_dissector("EtherSIO", ndpi_struct,
                     ndpi_search_ethersio,
                     NDPI_SELECTION_BITMASK_PROTOCOL_V4_V6_UDP_WITH_PAYLOAD,
                     1, NDPI_PROTOCOL_ETHERSIO);
}
```

Protocol Detected

Not This Protocol

ntop

https://github.com/ntop/nDPI/

Application Code Simplified

```
 struct ndpi_detection_module_struct *ndpi_s = ndpi_init_detection_module(NULL);

....

 proto_id = ndpi_detection_process_packet(ndpi_s,
                          ndpiFlow, ip_packet,
                          ip_len, packet_time, NULL);

 if(!ndpi_is_protocol_detected(proto_id) && hasDissectedTooManyPackets()) {
      updateProtocol(ndpi_detection_giveup(ndpi_s, ndpiFlow,
                                           &proto_guessed));

 }

....

ndpi_exit_detection_module(ndpi_struct);
```

nDPI Dissection Overhead

- nDPI dissection requires flows to be inspected since the beginning, where relevant information can be detected.
- Corollary: the longer is the flow, the less DPI is computationally expensive.
- Typical figures (ndpiReader):

```
DPI Packets (TCP):     301             (5.38 pkts/flow)
DPI Packets (UDP):     182             (1.88 pkts/flow)
DPI Packets (other):   12              (1.00 pkts/flow)
```

nDPI Runtime Resource Usage

- nDPI has a static library resource cost (e.g. load protocols and internal mapping) and a per-flow cost.
- Flow metadata can use additional memory (e.g. HTTP URL). NOTE: you can use nDPI just to detect the protocol and thus avoid additional memory runtime costs.
- Typical memory overhead (ndpiReader)

```
nDPI Memory statistics:
  nDPI Memory (once):      3.08 KB
  Flow Memory (per flow):  1.13 KB
  Setup Time:              62 msec
```

Non-Dissector Detection Methods

Host Name (HTTP, DNS, TLS/QUIC...)

```
{ "xbox.com",                          "Xbox",  NDPI_PROTOCOL_XBOX, NDPI_PROTOCOL_CATEGORY_GAME, NDPI_PROTOCOL_FUN, NDPI_PROTOCOL_DEFAULT_LEVEL },
{ "xboxlive.com",                      "Xbox",  NDPI_PROTOCOL_XBOX, NDPI_PROTOCOL_CATEGORY_GAME, NDPI_PROTOCOL_FUN, NDPI_PROTOCOL_DEFAULT_LEVEL },
{ "xboxlive.com.akadns.net",           "Xbox",  NDPI_PROTOCOL_XBOX, NDPI_PROTOCOL_CATEGORY_GAME, NDPI_PROTOCOL_FUN, NDPI_PROTOCOL_DEFAULT_LEVEL },
{ "xboxlive.com.c.footprint.net",      "Xbox",  NDPI_PROTOCOL_XBOX, NDPI_PROTOCOL_CATEGORY_GAME, NDPI_PROTOCOL_FUN, NDPI_PROTOCOL_DEFAULT_LEVEL },
{ "xboxservices.com",                  "Xbox",  NDPI_PROTOCOL_XBOX, NDPI_PROTOCOL_CATEGORY_GAME, NDPI_PROTOCOL_FUN, NDPI_PROTOCOL_DEFAULT_LEVEL },
{ "xboxab.com",                        "Xbox",  NDPI_PROTOCOL_XBOX, NDPI_PROTOCOL_CATEGORY_GAME, NDPI_PROTOCOL_FUN, NDPI_PROTOCOL_DEFAULT_LEVEL },
```

Certificates (TLS/QUIC)

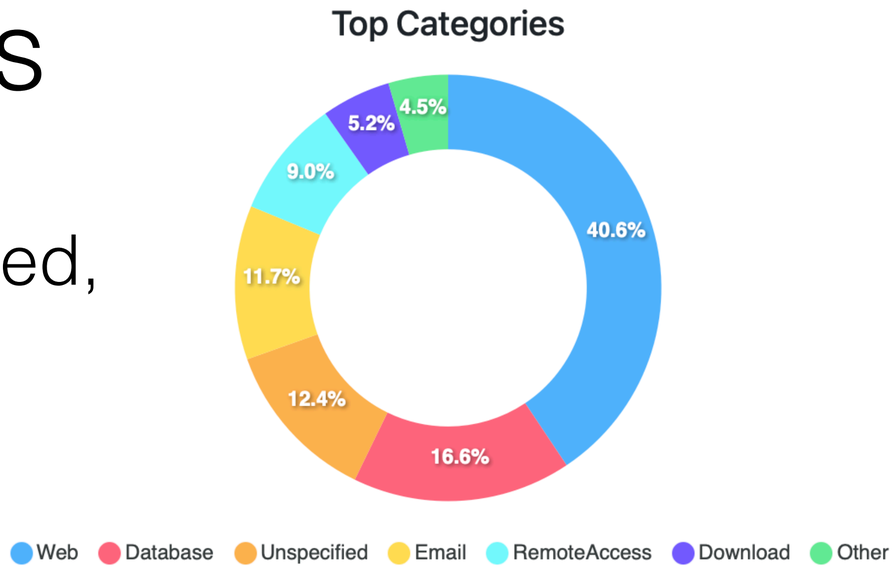
```
{ "CN=AnyDesk Client",                 NDPI_PROTOCOL_ANYDESK     },
{ "O=philandro Software GmbH",         NDPI_PROTOCOL_ANYDESK     },
{ "O=Kakao",                           NDPI_PROTOCOL_KAKAOTALK   },
{ "O=ntop.org",                        NDPI_PROTOCOL_NTOP        },
{ "O=Netflix",                         NDPI_PROTOCOL_NETFLIX     },
{ "O=Cloudflare",                      NDPI_PROTOCOL_CLOUDFLARE  },
{ "CN=simplednscrypt.org",             NDPI_PROTOCOL_DNSCRYPT    },
{ "CN=*.gateway.messenger.live.com",   NDPI_PROTOCOL_MSTEAMS     },
{ "OU=FortiGate",                      NDPI_PROTOCOL_FORTICLIENT },
```

IP Server Addresses

Useful for IXPs

```
{ 0xA29FC100 /* 162.159.193.0/24 */, 24, NDPI_PROTOCOL_CLOUDFLARE_WARP },
{ 0xA29FC000 /* 162.159.192.0/24 */, 24, NDPI_PROTOCOL_CLOUDFLARE_WARP },
{ 0xA29FC500 /* 162.159.197.0/24 */, 24, NDPI_PROTOCOL_CLOUDFLARE_WARP },
```

Categories



- Sometimes protocols are too fine grained, hence they better be clustered into categories (~120 as of now).
- Advantages:
  - Simple to implement "block all P2P", rather than block "BitTorrent, eMule..." As
    - We're constantly adding new protocols, so blocking a category is simpler.
    - Hardly all protocol names belonging to a category are known.
    - Monitoring traffic reports are easier to read.

Protocol Breeds

- While categories are designed to characterize the traffic in terms of content, they are not good for determining the nature of the traffic.
- Example: block all dangerous protocols

