

SharkFest`26 Europe

Transformers in the Wire: Advanced DPI Fingerprinting for AI/Stealth VPN Detection

Luca Deri <deri@ntop.org>, @lucaderi
Daniele Sartiano <daniele.sartiano@gmail.com>

Sponsored by: PROFITAP DEEP NETWORK OBSERVABILITY, cPacket, endace Record. Respond.

About Us

- Luca is the founder of the ntop project and he develops open source network traffic monitoring applications.
  His home page is https://luca.ntop.org



- Daniele is a computer scientist working at the .it Registry, where he develops AI-based software for the domain name space and network traffic analysis.



This talk is a follow-up to the Sharkfest EU 2024 presentation "A Deep Dive Into Traffic Fingerprints using Wireshark"

Class Material



Agenda



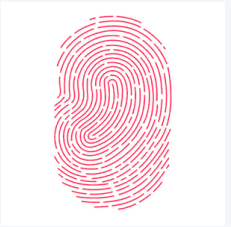
1. Introduction to Passive Traffic Fingerprinting
    1. Header-based Fingerprinting
    2. Payload-based Fingerprinting
    3. TLS/QUIC Traffic Fingerprinting
2. From Theory to Practice
    1. Fingerprinting Stealth VPNs
    2. Fingerprinting AI-generated Traffic
3. From Fingerprints to Transformers
    1. Modelling an AI model to recognize network traffic
    2. Positioning AI against Fingerprints

Part I:
Introduction to Passive Traffic Fingerprinting

What is a Network Fingerprint (IP and TCP Layers)

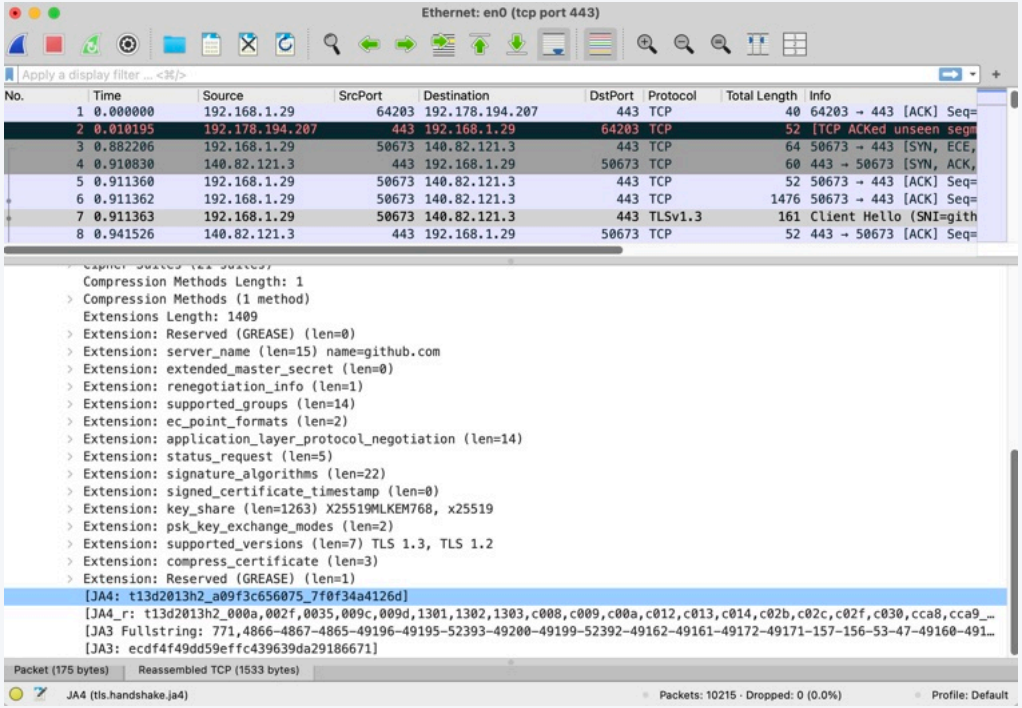
- A network traffic *fingerprint* is a (hopefully) unique set of *observable characteristics and patterns* extracted from data packets that identifies a specific device, operating system, application, or network protocol.

- When computers talk over a network, every piece of data is wrapped in standard protocols (like TCP, IP, or TLS) that act like envelope markings. Because different operating systems, apps, and devices construct these "envelopes" *slightly* differently, security systems can read these subtle variations to build a *fingerprint*.



Active vs Passive [1/2]

- Fingerprints can be determined using passive or active probing techniques with usual pro (no traffic, no fingerprints) / cons (traffic is injected in the network, hence we're not invisible).
- Passive
  Fingerprints are calculated by passively observing network traffic and producing the fingerprint according to "de-facto" techniques (e.g. JA4).
- As shown later, fingerprinting encrypted traffic has interesting features as ciphers and extensions ease fingerprint calculation.



Active vs Passive [2/2]



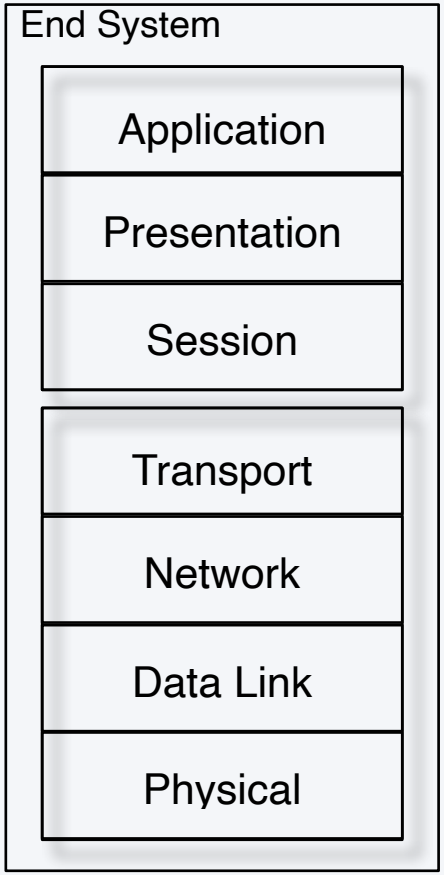
- Active fingerprinting is implemented by actively sending packets to a target machine in order to receive a response.
- Port scan can be considered a basic fingerprinting technique as it can be used to determine the operating system or read the version of specific services (e.g. read the HTTP server version and use it to find vulnerabilities) for attacking it.
- Some active fingerprinting tools:
  - nmap a popular network scanner including host discovery and service and operating system detection.
  - JARM a TLS server fingerprinting application developed by Salesforce. It provides the ability to identify and group malicious TLS servers on the Internet.
- NOTE: Throughout this talk, we'll focus only on passive fingerprints.

Active vs Passive: Pros and Cons



	Passive Fingerprinting	Active Fingerprinting
Pros	• Zero Stealth Risk: Completely invisible; cannot be blocked by target firewalls. • No Network Overhead: Does not inject artificial traffic or impact system performance. • Safe for Sensitive Environments: Ideal for Operational Technology (OT) and medical systems that crash easily. • Continuous Monitoring: Captures data 24/7 as traffic moves.	• High Accuracy: Highly reliable for pinpointing exact software and OS versions. • On-Demand Speed: Collects device profiles instantly without waiting for traffic. • Granular Inspection: Can validate passive assumptions and find hidden vulnerabilities. • Targeted Control: Allows testers to probe specific ports and flags.
Cons	• Traffic Dependent: If a device is silent or doesn't pass the monitoring node, it cannot be fingerprinted. • Encryption Blindspots: Encrypted payloads can hide deep application details. • Ambiguity: Shared network stacks or generic traffic patterns can lead to vague results.	• High Detection Risk: Triggers Firewalls, IDS, and IPS instantly. • Risk of Disruption: Can saturate thin pipes or freeze fragile legacy/OT systems. • Misses Quiet Devices: Only profiles devices online during the brief scan window.
Tools	• Wireshark/tshark	• nmap • JARM

Fingerprinting Families: Can Happen Anywhere



App (RTP/RTSP fingerprint Meet vs Teams vs Zoom)

Network Library Fingerprinting (TLS/QUIC)

SNMP, NetBIOS

TCP/IP stack (Linux vs Win vs macOS)

IEEE 802.11

CDP Protocol, ARP

Fingerprints: IP Stack [1/4]



Every operating system handles basic traffic a little differently. Wireshark can be used to look at IPv4 and IPv6 Headers and extract:

Header-based Fingerprinting

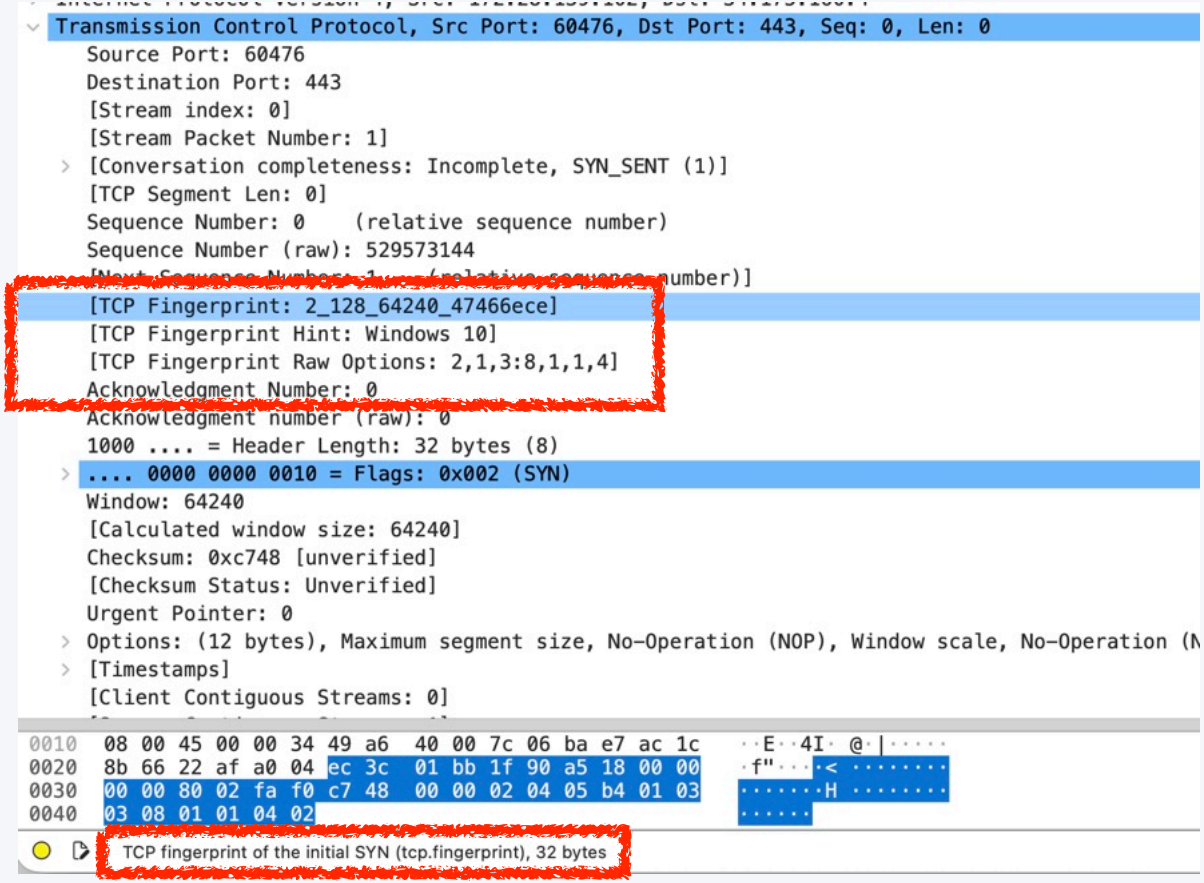
- Time-To-Live (TTL) / Hop Limit: A number that prevents packets from looping forever. Linux usually starts this at 64, Windows at 128, and macOS/iOS at 64.
- TCP Window Size: Dictates how much data a device can accept before sending an acknowledgement. On modern OSs there is no longer a single "fixed" default TCP window size (RFC 7323 (TCP Window Scaling)). However, the *initial* values, maximum bounds, and auto-tuning behaviors still differ heavily by OS kernel implementations. Example: Linux 64-128 KB, Windows 64 KB, macOS/iOS 128-256 KB.
- TCP Options: Extra parameters like Maximum Segment Size (MSS) or Selective Acknowledgments (SACK). The exact *order* and presence of these options are highly distinct signatures for specific operating systems.

Fingerprints: IP Stack [2/4]



- Recently I have implemented TCP fingerprinting in Wireshark based on the nDPI TCP Fingerprint specification (https://github.com/ntop/nDPI/blob/dev/fingerprint/nDPI_TCP_fingerprint.md)
- Core Purpose: Identifies client operating systems passively from the first connection-opening SYN packet —requiring no handshake completion or bidirectional visibility.
- The String Format: The fingerprint is an underscore-separated ASCII string structured as: **<tcp_flags>_<ttl_bucket>_<tcp_window>_<options_hash>**
- Options Serialization (R): TCP options are written in wire-order as human-readable, comma-separated decimal tokens (e.g., 2,4,8,1,3:7 for Linux).
- Advanced Noise Reduction: To prevent context drift, the specification actively strips *ephemeral, session-specific data* before hashing:
  - Value-Skipped: MSS (path-dependent) and Timestamps/MACs (per-connection/segment) are omitted from the hash.
  - Fully Dropped: TCP Fast Open (TFO) cookies and their adjacent NOP padding are removed entirely.
- Informative Sidecar Data: Elements that vary by path or application state (tcp_mss and tcp_fastopenstatus) are exported next to the fingerprint rather than inside it.
- Hashing Mechanism: The ASCII string of options is processed using a fast, deterministic 32-bit MurmurHash3 (seed 0), producing a compact 8-character hex layout.

Fingerprints: IP Stack [3/4]



Fingerprints: IP Stack [4/4]



Header-based Fingerprinting

Example of TCP signatures

```
static struct os_fingerprint tcp_fps[] = {
  { "2_64_65535_2b9b4c52",     ndpi_os_freebsd      }, /* 2,1,3:6,4,8       */
  { "2_64_64240_789c1d1f",     ndpi_os_linux        }, /* 2,4,8,1,3:7       */
  { "2_64_65535_58b6a464",     ndpi_os_android      }, /* 2,4,8,1,3:10      */
  { "194_64_65535_e76f0f14",   ndpi_os_apple        }, /* 2,1,3:6,1,1,8,4,0 */
  { "2_64_65535_e76f0f14",     ndpi_os_apple        }, /* 2,1,3:6,1,1,8,4,0 */
  { "194_64_65535_2b6d3038",   ndpi_os_apple        }, /* 2,1,3:5,1,1,8,4,0 */
  { "2_128_64240_47466ece",    ndpi_os_windows      }, /* 2,1,3:8,1,1,4     */
  { "2_128_65535_47466ece",    ndpi_os_windows      }, /* 2,1,3:8,1,1,4     */
  { NULL,                      ndpi_os_unknown    },
};
```

What can you do with IP stack fingerprinting ?
Detect the operating system family (e.g. Windows vs macOS/iOS vs Android) useful for detecting inconsistencies (e.g. too many OSs per IP -> hotspot or NAT).

What you cannot do with IP stack fingerprinting:
Recognize application (e.g. a stealth VPN) but it can help to characterize the traffic nature if the application is recognized (e.g. XRP-Tunnel Plus for iOS).

Fingerprints: Upper Layers

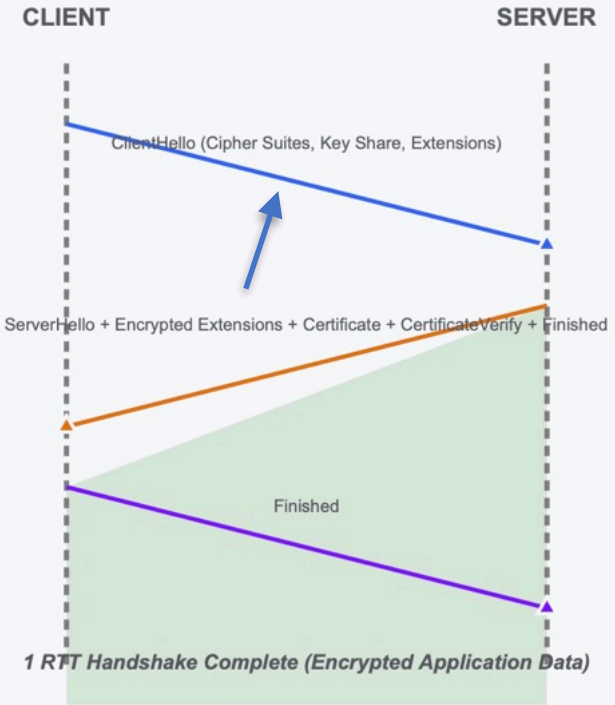


Header-based Fingerprinting

- Today most Internet traffic is encrypted (90%+).
- The paradox: while encryption prevents intruders to inspect the data being exchanged, it offers new fingerprinting methods not available with clear-text traffic.
  - When a device establishes a secure HTTPS connection, it performs a TLS handshake. Before any actual data is encrypted, the device openly broadcasts its capabilities:
    - Cipher Suites**:** A long list of cryptographic algorithms the client supports, ordered by preference.
    - TLS Extensions**:** Extra features the client wants to use (e.g., Supported Versions, Application-Layer Protocol Negotiation).
  - Handshake data can be used to create an application-level fingerprint.
- Later in this talk we'll discuss TLS/QUIC fingerprinting in detail.