```
typedef enum {
  NDPI_PROTOCOL_SAFE = 0,                 /* Surely doesn't provide risks for
                                             the network. (e.g., a news site) */
  NDPI_PROTOCOL_ACCEPTABLE,               /* Probably doesn't provide risks,
                                             but could be malicious (e.g., Dropbox) */
  NDPI_PROTOCOL_FUN,                      /* Pure fun protocol, which may be prohibited by
                                             the user policy (e.g., Netflix) */
  NDPI_PROTOCOL_UNSAFE,                   /* Probably provides risks, but could be a normal
                                             Traffic. Unencrypted protocols with clear pass
                                             should be here (e.g., telnet) */
  NDPI_PROTOCOL_POTENTIALLY_DANGEROUS,    /* Possibly dangerous (ex. Tor). */
  NDPI_PROTOCOL_DANGEROUS,                /* Surely is dangerous (ex. smbv1). */
  NDPI_PROTOCOL_TRACKER_ADS,              /* Trackers, Advertisements... */
  NDPI_PROTOCOL_UNRATED                   /* Not implemented or impossible to classify */
} ndpi_protocol_breed_t;
```

Metadata Extraction [1/2]

- nDPI is designed to be used in two fashions:
  - Quick: report just the application protocol. Useful for inline traffic processing where the protocol needs to be detected ASAP in order to apply the policy.
  - Full: report protocol and metadata. Best for rich monitoring applications where extracted metadata can help during operations.
- Note that nDPI has not beed designed to extract everything or to allow a callback during metadata extraction (e.g. trigger a function when a URL is extracted).

Metadata Extraction [2/2]

```
    struct {
      char *server_names, *advertised_alpns, *negotiated_alpn, *tls_supported_versions, *issuerDN, *subjectDN;
      u_int32_t notBefore, notAfter;
      char ja3_server[33], ja4_client[37], *ja4_client_raw;
      u_int16_t server_cipher;
      u_int8_t sha1_certificate_fingerprint[20];
      u_int8_t client_hello_processed:1, ch_direction:1, subprotocol_detected:1,
               server_hello_processed:1, fingerprint_set:1, webrtc:1, _pad:2;

#ifdef TLS_HANDLE_SIGNATURE_ALGORITMS
      /* Under #ifdef to save memory for those who do not need them */
      u_int8_t num_tls_signature_algorithms;
      u_int16_t client_signature_algorithms[MAX_NUM_TLS_SIGNATURE_ALGORITHMS];
#endif

      struct tls_heuristics browser_heuristics;

      u_int16_t ssl_version, server_names_len;

      struct {
        u_int16_t version;
      } encrypted_ch;

      ndpi_cipher_weakness server_unsafe_cipher;

      u_int32_t quic_version;
      u_int32_t quic_idle_timeout_sec;
    } tls_quic; /* Used also by DTLS and POPS/IMAPS/SMTPS/FTPS */
```

Example of TLS/QUIC metadata

Protocol Detection and Caching

- Every dissected flow is independent (i.e. flow X does not influence dissection of flow Y).
- However some flows influence dissection of correlated flows. Example in BitTorrent/MS Teams after the initial negotiation, multiple flows have birth and are correlated to the initial flow.
- For this reason nDPI includes some LRU (Least Recently Used) caches that store flow information for relevant protocols that rely on this method.

```
typedef enum {
  NDPI_LRUCACHE_OOKLA = 0,
  NDPI_LRUCACHE_BITTORRENT,
  NDPI_LRUCACHE_STUN,
  NDPI_LRUCACHE_TLS_CERT,
  NDPI_LRUCACHE_MINING,
  NDPI_LRUCACHE_MSTEAMS,
  NDPI_LRUCACHE_FPC_DNS, /* FPC DNS cache */
  NDPI_LRUCACHE_SIGNAL,
  NDPI_LRUCACHE_MAX     /* Last one! */
} lru_cache_type;
```

FPC (First Packet Classification)

- Ideally it would be nice to detect a protocol at the very first packet with payload.
- While for many (e.g. DNS, SNMP, QUIC) UDP-based protocols this is the case, in the case of TCP, this is not possible.
- nDPI tries to reduce classification time by caching results and using them to implement FPC.
- Note: the use of FPC might potentially introduce false positives income cases (e.g. a CDN), however this risk is very low but there are advantages in detection time.

How FPC Work?

- The idea behind FPC is very simple:
  - Suppose nDPI observes this flow:
    192.168.20.13:1234 <-> a.b.c.d:443 [www.github.com]
    TLS.GitHub
  - nDPI stores in cache a.b.c.d:443 = TLS.GitHub
  - Future flows going towards a.b.c.d:443 will be labelled as TLS.GitHub without DPI
- Note: you can disable FPC at any time (at runtime) by means of the nDPI configuration.

DPI vs FPC

What DPI said

What FPC said and how

```
TCP 192.168.1.117:54871 <-> 109.94.160.99:443 [proto: 91.189/TLS.Zoom][IP: 0/Unknown][Encrypted]
[Confidence: DPI][FPC: 189/Zoom, Confidence: DNS][DPI packets: 11][cat: Video/26][127 pkts/54118
bytes <-> 83 pkts/17526 bytes][Goodput ratio: 84/69][2.00 sec][Hostname/SNI: zoomfrn99mmr.zoom.us]
[bytes ratio: 0.511 (Upload)][IAT c2s/s2c min/avg/max/stddev: 0/0 17/9 950/156 93/24][Pkt Len c2s/
s2c min/avg/max/stddev: 66/66 426/211 1506/1506 458/364][Risk: ** TLS (probably) Not Carrying HTTPS
**][Risk Score: 10][Risk Info: No ALPN][TCP Fingerprint: 2_64_65535_15db81ff8b0d/Unknown][TLSv1.2]
[JA4: t12d930700_72a4e8475a2e_4446390ac224][ServerNames: *.zoom.us,zoom.us][JA3S:
ada793d0f02b028a6c840504edccb652][Issuer: C=US, ST=Arizona, L=Scottsdale, O=GoDaddy.com, Inc.,
OU=http://certs.godaddy.com/repository/, CN=Go Daddy Secure Certificate Authority - G2][Subject:
OU=Domain Control Validated, CN=*.zoom.us][Certificate SHA-1:
F7:5A:83:A8:77:24:55:D7:6D:2E:93:F6:6E:9C:C9:7E:AD:9B:3B:E8][Firefox][Validity: 2019-03-25 19:38:42
- 2021-03-25 19:38:42][Cipher: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256][Plen Bins:
0,20,12,4,2,6,3,2,2,2,4,5,10,1,0,0,1,0,0,1,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,17,0,0
]
```

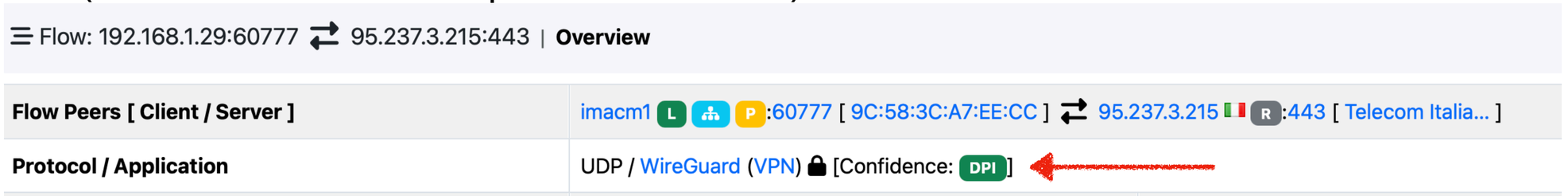
DPI Confidence [1/2]

- (Unless there is a bug in the code) nDPI detection is correct when packet payload is inspected.
- However sometimes:
  - The dissected flow already started and we missed the initial bytes that are necessary for detection.
  - Packet payload is not available (NetFlow, IXPs...).
  - There isn't a dissector for a protocol and we need to detect it by other means (e.g. IP address or port).

DPI Confidence [2/2]

```
typedef enum {
  NDPI_CONFIDENCE_UNKNOWN           = 0,    /* Unknown classification */
  NDPI_CONFIDENCE_MATCH_BY_PORT,            /* Classification obtained looking only at the L4 ports */
  NDPI_CONFIDENCE_NBPF,                     /* PF_RING nBPF (custom protocol) */
  NDPI_CONFIDENCE_DPI_PARTIAL,              /* Classification results based on partial/incomplete DPI information */
  NDPI_CONFIDENCE_DPI_PARTIAL_CACHE,        /* Classification results based on some LRU cache with partial/incomplete DPI information */
  NDPI_CONFIDENCE_DPI_CACHE,                /* Classification results based on some LRU cache (i.e. correlation among sessions) */
  NDPI_CONFIDENCE_DPI,                      /* Deep packet inspection */
  NDPI_CONFIDENCE_MATCH_BY_IP,              /* Classification obtained looking only at the IP addresses */
  NDPI_CONFIDENCE_DPI_AGGRESSIVE,           /* Aggressive DPI: it might be a false positive */
  NDPI_CONFIDENCE_CUSTOM_RULE,              /* Matching a custom rules */
```

- Based on the protocol, the dissection is influenced by past flows (e.g. by caches and FPC).
- Matching on IP/port is not reliable but sometimes that is the only choice (e.g. no packets) or the last resort (when all other options failed).



Confidence and Heuristics

- For some protocols it is not possible to do a full DPI due to the nature of the protocol.
- A typical example is OpenVPN and sub-dialects (e.g. NordVPN) where some heuristics are used (NDPI_CONFIDENCE_DPI_AGGRESSIVE), or BitTorrent.
- Other protocols such as most STUN/RTP-based protocol (e.g. Teams and WhatsApp) change flow protocol during lifetime, so nDPI needs to wait a bit before reporting the protocol.

Defining Custom Protocols

- A protocol dissector has to be written in C. However most modern protocols are sub-protocols of existing one (e.g. TLS or QUIC based).
- In other cases, some proprietary protocols are not supported by nDPI yet, and it's enough to label them based on IP:port pf the server (e.g. a mainframe).
- nDPI allows new protocols to be defined via its API but it has the ability to also define protocols at runtime via a configuration file.

Custom Protocols: protos.txt

```
$ cd example
$ cat protos.txt
#  Format:
#  <tcp|udp>:<port>,<tcp|udp>:<port>,.....@<proto>

tcp:81,tcp:8181@HTTP
udp:5062@SIP
tcp:860,udp:860,tcp:3260,udp:3260@iSCSI
tcp:9200@Elasticsearch
tcp:5601@Kibana
tcp:65535@TestProto

#
# nBPF filters (https://github.com/ntop/PF_RING/tree/dev/userland/nbpf))
#
# NOTE: they are evaluated in the same order they are defined !
#
nbpf:"host 192.168.1.1 and port 80"@HomeRouter


#  Subprotocols
#  Format:
#  host:"<value>",host:"<value>",.....@<subproto>
host:"disneyplus.com",host:"cdn.registerdisney.go.com",host:"disney-portal.my.onetrust.com",host:"disneyplus.bn5x.net",host:"disney-
plus.net"@DisneyPlus
host:"*.lvlt.dash.us.aiv-cdn.net.c.footprint.net"@AmazonVideo
host:"api-global.netflix.com"@Netflix


ip:3.3.3.3:446@CustomProtocolC=800
ipv6:[3ffe:507:0:1:200:86ff:fe05:80da]@CustomProtocolD=1024
ipv6:[247f:855b:5e16:3caf::]/64:100@CustomProtocolE=2048
ipv6:[247f:855b:5e16:3caf::]/64@CustomProtocolF=2049
ipv6:[fe80::76ac:b9ff:fe6c:c124]:12717@CustomProtocolG=2050
```

Combining nDPI with Cybersecurity

- As nDPI dissect the initial flow packets, it can be used to report unexpected communication conduct called "flow risk".
- nDPI does not include signatures (such as Snort/Suricata) but it's behavioral based.
- Risks are reported by protocol dissectors during traffic analysis. The idea is to report both protocol information, metadata and cybersecurity indicators that are valuable to AI-based traffic analysis.

nDPI Flow Risks

- HTTP suspicious user-agent
- HTTP numeric IP host contacted
- HTTP suspicious URL
- HTTP suspicious protocol header
- TLS connections not carrying HTTPS (e.g. a VPN over TLS)
- Suspicious DGA domain contacted
- Malformed packet
- SSH/SMB obsolete protocol or application version
- TLS suspicious ESNI usage
- Unsafe Protocol used
- Suspicious DNS traffic
- TLS with no SNI
- XSS (Cross Site Scripting)
- SQL Injection
- Arbitrary Code Injection/Execution
- Binary/.exe application transfer (e.g. in HTTP)
- Known protocol on non standard port
- TLS self-signed certificate
- TLS obsolete version
- TLS weak cipher
- TLS certificate expired
- TLS certificate mismatch
- DNS suspicious traffic
- HTTP suspicious content
- Risky ASN
- Risky Domain Name
- Malicious JA3 Fingerprint
- Malicious SHA1 Certificate
- Desktop of File Sharing Session
- TLS Uncommon ALPN
- TLS Certificate Validity Too Long
- Suspicious TLS Extension
- TLS Fatal Alert
- Suspicious Protocol traffic Entropy
- Clear-text Credentials Exchanged
- DNS Large Packet
- DNS Fragmented Traffic
- Invalid Characters Detected
- Possible Exploit Detected
- TLS Certificate Close to Expire
- Punycode/IDN Domain
- Error Code Detected
- Crawler/Bot Detected
- Anonymous Subscriber
- Unidirectional Traffic
- HTTP Obsolete Server
- ALPN/SNI Mismatch
- Client Contacted A Malware Host
- Binary File/Data Transfer (Attempt)
- Probing Attempt
- Obfuscated Traffic

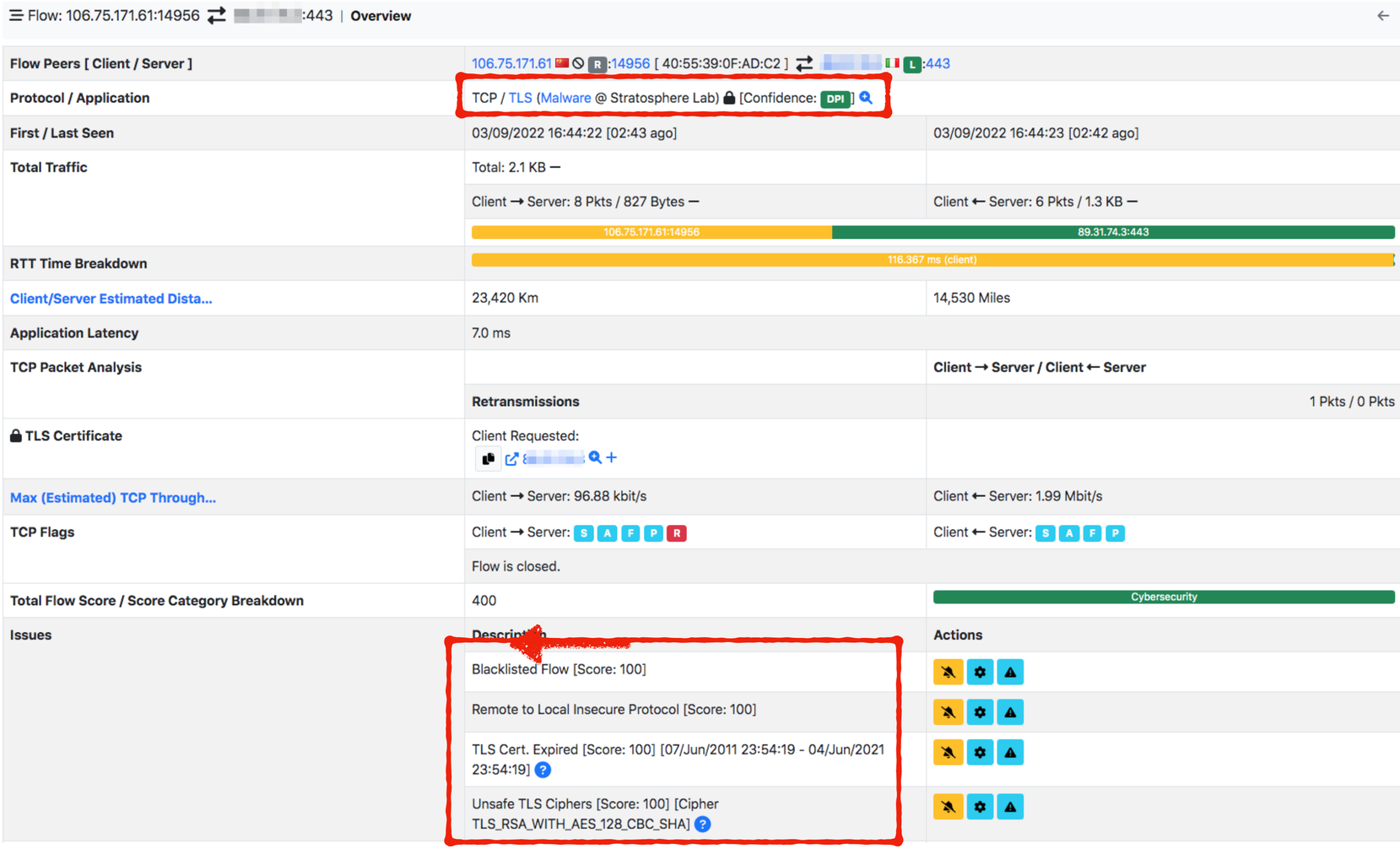
Legenda: Clear Text Only, Encrypted/Plain Text, Encrypted Only

Defining Risk Exceptions

- nDPI reports flow risk based on flow content.
- Sometimes not all risk are relevant (e.g. some alerts can be ignored as devices are already well protected albeit obsolete).
- It is possible to instruct nDPI not to generate specific risk alerts by means of protos.txt