How TLS Negotiation Works [1/3]

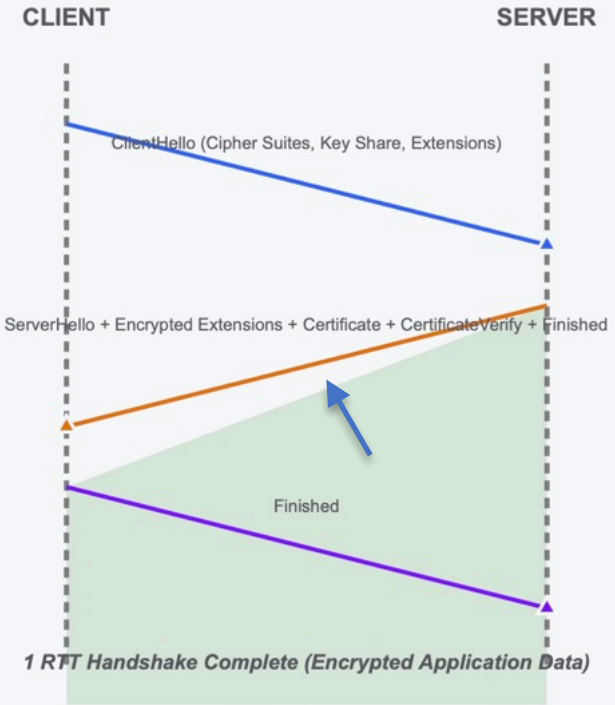
- Step 1: The Client Hello (Key Share Proposal)
  - The client starts the handshake by sending a ClientHello message to the server. To achieve the fast 1-RTT speed, the client does not wait for a negotiation; it actively guesses the server's preferred cryptographic algorithms. It sends:
  - A list of supported Cipher Suites (encryption algorithms).
  - A Key Share: Pre-generated public keys based on popular key-exchange algorithms (like ECDHE).
  - Supported TLS versions (e.g. TLS 1.3).



How TLS Negotiation Works [2/3]

- Step 2: The Server Response (Key Agreement & Authentication)

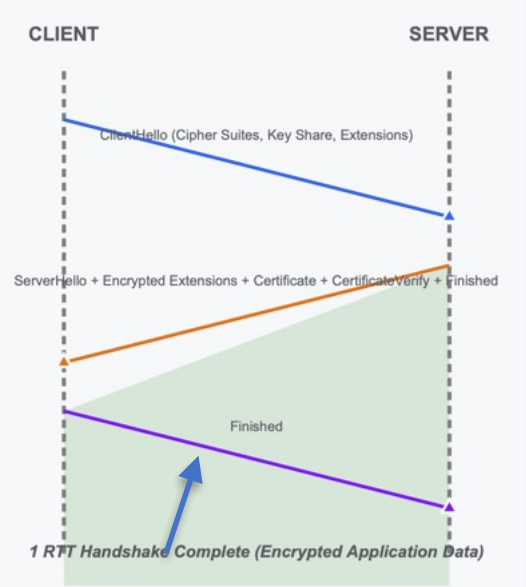
- The server processes the client's proposal. If it accepts one of the client's guessed key-exchange algorithms, it responds with its own parameters. This entire response is packed together:
  - ServerHello: Confirms the selected cipher suite and provides the server's matching Key Share. At this precise moment, both sides compute the shared secret key, and all subsequent handshake messages are encrypted.
  - Encrypted Extensions: Internal cryptographic configurations.
  - Certificate: The server's public TLS certificate, proving its identity.
  - CertificateVerify: A digital signature created using the server's private key, proving the server actually owns the certificate it just sent.
  - Finished: A cryptographic check confirming the handshake has not been tampered with.



How TLS Negotiation Works [3/3]

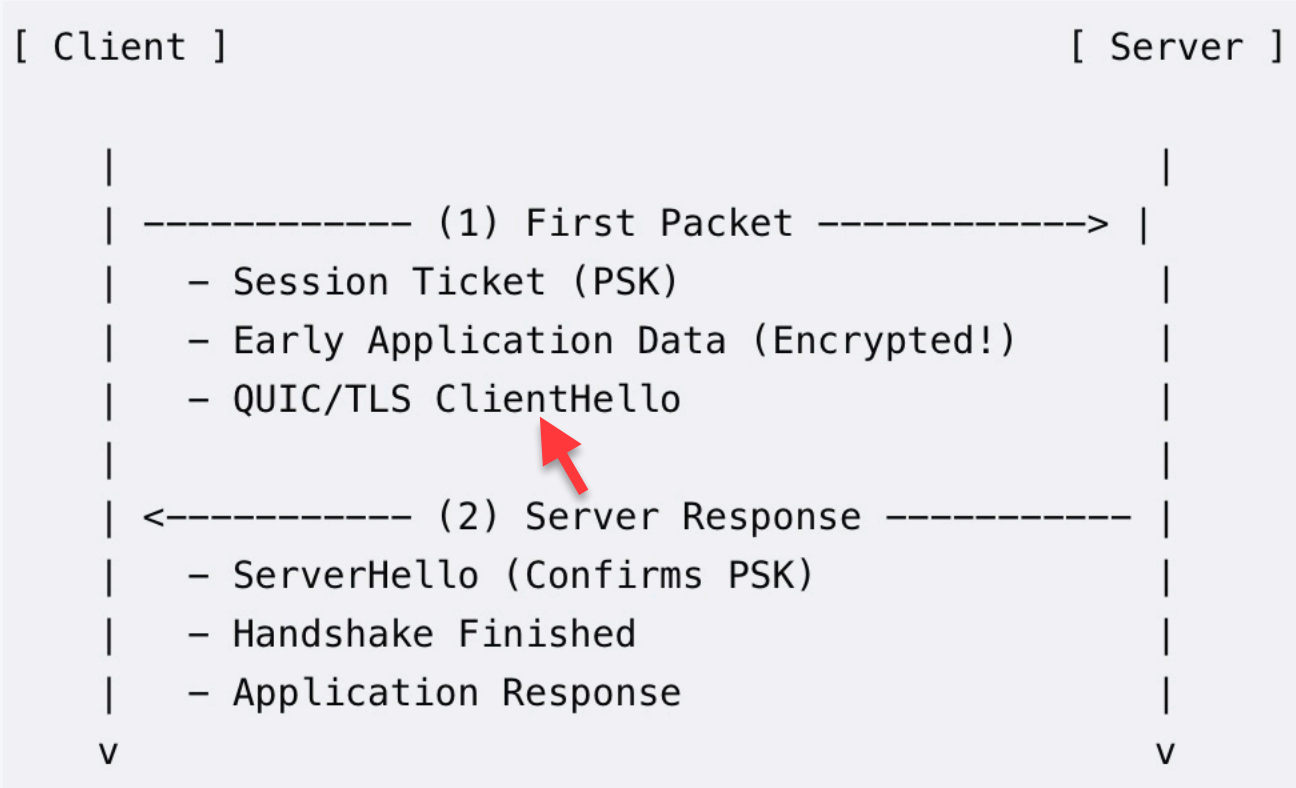
- Step 3: Client Verification and Finish

- The client receives the server's response and executes its checks:
  - It verifies the Certificate against its trusted Certificate Authorities (CAs) and validates the CertificateVerify signature.
  - It uses the server's Key Share to derive the exact same shared encryption keys.
  - It sends its own Finished message to the server to confirm it has successfully switched to encrypted communication.



No.	Time	Source	Destination	Protocol	Length	TLS Length	Info
32	3.009802	192.168.1.117	52.202.62.238	TLSv1.2	571	512	Client Hello (SNI=log.zoom.us)
37	3.119716	52.202.62.238	192.168.1.117	TLSv1.2	1506	111	Server Hello
41	3.120232	52.202.62.238	192.168.1.117	TLSv1.2	1306	5140,333,4	Certificate, Server Key Exchange, Server Hello Done
45	3.132813	192.168.1.117	52.202.62.238	TLSv1.2	245	70,1,60,40	Client Key Exchange, Change Cipher Spec, Encrypted Handshake Mes…
52	3.240003	52.202.62.238	192.168.1.117	TLSv1.2	105	1,40	Change Cipher Spec, Encrypted Handshake Message
54	3.242563	192.168.1.117	52.202.62.238	TLSv1.2	812	753	Application Data
84	3.350003	52.202.62.238	192.168.1.117	TLSv1.2	228	169	Application Data

QUIC vs TLS Handshake [1/2]



QUIC vs TLS Handshake [2/2]



No.	Time	Source	Destination	Protocol	Length	TLS Length	Info
1	0.000000	187.227.136.152	211.247.147.90	QUIC	1392		Initial, DCID=ff9415b5b005860d, PKN: 1, CRYPTO
2	0.005872	211.247.147.90	187.227.136.152	QUIC	1392		Initial, SCID=ff9415b5b005860d, PKN: 1, CRYPTO
3	0.069285	187.227.136.152	211.247.147.90	QUIC	1392		Initial, DCID=ff9415b5b005860d, PKN: 2
4	0.116640	211.247.147.90	187.227.136.152	QUIC	1392		Handshake, SCID=ff9415b5b005860d
5	0.116659	211.247.147.90	187.227.136.152	QUIC	1392		Handshake, SCID=ff9415b5b005860d
6	0.116692	211.247.147.90	187.227.136.152	QUIC	1254		Handshake, SCID=ff9415b5b005860d
7	0.182846	187.227.136.152	211.247.147.90	QUIC	83		Handshake, DCID=ff9415b5b005860d
8	0.185163	187.227.136.152	211.247.147.90	QUIC	83		Handshake, DCID=ff9415b5b005860d
9	0.190998	187.227.136.152	211.247.147.90	QUIC	115		Handshake, DCID=ff9415b5b005860d

```
..00 .... = Packet Type: Initial (0)
[.... 00.. = Reserved: 0]
[.... ..00 = Packet Number Length: 1 bytes (0)]
Version: Google T051 (0x54303531)
Destination Connection ID Length: 8
Destination Connection ID: ff9415b5b005860d
Source Connection ID Length: 0
Token Length: 0
Length: 1332
[Packet Number: 1]
Payload […]: cd6c78ab2880cd312769b2e008aaa61f91ff63764e5a29bfa2f509327be26808a2975e4ccb1b7589a66473fcd0e5e2cf3a2d6ba6051d567d55e01c119a3ca601
CRYPTO
  Frame Type: CRYPTO (0x08)
  Offset: 0
  Length: 307
  Crypto Data
  TLSv1.3 Record Layer: Handshake Protocol: Client Hello
    Handshake Protocol: Client Hello
PADDING Length: 1003
```

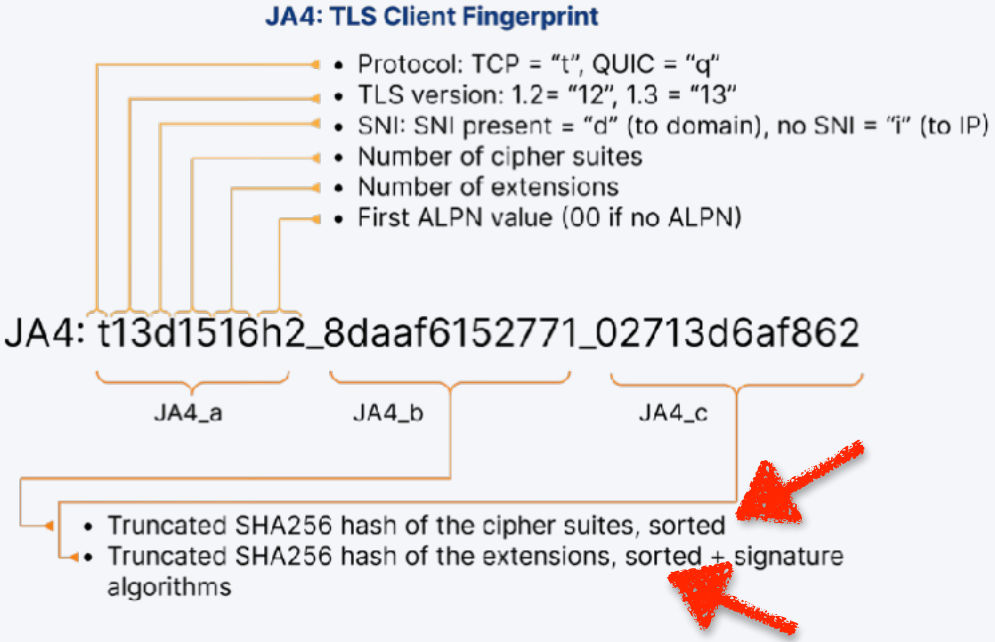
```
0000  08 00 41 33 01 00 01 2f  03 03 a5 ee 58 f8 26 2d   ··A3···/ ····X·&-
0010  48 44 99 6d 0d 81 41 cf  e8 24 e4 63 d4 6c 83 17   HD·m··A· ·$·c·l··
0020  d9 21 3a f3 31 86 28 5b  24 a9 00 00 06 13 01 13   ·!:·1·([ $·······
0030  02 13 03 01 00 01 00 00  00 00 13 00 11 00 00 0e   ········ ········
0040  77 77 77 2e 67 6f 6f 67  6c 65 2e 63 6f 6d 00 0a   www.goog le.com··
0050  00 08 00 06 00 1d 00 17  00 18 00 10 00 0a 00 08   ········ ········
0060  07 68 33 2d 54 30 35 31  00 0d 00 14 00 12 04 03   ·h3-T051 ········
```

Packet (1392 bytes) | Decrypted QUIC (1315 bytes)

Handshake protocol message (tls.handshake), 307 bytes — Packets: 12

TLS/QUIC Client Fingerprinting

JA4 is a TLS/QUIC client fingerprinting method analyzes the TLS Client Hello sent. While JA4 has been released under BSD 3-Clause, all other (named JA4+ for TCP, HTTP, SSH...) are patent pending and subject to license. Wireshark implements only JA4.



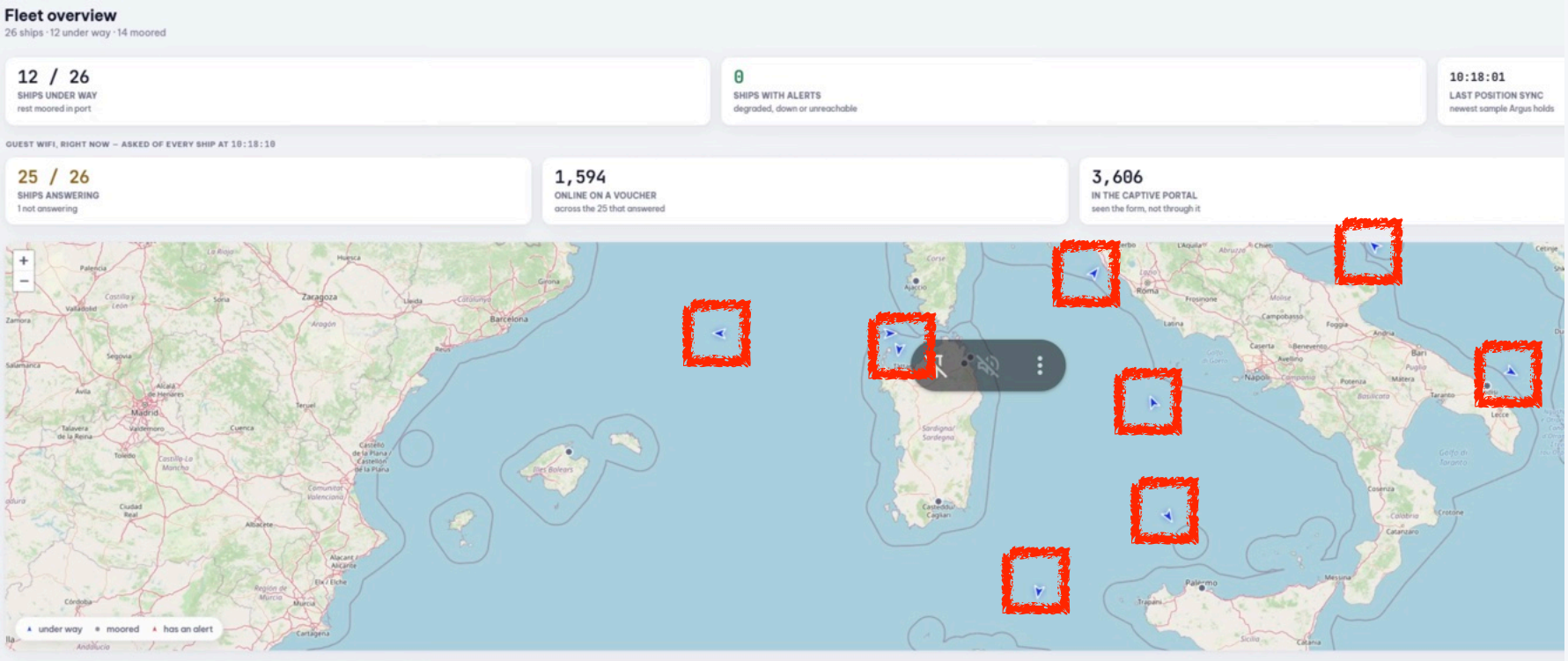
JA4 in Wireshark



```
Frame 2: Packet, 161 bytes on wire (1288 bits), 161 bytes captured (1288 bits) on interface en0, id 0
Ethernet II, Src: Apple_a7:ee:cc (9c:58:3c:a7:ee:cc), Dst: VantivaTechn_60:ed:80 (a4:91:b1:60:ed:80)
Internet Protocol Version 4, Src: 192.168.1.29, Dst: 116.203.142.127
Transmission Control Protocol, Src Port: 55172, Dst Port: 443, Seq: 1441, Ack: 1, Len: 95
[2 Reassembled TCP Segments (1535 bytes): #1(1440), #2(95)]
Transport Layer Security
    [Stream index: 0]
    TLSv1 Record Layer: Handshake Protocol: Client Hello
        Content Type: Handshake (22)
        Version: TLS 1.0 (0x0301)
        Length: 1530
        Handshake Protocol: Client Hello
            Handshake Type: Client Hello (1)
            Length: 1526
            Version: TLS 1.2 (0x0303)
            Random: f95de09030a880c8c94eb02110565c7c96ff4520c04415791507b4795b85bbca
            Session ID Length: 32
            Session ID: 60750858e8d8fb8601166309bc1b7a5749b099ef8af91c20ad851637a11043cd
            Cipher Suites Length: 42
            Cipher Suites (21 suites)
            Compression Methods Length: 1
            Compression Methods (1 method)
            Extensions Length: 1411
            Extension: Reserved (GREASE) (len=0)
            Extension: server_name (len=17) name=www.ntop.org
            Extension: extended_master_secret (len=0)
            Extension: renegotiation_info (len=1)
            Extension: supported_groups (len=14)
            Extension: ec_point_formats (len=2)
            Extension: application_layer_protocol_negotiation (len=14)
            Extension: status_request (len=5)
            Extension: signature_algorithms (len=22)
            Extension: signed_certificate_timestamp (len=0)
            Extension: key_share (len=1263) X25519MLKEM768, x25519
            Extension: psk_key_exchange_modes (len=2)
            Extension: supported_versions (len=7) TLS 1.3, TLS 1.2
            Extension: Reserved (GREASE) (len=1)
            [JA4: t13d2013h2_a09f3c656075_7f0f34a4126d]
            [JA4_r: t13d2013h2_000a,002f,0035,009c,009d,1301,1302,1303,c008,c009,c00a,c012,c013,c014,c02b,c
            [JA3 Fullstring: 771,4866-4867-4865-49196-49195-52393-49200-49199-52392-49162-49161-49172-49171
            [JA3: ecdf4f49dd59effc439639da29186671]
```