```
#
# Risk Exceptions
#
# ip_risk_mask:   used to mask flow risks for IP addresses
# host_risk_mask: used to mask exceptions for domain names and hosts
#
# Syntax: <name>=<64 bit mask to be put in AND with the risk
#
# For IPs, the flow risk is put in AND (source IP mask OR destination IP mask)
# For Flows with a hostname (e.g. TLS) the risk is also put in AND with the host_risk_mask
#ip_risk_mask:192.168.1.0/24=0
ip_risk_mask:10.10.120.0/24=0
ip_risk_mask:10.196.157.228=0
ipv6_risk_mask:[fe80::356b:e047:3695:0]/112=0
ipv6_risk_mask:[fe80::7c0:e74e:87c3:5d93]=0
host_risk_mask:".home"=0


# Custom certification autorities we trust
trusted_issuer_dn:"CN=813845657003339838, O=Code42, OU=TEST, ST=MN, C=US"
```

nDPI in Passive Traffic Analysis



ntop

https://github.com/ntop/nDPI/

Fingerprinting Methods

- Protocol Fingerprint
  - Analyze a specific protocol (e.g. DHCP fingerprint, or TCP behavior for OS fingerprinting) in order to compute the expected fingerprint. Example: Window hosts do not set the Timestamps option in TCP SYN packets.
- Content Fingerprint
  - Create the fingerprint based on the content of specific protocol. Examples:
    - HTTP User-Agent
    - Android vs iOS vs Windows can be passively detected looking at DNS domain names queries (e.g. thinkdifferent.us and connectivitycheck.android.com)
    - Firefox connects via TLS to firefox.settings.services.mozilla.com

Some Network Fingerprints

- TCP Fingerprint
- Application Fingerprint
  - TLS/QUIC (JA3/JA4) and Web Browser Fingerprint
  - DHCP
  - RDP (Remote Desktop Protocol)
  - SSH (Secure Shell)
  - DHCP (Dynamic Host Configuration Protocol)
  - OpenVPNs (and dialects)
  - Obfuscated TLS (encrypted tunnels based on a TLS dialect)
  - Fully Encrypted Protocols (ShadowSocks, VMess, Trojan,...)

nDPI Fingerprints

```
static struct os_fingerprint tcp_fps[] = {
    { "2_64_65535_8bf9e292397e",    ndpi_os_freebsd     },

    { "2_64_64800_83b2f9a5576c",    ndpi_os_linux       },
    { "2_64_64240_2e3cee914fc1",    ndpi_os_linux       },
    { "2_64_29200_2e3cee914fc1",    ndpi_os_linux       },
    { "2_64_29200_d853e95bd80f",    ndpi_os_linux       }, /* Sonos */
    { "2_64_14600_8c07a80cc645",    ndpi_os_linux       }, /* QNAP */
    { "2_64_64240_2e3cee914fc1",    ndpi_os_linux       }, /* rPI */
    { "2_64_32120_2e3cee914fc1",    ndpi_os_linux       }, /* rPI */
    { "2_64_29200_90541420d839",    ndpi_os_linux       }, /* Suse Linux */
    { "2_64_64240_41a9d5af7dd3",    ndpi_os_linux       },

    { "2_64_65535_d876f498b09e",    ndpi_os_android     },
    { "2_64_65535_685ad951a756",    ndpi_os_android     },
    { "2_64_65535_41a9d5af7dd3",    ndpi_os_android     },
    { "2_64_65535_148107a0d970",    ndpi_os_android     },
    { "2_64_65535_f518bfb025b0",    ndpi_os_android     },

    { "2_128_64240_6bb88f5575fd",   ndpi_os_windows     },
    { "2_128_8192_4697958db063",    ndpi_os_windows     }, /* Windows 7 */
```

```
local ja4_db = {
   ['02e81d9f7c9f_736b2a1ed4d3'] = 'Chrome',
   ['07be0c029dc8_ad97e2351c08'] = 'Firefox',
   ['07be0c029dc8_d267a5f792d4'] = 'Firefox',
   ['0a330963ad8f_c905abbc9856'] = 'Chrome',
   ['0a330963ad8f_c9eaec7dbab4'] = 'Chrome',
   ['168bb377f8c8_a1e935682795'] = 'Anydesk',
   ['24fc43eb1c96_14788d8d241b'] = 'Chrome',
   ['24fc43eb1c96_14788d8d241b'] = 'Safari',
   ['24fc43eb1c96_845d286b0d67'] = 'Chrome',
   ['24fc43eb1c96_845d286b0d67'] = 'Safari',
   ['24fc43eb1c96_c5b8c5b1cdcb'] = 'Safari',
   ['2a284e3b0c56_12b7a1cb7c36'] = 'Safari',
   ['2a284e3b0c56_f05fdf8c38a9'] = 'Safari',
   ['2b729b4bf6f3_9e7b989ebec8'] = 'IcedID',
   ['39b11509324c_ab57fa081356'] = 'Chrome',
   ['39b11509324c_c905abbc9856'] = 'Chrome',
   ['39b11509324c_c9eaec7dbab4'] = 'Chrome',
   ['41f4ea5be9c2_06a4338d0495'] = 'Chrome',
```

Fingerprints enable accurate device detection

iOS/iPadOS/macOS (Intel)

- Send SYN+ECE+CRW. Others (including macOS Silicon) just SYN.
- Options (iOS but not iPadOS) end with a double EOL.

Windows

- Does not use the timestamp (8) option.
- Has a default TTL of 128, vs 64 used on Linux etc.

Fingerprinting in Cybersecurity

- Fingerprinting plays a crucial role in cyber security as it helps in detecting threats, securing networks, and implementing targeted security measures.
- Defenders:
  - Match malware signatures (e.g. TLS fingerprint or SSL certificate hash) and block malicious traffic.
  - Prevents massive scanners from exploring network services.
- Attackers
  - Use fingerprinting to detect flaws (e.g. CVEs) that can be used to attack the system.
  - During reconnaissance, identify application/OS version in order to target attacks towards weak victims.

Anticipate Problems with Fingerprints

```
Frame 1: 60 bytes on wire (480 bits), 60 bytes captured (480
Ethernet II, Src: 76:ac:b9:35:30:da (76:ac:b9:35:30:da), Dst:
Internet Protocol Version 4, Src: 192.168.10.145 (192.168.10.
Transmission Control Protocol, Src Port: 49175, Dst Port: 888
    Source Port: 49175
    Destination Port: 8888
    [Stream index: 0]
    [Stream Packet Number: 1]
  > [Conversation completeness: Incomplete (35)]
    [TCP Segment Len: 0]
    Sequence Number: 0    (relative sequence number)
    Sequence Number (raw): 253744456
    [Next Sequence Number: 1    (relative sequence number)]
    Acknowledgment Number: 0
    Acknowledgment number (raw): 0
    0101 .... = Header Length: 20 bytes (5)
  > Flags: 0x002 (SYN)
    Window: 65535
    [Calculated window size: 65535]
    Checksum: 0x5297 [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
  > [Timestamps]
```

https://zmap.io/

```
Frame 1: 60 bytes on wire (480 bits), 60 bytes captured (480 bits)
Ethernet II, Src: 76:ac:b9:35:30:da (76:ac:b9:35:30:da), Dst: PCSSyste
Internet Protocol Version 4, Src: 192.168.10.145 (192.168.10.145), Dst
Transmission Control Protocol, Src Port: 46998, Dst Port: 8888, Seq: 0
    Source Port: 46998
    Destination Port: 8888
    [Stream index: 0]
    [Stream Packet Number: 1]
  > [Conversation completeness: Incomplete (35)]
    [TCP Segment Len: 0]
    Sequence Number: 0    (relative sequence number)
    Sequence Number (raw): 1163206847
    [Next Sequence Number: 1    (relative sequence number)]
    Acknowledgment Number: 0
    Acknowledgment number (raw): 0
    0101 .... = Header Length: 20 bytes (5)
  > Flags: 0x002 (SYN)
    Window: 1024
    [Calculated window size: 1024]
    Checksum: 0xd56b [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
  > [Timestamps]
```

https://github.com/robertdavidgraham/masscan

ntop

https://github.com/ntop/nDPI/

What about Traffic Classification ? [1/2]

- Traffic fingerprinting allows network traffic to be clustered according to the sender OS (TCP Fingerprinting) and Application (e.g. JA3/4).

```
194_64_65535_dd5737e4fedb-t13d1516h2_8daaf6152771_9b887d9acb53 [ tiktokv.eu tiktokcdn.com snapchat.com tiktokv.com ]
194_64_65535_dd5737e4fedb-t13d1516ht_8daaf6152771_9b887d9acb53 [ tiktokv.eu ]
2_64_65535_dd5737e4fedb-t13d1516h2_8daaf6152771_e5627efa2ab1 [ googlevideo.com pinimg.com pinterest.com ]
194_64_65535_dd5737e4fedb-t13d1516h2_8daaf6152771_e5627efa2ab1 [ tiktokv.eu tiktokcdn.com snapchat.com tiktokcdn-us.com ]
194_64_65535_dd5737e4fedb-t13d181100_e8a523a41297_d5fe2c511efa [ tiktokcdn.com tiktokv.eu tiktokcdn-eu.com ]
2_64_65535_dd5737e4fedb-t13d1516h2_8daaf6152771_9b887d9acb53 [ tiktokcdn.com ]
2_64_65535_dd5737e4fedb-t12d220700_0d4ca5d4ec72_3304d8368043 [ microsoft.com ryanair.com ]
194_64_65535_dd5737e4fedb-t00d030800_55b375c5d22e_566d5108064c [ facebook.com ]
194_64_65535_dd5737e4fedb-t13d1314h2_f57a46bbacb6_14788d8d241b [ appsflyersdk.com ]
2_64_65535_dd5737e4fedb-t13d2015h2_a09f3c656075_3d00e4afe3b1 [ apple.com ]
2_64_65535_dd5737e4fedb-t00d0310h2_55b375c5d22e_50cc996d9024 [ facebook.com ]
2_64_65535_dd5737e4fedb-t00d030600_55b375c5d22e_8f5d6a331b25 [ facebook.com ]
194_64_65535_dd5737e4fedb-t13d0713gr_04ca88ad2b9b_d8054c94196c [ snapchat.com ]
194_64_65535_dd5737e4fedb-t13d181100_e8a523a41297_ef7df7f74e48 [ tiktokcdn-eu.com ]
194_64_65535_dd5737e4fedb-t13d2015h2_a09f3c656075_3d00e4afe3b1 [ apple.com ]
194_64_65535_dd5737e4fedb-t13d2014ht_a09f3c656075_14788d8d241b [ apple.com icloud.com ]
2_64_65535_dd5737e4fedb-t13d2014ht_a09f3c656075_14788d8d241b [ apple.com spotify.com cdn-apple.com ]
194_64_65535_d3a424420f2a-t13d2015h2_a09f3c656075_3d00e4afe3b1 [ icloud.com apple.com ]
2_64_0_dd5737e4fedb-t13d2014ht_a09f3c656075_14788d8d241b [ apple.com ]
2_64_65535_dd5737e4fedb-t12d220600_0d4ca5d4ec72_3304d8368043 [  ]
2_64_65535_d3a424420f2a-t13d2015h2_a09f3c656075_3d00e4afe3b1 [ apple.com ]
194_64_65535_dd5737e4fedb-t13d0311ap_55b375c5d22e_14aed462abe7 [ apple.com ]
194_64_65535_dd5737e4fedb-t13d181200_e8a523a41297_02c8e53ee398 [ tiktokcdn-eu.com ]
194_64_0_dd5737e4fedb-t13d2014h2_a09f3c656075_14788d8d241b [ icloud.com ]
```

What about Traffic Classification ? [2/2]

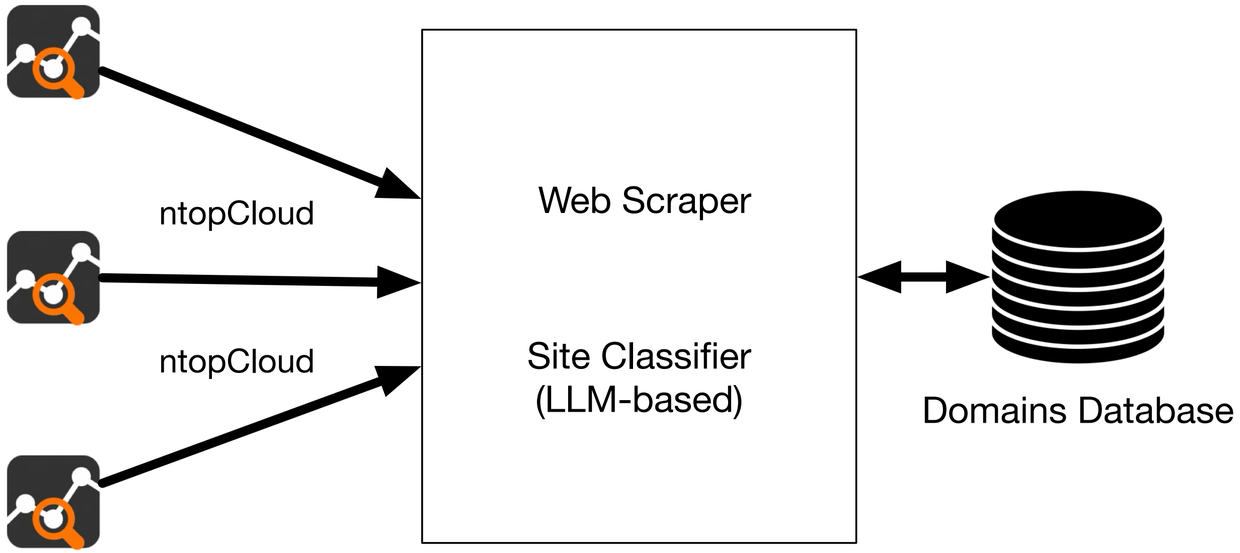
- Ok but what is tiktok.com or neacademy.it ?
- Is it possible to classify automatically traffic content ?
- AI can definitely help to do it automatically:
  - Domain name classification:< 5 sec including download time on a host without a GPU.
  - Multi-language support