Part II:
Stealth VPN Traffic Analysis

Everything Started Here [1/2]

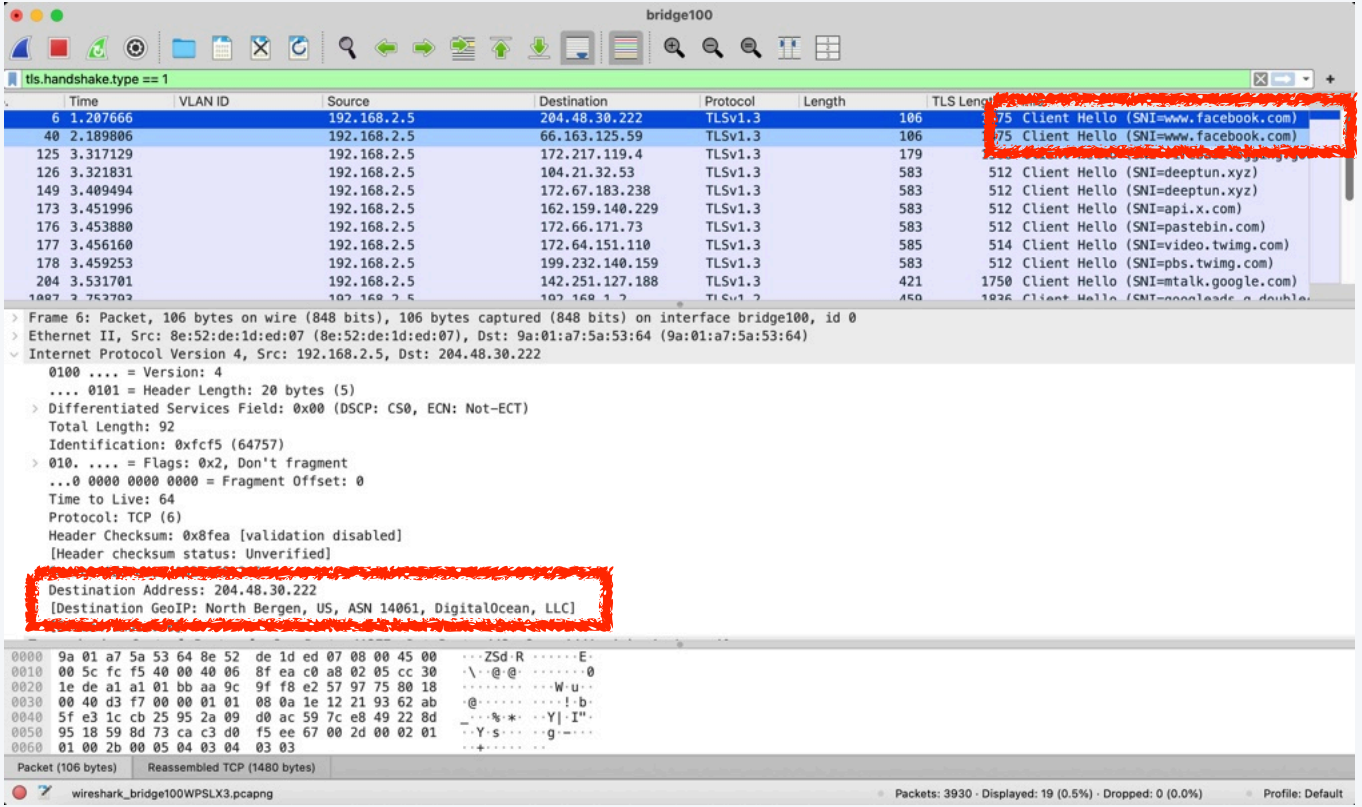


Everything Started Here [2/2]



- Playground Features
  - Tenth of thousand different WiFi clients per day.
  - "Annotated traffic": authentication returned information about monitored devices.
  - "Ghost" users are (relatively) easy to find:
    - Looking at their traffic volume compared to allowed protocols.
    - Checking traffic properties:
      - Very few large "elephant" flows.
      - A lot of DNS (resolve) activities, but not traffic to those resolved addresses.

What's Wrong With This Packet ?



File: pcap/HA-TunnelPlus/hatunnelplus_tls_facebook.com.pcapng

What is a Stealth VPN ? [1/2]



A **stealth VPN** (also known as an obfuscated VPN or VPN masking) is a VPN connection that uses obfuscation technology to disguise encrypted VPN traffic as regular internet traffic.

How a Stealth VPN Works

- **Encryption**: The VPN encrypts your data packets as usual.
- **Obfuscation** (The Disguise): The stealth mechanism strips away recognizable VPN headers, alters the connection handshake, or wraps the data inside an additional layer of SSL/TLS encryption.
- **Port 443 Routing**: The disguised traffic is sent through port 443 (or over 80 with HTTP-proxy traffic, or tunneling over SSH/DNS). This is the exact same port used by billions of secure websites daily (like online banking or social media).
- **Filter Bypass**: Network firewalls performing Deep Packet Inspection (DPI) see the traffic and mistake it for standard web browsing, letting it pass through seamlessly.

What is a Stealth VPN ? [2/2]



Feature	Standard VPN	Stealth VPN
Primary Goal	Encrypt data and hide your IP address.	Hide the fact that you are using a VPN at all.
Detection	Network admins and ISPs can easily see a VPN is active using DPI toolkits.	Appears as normal HTTPS web traffic.
Common Use Case	Daily privacy, secure public Wi-Fi, bypassing geo-blocks.	Overcoming strict national censorship, workplace blocks (e.g. do not see streaming during working hours), ISP throttling (e.g. slowing down your connection if they purposely target VPN traffic)
Examples	• Cisco Secure Client (formerly AnyConnect) • Fortinet FortiClient • Palo Alto Networks GlobalProtect	• TunnelBear • ProtonVPN • UltraSurf

Commercial vs Stealth VPNs



Feature	Commercial VPN	Stealth VPN
Packet Obfuscation	Any modern DPI-based firewall can easily identify over TLS/443 the distinct cryptographic handshake, packet metadata, and protocol headers unique to that vendor. Corporate VPNs *want* to be identifiable so enterprise IT admins can log and manage them properly.	A true stealth VPN uses active **obfuscation** specifically to alter or strip away those handshake signatures.
Differing Primary Objectives	Commercial SSL VPNs are built for Secure Access & Authentication. Their goal is to safely bridge a remote employee into a private corporate network using strict access controls and multi-factor authentication (MFA). They are not designed to hide from a hostile government firewall.	Stealth VPNs are built for Untraceability & Censorship Evasion. Their core purpose is to trick the network into thinking no VPN is being used at all, allowing users to bypass heavy censorship blocks or internet restrictions.

Stealth VPN Detection: The Easy Ones

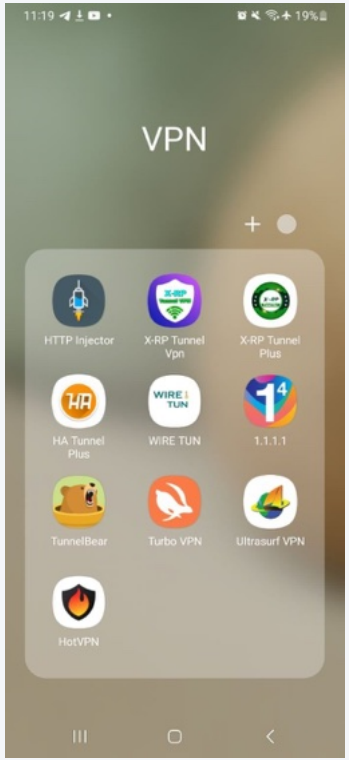


- DPI toolkits such as ntop's nDPI detect 35 VPN types including the stealth VPNs listed in the previous slide, i.e. the popular ones.
- They can be detected using some heuristics and via fingerprinting or certificate analysis.

TCP 10.0.2.15:37810 <-> 185.159.159.148:443 [proto: 91.344/TLS.ProtonVPN][Stack: TLS.ProtonVPN][IP: 0/Unknown][Encrypted][Confidence: DPI][DPI packets: 11][cat: VPN/2] [Breed: Acceptable][12 pkts/1454 bytes <-> 14 pkts/6607 bytes][Goodput ratio: 52/88][0.09 sec] [Hostname/SNI: vpn-api.proton.me][(Advertised) ALPNs: h2;http/1.1][(Negotiated) ALPN: h2] [bytes ratio: -0.639 (Download)][IAT c2s/s2c min/avg/max/stddev: 0/0 8/5 22/21 9/7][Pkt Len c2s/s2c min/avg/max/stddev: 60/54 121/472 358/1514 88/611][Risk: ** TLS Cert Expired **][Risk Score: 100][Risk Info: 29/May/2023 13:13:28 - 27/Aug/2023 13:13:27][nDPI Cli Fingerprint: c7206cddee635388c8e68a7bdd2a1812][nDPI Srv Fingerprint: 3c54ddcd036c][TCP Fingerprint: 2_64_64240_5ec4846073b9/Linux][TLSv1.2][JA4: t12d1209h2_d34a8e72043a_b39be8c56a14] [TLS-C: t12d1208h2_d34a8e72043a_036209cd1ead_30e06788707b][ServerNames: *.pr.tn,*.proton.me,*.storage.proton.me,pr.tn,proton.me][Issuer: C=US, O=Let's Encrypt, CN=R3] [Subject: CN=proton.me][Certificate SHA-1: AC:31:4E:05:15:6C:29:0B:D7:4F:31:3D:DE:CA:0F:C8:FF:E9:C6:4D][Safari][Validity: 2023-05-29 13:13:28 - 2023-08-27 13:13:27][Cipher: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256]

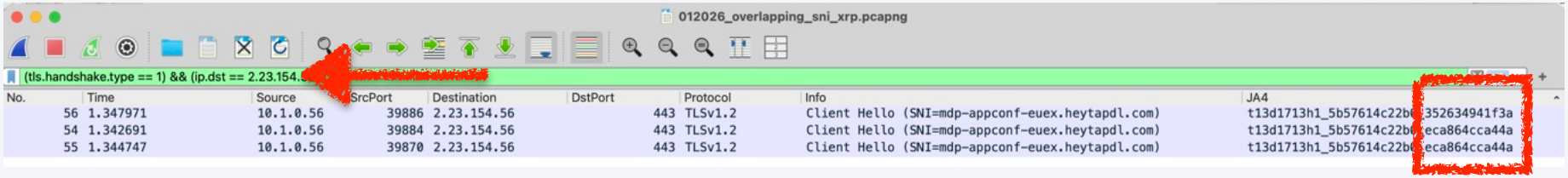
Stealth VPN Detection: The Hard Ones

- Google Play store contains many stealth VPNs we have used for this presentation (for some of them an iOS version often exists).
- Most VPNs tunnel traffic over TLS, but some also over SSH and DNS.
- Contrary to the "easy ones" the contact servers often hosted on cheap ISPs (contrary to commercial ones that use high-quality hosting), use random or configurable SNIs, open multiple concurrent TLS connections over different servers this to spread traffic across the Internet.
- Many (but not all) stealth VPNs we tested "leak" DNS traffic meaning that it is not sent inside the tunnel.
- The IP/ASN "inconsistency trick" allowed us to identify "in the wild" traffic that was "special" -> identify "suspicious" and use these hosts are starting point for our investigation activities.



Stealth VPNs: X-RP Tunnel Plus and JA4

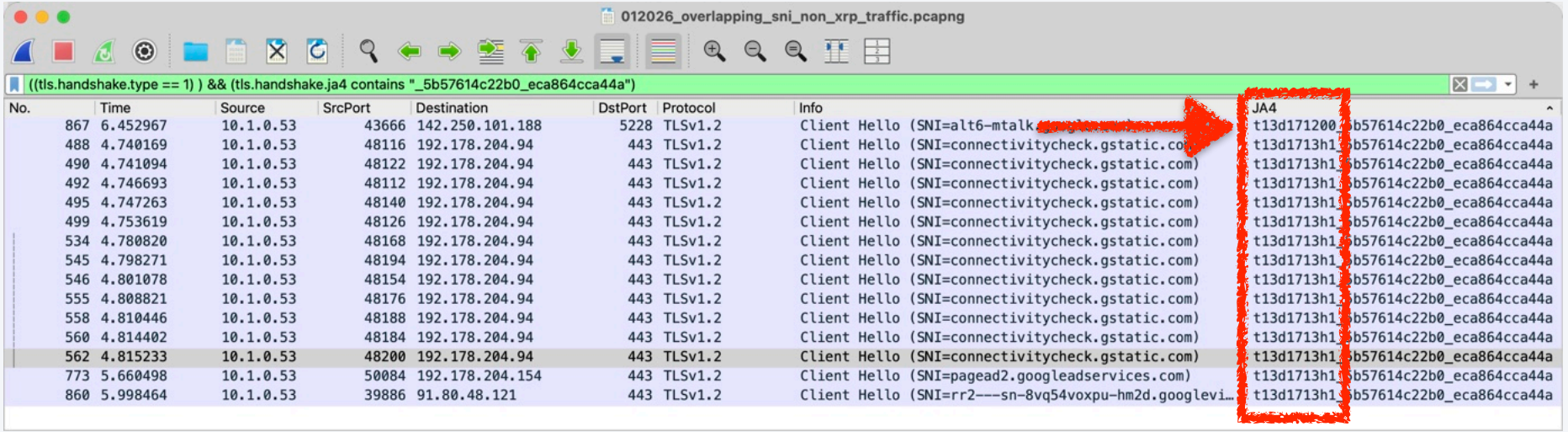
- This is a popular tunneling application that we have observed in live traffic across several device types and summer months.
- We have initially used the JA4 application fingerprint to see it was good enough to identify stealth VPN traffic:
  - Observing recurrent customers (i.e. the crew) across different phone types we confirmed that JA4 was not changing across devices.
  - However JA4 was not "stable enough" to be used as quick string comparison for our needs.