```
deri@super 218> duckdb ./domains.duck
v1.2.1 8e52ec4395
Enter ".help" for usage hints.
D SELECT domain,category,description FROM domains where domain = 'neacademy.it';
```

domain varchar	category varchar	description varchar
neacademy.it	education	neacademy.it is an educational website that offers a variety of courses focusing on technology, programming, data science, an…

```
D
```

Site Classification Architecture



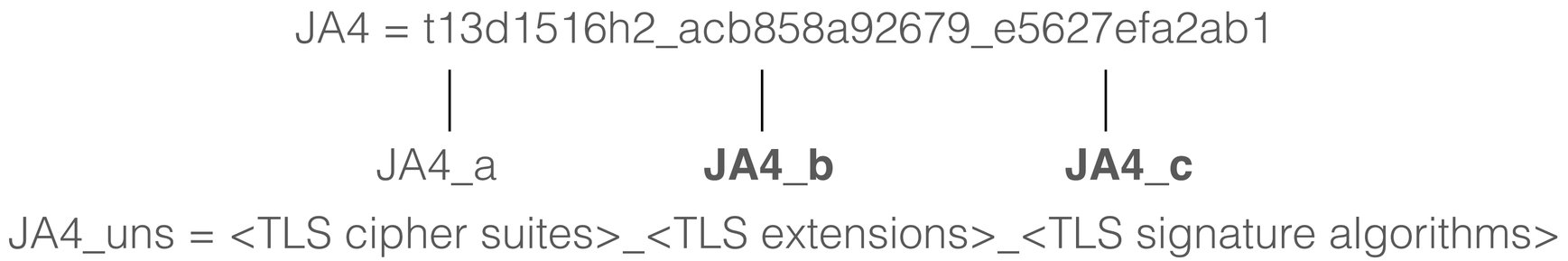
Semi-automated Creation of Fingerprints [1/5]

- Goals:
  - Identify unknown/unexpected apps using a given network.
  - Detect unauthorized applications from being installed on business devices.
  - Cluster flows belonging to the same application in order to separate their traffic from background noise.
  - Classify the application in categories in order to detect harmful traffic.

- In essence:
  - This is to ease the semi-automatic creation of application fingerprints to be used extending nDPI capabilities without coding.

Semi-automated Creation of Fingerprints [2/5]

- JA4 Client fingerprint (unsorted and unhashed version):



- Note:
  - As of today, this work is limited to TLS/QUIC.

Semi-automated Creation of Fingerprints [3/5]

Operational Steps:

- ndpiReader is used to parse a given pcap file and to generate a JSON file containing information on the flows.
- The JSON file is then analyzed, extracting, among other things, TCP and JA4_uns fingerprints and the contacted hostname.
- The hostname (SNI) is compared and filtered against a database of known apps and OS's hostnames, in order to cluster apps together.

Current output contains (subject to change):

- a list of pairs <JA4_uns>:<app names>
- a list of pairs <app name>:<JA4_uns fingerprints>

Semi-automated Creation of Fingerprints [4/5]

- Currently, the tool analyses pcap files that have been generated on mobile devices, including both Android and iOS.
- When clustering all the flows, 33 different JA4_uns are obtained: only 7 of them are shared among different apps (25 JA4_uns are unique for a single app).
- Some of the peculiarities are that the shared JA4_uns are very similar:
  - Two of the most common (12 apps) JA4_uns are almost identical: one is contained in the other and they are in the same order.
  - Three of the shared JA4_uns contain the same set of ciphers, extensions, and signature algorithm, but they are in a slightly different order.

Semi-automated Creation of Fingerprints [5/5]

- A : 1301,1302,1303,c02b,c02c,cca9,c02f,c030,cca8,c009,c00a,c013,c014,009c,009d,002f,0035_0017,ff01,000a,000b,0023,0005,000d,0033,002d,002b,0015_0403,0804,0401,0503,0805,0501,0806,0601,0201 : ['APP_Whatsapp', 'APP_Linkedin', 'APP_Facebook', 'APP_Microsoft', 'APP_Sony Music Center']
- B : 1301,1302,1303,c02c,c02b,cca9,c030,c02f,cca8,c00a,c009,c014,c013,009d,009c,0035,002f,c008,c012,000a_0017,ff01,000a,000b,0005,000d,0012,0033,002d,002b,001b,0015_0403,0804,0401,0503,0203,0805,0805,0501,0806,0601,0201 : ['APP_Microsoft', 'APP_Twitter', 'APP_Linkedin', 'APP_Pinterest', 'APP_Subito', 'APP_Spotify']
- C : 00ff,c02c,c02b,c024,c023,c00a,c009,c008,c030,c02f,c028,c027,c014,c013,c012,009d,009c,003d,003c,0035,002f,000a_000a,000b,000d,0005,0012,0017_0401,0201,0501,0601,0403,0203,0503,0603 : ['APP_Discord', 'APP_Microsoft', 'APP_Ryanair']

- Ongoing Work
  - Combine the hostname (SNI) with the signature to reduce multiple signature matches.
  - Use host/domain classification/reputation to identify the nature of the application.

nDPI as a Lightweighted IDS

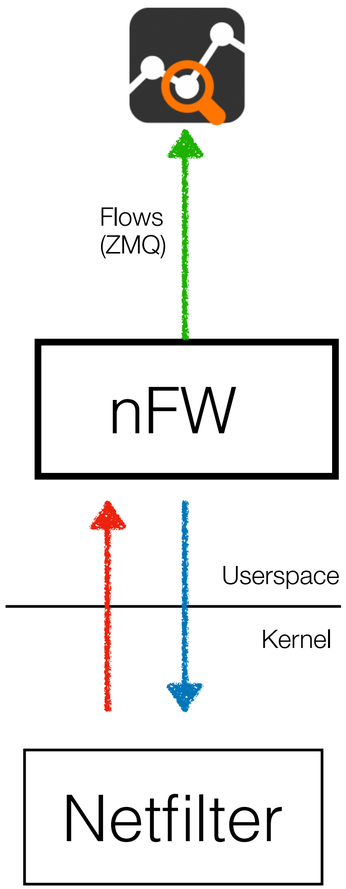
- ndpiReader -i http.pcap -k ndpi.txt -v 2 -K json