JA4_c
(Extensions)

File: pcap/XRP-TunnelPlus/01_2026/012026_overlapping_sni_xrp.pcapng

Stealth VPNs: Google Services and JA4



No.	Time	Source	SrcPort	Destination	DstPort	Protocol	Info	JA4
867	6.452967	10.1.0.53	43666	142.250.101.188	5228	TLSv1.2	Client Hello (SNI=alt6-mtalk	t13d171200_5b57614c22b0_eca864cca44a
488	4.740169	10.1.0.53	48116	192.178.204.94	443	TLSv1.2	Client Hello (SNI=connectivitycheck.gstatic.co	t13d1713h1_5b57614c22b0_eca864cca44a
490	4.741094	10.1.0.53	48122	192.178.204.94	443	TLSv1.2	Client Hello (SNI=connectivitycheck.gstatic.com)	t13d1713h1_5b57614c22b0_eca864cca44a
492	4.746693	10.1.0.53	48112	192.178.204.94	443	TLSv1.2	Client Hello (SNI=connectivitycheck.gstatic.com)	t13d1713h1_5b57614c22b0_eca864cca44a
495	4.747263	10.1.0.53	48140	192.178.204.94	443	TLSv1.2	Client Hello (SNI=connectivitycheck.gstatic.com)	t13d1713h1_5b57614c22b0_eca864cca44a
499	4.753619	10.1.0.53	48126	192.178.204.94	443	TLSv1.2	Client Hello (SNI=connectivitycheck.gstatic.com)	t13d1713h1_5b57614c22b0_eca864cca44a
534	4.780820	10.1.0.53	48168	192.178.204.94	443	TLSv1.2	Client Hello (SNI=connectivitycheck.gstatic.com)	t13d1713h1_5b57614c22b0_eca864cca44a
545	4.798271	10.1.0.53	48194	192.178.204.94	443	TLSv1.2	Client Hello (SNI=connectivitycheck.gstatic.com)	t13d1713h1_5b57614c22b0_eca864cca44a
546	4.801078	10.1.0.53	48154	192.178.204.94	443	TLSv1.2	Client Hello (SNI=connectivitycheck.gstatic.com)	t13d1713h1_5b57614c22b0_eca864cca44a
555	4.808821	10.1.0.53	48176	192.178.204.94	443	TLSv1.2	Client Hello (SNI=connectivitycheck.gstatic.com)	t13d1713h1_5b57614c22b0_eca864cca44a
558	4.810446	10.1.0.53	48188	192.178.204.94	443	TLSv1.2	Client Hello (SNI=connectivitycheck.gstatic.com)	t13d1713h1_5b57614c22b0_eca864cca44a
560	4.814402	10.1.0.53	48184	192.178.204.94	443	TLSv1.2	Client Hello (SNI=connectivitycheck.gstatic.com)	t13d1713h1_5b57614c22b0_eca864cca44a
562	4.815233	10.1.0.53	48200	192.178.204.94	443	TLSv1.2	Client Hello (SNI=connectivitycheck.gstatic.com)	t13d1713h1_5b57614c22b0_eca864cca44a
773	5.660498	10.1.0.53	50084	192.178.204.154	443	TLSv1.2	Client Hello (SNI=pagead2.googleadservices.com)	t13d1713h1_5b57614c22b0_eca864cca44a
860	5.998464	10.1.0.53	39886	91.80.48.121	443	TLSv1.2	Client Hello (SNI=rr2---sn-8vq54voxpu-hm2d.googlevi…	t13d1713h1_5b57614c22b0_eca864cca44a

File: pcap/XRP-TunnelPlus/01_2026/012026_overlapping_sni_xrp.pcapng JA4_a

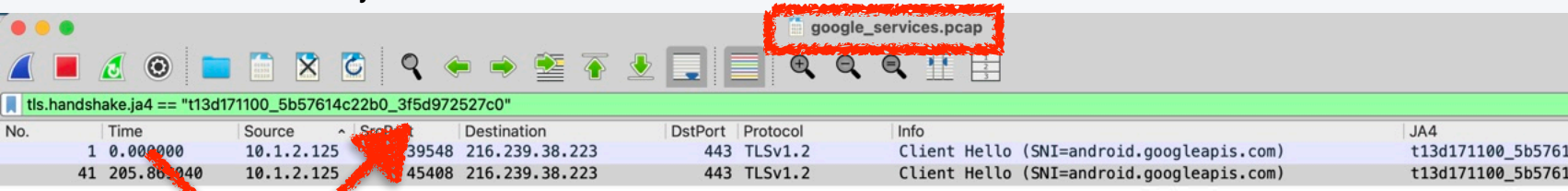
JA4 Observations

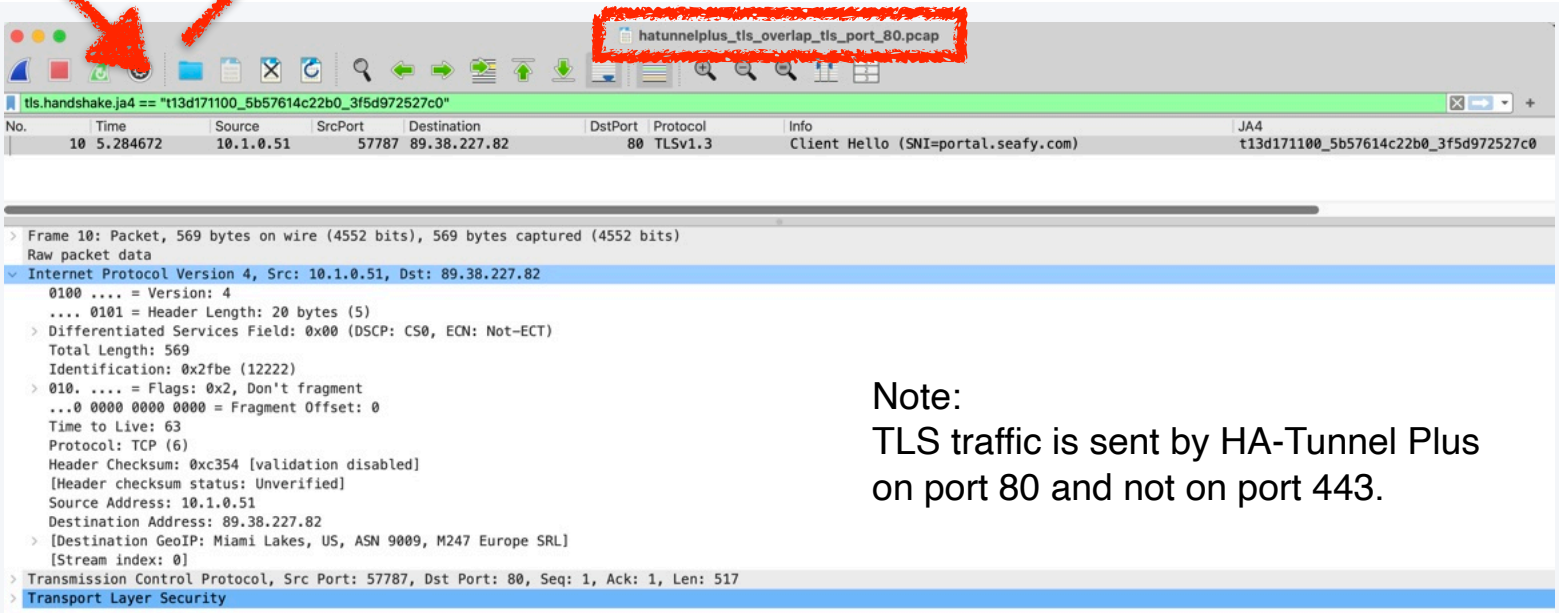


- Bad
  - JA4 hash is instabile. On a single device, the same application has multiple JA4. One would be desirable instead.
- Good
  - JA4 is different across different applications (XR-P Tunnel Plus vs Google service).
- Umm.. are really sure JA4 is different across applications?

Same JA4, Different Applications

Filter: tls.handshake.ja4 == "t13d171100_5b57614c22b0_3f5d972527c0"





Note:
TLS traffic is sent by HA-Tunnel Plus on port 80 and not on port 443.

Are False Positives a Problem?



- JA4 is a hash, and a hash has collisions by nature.
  - Hash collisions are mathematically unavoidable because a hash function maps an infinite (or incredibly large) set of possible inputs to a finite, fixed-size set of outputs.

- Consequences of a hash collision:
  - Misclassify traffic: not nice but *mostly* ok with passive monitoring.
  - Very bad if used in traffic enforcement (e.g. IPS) as you block legitimate traffic.

- If collisions are very little (<< 1%) you can still accept (but not like) them. Unfortunately in our experience we have collisions with very popular traffic as Google Android services that makes JA4 completely useless in this use case.
- Note that TCP fingerprint won't help as both clients are running on Android, so their fingerprint is identical.

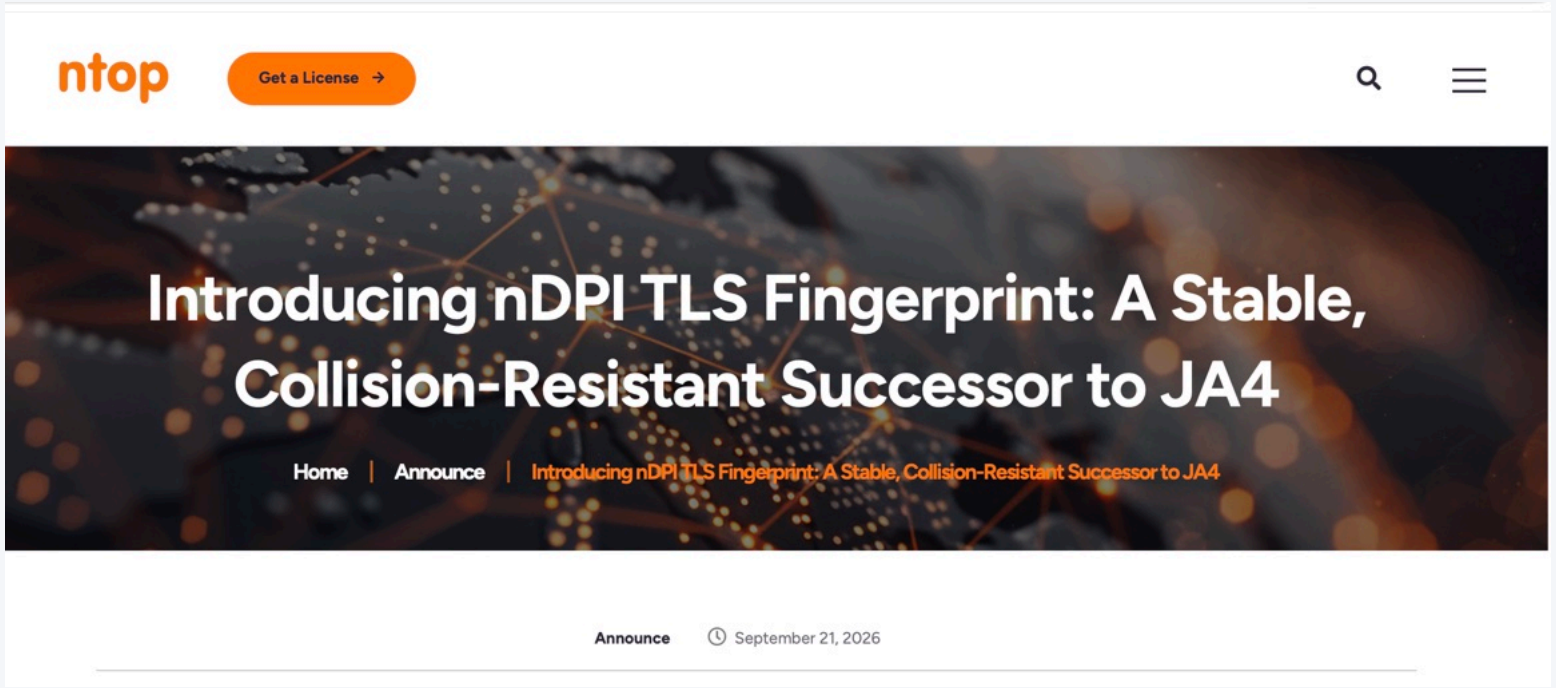
How can we create a "better" JA4 ?

- Client Hello "almost" identical, only minor differences that would make JA4 even more unstable if taken into account.

```
Frame 10: Packet, 569 bytes on wire (4552 bits), 569 bytes captured (4552 bits)
Raw packet data
Internet Protocol Version 4, Src: 10.1.0.51, Dst: 89.38.227.82
Transmission Control Protocol, Src Port: 57787, Dst Port: 80, Seq: 1, Ack: 1, Len:
Transport Layer Security
  [Stream index: 0]
  TLSv1.3 Record Layer: Handshake Protocol: Client Hello
    Content Type: Handshake (22)
    Version: TLS 1.0 (0x0301)
    Length: 512
    Handshake Protocol: Client Hello
      Handshake Type: Client Hello (1)
      Length: 508
      Version: TLS 1.2 (0x0303)
      Random: 1352dbfb0f733a54e49f67fde286779dc17d01499e6867d4e060aec0faa40143
      Session ID Length: 32
      Session ID: 53fd6382f85cfbf394d6b71b9cc4096dea992ebe66eaf9387d1cc1b437acd2
      Cipher Suites Length: 34
      Cipher Suites (17 suites)
      Compression Methods Length: 1
      Compression Methods (1 method)
      Extensions Length: 401
      Extension: server_name (len=21) name=portal.seafy.com
      Extension: extended_master_secret (len=0)
      Extension: renegotiation_info (len=1)
      Extension: supported_groups (len=8)
      Extension: ec_point_formats (len=2)
      Extension: status_request (len=5)
      Extension: signature_algorithms (len=20)
      Extension: key_share (len=38) x25519
      Extension: supported_versions (len=9) TLS 1.3, TLS 1.2, TLS 1.1, TLS 1.0
      [JA4: t13d171100_5b57614c22b0_3f5d972527c0]
      [JA4_r: t13d171100_002f,0035,009c,009d,1301,1302,1303,c009,c00a,c013,c014,
      [JA3 Fullstring: 771,4865-4866-4867-49195-49196-52393-49199-49200-52392-49
      [JA3: 9c815150ea821166faecf80757d8826a]
```

```
Frame 1: Packet, 583 bytes on wire (4664 bits), 583 bytes captured (4664 bits)
Ethernet II, Src: 76:98:66:1c:7c:6c (76:98:66:1c:7c:6c), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
Internet Protocol Version 4, Src: 10.1.2.125, Dst: 216.239.38.223
Transmission Control Protocol, Src Port: 39548, Dst Port: 443, Seq: 1, Ack: 1, Len: 517
Transport Layer Security
  [Stream index: 0]
  TLSv1 Record Layer: Handshake Protocol: Client Hello
    Content Type: Handshake (22)
    Version: TLS 1.0 (0x0301)
    Length: 512
    Handshake Protocol: Client Hello
      Handshake Type: Client Hello (1)
      Length: 508
      Version: TLS 1.2 (0x0303)
      Random: f47eb6ad7f6722098232ebd7f4746910d1f1a240b0c85d1b182319c6902aee30
      Session ID Length: 32
      Session ID: ca1b5053de46f3193a186bfc9ba619cc01f9b928523dd350ca7d38f20b96e440
      Cipher Suites Length: 34
      Cipher Suites (17 suites)
      Compression Methods Length: 1
      Compression Methods (1 method)
      Extensions Length: 401
      Extension: server_name (len=27) name=android.googleapis.com
      Extension: extended_master_secret (len=0)
      Extension: renegotiation_info (len=1)
      Extension: supported_groups (len=8)
      Extension: ec_point_formats (len=2)
      Extension: status_request (len=5)
      Extension: signature_algorithms (len=20)
      Extension: key_share (len=38) x25519
      Extension: supported_versions (len=5) TLS 1.3, TLS 1.2
      [JA4: t13d171100_5b57614c22b0_3f5d972527c0]
      [JA4_r: t13d171100_002f,0035,009c,009d,1301,1302,1303,c009,c00a,c013,c014,c02b,c02c,c02f,c03
      [JA3 Fullstring: 771,4865-4866-4867-49195-49196-52393-49199-49200-52392-49161-49162-49171-49
      [JA3: 9c815150ea821166faecf80757d8826a]
```

Yes we can. This is just a preview.



https://www.ntop.org/introducing-tlsfp-a-stable-collision-resistant-successor-to-ja4/

Fingerprints: Behavior and Patterns



Even if the traffic is completely encrypted and no header values are unique, the physical *way* the data moves can be analyzed using Wireshark:

Payload-based Fingerprinting

- Packet Length Sequences: A specific video streaming app or video game sends bursts of highly predictable packet sizes. For instance, a series of packet lengths like [512, 1024, 64, 1500] can indicate a specific software function.
- Inter-Arrival Time (IAT): The exact millisecond spacing between back-to-back packets. Automated bots or malicious command-and-control servers send data at precise rhythmic intervals, whereas human behavior is erratic.
- Directional Ratios: The ratio of uploaded packets to downloaded packets over a given time window.

Fingerprints: SPLT [1/2]



Payload-based Fingerprinting

Sequence of Packet Lengths and Times (*SPLT*) is used to measure the "shape" of a conversation to classify traffic, identify applications, and spot malicious behaviors without decrypting the data. It models network traffic as a chronological time series of the first several (usually 30 to 50, enough to capture the handshake and initial application response) packets of a flow.

- Packet Length (Size): The exact number of bytes in the application payload after transport headers (e.g., TCP/UDP headers) are stripped.
- Packet Direction: Formatted natively or as a multiplier (e.g., +1 for inbound/client-to-server and -1 for outbound/server-to-client).
- Inter-Arrival Time (IAT): The delta time—measured in milliseconds or nanoseconds—indicating how much time elapsed since the immediately preceding packet in the flow was observed.

Fingerprints: SPLT [2/2]



Payload-based Fingerprinting

- Encrypted Traffic Analytics (ETA): Pioneered at scale by enterprise architectures like Cisco Systems Encrypted Traffic Analytics [1], SPLT exports metadata using IP Flow Information Export (IPFIX) profiles to let security software flag malware hiding inside TLS sessions.
- Website & App Fingerprinting: Different sites or mobile applications trigger unique structural sequences of data requests and responses. Machine learning classifiers analyze SPLT arrays to identify what app or website a user is visiting based purely on behavioral timing and bursts.
- Malware Detection: Command and Control (C2) channels often rely on automated beacons that generate highly predictable SPLT patterns compared to normal human web-browsing behavior.

[1] https://github.com/cisco/mercury

SPLT and Tshark [1/2]



Payload-based Fingerprinting

```
$ cd SPLT
$ ./tshark.sh
$ ./analysis.py
[[ 1.0000e+00  0.0000e+00         nan]
 [-1.0000e+00  0.0000e+00  4.2060e-02]
 [ 1.0000e+00  0.0000e+00  1.7400e-04]
 [ 1.0000e+00  3.2200e+02  1.3480e-03]
 [-1.0000e+00  0.0000e+00  4.1064e-02]
 [-1.0000e+00 -1.4400e+03  1.9410e-03]
 [ 1.0000e+00  0.0000e+00  1.2200e-04]
 [-1.0000e+00 -1.4400e+03  4.9000e-05]
 [-1.0000e+00 -9.1900e+02  5.4000e-05]
 [ 1.0000e+00  0.0000e+00  3.0000e-05]
 [ 1.0000e+00  0.0000e+00  1.2600e-04]
 [ 1.0000e+00  6.0000e+00  4.9480e-03]
 [ 1.0000e+00  5.8000e+01  1.0000e-06]
....
```

Direction
1=c->s
-1=s->c

(+/-) Lenght

Interarrival Time

SPLT and Tshark [2/2]



- Things you can do with SPLT
  - Website and Application Fingerprinting
    Identify what an encrypted flow is doing (e.g., Netflix vs. Tor vs. standard web browsing):
    - Feed your vectorized SPLT profiles into a classic machine learning model like a Random Forest Classifier or a Support Vector Machine (SVM).
    - Train the model using labeled SPLT data. As different apps load resources in highly specific structural sequences, the model can accurately predict the application layer protocol without ever looking at the encrypted payload.
  - Threat Detection and Malware Beaconing
    - C2 Channel Detection: Command and Control (C2) software often uses automated scripts to *beacon* back to a malicious server. Look for high regularity or repeating patterns in your IAT.
    - Anomaly Detection: Train an unsupervised model like an Isolation Forest or an *Autoencoder* on standard corporate network SPLT data. When malware communicates via a non-standard protocol or custom encrypted channel, its SPLT sequence will flag as an outlier.

Payload-based Fingerprinting

Introducing TLS Handshake Fingerprint



- SPLT is a good idea but it has some problems when not used for malware. The main problem is that the time might be unstable across different networks (on ships we use geostationary satellites or Starlink) making it complex to create a fingerprint also usable on faster networks.
- Idea: what about fingerprinting TLS blocks (not packets), their direction and length?
- TLS Handshake Fingerprint on a nutshell: [<block type>,<block length>*<direction>]*
  - <block type>: enum of TLS block types (see ndpi_tls_block_type)
  - <block length>: block length without SNI length (ephemeral).
  - <direction>: 1=client->server, -1=server->client
  - Only the first 8 TLS blocks are considered.
  - Example 4=301,5=-118,1=-1,15=-64,15=-3432,15=-97,15=-53,1=1

```
typedef enum {
  tls_unknown = 0,
  tls_change_cipher,
  tls_alert,
  tls_handshake_encrypted_message,
  tls_handshake_client_hello,
  tls_handshake_server_hello,
  tls_handshake_new_session_ticket,
  tls_handshake_encrypted_extn,
  tls_handshake_certificate,
  tls_handshake_server_key_exchange,
  tls_handshake_certificate_request,
  tls_handshake_server_hello_done,
  tls_handshake_certificate_verify,
  tls_handshake_client_key_exchange,
  tls_handshake_finished,
  tls_application_data,
  tls_heartbeat,
} ndpi_tls_block_type;
```

Using TLS Handshake Fingerprint



- Command: `nDPI/example/ndpiReader -v 2 -i ~/Dropbox/Sharkfest/SharkfestEU2026/pcap/TLSBlocksFingerprint/<pcap> --cfg tls,application_blocks_tracking,1 --cfg tls,max_num_blocks_to_analyze,8 -K json -k /tmp/a.json`
- Same JA4, different TLS blocks


```
"tls": {
   "version": "TLSv1.3",
   "ja3s": "eb1d94daa7e0344597e756a1fb6e7054",
   "ja4": "t13d1713h1_5b57614c22b0_eca864cca44a",
   "tlsfp": "t13d1710h1_5b57614c22b0_8074ade24f04_30e06788707b",
   "unsafe_cipher": 0,
   "cipher": "TLS_AES_128_GCM_SHA256",
   "advertised_alpns": "http/1.1",
   "tls_supported_versions": "TLSv1.3,TLSv1.2",
   "tls_blocks": [
     "22:1=489,22:2=-118,20=-1,23=-5210,20=1,23=53,23=1294,23=-559"
   ]
 }
```

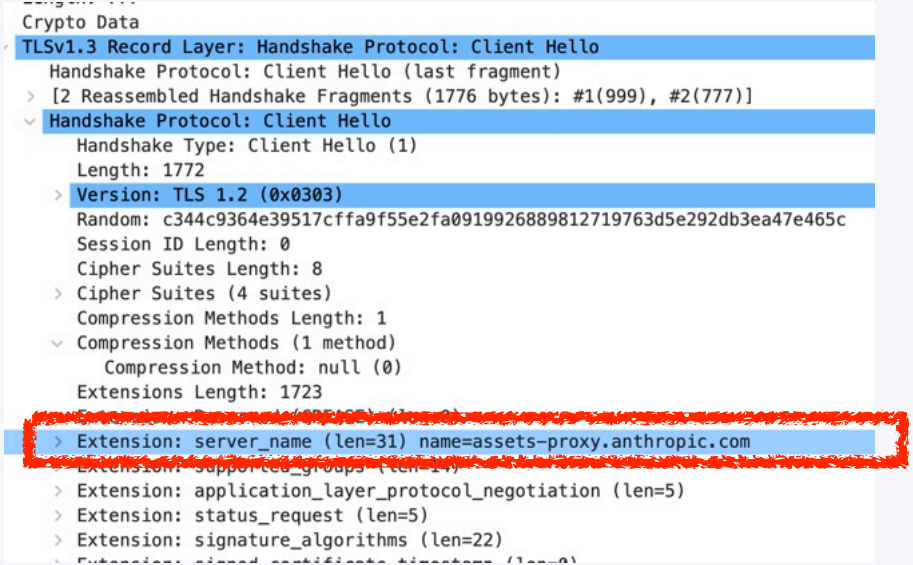
```
"tls": {
   "version": "TLSv1.3",
   "ja3s": "f4febc55ea12b31ae17cfb7e614afda8",
   "ja4": "t13d1713h1_5b57614c22b0_eca864cca44a",
   "tlsfp": "t13d1710h1_5b57614c22b0_8074ade24f04_30e06788707b",
   "unsafe_cipher": 0,
   "cipher": "TLS_AES_128_GCM_SHA256",
   "advertised_alpns": "http/1.1",
   "tls_supported_versions": "TLSv1.3,TLSv1.2",
   "tls_blocks": [
     "22:1=490,22:2=-118,20=-1,23=-1017,23=-1517,23=-615,20=1,23=53"
   ]
 }
```

Part II:
AI Traffic Analysis



AI Traffic Detection

- Traffic generated by commercial/popular AI companies such as OpenAI or Anthropic is pretty simple to detect.



File: pcap/AI/anthropic.pcapng

- But what about traffic generated by systems powered by applications such as llama.cpp (https://github.com/ggml-org/llama.cpp) or vllm (https://vllm.ai/) ?

AI Traffic: Streamed Response



```
POST /v1/chat/completions HTTP/1.1
Host: aitest:2999
User-Agent: curl/8.5.0
Accept: */*
Authorization: Bearer ntopng_test
Content-Type: application/json
Content-Length: 116

{"model": "Qwen3.6-27B", "messages": [{"role": "user", "content": "Hello! Tell me a 3-word joke."}], "stream":
true}
HTTP/1.1 200 OK
Keep-Alive: timeout=5, max=100
Content-Type: text/event-stream
Server: llama.cpp
Transfer-Encoding: chunked
Access-Control-Allow-Origin:

212
data: {"choices":[{"finish_reason":null,"index":0,"delta":
{"role":"assistant","content":null}}],"created":1784915896,"id":"chatcmpl-
xmlXHHxA7xKVvKGqWt8ZStqZ1TLxdU2U","model":"Qwen3.6-27B-Q4_K_M.gguf","system_fingerprint":"b9334-
fca6b9350","object":"chat.completion.chunk"}

data: {"choices":[{"finish_reason":null,"index":0,"delta":{"content":"Why"}}],"created":1784915896,"id":"chatcmpl-
xmlXHHxA7xKVvKGqWt8ZStqZ1TLxdU2U","model":"Qwen3.6-27B-Q4_K_M.gguf","system_fingerprint":"b9334-
fca6b9350","object":"chat.completion.chunk"}
```

AI Traffic: Non Streamed Response



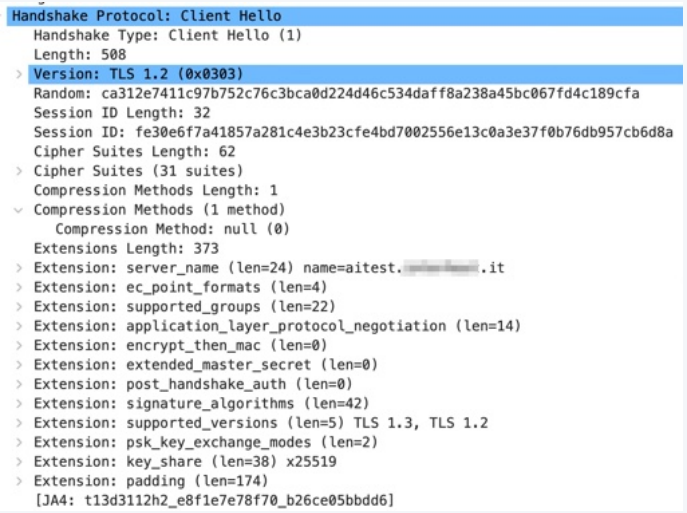
```
POST /v1/chat/completions HTTP/1.1
Host: aitest:2999
User-Agent: curl/8.5.0
Accept: */*
Authorization: Bearer ntopng_test
Content-Type: application/json
Content-Length: 143

{"model": "Qwen3.6-27B", "messages": [{"role": "user", "content": "Hello! Write a 500 words essay on large
language models"}], "stream": false}
HTTP/1.1 200 OK
Keep-Alive: timeout=5, max=100
Content-Type: application/json; charset=utf-8
Server: llama.cpp
Content-Length: 4909
Access-Control-Allow-Origin:

{"choices":[{"finish_reason":"stop","index":0,"message":{"role":"assistant","content":"**\n\n**The Rise and
Implications of Large Language Models**\n\nLarge Language Models (LLMs) have emerged as one of the most
transformative technologies of the twenty-first century, fundamentally altering how humans interact with
information, create content, and solve complex problems. At their core, LLMs are artificial intelligence
systems trained on vast datasets of text to understand, generate, and manipulate human language. By leveraging
deep learning architectures, particularly the transformer model, these systems have achieved unprecedented
levels of linguistic proficiency, enabling them to perform tasks ranging from translation and summarization to
coding and creative writing.\n\nThe foundational architecture of LLMs relies on self-supervised learning, where
the model predicts the next word in a sequence based on the preceding context. This approach allows the model
to learn intricate patterns of syntax, semantics, and even stylistic nuances without explicit instruction for
every possible task. As the scale of these models inc
```

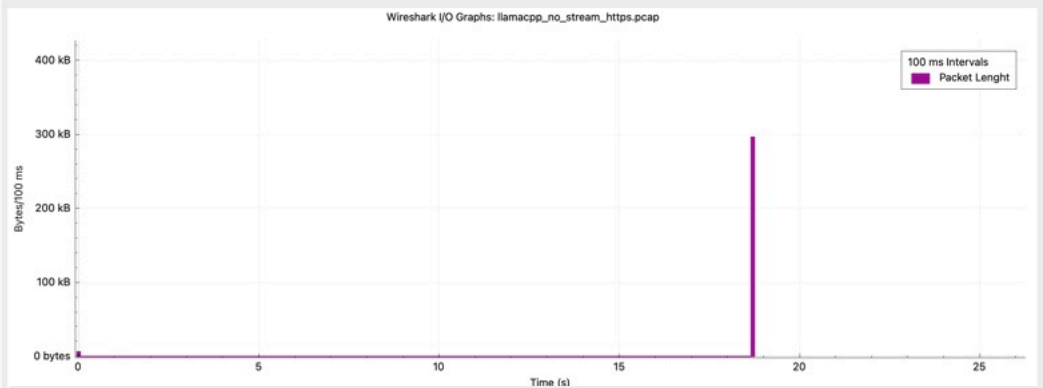
AI Traffic Detection

- When AI traffic is over HTTP, it's trivial to detect it as all clients follow the OpenAI API specifications (https://github.com/openai/openai-openapi, https://developers.openai.com/api/reference/overview). Just parse the URLs and check the JSON content.
- What about HTTPS instead ? How can we detect local users using "custom" AI services or on "not-so-popular" LLMs services that might exfiltrate data from our company.
- Can we create a fingerprint of this traffic and detect it, similar to what we did with stealth VPNs ?
- This is an example of this traffic, there is nothing special to report from a header-based, we need to combine this with flow analysis over time without limiting to the first few connection packets.

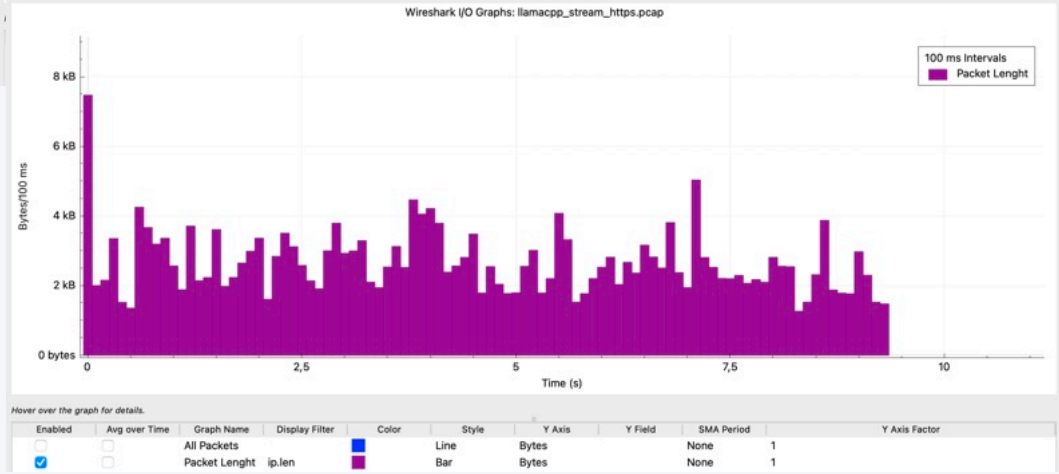


AI Traffic: Streaming vs non Streaming

No.	Time	Source	SrcPor
66	2.108024	93.48.237.…	
67	2.108034	212.54.224…	
68	2.109542	93.48.237.…	
69	2.109552	212.54.224…	
70	2.111025	93.48.237.…	
71	2.112220	93.48.237.…	
72	2.112230	212.54.224…	
73	2.114063	93.48.237.…	
74	2.115241	93.48.237.…	
75	2.115252	212.54.224…	
76	2.117023	93.48.237.…	
77	2.118299	93.48.237.…	
78	2.118309	212.54.224…	
79	2.119540	93.48.237.…	
80	2.119567	212.54.224…	
81	18.733937	212.54.224…	
82	18.733952	212.54.224…	
83	18.733954	212.54.224…	
84	18.733978	212.54.224…	
85	18.733981	212.54.224…	
86	18.733983	212.54.224…	
87	18.733984	212.54.224…	
88	18.733987	212.54.224…	
89	18.733992	212.54.224…	
90	18.733995	212.54.224…	
91	18.735465	151.11.51.…	
92	18.735499	212.54.224…	
93	18.735504	212.54.224…	
94	18.735507	212.54.224…	
95	18.735509	212.54.224…	
96	18.735511	212.54.224…	
97	18.735512	212.54.224…	
98	18.735514	212.54.224…	
99	18.735516	212.54.224…	
100	18.735517	212.54.224…	
101	18.735529	212.54.224…	
102	18.735531	212.54.224…	



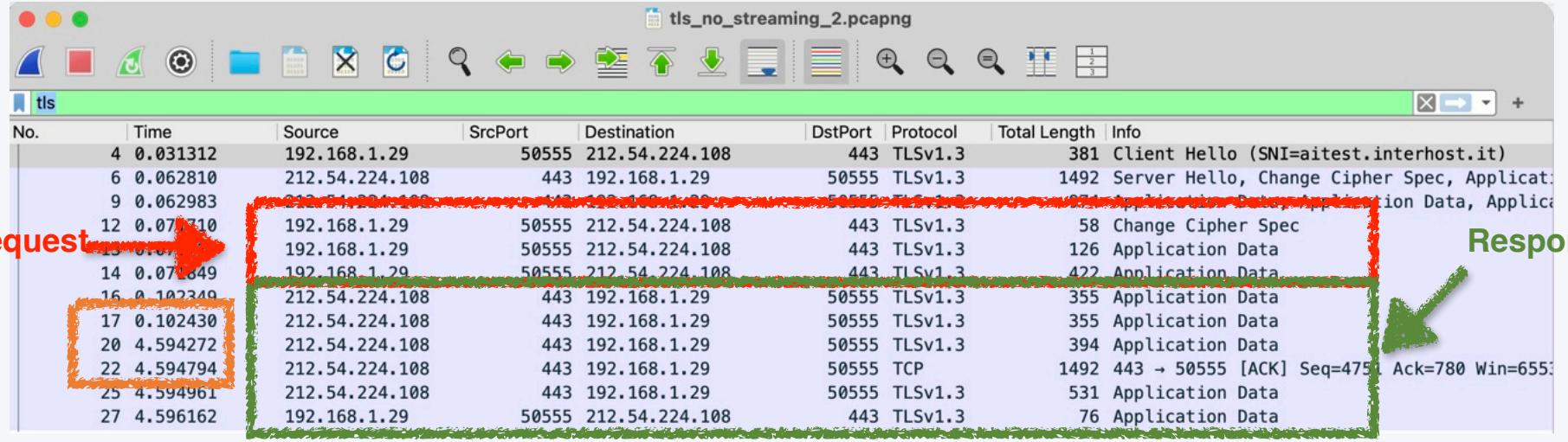
Without Streaming



With Streaming

AI Traffic Detection [1/2]

- Simplified detection algorithm
  - After TLS negotiation, client sends the request whose length can change according to its length (a few or many TLS application blocks).
  - The server starts replying sending short (< 500 bytes) TLS blocks. This is the initial response, but LLM is still working, no response yet.
  - Inference takes a bit of time (see packet 20), then the reply is sent.