```
{"src_ip":"192.138.251.242","dest_ip":"131.114.21.22","src_port":51368,"dst_port":80,"ip":4,"tcp_fingerprint":"2_
64_14600_41a9d5af7dd3","proto":"TCP","ndpi": {"flow_risk": {"11": {"risk":"HTTP Susp User-
Agent","severity":"High","risk_score": {"total":310,"client":275,"server":35}},"47": {"risk":"HTTP Obsolete
Server","severity":"Medium","risk_score": {"total":510,"client":435,"server":75}}},"confidence":
{"6":"DPI"},"proto":"HTTP.ntop","proto_id":"7.26","proto_by_ip":"Unknown","proto_by_ip_id":0,"encrypted":0,"breed
":"Safe","category_id":14,"category":"Network","hostname":"version.ntop.org","domainame":"version.ntop.org","http
": {"url":"version.ntop.org\/version.xml","code":200,"content_type":"application\/xml","user_agent":"ntop\/
5.0+Fedora+RPM host\/x86_64-2.6.32-220.17.1.el6.x86_64-linux-gnu distro\/fedora release\/18 kernrlse\/
3.9.4-200.fc18.x86_64 GCC\/4.7.1 config(build=; host=; programprefix=; dependencytracking; execprefix=; bindir=;
sbindir=; datadir=; includedir=; libdir=; libexecdir=; localstatedir=; sharedstatedir=; mandir=; infodir=; snmp;
static) run() libpcap\/1.3.0 gdbm\/1.10. openssl\/1.0.1e-fips zlib\/1.2.7 access\/http interfaces(eth0,
eth1)"}},"detection_completed":1,"check_extra_packets":0,"flow_id":34,"first_seen":1380114096.448,"last_seen":138
0114097.063,"duration":0.615,"vlan_id":0,"bidirectional":1,"xfer":
{"data_ratio":-0.450,"data_ratio_str":"Download","src2dst_packets":5,"src2dst_bytes":850,"src2dst_goodput_bytes":
512,"dst2src_packets":6,"dst2src_bytes":2242,"dst2src_goodput_bytes":1838},"iat":
{"flow_min":2,"flow_avg":204.0,"flow_max":305,"flow_stddev":142.8,"c_to_s_min":0,"c_to_s_avg":153.2,"c_to_s_max":
308,"c_to_s_stddev":153.3,"s_to_c_min":0,"s_to_c_avg":102.7,"s_to_c_max":306,"s_to_c_stddev":143.8},"pktlen":
{"c_to_s_min":66,"c_to_s_avg":170.0,"c_to_s_max":578,"c_to_s_stddev":204.0,"s_to_c_min":66,"s_to_c_avg":373.7,"s_
to_c_max":1514,"s_to_c_stddev":529.3},"tcp_flags":
{"cwr_count":0,"ece_count":0,"urg_count":0,"ack_count":10,"psh_count":2,"rst_count":0,"syn_count":2,"fin_count":2
,"src2dst_cwr_count":0,"src2dst_ece_count":0,"src2dst_urg_count":0,"src2dst_ack_count":4,"src2dst_psh_count":1,"s
rc2dst_rst_count":0,"src2dst_syn_count":1,"src2dst_fin_count":1,"dst2src_cwr_count":0,"dst2src_ece_count":0,"dst2
src_urg_count":0,"dst2src_ack_count":6,"dst2src_psh_count":1,"dst2src_rst_count":0,"dst2src_syn_count":1,"dst2src
_fin_count":1},"c_to_s_init_win":74,"s_to_c_init_win":74}
```

Using nDPI to Enforce Traffic Policies

- In addition to massive traffic analysis, nDPI can also be used to block unwanted traffic for various reasons:
  - Cybersecurity: block scanners, threats etc.
  - Policy: block protocols (e.g. VPNs or Tor) that circumvent the network policies, or specific content (e.g. porn).
  - Safer surfing: block trackers.
  - Better experience: block advertising.
- Both ntopng (Edge) and nProbe (IPS) can be used inline.

Traffic Enforcement on Linux



- Netfilter primer

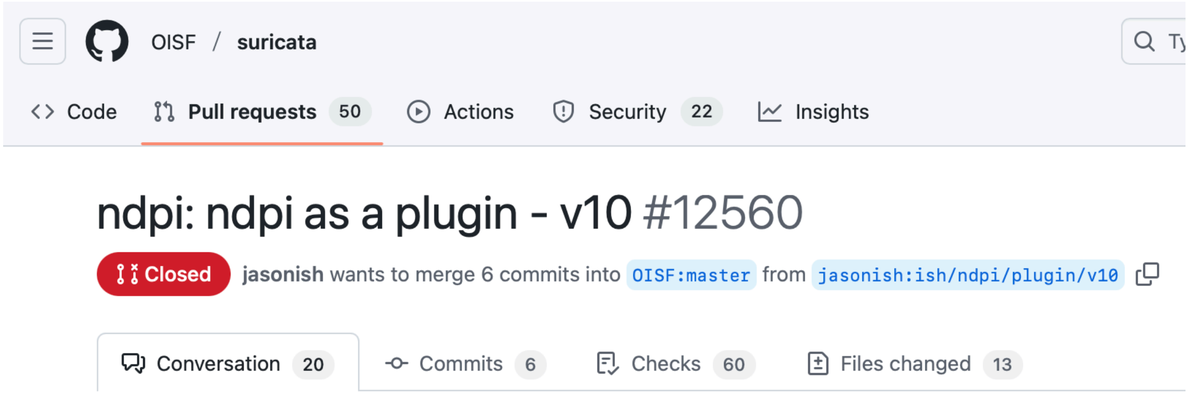
```
iptables -t mangle -A PREROUTING -j CONNMARK --restore-mark

iptables -A INPUT -i $IFNAME -m mark --mark 0 -j NFQUEUE --queue-num 0 --queue-bypass
iptables -A OUTPUT -o $IFNAME -m mark --mark 0 -j NFQUEUE --queue-num 0 --queue-bypass