AI Traffic Detection [2/2]



```
$ nDPI/example/ndpiReader -v 2 -i /Volumes/T7/deri/Dropbox/Sharkfest/
SharkfestEU2026/pcap/AI/llamacpp_stream_https.pcap

        1        TCP 151.11.51.52:36504 <-> 212.54.224.108:443 [proto: 91/
TLS][Stack: TLS][IP: 0/Unknown][Encrypted][Confidence: DPI][FPC: 0/Unknown,
Confidence: Unknown][DPI packets: 177][cat: Web/5][Breed: Safe][390 pkts/
26727 bytes <-> 441 pkts/223998 bytes][Goodput ratio: 4/87][9.31 sec]
[Hostname/SNI: aitest.interhost.it][(Advertised) ALPNs: h2;http/1.1][TLS
Supported Versions: TLSv1.3;TLSv1.2][bytes ratio: -0.787 (Download)][IAT c2s/
s2c min/avg/max/stddev: 0/0 22/20 107/66 16/16][Pkt Len c2s/s2c min/avg/max/
stddev: 66/66 69/508 583/1514 32/253][Risk: ** Possible AI Inference Traffic
**** Non PQC Compliant Flow **][Risk Score: 60][Risk Info: TLS 1.3 negotiated
non-PQ group 0x001D;AI streaming][nDPI Srv Fingerprint: 81dba73ae426][TCP
Fingerprint: 2_64_64240_789c1d1f/Linux][MSS: 1460][TLSv1.3][JA4:
t13d3112h2_e8f1e7e78f70_b26ce05bbdd6][TLS-C:
t13d3110h2_e8f1e7e78f70_75e657529f98_8180606ab83d][JA3S:
15af977ce25de452b96affa2addb1036][Firefox][Cipher: TLS_AES_256_GCM_SHA384]
```

From Packet Fingerprints to Application Behaviour



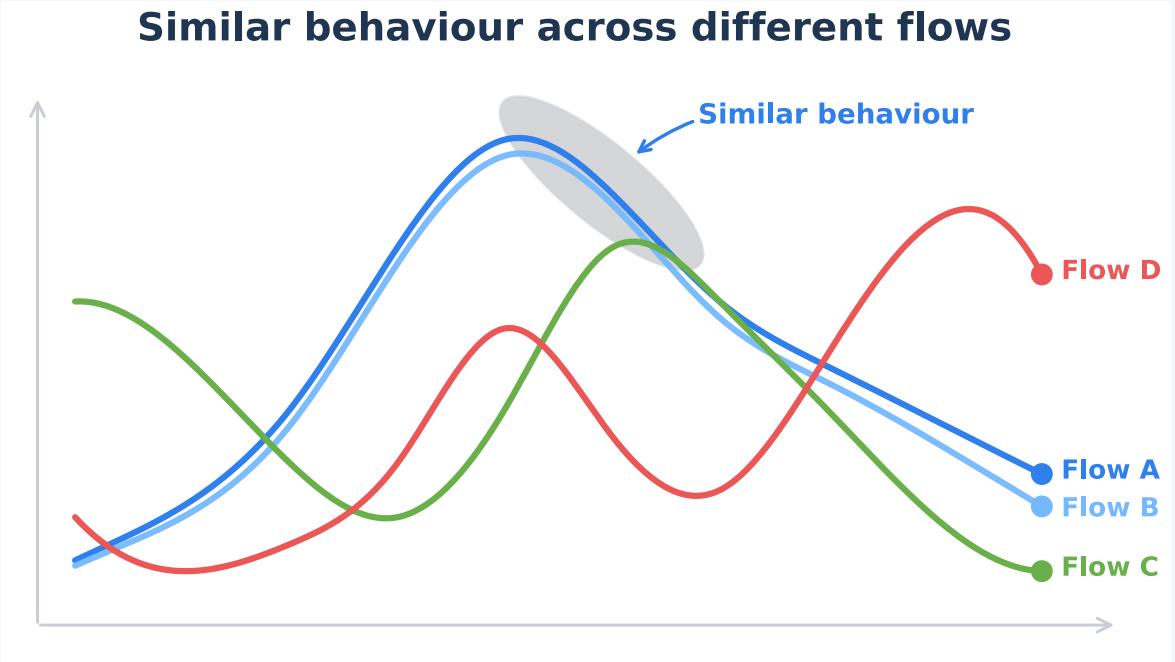
- There are many other examples (e.g. identify transactions for crypto currencies) where this technique can be used to “unveil” hidden activities.
- Fingerprints are a great way to detect application traffic.
- They have limitations when the hash fingerprint is not stable, but we have seen that there are effective workarounds to make it a robust solution.
- Initial TLS negotiation can be easily represented with a compact representation, but more complex behaviour as AI traffic detection can be more challenging.
- For advanced application detection, AI can be a better alternative to fingerprints whenever:
  - Application behaviour to be modelled is not trivial.
  - Application fingerprint is unknown (i.e. when we need to find steal VPNs in the wild).

- This has been the motivation behind the use of AI transformers.

Part III:
From Fingerprint to Transformers

Stealth VPN traffic detection through network behaviour

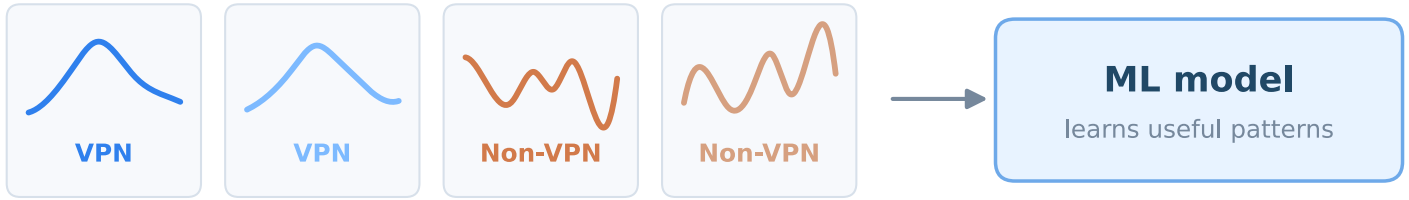
- **Fingerprinting**: Signature-based detection
  - Identifies traffic using known signatures and patterns
  - Fast and efficient
- **Machine Learning**: Data-driven detection
  - Learns behavioural patterns from labeled network traffic examples
  - Does not required knowing explicit signatures
- **Complementary** approaches
  - ML can help when fingerprints are unavailable, ambiguous, or unreliable.



How does Supervised Machine Learning work?

TRAINING

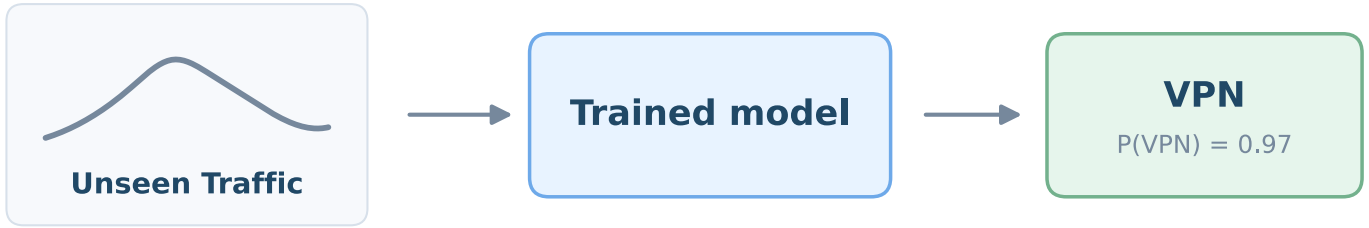
Learn from labeled traffic examples (VPN / Non-VPN)



Labeled examples

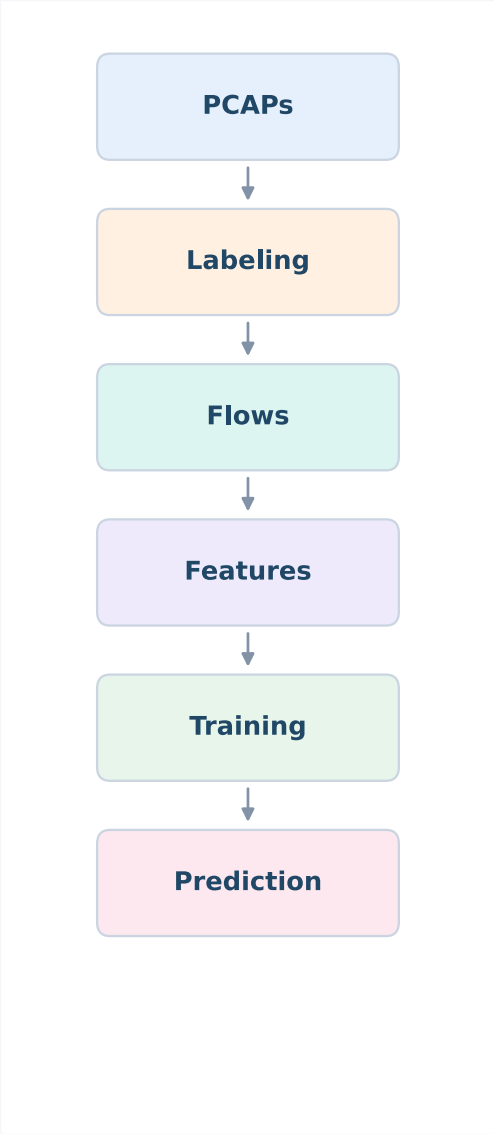
INFERENCE

Recognize previously unseen traffic



From PCAP to Prediction

- **PCAPs**: Capture network traffic or use existing PCAP datasets
- **Labeling**: Assign VPN/Non-VPN labels to network traffic
- **Flows**: Identify network flows and associate them with a proper label
- **Features**: Extract features describing traffic behaviour over temporal windows
- **Training**: Train the model using labeled traffic features
- **Prediction**: Use the trained model to classify unseen network traffic



Dataset: Where does our data come from?

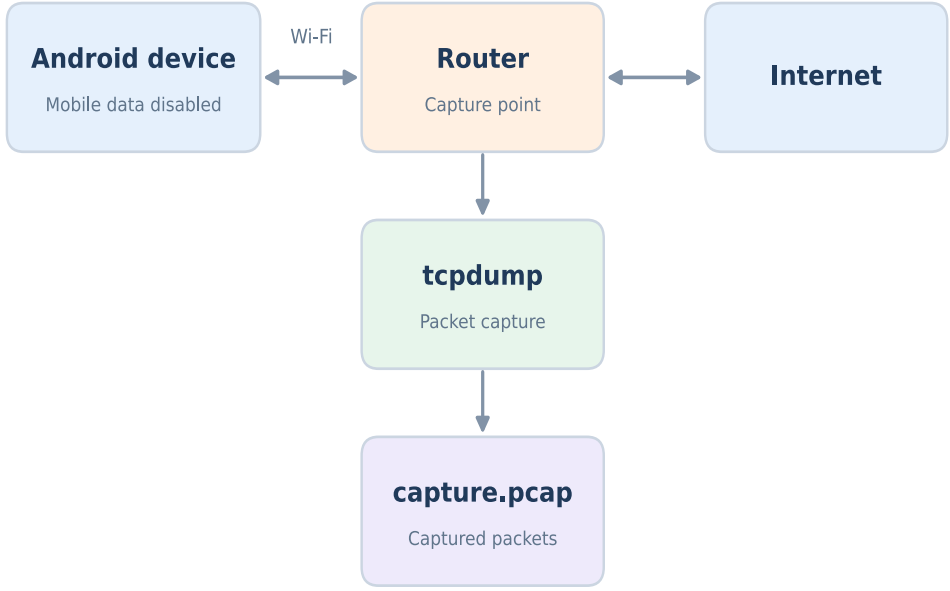
- **Public datasets**: Traditional VPN Traffic used to train our base model
  - VNAT [1]
  - ISCXVPN2016 [2]
- **Our Own Captures [3]**: Stealth VPN Traffic used to train our fine-tuned model
  - Controlled captures of stealth VPNs and Non-VPN traffic
  - Traffic collected from Android devices using different applications:
    - HA Tunnel Plus
    - XU Tunnel
    - TLS Tunnel
    - Psiphon
    - Geph

Dataset	Model	#PCAPs	VPN flows	Non-VPN flows
VNAT	Base Model	165	382 (1.63%)	23050 (98.37%)
ISCX VPN	Base Model	140	17778 (9.42%)	170848 (90.58%)
Own Captures	Fine Tuning + Test	31	44 (0.28%)	15645 (99.72%)

[1] https://archive.ll.mit.edu/datasets/vnat/VNAT_release_1.zip
[2] https://www.unb.ca/cic/datasets/vpn.html
[3] https://drive.google.com/file/d/1NqxvlDbUAbC_Xt3Zj8PCv_A7LuqRwjC7/view?usp=sharing

Dataset: Capturing and Labeling Network Traffic

- Controlled traffic capture setup
  - Android smartphone connected to a router via Wi-Fi
  - Mobile data disabled
  - Network traffic captured on the router using tcpdump
  - `tcpdump -i br-lan -s 0 -w capture.pcap`
- Record VPN sessions
  - Record the VPN application and its activation and disconnection times
  - Each capture contains normal traffic and traffic generated while the VPN is active
- Annotate VPN flows
  - Identify and manually verify candidate VPN streams
  - Store the reviewed annotations in csv file



From Capture to Reviewed Flow Annotations

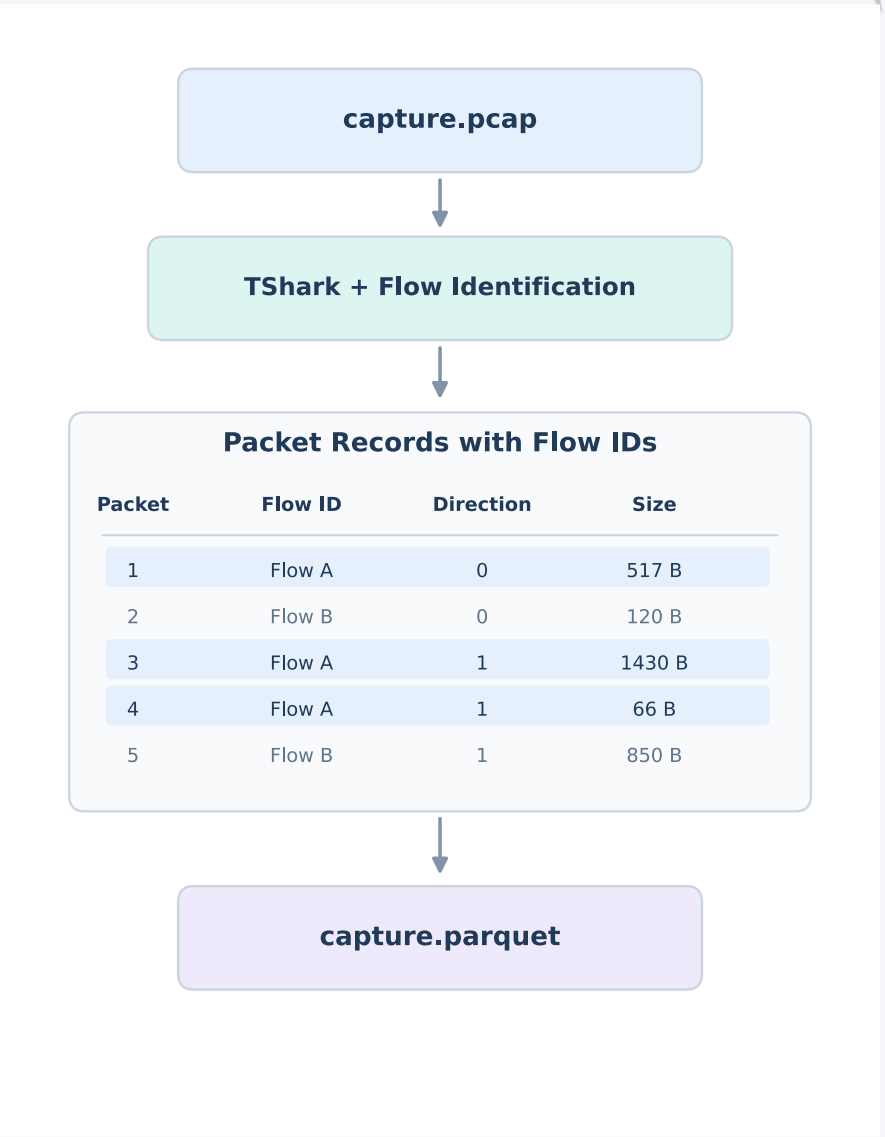
- The challenge
  - Every captures contains multiple network connections
  - Not every connection belongs to the VPN tunnel
- Discover candidate streams
  - Identify active streams during the VPN session
- Verify the stream
  - Use TShark to examine individual streams
  - `tshark -r captures/capture.pcap -Y “tcp.stream == 76”`
- Annotate
  - Save verified VPN labels in CSV
- Only manually verified streams are labeled as VPN
- Key commands
  - `make inspect-stream`
  - `tshark …`
  - `make annotate-add`



Annotation still needs a human reviewer

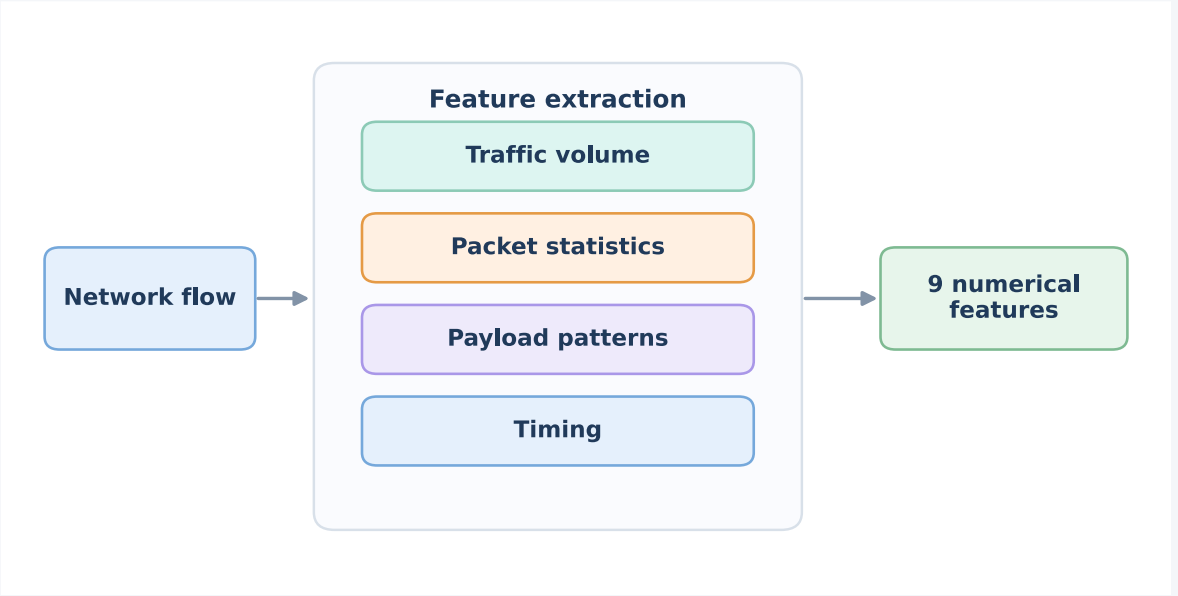
From Packets to Flows

- **TShark:** extracts packets level fields
- **Parser**: builds normalized bidirectional flow IDs using IP addresses, ports and protocol
  - `IP endpoint A|IP endpoint B|port A|port B|protocol`
- **Direction**: is assigned relative to the first observed direction in each flow
- **Labels**: VPN or Non-VPN labels are assigned using the reviewed annotations
- **Output**: the resulting structured data is stored in Parquet files with one row per packet
- **Command**:
  - ```
    make parse \
        CAPTURES=data/captures \
        ANNOTATIONS=data/captures/annotation.csv \
        PARSED=data/parsed
    ```
- **Output example**:
  - `data/parsed/captured_01.parquet`


    ```

From Flows to Features

- The network flows are transformed into 9 numerical representations that describe their behaviour
  - Traffic volume:
    - forward and backward bytes
    - forward and backward packets
  - Packet statistics:
    - average packet size
    - payload-length variation
  - Payload patterns:
    - entropy of payload lengths
  - Timing:
    - mean inter-arrival time
    - standard deviation of inter-arrival times
- Goal: Represent traffic behaviour in a format that a Machine Learning model can process (no payload inspection)



Capturing Traffic Behaviour with Windows and Bins

- To capture traffic behaviour over time:
  - **Windows**: Divide each flow into fixed-duration time windows with overlap between consecutive windows
  - **Bins**: Divide each window into smaller temporal intervals
  - **Features**: Aggregate packet-level observations within each bin to compute nine numerical features
- Configuration used in our experiments:
  - **Windows size**: 100 seconds
  - **Bin size**: 10 seconds
  - **Stride**: 50 seconds (50% overlap)
  - **Features**: 9 per bin
- Output: A fixed-size representation of 9 features × 10 temporal bins
- Command:
  - make features-all FEATURE_WINDOW_SIZE=100\ FEATURE_BIN_SIZE=10 FEATURE_STRIDE=50



From PCAP to Packet Fields



- Real capture HA Tunnel Plus
  - Filename 2026-09-07_16-16-17_ha_tunnel_plus_android_sni_sartiano_info_preserve_sni_ee_b6_09_9a_11_8c.pcap
  - TCP stream: 2
  - Client: 192.168.1.140:38393
  - Server: 167.17.69.14:80
  - TLS SNI: sartiano.info
- TShark extracts packet-level fields
  - Timestamps, source and destination endpoints
  - Transport protocol and packet lengths

Time (s)	Source	Destination	Frame (bytes)	TCP payload (bytes)
0	A	B	74	0
0.099	B	A	74	0
0.196	A	B	66	0
55.298	A	B	66	0
82.45	A	B	66	0
89.036	B	A	1506	1440
127.015	B	A	1506	1440
606.365	B	A	54	0

A = 192.168.1.140:38393 and B = 167.17.69.14:80

- TShark command:
  - `tshark -r capture.pcap -T fields -e frame.time_epoch -e ip.src -e ip.dst -e tcp.srcport -e udp.srcport -e tcp.dstport -e udp.dstport -e ip.proto -e frame.len -e tcp.len -E header=y -E separator=,`
- The parser creates a normalized bidirectional flow ID
  - `Flow ID: 167.17.69.14|192.168.1.140|80|38393|6`
  - `Direction: 0|1|0|0|0|1|1|1`

From Packets to Flows



- The parser enriches the packet records with:
  - Flow ID: A normalized identifier shared by packets travelling in both directions
  - Direction: 0 for the first observed packet direction, 1 for the reverse direction
  - Label: 1 for VPN, 0 for non-VPN, based on reviewed annotations
- Real example:
  - Flow ID → 167.17.69.14|192.168.1.140|80|38393|6
  - Direction
    - 0: 192.168.1.140:38393 → 167.17.69.14:80
    - 1: 167.17.69.14:80 → 192.168.1.140:38393

Time (s)	pkt_size (bytes)	direction	payload_len (bytes)	label
0	74	0	0	1
0.099	74	1	0	1
0.196	66	0	0	1
55.298	66	0	0	1
82.45	66	0	0	1
89.036	1506	1	1440	1
127.015	1506	1	1440	1
606.365	54	1	0	1

- **Output:** Parquet file containing one record per packet, ready for feature extraction.

From Flows to Temporal Windows



- Real Example
  - Flow duration: 606.36s
  - Total packets: 281,438
  - Packets are grouped using their normalized flow ID
- Temporal Window configuration used in our experiments:
  - **Windows size**: 100 seconds
  - **Bin size**: 10 seconds
  - **Stride**: 50 seconds (50% overlap)
  - **Features**: 9 per bin
- Each window captures the flow's behaviour during a specific time interval. The same packet may appear in two overlapping windows.

Window	Relative interval	Absolute start	Absolute end	Packets	Forward	Backward
W1	0–100 s	1788790605	1788790705	17908	6163	11745
W2	50–150 s	1788790655	1788790755	34729	8638	26091
W3	100–200 s	1788790705	1788790805	114512	17633	96879
W11	500–600 s	1788791105	1788791205	3732	1265	2467

- Consecutive windows overlap by 50 seconds. The flow continues until approximately 606.36 s, but no final partial window is created after the last complete window. For short flows, the implementation creates one 100-second window starting at the first packet. Empty bins remain zero-filled; no artificial packets are added.

From Windows to Numerical Features [1/2]

- The first window contains 10 bins

Bin	Relative interval	Packets	Forward	Backward
1	0–10 s	357	166	191
2	10–20 s	939	397	542
3	20–30 s	1410	289	1121
4	30–40 s	111	52	59
5	40–50 s	187	85	102
6	50–60 s	4141	1412	2729
7	60–70 s	632	278	354
8	70–80 s	1334	484	850
9	80–90 s	7112	2402	4710
10	90–100 s	1685	598	1087

- Window 1
  - Duration: 100 seconds
  - Total packets 17,908
  - Divided into 10 bins of 10 seconds each

- The 9 features are computed as follows:
  - **fwd_bytes**: sum of pkt_size for direction 0
  - **bwd_bytes**: sum of pkt_size for direction 1
  - **fwd_pkts**: number of direction 0 packets
  - **bwd_pkts**: number of direction 1 packets
  - **avg_pkt_size**: total frame bytes divided by packet count
  - **payload_size_variation**: std(payload_len) / mean(payload_len)
  - **payload_entropy**: base-2 entropy of the payload-length distribution
  - **iat_mean**: mean inter-arrival time within the bin
  - **iat_std**: standard deviation of inter-arrival times within the bin

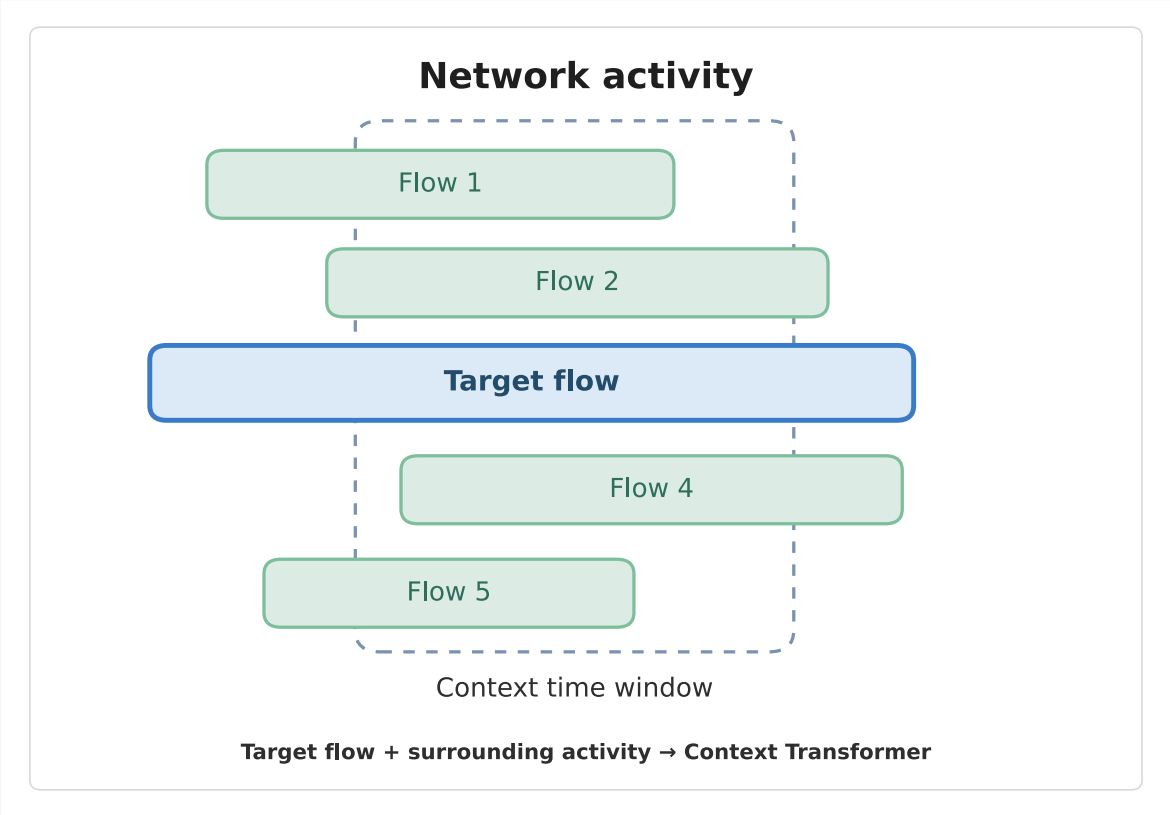
From Windows to Numerical Features [2/2]

- Raw feature matrix
  - 9 features calculated for each of the 10 temporal bins (9, 10)
- Logarithmic transformation
  - Apply log1p(abs(feature)) to each feature value
  - Compress large value to reduce the errors
  - Example – Bin 1
    - fwd_bytes = 57,917 → 10.9668
- Final output
  - One transformed feature matrix per target window
  - Shape (9, 10)
  - Ready to be combined with network context by the Context Transformer

Feature	Raw	Transformed
fwd_bytes	57917	10.9668
bwd_bytes	84977	11.3501
fwd_pkts	166	5.118
bwd_pkts	191	5.2575
avg_pkt_size	400.263	5.9946
payload_size_variation	1.446	0.8945
payload_entropy	3.884	1.586
iat_mean	0.026	0.0255
iat_std	0.127	0.1194

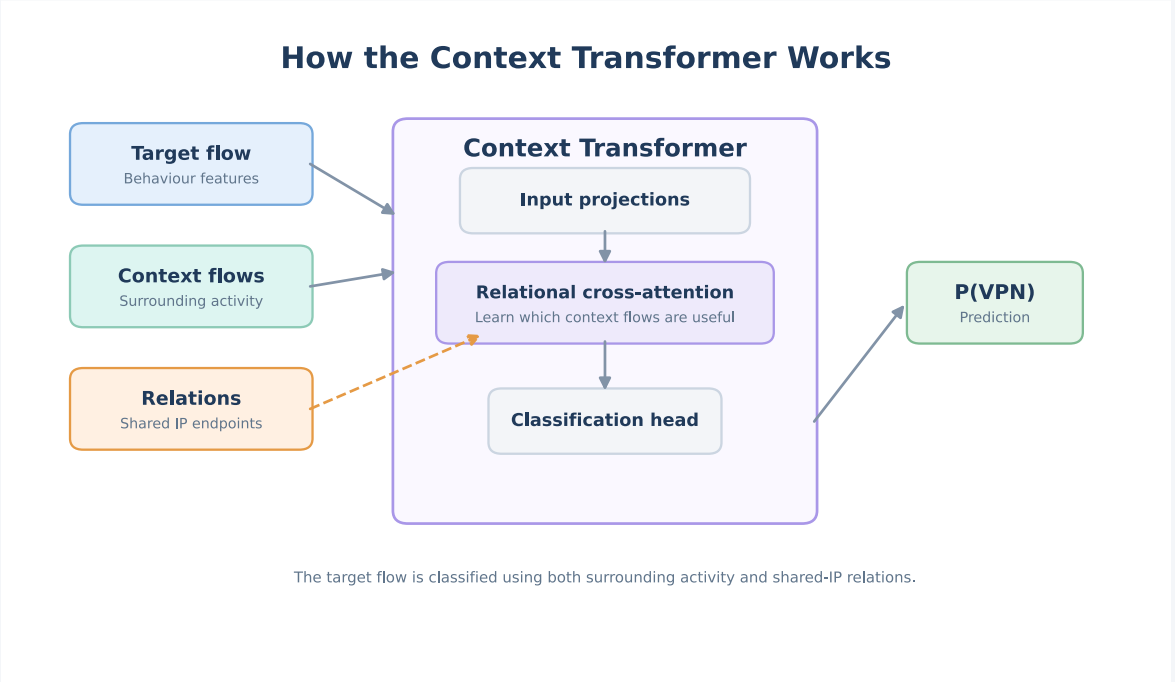
Context: Adding surrounding information

- A single flow may not provide enough information to identify VPN traffic
  - Surrounding network activity may provide additional clues
- Context representation
  - Select temporally nearby windows from the same PCAP
  - Each target window is enriched with up to 20 context elements
  - Relations: encode whether the target and context flows share IP endpoints
- Goal: Combines the target flow's behaviour with information from surrounding network activity
- Key idea: Temporal proximity selects the context; shared IP endpoints describe the relationships.
- Command:
  - make features-all FEATURE_CONTEXT_SIZE=20



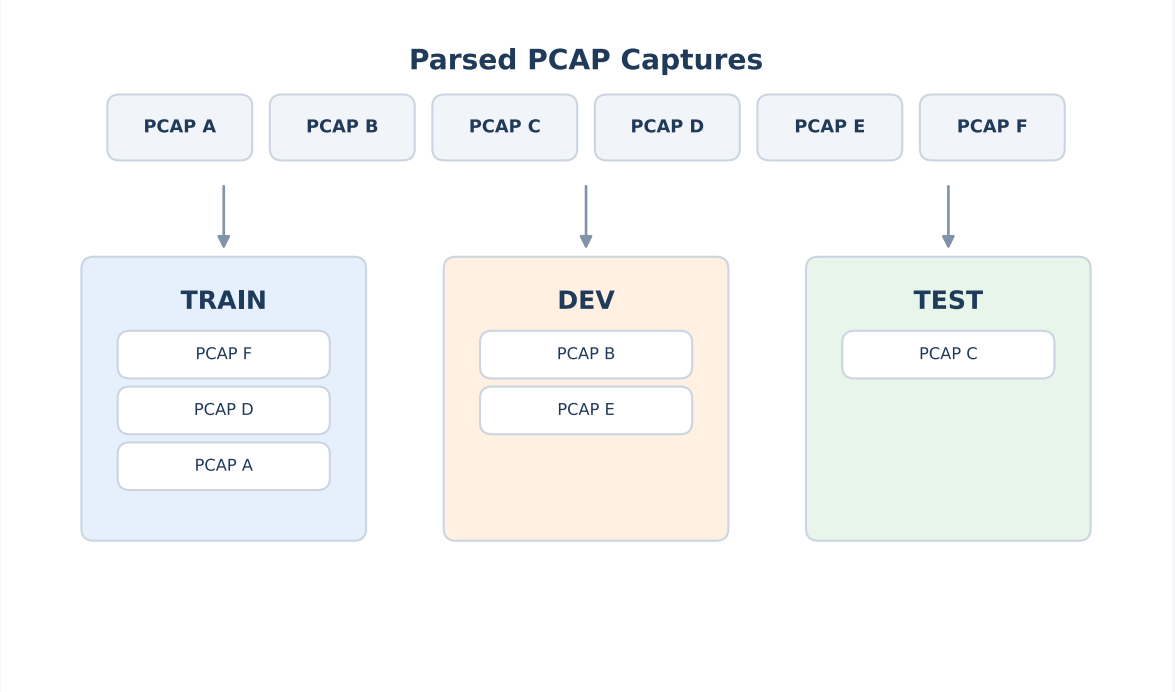
The Context Transformer

- Model Input
  - **Target features**: describe the behaviour of the window to classify
  - **Context features**: describe surrounding network activity
  - **Relations**: shared IP endpoints between target and context flows
- Relational Cross-Attention
  - The target attends to surrounding context elements
  - Relation information helps the model learn which context elements are relevant
- Output
  - A VPN probability for each target window



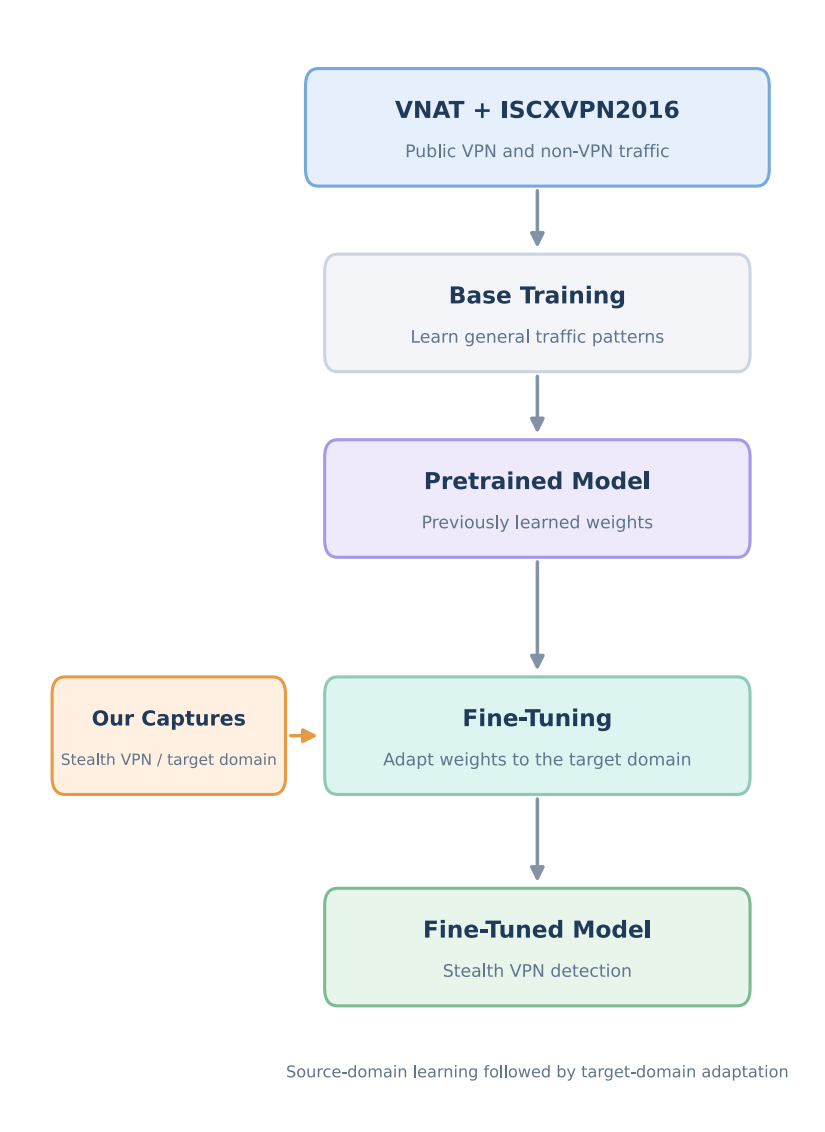
Splitting the Dataset

- The dataset is split in three data subset
  - Training set: used to train the model
  - Validation set: used to tune the model evaluating during the training
  - Test: used to evaluate the final model on unseen traffic
- Avoiding Data leakage
  - Split by PCAP capture, not by individual flows or windows
  - All flows and windows from the same PCAP belong to the same subset
  - Perform the split after parsing and before feature extraction
- Command:
  - make split PARSED=data/parsed \
    SPLITS=data/splits SPLIT_SEED=42
- Output: train.csv, dev.csv, test.csv



Training and Fine-Tuning for Domain Adaptation

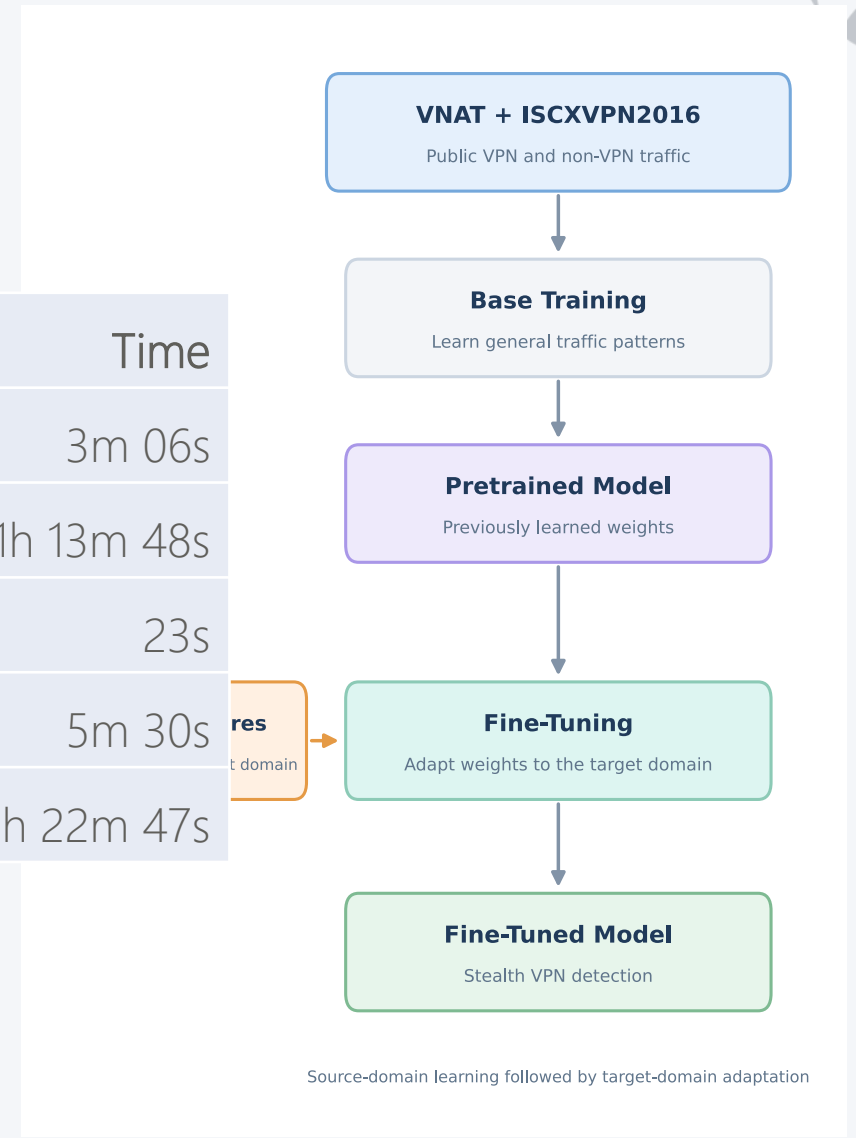
- Base Training
  - Train the model using VNAT and ISCX VPN 2016
  - Learn general patterns of VPN and Non-VPN traffic
- Why Fine-Tuning?
  - Our Stealth VPN captures are much smaller and represent a different traffic distribution
- Fine-Tuning
  - Start from the pretrained base model
  - Adapt the model weight's to the target domain using our Stealth VPN captures
  - Use only the training subset, keeping validation and test captures separate
- Commands
  - make paper-base-train
  - make finetune-train



Training and Fine-Tuning for Domain Adaptation

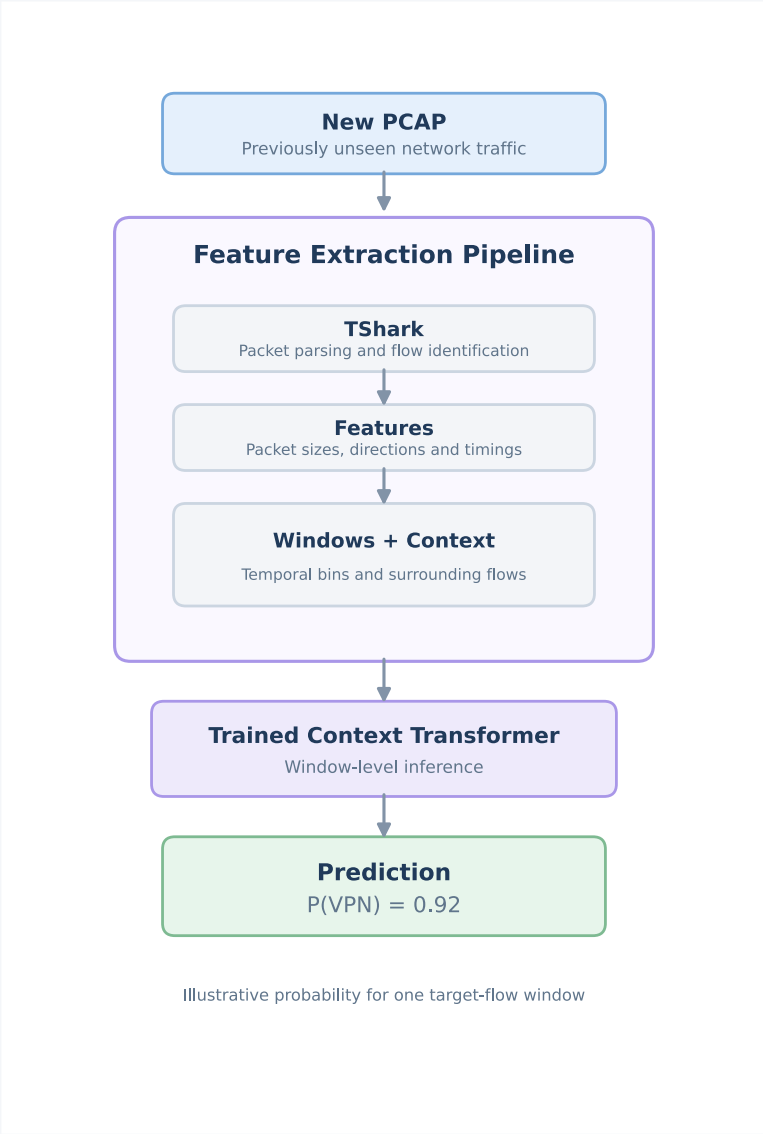
- Base Training
  - Train the model using VNAT and ISCX VPN 2016
  - Learn general patterns of VPN and Non-VPN traffic
- Why Fine-Tuning?
  - Our Stealth VPN capture
    represent a different traff
- Fine-Tuning
  - Start from the pretrained
  - Adapt the model weight'
    using our Stealth VPN ca
  - Use only the training sub
    test captures separate
- Commands
  - make paper-base-train
  - make finetune-train

Stage	Time
Base feats extraction	3m 06s
Base training	1h 13m 48s
Fine-tuning feats extraction	23s
Fine-tuning training	5m 30s
Total	1h 22m 47s



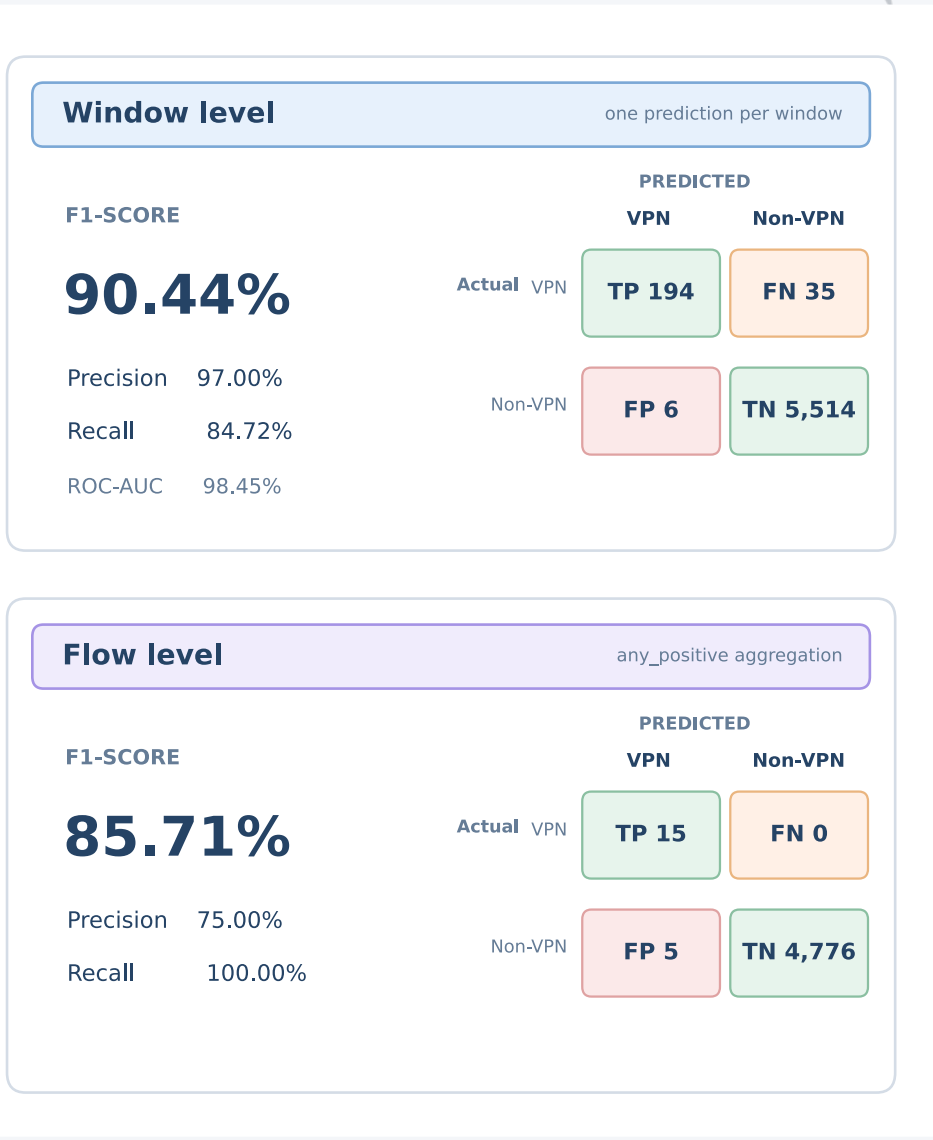
From Features to Predictions

- Feature Extraction
  - Process the PCAP using TShark and our Parser
  - Generate temporal feature windows
- Model Inference
  - Load the fine-tuned Context Transformer model
  - Classify each target window using its features and surrounding context
- Prediction
  - Output a VPN probability, P(VPN), for each target window
- Command
  - make predict \
    CHECKPOINT=tmp/models/finetuned/best.pt \
    PREDICTION_FEATURES=tmp/sharkfest/features/test.pt \
    PREDICTIONS=tmp/reports/test_predictions.npz



Evaluating the Detector

- **Evaluation** on held-out PCAP captures
  - **Precision:** How many VPN detections are correct
  - **Recall:** How many actual VPN instances are detected
  - **F1-score:** Balances precision and recall
- **Two evaluation levels**
  - **Window level:** Evaluate individual target windows
  - **Flow level:** Classify a flow as VPN if at least one window is positive (any_positive)
- **Key Results**
  - Window-level F1-score: **90.44%**
  - Flow-level F1-score: **85.71%**
  - All **15 VPN flows** detected, with **5 false-positive flows**.



Try It Yourself

- Reproduce the VPN detection pipeline
- Environment Setup (Ubuntu / Debian)
  - cd sharkfest26
  - sudo apt-get install -y tshark
  - python3 -m venv .venv
  - source .venv/bin/activate
  - pip install -r requirements.txt
  - make check
- Prepare the datasets
  - Download, decompress and put the datasets into:
    - data/VNAT_release_1/
    - data/ISCXVPN2016/
  - Own Captures:
    - data/own_captures/
    - data/own_captures/annotation.csv
- Run the complete experiment
  - make experiment \