iptables -t mangle -A POSTROUTING -j CONNMARK --save-mark
```

- In essence:
  - Unmarked flows are forwarded to NFQUEUE (i.e. nDPI).
  - Marked flows (i.e. those already detected by nDPI) are not, thus not introducing extra latency of CPU overhead.

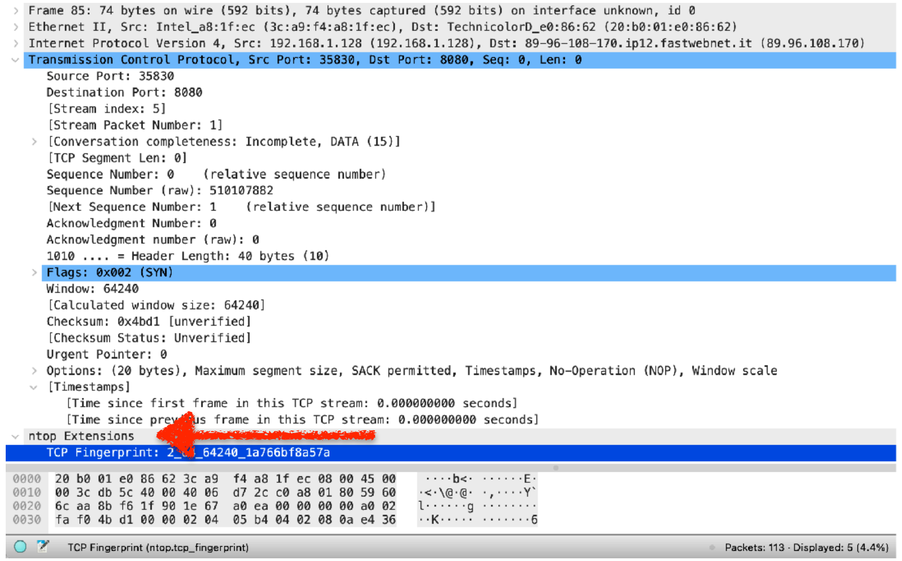
Legenda:
- Packets
- Verdict

nDPI in OpenSource Tools

- Suricata IDS (suricata.io)



- Wireshark (wireshark.org)

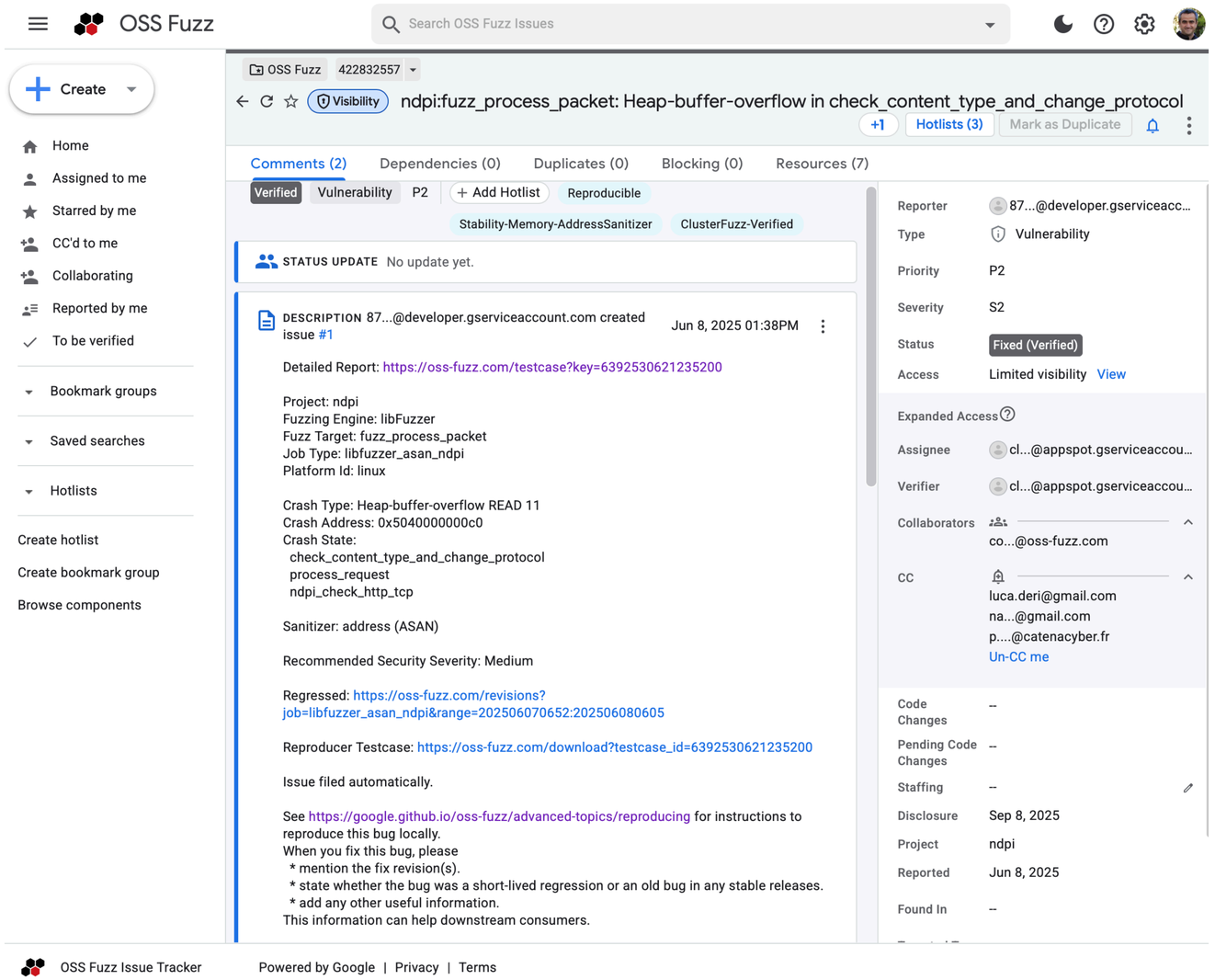


https://github.com/ntop/nDPI/

nDPI as a Library

- nDPI is more than a DPI toolkit as it includes various algorithms used my many network monitoring applications, including:
  - String Searching: Aho-Corasick
  - IP Matching: Radix Tree
  - Probabilistic Counting: HyperLogLog
  - Anomaly Detection: Exponential Smoothing
  - Data Comparison: Binning
  - Data Clustering (Unsupervised Machine Learning)
  - Streaming Data Analysis
  - Data Serialisation, Jitter/Entropy…

Google Continuous Code Fuzzing



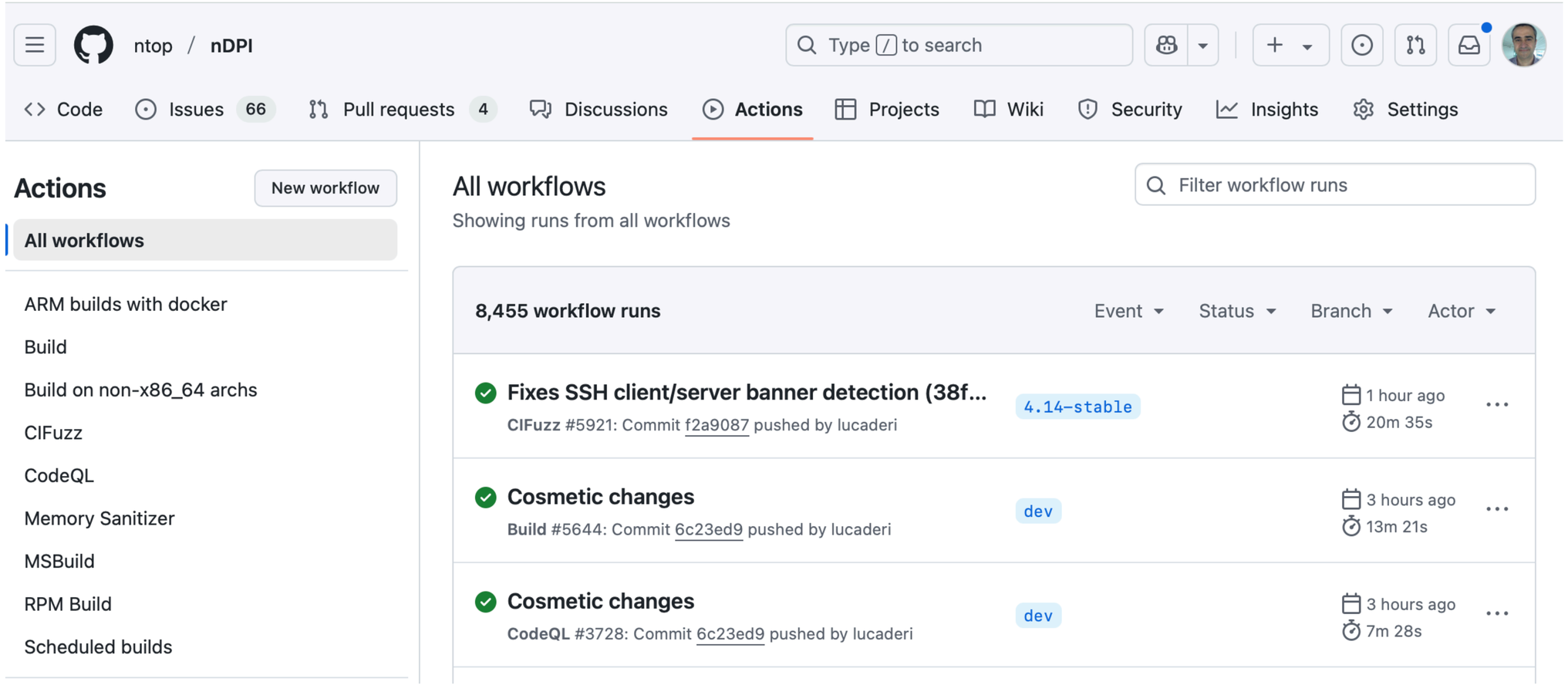
ntop

https://github.com/ntop/nDPI/

nDPI Regression Testing

```
$ cd tests
$ ./do.sh
Run configuration "caches_cfg" [--cfg=lru.ookla.size,0 --cfg=lru.msteams.ttl,1]
ookla.pcap                                        OK
teams.pcap                                        OK
Run configuration "caches_global" [--cfg=lru.ookla.scope,1 --cfg=lru.bittorrent.scope,1 --cfg=lru.stun.scope,1 --
cfg=lru.tls_cert.scope,1 --cfg=lru.mining.scope,1 --cfg=lru.msteams.scope,1 --cfg=lru.fpc_dns.scope,1]
bittorrent.pcap                                   OK
lru_ipv6_caches.pcapng                            OK
mining.pcapng                                     OK
ookla.pcap                                        OK
teams.pcap                                        OK
zoom_p2p.pcapng                                   OK
Run configuration "classification_only" [--conf=../../../example/only_classification.conf]
bittorrent.pcap                                   OK
bittorrent_tcp_miss.pcapng                        OK
forticlient.pcap                                  OK
http-basic-auth.pcap                              OK
http-pwd.pcapng                                   OK
http_auth.pcap                                    OK
ookla.pcap                                        OK
sip.pcap                                          OK
teams.pcap                                        OK
tls_1.2_unidir_client_no_cert.pcapng              OK
tls_1.2_unidir_server_no_cert.pcapng              OK
tls_1.2_unidirectional_client.pcapng              OK
tls_1.2_unidirectional_server.pcapng              OK
tls_1.3_unidirectional_client.pcapng              OK
tls_1.3_unidirectional_server.pcapng              OK
```

GitHub Testing



https://github.com/ntop/nDPI/