```
RUN_DIR=tmp/sharkfest_test \
BASE_VNAT_CAPTURES=data/VNAT_release_1 \
BASE_ISCX_CAPTURES=data/ISCXVPN2016 \
FINETUNE_CAPTURES=data/captures \
FINETUNE_ANNOTATIONS=data/captures/annotation.csv \
PARSE_WORKERS=8 \
TRAIN_DEVICE=auto
```

- Outputs:
  - tmp/sharkfest_test/models/
  - tmp/sharkfest_test/reports/test_metrics.json

Run the VPN Detector as a Service

- A lightweight FastAPI service for VPN detection
  - PCAP upload → TShark → Feature Extraction → VPN predictions
- Start the service
  - make serve SERVICE_CHECKPOINT=models/finetuned/best.pt SERVICE_DEVICE=auto
- Pass the pcap
  - curl -X POST -F 'file=@data/captures/2026-09-11_18-50-05_xu_tunnel_android_ee_b6_09_9a_11_8c.pcap' http://127.0.0.1:8000/predict/pcap
- Output
  - Detected VPN Flows
  - Positive windows / total windows
  - VPN window fraction


```
{
 "file": "2026-09-11_18-50-05_xu_tunnel_android_ee_b6_09_9a_11_8c.pcap",
 "total_windows": 1158,
 "total_flows": 816,
 "returned_flows": 3,
 "predictions": [
  {
   "source_file": "2026-09-11_18-50-05_xu_tunnel_android_ee_b6_09_9a_11_8c.pcap",
   "flow_id": "161.97.116.249|192.168.1.140|80|48896|6",
   "prediction": 1,
   "probability": 0.9906897175879705,
   "vpn_windows": 21,
   "total_windows": 21,
   "vpn_fraction": 1
  },
  {
   "source_file": "2026-09-11_18-50-05_xu_tunnel_android_ee_b6_09_9a_11_8c.pcap",
   "flow_id": "185.196.20.49|192.168.1.140|80|46214|6",
   "prediction": 1,
   "probability": 0.9855311142431723,
   "vpn_windows": 37,
   "total_windows": 37,
   "vpn_fraction": 1
  },
  {
   "source_file": "2026-09-11_18-50-05_xu_tunnel_android_ee_b6_09_9a_11_8c.pcap",
   "flow_id": "192.168.1.140|62.171.150.156|49762|80|6",
   "prediction": 1,
   "probability": 0.9889576236406962,
   "vpn_windows": 18,
   "total_windows": 18,
   "vpn_fraction": 1
  }
 ]
}
```

Thank You!

WIRESHARK

